DATA STRUCTURES AND ALGORITHMS

For
COMPUTER SCIENCE

DATA STRUCTURES & ALGORITHMS

SYLLABUS

Programming and Data Structures: Programming in C. Recursion. Arrays, stacks, queues, linked lists, trees, binary search trees, binary heaps, graphs.

Algorithms: Searching, sorting, hashing. Asymptotic worst case time and space complexity. Algorithm design techniques: greedy, dynamic spanning trees, shortest paths.

ANALYSIS OF GATE PAPERS

Data Structures				Algorithms			
Exam Year	1 Mark Ques.	2 Mark Ques.	Total	Exam Year	1 Mark Ques.	2 Mark Ques.	Total
2003	5	7	19	2003	3	5	13
2004	7	10	27	2004	1	7	15
2005	5	6	17	2005	2	7	16
2006	1	7	15	2006	7	5	17
2007	2	7	16	2007	2	8	18
2008	3	7	17	2008	-	11	22
2009	1	4	9	2009	3	5	13
2010	1	5	11	2010	1	3	7
2011	1	3	7	2011	1	3	7
2012	1	4	9	2012	3	3	9
2013	-	3	6	2013	4	3	10
2014 Set-1	3	3	9	2014 Set-1	2	2	6
2014 Set-2	3	3	9	2014 Set-2	2	2	6
2014 Set-3	3	3	9	2014 Set-3	1	3	5
2015 Set-1	3	3	6	2015 Set-1	3	3	9
2015 Set-2	3	2	4	2015 Set-2	3	3	9
2015 Set-3	4	6	16	2015 Set-3	2	3	8
2016 Set-1	3	5	13	2016 Set-1	2	3	8
2016 Set-2	3	6	15	2016 Set-2	2	2	6
2017 Set-1	4	3	10	2017 Set-1	2	1	4
2017 Set-2	2	4	10	2017 Set-2	1	3	7
2018	4	3	10	2018	1	3	7

CONTENTS

1 INTRODUCTION TO DATA STRUCTURES

1.1 INTRODUCTION

A computer is a machine that processes information and data. The study of data structures includes how this information and data is related and how to use this data efficiently for various applications.

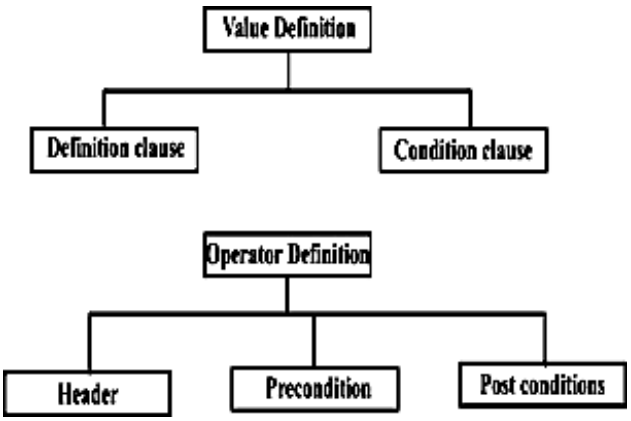


Fig 1: Value and Operator Definition

- A logical property of a data type is specified by abstract data type, or ADT.
- A data type definition consist of:
  (i) Value definition i.e. values and
  (ii)Operator definition i.e. set of operations on these values.
- These values and the operations on them form a mathematical construct that can be implemented using hardware or software data structures.
  Note: ADT is not concerned with implementation details.
- Any two values in an ADT are equal if and only if the values of their components are equal.

1.2 LINKED LISTS

- There are certain drawbacks of using sequential storage to represent stacks and queue.
  i) A fixed amount of storage remains allocated to the stack or queue even when the structure is actually using a smaller amount or possibly no storage at all.
  ii) Only fixed amount of storage may be allocated, making overflow a possibility.
- In a sequential representation, the items of a stack or a queue are implicitly ordered by the sequential order of storage. If q[x] represents an element of a queue, the next element will be q [(x+1) % MAXQUEUE] i.e. if x equals MAXQUEUE -1 then next element is q [0].
- Suppose that the items of a stack or a queue are explicitly ordered i. e. each item contain the address of the next item within itself. Such an explicit ordering gives rise to a data structure known as a Linear linked list as shown in the figure 2.

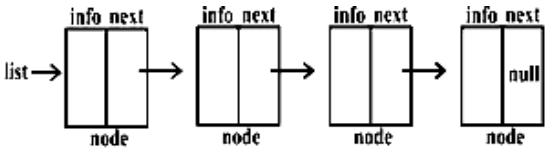


Fig 2: Linked List

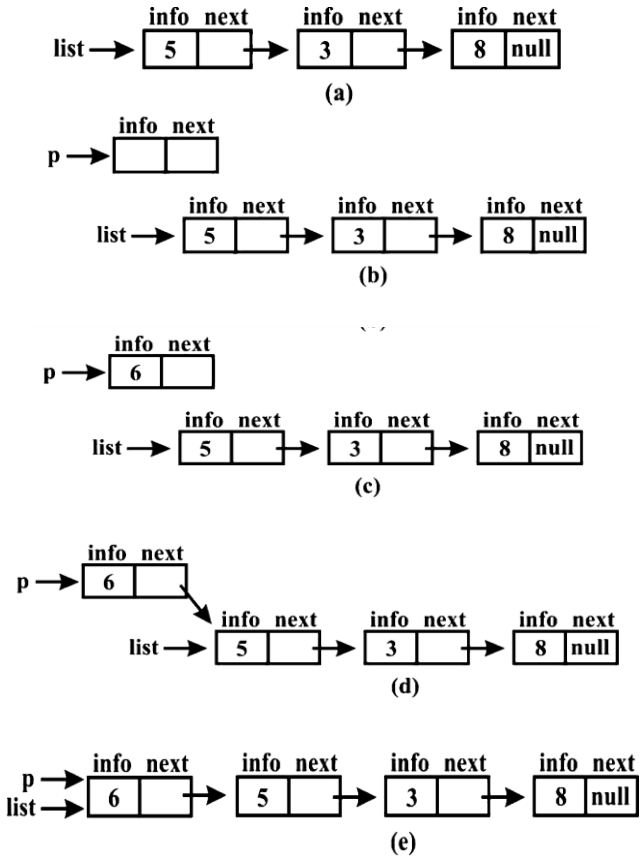
- Each item in the list is called a node and it contains two fields.
  i) Information field: It holds the actual element on the list.
  ii) The next address field: It contains the address of the next node in the list. Such an address, which is used to access a particular node, is known as a pointer.
- The entire linked list is accessed from an external pointer (list) that points to (contains the address of) the first node in the list.

- The next address field of the last node in the linked list contains a special value called as null. This is not a valid address and used to signal the end of a list.
- The linked list with no nodes is called the empty list or the null list. The value of the external pointer, list, to such a linked list is the null. The list can be initialized to form the empty list by the operation list = null.

1.2.1 INSERTING AND REMOVING NODES FROM A LIST

1.2.1.1 Insertion

A list is a dynamic data structure, the number of nodes on a list may vary as elements are inserted and removed. The dynamic nature of a list may be contrasted with the static nature of an array, whose size remains constant.



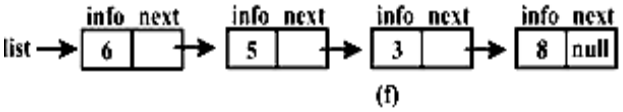


Fig 3: Adding an element to front of linked list

For example:
Suppose that we are given a list of integers as illustrated in fig 3(a) and we have to add the integer 6 to the front of that linked list.

- Assume the existence of mechanism for obtaining empty nodes as

  P = getNode ( );

  The operation obtains an empty node and sets the contents of a variable named P to the address of that node. The value of P is then a pointer to this newly allocated node.
- fig 3(b), shows the list and the new node after performing the get node operation. The next step is to insert the integer 6 into the info portion of the newly allocated node. This is done by info (P) = 6; (The result is shown in fig. 3(c))
- After setting the info portion of node (P), it is necessary to set the next portion of that node. Since node (P) is to be inserted at the front of the list, the node that follows should be the first node of the current list. Since the variable list contains the address of that first node, node (P) can be added to the list by performing the operation

  next (P) = list;

  [This operation places the value of list (which is the address of first node fn the list) into the next field of node(P)]
- At this point, P points to the list with the additional item included. However, since list is the external pointer to the desired list, its value must be modified to the address of the new first node of the list. This can be done by performing the operation –

  list = P;

Which changes the value of list to the value of P. Fig 3(e) illustrates the result of this operation.

- Generalized Algorithm
    1. P=getNode ( ); //Allocate memory for new node
    2. Info (P) =x;
    3. Next (P) =list;
    4. List=P;

1.2.1.2 Deletion

The following figure 4 shows the process of removing the first node of a nonempty list and storing the value of its info into a variable x. The initial step and final step is as shown in fig 4(a) and 4(f) respectively-

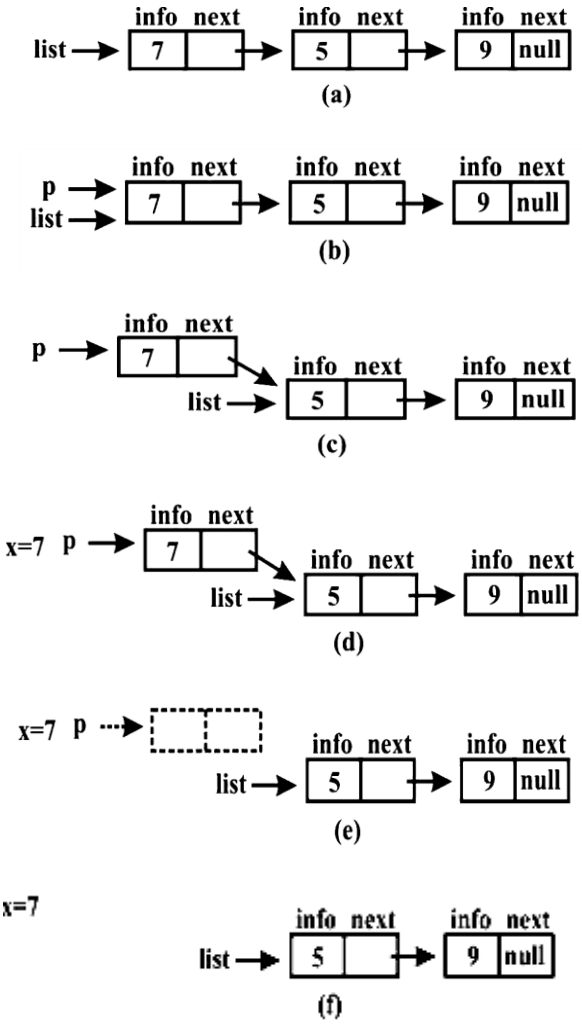


Fig 4: Removing a node from the front of the linked list

Note: The process is exactly opposite to the process of adding a node

During the process of removing the first node from the list, the variable p is used as an auxiliary variable. The starting and ending configuration of the list make no reference to P. But once the value of p is changed there is no way to access the node at all, since neither an external pointer nor a next field contains its address. Therefore the node is currently useless and cannot be reused. The get node creates a new node, whereas free node destroys a node.

Linked List as a Data Structure

Linked Lists are not only used for implementing stacks and queues but also other data structures. An item is accessed in a linked list by traversing the list from its beginning. A list implementation requires n operation to access the n^{th} item in a group but an array requires only single operation. Inserting or deleting an element in the middle of a group of other elements list is superior in an array.

For example:

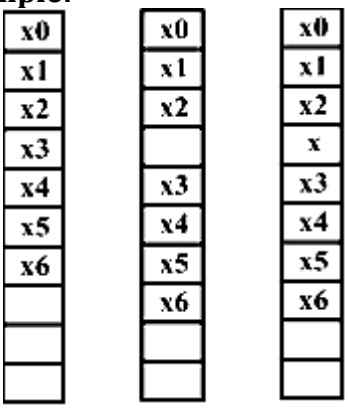


Fig 5: Insertion operation

We want to insert an element x between the 3^{rd} and 4^{th} elements in an array of size 10 that currently contains seven items (X [0] through X [6]).

Items 6 to 3 first are moved one slot and the new element is inserted in the newly available position 3 as shown in the fig 5.

- In this case insertion of one item involves moving four items in addition

to the insertion itself. If the array contained 500 or 1000 elements, a correspondingly larger number of elements would have to be moved. Similarly to delete an element from an array without any gap, all the elements beyond the element deleted must be moved one position.

- Suppose the items are stored as a list, if p points to an element of the list, inserting a new element involve allocating a node, inserting the information and adjusting two pointers. The amount of work required is independent of the size of the list as shown in fig 6.

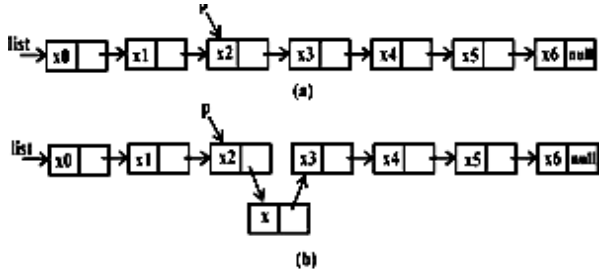


Fig 6: inserting a node

- An item can be inserted only after a given node, because there is no way to proceed from a given node to its predecessor in a linear list without traversing the list from its beginning i.e. one can only go forward for accessing the node of the list, not backward.
- For insertion of an item before desire node, the next field of its predecessor must be changed to point to a newly allocated node.
- To delete a node from a linear list it is insufficient for only one given pointer to that node. Since the next field of the node's predecessor must be changed to point to the node's successor and there is no directed way of reaching the predecessor of a given node.

1.2.2 OTHER LIST STRUCTURES

1.2.2.1 Circular Lists

- Let p be a pointer to a node in a linear list, but we cannot reach any of the nodes that precede node (p). If a list is traversed, the external pointer to the list must be preserved to be able to reference the list again.
  Suppose that a small change is made to the structure of a linear list, so that the next field in the last node contains a pointer back to the first node rather than the null pointer. Such a list is called a circular list
- Circular list is as shown in fig. 7(a), from any pointy in such a list it is possible to reach any other point in the list. If we begin at a given node and traverse the entire list, we end up at the starting point.

Note: Circular list does not have a natural "first" or "last" node. We must therefore, establish a first and last node by convention.

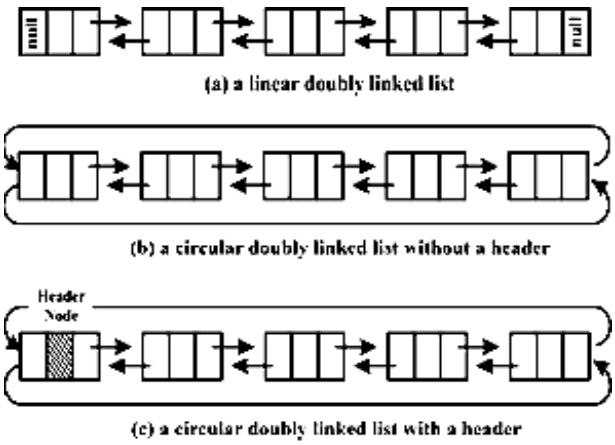


Fig 7: Circular linked list

One convention: Let the external pointer to the circular list point to the last node, and to allow the following node to be the first node as shown in fig. 7 (b). If p is an external pointer to a circular list, this convention allows access to the last node , this convention allows access to the last node of the list by referencing node(p) and

to the first node of the list by referencing node (next(p)).
A circular list can be used to represent a stack or queue.

1.2.2.2 Doubly Linked Lists

Although a circularly linked list has advantages over a linear list but still it has some drawbacks listed as follows:

- One cannot traverse such a list backward, nor can a node be deleted from a circularly linked list, given only a pointer to that node. In case where these facilities are required, the suitable data structure is a doubly linked list.
- In Doubly Linked Lists each node contains two pointers: one to its predecessor and another to its successor.
- Doubly linked list may be either linear or circular and may or may not contain a header node.
- Nodes on a doubly linked list consists of three fields
  (i) An info field that contains the value stored in the node.
  (ii) Left and right field contains pointers to the nodes on either side
- Dynamic implementation and array implementation of each node is as shown in fig. 8.

Array Implementation	**Dynamic Implementation**

Struct node type
 	
{int info;int left, right;
 	
};
 	
struct nodetype node[NUMNODES];
	Struct node {
	
	into info ;
	
	struct node *left, *right;
	
	};
	
	typeset struct node *NODEPTR;

Fig 8: Array and dynamic implementation

Note: In the array implementation the available list for such a set of nodes need not be doubly linked, since it is not traversed bidirectional. The available list must be linked together by using either a left or a right pointer.

2 STACK AND QUEUE

2.1 STACK

A stack is an ordered collection of items into which new items may be inserted or deleted only at one end, called the top of the stack.

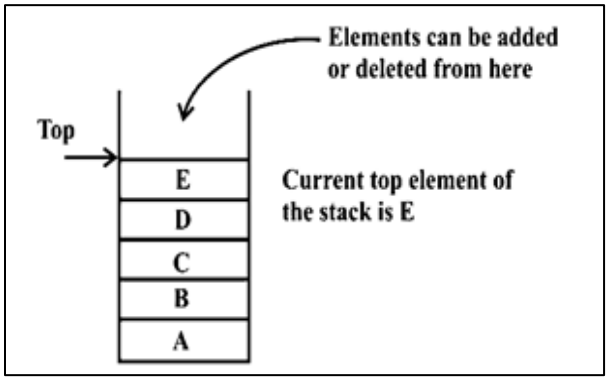


Fig 1: Stack containing elements

- A stack is different from an array as it has the provision for the insertion and deletions of items, thus a stack is a dynamic, constantly changing object.
- The last element inserted into a stack is the first element deleted. Thus stack is called a last-in, First-out (or-LIFO) list.

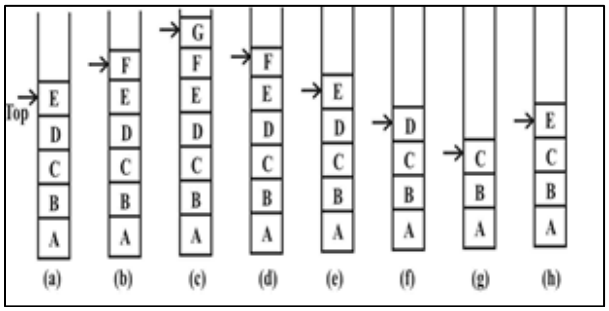


Fig 2: Motion picture of stack

- One cannot distinguish between frame (A) and frame (E) by looking at the stack's state at the two instances. No record is kept on the stack of the fact that items had been pushed and popped in the meantime.
- The true picture of a stack is given by a view from the top looking down, rather than from a side looking in. Thus there is no perceptible difference between frame (E) and (D). One can compare the stacks at the two frames by removing all the elements on both stacks and compare them individually.

2.1.1 OPERATIONS ON STACK

- When an item is added to a stack, it is pushed onto the stack.
- When an item is removed, it is popped from the stack.
- Given a stack s and an item i, performing the operation push (s, i) adds item i to the top of stack s.
- The operation i = pop(s); removes the element at the top of s and assigns its value to i.
- As push operation adds elements on to a stack, so a stack is sometimes called a pushdown list.
- There is no upper limit on the number of items that may be kept in a stack; though pop operation cannot be applied to the empty stack as such a stack has no elements to pop.
- The operation empty(s) determines whether stacks are empty or not. If the stack is empty, empty or not. If the stack is empty, empty(s) returns the value TRUE, or else it will return the value FALSE.
- The operation stack top(s) returns the top element of stack s. The stack top(s) operations can be decomposed into a pop and a push. This is also called as peek operation.

  i = stack top(s); is equivalent to

  i = pop(s);

  push (s, i);

The result of an illegal attempt to pop or access an item from an empty stack is called underflow	
Operation	Function
Push(s, i)	Adds item i on the top of stack s.
Pop(s)	Removes the top element of the stacks
Empty(s)	Determines whether or not a stack
Stack top(s)	Return the top element of stacks without popping it

stack top is not defined for an empty stack.

2.1.2 INFIX, POSTFIX AND PREFIX

- If the operator is situated in between the operands, then the representation is called infix.

  A + B → infix
- If the operator is preceding the operands, then the representation is called prefix.

  +AB → prefix
- If the operator is following the operands, then the representation is called postfix.

  AB+ → postfix
- Examples of Polish / Prefix notation
  1. (A + B) * C = (+ AB) * C = * + ABC
  2. A + (B * C) = A + (*BC) = +A * BC
  3. (A+B)/(C-D) = (+AB) / (-CD) = /+AB – CD
- Examples of Reverse polish / postfix Notation
  1. A $ B * C- D + E / F/ (G + H) = AB $ C * D – EF / GH +/+
  2. ((A + B) *C-(D-E)) $ (F + G) = AB +C * DE - - FG + $
  3. A –B /(C * D $ E) = ABCDE $ * / -
- The computer evaluates an arithmetic expression written in infix notation in two steps, i.e. first it converts the expression to postfix notation and then it evaluates the postfix expression.

2.1.3 Evaluation of a Postfix Expression

Let P be an arithmetic expression written in postfix notation. The following algorithm evaluates P using a STACK to hold operands.

Algorithm:

1. Add a right parenthesis")" at the end of P.
2. Scan P from left to right and repeat steps 3 and 4 for each elements of P until the ")" is encountered.
3. If an operand is encountered, push it on STACK.
4. If an operator is encountered, then
   (a) Remove the two top elements of STAC, where X is the top element and Y is the next to top element.
   (b) Evaluate Y operator X.
   (c) Place the result of
   (d) back on STACK.
   [End of loop]
5. Set VALUE equal to the element on STACK.
6. Exit.

Example of Above Concept:

Consider the following arithmetic expression P written in postfix notation.

P: 4,5,2, +, *, 16, 4, /, - [, is for separation it's not an operator]

Below is the content of STACK as each element of P is scanned.

Symbol Scanned	STACK
4	4
5	4,5
2	4,5,2
+	4,7
*	28
16	28,16
4	28,16,4
/	28,4
-	24
)	

The final number in STACK is 24, which is assigned to VALUE when ")" is scanned and thus is the final value of P.

2.1.4 Infix to Postfix Conversion

Let Q be an arithmetic expression written in infix notation, besides operands and operators, Q also contain left and right parentheses.
The following algorithm is used to transform the infix expression Q to its equivalent postfix expression P. It uses a stack to temporarily hold operators and left parentheses. The postfix expression P will be constructed from left to right using the operands from Q and the operators which are removed from STACK. Push a left parenthesis onto STACK and add a right parenthesis at the end of Q. The algorithm is completed when STACK is empty.

Algorithm:

1. Push "(" onto STACK, and add ")" to the end of Q.
2. Scan Q from left to right and repeat steps 3 to 6 for each elements of Q until the STACK is empty.
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered push it onto STACK
5. If an operator ⊗ is encountered, then:
   (a) Repeatedly pop from STACK and add to P, each operator (on the top of STACK) which has the same precedence or higher precedence than ⊗.
   (b) Add ⊗ to STACK
   [End of if structure]
6. If a right parenthesis is encountered, then:
   (a) Repeatedly pop form STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.
   (b) Remove the left parenthesis.
   [Do not add the left parenthesis to P]
   [End of If structure]
   [End of step 2 loop]
7. Exit

Example -
Consider the following arithmetic infix expression Q:
Q: A + (B * C-(D/E↑F)*G)*H
Following is the process of conversion of infix expressions into its equivalent expression P.

Symbol Scanned	STACK	Expression P
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
(	(*(-	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
↑	(+(-(/↑	ABC*DE
F	(+(-(/↑	ABC*DEF
)	(+(-	ABC*DEF↑/
*	(+(-*	ABC*DEF↑/
G	(+(-*	ABC*DEF↑/G
)	(+	ABC*DEF↑/G*-
*	(+*	ABC*DEF↑/G*-
H	(+*	ABC*DEF↑/G*-H
)		ABC*DEF↑/G*-H*+

After last step, the STACK is empty and postfix equivalent of Q is
P: ABC * DEF↑/G* -H * +

2.1.5 LINKED IMPLEMENTATION OF STACKS

The operation of adding an element to the front of a linked list is similar to that of pushing an element onto a stack. A new item is addressed as the only immediately accessible item in a collection –in both cases.

Note: A stack can be accessed only through its top element and a list can be accessed only from the pointer to its first element and the operation of removing the first element from a linked list is similar to popping a stack.

A stack may be represented by a Linear Linked List. The first node of the list represents the top of the stack.

If an external pointer s points to such a linked list, the operation push(s, x) is implemented as:

1. P=getNode ( );
2. Info (p) =x;
3. next (p)=s;
4. s=p;

Consider the following figure

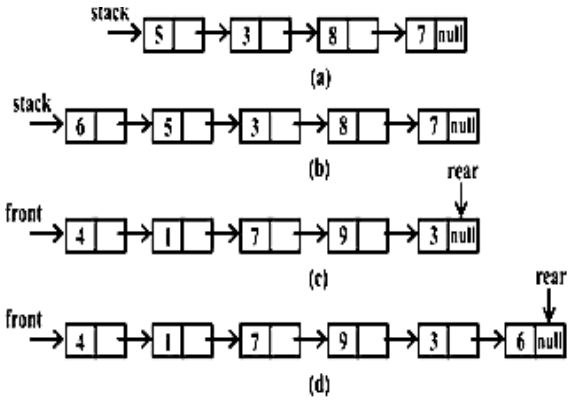


Fig 3: Stack and queue as linked list

Fig. 3(a) shows a stack implemented as a linked list fig. 3(b) shows the same stack after another element has been pushed onto it.

Advantages:

- All stacks being used by a program can share the same available list and when a stack needs a node, it can obtain it from the single available list.
- When a stack no longer needs a node, it returns the node to that same available list.
- As long as the total amount of space needed by all the stacks at any one time is less than the amount of space initially available to them, each stack is able to grow and shrink to any size
- No space has been pre-allocated to any single stack and no stack is using space that it does not need.

2.2 QUEUE

A queue is an ordered collection of items from which items may be deleted at one end called the front of the queue and may be inserted at the other end called the rear or the queue.
The first element inserted into a queue into a queue is the first element to be removed. Thus queue is called a FIFO (first – In – First – Out) list.

2.2.1 Operations on Queue

Operation	Function
Insert	Inserts item x at the rear of the queue q
x=remove(q)	Deletes the front elements from the queue q and sets x to its contents
Empty	Returns false on true depending on whether or not the queue contains any elements.

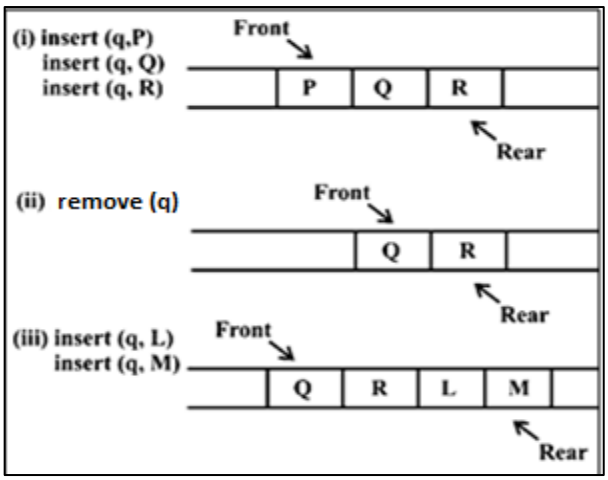


Fig. 4: Insertion of element in Queue

Example:

- As there is no limit to the number of elements a queue may contain, so an insert operation can always be performed. There is no way to remove an element from a queue containing no elements, thus remove operation can be applied only if the queue is non-empty.
- The result of an illegal attempt to remove an element from empty queue is called underflow

2.2.2 LINKED IMPLEMENTATION OF QUEUES

- We know that, items are deleted from the front of the queue and inserted at the rear. Let a pointer to the first element of a list represent the front of the queue and another pointer to the last element of the list represents the rear of the queue as shown in Fig. 5 illustrate the same queue after a new item has been inserted.

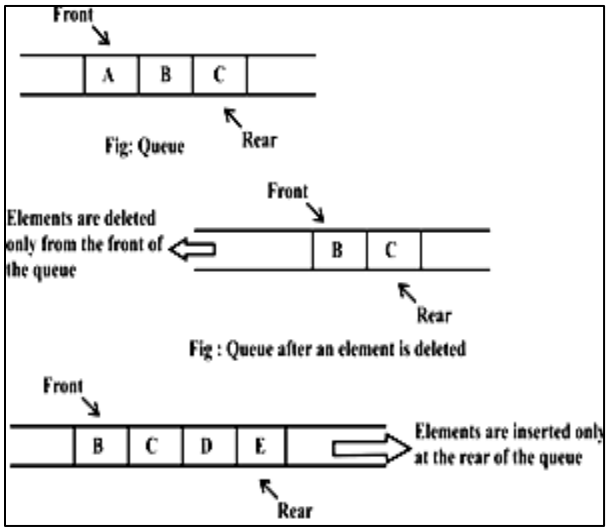


Fig 5: Queue after elements is inserted

- Under the list representation, a queue of consists of a list and two pointers q front and q rear. The operation empty (q) and x = remove (q) are completely similar to empty (s) and x = pop (s). with the pointer q front replacing s.
- Also, important consideration is required when the last element is removed from a queue. In that case, q rear must also be set to null, because in an empty queue both q front and q rear must be null.
- Insert (q, x) is same as adding an element x after the last element of list.

Disadvantages of representing a stack or queue by a linked list:

(i) A node in a linked list occupies more storage than a corresponding element in an array, since two pieces of information is needed in the array implementation.
[**Note:** The space used for a list node is usually not twice the space used by an array element, since the elements in such a list usually consist of structures with many subfields.]

(ii) Additional time is needed in managing of available list. Addition and deletion of each element from a stack or a queue involves a corresponding deletion or addition to the available list.

Advantages:

(i) All the stacks and queues of a program have access to the same free list of nodes.

(ii) Nodes not used by one stack may be used by another, as long as the total number of nodes in use at any one time is not greater than the total number of nodes available.

2.2.3 PRIORITY QUEUE

The result of its basic operation. The priority queue is a data structure in which the intrinsic ordering of the elements determine Type of priority queue

- An ascending priority queue is a collection of items into which items can be inserted arbitrarily and form which only the smallest item can be removed.
- A descending priority queue is collection of items into which can be inserted arbitrarily and it allows deletion of only the largest item.

2.2.3.1 Priority queue has following rules:

- An element of higher priority is processed before any element of lower priority.
- Two elements with the same priority are processed according to the order in which they were added to the queue.

2.2.3.2 Prototype of Priority Queue:

- A prototype of a priority queue is a timesharing system: a program of high priority is processed first and programs with the same priority form a standard queue.

2.2.4 Representation of a Priority Queue:

There are basically 2 different representations of priority queue:

1. One way list representation of a priority queue
2. Array Representation of a Priority Queue
3. List implementation of Priority Queues

2.2.4.1 One way list representation

(a) Each node in the list will contain there items of information: an information field INFO, q priority numbers PRN and a link number LINK.

(b) A node x precedes a node y in the list
   (i) When x has higher priority than y or
   (ii) When both have the same priority but x was added to the list before y.

Thus the order in the one-way list corresponds to the order of the priority queue.

2.2.4.2 Array Representation

Another way to maintain a priority queue in memory is to use a separate queue for each level of priority (or for each priority number). Each such queue will appear in its own circular array and must have its own pair of pointers. FRONT and REAR. In fact, if each queue is allocated the same amount of space, a two-dimensional array QUEUE can be used instead of the linear arrays. A indicates this representation for the priority queue in b. Observe that FRONT[K] and REAR[K] contain, respectively, the front and rear elements of row K of QUEUE, the row that maintains the queue of elements with priority number K.

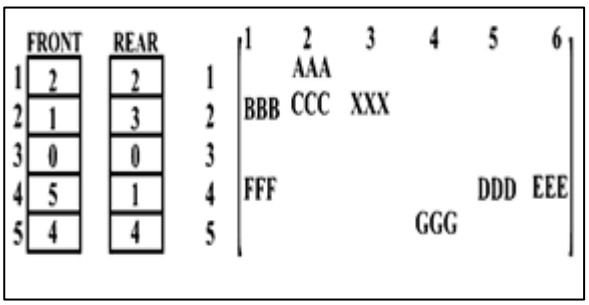


Fig 6: Array implementation of Priority Queue

2.2.4.3 List implementation

- An ordered list can be used to represent a priority queue. For an ascending priority queue -
  (i) Insertion is implemented by place operation which keeps the list ordered.
  (ii) Deletion of the minimum element is implemented by keeping the list in descending rather than ascending order. A priority queue implemented as an ordered linked list requires examining an average of approximately n/2 nodes for insertion but only one node for deletion.
- An unordered list may also be used as a priority queue. Such a list require examining only one node for insertion but always requires examining n elements for deletion traverse the entire list to find the minimum or maximum and hen delete that node.
- Thus an ordered list is somewhat more efficient than an unordered list in implementing a priority queue.
- **Advantages:** List is somewhat more efficient than an array in implementing a priority queue.
  (i) No shifting of elements i.e. gaps is necessary in a list
  (ii) An item can be inserted into a list without moving any other items, whereas this is impossible for an

array unless extra space is left empty.

2.3 DEQUEUE

A de-queue is a linear list in which elements can be added or removed at either end but not in the middle.

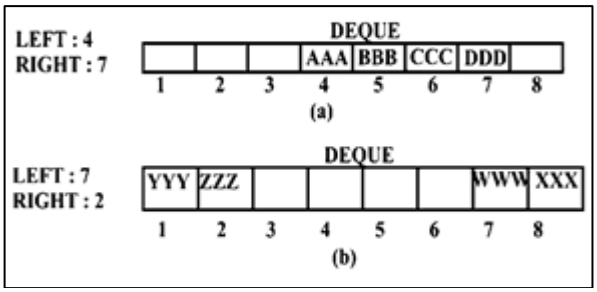


Fig 7: Pointers in de-queue

- De-queue is maintained by a circular array DEQUE with pointer LEFT and RIGHT, point to the two ends of the de-queue.
- The condition LEFT = NULL will be used to indicate that a de-queue is empty.
- An input-restricted de-queue allows insertion at only one end of the list but allows deletion at both ends of the list.
- An output-restricted deque allows deletion at only one end of the list but allows insertion at both ends of the list.

2.4 Abstract Data Types

A useful tool for specifying the logical properties of a data types is the abstract data type, or ADT. Fundamentally, a data type is a collection of values and a set of operations on those values. That collection and those operations from a mathematical construct that may be implemented using a particular hardware or software data structure. The term "abstract data type" refers to the basic mathematical concept that defines the data type.

In defining an abstract data type as a mathematical concept, we are not concerned with space or time efficiency. Those are implementation details. It may not concern to implement a particular ADT on a particular piece of hardware or using a particular software system.

For example, we have already seen that the ADT integer is not universally implementable. Nevertheless, by specifying the mathematical and logical properties of a data type or structure, the ADT is useful guideline to implementers and a useful tool to programmers who wish to use the data type correctly.

2.4.1 Queue as an abstract Data type

The representation of a queue as an abstract data type is straight forward. We use ectype to denote the type of the queue element and parameterize the queue type with ectype.

- Abstract typeset <<ectype>> QUEUE (ectype):
- Abstract empty (q);
- QUEUE (ectype) q;
- Post condition empty = = (Len (q) = 0);
- Abstract ectype remove (q);
- QUEUE (ectype) q;
- Precondition empty (q) = = FALSE;
- Post condition remove = = first (q`);
- q = = sub (q`, 1, len(q`)-1);
- abstract insert (q, elt);
- QUEUE (ectype) q;
- ectype elt;
- Post condition q = = q` + <elt>;

2.5 PERFORMANCE ANALYSIS

The space complexity of an algorithm is the amount of memory it needs to run to completion. The time complexity of an algorithm is the amount of computer time it needs to run to completion. Time Complexity: The time T(p) taken by a program p is the sum of the compile time and the run (or execution) time. The compile time does not depend on the instance characteristics. The runtime is denoted by T_P (instance characteristics).

To obtain such time at instance, we need to know that how many computations i. e. additions, multiplication, subtraction,

divisions performed by an algorithm. But still obtaining correct formula for each algorithm is impossible task, since the time needed for an addition, subtraction, multiplication and so on, often depends on the numbers being added, subtracted, and multiplied and so on. The value of Tp (n) for any given n can be obtained only experimentally. The program is typed, complied and run on a particular machine. The execution time is physically clocked and Tp(n) is obtained.

We can determine the number of steps needed by a program to solve a particular problem instance in one of two ways-

(i) In the first method, we introduce a new variable count into the program. This is a global variable with initial value 0. Statements to increment count by the appropriate amount are introduced into the program. This is done so that each time a statement in the original program is executed; count is incremented by the step count of that statement.

(ii) The second method to determine the step count of an algorithm is to build a table in which we list the total number of steps contributed by each statement. This number is often arrived at, by first determining the number of steps per execution (s/e) of the statement and the total number of times (i. e. frequency) each statement is executed. The s/e of a statement is the amount by which the count changes as result of the execution of that statement. By combining these two quantities the total contribution of each statement is obtained. By adding the contributions of all statements, the step count for entire algorithm is obtained.

2.6 ASYMPTOTIC NOTATION (O, Ω, θ)

(1) Big oh (O)

The function f(n) = O(g(n)) (read as f of n is big oh of g of n) iff exists positive constants c and n_0, such that $f(n) \leq c^* g(n)$ for all c, $n \geq n_0$.

For Example

1. The function $3n + 2 = O(n)$ as $3n + 2 \leq 4n$ for all $n \geq 2$.
2. The function $3n + 3 = O(n)$ as $3n + 3 \leq 4n$ for all $n \geq 3$.
3. The function $3n + 2 \neq O(1)$ as $3n + 2$ is not less than or equal to c for any constant c and all $n \geq n_0$.

- We write O (1) to mean a computing time that is a constant, Similarly -
  - O(n) is called linear
  - $O(n^2)$ is called quadratic
  - $O(n^3)$ is called cubic
  - $O(2^n)$ is called exponential

- If an algorithm takes time O(log n), it is faster, for sufficiently large n, than if it had taken O(n). Similarly O(n log n) is better than $O(n^2)$ but not as good as O(n).

- From the definition of O(Big-oh), it should be clear that f(n) = O(g(n) is not the same as O(g(n)) = f(n).

- If $f(n) = a_m n^m + \ldots\ldots + a_1 n + a_0$, then $f(n) = O(n^m)$

  Proof:

  $$F(n) \leq \sum_{i=0}^{m} |a_i| n^i$$

  $$\leq n^m \sum_{i=0}^{m} |a_i| n^{i-m}$$

  $$n^m \sum_{i=0}^{m} |a_i| \quad \text{for } n \geq 1$$

  so $f(n) = O(n^m)$

2.6.1 Omega (Ω)

The function f(n) = Ω(g(n)) (read as f of n is omega of g of n) if there exist positive

constant C and n_0 such that $f(n) \geq C * g(n)$ for all $n, n \geq n_0$.

For example:

1. The function $3n + 2 = \Omega(n)$ as $3n+2 \geq 3n$ for $n \geq 1$ the inequality holds for $n \geq 0$, but the definition of Ω requires an $n_0 > 1$.
2. The function $3n + 3 = \Omega(n)$ as $(3n+3) \geq 3n$ for $n \geq 1$.
3. The function $3n + 3 = \Omega(1)$

- As in the case of big-oh notation, there are several functions g(n) for which $f(n) = \Omega(g(n))$. The function g(n) is only a lower sound on f(n).
- For the statement $f(n) = \Omega(g(n))$ to be informative g(n) should be as large as possible function of n for which the statement $f(n) = \Omega(g(n))$ is true.
- So, when we say that $3n + 3 = \Omega(n)$ $6 \times 2^n + n^2 = \Omega(2^n)$ we almost never say that $3n + 3 = \Omega(1)$ or $6 \times 2^n + n^2 = \Omega(1)$ even though both of these statements are correct.
- if $f(n) = a_m n^m + \ldots\ldots\ldots + a_1 n + a_0$ and $a_m > 0$ then $f(n) = \Omega(n^m)$.

2.6.2 Theta (θ)

The function $f(n) = \theta(g(n))$ (read as f of n is theta of g of n) iff there exist positive constant C_1, C_2, and h_o such that $C_1 g(n) \leq f(n) \leq C_2 g(n)$ for all $n, n \geq n_0$.

For example

1. The function $3n + 2 = \theta(n)$ as $3n + 2 \geq 3n$ for all $n \geq 2$ & $3n+2 \leq 4n$ for all $n \geq n$, so $C_1 = 3, C_2 = 4$ and $n_0 = 2$
2. The function $3n + 3 = \theta(n)$

- The theta notation is more precise both the big-oh and omega notations. The function $f(n) = \theta(g(n))$ if g(n) is both on upper and lower bound on f(n).
- If $f(n) = a_m n^m + \ldots\ldots\ldots + a_1 n + a_0$ and $a_m > 0$ then $f(n) = \theta(n^m)$

2.6.3 Little "Oh" (o)

The function $f(n) = o(g(n))$ (read as f of n is little oh of g of n) iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

For example:

1. The function $3n + 2 = o(n^2)$ since $\lim_{n \to \infty} \frac{3n+2}{n^2} = 0$
2. The function $6 \times 2^n + n^2 = o(3^n)$

2.6.4 Little "omega" (ω)

The function $f(n) = \omega(g(n))$ (read as f of n is little omega of g of n) iff $\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$

3 SORTING & SEARCHING

3.1 SORTING

Sorting refers to the operation of arranging data in some given order, such as increasing or decreasing with numerical data or alphabetically with character data. A file of size b is a sequence of n items r[0], r[1],.....[n – 1]. Each item in the file is called a record.

A sort is internal if the records that it's sorting are in main memory or external if some of the records that it's sorting are in auxiliary storage.

A sorting technique is called stable if for all records i and j such that k[i] equals k[j], if r[i] precedes r[i] in the original file, r[i] precedes r[j] in the sorted file.

Efficiency Consideration:

There most important things that determine the efficiency of sorting are:

1. Amount of time that must be spent by the programmer in coding a particular sorting program.
2. The amount of machine time necessary for running the program.
3. The amount of space necessary for the program.

3.2 SORTING ALGORITHMS

3.2.1 BUBBLE SORT

In bubble sort, pass goes through the file sequentially, several times. Each pass consists of comparing each element in the file with its successor (x[i] with x[i + 1]) and interchanging the two elements if they are not in proper order.

Algorithm
Given Array `a` of `n` integers [0.....n-1]
Aim : To sort `a` in ascending order.

Working:

Bubble sort uses two counter i and j such that i begins with (n -2) and decrements till to 0. For every value of i, j begins with 0 and goes upto i.

Example:

If n = 5, then i ranges from (n-2) down to 0 i.e. 3 down to 0 and for each value of i, j varies from 0 to i.

```
For (i=n-2; i>=0; i--)

For (j=0; j<=i; j++)
```

For each value of j we check whether (a[j] > a [j + 1]) and if true, exchange a[j] with a [j + 1]

Hence

```
for(i= n -2; i>=0;i--)
        for (j = 0; j <= i; j++)
                if(a[j] > a[j + 1] )
                {
                        t = a[j];
                        a[j] = a[j + 1];
                        a[j + 1] = t;
                }
```

Let us show working with diagram.
Let 'a' be array of 5 elements i. e. n = 5.

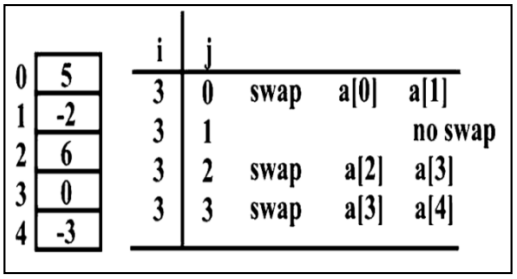


Fig 1: Array during pass I

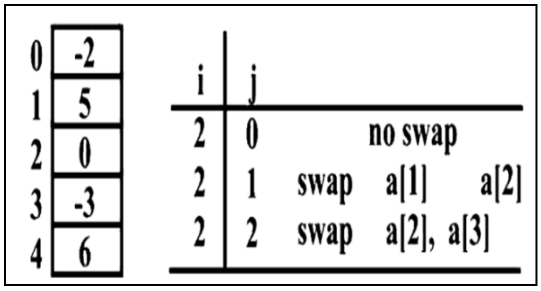


Fig 2: Array after pass I

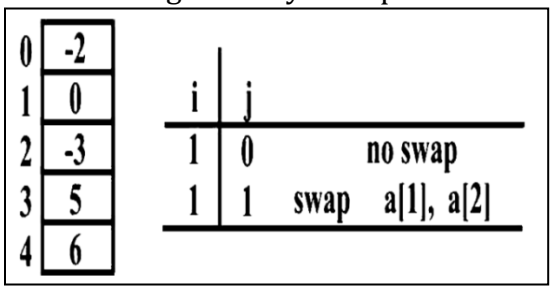


Fig 3: Array after pass II

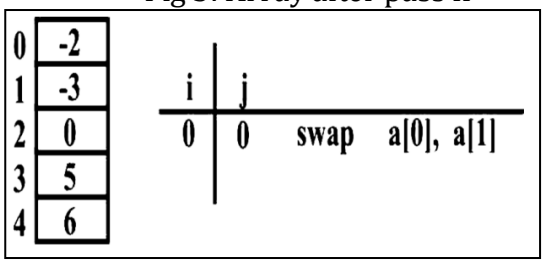


Fig 4: Array after pass III

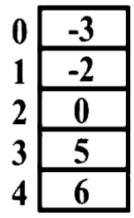


Fig 5: Array after pass IV

Note: Bubble sort requires (n – 1) passes to sort array of `n` elements.

Example:
Consider the following file
25, 57, 48, 37, 12, 92, 86, 33

1. First pass:

X[0] with x[1]	(25 with 57)	No interchange
X[1] with x[2]	(25 with 57)	Interchange
X[2] with x[3]	(25 with 57)	Interchange
X[3] with x[4]	(25 with 57)	Interchange
X[4] with x[5]	(25 with 57)	No interchange
X[5] with x[6]	(25 with 57)	Interchange
X[6] with x[1]	(25 with 57)	Interchange

∴ After the first pass, the file is in the order
25, 48, 37, 12, 57, 86, 33, 92

2. After second pass:
   25, 12, 37, 48, 57, 33, 86, 92
3. After Third pass:
   25, 12, 37, 48, 33, 57, 86, 92
4. After Fourth pass:
   12, 25, 37, 33, 48, 57, 86, 92
5. After Fifth pass:
   12, 25, 33, 37, 48, 57, 86, 92
6. After Sixth pass:
   12, 25, 33, 37, 48, 57, 86, 92
7. After Seventh pass:
   12, 25, 33, 37, 48, 57, 86, 92

- After the n^{th} pass, the n^{th} largest element is in its proper position within the array.
- The sorting method is called bubble sort because each number slowly "bubbles" up to its proper position.
- As each iteration places a new elements into its proper position, a file of n elements requires not more than (n – 1) iterations.
- As all the elements in positions greater than or equal to (n – i) are already in proper position after the i^{th} iteration, they need not be considered in succeeding iterations. If the file can be sorted in fewer than (n -1) passes, the final pass makes no interchanges.

3.2.1.2 Comparisons

On the first pass (n – 1) comparisons are made, on the second pass (n – 2) comparisons and on the $(n-1)^{th}$ pass only one comparison is made. Thus there are (n

– 1) passes and (n – 1) comparisons on each pass.
Total number of comparison=(n–1)*(n -1)
$= n^2 – 2n + 1$
$= O(n^2)$

Note: It is the number of interchanges rather than the number of comparisons that takes up most time in the program`s execution.
Now as all the elements in positions greater than or equal to (n– i) are already in proper position after iteration i, so they need not be considered in succeeding iterations. Thus number of comparisons on iteration i is (n – i). If there are k iterations, the total number of comparisons is -
$=(n-1)+(n-2)+(n-3)+\cdots+(n-k)$
$= kn - [1+2+3+4 \ldots\ldots\ldots\ldots\ldots\ldots k)]$
$= kn - [k(k+1)/2]$
$= \frac{2nk-k^2-k}{2}$
Thus bubble sort can be speed up by considering (n – 1) comparisons on the first pass, (n– 2) comparisons on the second pass and only one comparison on (n – 1)th pass.

3.2.2 QUICK SORT

Quick sort works on Divide and Conquer policy. It is also called partition Exchange Sort.

Divide:
The array X[P…r] is partitioned into two non-empty sub arrays X[p…q] and X[(q+1) …. r]. The index q is computed as part of this partitioning procedure.

Conquer:
The two sub arrays X[p….q] and X[(q + 1)….r] are sorted by recursive calls to quick sort

Example:
Consider the following unsorted array
34, 42, 67, 82, 90, 17, 12

Quicksort (pivot, i, j)

1. Increment the value of i till Pivot > a[i]
2. Decrement the value of j till Pivot ≤ a[j]
3. If i and j are not crossing each other, then interchange the values of i and j.
4. If i and j are crossing each other, then interchange the value of j with pivot.
5. Interchange of j with pivot, divide the array into two unsorted array, one at the left of pivot and one at the right of pivot.
6. Apply the same rule on left and right arrays to sort them and then merge the two to get final sorted array.
7. Example -
   (a) **34**, 12, 67, 82, 90, 17, 42
   (b) **34**, 12, 17, 82, 90, 67, 42
   (c) 17, 12, **34**, 82, 90, 67, 42
   Apply Quicksort (pivot, i, j) on Right sub-array 82, 90, 67, 42
   (d) 82, 42, 67, 90
   (e) 67, 42, 82, 90
   (f) 42, 67, 82, 90
   Apply Quicksort (pivot, i, j) on Left sub-array 12, 17
   Merge left and right sorted sub-arrays
   12, 17, 34, 42, 67, 82, 90

Note: It is not necessary to choose the first elements as pivot, one can choose any item as Pivot.

3.2.2.1 EFFICIENCY

Assume array size is n as a power of 2 i. e. $n = 2^m$, so that $m = \log_2 n$. Also assume proper position for the pivot always is the middle of the sub array.
Thus there will be n (actually n – 1) comparisons on the first pass, after which the file is split into two sub array each of size n/2. For each of these two arrays, there are n/2 comparisons approximately and thus a total of 2(n/2) comparisons. So after halving the sub array m times, there are n arrays of size 1.
Thus the total number of comparisons for the entire sort is approximately:
= n + 2*(n/2) + 4*(n/4) + 8*(n/8) + ……. + n* (n/n)

= n + n + n+……+n (m times)
Thus total number of comparisons is O(n * m) or O(n log n).

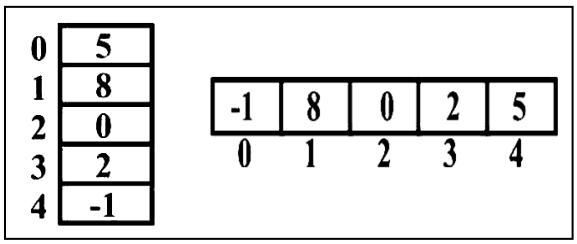
3.2.3 SELECTION SORT

In selections sort successive elements are selected in order and placed into their proper sorted position. Selection sort consists of a selection phase in which the largest of the remaining element is repeatedly placed in its proper position, i. e. at the end of the array. This largest elements is interchanged with the element at the end of the array.

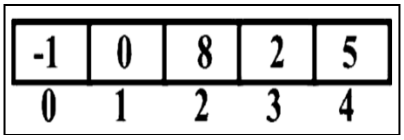
Thus the initial n element priority queue is reduced by one element after each selection. Thus after (n – 1) selections the entire array would be sorted as selection process needs to be done only from (n – 1) down to 1 rather than down to 0.
This method also requires (n – 1) passes for sorting array of n elements. For Example, to sort array of 5 elements, selection sort needs 4 passes. In pass number `i`, selection sort finds minimum elements between a[i] and a[n–1] (i. e. from position i to n–1) and then interchange the minimum element with a[i].

Let us show working of selection sort using diagrams. Let `a` be array of 5 integers

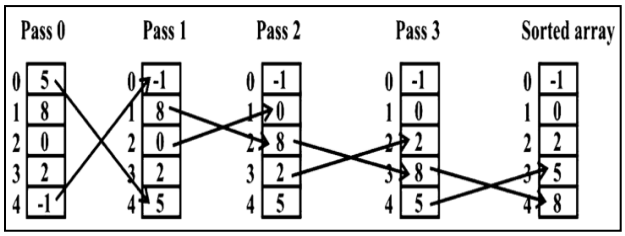
Pass 0: Check minimum element between a[0] to a[4], which is a[4] = -1
Interchange this minimum with a[0]
∴ The array after pass 0 is



Pass 1: Check minimum elements between a[1] to a[n-1] which is a[2]=0 and interchange this with a[1]. The array after pass 1 is-



Similarly all passes can be shown as follows-



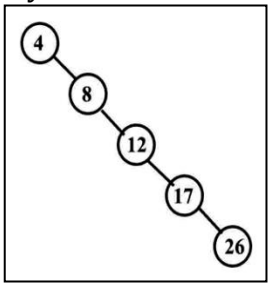
3.2.4 BINARY TREE SORT

Binary tree sort uses a binary search tree. It involves scanning each element of the input file and placing it into its proper position in a binary tree. To find that proper position of an element `a`, a left or a right branch is taken at each node, depending on whether a is less than the element in the node or greater than or equal to it.

As soon as the input elements are in their proper position in the tree, the sorted array can be retrieved by an in order traversal of the tree. This has two phases. First phase is creating a binary search tree using the given array elements. Second phase is to traverse the created binary search tree in order thus resulting in a sorted array.

3.2.4.1 Performance of the algorithm:

The average number of comparisons for this method is O(n $\log_2$ n). but in the worst case, the number comparisons requires is O(n^2), a case which arises when the sort tree is severely unbalanced.



e. g., Original data: 4, 8, 12, 17, 26.
In this case the insertion of the first node requires no comparisons, the second node requires two comparisons, and soon. Thus the total number of comparisons is

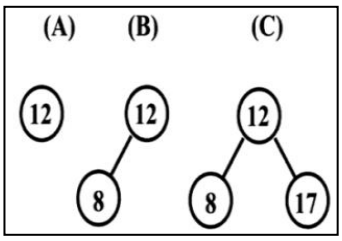
$$2 + 3 + \ldots + n = \frac{n*(n+1)}{2} - 1 = O(n^2)$$

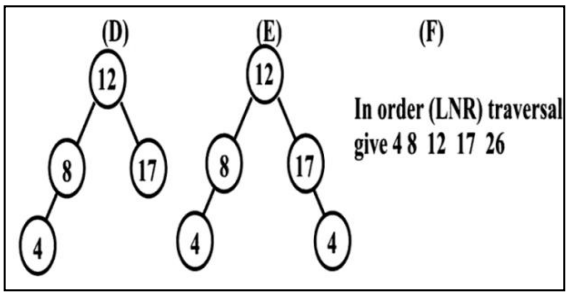
Example:
Consider the following unsorted array

(i) Construct a binary tree
(a)Elements on left of node are smaller
(b)Elements on the right of node are greater

(ii) Perform in order transversal of the tree to get sorted array.

(iii)





3.2.4.2 Comparison

If original array is completely sorted (or sorted in reverse order), then insertion of the first node requires no comparisons, the second requires two comparisons, the third node requires three comparisons and so on. Thus the total number of comparison is

$$2 + 3 + \ldots + n = \frac{n*(n+1)}{2} - 1 = O(n^2)$$

3.2.5 HEAP SORT

Binary heap data structure is an array those object that can be viewed as a complete binary tree.

A heap is binary tree satisfying the property: Every node has a greater value than its child node. For a given unsorted array, when a heap is formed, the element at the root node is the largest element and it is removed from the heap and placed at the end of the array. The element at the end of array which has been replaced by root node is placed at the root node and again with shuffling a new heap is formed and the same process is repeated. At the end we are left with only one element in the heap and complete sorted array.

The drawbacks of the binary tree sort are remedied by the heap sort, an in place sort that requires only O(n log n) operations regardless of the order of the input.

3.2.5.1 Heap (or descending heap):

A heap (descending heap) of size n can be defined as an almost complete binary tree of n nodes such that the contents of each node is less than or equal to the content of its father. If the sequential representation of an almost complete binary tree is used, this condition reduces to the inequality.

$$A[j] \le A[j \text{ div } 2] \text{ for } 1 \le (j \text{ div } 2), j \le n.$$

It is clear from this definition of a descending heap that the root of the tree (or the first element of the array) contains the largest element in the heap. We now formulate an algorithm which will have as input an unsorted array and produce as output a heap

3.2.5.2 Algorithm for creating a heap

1. Repeat through step 7 while there is another element to be placed in the heap.
2. Obtain child to be placed at leaf level.
3. Obtain position of parent for this child.
4. Repeat though step 6 while the child has a parent and the value of the child is greater than that of its parent.
5. Move the parent down to position of child.

6. Obtain position of new parent for the child.
7. Copy child element into its proper position.

3.2.5.3 General algorithm for the heap sort

1. Create the initial heap.
2. Repeat through step 8 for n - 1 times.
3. Exchange the first element with the last unsorted element.
4. Obtain the index of the largest son of the new element.
5. Repeat through step 8 for the unsorted element in the heap and while the current element is greater than the first element.
6. Interchange the elements and obtain the next left son.
7. Obtain the index of the next left son.
8. Copy of the element into its proper place.

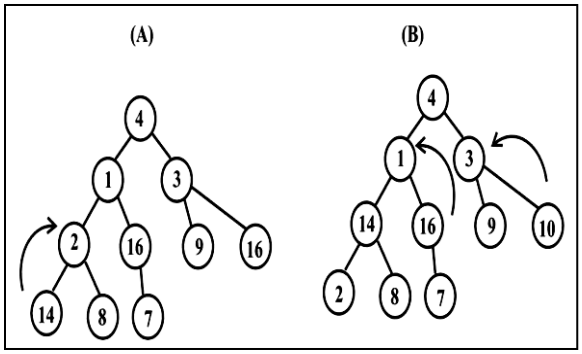
Consider the timing analysis of the heap sort. Since we are using a complete binary tree, the worst case analysis is easier than the average case. Note that depth of a complete binary tree of n nodes is $\log_2 n$. Recall that to sort a given array we must first create a heap and then sort that heap. The worst case at each step involves performing a number of comparisons which is given by the depth of the tree. This observation implies that the number of comparisons is $O(n \log_2 n)$. Average case behaviour is also $O(n \log_2 n)$. Also no extra working storage area, except for one element position, is required. That is why it is known as an in place sorting technique.

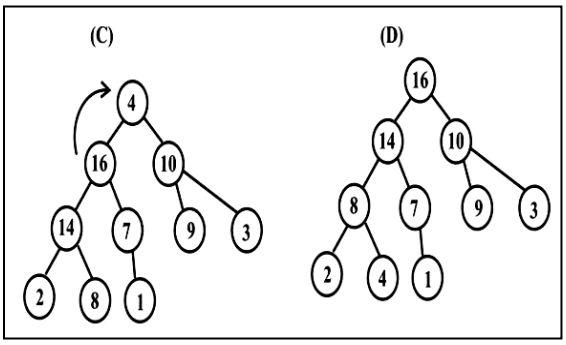
Example:

Consider following unsorted array

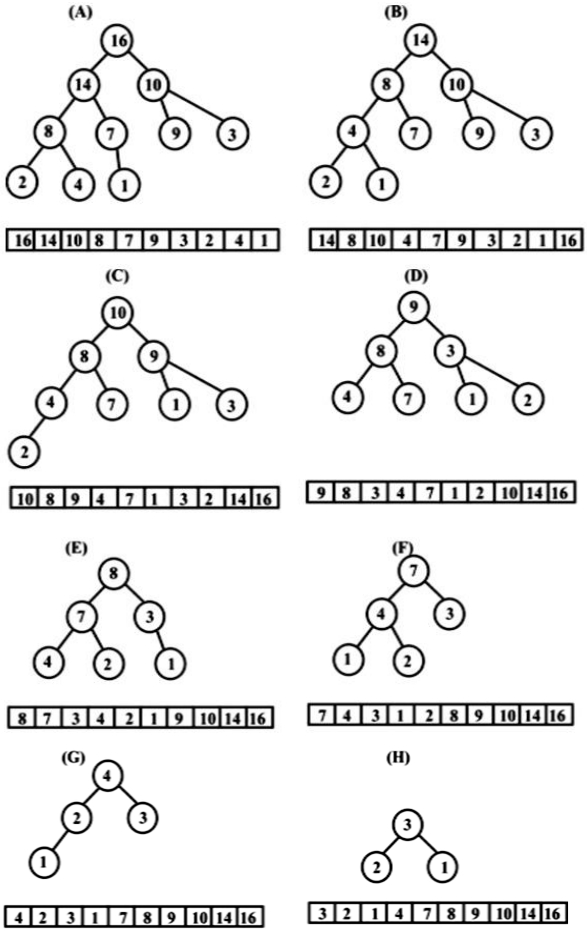
4	1	3	2	16	9	10	14	8	7
1	2	3	4	5	6	7	8	9	10

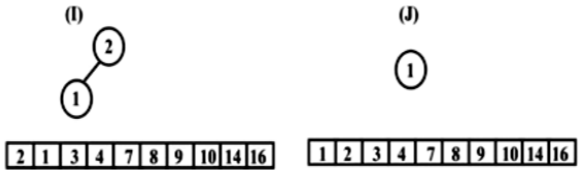
First construct heap tree as:





Sorting being as:



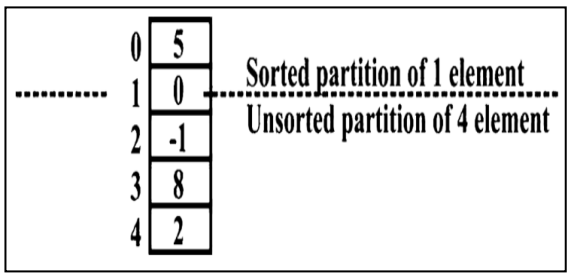


3.2.6 INSERTION SORT

Insertion sort places the largest key in its proper position. This is done by looking at all the elements in the list and pushing the largest key down the list, till its position is found.

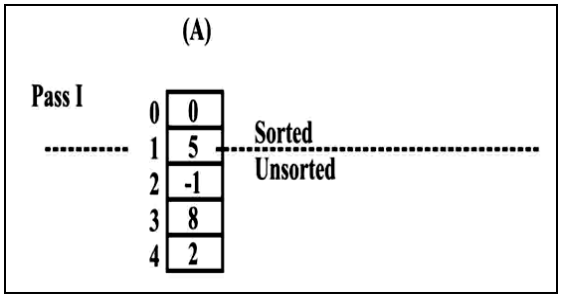
An insertion sort is one that sorts a set of records by inserting records into an existing sorted file. This requires (n – 1) passes to sort an array of n elements. It starts with an assumption that the array is divided into sorted partition of length 1 and unsorted partition of length (n -1)

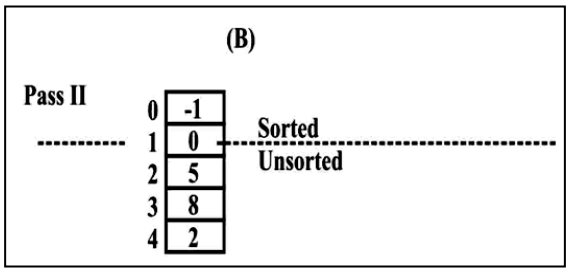
Example:
Consider the following unsorted array 50-182
For the above 5 integers, sorted and unsorted partitions can be shown as

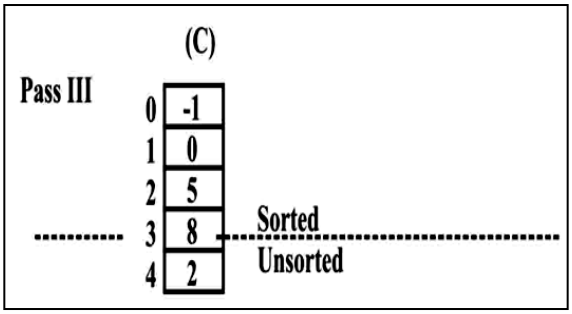


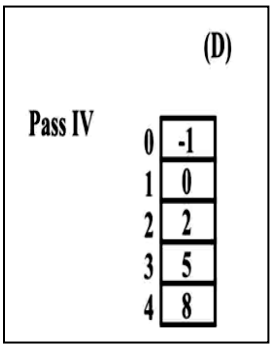
In each pass, insertion sort picks next element of unsorted partition and inserts i in right place in sorted part. During this process greater elements are pushed down to make a place for element getting inserted.

Thus we pick a[1] and insert it in the right place in the sorted part. During this process a[0] is pushed down to a[1].









3.2.6.1 Disadvantage:

Even after most items have been sorted property into the first part of the list, the insertion of a later item may require that many of them to the moved.

3.2.6.2 Comparison:

Worst case:
If the input array is in reverse order, the number of comparisons required to push down the heavies element is 2 time when i

is 2, 3 times when i is 3. Precisely test repeats i times (i range from 2 to n.)

$$\therefore \sum_{i=2}^{N} i = 2 + 3 + ... + N = \frac{N(N+1)}{2} - 1 = O(n^2)$$

In the best case (i. e. if input is already sorted) the running time is O(n) because the test to push down heavier element always fails.

3.2.7 SHELL SORT

Shell sort works on comparative basis. It works very much the way bubble sort works except that it does compare alternate elements, rather it makes comparison with the elements at fixed distance.

Shell sort is also known as diminishing increment sort and is a modification of the insertion sort. The algorithm starts with comparing the elements at distance `d` and swap them if needed. The process is repeated by reducing the distance d.

If there are unsorted elements, then we will have initial distance d = 8 / 2 = 4. So do the inter change.

For the next pass,

D=previous d/2 = 4 /2 = 2

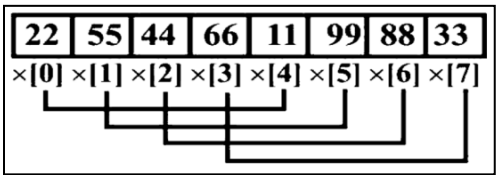
Thus now we need to compare (0, 2) (1, 3), (4, 6) and (5,7). The process is repeated till d is greater than one.

Example:

Consider the following unsorted array:

22	55	44	66	11	99	88	33
×[0]	×[1]	×[2]	×[3]	×[4]	×[5]	×[6]	×[7]

Array has 8 elements, so d = 8/2 = 4.



×[0] is compared with × [4]	22 with 11	Interchange
×[1] is compared with × [5]	55 with 99	No Interchange
×[2] is compared with × [6]	44 with 88	No Interchange
×[3] is compared with × [7]	66 with 33	Interchange

11	55	44	33	22	99	88	66
×[0]	×[1]	×[2]	×[3]	×[4]	×[5]	×[6]	×[7]

For the second pass, d=previous d/2=4/2= 2

×[0] is compared with × [2]	11 with 44	No Interchange
×[1] is compared with × [3]	55 with 33	Interchange so× 1 [1]=33, ×[3]= 55
×[2] is compared with × [4]	44 with 22	Interchange so× 1 [1]=33, ×[3]= 55
×[3] is compared with × [5]	55 with 99	No Interchange
×[4] is compared with × [6]	44 with 88	No Interchange
×[5] is compared with × [7]	99 with 66	Interchange so× 1 [5]=66, ×[7]= 99

Thus second pass would give

11	33	22	55	44	66	88	99

For the third pass d = a. So adjacent elements will be compared, thus it would give.

11	33	22	55	44	66	88	99

3.2.8 ADDRESS CALCULATION SORT

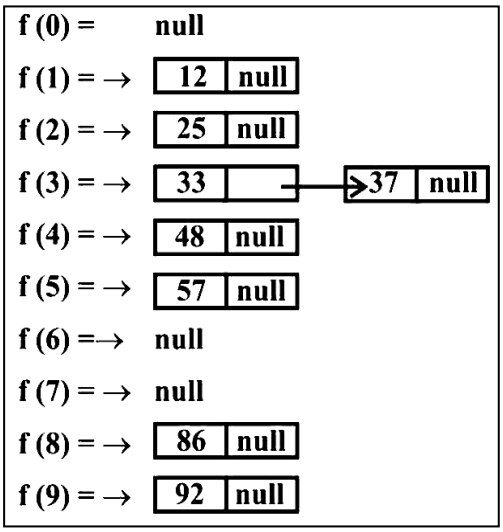
In address calculation sort, a function f is applied to each key. Function result determines, into which of several sub files the record is to be placed. The property of function should be such that if $x \leq y$, then

f(x) ≤f(y). This function is called order preserving

All of the records in one sub file will have keys that are less than or equal to the key of the records in another sub file. Any of the sorting method (Simple insertion is used) can be used to place into a sub file in correct sequence. When all the items into a sub file in correct sequence. When all the items of original file have been placed into sub files, the sub files are concatenated to produce the sorted result.

Example:
Consider the following unsorted file 25, 57, 37, 12, 92, 33

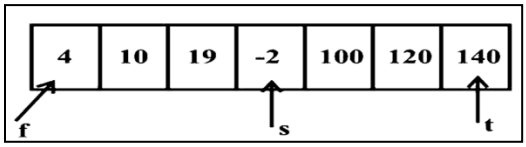
We will create ten sub files, one for each of the ten possible first digits. Initially each of the sub files is empty. An array of pointers f[10] is declared, where f[i] points to the first element in the file whose first digit is i. The first element (25) is placed into the file head by f [2]. Each of the sub files is maintained as a sorted linked list of the original array elements.
Thus the sub file would appear as:



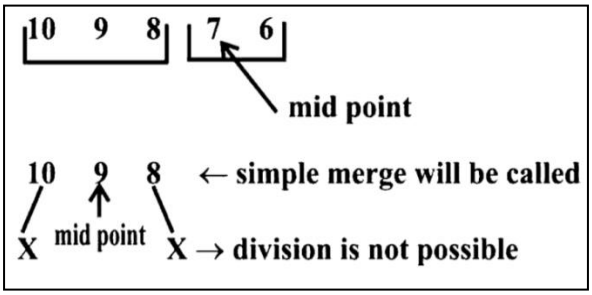
3.2.9 MERGE SORT

Merge sort works on the principle of divide and conquer technique. It divides the list into two sub lists of almost equal sizes. Continue dividing the list till sub list reduces to unit length and then merge the sub lists which will be in sorted order.

