TYPICAL QUESTIONS & ANSWERS

PART I

OBJECTIVE TYPE QUESTIONS

Each Question carries 2 marks.

Q.1 If h is any hashing function and is used to hash n keys in to a table of size m, where n<=m, the expected number of collisions involving a particular key x is :

(A) less than 1. **(B)** less than n.
(C) less than m. **(D)** less than n/2.

Ans:A

Q.2 Let A be an adjacency matrix of a graph G. The ij^{th} entry in the matrix A^K, gives

(A) The number of paths of length K from vertex Vi to vertex Vj.
(B) Shortest path of K edges from vertex Vi to vertex Vj.
(C) Length of a Eulerian path from vertex Vi to vertex Vj.
(D) Length of a Hamiltonian cycle from vertex Vi to vertex Vj.

Ans:B

Q.3 The OS of a computer may periodically collect all the free memory space to form contiguous block of free space. This is called

(A) Concatenation **(B)** Garbage collection
(C) Collision **(D)** Dynamic Memory Allocation

Ans:B

Q.4 What is the following code segment doing?

```
void fn( ){
char c;
cin.get(c);
 if (c != ‘\n’) {
          fn( );
          cout.put(c);
           }
   }
```

(A) The string entered is printed as it is.
(B) The string entered is printed in reverse order.
(C) It will go in an infinite loop.
(D) It will print an empty line.

Ans:B

Q.5 You have to sort a list L consisting of a sorted list followed by a few "random" elements. Which of the following sorting methods would be especially suitable for such a task?

(A) Bubble sort **(B)** Selection sort
(C) Quick sort **(D)** Insertion sort

Ans:D

Q.6 B Trees are generally

(A) very deep and narrow **(B)** very wide and shallow
(C) very deep and very wide **(D)** cannot say

Ans:D

Q.7 A technique for direct search is

(A) Binary Search **(B)** Linear Search
(C) Tree Search **(D)** Hashing

Ans:D

Q.8 If a node having two children is deleted from a binary tree, it is replaced by its

(A) Inorder predecessor **(B)** Inorder successor
(C) Preorder predecessor **(D)** None of the above

Ans:B

Q.9 The searching technique that takes O (1) time to find a data is

(A) Linear Search **(B)** Binary Search
(C) Hashing **(D)** Tree Search

Ans:C

Q.10 A mathematical-model with a collection of operations defined on that model is called

(A) Data Structure **(B)** Abstract Data Type
(C) Primitive Data Type **(D)** Algorithm

Ans:B

Q.11 The number of interchanges required to sort 5, 1, 6, 2 4 in ascending order using Bubble Sort is

(A) 6 **(B)** 5
(C) 7 **(D)** 8

Ans:B

Q.12 The postfix form of the expression $(A+B)*(C*D-E)*F/G$ is

(A) $AB+CD*E-FG/**$ **(B)** $AB+CD*E-F**G/$
(C) $AB+CD*E-*F*G/$ **(D)** $AB+CDE*-*F*G/$

Ans: A

Q.13 The complexity of multiplying two matrices of order m*n and n*p is

(A) mnp **(B)** mp
(C) mn **(D)** np

Ans:A

Q.14 Merging 4 sorted files containing 50, 10, 25 and 15 records will take____time

(A) O (100) **(B)** O (200)
(C) O (175) **(D)** O (125)

Ans:A

Q.15 For an undirected graph with n vertices and e edges, the sum of the degree of each vertex is equal to

(A) 2n **(B)** (2n-1)/2
(C) 2e **(D)** $e^2/2$

Ans:C

Q.16 In worst case Quick Sort has order

(A) O (n log n) **(B)** O $(n^2/2)$
(C) O (log n) **(D)** O $(n^2/4)$

Ans:B

Q.17 A full binary tree with 2n+1 nodes contain

(A) n leaf nodes **(B)** n non-leaf nodes
(C) n-1 leaf nodes **(D)** n-1 non-leaf nodes

Ans:B

Q.18 If a node in a BST has two children, then its inorder predecessor has

(A) no left child **(B)** no right child
(C) two children **(D)** no child

Ans:B

Q.19 A binary tree in which if all its levels except possibly the last, have the maximum number of nodes and all the nodes at the last level appear as far left as possible, is known as

(A) full binary tree. **(B)** AVL tree.
(C) threaded tree. **(D)** complete binary tree.

Ans:A

Q.20 A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as a

(A) queue. **(B)** stack.

(C) tree. **(D)** linked list.

Ans:A

Q.21 What is the postfix form of the following prefix expression -A/B*C$DE
(A) ABCDE$*/- **(B)** A-BCDE$*/-
(C) ABC$ED*/- **(D)** A-BCDE$*/

Ans:A

Q.22 A full binary tree with n leaves contains
(A) n nodes. **(B)** $\log_2 n$ nodes.
(C) $2n-1$ nodes. **(D)** 2^n nodes.

Ans:C

Q.23 A sort which relatively passes through a list to exchange the first element with any element less than it and then repeats with a new first element is called
(A) insertion sort. **(B)** selection sort.
(C) heap sort. **(D)** quick sort.

Ans:D

Q.24 Which of the following sorting algorithms does not have a worst case running time of $O(n^2)$?
(A) Insertion sort **(B)** Merge sort
(C) Quick sort **(D)** Bubble sort

Ans:B

Q.25 An undirected graph G with n vertices and e edges is represented by adjacency list. What is the time required to generate all the connected components?
(A) O (n) **(B)** O (e)
(C) O (e+n) **(D)** $O(e^2)$

Ans:C

Q.26 Consider a linked list of n elements. What is the time taken to insert an element after an element pointed by some pointer?
(A) O (1) **(B)** $O(\log_2 n)$
(C) O (n) **(D)** $O(n \log_2 n)$

Ans:A

Q.27 The smallest element of an array's index is called its
(A) lower bound. **(B)** upper bound.
(C) range. **(D)** extraction.

Ans:A

Q.28 In a circular linked list

(A) components are all linked together in some sequential manner.
(B) there is no beginning and no end.
(C) components are arranged hierarchically.
(D) forward and backward traversal within the list is permitted.

Ans:B

Q.29 A graph with n vertices will definitely have a parallel edge or self loop of the total number of edges are

(A) more than n **(B)** more than n+1
(C) more than (n+1)/2 **(D)** more than n(n-1)/2

Ans: D

Q.30 The minimum number of multiplications and additions required to evaluate the polynomial $P = 4x^3+3x^2-15x+45$ is

(A) 6 & 3 **(B)** 4 & 2
(C) 3 & 3 **(D)** 8 & 3

Ans: C

Q.31 The maximum degree of any vertex in a simple graph with *n* vertices is

(A) $n-1$ **(B)** $n+1$
(C) $2n-1$ **(D)** n

Ans: A

Q.32 The data structure required for Breadth First Traversal on a graph is

(A) queue **(B)** stack
(C) array **(D)** tree

Ans: A

Q.33 The quick sort algorithm exploit _________ design technique

(A) Greedy **(B)** Dynamic programming
(C) Divide and Conquer **(D)** Backtracking

Ans: C

Q.34 The number of different directed trees with 3 nodes are

(A) 2 **(B)** 3
(C) 4 **(D)** 5

Ans: B

Q.35 One can convert a binary tree into its mirror image by traversing it in

(A) inorder **(B)** preorder
(C) postorder **(D)** any order

Ans:C

Q.36 The total number of companions required to merge 4 sorted files containing 15, 3, 9 and 8 records into a single sorted file is

(A) 66 **(B)** 39
(C) 15 **(D)** 3

Ans: 33 (option is not available)

Q.37 In a linked list with n nodes, the time taken to insert an element after an element pointed by some pointer is

(A) 0 (1) **(B)** 0 (log n)
(C) 0 (n) **(D)** 0 (n 1og n)

Ans:A

Q.38 The data structure required to evaluate a postfix expression is

(A) queue **(B)** stack
(C) array **(D)** linked-list

Ans:B

Q.39 The data structure required to check whether an expression contains balanced parenthesis is

(A) Stack **(B)** Queue
(C) Tree **(D)** Array

Ans:A

Q.40 The complexity of searching an element from a set of n elements using Binary search algorithm is

(A) O(n) **(B)** O(log n)
(C) $O(n^2)$ **(D)** O(n log n)

Ans:B

Q.41 The number of leaf nodes in a complete binary tree of depth d is

(A) 2^d **(B)** $2^{d-1}+1$
(C) $2^{d+1}+1$ **(D)** 2^d+1

Ans:A

Q.42 What data structure would you mostly likely see in a nonrecursive implementation of a recursive algorithm?

(A) Stack **(B)** Linked list
(C) Queue **(D)** Trees

Ans:A

Q.43 Which of the following sorting methods would be most suitable for sorting a list which is almost sorted

(A) Bubble Sort **(B)** Insertion Sort
(C) Selection Sort **(D)** Quick Sort

Ans:A

Q.44 A B-tree of minimum degree t can maximum _____ pointers in a node.

(A) t–1 **(B)** 2t–1
(C) 2t **(D)** t

Ans:D

Q.45 The process of accessing data stored in a serial access memory is similar to manipulating data on a

(A) heap **(B)** queue
(C) stack **(D)** binary tree

Ans:C

Q.46 A graph with n vertices will definitely have a parallel edge or self loop if the total number of edges are

(A) greater than n–1 **(B)** less than n(n–1)
(C) greater than n(n–1)/2 **(D)** less than $n^2/2$

Ans:A

Q.47 A BST is traversed in the following order recursively: Right, root, left
The output sequence will be in

(A) Ascending order **(B)** Descending order
(C) Bitomic sequence **(D)** No specific order

Ans:B

Q.48 The pre-order and post order traversal of a Binary Tree generates the same output. The tree can have maximum

(A) Three nodes **(B)** Two nodes
(C) One node **(D)** Any number of nodes

Ans:C

Q.49 The postfix form of A*B+C/D is

(A) *AB/CD+ **(B)** AB*CD/+
(C) A*BC+/D **(D)** ABCD+/*

Ans:B

Q.50 Let the following circular queue can accommodate maximum six elements with the following data

front = 2 rear = 4

queue = _______; L, M, N, ___, ___

What will happen after ADD O operation takes place?

(A) front = 2 rear = 5
queue = ______; L, M, N, O, ___

(B) front = 3 rear = 5
queue = L, M, N, O, ___

(C) front = 3 rear = 4
queue = ______; L, M, N, O, ___

(D) front = 2 rear = 4
queue = L, M, N, O, ___

Ans:A

Q.51 A binary tree of depth "d" is an almost complete binary tree if

(A) Each leaf in the tree is either at level "d" or at level "d–1"
(B) For any node "n" in the tree with a right descendent at level "d" all the left descendents of "n" that are leaves, are also at level "d"
(C) Both **(A)** & **(B)**
(D) None of the above

Ans:C

Q.52 A linear collection of data elements where the linear node is given by means of pointer is called

(A) linked list **(B)** node list
(C) primitive list **(D)** None of these

Ans:A

Q.53 Representation of data structure in memory is known as:

(A) recursive **(B)** abstract data type
(C) storage structure **(D)** file structure

Ans:B

Q.54 If the address of A[1][1] and A[2][1] are 1000 and 1010 respectively and each element occupies 2 bytes then the array has been stored in _________ order.

(A) row major **(B)** column major
(C) matix major **(D)** none of these

Ans:A

Q.55 An adjacency matrix representation of a graph cannot contain information of :
(A) nodes **(B)** edges
(C) direction of edges **(D)** parallel edges

Ans:D

Q.56 Quick sort is also known as
(A) merge sort **(B)** heap sort
(C) bubble sort **(D)** none of these

Ans:D

Q.57 One of the major drawback of B-Tree is the difficulty of traversing the keys sequentially.
(A) True **(B)** False

Ans:A

Q.58 O(N) (linear time) is better than O(1) constant time.
(A) True **(B)** False

Ans:B

Q.59 An ADT is defined to be a mathematical model of a user-defined type along with the collection of all ____________ operations on that model.
(A) Cardinality **(B)** Assignment
(C) Primitive **(D)** Structured

Ans:C

Q.60 An algorithm is made up of two independent time complexities *f (n)* and *g (n).* Then the complexities of the algorithm is in the order of
(A) $f(n) \times g(n)$ **(B)** $Max\ (f(n),g(n))$
(C) $Min\ (f(n),g(n))$ **(D)** $f(n) + g(n)$

Ans:B

Q.61 The goal of hashing is to produce a search that takes
(A) $O(1)$ time **(B)** $O(n^2)$ time
(C) $O(\log n)$ time **(D)** $O(n \log n)$ time

Ans:A

Q.62 The best average behaviour is shown by
(A) Quick Sort **(B)** Merge Sort
(C) Insertion Sort **(D)** Heap Sort

Ans:A

Q.63 What is the postfix form of the following prefix **+ab–cd*

(A) *ab+cd–** **(B)** *abc+*–*
(C) *ab+*cd–* **(D)** *ab+*cd–*

Ans:A

Q.64 Time complexities of three algorithms are given. Which should execute the slowest for large values of N?

(A) $O(N^{1/2})$ **(B)** $O(N)$
(C) $O(\log N)$ **(D)** *None of these*

Ans:B

Q.65 How does an array differ from an ordinary variable?

Ans.
<u>Array Vs. Ordinary Variable</u>
Array is made up of similar data structure that exists in any language. Array is set of similar data types. Array is the collection of similar elements. These similar elements could be all int or all float or all char etc. Array of char is known as string. All elements of the given array must be of same type. Array is finite ordered set of homogeneous elements. The number of elements in the array is pre-specified.
An ordinary variable of a simple data type can store a single element only.
For example: *Ordinary variable*: - int a
Array: - int a[10]

Q.66 What values are automatically assigned to those array elements which are not explicitly initialized?

Ans.
Garbage values are automatically assigned to those array elements that are not explicitly initialized. If garbage class is auto then array is assumed to contain garbage values. If storage class were declared static or external then all the array elements would have a default initial value as zero.

Q.67 A stack is to be implemented using an array. The associated declarations are:

```
 int stack [100];
int top = 0;
```

Give the statement to perform push operation.

Ans.
Let us assume that if stack is empty then its top has value –1.
Top ranges from 0 – 99. Let item be the data to be inserted into stack

```
int stack [100];
int top = 0;
int item ;
If (top == 99)
  Printf (“stack overflow”);
```

```
        Else
                stack[top++] = item;
```

Q.68 Assume that a queue is available for pushing and popping elements. Given an input sequence a, b, c, (c be the first element), give the output sequence of elements if the rightmost element given above is the first to be popped from the queue.

Ans.

A	B	C

Front Rear

c b a

Q.69 A two dimensional array TABLE [6] [8] is stored in row major order with base address 351. What is the address of TABLE [3] [4]?

Ans.

TABLE [6] [8]

Base address 351

Let us assume that TABLE[6] [8] is of type integer.

The formula for row major form to calculate address is

Loc(a[i] [j]) = base (a) + w [n (i –lbr) + (j-lbc)]

where

n	no. of column in array
w	no of bytes per storage location
lbr	lower bound for row index
lbc	lower bound for column index.

Loc(TABLE[3] [4]) = 351 + 2[8(3-0) + (4-0)]

= 351 +2[24+4]

= 351 + 56

=407

Q.70 Which sorting algorithm is best if the list is already sorted? Why?

Ans.

Insertion sort as there is no movement of data if the list is already sorted and complexity is of the order O(N)

Q.71 What is the time complexity of Merge sort and Heap sort algorithms?

Ans.

Time complexity of merge sort is O(N $\log_2$ N)

Time complexity of heap sort is O(n$\log_2$n)

Sort	**Worst Case**	**Average Case**	**Best Case**
Merge Sort	O(nlog n)	O(nlog n)	O(nlog n)
Heap Sort	O(nlog n)	O(nlog n)	O(nlog n)

Q.72 What is the maximum possible number of nodes in a binary tree at level 6?

Ans.
$2^6 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 64$

Q.73 A queue is a,

(A) FIFO (First In First Out) list. **(B)** LIFO (Last In First Out) list.
(C) Ordered array. **(D)** Linear tree.

Ans. (A)

Q.74 Which data structure is needed to convert infix notation to postfix notation?
(A) Branch **(B)** Queue
(C) Tree **(D)** Stack

Ans. (D)

Q.75 Which of the following operations is performed more efficiently by doubly linked list than by singly linked list?
(A) Deleting a node whose location in given
(B) Searching of an unsorted list for a given item
(C) Inverting a node after the node with given location
(D) Traversing a list to process each node

Ans. (A)

Q.76 The extra key inserted at the end of the array is called a,

(A) End key. **(B)** Stop key.
(C) Sentinel. **(D)** Transposition.

Ans. (C)

Q.77 The prefix form of A-B/ (C * D ^ E) is,
(A) -/*^ACBDE **(B)** -ABCD*^DE
(C) -A/B*C^DE **(D)** -A/BC*^DE

Ans. (C)

Q.78 Consider that n elements are to be sorted. What is the worst case time complexity of Bubble sort?

(A) $O(1)$ **(B)** $O(\log_2 n)$
(C) $O(n)$ **(D)** $O(n^2)$

Ans. (D)

Q.79 A characteristic of the data that binary search uses but the linear search ignores is the___________.

(A) Order of the elements of the list.
(B) Length of the list.
(C) Maximum value in list.
(D) Type of elements of the list.

Ans. (A)

Q.80 In Breadth First Search of Graph, which of the following data structure is used?

(A) Stack. **(B)** Queue.
(C) Linked List. **(D)** None of the above.

Ans. (B)

Q.81 The largest element of an array index is called its

(A) lower bound. **(B)** range.
(C) upper bound. **(D)** All of these.

Ans. (C)

Q.82 What is the result of the following operation

Top (Push (S, X))

(A) X **(B)** null
(C) S **(D)** None of these.

Ans. (A)

Q.83 How many nodes in a tree have no ancestors.

(A) 0 **(B)** 1
(C) 2 **(D)** n

Ans. (B)

Q.84 In order to get the contents of a Binary search tree in ascending order, one has to traverse it in

(A) pre-order. **(B)** in-order.
(C) post order. **(D)** not possible.

Ans. (B)

Q.85 Which of the following sorting algorithm is stable

(A) insertion sort. **(B)** bubble sort.
(C) quick sort. **(D)** heap sort.

Ans. (D)

Q.86 The prefix form of an infix expression $p + q - r * t$ is

(A) + pq − *rt . **(B)** − +pqr * t .
(C) − +pq * rt . **(D)** − + * pqrt .

Ans. (C)

Q.87 Which data structure is used for implementing recursion?
(A) Queue. **(B)** Stack.
(C) Arrays. **(D)** List.

Ans. (B)

Q.88 In binary search, average number of comparison required for searching an element in a list if n numbers is
(A) $\log_2 n$. **(B)** n / 2 .
(C) n. **(D)** n – 1.

Ans. (A)

Q.89 In order to get the information stored in a Binary Search Tree in the descending order, one should traverse it in which of the following order?
(A) left, root, right **(B)** root, left, right
(C) right, root, left **(D)** right, left, root

Ans. (C)

Q.90 The equivalent prefix expression for the following infix expression (A+B)-(C+D*E)/F*G is
(A) -+AB*/+C*DEFG **(B)** /-+AB*+C*DEFG
(C) -/+AB*+CDE*FG **(D)** -+AB*/+CDE*FG

Ans. (A)

Q.91 The time required to delete a node x from a doubly linked list having n nodes is
(A) O (n) **(B)** O (log n)
(C) O (1) **(D)** O (n log n)

Ans. (C)

Q.92 Ackerman's function is defined on the non-negative integers as follows

$$a(m,n) = n+1 \text{ if } m=0$$
$$= a(m-1, 1) \text{ if } m \neq 0, n=0$$
$$= a(m-1, a(m, n-1)) \text{ if } m \neq 0, n \neq 0$$

The value of a (1, 3) is
(A) 4. **(B)** 5.
(C) 6. **(D)** 7.

Ans. (B)

Q.93 The result of evaluating the postfix expression 5, 4, 6, +, *, 4, 9, 3, /, +, * is

(A) 600. **(B) 350**.
(C) 650. **(D)** 588.

Ans. (B)

Q.94 The worst case of quick sort has order

(A) $O(n^2)$ **(B)** $O(n)$
(C) $O(n \log_2 n)$ **(D)** $O(\log_2 n)$

Ans. (A)

Q.95 For an undirected graph G with n vertices and e edges, the sum of the degrees of each vertex is

(A) ne **(B)** 2n
(C) 2e **(D)** e^n

Ans. (C)

Q.96 The time required to delete a node x from a doubly linked list having n nodes is

(A) O (n) **(B)** O (log n)
(C) O (1) **(D)** O (n log n)

Ans. (C)

PART II

DESCRIPTIVES

Q.1 What is an algorithm? What are the characteristics of a good algorithm? **(4)**

Ans:

An **algorithm** is “a step-by-step procedure for accomplishing some task" An algorithm can be given in many ways. For example, it can be written down in English (or French, or any other "natural" language). However, we are interested in algorithms which have been precisely specified using an appropriate mathematical formalism--such as a programming language.
Every algorithm should have the following five characteristics:
1.Input: The algorithm should take zero or more input.
2. Output: The algorithm should produce one or more outputs.
3. Definiteness: Each and every step of algorithm should be defined unambiguously.
4. Effectiveness: A human should be able to calculate the values involved in the procedure of the algorithm using paper and pencil.
5. Termination: An algorithm must terminate after a finite number of steps.

Q.2 How do you find the complexity of an algorithm? What is the relation between the time and space complexities of an algorithm? Justify your answer with an example. **(5)**

Ans:

Complexity of an algorithm is the measure of analysis of algorithm. Analyzing an algorithm means predicting the resources that the algorithm requires such as memory, communication bandwidth, logic gates and time. Most often it is computational time that is measured for finding a more suitable algorithm. This is known as time complexity of the algorithm. The running time of a program is described as a function of the size of its input. On a particular input, it is traditionally measured as the number of primitive operations or steps executed.

The analysis of algorithm focuses on time complexity and space complexity. As compared to time analysis, the analysis of space requirement for an algorithm is generally easier, but wherever necessary, both the techniques are used. The space is referred to as storage required in addition to the space required storing the input data. The amount of memory needed by program to run to completion is referred to as space complexity. For an algorithm, time complexity depends upon the size of the input, thus, it is a function of input size ‘n’. So the amount of time needed by an algorithm to run to its completion is referred as time complexity.

The best algorithm to solve a given problem is one that requires less memory and takes less time to complete its execution. But in practice it is not always possible to achieve both of these objectives. There may be more than one approach to solve a same problem. One such approach may require more space but takes less time to complete its execution while the other approach requires less space but more time to complete its execution. Thus we may have to compromise one to improve the other. That is, we may be able to reduce space requirement by increasing running time or reducing running time by allocating more memory space. This situation where we compromise one to better the other is known as Time-space tradeoff.

Q.3 Compare two functions n^2 and $2^n/4$ for various values of n. Determine when second becomes larger than first. **(5)**

n	n^2	$2^n/4$
1	1	0.5
2	4	1
3	9	2
4	16	4
5	25	8
6	36	16
7	49	32
8	64	64
9	81	128
10	100	256

When n=8, the value of n^2 and $2^n/4$ is same. Before that $n^2 > 2^n/4$, and after this value $n^2 < 2^n/4$.

Q.4 Explain an efficient way of storing a sparse matrix in memory. Write a module to find the transpose of a sparse matrix stored in this way. **(10)**

Ans:

A matrix in which number of zero entries are much higher than the number of non zero entries is called *sparse matrix.* The natural method of representing matrices in memory as two-dimensional arrays may not be suitable for sparse matrices. One may save space by storing only nonzero entries. For example matrix A (3*3 matrix) represented below

$$\begin{array}{ccc} 0 & 2 & 0 \\ 5 & 0 & 0 \\ 0 & 6 & 9 \end{array}$$

can be written in sparse matrix form as:

$$\begin{array}{ccc} 3 & 3 & 4 \\ 0 & 1 & 2 \\ 1 & 0 & 5 \\ 2 & 2 & 6 \\ 2 & 3 & 9 \end{array}$$

where first row represent the dimension of matrix and last column tells the number of non zero values; second row onwards it is giving the position and value of non zero number.