The idea of merge sort comes the case of merging two sub arrays which are already sorted in increasing order. For e.g.



The array A can be considered as two sub arrays ranging from f to s – 1 and s to t. Both these sub arrays are already sorted. These two sub arrays can be merge into a single array say `C` such that C contains all the element of array A in increasing order. We may extend the idea of two sub arrays to `n` sub arrays such that each sub arrays is sorted in increasing order. For example, consider an array of size n = 50. which can be divided into 2 sub arrays of 25 elements each. These 2 sub arrays can again be divided in four sub arrays of 12 and 13 elements each. This division can go as till each sub arrays has only one element. That is for original array having n elements, n sub arrays can be formed (each sub array has exactly one element) and then these arrays can be merged using simple merge technique.
As an example consider the array
(10, 9, 8,7 6, 5, 4, 3, 2, 1)
Step 1: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
Mid point is at position 5, so that left and right sub arrays consist 5 elements (position 1 to 5) and right sub array consists 5 elements.
Step 2: Left sub array is divided into 2 equal parts.
10, 9, 8, 7, 6, 5, 4, 3, 2, 1



At this point when division of sub arrays is not possible, a simple merge procedure is called for (10, 9, 8) to sort it as (8, 9, 10). This process goes on till all sub arrays have been sorted.

Example:

Consider following unsorted array:

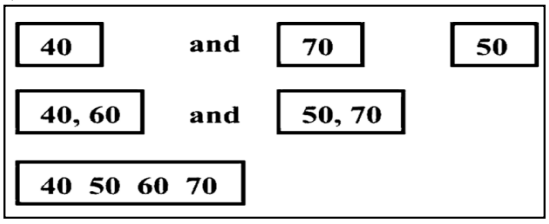
40, 60, 70, 50, 20, 10, 30

Break the list into two sub array

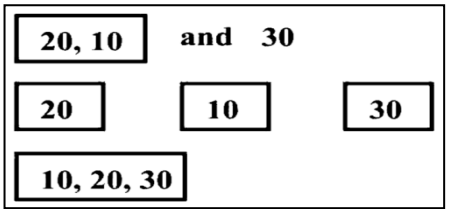
First sub array: 40 60 70 50

Second sub array: 20 10 30

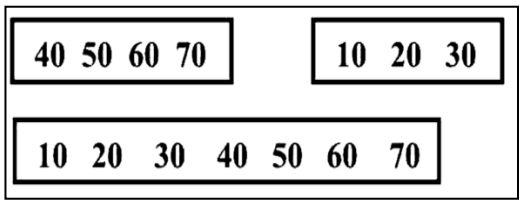
Again subdivide first sub list: 40, 60 and 70,50



Divide the second sub list:



Merge the two sub list



3.2.9.1 Comparison:

For any array of size n (where n is power of 2 i. e. $n = 2^m$ or $m = \log_2 n$) merge sort divides the array in two halves (of size n/2 each) and then the two sub arrays of size n/2 are merged. The two sub arrays of size n/2 are again divided in four equal size arrays of n/4 elements and then merged. This process would go on till n sub arrays of size one each are created. Thus the time can be calculated as

$= n + 2 (n/2) + 4 (n/4) + \ldots + n (n/n)$

$= n + n + \ldots + n$ (m terms)

$= m = n \log_2 n$

Therefore merge sort has running time of $O(n \log_2 n)$

3.2.10 RADIX SORT

Radix sort is based on the value of the actual digits in the positional representation of the number being sorted.

Larger integer can be determined as follows: Start at the most significant digit and advance through the least significant digits as long as the corresponding digits in the two numbers match. In this sort, we take each number in the order in which they appear in the file and place it into one of ten queues, depending on the value of the digit currently being processed. After this restore each queue to the original file starting with the queues of numbers with a 9 digit. When the actions have been performed for each digit, starting with the least significant and ending with the most significant, the file is sorted

Example:

Consider the following unsorted array

25, 57, 37, 12, 92, 86, 33

Queues based on least significant digit

Queue [0]

Queue [1]

Queue [2] 12, 93

Queue [3] 33

Queue [4]

Queue [5] 25

Queue [6] 86

Queue [7] 57, 37

Queue [8] 48

Queue [9]

After first pass:

12, 92, 33, 25, 86, 57, 37, 48

Queues based on most significant digit

Queue [0]

Queue [1] 12

Queue [2] 25

Queue [3] 33, 37

Queue [4] 48

Queue [5] 57
Queue [6]
Queue [7]
Queue [8] 86
Queue [9] 92

Thus sorted file is:
12, 25, 33, 37, 48, 57, 86, 92

3.3 SEARCHING

A table or a file is a group of elements, each of which is called a record. For each record there is a key associated, which is used to differentiate among different records. The association between a record and its key may be simple or complex. In simple way, the key is contained within the record at a specific offset from the start of the record. Such a key is called an internal key or an embedded key whereas a separate table of keys that includes pointer to the records, such keys is called as external keys. For every file there is at least one set of keys that is unique, such a key is called a primary key.

A search algorithm is an algorithm that accepts an argument b and tries to find record a pointer to that b. The algorithm may return the entire record or more commonly it may return a pointer to that record. It is possible that the search for a particular argument in a table is unsuccessful; i.e. there is no record or a null pointer. If a search is unsuccessful it may be desirable to add a new record with the argument as its key.

The algorithm that does the above task is called a searching and insertion algorithm. A successful search is called a retrieval. A table of records in which a key is used for retrieved is often called a search table or a dictionary.

BASIC SEARCHING TECHNIQUES

3.3.1 Sequential searching

The simplest technique for searching an unordered table for a particular record is to scan each entry in the table in a sequential manner until the desired record is found. An algorithm for such a search procedure is as follows.

Function LINEAR SERCH(K, N, X). Given an unordered vector K consisting of N + 1($N \geq 1$) elements, this algorithm searches the vector for a particular element having the value X. Vector elements K[N + 1] serves as a sentinel element and receives the value of X prior to the search. The function returns the index of the vector element if the search is successful and returns 0 otherwise.

1. [Initialize search]
   $I \leftarrow 1$
   $K[N+1] \leftarrow X$
2. [Search the vector]
   Repeat the vector [1] $\neq X$
3. [Successful search?]
   If I = N + 1
   Then Write (`UNSUCCESSFUL SEARCH`)
   Return (0)
   Else Write (`SUCCESSFUL SEARCH`)
   Return (I)

The first step of the algorithm initializes the key value of the sentinel record to x. In the second step, a sequential search is then performed on the n + 1 records. If the index of the record found denotes record R_{n+1} then the search has failed; otherwise, the search is successful and I contains the index of the desired record.

Recall that the performance of a search method can be measured by counting the number of key comparisons taken to find a particular record. There are two cases which are important, namely, the average case and the worst case. The worst case for the previous algorithm consists of n + 1 key comparisons, while the average case takes (n + 1)/2 key comparisons. The average and worst search times for this method are both proportional to n, that is, of O(n). These estimates are based on the probability of a request for a particular record is the same as for any other record. Let Pi be the probability for the request of

record R_1 for $1 \le i \le n$. The average length of search (ALOS) for n records is given by

$E[ALOS] = 1 * P_1 + 2 * P_2 \ldots\ldots + n * P_n$
Where $P_1 + P_2 + \ldots\ldots + P_n = 1$.

Now suppose that the probabilities for request for particular records are not equally likely, that is, $P_1 \ne 1/n \le i \le n$. the question which naturally arises is: Can we rearrange the table so as to reduce the ALOS? The answer is yes and the desired arrangement can be obtained by looking at the previous equation for the expected ALOS. This quantity will be minimized if the records are ordered such that

$P_1 \ge P_2 \ge \ldots\ldots\ldots \ge P_n$. .....(1)

For example, letting n = 5 and $P_1 = 1/5$ for $1 \le i \le 5$ yields

$E[ALOS] = 1 * 1/5 + 2 * 1/5 + 3 * 1/5 + 4 * 1/5 = 3$

Now, assuming that $P_1 = 0.4$, $P_2 = 0.3$, $P_3 = 0.2$, $P_4 = 0/07$ and $P_5 = 0.03$, the average length of search in this case is

$E[ALOS] = 1*0.4 + 2*0.3 + 3*0.2 + 4*0.07 + 5*0.03 = 2.03$

This number is substantially less than 3. The rearrangement of the initial t according to Eq. (1) is called preloading.

If the table is subjected to many deletions, then it should be represented as a linked list. The traversal of a linked table is almost as fast as the traversal of its sequential counterpart. Note that insertions can be performed very efficiently when the table is sequentially represented, assuming that the table is not ordered.

3.3.2 Indexed sequential search

The indexed sequential search improves search efficiency for a sorted file, but it involves an increase in the amount of space required.

An auxiliary table called an index is set aside in addition to the sorted file itself.

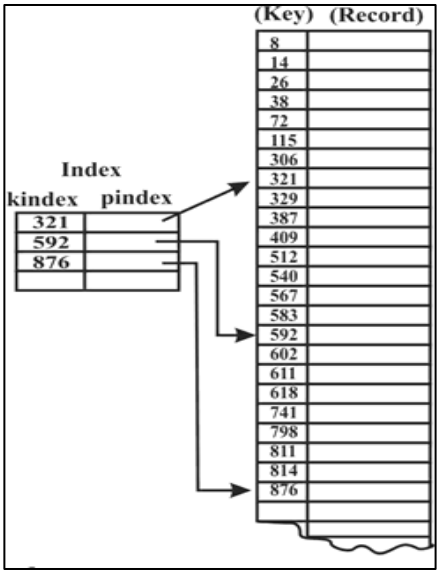


Fig 1: Indexed Sequential file

- Each element in the index consists of a key k index and a pointer the record in the file that corresponds to k index. The elements in the index, as well as the elements in the file, must be sorted on the key. If the index is $1/8^{th}$ the size of the file, every $1/8^{th}$ record of the file is represented in the index as shown in the figure.
- The algorithm used for searching an indexed sequential file is simple. Let index be an array of the keys in the index and let p index be the array of pointers within the index to the actual records in the file.

Note: We assume that the file is stored as an array, that n is the size of the file and that index size is the size of the index.

```
For (i=0;i< index size && k index (i)<= key ;i++)
Lowly = (i= = 0)? 0 : p index (i) – 1;
for(j=lowly;<=helm && k(j) !=key !=key; j++);
return ((j > hilim)? – 1 : j);
```

In the case of multiple records with the same key this algorithm does not necessarily return a pointer to the first such record in the table.

- **Advantage:**

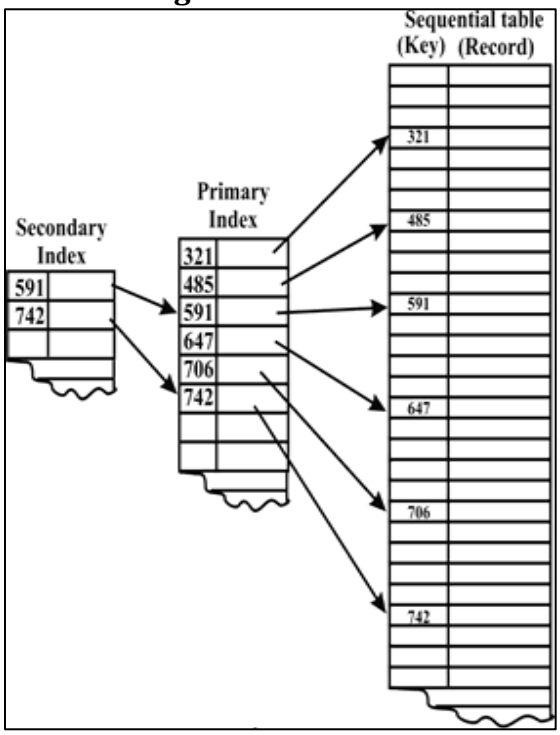


Fig 2: use of secondary index

The items in the table can be examined sequentially if all the records in the file must be accessed, yet the search time for a particular item is sharply reduced.

- A sequential search is performed on the smaller index rather than on the larger table. Once the correct position has been found, a second sequential search is performed on a small portion of the record table itself.

- The index can be use in both cases i. e. sorted table is stored as a linked list as well as array.
  If the table is so large that even the use of an index does not achieve sufficient efficiency. In this case secondary index can be used and it acts as an index to the primary index which points to entries in the sequential table as shown in figure shown above.

- Deletion from an indexed sequential table can be made easily by flagging deleted entries. In sequential searching through the table, deleted entries are ignored.

  Insertion into an indexed sequential table is more difficult, since there may not be space between tow already existing table entries. Therefore it is necessary to shift in a large number of table elements. But it a herby item has been flagged as deleted in the table only a few items need to be shifted and the deleted item can be overwritten. This may require alternation of the index if an item pointed to be an index element is shifted.

3.3.3 BINARY SEARCH

Binary searching is the most efficient method of searching a sequential table without the use of auxiliary indices or tables. Basically, the argument is compared with the key of the middle element of the table. If they are equal the search ends successfully otherwise either the upper or lower half of the table must be searched in a similar manner.

- Algorithm:
  1. Low = 0;
  2. Hi = n – 1;
  3. While (low < = hi) {
  4. mid = (low + hi) /2;
  5. if (key = = k(mid);
  6. return (mid))
  7. if (key < k(mid))
  8. hi = mid – 1;
  9. Else
  10. low = mid + 1;
  11. } /* end while */
  12. return (-1);

- Each comparison in the binary search reduces the number of possible candidates by a factor of 2. Thus, the maximum number of key comparisons is approximately $\log_2$
  We may say that the binary search algorithm is O(log n)

- The binary search may be used in conjunction with the indexed sequential table organization. In that, instead of searching the index sequentially, a binary search can be used. The binary search can also be used in searching the main table once tow boundary records are identified. But, the size of this table segment is likely to be small enough so that a binary search is not more advantageous than a sequential search. The binary search algorithm can only be used if the table is stored as an array.

Binary search in the presence of insertions and deletions with maximum number of elements

- A data structure known as `padded List` is used for utilizing binary search in the presence of insertions and deletions if the maximum number of elements are available.

- This method uses two arrays:
  i) An elements array
  ii) Parallel flag array
  iii) The element array contains the sorted keys in the table with `empty` slots initially evenly interspersed among the key of the table to allow for growth.
  iv) An empty slot is indicated by a 0 value in the corresponding flag array element, whereas a full slot is indicated by the value 1. Each empty slot in the element array contains a key value greater than or equal to the key value in the previous full slot and less than the key value in the following full slot. Thus the entire element, perform a binary search on the array. If they performed on it. To search for an element does not exist in the table. If it is found and the corresponding flag value is 1, the element has the argument key. If it does the element has been located, if it does not, the element does not exist in the table.

4 TREE

TREE

Trees are nonlinear data structures whereas strings, arrays, lists, stacks and queues are linear data structure. The structure is mainly used to represent data containing a hierarchical relationship between elements e.g. records, family trees and tables of contents.

4.1 BINARY TREES

A binary tree T is defined as a finite set of elements, called nodes, such that T is empty(called the null tree or empty tree), or T contains a distinguished node R, called the root of T, and the remaining nodes of T form as ordered pair of disjoint binary trees T_1 and T_2

- If T contains a root R then two trees T_1 and T_2 are called the left and right sub trees of R.
- If T_1 is nonempty, then its root is called the left successor of R; similarly, if T_2 is nonempty, then its root is called the right successor of R.
- A conventional method to represent a binary tree is shown in figure below.

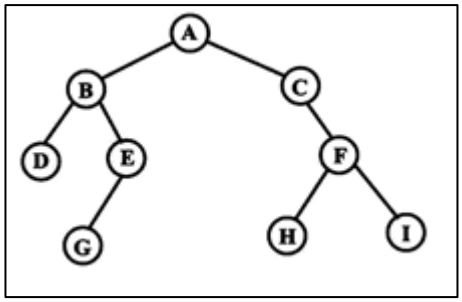


Fig 1: Binary Tree

4.1.1 Description

This tree consists of nine nodes with A as its root. It's left sub tree is rooted at B and its right sub-tree is rooted at C. (This is shown by the two branches emanating from A to B on the left and to C on the right) The absence of a branch indicates an empty sub tree.

For example:
In fig. 1 the left sub tree of the binary tree rooted at C and the right sub tree of the binary tree rooted at E are both empty. The binary tree rooted at D, G, H, I have empty right and left sub trees.

4.1.2 BASIC TERMINOLOGY

- If A is the root of a binary tree and B is the root of its left or right sub-tree, then A is said to be the father of B and B is said to be the left or right son of A.
- A node that has no son, such as D, G, H, and I in figure 2 are called a leaf.
- Node n_1 is an ancestor of node n_2 (and n_2 is a descendent of n_1) if n_1 is either the father of n_2 or the father of some ancestor of n_2.

For example:
In figure 1, A is an ancestor of G, and H is a descendent of C but E is neither an ancestor nor a descendant of C.

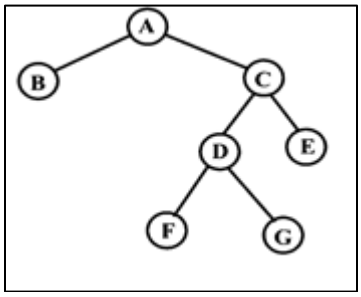


Fig 2: Strictly binary tree

- A node n_2 is a left descendant of node n_1 is either the left son of n_1 (A right descendent may be similarly defined).

- Two nodes are brothers if they are left and right sons of the same father.
- Strictly Binary Tree: If every non leaf node in a binary tree has non empty left and right subtrees, the tree is called as strictly binary tree. Tree in figure 3 is a strictly binary tree.

  A strictly binary tree with n leaves always contains 2n – 1 node.
- **Level:** The root of the tree has level 0, and the level of any other node in the tree is one more than the level of its father.
  For example: In figure 1, node E is at level 2 and node H is at level 3.
- **Depth:** The depth of a binary tree is the maximum level of any leaf in the tree and equals to the length of the longest path from the root to any leaf. Thus depth of binary tree in figure 1 is 3.
- If a binary tree contains n nodes at level m, it contains at most 2n nodes at level m + 1. Since a binary tree can contain at most one node at level 0 (the root), if can contains at most 2^m nodes at level m.
- A complete binary tree of depth d is the binary tree of depth d that contains exactly 2^d nodes at each level m between 0 and d. (i. e. the binary tree of depth d that contains exactly 2^d nodes at level d.)
- The total number of nodes in a complete binary tree of depth d is t_n and it equals to the sum of the number of nodes at each level between 0 and d. Thus, by induction

  $$T_n = 2^{d+1} - 1$$

  Since all total leaves in such a tree at level d, the tree contains 2^d leaves and therefore 2^d – 1nonleaf nodes
- **Almost complete binary tree:** A binary tree of depth d is an almost complete binary tree if –

  1. Any node and at level less than (d – 1) has two sons.
  2. For any node and in the tree with a right descendent at level d, and must have a left son and every left descendent of and is either a leaf at level d or has two sons. The figure shown below shows an almost complete binary tree.

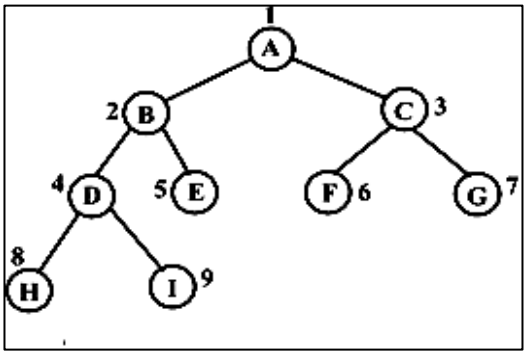


Fig 3: Node numbering for almost complete binary tree

  3. The nodes of an almost complete binary tree can be numbered so that root is assigned the number 1, a left son is assigned twice the number assigned its father and a right son is assigned one more than twice the number assigned its father (It is as shown in figure 3).

  4. An almost complete strictly binary tree with n leaves has 2n – 1 nodes, as same of other strictly binary tree with n leaves. An almost complete binary tree with n leaves that is not strictly binary has 2n nodes. There are two distinct almost complete binary trees with n leaves, one of which is strictly binary and one of which is not.

There is only a single almost complete binary tree with n nodes. The tree is strictly binary if and only if n is odd.

5. The tree of figure 3 is the only almost complete binary tree with 9 nodes and is strictly because 9 is odd, whereas the tree of figure 5 is the only almost complete binary tree with ten nodes and is not strictly binary because 10 is even.

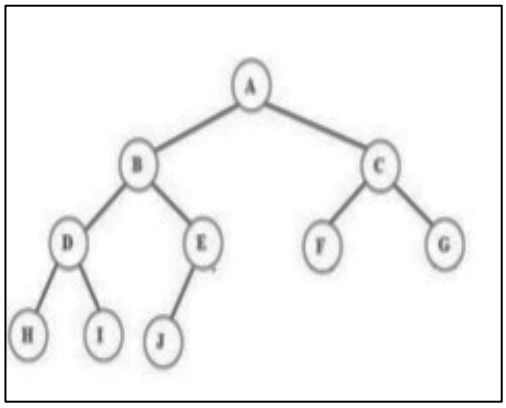


Fig. 5: Almost complete binary tree but not strictly binary tree

6. An almost complete binary tree of depth d is intermediate between the complete binary tree of depth d –1, that depth d, which contains 2^d-1 nodes and the complete binary tree of depth d, which contains 2^{d+1} - 1 nodes.

If t_n is the total number of nodes in an almost complete binary tree, its depth is the largest integer less than or equal to $\log_2 t_n$

Example:
The almost complete binary trees with 4, 5, 6 and 7 nodes have depth 2 and the almost complete binary trees with 8, 9, 10, 11, 13, 14, and 15 nodes have depth 3.

4.1.3 APPLICATION OF BINARY TREES

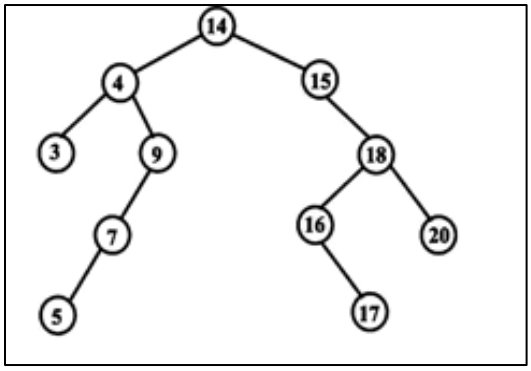


Fig. 6: Binary tree constructed for finding duplicates

1. A binary tree is a useful data structure when two-way decisions must be made at each point in a process.
2. The number of comparisons can be reduced by using a binary tree. The first number in the list is placed in a node that is established as the root of a binary tree with empty left and right sub trees. Each successive number in the list is then compared to the number in the root.

- If it matches, we have duplicates
- If it is smaller, we observe the left sub tree
- If it is larger, we observe the right sub tree
- If the sub tree is empty, the number is not a duplicate and is placed into a new node at that position in the tree
- If the sub tree is non-empty, we compare the number to the contents of the root of the sub-tree and the process is repeated with the sub tree.

The figure 6 shows the tree constructed from the input 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9, 14, 5.

3. Traversing a Binary Tree:

Traversing means to pass through the tree, enumerating each of its nodes once. All the nodes of the linear list are visited in a traversal from first to last. There is no natural linear order for the nodes of a tree, therefore different ordering are used for traversal in different cases. We will define three recursive traversal methods so that traversing a binary tree involves visiting the root and traversing its left and right sub-trees.

a. Preorder (depth – first order)

To traverse a nonempty binary tree in pre-order also known as depth-first order. We perform the following three operations:

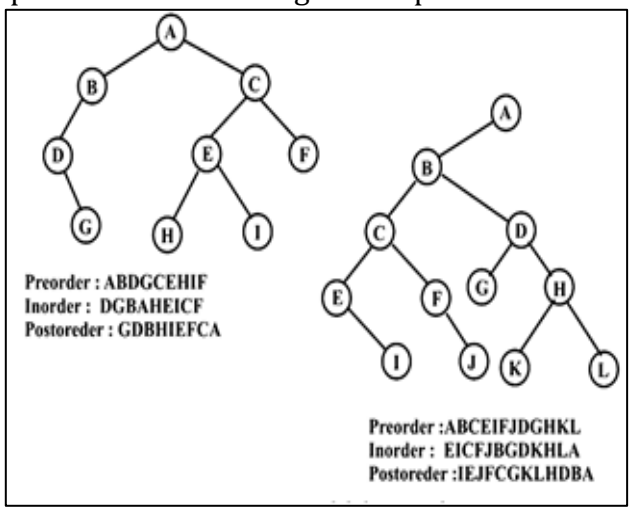


Fig. 7: Traversal of binary trees

1. Visit the root
2. Traverse the left sub-tree in pre-order
3. Traverse the right sub-tree in pre-order.

b. In-order (Symmetric order)

To traverse a nonempty binary tree in in order or symmetric order:

1. Traverse the left sub tree in in-order
2. Visit the root
3. Traverse the right sub tree in in-order

c. Post order

To traverse a nonempty binary tree in post order

1. Traverse the left sub tree in post order
2. Traverse the right sub tree in post order
3. Visit the root

4. Many algorithms that use binary trees proceed in two phases. The first phase builds a binary tree, and the second phase traverses the tree.

Example: Consider the following sorting method:
Given a list of numbers in an input file, we wish to print them in ascending order. All the numbers are inserted into the binary tree with duplicate values also. When a number is compared with the contents of a node in the tree, a left branch is taken if the number is smaller than the contents of the node and right branch if it is greater or equal to the contents of the node. Thus, if the input list is 14, 4, 15, 3, 9, 14, 18, 7, 9, 16, 20, 5, 17, 4, 5 then the binary tree is constructed as shown in the figure below.

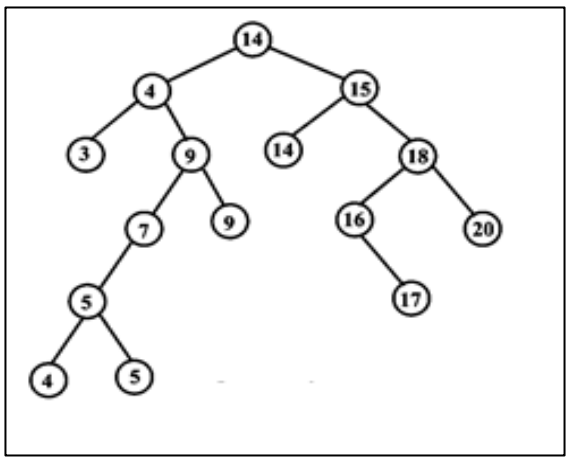


Fig. 8: Binary tree constructed for sorting

Note: Such a binary tree with a property that, all elements in the left sub tree of a node n are less than the contents of n, and

all elements in the right subtree of n is greater than or equal to the contents of n.

A binary tree with such a property is called binary search tree. If a binary search tree is traversed in order and the contents of each node are printed as the nodded is visited, the numbers are printed in ascending order.

5. Expressions and their binary tree representation:

The expression containing operands and binary operators can be represented by binary trees as shown in following figure.

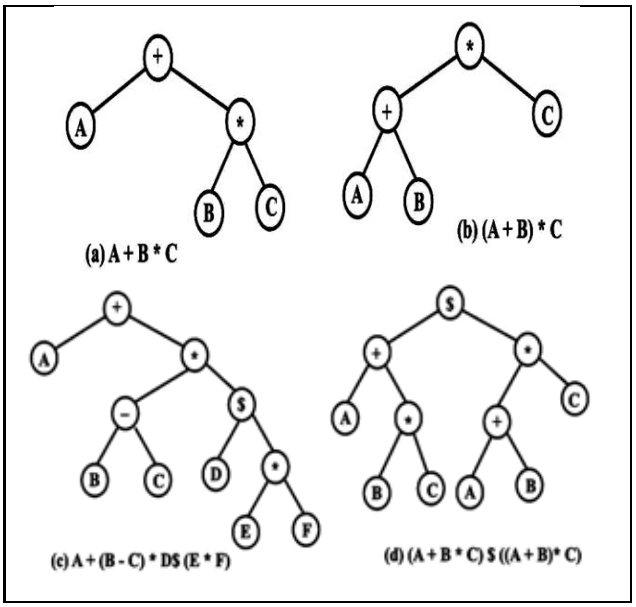


Fig. 9: Expression and their binary representation

4.1.4 REPRESENATION OF BINARY TREES IN MEMORY

Let T a binary tree, we will discuss the two methods to represent T in memory.

i) Link representation of T

ii) Single array or sequential representation of T.

The important requirement of any representation of T is that one should have direct access to the root R of T and given any node N of T, one should have direct access to the children of N.

4.1.5 Linked Representation of Binary Trees

Consider a binary tree T, T will be maintained in memory by means of a linked representation which uses three parallel arrays, INFO, LEFT and RIGHT and a pointer variable ROOT as follows. Each node N of T will correspond to a location k such that

a) INFO[k] ⇒ contains the data at node N
b) LEFT[k] ⇒ contains the location of the left child of node N.
c) RIGHT[k] ⇒ contains the location of the right child of node N.

- Also, ROOT will contain the location of the root R of T. If any subtree is empty, then the corresponding pointer will contain the null value, if the tree T is itself is empty, then ROOT will contain the null values.

  For example: A schematic linked representation of binary tree T is as shown in figure below.

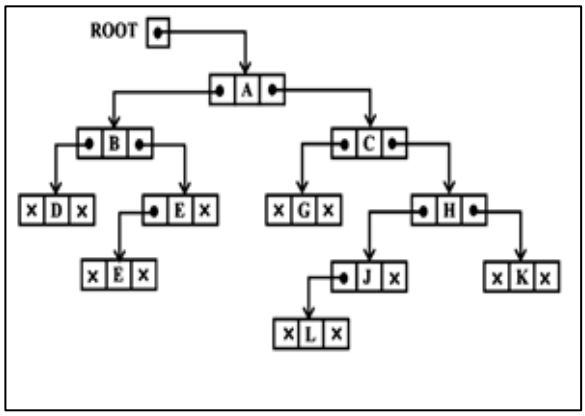


Fig. 10: Binary tree representation

- Each node is shown with three fields and that empty sub trees are pictured by using X for the null entries. Figure below shows how this linked representation may appear in memory. The choice of 20 elements for the array is arbitrary and AVAIL list is maintained as a one – way list using the array LEFT.

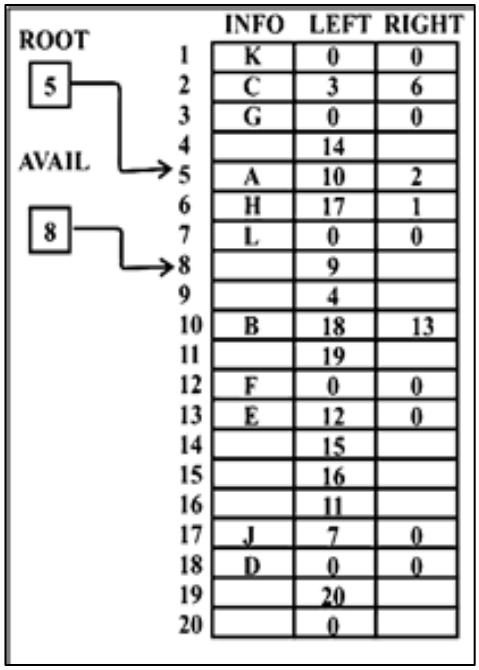


Fig. 11: Linked representation of binary tree in memory

4.1.6 Sequential Representation of Binary Trees

- Suppose T is a binary tree that is complete or nearly complete. Then there is an efficient way of maintaining T in memory called the sequential representation of T.
  This representation uses only a single linear array TREE as follows:
  a) The root R of T is stored in TREE [1]
  b) If a node N occupies TREE [k], then its left child is stored in TREE [2*k] and its right child is stored in TREE [2*k + 1].

- Also, NULL is used to indicate an empty subtree. In particular TREE[1] = NULL indicates that the tree is empty.
  The sequential representation of the binary tree T in figure 12(a) below is shown in figure 12(b).

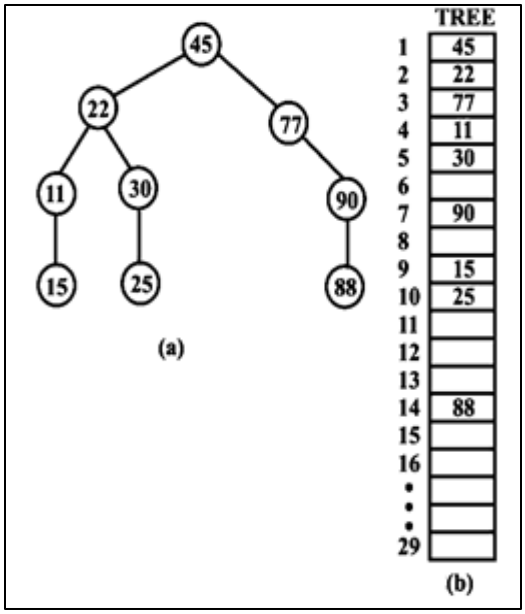


Fig. 12: Sequential representation of the binary tree

- The sequential representation of a tree with depth d will require an array with approximately 2^{d+1} elements. Accordingly, this sequential representation is usually inefficient unless as stated above, binary tree T is complete or nearly complete.

4.1.7 Node representation of a Binary tree

1. Array node representation: A tree node may be implemented as array elements or as allocations of a dynamic variable. Each node contains info, left, right and father fields. The left, right and father fields 3 of a node point to the node`s left son, right son and father respectively.

- Array implementation of a node

```
# define NUMNODES 500
struct node_type {
 int info;
 struct node_type * left, *right,
 *father;
 }node[NUMNODES];
```

- Under this, the operations info(p), right(p) and father (p) are implemented by reference to node [p].info, node [p].

left, node[p].right and node[p].father respectively and the operations is left (p), is right(p) and brother(p) can be implemented using above operations.

- The available, get node, free node etc. are some operations required to implement above functions.
  Note: The available list is not a binary tree but a linear list whose nodes are linked together by the left field is a binary tree.

- Each node in a tree taken from the available pool when needed and returned to available pool when no longer in use. This representation is called the linked array representation of a binary free

2. **Dynamic node representation:** The operations info(p), right (p) and father (p) would by implemented by reference to P $\rightarrow$info, p $\rightarrow$left, p $\rightarrow$right and p $\rightarrow$ father, respectively. Also, explicit available list is not needed. The routines get node and free node simply allocated and free nodes using the routines malloc and free. This representation is called Dynamic node representation.

Note: Both the linked array representation and the dynamic node representation are implementation of an abstract linked representation in which explicit pointers link together the nodes of a binary tree.

4.1.8 COMPARISON OF TREE AND BINARY TREE

A tree is a finite non-empty set of elements in which one element is called the root and the remaining elements are partitioned into m > = 0 disjoint subsets, each of which is itself a tree.

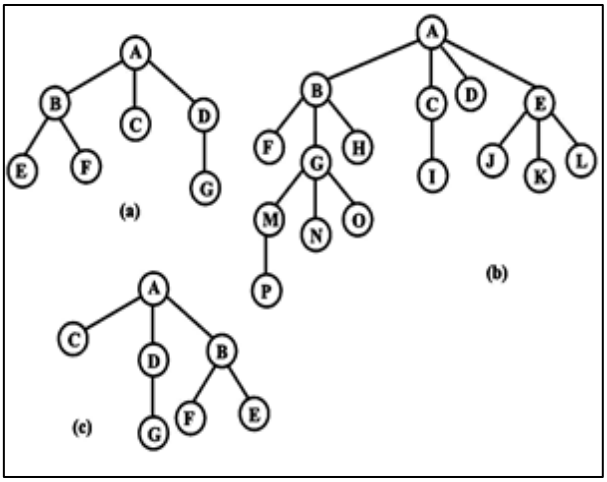


Fig. 13: Examples of tree

Example:

- Each node may be the root of a tree with zero or more sub-trees.

- A node with no sub trees is a leaf.
  (Note: The terms father, son, brother, ancestor, descendant level and depth is also available for trees and have same meaning.)

- The degree of a node in a tree is equal to the number of its sons. Thus, in fig 13(a) shown above node C has degree 0 node D has degree 1, node B has degr e e 2 etc. There is no upper limit on the degree of a node.

Equivalent Trees:

- The tree shown in fig. 13(a) and (c) are equivalent trees.
- Each has A as its root and there sub trees one of those sub trees has root C with no sub-trees, another has root D with a single sub-tree rooted at G and the third has root B with two sub trees rooted at E and F.

Note: The only difference is the illustration of the order in which the sub-trees are arranged.

- The definition of a tree makes no distinction among sub trees of a general tree, while a binary tree, in which a

distinction is made between the left and right sub trees.

Ordered tree: An ordered tree is defined as a tree in which the sub trees of each node form an ordered set.

- The first son of a node in an ordered tree is often called the **oldest son** of that node, and the last on is called the **youngest**.
- The trees of fig. 17(a) and (c) are equivalent as unordered trees; they are different as ordered trees
- A forest is an ordered set of ordered trees

Comparison of tree and binary tree:

1. Every binary tree except for the empty binary tree is indeed a tree; however, not every tree is binary.
2. A tree node may have more than two sons, whereas a binary tree node may not.
3. Even a tree whose **nodes** have almost two sons is not necessarily a binary tree because only son in a general tree is not designated as being a "left" or a "right" son, whereas in a binary tree, every son must be either a "left" son or a "right" son.
4. A non-empty binary tree is a tree, the designation of left and right have no meaning within the context of a tree (except perhaps to order the two sub trees of those node with two sons.)
5. A non-empty binary tree is a tree each of whose nodes has maximum of two sub trees which have the added designation of "left" or "right".

4.2 HEADER NODES: THREADS

Consider a binary tree T. Variation of the linked representation of T are frequently used because certain operation on T are easier to implement by using modifications.

4.2.1 Header Nodes

Suppose a binary tree T is maintained in memory by means of a linked representation. Sometimes an extra, special node, called a header node, is added to the beginning of T. When this extra node is used, the tree pointer variable, which we will call HEAD, will point to the header node, and the left pointer of the header node will point to the root of T. Fig. 14(b) shown below shows schematic representation of the binary tree, in fig. 18(a). It uses a linked representation with a header node.

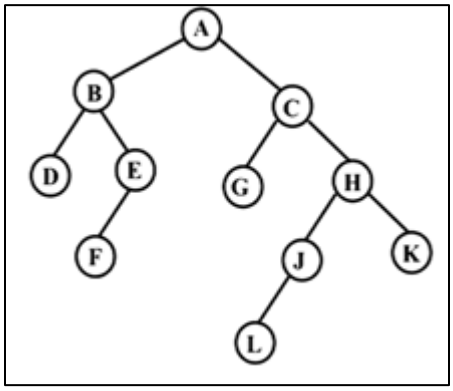


Fig. 14(a): Binary tree

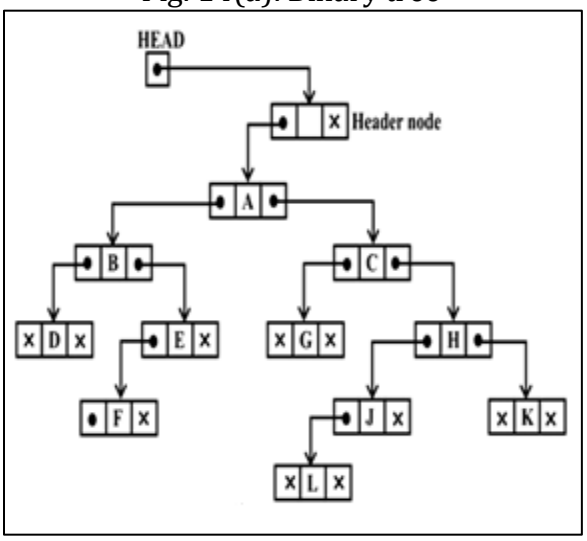


Fig. 14 (b): Schematic representation of the binary tree

- Suppose a binary tree T is empty. Then T will still contain a header node, but

the left pointer of the header node will contain the null value.
i.e. LEFT [HEAD] = NULL indicate an empty tree.

- Another variation of the above representation of a binary tree T, is to use the header node as a sentinel i. e. if a node has an empty subtree, then the pointer field for the sub tree will contains the address of the header node instead of the null value. No pointer will ever contain an invalid address and the condition, LEFT [HEAD] = HEAD Will indicate an empty sub tree.

4.2.2 Threads and In order Threading

- In the linked representation of a binary tree T, approximately fifty percent of the entries in the pointer fields LEFT and RIGHT will contain null elements. This space may be more efficiently used by replacing the null entries by some other type of information specification, we will replace certain null entries by special pointers which points to node higher in the tree. These special pointers are called threads and binary trees with such pointer are called threaded trees.
- From ordinary pointers, the threads in threaded tree must be distinguished. The threads in the diagram of a threaded tree are usually denoted by dotted lines. In memory, an extra 1 – bit TAG field may be used to distinguish threads from ordinary pointers or alternatively, threads may be denoted by negative integers when ordinary pointers are denoted by positive integers.

- There are many ways to thread by which we can thread a binary tree T, but each threading will correspond to a particular traversal of T. Also, one may select a one way or two threading.
(**Note:** Here we will see in-order treading only)

1. In the one-way threading of T, a thread will appear in the right field of a node and will point to the next node in the in order traversal of T.

2. In the two-way threading of T, a thread will also appear in the LEFT field of a node and will point to the preceding node in the in order traversal of T.

3. Also the left pointer of the first node and the right pointer of the last node (in the in order traversal of T) will contain the null value when T does not have a header node, but will point to the header node when T does have a header node.

Note: There is a similar one way, there is no threading of T which corresponds to the post order traversal of T on the other way, and there is no threading of T which corresponds to the post order traversal of T. The following figure shows threading for a binary tree shown in figure 14(a).

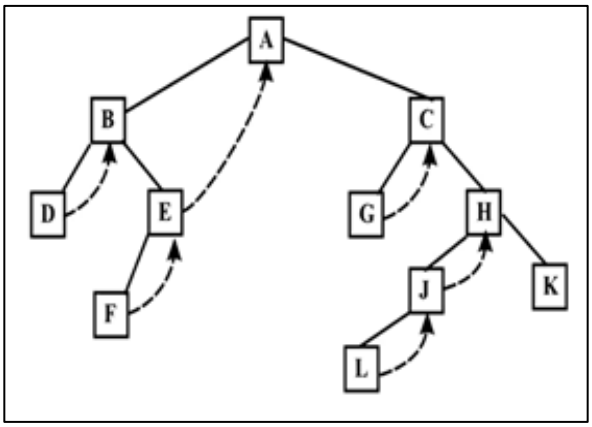


Fig. 15(a): One way inorder threading

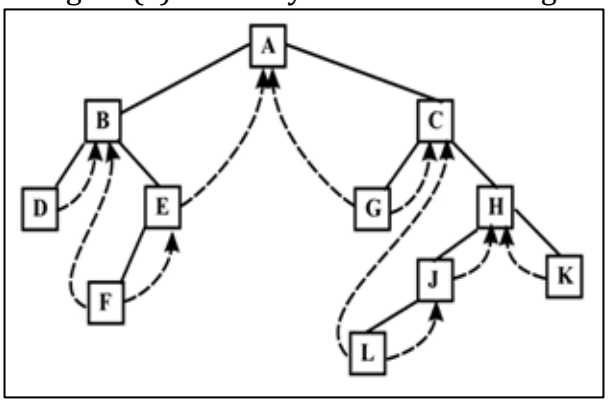


Fig. 15(b): Two way inorder threading

4.3 BINARY SEARCH TREES

Suppose T is a binary tree, then T is called a binary search tree (or binary sorted tree) if each node N of T has the following property:

The value at N is greater than every value in the left sub tree of N and is less than every value in the right sub tree of N. (Assume that all nodes are distinct).

OR

The value at N is greater than every value in the left sub tree of N and is less than or equal to every value in the right sub tree onN. This is a very important data structure and it enable one to search for and element with an average running time f(n) = $O(\log_2 n)$. It also enables one to easily insert and delete an item. This structure associates with the following structures:

a) Sorted Linear Array:

Here one can search for and find an element with a running time f(n) = $O(\log_2 n)$ but it is expensive to insert and delete elements.

b) Linked List:

Here one can easily insert or delete elements but it is expensive to search and find an element, since one must use a linear search with running time f(n) = O(n).

Example:

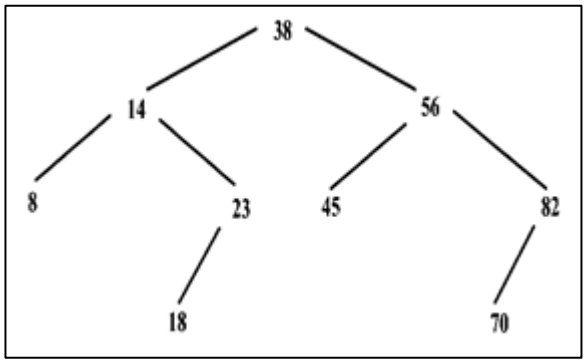


Fig. 16: Binary tree

4.3.1 SEARCHING AND INSERTING IN BINARY SEARCH TREES

Let T be a binary search tree, suppose ITEM information is given. The following algorithm finds the location of ITEM in the binary search tree T or insert or ITEM as a new node in its appropriate place in the tree.

a) Compare ITEM with the root node N of the tree
   i) If ITEM < N, proceed to the left child of N.
   ii) If ITEM > N, proceed to the right child of N.

b) Repeat Step (a) until one of the following occurs:
   i) We meet a node N such that ITEM = N. In this case the search is successful, and we insert ITEM place of the empty sub tree. In other words, proceed from the root R down through the tree T until finding ITEM in T or inserting ITEM as a terminal node in T.

Example:

Consider the binary search tree T in figure 20. Suppose ITEM = 20 is given.
Simulating the above algorithm, we obtain the following steps:

1. Compare ITEM = 20 with the root, 38, of the tree T. Since 20 < 38, proceed to the left child of 38, which is 14.
2. Compare ITEM = 20 with 14. Since 20 >14, proceed to the right child of 14, which is 23.
3. Compare ITEM = 20 with 23. Since 20 > 23, proceed to the left child of 23, which is 18.
4. Compare ITEM = 20 with 18. Since 20 > 18 does not have a right child, insert 20 as the right child of 18. Figure 17 shows the new tree with ITEM = 20 inserted in tree shown in figure 16.. The shaded edges indicate the path down through the tree during the algorithm.

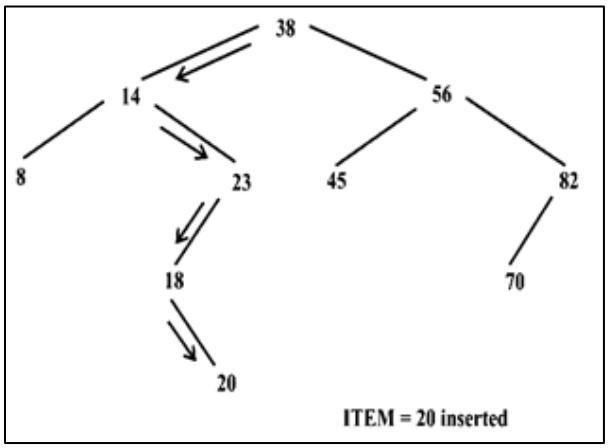


Fig. 17: Binary tree with insertion of item 20

Procedure:

The formal presentation of our search and insertion algorithm will use the following procedure which finds the location of a given ITEM and its parent. The procedure traverses down the tree using the pointer PTR and the pointer SAVE FOR THE PARENT NODE. This procedure will also be used in the next section, on deletion. Observe that in Step 6, we move to the left child or the right child according to whether ITEM < INFO [PTR] or ITEM > INFO [PTR]

Algorithm:

The formal statement of our search and insertion algorithm follows.
Observe that, in Step 4, there are there possibilities: (1) the tree is empty, (2) ITEM is added as a left child and (3) ITEM is added as a right child.

4.3.2 Complexity of the searching Algorithm

Suppose we are searching for an item of information in a binary search tree T. Observe that the number of comparisons is bounded by the depth of the tree. This comes from the fact that we proceed down a single path of the tree. Accordingly, the running time of the search will be proportional to the depth of the tree.
Suppose we are given n data items, A_1, A_2, ……A_N, and suppose the items are inserted in order into a binary search tree T. Recall that there are n! Permutations of the n items. Each such permutation will give rise to a corresponding tree. It can be shown that the average depth of the n! trees is approximately ($c \log_2 n$), where c = 1.4. Accordingly, the average running time f(n) to search for an item in a binary tree T with n elements is proportional to $\log_2 n$, $f(n) = O(\log_2 n)$.

4.3.3 Application of Binary Search Trees

Consider a collection of n items, A_1, A_2, ……A_N. Suppose we want to find and delete all duplicates in the collection. One straightforward way to do this is as follows:

Algorithm A:

Scan the elements from A_1 to A_N (that is, from left to right).

a) For each element A_k, compare A_k with A_1, A_2,……,A_{k-1}, that is, compare A_k with those elements which precede A_k.
b) If A_k does occur among A_1, A_2,……,A_{k-1}, then delete A_k.

After all elements have been scanned, there will be no duplicates.

EXAMPLE
Suppose Algorithm A is applied to the following list of 15 numbers:
14, 10, 17, 12, 10, 11, 20, 12, 18, 25, 20, 8, 22, 11, 23
Observe that the first four numbers (14, 10, 17 and 12) are not deleted. However,

$A_5 = 10$ is deleted since $A_5 = A_2$

$A_8 = 12$ is deleted since $A_8 = A_4$

$A_{11} = 20$ is deleted since $A_{11} = A_7$

$A_{14} = 11$ is deleted since $A_{14} = A_6$

When Algorithm A is finished running, the 11 numbers - 14, 10, 17, 12, 11, 20, 18, 25, 8, 22, 23 which are all distinct, will remain.

Consider now the time complexity of algorithm A, which is determined by the number of comparisons. First of all, we assume that the number d of duplicates is very small compare with the number n of data items. Observe that the step involving A_k will require approximately k - 1comparisons, since we compare A_k with items $A_1, A_2, \ldots\ldots A_{k-1}$(less the few that may already have been deleted). Accordingly, the number f(n) of comparisons required by Algorithm A is approximately

$0+1+2+3+\ldots\ldots\ldots\ldots+(n-2)+(n-1)=$

$$\frac{(n-1)n}{2} = O(n^2)$$

For example, for n = 1000 items, Algorithm A will require approximately 500000 comparisons. In other words, the running time of Algorithm A is proportional to n. Using a binary search tree, we can give another algorithm to find the duplicates in the set $A_1, A_2, \ldots\ldots, A_N$ of n data items.

Algorithm B:

Build a binary search tree T using the elements $A_1, A_2, \ldots\ldots A_n$. In building the tree, delete A_x from the list whenever the value of A_k already appears in the tree.

The main advantage of Algorithm B is that each element A_k is compared only with the elements in a single branch of the tree. It can be shown that the average length of such a branch is approximately $c \log_2 k$, where c = 1.4. Accordingly, the total number f(n) of comparisons required by Algorithm B is approximately $O(n\log_2 n)$. That is, $f(n) = O(n \log_2 n)$. For example, for n = 1000, Algorithm B will require 10000 comparisons rather than the 500000 comparisons with Algorithm A(We note that, for the worst case, the number of comparisons for Algorithm B is the same as for Algorithm A.)

EXAMPLE

Consider again the following list of 15 numbers:

14, 10, 17, 12, 10, 11, 20, 12, 18, 25, 20, 8, 22, 11, 23

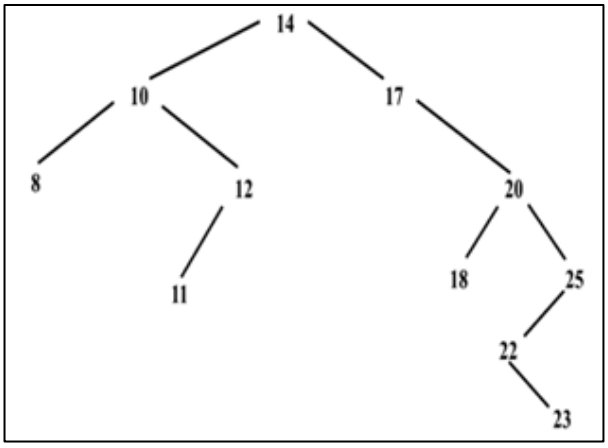


Fig. 18: Binary tree

Applying Algorithm B to this list of numbers, we obtain the tree in fig. 18.

The exact number of comparisons is

$0+1+1+2+2+3+3+3+3+2+4+4+5=38$

On the other hand, Algorithm A requires

0 + 1 + 2 + 3 + 2 + 4 + 5 + 4 + 6+ 7 + 6 + 8 + 9 + 5 + 10 = 72 comparisons.

4.3.4 Deleting in a Binary Search Tree

Suppose T is a binary search tree and suppose an ITEM of information is given. This section given an algorithm which deletes ITEM from the tree T.

The deletion algorithm first uses Procedure 1 to find the location of the node N which contains ITEM and also the location of the parent node P(N). The way N is deleted from the tree depends primarily on the number of children of node N. There are three cases:

Case1:

N has no children. Then N is deleted from T by simply replacing the location of N in the parent node P(N) by the null pointer.

Case 2:

N has exactly one child. Then N is deleted from T by simply replacing the location of N in P(N) by the location of the only child of N.

Case 3:

N has two children. Let S(N) denote the in order successor of N. Then N is deleted

from T first deleting S(N) from T (by using case 1 or case 2) and then replacing node N in T by the node S(N).

Observe that the third case is much more complicated than the first two cases. In all three cases, the memory space of the deleted node N is returned to the AVAIL list.

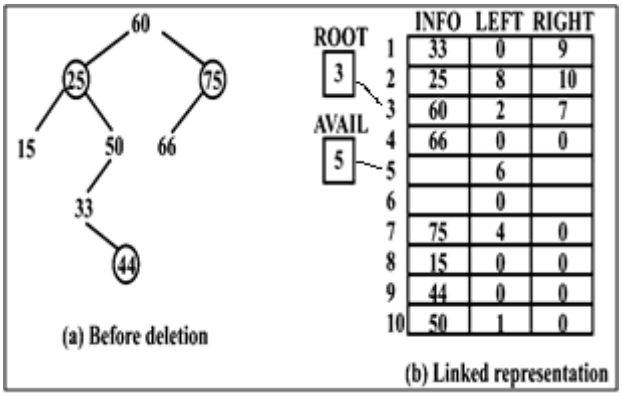


Fig. 19: Linked representation before deletion

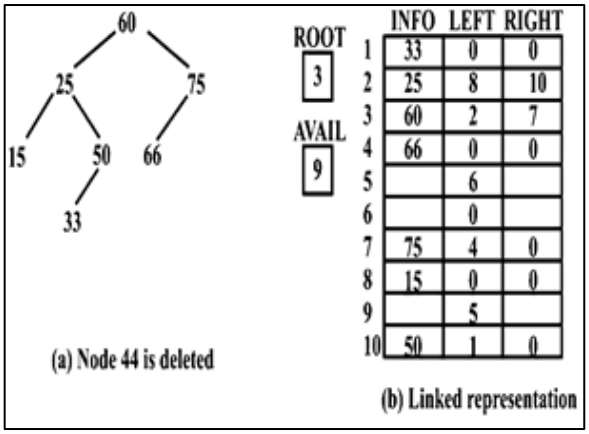


Fig. 20: Linked representation after deletion

Our deletion algorithm will be stated in terms of Procedures 1 and 2 which follow. Procedure 1 refers to cases 1 and 2, where the deleted node N does not have two children and procedure 2 refers to case 3, where N does have two children. There are many sub cases which reflect the fact that N may be a left child, a right child or the root. Also, in case 2, N may have a left child or a right child.

Procedure 2 treats the case that the deleted node N has two children. We note that the in-order successor of N can be found by moving to the right child of N and them moving repeatedly to the left until meeting a node with an empty left sub-ree.

5 HEAP & HEIGHT BALANCED TREE

5.1 HEAP

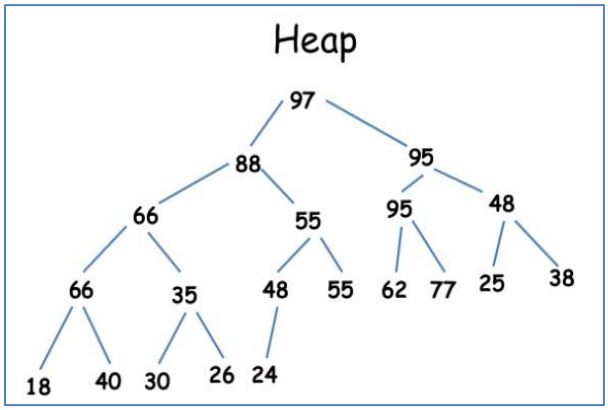


Fig. 1(a): Heap

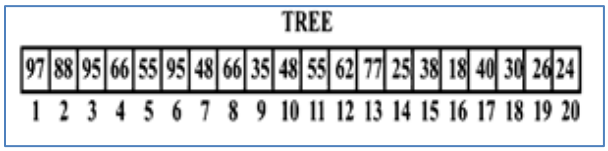


Fig. 1(b): Sequential representation of heap tree

Suppose H is a complete binary tree with n elements. (Unless otherwise stated, we assume that H is maintained in memory by a linear array TREE using the sequential representation of H, not a linked representation.) H is called a heap or a max heap, if each node N of H has the following property; The value at N is greater than or equal to the value at each of the children N. Accordingly, the value at N is greater than or equal to the value at any of the descendants of N. (A minheap is defined analogously; The value at N is less than or equal to the value at any of the children of N.)

Example

Consider the complete tree H in Fig. 1. Observe that H is heap. This means, in particular, that the largest element in H appears at the "top" of the heap, that is, at the root of the tree. Figure 1(b) shows the sequential representation of H by the array TREE. That is, TREE [1] is the root of the tree H and the left and right children of node TREE [K] are, respectively, TREE [2K] and TREE [2K+1]. This means, that the parent of any node TREE [J] is the node TREE [J/2] (where J/2 means integer division).

Observe that the nodes of H on the same level appear one after the other in the array TREE.

5.1.1 Insertion into a Heap:

INSHEAP (TREE, N, ITEM)

A heap H with N elements is stored in the array TREE, and ITEM information is given. This procedure inserts ITEM as a new element of H. PTR gives the location of ITEM as it rises in the tree and PAR denotes the location the parent of ITEM.

1. [Add new node to H and initialize PTR.]
   Set N: = N + 1 and PTR: = N.
2. [Find locations to insert ITEM.]
   Repeat Steps 3 to 6 while PTR < 0.
3. Set PAR: = [PTR/2] [Location of parent node.]
4. If ITEM ≤ TREE [PAR], then:
   Set TREE [PTR]: = ITEM, and Return.
   [End of If structure.]
5. Set TREE [PTR]: = TREE [PAR]. [Moves node down.]
6. Set PTR: = PAR. [Updates PTR.]
   [End of Step 2 loop.]
7. [Assign ITEM as the root of H]
   Set TREE [1]:=ITEM.
8. Return

Observe that ITEM is not assigned to an element of the array TREE until the approximate place for ITEM is found. Step 7 takes care of the special case that ITEM raises to the root TREE [1].

Suppose an array A with N elements is given. By repeatedly applying the above procedure to A, that is, by executing

Call INSHEAP (A, J, A [J+1])

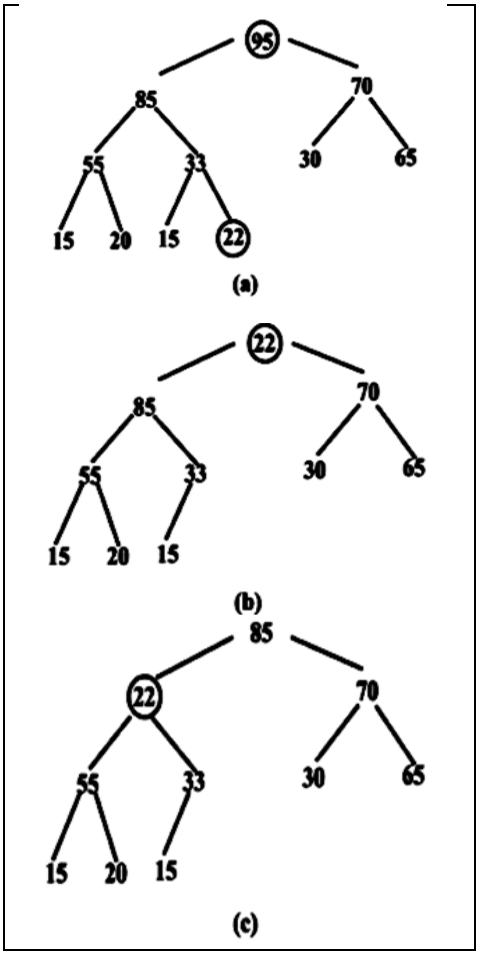
For J = 1, 2,……., N – 1, we can build a heap H out of the array A.

5.1.2 Deleting the Root of a Heap

Suppose H is heap with N elements, and suppose we want to delete the root R of H. This is accomplished as follows:

1. Assign the root R to some variable ITEM.
2. Replace the deleted node R by the last node L of H so that H is still a complete tree, but not necessarily a heap.
3. (Heapify) Let sink to its appropriate place in H so that H is finally a heap. Again we illustrate the way the procedure works before stating the procedure formally.

Example:



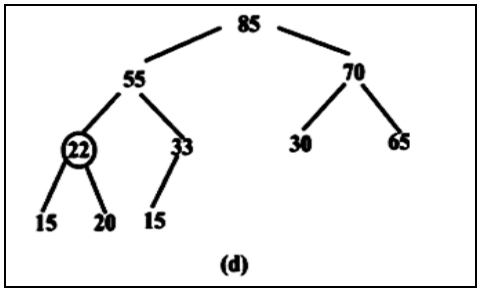


Fig. 2: Heap tree

Consider the heap H in Fig. 2(a), where R = 95 is the root and L = 22 is the last node of the tree. Step 1 of the above procedure deletes R = 95 and Step 2 replaces R = 95 by L = 22. This gives the complete tree in Fig. 2(b), which is not a heap. Observe, however, that both the right and left sub trees of 22 are still heaps. Applying Step 3, we find the appropriate place of 22 in the heap as follows:

a) Compare 22 with its two children, 85 and 70. Since 22 is less than the larger child, 85, interchange 22 and 85 so the tree now looks like Fig. 2(c).

b) Compare 22 with its two new children, 55 and 33. Since 22 is less than the larger child 55, so interchange 22 and 55 and now it's look like Fig. 2(d).

c) Compare 22 with its new children, 15 and 20. Since 22 are greater than both of its children, node 22 has dropped to its appropriate place in H.
Thus fig. 2(d) is the required heap H without its original root R.

Note: As with inserting an element into a heap, one must verify that the above procedure does always yield a heap as a final tree. We also note that Step 3 of the procedure may not end until the node l reaches the bottom of the tree, i. e., until L has no children. The formal statement of our procedure follows.

5.1.2 Application to Sorting

Suppose an array A with N elements is given. The heap sort algorithm to sort A consists of the two following phases:

Phase A:
Build a heap H out of the element of A.

Phase B:
Repeatedly delete the root elements of A. Since the root of H always contains the largest node in H, Phase B deletes the elements of A in decreasing order.

Algorithm:

HEAPSORT (A, N)
An array A with N elements is given. This algorithm sorts the elements of A.

1. [Build a heap H]
   Repeat for J = 1 to N – 1:
   Call INSHEAP (A, J, A [J+1])
   [End of loop.]
2. [Sort A repeatedly deleting the root of H]
   Repeat while N > 1:
   (a) Call DELHEAP (A, N, ITEM).
   (b) Set A [N + 1]: = ITEM.
   [End of Loop]
3. Exit.

The purpose of Step 2(b) is to save space. That is, one could use another array B to hold the sorted elements of A and replace Step 2(b) by Set B [N+1]:= ITEM
However, Step 2(b) does not interfere with the algorithm, since A [N+1] does not belongs to heap H.

5.1.3 Complexity of Heap sort

Suppose the heap sort algorithm is applied to an array A with n elements. The algorithm has two phases and we analyze the complexity of each phase separately.

Phase A
Suppose H is a heap. Observe that the number of comparisons to find the appropriate place of a new elements ITEM in H cannot exceed the depth of H. Since H is a complete tree, its depth is bounded by $\log_2 m$ where m is the number of elements in H. Accordingly, the total number g(n) of comparisons to insert the n elements of A into H is bounded as follows:
$g(n) \leq n \log_2 n$. Consequently, the running time of Phase A of heap sort is proportional to $n \log_2 n$.

Phase B

Suppose H is a complete tree with m elements and suppose that the left and the right sub trees of H are heaps and L is the root of H. Observe that re-heaping uses 4 comparisons to move the node L one step down the tree H. Since the depth of H does not exceed $\log_2 m$, re-heaping uses at most 4 $\log_2 m$ comparisons to find the appropriate place of L in tree H. This means that the total number h (n) of comparisons to delete the n elements of A from H, which requires re-heaping n times, is bounded as follows:

$$h(n) \leq 4n \log_2 n$$

Accordingly, the running time of phase B of heap sort is also proportional to $n \log_2 n$.

Since each phase requires time proportional to n log, n, the running time to sort the n-element array A using heap sort is proportional to $n \log_2 n$, that is, $f(n) = O(n \log_2 n)$. Observe that this gives a worst- case complexity of the heap sort algorithm.

5.2 TREE SEARCHING

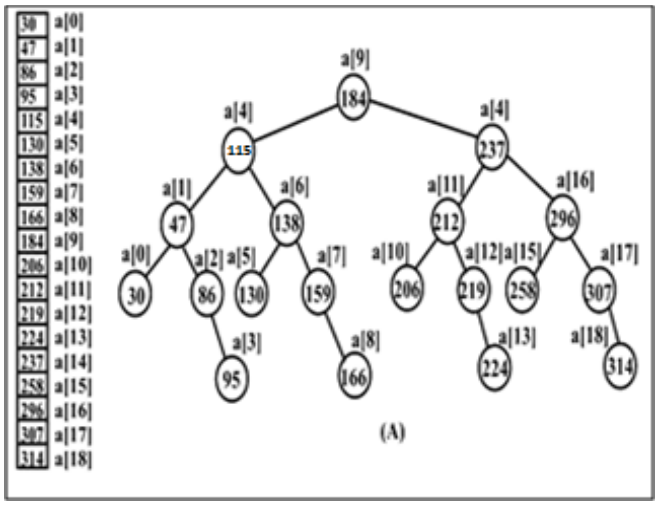


Fig. 3(a): Sorted array and its binary tree representation

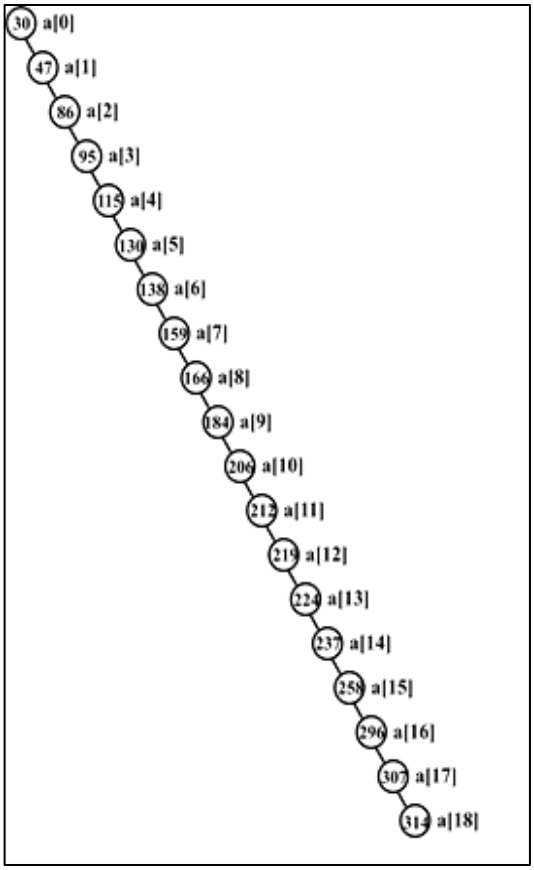


Fig. 3(b): Sorted array and two of its binary tree representations

- Using the binary tree notation, the algorithm for searching for the key `key` is as follows. Here we assume that each node contains four fields
  k: holds the record's key value
  r: holds the record itself
  Left & right: pointers to the sub tree

  Algorithm:

  1. p = tree;
  2. While (p! = null && key! = k(p))
  3. p = (key < k(p))? Left (p) : right (p);
  4. return (p);

- A sorted array can be produced from a binary search tree by traversing the tree in in-order and inserting each element sequentially into the array as it is visited. There are many binary search trees that correspond to a it is visited.
  Viewing the middle element of the array as the root of a tree and observing the remaining elements recursively as left and right subtrees produces a relatively balanced binary search tree as shown in figure above.
  Also, viewing the first element of the array as the root of a tree, and each successive element as the right son of its predecessor produces a very unbalanced binary tree.

5.2.1 Efficiency of Binary search tree operations

- The time required to search an element in binary search tree varies between O(n) and O(log n), depending on the structure of the tree. If the records are inserted in sorted (or reverse) order, the resulting tree contains all null left (or right) links, so that the tree search reduces to a sequential search.
- If the records are presented in random order (i.e. any permutation of n elements is equally likely) balanced tree results more often than not, so that on the average, search time remains O(log n)

Internal path length (i): It is defined as the sum of the levels of all the nodes in the tree.
In the tree shown in figure 4, i= 30.
i. e. 1 node at level 0, 2 at level 1, 4 at level 2, 4at level 3 and 2 at level 4:
1 * 0 + 2 * 1 + 4 * 2 + 4 * 3 + 2 * 4 = 30.

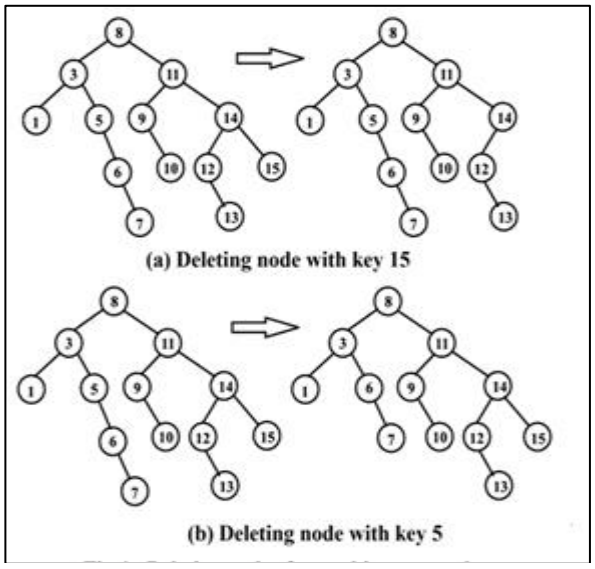


Fig. 4: Deleting nodes from a binary search tree

Since the number of comparisons required to access a node in a binary search tree is one greater than the node`s level, the average number of comparisons required for a successful search in a binary search tree with n nodes equals (i + n)/n (assuming equal like hood for accessing every node in the tree)

Thus, for above tree number of comparisons = $\frac{(30+13)}{13} = 3.31.$

Let S_n equal the average number of comparisons required for a successful search in a random binary search tree of n nodes in which the search argument is equally likely to be any of the n keys and let i_n be the average internal path length of a

Thus, $S_n = \frac{(i_n + n)}{n}$.

Let level of an external node equals the number of comparisons in an unsuccessful search for a key in the range represented by that external node.

Then, if e_n is the average external path length of a random binary search tree of a n nodes.

$$U_n = e_n / (n + 1)$$

This assumes that, each of the (n + 1) keys range equally likely in an unsuccessful search However, it can be shown that e = i+ 2n for any binary tree of n node. Since,

$S_n = (u_n + n) / n$ and $u_n = e_n / (n + 1)$ this implies that

$$S_n = \left(\frac{(n+1)}{n}\right)u_n - 1$$

The average search time in a random binary search tree is O(log n) and on an average , requires approximately only 39% more comparisons than in a balanced binary tree.

5.3 OPTIMUM SEARCH TREES

A binary search tree that minimizes the expected number of comparisons for a given set of keys and probabilities is called optimum.

The fastest – known algorithm to produce an optimum binary search tree is $O(n^2)$ in the general case. But unless the tree is maintained unchanged over a very large number of searches, is too expensive.

STRUCTRUE

However, although an efficient algorithm to construct an optimum tree in the general case does not exist, there are several methods for constructing near – optimum trees in O(n) time. Assume n keys, k(1) through k(n). Let p(i) be the probability of searching for a key k(i) and q(i) the probability of an unsuccessful search between k(1) and k(i) (with q(0) the probability of an unsuccessful search for a key below k(1), and q(n) the probability of an unsuccessful search for a key above k (n). Define s(i, j) as q(i) + q(i+1) +……+q(j).

One method, called the balancing method, attempts to find a value i that minimizes the absolute value of s(0, i-1) – s(i, n) and establishes k(i) as the root of the binary search tree, with k(1) through k(i- 1) in its left sub tree and k(i+ 1) through k(n) in its right subtree. The process is then applied recursively to build the left and right subtrees.

Locating the value i at which abs (s(0, i-1)–s(i, n)) is minimized can be done efficiently as follows. Initially, set up an array s0[n + 1] such that s0[i] equals s(0, i). This can be done by initializing s0[0] to q(0) and s0[j–1] + p(j) + q(j) for j from 1 to n in turn. Once s0 has been initialized, s(i, j) can be computed for any i and j as s0[j] – s0[i- 1] –p(i) whenever necessary. We define si(j) as s0(j-1)–s(j, n). We wish to minimize abs(si(i)).

After s0 has been initialized, we begin the process of finding an i to minimize abs(s0(i-1) – s(i, n)), or si(i). Note that si is a monotonically increasing function. Note also that si(0) = q(0)–1, which is negative, and si(n) = 1–q(n+1), which is positive. Cheek the values of si (1), si(n), si(2), si(n- 1), si(4), si(n-3),……., si(2^j), si(n+1-2^j) in turn until discovering the first positive si(2^j) of the

first negative si(n+1-2^j). If a positive si(2^j) is found first, the desired i that minimizes abs(si(i)) lies within the interval [2^{j-1}, 2^j]; if a negative si (n+1–2^j) is found first, the desired i lies within the interval [n +1-2^j, n+1–2^{j-1}]. In either case, i has been narrowed down to an interval of size 2^{j-1}. Within the interval, use a binary search to narrow down on i. The doubling effect in the interval size guarantees that the entire recursive process is O(n), whereas if a binary search were used on the entire interval[0, n] to start, the process would be 0 (n log n).

A second method used to construct near–optimum binary search trees is done by greedy method. Instead of building the tree from top to down, as in the balancing method, the greedy method builds the tree from the bottom to up. The method uses a doubly linked linear list in which each list element contains four pointers, one key value and three probability values. The four pointers are left and right list pointers used to organize the doubly linked list and left and right sub tree pointers are left and right list pointers used to organize the doubly linked list and left and right sub tree pointers used to keep track of binary search subtrees containing keys less than and greater than the key value in the node. The three probability values are the sum of the probabilities in the left subtree, called the left probabilities, the probability p(i) of the node`s key value fc(i), called the key probability and the sum of the probabilities in the right subtree, called the right probability. The total probability of a node is key value in the i^{th} node is k(i), its left probability is q(i- 1), its right probability is q(i) its key probability is p(i), and its left and right subtree pointers are null.

5.3.1 BALANCED TREES (AVL TREES)

AVL tree sometimes also called Balance Tree.

A Balance Binary Tree is a binary tree in which the height of the two sub trees of every node never differ by more than 1. The balance of a node in a binary tree is defined as the height of its left sub tree minus the height of its right sub tree.

The following figure shows a balance binary tree. Each node in a balanced binary tree has a balance of 1, -1 or 0, depending on whether the height of its left subtree is greater than, less than, or equal to the height of its right subtree. The balance of each node is indicated in figure 5.

Suppose that we are given a balance tree and use the preceeding search and insertion algorithm to insert a new node `p` into the tree. Then the resulting tree may or may not remain balanced.

Figure 5 illustrates all possible insertions that may be made.

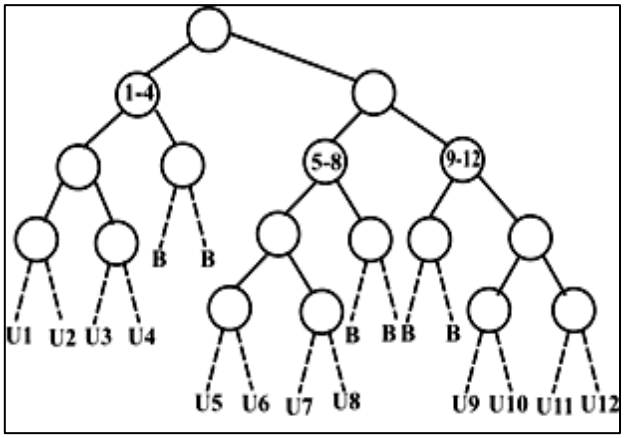


Fig. 5: Possible insertions in tree

Each insertion that yields a balanced tree is indicated by B. The unbalanced insertions are indicated by U and are numbered from 1 to 12.

Tree becomes unbalanced only if the newly inserted node is a left descendant of a node that previously had a balance of 1 or if it is a right descendant of a node that previously had a balance of -1.

The youngest ancestor that becomes unbalanced in each insertion is indicated by the numbers contained in three of the nodes as shown in figure 5.

A newly created node is inserted into left sub tree of B, changing the balance of B to 1 and the balance of A to 2 is as shown in figure 6(a).

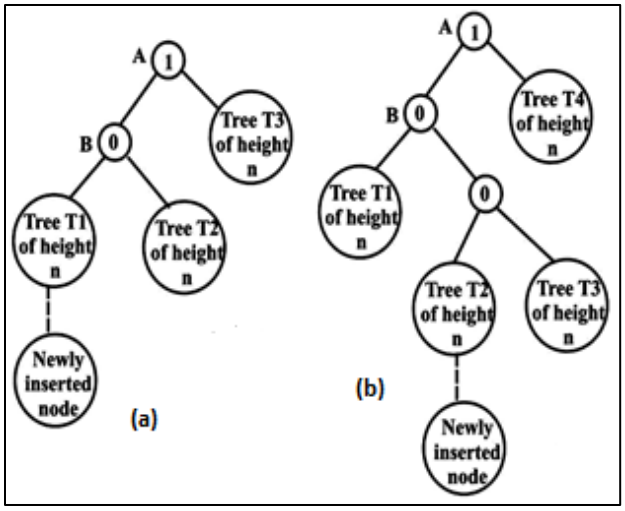


Fig. 6: insertion of a node in left and right sub-tree

Figure 6(b) shows the newly created node is inserted into the right sub tree of B, changing the balance of B to -2 and balance of A to 3.

To maintain a balance tree it is necessary to perform a transformation on the tree so that

i) The in-order traversal of the transformed tree is the same as for the original tree (i.e., the transformed tree remains a binary search).

ii) The transformed tree is balanced.

Let the following fig. 7 shown is original tree.

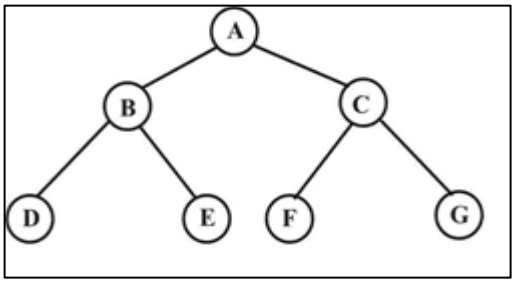


Fig. 7: Original tree

The following two figures show the left rotation and right rotation respectively.

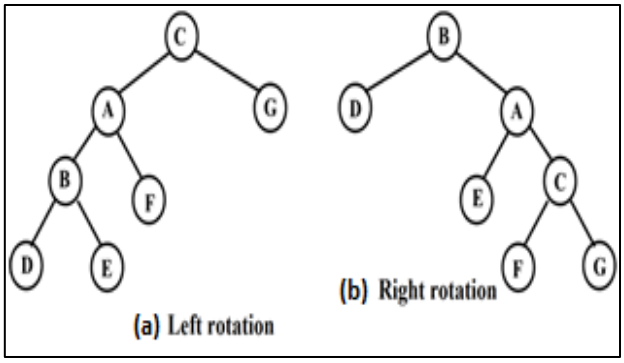


Fig. 8: Trees after left and right rotation

An algorithm to implement a left rotation of a subtree rooted at p is as follows:

```
q = right (p);
hold  = left (q);
left(q) = P;
right (p) = hold;
```

The maximum height of a balanced binary search tree (AVL tree) is 1.44 $\log_2 n$, so that a search in such a tree never requires more than 44 percent more comparisons than that for a completely balance tree.

In actual practice, balance binary search trees behaves even better, yielding search times of $\log_2(n+0.25)$ for large n. On an average, a rotation is required in 46.5 percent of the insertions.

Insertions requires almost a double rotation, deletion may require one (single or double) rotation at each level of the tree or O(log n) rotations. In practice, an average of only 0.214 (single or double) rotations has been found to be required per deletion.

The balance binary search trees that we have looked at are called height balanced tree because their height is used as the criterion for balancing.

There are a number of ways of defining trees. In one method, the weight of a tree is defined as the number of external nodes in the tree. Balanced trees may also be used for efficient implementation of priority queues.

Inserting a new element requires at most O(log n) steps to find its position and O(1) steps to access the elements and O(log n) or O(1) steps to delete that leaf.

5.4 GENERAL SEARCH TREES

General non binary trees are also used a search tables, particularly in external storage. There are two type of such trees: multiday search trees and digital search trees.

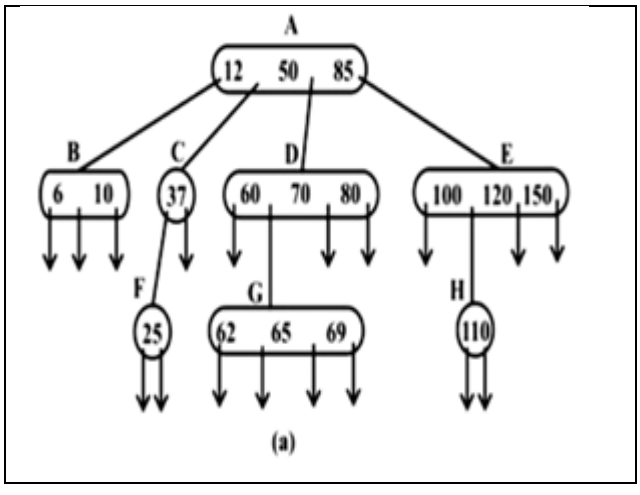
5.5 MULTIDAY SEARCH TREES

A multiday search tree of order n is a general tree in which each node has n or fewer sub trees and contains one fewer key than it has sub trees.

- If a node has four sub trees, it contains tree keys. In additions, if S_0, S_1,……S_{m-1} are the m sub trees of a node containing keys k_0, k_1,……..,k_{m-2} in ascending order, all keys in sub tree S_0 are less than or equal to k_0, all keys in the sub tree S_j (where j is between 1 and m – 2) are greater than k_{j-1} are less than or equal to k_j and all keys in the sub tree S_{m-1} are greater than k_{m-2}.

- The sub tree S_j is called the left sub tree of key k_j and its root is called the left son of key k_j. Similarly S_j is called the right sub tree and its root is the right son of key k_{j-1}.

The following figure 9 illustrates a number of multiday search trees.



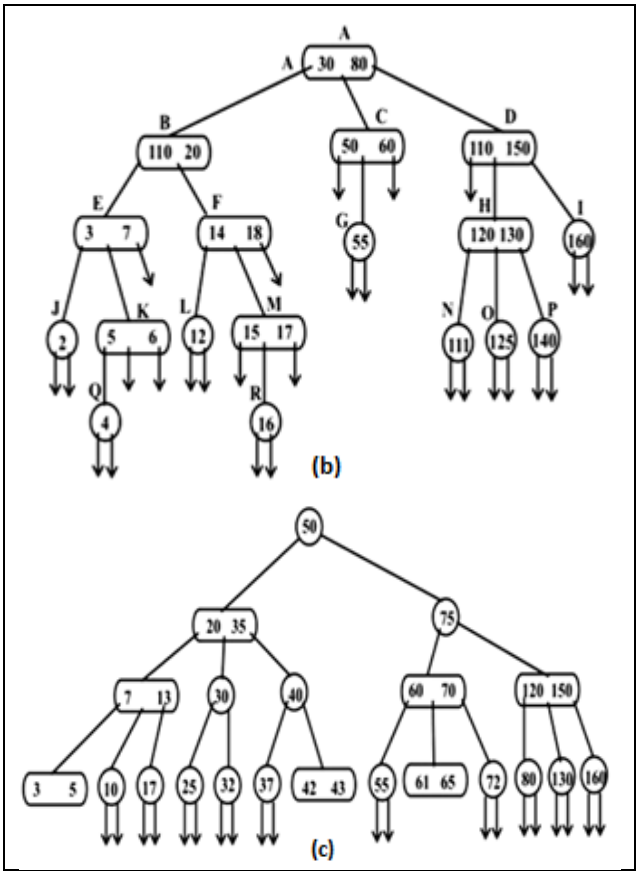


Fig. 9: Multi-day search tree

In fig. 9(a)

- It is a multiday search tree of order 4.
- The 8 nodes of that tree have been labelled A through H, Nodes A, D, E, and G contains the maximum number of sub trees, 4 and the maximum number of keys 3. Such nodes are called full nodes.
- Some of the sub tree of nodes D and E and all of the sub trees of node G are empty, as indicated by arrows emanating from the approximate positions in the nodes.
- Nodes B, C, F and H are not full and also contain some empty sub trees.

In fig. 9(b)

- It shows a top – down multiday search tree of order 3.
- Such a tree is characterized by the condition by the condition that any non-full node is a leaf in fig 9(a).
- Simi leaf defined as a node with at least one empty sub tree. Nodes B through H

in figure 9(a) are all semi leaves. In fig. 9(b) nodes B through G and I through R are semi leaves.

- In a top –down multiday tree, a semi leaf must be either full or a leaf.

In fig. 9(c)

- It is another multiday search tree of order 3.
- It is not top–down, since there are four nodes with only one key and nonempty sub trees.
- It does have another special property in that it is balanced i. e. all its semi leaves are at the same level (3). This implies that all semi leaves are leaves.

5.5.1 Searching a Multiday Tree

The algorithm to search a multiday search tree, regardless of whether it is top-down, balanced or neither, is straightforward. Each node contains a single integer field, a variable number of pointer fields and a variable number of key fields. If node (p) is a node, the integer field numtrees(p) equals the number of sub trees of node(p) numtrees(p)=1) point to the sub trees of node(p). The key fields k(p, 0) through k(p, numtrees(p)-2) are keys contained in node(p) – 2 inclusive) contains all keys in the tree between k(p,i-1) and k(p, i). son (p, 0) points to a sub tree containing only keys less than k(p, 0) and son(p, numtrees(p)–1) points to a sub tree containing only keys greater than k(p,numtrees(p)–2). We also assume a function node search (p, key) that returns the smallest integer; such that key <= k(p, j), or numtrees(p)– 1 if key is greater than all the key in node(p). (We will discuss shortly how node search is implemented.). The following recursive algorithm is for a function search (tree) that returns a pointer to the node containing key (or –1 [representing null] if there is no such node in the tree) and sets the global variable position to the position of key in that node:

```
1. P = tree;
2. If (p = = null) {
3. Position = -1;
4. return (-11); }/* end if */
5. i= node search(p, key);
6. if (i<numtrees(p, key);
7. if (i<numtrees(p) – 1 && key = = k(p, i)) {
8. position = i;
9. return(p); }/* end if */
10. return (search(son(p, i)));
```

Note that after setting i to node search (p, key), we insist on checking the i< numtrees (p) -1 before accessing k(p, i). This is to avoid using the possibly nonexistent or erroneous k(p, numtrees(p) – 1), in case key is greater than all the keys in node(p). The following is a no recursive version of the foregoing algorithm:

```
1. P = tree ;
2. While (p ! = null) {
/* search the sub trees rooted at node(p) */
3. i = node search(p, key);
4. if (i< numtrees(p) – 1 && key = = k(p, i)){
5. position = 1;
6. return(p);} /*end if */
7. p = son(p, i);}/* end while */
8. position = -1;
9. return(-1);
```

The function node search is responsible for locating the smallest key in a node greater than or equal to the search argument. The simplest technique for doing this is a sequential search through the ordered set of keys in the node. If all keys are of fixed equal length, a binary search can also be used locate the appropriate key. The decision whether to use a sequential or binary search depends on the order of the tree, which determines how many keys must be searched. Another possibility is to organize the keys within the node as a binary search tree.

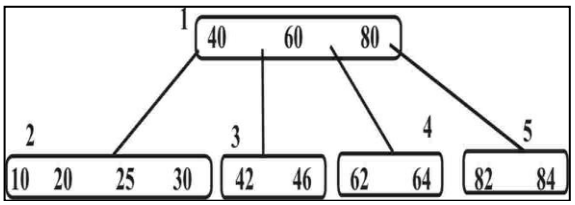
5.6 B-TREE AND B+ TREE

In binary search tree each node contains a single key and points to two sub trees. A B-

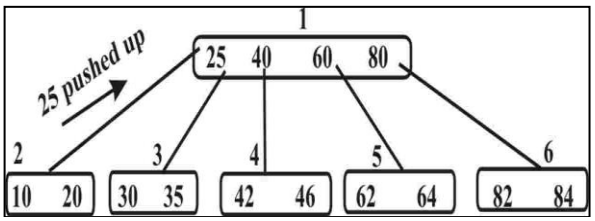
trees is a multiday search tree. A B-tree of order n is a tree in which each non root node contains at least n div 2 keys. A B-tree of order n is also called n–(n -1) tree. This means each node in the tree has a maximum of (n – 1) keys and n sons. Thus 4-5 tree is a B-tree of order 5.

Following diagrams show the insertions process in a B-tree of order 5. When a node becomes full (i.e. has 4 keys) and an attempt is made to insert additional key, a split occurs such that medians of keys is sent up to the parent.

Initial B – Tree:

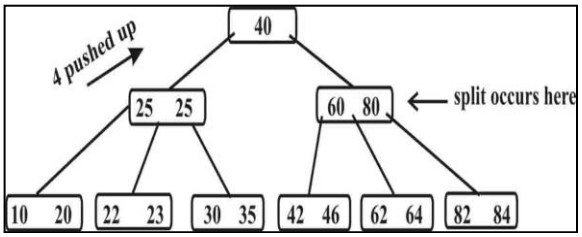


Thus inserting 35in the tree results splitting of node 2



Also inserting 21, 22 will be straight forward. But on insertion of a Key = 23 will again split node 2.

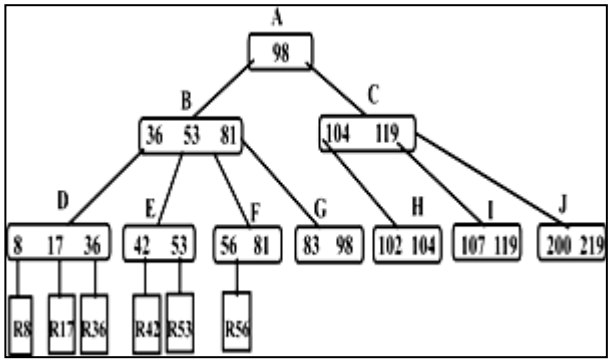
which will in turn split node 1 to push 40.



Note: All the leaves are at same level in a B-tree. Hence it is called Balance - tree.

B + tree:

A major disadvantage of B-tree is the difficulty of traversing the keys sequentially. A variation to B – tree is B+ tree in which all the keys are maintained in the leafs and keys are replicated in the non-leaf nodes to define paths for locating individual records. The leafs are linked together to provide a sequential path for traversing the keys in the tree.



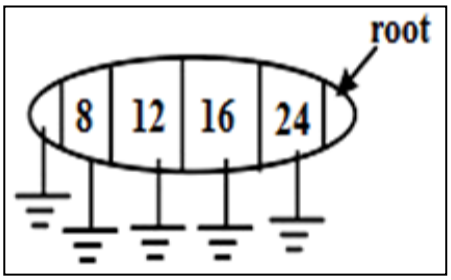
To locate a record associated with key = 53 (random access), the key is first compared with 98. As key is less than 98, left sub tree is visited. 53 is then compared with 36 and then with 53 in node B. Since it is less than or equal to 53 processed to node E where actual record can be found from there on the tree can also be read sequentially. Since record for 53 is found in node E which has a link to node F with keys 56, 81 and so on. Thus B+ tree can search a key randomly and then traverse the tree sequentially.

Construction of B-Tree having order 4

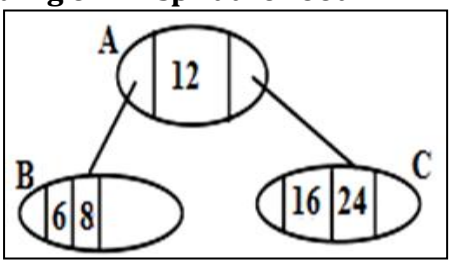
Consider the following key set:

12, 8, 16, 24, 6, 18, 28, 100, 15, 49, 68, 20, 80, 82, 85, 88

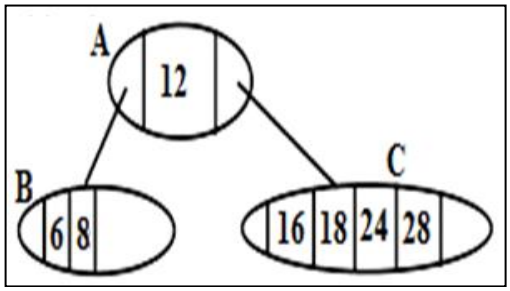
1.



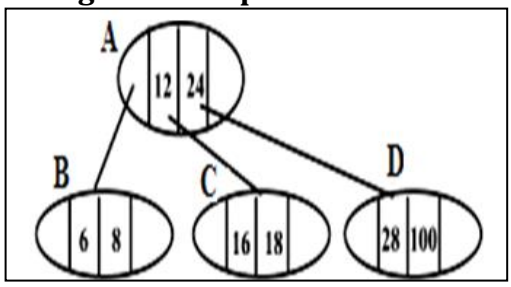
2) Adding 6 will split the root



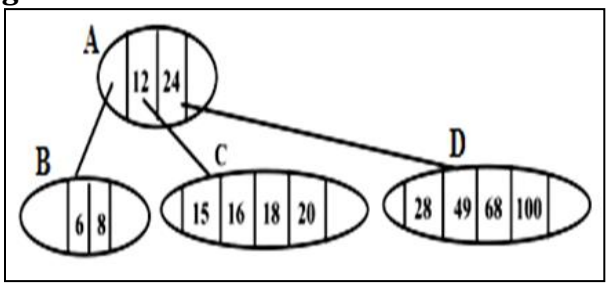
3) 18, 28 are added in node C



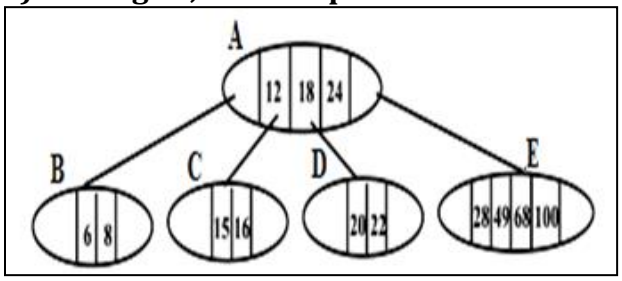
4) Adding 100 will split C



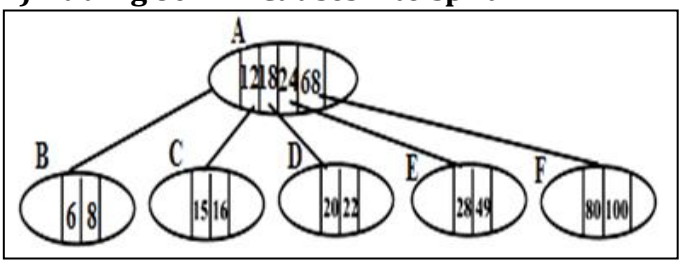
5) 15 goes in C and 49, 68 goes in D, 20 goes in C



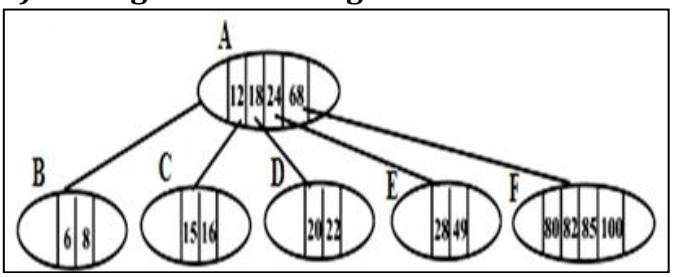
6) Adding 22, Causes split at node C.



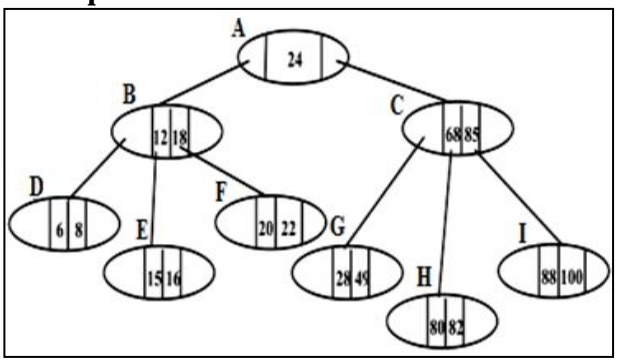
7) Adding 80 will Causes E to split.



8) Adding 82 & 85 changes F



9) 88 will cause F to solit which will in turn split the root node A.



5.7 DIGITAL SEARCH TREES

Another method of using trees to expedite searching is to form a general tree based on the symbols of which the keys are composed. For example, if the keys are integers, then each digit position determines one of ten possible sons of a given node. A forest representing one such set of keys is illustrated in fig. 10. If the keys consist of alphabetic characters, each letter of the alphabet determines a branch in the tree. Note that every leaf node contains the special symbol eok, which represents the end of a key. Such a leaf node must also contain a pointer to the record that is being stored.

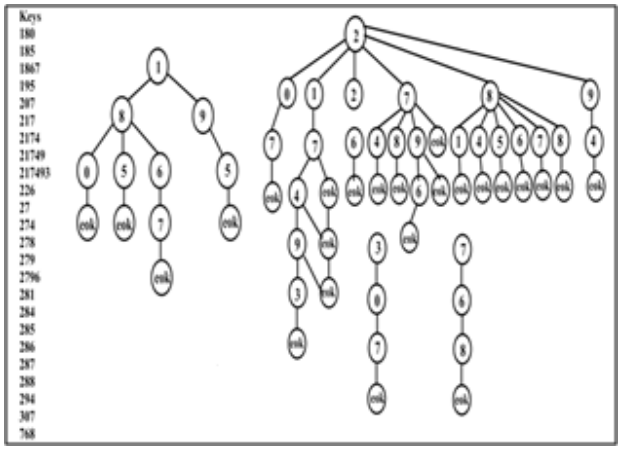


Fig. 10: forest

Using this binary tree representation, we may present an algorithm to search and insert into such a nonempty digital tree. As usual, key is the key for which we are searching and rec is the record that we wish to insert if key is not found. We also let key(i) be the rth symbol of the key. If the key has n symbols, we assume that key(n) equals eok. The algorithm uses the get node operation to allocate a new tree node when necessary. We assume that recptr is a pointer to the record rec to be inserted. The algorithm returns a pointer to the record that is being sought and uses an auxiliary function insert, whose algorithm is also given as-

```
1.  P = tree;
2.  father = null;     /* father is the father of
                          p */
3.  for (f = 0;; f++) {
4.  q=null;  /*q points to the other brother
                 of  p*/
5.  While (p ! = null && symbol (p) < key(i)){
6.  sq = p;
7.  p = brother (p); }/* end while */
8.  if (p = = null | | symbol(p) > key(i)) {
9.  ins val = insert(i, p);
10. return (ins val); }/* end if */
11. if (key(i) = = eok)
12. return(son (p));
13. else {
14. father = p;
15. p = son(p); }/* end else */
16. } /*end for */
```

The algorithm for insert is as follows:

```
/* insert the jth symbol of the key */
1.  s = getnode();
2.  symbol(s) = key(i);
3.  if (tree = = null)
4.  tree = s;
5.  else
6.  if (q ! = null)
7.  Brother(q) = s;
8.  else
9.  (father == null)? tree = s : son(father) = s;
/* insert the remaining symbols of the key */
10. for (j = 7; key (j)! = eok; j++) {
11. father = s;
12. s = get node( );
13. symbol(s) = key(j + 1);
14. son(father) = s;
15. brother(s) = null; }/* end for */
16. son(s) = addr(rec);
17. return(son(s));
```

Note: That by keeping the table of keys as a general tree, we need to search only a small list of sons to find whether a given symbol appears at a given position within the keys of the table. However, it is possible to make the tree even smaller by eliminating those nodes from which only a single leaf can be reached. For example, in the keys of figure 11, once the symbol `7` is recognized the only key that can possibly match is 768. Similarly, upon recognizing the two symbols '1' and '9', the only matching key is 195. Thus the forest of Figure 11 can be abbreviated to the one of figure 11. In figure 11 a box indicates a key and a circle indicates a tree node. A dashed line is used to indicate a pointer from a tree node to a key.

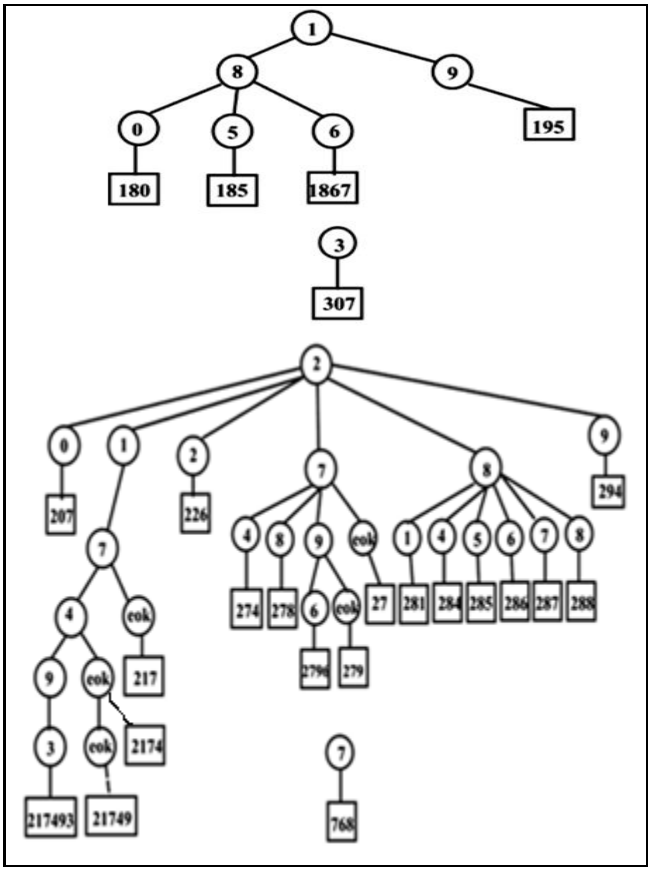


Fig. 11: Condensed forest representing a table of keys

The binary tree representation of a digital search tree is efficient when each node has relatively few sons. The process of searching through the list of sons to match the next symbol in the key is relatively efficient. However, if the set of keys is dense within the set of all possible keys (that is, if almost any possible combination of symbols actually appears as a key), most nodes will have a large number of sons and the cost of the search process becomes prohibitive.

6 INTRODUCTION TO GRAPH THEORY

6.1 GRAPH THEORY TERMINOLOGY

A graph is one type of nonlinear data structure.

Graph and Multi-graphs:

A graph G consists of:

1. A set V of elements called nodes (or points or vertices)
2. A set E of edges such that each edge e in E is identified with a unique unordered pair [u, v] of nodes in V, denoted by e = [u, v].

We denote graph as, G = (V, E), let's assume e [u, v]

- Then the nodes u and v are called the endpoints of e and u and v are said to be adjacent nodes or neighbors.
- The degree of a node u (denoted as deg (u) is the number of edges containing u.

  If deg(u)=0 $\Rightarrow$ if u does not belong to any edge then u is called an isolated node.
- A path P of length n from a node u to a node v is defined as a sequence of n + 1 nodes such that v_{i-1} is adjacent to v_i for i=1 to n.

  P = (v_0, v_1, v_2,.....,v_n)

  Such that u = v_0; v_{i-1} is adjacent to v_1 for i = 1, 2,……..,n; and v_n = v

1. The path P is said to be closed if v_0 = v_n
2. The path P is said to be simple if all the nodes are distinct, with the exception that $v_0$may equal v_n (i.e. P is simple if the nodes v_0, v_1,……..v_{n-1} are distinct and the nodes v_1,v_2,……..v_n are distinct)
3. A cycle is a closed simple path with length 3 or more. A cycle of length k is called a k–cycle.

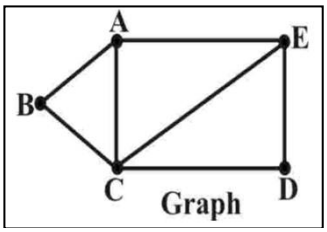
- A graph G is said to be connected if there is a path between any two of its nodes.
- A graph G is said to be complete if every node u in G is adjacent to every other node v in graph G. A complete graph T without any cycle is called a tree graph or simply a tree. Tree is not a complete graph always, this means that, in particular, there is a unique simple path P between any two nodes u and v in T and if T is finite tree with m nodes, then T will have m – 1 edge.
- A graph G is said to be labeled if its edges are assigned with some data.
- G is said to be weighted, if each edge e in G is assigned a nonnegative numerical value w (e) called the weight or length of e.
- In such a case, each path P in G is assigned a weight or length which is the sum of the weights of the edges along the path P. If we are given no other information about weights, we may assume any graph G to be weighted by assigning the weight w (e) = 1 to each edge e in G.
- **Multiple edges:** Distinct edges e and e` are called multiple edges if they connect the same endpoints, i. e. if e = [u, v] and e` = [u, v]. The graph containing multiple edges between two vertices is called a multi-graph.
- **Loops:** An edge e is called a loop if it has identical endpoints, i. e. if e = [u, v]
- A multi-graph M is said to be finite if it has a finite number of nodes and a finite number of edges.
- A graph G with a finite number of nodes must automatically have a finite number of edges and so must

be finite; but this is not necessarily true for a multi-graph M, since M may have multiple edges.

- Distance denoted by d(u, v) between two vertices u and v is defined as the length of the shortest path joining u and v.
- A path is said to be a directed path if all arcs of the path are directed towards the same direction. In this, $v_2e_2v_4e_3v_3$ is a directed path but $v_1e_1v_2e_2v_4$ is not a directed path. Also v_2 and v_4 are adjacent vertices but v_2 and v_3 are not adjacent.

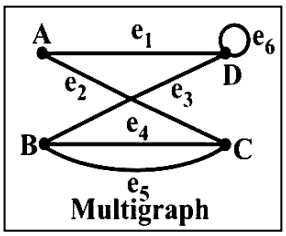
Example to illustrate Terminology:

1.



- It is a connected graph with 5 nodes and 7 edges.
- There are two simple paths of length 2 from B to E :(B, A, E) and (B, C, E).
- There is only one simple path of length 2 from B and D: (B, C, and D).
- There are two 4-cycles in the graph [A, B, C, E, A] and [A, C, D, E, A]
- deg(A) = 3, since A belong to 3 edges
  deg(C) = 4,
  deg(D) = 2

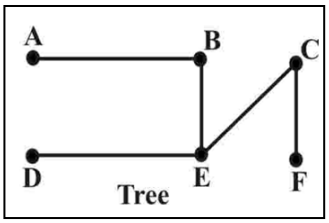
2.



- It is not a graph but a multi-graph
- It has multiple edges e_4 = [B, C] and e_5 = [B, C] and it has a loop, e_6 = [D, D]

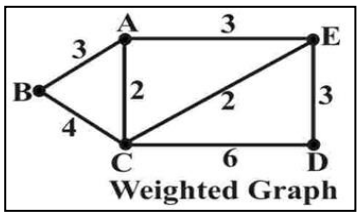
(From the definition of a graph usually does not allow either multiple edges or loops)

3.



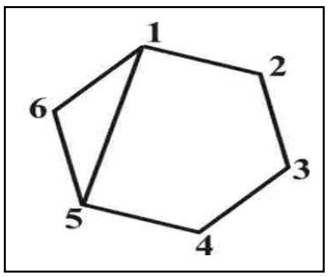
- A tree graph with 6 nodes and 5 edges
- There is a unique simple path between any two nodes of the tree graph.

4.



- This graph is same as that of Graph shown in Example 1 except that now the graph is weighted.
- P_1 = (B, C, D) and P_2 = (B, A, E, D) are both path from node B to node D. Although P_2 contains more edges than P_1, the weight $W(P_2)=9$ is less than the weight $W(P_1) = 10$.

5.



This graph has 3 cycles of lengths 3, 5, 6

Cycles	Length
1-5-6-4	3
1-2-3-4-5-1	5
1-2-3-4-5-6-1	6

6.2 DIRECTED GRAPH:

Directed graph G, also called a diagraph or graph, is the same as a simple-graph except that each edge e in G is assigned a direction or in other way, each edge e is identified with an ordered pair (u, v)of nodes in G.

- Suppose G is a directed graph with a directed edge e = (u, v). Then e is also called an arc. The following terminology is used with respect to directed graph-
  a) u is the origin or initial point of e and v is the destination or terminal point of e.
  b) u is predecessor of v and v is a successor or neighbour of u.
  c) u is adjacent to v but v is not adjacent to u.
- The out degree of a node u in G (denoted as outdeg(u)) is the number of edges beginning at u. The in degree of u (denoted as indeg(u)), is the number of edges ending at u.
- A node u is called a source if it has a positive out degree but zero in degree. Similarly, u is called a sink if it has a zero out degree but a positive in degree.
- A node V is said to be reachable from a node u if there is a (directed) path from u to v.
- A directed graph G is said to be connected or strongly connected if for each pair u, v of nodes in G there is a path from u to v and there is also a path from v to u.
- A graph G is said to be unilaterally connected if for any pair u, v of nodes in G there is a path from u to v or a path from v to u.

Example:

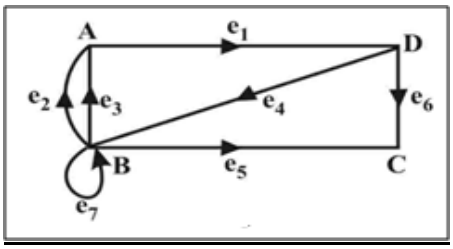


Fig. 1: Directed Graph

The graph in fig. 1 consists of:

1. 4 nodes and 7 (directed) edges
2. The edges e_2 and e_3 are said to be in parallel, since each begins at B and ends at A.
3. The edge e_7 is a loop, since it begins and ends at the same point B.
4. The sequence P_1 = (B, C, B, A) is not a path, since (C, B) is not an edge, i. e., the direction of the edge e_5 = (B, C) does not agree with direction of the path P_1 on the other hand P_2 = (D, B, A) is a path from D to A, since (D, B) and (B, A) are edges. Thus A is reachable from D.
5. There is no path from C to any other node, so G is not strongly connected. However, G is unilaterally connected.
   Also, indeg(D) = 1 and outdeg(D) = 2
   Node C is a sink, since indeg(C) = 2 but outdeg(C) = 0. No node in G is a source.

6.2.1 Advantage of Directed Graph:

- Let T be any nonempty tree graph. Suppose we choose any node R in T. Then T with this designated node R is called a rooted tree and R is called its root. This defines a direction to the edges in T, so the rooted tree T may be viewed as a directed graph.
- Suppose we also order the successors of each node v in T. Then T is called an ordered rooted tree. Ordered trees are almost same as the general trees.
- A directed graph G is said to be simple if G has no parallel edges. A simple graph G may have loops, but it cannot have more than one loop at a given node.

- A non-directed graph G may be viewed as a simple directed graph by assuming that each edge [u, v] in G represents two directed edges (u, v) and (v, u)

6.3 IN DEGREES AND OUT DEGREES OF VERTICES OF A DIGRAPH:

Consider the following graph -

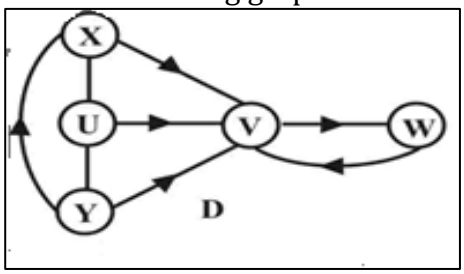


Fig. 2: In-degree and Out-degree

Let us consider a vertex U of a digraph D. The in degree of U is defined as the number of arcs for which U is head and out degree is the number of arcs of which u is the tail.

Note: $\text{indeg}(U) \Leftrightarrow d_D^+(U) \Rightarrow$ indegree of U in graph D

$\text{Outdeg}(U) \Leftrightarrow d_D^-(U) \Rightarrow$ out degree of U in graph D

6.4 NULL GRAPH:

A graph is said to be null if all its vertices are isolated.

6.5 FINITE GRAPHS:

A multi-graph is said to be finite if it has a finite number of vertices and a finite number of edges.

Note: A simple graph with a finite number of vertices must automatically have a finite number of edges and so must be finite.

6.6 TRIVIAL GRAPH:

The finite graph with one vertex and no edges i. e. a single point is called the trivial graph.

6.7 SUBGRAPHS:

Consider a graph G = G(V, E). A graph H = H(V', E') is called a sub-graph of G if the vertices and edges of H are contained in the vertices and edges of G, i.e. if $V` \subseteq V$ and $E` \subseteq E$.

6.7.1 Advantage of Sub-graphs

1) A sub-graph H(V`, E`) of G(V, E) is called the sub-graph induced by its vertices V` if its edges set E` contains all edges in G whose endpoints belong to vertices in H.
2) if v is vertex in G, then G – v is the sub-graph of G obtained deleting v from G and deleting all edges in G which contain v.
3) If e is an edge in G, then G – e is the sub-graph of G obtained by simply deleting the edge from G.

6.8 SEQUENTIAL REPRESENTATION OF GRAPHS, ADJACENCY MATRIX, PATH MATRIX

There are two ways of maintaining a graph G in the memory of a computer.

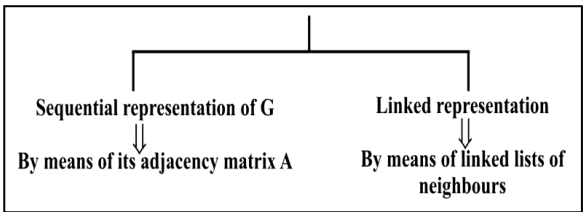


Fig. 3: Graph representation

6.8.1 Adjacency Matrix:

Suppose G is simple directed graph with m nodes and suppose the nodes of G have been ordered and are called $v_1, v_2, \ldots\ldots v_m$. Then the adjacency matrix A = (a_{ij}) of the graph G is the m × m matrix defined as follows:

$$a_{ij} \begin{cases} 1 \Rightarrow \text{if } v_i \text{ is adjacent to } v_j, \text{ i.e. if there is an edge } (v_i, v_j) \\ 0 \Rightarrow \text{otherwise} \end{cases}$$

Such a matrix A, which contains entries of only 0 and 1, is called a bit matrix or a Boolean matrix.

- The adjacency matrix A of the graph G does depend on the ordering of the nodes of G i.e. a different ordering of the nodes may result in a different adjacency matrix.

  Note: The matrices resulting from two different ordering are closely related in that one can be obtained from the other by simply interchanging rows and columns.
- Let G be an undirected graph, then the adjacency matrix A of G will be a symmetric matrix (i. e. one in which $a_{ij} = a_{ji}$ for every i and j). Each undirected edge [u, v] corresponds to the two directed edges (u, v) and (v, u).
- The above matrix representation of a graph may be extended to multi-graph. If G is a multi-graph, then the adjacency matrix of G is the m × m matrix $A = (a_{ij})$ defined by setting a_{ij} equal to the number of edges from v_i to v_j.

Example:

Consider the graph G, suppose the nodes stored in memory a linear array `X` as follows:

X : A, B, C, D

Then we assume that the ordering of nodes in G is as follows:

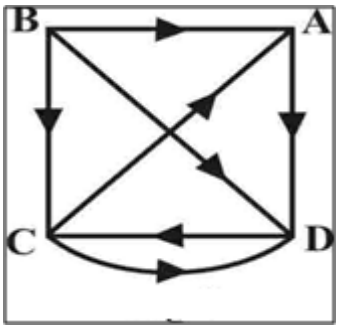


Fig. 4: Graph

V_1 = A, V_2 = B, V_3 = C, v_4= D. The adjacency matrix M of G is as follows:

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Note: The number of 1's in A is equal to the number of edges in G.

Consider the powers M, M^2, M^3……of the adjacency matrix A of a graph G.

Let m_k (i, j) = the ij entry in the matrix M^k

Note: $m_1(i, j) = m_{ij}$ gives the number of paths of length 1 from node v_i to v_j, Also $m_2(i, j)$ gives the number of paths of length 2 from v_i to v_j.

6.8.2 Path Matrix:

Let G be a simple directed graph with m nodes, $v_1, v_2, \ldots v_m$. The path matrix or reach ability matrix of G is the m-square matrix P = (P_{ij}) defined as follows:

$$P_{ij} = \begin{cases} 1 \Rightarrow \text{if there is a path from } v_i \text{ to } v_j \\ 0 \Rightarrow \text{otherwise} \end{cases}$$

- Suppose there is a path from v_i to v_j. Then there must be a simple path from v_i to v_j when $v_i \neq v_j$ or there must be a cycle from v_i to v_j when v_i to v_j when $v_i = v_j$. Since G has only m nodes, such a simple path must have length m – 1 or less or such a cycle must have length m or less.
- A directed graph G is said to be strongly connected if, for any pair of nodes u and v in G, there is a path from u to v and also a path from v to u. Accordingly, G is strongly connected iff the path matrix P of G has no zero entries. Thus, the graph shown in fig. 4 is not strongly connected.

6.8.3 Transitive Closure:

The transitive closure of a graph G is defined to be graph G` has the same node as G and there is an edge $(v_i, v_j)$ in G`. Whenever there is a path from v_i to v_j in G.

The path matrix P of the graph G is precisely the adjacency matrix of its transitive closure G. A graph G is strongly

connected if its transitive closure is a complete graph.

WARSHALL'S ALGORITHM

Warshall's algorithm is an efficient way of finding the matrix P of the graph G. (where G is the directed graph with m nodes v_1, v_2,……,v_m.

A directed graph G with M nodes is maintained in memory by its adjacency matrix. This algorithm finds the (Boolean) path matrix P of the graph G.

ALGORITHM

We initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, ... k-1} as intermediate vertices. For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

1) k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.

2) k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j].

6.9 SHORTEST PATH ALGORITHM

- Let G be a directed graph with m nodes v_1, v_2,……..,v_m. Suppose G is weighted i. e. suppose each edge e in G in assigned a nonnegative number w(e) called the weight or length of the edge e.
- G may be stored in memory by its weight matrix, W = (W_{ij}), defined as follows:

$$W_{ij} = \begin{cases} W(e) & \text{if there is an edge e from } v_i \text{ to } v_j \\ 0 & \text{if there is no edge from } v_i \text{ to } v_j \end{cases}$$

The path matrix P tells us whether or not there are paths between the nodes. If we want to find a matrix Q which will tell us the lengths of the shortest paths between the nodes or, more exactly a matrix Q = (q_{ij}) where q_{ij} = length of a shortest path from v_i to v_j

Algorithm:

A weighted graph G with M nodes is maintained in memory by its matrix W. This algorithm finds a matrix Q such that Q [I, J] is the length of a shortest path from node V_I to V_J. INFINITY is a very large number, and MIN is the minimum value function:

1. Repeat for I, J = 1, 2,……..,M : [Initializes Q] W[I, J] = 0, then : Set Q[I, J]: = INFINITY;
   [End of loop]
2. Repeat Step 3 and 4 for K = 1, 2,……..,M : [Updates Q]
3. Repeat Step 4 for I = 1, 2,……..,M:
4. Repeat for J = 1, 2,……..,M:
   Set Q[I, J] : = MIN (Q[I, J], Q[I, K] + Q[K, J]).
   [End of loop.]
   [End of Step 3 loop.]
   [End of step 2 loop.]
5. Exit.

6.10 LINKED REPRESENTATION OF A GRAPH

Need for Linked representation of a graph:

Let G be a directed graph with m nodes. The sequential representation of G in memory i.e. the representation of G by its

adjacency matrix A has some disadvantages.

i) It may be difficult to insert and delete nodes in G. Because the size of A may need to be changed and the nodes may need to be changed and the nodes may need to re-ordered. So there may be many, many changes in the matrix A.

ii) Also, if the number of edges is O(m) or O(mlog$_2$ m), then the matrix A will be sparse(i. e. it will contain many zeros) hence more space is wasted. To avoid this, G is usually represented in memory by linked representation, also called an adjacency structure.

- **Consider the graph G in the following figure 5(a).**

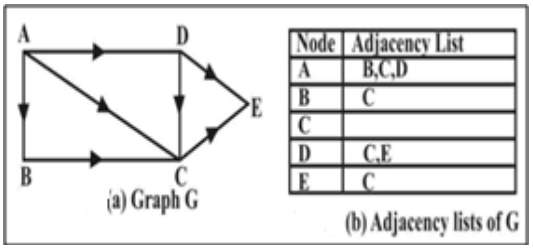


Fig. 5: Graph with adjacency list representation

The table in fig. 5(b) Shows, each node in G followed by its adjacency list, which is its list of adjacent nodes, also called its successors or neighbours.

- The following fig. 6(c) and 6(d) shows a linked representation of G in memory. Specifically, the linked representation will contain two lists (or files)
  1. A node list NODE
  2. An edge list EDGE

6.10.1 Node List:

Each element in the list NODE will correspond to a node in G and it will be a record of the form

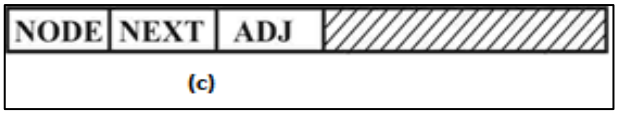


Fig. 5(c): NODE list linked representation of G in memory

NODE: The name or key value of the node
NEXT: A pointer to the next node in the list NODE
ADJ: A pointer to the first element in the adjacency list of the node, which is maintained in the list EDGE. Shaded Area: Indicates that there may be other information in the record, such as the in degree during the execution of an algorithm, and so on.
Note: One may assume that NODE is an array of records containing fields such as NAME, INDEG, OUTDEG, STATUS, etc.

6.10.2 Edge List:

Each element in the list EDGE will correspond to an edge of G and will be a record of the form:

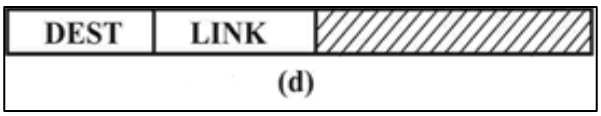


Fig. 5(d): EDGE list linked representation of G in memory

DEST: It will point to the location in the list NODE of the destination or terminal node of the edge.

LINK: It will link together the edges with the same initial node i. e. the nodes in the same adjacency list.

Shaded Arcs:
It indicates that there may be other information in the record corresponding to the edge, such as afield EDGE containing the labelled data of the edge when G is a labelled graph, a field WEIGHT containing the weight of the edge when G is a weighted graph and so, on.
AVAIL:
We need a pointer variable AVAIL for the list of available space in the list EDGE. The

graph G shown in fig 5(a) may appear in memory as shown in figure 6.

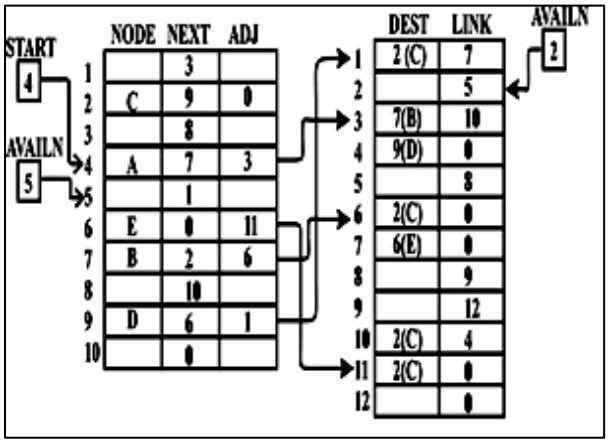


Fig. 6: Graph G in fig 5(a) with pointer AVAIL

6.11 GRAPH TRAVERSAL

Graph traversal is the systematically assessment of all the nodes of a graph. It is always possible to traverse a graph efficiently by visiting the graph nodes in an implementation dependent manner.

Graph traversal is more complex than for a list or a tree:

1) There is no natural "first" node in a graph from which the traversal should start, as there is a first node in a list or a root in a tree. Also, once a starting node has been determined and all nodes reachable from that node have been visited, there may remain other nodes in a graph that have not been visited because they are not reachable from the starting node. Thus, once all reachable nodes in a graph have been visited, the traversal algorithm again has the problem of selecting another starting node.

2) There is no natural order among the successors of a particular node. Thus there is no prior order in which the successor of a particular node should be visited.

3) Unlike a node of a list or a tree, a node of a graph may have more than one predecessor. If node x is a successor of both nodes y and z, x may be visited after u but before z. It therefore possible for a node to be visited before one of its predecessor. Also, if a graph is cyclic, every traversal must include some node that is visited before one of its predecessors.

6.12 SPANNING FORESTS:

Forest:
It may be defined as an acyclic graph in which every node has one or no predecessors.

Tree:
A tree may be defined as a forest in which only a single node (i. e. root) has no predecessors. An ordered Forest is one whose component trees are ordered.

Spanning Forest:

If G is a graph, then F is a spanning forest of G if

1. F is a sub graph of G containing all the nodes of G.
2. F is an ordered Forest containing tree $T_1, T_2, \ldots T_n$.
3. T_i contains all nodes that are reachable in G from the root of T_i are not contained in T_j for some $j < i$. F is a spanning tree of G if it is a spanning forest of G and consists of a single tree.

Example:

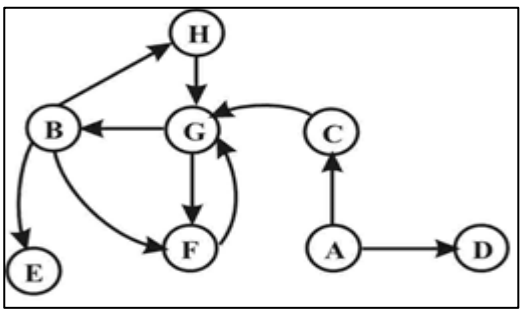


Fig. 7: Graph G

Possible Spanning Forests

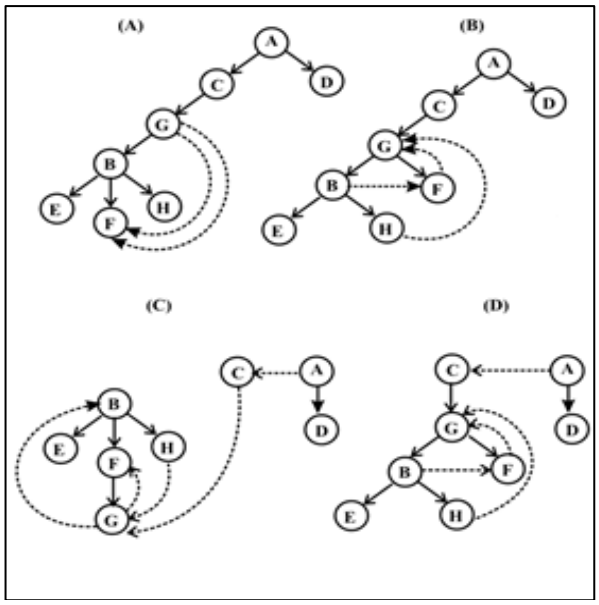


Fig. 8: different spanning trees for Graph G

Note: In each forest the arcs of the graph that are not included in the forest are shown as dotted arrows and the arcs included in the forest as solid arrows.
Any spanning tree divides the edges of a graph into four groups:

1) Tree edges: These are arcs of the graph that are included in the spanning Forest.
2) Forward edges: These are arcs of the graph from a node to a spanning forest non-son descendant.
3) Cross edges: It is an arc from one node to another node that is not the first node is descendant or ancestor in the spanning forest.
4) Back edges: These arcs from a node to a spanning forest ancestor.
Consider a method that visits all nodes reachable from a previously node before visiting any node reachable from a previously visited node. In such traversal a node is visited either arbitrarily or as the successor of a previously visited node.
The traversal defines a spanning forest in which an arbitrarily selected node is the root of a tree in the spanning forest and in which a node n_1 selected as the successor of n_2 is a son of n_2 in the spanning forest.

6.13 UNDIRECTED GRAPHS AND THEIR TRAVERSALS:

- An undirected graph may be represented as a directed graph by using either the adjacency matrix or adjacency list method., A>
<B must exist whenever an arc <A, B> exists.
- The undirected arc (A, B) represents the two directed arcs <A, B> and <B, A>.
- An adjacency matrix representing an undirected graph must be symmetric i. e.value in row i, column j and in row j, column I must be either both false. (i. e. the arc (i. j.) does not exist in the graph) or both true (the arc (i, j) does exist).
In the adjacency list representation, if (i, j) is an undirected arc, the arc list emanating from node (i) contains a list node representing directed are <i, j> and the list emanating from node (j) contains a list node representing directed arc <i, j> and the list emanating from node (j) contains a list node representing directed arc <j, i>. In an undirected graph, if a node x is reachable from a node y (i.e. there is a path from y to x), y is reachable from x as well as along the reversal path.

An undirected graph is represented by a directed graph, any traversal method for undirected graphs as well.

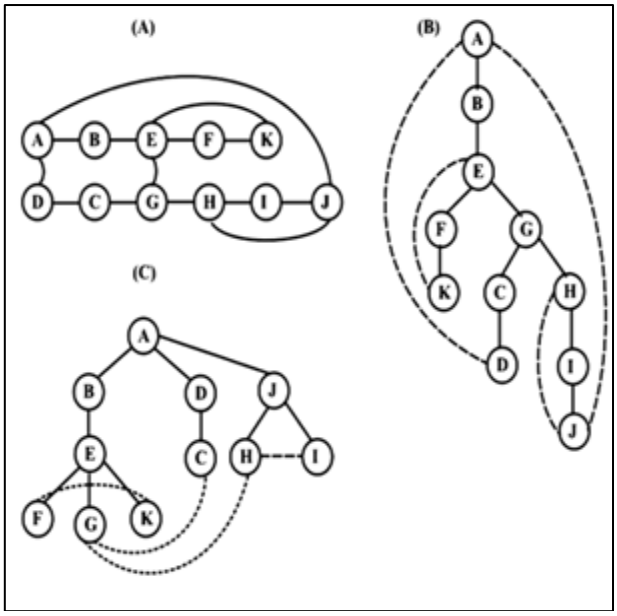


Fig. 9: Undirected graph with it's spanning trees

Example:
The figure 10 shows, an undirected graph and two spanning trees for that graph. The tree in (b) is formed by either of the traversal ABEFGKCHDIJ or ABEFKGCDHIJ among others. The tree in figure 10 (c) is created by the traversals ABDJECHIFGK or ABEFGKDCJHI.

Spanning Forest constructed by undirected graph traversal has several special properties-

1. There is no distinction between forward edges (or tree edges) and back edges.
   Note: In an undirected graph, any are between a node and its non-son descendant is called back edge.
2. In an undirected graph all cross edges are within a single tree. Cross edges between trees arise in a directed graph traversal when there is an arc <x, y> such that y is visited before x and there is no path from y to x. Therefore, there <x, y> is a cross edge.
3. In an undirected graph containing an arc (x,y), x and y must be part of the same tree, since each is reachable from the other via that arc at least.
5. A cross edge in an undirected graph is possible only if there nodes x, y and z are part of a cycle and y and z are in separate sub-trees of a sub-tree whose root is x.
5. The path between y and z must then include across edge between the two sub trees.
   Note: Spanning Forests for an undirected graph have double the edges of directed graph, therefore forests tends to have fewer but larger trees.

6.13.1 Terms related to undirected graphs

1) Connected:
An undirected graph is connected if every node in it, is reachable from every other node.

Example:

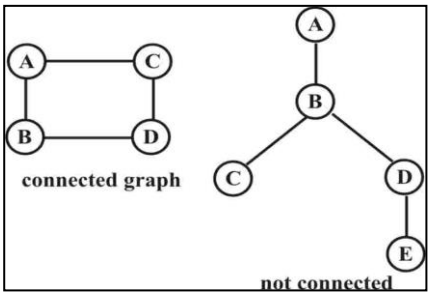


Fig. 10:Connected and not connected graph

2) Connected Component:
A connected component of all undirected graph is a connected sub graph containing all arcs incident to any of its such that no graph node outside the sub graph reachable from any node in the sub graph.

Example:

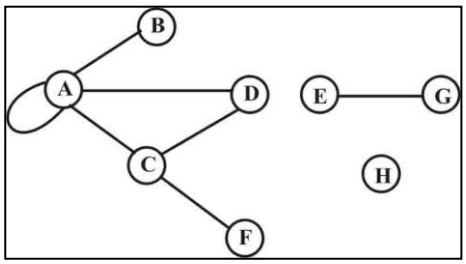


Fig. 11: connected components of Graph

It has there connected components:
→ nodes A, B, C, D, F
→ nodes E and G
→nodes H
A connected graph has a single connected component.

3) Spanning Forest:

The spanning Forest of a connected graph is a spanning tree. Each tree in the spanning forest of an undirected graph contains all the nodes in a single connected component of a graph.

6.13.2 Traversing Methods:

Many graph algorithms require one to properly observe the nodes and edges of a graph G. There are two standard ways of doing this:

1) Breadth – First search or breadth – first traversal
2) Depth – First search or depth – first traversal.
Breadth First search will use a queue as an auxiliary structure to hold nodes for future processing and the depth first search will use a stack. During the execution of an algorithm, each node N of G will be in one of three states, called the status of N as follows:

STATUS = 1: (Ready state) The initial of N as follows:
STATUS =2: (Waiting state) The node N is on the queue or stack, waiting to be processed.
STATUS = 3: (Processed state). The node N has been processed.

1) Breadth First Search:
A breadth first search beginning at a starting node A is as follows:
→ First we examine the starting node A
→ Then we examine all the neighbor of A
→ Then we examine all the neighbors of the neighbors of A and so on. We need to keep track of the neighbors of a node and we need to guarantee that no node is processed more than once. This is accomplished by using a queue to hold nodes that are waiting to be processed and by using a field STATUS which tells us the current status of any node.

Algorithm:

This algorithm executes a breadth –first search on a graph G beginning at a starting node A.

1. Initialize all nodes to the ready state (STATUS = 1).
2. Put the starting node A in QUEUE and change its status to the waiting state (STATUS= 2).
3. Repeat Steps 4 and 5 until QUEUE is empty:
4. Remove the front node N of QUEUE. Process N and change the status of N to the processed state (STATUS = 3)
5. Add to rear of QUEUE all the neighbors of N that are in the steady state (STATUS = 1), and change their status to the waiting state(STATUS = 2).
[End of Step 3 loop.]
6. Exit.

This algorithm will process only those nodes which are reachable from the starting node A. Suppose we wants to examine all the nodes in the graph G. Then the algorithm must be modified so that it begins with another node (let say B) that is still in the ready state. This node B can be obtained by traversing the list of nodes.

2) Depth – First search /Depth first traverse

A depth – first search beginning at a starting node A is as follows:
→ First we examine the starting node A
→ Then we examine each node N along a path P which begins at A i.e. we process a neighbor of A, then a neighbor of A, then a neighbor of a neighbor of A and so on.
→ After coming to a "dead end" i.e. the end of the path P, We back track on P until we can continue along another path P and so on.

Note: The algorithm is very similar to the breadth – first search except that we now use a stack instead of the queue. A field STATUS is used to tell us the current status of anode.

Algorithm:
This algorithm executes a depth – first search on a graph G beginning at a starting node A.

1. Initialize all nodes to the ready state (STATUS= 1).
2. Push the starting node A onto STACK and change its status to the waiting state (STATUS = 2).
3. Repeat Steps 4 and 5 until STACK is empty.
4. Pop the top node N of STACK. Process N and change its status to the processed state (STATUS = 3).
5. Push onto STACK all the neighbors of N that are still in the ready state (STATUS =1), and change their status to the waiting state (STATUS = 2).
   [End of Step 3 loop.]
6. Exit.

The above algorithm will process only those nodes which are reachable from the starting node A. Suppose we wants to examine all the nodes in G. Then the algorithm must be modified so that it begins again with another node (say node B) that is still in ready state. This node B can be obtained by traversing the list of nodes.

6.13.3 Applications of Depth – First traversal

1. Depth first traversal creates a spanning forest than can be used to determine if an undirected graph is connected and to identify the connected components of an undirected graph.
2. Depth First traversal can also be used to determine if a graph is acyclic. In both directed and undirected graphs, a cycle exists if a back edge exists in a depth first spanning forest.
3. Depth First traversal can be used to produce are verse topological ordering of the nodes.
4. An algorithm to determine a reverse topological ordering of the nodes of DAG consists of a depth first search of the DAG followed by an inorder traversal of the resulting spanning forest.

6.13.4 Efficiency of Depth–First traversal

The depth First traversal routine visits every node of a graph and traverses all the successors of each node.

For adjacency matrix implementation

$\rightarrow$Traversal all successors routine of a node $\Rightarrow O(n)$
$\rightarrow$Traversing the successors of all the nodes $\Rightarrow O(n^2)$
$\therefore$ Depth–first search using the adjacency matrix representation is $O(n + n^2)$ (n node visits and n^2 possible successor examination)= $O(n^2)$ (By sum Rule)
Note: Where n is the number of graph nodes.

For adjacency list representation

- $\rightarrow$ Traversing all successors of all nodes $\Rightarrow O(e)$
- $\rightarrow$ Assuming that the graph nodes are organized as an array or a linked list, visiting all n nodes is O(n), so that the efficiency of depth – first traversal using adjacency list is O(n + e).Since e is usually much smaller than n^2, the adjacency list representation yields more efficient traversal.
- $\rightarrow$ Depth first traversal is often considered O(e),since e is usually larger than n.

Note: e is the number of edges in a graph.

Application of breadth traversal

1. Breadth first traversal can be used for some of the same application as depth first traversal. In particular breadth first traversal can be used to determine if an undirected graph is connected to identify the graphs connected components.
2. It can also be used to determine if a graph is cyclic. For a directed graph, this is detected when a back edge is found, for an undirected graph it is detected when a cross edge within the same tree is found.
3. For an un-weighted graph, first traversal canal so be used to find the shortest path from one node to another. The breadth – first tree path from the root to the target is the shortest path between the two nodes.

Efficiency of breadth first traversal

The efficiency of breadth first traversal is the same as that of depth first traversal, each node is visited once and all arcs emanating from every node are considered. Efficiency for the adjacency list graph representation = O(n + e).

6.14 MINIMUM SPANNING TREES

Given a connected weighted graph G, it is often desired to create a spanning tree T for G such that sum of the weights of the tree edges in T is as small as possible. Such a tree is called a minimum spanning tree.

Prim's Algorithm:

- An arbitrary node is chosen initially as the tree root.
  Note: In an undirected graph and its spanning tree, any node can be considered the tree root and the nodes adjacent to it as its sons.
  The nodes of the graph are then appended to the tree one at a time until all nodes of the graph are included.

Prim`s algorithm may therefore be implemented as follows:

```
Root= an arbitrary node chosen as root;
(every node nd in the graph){
Distance [nd] = weight (root, nd);
Closest[nd] = root ;
}/* end for */
distance[root] = INFINITY;
current = root;
for (i= 1; i<number of nodes in the graph;++i){
/* find the node closed to the tree */
mindist = INFINITY;
for (every node nd in the graph)
if (distance[nd] < mindinst){
        current = nd;
        mindist = distance[nd];
        }
        /* end if */
/* add the closest node to the tree */
/* and adjust distance  */
addson(closest[current],current);
distance[current] = INFINITY;
for (every node nd adjacent to current)
    if (distance[nd] < INFINITY)
    && (weight (current, nd) <
    distance [nd] ) {
        distance[nd] = weight
        (current, nd);
        closes[nd] = current ;
        }       /* end if */
    }       /* end for */
```

- If the graph is represented by an adjacency matrix, each for loop in Prim`s algorithm must examine O(n) nodes. Since the algorithm contains a nested for loop, it is $O(n^2)$

- Prim's algorithm can be made more efficient by maintaining the graph using adjacency lists and keeping a priority queue of the nodes not in the partial tree.