A function to find transpose of a sparse matrix is:

```
void  transpose(x,r,y)
int x[3][3],y[3][3],r;
{
     int i,j,k,m,n,t,p,q,col;
     m=x[0][0];
      n=x[0][1];
      t=x[0][2];
    y[0][0]=n;
    y[0][1]=m;
    y[0][2]=t;
  if(t>0)
{
     q=1;
      for (col=0;col<=n;col++)
         for(p=1;p<=t;p++)
           if(x[p][1]==col)
           {
                y[q][0]=x[p][1];
      y[q][1]=x[p][0];
      y[q][2]=x[p][2];
     q++;
  }
}
return;
}
```

Q.5 Explain an efficient way of storing two symmetric matrices of the same order in memory. **(4)**

Ans:

A n-square matrix array is said to be symmetric if a[j][k]=a[k][j] for all j and k. For a symmetric matrix, we need to store elements which lie on and below the diagonal or those on and above the diagonal. Two symmetric matrix of A and B of same dimension can be stored in an n*(n+1) array C where c[j][k]=a[j][k] when j≥k but c[j][k-1]=b[j][k-1] when j<k.

$$\begin{pmatrix} a_{11} & b_{11} & b_{12} & b_{13} & \dots & b_{1,n-1} & b_{1n} \\ a_{21} & a_{22} & b_{22} & b_{23} & \dots & b_{2,n-1} & b_{2n} \\ a_{31} & a_{32} & a_{32} & b_{33} & \dots & b_{3,n-1} & b_{3n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \dots & a_{n,n} & b_{nn} \end{pmatrix}$$

Q.6 Write an algorithm to evaluate a postfix expression. Execute your algorithm using the following postfix expression as your input : a b + c d +*f ↑. **(7)**

Ans:

To evaluate a postfix expression:

```
clear the stack
symb  = next input character
while(not end of input)
{
  if (symb is an operand)
     push onto the stack;
  else
  {
     pop two operands from the stack;
     result=op1 symb op2;
     push result onto the stack;
     }
     symb = next input character;
}
return (pop (stack));
```

*The given input as postfix expression is:- ab+cd+*f^*

Symb	Stack	Evaluation
a	a	
b	a b	
+	pop a and b	(a + b)
	Push (a + b)	
c	(a + b) c	
d	(a + b) c d	
+	pop c and d	(c + d)
	Push (c + d)	
*	pop (a + b) and (c + d)	(a + b) * (c + d)
f	(a + b) * (c + d) f	
^	pop (a + b) * (c + d) and f	(a + b) * (c + d) ^ f

The result of evaluation is ***((a + b) * (c + d)) ^ f***

Q.7 What are circular queues? Write down routines for inserting and deleting elements from a circular queue implemented using arrays. **(7)**

Ans:

Circular queue: Static queues have a very big drawback that once the queue is FULL, even though we delete few elements from the "front" and relieve some occupied space, we are not able to add anymore elements as the "rear" has already reached the Queue's rear most position.

The solution lies in a queue in which the moment "rear" reaches its end; the "first" element will become the queue's new "rear". This type of queue is known as circular queue having a structure like this in which, once the Queue is full the "First" element of the Queue becomes the "Rear" most element, if and only if the "Front" has moved forward;

Circular Queue using array:-

insert(queue,n,front,rear,item)

```
This procedure inserts an element item into a queue.
     1. If front = 1 and rear = n, or if front = rear
     + 1, then:
       Write: overflow, and return
     2. [find new value of rear]
     If front = NULL , then : [queue initially empty.]
       Set front = 1 and rear=1.
     Else if rear = n, then:
       Set rear=1.
     Else:
       Set  rear = rear + 1.
     [end of structure.]
     3. Set queue[rear] = item. [this inserts new
     element.]
     4.Return .
```

delete(queue,n,front,rear,item)

```
This procedure deletes an element from a queue and
assigns it to the variable item.
      1.   [queue already empty?]
      If front = NULL , then:
        Write: underflow, and return.
      2.   Set item = queue[front].
      3.   [find new value of front]
      If front = rear , then : [queue has only one
      element to start].
        Set front = NULL and rear = NULL.
      Else if front = n, then:
        Set front=1.
      Else:
           Set  front = front + 1.
      [end of structure.]
      4.   Return .
```

Q.8 Given a set of input representing the nodes of a binary tree, write a non recursive algorithm that must be able to output the three traversal orders. Write an algorithm for checking validity of the input, i.e., the program must know if the input is disjoint, duplicated and has a loop. **(10)**

Ans:

Pre-order Traversal

The process of doing a pre-order traversal iteratively then has the following steps(assuming that a stack is available to hold pointers to the appropriate nodes):

```
1. push NULL onto the stack
2. start at the root and begin execution
3. write the information in the node
4. if the pointer to the right is not NULL push
the pointer onto the stack
```

```
5. if the pointer to the left is not NULL move
the pointer to the node on the left
6. if the pointer to the left is NULL pop the
stack
7. repeat steps 3 to 7 until no nodes remain
```

In-order Traversal

This process when implemented iteratively also requires a stack and a Boolean to prevent the execution from traversing any portion of a tree twice. The general process of in-order traversal follows the following steps:

```
1.   push a NULL onto the stack.
2.   beginning with the root, proceed down the
left side of the tree. As long as there is
another left node following the current node,
push the pointer to the current node onto the
stack and move to the left node which follows
it.
3.   when there are no more left nodes following
the current node, process the current node.
4.   check to see if the last left node has a
right node(it obviously doesn't have a left
node).
5.   if there is a right node then there is
subtree to the right. This should be processed
in following steps 2 to 4.
6.   if there is no right node then pop the last
node from the stack (back up one node). Process
this node and since this node has a left subtree
that has already been processed, move to the
right of the node and begin looking for a left
and right subtree again.
```

Post-order Traversal

This can be done both iteratively and recursively. The iterative solution would require a modification of the in-order traversal algorithm.

Q.9 Two linked lists contain information of the same type in ascending order. Write a module to merge them to a single linked list that is sorted. **(7)**

Ans:

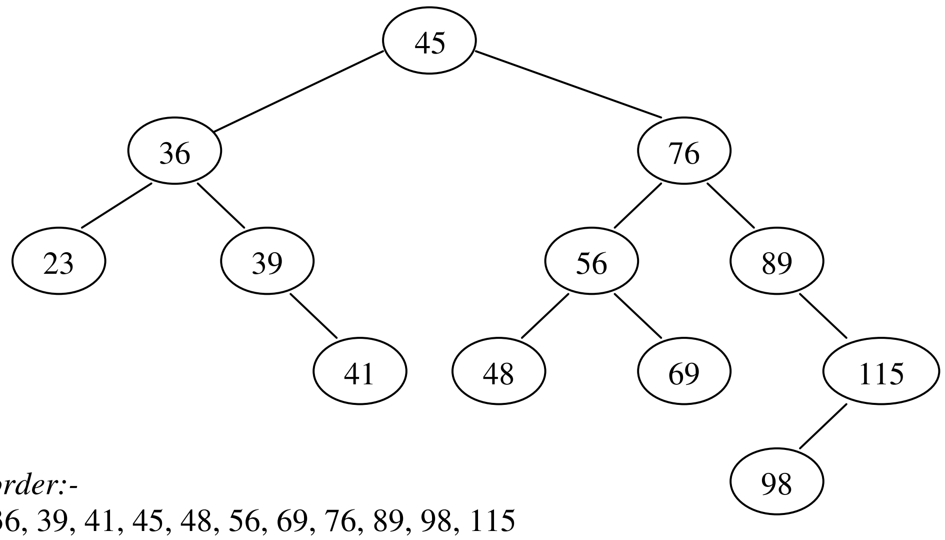
```
merge(struct node *p, struct node *q, struct **s)
{
 struct node *z;
 z = NULL;
 if((x= =NULL) && (y = =NULL))
    return;
 while(x!=NULL && y!=NULL)
 {
    if(*s= =NULL)
    {
    *s=(struct link *)malloc(sizeof(struct node *z));
       z=*s;
    }
    else
    {
z-->link=(struct link *)malloc(sizeof(struct node *));
       z=z-->link;
    }
    if(x-->data < y-->data)
    {
       z-->data=x-->data;
       x=x-->link;
    }
    else if(x-->exp > y-->exp)
    {
       z-->data=y-->data;
       y=y-->link;
    }
    else if(x-->data= =y-->data)
    {
       z-->data=y-->data;
       x=x-->link;
       y=y-->link;
    }
 }
 while(x!=NULL)
 {
 z→link = struct link *malloc(sizeof(struct node *));
    z=z→link;
    z-->data=x-->data;
    x=x-->link;
 }
 while(y!=NULL)
 {
 z→link = struct link *malloc(sizeof(struct node *));
    z=z→link;
    z-->data=y-->data;
    y=y-->link;
 }
 z-->link=NULL;
}
```

Q.10 What is a Binary Search Tree (BST)? Make a BST for the following sequence of numbers.
45, 36, 76, 23, 89, 115, 98, 39, 41, 56, 69, 48
Traverse the tree in Preorder, Inorder and postorder. **(8)**

Ans:

A binary search tree B is a binary tree each node of which satisfies the following conditions:

1. The value of the left-subtree of 'x' is less than the value at 'x'
2. The value of the right-subtree of 'x' is greater than value at 'x'
3. the left-subtree and right-subtree of binary search tree are again binary search tree.



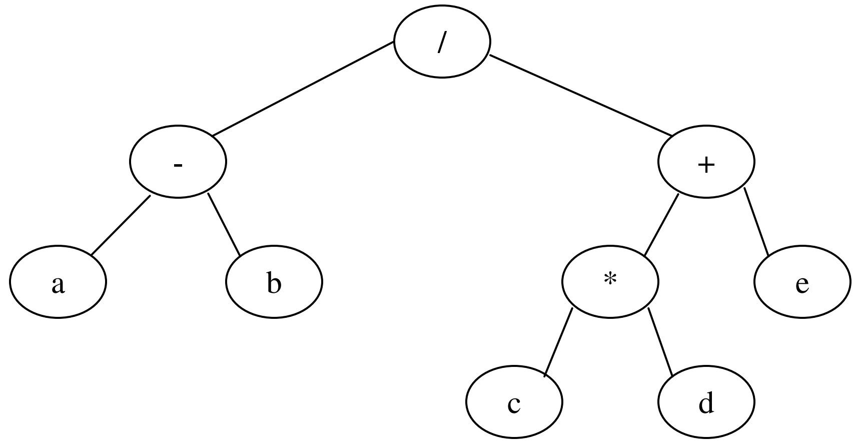
Preorder:-
23, 36, 39, 41, 45, 48, 56, 69, 76, 89, 98, 115
Inorder:-
45, 36, 23, 39, 41, 76, 56, 48, 69, 89, 115, 98
Postorder:-
23, 41, 39, 36, 48, 69, 56, 98, 115, 89, 76

Q.11 What are expression trees? Represent the following expression using a tree. Comment on the result that you get when this tree is traversed in Preorder, Inorder and postorder. (a-b) / ((c*d)+e) **(6)**

Ans:

The leaves of an expression tree are operands, such as constants or variable names, and the other nodes contain operators. This particular tree happens to be binary, because all of the operations are binary, and although this is the simplest case, it is possible for nodes to have more than two children. It is also possible for a node to have only one child, as is the case with the unary minus operator. We can evaluate an expression tree, T, by applying the operator at the root to the values obtained by recursively evaluating the left and right subtrees.

The expression tree for the expression: ***(a – b) / ((c * d) + e))***



The traversal of the above expression tree gives the following result:-
Preorder:- **(/ - a b + * c d e)**
This expression is same as the "prefix notation" of the original expression.
Inorder:- **(a – b) / ((c * d) + e)**
Thus the inorder traversal gives the actual expression.
Postorder:- **(a b – c d * e + /)**
Thus the postorder traversal of this gives us the "posfix notation" or the "Reverse Polish notation" of the original expression.

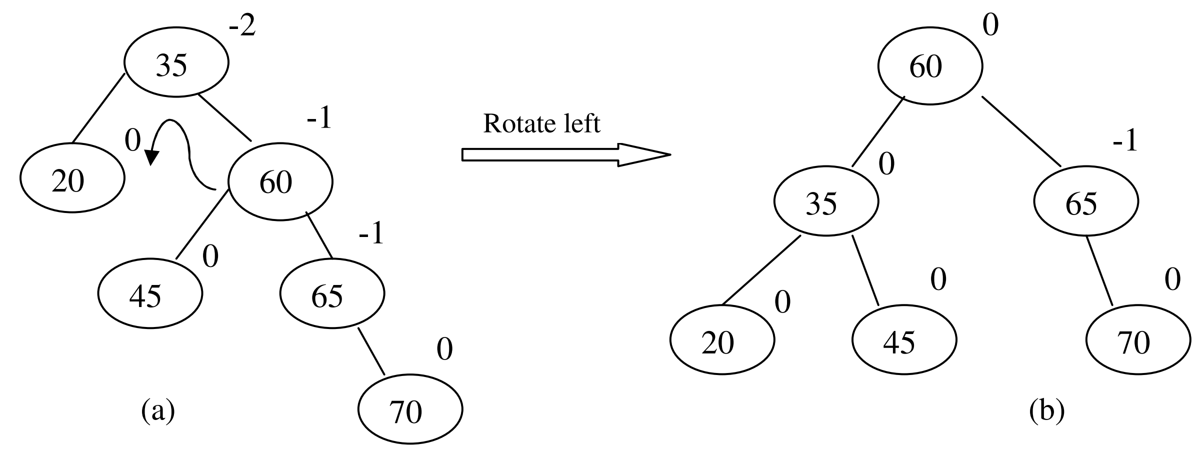
Q.12 How do you rotate a Binary Tree? Explain right and left rotations with the help of an example. **(8)**

Ans:
Rotations in the tree:

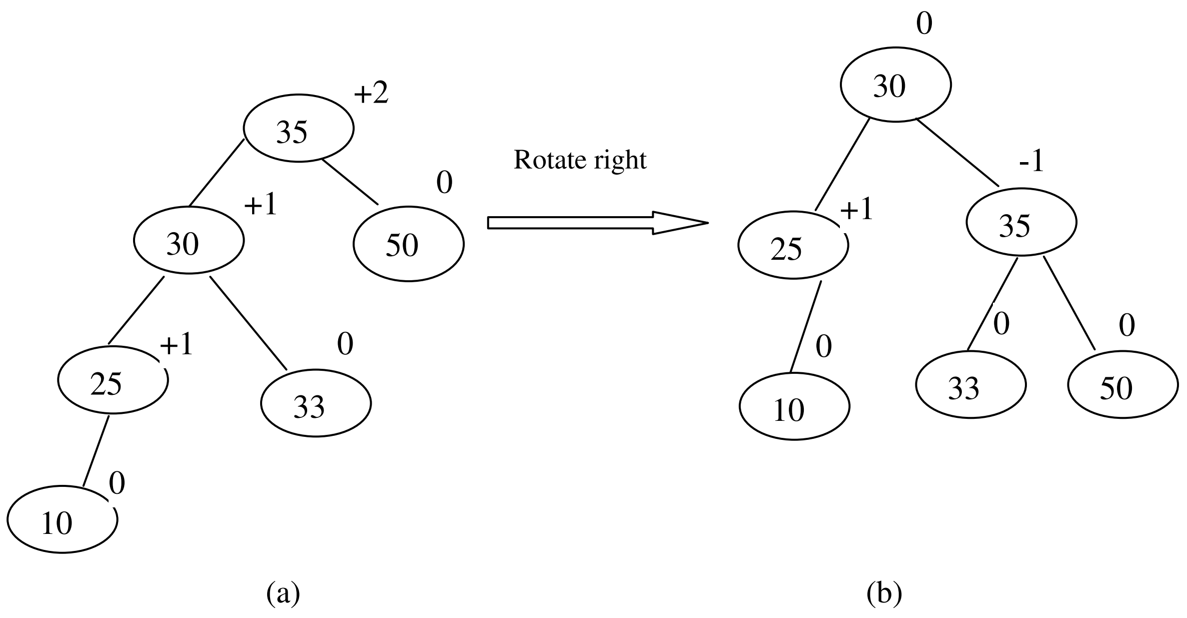
If after inserting a node in a Binary search tree, the balancing factor (height of left subtree - height of right subtree) of each node remains 0, 1 and -1 then there is no need of modification of original tree. If balancing factor of any node is either +2 or -2, the tree becomes unbalanced. It can be balanced by using left rotation or right rotation or a combination of both.

For example

1. In the following tree, after inserting node 70 , the balance factor of tree at the root node (2) so the tree is rotated left to have a height balanced tree shown by (a) to (b):



2. In the following tree, after inserting 10 the balance factor of tree at root node is +2 so the tree is rotated right to have height balanced tree as shown by(a) to (b) below:



Q.13 Taking a suitable example explains how a general tree can be represented as a Binary Tree. **(6)**

Ans:

Conversion of general trees to binary trees:

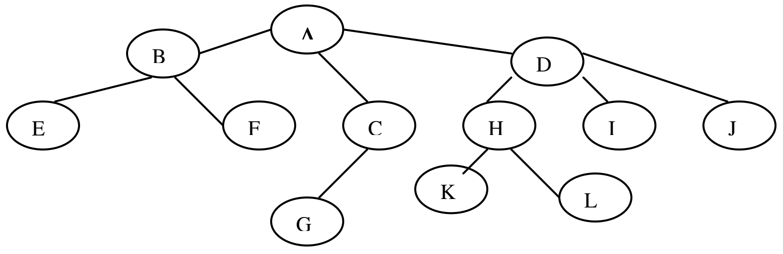
A general tree can be converted into an equivalent binary tree. This conversion process is called the natural correspondence between general and binary trees.

The algorithm is as follows:

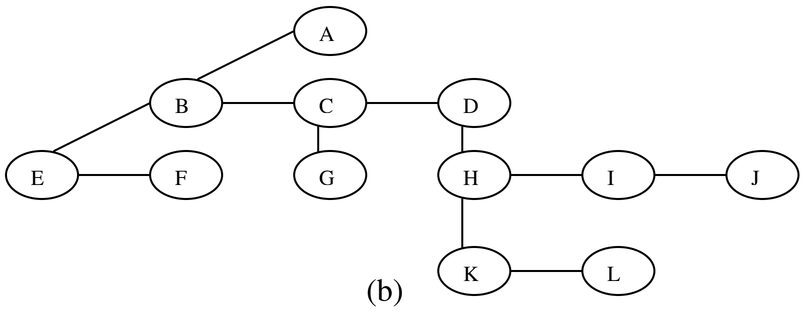
(a) Insert edges connecting siblings from left to right at the same level.

(b) Delete all edges of a parent to its children except to its left most offspring.

(c) Rotate the resultant tree 45^0 to mark clearly left and right subtrees.
A general tree shown in figure (a) converted into a binary tree shown in (b)

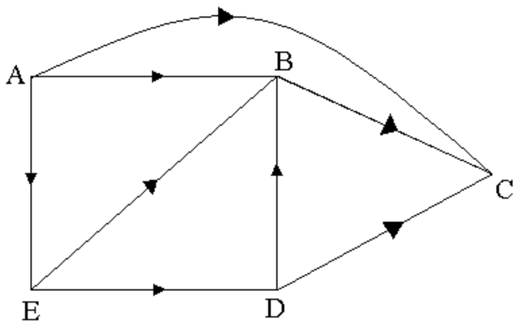


(a)

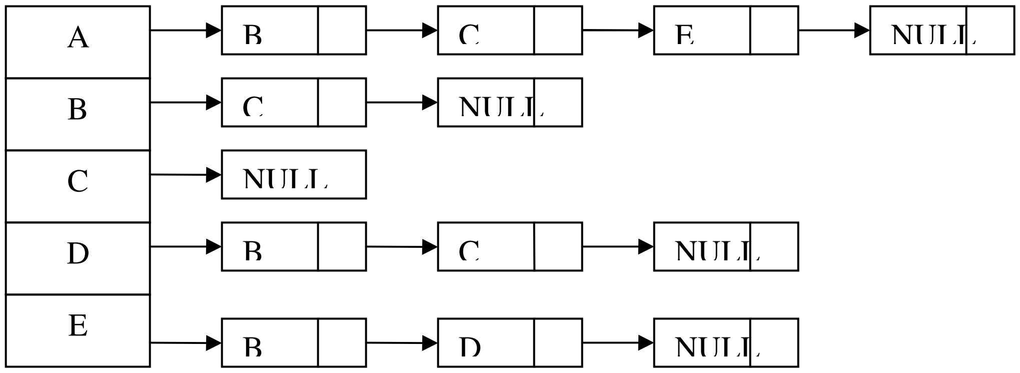


(b)

Q.14 What are the different ways of representing a graph? Represent the following graph using those ways. **(6)**



Ans:
The different ways of representing a graph is:
***Adjacency list representation**:* This representation of graph consists of an array Adj of |V| lists, one for each vertex in V. For each uεV, the adjacency list Adj[u] contains all the vertices v such that there is an edge (u,v)ε E that is Adj[u] consists of all the vertices adjacent to u in G. The vertices in each adjacency list are stored in an arbitrary order. The adjacency list representation of given graph is depicted in the following figure:



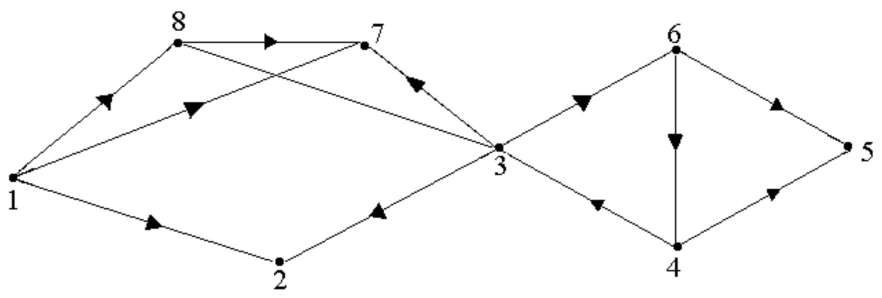
Adjacency Matrix representation: This representation consists of |V|*|V| matrix A= (a_{ij}) such that

$$a_{ij} = \begin{cases} 1 \text{ if}(i,j) \varepsilon E \\ 0 \text{ otherwise} \end{cases}$$

The adjacency matrix representation of given graph is given below:

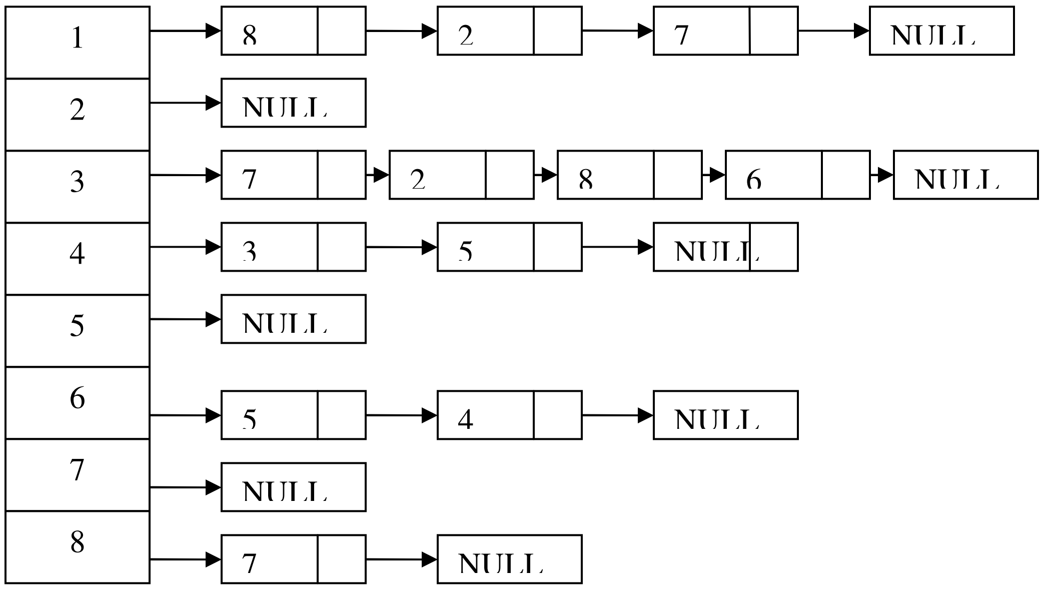
$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 0 & 1 \\ B & 0 & 0 & 1 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 \\ D & 0 & 1 & 1 & 0 & 0 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