→ The first inner loop can then be replaced by removing the minimum distance node from the priority queue and adjusting the priority queue.

→ The second inner loop simply traverses an adjacency list and adjusts the position of any nodes whose distance is modified in the priority queue. Under this implementation Prim`s algorithm is O((n + e) log n)

Kruskal's Algorithm

- Like Prim's algorithm, Kruskal's algorithm also constructs the minimum spanning tree of a graph by adding edges to the spanning tree one by one. At all points during its execution the set of edges selected by Prim`s algorithm forms exactly one tree. On the other hand, the set of edges selected by Kruskal's algorithm terms a forest of trees.
- The set T of edges is initially empty. As the algorithm progresses, edges are added to T. So long as it has not found a solution, the partial graph formed by nodes of G and edges in T consists of several connected components.
- The elements of T included in a given connected component form a minimum spanning tree for the nodes in the component. At the end of the algorithm only one connected component remains, so T is then a minimum spanning tree for all the nodes of G.
- We observe the edges of G in order of increasing length. If an edge joints two nodes in different connected components, we add it to T. Also the two connected components now form only one component. Otherwise the edge is rejected, it joins two nodes in the same connected component and therefore cannot be added to T without forming a cycle (because the edges in T form a tree for each component) l. The algorithm stops when only one connected component remains.
- aTo illustrate this algorithm consider the fig. 15 diagram In increasing order of length the edges are:{1, 2}, {2, 3}, (4, 5}, {6, 7}, {1, 4}, (2, 5}, {4, 7},{3, 5}, {2, 4}, {3, 6},{5, 7} and {5, 6}, the algorithm proceeds as follows:

Step Initialization	Edge Considered	Connected Components {1},{2},{3},{4}{5}{6}{7}
1	({1,2}	{1,2}{3}{4}{5}{6}{7}
2	{2,3}	{1,2}{3}{4}{5}{6}{7}
3	{4,5}	{1,23}{4,5}{6}{7}
4	{6,7}	{1,2,3}(4,5}{6,7}
5	{1,4}	{1,2,3,4,5}{6,7}
6	{2,5}	Rejected
7	{4,7}	{1,2,3,4,5,6,7}

When the algorithm stops, T contains the chosen edges {1, 2}, {2, 3}, {4, 5}, {6, 7}, {1, 4} and {4m, 7}

Note: The minimum spanning tree is shown by dark lines in the above figure and its total length is17.

Running Time Analysis

We can evaluate the execution time of the algorithm as follows. On a graph with n nodes and e edges the number of operation s is in

1) O(eloge) to sort the edges, which is equivalent to O(e log n) because $(n-1) \le a \le n(n-1)/2$

2) O(n) to initialize the n distinct sets.

3) O(2eα(2e, n)) for all find and merge operations, (where α is the slow growing function, and there are almost 2e find operation s and n – 1 merge operation on a universe containing n elements.)

4) At worst O(e) for the remaining operation.

$\therefore$ Total time for the algorithm is in O(e log n) because O (α(2e, n) $\subset$ O(log n).

7 DESIGN TECHNIQUES

Introduction

In an algorithm design there is no perfect technique that is a cure for all computation problems. Different problems require the use of different kinds of techniques. A good programmer uses all these techniques based on the type of problem. Some commonly-used techniques are:

- Divide and conquer
- Randomized algorithms
- Greedy algorithms
- Dynamic programming

7.1 Greedy Algorithm:

A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment. This means that it makes a locally-optimal choice with the hope that it will lead to a globally-optimal solution.

Greedy Method:

The greedy method is one of the most straight forward algorithm design technique; and it can be applied to a wide variety of problems.

The greedy method suggests that one can devise an algorithm which works in stages, considering one input at a time. At each stage, a decision is made regarding whether or not a particular input is in an optimal solution.

This is done by considering the inputs in an order determined by some selection procedure. If the inclusion of next input into the partially constructed optimal solution then this input is not added to the partial solution. Most problems have 'n' input and require us to obtain subset that satisfied some constraints is called a feasible solution. We are required to find a feasible solution that optimizes (minimum or maximizes) a given objective function. A feasible solution that does this is called an optimal solution.

Greedy algorithms have some advantages and disadvantages:

1. It is quite easy to come up with a greedy algorithm (or even multiple greedy algorithms) for a problem.
2. Analyzing the run time for greedy algorithms is easier than for other techniques (like Divide and conquer). For the Divide and conquer technique, it is not clear whether the technique is fast or slow. This is because at each level of recursion the size of gets smaller and the number of sub-problems increases.
3. It is very much hard to understand correctness issues in Greedy technique. Even with the correct algorithm, it is hard to prove why it is correct. Proving that a greedy algorithm is correct is more of an art than a science. It involves a lot of creativity.

Greedy Algorithm Based Problems:

1. Activity Selection Problem
2. Job Sequencing Problem
3. Huffman Coding
4. Kruskal's Minimum Spanning Tree
5. Prim's Minimum Spanning Tree
6. Dijkastra's Shortest Path Algorithm

Shortest Path Problem:

Consider a directed graph G in which every edges has a non-negative weight attached, and our problem is to find a path from one vertex 'v' to another 'w' such that the weights on the path is as small as possible such a path is called 'shortest path'. The weights may represent costs, time or some quantity other than distance.

Consider example of airline routes, with each vertex representing a city and the weight on each edge the cost of flying from one city to the second. The problem is to find routine from city 'v' to city 'w' such that total cost is a minimum.

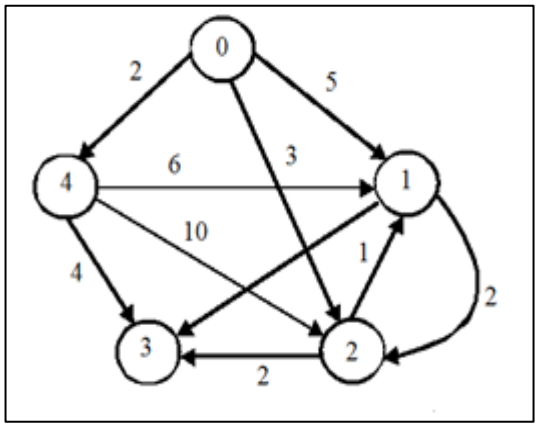
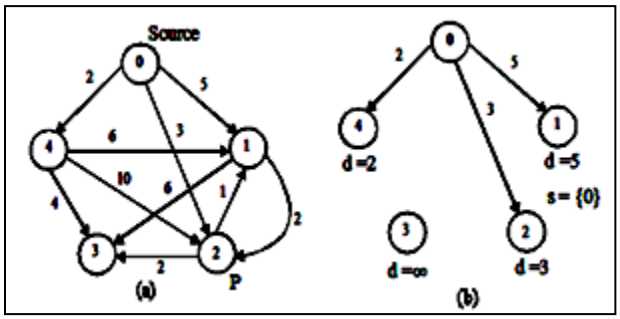


Fig. 1: Directed graph with weights

Consider the directed graph as shown in fig.1. The shortest path from vertex 0 to vertex 1 goes via vertex 2 and has a total cost of 4, compared to the cost of 5 for the edge directly from 0 to 1 and the cost of 8 for the path via vertex 4.

From starting node (vertex) called the source and finding the shortest path to every other vertex for simplicity, we take the source to be vertex 0, and our problem then consist of finding the smallest path from vertex 0 to every other vertex in the graph. The basic requirement is that the weights are all non–negative.



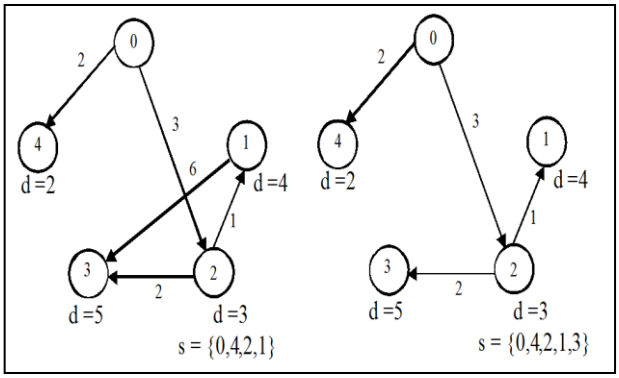


Fig. 2: shortest path for given graph

The directed graph shown in part (a), the initial situation is shown in part (b). The set S consist of 0 alone, and the entries of the distances table D.

The distances to vertex 4 is shortest, so 4 is added to S in part (c) and the distance D[4] is updated to the value 6. Since the distance to vertices 1 and 2 via vertex 4 and are greater than those already recorded in T, their entries remain unchanged.

The next closet vertex to 0 is vertex 2, and it is added in part (d), which also shows the effect of updating the distance to vertices 1 and 3. Its paths are via vertex 2 and are shortest than those preciously recorded. The finials two steps, shows in part (e) and (f), add vertices 1 and 3 to 5 and yield the paths and distance shown in the finial diagram.

7.2 Divide and Conquer Algorithm:

This paradigm, divide-and-conquer, breaks a problem into sub-problems that are similar to the original problem, recursively solves the sub-problems, and finally combines the solutions to the sub-problems to solve the original problem as shown in fig 3. Because divide-and-conquer solves sub-problems recursively, each sub-problem must be smaller than the original problem, and there must be a base case for sub-problems. You should think of a divide-and-conquer algorithm as having three parts:

1. **Divide:** Break the given problem into sub-problems of same type.
2. **Conquer:** Recursively solve these sub-problems
3. **Combine:** Appropriately combine the answers

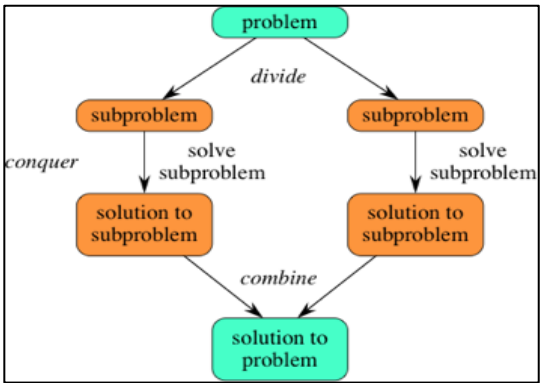


Fig. 3: Divide and conquer strategy

Divide and Conquer based Problems:

1. Binary Search
2. Quick Sort
3. Merge Sort
4. Matrix Multiplication (Strassen's-algorithm)
5. Maximal Subsequence
6. Fractional knapsack

Binary Search: A Divide and Conquer algorithm to find a key in an array.

Precondition: S is a sorted list

```
index binsearch (number n, index low, index high, const keytype S[ ], keytype x)
    if low ≤ high then
      mid = (low + high) / 2
    if x = S[mid] then
      return mid
    elsif x < s[mid] then
      return binsearch(n, low, mid-1, S, x)
    else
      return binsearch(n, mid+1, high, S, x)
    else
      return 0
  end binsearch
```

Divide: search lower or upper half
Conquer: search selected half
Combine: None

Performance:
$T(n)=T(n/2)+\Theta(1)$
$T(n)=\Theta(\log n)$

7.3 Dynamic programming:

Dynamic Programming is an algorithmic paradigm that solves a given complex problem by breaking it into sub-problems and stores the results of sub-problems to avoid computing the same results again.

If the given problem can be broken up in to smaller sub-problems and these smaller subproblems are in turn divided in to still-smaller ones, and in this process, if you observe some over-lapping subproblems, then it's a big hint for DP. Also, the optimal solutions to the sub-problems contribute to the optimal solution of the given problem (referred to as the Optimal Substructure Property).

There are two ways of doing this.

1. Top-Down: Start solving the given problem by breaking it down. If you see that the problem has been solved already, then just return the saved answer. If it has not been solved, solve it and save the answer. This is usually easy to think of and very intuitive. This is referred to as Memoization.

2. Bottom-Up: Analyze the problem and see the order in which the sub-problems are solved and start solving from the trivial subproblem, up towards the given problem. In this process, it is guaranteed that the subproblems are solved before solving the problem. This is referred to as Dynamic Programming.

Dynamic Programming problems:

1. Longest Common Subsequence
2. Subset Sum Problem
3. 0-1 Knapsack Problem

4. All pair shortest path by Floyd-Warshall
5. Shortest path by Dijkstra

7.4 Backtracking:

Sometimes we are facing the task of finding an optical solution to a problem, there is no applicable theory to help us to find the optimum expect by resorting to exhaustive search. So new systematic, exhaustive technique is used known as 'backtracking'.
In this technique a partial solution is derived at each step and validity of partial solution is checked and if incorrect we backtrack and repair the solution.

Examples:
Chess, 8 queen problem, puzzle, tic –tac-toe problem.
Consider the puzzle of how to place eight queens on a chessboard, so that no queen can attack another. One of the rule for chess is that, a queen can take another piece that lie on the same row, same column, or the same diagonal as that of queen.

ARRAYS

1. Is it possible to have negative index?

2. What is zero based addressing?

3. What is array initialization?

4. What is a string?

5. Differentiate between gets ( ) and scanf ( ) using %s conversion specification?

6. How to initialize a character array?

7. What is a subscripted variable?

8. What are the characteristics of arrays in C?

9. What is the allowed data type for subscript?

10. Why is it necessary to give the size of an array in an array declaration?

11. How to get the size of an array in a program?
```
int a[10];
print f("%d\n", size of (a)/size of (a[0]));
```

12. When does the compiler not implicitly generate the address of the first element of an array?

13. Does the following code work? Justify.
```
const int n = 10; int x[n];
```

14. What is the output of the program?
```
Main ( )
{
int a[ ] = {0, 0×4, 4, 9};
int i = 2;
print f("%d %d", a[i], i[a]);
}
```

15.
```
#define void int
int i = 300;
void main ( )
{
int i = 200;
{
int  i = 100;
print f("%d", i);
}
print f("%d", i);
}
```
What is the output of the above program?

16. How many bytes of memory will the following arrays need?
```
(a) char s[80];/* 80*/
(b) char s[80][10];/* 800*/
(c) int d[10];/*10* size of(int)*/
(d) float d[10][5];/*50*size of(float)*/
```

17. What is the output of the program?
```
main ( )
{
int i, j;
int mat [3] [3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
for (i=2; i>= 0; i --)
for (j=2; j>=0; j--)
print f("%d", *(*(mat + j) + i));
}
```

18. What is the output of the program?

```
#include <stdio.h>
main ( )
{
char s1 [ ] = "Ramco";
char s2 [ ] = "Systems";
s1 = s2;
print f("%s", s1);
}
```

19. What is the output of the program?

```
#include <stdio.h>
main ( )
{
int a[10];
printf("%d", ((a + 9) + (a + 1)));
}
```

20. What is the output of the program?

```
#include <stdio.h>
main ( )
{
char numbers [5] [6] = {"zero", "one", "two", "three", "four"};
printf("%s is %c", & numbers [4] [0], numbers [0] [0]); }
```

21. Write a program to reverse a string using theoperator

```
#include <stdio.h>
int main ( )
{
char s[20];
void revstr ( char *);
print f("Enter the string to be reversed\n");
scanf ("%s", s);
revstr (s);
printf ("Reverse string is %s \n", s);
return 0;
}
void revstr (char *s)
{
int length = strlen (s);
int i, j;
if (length > 0)
for (i =0, j = length -1; i < j; i ++, j --)
s[i] ^ = s[j], s[j] ^ = s [i], s[i] ^ =s[j];
}
```

GATE QUESTIONS

DATA STRUCTURES

ALGORITHMS

GATE QUESTIONS(PROGRAMMING)

Q.1 The value of j at the end of the execution of the following C program

```
int incr (int i){
        static int count = 0;
        count = count + i;
        return (count);
}
        main () {
        int i, j;
        for (i = 0; i <=4; i++)
        j= incr (i);
}
```

is

a) 10 b) 4
c) 6 d) 7

[GATE -2000]

Q.2 The most approximate matching for the following pairs

List-I

X: m = malloc (5); m = NULL;
Y: free (n); n->value =5;
Z: char *p; *p= 'a';

List-II

1: using dangling pointers
2: using uninitialized pointers
3: lost memory

Codes:

a) X—1, Y— 3, Z— 2
b) X— 2, Y — 1, Z—3
c) X— 3, Y— 2, Z—1
d) X — 3, Y— 1, Z —2

[GATE- 2000]

Q.3 The following C declarations

```
struct node {
        int i:
        float j;
};
        struct node * s[10];
```

Define s to be -

a) An array, each element of which is a pointer to a structure of type node
b) A structure of 2 fields, each field being a pointer to an array of 10 elements
c) A structure of 3 fields: an integer, a float, and an array of 10 elements
d)An array, each element of which is a structure of type node

[GATE -2000]

Q.4 What is printed by the print statements in the program P1 assuming call by reference parameter passing?

```
Program P1 ( ) {
        x = 10;
        y = 3;
        func1 (y, x, x);
        print  x;
        print  y;
}
func1 (x, y, z) {
        y =y+4;
        z = x + y +z;
}
```

a) 10, 3 b) 31, 3
c) 27, 7 d) None of these

[GATE -2001]

Q.5 In the C language

a) At most one activation record exists between the current activation record and the activation record for the main
b) The number of activation records between the current activation record and the activation record for the main depends on the actual function calling sequence
c) The visibility of global variables depends on the actual function calling sequence
d) Recursion requires the activation record for the recursive function to be saved on

a different stack before the recursive function can be cal led

[GATE -2002]

Q.6 The results returned by function under value-result and reference parameter passing conventions

a) Do not differ
b) Differ in the presence of loops
c) Differ in all cases
d) May differ in the presence of exception

[GATE- 2002]

Q.7 Assume the following C variable declaration

int * A[10], B[10][10];

Of the following expressions which will not give compile time errors if used as left hand sides of assignment statements in a C program?

I. A [2]
II. A[2][3]
III. B[1]
IV. B [2][3]

a) I, II, and IV only
b) II, III, and IV only
c) II and IV only
d) IV only

[GATE- 2003]

Q.8 Consider the C program shown below:

```
# include <stdio.h>
# define print(x) printf ("%d", x)
int x;
void Q(int z); {
        z +=x;
        print (z);
}
void P(int *y) {
        int x= *y+2;
        Q(x);
        *y = x-1;
        print (x);
}
main (void) {
        x = 5;
        P (&x)
        print (x);
}
```

The output of this program is

a) 12 7 6 b) 22 12 11
c) 14 6 6 d) 7 6 6

[GATE 2003]

Q.9 In the following C program fragment j, k, n and Two Log _ n are integer variables, and A is an array of integers. The variable n is initialized to an integer ≥3 and Two Log _ n is initialized to the value of $2^{*}\lfloor \log_2(n) \rfloor$

```
for (k=3; k <=n; k++)
        A[k] = 0;
for (k=2; k <= Two Log _ n; k++)
        for (j=k+1; j <=n; j++)
A[j] = A[j] || (j % k);
for (j=3; j<=n; j++)
        if (!A[j]) printf ("%d", j);
```

The set of numbers printed by this program fragment is

a) {m | m≤n, (∃i) [m = !1]}
b) {m | m≤n, (∃i) [m = i^2]}
c) {m | m≤n, m is prime}
d) None

[GATE -2003]

Q.10 Consider the following C function;

```
void swap {int a, int b){
        int temp;
        temp a;
        a = b;
        b = temp;
}
```

In order to exchange the values of two variables x and y

a) Call swap (x, y)
b) Call swap (&x, &y)
c) Swap (x, y) cannot be used as it does not return any value
d) Swap (x, y) cannot be used as the parameters are passed by value

[GATE -2004]

Q.11 The goal of structured programming is to

a) Have well indented programs
b) Be able to infer the flow of control from the compiled code
c) Be able to infer the flow of control from the program text
d) Avoid the use of GOTO statements

[GATE -2004]

Q.12 Consider the following C-program

```
double foo (double);/* Line 1 */
int main 0 {
        double da, db;
        // input da
        db = foo (da);
}
double foo (double a) {
        return a;
}
```

The above code compiled without any error or warning. If Line 1 is deleted, the above code will show

a) No compile warning or error
b) Some compiler-warnings not leading to unintended results
c) Some compiler-warnings due to type-mismatch eventually leading to unintended results
d) Compiler errors

[GATE- 2005]

Q.13 Consider the following C- program:

```
void foo (int n, int sum) {
        int  k = 0, j = 0;
        if  (n==0) return;
        k = n % 10; j =n/10;
        sum = sum + k;
        foo (j, sum);
        printf ("%d", k);
}
int main () {
        int a= 2048, sum = 0;
        foo (a, sum);
        printf ("%d\ n", sum);
}
```

What does the above program print?

a) 8, 4, 0, 2, 14 b) 8, 4, 0, 2, 0
c) 2, 0, 4, 8, 14 d) 2, 0, 4, 8, 0

[GATE 2005]

Q.14 Which one of the following are essential features of an object-oriented programming language?

1. Abstraction and encapsulation
2. Strictly-typedness
3. Type-safe property coupled with sub-type rule
4. Polymorphism in the presence of inheritance

a) 1 and 2 b) 1 and 4
c) 1, 2 and 4 d) 1, 3 and .4

[GATE 2005]

Q.15 A common property of logic programming languages and functional languages is

a) Both are procedural languages
b) Both are based on λ- calculus
c) Both are declarative
d) Both use Horn-clauses

[GATE 2005]

Q.16 An Abstract Data Type (ADT) is

a) Same as an abstract class
b) A data type that cannot be instantiated
c) A data type for which only the operations defined on it can be used, but none else
d) All of the above

[GATE 2005]

Q.17 What does the following C-statement declares?

```
int (*f) (int *);
```

a) A function that takes an integer pointer as argument and returns an integer
b) A function that takes an integer as argument and returns an integer pointer
c) A pointer to a function that takes an integer pointer as argument and returns an integer
d) A function that takes an integer pointer as argument and returns a function pointer

[GATE 2005]

Q.18 Consider this C code to swap two integers and these five statements: The code

```
void swap (int *px, int *py) {
        *px = *px - *py;
        *py = *px + *py;
        *px = *py - *px;
}
```

S1: will generate a compilation error
S2: may generate a segmentation fa.0 It at runtime depending on the arguments passed
S3: correctly implements the swap procedure for all input pointers referring to integers stored in memory locations accessible to the process
S4: implements the swap procedure correctly for some but not all valid input pointers
S5: may add or subtract integers and pointers

a) S1 only b) S2 and S3
c) S2 and S4 d) S2 and S5

[GATE 2006]

Q.19 Consider these two functions and two statements S1 and S2

```
int work1 (int *a, int I,
int j)
{
      int x = a[i + 2];
      A[j] = x + 1;
      return a [i + 2] –
3;
}
```

```
int work2 (int *a, int I,
int j)
{     int t1 = i + 2;
      int t2 = a [t1];
      A[j] = t2 + 1;
      return t2-3
}
```

S1: The transformation from work1 to work2 is valid, i.e., for any program state and input arguments, work2 will compute the same output and have the same effect on program state as work1
S2: lf the transformations applied to work to get work2 will always improve the performance (i.e.,. reduce CPU time) of work2 compared to work1

a) S1 is false and S2 is false
b) S1 is false and S2 is true
c) S1 is true and S2 is false
d) S1 is true and S2 is true

[GATE -2006]

Q.20 Consider the following C-function in which a[n] and b[m] are two sorted integer arrays and c[n+m] be another integer array.

```
void xyz (int a[ ], int b[ ] int c[ ]) {
        int i, j, k;
        i=j=k=0;
        while ((i<n) && (j<m))
        if (a[i] < b[j])
                c[k++]= a[i++];
        else
                c[k ++] = b[j++];
}
```

Which of the following condition hold(s) after the termination of the while loop?

(i) $j < m$, $k = n + j - 1$, and $a[n-1] < b[j]$, if $i = n$
(ii) $i < n$, $k = m + i - 1$, and $b[m-1] \leq a[i]$, if $j = m$

a) only (i)
b) only (ii)
c) either (1) or (ii) but not Both
d) neither (i) nor (ii)

[GATE -2006]

Q.21 Consider the following C function:

```
int f (int n){
        static int r = 0;
        if (n <= 0) return 1;
        if (n > 3){
                r = n;
                return f (n-2) + 2;
        }
        return  f (n-1) + r;
}
```

What is the value of f (5)?

a) 5 b) 7
c) 9 d) 18

[GATE -2007]

Q.22 Which combination of the integer variables x, y and z makes the variable a get the value 4 in the following expression?

a=(x>y)? ((x>z)? x: z): ((y> z)? y: z)
a) x = 3, y= 4, z = 2
b) x= 6, y =5 z =3
c) x =6, y=3,z = 5
d) x= 5, y = 4, z = 5

[GATE- 2008]

Q.23 Choose the correct to fill ?1and ?2 so that the program below prints an input string in reverse order. Assume that the input string is terminated by a newline character.

```
void reverse (void){
        int c;
        if (?1) reverse ( );
        ?2;
}
main ( ) {
        printf ("Enter Text");
        printf ("/n");
        reverse ( ); printf ("/n");
}
```

a) ?1 is (getchar ( )!= '\n')
b) ?1 is (c = getchar ( ))!= '\n')
   ?2 isgetchar (c);?2 is getchar (c);
c) ?1 is (c !='\n')
   ?2 isputchar (c);
d) ?1 is (c = getchar ( ))!= '\n')
   ?2 is putchar (c);

[GATE -2008]

Q.24 What is printed by the following C program?

```
int f (int x, int *py, int **ppz {
        int y, z;
        **ppz +=1;z = *ppz;
        *py+=2; y =*py;
        x +=3;
        return x + y+ z;
}
void main ( )
{
        int c, *b, **a,
        c=4; b=&c; a=&b;
        printf ("%d", f(c, b, a));
}
```

a) 18 b) 19
c) 21 d) 22

[GATE -2008]

Q.25 Consider the program below:

```
# include <studio.h>
int fun (int n, int*f_ p) {
        int t, f;
        if (n <=1) {
        *f_p=1;
        return 1;
}
        t= fun (n—1,*f _ p);
        f= t + *f _ p;
        *f_ p + t;
        return f;
}
int main {}
{
        int x=15;
        printf ("%d\n", fun (5, &x)};
        return 0;
}
```

The value printed is
a) 6 b) 8
c) 14 d) 15

[GATE- 2009]

Q.26 What does the following program print?

```
# include <stdio. h>
void f (int*p, int*q) {
        p=q;
        *p=2;
}
int i = 0, j = 1;
int main () {
        f (&i, & j};
        printf ("%d %d/n", i, j);
        return0 ;
}
```

a) 2 2 b) 2 1
c) 0 1 d) 0 2

[GATE-2010]

Q.27 The following program is to be tested for statement coverage begin

```
if (a == b) {S1; exit; }
else if (c = = d) {S2;}
else    {S3;exit;}
S4;
end
```

The test cases T_1, T_2, T_3 and T_4 given below are expressed in terms of the properties satisfied by the values of variables a, b, c and d. The exact values are not given.
T_1: a, b, c and d are all equal
T_2: a, b, c and d are all distinct
T_3: a=b and c! =d
T_4: a! = b and c=d
Which of the test suites given below ensures coverage of statements S_1, S_2, S_3 and S_4?

a) T_1, T_2, T_3 b) T_2,T_4
c) T_3, T_4 d) T_1, T_2, T_4

[GATE- 2010]

Q.28 What is the value printed by the following C program?

```
#include < stdio.h>
int f(int *a, int n)
{
if (n<=0) return 0;
else if (*a %2= =0)
        return *a +f(a + 1, n-1);
else
        return *a - f(a+1, n-1);
}
int main ( )
{
        int a[ ] = {12, 7, 13, 4, 11 , 6};
        printf("%d*, f(a, 6));
        return 0;
}
```

a) -9 b) 5
c) 15 d) 19

[GATE -2010]

Common Data for Questions 29 and 30
Consider the following recursive C function that takes unsigned int foo (unsigned int, n, unsigned int r)

```
{
        if (n > 0) return (n%r + foo (n/r, r));
        else return 0;
}
```

Q.29 What is the return value of the function foo, when it is called as foo (513, 2)?

a) 9 b) 8
c) 5 d) 2

[GATE -2011]

Q.30 What is the return value of the function foo, when it is called as foo (345, 10)?

a) 345 b) 12
c) 5 d)3

[GATE -2011]

Q.31 What does the following fragment of C-program print?

```
char c [ ] = "GATE 2011"
char *p = c;
printf ("%s", p+p[3]-p[1]);
```

a) GATE 2011 b) E2011
c) 2011 d) 011

[GATE-2011]

Common Data for Questions 32 and 33
Consider the following C code segment

```
int a, b, c = 0;
void prtFun (void);
int main ()
{
    static int a = 1; /* line 1 */
    prtFun();
    a += 1;
    prtFun();
    printf ( "\n %d %d " , a, b) ;
}
```

```
void prtFun (void)
{
   static int a = 2; /* line 2 */
   int b = 1;
   a += ++b;
   printf (" \n %d %d " , a, b);
}
```

Q.32 What output will be generated by the given code segment?

a)3 1 b)4 2
4 1 6 1
4 2 6 1

c)4 2
6 2
2 0

d)3 1
5 2
5 2

[GATE-2012]

Q.33 What output will be generated by the given code segment if -
Line 1 is replaced by auto int a = 1;
Line 2 is replaced by register int a= 2?

a)3 1
4 1
4 2

b)4 2
6 1
6 1

c)4 2
6 2
2 0

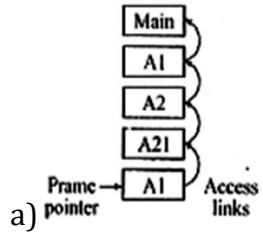
d)4 2
4 2
2 0

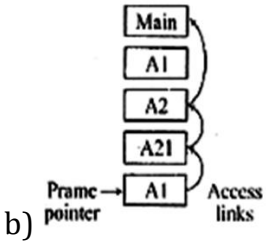
[GATE-2012]

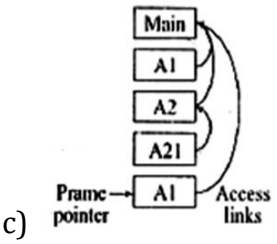
Q.34 Consider the program given below, in a block-structured pseudo-language with lexical scoping an nesting of procedures permitted.

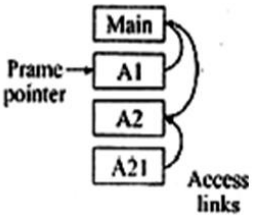
```
Program main;
Var ……..
Procedure A1;
        Var …
        Call A2 ;
        End A1
Procedure A2;
        Var ……..
Procedure A21;
        Var …
Call A1;
        End A2
        Call A21;
        End A2
        Call A1;
End main.
```

Consider the calling chain:
Main → A1 → A2 → A21 → A1
The correct set of activation records along with their access links is given by

a)


b)


c)


d)


[GATE-2012]

Q.35 What is the return value of (p , p) , if the value of p is initialized to 5 before the call? Note that the first parameter is passed by reference, whereas the second parameter is passed by value.

```
int   f(int &x, int  c){
        c = c – 1;
        int  (c = = 0)  return  1 ;
        x = x + 1;
        return   f(x, c)  *  x :
}
```

a) 3024
b) 6561
c) 55440
d) 161051

[GATE-2013]

Q.36 Consider the C function given below

```
int f(int j)
{
static int i = 50;
```

```
int k;
if (i == j)
{
printf("something");
k = f(i);
return 0;
}
else return 0;
}
```

Which one of the following is TRUE?

a) The function returns 0 for all values of j.
b) The function prints the string something for all values of j.
c) The function returns 0 when j=50.
d) The function will exhaust the runtime stack or run into an infinite loop when j = 50

[GATE -2014]

Q.37 Consider the following pseudo code. What is the total number of multiplications to be performed?

```
D = 2
for i = 1 to n do
  for j = i to n do
     for k = j + 1 to n do
        D = D *3;
```

a) Half of the product of 3 consecutive integers
b) One third of the product of 3 consecutive integers
c) One-sixth of the product of 3 consecutive integers
d) None of the above

[GATE -2014]

Q.38 Consider the function func shown below:

```
int func(int num) {
int count = 0;
while (num) {
count++;
num>>= 1;
}
return (count);
}
```

The value returned by func(435) is ……

[GATE -2014]

Q.39 Suppose n and p are unsigned int variables in a C program. We wish to set p to nC3.If n is large, which one of the following statements is most likely to set p correctly?

a) p = n * (n-1) * (n-2) / 6;
b) p = n * (n-1) / 2 * (n-2) / 3;
c) p = n * (n-1) / 3 * (n-2) / 2;
d) p = n * (n-1) * (n-2) / 6.0;

[GATE -2014]

Q.40 Consider the following function

```
double f (double x)
{
if ( abs( x*x – 3) < 0.01) return x;
else return f (x/2+1.5/x);
}
```

Give a value q (to 2 decimal) such that f(q) will return q: ……..

[GATE- 2014]

Q.41 Consider the following program in C language

```
#include<stdio.h>
        void main()
        {
            int i;
            int *pi = &i;
            scanf ("%d",pi);
            printf ("%d\n",i+5);
        }
```

Which of the following is true?

a) Compilation fails
b) Execution results in a run time error
c) On execution, the value printed is 5 more than the address of variable i
d) On execution, the value printed is 5 more than the integer value entered.

[GATE -2014]

Q.42 Consider the following C function.

```
int fun (int n)
```

```
{
        int x = 1, k;
         if (n==1)
        return x;
        for (k = 1; k < n; ++k )
        x= x+fun( k ) * fun (n-k) ;
        return x;
}
```

The return value of fun (5) is _____

[GATE- 2015]

Q.43 Consider the following recursive C function.

```
void get (int n) {
        if (n<1) return;
        get (n-1);
        get(n- 3) ;
        print f (" %d",n) ;
}
```

If get (6) function is being called in main () then how many times will the get () function be invoked before returning to the main ( )?

a) 15 b) 25
c) 35 d) 45

[GATE -2015]

Q.44 Consider the following C program

```
#inclue <stdio.h>
int main()
{
int i, j, k=0;
j = 2*3 / 4 + 2.0 / 5 + 8 / 5;
k - = --j;
for (i=0; i<5; i++)
{
        switch(i+k)
        {
        case1:
        case 2 : printf("\ n %d", i+k) ;
        case 3: printf("\ n %d", i+k);
        default : printf("\ n%d", i+k) ;
        }
}
return 0;
}
```

The number of times printf statement is executed is _____

[GATE -2015]

Q.45 Consider the following C program.

```
#include <stdio.h>
int f1(void) ;
int f2(void) ;
int f3(void);
int x = 10;
int main()
{
        int x = 1;
        x  += f1() + f2() + f3() + f2();
        printf("%d", x) ;
        return 0;
}
int f1()
{
        int x = 25;
        x++ ;
        return x;
}
int f2()
{
        static int x = 50;
        x++ ;
        return x;
}
int f3()
{
        x*=10;
        return x ;
}
```

The output of the program is_____

[GATE- 2015]

Q.46 Consider the following function written the C programming language.

```
void foo (char *a)
 {
        if (*a && *a != ' ')
        {
                foo(a+1);
                putchar (*a) ;
        }
}
```

The output of the above function on input "ABCD EFGH" is

a) ABCD EFGH b) ABCD
c) HGFE DCBA d) DCBA

[GATE -2015]

Q.47 Consider the following C program segment.

```
#include<stdio.h>
int main()
{
        char s1[7]="1234", *p;
        p = s1+2;
        *p = '0';
        printf("%s"s1);
}
```

What will be printed by the program?

a) 12 b) 120400
c) 1204 d) 1034

[GATE- 2015]

Q.48 Consider the following C program

```
#include <stdio.h>
 int main ()
{
static int a[ ] = { 10, 20, 30 40, 50} ;
static int *p[ ] ={ a, a+3, a+4, a+1, a+2} ;
int **ptr = p;
ptr++ ;
printf ("%d%d", ptr - p, **ptr) ;
}
```

The output of the program is _____

[GATE -2015]

Q.49 Consider the following C program.

```
void f(int, short);
void main( )
{
int i = 100;
short s = 12;
short *p = &s;
_____ ; // call to f( )
}
```

Which one of the following expressions, when placed in the blank above, will NOT result in a type checking error?

a) f(s,*s) b) i = f(i,s)
c) f(i,*s) d) f(i,*p)

[GATE- 2016]

Q.50 Consider the following C program.

```
#include<stdio.h>
void mystery(int *ptra, int *ptrb)
{
int *temp;
temp = ptrb;
ptrb = ptra;
ptra = temp;
}
int main( )
{
int a = 2016, b = 0, c = 4, d = 42;
mystery (&a, &b);
if (a < c)
mystery(&c, &a);
mystery(&a, &d);
printf("%d\n", a);
}
```

The output of the program is _____.

[GATE -2016]

Q.51 The following function computes the maximum value contained in an integer array p[ ]of size n (n >= 1).

```
int max(int *p, int n)
{
int a = 0, b = n – 1;
while ( _____ )
{
if (p[a] <= p[b])
        { a = a+1; }
else
        { b = b-1; }
}
return p[a];
}
```

The missing loop condition is

a) a != n b) b != 0
c) b > (a + 1) d) b != a

[GATE -2016]

Q.52 What will be the output of the following pseudo-code when parameters are passed by reference and dynamic scoping is assumed?

```
a = 3;
void n(x)
{
x = x * a;
```

```
print(x);
}
void m(y)
{
a = 1;
a = y - a;
n(a);
print(a);
}
void main()
{
m(a);
}
```

a) 6, 2 b) 6, 6
c) 4, 2 d) 4, 4

[GATE -2016]

Q.53 The value printed by the following program is_____.

```
void f(int* p, int m)
{
m = m + 5;
*p = *p + m;
return;
}
void main( )
{
int i=5, j=10;
f(&i, j);
printf("%d", i+j);
}
```

[GATE- 2016]

Q.54 The following function computes X^Y for positive integers X and Y.

```
int exp(int X, int Y)
{
int res = 1, a = X, b = Y;
while (b != 0 )
{
        if ( b%2 == 0)
        {
                a = a*a;
                b = b/2;
        }
        else
        {
                res = res*a;
                b = b-1;
        }
}
return res;
}
```

Which one of the following conditions is TRUE before every iteration of the loop?

a) $X^Y = a^b$
b) $(res*a)^Y = (res*X)^b$
c) $X^Y = res*a^b$
d) $X^Y = (res*a)^b$

[GATE- 2016]

Q.55 Consider the following program:

```
int f(int *p, int n)
{
if (n <= 1) return 0;
else
return max(f(p+1, n–1), p[0] – p[1]);
}
int main()
{
int a[ ] = {3, 5, 2, 6, 4};
printf ("%d", f(a, 5));
}
```

Note: max(x, y) returns the maximum of x and y.

The value printed by this program is_____.

[GATE -2016]

Q.56 Consider the C struct defined below:

```
Struct data {
        int marks [100];
        char grade;
        int cnumber;
};
Struct data student;
```

The base address of student is available in register R1. The field student, grade can be accessed efficiently using

a) Post- increment addressing mode (R1)
b) Pre– decrement addressing mode.-(R1)
c) Register direct addressing mode E1.

d) Index addressing mode X (RI). Where X is an offset represented in 2's complement 16-bit representation.

[GATE -2017]

Q.57 Consider the following C code:

```
#include < stdio. h >
int* assignval (int *x, int val)
{
  *x = val;
  return x;
}
void main ( ) {
int *x= malloc (size of (int));
If (NULL= = x) return;
x= assignval (x, 0);
if (x) {
x= (int*) malloc (sizeof (int));
        if (NULL= = x) return;
        x= assignval (x, 10);
}
printf ("%d\n", *x);
free (x);
}
```

The code suffers from which one of the following problems:

a) Complier error as the return of malloc is not typecast appropriately
b) Compiler error because the comparison should be made as x= = NULL and not as shown
c) complies successfully but execution may result in dangling pointer
d) compiles successfully but execution may result in memory leak

[GATE -2017]

Q.58 The output of executing the following C program is ____

```
#include <stdio h>
int total (int v) {
static int  count = 0;
while (v) {
   count + = v & 1;
   v >>=1;
   }
   return count;
}
void main () {
static int x= 0;
 int i = 5;
  for (; i > 0; i - -) {
  x=x+ total (i);
}
 printf ("%d\n", x);
}
```

[GATE -2017]

Q.59 Consider the C functions foo and bar given below

```
int foo(int val )
{
        int x= 0;
        while (val > 0)
        {
         x = x+ foo (val- -);
        }
         return val;
}
int bar (int val) {
        int x= 0;
        while (val > 0)
        {
        x = x+ bar (val-1);
        }
        return val;
}
```

Invocations of foo (3) and bar (3) will result in:

a) Return of 6 and 6 respectively.
b) Infinite loop and abnormal termination respectively
c) Abnormal termination and infinite loop respectively.
d) Both terminating abnormally.

[GATE -2017]

Q.60 Consider the following C program.

```
# include <stdio.h>
# include <string.h>
void printlength (char *s, char *t) {
unsigned int c= 0;
int  len=((strlen(s)  -  strlen(t))  >  c)
strlen(s) : strelen(t);
```

```
printf ("%d\n" , len);
}
void main ( ) {
        char *x= "abc";
        char *y= "defgh";
        printlength (x, y);
```

recall that strlen is defined in sting, h as returning a value of type size_ t, which is an unsigned int. The output of the program is ___.

[GATE -2017]

Q.61 Consider the following two functions:

```
Void fun1 (int n{     Void fun2 (int n{
If (n= = 0) return;   If(n==0) return;
Printf("%d",n)        Printf("%d",n)
Fun2(n-2);            fun 1(++n);
Printf("%d",n);       printf("%d",n);
}                     }
```

The output printed when fun 1 (50 is called is

a) 53423122233445
b) 53423120112233
c) 53423122132435
d) 53423120213243

[GATE -2017]

Q.62 Match the following :

List-I

P) static char var;
Q) m= malloc(10);m= NULL;
R) char*ptr [10];
S) register intvar1;

List-II

i) Sequence of memory locations to store addresses
ii) A variable located in data section of memory
iii) Request to allocate a CPU register to store data
iv) A lost memory which cannot be freed

a) P→(ii),Q→(iv),R→(i),S→(iii)
b) P→(ii),Q→(i),R→(iv),S→(iii)
c) P→(ii),Q→(iv),R→(iii),S→(i)
d) P→(iii),Q→(iv),R→(i),S→(ii)

[GATE -2017]

Q.63 Consider the following function implemented in C:

```
Void printxy(ints, inty)
{
     int*ptr;
    x=0
    ptr =&x;
    y=*ptr;
   *ptr=1;
   printf("%d,%d",x,y);
 }
```

The output of invoking printxy(1,1)is

a) 0,0 b) 0,1
c) 1,0 d)1,1

[GATE -2017]

Q.64 Consider the C program fragment below which is meant to divide x by y using repeated subtractions . The variables x,y,q are all unsigned int.

```
while (r > = y)
{
 r = r-y ;
q = q+1;
}
```

Which of the following conditions on the variables x,y,q and r before the execution of the fragement will ensure that the loop terminates in a state satisfying the condition x= =(y× q + r)?

a) (q= =r) & &(r = =0)
b) (x > 0) & &(r = =x) & & (y > 0)
c) (q= = 0) & &(r= =x)& & (y>0)
d) (q= =0) & & (y> 0)

[GATE -2017]

Q.65 Consider the following C program

```
#include < stidio. H >
Int main( )
{ int m = 10;
int n,n1;
 n1= + + m;
 n- -;
n- =n1;
printf("%d",n)'
 return 0;
}
```

The output of the program is ___.

[GATE -2017]

Q.66 Consider the following C program:

```
#include <stdio.h>
int counter = 0;

int calc(int a, int b) {
 int c;
 counter++;
 if (b == 3)
return (a * a * a);
else {
 c = calc(a, b / 3);
 return (c * c * c);
 }
}
int main() {
 calc(4, 81);
 printf("%d", counter);
}
```

The output of this program is _____ .

a) 5 b) 4

c) 3 d) None of these

[GATE -2018]

Q.67 Consider the following program written in pseudo-code. Assume that x and y are integers.

```
Count (x, y) {
   if (y !=1 ) {
      if (x !=1) {
         print("*");
         Count (x/2, y);
      }
      else {
         y=y-1;
         Count (1024, y);
      }
   }
}
```

The number of times that the print statement is executed by the call Count (1024, 1024) is _____ .

(A) 10230

(B) 10

(C) 1023

(D) 23010

[GATE -2018]

Q.68 Consider the following C program:

```
#include <stdio.h>
void fun1(char *s1, char *s2) {
 char *temp;
 temp = s1;
 s1 = s2;
 s2 = temp;
}
void fun2(char **s1, char **s2) {
 char *temp;
 temp = *s1;
 *s1 = *s2;
 *s2 = temp;
}
int main() {
 char *str1 = "Hi", *str2 = "Bye";
 fun1(str1, str2);
 printf("%s %s", str1, str2);
 fun2(&str1, &str2);
 printf("%s %s", str1, str2);
 return 0;
}
```

The output of the program above is

a) Hi Bye Bye Hi

b) Hi Bye Hi Bye

c) Bye Hi Hi Bye

d) Bye Hi Bye Hi

[GATE -2018]

Q.69 Consider the following C code. Assume that unsigned long int type length is 64 bits.

```
unsigned long int fun(unsigned long
int n) {
  unsigned long int i, j, sum = 0;
     for( i = n; i > 1; i = i/2) j++;
     for( ; j > 1; j = j/2) sum++;
     return sum;
}
```

The value returned when we call fun with the input 240 is

a) 4

b) 5

c) 6

d) 40

[GATE -2018]

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
(a)	(d)	(a)	(b)	(a)	(b)	(c)	(a)	(b)	(d)	(c)	(c)	(d)	(b)
15	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**
(c)	(c)	(c)	(b)	(d)	(c)	(d)	(a)	(d)	(b)	(b)	(d)	(d)	(c)
29	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**
(d)	(b)	(c)	(c)	(d)	(d)	(b)	(d)	(c)	9	(b)	1.73	(d)	51
43	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**
(b)	10	230	(d)	1204	140	(d)	2016	(d)	(d)	30	(c)	3	(d)
57	**58**	**59**	**60**	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	
(a)	23	(c)	3	(c)	(a)	(c)	(c)	0	(b)	(c)	(a)	(b)	

EXPLANATIONS

Q. 1 (a)

Count is static variable in incr(). Statement static int count = 0 will assign count to 0 only in first call. Other calls to this function will take the old values of count.

Count becomes 0 after the function call incr(0)

Count becomes 1 after the function call incr(1)

Count becomes 3 after the function call incr(2)

Count becomes 6 after the function call incr(3)

Count becomes 10 after the function call incr(4)

Q. 2 (d)

X corresponds to a lost memory thus, X-3

Y corresponds to usage of dangling pointers thus, Y-1

Z corresponds to usage of uninitialized pointers thus, Z-2

Q. 3 (a)

In the given declaration s is an array denoted by s[10], containing 10 elements which are pointers indicated by the symbol, *, to structure of type node as defined by structure node statement.

Q. 4 (b)

Let the formal parameters be X, Y, Z to avoid confusion.

```
func1 (X, Y, Z)
{
        Y = Y + 4;
        Z = X + Y + Z;
}
```

y and X refer same location. x, Y, Z refer to same location. So the changes in func1 reflects in x, y, so x = 10 and y = 3 is printed.

Q. 5 (a)

Activation record or frame, having a collection of fields, gives the information required by a single execution of a procedure that is managed by a contiguous block of storage. Such type of storage are used by languages like C and Pascal

so when a procedure is called, it is necessary to push the activation record of the procedure on a run time stack and when the control is returned back to the calling procedure the activation record is popped up from the stack.

Q. 6 (b)

The results returned by the functions under value-result and reference parameter passing conventions differ in the presence of loops.

Q. 7 (c)

From the given declaration it is clear that int* A [10], is an array of 10 pointers.

A[2][3] can be used on the left hand side of assignment when A[2] can hold the pointer of an integer array.

Thus, B [2] [3] can be used as left hand side as it gives the element of second row in the third column.

Q. 8 (a)

From the code it is found that

x = 5

x = * y + 2 = 5 + 2 = 7

Q(x)

z = 7

z = z + x = 12

print (z) = 12

*y = x - 1 = 6

Print(x) = 7

Print(x) = 6

Q. 9 (b)

If (! A[j]) condition was as follows (! A[j] == 0)

Then, it prints a nonzero value in the array $\{m \mid m < n, (\exists i)\ [m= i^2]\}$

Q.10 (d)

From the given code it is determined that there will be no interchange in the values of x and y, since the parameters are passed by value, when the function swap (x, y) is called. As the scope of a and b lies within the function but not in the main program so there is interchanging in the formal parameters a and b but no exchange in actual parameters x and y will take place. It is only possible to exchange the values of x and y, if they are called by reference.

Q.11 (c)

Structured programming refers to programming in which control is passed from one instruction to another in a sequential order. Some of the examples of this type of programming are C and Pascal. This type of programming thus, has a goal to be able to infer the flow of control from the program text which means the user can execute the program according to his need. In structure programming various control structures such as switch-case, if-then-else, while, etc. allows a programmer to decode the flow of the program easily.

Q. 12 (c)

Whenever the a function's data type is not declared in the program, the default declaration is taken into, consideration. By default, the function foo in this is assumed to be of in type then also it returns a double type of value. This situation will generate compiler warning due to this mismatch leading to unintended results.

Q. 13 (d)

From the given code it is found that foo is a recursive function and when the function foo(a, sum) is called where a=2048 and sum=0 as per given conditions. Then, the execution of the function takes in the following manner

For n = 2048
k=2048% 10=foo (204, 8)
j= 2048/10 = 204
therefore, sum = 0+8=8
n = 204
k = 204%10 = foo(20, 12)
j=204/10=20 therefore,
sum=8+4=12
n = 20
k = 20% 10 = foo (2, 12)
j=20/10=2 therefore,
Sum =12+0 = 12
n = 2
k = 2% 10 = foo(0, 14)
y = 2/10 = 0 therefore,
sum = 12 + 2 = 14
at this point function will be terminated and the print sequence will be 2, 0, 4, 8, 0 since, sum is the local variable.

Q. 14 (b)
Object Oriented Programming (OOP) is a programming paradigm. The language is object oriented as it use objects. Objects are the data structures that contain data fields and methods together with their interactions.
The main features of the Programming techniques 'are
1. data abstraction
2. encapsulation
3. modularity
4. polymorphism
5. inheritance
Therefore, the essential features are given by statements (i) and (iv).

Q. 15 (c)
Languages needs declaration of any statement that we write before its use thus, the common property of both the languages is that both are declarative.

Q. 16 (c)
An abstract data type is a mathematical model for a certain class of data structures that have similar behavior. An abstract data type is indirectly defined by the operations and mathematical constraints thus, is a user defined data type, not a system defined, that can perform operations defined by it on that data.

Q. 17 (c)
A '*' in the C statement indicates a pointer. Thus, *f is a, pointer to a function which takes argument as (int*) thus an integer pointer and an int in the start of the statement indicates that the statement returns an int value, i.e., an integer value, so finally the given statement means that a pointer to a function that takes an integer pointer as an argument returns an integer.

Q. 18 (b)
For the given code only the statements S2 and S3, are valid and thus, are true. Since, the code may generate a segmentation fault at runtime depending on the arguments passed as the arguments are passed by reference and also the swap procedure is correctly implemented here.

Q. 19 (d)
From the given code and statement it can be concluded that the same work is performed by both work1 and work2, hence the statement S1 is true. To prove statement S2 true we consider work1 where, int x=a [/+2] is computed twice and the value returned by it is a[i+2]-1, however in work2, the assignment statement of variables £1 and t2 return f2-3 which implies that computation is done once only thus, S2 is also true.

In work 1	**But in work 2**
int x=a[i+2];	t, = i + 2

here a[j+2j is $t_2 = a[t_1]$ computed twice
return a[i+2J-1 return t_2 -3
so need for 2nd computation

Q. 20 (c)

In the while loop it is given that i<n and j<m and also a[il<b[j]so for this a[n-1]<b[j] thus, k=m+i-1 when i<n
This implies that 6[m-1]<=a[i]
Also, if 1<m then, due to the value of k being equal to n+y-1 the condition a[n-1]<b[j].

Q.21 (d)

We follow, the following steps to obtain the value of f(5)
f(5)
r=5
f(3)+2 = 18
↑
f(2) + 5 =16:
↑
f(1) + 5 = 11
↑
f(0) + 5 = 6
↑
1

Q. 22 (a)

The operator "?:" in C is the ternary operator which means that, if the expression is exp1 ? exp2: exp3, so it means, if exp1 is true then exp2 is returned as the answer else exp3 is the required answer.
So, in the given expression let us consider x = 3, y = 4, Z = 2
Then, expression becomes a
= (3>4)?((3>2)?3:2):((4>2?4:2)
From this, we get that 3>4 is false so we go for the else part of the statement which is 4>2 and is true thus/the answer is 4, the true part of the statement.

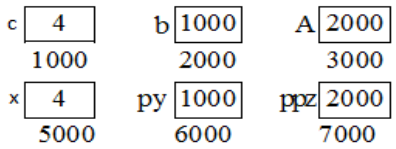
Q. 23 (d)

?1 is ((c = getchar ())! ='\n')
?2 is putchar (c);

Because the operator '1=' has higher priority than '=' Operator so according to this = getchar () should be contained in brackets and when the string is reversed then the function putchar (c) is used so that the characters can be printed.

Q. 24 (b)

The program gets executed in the following manner Graphical Representation



Now, considering,
int y, z;
**ppy += 1; z = *ppz = 6
*py += 2; y = *py = 6
x = 4 + 3 = 7
return x + y + z;
and
c = 4; b = &c; a = &b;
print f ("%d", f(c, b, a));
From the code,
The output is printed as 6+6+7=19.

Q. 25 (b)

As x = 15
fun (5, &x) and let &x = 200
then fun (5, 200) so, then t = 5f = 5 + 3 *f _ p = 5
this will return 8
when fun(4,200), then t = 3f= 3+2 *f _ p = 3
this will return 5
when fun(3,200) then t =2f=2+1*f_ p =2
this will return 3 .
for fun(2,200) t = 1 f =1 +-1*f _ p = 1
this will return 2
last for fun (1,200)*f _ p=1
will return 1
So fun(5, &x) returns 8 and this is the answer that will be printed.

Q. 26 (d)

In the given program it begins from main i & j globally initialized by 0

and 1 so when we call function f(&i, &j) the address of i and j passed, when p = q and *p = 2 means *q = 2, so value of *q is passed 2 and value of *q return to j & the value of i and j as 2, so, print f ("%d %d", i, j) give output (0, 2).

Q. 27 (d)

In a given program we take the test cases and apply.
First take T1, if all value equal means.
a = b = c = d
So, due to T_1, a = b condition satisfied and S_1 and S_4 executed.
So, from T_2 when all a, b, c, d distinct.
S_1, S_2 not execute, S_3 execute.
from T_3 when a = b then, S_1 execute but c = d so, S_2 not execute but S_3 and S_4 execute but we have no need of T_3 because we get all result from above two.
By using T_4. If a! = b and c = d.
So, S_1 not execute and S_2 and S_4 execute so all of S_1, S_2, S_3, S_4 execute and covered by T_1, T2 and T_4.

Q.28 (c)

The numbers given are 12, 7, 13, 4, 11, 6 Now, just move with the steps of the program in the sequence of the numbers are given. The sequence, in which numbers are entered is

12	7
13	4
11	6

Now, as per the program, we get,
(12 (7 - (13 - (4 + (11- (6 + 0)))))
=15.

Q.29 (d)

foo (513, 2)
= (513%2) + foo (513/2, 2)
=1 + foo (256, 2)
= 1 + 256%2 + foo (128, 2)
= 1 + 0+128%2 + foo (64, 2)
= 1 + 0+0 + foo (32, 2)
= 1 + 0 + 0 + 32%2 + foo (16, 2)
= 1 + 0+0+ 0+16%2+foo (8, 2)
= 1 + 0 + 0^r0 + 0 + 8%2 + foo (4, 2)
= 1+0+0+0+0+0+4%2+foo (2, 2)
= 1+0+0+0+0+0+0+1 %2 + foo (1, 2)
=1+o+0+0+0+0+0+01 %2+ foo(0, 2)
= 1+ 0+0+0 + 0 + 0 + 0 + 0 +1 + 0 = 2

Q. 30 (b)

foo (345, 10)
= 345%10 + foo (345/10, 10)
= 5 + foo (34, 10)
= 5 + 4 + foo (3910)
= 5 + 4 + 3= 12

Q. 31 (c)

GATE 2011
Suppose p base address of string.
p[3] is 'E', p[1] is 'A'
p + p [3] – p [1] = BA+'E'-'A' = BA+4
print f ("%s",BA+4) is 2011

Q. 32 (c)

In main a is initialized with 1 and it is a static variable.
Static variable will not change its value in any calling of function.In first call of ptrFun (void) function a is initialized with 2 as a static variable and b is normal auto type variable b = 1
a + = + + b
a = a + 2
a=4
Now 4, 2 will be printed.
Now, CPU control backs to main function
a = a +1
a = 1 + 1
a = 2 .
Again ptrFun (veid) function is called.
At this time a will not be initialized but b will be intialized.
Previously a= 4 and at this time b= 1
a + = + + b;
a + = 2,
a = 6;
Now, 6, 2 will be printed.

and CPU control returns back to main function a and b will be printed as 2, 0.

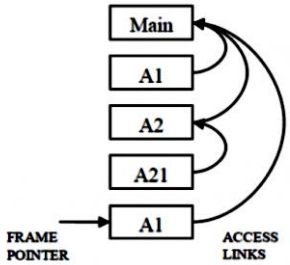
Q. 33 (d)

Auto int a = 1 is same as int a = 1
Register int a = 2 is same as int a = 2
When we call ptrFun (void) function then a is initialized with 2 and b with 1.
a + = ++ b;
(pre increment of b will give incremented value of b)
or a + = 2;
a = a + 2;
a = 2+2 = 4;
Now, a and b are printed as 4, 2. Now control returns back to main function a is incremented by 1 and maiin a is initialized with1.
So, now
a = a +1
a=1+1
a = 2
Again ptrFun (void) function is called then again a is intialized with 2 and b with 1 .
a + = + + b;
a + = 2;
a = 4;
and this function prints a, b as 4, 2.
Now, control of CPU returns back to main. In main print the value of a and b. In main a - T and b is not initialized and b is a global or external variable, so by default it is zero.

4	2
4	2
2	0

Q. 34 (d)

Link to activation record of closest lexically enclosing block in program text. It depends on the static program text.



Main → A_1 → A_2 → A_{21} → A_1

Q. 35 (b)

Return value f (p , p) if the value of p is initialized to 5 before the call.
Since, reference of p is passed as 'x'
Thus, any change in value of x in f would be reflected globally.
The recursion can be broken down as

$f(x,c)^{5\ 5}$

$x * f(x,c)^{6\ 4}$

$x * f(x,c)^{7\ 3}$

$x * f(x,c)^{8\ 2}$

$x * f(x,c)^{9\ 1}$

1 (∵ c = 0 in this call)
∴ Answer is = $x^4 * x * x * x$
The final value of x = 9
9^4 = 6561 i.e., Option (b)

Q. 36 (d)

For any value of 'j' other than 50 the function will return 0, for j=50, then condition (i==j) will be true, it will print "something" and function will be called recursively with same value till the run time stack overflows.

Q.37 (c)

For i=1, the multiplication statement is executed (n-1)+(n-2)+..2+ 1 times.
For i = 2, the statement is executed (n-2) + (n-3) + .. 2 + 1 times…….
For i = n-1, the statement is executed once.
For i=n, the statement is not executed at all

So the statement executes for [(n-1) + (n-2) + .. 2 + 1] + [(n-2) + (n-3) + .. 2 + 1] + … + 1 + 0 times.
It can be written as S = [n*(n-1)/2 + (n-1)*(n-2)/2 + ….. + 1]
We know that Series S1 = n^2 + (n-1)^2 + .. 1^2.
The sum of this series is n*(n+1)*(2n+1)/6
S1– 2S = n + (n-1) + … 1 = n*(n+1)/2
2S = n*(n+1)*(2n+1)/6 – n*(n+1)/2
S = n*(n+1)*(n-1)/6

Q.38 (9)
Initially num = 110110011, count= 0
count = 1; num = 101100110 after 1st right shift
count =2; num = 011001100 after 2nd right shift
:
:
Count = 9; num = 000000000 after 9th right shift.
After nine right shifts, num = 0; and while loop terminates count = 9 will be returned

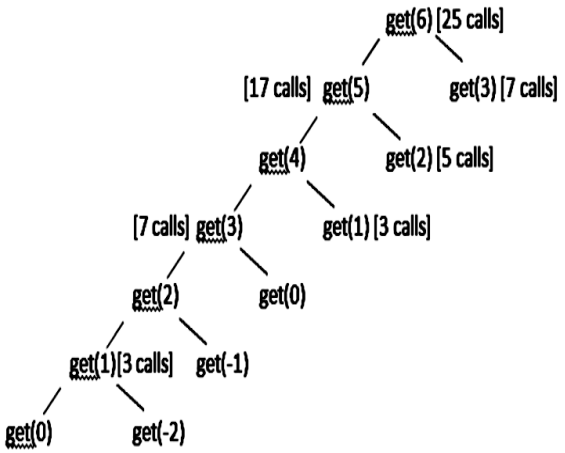
Q.39 (b)
P = n*(n-1)*(n-2)/6
It we multiply n, (n-1), (n-2) at once, it might go beyond the range of unsigned integer
(resulting overflow). So options (A) and (D) are cannot be the answer. If n is even or odd n×(n-1)/2 will always result in integer value (no possibility of truncation, so more accuracy) where as incase of n*(n-1)/3, its not certain to get integer always truncation possible, so less accuracy).

Q.40 (1.73)
If condition given in function definition should be 'TRUE', for f (q) to return value q .
The condition is as follows:
if (abs(x*x –3)<0.01) return x;
The above condition will be true when x^2-3 is almost 0 so x=1.73

Q.41 (d)
pi contains the address of i. So scanf("%d",pi) places the value entered in console into variable i. So printf("%d\n",i+5), prints 5 more than the value entered in console.

Q.42 (51)
fun(5)= 1+ fun(1)*fun(4) + fun(2)*fun(3) + fun(3) * fun(2) + fun(4)* fun(1)
Similarly calculate fun(1), fun(2), fun(3)
fun(1) = 1, fun(2) =2, fun(3) = 5, fun(4) = 15
Finally fun(5) = 51

Q.43 (b)



Q.44 (10)
The value of j with the expression j = 2*3 / 4 + 2.0 / 5 + 8 / 5 will be 2.
k -= --j; makes k value as -1.
For i=0: i+k will be -1, default block gets executed. So printfcount=1
For i=1: i+kwill be 0, default block gets executed. So printfcount=2
For i=2: i+k will be 1, all blocks are executed. So printfcount=5
For i=3: i+k will be 2, all blocks are executed. So printfcount=8
For i=4: i+k will be 3, case 3 and default blocks are executed. So print f count=10.

Q.45 (230)
x = x+ f1() + f2() + f3() + f2();
f1() returns 26, f2() returns 51, f3() returns 100.
Second call to f2() returns 52 since x is static variable.
So x= 1+26+51+100+52=230

Q.46 (d)
The program prints all characters before space in reverse order i.e. DCBA because printing is after the recursive call.

Q.47 (1204)
P=S1+2; which means p holds the address of 3rd character
*p='0'; the 3rd character becomes 0
So the string printed is 1204

Q.48 (140)
The value of ptr-p is 1 and value of **ptr is 40.

Q.49 (d)
Since function prototype is void f(int, short) i.e., f is accepting, arguments int, short and its return type is void. So f(i, *p) is correct answer.

Q.50 (2016)
Whatever modifications are performed in mystery ( ) function, those modifications are not reflected in main ( ) function so it will print 2016.

Q.51 (d)
While(b! = a) is the condition because, a is moving from starting of array and b is moving from end of array. When they both are equal we stop the process.

Q.52 (d)

Q.53 (30)
Parameter i is passed by reference and j is passed by value. So in the function f, value of m is 15 (but j is not effected). Value of I is 5+15=20 (*p is nothing but i). In main, i+j becomes 20+10=30

Q.54 (c)
This can be verified by taking values for X and Y.

Q.55 (3)

3	5	2	6	4
2000	2002	2004	2006	2008

f(2000,5) :return max(f(2002,4), -2)
f(2002,4) : return max(f(2004,3), 3)
f(2004,3): return max(f(2006,2), -4)
f(2006,2): return max(f(2008,1),2)
f(2008,1): return max(f(2010,0),0)
During back track, the maximum value returned is 3

Q.56 (d)
Since direct access is possible with only index addressing mode is given options. So option (d) is the correct answer.

Q. 57 (d)
The code will run and give output = 10, so option A and B are discarded.
int * x= malloc (sizeof (int)); This statement assigns a location to x. Now, (int*) malloc (sizeof (int)); again assigns a new location to x, previous memory location is lost because now we have no reference to that location resulting in memory leak. Therefore, option D is correct.

Q. 58 23
Digits: 5-0101, 4-0100, 3-0011, 2-0010, 1-0000
Count of 1s: 2, 3, 5, 6, 7
Total: 2+3+5+6+7 = 23
Total (i) = 23

Therefore, the output is 23

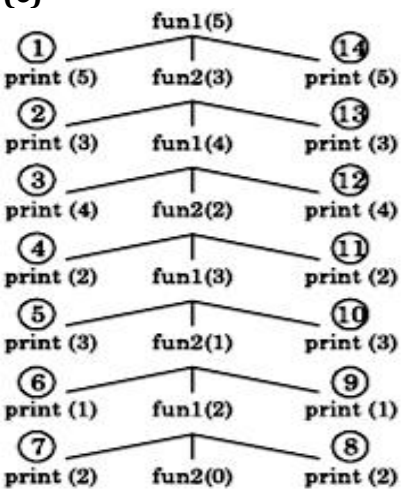
Q.59 (c)

In the function foo every time in the while foo is called with the value 3 because val is passed with post decrement operator so the value 3 is passed and val is decremented later. Every time the function is called a new variable is created as the passed variable is passed by value, with the value 3. So the function will close abruptly without returning any value.
In the function bar, in the while loop value the value of val variable is not decrementing, it remains 3 only. Bar function in the while loop is called with val-1 i.e 2 but value of val is not decremented, so it will result in infinite loop.

Q.60 (3)

Strlen (s) =3
Strlen (t) =5
Value of c is 0
[Strlen (s) –strlen (t) > 0] : Strlen(s) : Strlen(t);
[Strlen(s) - Strlen(t)]=-2
Since it is givne in question Strlen (s)–strlen(t) will give unsigned value.
So, [Strlen (s) - Strlen (t)]=[-2]=2
2 > 0: Strlen (s) : Strlen (t) ;since 2 > 0 will evaluate True so it will give output as strlen (s)=3
Hence '3' will be printed.

Q.61 (c)



Hence the output is : 53423122132435

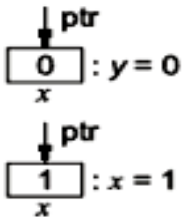
Q.62 (a)

- **Static char var**; Initialization of a variable located in data section of memory .
- **m = malloc(10); m=NULL**; A lost memory which can't be freed because free (m) is missed in code.
- **Char*ptr[10];** : Sequence of memory locations to store address.

register int var 1: Request to allocate a CPU register to store data.

Q.63 (c)

Print xy(1,1)



Hence the output will be (1,0)

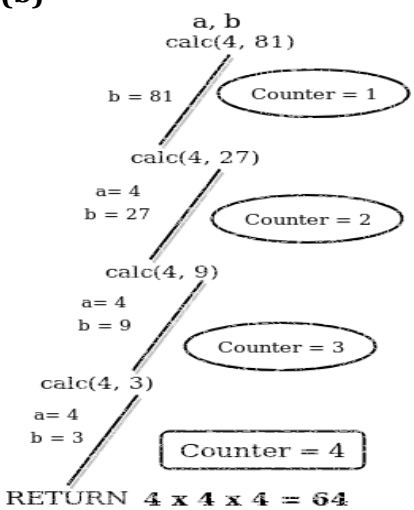
Q.64 (c)

Q.65 (0)

1. int m=10; // m= 10
2. int n,n1;
3. n = + + m; //n=11
4. n1= m + +; //n1=11,m =12
5. n- -; //n=10
6. - - n1; //n1=10
7. n-= n1; //n = 0
8. printf ("%d",n);

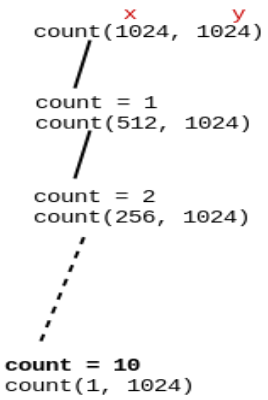
The output will be 0.

Q.66 (b)



Q.67 (c)

Firstly, Count (1024, 1024) will be called and value of x will be deducted by x/2. 'x' will be printed 10 times. On count=10, value of x will become 1.



Since x=1, inner else loop will start executing. On each call value of y will be decreased by 1 and will be executed until value of y becomes 1 (which is on 1023th call). Since count() is called recursively for every y=1023 and count() is called 10times for each y. Therefore, 1023 x 10 = 10230 Answer is 10230.

Q.68 (A)

fun1(char *s1, char *s2)

Above function scope is local, so the value changed here won't affect actual parameters. SO the values will be 'Hi Bye'.

fun2(char **s1, char **s2)

In this function value is pointer to pointer, so it changes pointer of the actual value. So values will be 'Bye Hi'

Answer is 'Hi Bye Bye Hi'

Q.69 (b)

```
// n takes 2^40

unsigned long int fun(unsigned long int n)
{

// initialized sum = 0

unsigned long int i, j, sum = 0;

//First it takes i = n = 2^40,

//then it divides i by 2 and incremented once j

//each time, that's will make makes j = 40,

for( i=n; i>1; i=i/2) j++;

//Now the value of j = 40,

//it divides j by 2 and incremented once sum

//each time, that's will make makes sum = 5,

for( ; j>1; j=j/2) sum++;

//returns sum = 5

return sum;

}
```

So, answer is 5.

GATE QUESTIONS(ARRAYS)

Q.1 Suppose you are given an array s[1...n] and a procedure reverse (s, i, j) which reverses the order of elements in a between positions i and j (both inclusive) What does the following sequence do, where 1≤ k≤n:

reverse (s, 1 , k);
reverse (s, k + 1, n);
reverse (s, 1, n);

a) Rotates s left by k positions
b) Leaves s unchanged
c) Reverses all elements of s
d) None of these

[GATE -2000]

Q.2 A program P reads in 500 integers in the range [0, 100] representing the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies?

a) An array of 50 numbers
b) An array of 100 numbers
c) An array of 500 numbers
d) A dynamically al located array of 550 numbers

[GATE- 2005]

Q.3 Consider following C function in which size is the number of elements in the array E:

```
int MyX(int *E, unsigned int size)
        {
          int Y = 0;
          int Z;
          int i, j, k;
          for(i = 0; i < size; i++)
               Y = Y + E[i];
           for(i = 0; i < size; i++)
             for(j = i; j < size; j++)
             {
                 Z = 0;
                 for(k = i; k <= j; k++)
                     Z = Z + E[k];
                 if (Z > Y)
                   Y = Z;
             }
          return Y;
          }
```

The value returned by the function MyX is the

a) Maximum possible sum of element in sub-array of array E
b) Maximum element in any sub-array of array E
c) Sum of maximum elements in all possible sub-arrays of array E
d) Sum of all the elements in the array E

[GATE- 2014]

Q.4 Let A be a square matrix of size n X n. Consider the following pseudo code. What is the expected output?

```
C = 100 ;
for i =1 to n do
for j = 1 to n do
  {
        Temp = A[i][j] + C;
        A[i][j] = A[j][i];
        A[j][i] = Temp – C;
  }
        for i = 1 to n do
          for j = 1 to n do
            Output (A[i][j]);
```

a) The matrix A itself
b) Transpose of matrix A
c) Adding 100 to the upper diagonal elements and subtracting 100 from lower diagonal elements of A
d) None of the above

[GATE -2014]

Q.5 What is the output of the following C code? Assume that the address of x is 2000 (in decimal) and an integer requires four bytes of memory.

```
int main()
```

{
unsigned int x[4][3]={{1,2,3}, { 4,5,6} , {7,8,9} , {10,11,12}};
print f(" %u, %u, %u", x+3, *(x+3) , *((x+2)+ 3));
}
a) 2036, 2036, 2036
b) 2012, 4, 2204
c) 2036, 10, 10
d) 2012, 4, 6

[GATE- 2015]

Q.6 Suppose c=< c[0] ,....,c[k- 1]> is an array of length k, where all the entries are from the set {0, 1}. For any positive integers a and n, consider the following pseudo code.

DOSOMETHING (c, a, n)
z := 1
for i :=0 to k-1
    z := z^2 mod n;
    if c[i] = 1;
        z := (z*a) mod n;
return z;

If k = 4, c =<1, 0, 1, 1>, a = 2 and n = 8, then the output of DOSOMETHING (c, a, n) is ……..

[GATE- 2015]

Q.7 Consider the following two C code segments. Y and X are one and two dimensional arrays of size n and n × n respectively, where 2<=n<=10. Assume that in both code segments, elements of Y are initialized to 0 and each element X[i] [j] of array X is initialized to i + j. Further assume that when stored in main memory all elements of X are in same main memory page frame

Code segment 1:
// initialize element of Y to 0
// initialize elements X[i][j] of X to 1+j
for (i = 0; i < n; i++)
Y[i] + = X[0][i];

Code segment 2:
//initialize elements of Y to 0
//initialize elements X[i][j] of X to 1+j
for (i = 0; i < n; i++)
Y[i] + = X[i][0];

Which of the following statements is/are correct?
S1: Final contents of array Y will be same in both code segments
S2: Elements of array X accessed inside the for loop shown in code segment 1 are contiguous in main memory
S3: Elements of array X accessed inside for loop shown in code segment 2 are contiguous in main memory.
a) Only S2 is correct
b) Only S3 is correct
c) Only S1 and S2 are correct
d) Only S1 and S3 are correct

[GATE -2015]

Q.8 Consider the following C program.

#include <stdio. h>
#include <string .h>
int main ( )
{
  char*c="GATECSIT2017";
   char*p = c
printf("%d".(int)strlen (c+2[p]-6[p]-1));
return 0;
}

The output of the program is _____.

[GATE-2017]

Q.9 Consider the following snippet of a C program. Assume that swap (&x, &y) exchanges the contents of x and y.

int main ( )
{
 int array [ ] ={3,5,1,4,6,2};
int done =0;
int i;
while(done = =0)
{
    done =1;

```
    for  (i =0;< = 4 ; i ++)
    {
        if (array[ i] < array [i+1])
        swap(& array[i], &array [i+1]);
            done =0;
     }
}
for (i=5; i >1; i - - )
{
       if (array) [i] > array [i-1])
       {
     swap(& array [i], & array [i-1]);
       done = 0;
        }
}
printf({"%d", array[3]);
}
```

The output of the program is ___.

[GATE-2017]

ANSWER KEY:

1	2	3	4	5	6	7	8	9
(a)	(a)	(a)	(a)	(a)	0	(c)	2	3

EXPLANATIONS

Q. 1 **(a)**
From the given conditions it can be clearly concluded that, the given sequence rotates s left by k positions.

Q.2 **(a)**
As per the given conditions it is known that the frequency of those students is printed whose score is greater than 50 so, the range of frequency contains score from 50 to 100 so; an array of 50 numbers is suitable for representing the frequency.

Q.3 **(a)**
The first for loop calculates the sum of elements in the array E and stores in Y. In the nested for loop, it calculates the sum of elements of all possible sub arrays.
In the condition Z>Y, it Checks whether sum of elements of each sub array is greater than the sum of elements of array if so, that sum is assigned to Y, if not 'Y' will be the sum of elements of complete array.

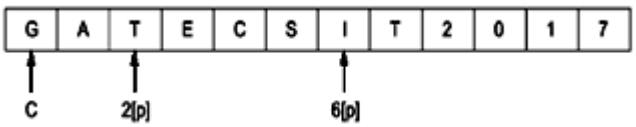
Q.4 **(a)**
In the computation of given pseudo code for each row and column of Matrix A, each upper triangular element will be interchanged by its mirror image in the lower triangular and after that the same lower triangular element will be again re-interchanged by its mirror image in the upper triangular, resulting the final computed Matrix A same as input Matrix A.

Q.5 **(a)**
Address of x is 2000
Since x is considered as pointer to an array of 3 integers and an integer takes 4 bytes, value of x+3= 2000+3*3*4=2036
The expression *(x+3) also prints same address as x is 2D array.
The expression *(x+2)+3= 2000+2*3*4+3*4=2036

Q.6 **(0)**
For i=0, z=1, c [0]=1, z=1*2 mod 8= 2
For i=1, z = 2*2 mod 8 = 4, c [1]=0 so z remains 4
For i=2, z=16 mod 8=0

Q.7 **(c)**
In C, 2D arrays are stored in row major order. So S2 is correct but not S3.

Q.8 **(2)**
(C+2[P]-6[P]-1)



C+T-I-1
= C+11-1
=C+10
Hence print will print '2'.

Q.9 **(3)**

3	5	1	4	6	2
0	1	2	3	4	5

First for loop:

(i = 0)	5	3	1	4	6	2
(i = 1)	5	3	1	4	6	2
(i = 2)	5	3	4	1	6	2
(i = 3)	5	3	4	6	1	2
(i = 4)	5	3	4	6	2	1

Second for loop

(i = 5) 5 3 4 6 2 1
(i = 4) 5 3 4 6 2 1
(i = 3) 5 3 6 4 2 1
(i = 2) 5 6 3 4 2 1
(i = 1) 6 5 3 4 2 1

Now since done is'0' hence the for loops will execute again.

First for loop:

(i = 0) 6 5 3 4 2 1
(i = 1) 6 5 3 4 2 1
(i = 2) 6 5 4 3 2 1
(i = 3) 6 5 4 3 2 1
(i = 4) 6 5 4 3 2 1

Second for loop:

(i = 5) 6 5 4 3 2 1
(i = 4) 6 5 4 3 2 1
(i = 3) 6 5 4 3 2 1
(i = 2) 6 5 4 3 2 1
(i = 1) 6 5 4 3 2 1

Value of done is still '0', hence the for loop will execute again First for loop:

This time there will be no change by the for loop.

The value of done is '1'. Hence the loop terminates as

6	5	4	3	2	1
0	1	2	3	4	5

The output of the program will be '3'.

GATE QUESTIONS(STACKS AND QUEUES)

Q.1 The best data structure to check whether an arithmetic expression has balanced parentheses is a

a) queue b) stack
c) tree d) list

[GATE -2004]

Q.2 An implementation of a queue Q, using two stacks S1 and S2, is given below

```
void insert (Q, x){
        push (Si , x);
}
void delete (Q) {
        if (stack-empty (S2)) then
        if (stack-empty (S1 )) then
        {
                print ("Q is empty");
                return;
        }
        else
        while(! (Stack-empty(S1)))
        {
                x = pop (S1);
                push (S2, x);
        }
x =pop (S2);
}
```

Let n insert and m delete operations be performed in an arbitrary order on an empty queue. Let x and y be the number of push and pop operations performed respectively in the process. Which one of the following is true for all m and n?

a) $n + m \leq x < 2n$ and $2m \leq y \leq n + m$
b) $n + m \leq x < 2n$ and $2m \leq y \leq 2n$
c) $2m \leq x < 2n$ and $2m \leq y \leq n + m$
d) $2m \leq x < 2n$ and $2m \leq y \leq 2n$

[GATE- 2006]

Q.3 The following postfix expression with single digit operands is evaluated using a stack :

8 2 3 ^ / 2 3 * +5 1 * -

Note that ^ is the exponential operator. The top two elements of the stack after the first * is evaluated are

a) 6, 1 b) 5, 7
c) 3, 2 d) 1, 5

[GATE-2007]

Q.4 Suppose a circular queue of capacity (n -1) elements is implemented with an array of n elements. Assume that the insertion and deletion operations are carried out using REAR and FRONT as array index variables, respectively. Initially REAR =FRONT = 0. The conditions to detect queue full and queue empty are

a) Full: (REAR + 1) mod n = = FRONT
   Empty: REAR = = FRONT
b) Full: (REAR + 1) mod n = = FRONT
   Empty: (FRONT + 1) mod n = = REAR
c) Full: REAR = =FRONT
   Empty: (REAR +1) mod n = = FRONT
d) Full: (FRONT + 1) mod n = = REAR
   Empty: REAR = = FRONT

[GATE-2012]

Q.5 Consider the following operation along with Enqueue and Dequeue operations on queues, where k is a global parameter.

```
MultiDequeue(Q){
        m = k;
        while (Q is not empty)  and
        (m > 0)
        {
                Dequeue(Q);
                m = m – 1;
        }
```

}

What is the worst case time complexity or a sequence of n queue operations on an initially empty queue?

a) $\theta(n)$ b) $\theta(n+k)$

c) $\theta(nk)$ d) $\theta(n^2)$

[GATE-2013]

Q.6 Suppose a stack implementation supports an instruction REVERSE, which reverses the order of elements on the stack, in addition to the PUSH and POP instructions. Which one of the following statements is TRUE with respect to this modified stack?

a) A queue cannot be implemented using this stack.

b) A queue can be implemented where ENQUEUE takes a single instruction and DEQUEUE takes a sequence of two instructions.

c) A queue can be implemented where ENQUEUE takes a sequence of three instructions and DEQUEUE takes a single instruction.

d) A queue can be implemented where both ENQUEUE and DEQUEUE take a single instruction each.

[GATE -2014]

Q.7 Consider the C program below.

```
#include<stdio.h>
int *A, stkTop;
int stkFunc (int opcode, int val)
{
static int size =0, stkTop=0;
switch (opcode)
{
  case -1 :
        size = val; break;
  case 0 :
        if (stkTop < size)
              A [stkTop++] = val;
        break;
  default :
        if (stkTop)
              return A [--stkTop];
}
return -1;
}
int main ( )
{
int B[20] ; A = B;
stkTop = -1;
stkFunc (-1, 10);
stkFunc (0, 5);
stkFunc (0, 10);
printf ("%d\n", stkFunc(1, 0) +
stkfunc(1, 0);
}
```

The value printed by the above program is _____.

[GATE -2015]

Q.8 The result of evaluating the postfix expression 10 5 + 60 6 / * 8 – is _____

a) 284 b) 213

c) 142 d) 71

[GATE- 2015]

Q.9 A queue is implemented using an array such that ENQUEUE and DEQUEUE operations are performed efficiently. Which one of the following statements is **CORRECT** (n refers to the number of items in the queue)?

a) Both operations can be performed in O(1) time

b) At most one operation can be performed in O(1) time but the worst case time for the other operation will be Ω(n)

c) The worst case time complexity for both operations will be Ω(n)

d) Worst case time complexity for both operations will be Ω(log n)

[GATE -2016]

Q.10 Let Q denote a queue containing sixteen numbers and S be an empty stack. Head(Q) returns the element

at the head of the queue Q without removing it from Q. Similarly Top(S) returns the element at the top of S without removing it from S. Consider the algorithm given below.

```
while Q is not Empty do
if S is Empty OR Top(S) ≤ Head(Q) then
x:= Dequeue(Q);
Push(S, x);
else
x:= Pop(S);
Enqueue(Q, x);
end
end
```

The maximum possible number of iterations of the while loop in the algorithm is____.

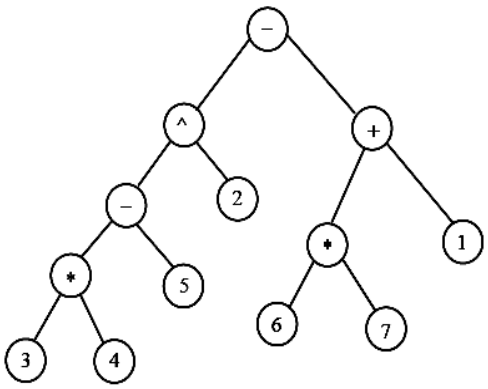
[GATE -2016]

Q.11 Consider the following New-order strategy for traversing a binary tree:

- Visit the root;
- Visit the right subtree using New-order;
- Visit the left subtree using New-order;

The New-order traversal of the expression tree corresponding to the reverse polish expression
4 * 5 – 2 ^ 6 7 * 1 + – is given by:

a) + – 1 6 7 * 2 ^ 5 – 3 4 *
b) – + 1 * 6 7 ^ 2 – 5 * 3 4
c) – + 1 * 7 6 ^ 2 – 5 * 4 3
d) 1 7 6 * + 2 5 4 3 * – ^ –



[GATE -2016]

Q.12 A circular queue has been implemented using a singly linked list where each node consists of a value and a single pointer pointing to the next node. We maintain exactly two external pointers **FRONT** and **REAR** pointing to the front node and the rear node of the queue, respectively. Which of the following statements is/ are **CORRECT** for such a circular queue, so that insertion and deletion operations can be performed in 0(1) time?

I. Next pointer of front node pints to the rear node.
II. Next pointer of rear node points to the front node

a) I only b) II only
c) Both I and II d) Neither I nor II

[GATE -2017]

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12
(b)	(a)	(a)	(a)	(a)	(c)	15	(c)	(a)	256	(c)	(b)

EXPLANATIONS

Q. 1 **(b)**

The best data structure that can be used is stack. Let's check the algorithm -

Q. 2 **(a)**

From the given implementation total numbers of push operations are given by n+m, where n tells no. of insertion and m tells the number of deletions. Maximum numbers of insert operations allowed are 2n, so $n+m \le x \le 2n-1$. Also the numbers of pop operations are n+m. But the numbers of delete operations are 2m that are less than the number of pop operations so, the required condition becomes 2m<=n+m-2.

From the above two conditions, we get $n+m \le x \le 2n$ and $2m \le n+m$

Q. 3 **(a)**

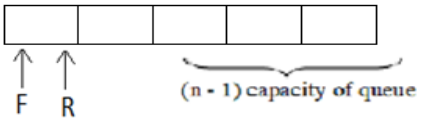
From the given expression,

Expression symbol Op1Op2 value tops(RL)

8				8
2				8, 2
3				8, 2, 3
∧	2	3	8	8, 8
/	8	8	1	1
2				1, 2
3				1, 2, 3
*	2	3	6	1, 6

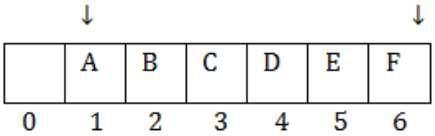
So, the top 2 elements of the stack are 6, 1 after the first * is evaluated.

Q. 4 **(a)**



Take a full queue e.g., n = 7

Front Rear



(insert six elements one by one)

(R + 1) mod n

⇒(6 +1) mod 7

⇒0 that means queue is full.

So, for full T (R +1) and n =Front

Take an empty queue and given front = rear= 0. Here front and rear is same so for empty queue.

Front = Rear

Q. 5 **(a)**

To compute the worst case time complexity of a sequence of n queue operations on an initially empty queue is θ(n).

Complexity of a sequence of 'n' queue operations = Total complexity of Enqueue operations (a) + Total complexity of Dequeue operations (β).

Total complexity of Dequeue operations (β). < Total complexity of Enqueue operations β<α ...(i)
Total complexity of queue operations
(β). < n ...(ii)
Total complexity of n operations =α+ β.
<α +α [From ...(i)]
<n + n [From ...(ii)]
<2 n
Worst Case Time Complexity of 'n' operations is (a).θ (n).

Q.6 (c)
Option (a) is false because queue can be implemented by using the modified stack as by reversing the stack. LIFO will become FIFO.
Implementation of ENQUEUE & DEQUEUE takes four sequence of instructions as follows:
1. Enqueue: Reverse, Push, Revesre
   Dequeue: POP
   (OR)
2. Enqueue: Push
   Dequeue: Reverse, POP, Reverse

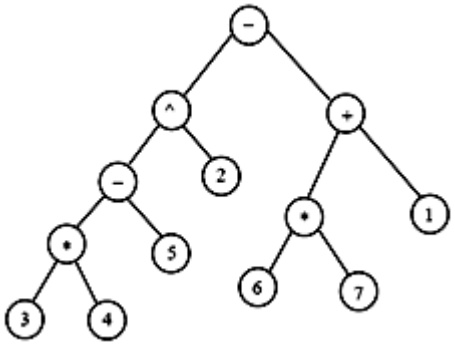
Q.7 (15)
stkFunc (-1,10); // initialize size as 10
stkFunc (0, 5); // push 5
stkFunc (0, 10); // push 10
printf ("%d\n", stkFunc(1,0)+ stkfunc(1, 0));
Pop the two elements, add them and print the result which is 5+10 =15

Q.8 (c)
Operator +: 10+5=15
Stack: 15 60 6
Operator /: 60/6=10
Stack: 15 10
Operator *: 15*10 = 150
Stack: 150 8
Operator -: 150 – 8 =142

Q.9 (a)
Queue has both front and rear pointers at which deletion and insertion is done respectively. So there is no dependency on number of elements.
++front deletes the element, Q[++rear] = x inserts the element. So both operations can be performed in O(1) time.

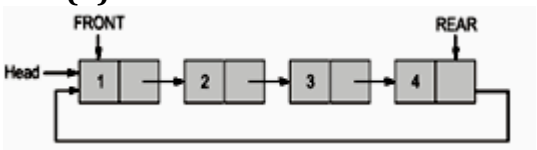
Q.10 (256)
Trying with queue elements, say n = 1, 2, 3, then maximum number of iterations are noticed as n^2. Here n = 16, So answer will be 256.

Q.11 (c)
The given expression is postfix expression and the corresponding expression tree is as below.



The new order traversal of the above tree is: – + 1 * 7 6 ^ 2 – 5 * 4 3

Q.12 (b)



Since insertion in a queue are always from REAR and deletion is always form FRONT. Hence having the next pointer of REAR node pointing to the FRONT node will lead to both insertion and deletion operation in O (1) time.

GATE QUESTIONS(LINKED LIST)

Q.1 In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is

a) $\log n$ b) $\frac{n}{2}$

c) $\log_2^n - 1$ d) n

[GATE-2002]

Q.2 Consider the function f defined below:

```
struct item {
        int data;
        struct item * next;
};
int f(struct item *p) {
        return ((p==NULL) || (p-> next==NULL) || ((p-> data <=p-> next ->data) &&
        f(p-> next)));
}
```

For a given linked list p, the function f returns 1, if and only, if

a) The list is empty or has exactly one element
b) The elements in the list are sorted in non-decreasing order of data value
c) The elements in the list are sorted in non-increasing order of data value
d) Not all elements in the list have the same data value

[GATE 2003]

Q.3 The following G function takes a single-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of list after the function completes execution?

```
struct node {
        int value;
        struct node *next;
};
void rearrange (struct node*list){
        struct node *p, *q;
int  temp;
        if(!list || !list ->next) return;
        p =list; q = list -> next;
        while (q){
                temp =p -> value;
                p -> value =q -> value;
                q -> value = temp;
                p = q-> next;
                q = p? p-> next: 0;
                        }}
```

a) 1, 2, 3, 4, 5, 6, 7 b) 2, 1, 4, 3, 6, 5, 7
c) 1, 3, 2, 5, 4, 7, 6 d) 2, 3, 4, 5, 6, 7, 1

[GATE 2008]

Q.4 The following C function takes a simply-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank.

```
typedef struct node {
        int value;
        struct node *next;
} Node;
Node *move_to_front (Node *head) {
Node *p, *q;
if (head ==NULL || (head->next==NULL))
return head;
q= NULL; p= head;
while (p-> next!= NULL) {
        q=p;
        p=p-> next;
}
_____;
return head;
}
```

Choose the correct alternative to replace the blank line.

a) q = NULL; p->next = head; head = p;
b) q-> next =NULL; head = p; p->next = head;
c) head = p; p->next=q; q-> next = NULL;
d) q-> next = NULL; p->next== head;head = p;

[GATE -2010]

ANSWER KEY:

1	2	3	4
(d)	(d)	(b)	(d)

EXPLANATIONS

Q.1 **(d)**

Linked list is also known as one way list. As we know that linked list is a list implemented by each item having a link to the next item. It is also a linear collection of data elements known as nodes. It needs a comparison with every node (in worst case) in the list in the near order. So number of comparisons is n.

Q.2 **(d)**

When all the elements in the list do not have the same value, the function f returns the value as 1 for p==NULL or p → next == NULL or p → data <= next → data and p → next

Q.3 **(b)**

The input sequence gets changed due to the reason that C code above is non-recursive.

Input sequence → 1, 2, 3, 4, 5, 6, 7

Output sequence → 2, 1, 4, 3, 6, 5, 7

So, the non-recursive nature of the C code interchanges the values in position of one and two then interchanges values in position of three and four and then interchanges values in fifth and six positions.

Q.4 **(d)**

The following code sequence will modify the last element to the front of the list and return the modified list.

q → next = NULL;
p → next = head;
head = p;

GATE QUESTIONS(TREES)

Q.1 Consider the following nested representation of Binary trees: (XYZ) indicates Y and are the left and right sub trees, respectively, of node Z. Note that Y and Z may be NULL, or further nested. Which of the following represents a valid binary tree?

a) (1 2 (4 5 6 7))
b) (1 (2 3 4) 5 6) 7)
c) (1 (2 3 4) (5 6 7))
d) (1 (2 3 NULL (4 5))

[GATE -2000]

Q.2 The number of leaf nodes in a rooted tree of n nodes, with each node having 0 or 3 children is

a) $\frac{n}{2}$ b) $\frac{(n-1)}{3}$
c) $\frac{(n-1)}{2}$ d) $\frac{(2n+1)}{3}$

[GATE-2002]

Q.3 suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual order in of n natural numbers. What is the in-order traversal sequence of the resultant tree?

a) 7 5 1 0 3 2 4 6 8 9
b) 0 2 4 3 1 6 5 9 8 7
c) 0 1 2 3 4 5 6 7 9
d) 9 8 6 4 2 3 0 1 5 7

[GATE 2003]

Q.4 Let T(n) be the number of different binary search trees on n distinct elements. Then,

$T(n)=\sum_{k=1}^{n} T(K-1)T(x)$, where x is

a) n – k + 1 b) n – k
c) n – k – 1 d) n – k - 2

[GATE 2003]

Q.5 The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, and 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)?

a) 2 b) 3
c) 4 d) 6

[GATE -2004]

Q.6 Consider the following C program segment

```
structCellNode{
        structCellNode *leftChild;
        int element;
        structCellNde *rightChild;
}
int Dosomething (structCellNode *ptr){
        int value = 0;
        if (ptr ! = NULL){
            if (ptr ->leftChild ! NULL)
            value = 1 + Domething (ptr->leftChild);
            if (ptr->rightChild ! = NULL)
            value = max (value, 1 + DoSomething (ptr->rightChild));
        }
return (value);
}
```

The value returned by the function Do Something when a pointer to the root of a non-empty tree is passed as argument is

a) the number of leaf nodes in the tree
b) the number of nodes in the tree
c) the number of internal nodes in the tree
d) the height of the tree

[GATE- 2004]

Q.7 Consider the label sequences obtained by the following pairs of traversals on a labelled binary tree. Which of these pairs identify a tree uniquely?

i) pre-order and post order
ii) in-order and post order
iii) pre-order and in order
iv) level order and post order

a) (i) only b) (ii) and (iii)
c) (iii) only d) (iv) only

[GATE-2004]

Q.8 In a complete k-ary tree, every internal node has exactly k children. The number of leaves in such a tree with n internal nodes is

a) nk b) (n-1)k+1
c) n(k-1)+1 d) n(k-1)

[GATE-2005]

Q.9 How many distinct binary search trees can be created out of 4 distinct keys?

a) 5 b) 14
c) 24 d) 42

[GATE-2005]

Q.10 A priority-queue is implemented as a max-head. Initially, it has 5 elements. The level-order traversal of the heap is given below.

10, 8, 5, 3, 2

Two new elements 1 and 7 are inserted in the heap in that order. The level-order traversal of the head after the insertion of the element is

a) 10, 8, 7, 5, 3, 2, 1
b) 10, 8, 7, 2, 3, 1, 5
c) 10, 8, 7, 1, 2, 3, 5
d) 10, 8, 7, 3, 2, 1, 5

[GATE-2005]

Q.11 Post order traversal of a given binary search tree, T produces the following sequence of keys

10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29

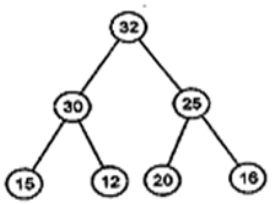
Which in one of the following sequences of keys can be the result of an in-order traversal of the tree T?

a) 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60,95
b) 9, 10, 15, 22, 40, 50, 60, 95, 23, 25, 27,29
c) 29, 15, 9, 10, 25, 22, 23, 27, 40, 60, 50,95
d) 95, 50, 60, 40, 27, 23, 22, 25, 10, 9,15,29

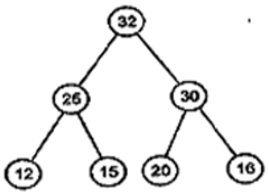
[GATE 2005]

Q.12 The elements 32, 15, 20, 30, 12, 25, 16, are inserted one by one in the given order into a max-heap. The resultant max-heap is

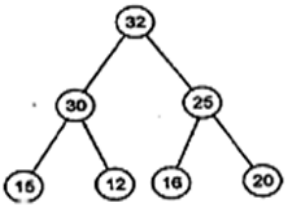
a)



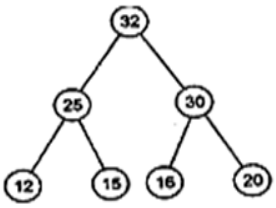
b)



c)



d)



[GATE 2005]

Statements for Linked Answer Questions Q.13 and Q.14

A 3-ary max heap is like a binary max heap, but instead of 2 children, nodes have 3 children. A 3-ary heap can be represented by an array as follows : The root is stored in the first location, a[0], nodes in the next level, from left to right is stored from left to right is stored from a[1] to a[3]. The nodes from the second level of the tree from left to right are stored from a[4] location onward. An item x can be inserted into a 3-ary heap containing n items by

placing x in the location a[n] and pushing it up the tree to satisfy the heap property.

Q.13 Which one of the following is a valid sequence of elements in an array representing 3-ary max heap?

a) 1, 3, 5, 6, 8, 9 b) 9, 6, 3, 1, 8, 5
c) 9, 3, 6, 8, 5, 1 d) 9, 5, 6, 8, 3, 1

[GATE 2006]

Q.14 Suppose the elements 7, 2, 10 and 4 are inserted, in that order, into the valid3-ary max heap found in the above question 60. Which one of the following is the sequence of items in the array representing the resultant heap?

a) 10, 7, 9, 8, 3, 1, 5, 2, 6, 4
b) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
c) 10, 9, 4, 3, 7, 6, 8, 2, 1, 3
d) 10, 8, 6, 9, 7, 2, 3, 4, 1, 5

[GATE -2006]

Q.15 In a binary max heap containing n numbers, the smallest element can be found in time

a) O(n) b) O(log n)
c) O(log log n) d) O(1)

[GATE-2006]

Q.16 The maximum number of binary trees that can be formed with three unlabelled nodes is

a) 1 b) 5
c) 4 d) 3

[GATE-2007]

Q.17 The height of a binary tree is the maximum number of edges in any root to leaf path. The maximum number of nodes in a binary tree of height h is

a) $2^h - 1$ b) $2^{h-1} - 1$
c) $2^{h+1} - 1$ d) 2^{h+1}

[GATE-2007]

Q.18 Consider the process of inserting an element into a Max Heap, where the Max Heap is represented by an array. Suppose we perform a binary search on the path from the new leaf to the root to find the position for the newly inserted element, the number of comparisons performed is

a) $\Theta(\log_2 n)$ b) $\Theta(\log_2 \log_2 n)$
c) $\Theta(n)$ d) $\Theta(n \log_2 n)$

[GATE-2007]

Q.19 A complete n-ary tree is a tree in which each node has n children or no children. Let be the number of internal nodes and L be the number of leaves in a complete n-array tree. If L = 41, and l = 10, what is the value of n?

a) 3 b) 4
c) 5 d) 6

[GATE-2007]

Q.20 Consider the following C program segment where Cell Node represents a node in a binary tree:

```
Struct Cell Node{
        Struct CeII Node *leftChild;
        int element;
        struct Cell Node *rightChild;
}
intGetValue (structCeIINode *ptr
int value = 0;
if (ptr != NULL)
if ((ptr->leftChild = = NULL) &&
(ptr ->rightChild == NULL)){
        value = 1;
        else
        value = value + GetValue (ptr
        ->leftChild)
        + Getvalue (ptr ->rightChiId);
        }
return (value);
}
```

The value returned by GetValue when a pointer to the root of a binary tree is passed as its argument is

a) the number of nodes in the tree
b) the number of internal nodes in the tree

c) the number of leaf nodes in the tree
d) the height of the tree

[GATE 2007]

Q.21 The in-order and pre-order traversal of a binary tree are d b e a f c g and a b d e c f g, respectively the post order traversal of the binary tree is

a) d e b f g c a
b) e d b g f c a
c) e d b f g c a
d) d e f g b c a

[GATE 2007]

Q.22 You are given the postorder traversal P, of a binary search tree on the n elements 1, 2, ..., n. You have to determine the unique binary search tree that has P as its postorder traversal. What is the time complexity of the most efficient algorithm for doing this?

a) $\Theta(\log n)$
b) $\Theta(n)$
c) $\Theta(n \log n)$
d) None of the above, as the tree cannot be uniquely determined

[GATE-2008]

Statements for Linked Answer Questions Q.23, 24 and 25

Consider a binary max-heap implemented using an array

Q.23 Which one of the following arrays represents a binary max-heap?

a) {25, 12, 16, 13, 10, 8, 14}
b) {25, 14, 13, 16, 10, 8, 12}
c) {25, 14, 16, 13, 10, 8, 12}
d) {25, 14, 12, 13, 10, 8, 16}

[GATE 2009]

Q.24 What is the content of the array after two delete operations on the correct answer to the previous question?

a) {14, 13, 12, 10, 8}
b) {14, 12, 13, 8, 10}
c) {14, 13, 8, 12, 10}
d) {14, 13, 12, 8, 10}

[GATE 2009]

Q.25 What is the maximum height of any AVL-tree with 7 nodes? Assume that the height of a tree with a single node is 0.

a) 2
b) 3
c) 4
d) 5

[GATE 2009]

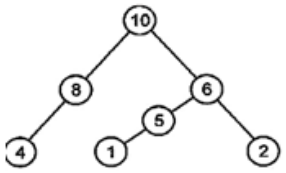
Q.26 We are given a set of n distinct elements and an unlabeled binary tree with n nodes. In how many ways can we populate the tree with the given set so that it becomes a binary search tree?

a) 0
b) 1
c) n!
d) $\frac{^{2n}C_n}{n+1}$

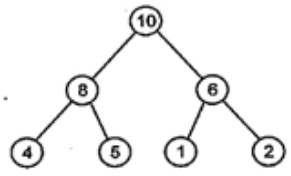
[GATE 2011]

Q.27 A max-heap is a heap where the value of each parent is greater than or equal to the value of its children. Which of the following is a max-heap?

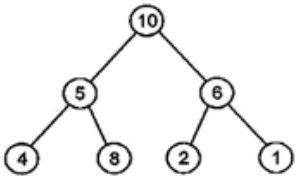
a)



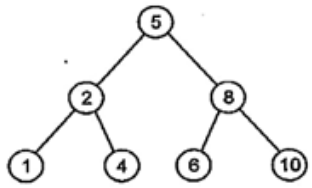
b)



c)



d)



Q.28 The height of a tree is defined as the number of edges on the longest path in the tree. The function shown in the pseudo code below in invoked as height(root) to compute the height of a binary tree rooted at the tree pointer root.

```
int height (treeptr in){
        if (n = = NULL)  return -1 ;
        If (n → left = = NULL)
        If  (n  →  right  =  =  NULL)
        return 0;
        Else [B1] return;//  Box 1
        Else {h1 = height  (n → left) ;
        If  (n → right = NULL) return
        (1 + h1) ;
        Else {h2 = height (n → right) ;
        [B2]
        Return;// Box 1
}
```

The appropriate expressions for the two boxes B1 and B2 are

a) B1: (1 + height (n → right) B2: (1 + max (h1 , h2))
b) B1: (height (n →right) B2: (1 + max(h1 , h2))
c) B1: height (n →right) B2: max (h1,h2)
d) B1: (1 + height (n → right) B2: max (h1 , h2)

[GATE-2012]

Q.29 The pre-order traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the post order traversal sequence of the same tree?

a) 10, 20, 15, 23, 25, 35, 42, 39, 30
b) 15, 10, 25, 23, 20, 42, 35, 39, 30
c) 15, 20, 10, .23, 25, 42, 35, 39, 30
d) 15, 10, 23. 25, 20, 35, 42, 39, 30

[GATE-2013]

Q.30 Consider the C function given below. Assume that the array ListA contains n (>0) elements, sorted in ascending order.

```
int processArray(int *listA, int x, int n)
 {
   int i, j, k;
   i = 0;  j = n-1;
   do {
          k = ( i + j ) / 2;
          if ( x <= listA[k])
                 j = k-1;
          if ( listA[k]  <= x)
                 i = k+1;
     } while (i<=j);

   if ( listA[k] == x)
          return k;
   else
          return -1;
}
```

Which of the following statements about the function process Array is CORRECT?

a) It will run into an infinite loop when x is not in list A
b) It is an implementation of binary search
c) It will always find the maximum element in list A
d) It will return -1 even when x is present in list A.

[GATE -2014]

Q.31 A priority queue is implemented as a Max-Heap. Initially, it has 5 elements. The level-order traversal of the heap is: 10, 8, 5, 3, 2. Two new elements 1 and 7 are inserted into the heap in that order. The level-order traversal of the heap after the insertion of the elements is:

a) 10, 8, 7, 3, 2, 1, 5
b) 10, 8, 7, 2, 3, 1, 5
c) 10, 8, 7, 1, 2, 3, 5
d) 10, 8, 7, 5, 3, 2, 1

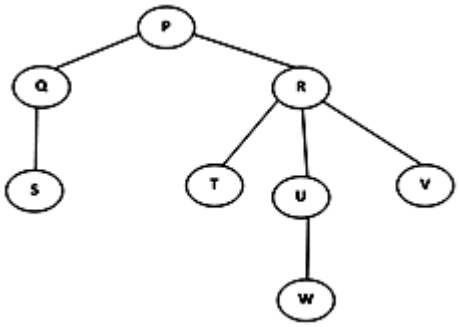
[GATE -2014]

Q.32 Consider a rooted Binary tree represented using pointers. The best

upper bound on the time required to determine the number of subtrees having having exactly 4 nodes $O(n^a \text{ Log } n^b)$. Then the value of a + 10b is _____

[GATE- 2014]

Q.33 Consider the following rooted tree with the vertex labeled P as root:



The order in which the nodes are visited during an in-order traversal of a tree is

a) SQPTRWUV b) SQPTUWRV
c) SQPTWUVR d) SQPTRUWV

[GATE -2014]

Q.34 Consider the pseudo code given below. The function DoSomething() takes as argument a pointer to the root of arbitrary tree represented by the leftmost child right sibling representation. Each node of the tree is of type treeNode.

```
typedef struct treeNode* treeptr;
struct treeNode
{ treeptr  leftmostchild, rightsibling;
};
int DoSomething ( treeptr tree)
{
  int value = 0;
  if (tree != NULL)
  {
       if  ( tree→leftmostchild == NULL)
                  Value = 1;
      else
                value = DoSomething ( tree→leftmostchild);
       value = value + DoSomething ( tree→rightsibling);
  }
  return (value);
}
```

When the pointer to the root of the tree is passed as argument to DoSomething, the value returned by the function corresponds to the

a) Number of internal nodes in the tree
b) Height of the tree
c) Number of nodes without a right sibling in the tree
d) Number of leaf nodes in the tree

[GATE -2014]

Q.35 The pre-order traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 42 which one of the following is the post-order traversal sequence of the same tree?

a) 10, 20, 15, 23, 25, 35, 42, 39, 30
b) 15, 10, 25, 23, 20, 42, 35, 39, 30
c) 15, 20, 10, 23, 25, 42, 35, 39, 30
d) 15, 10, 23, 25, 20, 35, 42, 39, 30

[GATE-2013]

Q.36 The height of a tree is the length of the longest root-to-leaf path in it. The maximum and minimum number of nodes in a binary tree of height 5 are

a) 63 and 6 b) 64 and 5
c) 32 and 6 d) 31 and 5

[GATE -2015]

Q.37 Which of the following is/are correct inorder traversal sequence(s) of binary search tree(s)?

1) 3, 5, 7, 8, 15, 19, 25
2) 5, 8, 9, 12, 10, 15, 25
3) 2, 7, 10, 8, 14, 16, 20
4) 4, 6, 7, 9, 18, 20, 25

a) 1 and 4 only b) 2 and 3 only
c) 2 and 4 only d) 2 only

[GATE -2015]

Q.38 Consider a max heap, represented by the array: 40, 30, 20, 10, 15, 16, 17, 8,

Array Index	1	2	3	4	5	6	7	8	9
Value	40	30	20	10	15	16	17	8	4

Now consider that a value 35 is inserted into this heap. After insertion, the new heap is

a) 40, 30, 20, 10, 15, 16, 17, 8, 4, 35
b) 40, 35, 20, 10, 30, 16, 17, 8, 4, 15
c) 40, 30, 20, 10, 35, 16, 17, 8, 4, 15
d) 40, 35, 20, 10, 15, 16, 17, 8, 4, 30

[GATE -2015]

Q.39 A binary tree T has 20 leaves. The number of nodes in T having two children is _____.

[GATE- 2015]

Q.40 Consider the following array of elements.
<89,19,50,17,12,15,2,5,7,11,6,9,100>
The minimum number of interchanges needed to convert it into a max-heap is

a) 4 b) 5
c) 2 d) 3

[GATE -2015]

Q.41 While inserting the elements 71,65, 84,69,67,83 in an empty binary search tree (BST) in the sequence shown, the element in the lowest level is

a) 65 b) 67
c) 69 d) 83

[GATE- 2015]

Q.42 The number of ways in which the numbers 1, 2, 3, 4, 5, 6, 7 can be inserted in an empty binary search tree, such that the resulting tree has height 6, is _____.
Note: The height of a tree with a single node is 0

[GATE- 2016]

Q.43 An operator delete(i) for a binary heap data structure is to be designed to delete the item in the i^{th} node. Assume that the heap is implemented in an array and i refers to the i-th index of the array. If the heap tree has depth d (number of edges on the path from the root to the farthest leaf), then what is the time complexity to re-fix the heap efficiently after the removal of the element?

a) O(1)
b) O(d) but not O(1)
c) $O(2^d)$ but not O(d)
d) $O(d2^d)$ but not $O(2^d)$

[GATE-2016]

Q.44 Let T be a binary search tree with 15 nodes. The minimum and maximum possible heights of T are:
Note: The height of a tree with a single node is 0.

a) 4 and 15respectively
b) 3 and 14 respectively
c) 4 and 14 respectively
d) 3 and 15 respectively

[GATE-2017]

Q.45 The pre- order traversal of a binary search tree is given tree by 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20. Then the post order traversal of this tree is:

a) 2,6,7,8,9,10,12,15,16,17,19,20
b) 2,7,6,10,9,8,15,17,20,19,16,12
c) 7,2,6,8,9,10,20,17,19,15,16,12
d)7,6,2,10.98.15.16.17.20.19.12

[GATE-2017]

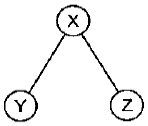
ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(c)	(d)	(c)	(a)	(b)	(d)	(b)	(c)	(b)	(d)	(a)	(a)	(d)	(a)	(a)
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
(b)	(c)	(b)	(c)	(c)	(a)	(b)	(c)	(d)	(b)	(b)	(b)	(a)	(d)	(b)
31	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**
(a)	1	(a)	(d)	(d)	(a)	(a)	(b)	19	(d)	(b)	64	(b)	(b)	(b)

EXPLANATIONS

Q. 1 **(c)**

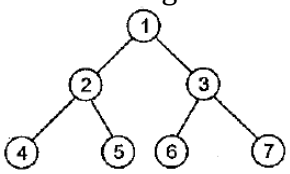
(XYZ) indicates that Y is left subtree and Z is right subtree Node is X



As per given in the question:

(1(234) (567))

We get, the following tree



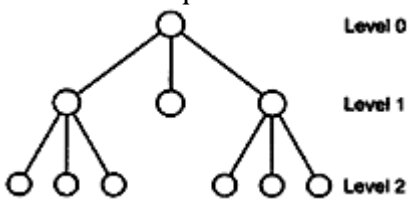
1 is the root node

2 and 3 are the non leaf node

4, 5, 6, 7 are the leaf node which may be null or further nested because in a binary tree every node has 0 or children and not just 1.

Q.2 **(d)**

Drawing a tree is more than sufficient to find the solutions to such kind of questions



At level 0 number of nodes =1

Total nodes =1

At level 1 leaf of nodes =3

Total nodes upto level 1=4

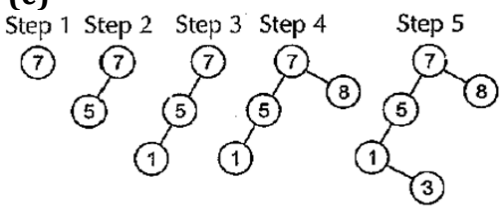
At level 2 leaf of nodes =7

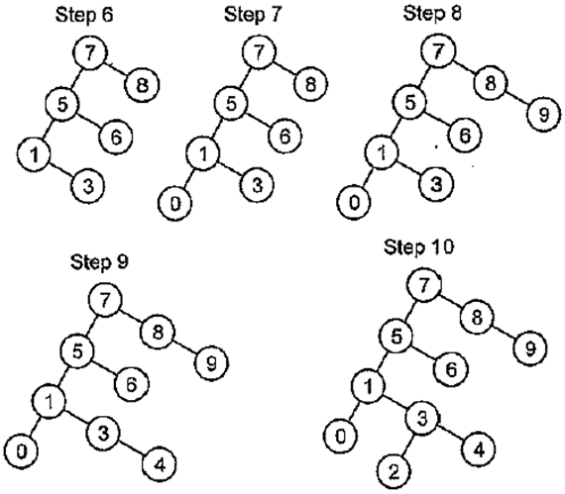
Total nodes =10

Therefore, total number of leaf nodes = (2n +1)/3 n=10

So, leaf nodes = (2 × 10 + 1)/3=7

Q.3 **(c)**





Therefore, the in order traversal came out to be 0 1 2 3 4 5 6 7 8 9.

Note: The in order traversal of a BST is in always sorted order.

Q.4 **(a)**

Let the number of nodes in the left subtree of the binary search tree be k

So,

T(0) = 1 '

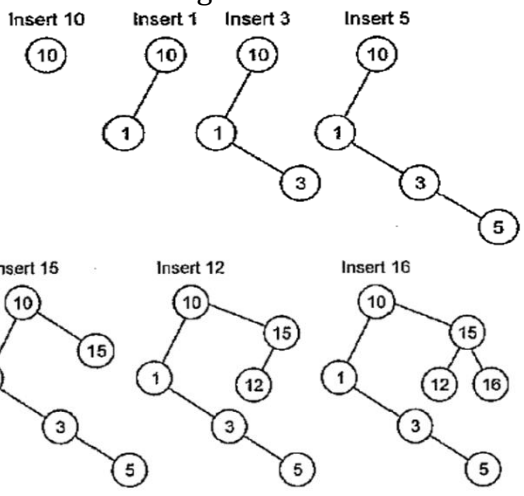
$$T(n)=\sum_{k=1}^{n} T(k-1)T(x),$$

= T(x) [T(0) + ...+ T(n - 1)]

Then, x = n - k + 1

Q.5 **(b)**

Binary tree required is obtained in the following manner



Thus the maximum height, i.e., the maximum distance of a leaf node from the root is 3.

Q.6 (d)

The total number of edges in left sub tree is calculated by the do Something function recursively as this function is are cursive function and the root node is included when 1 is added to the count stored in the variable value. Similar thing is done with right sub tree and finally max function in the program return the height of the tree as it calculates the total number of edges from root node to leaf node for either the left or right sub tree.

Q.7 (b)

We must know the following.
Either Preorder or Post order with the combination of inorder can identify the binary tree uniquely. But only preorder and post order combination cannot identify binary tree uniquely.

Q.8 (c)

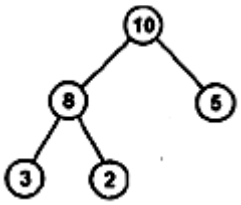
Let us consider a binary tree. In a binary tree :-
Number of leaves -1= Number of nodes
In k- ary tree where is each node has k children, this implies that k-1 key have no leaves.
Now, it is given that number of nodes are n, therefore number of leave will be n(k-1)+1.

Q.9 (b)

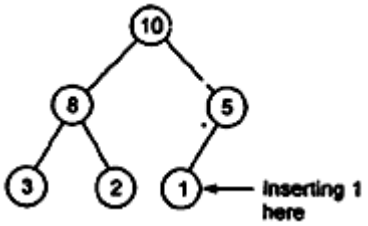
The number of keys as per given are 4
Applying the direct formula
$B_n = 1/(n+1)\times(2!/n!n!)$
Where B_n is number of binary trees and n is the number of tree
$\rightarrow B_n = 1/(4+1)\times(8!/4!4!)$
$\rightarrow B_n = 1/5\times(8\times7\times6\times5\times4!)/4!4!$
$\rightarrow B_n = 8\times7\times6/(4\times3\times2)$
$\rightarrow B_n = 56/4$
$\rightarrow B_n = 14$
The total number of binary tree with n = 4 is 14.

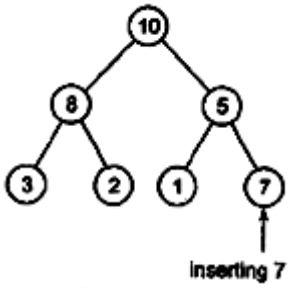
Q.10 (d)

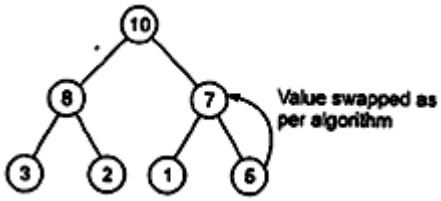
Initial level order traversal with 10, 8, 5, 3, 2



Now, let us insert the value







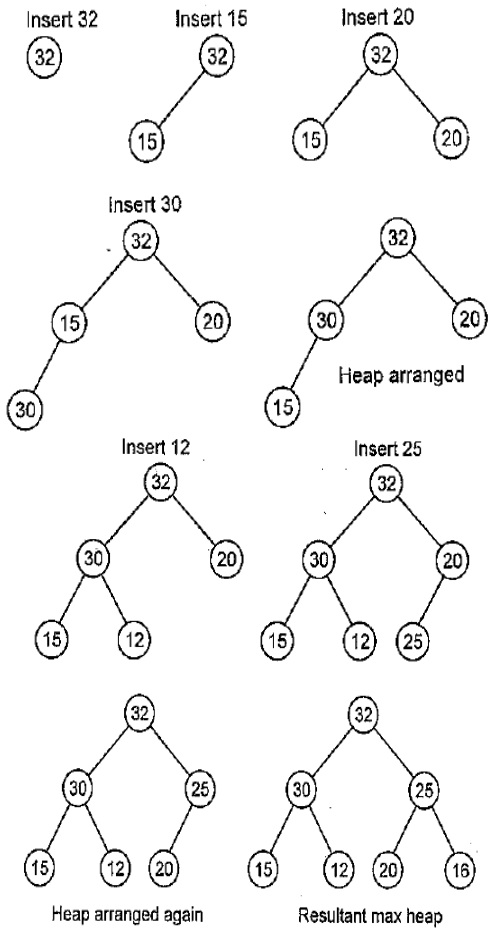
Therefore, the level order traversal comes out to be 10, 8, 7, 3, 2, 1, 5

Q.11 (a)

For representing the tree in an in order traversal, the tree is always drawn in a way such that the given order is in an increasing order thus, the tree consists of sequence 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95

Q.12 (a)

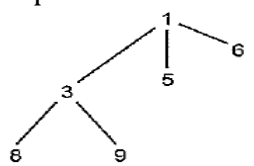
The insertion of elements takes place in the following manner and in this order



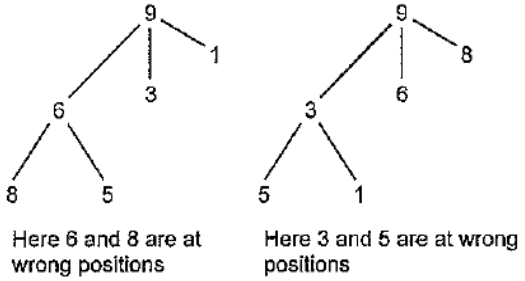
Q.13 (d)

The condition that needs to satisfy for maximum heap is that the root node is the largest element in tree as compared to all its children.

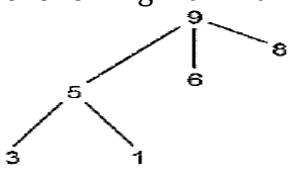
Thus, in option (a) the heap obtained is not the maximum heap as the root element is not the largest one the heap obtained is



Also the heap obtained from options (b) and (c) are not maximum heap



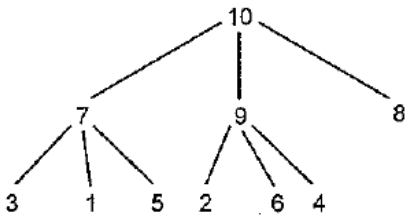
Thus, option (d) , is correct as it gives the following maximum heap.



Q.14 (a)

Whenever we insert an element into a heap such that the resultant heap is maximum one the element is inserted in the last position and then it is positioned to a correct place to obtain a maximum heap.

The heap obtained when the elements are inserted in the order 7, 2, 10 and 4, is as follows



Q.15 (a)

Time takes by binary max heap to identify max element is O(1). Therefore, the time taken by binary max heap to identify the smallest element is O(n).

Q.16 (b)

It is given that n=3

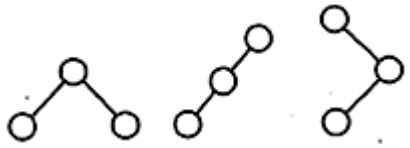
$$\frac{1}{n+1}\left({}^{2n}C_n\right)$$

$$\frac{1}{3+1}\left({}^{6}C_3\right)$$

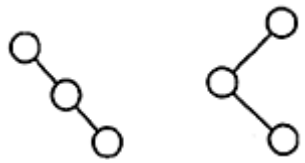
$$\frac{1.6!}{4\times 3!3!}=5$$

Therefore, there are 5 trees that can be formed with three unlabelled node.

1 **2** **3**



4 5



Q.17 (c)

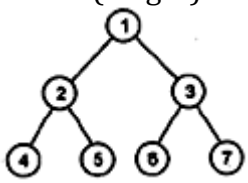
This is a formula to calculate the total no of nodes. It is $2^{h+1} - 1$.

Lets consider the same examples to prove this.

1. Simplest could be taking the binary tree of h(height) = 0.
   Now, the binary tree of the height h will have only 1 node.

   Using formula $2^{(0+1)}-1=1$. Hence, the formula is correct.
2. Binary tree of h (height) =2.

   Using formula 2^(2+1)-1= 7. Hence, the formula is correct.

Q.18 (b)

Max heap is a particular kind of binary tree with the following properties:

The value of each node is not less than the values stored in each of its children.

The tree is perfectly balanced, and the leaves in the last level are all in the leftmost positions.

The height of heap tree is logn. So there are logn elements from root to leaf path. The binary search on these logn elements take O(loglogn).

Q.19 (c)

Formula used: l(n-1)+1=1

→ 10(n-1)+1 =41

→ 10n-9 = 41

→ 10n = 50

→ n = 5

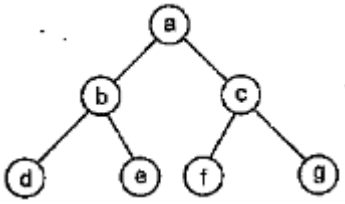
The node which is have 5 children.

Q.20 (c)

Since, the value is initialized to zero as depicted by the statement int value=0 also from the statement value =1 it is clear that the binary tree, if doesn't have. Any left child or right child and have only the root 'node then the number of leaf node in the, tree is 1 this is what was depicted in the, 'if' part of the loop. Now, the else part tells that, if the tree contains left and right child then, the number of leaf nodes are found recursively and the value returned is the number of leaf nodes in the tree.

Q.21 (a)

The in order traversal sequence is dbeafcg and the preorder traversal sequence is abdecfg so, the tree is



Q.22 (b)

The traversal of binary search tree will be of the sequence 1,2,3,....n since it's a post order traversal of a binary search tree. Therefore, by using the post order, unique binary tree can be constructed in O(n) time.

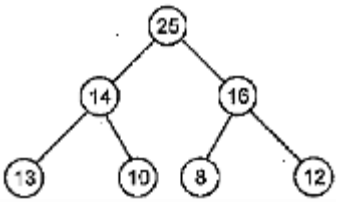
Q.23 (c)

Suppose that we have a node x, then for the condition

1<= x <= n/2 and A[x] >= A [2x+1]

where 2x and 2x+1 are left child and right child of the node x respectively of a binary max-heap.

Thus, under given cases the tree obtained is:



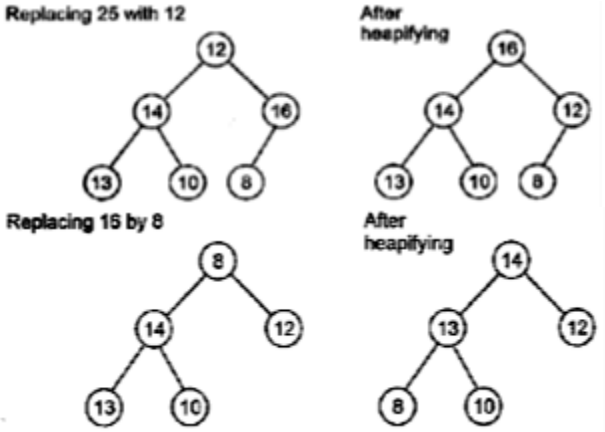
Q.24 (d)

Always a greater element is deleted from the binary heap first. The answer of previous question gave the array as [25, 14, 16, 13, 10, 12, 8]

So, when the greatest element, i.e., 25 is deleted the array becomes [14, 16, 13, 10, 12, 8]

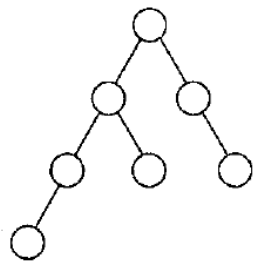
And next after second deletion the array becomes [14, 13, 10, 12, and 8]

Thus, the procedure for obtaining the final tree is as follows.



Q.25 (b)

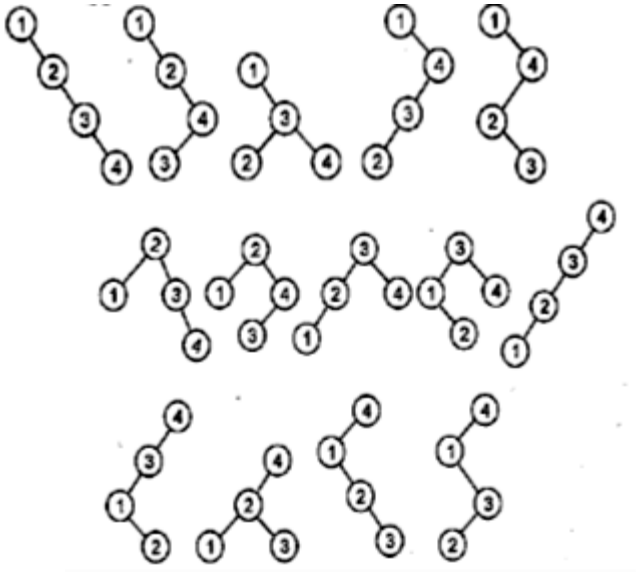
There are various ways of drawing an AVL tree. One of them is



Thus, according to this the maximum height of the tree is 3.

Q.26 (b)

For 4 elements 1, 2, 3, 4 the binary search tree will be



$$\Rightarrow 15 \Rightarrow \frac{1}{n+1}\,{}^{2n}C_n$$

This is only when only nodes are given.

But in question, the unlabelled binary tree is given, so we can put the numbers only in 1 way so that it becomes binary search tree.

Q.27 (b)

As heap is a complete binary tree with parent dominance property.

Q.28 (a)

Height of a tree is obtained by calculating 1+ height (right sub tree) and 1+ height (left sub tree) and picking the max among these two.

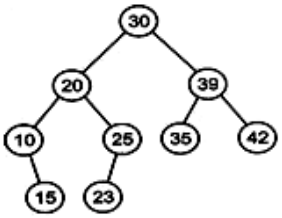
Q.29 (d)

Preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42.

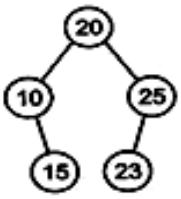
To compute post order traversal sequence of the same tree.

The Preorder traversal sequence is formed as follows First root, then left subtree, then right subtree and then recursively repeating it.

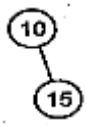
30 is the root and all elements < 30 will be in the left subtree and all > 30 in right subtree, which is a, property of a Binary Search Tree.



Moving left subtree of tree rooted at 30

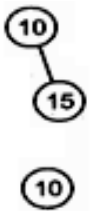


Moving to the left subtree of element.

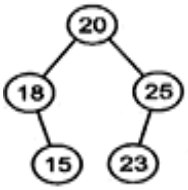


Left subtree is empty, so going to the right subtree of element (10)

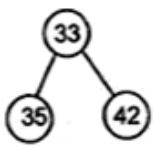
Left and Right subtrees are empty, so print root (15) going back from right subtree



Print root
Reversing from left subtree and going to right.



Similarly here (25) and@ would be printed and then moving to its right subtree.



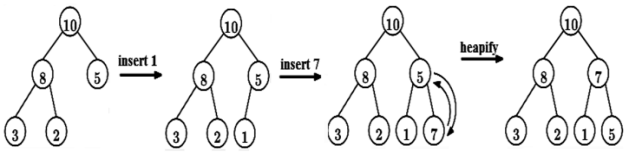
Here print order is (35), @, (39) and at last root 30.
The order is given by

(d) 15, 10, 23, 25, 20, 35, 42, 39, 30

Q.30 (b)

By the logic of the algorithm it is clear that it is an attempt to implement Binary Search. So option c is eliminated. Let us now check for options a and d. A good way to do this is to create small dummy examples (arrays) and implement the algorithm as it is. One may make any array of choice. Running iterations of the algorithm would indicate that the loop exits when the x is not present. So option A is wrong. Also, when x is present, the correct index is indeed returned. d is also wrong.
Correct answer is b. It is a correct implementation of Binary Search.

Q.31 (a)



Q.32 (1)

```
int print_subtrees_size_4(node *n)
{
        int size=0;
        if(node==null)
        return 0;
        size=print_subtrees_size_4(n
        ode>left)+print_subtrees_siz
        e_4(node->right)+1;
        if(size==4)
        printf("this is a subtree of
        size 4");
        return size;
}
```

The above function on taking input the root of a binary tree prints all the subtrees of size 4 in O(n) time
so a=1 , b=0 and then a+10*b=1

Q.33 (a)

The In-order Traversal of Ternary Tree is done as follows:

Left → Root →Middle →Right
So the nodes are visited in SQPTRWUV order

Q.34 (d)

Here, that condition is if (tree → leftMostchild == Null)

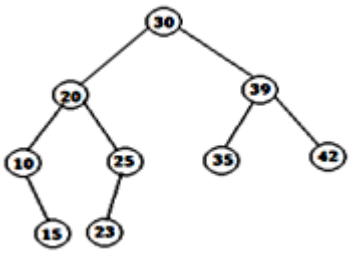
- Which means if there is no left most child of the tree (or the sub-tree or the current nodes called in recursion)
- Which means there is no child to that particular node (since if there is no left most child, there is no child at all).
- Which means the node under consideration is a leaf node.
- The function recursively counts, and adds to value, whenever a leaf node is encountered.
- The function returns the number of leaf nodes in the tree

It can also be solved by taking a sample tree and tracing the logic.

Q.35 (d)

In preorder traversal, the root lies at the front. Thus, 30 is the root of the binary search tree. All the elements greater than 30 lie in the right subtree. Repeat the above procedure to construct the complete binary search tree.

The complete binary search tree is shown in the following figure



Q.36 (a)

The maximum number of nodes in a binary tree of height h is $2^{h+1}-1$.
The minimum number of nodes in a binary tree of height h is h+1.

Q.37 (a)

In-order traversal of a BST is always ascending.

Q.38 (b)

Once 35 is inserted into the heap, two swaps are needed between 35, 15 and then 35, 30. The level order of the final tree is 40, 35, 20, 10, 30, 16, 17, 8, 4, 15.

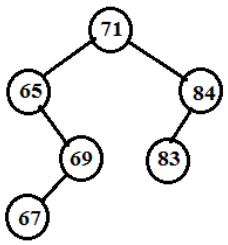
Q.39 (19)

The number of internal nodes = Number of leaf nodes – 1.

Q.40 (d)

The swaps needed are (100, 15), (100,50), (100,89).

Q.41 (b)



Q.42 (64)

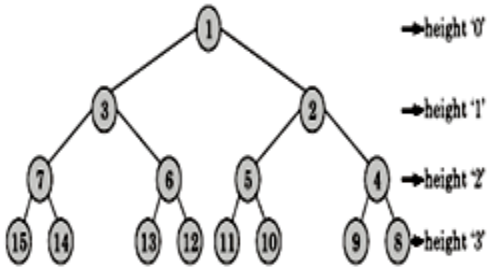
Formula is 2^n, here n is 6
$2^6 = 64$

Q.43 (b)

Deletion of element needs heapify process which depends on depth of tree i.e. So time complexity is O(d) and it cannot be O(1) because they are not constant number of operations. It depends on depth of the tree.

Q.44 (b)

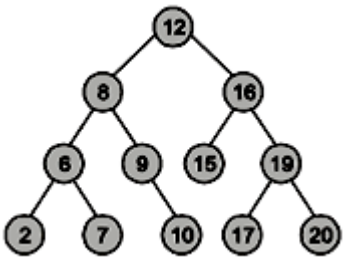
Since there are 15 nodes, hence the minimum height of the tree will be 3(when tree will be balanced).



The maximum height will be when the tree is skew tree, which will give rise to height 14.

Q.45 (b)

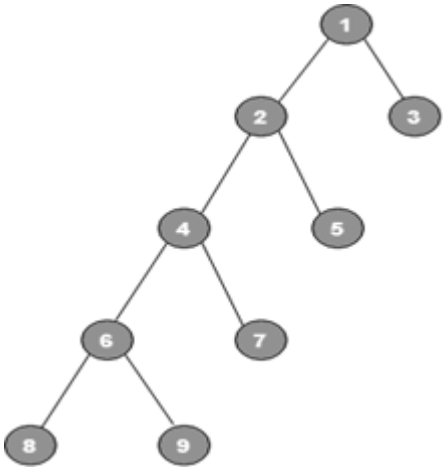
Pre order: 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 20
In order: 2, 6, 7, 8, 9, 10, 12, 15, 16, 17, 19, 20
Tree will be,



Post order will be
2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12

Q.46 (c)

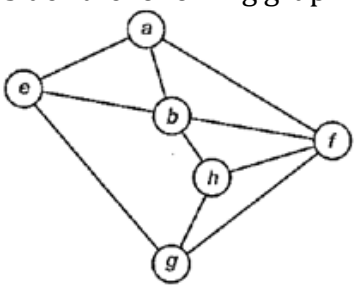
Post-order: 8, 9, 6, 7, 4, 5, 2, 3, 1
In-order: 8, 6, 9, 4, 7, 2, 5, 1, 3



The height of the binary tree above is 4.

GATE QUESTIONS(GRAPHS)

Q.1 Consider the following graph:



Among the following sequences
I. abeghf II. abfehg
III. abfhge IV. afghbe
Which are depth first traversals of the above graph?
a) I, II and IV b) I and IV
c) II, III and IV d) I, III and IV

[GATE-2003]

Q.2 Let G = (V, E) be a directed graph with n vertices. A path from vi to vj in G is a sequence of vertices (v_i , v_{i+1} , …. v_j) such that (v_k , v_{k+1}) ∈ E for all k in i through j – 1. A Simple path is a path in which no vertex appears more than once. Let A be an n × n array initialized as follows

$$A[i, j] = \begin{cases} 1, & \text{if } (j, k) \in E \\ 0, & \text{otherwise} \end{cases}$$

Consider the following algorithm:

```
for i = 1 to n
  for j = 1 to n
    for k = 1 to n
A[j , k] = max (A[j , k], A[j , i] +
A[i , k]);
```

Which of the following statements is necessarily true for all j and k after termination of the above algorithm?

a) A[j , k] ≤ n
b) A[j , k] ≥ n – 1, then G has a Hamiltonian cycle
c) If there exists a path from j to k, A[j , k] contains the longest path length from j to k
d) If there exists a path from j to k, every simple path from j to k contains at most A[j , k] edges.

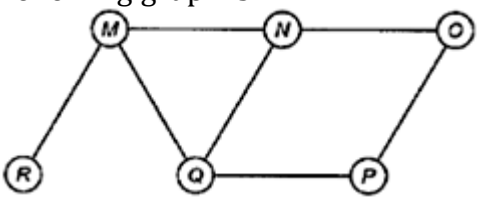
[GATE-2003]

Q.3 Let T be a depth first search tree in an undirected graph G. Vertices u and v are leaves of this tree T. The degrees of both u and v in G are at least 2. Which one of the following statements is true?

a) There must exist a vertex w adjacent to both u and v in G
b) There must exist a vertex w whose removal disconnects u and v in G
c) There must exist a cycle in G containing u and v
d) There must exist a cycle in G containing u and all its neighbours in G

[GATE-2006]

Q.4 The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is



a) MNOPQR b) NQMPOR
c) QMNPRO d) QMNPOR

[GATE-2008]

Q.5 The most efficient algorithm for finding the numbers of connected components is an undirected graph on n vertices and m edges has time complexity.

a) θ(n) b) θ(m)
c) θ(m + n) d) θ(mn)

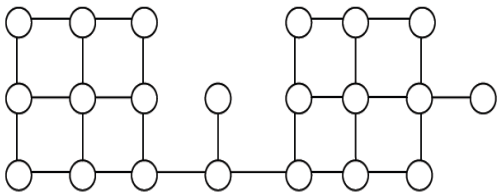
[GATE-2008]

Q.6 Let G be a graph with n vertices and m edges. What is the tightest upper bound on the running time of Depth first search on G when G is represented as an adjacency matrix?

a) $\theta(n)$ b) $\theta(n+m)$
c) $\theta(n^2)$ d) $\theta(m^2)$

[GATE-2014]

Q.7 Suppose depth first search is executed on the graph below starting at some unknown vertex. Assume that a recursive call to visit a vertex is made only after first checking that the vertex has not been visited earlier. Then the maximum possible recursion depth (including the initial call) is.....



a) 17 b) 18
c) 19 d) 20

[GATE-2014]

Q.8 Breadth First Search(BFS) is started on a binary tree beginning from the root vertex. There is a vertex t at a distance four from the root. If t is the n-th vertex in this BFS traversal, then the maximum possible value of n is____

[GATE-2016]

Q.9 G is an undirected graph with n vertices and 25 edge such that each vertex of G has degree at least 3. Then the maximum possible value of n is__

[GATE-2017]

Q.10 Let G= (VE) be any connected undirected edge- weighted graph. The weights of the edges in E are positive and distinct, consider the following statements:

I. Minimum Spanning Tree of G is always unique.
II. Shortest path between any two vertices of G is always unique.

Which of the above statements is/ are necessarily true?

a) I only b) II only
c) Both I nor II d) Neither I nor II

[GATE-2017]

Q.11 Let G be a simple undirected graph. Let T_D be a depth first search tree of G. Let T_B be a breadth first search tree of G. Consider the following statements.