Q.15 Show the result of running BFS and DFS on the directed graph given below using vertex 3 as source. Show the status of the data structure used at each stage. **(8)**



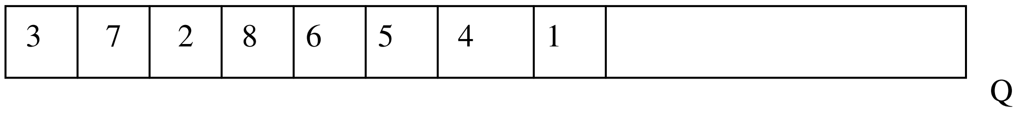
Ans:

Depth first search (DFS):



Depth first search (DFS) starting from vertex 3 as source and traversing above adjacency list one by one, we get following result:
3-7-2-8-6-5-4-1

Breath First Search(BFS):



Elements are inserted from rear and deleted from front. Before removal of an element, insert its element in the queue. So result of BFS in the given graph is:
3-7-2-8-6-5-4-1

Q.16 Write an algorithm for finding solution to the Tower's of Hanoi problem. Explain the working of your algorithm (with 4 disks) with diagrams. **(5)**

Ans:

```
void Hanoi(int  n, char initial, char temp, char
final)
{      if(n==1)
       { printf("move disk 1 from peg %c to
peg%c\n",initial,final);
 return;
        }
Hanoi(n-1,initial, final, temp);
printf("move disk %d from peg %c to peg%c\n", n,
initial, final);
Hanoi(n-1,temp,initial,final);
}
```

For n=4, let A, B and C denotes the initial, temp and final peg. Using above algorithm, the recursive solution for n=4 disks consists of the following 15 moves:

$A \longrightarrow B$ $A \longrightarrow C$ $B \longrightarrow C$ $A \longrightarrow B$

$C \longrightarrow A$ $C \longrightarrow B$ $A \longrightarrow B$ $A \longrightarrow C$

$B \longrightarrow C$ $B \longrightarrow A$ $C \longrightarrow A$ $B \longrightarrow C$

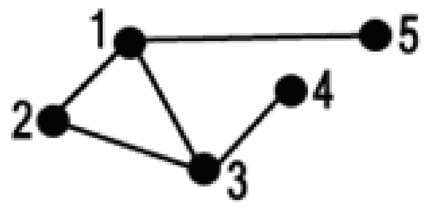
$A \longrightarrow B$ $A \longrightarrow C$ $B \longrightarrow C$

Q.17 Reverse the order of elements on a stack S
(i) using two additional stacks.
(ii) using one additional queue.
(9)

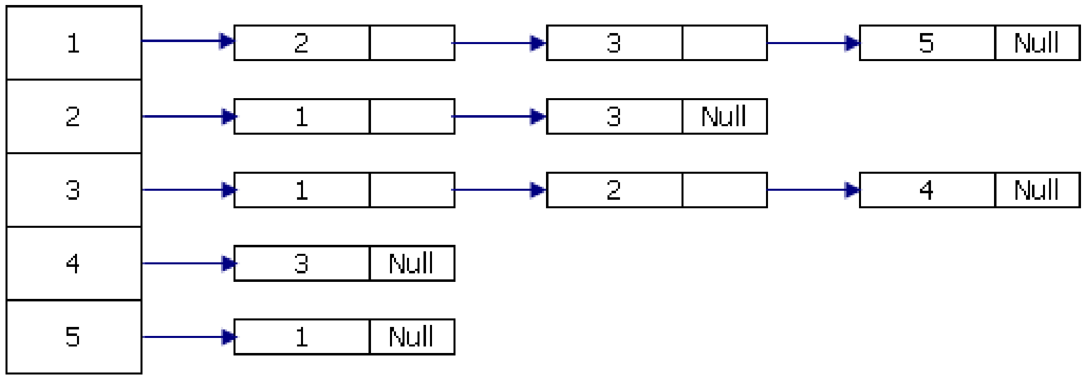
Ans*:* *Let S be the stack having n elements. We need to reverse the elements of S*

(i) using two additional stack S1 and S2
```
while not empty (S)
{       C=pop(S);
         push(S1,C);
}
While not empty(S1)
{     C=pop(S1);
         push(S2,C);
}
While not empty(S3)
{     C=pop(S3);
         push(S,C);
}
```
(ii) using one additional queue Q
```
while not empty (S)
{       C=pop(S);
         enque(Q,C);
}
While not empty(Q)
{     C=deque(Q);
         push(S,C);
}
```

Q.18 Explain the representations of graph. Represent the given graph using any two methods **(8)**



Ans:
The different ways of representing a graph is:
Adjacency list representation*:* This representation of graph consists of an array Adj of |V| lists, one for each vertex in V. For each uεV, the adjacency list Adj[u] contains all the vertices v such that there is an edge (u,v)ε E that is Adj[u] consists of all the vertices adjacent to u in G. The vertices in each adjacency list are stored in an arbitrary order. The adjacency list representation of given graph is depicted in the following figure:



Adjacency Matrix representation: This representation consists of |V|*|V| matrix A=(a_{ij}) such that

$$a_{ij} = \begin{cases} 1 \text{ if}(i,j)\varepsilon\ E \\ 0 \text{ otherwise} \end{cases}$$

The adjacency matrix representation of given graph is given below

$$\begin{array}{c|ccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \hline \mathbf{1} & 0 & 1 & 1 & 0 & 1 \\ \mathbf{2} & 1 & 0 & 1 & 0 & 0 \\ \mathbf{3} & 1 & 1 & 0 & 1 & 0 \\ \mathbf{4} & 0 & 0 & 1 & 0 & 0 \\ \mathbf{5} & 1 & 0 & 0 & 0 & 0 \end{array}$$

Q.19 Write short notes on any **FOUR**:-

(i) B Tree.
(ii) Time Complexity, Big O notation.
(iii) Merge Sort.
(iv) Threaded Binary Tree.
(v) Depth First Traversal.

(3.5 x 4 = 14)

Ans:

(i) **B Tree**:

Unlike a binary-tree, each node of a B-tree may have a variable number of keys and children. The keys are stored in non-decreasing order. Each key has an associated child that is the root of a subtree containing all nodes with keys less than or equal to the key but greater than the preceeding key. A node also has an additional rightmost child that is the root for a subtree containing all keys greater than any keys in the node.

A B-tree has a minimum number of allowable children for each node known as the *minimization factor*. If *t* is this *minimization factor*, every node must have at least *t - 1* keys. Under certain circumstances, the root node is allowed to violate this property by having fewer than *t - 1* keys. Every node may have at most *2t - 1* keys or, equivalently, *2t* children.

For *n* greater than or equal to one, the height of an *n*-key b-tree T of height *h* with a minimum degree *t* greater than or equal to 2, $h \leq \log_t (n+1/2)$

(ii) **Time Complexity, Big O notation**: The amount of time needed by an algorithm to run to its completion is referred as time complexity. The asymptotic running time of an algorithm is defined in terms of functions. The upper bound for the function 'f' is provided by the big oh notation (O). Considering 'g' to be a function from the non-negative integers to the positive real numbers, we define O(g) as the set of function f such that for some real constant c>0 and some non negative integers constant n_0 , $f(n) \leq cg(n)$ for all $n \geq n_0$. Mathematically,$O(g(n))=\{f(n)$: there exists positive constants such that $0 \leq f$ $f(n) \leq cg(n)$ for all n, $n \geq n_0\}$, we say "f is oh of g"

(iii) **Merge sort:** Merge sort is a sorting algorithm that uses the idea of divide and conquers. This algorithm divides the array into two halves, sorts them separately and then merges them. This procedure is recursive, with the base criterion-the number of elements in the array is not more than 1. Suppose variable low and high represents the index of the first and last element of the array respectively, the merge sort can be defined recursively as

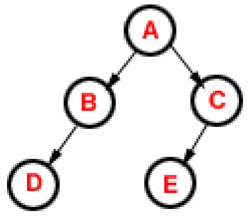
```
If(low<high) then
Divide the list into two halves
Mergesort the left half
Mergesort the right half
Mergesort the two sorted halves into one sorted list
Endif.
```

(iv) Threaded Binary Tree : If a node in a binary tree is not having left or right child or it is a leaf node then that absence of child node is represented by the null pointers. The space occupied by these null entries can be utilized to store some kind of valuable information. One possible way to utilize this space is to have special pointer that point to nodes higher in the tree that is ancestors. These special pointers are called threads and the binary tree having such pointers is called threaded binary tree. There are many ways to thread a binary tree each of these ways either correspond either in-order or pre-order traversal of T.A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. By doing this threading we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

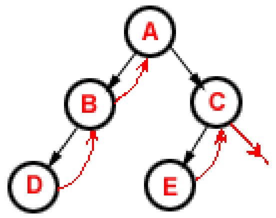
The node structure for a threaded binary tree varies a bit and its like this

```
struct NODE
{
 struct NODE *leftchild;
 int node_value;
 struct NODE *rightchild;
 struct NODE *thread;
}
```

Let's make the Threaded Binary tree out of a normal binary tree...



The INORDER traversal for the above tree is -- D B A E C. So, the respective Threaded Binary tree will be --



B has no right child and its inorder successor is A and so a thread has been made in between them. Similarly, for D and E. C has no right child but it has no inorder successor even, so it has a hanging thread.

(v) depth-first traversal :A depth-first traversal of a tree visits a node and then recursively visits the subtrees of that node. Similarly, depth-first traversal of a graph visits a vertex and then recursively visits all the vertices adjacent to that node. The catch is that the graph may contain cycles, but the traversal must visit every vertex at most once. The solution to the problem is to keep track of the nodes that have been visited, so that the traversal does not suffer the fate of infinite recursion.

Q.20 Define the term array. How are two-dimensional arrays represented in memory? Explain how address of an element is calculated in a two dimensional array. **(8)**

Ans:

An *array* is a way to reference a series of memory locations using the same name. Each memory location is represented by an array element. An *array element* is similar to one variable except it is identified by an index value instead of a name. An *index* value is a number used to identify an array element.

Declaring a two dimensional array- A two dimensional array is declared similar to the way we declare a one-dimensional array except that we specify the number of elements in both dimensions. For example,

int grades[3][4];

The first bracket ([3]) tells the compiler that we are declaring 3 pointers, each pointing to an array. We are not talking about a pointer variable or pointer array. Instead, we are saying that each element of the first dimension of a two dimensional array reference a corresponding second dimension. In this example, all the arrays pointed to by the first index are of the same size. The second index can be of variable size. For example, the previous statement declares a two-dimensional array where there are 3 elements in the first dimension and 4 elements in the second dimension.

Two-dimensional array is represented in memory in following two ways:

1. Row major representation: To achieve this linear representation, the first row of the array is stored in the first memory locations reserved for the array, then the second row and so on.
2. Column major representation: Here all the elements of the column are stored next to one another.

In row major representation, the address is calculated in a two dimensional array as per the following formula.The address of a[i][j]=base(a)+(i*m+ j)*size where base(a) is the address of a[0][0], m is second dimension of array a and size represent size of the data type.

Q.21 An, array, A contains n unique integers from the range x to y (x and y inclusive where n=y-x). That is, there is one member that is not in A. Design an O(n) time algorithm for finding that number. **(8)**

Ans:

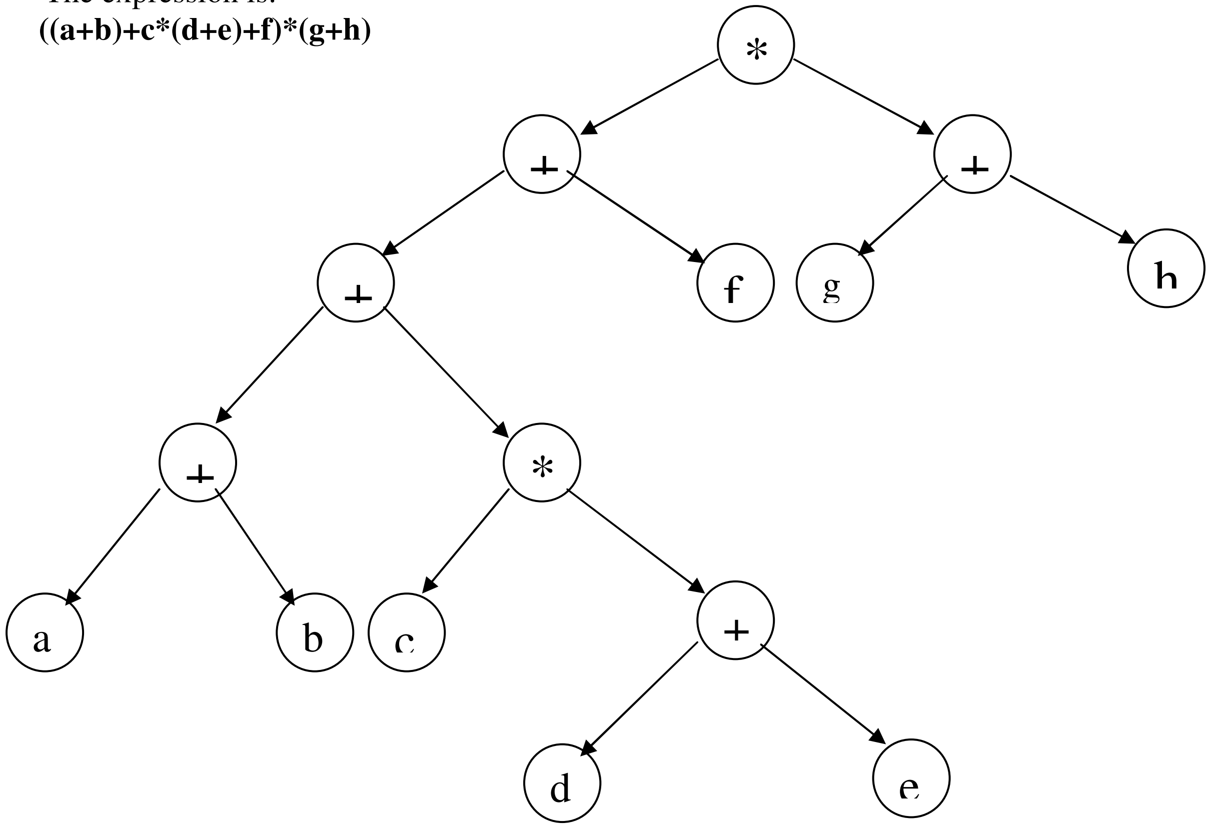
The algorithm for finding the number that is not array A where n contains n unique (n = x - y):

```
find(int A[],n,x,y)
{
                    int i,missing_num,S[n];
     for(i=0, i<n, i++)S[i] = -999;
     for(i=0, i<n, i++)S[A[i]-x]=A[i];
     for(i=0, i<n, i++)
     {if(S[i] == -999)
        {
           missing_num = i + x;
           break;
         }
       }
}
```

Q.22 Draw the expression tree of the following infix expression. Convert it in to Prefix and Postfix expressions.

$((a+b)+c*(d+e)+f)*(g+h)$ **(9)**

Ans:
The expression is:
((a+b)+c*(d+e)+f)*(g+h)



The postfix expression is:
((a+b)+c*(d+e)+f)*(g+h)
= ((ab+)+c*(de+)+f)*(gh+)
= ((ab+)+(cde+*)+f)*(gh+)
= ((ab+cde+*+)+f)*(gh+)
= (ab+cde+*+f+)*(gh+)
=(ab+cde+*+f+gh+*)

The prefix expression is:
((a+b)+c*(d+e)+f)*(g+h)
= ((+ab)+c*(+de)+f)*(+gh)
= ((+ab)+(*c+de)+f)*(+gh)
= ((++ab*c+de)+f)*(+gh)
= (+++ab*c+def)*(+gh)
= (*+++ab*c+def+gh)

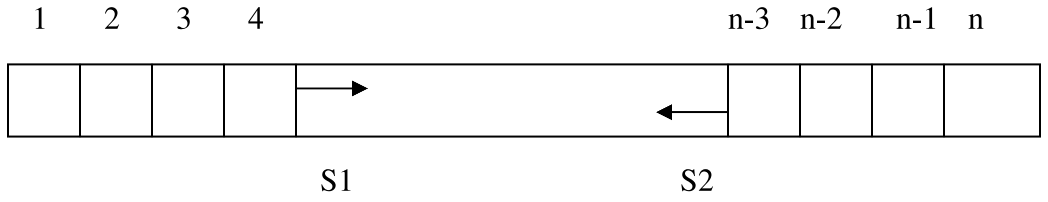
Q.23 Implement a Queue using a singly linked list L. The operations INSERT and DELETE should still take O (1) time. **(6)**

Ans*:*
Implementation of queue using a singly linked list:
While implementing a queue as a single liked list, a queue q consists of a list and two pointers, q.front and q.rear.

inserting an element is as follows:
```
insert(q,x)
{
  p=getnode();
  info(p) = x;
  next(p) = null;
  if(q.rear == null)
     q.front = p;
  else
     next(q.rear) = p;
  q.rear = p;
}
```
deletion is as follows:
```
remove(q)
{
  if(empty(q))
  {
     printf("queue overflow");
     exit(1);
  }
  p=q.front;
  x=info(p);
  q.front=next(p);
  if(q.front == null)
  q.rear = null;
  freenode(p);
  return(x);
}
```

Q.24 Explain how to implement two stacks in one array A[1..n] in such a way that neither stack overflows unless the total number of elements in both stacks together is n. The PUSH and POP operations should run in O(1) time. **(10)**

Ans:
Two stacks s1 and s2 can be implemented in one array A[1,2,…,N] as shown in the following figure



We define A[1] as the bottom of stack S1 and let S1 "grow" to the right and we define A[n] as the bottom of the stack S2 and S2 "grow" to the left. In this case, overflow will occur only S1 and S2 together have more than n elements. This technique will usually decrease the number of times overflow occurs. There will separate push1, push2, pop1 and pop2 functions to be defined separately for two stacks S1 and S2.

Q.25 Two Binary Trees are similar if they are both empty or if they are both non-empty and left and right sub trees are similar. Write an algorithm to determine if two Binary Trees are similar. **(8)**

Ans:
We assume two trees as tree1 and tree2. The algorithm to determine if two Binary trees are similar or not is as follows:

```
similar(*tree1,*tree2)
{
  while((tree1!=null) && (tree2!=null))
  {
     if((tree1->root) == (tree2->root))
        similar(tree1->left,tree2->left);
        similar(tree1->right,tree2->right);
 elseif(tree->root!=    tree2->node){printf("Trees
 are not equal");
 return;}
  }
}
```

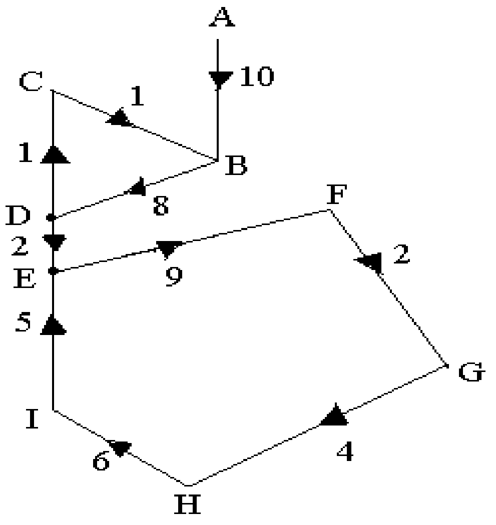
Q.26 The degree of a node is the number of children it has. Show that in any binary tree, the number of leaves are one more than the number of nodes of degree 2 **(8)**

Ans:
Let n be the number of nodes of degree 2 in a binary tree T. We have to show that number of leaves in T is n+1. Let us prove it by induction.
For n=1, the result is obvious that the leaves are two i.e. 1+1.
Let the formulae is true for n=k, i.e. if there are n nodes of degree 2 in T then T has k+1 leaves.
If we add two children to one of the leaf node then, number nodes with degree two will be k+1 and number of leaf node will be (k+1)-1+2 = (k+1)+1. Hence we can conclude by induction that if a binary tree t has n nodes of degree 2 then it has n+1 leaves.

Q.27 Write the non-recursive algorithm to traverse a tree in preorder. **(8)**

Ans:
The Non- Recursive algorithm for preorder traversal is as follows: Initially push NULL onto stack and then set PTR=Root. Repeat the following steps until PTR= NULL.
1. Preorder down the left most path routed at PTR.
2. Processing each node N on the path and pushing each right child if any onto the stack.[The traversing stops when (PTR)=NULL]
3. Backtracking: pop and assign to PTR the top element on stack. If PTR not equal to NULL then return to step 1 otherwise exit.

Q.28 Give the adjacency matrix for the following graph: **(4)**



Ans:

Adjacency matrix for the given graph

	A	B	C	D	E	F	G	H	I
A	0	1	0	0	0	0	0	0	0
B	0	0	0	1	0	0	0	0	0
C	0	1	0	0	0	0	0	0	0
D	0	0	1	0	1	0	0	0	0
E	0	0	0	0	0	1	0	0	0
F	0	0	0	0	0	0	1	0	0
G	0	0	0	0	0	0	0	1	0
H	0	0	0	0	0	0	0	0	1
I	0	0	0	0	1	0	0	0	0

Q.29 Draw the complete undirected graphs on one, two, three, four and five vertices. Prove that the number of edges in an n vertex complete graph is n(n-1)/2. **(8)**

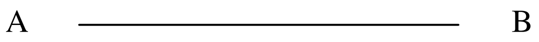
Ans:

The following are the Graphs:

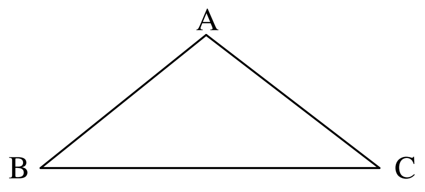
i. One vertex:-

.A

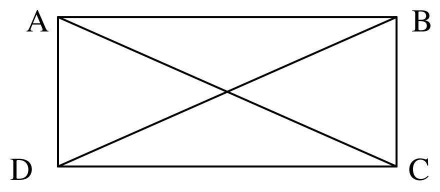
ii. Two vertices:-



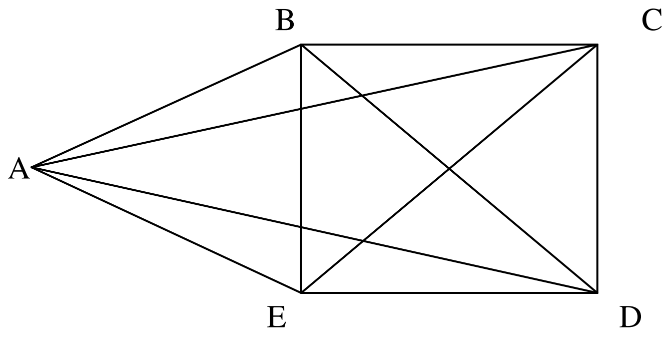
iii. Three vertices:-



iv. Four vertices:-



v. Five vertices:-



From the above Graphs we can see

i. When n=1,

Then number of edges
=n(n-1)/2
= 1(1-1)/2
=1(0)/2
=0/2
=0
Therefore, number of edges = 0.

ii. When n=2,

Then number of edges
=n(n-1)/2
=2(2-1)/2
=2(1)/2
=2/2
=1
Therefore, number of edges = 1.

iii. When n=3,

Then number of edges
=n(n-1)/2

=3(3-1)/2
=3(2)/2
=6/2
=3
Therefore, number of edges = 3.

iv. When n=4,
Then number of edges
=n(n-1)/2
=4(4-1)/2
=4(3)/2
=12/2
=6
Therefore, number of edges = 6.

v. When n=5,
Then number of edges
=n(n-1)/2
=5(5-1)/2
=5(4)/2
=20/2
=10
Therefore, number of edges = 10.