(I) No edge of G is a cross edge with respect to T_D. (A cross edge in G is between two nodes neither of which is an ancestor of the other in T_D).

(II) For every edge (u, v) of G, if u is at depth i and v is at depth j in T_B, then $|i - j| = 1$.

Which of the statements above must necessarily be true?

a) I only
b) II only
c) Both I and II
d) Neither I nor II

[GATE-2018]

Q.12 Let G be a graph with 100! vertices, with each vertex labelled by a distinct permutation of the numbers 1, 2, ..., 100. There is an edge between vertices u and v if and only if the label of u can be obtained by swapping two adjacent numbers in the label of v. Let y denote the degree of a vertex in G, and z denote the number of connected components in G. Then y + 10z = _____.

a) 109
b) 110
c) 119
d) none of these

[GATE-2018]

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12
(d)	(d)	(b)	(c)	(c)	(c)	19	31	16	(a)	(a)	(a)

EXPLANATIONS

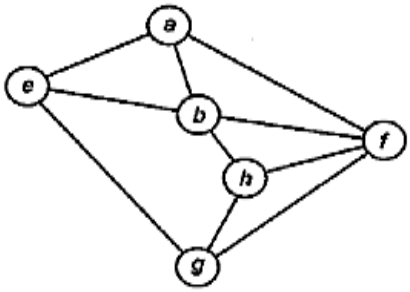
Q.1 **(d)**

In such type of questions, we need to check the options so as to save the time in the examination. However, this is all depends upon a practices and the regular way of solving questions.

Considering the diagram, lets check options:

Option l; abeghf

Search (a) = b add (b) a, b



I, III, IV are DFS traversals. But II is not because once we visit f, either we can visit h or g but not e.

Q. 2 **(d)**

With the given algorithm, we can say that if there exists a path from j to k, every simple path from j to k contains at most A [j, k] edges

Now since A [j, k] =1, if (j, k) ∈ E and A is n x n array and also the values of i , j, k, are never 0.

There is a path from j to k and the path will contain maximum of A (j, k) edges.

Q. 3 **(b)**

Since they are leaves in DFS, they must be separable by a vertex.

Q. 4 **(c)**

Option (a) is MNOPQR. It cannot be a BFS as the traversal starts with M, but O is visited before N and Q. In BFS all adjacent must be visited before adjacent of adjacent.

Option (b) is NQMPOR. It also cannot be BFS, because here, P is visited before O.

(c) and (d) match up to QMNP. We see that M was added to the queue before N and P (because M comes before NP in QMNP). Because R is M's neighbor, it gets added to the queue before the neighbor of N and P (which is O). Thus, R is visited before O.

Q.5 **(c)**

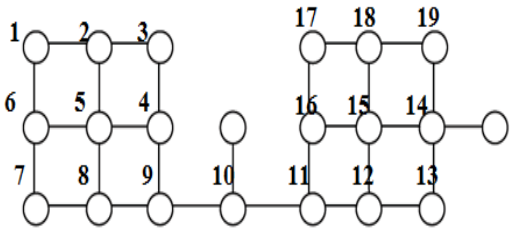
Assume n ≤ m.

Now, time complexity of most efficient algorithm for finding the number of connected components in an undirected graph on n vertical and m edge can be found using depth first search. The time complexity is Θ (m+n).

Q.6 **(c)**

DFS visits each vertex once and as it visits each vertex, we need to find all of its neighbors to figure out where to search next. Finding all its neighbors in an adjacency matrix requires O(V) time, so overall the running time will be $O(V^2)$.

Q.7 **(19)**



Q.8 **(31)**

Maximum possible value happens when we have complete tree and our t node is the last leaf node at

height 4. So, 1+2+4+8+16 = 31st is the t node.

Q.9 **(16)**

n ≤ ?

e =25

Now since each vertex has at least 3 degree

And 2e = Σdegree

i.e., 2e ≥3n

n ≤ 2e/3

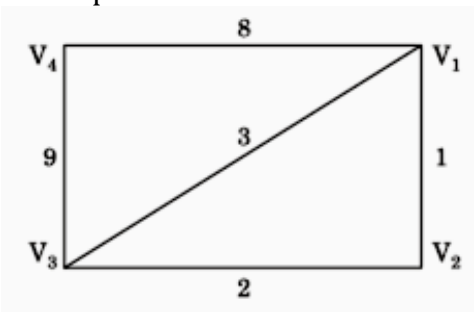
$$n \le \frac{2\times 25}{3} \le 16.66$$

so n is at most 16.

Q.10 **(a)**

- Since all the edge weights are unique, hence the minimum spanning tree of the graph will be unique.
- Shortest path between the two vertices need not to be unique. A counter example for the statement can be ,

The path from



The path from $V_1 \to V_3$ can be

I. $V_1 \to V_2 \to V_3 : 1+2=3$

II. $V_1 \to V_3 : 3$

Hence the path is not unique

Q.11 **(a)**

There are four types of edges that can yield in DFS. These are tree, forward, back, and cross edges. In undirected connected graph, forward and back egdes are the same thing. A cross edge in a graph is an edge that goes from a vertex v to another vertex u such that u is neither an ancestor nor descendant of v. Therefore, cross edge is not possible in undirected graph. So, statement (I) is correct.

For statement (II) take counterexample of complete graph of three vertices, i.e., K3 with XYZ, where X is source and Y and Z are in same level. Also, there is an edge between vertices Y and Z, i.e., |i-j| = 0 ≠ 1 in BFS. So, statement became false.

Option (A) is correct.

Q.12 **(a)**

There is an edge between vertices u and v iff the label of u can be obtained by swapping two adjacent numbers in the label of v.

Then the set of swapping numbers will be {(1, 2), (2, 3), ……..(9, 9)}

There will be 99 such sets, i.e. number of edges = 99

and each vertex will have 99 edges corresponding to it.

Say graph with 3! vertices, then vertices will be like {1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}…

Let's pick vertex {123}, degree will be 2 since it will be connected with two other vertices {213} and {132}.

We can conclude that for n, degree will be n-1.

So, degree of each vertex = 99 (as said y)

As the vertices are connected together, the number of connected components formed will be 1 (as said z).

y+10z = 99+10(1) = 109

GATE QUESTIONS(HASHING)

Q.1 Given the following input (4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199) and the hash function x mod 10. Which of the following statements are true?

1. 9679, 1989, 4199 hash to the same value.
2. 1471, 6171 hash to the same value.
3. All elements hash to the same value.
4. Each elements hashes to a different value.

a) 1 only b) 2 only
c) 1 and 2 d) 3 and 4

[GATE-2004]

Q.2 The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and linear probing. What is the resultant hash table?

a)

0	
1	
2	2
3	23
4	
5	15
6	
7	
8	18
9	

b)

0	
1	
2	12
3	13
4	
5	5
6	
7	
8	18
9	

c)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

d)

0	
1	
2	12,2
3	13,3,23
4	
5	5,15
6	
7	
8	18
9	

[GATE 2009]

Statements for Linked Answer Questions 3 and 4

A hash table of length 10 uses. Open addressing with hash function h (k) = k mod 10, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Q.3 Which one of the following choices gives a possible order in which the key values could have been inserted in the table?

a) 46, 42, 34, 52, 23, 33
b) 34, 42, 23, 52, 33, 46
c) 46, 34, 42, 23, 52, 33
d) 42, 46, 33, 23, 34, 52

[GATE 2010]

Q.4 How many different insertion sequences of the key values using the same hash function and linear probing will result in the hash table shown above?

a) 10 b) 20
c) 30 d) 40

[GATE 2010]

Q.5 Consider a hash table with 9 slots. The hash function is h(k) = k mod 9. The collisions are resolved by chaining. The following 9 keys are inserted in the order: 5, 28, 19, 15, 20, 33, 12, 17, 10. The maximum, minimum, and average chain lengths in the hash table, respectively, are

a) 3, 0, and 1 b) 3, 3, and 3
c) 4, 0, and 1 d) 3, 0, and 2

[GATE-2014]

Q.6 Consider a hash table with 100 slots. Collisions are resolved using chaining. Assuming simple uniform hashing, what is the probability that the first 3 slots are unfilled after the first 3 insertions?

a) $(97 \times 97 \times 97)/100^3$
b) $(99 \times 98 \times 97)/100^3$
c) $(97 \times 96 \times 95)/100^3$
d) $(97 \times 96 \times 95)/(3! \times 100^3)$

[GATE-2014]

Q.7 Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020?

a) $h(i) = i^2 \bmod 10$
b) $h(i) = i^3 \bmod 10$
c) $h(i) = (11*i^2) \bmod 10$
d) $h(i) = (12*i) \bmod 10$

[GATE-2015]

Q.8 Given a has table T with 25 slots that stores 2000 elements, the load factor α for T is___

[GATE-2015]

ANSWER KEY:

1	2	3	4	5	6	7	8
(c)	(c)	(c)	(c)	(a)	(a)	(b)	80

EXPLANATIONS

Q. 1 **(c)**

The solution can be achieved by finding the hash values of the input.

Index	Hash value
4322	2
1334	4
1471	1
9679	9
1989	9
6171	1
6173	3
4199	9

By the table above, it is observed that statement 1 and statement 2 are correct.

Q.2 **(c)**

12 mod 10 = 2
18 mod 10 = 8
13 mod 10 = 3
2 mod 10 = 2 collision
(2 + 1) mod 10 = 3 again collision (using linear probing)
(3 + 1) mod 10 = 4
3 mod 10 = 3 collision
(3 + 1) mod 10 = 4 again collision (using linear probing)
(4 + 1) mod 10 = 5
23 mod 10 = 3 collision
(3 + 1) mod 10 = 4 collision
(4 + 1) mod 10 = 5 again collision
(5 + 1) mod 10 = 6
5 mod 10 = 5 collision
(5 + 1) mod 10 = 6 again collision
(6 + 1) mod 10 = 7
15 mod 10 = 5 collision
(5 + 1) mod 10 = 6, collision
(6 + 1) mod 10 = 7 collision
(7 + 1) mod 10 = 8 collision
(8 + 1) mod 10 = 9
So, resulting hash table

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

Q.3 **(c)**

In this method, we simply checking the options by using linear probing so, option (a) 46, 42, 34, 52, 23, 33,

0	
1	
2	42
3	52
4	34
5	23
6	46
7	33
8	
9	

According to linear probing this is hash table

Option (b) 34, 42, 23, 52, 33, 46

0	
1	
2	42
3	23
4	34
5	52
6	33
7	46
8	
9	

Hash table

Option (c) 46, 34, 42, 23, 52, 33

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Hash table

Q.4 **(c)**

Different insertion sequences of key values using same hash function and linear probing.

46, 34, 42, 23, 52, 33

No problem to enter 46, 34, 42, 23

0	
1	
2	42
3	23
4	34
5	
6	46
7	
8	
9	

52 →

0	
1	
2	42
3	23
4	34
5	52
6	46
7	
8	
9	

To enter 52 there are 6 possible options

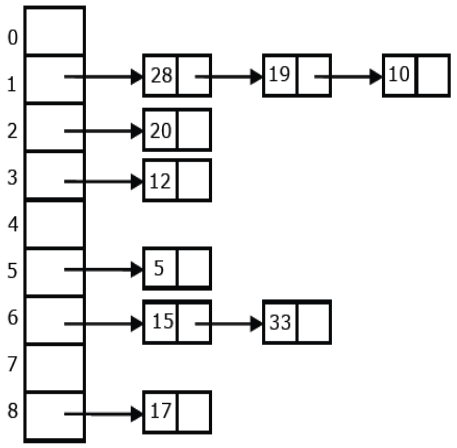
After entering 52, we gate hash table

Now, to enter 23 there are 5 possible options.

So, total =6× 5 = 30 possible options.

Q.5 **(a)**

The resulting hash table is



The maximum and minimum lengths of chains are 3, 0 respectively.

The average length of chain = (0+3+1+1+0+1+2+0+1) / 9 = 1

Q.6 **(a)**

Probability of 1st element occupying one of 97 slots = 97 / 100

Probability of 2nd element occupying one of 97 slots =97/100 [since collision resolution is chaining]

Probability of 3rd element occupying one of 97 slots= 97/100 [since collision resolution is chaining]

Q.7 **(b)**

Based on the table below,

i	i^3	i^3 mod 10
0	0	0
1	1	1
2	8	8
3	27	7
4	64	4
5	125	5
6	216	6
7	343	3
8	512	2
9	729	9

The numbers from 0 to 2020 are equally divided in 10 buckets. But it is not possible with squares. The buckets 2,3,7,8 would be empty.

Q.8 **(80)**

Load factor = (no. of elements)/ (no. of slots in table) = 2000 / 25 =80

GATE QUESTIONS (ALGORITHM ANALYSIS & ASYMPTOTIC NOTATION)

Q.1 Consider the following functions
$f(n) = 3n\sqrt{n}$
$g(n) = 2^{\sqrt{n}\log_2^n}$
$h(n) = n!$
Which of the following is true?
a) h(n) is O(f(n))
b) h(n) is O(g(n))
c) g(n) is not O(f(n))
d) f(n) is O(g(n))

[GATE-2000]

Q.2 Let s be a sorted array of n integers. Let t(n) denotes the time taken for the most efficient algorithm to determine if there are two elements with sum less than 1000 in s. Which of the following statements is true?
a) t(n) is O(1)
b) $n \leq t(n) \leq n\log_2 n$
c) $n\log_2 n \leq t(n) \leq \left(\frac{n}{2}\right)$
d) $t(n) = \left(\frac{n}{2}\right)$

[GATE-2000]

Q.3 Let $f(n) = n^2 \log n$ and $g(n) = n(\log n)^{10}$ be two positive functions of n. Which of following statements is correct?
a) f(n) = O(g(n)) and g(n) ≠ O(f(n))
b) g(n) = O(f(n)) and f(n) ≠ O(g(n))
c) f(n) ≠ O(g(n)) and g(n) ≠ O(f(n))
d) f(n) = O(g(n)) and g(n) = O(f(n))

[GATE-2001]

Q.4 The running time of the following algorithm
Procedure A (n)
If n ≤ 2 return (1) else return
$\left(A\left(\left\lceil\sqrt{n}\right\rceil\right)\right)$;
is best described by
a) O(n) b) O(log n)
c) O(log log n) d) O(1)

[GATE-2002]

Q.5 Consider the following algorithm for searching for a given number x in an unsorted array A[l...n] having n distinct values
1. Choose an i uniformly at random from 1..n
2. If A[i] = x then stop else Goto 1;
Assuming that x is present A, what is the expected number of comparisons made by the algorithm before it terminates?
a) n b) n - 1
c) 2n d) $\frac{n}{2}$

[GATE-2002]

Q.6 In a heap with n elements with the smallest element at the root, the 7th smallest element can be found in time
a) θ(n log n) b) θ(n)
c) θ(log n) d) θ(1)

[GATE-2003]

Q.7 The usual $\theta(n^2)$ implementation of Insertion Sort to sort an array uses Linear search to identify the position where an element is to be inserted into the already sorted part of the array. If, instead, we use binary search to identify the position, the worst case running time will
a) remain $\theta(n^2)$
b) become $\theta(n(\log n)^2)$
c) become θ(n log n)
d) become θ(n)

[GATE-2003]

Q.8 Consider the following three claims:
I. $(n + k)^m = \theta(n^m)$, where k and m are constants
II. $2^{n+1} = O(2^n)$
III. $2^{2n+1} = O(2^n)$
Which of these claims are correct?

a) I and II b) I and III
c) II and III d) I, II and III

[GATE-2003]

Q.9 The recurrence equation
$T(1) = 1$
$T(n) = 2T(n - 1) + n , n \geq 2$
Evaluates to
a) $2^{n+1} - n - 2$ b) $2^n - n$
c) $2^{n+1} - 2n - 2$ d) $2^n + 2$

[GATE-2004]

Q.10 The time complexity of the following C function is (assume n > 0)

```
int recursive (int n)
if (n == 1)
        return (1);
else
return (recursive (n - 1) + recursive (n – 1);
}
```

a) O(n) b) O(n log n)
c) $O(n^2)$ d) $O(2^n)$

[GATE-2004]

Q.11 Let A [1,…,n] be an array storing a bit (1 or 0) at each location, and f(m) is a function whose time complexity is θ(m). Consider the following program fragment written in a C like language:

```
counter = 0
for (i=1; i<n; i++)
{
        if(A[i]==1)counter ++;
        else {
        f(counter);
        counter = 0;
        }
}
```

The complexity of this program fragment is
a) $\Omega(n^2.)$
b) $\Omega(n\log n)\text{and}O(n^2)$
c) $\theta(n)$
d) $O(n)$

[GATE-2004]

Q.12 The time complexity of computing the transitive closure of a binary relation on a set of n elements is known to be
a) O(n) b) O(n log n)
c) $O(n^{3/2})$ d) $O(n^3)$

[GATE-2005]

Statements for Linked Answer Question 13 and 14

Consider the following C function:

```
Double foo (int n) {
        int i;
        double sum;
        If (n==0) return 1.0;
        else {
                sum =0.0;
                for (i=0; i<n; i++)
                        sum +=foo(i);
                        return sum;
        }
}
```

Q.13 The space complexity of the above function is
a) O(1) b) O(n)
c) O(n!) d) $O(n^3)$

[GATE-2005]

Q.14 Suppose we modify the above function foo () and store the values of foo (i), 0≤i<n, as and when they are computed. With this modification, the time complexity for function foo () is significantly reduced. The space complexity of the modified function would be
a) O(1) b) O(n)
c) $O(n^2)$ d) O(n!)

[GATE-2005]

Q.15 Suppose

$$T(n) = 2T\left(\frac{n}{2}\right) + n,\ T(0) = T(1) = 1$$

Which one of the following is false?
a) $T(n) = O(n^2)$
b) $T(n) = \theta(n\log n)$
c) $T(n) = \Omega(n^2)$

d) $T(n) = O(n \log n)$

[GATE-2005]

Q.16 Give two arrays of numbers $a_1,....,a_n$ and $b_1,....,b_n$ where each number is 0 or 1, the fastest algorithm to find the largest span (i, j) such that $a_i + a_{i+1} + ... + a_j = b_i + b_{i+1} + .. + b_j$, or report that there is not such span,

a) Takes $O(3^n)$ and $\Omega(2^n)$ time if hashing is permitted

b) Takes $O(n^3)$ and $\Omega(n^{2.5})$ time in the key comparison model

c) Takes $\theta(n)$ time and space

d) Takes $O(\sqrt{n})$ time only if the sum of the 2n elements is an even number

[GATE-2006]

Q.17 Consider the following recurrence:
$T(n) = 2T([\sqrt{n}])+1, T(1) = 1$
Which one of the following is true?
a) $T(n) = \theta(\log \log n)$
b) $T(n) = \theta(\log n)$
c) $T(n) = \theta(\sqrt{n})$
d) $T(n) = \theta(n)$

[GATE-2006]

Q.18 A set x can be represented by an array x[n] as follows X[i] = $\begin{cases} 1, \text{if } i \in x \\ 0, \text{otherwise} \end{cases}$

Consider the following algorithm in which x, y and z are Boolean arrays of size n :

```
algorithm zzz (x [ ], y [ ], z [ ]) {
int i;
for (i=0; i<n; ++i)
z[i] = (x[i] ∧ ~y[i]) ∨ (~x[i] ∧ y[i])
}
```

The set Z computed by the algorithm is

a) $(X \cup Y)$ b) $(X \cap Y)$

c) $(X-Y) \cap (Y-X)$ d) $(X-Y) \cup (Y-X)$

[GATE-2006]

Q.19 An element in an array X is called a leader, if it is greater than all elements to the right of it in X. The best algorithm to find all leaders in an array

a) Solves it in linear time using a left to right pass of the array

b) Solves it in linear time using a right to left pass of the array

c) Solves it using divide and conquer in time $\theta(n \log n)$

d) Solves it in time $\theta(n^2)$

[GATE-2006]

Q.20 Consider the following C program fragment in which i , j and n are integer variables.

```
for( i = n , j = 0; i > 0; i /= 2, j += i);
```

Let val (j) denotes the value stored in the variable j after termination of the for loop. Which one of the following is true?

a) val (j)=$\theta(\log n)$ b) val (j)=$\theta(\sqrt{n})$

c) val (j)=$\theta(n)$ d) val (j)=$\theta(n \log n)$

[GATE-2006]

Q.21 Which one of the following in place sorting algorithms needs the minimum number of swaps?

a) Quick sort b) Insertion sort

c) Selection sort d) Heap sort

[GATE-2006]

Q.22 Consider the polynomial $p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$, where $a_i \neq 0, \forall$ i. The minimum number of multiplication needed to evaluate p on the input x is

a) 3 b) 4

c) 6 d) 9

[GATE-2006]

Q.23 Consider the following C code segment:

```
int IsPrime (n)
{
int i, n;
        for (i=2; i<=sqrt (n); i++)
        if (n% i == 0)
{
```

```
printf("Not Prime\n"); return 0;
}
        return 1;
}
```

Let T(n) denotes the number of times the for loop is executed by the program on input n. Which of the following is true?

a) $T(n) = O(\sqrt{n})$ and $T(n) = \Omega(\sqrt{n})$
b) $T(n) = O(\sqrt{n})$ and $T(n) = \Omega(1)$
c) $T(n) = O(n)$ and $T(n) = \Omega(\sqrt{n})$
d) None of these

[GATE-2007]

Q.24 What is the time complexity of the following recursive function?

```
int DoSomething (int n) {
it (n <= 2)
return 1;
else
return (DoSomething (floor
(sqrt (n))) + n);
 }
```

a) $\Theta(n^2)$ b) $\Theta(n \log_2 n)$
c) $\Theta(\log_2 n)$ d) $\Theta(\log_2 \log_2 n)$

[GATE-2007]

Q.25 In the following C function let n ≥ m

```
int gcd (n, m)
        {
        if (n % m == 0) return m;
        n = n % m;
        return gcd (m, n);
        }
```

How many recursive calls are made by this function?

a) $\Theta(\log_2 n)$ b) $\Omega(n)$
c) $\Theta(\log_2 \log_2 n)$ d) $\Theta(\sqrt{n})$

[GATE-2007]

Common Data for Questions 26 and 27

Consider the following C functions:

```
int f1 (int n)
{
        if (n == 0 || n ==1)
         return n;
        else
        return (2*f1(n-1) + 3*f1(n-
                2));
}
int f2 (int n)
{
int i;
int X[N], Y[N], Z[N];
X[0] = Y[0] = Z[0] = 0;
X[1] = 1; Y[1] =2; Z[1] =3;
for (i =2; i<=n; i++){
        X[i] = Y[i-1] + Z[i-2];
        Y[i] = 2*X[i];
        Z[i] = 3*X[i];
}
return X[n];
}
```

Q.26 The running time of f1 (n) and f2 (n) are

a) $\Theta(n)$ and $\Theta(n)$
b) $\Theta(2^n)$ and $\Theta(n)$
c) $\Theta(n)$ and $\Theta(2^n)$
d) $\Theta(2^n)$ and $\Theta(2^n)$

[GATE-2008]

Q.27 f1 (8) and f2 (8) return the values

a) 1661 and 1640
b) 59 and 59
c) 1640 and 1640
d)1640 and1661

[GATE-2008]

Q.28 We have a binary heap on n elements and wish to insert n more elements (not necessarily one after another) into this heap. The total time required for this is

a) $\Theta(\log n)$ b) $\Theta(n)$
c) $\Theta(n \log n)$ d) $\Theta(n^2)$

[GATE-2008]

Q.29 The minimum number of comparisons required to determine if an integer appears more than n/2 times in a sorted array of n integers is

a) θ(n) b) θ(log n)
c) θ(log*n) d) θ(1)

[GATE-2008]

Q.30 Consider the following functions:
$f(n) = 2^n$
$g(n) = n!$
$h(n) = n^{\log n}$
Which of the following statements about the asymptotic behaviour of f(n), g(n), and h(n) is true?
a) $f(n) = O(g(n)); g(n) = O(h(n))$
b) $f(n) = \Omega(g(n)); g(n) = O(h(n))$
c) $g(n) = O(f(n)); h(n) = O(f(n))$
d) $h(n) = O(f(n)); g(n) = \Omega(f(n))$
[GATE-2008]

Q.31 The running time of an algorithm is represented by the following recurrence relation

$$T(n) = \begin{cases} n, n \le 3 \\ T\left(\frac{n}{3}\right) + cn, \text{otherwise} \end{cases}$$

a) $\theta(n)$ b) $\theta(n \log n)$
c) $\theta(n^2)$ d) $\theta(n^2 \log n)$
[GATE-2009]

Q.32 Two alternative packages A and B are available for processing a database having 10k records. Packages A requires $0.0001n^2$ time units and package B requires $10n\log_{10} n$ time units to process n records. What is the smallest value of k for which package B will be preferred over A?
a) 12 b) 10
c) 6 d) 5
[GATE-2010]

Q.33 Let W(n) and A(n) denote respectively , the worst case and average case running time of an algorithm executed on an input size n. Which of the following is always TRUE?
a) $A(n) = \Omega(W(n))$ b) $A(n) = \Theta(W(n))$
c) $A(n) = O(W(n))$ d) $A(n) = o(W(n))$
[GATE-2012]

Q.34 Consider the following functions

```
int unknown (int n){
int i , n , k =0;
for ( i = n/2 , i <= n; i++)
for (j = 2; j <= n; j = j * 2)
k = k +n/2;
return (k);
}
```

The return value of the function is
a) $\theta(n^2)$ b) $\theta(n^2 \log n)$
c) $\theta(n^3)$ d) $\theta(n^3 \log n)$
[GATE-2013]

Q.35 Which one of the following correctly determines the solution of recurrence relation with T(1) = 1?
$T(n) = 2\,T(n/2) + \log n$
a) $\theta(n)$ b) $\theta(n \log n)$
c) $\theta(n^2)$ d) $\theta(\log n)$
[GATE-2014]

Q.36 The number of arithmetic operations required to evaluate the polynomial $P(X) = X^5 + 4X^3 + 6X + 5$ for a given value of X using only one temporary variable.
a) 6 b) 7
c) 8 d) 9
[GATE-2014]

Q.37 Let a_n represent the number of bit strings of length n containing two consecutive 1s. What is the recurrence relation for a_n?
a) $a_{n-2} + a_{n-1} + 2^{n-2}$
b) $a_{n-2} + 2a_{n-1} + 2^{n-2}$
c) $2a_{n-2} + a_{n-1} + 2^{n-2}$
d) $2a_{n-2} + 2a_{n-1} + 2^{n-2}$
[GATE-2015]

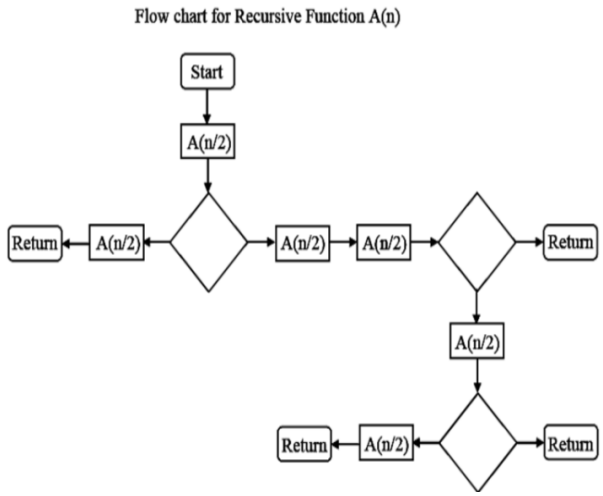
Q.38 Consider the equality $\sum i^3 = X$, for i=0 to n and the following choices for X
I) $\theta(n^4)$ II) $\theta(n^5)$
III) $O(n^5)$ IV) $\Omega(n^3)$
The equality above remains correct if X is replaced by
a) only I
b) Only II
c) I or III or IV but not II
d) II or III or IV but not I
[GATE-2015]

Q.39 Let f(n)=n and g(n)= $n^{(1+\sin n)}$, where n is a positive integer. Which of the following statement is/are correct?
I. f(n) = O(g(n))
II. f (n)= Ω(g(n))
a) Only I b) Only II
c) Both I and II d) Neither I nor II
[GATE-2015]

Q.40 A list contains n distinct elements. The number of comparisons to find an element in this list that is neither maximum nor minimum is
a) θ(n log n) b) θ(n)
c) θ(log n) d) θ(1)
[GATE-2015]

Q.41 Consider a carry look ahead adder for adding two n-bit integers, built using gates of fan-in at most two. The time to perform addition using this adder is
a) θ(1) b) θ(logn)
c) θ(sqrt(n)) d) θ(n)
[GATE-2016]

Q.42 The given diagram shows the flowchart for a recursive function A(n). Assume that all statements, except for the recursive calls, have O(1) time complexity. If the worst case time complexity of this function is $O(n^{\alpha})$, then the least possible value(accurate up to two decimal positions) of α is_____.



[GATE-2016]

Q.43 In an adjacency list representation of an undirected simple graph G = (V, E), each edge (u, v) has two adjacency list entries: [v] in the adjacency list of u, and [u] in the adjacency list of v. These are called twins of each other. A twin pointer is a pointer from an adjacency list entry to its twin. If |E| = m and |V| = n, and the memory size is not a constraint, what is the time complexity of the most efficient algorithm to set the twin pointer in each entry in each adjacency list?
a) $\theta(n^2)$ b) θ(n+m)
c) $\theta(m^2)$ d) $\theta(n^4)$
[GATE-2016]

Q.44 Consider the following C function:

```
float f (flat x, int y) {
        float p, s; int i;
        for (s=1, p=1,i=1; i<y; i++) {
                p*=x/i;
                s+=p;
        }
        return s;
}
```

For large values of y, the return value of the function f best approximates
a) x^y b) e^x
c) in (1+x) d) x^x
[GATE 2003]

Q.45 What does the following algorithm approximate?

```
x = m;
y = 1;
while (x – y >∈){
        x = (x+ y)/2;
        Y = m/x;
}
Print (x);
```

a) log m b) m^2
c) $m^{1/2}$ d) $m^{1/3}$
[GATE 2005]

Q.46 Consider the following segment of C-code

```
int j, n;
```

```
j = 1;
while (j <=n)
j= j*2;
```

The number of comparisons made in the execution of the loop for any n> 0 is

a) CEIL($\log_2$ n) +1
b) n
c) CEIL($\log_2$ n)
d) FLOOR($\log_2$ n)+ 1

[GATE 2007]

Q.47 Consider the following function.

```
int unknown  (int n){
int  i,  j,  k = 0;
for  (i = n / 2; 1 < = n; i++)
for  (j = 2; j < = n; j = j * 2)
k = k + n / 2;
return  (k) :
}
```

The return value of the function is

a) $\theta(n)^2$ b) $\theta(n^2 \log n)$
c) $\theta(n)^3$ d) $\theta(n^3 \log n)$

[GATE-2013]

Q.48 Suppose we have a balanced binary search tree T holding n numbers. We are given two numbers L and H and wish to sum up all the numbers in T that lie between L and H. Suppose there are m such numbers in T. If the tightest upper bound on the time to compute the sum is $O(n^a \log^b n + m^c \log^d n)$, the value of a+ 10b + 100c + 1000d is ___

[GATE 2014]

Q.49 An algorithm performs $(\log N)^{1/2}$ find operations, N insert operations, $(\log N)^{1/2}$ delete operations, and $(\log N)^{1/2}$ decrease-key operations on a set of data items with keys drawn from a linearly ordered set. For a delete operation, a pointer is provided to the record that must be deleted. For the decrease-key operation, a pointer is provided to the record that has its key decreased. Which one of the following data structures is the most suited for the algorithm to use, if the goal is to achieve the best total asymptotic complexity considering all the operations?

a) Unsorted array
b) Min-heap
c) Sorted array
d) Sorted doubly linked list

[GATE 2015]

Q.50 Consider the following C function.

```
int fun1 (int n)
{
int i, j, k, p, q = 0;
for (i = 1; i<n; ++i)
{
  p = 0;
 for (j=n; j>1; j=j/2)
      ++p;
 for (k=1; k<p; k=k*2)
      ++q;
}
return q;
}
```

Which one of the following most closely approximates the return value of the function fun1?

a) n^3 b) $n(\log n)^2$
c) nlogn d) nlog(logn)

[GATE 2015]

Q.51 Consider the following functions from positive integers to real numbers:

$$10, \sqrt{n}, n\log_2 n, \frac{100}{n}$$

Tire CORRECT arrangement of the above functions in increasing order of asymptotic complexity is:

a) $\log_2 n, \frac{100}{n}, 10, \sqrt{n}, n$

b) $\frac{100}{n}, 10\log_2 n, \sqrt{n}, n$

c) $10, \frac{100}{n}, \sqrt{n}, \log_2 n, n$

d) $\frac{100}{n}, \log_2 n, 10, \sqrt{n}, n$

[GATE -2017]

Q.52 Consider the recurrence function

$$T(n) = \begin{cases} 2T(\sqrt{n}) + 1, & n > 2 \\ 2 & 0 < n \le 2 \end{cases}$$

Then T(n) in terms of Θ notation is

a) $\Theta(\log\log n)$ b) $\Theta(\log n)$

c) $\Theta(\sqrt{n})$ d) $\Theta(n)$

[GATE -2017]

Q.53 Consider the following C function.

```
int fun {int n)
{
    int i.j.;
    for (i=1;i< = n;i++)
    {
       for (j=1;j<n;j+ = i)
       {
       Printf("%d", i , j);
       }
    }
 }
```

Time complexity of fun in terms of Θ notation is

a) $\Theta(n\sqrt{n})$ b) $\Theta(n^2)$

c) $\Theta(n\log n)$ d) $\Theta(n^2\log n)$

[GATE -2017]

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(d)	(a)	(b)	(c)	(a)	(c)	(a)	(a)	(a)	(d)	(c)	(d)	(b)	(b)	(c)
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
(c)	(b)	(d)	(b)	(c)	(c)	(a)	(b)	(d)	(a)	(b)	(c)	(b)	(b)	(d)
31	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**
(a)	(c)	(c)	(b)	(a)	(b)	(a)	(c)	(d)	(d)	(b)	2.32	(b)	(b)	(c)
46	**47**	**48**	**49**	**50**	**51**	**52**	**53**							
(c)	(b)	110	(a)	(d)	(b)	(b)	(c)							

EXPLANATIONS

Q. 1 **(d)**

$f(n) = 3n\sqrt{n}$

$g(n) = 2^{\sqrt{n}\log_2^n} = n\sqrt{n}$

So $f(n) = O(g(n))$ and $g(n) = O(f(n))$

Q. 2 **(a)**

Let array be sorted in ascending order, if sum of first two elements is less than 1000 then there are two elements with sum less than 1000 otherwise not. For array sorted in descending order we need to check last two elements. For an array data structure, number of operations are fixed in both the cases and not dependent on n, complexity is O(1)

Q. 3 **(b)**

$f(n)=n^2 \log n$

$g(n) =n(\log n)^{10}$

Any constant power of Log n is asymptotically smaller than n.

Q. 4 **(c)**

$T(n) = T(\sqrt{n}) + C1$ if $n > 2$

Let $n = 2^m$, $T(n) = T(2^m)$

Let $T(2^m) = S(m)$

From the above two, $T(n) = S(m)$

$S(m) = S(m/2) + C1$

$S(m) = O(\log m) = O(\log\log n)$

Now, let us go back to the original recursive function T(n)

$T(n) = S(m) = O(\text{LogLogn})$

Q. 5 **(a)**

Since the array is unsorted and since the condition-

if A [i] = x then stop else go to 1.

So, the comparisons will be made till the value of i reach n.

Therefore, before termination, expected numbers of comparisons made by algorithm is n.

Q. 6 **(d)**

The 7^{th} smallest element must be in first 7 levels. Total number of nodes in any Binary Heap in first 7 levels is at most 1 + 2 + 4 + 8 + 16 + 32 + 64 which is a constant. Therefore we can always find 7th smallest element in $\Theta(1)$ time.

Q. 7 **(a)**

If we use binary search then there will be $\lceil \log_2(n!) \rceil$ comparisons in the worst case, which is $\Theta(n \log n)$. But the algorithm as a whole will still have a running time of Θ (n^2) on average because of the series of swaps required for each insertion.

Q. 8 **(a)**

Statement 1 is correct

Consider k to be constant

$f(n) = (n+k)^m$

$\rightarrow f(n) = (1+n)^m$

$\rightarrow f(n) = O(n^m)$

Statement 2 is correct

$f(n) = 2^{n+1}$

$\rightarrow f(n) = 2^n.2$

$\rightarrow f(n) = O(2^n)$

Statement 3:

$f(n) = 2^{2n+1}$

$\rightarrow \quad f(n) = 2^{2n}.2$

$\rightarrow \quad f(n) > 2^n$

Therefore, we can see that the only statement 3 is false.

Q. 9 **(a)**

$T(n)=2T(n-1)+n,\ n \geqslant 2,\ T(1)=1$

$T(n)=n+2(n-1)+2^2(n-2)+\cdots+2^{(n-1)}(n-(n-1))$

$=n(1+2+\cdots+2^{n-1})-(1.2+2.2^2+3.2^3+\cdots+(n-1).2^{n-1})$

$=n(2^n-1)-(n.2^n-2^{n+1}+2)$

$=2^{n+1}-n-2$

Q. 10 (d)

$T(n)=2T(n-1)+a$ is the recurrence equation found from the pseudo code .

Solving the Recurrence Equation by Substitution Method

$T(n)=2T(n-1)+a$ -------- Equation 1
$T(n-1)=2T(n-2)+a$
$T(n-2)=2T(n-3)+a$
We can re write Equation 1 as

$T(n)=2[2T(n-2)+a]+a$
$= 4T(n-2)+3a$
$= 4[2T(n-3)+a]+3a$
$= 8T(n-3)+7a\ldots$
$= 2^kT(n-k)+(2k-1)a$ --- Equation 2

On Substituting Limiting Condition

$T(1)=1$ implies $n-k=1 \Longrightarrow k=n-1$
Therefore Equation 2 becomes

$= 2n-1+(2n-1-1)a$
$= O(2^n)$

Q. 11 (c)

The lines below
If (a[i] ==1) counter ++;
else {f (counter);counter =0;}
contains only one loop.
Now, as there is only one loop so it will be computed using linear complexity and we know that the complexity will be θ (n).

Q. 12 (d)

In computer science the concept of transitive closure can be thought of as constructing a data structure that makes it possible to answer reachability questions. That is, can one get from node a to other node b in one or more hops? A binary relation tells you only that node a is connected to node b, and that node b is connected to node c, etc. After the transitive closure is constructed in an O(1) operation one may determine that node c is reachable from node a.
warshall's algorithm can be used to construct the Transitive closure of directed graphs (). In warshall's original formulation of the algorithm, the graph is un-weighted and represented by a Boolean adjacency matrix. Then the addition operation is replaced by logical conjunction (AND) and the minimum operation by logical disjunction (OR). So, Applying Warshall theorem, we get the time complexity of computing transitive closure of a binary relation on a set of n elements. Warshall theorem, time complexity is O (n^3) which is achieved as there are three nested for loops with n frequency.

Q. 13 (b)

Note that the function foo() is recursive. Space complexity is O(n) as there can be at most O(n) active functions (function call frames) at a time.

Q. 14 (b)

As given, The value foo(i) is such that $0 \leq i < n$. As per the condition the longest size that can be stored is a string of n bits Therefore, the space complexity of the modified function becomes O (n).

Q. 15 (c)

$T(n) = 2T(n/2) + n$
$= 2(2T(n/4) + n/2) + n$
$= 4T(n/4) + n + n$
$= 4T(n/4) + 2n$
$= 4(2T(n/8) + n/4) + 2n$
$= 8T(n/8) + n + 2n = 8T(n/8) + 3n$
$= 8(2T(n/16) + n/8)+ 3n$
$= 8T(n/16)+ n + 3n = 16T(n/16) + 4n$
...
$= 32T(n/32) + 5n$ ...

= n*T(1) + log2(n)*n
= O(n*log$_2$(n))
= O(n^2) also.
So the false statement is $T(n)=\Omega(n^2)$

Q. 16 (c)

Given that each number is 0 or 1.
Now, to achieve
$a_1 + a_{i+1} + ... + a_i = b_i + b_{i+1} + ... + b_j$
We need to find the sum of LHS and RHS and then have to compare.
We need to do this from very starting like first we need to check.
Whether $a_1 = b_1$
If not then whether $a_1 + a_2 = b_1 + b_2$
And so on……
This will taken n comparisons.
Therefore, the fastest algorithm to find the largest span takes $\Theta(n)$ time and space.

Q. 17 (b)

This question can be solved by first change of variable and then Master Method.
Let n = 2^m
$T(2^m) = T(2^{(m/2)}) + 1$
Let $T(2^m) = S(m)$
$S(m) = 2S(m/2) + 1$
Above expression is a binary tree traversal recursion whose time complexity is θ(m). You can also prove using Master theorem.
S(m) = θ(m)
= θ(logn)
Now, let us go back to the original recursive function T(n)
$T(n) = T(2^m) = S(m)$
= θ(logn)

Q. 18 (d)

The condition given in the FOR loop is $z[i]=(x[i] \wedge \sim y[i]) \vee (\sim x[i] \wedge y[i])$
Now, we clearly can see that the condition reflect the XOR operation set obtained is$(X \cap Y') \cup (X' \cap Y)$
Since, the formula is $X \cap Y' = X' \cap Y$
Result obtained is (X - Y) ∪ (Y - X)

Q. 19 (b)

Let array x contains n elements.
Solving the question in linear time from right to left movement is possible. Maintain max element while moving from right to left and every element is compared only with max to decide whether it is leader.

Q. 20 (c)

The frequency of the for loop can given as
i = i/2
n= n/2
After termination of loop
val (j) = n + n/2+…..n/2 log$_2$ n.
= 2n (1-1/2 log$_2$ n)
= 2n (1-1/n)
= 2(n-1) = θ (n)

Q. 21 (c)

Out of the algorithms provided in the option, selection sort takes minimum number of swaps. Working of selection sort can be seen as−First find the smallest in the array and exchange it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continue in this way until the entire array is sorted.

Q. 22 (a)

The minimum number of multiplication can be found out by simplifying the given expression
$p(x) = a_0 + x(a_1 + a_2x + a_3x^2)$
$p(x) = a_0 + x(a_1 + (a_3x + a_2)x)$
Here we can see three multiplications
1. $a_3 * x$
2. $x (a_2 + a_3x)$
3. $x* (a_1 + x(a_2 + a_3x))$

Q. 23 (b)

Best case occurs when the given number is even, So $T(n) = \Omega(1)$ (Best case)

Worst case occurs when the given number is square of a prime number.

$T(n) = O(\sqrt{n})$ (worst case)

Q. 24 (d)

$T(n) = T(\sqrt{n}) + C1$ if $n > 2$

Let $n = 2^m$, $T(n) = T(2^m)$

Let $T(2^m) = S(m)$

From the above two, $T(n) = S(m)$

$S(m) = S(m/2) + C1$

$S(m) = O(\log_2 m)$

$= O(\log_2 \log_2 n)$

Now, let us go back to the original recursive function T(n)

$T(n) = S(m) = O(\text{Log}_2 \text{Log}_2 n)$

Q. 25 (a)

In algorithm you will see that the remainder is 'cut' into half in every 2 steps. And since it cannot go less than 1, there can be at most $2.[\log_2 n]$ steps/recursions. Each step/recursion requires constant time, $\theta(1)$ so this can be at most $2.[\log_2 n].\theta(1)$ time and that's $\theta(\log_2 n)$

Q. 26 (b)

As we can observe f(1) is a recursive function, the recurrence equation can be further observed as

$T(n) = 2T(n-1) + 3T(n-2)$

The solution to the equation is 2^n.

Also, f2 has a loop from 2 to N. Therefore, the average running time comes out to be $\Theta(2^n)$ and $\Theta(n)$.

Q. 27 (c)

For f(1), we are given that

(2*f1(n-1)+3*f1(n-2))

For f(2), we are given that

X[0] = Y[0] = Z[0] = 0

X[1] = 1; Y[1] = 2; Z[1] = 3

And for loop we are given that:

X[i] = Y[i-1] + Z[i-2];

Y[i] = 2*X[i];

Z[i] = 3*X[i];

Calculating f(1),

f(n) = (2* f1(n-1) + 3 * f1(n-2))

f(2) = (2* f1(2-1) + 3 * f1(2-2))

f(2) = (2 * 1+3*0) = 2

Similarly,

f(3) = 7

f(4) = 20

f(5) = 61

f(6) = 182

f(7) = 547

f(8) = 1640

Therefore, f1(8) returns 1640.

f2(8) will also return the same value as f2(n) is the non-recursive function of f1(n).

Q. 28 (b)

Following is algorithm for building a Heap of an input array A.

```
BUILD-HEAP(A)
 heapsize := size(A);
 for i := floor(heapsize/2) downto 1
     do HEAPIFY(A, i);
   end for
END
```

Although the worst case complexity looks like O(nLogn), upper bound of time complexity is O(n)

Q. 29 (b)

Let's consider an array

A[1]
A[2]
-
-
-
A[n]

Below is an array in which there is an element i. This element i appears more than n/2 time in array.

Generally, in a binary search, expected case comparison is not greater than Log (n+1) and if the array is sorted then it takes $\Theta(1)$ times.

But, given to us is the following:

1. The array is already sorted.
2. Integer appears more than n/2 times.

Therefore, the total number of comparisons cannot be greater than $\Theta(\log n)$.

Q. 30 (d)

Since,
$f(n) = 2^n$
$g(n) = n!$
$h(n) = n^{\log n}$
it can also be shown as
$f(n) = O(2^n)$
$g(n) = O(n^!)$
$h(n) = O(n^{\log n})$
Now,
As per the asymptotic order of function $n \log n \le C\, 2^n$ for all $n \ge n_0$
Let us assume $C = 1$ and $n_0 = 2$
It comes out to be $2 \log \le 2^2$
$h(n) = O(f(n)) \rightarrow$ one result
Now,
$g(n) = n!$ & $f(n) = 2^n$
As per the asymptotic order of function
$n! \ge C2^n$ for all $n \ge n_0$
$24 > 2^4 \ge 16$
$g(n) = \Omega(f(n)) \rightarrow$ second result
Therefore, the two results are
$h(n) = O(f(n)); g(n) = \Omega(f(n))$

Q. 31 (a)

$T(n) = cn + T(n/3)$
$= cn + cn/3 + T(n/9)$
$= cn + cn/3 + cn/9 + T(n/27)$
Taking the sum of infinite GP series. The value of T(n) will be less than this sum.
$T(n) <= cn(1/(1-1/3))$
$<= 3cn/2$
or we can say
$cn <= T(n) <= 3cn/2$
Therefore $T(n) = \theta(n)$
This can also be solved using Master Theorem for solving recurrences.

Q. 32 (c)

Since, $10n\log 10n \le 0.0001n2$
Given $n = 10^k$ records. Therefore,
$=10\times(10^k)\log_{10}10^k \le 0.0001(10^k)^2$
$=10^{k+1}k \le 0.0001 \times 10^{2k}$
$=k \le 10^{2k-k-1-4}$
$=k \le 10^{k-5}$
Hence, value 5 does not satisfy but value 6 satisfies.
6 is the smallest value of k for which package B will be preferred over A.

Q. 33 (c)

The average case time complexity is always less than or equal to the worst case time complexity, Thus, $A(n) \le W(n)$ or $A(n) = O(W(n))$.
For example, for merge sort $A(n) = O(n\log n)$, $W(n) = O(n\log n)$.
For quick sort, $A(n) = O(n\log n)$ and $W(n) = O(n^2)$

Q. 34 (b)

The outer for loop executes $n - (n/2) + 1 = (n/2) + 1$ times. The inner for loop executes $\log_2 n$ times. In the body of inner for loop (n/2) is added to the value of k during each execution. The initial value of

$$K = \left(\frac{n}{2}+1\right)\left(\log_2 n.\frac{n}{2}\right)$$

$$= \left(\frac{n^2}{4}\log n + \frac{n}{2}\log n\right) = \theta(n^2 \log n)$$

Q.35 (a)

By Master's theorem case (i) T(n) is O(n)

Q.36 (b)

By using Horner's rule, expression can be written as $5+x(6+x^2(5+x^2)$
So there are 4 multiplications and 3 additions needed to evaluate the polynomial.

Q.37 (a)

Give value for n and check.

a_0= 0; a_1 =0; a_2= 1 [“11”]

a_3=3 [“011”,”110”,”111”]

a_4=8

[“0011”,”0110”,”0111”,”1101”,”1011”,”1100”,”1110”,”1111”]

If we check for a3, we can see that only (a) and (c) satisfy the value. Among (a) and (c), only (a) satisfies for a_4

Q.38 (c)

Sum of cubes of first n natural numbers is $n^2(n+1)^2/4 = \theta(n^4)$

So it clearly stays I, III, IV are true but not II.

Q.39 (d)

The value of sine function varies from -1 to 1.

For sin = -1 or any other negative value, I becomes false.

For sin = 1 or any other negative value, II becomes false

Q.40 (d)

We only need to consider any 3 elements and compare them. So the number of comparisons is constant. So the time complexity is θ (1).

Let us take an array {10, 20, 15, 7, 90}. Output can be 10 or 15 or 20

Pick any three elements. Let the three elements be 10, 20 and 7. Using 3 comparisons, we can find that the middle element is 10.

Q.41 (b)

Look ahead carry generator gives output in constant time if fan-in = number of inputs.

For Example:

It will take O(1) to calculate

c4 = g3 + p3g2 + p3p2g1 + p3p2p1g0 + p3p2p1p0c0c4

= g3 + p3g2 + p3p2g1 + p3p2p1g0 + p3p2p1p0c0,

if OR gate with 5 inputs is present.

And, if fan-in != number of inputs then we will have delay in each level, as given below.

If we have 8 inputs, and OR gate with 2 inputs, to build an OR gate with 8 inputs, we will need 4 gates in level-1, 2 in level-2 and 1 in level-3. Hence 3 gate delays, for each level.

Similarly an n-input gate constructed with 2-input gates, total delay will be O(log n).

Q.42 (a)

The time complexity of a recurrence relation is the worst case time complexity. First we have to find out the number of function calls in worst case from the flow chart. The worst case will be when all the conditions (diamond boxes) turn out towards non-returning paths taking the longest root. In the longest path we have 5 function calls. So the recurrence relation will be–

A(n) = 5A(n/2) + O(1)

Solving this recurrence relation using Master theorem –

a = 5 , b= 2 , f(n) =O(1) , $n\log_b a = n\log_2 5$ (case 1 of master theorem)

$A(n)=n\log_b a$

value of $\log_2 5$ is 2.3219, so, the best option is option a.

Q.43 (b)

Twin Pointers can be setup by keeping track of parent node in BFS or DFS of graph.

Q. 44 (b)

We will consider an iteration method since, the given function ‘/’ doesn't undergo recursion thus, we have the following:

i	p	s
	p = p*x/i	s = s + p
initialize 1	1	1
1 p = x		s = x +1
2 p = x. x/2		s = 1 + x + x ^ 2/2
3 p = x ^ 2/2.x/3		s = 1 + x + x ^ 2/2 + x ^ 3/6
4 p = x ^ 3/6.x/4		s = 1 + x + x ^ 2/2 + x ^ 3/6 + x^3/6
5		p = x^4/24.x/5
		s = 1 + x + x^2/2+x^ 3/6+x^4/24+x^5/120

When, y tends to larger values like infinity then *i* also tends to infinity. Hence, for loop instrumentation may also lead to infinity. The value of y; is taken as large integer but not infinite for the given function thus, the value of f is $ and is obtained as follows = $1+ x + x^2/2 + x^3/6 + x^4/24 + x^5/120$ ..... $+\infty$

$s= 1 + x + x^2/2! + x^3/3! + x^4/4! + x^5/5!$ ..... $+\infty$

Therefore, $s = e^x$

Q. 45 (c)

Let us suppose that m = 2

1st loop

x – y = 2

x = 3/2 = 1.5

y = 2/1.5 = 1.33

2nd loop

x – y = .16

x = 1.415

y = 1.413

And so on

And the loop will stop when x – y = 0

Thus, this program calculates square root.

Q. 46 (c)

From the statement *j* = j*2 in the code we get to know that j increases in power of 2's. Lets say that this statement executes x times then, according to the question for while loop $2^x \le n$

Therefore, $x \le \log_2 n$

And also for termination of while loop there will be an extra comparison required. Thus, total number of comparisons = x + 1

= CEIL ($\log_2 n$) + 1

Q. 47 (b)

The return value of the function is $\theta(n^2 \log n)$

The inner for loop for $\frac{n}{2}+1$ iterations.

The inner for loop runs independent of outer loop

And for each inner, $\frac{n}{2}$ gets added to k.

$\therefore \frac{n}{2} \times$ # outer loops $\times$ # inner loops per outer loop

#inner loops = $\theta(\log n)\left[\because 2^{\theta(\log n)} = \theta(n)\right]$

$\therefore \frac{n}{2} \times \left\lfloor \frac{n}{2}+1 \right\rfloor . \theta(\log n) = \theta(n^2 \log n)$

Q.48 (110)

It takes (log n) time to determine numbers n1 and n2 in balanced binary search tree T such that

1. n^1 is the smallest number greater than or equal to L and there is no predecessor n'1 of n1 such that n'1 is equal to n1.
2. n^2 is the largest number less than or equal to H and there is no successor of n'2 of n2 such that is equal to n2.

   Since there are m elements between n1 and n2, it takes 'm' time to add all elements between n1 and n2.

So time complexity is O (log n + m)

So the given expression becomes O ($n^0 \log^1 n + m^1 \log^0 n$)

a = 0, b = 1, c = 1 and d =0

So a+ 10b+ 100c+1000d = 0 + 10*1 + 100*1 + 1000*1 = 10 + 100 = 110

Q.49 (a)

The time complexity of insert in unsorted array is O(1), min-heap is O(logn), sorted array is O(n). Since number of insertion operations are asymptotically higher, unsorted array is preferred.

Q.50 (d)

```
for (i = 1; i<n; ++i)  // runs for n times
{
p = 0;
For (j=n; j>1; j=j/2) // runs for logn times
++p;
For (k=1; k<p; k=k*2) // runs for loglogn times
++q;
}
```

Since the value returned by the function is q, that approximates n*log (log n)

Q.51 (b)

$\frac{100}{n} < 10\log_2 n < \sqrt{n} < n$ so correct answer is (b)

Q.52 (b)

$T(n) = 2T(\sqrt{n}) + 1$ ... (1)

$T(n) = 2T(\sqrt[2]{n}) + 1$ ...(2)

Substituting (2) in (1)

$T(n) = 2.2T(\sqrt[2]{n}) + 2$

$T(n) = 2^2 T(\sqrt[2]{n}) + 2$ ...(3)

$T(\sqrt[2]{n}) = 2T(\sqrt[3]{n}) + 2$ ... (4)

Substituting (4) in (3)

$T(n) = 2^3 T(\sqrt[3]{n}) + 2$

Running the same till K times

$T(n) = 2^K T(\sqrt[K]{n}) + K$

$\sqrt[K]{n} = 2$

$K = \log_2 n$

Solving this will give $T(n) = \Theta(\log n)$

Q.53 (c)

First loop will execute 'n' times and the inner loop will execute Θ (log n) times, hence the complexity will $\Theta(n \log n)$

GATE QUESTIONS (DIVIDE & CONQUER)

Q.1 Randomized quick sort is an extension of quick sort where the pivot is chosen randomly. What is the worst case complexity of sorting n numbers using randomized quick sort?

a) O(n) b) O(n log n)
c) $O(n^2)$ d) O(n!)

[GATE-2001]

Q.2 Suppose there are $\lfloor \log n \rfloor$ sorted lists of $\lfloor n / \log n \rfloor$ elements each. The time complexity of producing a sorted list of all these elements is **(Hint:** Use a heap data structure**)**

a) O(log log n) b) $\theta(n \log n)$
c) $\Omega(n \log n)$ d) $\Omega(n^{3/2})$

[GATE-2005]

Q.3 The median of n elements can be found in O(n) time. Which one of the following is correct about the complexity of quick sort, in which median is selected as pivot?

a) θ(n) b) θ(n log n)
c) $\theta(n^2)$ d) $\theta(n^3)$

[GATE-2006]

Q.4 Which of the following sorting algorithms has the lowest worst-case complexity?

a) Merge sort b) Bubble sort
c) Quick sort d) Selection sort

[GATE-2007]

Q.5 An array of n numbers is given, where n is even number. The maximum as well as the minimum of these n numbers needs to be determined. Which of the following is true about the number of comparisons needed?

a) At least 2n-c comparisons, for some constant c, are needed
b) At most 1.5n-2 comparisons are needed
c) At least $n \log_2 n$ comparisons are needed
d) None of the above

[GATE-2007]

Q.6 Consider the Quick sort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sub-lists each of which contains at least one-fifth of the elements. Let T(n) be the number of comparisons required to sort n elements. Then

a) $T(n) \le 2T(n/5) + n$
b) $T(n) \le T(n/5) + T(4n/5) + n$
c) $T(n) \le 2T(4n/5) + n$
d) $T(n) \le 2T(n/2) + n$

[GATE-2008]

Q.7 What is the number of swaps required to sort n elements using selection sort, in the worst case?

a) θ(n) b) θ(n log n)
c) $\theta(n^2)$ d) $\theta(n^2 \log n)$

[GATE-2009]

Q.8 In quick sort, for sorting n elements, the (n/4)th smallest elements is selected as pivot using an O(n) time algorithm. What is the worst case time complexity of the quick sort?

a) θ(n) b) θ(n log n)
c) $\theta(n^2)$ d) $\theta(n^2 \log n)$

[GATE-2009]

Q.9 A list of n strings, each of length n, is sorted into lexicographic order using the merge-sort algorithm. The worst case running time of this computation is

a) O(nlog n) b) $O(n^2 \log n)$
c) $O(n^2 + \log n)$ d) $O(n^2)$

[GATE-2012]

Q.10 Let P be a Quick Sort Program to sort numbers in ascending order using the first element as pivot. Let t1 and t2 be the number of comparisons made by P for the inputs {1, 2, 3, 4, 5} and {4, 1, 5, 3, 2} respectively. Which one of the following holds?

a) t1 = 5 b) t1 < t2
c) t1 > t2 d) t1 = t2

[GATE-2014]

Q.11 The minimum number of comparisons required to find minimum and maximum of 100 numbers is ----

[GATE-2014]

Q.12 Suppose P, Q, R, S, T are sorted sequences having lengths 20, 24, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequences at a time. The number of comparisons that will be needed in the worst case by the optimal algorithm for doing this is _____

[GATE-2014]

Q.13 You have an array of n elements. Suppose you implement quick sort by always choosing the central element of the array as the pivot. Then the tightest upper bound for the worst case performance is

a) $O(n^2)$ b) $O(n \log n)$
c) $\theta(n \log n)$ d) $O(n^3)$

[GATE-2014]

Q.14 Which one of the following is the recurrence equation for the worst case time complexity of the Quicksort algorithm for sorting n (≥ 2) numbers? In the recurrence equations given in the options below, c is a constant.

a) $T(n) = 2T(n/2) + cn$
b) $T(n) = T(n-1) + T(0) + cn$
c) $T(n) = 2T(n-2) + cn$
d) $T(n) = T(n/2) + cn$

[GATE-2015]

Q.15 Suppose you are provided with the following function declaration in the C programming language int partition (int a [ ], int n); The function treats the first element of a [ ] as a pivot, and rearranges the array so that all elements less than or equal to the pivot is in the left part of the array , and all elements greater than the pivot is in the right part. In addition, it moves the pivot so that the pivot is the last elements of the left part. The return value is the number of elements in the left part. The following partially given function in the C programming language is used to find the K^{th} smallest element in an array a [ ] of size n using the partition function We assume k<=n

```
int kth _smallest (int a[ ], int n, int k)
{
int left _ end = partition (a,n) ;
if (left _ end+1 == k)
        return a left _ end ;
if (left _ end +1 > k)
        return
kth_smallest(……..);
else
        return
kth_smallest(……..);
}
```

The missing argument lists are respectively

a) (a, left_end, k) and (a+left_end+1, n-left_end-1, k-left_end-1)
b) (a, left_end, k) and (a, n-left_end-1, k left_end-1)
c) (a+left_end+1,n-left_end-1,k left_end-1) and (a, left_end, k)
d) (a, n-left_end-1,k-left_end-1) and (a, left_end, k)

[GATE-2015]

Q.16 Assume that a merge sort algorithm in the worst case takes 30 second

for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes?

a) 256 b) 512
c) 1024 d) 2048

[GATE-2015]

Q.17 What is the number of swaps required to sort n elements using selection sort, in the worst case?

a) $\theta(n)$ b) $\theta(n \log n)$
c) $\theta(n^2)$ d) $\theta(n^2 \log n)$

[GATE-2015]

Q.18 The worst case running times of Insertion sort, Merge sort and Quick sort, respectively, are:

a) $\Theta(n\log n)$, $\Theta(n\log n)$, and $\Theta(n^2)$
b) $\Theta(n^2)$, $\Theta(n^2)$, and $\Theta(n\log n)$
c) $\Theta(n^2)$, $\Theta(n\log n)$, and $\Theta(n\log n)$
d) $\Theta(n^2)$, $\Theta(n\log n)$, and $\Theta(n^2)$

[GATE-2016]

Q.19 Assume that the algorithms considered here sort the input sequences in ascending order. If the input is already in ascending order, which of the following are TRUE?

I. Quicksort runs in $\theta(n^2)$ time
II. Bubble sort runs in $\theta(n^2)$ time
III. Merge sort runs in $\theta(n)$ time
IV. Insertion sort runs in $\theta(n)$ time

a) I and II only b) I and III only
c) II and IV only d) I and IV only

[GATE-2016]

Q.20 Four matrices M_1, M_2, M_3 and M_4 are of dimensions p×q, q×r, r×s and s×t respectively can be multiplied in several ways with different number of total scalar multiplications. For example, when multiplied as $(M_1 \times M_2) \times M_3) \times M_4$) the total number of scalar multiplications is pqr = rst = prt. When multiplied as $(((M_1 \times M_2) \times M_3) \times M_4)$ the total number of scalar multiplications is and t=80, then the minimum number of scalar multiplications needed is

a) 248000 b) 44000
c) 19000 d) 25000

[GATE -2011]

Q.21 Which one of the following is the tightest upper bound that represents the time? Complexity of inserting an object into a binary search tree of n nodes?

a) O (1) b) 0 (log n)
c) O (n) d) O (n log n)

[GATE-2013]

Q.22 What are the worst-case complexities of insertion and deletion of a key in a binary search tree?

a) $\theta(\log n)$ for both insertion and deletion
b) $\theta(n)$ for both insertion and deletion
c) $\theta(n)$ for insertion and $\theta(\log n)$ for deletion
d) $\theta(\log n)$ for insertion and $\theta(n)$ for deletion

[GATE -2015]

Q.23 Match the algorithms with their complexities:

List-I (Algorithm)

P) Towers of Hanoi with n disks
Q) Binary search given n sorted numbers
R) Heap sort given n numbers at the worst case
S) Addition of two n×n matrices

List- II (Time complexity)

i) $\Theta(n^2)$ ii) $\Theta(n\log n)$
iii) $\Theta(2^n)$ iv) $\Theta(\log n)$

a) P-(iii), Q-(iv), R- (i), S-(ii)
b) P-(iv), Q-(iii) , R-(i), S(ii)
c) P-(iii), Q-(iv), R-(ii), S-(i)
d) P-(iv), Q- (iii), R –(ii) S-(i)

[GATE -2017]

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(b)	(a)	(b)	(a)	(b)	(b)	(a)	(b)	(b)	(c)	148	358	(a)	(b)	(a)
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**							
(b)	(a)	(d)	(d)	(c)	(c)	(b)	(c)							

EXPLANATIONS

Q. 1 **(b)**

Randomized quick sort can be explained as; to avoid getting slammed with a worst-case input, pivots are picked randomly from the set of possible pivots

The algorithm can be given as

QUICKSORTRAND (A, i , j)
1. if i ≤ j
2. then return
3. l← random number between i and j
4. k ←PARTITON (A, i ,j, l)
5. QUICKSORTRAND (A, i, k-1)
6. QUICKSORTRAND (A, k +1, j)
7. return

Worst case complexity of the Quick sort is $O(n^2)$ but when it come to randomized quick sort the worst case complexity changes to O(n log n)

Q. 2 **(a)**

As per the hint given along the question use heap data structure.

It is also given that

There are [log n] sorted lists of $\lfloor n/\log n \rfloor$ element's each

We know that construction of heap takes O(log log n)time

Therefore in worst case, the time complexity is O(n log log n)

(as element are $\lfloor n/\log n \rfloor$)

Q. 3 **(b)**

If we use median as a pivot element, then the recurrence for all cases becomes T(n) = 2T(n/2) + O(n)

The above recurrence can be solved using case 2 of Master Method.

Q. 4 **(a)**

Lets the inputs be n. Now, consider the table below defining the order of t sorting algorithm worst case to be case

Sorting Algorithm	Worst case complexity
Merge sort	O(n log n)
Bubble sort	$O(n^2)$
Quick sort	$O(n^2)$
Selection sort	$O(n^2)$

Q. 5 **(b)**

Herein we apply dived and conquer method. Divide and conquer method is based on a basic idea of dividing the original problem into two or more sub-problems. These sub problem, if possible are further sub divided into sub problems. Individually, each sub problem is then solved using the similar technique. The solutions of the sub problems are then combined to result in a final solution.

N is divided into 2 parts, n/2 and n/2

T(n) = 2T(n/2) + 2 for n>2

$T(n) = 3n/2-2$
$= 1.5n-2$

Q. 6 (b)
As given, pivot element splits into two sub list containing atleast 1/5th element.
There are two lists; One list with 1/5th element and other are with 4/5th elements (1-1/5 =4/5).
For $1/5^{th}$ elements number of comparisons T(n/5) & for $4/5^{th}$ element number of comparisons T(4n/5) and n is the time to split
Therefore, recurrence relation is $T(n) \leq T(n/5)+ T(4n/5)+ n$

Q.7 (a)
In the selection sort, element are not swapped with every move. The algorithm finds out the smallest element in the list and then is placed in single swap. This determines that is the worst case maximum swap that could happen is n. Therefore, the number of swap is θ (n).

Q. 8 (b)
According to the problem, the pivot element is selected in a way that it divides the array in $1/4^{th}$ and $3/4^{th}$ elements. So, we get the relation -
$T(n)=T(n/4) +T(3n/4)+n$
Solving the relation, we get O (n log n). Therefore, the average complexity of quick sort is O (n log n).

Q. 9 (b)
When we are sorting an array of n integers, Recurrence relation for Total number of comparisons involved will be,
$T(n) = 2T(n/2) + n$, where f(n)=n is the number of comparisons in order to merge 2 sorted subarrays of size n/2.
$T(n) = (n\log_2 n)$
Instead of integers whose comparison take O(1) time, we are given n strings. We can compare 2 strings in O(n) worst case. Therefore, Total number of comparisons now will be $(n^2\log_2 n)$ where each comparison takes O(n) time now.
In general, merge sort makes $(n\log_2 n)$ comparisons, and runs in $(n\log_2 n)$ time if each comparison can be done in O (1) time.

Q.10 (c)
If elements are already sorted, Quick Sort's behaves in worst case. In every step of quick sort, numbers are divided as per the following recurrence.
$T(n)=T(n-1)+O(n)$

Q.11 (148)
Total number of comparisons
$= 3n/2-2 =148$

Q.12 (358)
The number of comparisons for merging two sorted sequences of length m and n is m+n-1.
Total numbers of comparisons are-
(44-1)+ (94-1) + (65-1) + (159-1) =358

Q.13 (a)
The Worst case time complexity of quick sort is O (n^2). This will happen when the elements of the input array are already in sorted order (ascending or descending), irrespective of position of pivot element in array.

Q.14 (b)
In worst case, the pivot element goes to one of the extremes and divides the array into two parts of size n-1 and 0. So the recurrence relation is
$T(n) = T(n - 1) + T(0) + cn$

Q.15 (a)

If k is smaller than the pivot element we continue the process in the left array. Otherwise we continue with right array.

Q.16 (b)

Time complexity of Merge sort is n*logn.

C*64log64= 30. So C=5/64

For time 6 minutes 5/64*n logn = 6*60

n logn = 72*64 = 512*9

So n=512

Q.17 (a)

In selection sort worst case, it is required to do one swap in each iteration. There will be n-1 iterations. So the number of swaps are θ(n).

Q.18 (d)

Worst case time complexity of Insertion sort is $O(n^2)$ when the elements are in reverse sorted order. Best, Average, Worst case time complexity of Merge sort is O(n logn).

Worst case time complexity of Quick sort is $O(n^2)$.

Q.19 (d)

1. Quicksort will take worst case, if the input is in ascending order i.e $\theta(n^2)$
2. Insertion sort takes θ(n)

Q.20 (c)

$M_1 \times M_2 \times M_3$

For $(M_1 \times M_2) \times M_3$

$= (p \times q \times r) + (p \times r \times s)$

$= (10 \times 100 \times 20) + (10 \times 20 \times 5)$

$= 20000 + 1000 = 21000$

$M_1 \times (M_2 \times M_3)$

$= \{p \times q \times s) + (q \times r \times s)$

$= (10 \times 100 \times 5) + (100 \times 20 \times 5)$

$=5000 + 10000 = 15000$

$M_1 (M_2 \times M_3) < (M\text{-}_1 \times M_2) \times M_3$

This, $(M_1 \times (M_2 \times M_3))\ M_4$

$= 15000 + p \times s \times t$

$=15000 + 10 \times 5 \times 80$

$=15000 + 4000 = 19000$

Q. 21 (c)

The tightest upper bound that represents the time complexity of inserting an object into a binary search tree with n nodes is 0 (n), When inserting a new object (element) into a binary search tree we need to find its exact position in the tree.

The time for this operation is upper bounded by the height of the binary tree.

Since, the tree is unbalanced, balancing operation is not required upon insertion so only time consumed, is in finding the new element's right position in the tree. Thus, the time conserved is in finding the position which is equal to the height of the tree.

Maximum height of the tree =0(n)

Q.22 (b)

Binary search tree can be skewed tree where the height of tree is n-1. So the worst case time complexity of insertion and deletion will be θ(n).

Q.23 (c)

- Towers of Hanoi with n disks=$2T(n-1)+1=\Theta(2^n)$
- Binary search given n sorted numbers = $T(n/2)+1=\Theta(\log n)$
- Heap sort given n numbers at the worst case =$2T(n/2)+n=\Theta(n \log n)$
- Addition of two n ×n matrices = $4T(n/2)+1=\Theta(n^2)$

GATE QUESTIONS (GREEDY METHOD)

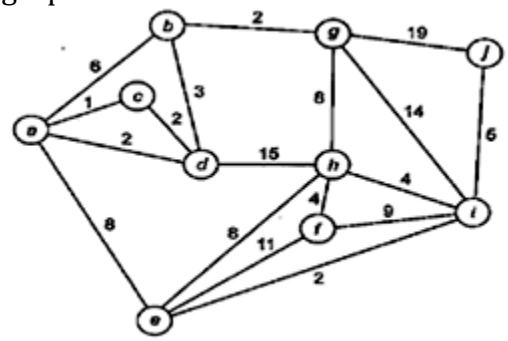
Q.1 The following are the starting and the ending times of activities A, B, C, D, E, F, G and H respectively in chronological order.

a_s b_s c_s a_e d_s c_e e_s f_s b_e d_e g_s e_e f_e h_s g_e h_e . Here x_s denotes the starting time and x_e denotes the ending time of activity X. we need to schedule the activities in a set of rooms available to us. An activity can be scheduled in room only if the room is reserved for the activity for its entire duration. What is the minimum number of rooms required?

a) 3 b) 4
c) 6 d) 5

[GATE-2003]

Q.2 What is the weight of a minimum spanning tree of the following graph?



a) 29 b) 31
c) 38 d) 41

[GATE-2003]

Q.3 Let G = (V, E) be an undirected graph with a sub graph $G_1 = (V_1, E_1)$. Weights are assigned to edges of G as follows

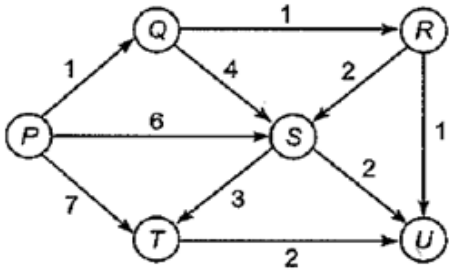
$$w(e) = \begin{cases} 0 \text{ if } e \in E_1 \\ 1 \text{ otherwise} \end{cases}$$

A single-source shortest path algorithm is executed on the weighted graph (V, E, W) with an arbitrary vertex v_1 of V_1 as the source. Which of the following can always be inferred from the path costs computed?

a) The number of edges in the shortest paths from v_1 to all vertices of G
b) G_1 is connected.
c) V_1 forms a clique in G
d) G_1 is a tree

[GATE-2003]

Q.4 Suppose we run Dijkstra's single source shortest-path algorithm on the following edge-weighted directed graph with vertex P as the source.



In what order do the nodes get included into-the set of vertices for which the shortest path distance is finalized?

a) P, Q, R, S, T, U b) P,Q,R, U, S, T,
c) P, Q, R, U, T, S d) P,Q, T, R, U, S

[GATE-2004]

Statements for Linked Answer Question 5 and 6

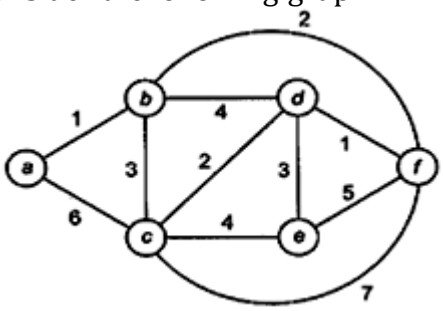
We are given 9 tasks $T_1, T_2, \ldots T_9$. The execution of each task requires one unit of time. We can execute one task at a time. Each task T_i has a profit P_i and a deadline d_i . Profit P_i is earned if the task is completed before the end of the d_i the unit of time.

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Profit	15	20	30	18	18	10	23	16	25
Dead line	7	2	5	3	4	5	2	7	3

Q.5 Are all takes complemented in the schedule that gives maximum profit?
a) All tasks are completed
b) T_1 and T_6 are left out
c) T_1 and T_8 are left out
d) T_4 and T_6 are left out
[GATE-2005]

Q.6 What is the maximum profit earned?
a) 147 b) 165
c) 167 d) 175
[GATE-2005]

Q.7 Consider the following graph:



Which one of the following cannot be the sequence of edges added, in that order, to a minimum spanning tree using Kruskal's algorithm?
a) (a–b), (d–f), (b–f), (d–c), (d – e)
b) (a–b), (d–f), (d–c), (b–f), (d – e)
c) (d–f), (a–b), (d–c), (b–f), (d – e)
d) (d–f), (a–b), (b–f), (d–e), (d–c)
[GATE-2006]

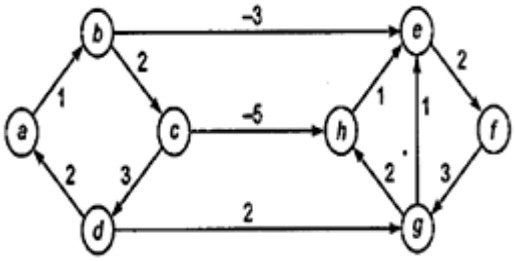
Q.8 To implement Dijkstra's shortest path algorithm on unweighted graphs so that it runs in linear time, the data structure to be used is
a) queue b) stack
c) heap d) B-Tree
[GATE-2006]

Statements for Linked Answer Q.9 & Q.10
Suppose the letters a, b, c, d, e, f have probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}$ respectively.

Q.9 Which of the following is the Huffman code for the letter a, b, c, d, e, f?
a) 0, 10, 110, 1110, 11110, 11111
b) 11, 10, 011, 010, 001, 000
c) 11, 10, 01, 001, 0001, 0000
d) 110, 100, 010, 000, 001, 111
[GATE-2007]

Q.10 What is the average length of the correct answer to Q. 9?
a) 3 b) 2.1875
c) 2.25 d) 1.9375
[GATE-2007]

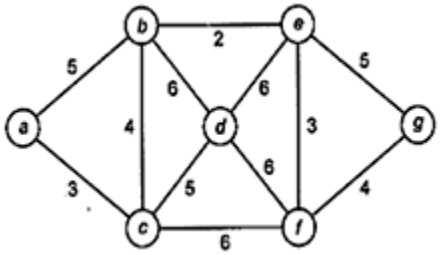
Q.11



Dijkstra's single source shortest path algorithm when run from vertex a in the above graph, computers the correct shortest path distance to
a) only vertex a
b) vertices a, e, f, g, h
c) vertices a, b, c, d
d) all the vertices
[GATE-2008]

Q.12 Which of the following statements is/are correct regarding Bellman-ford shortest path algorithm?
P. Always finds a negative weighted cycle, if one exists.
Q. Finds whether any negative weighted cycle is reachable from the source.
a) P only
b) Q only
c) Both P and Q
d) Neither P nor Q
[GATE-2009]

Q.13 Consider the following graph :



Which one of the following is not the sequence of edges added to the minimum spanning tree using Kruskal's algorithm?

a) (b, e)(e, f)(a, c)(b, c)(f, g)(c, d)
b) (b, e)(e, f)(a, c)(f, g)(b, c)(c, d)
c) (b, e)(a, c)(e, f)(b, c)(f, g)(c, d)
d) (b, e)(e, f)(b, c)(a, c)(f, g)(c, d)

[GATE-2009]

Common Data for Questions 14 and 15

Consider a complete undirected graph with vertex set (0, 1, 2, 3, 4). Entry W_{ij} in the matrix W below is the weight of the edge (i,j)

$$W = \begin{pmatrix} 0 & 1 & 8 & 1 & 4 \\ 1 & 0 & 12 & 4 & 9 \\ 8 & 12 & 0 & 7 & 3 \\ 1 & 4 & 7 & 0 & 2 \\ 4 & 9 & 3 & 2 & 0 \end{pmatrix}$$

Q.14 What is the minimum possible weight of a spanning tree T in this graph such that vertex 0 is a leaf node in the tree T?

a) 7 b) 8
c) 9 d) 10

[GATE-2010]

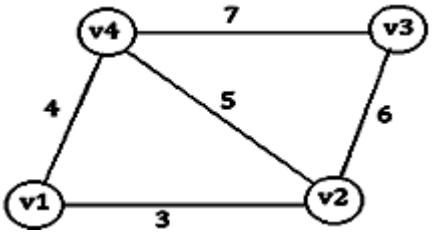
Q.15 What is the minimum possible weight of a path P from vertex 1 to vertex 2 in this graph such that P contain at most 3 edges?

a) 7 b) 8
c) 9 d) 10

[GATE-2010]

Common data for questions 16 and 17

An undirected graph G(V , E) contains n(n > 2) nodes named v1, v2,....vn. Two nodes v_p, v_f are connected if and only if $0 < |i - f| \leq 2$. Each edge (v_p, v_f) is assigned a weight i + j. A sample graph with n = 4 is shown below.



Q.16 What will be the cost of the minimum spanning Tree (MST) of such a graph with n nodes?

a) $1/12(11n^2 - 5n)$ b) $n^2 - n + 1$
c) 6n - 11 d) 2n + 1

[GATE-2011]

Q.17 The length of the path from v_5 to v_6 in the MST of previous question with n=10 is

a) 11 b) 25
c) 31 d) 41

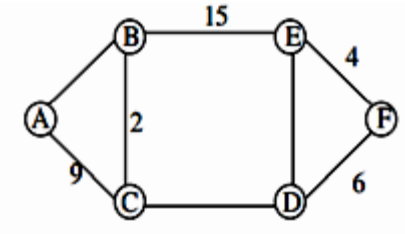
[GATE-2011]

Q.18 What is the time complexity of Bellman-ford single source shortest path algorithm on a complete graph of n vertices?

a) $\theta(n^2)$ b) $\theta(n^2 \log n)$
c) $\theta(n^3)$ d) $\theta(n^3 \log n)$

[GATE-2013]

Q.19 The graph shown below 8 edges with distinct integer edge weights. The minimum spanning tree (MST) is of weight 36 and contains the edges: {(A, C), (B, C), (B, E), (E, F), (D, F)}. The edge weights of only those edges which are in the MST are given in the figure shown below. The minimum possible sum of weights of all 8 edges of this graph is _____.



[GATE-2015]

Q.20 Let G be a connected undirected graph of 100 vertices and 300 edges. The weight of a minimum spanning tree of G is 500. When the weight of each edge of G is increased by five, the weight of a minimum spanning tree becomes ___.

[GATE-2015]

Q.21 Let G be a weighted connected undirected graph with distinct positive edge weights. If every edge weight is increased by the same value, then which of the following statements is/are **TRUE**?

P: Minimum spanning tree of G does not change

Q: Shortest path between any pair of vertices does not change

a) P only b) Q only

c) Neither P nor Q d) Both P and Q

[GATE-2016]

Q.22 Let G be a complete undirected graph on 4 vertices, having 6 edges with weights being 1, 2, 3, 4, 5 and 6. The maximum possible weight that a minimum weight spanning tree of G can have is ___.

[GATE-2016]

Q.23 Let G be a weighted graph with edge weights greater than one and G' be the graph constructed by squaring the weights of edges in G. Let T and T' be the minimum spanning trees of G and G' respectively, with total weights t and T'. Which of the following statements is true?

a) T'=T with total weight $t'=t^2$

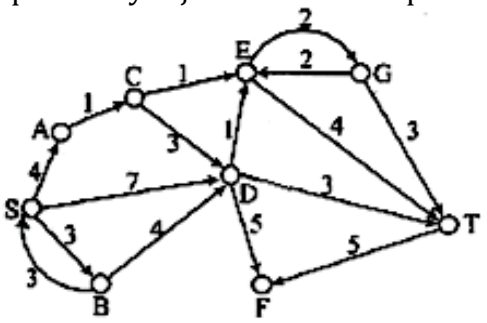
b) T'=T with total weight $t'<t^2$

c) T'≠T but total weight $t'=t^2$

d) None of the above

[GATE-2012]

Q.24 Consider the directed graph shown in the figure below. There are multiple shortest paths between vertices S and T. Which one will be reported by Dijkstra's shortest path



algorithm? Assume that in any iteration, the shortest path to a vertex v is updated only when a strictly shorter path to V is discovered.

a) SDT b) SBDT

c) SACDT d) SACET

[GATE-2012]

Q.25 Which one of the following is the tightest upper bound that represents the number of swaps required sorting it numbers using selection sort?

a) O (log n) b) O (n)

c) O (n log n) d) O (n^2)

[GATE-2013]

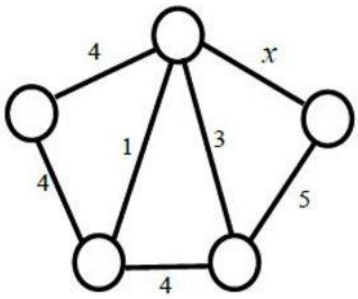
Q.26 A message is made up entirely of characters from the set X= {P,Q,R,S,T}. the table or probabilities for each of he characters is shown below:

Character	Probability
P	0.22
Q	0.34
R	0.17
S	0.19
T	0.08
Total	1.00

If a message of 100 characters over X is encoded using Huffman coding, then the expected length of the encoded message in bits is _____.

[GATE-2017]

Q.27 Consider the following undirected graph G:



Choose a value for x that will maximize the number of minimum weight spanning trees (MWSTs) of G. The number of MWSTs of G for this value of x is _____ .

a) 4 b) 5
c) 2 d) 3

[GATE-2018]

Q.28 Consider the weights and values of items listed below. Note that there is only one unit of each item.

Item number	Weight (in Kgs)	Value (in Rupees)
1	10	60
2	7	28
3	4	20
4	2	24

The task is to pick a subset of these items such that their total weight is no more than 11 Kgs and their total value is maximized. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by V_{opt}. A greedy algorithm sorts the items by their value-to-weight ratios in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by V_{greedy}.

The value of $V_{opt} - V_{greedy}$ is _____ .

a) 16 b) 8
c) 44 d) 60

[GATE-2018]

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(b)	(b)	(b)	(b)	(d)	(a)	(d)	(c)	(a)	(d)	(c)	(b)	(d)	(d)	(b)
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**		
(b)	(c)	(c)	69	995	(d)	7	(b)	(d)	(b)	225	(a)	(a)		

EXPLANATIONS

Q. 1 **(b)**

An activity can be scheduled in a room only if the room is reserved for the activity for its entire duration. The answer can be obtained form

$a_s\ b_s\ c_s\ a_e\ d_s\ c_e\ e_s\ f_s\ b_e\ d_e\ g_s\ f_e\ h_s\ g_e\ h_e$

You may have observed that in between b_s and b_e, two activities ended, they are ae and ce.

The maximum is gap is in between bs and be, which is 6 also 2 activities ended so minimum number of rooms required =6-2=4

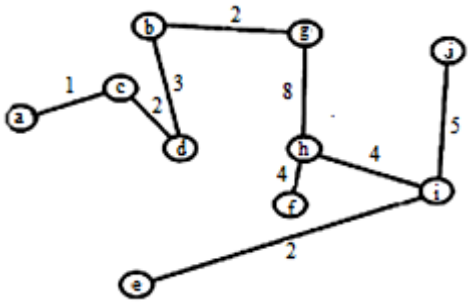
We can verify the answer by solving the question diagrammatically

a_s b_s c_s	a, b,c starts
d_s b_s c_s	a ends
d_s b_s e_s	c ends
d_s b_s e_s f_s	f also starts
e_s f_s	d and b ends
g_s e_s f_s	g starts
g_s	e and f ends
g_s h_s	h starts
	g and h ends thus vacating the room

Q. 2 **(b)**

Minimum spanning tree connects all the vertices considering the weights.

Here, the tree formed will be



This implies, weight of the minimum spanning tree is 1 + 2 + 2 + 2 + 3 + 4 + 4 + 5 + 8 = 31 .

Q. 3 **(b)**

When shortest path from v1 (one of the vertices in V1) is computed. G1 is connected if the distance from v1 to any other vertex in V1 is greater than 0, otherwise G1 is disconnected.

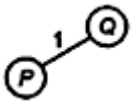
Q. 4 **(b)**

The given diagram of the graph is to be arranged as per the minimum weight.

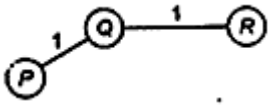
The steps are illustrated as below.

Step 1 Add P

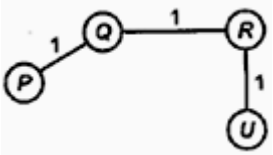
Step 2 Insert Q



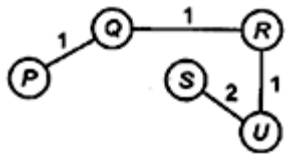
Step 3 Insert R



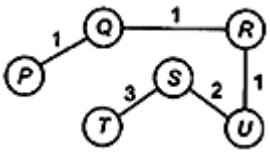
Step 4 Insert U



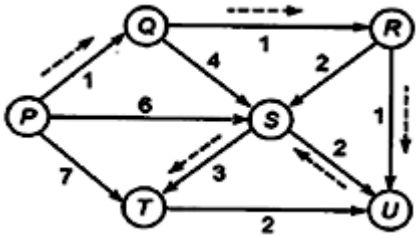
Step 5 Insert S



Step 6 Insert T



Finally



Total distance=1+1+1+2+3=8
Therefore the sequence came out to be P, Q, R, U, S, T .

Q. 5 (d)

As per given, the total tasks are 9.
Let's consider R as empty initially; Now, R is to be fed with the tasks.
The tasks will be allotted as per the given profit and associated deadlines. To maximize profit, we can finish tasks in following order T7, T2, T9, T5, T3, T8, T1.
T_4 and T_6 will be left out as the maximum profit is achieved without entering them.

Q. 6 (a)

To maximize profit, we can finish tasks in following order T7, T2, T9, T5, T3, T8, T1.
We get the maximum profit as 23 + 20 + 25 + 18 + 30 + 16 + 15 = 147

Q. 7 (d)

As given,
The algorithm of Kruskal algorithm
1. Set i = 1 and let E= { }
2. Select an edge e_i of minimum value not in E_{i-1} such that Ti = < E_{i-1} cup {e_i} > is acyclic and define E_i = E_{i-1} cup {e_i}. If no such edge exists, let T=< E_i> and stop.
3. Replace i by i+1. Return to step 2.
It can be explained as- the kruskal algorithm starting with a forest. The forest consists of n tree. Every tree consists only by one node. Now, in every step of the algorithm, two different tree of the forest are connected to form a bigger tree. This is how the quantity is decreased and the size of tree increases. This is done until we end up in a tree which is the minimum spanning tree. In every step the side with the lest cost is chosen. If the chosen side connects node which belong n the same tree the side is rejected, and not examined again because it could produce a circle which will destroy our tree.
Now, considering the options
Option (a)
(a-b)+(d-f)+(b-f)+(d-c)+(d-e)
=1+1+2+2+3=9
Option (b)
(a-b)+(d-f)+(d-c)+(b-f)+(d-e)
=1+1+2+2+3=9
Option (c)
(d-f)+(a-b)+(d-c)+(b-f)+(d-e)
=1+1+2+2+3=9
Option (d)
(d-f)+(a-b)+(b-f)+(d-e)+(d-c)
=1+1+2+3+2=9
In the option (d), the edge with more weight is coming first which is against the algorithm.

Q. 8 (c)

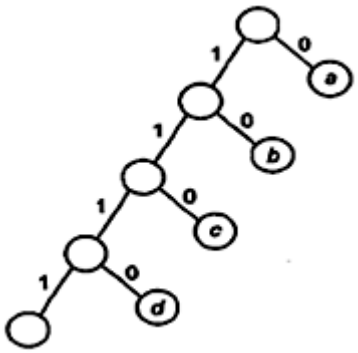
Heap is used to implement Dijakstra's shortest path algorithm on un-weighted graphs so that it runs on linear time because of heap discussed below.
Heap is a data structure that allows the following:

1. Add: Heap allows the addition of an element with some priority associated with each element

2. Remove: The element with the highest priority is removed and returned.
3. Peak: Heap increases the priority of an element to the highest without removing the element form the list.
   Now, to implement a heap, take a list of elements and according to the priority. The highest priority is O(n)time to implement the Dijkstra s shortest path algorithm on unweighted graphs

Q. 9 (a)

We know that characters with high probability need less number of bits and vice-versa. The number of bits required are 1, 2, 3, 4, 5, 5.
a, b, c, d, e, f, are represented by 0, 10, 110, 1110, 11110, 11111
Diagrammatically,



Q. 10 (d)

The formula used here is
Average length = ∑ Bits required × Probability
= 1/2 ×1 + 2 × 1/4 + 3 × 1/8 + 4 × 1/16 + 5 × 1/32 + 5 × 1/32
= 1.9375

Q. 11 (c)

Dijkstra's single source shortest path is not guaranteed to work for graphs with negative weight edges, but it works for the given graph. Let us see...

Let us run the 1st pass
b 1
b is minimum, so shortest distance to b is 1.

After 1st pass, distances are
c 3, e -2.
e is minimum, so shortest distance to e is -2

After 2nd pass, distances are
c 3, f 0.
f is minimum, so shortest distance to f is 0

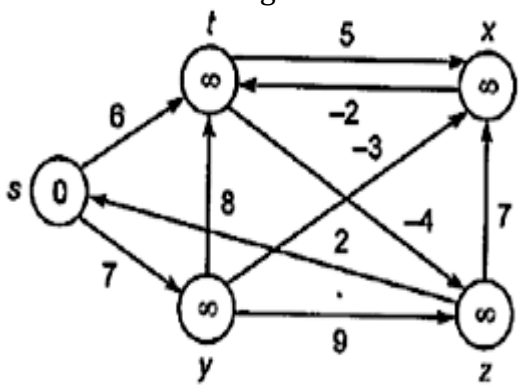
After 3rd pass, distances are
c 3, g 3.
Both are same; let us take g. so shortest distance to g is 3.

After 4th pass, distances are
c 3, h 5
c is minimum, so shortest distance to c is 3

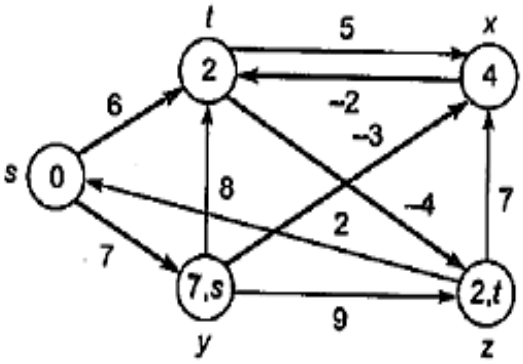
After 5th pass, distances are
h -2
h is minimum, so shortest distance to h is -2

Q. 12 (b)

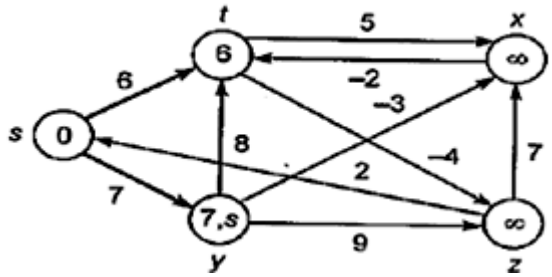
The Bellman-Ford algorithm solves the single source shortest paths problem for a graph. This is done with both positive and negative edge weight. Lets pictorially understand the concept of the Bellman-ford algorithm.



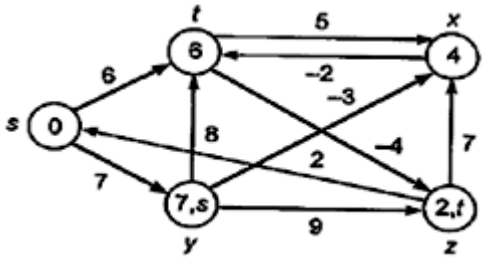
Initialization



After Pass 1



After Pass 2

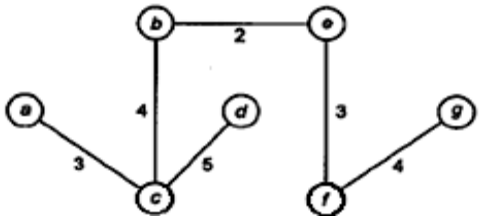


After Pass 3

The order of edges examined in each pass
(t, x), (t, z), (x, t), (y, x), (y, t), (y, z), (z, x), (z, s), (s, t), (s, y)

Q. 13 (d)

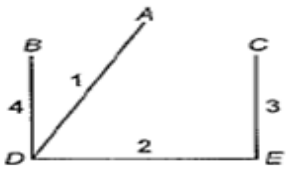
According to the kruskal algorithm, on the graph G, the edges with the smallest weight is selected first.
The weights of the edges are
(b, e) = 2
(e, f) =3
(b, c) =4
(a, c) =8
(f, g) =4
(c, d) =5
Therefore, according to the weights above, we get the following graph



Q. 14 (d)

$$W = \begin{pmatrix} 0 & 1 & 8 & 1 & 4 \\ 1 & 0 & 12 & 4 & 9 \\ 8 & 12 & 0 & 7 & 3 \\ 1 & 4 & 7 & 0 & 2 \\ 4 & 9 & 3 & 2 & 0 \end{pmatrix}$$

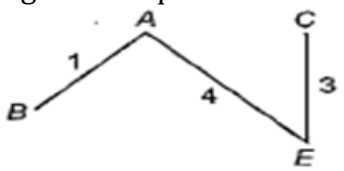
The leaf node has vertex 0,
The tree is 0-1-2-3-4.This determines that the weights of the tree is 10.



Therefore, the minimum possible weight of spanning tree T is O+1+2+3+4=10.

Q. 15 (b)

The part from vertex 1 to vertex 2 is 1-0-4-2.
The length of the path is 1+0+4+3=8



Q. 16 (b)

Minimum spanning tree for 3 nodes would be
Total weight= 3 + 4 = 7
Minimum spanning tree for 4 nodes would be
Total weight= 3 + 4 + 6 = 13
Minimum spanning tree for 5 nodes would be
Total weight= 3 + 4 + 6 + 8 = 21
Minimum spanning tree for 6 nodes would be
Total weight= 3 + 4 + 6 + 8 + 10 = 31

If we check these in the given options, option B satisfies the value.

Q. 17 (c)

Any MST which has more than 5 nodes will have the same distance between v5 and v6 as the basic structure of all MSTs (with more than 5 nodes) would be following.

(v1) _ (v2) _ (v4) _ (v6) _ . . (more even numbered nodes)

|

(v3)

|

(v5)

|

(More odd numbered nodes)

Distance between v5 and v6

= 3 + 4 + 6 + 8 + 10 = 31

Q.18 (c)

The complexity of Bellman-Ford single-source shortest path algorithm is

$\theta(|V|.|E|)$. In a complete graph with n vertices, there are $[n(n-1)]/2$ edges. Thus, the complexity of Bellman-Ford algorithm on a complete graph is $\theta(n^3)$.

Q.19 (69)

In every cycle, the weight of an edge that is not part of MST must be greater than or equal to weights of other edges which are part of MST. Since all edges weights are distinct, the weight must be greater. So the minimum possible weight of ED is 7, the minimum possible weight of CD is 16 and that of AB is 10. So the minimum possible sum of weights is 69.

Q.20 (995)

Since there are 100 vertices, there must be 99 edges in MST. If every edge weight is increased by 5, the total increase in MST weight = 99*5 = 495

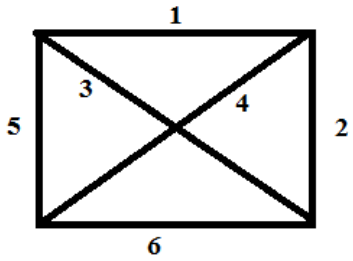
The new weight of MST will be=500+495 = 995.

Q.21 (d)

The edge weights in 2nd graph are increased proportional to edge weights in 1st graph. So the MST does not change and also the shortest path between pairs does not change.

Q.22 (7)

The graph is as below. The weight of MST is: 1+2+4 = 7



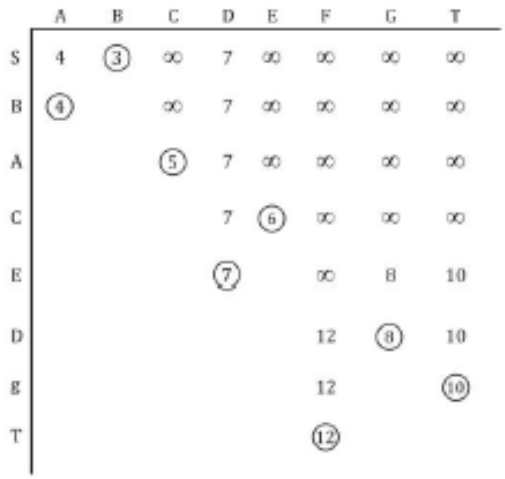
Q.23 (b)

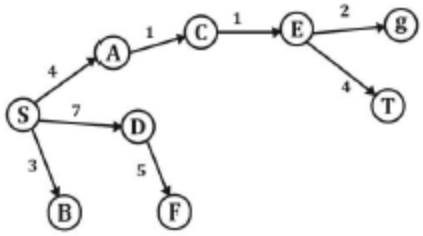
In G' the edge is square of the edge in G. So the same edges which were picked in construction of MST for G will be picked for G' also. So T and T' are same. But the cost of T' is not equal to square of cost of T. Since

If the weights of edges in T are 1, 2, 3. So t=6.

Then weights of edges in T' are 1, 4, 9. So t'=14.

Q. 24 (d)





Q.25 (b)

The tightest upper bound that represents the number of swaps required to sort n numbers using selection sort are 0 (n).

In an unsorted array in selection sort we find the minimum value and swap it with the value placed at the index where the unsorted array starts.

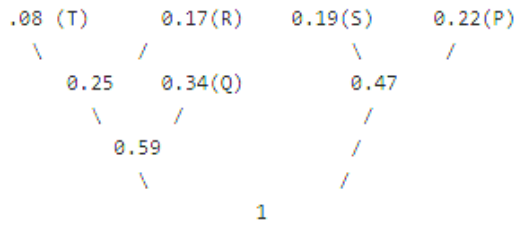
Hence, a number of swaps would be done in order to place each element in its sorted position. ”

There are n such iterations required to sort n numbers

There would be n. O(1) swaps The solution is (b).

Q. 26 225

In Huffman coding, we pick the least two frequent (or probable) character, combine them and create a new node.



Looking at above tree structure,

Number of bits required by each:

P – 2

Q – 2

R – 3

S – 2

T – 3

Therefore, excepted length of the encoded message

= 3*0.8 + 3*0.17 + 2*0.19 + 2 *0.22 + 2*0.34=225

Q.27 (a)

To maximize the number of minimum weight spanning trees of G, the value of x will be 5 because it will have two more choices for corner vertex which will maximize maximum number of MSTs.

Now, according to kruskal algorithm for MST:

1. Edges with weights 1 and 3 will be selected first,
2. Now bottom edge with weight 4 will not be selected as will cause cycle on MST,
3. both corner vertices have two-two choices to select the vertices, so these corner edges with weights 4 and 5 will resultant 2*2 = 4 MSTs.

Therefore, total number of MSTs are 2*2 = 4, which is answer.

Q.28 (a)

Item No	Weight	Value	Value/Weight
1	10	60	6
2	7	28	4
3	4	20	5
4	2	24	12

After sorting :

Item No	Weight	Value	Value/Weight
4	2	24	12
1	10	60	6
3	4	20	5
2	7	28	4

First we will pick item_1 (Value weight ratio is highest). Second highest is item_1, but cannot be picked because of its weight. Now item_3 shall be picked. item_2 cannot be included because of its weight.
Therefore, overall profit by V_{greedy} = 20+24 = 44

Hence, $\mathbf{V_{opt} - V_{greedy} = 60\text{-}44 = 16}$
So, answer is 16.

GATE QUESTIONS (DYNAMIC PROGRAMMING)

Q.1 In an unweighted, undirected connected graph, the shortest path from a node S to every other node is computed most efficiently, in terms of time complexity, by

a) Dijkstra's algorithm starting from S
b) Warshall's algorithm
c) Performing a DFS starting from S
d) Performing a BFS starting from S

[GATE-2007]

Statements for Linked Answer Questions 2 and 3

The subset-sum problem is defined as follows: Given a set of n positive integers, S = $\{a_1, a_2, a_3, ..., a_n\}$, and positive integer W, is there a subset of S whose elements sum to W? A dynamic program for solving this problem uses a 2-dimensional Boolean with n rows and W+1 columns. X[i, j], $1 \le i \le n$, $0 \le j \le W$, is true if and only if there is a subset of $\{a_1, a_2, a_3, ..., a_i\}$ whose elements sum to j.

Q.2 Which of the following is valid for $2 \le i \le n$ and $a_i \le j \le W$?

a) $X[i, j] = X[i-1, j] \vee X[i, j-a_i]$
b) $X[i, j] = X[i-1, j] \vee X[i-1, j-a_i]$
c) $X[i, j] = X[i-1, j] \wedge X[i, j-a_i]$
d) $X[i, j] = X[i-1, j] \wedge X[i-1, j-a_i]$

[GATE-2008]

Q.3 Which entry of the array X, if true, implies that there is a subset whose elements sum to W?

a) X[1, W] b) X[n, 0]
c) X[n, W] d) X[n-1, n]

[GATE-2008]

Q.4 The subset-sum problem is defined as follows: Given a set s of n positive integers and a positive integer W, determine whether there is a subset of S whose elements sum to W.

An algorithm Q solves this problem in O(nW) time. Which of the following statements is false?

a) Q solves the subset-sum problem on polynomial time when the input is encoded in unary
b) Q solves the subset-sum problem in polynomial tome when the input is encoded in binary
c) The subset sum problem belongs to the class NP
d) the subset sum problem is NP hard

[GATE-2008]

Statements for Linked Answer Questions 5 and 6

A sub-sequence of a given sequence is just the given sequence with some elements (possibly none or all) left out. We are given two sequence X[m] and Y[n] of lengths m and n, respectively, with indices of X and Y starting from 0.

Q.5 We wish to find the length of the Longest Common Sub-sequence (LCS) of X[m] and Y[n] as l(m, n), where an incomplete recursive definition for the function l(i, j) to compute the length of the LCS of X[m] and Y[n] is given below

l(i, j)= 0, if either I = 0 or j = 0
= expr 1, if i, j>0 and X[i-1] =[j-1]
= expr 2, if i, j>0 and X[i-1] =Y[j-1]

Which one of the following options is correct?

a) expr1 = l(i-1, j) + 1
b) exp 1 = l(i,j – 1)
c) expr2 = max{(l(i-1, j), l(i, j-1)}
d) expr2 = max {l(i-1, j-1), l(i, j)}

[GATE-2009]

Q.6 The values of l(i, j) could be obtained by dynamic programming

based on the correct recursive definition of l(I, j) of the form given above, using an array L(M, N), where M = m + 1 and N = n + 1, such that L[i, j] = l(i, j).

Which one of the following statements would be true regarding the dynamic programming solution for the recursive definition of l(i, j)?

a) All elements L should be initialized to 0 for the values of l(i, j) to be properly computed
b) The values of l(i, j) may be computed in a row major order or column major order of L(M, N)
c) The values of l(i, j) cannot be computed in either row major order or column major order of L(M, N)
d) L(p, q) needs to be computed before L[r, s] if either p<r or q <s

[GATE-2009]

Q.7 Four matrices M1, M2, M3 and M4 of dimensions p × q , q × r , r ×s , s× t, respectively, can be multiplied in several ways with different number of total scalar multiplications. For example, when multiplied as ((M_1 × M_2) × (M_3 × M_4)), the total number of scalar multiplications is pqr + rst + prt. When multiplied as ((M_1 × M_2) ×M_3)× M_4, the total number of scalar multiplications is pqr + prs + pst. If p = 10, q = 100, r = 20, s = 5 and t = 80, then the number of scalar multiplications needed is

a) 248000 b) 44000
c) 19000 d) 25000

[GATE-2011]

Q.8 An algorithm to find the length of the longest monotonically increasing sequence of numbers in an array A[0:n-1] is given below. Let L_i denote the length of the longest monotonically increasing sequence starting at index i in the array. Initialize L_{n-1} = 1.

For all i such that 0 ≤ i ≤ n – 2

$$L_1 = \begin{cases} 1+L_{i+1}, & \text{if } A[i]<A[i+1] \\ 1, & \text{otherwise} \end{cases}$$

Finally, the length of the longest monotonically increasing sequence in Max (L_0, L_1, L_2... L_{n+1}), which of the following statements is TRUE?

a) The algorithm uses dynamic programming paradigm
b) The algorithm has a linear complexity and uses branch and bound paradigm.
c) The algorithm has a nonlinear polynomial complexity and uses branch and bound paradigm.
d) The algorithm uses divide-and-conquer paradigm.

[GATE-2011]

Q.9 Consider two strings A="qpqrr" and B="pqprqrp". Let x be the length of the longest common subsequence (not necessarily contiguous) between A and B and let y be the number of such longest common subsequences between A and B. Then x+10y= _____.

[GATE-2014]

Q.10 Given are some algorithms, and some algorithm design paradigms.

1) Dijkstra's shortest Path
2) Floyd-Warshall algorithm to compute all pair shortest path
3) Binary search on a sorted array
4) Backtracking search on a graph

i) Divide and Conquer
ii) Dynamic programming
iii) Greedy design
iv) Depth-first search
v) Breadth-first search

Match the above algorithms on the left to the corresponding design paradigm they follow

a)1-i, 2-iii, 3-i, 4-v
b)1-iii, 2-iii, 3-i, 4-v
c)1-iii, 2-ii, 3-i, 4-iv
d)1- iii, 2- ii, 3- i, 4- v

[GATE-2015]

Q.11 Consider the weighted undirected graph with 4 vertices, where the weight of edge {i,j} is given by the entry Wij in the matrix W.

$$W = \begin{vmatrix} 0 & 2 & 8 & 5 \\ 2 & 0 & 5 & 8 \\ 8 & 5 & 0 & x \\ 5 & 8 & x & 0 \end{vmatrix}$$

The largest possible integer value of x, for which at least one shortest path between some pair of vertices will contain the edge with weight x is ____.

[GATE-2016]

Q.12 The Floyd-Warshall algorithm for all-pair shortest paths computation is based on

a) Greedy paradigm.
b) Divide-and-Conquer paradigm.
c) Dynamic Programming paradigm.
d) Neither Greedy nor Divide-and-Conquer nor Dynamic Programming paradigm

[GATE-2016]

Q.13 Let A1, A2, A3, and A4 be four matrices of dimensions 10×5, 5×20, 20×10, and 10×5 respectively. The minimum number of scalar multiplications required to find the product A1A2A3A4 using the basic matrix multiplication method is_____.

[GATE-2016]

Q.14 What is the time complexity of Bellman-Ford single source shortest path algorithm on a complete graph of n vertices?

a) $\theta(n)^2$ b) $\theta(n^2 \log n)$
c) $\theta(n)^3$ d) $\theta(n^3 \log n)$

[GATE-2013]

Q.15 Consider the following table:
Match the algorithms to the design paradigms they are based on.

Algorithms	Design Paradigms
(P) Kruskal	(i) Divide and Conquer
(Q) Quicksort	(ii) Greedy
(R) Floyed – Warshall	(iii)Dynamic Programming

a) (P) ↔ (ii) (Q)↔ (iii), (R) ↔(i)
b) (P) ↔ (iii),(Q) ↔(i),(R)↔(ii)
c) (P) ↔(ii),(Q) ↔(i),(R) ↔(iii)
d) (P) ↔(i),(Q) ↔(ii),(R) ↔(iii)

[GATE-2017]

Q.16 Assume that multiplying a matrix G_1 of dimension p×q with another matrix G_2 of dimension q×r requires pqr scalar multiplications. Computing the product of n matrices $G_1G_2G_3$ ….. G_n can be done by parenthesizing in different ways. Define G_iG_{i+1} as an explicitly computed pair for a given paranthesization if they are directly multiplied. For example, in the matrix multiplication chain $G_1G_2G_3G_4G_5G_6$ using parenthesization $(G_1(G_2G_3))(G_4(G_5G_6))$, G_2G_3 and G_5G_6 are only explicitly computed pairs.

Consider a matrix multiplication chain $F_1F_2F_3F_4F_5$, where matrices F_1,F_2,F_3,F_4 and F_5 are of dimensions 2×25,25×3,3×16,16×1 and 1×1000, respectively. In the parenthesization of $F_1F_2F_3F_4F_5$ that minimizes the total number of scalar multiplications, the explicitly computed pairs is/are

a) F_1F_2 and F_3F_4 only
b) F_2F_3 only
c) F_3F= only
d) F_1F_2 and F_4F_5 only

[GATE-2018]

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(d)	(b)	(c)	(b)	(c)	(b)	(c)	(a)	34	(c)	11	(c)	1500	(c)	(c)	(c)

EXPLANATIONS

Q. 1 (d)

Dijkstra algorithm is not suitable as it is good for weighted graph only.
Warshall's algorithm is not suitable as it is also good weighted graph only.
Depth first search it is not suitable to find the shortest path in team of time complexity.
Breadth first search is the only option.

Q. 2 (b)

The subset sum problem given above is a NP complete problem as this problem in which both NP (verifiable in non-deterministic polynomial time) and NP-hard (any NP-problem can be translated into this problem).
Now, here x is a Boolean array that contains N rows and W+1 columns
Also i^{th} row determines the elements of subject S and j^{th} column determines the corresponding weight.
Given is $2 \leq i \leq n$ and $a_i \leq j \leq W$
Let's subset $S = \{a_1, a_2, a_3, \ldots a_i\}$
And weight is given as W
$X[i, j] = X[i-1, j] \vee X[i-1, j- a_i]$

Q. 3 (c)

X[n, W] is true as in the previous question, the weight W of subject S is found which contains the n elements.

Q. 4 (b)

Statement I is true. Q solve the subset-sum problem in polynomial time, when the input is encoded in unary.
Statement II is false. It is given that an algorithm Q solve the problem in O(nW) time. Here, W is an integer so it is definitely a constant so the input must be encoded in binary. It is can be said that Q solve the problem in O(n) time.
Statement III is true. The subset sum problem belongs to the class NP.
Statement IV is true. The subset sum problem is NP-hard.

Q. 5 (c)

Here, we get the two correct expressions on solving the given
expr 1 = l(i-1, j-1)+1
expr 2 =max (l (i-1, j), (l, j-1))
The answer available is the expr2.

Q. 6 (b)

The solution is continued form the previous solution. The values of l (i, j) may be computed in a row major order or column major order of L (M, N).

Q. 7 (c)

$M_1 \times M_2 \times M_3$
For $(M_1 \times M_2) \times M_3$
$= (p \times q \times r) + (p \times r \times s)$
$= (10 \times 100 \times 20) + (10 \times 20 \times 5)$
$= 20000 + 1000 = 21000$

For $M_1 \times (M_2 \times M_3)$
$= \{p \times q \times s) + (q \times r \times s)$
$= (10 \times 100 \times 5) + (100 \times 20 \times 5)$
$=5000 + 10000 = 15000$
$M_1 (M_2 \times M_3) < (M\text{-}_1 \times M_2) \times M_3$
This, $(M_1 \times (M_2 \times M_3))\ M_4$
$= 15000 + p \times s \times t$
$=15000 + 10 \times 5 \times 80$
$=15000 + 4000 = 19000$

Q. 8 (a)

The algorithm is storing the optimal solutions to sub-problems at each point (for each i), and then using it to derive the optimal solution of a bigger problem. And that is dynamic programming approach. And the program has linear time complexity.

Now, branch and bound comes when we explore all possible solutions (branch) and backtracks as soon as we find we won't get a solution (in classical backtracking we will retreat only when we won't find the solution).In backtracking : In each step, you check if this step satisfies all the conditions
If it does: you continue generating subsequent solutions
If not: you go one step backward to check for another path
So, backtracking gives all possible solutions while branch and bound will give only the optimal one.

The given algorithm here is neither backtracking nor branch and bound. Because we are not branching anywhere in the solution space. And the algorithm is not divide and conquer as we are not dividing the problem and then merging the solution as in the case of merge sort (where merge is the conquer step).

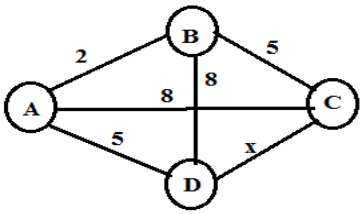
Q.9 (34)

A = "qpqrr" B = "pqprqrp"
The longest common subsequence (not necessarily contiguous) between A and B is having 4 as the length, so x=4 and such common subsequences are as follows:
1) qpqr
2) pqrr
3) qprr
So y = 3 (the number of longest common subsequences) hence x+10y = 4+10*3 = 34

Q.10 (c)

Dijkstra's shortest path is Greedy design. All pairs shortest path is Dynamic programming problem. Binary Search is Divide and Conquer Depth first search is back tracking approach

Q.11 (11)



Let vertices be A, B, C and D. x directly connects C to D. The shortest path (excluding x) from C to D is of weight 12 (C-B-A-D). So to include edge with x it must be at least 11.

Q.12 (c)

All pairs shortest path algorithm is a Dynamic programming problem solving technique.

Q.13 (1500)

There are 5 possible cases
A1 (A2 (A3 A4)), A1 ((A2 A3) A4), ((A1 A2) A3) A4, (A1 (A2 A3)) A4, (A1 A2)(A3 A4).
The scalar multiplications required are 1750, 1500, 3500, 2000, 3000 respectively.
Minimum number of scalar multiplications = 1500

Q. 14 (c)

The time complexity Of Bellman-Ford single source shortest path algorithm on a complete graph of n vertices is 0 (n^3).

Explanation: The time complexity of Bellman-Ford algorithm on a graph with n vertices and m edges is O(n^3).

For a complete graph, m = ${}^nC_2$ = O (n^2), since there is an edge between all pair of vertices.

Time complexity= O (n^2. n)= O (n^3)

Q.15 (c)

- Kruskal's algorithms which is used to find MST uses greedy approach.
- Quick sort uses divide and conquer approach by dividing the input array according to pivot element.
- Floyed Warshall which is used to find all pair shortest path uses dynamic programming.

Q.16 (C)

Matrix F_5 is of dimension 1 X 1000, which is going to cause very much multiplication cost. So evaluating F_5 at last is optimal.

Total number of scalar multiplications are 48 + 75 + 50 + 2000 = 2173 and optimal parenthesis is ((F_1(F_2(F_3 F_4)))F_5).

As concluded F_3, F_4 are explicitly computed pairs.

Option (C) is Correct.

GATE QUESTIONS (P & NP CONCEPTS)

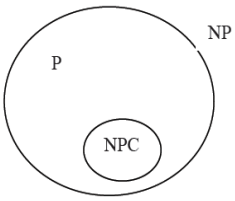
Q.1 Let π_A be a problem that belongs to the class NP. Then, which one of the following is true?

a) There is no polynomial time algorithm for π_A.
b) If π_A can be solved deterministically in polynomial time, then P = NP
c) If π_A is NP-hard, then it is NP-complete.
d) π_A may be undecidable.

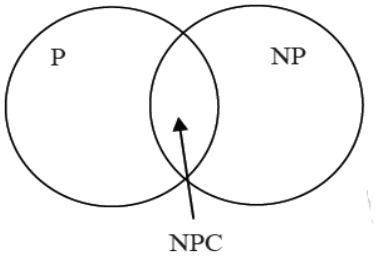
[GATE-2009]

Q.2 Suppose a polynomial time algorithm is discovered that correctly computes the largest clique in a given graph. In this scenario, which one of the following represents the correct Venn diagram of the complexity classes P, NP and NP Complete (NPC)?

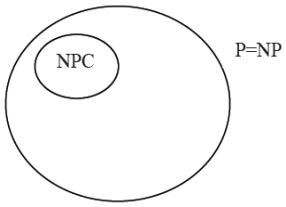
a)



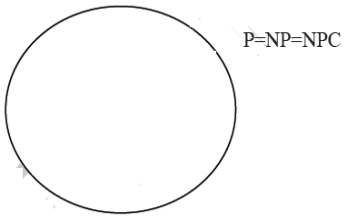
b)



c)



d)



[GATE-2014]

Q.3 Consider the decision problem 2-CNF-SAT defined as follows:

{**π** | **π** is a satisfiable propositional formula in CNF with at most two literals per clause }

For example=$(x_1 \vee x_2) \wedge (x_1 \vee \sim x_3) \wedge (x_2 \vee x_4)$ is a Boolean formula and it is in 2-CNF-SAT.

The decision problem 2-CNF-SAT is

a) NP-Complete.
b) Solvable in polynomial time by reduction to directed graph reach ability.
c) Solvable in constant time since any input instance is satisfiable.
d) NP-hard, but not NP-complete

[GATE-2014]

Q.4 Consider two decision problems Q1, Q2 such that Q1 reduces in polynomial time to 3-SAT and 3 - SAT reduces in polynomial time to Q2. Then which one of following is consistent with the above statement?

a) Q1 is in NP, Q2 in NP hard
b) Q2 is in NP, Q1 is NP hard
c) Both Q1 and Q2 are in NP
d) Both Q1 and Q2 are NP hard

[GATE-2015]

ANSWER KEY:

1	2	3	4
(c)	(d)	(b)	(a)

EXPLANATIONS

Q. 1 **(c)**

A problem which is in P, is also in NP- so A is false. If problem can be solved deterministically in Polynomial time, then also we can't comment anything about P=NP, we just put this problem in P. So, B is also false. C is TRUE because that is the definition of NP-complete.

D is false because all NP problems are not only decidable but decidable in polynomial time using a non-deterministic Turing machine.

Q.2 **(d)**

The most important open question in complexity theory is whether the P=NP, which asks whether polynomial time algorithms actually exist for NP-complete and all NP problems (since a problem "C" is in NP-complete, if C is in NP and every problem in NP is reducible to C in polynomial time). In the question it is given that some polynomial time algorithm exists which computes the largest clique problem in the given graph which is known NP-complete problem. Hence P=NP=NP-Complete.

Q.3 **(b)**

2-SAT is in P. This we can prove by reducing 2-SAT to directed graph reachability problem which is known to be in P.

Q.4 **(a)**

Q1 reduces in polynomial time to 3-SAT. So Q1 is in NP.

3-SAT reduces in polynomial time to Q2. So Q2 is NP-hard. If Q2 can be solved in P, then 3-SAT can be solved in P, but 3-SAT is NP-complete that makes Q2 NP-hard.

GATE QUESTIONS (MISCELLANEOUS CONCEPTS)

Q.1 Let G be an undirected graph. Consider a depth-first traversal of G, and let T be the resulting depth-first search tree. Let u be a vertex in G and let v be the first new (unvisited) vertex visited after visiting u in the traversal. Which of the following statements is always true?

a) {u, v} must be an edge in G, and u is a descendant of v in T
b) {u, v} must be an edge in G, and v is a descendant of u in T
c) {u, v} is not an edge in G then u is a leaf in T
d) {u, v} is not an edge in G then u and v must have the same parent in T

[GATE-2000]

Q.2 Consider any array representation of an n element binary heap where the elements are stored from index 1 to index n of the array. For the element stored at index i of the array(i ≤ n), the index of the parent is

a) i – 1
b) $\left\lfloor \frac{i}{2} \right\rfloor$
c) $\left\lceil \frac{i}{2} \right\rceil$
d) $\frac{(i+1)}{2}$

[GATE-2001]

Q.3 Consider an undirected unweighted graph G. Let a breadth-first traversal of G be done starting from a node r. Let d(r, u) and d(r, v) be the lengths of the shortest paths from r to u and v respectively in G. If u is visited before v during the breadth-first traversal, which of the following statements is correct?

a) d(r , u) < d(r , v)
b) d(r , u) > d(r , v)
c) d(r , u) ≤ d(r , v)
d) None of these

[GATE-2001]

Q.4 The cube root of a natural number n is defined as the largest natural number m such that $m^3 \le n$. The complexity of computing the cube root of n (n is represented in binary notation) is

a) O(n) but not $O(n^{0.5})$
b) $O(n^{0.5})$ but $O(\log n)^k$ for any constant k > 0
c) $O((\log n)^k)$ for some constant k > 0 but not $O((\log \log n)^m)$ for any constant m > 0
d) $O((\log \log n)^k)$ for some constant k >0.5 but not $O((\log \log n)^{0.5})$

[GATE-2003]

Statements for linked answer Questions 5 and 6

In a permutation $a_1 \ldots a_n$ of n distinct integers, an inversion is a pair (a_i, a_j) such that i < j and $a_i > a_j$

Q.5 If all permutations are equally likely, what is the expected number of inversions in a randomly chosen permutation of $1 \ldots n^2$

a) $\frac{n(n-1)}{2}$
b) $\frac{n(n-1)}{4}$
c) $\frac{n(n+1)}{4}$
d) $2n[\log_2 n]$

[GATE-2003]

Q.6 What would be the worst case time complexity of the insertion sort algorithm, if the inputs are restricted to permutations of 1...n with at most n inversions?

a) $\theta(n^2)$
b) $\theta(n \log n)$
c) $\theta(n^{1.5})$
d) $\theta(n)$

[GATE-2003]

Q.7 A program takes as input a balanced binary search tree with n leaf nodes and computes the value of a function g(x) for each node x. If the cost of computing g(x) is min (number of leaf-nodes in left-subtree of x, number of leaf-nodes in right-subtree of x) then the worst-case time complexity of the program is

a) $\theta(n)$ b) $\theta(n \log n)$
c) $\theta(n^2)$ d) $\theta(n^2 \log n)$

[GATE-2004]

Q.8 Suppose each set is represented as a linked list with elements in arbitrary order. Which of the operations among union, intersection, membership, cardinality will be the slowest?

a) union only
b) intersection, membership
c) membership, cardinality
d) union, intersection

[GATE-2004]

Q.9 Two matrices M_1 and M_2 are to be stored in arrays A and B respectively. Each array can be stored either in row-major or column-major order in contiguous memory locations. The time complexity of an algorithm to compute $M_1 \times M_2$ will be

a) best if A is in row-major, and B is in column major order
b) best if both are in row-major order
c) best if both are in column-major order
d) independents of the storage scheme

[GATE-2004]

Q.10 A scheme for sorting binary trees in an array X is as follows. Indexing of X starts at 1 instead of 0. The root is sorted at X[1]. For a node stored at X[i], the left child, if any, is stored in X[2i] and the right child, if any, in X[2i + 1]. To be able to store any binary tree on n vertices the minimum size of X should be

a) $\log_2 n$ b) n
c) 2n + 1 d) $2^n - 1$

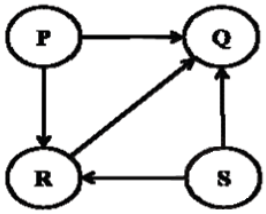
[GATE-2006]

Q.11 The number of elements that can be sorted in $\theta(\log n)$ time using heap sort is

a) $\theta(1)$
b) $\theta\left(\sqrt{\log n}\right)$
c) $\theta(\log n / \log \log n)$
d) $\theta(\log n)$

[GATE-2013]

Q.12 Consider the directed graph given below. Which one of the following is TRUE?



a) The graph doesn't have any topological ordering
b) Both PQRS and SRPQ are topological ordering
c) Both PSRQ and SPRQ are topological ordering
d) PSRQ is the only topological ordering

[GATE-2014]

Q.13 The number of elements that can be sorted in $\theta(\log n)$ time using heap sortis

a) $\theta(1)$ b) $\theta\left(\sqrt{\log n}\right)$
c) $\theta\left(\frac{\log n}{\log \log n}\right)$ d) $\theta(\log n)$

[GATE-2013]

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12	13
(c)	(b)	(c)	(c)	(b)	(d)	(b)	(d)	(d)	(d)	(c)	(c)	(c)

EXPLANATIONS

Q. 1 **(c)**

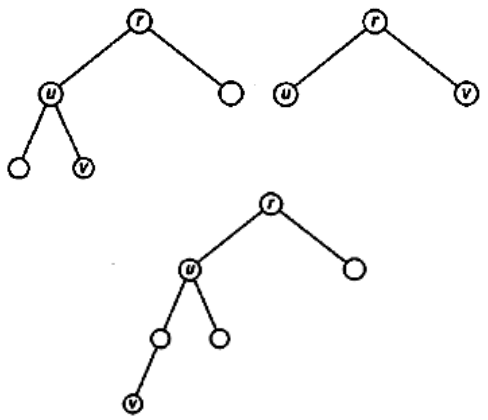
G is an undirected graph. Now, when G is traversed via depth first. The resultant obtained is T (depth first tree). V is the first vertex visited after visiting u. Now, if u and v are not connected, then no cycle is formed and u is a leaf in T. However, if u and v are connected, then a cycle will be formed.

Q. 2 **(b)**

This is a basic question. We know that to reach a node on level i, the distance to the root is i-1. This implies that if an element is stored at index i of the array then, index of the parent is floor (i/2).

Q. 3 **(c)**

Let's, construct diagram of all the possibilities of tree on the given condition that u is visited before v in the breadth first traversal



From, the diagrams, the statement that is correct is $d(r,u) \leq d(r,v)$

Q. 4 **(c)**

We can simply do a binary search in the array of natural numbers from 1..n and check if the cube of the number matches n (i.e., check if a[i]∗a[i]∗a[i]==n). This check takes O(logn) time and in the worst case we need to do the search O(logn) times. So, in this way we can find the cube root in $O(\log_2 n)$. So, options (a) and (b) are wrong.

Now, a number is represented in binary using logn bit. Since each bit is important in finding the cube root, any cube root finding algorithm must examine each bit at least once. This ensures that complexity of cube root finding algorithm cannot be lower than logn. (It must be Ω(logn)). So (d) is also false and (c) is the correct answer.

Q. 5 **(b)**

Since, permutations are equally likely,
Expected number of inversions in a randomly chosen permutation
$= 1/2({}^{n}C_2)$
$= (1/2)\, n!(2!.\, (n-2)!$
$= 1/2 .n(n-1)\, (n-2) \sqrt[1]{\ }\, (2.(n-1)!)$
$= n(n-1)/4$

Q. 6 **(d)**

Insertion sort runs in Θ(n + f(n)) time, where f(n) denotes the number of inversion initially present in the array being sorted. Therefore, the worst case becomes O(n)

Q. 7 **(b)**

The recurrence relation for the recursive function is

T(N) = 2 * T(N/2) + n/2

Where N is the total no. of nodes in the tree.

T(N) = 2 * (2*T(N/2) + n/2) + n/2

= 4 * T(N/2) + 3(n/2)

Solve this till T(1) i.e. till we reach the root.

$T(N)=c^* T(N / 2^i) + (2^*i - 1) * (n/2)$

Where i = log(N)

= log((2n - 1) / 2)

O(c * T(N / 2^i) + (2*i - 1) * (n/2))

reduces to O((2*i - 1) * (n/2))

O((2*(log((2n - 1) / 2)) - 1) * (n/2))

...sub the value of i.

O(n * log(n))

Q. 8 **(d)**

Each set is given as a linked list.

The difference between the traversing nature of linked list and set is that linked list traverse in sequential fashion while a set does not.

Due to this the traversal will begin from start node and ends at root node.

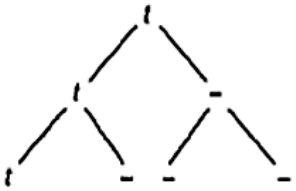
Now, considering options on the same concept. Membership and cardinality takes O(1) for an element so they are faster than intersection and union . therefore, the traversing in a linked list form star node to root is slowest in case of union and intersection.

Q. 9 **(d)**

This is a trick question. Note that the questions ask about time complexity, not time taken by the program. for time complexity, it doesn't matter how we store array elements, we always need to access same number of elements of M1 and M2 to multiply the matrices. It is always constant or O(1) time to do element access in arrays, the constants may differ for different schemes, but not the time complexity.

Q. 10 **(d)**

To find the minimum size of X, we need to consider the worst case size of the tree. The worst case is defined as the case in which the single node contains the date at each level of tree. This can be further explained using an example of worst case binary tree for 3 data items:



With the example we can sum up the concept in one line according to which, for n vertices, n is the level of tree required.

Memory required at each level=2^0 2^1 ,.... ,2^{n-1}

Minimum size = $2^0 + 2^1 + ...+2^{n-1}$

$= 2^n - 1$

Q.11 **(c)**

The time complexity of heap sort for sorting m elements is θ(mlogm). Thus, time complexity to sort (logn/log (logn) element is

$$\theta\left(\frac{\log n}{\log\log n}\log\left(\frac{\log n}{\log\log n}\right)\right)$$

$$=\left(\frac{\log n}{\log\log n}.[\log\log n-\log\log n]\right)$$

$$=\left(\log n-\frac{\log\log n}{\log\log n}\right)=\theta(\log n)$$

Q.12 **(c)**

Topological ordering of a directed graph is a linear ordering of its

vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering. Topological ordering is possible if graph has no directed cycles.

a) As the given graph doesn't contain any directed cycles, it has at least one topological ordering. So option (A) is false
b) PQRS cannot be topological ordering because S should come before R in the ordering as there is a directed edge from S to R. SRQP cannot be topological ordering, because P should come before Q in the ordering as there is a directed edge from P to Q
c) PSRQ and SPRQ are topological orderings as both of them satisfy the above mentioned topological ordering conditions.
d) PSRQ is not the only one topological ordering as SPRQ is another possibility

Q. 13 (c)

$\theta(\log n)$ time using heap sort is $\theta\left(\frac{\log n}{\log\log n}\right)$

Consider the number of elements is k. Which can be sorted in $\theta(k\log k)$ time.

Analyzing the options in decreasing order of complexity sine we need a tight bound i.e., θ

i.e.

$$\theta(\log n),\theta\left(\frac{\log n}{\log\log n}\right),\theta\left(\sqrt{\log n}\right),\theta(1)$$

So it $k \in \theta$ (log n) time required for heap sort is O (k log k) i.e., $\theta(\log n\times\log\log n)$, But this is not in $\theta(\log n)$

If $k\in\theta\left(\frac{\log n}{\log\log n}\right)$ time required for heap sort

$$\theta\left(\frac{\log n}{\log\log n}\times\log\left(\frac{\log}{\log\log n}\right)\right)$$

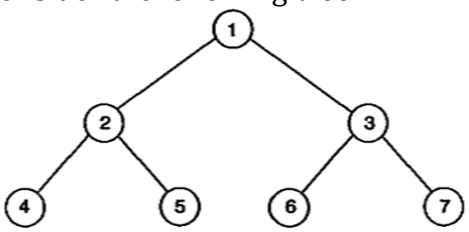
$$\text{i.e.,}\ \theta\left(\log n\times\left\lfloor\frac{\log\left(\frac{\log}{\log\log n}\right)}{\underbrace{\log\log n}_{\le 1}}\right\rfloor\right)$$

So, this is in $\theta(\log n)$

Hence, answer is (c) $\theta\left(\frac{\log n}{\log\log n}\right)$

ASSIGNMENT QUESTIONS(DATA STRUCTURES)

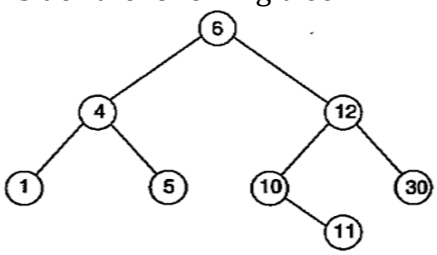
Q.1 Consider the following tree



If the post order traversal gives a b – c d *+ then the label of the nodes 1, 2, 3... Will be

a) +, -, *, a, b, c, d
b) a, -,b, +, c, *, d
c) a, b, c, d, -, *, +
d) -,a, b,+, *, c, d

Q.2 Consider the following tree.



If this tree is used for sorting, then a new number 8 should be placed as the

a) left child of the node labelled 30
b) right child of the node labelled 5
c) right child of the node labelled 30
d) left child of the node labelled 10

Q.3 The initial configuration of a queue is a, b, c, d, ('a' is in the front end). To get the configuration d, c, b, a, one needs a minimum of -

a) 2 deletions and 3 additions
b) 3 deletions and 2 additions
c) 3 deletions and 3 additions
d) 3 deletions and 4 additions

Q.4 The number of possible ordered trees with 3 nodes A, B, C is

a) 16
b) 12
c) 6
d) 10

Q.5 The number of swappings needed to sort the number 8, 22, 7, 9, 31, 19, 5, 13 in ascending order, using bubble sort is

a) 11
b) 12
c) 13
d) 14

Q.6 Given two sorted list of size 'm' and 'n' respectively. The number of comparisons needed in the worst case by the merge sort algorithm will be

a) m × n
b) Maximum of m, n
c) Minimum of m, n
d) m + n -1

Q.7 If the sequence of operation – push (1) , push (2) , pop , push (1) , push (2), pop, pop, pop, push (2), pop, are performed on a stark, the sequence of popped out values are

a) 2, 2, 1, 1, 2
b) 2, 2, 1, 2, 2
c) 2, 1, 2, 2, 1
d) 2, 1, 2, 2, 2

Q.8 A hash table with 10 buckets with one slot per bucket is depicted in Fig. shown below. The symbols, S1 to S7 are initially entered using a hash function with linear probing. The maximum number of comparisons needed in searching an item that is not present is

0	S7
1	S1
2	
3	S4
4	S2
5	
6	S5
7	
8	S6
9	S3

a) 4 b) 5
c) 6 d) 3

Q.9 A binary tree in which every non-leaf node has non-empty left and right Sub trees is called a strictly binary tree. Such a tree with 10 leaves
a) cannot have more than 19 nodes
b) has exactly 19 nodes
c) has exactly 17 nodes
d) cannot have more than 17 nodes

Q.10 The depth of a complete binary tree with 'n' nodes is (log is to the base two)
a) log (n+1)–1 b) log (n)
c) log (n-1)+1 d) log (n)+1

Q.11 Pre-order is same as
a) depth-first order
b) breath-first order
c) topological order
d) linear order

Q.12 Which of the following traversal techniques lists the nodes of a binary search tree in ascending order?
a) post-order
b) In-order
c) pre-order
d) none of the above

Q.13 The average successful search time for binary search on '10' items is
a) 2.6 b) 2.7
c) 2.8 d) 2.9

Q.14 A hash function f defined as f (key) = key mod 7, and linear probing, is used to insert the keys 37, 38, 72, 48, 98, 11, 56, into a table indexed from 0 to 6. What will be the location of key 11?
a) 3 b) 4
c) 5 d) 6

Q.15 The average successful search time for sequential search on 'n' items is
a) n/2 b) (n – 1)/2
c) (n + 1)/2 d) log (n) + 1

Q.16 The running time of an algorithm T(n),where 'n' is the input size is given by
$T(n) = 8T(n/2) + qn$, if $n > 1$
P, if $n = 1$
Where p, q are constants, the order of this algorithm is
a) n^2 b) n^n
c) n^3 d) n

Q.17 Let m , n be positive integers. Define Q(m, n) as
Q(m, n) = 0, if m < n
Q(m - n, n) + p, if m >= n
Then Q(m , 3) is (a div b, gives the quotient when a is divided by b)
a) a constant b)p×(m mod 3)
c) p × (m div 3) d) 3 × p

Q.18 Six files F1, F2, F3, F4, F5 and F6 have 100, 200, 50, 80, 120, 150 number of records respectively. In what order should they be stored so as to optimize access time? Assume each file is accessed with the same frequency?
a) F3, F4, F1, F5, F6, F2
b) F2, F6, F5, F1, F4, F3
c) F1, F2, F3, F4, F5, F6
d) Ordering is immaterial as all files are accessed with the same frequency.

Q.19 In Q. 18 the average access time will be
a) 268 units b) 256 units
c) 293 units d) 210 units

Q.20 An algorithm is made up of 2 modules M1 and M2. If order M1 is f(n) and M2 is g(n) then order of the algorithm is
a) max(f(n) , g(n))
b) min(f(n), g(n))

c) f(n) + g(n)
d) f(n) × g(n)

Q.21 The concept of order (Big O) is important because
a) It can be used to decide the best algorithm that solves a given problem.
b) It determines the maximum size of a problem that can be solved in a given system, in a given amount of time.
c) It is the lower bound of the growth rate of the algorithm
d) none of the above

Q.22 The running time T(n), where 'n' is the input size of a recursive algorithm is given as follows.
T(n) = c+T(n-1), if n>1
d, if n <= 1
The order of this algorithm is
a) n^2 b) n
c) n^3 d) n^n

Q.23 There are 4 different algorithm A1, A2, A3, A4 to solve a given problem with the order log(n), log(log(n)), n log(n), n/log(n) respectively. Which is the best algorithm?
a) A1 b) A2
c) A4 d) A3

Q.24 The number of possible binary trees with 3 nodes is
a) 12 b) 13
c) 5 d) 15

Q.25 The number of possible binary trees with 4 nodes is
a) 12 b) 13
c) 14 d) 15

Q.26 The time complexity of an algorithm T(n), where n is the input size is given by
T(n) = T(n - 1) + 1/n, if n > 1
1, otherwise
The order of this algorithm is
a) log n b) n
c) n^2 d) n^n

Q.27 Sorting is useful for
a) report generation
b) minimize the storage needed
c) making searching easier and efficient
d) responding to queries easily

Q.28 Choose the correct statements.
a) Internal sorting is used if the number of items to be shorted is very large
b) External sorting is used if the number of items to be shorted is very large
c) External sorting needs auxiliary storage.
d) Internal sorting needs auxiliary storage.