Q.30 Bubble sort algorithm is inefficient because it continues execution even after an array is sorted by performing unnecessary comparisons. Therefore, the number of comparisons in the best and worst cases are the same. Modify the algorithm in such a fashion that it will not make the next pass when the array is already sorted. **(12)**

Ans:
Bubble sort continues execution even after an array is sorted. To prevent unnecessary comparisons we add a Boolean variable say switched and initialize it by True in the beginning. Along with the "for" loop, we add the condition (switched=true) and make it false inside the outer for loop. If a swapping is made then the value of switched is made true. Thus if no swapping is done in the first pass, no more comparisons will be done further and the program shall exit.
The algorithm after modifying it in the above mentioned manner will be as follows:-

```
void bubble(int x[],int n)
{
  int j,pass,hold;
  bool switched=true;
  for(pass=0;pass<n-1 && switched=true;pass++)
  {
     switched=false;
     for(j=0;j<n-pass-1;j++)
     {
        switched=true;
        hold=x[j];
        x[j]=x[j+1];
```

```
                x[j+1]=hold;
            }
        }
    }
```

Q.31 Which sorting algorithm is easily adaptable to singly linked lists? Explain your answer. **(4)**

Ans:
Simple ***Insertion sort*** is easily adabtable to singly linked list. In this method there is an array *link* of pointers, one for each of the original array elements. Initially link[i] = i + 1 for 0 < = i < n-1 and link[n-1] = -1. Thus the array can be thought of as a linear link list pointed to by an external pointer *first* initialized to 0. To insert the kth element the linked list is traversed until the proper position for x[k] is found, or until the end of the list is reached. At that point x[k] can be inserted into the list by merely adjusting the list pointers without shifting any elements in the array. This reduces the time required for insertion but not the time required for searching for the proper position. The number of replacements in the link array is O(n).

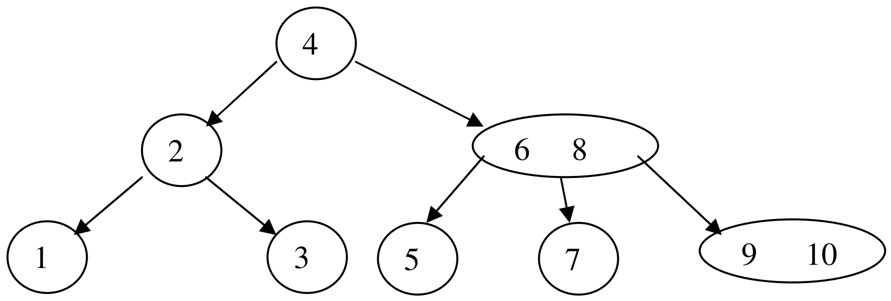
Q.32 Sort the following sequence of keys using merge sort.
66, 77, 11, 88, 99, 22, 33, 44, 55 **(8)**

Ans:
Sorting using Merge sort:-

```
Original File   [66] [77] [11] [88] [99] [22] [33] [44] [55]
    Pass1       [66  77]  [11  88]  [22  99]  [33  44]  [55]
    Pass2       [11  66   77   88]  [22  33   44   99]  [55]
    Pass3       [11  22   33   44   66   77   88   99]  [55]
    Pass4       [11  22   33   44   55   66   77   88   99]
```

Q.33 Draw a B-tree of order 3 for the following sequence of keys:
2, 4, 9, 8, 7, 6, 3, 1, 5, 10 **(8)**

Ans:
The B-tree of order 3 for the following data:-
2 4 9 8 7 6 3 1 5 10



Q.34 What do you mean by complexity of an algorithm? Explain the meaning of worst case analysis and best case analysis with an example. **(8)**

Ans:

The complexity of an algorithm M is the function f(n) which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data. Frequently, the storage space required by an algorithm is simply a multiple of the data size n. Therefore, the term "complexity" shall refer to the running time of the algorithm.

We find the complexity function f(n) for certain cases. The two cases one usually investigates in complexity theory are :-

i. Worst case:- the maximum value of f(n) for any possible input

ii. Best case:- the minimum possible value of f(n)

For example:-

If we take an example of linear search in which an integer Item is to searched in an array Data. The complexity if the search algorithm is given by the number C of comparisons between Item and Data[k].

Worst case:-

The worst case occurs when Item is the last element in the array Data or is it not there at all. In both the cases, we get

$$C(n)=n$$

The average case, we assume that the Item is present it the array and is likely to be present in any position in the array. Thus, the number of comparisons can be any of the numbers 1, 2, 3……..n and each number occurs with probability $p = 1/n$.

$C(n) = 1. 1/n + 2.1/n + … + n.1/n$

$= (n+1) / 2$

thus the average number of comparisons needed to locate the Item in to array Data is approximately equal to half the number of elements in the Data list.

Q.35 Explain the method to calculate the address of an element in an array. A 25×4 matrix array DATA is stored in memory in 'row-major order'. If base address is 200 and $\omega = 4$ words per memory cell. Calculate the address of DATA [12, 3] . **(8)**

Ans:

In a row major representation, the address of any element a[i][j] can be found using the following formula: a[i][j]=base(a)+(i*m+j)*size

where base(a) is base address, m is the no. of columns and size represent size of word.

DATA[12,3]=200+(12*4+3)*4

=200+(48+3)*4

=200+51*4

=200+204

=404

Q.36 What is the difference between a grounded header link list and a circular header link list? **(3)**

Ans:
A header linked list is a linked list which always contains a special node, called the header node, at the beginning of the list. The two kinds of header linked lists are:-
i. Grounder header linked list
ii. Circular header linked list
The difference between the two is that-
A grounded header list is a header list where the last node contains the null pointer.
A circular header list is a header list where the last node points back to the header node.

Q.37 Write an algorithm to insert a node in the beginning of the linked list. **(7)**

Ans:
/* structure containing a data part and link part */
```
struct node
{
 int data ;
 struct node * link ;
} ;
```
/* Following adds a new node at the beginning of the linked list */
```
void addatbeg ( struct node **q, int num )
{
 struct node *temp ;
 /* add new node */
 temp = malloc ( sizeof ( struct node ) ) ;
 temp -> data = num ;
 temp -> link = *q ;
 *q = temp ;
}
```

Q.38 Write an algorithm which does depth first search through an un-weighted connected graph. In an un-weighted graph, would breadth first search or depth first search or neither find a shortest path tree from some node? Why? **(8)**

Ans:
An algorithm that does depth first search is given below:
```
struct node
{
  int data ;
  struct node *next ;
} ; int visited[MAX] ;
void dfs ( int v, struct node **p )
{
  struct node *q ;
  visited[v - 1] = TRUE ;
```

```
        printf ( "%d\t", v ) ;
        q = * ( p + v - 1 ) ;
        while ( q != NULL )
        {
           if ( visited[q -> data - 1] == FALSE )
              dfs ( q -> data, p ) ;
           else
              q = q -> next ;}
                    }
```

Q.39 Convert the following infix expressions into its equivalent postfix expressions;

(i) $(A+B\uparrow D)/(E-F)+G$

(ii) $A*(B+D)/E-F*(G+H/K)$ **(8)**

Ans:

(i). (A + B ^ D) / (E -F) + G

= (A + B D ^) / (E F -) + G

= (A B D ^ +) / (E F -) + G

= (A B D ^ + E F - /) + G

= (A B D ^ + E F - / G +)

the postfix expression is :- **(A B D ^ + E F - / G +)**

(ii). A * (B + D) / E – F * (G + H / K)

= A * (B D +) / E – F * (G + H / K)

= A * (B D +) / E – F * (G + H K /)

= A * (B D +) / E – F * (G H K / +)

= (A B D + *) / E – F * (G H K / +)

= (A B D + * E /) – F * (G H K / +)

= (A B D + * E /) – (F G H K / + *)

= (A B D + * E / F G H K / + * -)

the postfix expression is :- **(A B D + * E / F G H K / + * -)**

Q.40 What is quick sort? Sort the following array using quick sort method.

24 56 47 35 10 90 82 31 **(7)**

Ans:

Quick sort is a sorting algorithm that uses the idea if divide and conquer. This algorithm chooses an element known as pivot element, finds its position in the array such that it divides the array into two sub arrays. The elements in the left sub array are less than and the elements in the right sub array are greater than the

dividing element. Then the algorithm is repeated separately for these two sub array.
The given data is :-

```
          24 56  47  35  10  90  82  31

Pass 1:-  (10)24  (56 47  35  90  82  31)
Pass 2:-   10 24  (56 47  35  90  82  31)
Pass 3:-  10  24  (47 35  31) 56  (90 82)
Pass 4:-  10  24  (35 31) 47  56  (90 82)
Pass 5:-  10  24  (31)35  47  56  (90 82)
Pass 6:-  10  24   31 35  47  56  (90 82)
Pass 7:-  10  24   31 35  47  56  (82)90
Pass 8:-  10  24   31 35  47  56   82 90
```

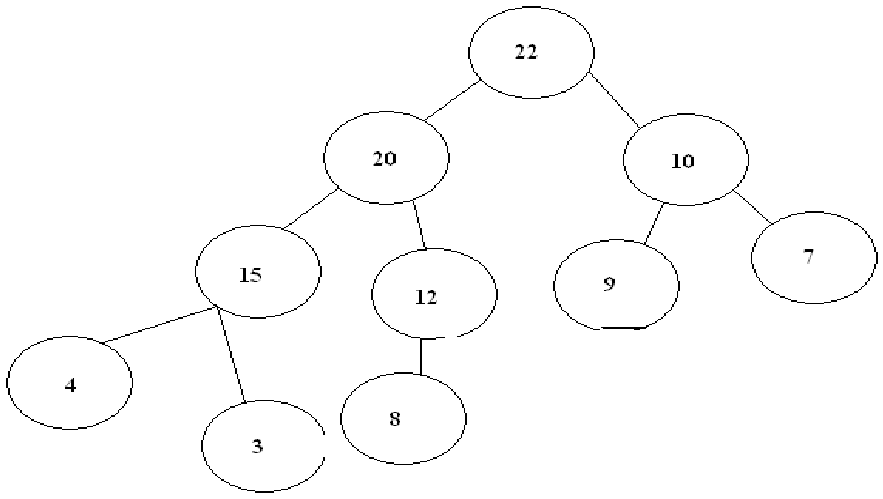
Q.41 How many key comparisons and assignments an insertion sort makes in its worst case? **(4)**

Ans:
The worst case performance in insertion sort occurs when the elements of the input array are in descending order. In that case, the first pass requires one comparison, the second pass requires two comparisons, third pass three comparisons,….kth pass requires (k-1), and finally the last pass requires (n-1) comparisons. Therefore, total numbers of comparisons are:

$$\begin{aligned} f(n) &= 1+2+3+\dots\dots\dots+(n-k)+\dots..+(n-2)+(n-1) \\ &= n(n-1)/2 \\ &= O(n^2) \end{aligned}$$

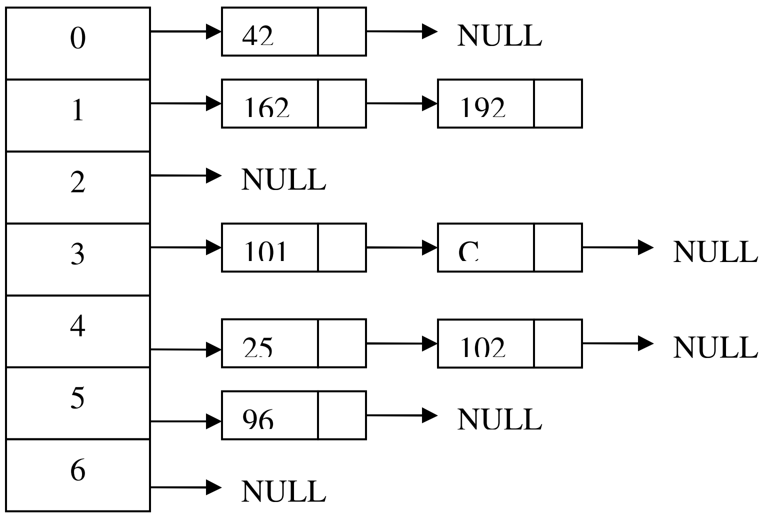
Q.42 Create a heap with following list of keys:
8, 20, 9, 4, 15, 10, 7, 22, 3, 12 **(5)**

Ans:
Creation of a heap:- 8 20 9 4 15 10 7 22 3 12



Q.43 The following values are to be stored in a hash table
25, 42, 96, 101, 102, 162, 197
Describe how the values are hashed by using division method of hashing with a table size of 7. Use chaining as the method of collision resolution. **(8)**

Ans:
Table size=7
25%7=4
42%7=0
96%7=5
101%7=3
102%7=4
162%7=1
197%7=1
So collision resolution can be resolved as follows:



Q.44 Write an algorithm INSERT that takes a pointer to a sorted list and a pointer to a node and inserts the node into its correct position in the list. **(8)**

Ans:
/* structure containing a data part and link part */
```
struct node
{
 int data ;
 struct node *link ;
} ;
/* Following inserts node to an ascending order linked list
*/
void INSERT ( struct node **q, int num )
{
 struct node *r, *temp = *q ;
 r = malloc ( sizeof ( struct node ) ) ;
 r -> data = num ;
 /* if list is empty or if new node is to be inserted
before the first node */
 if ( *q == NULL || ( *q ) -> data > num )
 {
    *q = r ;
    ( *q ) -> link = temp ;
 }
 else
 {
    /* traverse the entire linked list to search the
position to insert the
        new node */
    while ( temp != NULL )
    {
if ( temp -> data <= num && ( temp -> link -> data > num ||
temp -> link == NULL ))
        {
           r -> link = temp -> link ;
           temp -> link = r ;
           return ;
        }
    temp = temp -> link ;  /* go to the next node */ }
      }
}
```

Q.45 Write a non recursive algorithm to traverse a binary tree in inorder. **(8)**

Ans:
Non – recursive algorithm to traverse a binary tree in inorder:-
```
Initially push NULL onto STACK and then set PTR =
ROOT. Then repeat the following steps until NULL is
popped from STACK.
i.    Proceed down the left -most path rooted at PTR,
      pushing each node N onto STACK and stopping when
      a node N with no left child is pushed onto
      STACK.
ii.    [Backtracking.] Pop and process the nodes on
      STACK. If NULL is popped then Exit. If a node N
```

```
with a right child R(N) is processed, set PTR =
R(N) (by assigning PTR = RIGHT[PTR] and return
to Step(a)).
```

Q.46 Describe insertion sort with a proper algorithm. What is the complexity of insertion sort in the worst case? **(8)**

Ans:

Insertion Sort: One of the simplest sorting algorithms is the *insertion sort.* Insertion sort consists of *n* - 1 *passes.* For pass *p* = 2 through *n*, insertion sort ensures that the elements in positions 1 through *p* are in sorted order. Insertion sort makes use of the fact those elements in positions 1 through *p* - 1 are already known to be in sorted order.

```
Void insertion_sort( input_type a[ ], unsigned int n )
{
unsigned int j, p;
input_type tmp;
       a[0] = MIN_DATA;         /* sentinel */
       for( p=2; p <= n; p++ )
{
            tmp = a[p];
            for( j = p; tmp < a[j-1]; j-- )
                a[j] = a[j-1];
            a[j] = tmp;
   }
}
```

Because of the nested loops, each of which can take *n* iterations, insertion sort is $O(n^2)$. Furthermore, this bound is tight, because input in reverse order can actually achieve this bound. A precise calculation shows that the test at line 4 can be executed at most *p* times for each value of *p*. Summing over all *p* gives a total of $(n(n-1))/2 = O(n^2)$.

Q.47 Which are the two standard ways of traversing a graph? Explain them with an example of each. **(8)**

Ans:

The two ways of traversing a graph are as follows:-

i. The *depth-first traversal* of a graph is like the depth-first traversal of a tree. Since a graph has no root, when we do a depth-first traversal, we must specify the vertex at which to begin. Depth-first traversal of a graph visits a vertex and then recursively visits all the vertices adjacent to that node. The catch is that the graph may contain cycles, but the traversal must visit every vertex at most once. The solution to the problem is to keep track of the nodes that have been visited, so that the traversal does not suffer the fate of infinite recursion.

ii. The *breadth-first traversal* of a graph is like the breadth-first traversal of a tree. Breadth-first tree traversal first visits all the nodes at depth zero (i.e., the root), then all the nodes at depth one, and so on. Since a graph has no root, when we do a breadth-first traversal, we must specify the vertex at which to start the

traversal. Furthermore, we can define the depth of a given vertex to be the length of the shortest path from the starting vertex to the given vertex.
Thus, breadth-first traversal first visits the starting vertex, then all the vertices adjacent to the starting vertex, and the all the vertices adjacent to those, and so on.

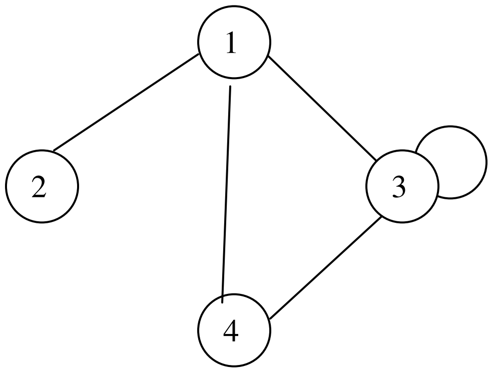
Q.48 Consider the following specification of a graph G

$V(G) = \{1,2,3,4\}$

$E(G) = \{(1,2),(1,3),(3,3),(3,4),(4,1)\}$

(i) Draw an undirected graph.
(ii) Draw its adjacency matrix. **(8)**

Ans:
V(G) = {1,2,3,4}
E(G) = {(1,2),(1,3),(3,3,),(3,4),(4,1)}
(i). The undirected graph will be as follows:-



(ii). The adjacency matrix will be as follows:-

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 1 \\ 4 & 1 & 0 & 1 & 0 \end{array}$$

Q.49 Write short notes on the following:
(i) B-tree.
(ii) Abstract data type.
(iii) Simulation of queues. **(5+6+5)**

Ans:
(i) B-tree:-
Just as AVL trees are balanced binary search trees, B-trees are balanced M-way search trees. By imposing a balance condition, the shape of an AVL tree is constrained in a way which guarantees that the search, insertion, and withdrawal

operations are all O(log n), where n is the number of items in the tree. The shapes of B-Trees are constrained for the same reasons and with the same effect.
A B-Tree of order M is either the empty tree or it is an M-way search tree T with the following properties:

a. The root of T has at least two subtrees and at most M subtrees.
b. All internal nodes of T (other than its root) have between [M / 2] and M subtrees.
c. All external nodes of T are at the same level.

(ii)Abstract Data Types:- A useful tool for specifying the logical properties of a data type is the abstract data type or ADT. The term "abstract data type" refers to the basic mathematical concept that defines the data type. In defining ADT as a mathematical concept, we are not concerned with space or time efficiency.
An ADT consists of two parts:- a value definition and an operator definition. The value definition defines the collection of values for the ADT and consists of two parts: a definition clause and a condition clause. For example, the value consist definition for the ADT RATIONAL states that a RATIONAL value consists of two integers, the second of which does not equal 0. We use array notation to indicate the parts of an abstract type.
(iii) Simulation of queues: Simulation is the process of forming an abstract model of a real world situation in order to understand the effect of modifications and the effect of introducing various strategies on the situation. The simulation program must schedule the events in the simulation so the activities will occur in the correct time sequence. For example, suppose a person visit an outlet at time 't1' to deposit his cellphone bill. The transaction may be expected to take a certain period of time 't2' to complete. If any of the service window is free, the person can immediately deposit the bill in time (t1+t2) spending exactly the time required to deposit the bill. If none of the service window is free and there is queue waiting at each window. The person joins the smallest queue and waits until all previous people deposited their bills and left the line. In this case, the time spends by the person in the bill office in t2+time spends in waiting. Such problems are solved by writing a program to simulate the actions of the persons. The various service windows are represented by different queues and each person waiting in that line is represented by a node in that queue. The node at the front of the queue represents the person currently being serviced at the window. We can define various events associated with the process like arrival of a person, departure of a person etc.
When a departure node is removed from the event list, the amount of time spent by the departing person is computed and added to a total and the node representing the person is removed from the front of the queue. After a node has been deleted from the front of the queue, the next person in the queue becomes the first to be serviced by that window and a departure node is added to the event list. At the end of simulation, when the event list is empty, the total will be divided by the number of persons to get the average time spent by a person.

Q.50 Why do we use a symptotic notation in the study of algorithm? Describe commonly used asymptotic notations and give their significance. **(8)**

Ans:

The running time of an algorithm depends upon various characteristics and slight variation in the characteristics varies the running time. The algorithm efficiency and performance in comparison to alternate algorithm is best described by the order of growth of the running time of an algorithm. Suppose one algorithm for a problem has time complexity as c_3n^2 and another algorithm has $c_1n^3 + c_2n^2$ then it can be easily observed that the algorithm with complexity c_3n^2 will be faster than the one with complexity $c_1n^3 + c_2n^2$ for sufficiently larger values of n. Whatever be the value of c_1, c_2 and c_3 there will be an 'n' beyond which the algorithm with complexity c_3n^2 is faster than algorithm with complexity $c_1n^3 + c_2n^2$, we refer this n as breakeven point. It is difficult to measure the correct breakeven point analytically, so Asymptotic notation are introduced that describe the algorithm efficiency and performance in a meaningful way. These notations describe the behavior of time or space complexity for large instance characteristics. Some commonly used asymptotic notations are:

Big oh notation (O): The upper bound for the function 'f' is provided by the big oh notation (O). Considering 'g' to be a function from the non-negative integers to the positive real numbers, we define O(g) as the set of function f such that for some real constant c>0 and some non negative integers constant n_0 , $f(n) \leq cg(n)$ for all $n \geq n_0$. Mathematically, $O(g(n))=\{f(n)$: there exists positive constants such that $0 \leq f$ $f(n) \leq cg(n)$ for all n, $n \geq n_0\}$, we say "f is oh of g".

Big Omega notation (Ω): The lower bound for the function 'f' is provided by the big omega notation (Ω). Considering 'g' to be a function from the non-negative integers to the positive real numbers, we define Ω(g) as the set of function f such that for some real constant c>0 and some non negative integers constant n_0 , $f(n) \geq cg(n)$ for all $n \geq n_0$. Mathematically, $\Omega(g(n))=\{f(n)$: there exists positive constants such that $0 \leq cg(n) \leq f(n)$ for all n, $n \geq n_0\}$.

Big Theta notation (θ): The upper and lower bound for the function 'f' is provided by the big oh notation (θ). Considering 'g' to be a function from the non-negative integers to the positive real numbers, we define θ(g) as the set of function f such that for some real constant c1 and c2 >0 and some non negative integers constant n_0 , $c1g(n) \leq f$ $f(n) \leq c2g(n)$ for all $n \geq n_0$. Mathematically, $\theta(g(n))=\{f(n)$: there exists positive constants c1 and c2 such that $c1g(n) \leq f$ $f(n) \leq c2g(n)$ for all n, $n \geq n_0\}$, we say "f is oh of g"

Q.51 Illustrate the steps for converting an infix expression into a postfix expression for the following expression $(a+b)*(c+d)/(e+f)\uparrow g$. **(8)**

Ans:

The given infix expression can be converted to postfix expression as follows:

(a+b)*(c+d)/(e+f)^g

=(ab+)*(cd+)/(ef+)^g

=(ab+)*(cd+)/(ef+g^)

=(ab+cd+*)/(ef+g^)

=(ab+cd+*ef+g^/)

The postfix expression is:-

(ab+cd+*ef+g^/)

Q.52 Let P be a pointer to a singly linked list. Show how this list may be used as a stack. That is, write algorithms to push and pop elements. Specify the value of P when the stack is empty. **(8)**

Ans:

Linked list as stack:-

```
//Declaration of the stack
typedef struct node *node_ptr;
struct node
{
    element_type element;
    node_ptr next;
};
typedef node_ptr STACK;
/* Stack implementation will use a header. */
//Routine to test whether a stack is empty
int is_empty( STACK S )
{
     return( S->next == NULL );
}
//Routine to create an empty stack
STACK create_stack( void )
{
     STACK S;
     S = (STACK) malloc( sizeof( struct node ) );
     if( S == NULL )
       fatal_error("Out of space!!!");
    return S;
}
void make_null( STACK S )
{
     if( S != NULL )
        S->next = NULL;
     else
     error("Must use create_stack first");
}
//Routine to push onto a stack
void push( element_type x, STACK S )
{
      node_ptr tmp_cell;
tmp_cell = (node_ptr) malloc( sizeof ( struct
                node ) );
      if( tmp_cell == NULL )
```

```
                    fatal_error("Out of space!!!");
              else
             {
                    tmp_cell->element = x;
                    tmp_cell->next = S->next;
                   S->next = tmp_cell;
              }
        }
        // Routine to pop from a stack
        void pop( STACK S )
        {
             node_ptr first_cell;
             if( is_empty( S ) )
                 error("Empty stack");
             else
             {
                 first_cell = S->next;
                 S->next = S->next->next;
                 free( first_cell );
              }
        }
```

Q.53 How will you represent a max-heap sequentially? Explain with an example. **(4)**

Ans:
A max heap, also called descending heap, of size n is an almost complete binary tree of n nodes such that the content 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.

Info[j] <= info[(j-1)/2] for 0 <= ((j-1)/2) < j <= n-1

It is clear from this definition that root of the tree contains the largest element in the heap in the descending heap. Any path from the root to a leaf is an ordered list in descending order.

Q.54 Write an algorithm to insert an element to a max-heap that is represented sequentially. **(8)**

Ans:
An algorithm to insert an element “newkey” to a max heap of size i-1 where i>=1:

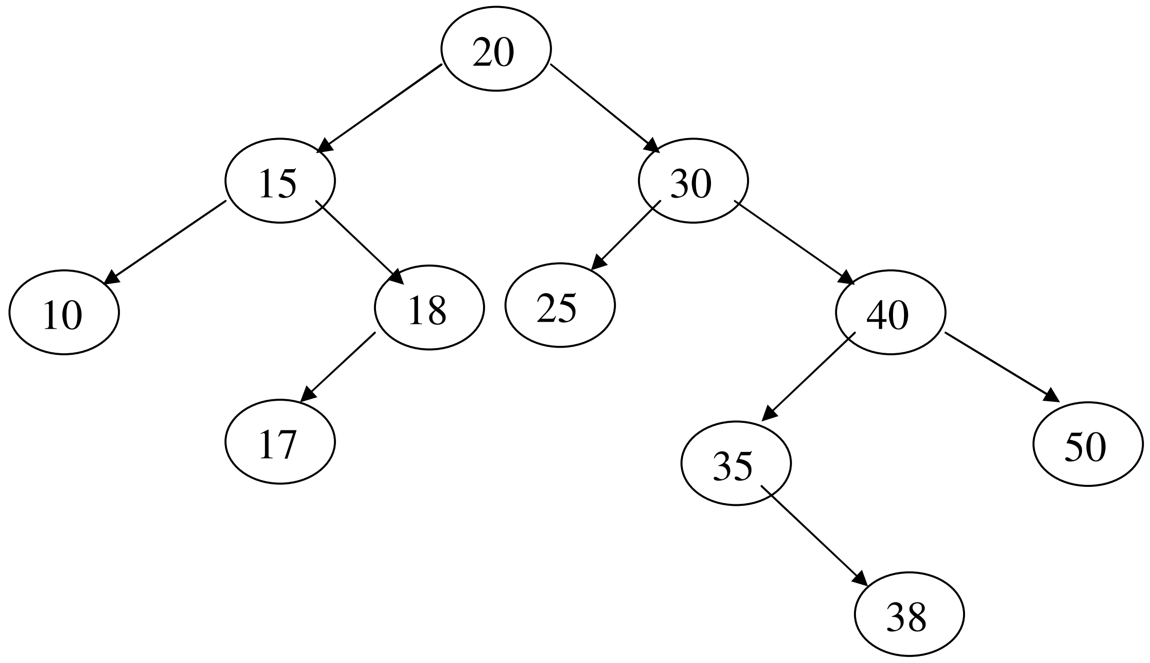
```
s=i;
parent=s div 2;
key[s]=newkey;
while(s<>0 && key[parent]<=key[s])
{
      /*interchange parent and child*/
      temp=key[parent];
       key[parent]=key[s];
       key[s]=temp;
        s=parent;
```

```
                parent=s div 2;
        }
```

Q.55 Construct a binary tree whose nodes in inorder and preorder are given as follows:
Inorder : 10, 15, 17, 18, 20, 25, 30, 35, 38, 40, 50
Preorder: 20, 15, 10, 18, 17, 30, 25, 40, 35, 38, 50 **(10)**

Ans:
Construction of a Binary tree whose nodes in inorder and preorder are given as follows:-
Inorder:- 10, 15, 17, 18, 20, 25, 30, 35, 38, 40, 50

Preorder:- 20, 15, 10, 18, 17, 30, 25, 40, 35, 38, 50



Q.56 Execute your algorithm to convert an infix expression to a post fix expression with the following infix expression on your input
$(m+n)*(k+p)/(g/b)\uparrow(a\uparrow b/c)$ **(8)**

Ans:
The infix expression is:- (m + n)*(k + p) / (g / b) ^ (a ^ b / c)

SYMB	STACK	POSTFIX EXPRESSION
(	(	
m	(	m
+	(+	m
n	(+	mn
)	empty	mn+

*	*	mn+
(	*(	mn+
k	*(	mn+k
+	*(+	mn+k
p	*(+	mn+kp
)	*	mn+kp+
/	/	mn+kp+*
(	/(	mn+kp+*
g	/(	mn+kp+*g
/	/(/	
b	/(/	mn+kp+*gb
)	/	mn+kp+*gb/
^	/^	
(	/^(	
a	/^(	mn+kp+*gb/a
^	/^(^	
b	/^(^	mn+kp+*gb/ab
/	/^(/	mn+kp+*gb/ab^
c	/^(/	mn+kp+*gb/ab^c
)	/^	mn+kp+*gb/ab^c/
		mn+kp+*gb/ab^c/^/

The postfix expression is ***mn+kp+*gb/ab^c/^/.***

Q.57 A double ended queue is a linear list where additions and deletions can be performed at either end. Represent a double ended queue using an array to store elements and write modules for additions and deletions. **(8)**

Ans:
```
#include <stdio.h>
 #include <conio.h>
 #define MAX 10
void addqatbeg ( int *, int, int *, int * ) ;
void addqatend ( int *, int, int *, int * ) ;
int delqatbeg ( int *, int *, int * ) ;
int delqatend ( int *, int *, int * ) ;
void display ( int * ) ;
int count ( int * ) ;
 void main( )
```

```
   {
 int arr[MAX] ;
 int front, rear, i, n ;
 clrscr( ) ;
 /* initialises data members */
 front = rear = -1 ;
 for ( i = 0 ; i < MAX ; i++ )
    arr[i] = 0 ;
 addqatend ( arr, 17, &front, &rear ) ;
 addqatbeg ( arr, 10, &front, &rear ) ;
 addqatend ( arr, 8, &front, &rear ) ;
 addqatbeg ( arr, -9, &front, &rear ) ;
 addqatend ( arr, 13, &front, &rear ) ;
 addqatbeg ( arr, 28, &front, &rear ) ;
 addqatend ( arr, 14, &front, &rear ) ;
 printf ( "\nElements in a deque: " ) ;
 display ( arr ) ;
 n = count ( arr ) ;
 printf ( "\nTotal number of elements in deque: %d", n );
 i = delqatbeg ( arr, &front, &rear ) ;
 printf ( "\nItem extracted: %d", i ) ;
 i = delqatbeg ( arr, &front, &rear ) ;
 printf ( "\nItem extracted:%d", i ) ;
 i = delqatbeg ( arr, &front, &rear ) ;
 printf ( "\nItem extracted:%d", i ) ;
 i = delqatbeg ( arr, &front, &rear ) ;
 printf ( "\nItem extracted: %d", i ) ;
 printf ( "\nElements in a deque after deletion: " ) ;
 display ( arr ) ;
 addqatend ( arr, 16, &front, &rear ) ;
 addqatend ( arr, 7, &front, &rear ) ;
    printf ( "\nElements in a deque after addition: " ) ;
 display ( arr ) ;
 i = delqatend ( arr, &front, &rear ) ;
 printf ( "\nItem extracted: %d", i ) ;
 i = delqatend ( arr, &front, &rear ) ;
 printf ( "\nItem extracted: %d", i ) ;
 printf ( "\nElements in a deque after deletion: " ) ;
 display ( arr ) ;
 n = count ( arr ) ;
 printf ( "\nTotal number of elements in deque: %d", n );
  getch( ) ;
 }
void addqatbeg ( int *arr, int item, int *pfront, int
*prear )
{
/* adds an element at the beginning of a deque */
 int i, k, c ;
 if ( *pfront == 0 && *prear == MAX - 1 )
```

```
{
   printf ( "\nDeque is full.\n" ) ;
   return ;
     }
if ( *pfront == -1 )
{
   *pfront = *prear = 0 ;
   arr[*pfront] = item ;
   return ;
}
if ( *prear != MAX - 1 )
{
   c = count ( arr ) ;
   k = *prear + 1 ;
   for ( i = 1 ; i <= c ; i++ )
   {
      arr[k] = arr[k - 1] ;
      k-- ;
   }
   arr[k] = item ;
   *pfront = k ;
( *prear )++ ;
}
else
{
   ( *pfront )-- ;
   arr[*pfront] = item ;
}
   }
  void addqatend ( int *arr, int item, int *pfront, int
  *prear )
  {
  /* adds an element at the end of a deque */
int i, k ;
if ( *pfront == 0 && *prear == MAX - 1 )
{
   printf ( "\nDeque is full.\n" ) ;
   return ;
}
if ( *pfront == -1 )
{
   *prear = *pfront = 0 ;
   arr[*prear] = item ;
   return ;
}
if ( *prear == MAX - 1 )
{
   k = *pfront - 1 ;
   for ( i = *pfront - 1 ; i < *prear ; i++ )
```

```
      {
         k = i ;
         if ( k == MAX - 1 )
            arr[k] = 0 ;
         else
            arr[k] = arr[i + 1] ;
      }
      ( *prear )-- ;
      ( *pfront )-- ;
   }
   ( *prear )++ ;
   arr[*prear] = item ;
      }
     int delqatbeg ( int *arr, int *pfront, int *prear )
     {
     /* removes an element from the *pfront end of deque */
   int item ;
   if ( *pfront == -1 )
   {
      printf ( "\nDeque is empty.\n" ) ;
      return 0 ;
   }
   item = arr[*pfront] ;
   arr[*pfront] = 0 ;

   if ( *pfront == *prear )
      *pfront = *prear = -1 ;
   else
      ( *pfront )++ ;
   return item ;
      }
     int delqatend ( int *arr, int *pfront, int *prear )
     {                                                  /*
     removes an element from the *prear end of the deque */
   int item ;
   if ( *pfront == -1 )
   {
   printf ( "\nDeque is empty.\n" ) ;
      return 0 ;
      }
      item = arr[*prear] ;
   arr[*prear] = 0 ;
   ( *prear )-- ;
   if ( *prear == -1 )
      *pfront = -1 ;
   return item ;
      }
     void display ( int *arr )
```

```
{                          /* displays elements of a
deque */
 int i ;
 printf ( "\n front->" ) ;
 for ( i = 0 ; i < MAX ; i++ )
    printf ( "\t%d", arr[i] ) ;
 printf ( " <-rear" ) ;
}
int count ( int *arr )
{
/* counts the total number of elements in deque */
 int c = 0, i ;

 for ( i = 0 ; i < MAX ; i++ )
 {
    if ( arr[i] != 0 )
       c++ ;
 }
 return c ;
}
```

Q.58 What do you mean by hashing? Explain any five popular hash functions. **(8)**

Ans:

Hashing provides the direct access of record from the file no matter where the record is in the file. This is possible with the help of a hashing function H which map the key with the corresponding key address or location. It provides the key-to-address transformation.

Five popular hashing functions are as follows:

Division Method: An integer key x is divided by the table size m and the remainder is taken as the hash value. It can be defined as

H(x)=x%m+1

For example, x=42 and m=13, H(42)=45%13+1=3+1=4

Midsquare Method: A key is multiplied by itself and the hash value is obtained by selecting an appropriate number of digits from the middle of the square. The same positions in the square must be used for all keys. For example if the key is 12345, square of this key is value 152399025. If 2 digit addresses is required then position 4th and 5th can be chosen, giving address 39.

Folding Method: A key is broken into several parts. Each part has the same length as that of the required address except the last part. The parts are added together, ignoring the last carry, we obtain the hash address for key K.

Multiplicative method: In this method a real number c such that 0<c<1 is selected. For a nonnegative integral key x, the hash function is defined as

H(x)=[m(cx%1)]+1

Here,cx%1 is the fractional part of cx and [] denotes the greatest integer less than or equal to its contents.

Digit Analysis: This method forms addresses by selecting and shifting digits of the original key. For a given key set, the same positions in the key and same rearrangement pattern must be used. For example, a key 7654321 is transformed to the address 1247 by selecting digits in position 1,2,4 and 7 then by reversing their order.

Q.59 What is the best case complexity of quick sort and outline why it is so. How could its worst case behaviour arise? **(6)**

Ans:

In the **best case** complexity, the pivot is in the middle. To simplify the math, we assume that the two sub-files are each exactly half the size of the original, and although this gives a slight overestimate, this is acceptable because we are only interested in a Big-Oh answer.

$T(n) = 2T(n/2) + cn$

which yields

$\boldsymbol{T(n) = cn \log n + n = O(n \log n)}$

Worst case:- The pivot is the smallest element, all the time. Then $i = 0$ and if we ignore $T(0) = 1$, which is insignificant, the recurrence is $T(n) = T(n - 1) + cn$, $n > 1$ which yields

$$T(n) = T(1) + c\sum_{i=2}^{n} i = O(n^2)$$

Q.60 Explain Dijkstra's algorithm for finding the shortest path in a given graph. **(6)**

Ans:

Dijkstra's algorithm: This problem is concerned with finding the least cost path from an originating node in a weighted graph to a destination node in that graph.

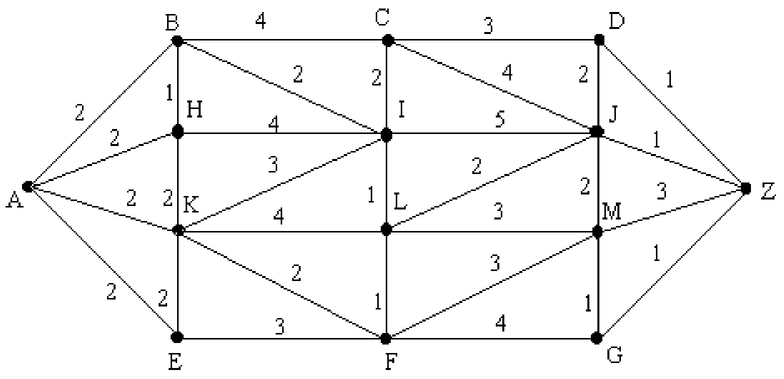
The algorithm is as follows:

```
Let G=(V,E) be a simple graph. Let a and z be any two
vertices of the graph. Suppose L(x ) denotes the label
of the vertex x which represent the length of shortest
path from the vertex a to the vertex x and $w_{ij}$ denotes
the weight of the edge $e_{ij}$=<$V_i$, $V_j$>
1.  Let P=φ where  P is the set of those vertices which
    have permanent labels and T={all vertices in the
    graph}
    Set L(a)=0, L(x)=∞ for all x≠a
2.  Select the vertex v in T which has the smallest
    label. This label is called permanent  label of v.
    Also set P=P {v} and T=T-{v}
    If v=z then L(z) is the length of the shortest path
    from the vertex a to z and stop.
3.  If v is not equal z then revise the labels of
    vertices of T that is the vertices which do not
    have permanent labels.
    The new label of vertex x in T is given by
     L(x)=min(old L(x), L(v)+w(v, x))
```

```
Where w(v, x) is the weight of the edge joining the
vertex v and x. If there is no direct edge joining
v and x then take w(v, x)= ∞
```

```
4. Repeat steps 2 and 3 until z gets the permanent
   label.
```

Q.61 Find the shortest path from A to Z using Dijkstra's Algorithm. **(10)**



Ans:

1. P=φ; T={A,B,C,D,E,F,G,H,I,J,K,L,M,Z}
   Let L(A)=0 and L(x)=∞ for all x in T (L(x) denotes label of x)
2. P={A}, T={B,C,D,E,F,G,H,I,J,K,L,M,Z}
   A≠Z so calculate new labels for vertices in T

L(B)=min(∞, 0+2)=2	L(H)=min(∞, 0+2)=2
L(C)=min(∞, 0+∞)=∞	L(I)=∞
L(D)=∞	L(J)=∞
L(E)=min(∞, 0+2)=2	L(K)=min(∞, 0+2)=2
L(F)=∞	L(L)=∞
L(G)=∞	L(M)= ∞ ; L(Z)= ∞

3. v=B, vertex with smallest label, L(v)=2
   P={A,B}, T={C,D,E,F,G,H,I,J,K,L,M,Z}
B≠Z so calculate new labels for vertices in T

L(C)=min(∞, 2+4)=6	L(I)=min(∞, 2+2)=4
L(D)=∞	L(J)=∞
L(E)=min(2,2+∞)=2	L(K)=min(2,2+∞)=2
L(F)=∞	L(L)=∞
L(G)=∞	L(M)=∞
L(H)=min(2,2+1)=2	L(Z)= ∞

4. v=E, vertex with smallest label, L(v)=2

P={A,B,E}, T={C,D,F,G,H,I,J,K,L,M,Z}
B≠Z so calculate new labels for vertices in T

L(C)=min(6, 2+∞)=6	L(J)= ∞
L(D)=∞	L(K)=min(2,2+2)=2
L(F)=min(∞,2+3)=5	
L(G)=∞	L(L)=∞
L(H)=∞	L(M)=∞
L(I)= ∞	L(Z)= ∞

5. v=K, vertex with smallest label, L(v)=2
   P={A,B,E,K}, T={C,D,F,G,H,I,J,L,M,Z}

K≠Z so calculate new labels for vertices in T

L(C)=min(6, 2+∞)=6	L(J)= ∞
L(D)=∞	L(L)=min(∞,2+4)=6
L(F)=min(5,2+2)=4	L(M)=∞
L(G)=∞	L(Z)= ∞
L(H)=min(∞,2+2)=4	
L(I)= min(∞,2+3)=5	

6. v=F, vertex with smallest label, L(v)=4
   P={A,B,E,K,F}, T={C,D,G,H,I,J,L,M,Z}

K≠Z so calculate new labels for vertices in T

L(C)=min(6, 4+∞)=6	L(I)= min(5,4+∞)=5
L(D)=∞	L(L)=min(6,4+1)=5
L(G)=min(∞,4+4)=8	L(M)=min(∞,4+3)=7
L(G)=∞	L(Z)= ∞
L(H)=min(4,4+∞)=4	L(J)= ∞

7. v=H, vertex with smallest label, L(v)=4
   P={A,B,E,K,F,H}, T={C,D,G,I,J,L,M,Z}

K≠Z so calculate new labels for vertices in T

L(C)=min(6, 4+∞)=6	L(J)= min(∞,4+∞)=∞
L(D)=∞	L(L)=min(5,4+∞)=5
L(G)=min(8,4+∞)=8	L(M)=min(7,4+∞)=7
L(I)=min(5,4+4)=5	L(Z)= ∞

8. v=I, vertex with smallest label, L(v)=5
   P={A,B,E,K,F,H,I}, T={C,D,G,J,L,M,Z}

I≠Z so calculate new labels for vertices in T

L(C)=min(6, 5+2)=6	L(L)= min(5,5+1)=5
L(D)=∞	L(M)=min(7,5+∞)=7
L(G)=min(8,5+∞)=8	L(Z)=∞
L(J)=min(∞,5+5)=10	

9. v=L, vertex with smallest label, L(v)=5
   P={A,B,E,K,F,H,I,L}, T={C,D,G,J,M,Z}

L≠Z so calculate new labels for vertices in T

L(C)=min(6, 5+∞)=6	L(J)= min(10,5+2)=7
L(D)=∞	L(M)=min(7,5+3)=7
L(G)=min(8,5+∞)=8	L(Z)=∞

10. v=C, vertex with smallest label, L(v)=6
    P={A,B,E,K,F,H,I,L,C}, T={D,G,J,M,Z}

C≠Z so calculate new labels for vertices in T

L(D)=min(∞,6+3)=9	L(M)= min(7,6+∞)=7
L(G)=min(8,6+∞)=8	L(Z)=∞
L(J)=min(7,6+4)=7	

11. v=J, vertex with smallest label, L(v)=7
    P={A,B,E,K,F,H,I,L,C,J}, T={D,G,M,Z}

J≠Z so calculate new labels for vertices in T

L(D)=min(9,7+2)=9	L(M)= min(7,7+2)=7
L(G)=min(8,7+∞)=8	L(Z)=min(∞,7+1)=8

12. v=M, vertex with smallest label, L(v)=7
    P={A,B,E,K,F,H,I,L,C,J,M}, T={D,G,Z}

M≠Z so calculate new labels for vertices in T

L(D)=min(9,7+∞)=9	L(Z)= min(8,7+3)=8
L(G)=min(8,7+1)=8	

13. v=G, vertex with smallest label, L(v)=8
    P={A,B,E,K,F,H,I,L,C,J,M,G}, T={D,Z}

G≠Z so calculate new labels for vertices in T

L(D)=min(9,8+∞)=9	L(Z)= min(8,8+1)=8

14. v=z, so we stop here, Shortest distance is 8
    Backtracking all steps we get, shortest path as: A-K-F-L-J-Z

Q.62 Define a B-Tree. **(4)**

Ans:

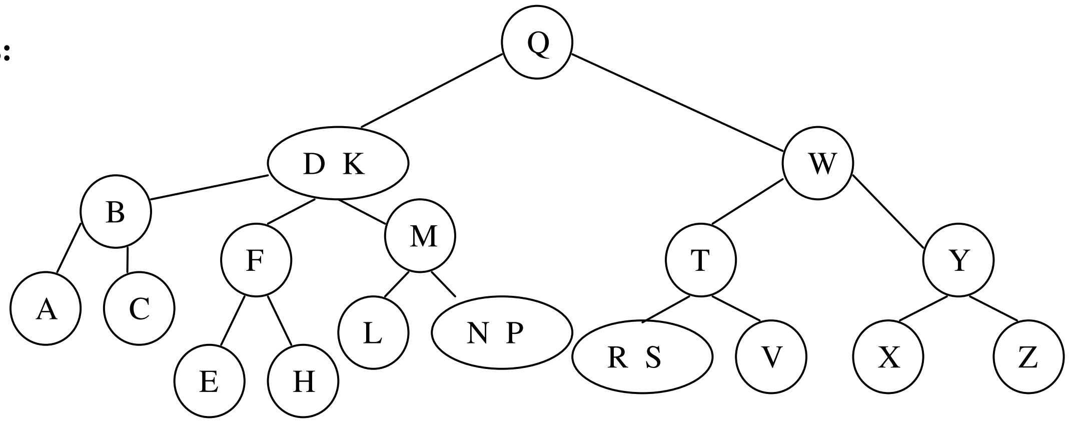
Just as AVL trees are balanced binary search trees, B-trees are balanced M-way search trees. A B-Tree of order M is either the empty tree or it is an M-way search tree T with the following properties:

1. Each non-leaf node has a maximum of M children and minimum of M/2 keys.
2. Each node has one fewer key than the number of children with a maximum of M-1 keys.
3. Keys are arranged in a defined order within the node. All keys in the subtree to the left of a key are predecessor of the key and that on the right are successors of the key.
4. All leaves are on the same level.

Q.63 Show the result of inserting the keys.