Q.29 A sorting technique that guarantees, that records with the same primary key occurs in the same order in the sorted list as in the original unsorted list is said to be
a) stable b) consistent
c) external d) linear

Q.30 A text is made up of the characters a, b, c, d, e each occurring with the probability .12, .4, .15, .08 and .25 respectively. The optimal coding technique will have the average length of
a) 2.15 b) 3.01
c) 2.3 d) 1.78

Q.31 In the previous question which of the following characters will have codes of length 3?
a) Only c b) Only b
c) b and c d) Only d

Q.32 The running time of an algorithm is given by
T(n)= T(n-1)+T(n -2)–T(n-3), if n >3
N, otherwise

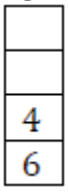
The order of this algorithm is
a) n b) log n
c) n^n d) n^2

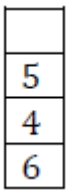
Q.33 What should be the relation between T(1), T(2) and T(3), so that Qn. 32, gives an algorithm whose order is constant?
a) T(1)=T(2)=T(3)
b) T(1)+T(3)=2T(2)
c) T(1) - T(3) = T(3)
d) T(1) + T(2) = T(3)

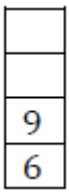
Q.34 The Ackermann's function
a) has quadratic time complexity
b) has exponential time complexity
c) can't be solved iteratively
d) has logarithmic time complexity

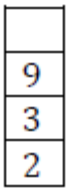
Q.35) The order of an algorithm that finds whether a given Boolean function of 'n' variables, produces a 1 is
a) constant b) linear
c) logarithmic d) exponential

Q.36 In evaluating the arithmetic expression 2*3– (4+5), using stacks to evaluate its equivalent post-fix form, which of the following stack configuration is not possible?

a) 

b) 

c) 

d) 

Q.37 The way a card game player arranges his cards as he picks them up one by one, is an example of
a) bubble sort b) selection sort
c) insertion sort d) merge sort

Q.38 You want to check whether a given set of items is sorted. Which of the following sorting methods will be the most efficient if it is already sorted in
sorted order?
a) bubble sort b)selection sort
c) insertion sort d) merge sort

Q.39 The average number of comparisons performed by the merge sort algorithm, in merging two sorted list of length 2 is
a) 8/3 b) 8/5
c) 11/7 d) 11/6

Q.40 Which of the following sorting method will be the best if number of swapping done, is the only measure of efficiency?
a) bubble sort b) selection sort
c) insertion sort d) merge sort

Q.41 You are asked to sort 15 randomly generated numbers. You should prefer
a) bubble sort b) selection sort
c) insertion sort d) merge sort

Q.42 As part of the maintenance work, you are entrusted with the work or rearranging the library books in a shelf in proper order, at the end of each day. The ideal choice will be
a) bubble sort b)selection sort
c) insertion sort d) merge sort

Q.43 The maximum number of comparison needed to sort 7 items using radix sort is (assume each item is a 4 digit decimal number)
a) 280 b) 40
c) 47 d) 38

Q.44 Which of the following algorithm exhibits the unnatural behavior that, minimum number of comparisons are needed if the list to be sorted is in the reverse order and maximum

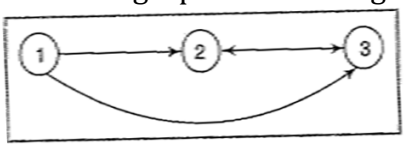
number of comparisons are needed if they are already in sorted order?
a) Heap sort
b) Radix sort
c) Binary insertion sort
d) There can't be any such sorting method

Q.45 Which of the following sorting algorithm has the worst time complexity of nlog(n)?
a) Heap sort b) Quick sort
c) Insertion sort d) Selection sort

Q.46 Which of the following sorting methods sorts a given set of items that is already in sorted order or in reverse sorted order with equal speed?
a) Heap sort b) Quick sort
c) Insertion sort d) Selection sort

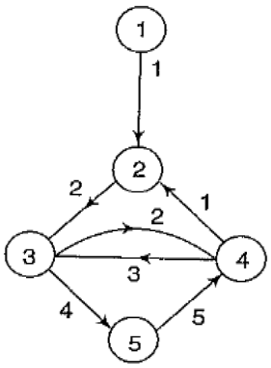
Q.47 Which of the following algorithm solves the all-pair shortest path problem?
a) Dijkstra's algorithm
b) Floyd's algorithm
c) Prim's algorithm
d) Warshall's algorithm

Q.48 Consider the graph in below Fig.



The third row in the transitive closure of the above graph is
a) 1, 1, 1 b) 1, 1, 0
c) 1, 0, 0 d) 0, 1, 1

Q.49 The eccentricity of the node labelled 5 in the graph in below Fig. is



a) 6 b) 7
c) 8 d) 5

Q.50 The centre of graph in Qn. 49 is the node labelled
a) 1 b) 2
c) 3 d) 4

Q.51 Stack A has the entries a, b, c (with a on top). Stack B is empty. An entry popped out of stack A can be printed immediately or pushed to stack B. An entry popped out of stack B can only be printed. In this arrangement, which of the following permutations of a, b, c is not possible?
a) b a c b) b c a
c) c a b d) a b c

Q.52 In the previous problem, if the stack A has 4 entities, then the number of possible permutations will be
a) 24 b) 12
c) 21 d) 14

Q.53 The information about an array that is used in a program will be stored in
a) symbol table b) activation table
c) system table d) dope vector

Q.54 Which of the following expressions access the $(i,j)^{th}$ entry of a (m * n) matrix stored in column major form?
a) n×(i-1)+j b) m×(j-1)+i
c) m×(n-j)+j d) n× (m- i)+j

Q.55 Sparse matrix has
a) many zero entries
b) many non-zero entries
c) higher dimension
d) none of the above

Q.56 In which of the following cases, linked list implementation of sparse matrices consumes the same memory space as the conventional way of storing the entire array? (Assume all data-types need the same amount of storage.)
a) 5×6 matrix with 9 non-zero entities
b)5×6 matrix with 8 non-zero entities
c) 6×5 matrix with 8 non-zero entities
d)6×5 matrix with 9 non-zero entities

Q.57 The linked list implementation of sparse matrices is superior to the generalized dope vector method became it is
a) conceptually easier
b) completely dynamic
c) efficient in accessing an entry
d) efficient if the sparse matrix is a band matrix

Q.58) If the dope vector stores the position is the first and last non-zero entries in each row, then (i, j)th entry in the array can be calculated as (L(x) and F(x) represent the last and first non-zero entries in row x)
a) $\sum_{k=1}^{i-1}\left(L(k)-F(k)+1\right)+\left(j-F(i)+1\right)$
b) $\sum_{k=1}^{i-1}\left(L(k)-F(k)+1\right)+\left(j-F(i)\right)$
c) $\sum_{k=1}^{i-1}\left(L(k)-F(k)+1\right)+\left(i-F(j)+1\right)$
d) $\sum_{k=1}^{i-1}+\left(L(k)-F(k)+1\right)+\left(i-F(j)\right)$

Q.59 The postfix equivalent of the prefix * + a b – c d is
a) ab + cd - *
b) ab cd + - *
c) ab + cd * -
d) ab + - cd *

Q.60 The order of the binary search algorithm is
a) n
b) n^2
c) nlog(n)
d) log(n)

Q.61 The average search time of hashing, with linear probing will be less if the load factor
a) is far less than one
b) equals one
c) is far greater than one
d) none of the above

Q.62 A hash table can store a maximum of 10 records. Correctly there are records in location 1, 3, 4, 7, 8, 9, 10. The probability of a new record going into location 2, With a hash function resolving collisions by linear probing is
a) 0.6
b) 0.1
c) 0.2
d) 0.5

Q.63 Consider a hashing function that resolves collision by quadratic probing. Assume the address space is indexed from 1 to 8. Which of the following location will never be probed if a collision occurs at position 4?
a) 4
b) 5
c) 8
d) 2

Q.64) A hash table has space for 100 records. What is the probability of collision before the table is 10% full?
a) 0.45
b) 0.5
c) 0.3
d) 0.34
(approximately)

Q.65) Which of the following remarks about Trie indexing are true?
a) It is efficient is dealing with strings of variable length.
b) It is effect in there are few number of data items.

c) The number of disk access can't exceed the length of the particular string that is searched.
d) It can handle insertions and deletions, dynamically and efficiently.

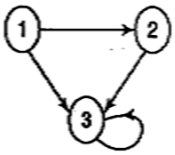
Q.66 Which of the following remarks about Tree indexing are true?
a) It is an m-ary tree.
b) It is a searched tree of order m.
c) Successful search should terminate in left nodes.
d) Unsuccessful search may terminate at any level of the tree structure.

Q.67 Pick the correct statements.
a) Sequential file organization is suitable for batch processing.
b) Sequential file organization is suitable for interactive processing.
c) Index Sequential file organization supports both interactive processing.
d) Relative file can't be accessed sequentially.

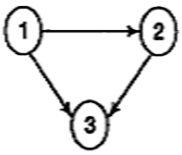
Q.68 Stacks cannot be used to
a) Evaluate an arithmetic expression in postfix form.
b) Implement recursion.
c) Convert a given arithmetic expression in infix form to its equivalent postfix form.
d) Allocate resources (like CPU) by the operating system.

Q.69 Let M be the 3 × 3 , adjacency matrix corresponding to a given graph of 3 nodes labelled 1, 2, 3. If entry (1 , 3)in M^3 is 2, then the graph could be

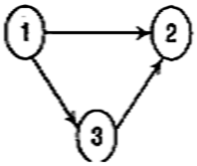
a)



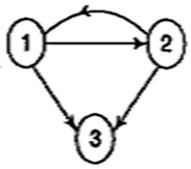
b)



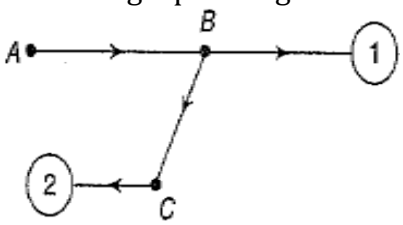
c)



d)



Q.70 Consider the graph in Fig below

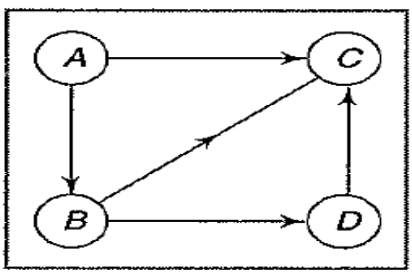


What should be the labels of nodes marked 1 and 2 if the breadth first traversal yields the list a b c d e?
a) D and E
b) E and D
c) unpredictable
d) none of the above

Q.71 If the depth first search of the given in Qn. 70 yields the list A B C D E, then the labels of the nodes marked 1 and 2 will be
a) E and D
b) D and E
c) unpredictable
d) none of the above

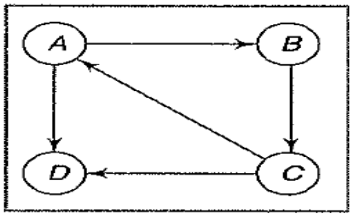
Q.72 Which of the following abstract data types can be used to represent a many to many relation?
a) Tree b) Plex
c) Graph d) Queue

Q.73 Consider the graph in below Fig.



Which of the following is a valid topological sorting?
a) A B C D b) B A C D
c) B A D C d) A B D C

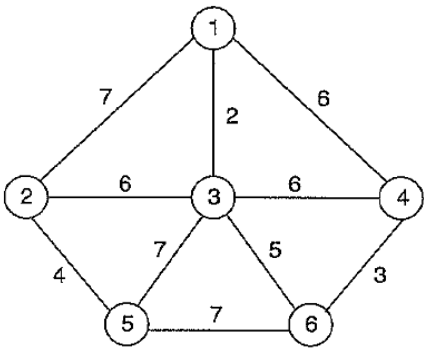
Q.74 Consider the graph in below Fig.



Which of the following is a valid strong component?

a) a, c, d
b) a, b, d
c) b, c, d
d) a, b, c

Q.75 Consider the undirected weighted graph in below Fig.



The minimum cost spanning tree for this graph has the cost

a) 18
b) 20
c) 24
d) 22

Q.76 Merge sort uses

a) divide and conquer strategy
b) backtracking approach
c) heuristic search
d) greedy approach

Q.77 The principle of locality justifies the use of

a) interrupts
b) DMA
c) polling
d)cache memory

Q.78) For merging two sorted lists of sizes m and n into a sorted list of size m + n, we require comparisons of

a) O(m)
b) O(n)
c) O(m + n)
d)O(log(m) + log(n))

Q.79 A binary tree has n leaf nodes. The number of nodes of degree 2 is in tree is

a) $\log_2 n$
b) n - 1
c) n
d) 2^n

Q.80 The minimum number of edges in a connected cyclic graph on n vertices is

a) n - 1
b) n
c) n + 1
d) none of the above

Q.81 The postfix expression for the infix expression
A + B * (C + D) / F + D * E is:

a) A B + CD + F * / D + E *
b) ABCD + * F / + DE * +
c) A * B + CD / F * DE ++
d) A + * BCD / F * DE ++

Q.82 Which of the following statement is true?

I. As the number of entries in the hash table increases, the number of collision increases.
II. Recursive programs are efficient.
III. The worst time complexity of Quick sort is $O(n^2)$.
IV. Binary search implemented using a linked list is efficient.

a) I and II
b) II and III
c) I and IV
d) I and III

Q.83 The number of binary trees with 3 nodes which when traversed in post-order gives the sequence A, B, C is

a) 3
b) 9
c) 2
d) 1

Q.84 The minimum number of colours needed to colour a graph having n (>3) vertices and 2 edges is

a) 4
b) 3
c) 2
d) 1

Q.85 Which of the following file organization is preferred for secondary key procession?

a) Indexed sequential file organization
b) Two-way linked list

c) Inverted file organization
d) Sequential file organization

Q.86 Mr. Fool designed a crazy language called STUPID that includes the following features.
+ has procedure over /
+ has procedure over – (binary)
- binary has procedure over *
* and ^ (exponentiation) have the same procedures.
+ and * associate from right to left.
The rest of the mentioned operators associate from left to right. Choose the correct stack priorities Mr. Fool should assign to+,*, ^, / respectively, for correctly converting an arithmetic expression in infix form to the equivalent postfix form.
a) 5, 2, 2, 4 b) 7, 5, 2, 1
c) 1, 1, 2, 4 d) 5, 4, 3, 1

Q.87 The infix priorities of +, *,^,/ could be
a) 5, 1, 2, 7 b) 7, 5, 2, 1
c) 1, 2, 5, 7 d) 5, 2, 2, 4

Q.88 Mr. Fool's STUPID language will evaluate the expression 2*2^3 * 4 to
a) 256 b) 64
c) 4^{12} d) 4^{81}

Q.89 The expression 1*2^3 * 4 ^ 5 * 6 will be evaluated to
a) 32^{30} b) 162^{30}
c) 49152 d) 173458

Q.90 In a circularly linked list organization, insertion of a record involves the modification of
a) no pointer b) 1 pointer
c) 2 pointers d) 3 pointers

Q.91 Stack is useful for implementing
a) radix sort
b) breadth first search
c) recursion
d) depth first search

Q.92 To store details of an employee, a storage space of 100 characters is needed. A magnetic tape with a density of 1000 characters per inch and an inter-record gap of a 1 inch is used to store information about all employees in the company. What should be the blocking factor so that the wastage doesn't exceed one-third of tape?
a) 0.05 b) 20
c) 10 d) 0.1

Q.93 A machine needs a minimum of 100 sec to sort 1000 names by quick sort. The minimum time needs to sort 100 names will be approximately
a) 50.2 sec b) 6.7 sec
c) 72.7 sec d) 11.2 sec

Q.94 A machine took 200 sec to sort 200 names, using bubble sort. In 800 sec, it can approximately sort
a) 400 names b) 800 names
c) 750 names c) 800 names

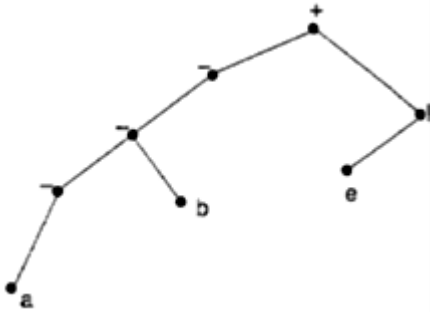
Q.95 The correct order of arrangement of the names Bradman, Lamb, May, Boon, Border, Underwood and Boycott, So that quick sort algorithm makes the least number of comparisons is
a) Bradman, Border, Boon, Boycott, May, Lamb, Underwood
b) Bradman, Border, Boycott, Boon, May, Underwood, Lamb
c) Underwood, Border, Boon, Boycott, May, Lamb, Bradman
d) Bradman, May, Lamb, Border, Boon, Boycott, Underwood

Q.96 Which of the following is useful in traversing a given graph by breadth first search?
a) Stack b) Set
c) Least d) Queue

Q.97 Which of the following is useful in implementing quick sort?
a) Stack b) Set
c) Least d) Queue

Q.98 Queue can be used to implement
a) radix sort
b) quick sort
c) recursion
d) depth first search

Q.99 The Expression tree given in Fig below evaluates to 1, if



a) a=-b and e=0 b)a=-b and e= 1
c) a=b and e=0 d) a=b and e= 1

Q.100 A hash function randomly distributes records one by one in a space that can hold x number of records. The probability that the m^{th} record is the first record to result in collision is
a) $(x-1)(x-2)...(x-(m-2))(m-1)/x^{m-1}$
b) $(x-1)(x-2)...(x-(m-1))(m-1)/x^{m-1}$
c) $(x-1)(x-2)...(x-(m-2))(m-1)/x^{m}$
d) $(x-1)(x-2)...(x-(m-1))(m-1)/x^{m}$

ANSWER KEY:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(a)	(d)	(c)	(b)	(d)	(d)	(a)	(a)	(b)	(a)	(a)	(b)	(d)	(c)	(c)
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
(c)	(c)	(a)	(a)	(a)	(a)	(b)	(b)	(c)	(c)	(a)	(a)	(b)	(a)	(a)
31	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**
(a)	(a)	(a)	(c)	(d)	(d)	(c)	(c)	(a)	(b)	(c)	(b)	(a)	(c)	(a)
46	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
(b)	(b)	(d)	(b)	(d)	(c)	(d)	(d)	(b)	(a)	(c)	(a)	(a)	(a)	(b)
61	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**
(a)	(a)	(d)	(a)	(a)	(a)	(a)	(d)	(a)	(a)	(a)	(b)	(d)	(d)	(b)
76	**77**	**78**	**79**	**80**	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**
(a)	(d)	(c)	(b)	(b)	(b)	(d)	(d)	(c)	(c)	(a)	(d)	(b)	(c)	(c)
91	**92**	**93**	**94**	**95**	**96**	**97**	**98**	**99**	**100**					
(c)	(b)	(b)	(a)	(a)	(d)	(a)	(a)	(a)	(a)					

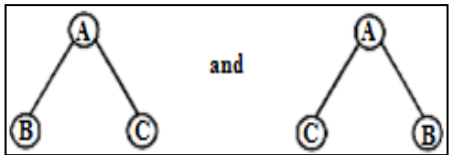
EXPLANATIONS (DATA STRUCTURES)

Q.1 **(a)**
Post order traversals yields 4, 5, 2, 6, 7, 3, 1. Comparing with a, b, -, c, d, *, +, we get the labels of nodes 1, 2, 3, 4, 5, 6, 7 as +, -, *, a, b, c, d respectively.

Q.2 **(d)**
If it is to be used for sorting, labels of left child should be less than the label of the current node. Coming down the tree, we get left child of node labelled 10 as correct slot for 8.

Q.3 **(c)**
Delete a, b, c and then insert c, then b and then a.

Q.4 **(b)**
It is 12. The tree may be of depth 2 or 1. If the depth is 2, we have 6 possible trees. This is because one of three tree nodes A, B, C may be the root and the next level may be one of the remaining two nodes. If the depth is 1, the root may be one of the 3 nodes A, B, C. Corresponding to a root say A, Two trees are possible as this.



This gives us 6 more possibilities.

Q.5 **(d)**
Number of swaps in 1st iteration: 5
Number of swaps in 2nd iteration: 4
Number of swaps in 3rd iteration: 2
Number of swaps in 4th iteration: 1
Number of swaps in 5th iteration: 1
Number of swaps in 6th iteration: 1
Total = 5+4+2+1+1+1 = 14

Q.6 **(d)**
The maximum number of comparisons needed in merging process of sizes m and n is m+n-1
Each comparison puts one element in the final sorted array. In the worst case m+n-1 comparisons are necessary.

Q.7 **(a)**
Simple stack operation. Take an empty stack and perform push and pop operations in the given sequence.

Q.8 **(a)**
It will be equal to the size of biggest cluster (which is 4 in this case). This is because, assume a search key hashing onto bin 8. By linear probing the next location for searching is bin 9 then 0 and then 1.
So, maximum comparison is 4. This logic may not work if deletion operation is done before the search.

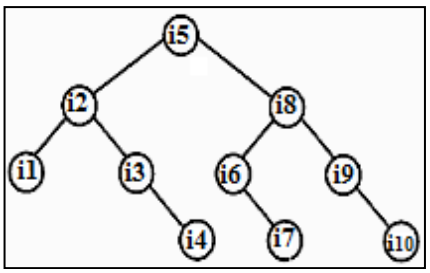
Q.9 **(b)**
A strictly binary tree with 'n' leaves must have (2n - 1) nodes. Verify for some small 'n'. This can be proved by the principle of mathematical induction.

Q.10 **(a)**
If the depth is d, the number of nodes n will be $2^{(d+1)} - 1$.
So, $n + 1 = 2^{(d+1)}$ or $d = \log(n + 1) - 1$

Q.11 **(a)**
Pre-order visits root and move to left most depth. During back tracking it visits right sub tree which is same as depth-first search.

Q.12 (b)
For example, consider the binary search tree given in Q. 2. The in-order listing will be 1, 4, 5, 6, 10, 11, 12, 30, i.e., the number arranged in ascending order.

Q.13 (d)
The 10 items i1, i2, ...i10 may be arranged in a binary search tree in Fig below. To search i5, the number of comparison needed is 1; for i2, it is 2; for i8 it is 2; for i1 it is 3, and so on. The average is (1 + (2 + 2) + (3 + 3 + 3 + 3) + (4 + 4 + 4 + 4))/10, i.e., 2.9.



Q.14 (c)
f(37) = 37 mode 7 = 2. So 37 will be put in location 2. f(38) = 3 so, 38 will be in third location. f(72) = 2 so, this results in a location with linear probing as the collision resolving strategy, the alternate location for 72 will be the location 4 (i.e., next vacant solved in the current configuration). Continuing this way, the final configuration will be 98, 56, 37, 38, 72, 11, and 48.

Q.15 (c)
If the search key matches the very first item, with one comparison we can terminate. If it's second, 2 comparisons, etc. So, as average is (1+2+....+n)/n, i.e., (n+1)/2

Q.16 (c)
By Masters's theorem.

Q.17 (c)
Let m > n. Let m/n yield quotient x and remainder y. So m = n*x+y and y < div 3 is the quotient when n is divided by 3. So, many items p is added, before we terminate recursion by satisfying the end condition. Q(m, n) = 0, if m < n, hence the result.

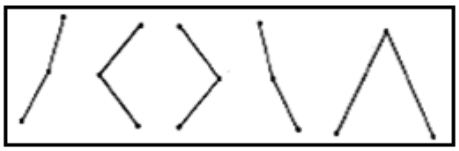
Q.18 (a)
Since access is sequential, greater the distance, greater will be access time. Since all the files are referenced with equal frequency, over all access time can be reduced by arranging them as in (a).

Q.19 (a)
Refer Q. 18. Since each file is referenced with equal frequency each record in a particular file can be referenced with equal frequency so, average access time will be (25 + (50 + 40) + (50+80+50) +...)/6=268 (approximately).

Q.20 (a)
By definition of order, there exist constant c1, c2, n1, n2 such that -
$T(n) \leq c1 \times f(n)$, for all $n \geq n1$.
$T(n) \leq c2 \times g(n)$, for all $n \geq n2$.
Let N = max (n1, n2) and C = max(c1, c2) so, $T(n) \leq C \times f(n)$, for all $n \geq N$.
$T(n) \leq C \times g(n)$, for all $n \geq N$.
Adding
$T(n) \leq C/2 \times (f(n), g(n))$
Without loss of generality, let max (f(n), g(n)) = f(n) So, $T(n) \leq C/2$ $(f(n) + f(n)) \leq C \times f(n)$.
So, order is f(n) which is max (f(n) , g(n)), by our assumption.

Q.22 (b)
By recursive applying the relation we finally arrive at
T(n-1) = c(n-1)+T(1)=c(n-1)+ d
So, order is n.

Q.24 (c)



Q.25 (c)

Eight possible binary trees of depth 3 and six possible binary trees of depth 2. So, altogether 14 binary trees are possible with 4 nodes.

Q.26 (a)

By substitution method

Q.30 (a)

Using Huffman's algorithm, code for a is 1111; b is 0; c is 110; d is 1110; e is 10. Average code length is $4\times12+1\times4+3\times.15+4\times.08+2\times.25= 2.15$

Q.32 (a)

Let us find what is T(4), T(5), T(6).
T(4)=T(3)+ T(2) – T(1)=3+2 -1 = 4
T(5)=T(4) + T(3)– T(2)=4+ 3 - 2 = 5
T(6)= T(5)+T(4)–T(3)= 5+ 4 - 3 = 6
By induction it can be proved that T(n) = n. Hence the order is n.

Q.33 (a)

Refer Q. 32. Let T(1) = T(2) = T(3) = k (say). Then T(4) = k + k – k = k
T(5) = k + k – k = k
By induction it can be proved that T(n) = k (where k is a constant).

Q.35 (d)

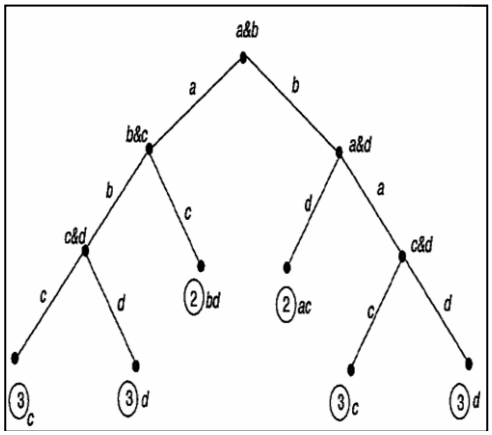
In the worst case it has to check all the 2^n possible input combinations, which is exponential.

Q.36 (d)

The postfix equivalent 2 3 * 4 5 + - . For evaluating this using stack, starting from the left, we have to scan one by one. If scanned element is an operand, push. If scanned element is an operator, pop two elements, apply the operator on the popped out entries and push the result onto the stack. If we follow this, we can find that configuration in (d) is not possible.

Q.39 (a)

Merge sort combines two given sorted lists into one sorted list. For this problem let the final sorted order be –a, b, c, d. The two list (of length two each) should fall into the one of the following 3 categories.
(I) a, b and c, d
(ii) a, c and b, d
(iii) a, d and b, c
The number of comparisons needed in each case will be 2, 3, 3. So, average number of comparisons will be (2 + 3 + 3)/3 = 8/3
Here is a better way of doing:
Let list L1 have the items a, c and L2 have the items b, d.
The tree drawn below depicts the different possible cases. (a & b means a is compared with b. If a is smaller, the edge will be labelled a. The number within a circle, beside the leaf nodes, is the number of comparisons, needed to reach it)
The five possible trees are –



From the trees, we find there are 6 possible ways. Total number of comparisons needed is 3 + 3 + 2 + 2

+ 3 +3=6 so, average number of comparisons is 16/6 = 8/3.

Q.43 (a)
The maximum number of comparison is number of items × number of digits. i.e., 7×10× 4 = 280

Q.47 (b)
Dijkstra's algorithm solved single source shortest path problem. Warshall's algorithm finds transitive closure of a given graph. Prim's algorithm constructs a minimum cost spanning tree for a given weighted graph.

Q.48 (d)
Third row corresponds to node 3. From 3 to 1 there is no path. So the entry (3, 1) should be zero. Since there is a path from 3 to 2 and also from 3 to 3 (i.e. 3 →2 →3), the third row should be 0, 1, 1.

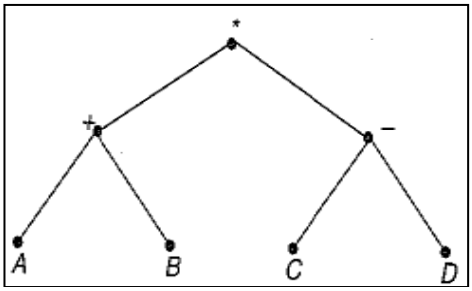
Q.49 (b)
Eccentricity of a given node is the maximum of minimum path from other nodes to the given node.
Cost of minimum path from 1 to 5 is 7
Cost of minimum path from 2 to 5 is 6
Cost of minimum path from 3 to 5 is 4
Cost of minimum path from 4 to 5 is 7
Since, the maximum is 7, eccentricity of node 5 is 7.

Q.50 (d)
Refer Qn. 49.
Find eccentricity of all nodes.
Eccentricity of node 1 is ∞
Eccentricity of node 2 is 6
Eccentricity of node 3 is 8
Eccentricity of node 4 is 5
Eccentricity of node 5 is 7
Center of graph is the node with minimum eccentricity.

Q.52 (d)
Total number of possible permutations for the previous problem is 5. For the four entries a, b, c, d the possibilities are a, followed by permutation of a, b, c which is 5. b followed by permutations of a, c, d. Which is 5. The other possibilities are c, b, a, d, c, d, b, a ; c, b,
d, a ; d, c, b, a. Totally is 14.

Q.56 (c)
Conventional way needs storage of m × n.
In the case of linked list implementation of sparse matrices, storage needed will be m + 3 × (the number of non-zero entries).
Only in case (c), both the methods need the same storage of 30.

Q.59 (a)
The tree whose pre-order traversal yields * + A B – C D, is given in below Fig. Write the post-order traversal of the tree. That is the post-fix form.



Q.60 (b)
Let there be 'n' items to be searched, after the first search the list is divided into two, each of length n/2. After the next search, 2 lists, each of length n/4 and so on. This successive division has to stop when the length of list becomes 1. Let it happen after k steps. After the k steps, $n/2^k = 1$.
Solving, $n = 2^k$. Hence the order is log(n)

Q.61 (a)
Load factor is the ratio of number of records that are currently present and the total number of records that can be present. If the load factor is less, free space will be more. This means probability of collision is less. So, the search time will be less.

Q.62 (a)
If the new record hashes onto one of the six locations 7, 8, 9, 10, 1 or 2, the location 2 will receive the new record. The probability is 6/10 (as 10 is the total possible number of locations).

Q.63 (d)
You can verify that the 1^{st}, 3^{rd}, 5^{th}, 7^{th}....probes check at location 5.
The 2^{nd}, 6^{th}, 10^{th}.... probes check at location 8.
The 4^{th}, 8^{th}, 12^{th}... probes check at location 4.
The rest of the address space will never be probed.

Q.64 (a)
If there is only one record, then the probability of collision will be 1/100. If 2, then 2/100 etc., If 9 then 9/100. So, the required probability is 1 + 2 + 3 ... 9/100 = 0.45.

Q.69 (a)
If the(1 , 3) entry is m^3 is 2, it means there are 2 paths of length 3, connecting nodes 1 and 3. If you see the graph in (a), there are 2 paths connecting 1 and 3, ($1 \rightarrow 2 \rightarrow 3 \rightarrow 3$ and $1 \rightarrow 3 \rightarrow 3 \rightarrow 3$).

Q.70 (a)
In breadth first traversal the nodes are searched level by level. Starting with vertex A the only next choice is B. Then C, then 1 and lastly 2. Comparing with ABCDE, (a) is the correct answer.

Q.71 (a)
In the depth first traversal, we go as deep as possible before we backtrack and look for alternate branches. Here it yields ABC21. So, labels of needs 1 and 2 should be E and D respectively.

Q.73 (d)
In topological sorting we have to list out all the nodes in such a way then whenever there is an edge connecting i and j, i should precede j in the listing. For some graphs, this is not at all possible, for some this can be done in more than one way. (d) is the only correct answer for this question.

Q.74 (d)
Strong component of a given graph is the maximal set of vertices i, j in the set, there is a path connecting i to j. Obviously vertex 'd' can't be in the maximal set (as no vertex Can be reached starting from vertex d). The correct answer is (d).

Q.75 (b)
Use Prim's algorithm or Kruskal's algorithm and verify the result.

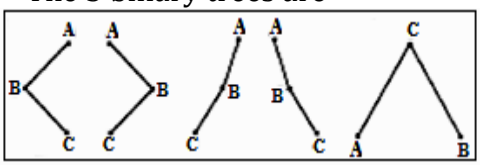
Q.78 (c)
Each comparison will append one item to the existing merge list. In the worst case one needs m + n -1 comparisons which is of order m+ n.

Q.79 (b)
It can be proved by induction that a strictly binary 3 with 'n' leaf nodes will have a total of 2n – 1 nodes.
So number of non-leaf nodes is (2n - 1) – n = n - 1.

Q.82 (d)
Recursive programs take more time than the equivalent non-recursive version and so not efficient. This is

because of the function call overhead.
In binary search, Since every time the current list in probed at the middle, random access is preferred. Since linked list does not support random access, Binary search implemented this way is inefficient.

Q.83 (d)
The 5 binary trees are



Q.92 (b)
Blocking factor is the number of logical records that is packed to one physical record. Here in every 3 inch, there should be 2 inch of information. Hence 2 × 10 = 20 logical records.

Q.93 (b)
In the best case quick sort algorithm makes nlog(n) comparisons. So 1000 × log (1000) = 9000 comparisons, which take 100 sec. To sort 100 names a minimum of 100 (log 100) = 600 comparisons are needed. This takes 100 × 600 / 9000 = 6.7 sec.

Q.94 (a)
Let the first element be the pivot element always. The best way in the one that splits the list into two equal parts each time. This is possible if the pivot element is the median. Consider the given set of names or the equivalent set 1, 2, 3, 4, 5, 6, 7. Four is the median and hence should be the pivot element. Since the first element is the pivot element, 4 should be the first element. After the first pass, 4 will be put in the correct place and we are left which two sub lists 1, 2, 3 and 5, 6, 7. Since 2 is the median of 1, 2, 3 the list should be rearranged as 2, 1, 3 or 2, 3,1. For similar reasons 5, 6, 7 should be rearranged as 6, 5, 7 or 6, 7, 5.

Q.96 (d)
Immediately after visiting a node, append it to the queue. After visiting all its children, The node currently in the head of the queue is deleted. This process is recursively carried out on the current head of the queue, till the queue becomes empty.

Q.99 (a)
The corresponding expression is –(-a- b) + e!. This is 1 if a = -b and e is either 1 or 0, Since 1! = 0! = 1.

Q.100 (a)
Probability for the first record not colliding is x/x.
Probability for the second record not colliding is x -1/x.
(This is because one place is already occupied. So, favorable number of cases is x - 1).
Probability for the third record not colliding is x – 2/x.
Probability for the $(n-1)^{th}$ record not colliding is x – (m - 2)/x.
Now the next (m^{th}) record is resulting in a collision. Out of the x places, Each should hash to one of the (m -1) places already, filled.
So probability is (m -1)/x.
The required probability is
(x/x) (x – 1/x) (x – 2/x)....(x – (m-2)/x) (m -1/x)

ASSIGNMENT QUESTIONS (ALGORITHMS)

Q.1 The recurrence relation for merge sort (where number of elements is greater than one) is _____ .

a) T(n) = 2T(n/2) + kn, where k is constant time required to solve the problem of size 1
b) T(n) = 4T(n/4) + kn, where k is constant time required to solve the problem of size 2
c) T(n) =2T(n/2) + k,where k is constant time to solve the problem of size 1.
d) T(n) = 8T(n/2) + 2n

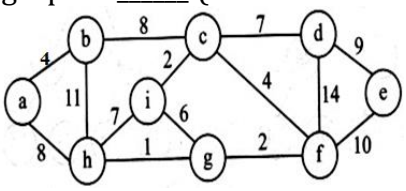
Q.2 The worst case time complexity to construct max heap is

a) O(n) b) O(logn)
c) O(nlogn) d) $O(n^2)$

Q.3 Let X=(x_1, x_2, ... , x_n) and Y=(y_1, y_2, ...y_n) be sequences and let Z = (z_1, z_2 ..... z_k) be any longest common subsequence of x and y. if $x_m \neq y_n$, then $z_k \neq x_m$ implies that z is a longest common subsequence of

a) X_{m-1} and Y_{n-1} b)X_{m-1} and Y_{n-1}
c) X and Y_{n-1} d) X and Y

Q.4 The cost of minimal spanning tree (using prim's algorithm) for below graph is _____ (root vertex I 'a').



a) 37 b) 38
c) 36 d) 28

Q.5 Consider the given below program:

```
int sum (int n)
{
        int sum = n;
        for (int k = 1; k < = n; k + + )
        {
sum + = k;
        }
return sum;
        }
```

What is the step count if assignment, arithmetic computation read & print will take one step?

a) n + 5 b) n + 3
c) n + 2 d) n + 9

Q.6 The time complexity for strassen's matrix multiplication algorithm is ____.

a) $O(n^{2.81})$ b) $O(n^{5.68})$
c) $O(n^{\log n})$ d) $O(n^{1/3})$

Q.7 Which algorithm makes a locally optimal choice step by step in the hope that this choice will lead to globally optimal solution?

a) Dynamic programming
b) Greedy algorithm
c) Branch and Bound
d) Backtracking

Q.8 Does greedy approach give an optimal solution for every instance of knapsack Problem?

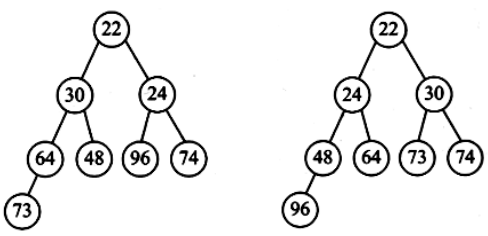
a) Yes
b) No
c) Never gives an optimal solution
d) Greedy approach is not applicable for knapsack problem

Common data questions: 9 and 10

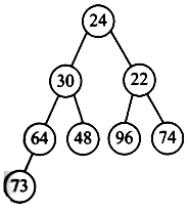
Consider the given below elements:
24, 30, 22, 64, 48, 96, 74, 73

Q.9 Identify the min heap:

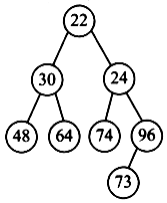
a) b)



c)

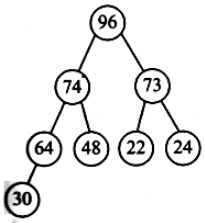


d)

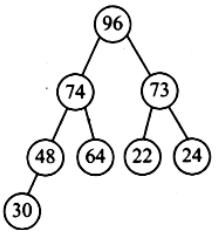


Q.10 Identify the max heap:

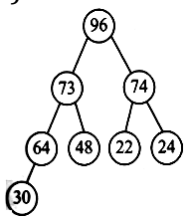
a)



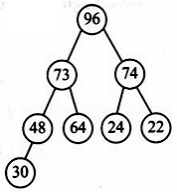
b)



c)



d)



Common data questions: 11 and 12

Consider the following data:

$W_1 = 2, W_2 = 3, W_3 = 4, W_4 = 1$

$P_1 = 10, P_2 = 15, P_3 = 5, P_4 = 12$ and knapsack capacity = 5

Q.11 What is the maximum possible profit, using fractional knapsack?

a) 25 b) 32

c) 17 d) 27

Q.12 Which of the following is the possible combination of weights for maximum profit using 0/1 knapsack concept?

a) (W_1, W_2) b) (W_2, W_4)

c) (W_3, W_4) d) (W_1, W_4)

Q.13 Which of the following uses Divide and Conquer Technique?

(i) Binary search (ii) Quick sort

(iii) Merge sort (iv) Heap sort

a) (i), (ii), (iii) b)(i), (iii), (iv)

c) (ii), (iii), (iv) d) (i), (ii), (iv)

Q.14 Which class consists of problems that can be solved in $O(n^k)$ time where k is a constant and n is size of the input to the problem, using deterministic turning machine.

a) Class NP b)ClassNP–Complete

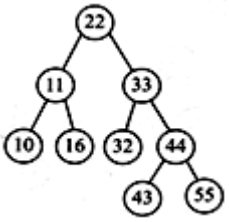
c) Class P d) Class NP – hard

Q.15 Satisfactory of Boolean formulas is _____ problem.

a) Class P b) Class NP

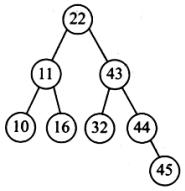
c) NP – complete d) NP – hard

Q.16 Consider the given below binary search tree

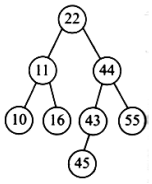


Which of the following is the resultant binary search tree after deletion of 33?

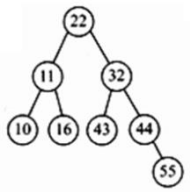
a)



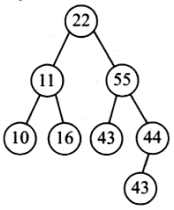
d)



c)



d)



Q.17 Code blocks allow many algorithms to be implemented with

a) Clarity b) Elegance

c) Efficiency d) All of the above

Q.18 The development of a dynamic-programming algorithm can be broken into a sequence of four steps, which are given below randomly.

I. Construct an optimal solution from computed information

II. Compute the value of an optimal solution in a bottom-up fashion.

III. Characterize the structure of an optimal solution.
IV. Recursively, define the value of an optimal solution.

The correct sequence of the above steps is

a) I, II, III, IV b) IV, III, I, II
c) IV, II, I, III d) III, IV, II, I

Q.19 Which of the following is not an application of greedy method?

a) Minimum – spanning tree algorithm
b) Dijkstra's algorithm for shortest paths from a single source.
c) Chavatal's greedy set – covering heuristic
d) Assembly line scheduling

Q.20 Which of the following is/are element(s) of the greedy strategy?

a) Determine the optimal structure of the problem.
b) Develop a recursive solution.
c) Convert the recursive algorithm to an iterative algorithm.
d) Assembly line scheduling

Q.21 Under reasonable assumptions, the expected time to search for an element in a hash Table is

a) lg n b) $O(n^2)$
c) O(n) d) 0(1)

Q.22 To handle "collisions" in which more than one key maps to the same array index. We can use

a) chaining
b) open addressing
c) both a and b
d) None of the above

Q.23 Under the assumption of simple uniform hashing. A hash table in which collisions are resolved by chaining, an unsuccessful search takes expected time.

a) $\theta(1 + 2\alpha)^2$ b) $O(\alpha/2)$
c) $\Omega(\alpha^2)$ d) $\theta(1 + \alpha)$

Q.24 Which of the following is collision Resolution Method in hashing?

a) Linear Probing
b) Quadratic Probing
c) Both a and b
d) None of the above

Q.25 Which of the following technique has the greatest number of probe sequences?

a) Linear Probing
b) Quadratic Probing
c) Double Hashing
d) All are having the same no. of probe sequences

Q.26) In Quadratic probing if two keys have the same initial prob position, then probe sequences are the same, since $h(k_1, 0) = h(k_2,0)$ implies $h(k_1,i) = h(k_2,i)$. This properly leads to a milder form of clustering called

a) primary clustering
b) square clustering
c) secondary clustering
d) super clustering

Q.27) Double hashing improves over linear or quadratic probing in that _____probe sequence is used rather than _____ for i= 0……m – 1.

a) $\theta(m^2)$, $\theta(m)$ b)$\theta(m)$, $\theta(m^2)$
c) $\theta(1)$, $\theta(m)$ d) $\theta(m)$, $\theta(1)$

Q.28) The heap sort procedure takes _____ time.

a) O(n) b) $\theta(n)$
c) $\Omega(n)$ d) O(nlogn)

Q.29) Heap sort algorithm can be implemented by using

a) Min heap tree
b) Max heap tree
c) Both a and b
d) None of the above

Q.30 Turing's halting problem can be solved easily by
a) any computer
b) only by super computer
c) it cannot be solved by any computer
d) None of these

Q.31 For NP-complete problems
a) several polynomial time algorithms are available
b) no polynomial-time algorithm is discovered yet
c) polynomial - time algorithms exist but not discovered
d) polynomial-time algorithms will not exist, hence cannot be discovered

Q.32 Which one of the following is a NP-complete problem finding?
a) longest simple path
b) Hamiltonian cycle
c) 3CNF formula is satisfiable
d) All are NP-complete Problems

Q.33 Given that $p(n) = 10n + 7$
$q(n) = 3n^2 + 2n + 6$
a) function P(n) is asymptotically bigger than q(n)
b) function P(n) is asymptotically smaller than q(n)
c) function P(n) is asymptotically equal to function q(n)
d) Cannot say

Q.34 Given that
$p(n) = 8n^4 + 9n^2$
$q(n) = 100n^3 - 3$
for the above mentioned functions
a) P(n) is bigger than q(n)
b) P(n) is smaller than q(n)
c) Both are equal
d) Cannot say

Q.35 Which one of the following is correct?
a) $1 < n < n \log n < n^3 < n! < n^2$
b) $n \log n < n < n^3 < n! < 1 < n^2$
c) $1 < n \log n < n^3 < n! < n < n^2$
d) $1 < n < n \log n < n^2 < n^3 < n!$

Q.36 If $P(n) = n^2$ then which is correct?
a) $P(n) = O(n^3)$
b) $P(n) = O(n^2)$
c) $P(n) = O(n^4)$
d) All of the above

Q.37 For the recurrence
$T(n)=2T\left(\lfloor \sqrt{n} \rfloor\right)$ +logn which is tight upper bound
a) $T(n) = O(n^2)$
b) $T(n) = O(n^3)$
c) $T(n) = O(\log n)$
d) $T(n) = O(\lg n \lg \lg n)$

Q.38 Consider
$T(n) = 9\,T(n/3) + n$
a) $T(n) = \theta(n^2)$ b) $T(n) = \theta(n^3)$
c) $T(n) = \Omega(n^3)$ d) $T(n) = O(n)$

Q.39 We measure the performance of B-trees
a) By how much computing time the dynamic - set operations consume.
b) By how many disk accesses are performed.
c) Both a and b
d) None of these

Q.40 If $n \geq 1$, then for any n-key, B-tree T of height h and minimum degree $t \geq 2$,
a) $h = n$ b) $h < \frac{n+1}{2}$
c) $h \leq \log_t \frac{n+1}{2}$ d) None

Q.41 The simplest B-Tree occurs for the minimum degree
a) $t = 0$ b) $t = 1$
c) $t = 2$ d) $t = \infty$

Q.42 For a B-Tree, if t is minimum degree, every internal node other than the root has at least
a) t children b) t -1 children
c) t + 1 children d) 2t children

Q.43 With minimum degree t=2 at the depth h=3 what will be the number of nodes (at least)?

a) 8 b) 7
c) 16 d) 9

Q.44 Prim's minimum-spanning tree algorithm and dijkstra's single-source shortest-path algorithm use ideas similar to

a) kruskal's algorithm
b) Depth first algorithm
c) Breadth first algorithm
d) None

Q.45 In a depth-first search of an undirected graph G, every edge of G is

a) Either forward edge or cross edge
b) Tree edge or Back edge
c) Back edge forward edge
d) Tree edge or Cross edge

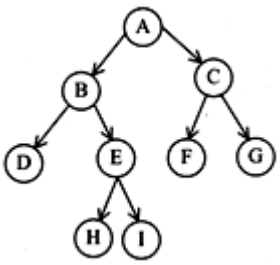
Q.46 Which of the following is false?

a) Minimum spanning tree algorithm is useful in the design of electronic circuit
b) Kruskal's algorithm and Prim's algorithm both can be speed up to run in time O(E + Vlogv)
c) Kruskal's algorithm and Prim's algorithm both are greedy algorithms
d) Prim's algorithm forms a forest and Kruskal's algorithm forms a tree

Q.47 Match the following:

I.	Inorder	1.	ABCDEFGHI
II.	Preorder	2.	DBHEIAFCG
III.	Postorder	3.	ABDEHICFG
IV.	Levelorder	4.	DHIEBFGCA

for the tree



a) I – 2, II – 3, III – 4, IV – 1
b) I – 3, II – 1, III – 4, IV – 2
c) I – 1, II – 2, III – 3, IV – 4
d) I – 4, II – 3, III – 2, IV – 1

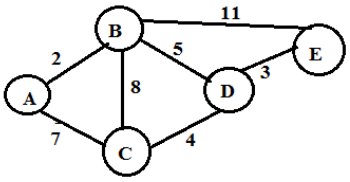
Q.48 A simple graph with 5 vertices and 2 components can have at most

a) 6 edges b) 5 edges
c) 3 edges d) 7 edges

Q.49 Let M be a connected map with V vertices, E edges and R regions. Then

a) V – E + R = 2 b) V + E – R =
c) V + 2 = E – R d) V + R = 2 E

Q.50 Starting with edge AB what will be the minimum spanning tree using kruskal's algorithm?



a) $\overline{AB},\overline{BC},\overline{CD},\overline{DE}$ b) $\overline{AB},\overline{DE},\overline{DC},\overline{DB}$
c) $\overline{AB},\overline{BD},\overline{DE},\overline{DC}$ d) $\overline{AB},\overline{BD},\overline{CD},\overline{BC}$

Q.51 In performance analysis we use _____ and in performance measurement we use_____.

a) Analytical methods, Conduct Experiments
b) Conduct Experiments, Analytical Methods
c) Analytical Method, Analytical Method
d) Conduct Experiments, Conduct Experiments

Q.52 Some computer system requires the user to provide an upper limit on the amount of time the program will run. Which one of the following is/are correct?

a) Once this upper limit is reached, the program is aborted
b) One would like to provide a time limit that is just slightly above the expected run time.

c) If program runs into an infinite loop caused by some discrepancy in the data and you actually get billed for the computer time used.
d) All of the above

Q.53 If c is a constant that denotes the fixed part of the space requirements and S_P denotes the variable components. Then the space requirement of any program may therefore be written as
a) $\frac{C}{S_P}$ b) $C * S_P$
c) $C + S_P$ d) S_P/C

Q.54 What is the time required to evaluate expression a^n?
a) O(n) b) O(logn)
c) $O(n^2)$ d) $O(n^3)$

Q.55 An NP-Complete problem
a) can be solved easily
b) can be solved but not easily
c) easy but cannot be solved
d) status is unknown

Q.56 What is the time required to evaluate expression X^n by using divide and conquer strategy?
a) O(logn) b) O(nlogn)
c) O(n) d) $O(n^2)$

Q.57 Given that
p(n) = 12n + 6
q(n) = 16n + 4
a) both functions are asymptotically equal
b) function p(n) is asymptotically smaller than q(n)
c) function p(n) is asymptotically bigger than q(n)
d) function p(n) and function q(n) asymptotically not comparable

Q.58 Given that
$p(n) = 8n^4 + 9n^2$
$q(n) = 100n^3 - 3$
for the above mentioned functions
a) P(n) is bigger than q(n)
b) P(n) is smaller than q(n)
c) Both are equal
d) Cannot say

Q.59 Which of the following is wrong?
a) $f(n) = \theta(g(n)) \Rightarrow g(n) = \theta(f(n))$
b) $f(n) = \theta(g(n)) \Rightarrow f(n) = O(g(n))$
c) $f(n) = \theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$
d) $f(n) = \theta(g(n)) \Rightarrow g(n) \neq \theta(f(n))$

Q.60 Which of the following is correct? Worst case _____
a) Provides an upper bound on running time.
b) An absolute guarantee that the algorithm would not run longer, no matter what the inputs are.
c) Both (a) and (b)
d) None of the above

Q.61 Which of the following is correct? Best case _____
a) Provides a lower bound on running time
b) Input is the one for which the algorithm runs the fastest
c) Both (a) and (b)
d) None of the above

Q.62 Algorithms for optimization problems typically go through a _____ with _____ at each step.
a) set of choices, sequence of steps.
b) set of steps
c) sequence of steps, set of choices
d) set of steps, sequence of choices

Q.63 A greedy algorithm always makes the choice that looks _____ at the moment.
a) Worst b) Best
c) Average d) Nice

Q.64 Consider the given statements, $f(n) = n^3, g(n) = n^2$

(i) $f(x) \in O(g(x))$
(ii) $g(x) \in O(f(x))$
(iii) $f(x) \in \Omega(g(x))$
Which of the following is true?
a) (i) and (ii) b) (ii) and (iii)
c) (i), (ii) and (iii) d) Only (ii)

Q.65 The optimal-substructure property is exploited by both the
a) Non greedy & static programming
b) Dynamic programming and Static programming
c) Greedy programming and non greedy programming
d) Greedy and Dynamic programming

Q.66 Which of the following is/are classical optimization problem?
a) 0-1 knapsack problem
b) Fractional knapsack problem
c) Both a and b
d) None of the above

Q.67 Under reasonable assumptions, the expected time to search for an element in a hash table is
a) lg n b) $O(n^2)$
c) O(n) d) 0(1)

Q.68 Direct addressing is applicable when we can afford to allocate an array that has
a) one position, for every impossible key.
b) multiple positions, for every possible key.
c) multiple possible positions for one duplicate key.
d) one position for every possible key.

Q.69 To handle "collisions" in which more than one key maps to the same array index, we can use _____
a) chaining
b) open addressing
c) both a and b
d) None of the above

Q.70 A "perfect hashing" can support searches in _____ worst case time.
a) O(n) b) $O(n^2)$
c) O(5) d) O(1)

Q.71 To construct a hash function using multiplication method which of the following steps does not involve? To map the key in the interval 0 to m.
a) First select a real number constant k
b) Compute the value of the fractional part of (key * k)
c) Get the integer_part of the expression m * fractional_part
d) All of above

Q.72 Which is not an application of Hash Table?
a) Compiler
b) Assembler
c) Both (a) and (b)
d) None of the above

Q.73 Heap sort combines the better attributes of the two sorting algorithms.
a) Bubble, selection
b) Selection, insertion
c) Insertion, merge
d) Merge, quick

Q.74 The technique(s) which is/are commonly used to compute the probe sequence required for open addressing.
a) Linear Probing
b) Quadratic Probing
c) Double Probing
d) All the above

Q.75 In the method of linear probing, clusters arise since an empty slot is preceded by i full slots, gets filled next with probability.
a) $\frac{(i+1)}{m}$ b) (i + 1) * m
c) $(i + 1)^m$ d) (i + 1) (m + 1)

Q.76 What is the worst case time complexity of quick sort?
a) $O(n^3)$ b) $O(n)$
c) $O(n^2)$ d) $O(n.\log n)$

Q.77 $an + b \in O(n^2)$ this statement is
a) true
b) false
c) cannot be determined
d) depends on input size

Q.78 For input size 'n' and hash table size m what is the load factor in double hashing?
a) $\alpha = n * m$ b) $\alpha = n + m$
c) $\alpha = n^m$ d) $\alpha = n/m$

Q.79 Assuming uniform hashing an open address hash table with load factor $\alpha=n/m<1$, the expected number of probes in an unsuccessful search is almost.
a) $\frac{1}{1-\alpha}$ b) $1 - \alpha$
c) $1 + \alpha$ d) α^2

Q.80 The expected height of a randomly built search tree is
a) O(n) b) O(lgn)
c) O(1) d) None of the above

Q.81 The heap sort procedure takes time in general
a) O(n) b) θ(n)
c) Ω(n) d) O(nlogn)

Q.82 Heap sort algorithm can be implemented by using
a) Min heap tree
b) Max heap tree
c) Both a and b
d) None of the above

Q.83 The binary search tree property allows us to print out all the keys in a binary search tree in sorted order by a simple recursive algorithm, called an
a) preorder tree walk
b) inorder tree walk
c) Postorder tree walk
d) levelorder tree walk

Q.84 Insertion sort
Merge sort
Heap sort
Quick sort
Which is wrong for the above mentioned sort algorithms?
a) All are comparison sort
b) The lower bound is $\Omega(n \lg n)$
c) The upper bound is $O(n^2)$
d) a, b, c are correct

Q.85 Which of the following sorting algorithms cannot beat the lower bound?
a) counting sort algorithm
b) radix sort algorithm
c) bucket sort algorithm
d) All A, B, C can beat the lower bound Ω(nlgn).

Q.86 Under reasonable assumptions, the expected time to search for an element in _____ is O(1)
a) Linear search algorithm
b) Binary search algorithm
c) Hashing
d) None of the above

Q.87 B trees are
a) Balanced search trees
b) Designed to work on magnetic disks
c) Designed to work on direct access secondary storage
d) All of the above

Q.88 What data structure is used to implement breadth First search?
a) array b) Linked list
c) Queue d) stack

Q.89 What data structure is used to implement Depth First search?
a) Linked list b) array
c) stack d) Queue

Q.90 If t is the minimum degree for a Btree. What is the minimum number of keys, every node other than the root must have?
a) t keys b) t + 1 keys
c) log t keys d) t – 1 keys

Q.91 For a B-Tree, if t is minimum degree, every internal node other than the root has atleast
a) t children b) t – 1 children
c) t +1 children d) 2t children

Q.92 For a B-Tree, if t is minimum degree, every node can contain atmost
a) t keys b) 2t keys
c) 2t – 1 keys d) 2t + 1 keys

Q.93 Which of the following is not a Depth
First Search?
a) Pre order b) In order
c) Level order d) Post order

Q.94 Finding the strongly connected components of a directed graph is
a) Breadth first search
b) Depth First Search
c) Both
d) None of these

Q.95 Which of the following is/are true about Adjacency lists?
a) can readily be adapted to represent weighted graphs.
b) There is no quicker way to determine if a given edge (u, v) is present in the graph
c) For both directed & undirected graphs, the desirable property that the amount of memory it requires is θ(V+E)
d) All of the above are true

Q.96 In any depth-first search of a graph G = (V, E), for any two vertices u and v, which of the following holds?
a) The intervals [d[u], f[u]] and [d[v],f[v]] are entirely disjoint, and neither u nor v is descendent of the other in the depth-first forest.
b) The internal [d[u],f[u]] is contained entirely within the interval [d[v],f[v]], and u is descendent of v is a depth-first tree.
c) The interval [d[v],f[v]] is contained entirely within the interval [d[u],f[u]], and v is descendent u in a depth-first tree
d) None

Q.97 In a breadth-first search of a directed graph, the following property(ies) hold
a) For each tree edge (u,v), we have $d[\upsilon] = d[u] + 1$
b) For each cross edge (u,v), we have $d[\upsilon] \leq d[u] + 1$
c) For each back edge (u, v) we have $0 \leq d[\upsilon] \leq d[u]$
d) All of the above

Q.98 A shortest-path tree rooted at s is a directed subgraph G' = (V', E') where $V' \subset V$ and E such that
a) V' is the set of vertices reachable from s in G.
b) G' forms a rooted tree with roots
c) for all $\upsilon \in V'$, the unique single path from s to υ in G' is a shortest path from s to υin G.
d) All the above

Q.99 What is the best-case time complexity of Insertion sort?
a) O(log n) b) O(n log n)
c) $O(n^2)$ d) O(n)

Q.100 Which of the following is correct?
a) Shortest - path algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it.
b) The above mentioned property is hallmark of the applicability of both dynamic programming and the greedy method.
c) The Floyd-Warshall algorithm, which finds shortest paths between all pairs of vertices is a dynamic - programming algorithm.
d) All of the above

EXPLANATIONS (ALGORITHMS)

Q. 1 **(a)**

Merge sort recurrence relation is
$T(n) = 2T(n/2) + kn$
Where k is constant time required to solve the problem of size 1.

Q. 3 **(b)**

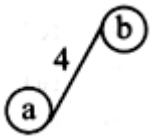
According to theorem optimal substructure of an LCS, Z is an ICS of X_{m1} and Y.

Q. 4 **(a)**

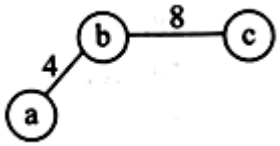
In prim's algorithm we start with root vertex and grow until the tree spans all vertices in v. At each step a light edge is added to the tree.

Step-1:

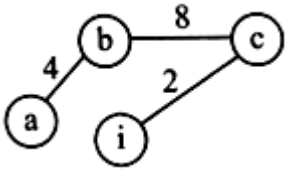
Step-2:



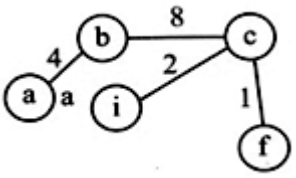
Step-3:



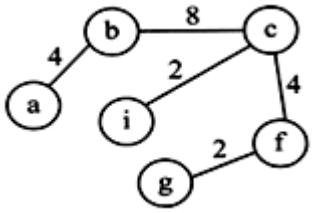
Step-4:



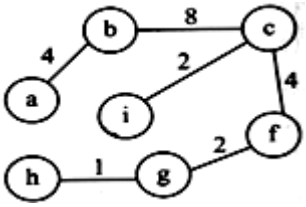
Step-5:



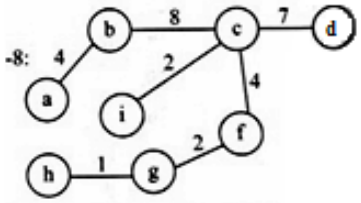
Step-6:



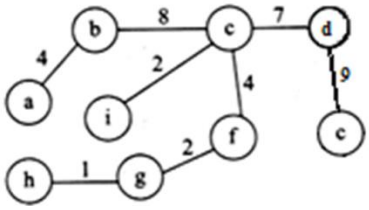
Step-7:



Step-8:



Step-9:



Cost = 4+8 + 2+ 4 + 2+ 1 + 7 + 9 = 37

Q. 5 **(c)**

There is no step count for function definition line, opening and closing braces.

Step count

```
Int     sum(int n)              0
{                               0
        Int sum=0               1
        For (int k = 1;
        K < n; k + +)           n
{
        Sum + = k;              1
}
Return to sum;          1
```

The loop contains single statement and is executed for n times.

So step count = 1 + n × 1 + 1 = n + 2

Q. 6 (a)

Strassen's matrix multiplication chooses sub-matrices to multiply and its recurrences relation is

$T(n) = 7T(n/2) + 0(n^2)$

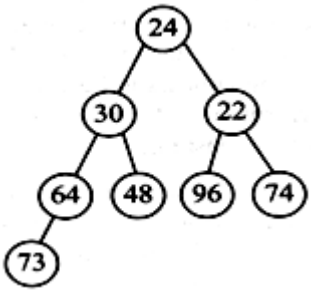
And is solved in $T(n) = 0(n^{\log_2 7}) = 0(n^{2.81})$

Q. 8 (b)

It does not give optimal solution of 0/1 knapsack problem.

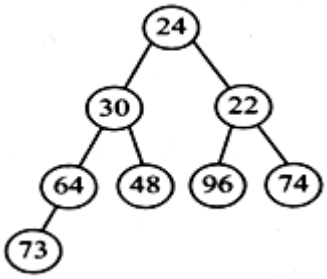
Q. 9 (a)

Arrange the given elements in a binary tree and the tree must be complete i.e., place a node the right child of a node if that node has left child i.e.,



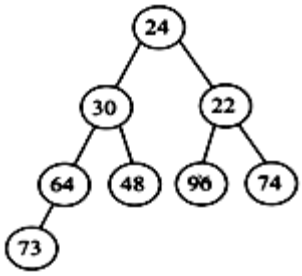
Now check from the last node to root node such that root contains minimum

Value compared to its children otherwise swap those two values.

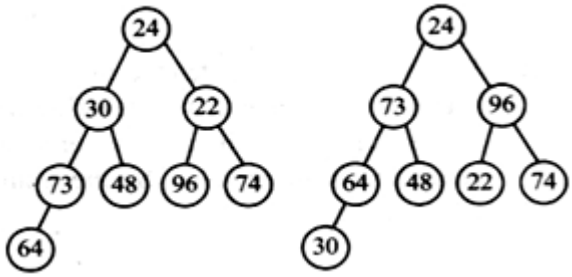


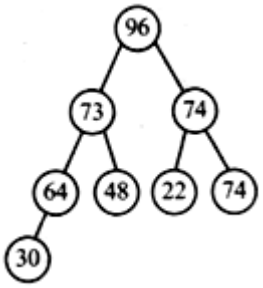
Q. 10 (c)

First construct complete binary tree and then check from the last node to the root such that root node contains maximum value compared to its child nodes otherwise swap the two values. Complete binary tree fro given data will be



Now check from the bottom of the tree to the root node such that root contain maximum value compared to its children





Q. 11 (b)

$W_2 = 3$ $P_2 = 15$

$W_4 = 1$ $P_4 = 12$

$1/2W_1 = 1$ $1/2P_1 = 5$

$W = 5$ $P = 32$

Profit = 32

Q. 12 (b)

Knapsack capacity is 5

In O/I knapsack, either you have to consider full weight or no weight, fraction is not allowed.

So, $W_4 = 1$

$W_2 = 3$

Weight = 4

We will get max profit =27

Q. 13 (a)

Heap sort does not use divide and conquer technique.

Q. 14 (c)

Class P can be solved in $O(n^k)$ times using deterministic turing machines.

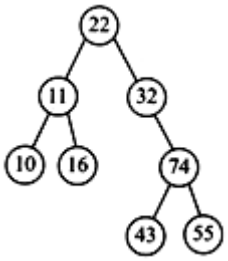
Q. 15 (c)

Satisfiability of Boolean formulas is NP-complete.

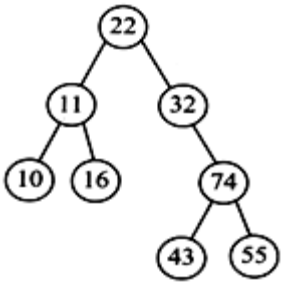
Q. 16 (a)

Whenever a node is detected in a binary search tree replace that node value With largest left child subtree value or smallest right child subtree value, so

The resultant tree will be



OR



Q. 23 (d)

Under the assumption of simple uniform hashing, and key k not already Stored in table is equally likely to hash to any of the m slots. The expected time to search to the end of list T[h(k)] = which has expected length E[nh (k)] = ∝. Thus the expected number of elements examined in an unsuccessful search is ∝, and the total time required is θ(1 + ∝).

Q. 26 (c)

As in linear probing, the initial probe determines the entire sequence, so only m distinct probe sequences are used.

Q. 27 (a)

Since each possible (h1(k), h2(k)) pair yields a district probe sequence. As a Result, the performance of double hashing appears to be very close to the Performance of the "ideal" scheme of uniform hashing.

Q. 29 (c)

Mostly Max heap tree is used but it is not mandatory on can use min heap Tree also instead of max heap tree. It will return a ascending order.

Q. 31 (b)

No polynomial-time algorithm has yet been discovered for an NP-complete Problem, nor has anyone yet been able to prove that no polynomial-time algorithm can exist for anyone of them.

Q. 33 (b)

By definition

$$\lim_{n\to\infty}\frac{10n+7}{3n^2+2n+6}=\frac{10/n+7/n^2}{3+2/n+6/n^2}$$

$= 0/3 = 0.$

Hence $3n^2 + 2n + 6$ is asymptotically bigger than $10n + 7$.

Q. 35 (d)

$1 < n < n \log n < n^2 < n^3 < n!$

Q. 36 (d)

We know $f(n) = O(f(n))$.

Also we know $n^2 < n^3 < n^4$.

Q. 37 (d)

Let $S(m) = T(2^m)$

$S(m) = 2\,S(m/2) + m$ ..... (1)

By yielding $m = \lg n$

$T(2^m) = 2T(2^{m/2}) + m$

for (1) $S(m) = O(m \lg m)$

So $T(n) = T(2^m) = O(m \lg m)$
$= O(\lg n \lg \lg n)$

Q. 38 (a)
a = 9, b = 3, f(n) = n we can apply case 1 of master theorem
$n^{\log}b^{a} = n^{\log_3^9} = \theta(n^2)$.

Q. 43 (a)
With minimum degree t and depth h the number of nodes can be calculated Using the formula $2t^{h-1}$.

Q. 44 (c)
Given a graph G=(V, E) and a distinguished source vertex s, breadth first Search systematically explores the edges of G to "discover" every vertex that is reachable from s.

Q. 45 (b)
Let (u, v) be an arbitrary edge of G, and suppose without loss of generality That d[u] < d[v]. then, υ must be discovered and finished before we finish u (while u is gray), since υ is gray), since υ is on u's adjancy list. If the edge (u, v) is explored first in the direction from u to v, then v is undiscovered (white) until that time, for otherwise we would have explored this edge already in the direction from v to u. thus (u, v) becomes a tree edge. If (u, v) is explored first in the direction from v to u, then (u, v) is a back edge, since u is still gray at the time the edge is first explored.

Q. 46 (d)
Prim's algorithm forms a simple tree, Kruskal's algorithm may form a forest.

Q. 47 (a)
Apply recursive algorithm

Preorder	Root	Left	Right
Inorder	Left	Root	Right
Postorder	Left	Right	Root

Q. 48 (a)
Formula $\frac{(n-k)(n-k+1)}{2}$

Q. 71 (d)
Constant k lies in the interval 0 to 1. Such that it is neither close to 0 nor too close to 1 fractional part of key * k is mapped into interval 0 to m.

Q. 73 (c)
Like merge sort, but unlike insertion sort, heapsort's running time 0(nlogn).
Like insertion sort, but unlike merge sort, heap sort sorts in place: only a constant number of array elements are stored outside the input array at time.

Q. 74 (d)
To perform insertion using open addressing, we successively examine, or probe the hash table until we find an empty slot in which to put the key.

Q. 75 (a)
Because the initial probe determines the entire probe sequence there are only m distinct probe sequences.

Q. 76 (c)
It occurs when pivot divides the array into two parts of size n-1 and 0.

Q. 77 (a)
Since $n = O(n^2)$

Q. 78 (d)
Like Analysis of chaining analysis of open addressing is expressed in terms of load factor $\propto = n/m$. of-course with open addressing, we have at most one element per slot, and thus $n \le m$, which implies
$\propto \le 1$.

Q. 79 (a)

Q. 80 (b)
In practice, we cannot always guarantee that binary search trees are built randomly but there are variations of binary search trees whose worst-case performance on basic operations can guaranteed to be height O(lgn)

Q. 82 (c)
Mostly Max heap tree is used but it is not mandatory on can use min heap tree also instead of max heap tree. It will return a ascending order.

Q. 93 (c)
Level order search is a Breadth first search.

Q. 95 (d)
Let G = (V, E) be a weighted graph with weight function ω. The weight ω(u , v) of the edge (u , v) ∈ E is simply stored with vertex ϑ is u's adjacency list. The adjacency list representation is quite robust in that it can be modified to support many other graph variants.

Q. 97 (d)
There is no forward edge.

Q. 99 (d)
Best case of insertion sort is O(n)