F, S, Q, K, C, L, H, T, V, W, M, R, N , P, A, B, X, Y, D, Z, E in the order to an empty B-tree of degree-3. **(12)**

Ans:



Q.64 What are stacks? How can stacks be used to check whether an expression is correctly parenthized or not. For eg(()) is well formed but (() or)()(is not. **(7)**

Ans:
A stack is a data structure that organizes data similar to how one organizes a pile of coins. The new coin is always placed on the top and the oldest is on the bottom of the stack. When accessing coins, the last coin on the pile is the first coin removed from the stack. If we want the third coin, we must remove the first two coins from the top of the stack first so that the third coin becomes the top of the stack and we can remove it. There is no way to remove a coin from anywhere other than the top of the stack.
A stack is useful whenever we need to store and retrieve data in last in, first out order. For example, The computer processes instructions using a stack in which the next instruction to execute is at the top of the stack.
To determine whether an expression is well parenthized or not:- two conditions should be fulfilled while pushing an expression into a stack. Firstly, whenever an opening bracket is pushed inside a stack, there must be an occurrence a closing bracket before the last symbol is reached. Whenever a closing bracket is encountered, the top of the stack is popped until the opening bracket is popped out and discarded. If no such opening bracket is found and stack is emptied, then this means that the expression is not well parenthized.
An algorithm to check whether an expression is correctly parenthized is as follows:

```
flag=TRUE;
clear the stack;
Read a symbol from input string;
while not end of input string and flag do
 {
     if(symbol= '( ' or symbol= '[' or symbol = '{' )
         push(symbol,stack);
    else  if(symbol= ') ' or symbol= '[' or symbol =
'{' )
```

```
    if stack is empty
             flag=false;
             printf("More right parenthesis than left
parenthises");
          else
             c=pop(stack);
             match c and the input symbol;
             If not matched
               {     flag=false;
                      printf("Mismatched
parenthesis");
                 }
          Read the next input symbol;
    }
if stack is empty then
   printf("parentheses are balanced properly");
else
    printf(" More left parentheses than right
parentheses");
```

Q.65 What is a linear array? Explain how two dimensional arrays are represented in memory. **(8)**

Ans:
An *array* is a way to reference a series of memory locations using the same name. Each memory location is represented by an array element. An *array element* is similar to one variable except it is identified by an index value instead of a name. An *index* value is a number used to identify an array element. The square bracket tells the computer that the value inside the square bracket is an index.

grades[0]
grades[1]

Representation of two-dimensional arrays in memory:-
Let grades be a 2-D array as grades[3][4]. The array will be represented in memory by a block of 3*4=12 sequential memory locations. It can be stored in two ways:- row major wise or column major wise. The first set of four array elements is placed in memory, followed by the second set of four array elements, and so on.

Q.66 Write an algorithm to merge two sorted arrays into a third array. Do not sort the third array. **(8)**

Ans:

```
  void merge(int *a,int *b,int n,int m)
     {
     int i=0,c[20],j=0,k=0,count=0;
     while(i<=n&&j<=m)
     {
      if(a[i]<b[j])
      {
            c[k]=a[i];
            i++;
            k++;
           }
           if(a[i]>b[j])
           {
            c[k]=b[j];
            j++;
            k++;
           }
           if(a[i]==b[j])
           {
            c[k]=a[i];
            k++;
            i++;
            j++;
            count++;
           }
                }
           if(i<=n&&j==m)
           {
            while(i<=n)
            {
             c[k]=a[i];
             i++;
             k++;
            }
           }
           if(i==n&&j<=m)
           {
            while(j<=m)
            {
             c[k]=b[j];
             i++;
             j++;
            }
           }
           for(i=0;i<m+n-count;i++)
              printf("%d\t",c[i]);
          }
```

Q.67 Write a complete programme in C to create a single linked list. Write functions to do the following operations
(i) Insert a new node at the end
(ii) Delete the first node **(8)**

Ans:

```
//Create a single list
  struct node
```

```
{
 int data;
 struct node *link
}
struct node *p,*q;
//Inserting a node at the end(append)
append(struct node **q,int num)
{
 struct node *temp;
 temp=*q;
 if(*q==NULL)
 {
    *q=malloc(sizeof(struct node *));
    temp=*q;
 }
 else
 {
    while(temp → link!=NULL)
    temp=temp→link;
    temp→link = malloc(sizeof(struct node *));
    temp=temp→link;
 }
 Temp→data=num;
 temp →link=NULL;
}
//Delete the first node
delete(struct node *q,int num)
{
 struct node *temp;
 temp=*q;
 *q=temp→link;
 free(temp);
}
```

Q.68 Define a sparse metrics. Explain the representation of a 4X4 matrix using linked list. **(8)**

Ans:

A matrix in which number of zero entries are much higher than the number of non zero entries is called sparse matrix. The natural method of representing matrices in memory as two-dimensional arrays may not be suitable for sparse matrices. One may save space by storing only nonzero entries. For example matrix A (3*3 matrix) represented below

```
0 2 0
 5 0 0
 0 6 9
```

can be written in sparse matrix form as:

```
 3 3 4
0 1 2
```

```
 1  0  5
 2  2  6
 2  3  9
```

where first row represent the dimension of matrix and last column tells the number of non zero values; second row onwards it is giving the position and value of non zero number.
Representation of a 4*4 matrix using linked list is given below:

```
#define MAX1 4
#define MAX2 4
struct cheadnode            /* structure for col
headnode */
{
 int colno ;
 struct node *down ;
 struct cheadnode *next ;
} ;
struct rheadnode           /* structure for row
headnode */
{
 int rowno ;
 struct node * right ;
 struct rheadnode *next ;
} ;
struct node                  /* structure for node to
store element */
{
 int row ;
 int col ;
 int val ;
 struct node *right ;
struct node *down ;
} ;
struct spmat                 /* structure for special
headnode */
{
 struct rheadnode *firstrow ;
 struct cheadnode *firstcol ;
 int noofrows ;
 int noofcols ;
 } ;
struct sparse
{
 int *sp ;
 int row  ;
 struct spmat *smat ;
 struct cheadnode *chead[MAX2] ;
 struct rheadnode *rhead[MAX1] ;
 struct node *nd ;
} ;
```

```
void initsparse ( struct sparse *p )             /*
initializes structure elements */
{
    int i ;
for ( i = 0 ; i < MAX1 ; i++ )             /* create
row headnodes */
 p -> rhead[i] = ( struct rheadnode * ) malloc (
           sizeof ( struct rheadnode ) ) ;
 for ( i = 0 ; i < MAX1 - 1 ; i++ )  /* initialize and
link row headnodes together */
 {
    p -> rhead[i] -> next = p -> rhead[i + 1] ;
    p -> rhead[i] -> right = NULL ;
    p -> rhead[i] -> rowno = i ;
 }
 p -> rhead[i] -> right = NULL ;
 p -> rhead[i] -> next = NULL ;
 for ( i = 0 ; i < MAX1 ; i++ )            /* create
col headnodes */
 p -> chead[i] = ( struct cheadnode * ) malloc (
sizeof ( struct cheadnode ) ) ;
 for ( i = 0 ; i < MAX2 - 1 ; i++ )                /*
initialize and link col headnodes together */
 {
    p -> chead[i] -> next = p -> chead[i + 1] ;
 p -> chead[i] -> down = NULL ;
    p -> chead[i] -> colno = i ;
 }
 p -> chead[i] -> down = NULL ;
 p -> chead[i] -> next = NULL ;
 /* create and initialize special headnode */
 p -> smat = ( struct spmat * ) malloc ( sizeof (
struct spmat ) ) ;
 p -> smat -> firstcol = p -> chead[0] ;
 p -> smat -> firstrow = p -> rhead[0] ;
 p -> smat -> noofcols = MAX2 ;
 p -> smat -> noofrows = MAX1 ;
}
void create_array ( struct sparse *p )     /* creates,
dynamically the matrix of size MAX1 x MAX2 */
{
 int n, i ;
 p -> sp = ( int * ) malloc ( MAX1 * MAX2 * sizeof (
int ) ) ;
    for ( i = 0 ; i < MAX1 * MAX2 ; i++ )           /*
get the element and store it */
 {
    printf ( "Enter element no. %d:", i ) ;
    scanf ( "%d", &n ) ;
```

```
            * ( p -> sp + i ) = n ;
        }
       }
```

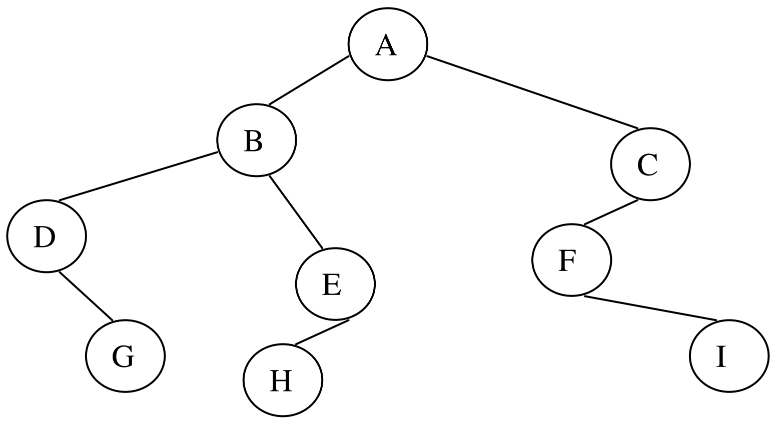
Q.69 Given the following inorder and preorder traversal reconstruct a binary tree

Inorder sequence	D, G, B, H, E, A, F, I, C
Preorder sequence	A, B, D, G, E, H, C, F, I

(8)

Ans:

The given *In-order sequence:-* D, G, B, H, E, A, F, I, C
Pre-order sequence:- A, B, D, G, E, H, C, F, I
The binary tree T is drawn from its root downward by choosing the first node in its pre-order as root of T then study left and right child recursively. The tree T is shown below:



Q.70 Devise a representation for a list where insertions and deletions can be made at either end. Such a structure is called Deque (Double ended queue). Write functions for inserting and deleting at either end. **(8)**

Ans:

```
struct Deque
{
   int info;
   struct Deque *left, *right;
}*head, *tail;
add(struct Deque *p, int x)
{
 struct Deque *temp;
 temp = malloc (sizeof(struct Deque), 1);
 temp -> info = x;
 if(p == head)
 {
  temp->right = head;
  temp->left = NULL;
  head->left = temp;
```

```
   head = temp;
  }
  else if(p == tail)
  {
   temp->left = tail;
   temp->right = NULL;
   tail->right = temp;
   tail = temp;
  }
 }
 delete(struct Deque *p, int x)
 {
   struct Deque *temp;
   if(p == head)
   {
    temp = head;
    head = head->right;
    head->left = NULL;
    temp->right = NULL;
    free(temp);
   }
   else if(p == tail)
   {
    temp = tail;
    tail = tail->left;
    tail->right = NULL;
    temp->left = NULL;
    free(temp);
   }
 }
```

Q.71 Define Hashing. How do collisions happen during hashing? Explain the different techniques resolving of collision. **(8)**

Ans:

Hashing provides the direct access of record from the file no matter where the record is in the file. This is possible with the help of a hashing function H which map the key with the corresponding key address or location. It provides the key-to-address transformation.

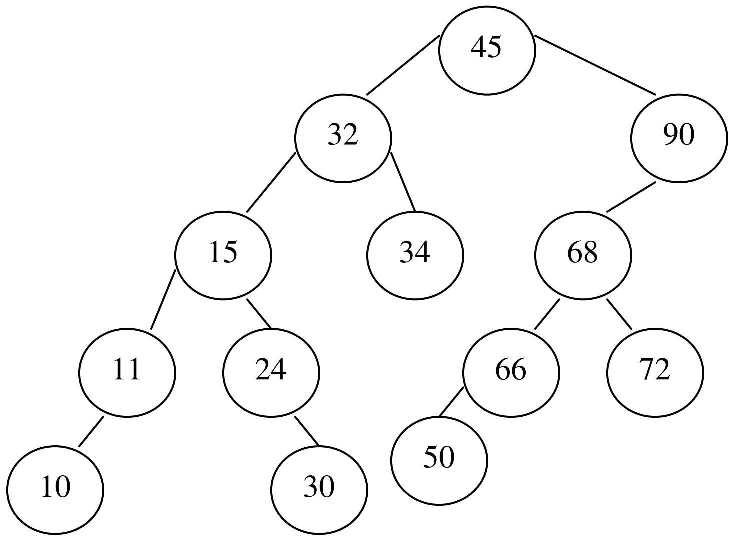
Usually the key space is much larger than the address space, therefore, many keys are mapped to the same address. Suppose that two keys K1 and K2 map to the same address. When the record with key K1 is entered, it is inserted at the hashed address, but when another record with key K2 is entered, it is a dilemma where to insert as a record with key K1 is already there. This situation is called a hash collision. Two broad classes of collision resolution techniques are: open addressing and chaining.

Open addressing: The simplest way to resolve a collision is to start with the hash address and do a sequential search through the table for an empty location.

The idea is to place the record in the next available position in the array. This method is called linear probing. An empty record is indicated by a special value called null. The major drawback of the linear probe method is clustering.
Chaining: In this technique, instead of hashing function value as location we use it as an index into an array of pointers. Each pointer access a chain that holds the element having same location.

Q.72 Make a BST for the following sequence of numbers.
45,32,90,34,68,72,15,24,30,66,11,50,10 Traverse the BST created in Pre-order, Inorder and Postorder. **(8)**

Ans:
The property that makes a binary tree into a binary search tree is that for every node, *X*, in the tree, the values of all the keys in the left subtree are smaller than the key value in *X*, and the values of all the keys in the right subtree are larger than the key value in *X*.
The Binary Search Tree for given data 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10

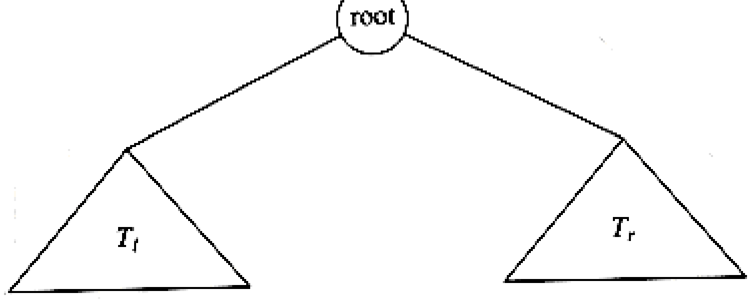


The traversals for the above tree is as follows:-
Pre-order traversal is-
45 , 32, 15, 11, 10, 24, 30, 34, 90, 68, 66, 50, 72
Inorder traversal is:-
10, 11, 15, 24, 30, 32, 34, 45, 50, 66, 68, 72, 90
Postorder traversal:-
10, 11, 30, 24, 15, 34, 32, 50, 66, 72, 68, 90, 45

Q.73 What is a Binary Tree? What is the maximum number of nodes possible in a Binary Tree of depth d. Explain the following terms with respect to Binary trees
(i) Strictly Binary Tree (ii) Complete Binary Tree (iii) Almost Complete Binary Tree **(8)**

Ans:

A binary tree is a tree in which no node can have more than two children.
Figure below shows that a binary tree consists of a root and two subtrees, T_l and T_r, both of which could possibly be empty.



The maximum numbers of nodes a binary tree of depth d can have is $2^{d+1}-1$.

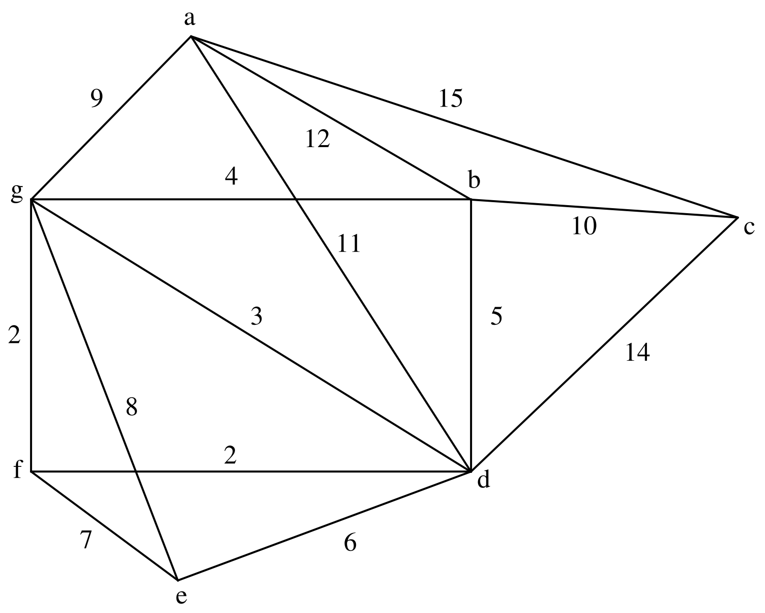
(i) *Strictly Binary Tree:-* If every non leaf node in binary tree has non empty left and right sub-trees , then the tree is called a strictly binary tree.

(ii) *Complete Binary Tree:-* A complete binary tree of depth d is that strictly binary tree all of whose leaves are at level D.

(iii) *Almost Complete Binary Tree:-*A binary tree of depth d is an almost complete binary tree if: 1.Any node nd at level less than d-1 has two children. 2. for any node nd in the tree with

(iv) a right descendant at level d, nd must have a left child and every descendant of nd is either a leaf at level d or has two children.

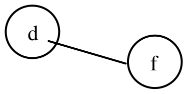
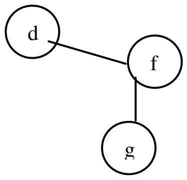
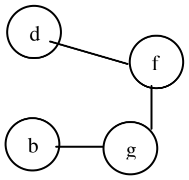
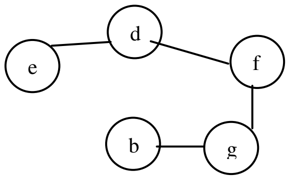
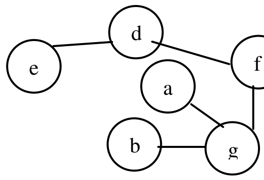
Q.74 What is a Spanning tree of a graph? What is minimum spanning tree? Execute Prim's Kruskal's algorithm to find the minimum spanning tree of the following graph. **(10)**



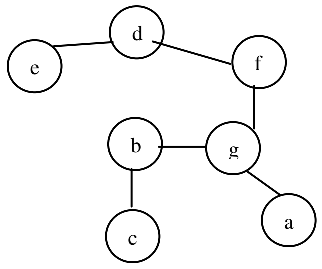
Ans:

A Spanning Tree is any tree consisting of vertices of graph tree and some edges of graph is called a spanning tree.

If graph has n vertices then spanning tree has exactly n-1 edges.
A minimum spanning tree of an undirected graph *G* is a tree formed from graph edges that connects all the vertices of *G* at lowest total cost. A minimum spanning tree exists if and only if *G* is connected.
Kruskal's algorithm to find minimum spanning tree of given graph:

S.No.	Edge	Cost	Action	Spanning Tree T
				T={φ}
1.	(d,f)	2	Add to T	
2.	(f,g)	2	Add to T	
3.	(d,g)	3	Discard as it forms a cycle	
4.	(g,b)	4	Add to T	
5.	(b,d)	5	Add to T	
6.	(d,e)	6	Add to T	
7.	(e,f)	7	Discard as it forms a cycle	
8.	(e,g)	8	Discard as it forms a cycle	
9.	(a,g)	9	Add to T	

10. (b,c) 10 Add to T



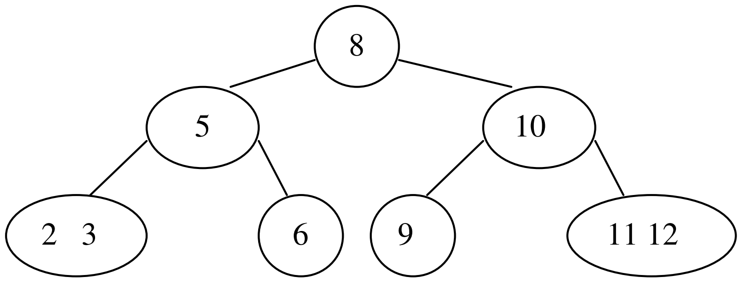
T is required spanning tree.

Q.75 What are B-trees? Draw a B-tree of order 3 for the following sequence of keys. 3,5,11,10,9,8,2,6,12 **(6)**

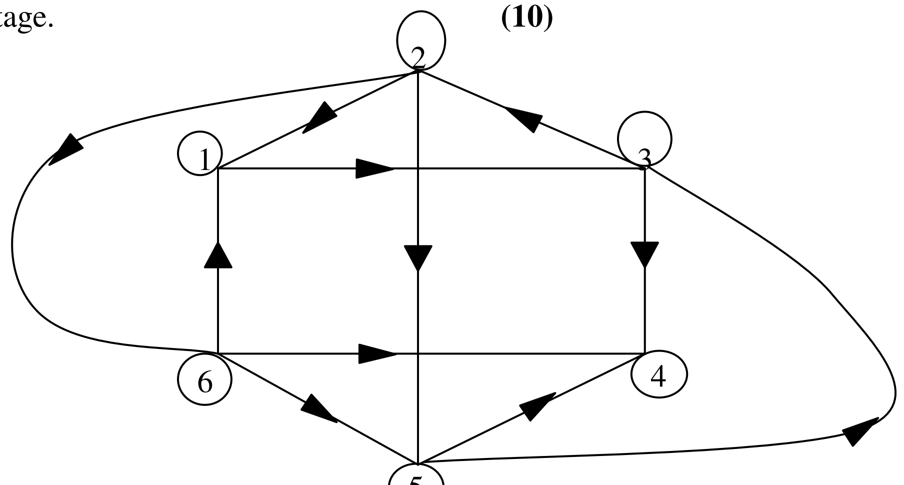
Ans:
A B-tree is a balanced multiway search tree.
A *B-Tree of order M* is either the empty tree or it is an *M*-way search tree *T* with the following properties:

1. The root of *T* has at least two subtrees and at most *M* subtrees.
2. All internal nodes of *T* (other than its root) have between [M / 2] and *M* subtrees.
3. All external nodes of *T* are at the same level.

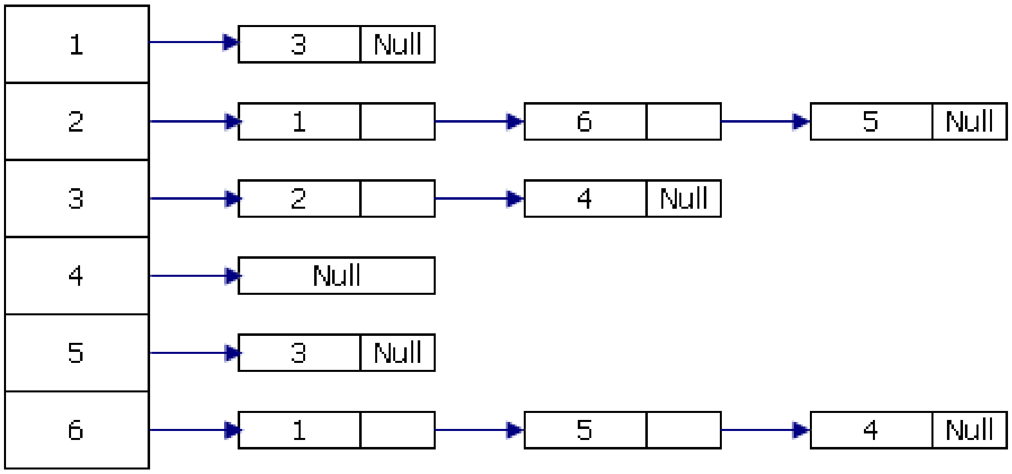
A B-tree of order one is clearly impossible.



Q.76 Show the result of running BFS and DFS on a directed graph given below using vertex 1 as source. Show the status of the data structure used at each stage. **(10)**



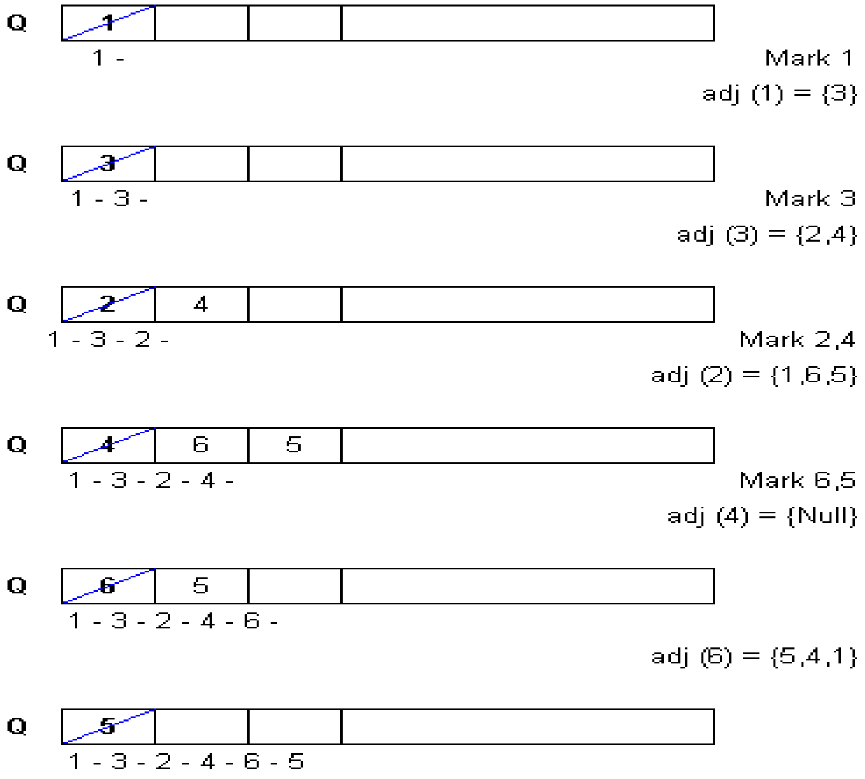
Ans: Depth First Transversal



1 –	Mark 1
1 – 3	Mark 3
1 – 3 – 2	Mark 2

1 is marked, so visit 6
1 – 3 – 2 – 6
Move to adj list of 6
1 is already marked
1 – 3 – 2 – 6 – 5
Move to adj list of 5
3 is already marked
So move to adj list of 6 again
Depth First Transversal is:
1 – 3 – 2 – 6 – 5 – 4

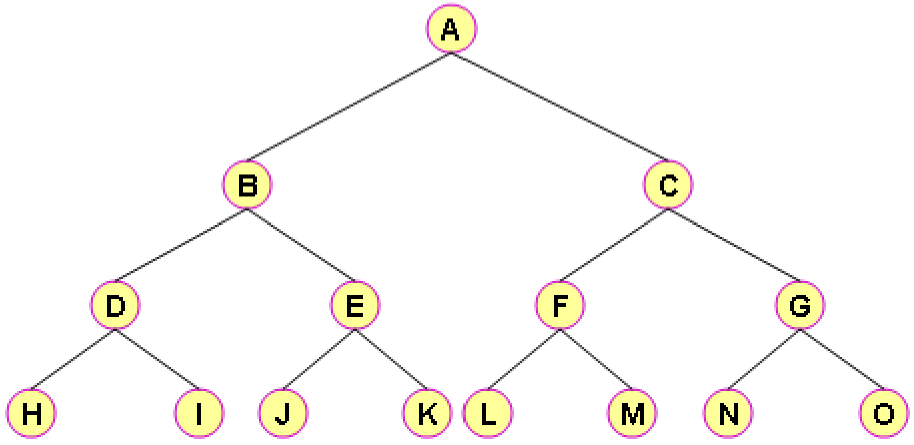
Breadth First Transversal



Breadth First Transversal is:
1 – 3 – 2 – 4 - 6 – 5

Q.77 Construct a complete binary tree with depth 3 for this tree which is maintained in memory using linked representation. Make the adjacency list and adjacency matrix for this tree. (6)

Ans:



Pointer	Info.	Left	Right
1	A	2	3
2	B	4	5
3	C	6	7
4	D	8	9
5	E	10	11
6	F	12	13
7	G	14	15
8	H	0	0
9	I	0	0
10	J	0	0
11	K	0	0
12	L	0	0
13	M	0	0
14	N	0	0
15	O	0	0

Q.78 Define data type and abstract data type. Comment upon the significance of both. **(8)**

Ans:
We determine the amount of memory to reserve by determining the appropriate abstract data type group to use and then deciding which abstract data type within the group is right for the data.
There are four data type groups:

- **Integer** Stores whole numbers and signed numbers.

- **Floating-point** Stores real numbers (fractional values). Perfect for storing bank deposits where pennies (fractions of a dollar) can quickly add up to a few dollars.
- **Character** Stores a character. Ideal for storing names of things.
- **Boolean** Stores a true or false value. The correct choice for storing a yes or no or true or false response to a question.

Abstract Data Types:-

A useful tool for specifying the logical properties of a data type is the abstract data type or ADT. The term "abstract data type" refers to the basic mathematical concept that defines the data type. In defining ADT as a mathematical concept, we are not concerned with space or time efficiency.

An ADT consists of two parts:- a value definition and an operator definition. The value definition defines the collection of values for the ADT and consists of two parts: a definition clause and a condition clause. For example, the value consist definition for the ADT RATIONAL states that a RATIONAL value consists of two integers, the second of which does not equal 0. We use array notation to indicate the parts of an abstract type.

Q.79 Write a procedure to reverse a singly linked list. **(8)**

Ans:

A procedure to reverse a singly linked list:

```
reverse(struct node **st)
{
struct node *p, *q, *r;
p = *st;
q = NULL;
while(p != NULL)
{
 r =q;
 q = p;
 p = p → link;
 q → link = r;
}
*st = q;
}
```

Q.80 Execute your algorithm to convert an infix expression to a post fix expression with the following infix expression as input

$$Q = [(A + B)/(C + D) \uparrow (E/F)] + (G + H)/I$$ **(8)**

Ans:

Given infix expression is:-

Q = [(A + B) / (C + D) ^ (E + F)] + (G + H) / I

Symb	Stack	Postfix Notation
[	[	
(	[(	
A	[(	A

+	[(+	
B	[(+	AB
)	[	AB+
/	[/	
(	[/ (	
C	[/ (	AB+C
+	[/ (+	
D	[/ (+	AB+CD
)	[/	AB+CD+
^	[/ ^	
(	[/ ^ (	
E	[/ ^ (	AB+CD+
/	[/ ^ (/	AB+CD+
F	[/ ^	AB+CD+/
]	EMPTY	AB+CD+EF/^/
+	+	
(	+ (	
G	+ (	AB+CD+EF/^/G
+	+ (+	
H	+ (+	AB+CD+EF/^/GH
)	+	AB+CD+EF/^/GH+
/	+ /	AB+CD+EF/^/GH+
I	+ /	AB+CD+EF/^/GH+I

POSTFIX:- **AB+CD+EF/^/GH+I/+**

Q81 Enumerate various operations possible on ordered lists and arrays. Write procedures to insert and delete an element in to array. **(8)**

Ans:
An ordered list is a container which holds a *sequence* of objects. Each object has a unique *position* in the sequence. In addition to the basic repertoire of operations supported by all searchable containers, ordered lists provide the following operations:
FindPosition
used to find the position of an object in the ordered list;
operator []
used to access the object at a given position in the ordered list;
Withdraw(Position&)
used to remove the object at a given position from the ordered list.
InsertAfter
used to insert an object into the ordered list *after* the object at a given position; and
InsertBefore
used to insert an object into the ordered list *before* the object at a given position.
Procedures to insert and delete an element in to an array:-

```
void insert ( int *arr, int pos, int num )
/* inserts an element num at given position pos */
{
 /* shift elements to right */
 int i ;
 for ( i = MAX - 1 ; i >= pos ; i-- )
    arr[i] = arr[i - 1] ;
 arr[i] = num ;
}
void del ( int *arr, int pos )
/* deletes an element from the given position pos */
{
 /* skip to the desired position */
 int i ;
 for ( i = pos ; i < MAX ; i++ )
    arr[i - 1] = arr[i] ;
 arr[i - 1] = 0 ;
}
```

Q.82 By taking an example show how multidimensional array can be represented in one dimensional array. **(8)**

Ans:

Multidimensional array: Multidimensional arrays can be described as "arrays of arrays". For example, a bidimensional array can be imagined as a bidimensional table made of elements, all of them of a same uniform data type.

int arr[3][5]; represents a bidimensional array of 3 per 5 elements of type `int`.

Similarly a three dimensional array like

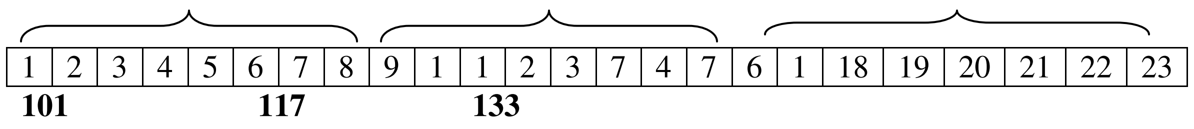
int arr[3][4][2]; represent an outer array of three elements , each of which is a two dimensional array of four rows, each of which is a one dimensional array of five elements.

Multidimensional arrays are not limited to two or three indices (i.e., two dimensional or three dimensional). They can contain as many indices as needed. However the amount of memory needed for an array rapidly increases with each dimension. For example:

char arr [100][365][24][60][60];declaration would consume more than 3 gigabytes of memory.

Memory does not contain rows and columns, so whether it is a one dimensional array or two dimensional arrays, the array elements are stored linearly in one continuous chain. For example, the multidimensional array

```
int arr[3][4][2]= {
            {  {1,2},{3,4},{5,6},{7,8} },
                     {  {9,1},{1,2},{3,7},{4,7} },
                      {  {6,1},{18,19},{20,21},{22,23} },
          };  is stored in memory just like an one-dimensional array shown below:
```



Multidimensional arrays are just an abstraction for programmers, since we can obtain the same results with a simple array just by putting a factor between its indices.

Q.83 Show the various passes of bubble sort on an unsorted list 11, 15, 2, 13, 6 **(8)**

Ans:
The given data is:-

```
Pass 1:-  11  15  2   13  6
          11  15  2   13  6
          11  2   15  13  6
          11  2   13  15  6
          11  2   13  6   [15]
Pass 2:-  11  2   13  6   15
          2   11  13  6   15
          2   11  13  6   15
          2   11  6   [13 15]
Pass 3:-  2   11  6   13  15
          2   11  6   13  15
          2   6   [11 13  15]
Pass 4:-  2   6   11  13  15
          2   [6  11  13  15]
```

Therefore, the sorted array is:-

```
          2   6   11  13  15
```

Q.84 Compare and contrast various sorting techniques with respect to memory space and computing time. **(8)**

Ans:
Insertion sort:- Because of the nested loops, each of which can take *n* iterations, insertion sort is $O(n^2)$. Furthermore, this bound is tight, because input in reverse order can actually achieve this bound. So complexity= (n(n-1))/2 = $O(n^2)$.
Shellsort: - The running time of Shellsort depends on the choice of increment sequence. *The worst-case running time of Shellsort, using Shell's increments, is* (n^2).
Heapsort:- The basic strategy is to build a binary heap of *n* elements. This stage takes *O(n)* time. We then perform *n delete_min* operations. The elements leave the heap smallest first, in sorted order. By recording these elements in a second array and then copying the array back, we sort *n* elements. Since each *delete_min* takes $O(\log n)$ time, the total running time is $O(n \log n)$.
Mergesort:- Mergesort is a classic example of the techniques used to analyze recursive routines. We will assume that *n* is a power of 2, so that we always split into even halves. For $n = 1$, the time to mergesort is constant, which we will

denote by 1. Otherwise, the time to mergesort *n* numbers is equal to the time to do two recursive mergesorts of size *n*/2, plus the time to merge, which is linear. The equations below say this exactly:

$T(1) = 1$

$T(n) = 2T(n/2) + n$

Quicksort:- Like mergesort, quicksort is recursive, and hence, its analysis requires solving a recurrence formula. We will do the analysis for a quicksort, assuming a random pivot (no median-of-three partitioning) and no cutoff for small files. We will take $T(0) = T(1) = 1$, as in mergesort. The running time of quicksort is equal to the running time of the two recursive calls plus the linear time spent in the partition (the pivot selection takes only constant time). This gives the basic quicksort relation

$$T(n) = T(i) + T(n - i - 1) + cn$$

Q.85 What do you mean by hash clash? Explain in detail any one method to resolve hash collisions. **(8)**

Ans:

Hashing is not perfect. Occasionally, a collision occurs when two different keys hash into the same hash value and are assigned to the same array element. Programmers have come up with various techniques for dealing with this conflict.

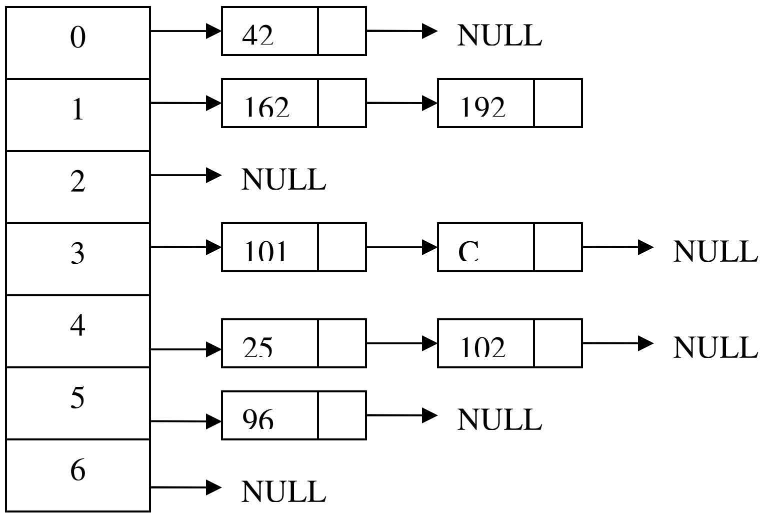
A common way to deal with a collision is to create a linked list of entries that have the same hash value. For example, let the key of each entry hashes to the same hash value and this result in both being assigned to the same array element of the hashtable.

Because two entries cannot be assigned the same array element, the programmer creates a linked list. The first user-defined structure is assigned to the pointer in the array element. The second isn't assigned to any array element and is instead linked to the first user-defined structure, thereby forming a linked list.

For example: in a table size of 7

42, 56 both are mapped to index 0 as 42%7=0 and 56%7=0.

25, 42, 96, 101, 102, 162, 197 can be mapped as follows:



Q.86 Describe the concept of binary search technique? Is it efficient than sequential search? **(8)**

Ans*:*

Binary search technique:-

This technique is applied to an ordered list where elements are arranged either in ascending order or descending order. The array is divided into two parts and the item being searched for is compared with item at the middle of the array. If they are equal the search is successful. Otherwise the search is performed in either the first half or second half of the array. If the middle element is greater than the item being searched for the search process is repeated in the first half of the array otherwise the process is repeated in the second half of the array. Each time a comparison is made, the number of element yet to be searched will be cut in half. Because of this division of array into two equal parts, the method is called a binary search.

In binary search, each iteration of the loop halves the size of the list to be searched. Thus the maximum number of key comparisons is approximately $2*\log_2 n$. This is quite an improvement over the sequential search, especially as the list gets larger. The binary search is, however, not guaranteed to be faster for searching for small lists. Even though the binary search requires less number of comparisons, each comparison requires more computation. When n is small, the constant of proportionality may dominate. Therefore, in case of less number of elements in the list, a sequential search is adequate.

Q.87 Prove the hypothesis that "A tree having 'm' nodes has exactly (m–1) edges or branches". **(8)**

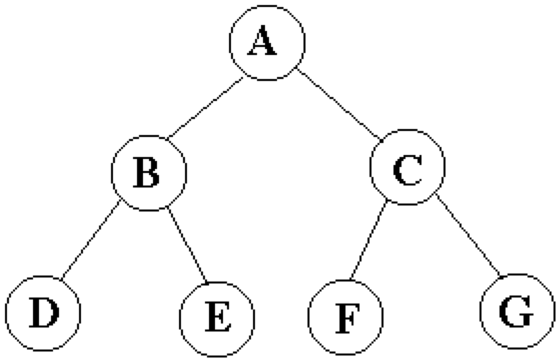
Ans:

A tree having m nodes has exactly (m-1) edges or branches

Proof: A root in a tree has indegree zero and all other nodes have indegree one. There must be (m-1) incoming arcs to the (m-1) non-root nodes. If there is any other arc, this arc must be terminating at any of the nodes. If the node is root, then its indegree will become one and that is in contradiction with the fact that root always indegree zero. If the end point of this extra edge

is any non-root node then its indegree will be two, which is again a contradiction. Hence there cannot be more arcs. Therefore, a tree with m vertices will have exactly (m-1) edges.

Q.88 What do you understand by tree traversal? Write a procedure for traversing a binary tree in preorder and execute it on the following tree. **(8)**



Ans:

The algorithm walks through the tree data structure and performs some computation at each node in the tree. This process of walking through the tree is called a *tree traversal.*

There are essentially two different methods in which to visit systematically all the nodes of a tree--*depth-first traversal* and *breadth-first traversal.* Certain depth-first traversal methods occur frequently enough that they are given names of their own: *preorder traversal*, *inorder traversal* and *postorder traversal.*

The algorithm for traversing a tree in preorder is as follows:-

```
i.      Process the root R.
ii.    Traverse the left sub tree of R in preorder.
iii.   Traverse the right sub tree of R in preorder.
```

The preorder for the given tree is:-

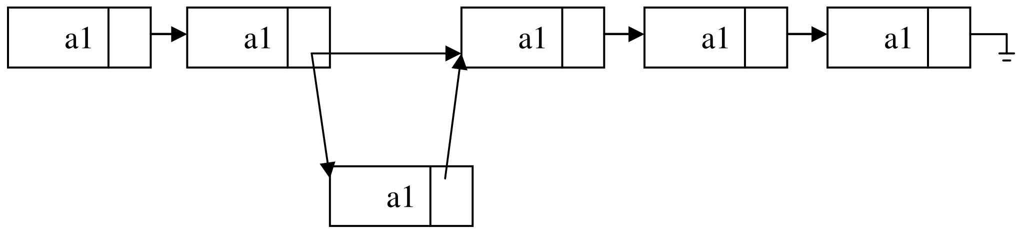
```
A  L1  R1
A  B   L1l  L1R  R1
A  B   D    L1R  R1
A  B   D    E    R1
A  B   D    E    C   R1L  R1R
A  B   D    E    C   F    R1R
A  B   D    E    C   F    G
```

Q.89 Write a procedure to insert a node into a linked list at a specific position and draw the same by taking any example? **(8)**

Ans:

```
* Insert (after legal position p).*/
 /* Header implementation assumed. */
 void insert( element_type x, LIST L, position p )
 {
 position tmp_cell;
  tmp_cell = (position) malloc( sizeof (struct node));
        if( tmp_cell == NULL )
              fatal_error("Out of space!!!");
 else
 {
              tmp_cell->element = x;
              tmp_cell->next = p->next;
              p->next = tmp_cell;
```

```
    }
    }
```

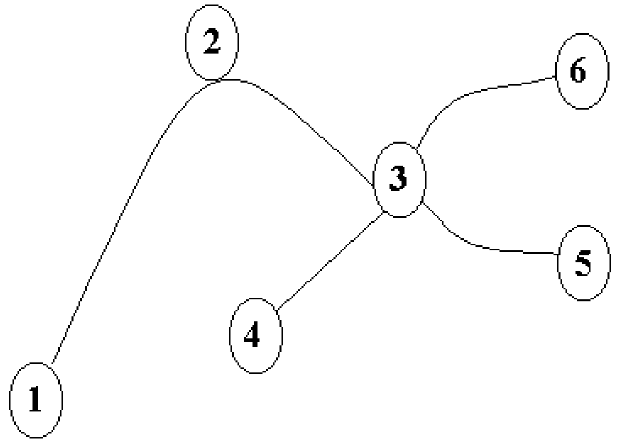


Q.90 Explain insertion into a B-tree. **(8)**

Ans:
Insertion into a B-tree:

1. First search is done for the place where the new record must be put. As the keys are inserted, they are sorted into the proper order.
2. If the node can accommodate the new record, insert the new record at the appropriate pointer so that number of pointers remains one more than the number of records.
3. If the node overflows because there is an upper bound on the size of a node, splitting is required. The node is split into three parts; the middle record is passed upward and inserted into parent, leaving two children behind. If n is odd (n-1 keys in full node and the new target key), median key int(n/2)+1 is placed in parent node, the lower n/2 keys are put in the left leaf and the higher n/2 keys are put in the right leaf. If n is even, we may have left biased or right biased means one key may be more in left child or right child respectively.
4. Splitting may propagate up the tree because the parent, into which splitted record is added, may overflow then it may be split. If the root is required to be split, a new record is created with just two children.

Q.91 Define adjacency matrix and make the same for the following undirected graph. **(8)**



Ans:
Adjacency Matrix representation: This representation consists of |V|*|V| matrix A=(a_{ij}) such that

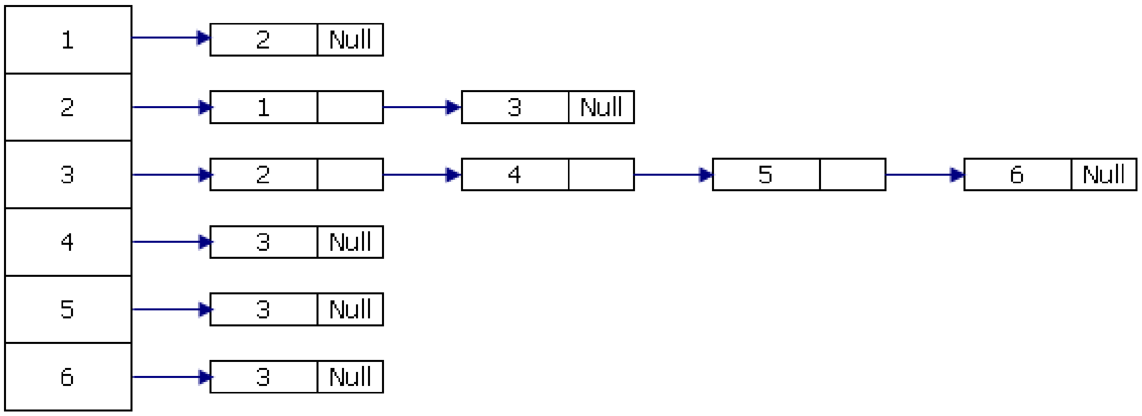
$$a_{ij} = \begin{cases} 1 \text{ if}(i,j) \varepsilon \text{ E} \\ 0 \text{ otherwise} \end{cases}$$

The adjacency matrix of given graph is given below:

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 1 & 1 \\ 4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Q.92 Show the linked representation of the above graph. **(8)**

Ans: *The linked representation of given graph is:*



Q.93 List various problem solving techniques. **(5)**

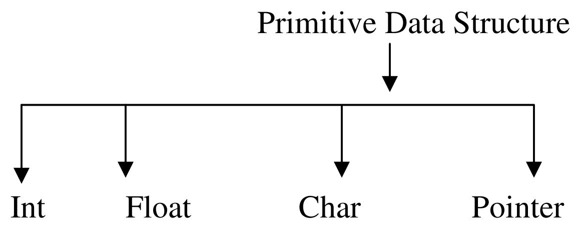
Ans.
Problem Solving Technique
There are two approaches to solve a problem. They are:-

1. **Top down**-A top down_design approach starts by identifying the major components of the system, decomposing them into their lower-level components & iterating until the desired level of detail is achieved it often result into some form of stepwise refinement stating from an abstract design, in each step the design is refined to more concrete level , until we reach a level where no more refinement is needed & the design can be implemented directly. It is suitable only if the specifications of the system are clearly known (waterfall model). Top down approaches require some idea about the feasibility of components specified during design.
2. **Bottom- up-** A bottom- up design approach starts with designing the most basic or primitive components & proceeds to the higher- level components that use these lower- level components. Bottom-up methods work with layer of abstraction starting from the very bottom, operations that provide a layer abstractions are implemented .The operations of this layer are then used to implement more powerful operations & a still higher layer of abstraction, until the stage is reached where the operations supported by the layer are those desired by the system. If a system is to is to be built from an existing system approach bottom-up is more suitable.

Here, pure top-down or pure bottom-up approaches are often not practical .A common approach to combine he two approaches is to provide a layer of abstraction for the application domain of interest through libraries of functions, which contain the functions of interest to the application domain.

Q94. Explain the concept of primitive data structures. **(4)**

Ans.
<u>**Primitive Data Structure**</u>
These are the basic structure and are directly operated upon by the machine instructions. These in general have different representations on different computer. Integer floating point number, character constraints, string constants, pointer etc fall in this category.



Q95. Determine the frequency counts for all statements in the following program segment.

```
for (i=1; i ≤ n; i ++)
for (j = 1; j ≤ i;  j++)
for (k =1; k ≤ j; k++)
y ++;
```
(3)

Ans.
```
S1:  for (i=1; i<= n; i++)
S2:  for (j=1; j,= i; j++)
```

```
S3 : for (k=1; k<= j; k++)
S4 :  y++;
Frequency counts of
         S1=n
         S2 = 1+ (1+2)+ (1+2+3) +……. (1+2+….n)
         S3 = 1+ [ 1+ (1+2)] + …..[ 1+ (1+2) +….
              (1+2+3+…n)]
         S4= same as S3.
```

Q96 . Write an algorithm to count number of nodes in the circular linked list. **(3)**

Ans.

Counting No of Nodes in Circular List

```
Let list be a circular header list in memory. This
algorithm traverse and counts number of nodes in a
LIST.
   0.   set COUNT: = 0
   1. Set PTR: = LINK [START]. {Initializes the
      pointer PTR}
   2. Repeat steps 3,/ 4, 5 while PTR   =   START;
   3. COUNT = COUNT + 1
   4. Set PTR = LINK [PTR].  [PTR now points to next
      node]
      [End of step 2 loop]
   5. Exit
```

Q97. Write an algorithm to insert a node in between any two nodes in a linked list **(4)**

Ans.

Insertion of a node after the given element of the list

```
Node *  InsertAfterElement (node * start, int item,
int after)
{
   node * ptr, * loc;
   ptr =(node *) malloc (sizeof (node));
   ptr->info = item;
loc=start;
while (loc != NULL && loc->info != after)
loc= loc->next;
if (start==NULL) { ptr->next= NULL;
start = ptr;}
else if (loc l = NULL){
ptr->next = loc->next;
loc->next= ptr;}
return start;
}
```

Q98. Write down any four application of a stack. **(4)**

Ans.

(i) Conversion of infix to postfix form
(ii) Reversing of a line.
(iii) Removal of recursion
(iv) Evaluating post fix expression

Q99. The system allocates memory for any multidimensional array from a large single dimensional array. Describe *two* mapping schemes that helps us to store a *two* dimensional metrics in a *one-dimensional* array. **(8)**

Ans.

A two dimensional array is declared similar to the way we declare a one-dimensional array except that we specify the number of elements in both dimensions. For example,
int grades[3][4];
The first bracket ([3]) tells the compiler that we are declaring 3 pointers, each pointing to an array. We are not talking about a pointer variable or pointer array. Instead, we are saying that each element of the first dimension of a two dimensional array reference a corresponding second dimension. In this example, all the arrays pointed to by the first index are of the same size. The second index can be of variable size. For example, the previous statement declares a two-dimensional array where there are 3 elements in the first dimension and 4 elements in the second dimension.
Two-dimensional array is represented in memory in following two ways:
1. Row major representation: To achieve this linear representation, the first row of the array is stored in the first memory locations reserved for the array, then the second row and so on.
2. Column major representation: Here all the elements of the column are stored next to one another. In row major representation, the address is calculated in a two dimensional array as per the following formula
The address of a[i][j]=base(a)+(i*m+ j)*size,where base(a) is the address of a[0][0], m is second dimension of array a and size represent size of the data type.

Q100. Write down the algorithm of quick sort.

Ans.

An algorithm for quick sort:

```
void quicksort ( int a[ ], int lower, int upper )
{
  int i ;
  if ( upper > lower ) {
      i = split ( a, lower, upper ) ;
      quicksort ( a, lower, i - 1 ) ;
      quicksort ( a, i + 1, upper ) ;  }
}
int split ( int a[ ], int lower, int upper ){
  int i, p, q, t ;
  p = lower + 1 ;
```

```
      q = upper ;
      i = a[lower] ;
      while ( q >= p )
      {
         while ( a[p] < i )
            p++ ;
         while ( a[q] > i )
            q-- ;
         if ( q > p )
         {
            t = a[p] ;
            a[p] = a[q] ;
            a[q] = t ;
         }
      }
      t = a[lower] ;
      a[lower] = a[q] ;
      a[q] = t ;
      return q ; }
```

Q101. Convert the following Infix expression to Postfix form using a stack: *x + y * z + (p * q + r) * s*, Follow usual precedence rule and assume that the expression is legal. **(7)**

Ans.

The given infix expression is :-

*x + y * z + (p * q + r) * s*

Symb	Stack	Postfix Expression
x	empty	x
+	+	x
y	+	xy
*	+*	xy
z	+*	xyz
+	+	xyz*+
(	+(	xyz*+
p	+(	xyz*+p
*	+(*	xyz*+p
q	+(*	xyz*+p
+	+(+	xyz*+pq*
r	+(+	xyz*+pq*r
)	+	xyz*+pq*r+
*	+*	xyz*+pq*r+
s	+*	xyz*+pq*r+s
		= xyz*+pq*r+s*+

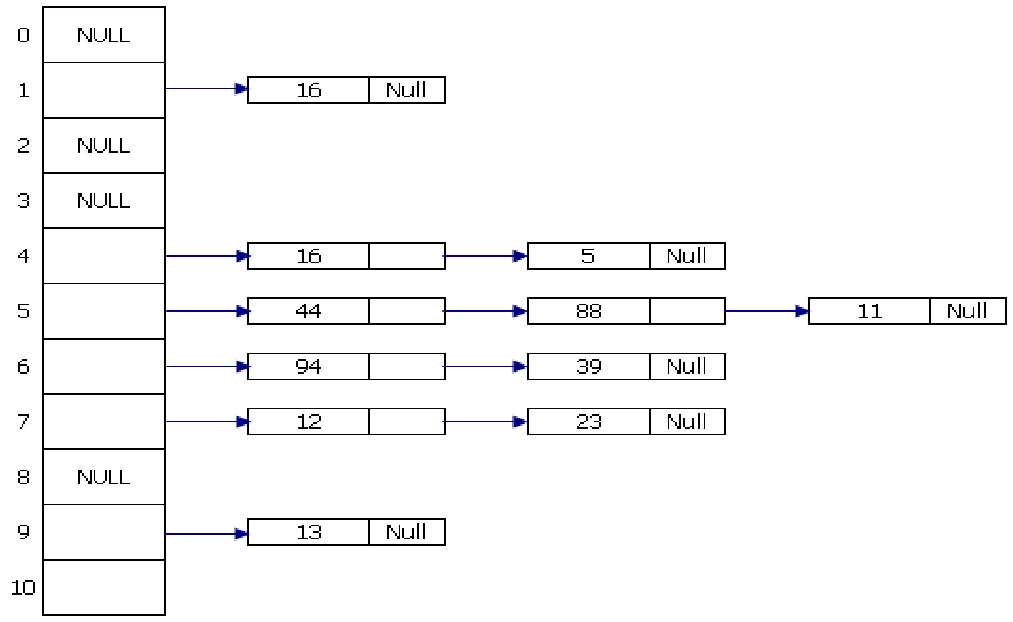
The postfix expression is :- ***xyz*+pq*r+s*+***

Q102. Draw the 11 item hash table resulting from hashing the keys: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16 and 5 using the hash function h(i) = (2i+5) mod 11. **(8)**

Ans.

table size is 11

h(12) = (24+5) mod 11 = 29 mod 11 = 7
h(44) = (88+5) mod 11 = 93 mod 11 = 5
h(13) = (26+5) mod 11 = 31 mod 11 = 9
h(88) = (176+5)mod 11= 181mod 11 = 5
h(23) = (46+5) mod 11 = 51 mod 11 = 7
h(94) = (188+5)mod 11= 193mod 11 = 6
h(11) = (22+5) mod 11 = 27 mod 11 = 5
h(39) = (78+5) mod 11 = 83 mod 11 = 6
h(20) = (40+5) mod 11 = 45 mod 11 = 1
h(16) = (24+5) mod 11 = 29 mod 11 = 4
h(5) = (10+5) mod 11 = 15 mod 11 = 4



Q103. Give the algorithm to construct a binary tree where the yields of preorder and post order traversal are given. **(6)**

Ans.

Tree cannot be constructed using Preorder and Post order traversal of a binary tree. For this Inorder traversal has to be given.

Therefore to construct a tree using Inorder and preorder traversal (as an example, traversals of Q 104 are used), the algorithm is as follows:

The tree T is drawn from the root downward as follows:

1) The root of T is obtained by choosing the first node in its preorder. Thus G is the root of the node.

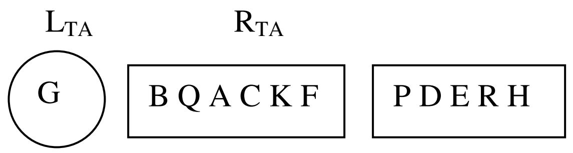
2) The left child of the node G is obtained as follows. First use the inorder of T to find the node in the left subtree L_{TA} of G. Thus L_{TA} consists of the nodes B,Q,A,C,K,F. Then the left child of G is obtained by choosing the first node in the preorder of L_{TA} (which appears in the Preorder of T) . Thus B is the left son of G.

3) Similarly the right subtree R_{TA} of G consists of the nodes P,D,E,R,H and P is the root of R_{TA}, that is P is the right child of G.
Repeating the above process with each new node, we finally obtain the required tree.

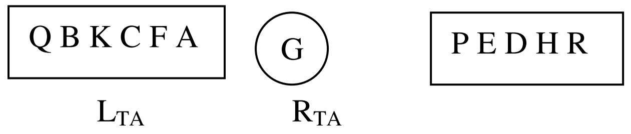
Q104. Construct the binary tree for the following sequence of nodes in preorder and inorder respectively.
Preorder : G, B, Q, A, C, K, F, P, D, E, R, H
Inorder: Q, B, K, C, F, A, G, P, E, D, H, R **(4)**

Ans.

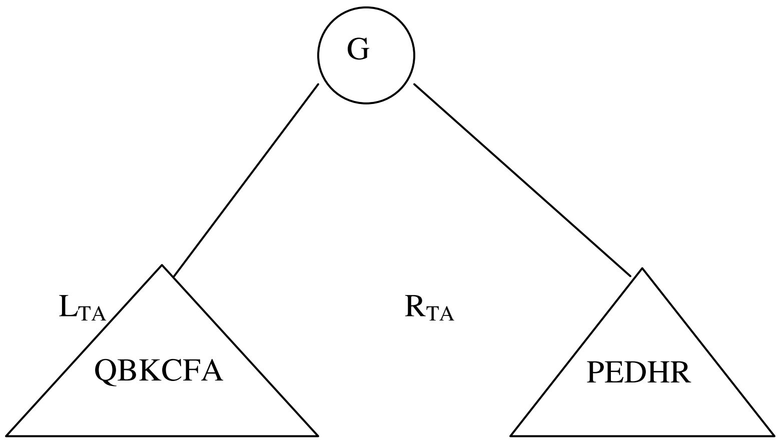
Preorder:



Inorder:



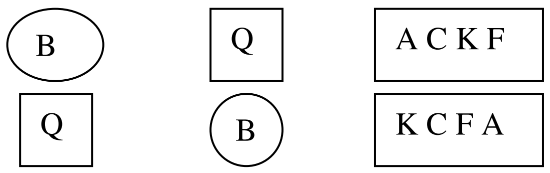
Root of the tree is G



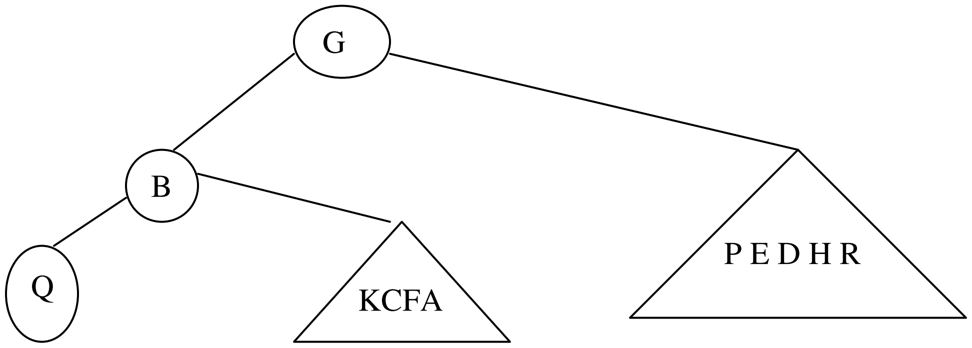
Consider L_{TA}

Pre:

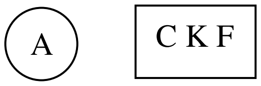
In :



Root B

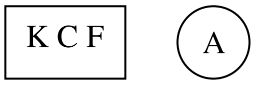


Let us consider R_{TB}

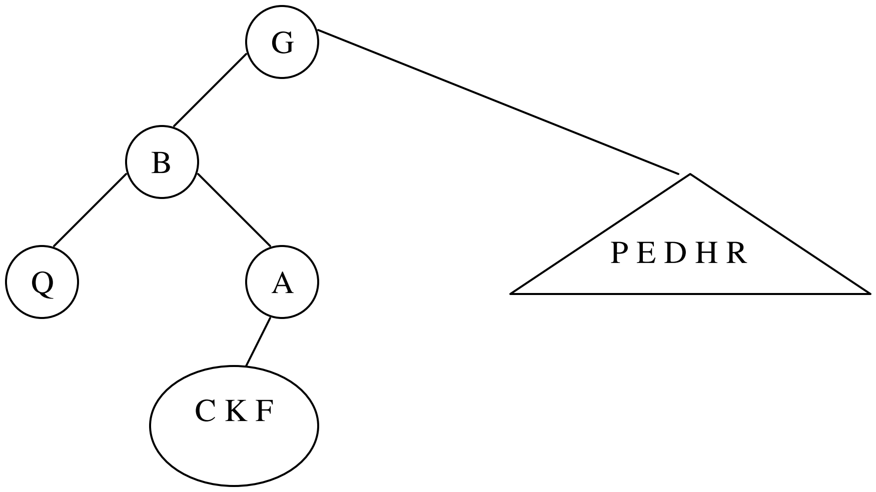


Pre :

In:



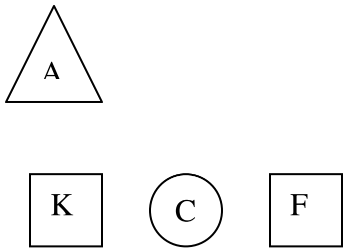
Root is A

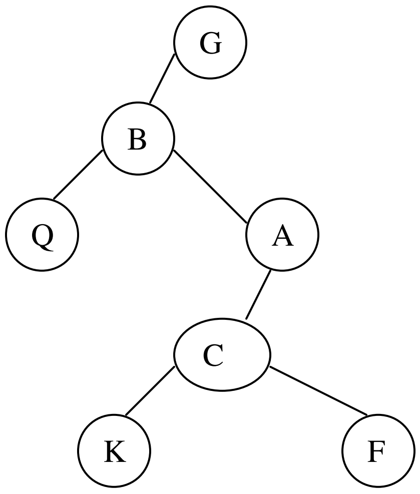


Let us consider

Pre:

In :

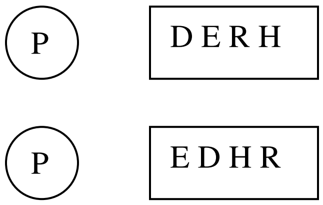


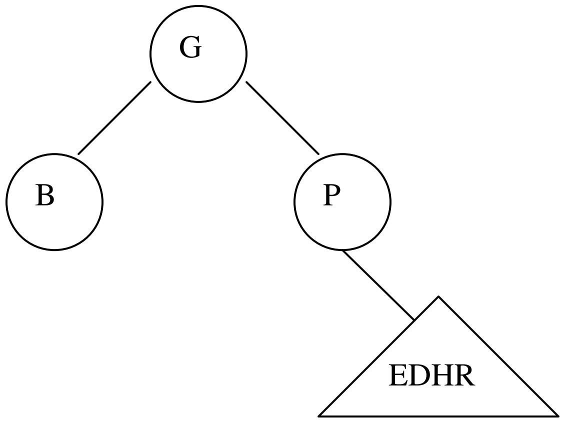


Now let us consider R_{TA}

Pre:

In:

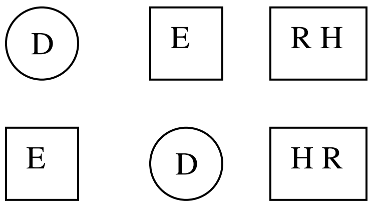


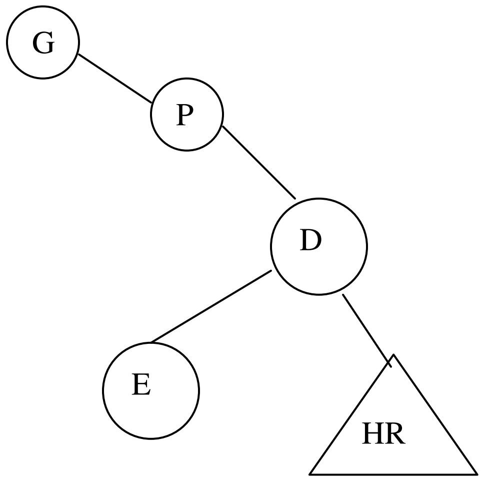


Let us consider R_{TE}

Pre :

In:

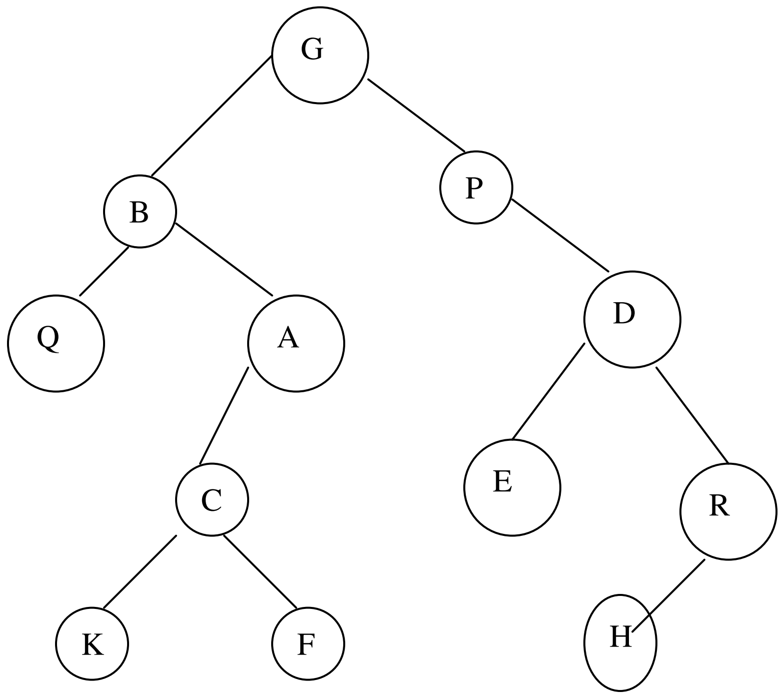




let us now consider HR

Pre: R H

In: H R

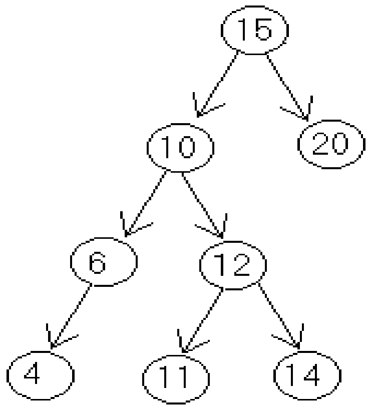


Q105. Convert the following infix expression into a postfix expression (Show steps)
$A * (B + D) / E - F(G + H / k)$ **(4)**

Ans.
Infix to Post fix

Symbol	**Postfix String**	**Opstk**
A	A	
*	A	*
(	A	*(
B	AB	*(
+	AB	*(+
D	ABD	*(+
)	ABD+	*(
/	ABD+*	/
E	ABD+*E	/
-	ABD+*E/	-
F	ABD+*E/F	-
*	ABD+*E/F	-
(	ABD+*E/F	-(
G	ABD+*E/FG	-(
+	ABD+*E/FG	-(+
H	ABD+*E/FGH	-(+
/	ABD+*E/FGH	-(+/
K	ABD+*E/FGHK	-(+/
)	ABD+*E/FGHK/+-	

Q106. Write an algorithm to delete a particular node from binary search tree. Trace your algorithm to delete a node (10) from the given tree.



(7)

Ans.

Algorithm To Delete Node From Binary Search Tree

To delete a node following possibilities may arise

- Node id a terminal node
- Node have only one child
- Node having 2 children.

```
DEL(INFO, LEFT, RIGT, ROOT, AVAIL, ITEM)
```

A binary search tree T is in memory, and an ITEM of information is given. This algorithm deletes ITEM from the tree.

```
1. [ Find the locations of ITEM and its parent]
Call FIND(INFO, RIGHT, ROOT, ITEM, LOC, PAR).
2. [ITEM in tree?]
if LOC=NULL, then write : ITEM not in tree, and Exit.
3. [Delete node containing ITEM.]
if RIGHT[LOC] != NULL and LEFT[LOC] !=NULL then:
Call CASEB(INFO,LEFT,RIGHT,ROOT,LOC,PAR).
Else:
Call CASEA (INFO,LEFT,RIGHT,ROOT,LOC,PAR).
[End of if structure.]
4. [Return deleted node to AVAIL list.]
Set LEFT[LOC]:=AVAIL and AVAIL:=LOC.
5. Exit.
CASEB(INFO,LEFT,RIGHT,ROOT,LOC,PAR)
```

This procedure will delete the node N at LOC, where N has two children. The pointer PAR gives the location of the parent of N, or else PAR=NULL indicates that N is the root node. The pointer SUC gives the location of the inorder successor of N, and PARSUC gives the location of the parent of the inorder successor.

```
1. [Find SUC and PARSUC.]
(a) Set PTR: = RIGHT[LOC] and SAVE:=LOC.
(b) Repeat while LEFT[PTR] ≠ NULL:
```

```
Set SAVE:=PTR and PTR:=LEFT[PTR].
   [End of loop.]
(c) Set SUC : = PTR and PARSUC:=SAVE.
2. [Delete inorder successor]
Call CASEA (INFO, LEFT, RIGHT, ROOT, SUC, PARSUC).
3. [Replace node N by its inorder successor.]
(a) If PAR≠NULL, then:
If LOC = LEFT[PAR], then:
 Set LEFT[PAR]:=SUC.
Else:
 Set RIGHT[PAR]: = SUC.
[End of If structure.]
   Else:
      Set ROOT: = SUC.
   [End of If structure.]
(b) Set LEFT[SUC]:= LEFT [LOC] and
RIGHT[SUC]:=RIGHT[LOC]
4. Return.
CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
```

This procedure deletes the node N at location LOC, where N does not have two children. The pointer PAR gives the location of the parent of N, or else PAR=NULL indicates that N is the root node. The pointer CHILD gives the location of the only child of N, or else CHILD = NULL indicates N has no children.

```
1. [Initializes CHILD.]
If LEFT[LOC] = NULL and RIGHT[LOC] = NULL, then:
 Set CHILD:=NULL.
Else if LEFT[LOC]≠NULL, then:
 Set CHILD: = LEFT[LOC].
Else
 Set CHILD:=RIGHT[LOC]
[End of If structue.]
2. If PAR ≠ NULL, then:
If LOC = LEFT [PAR], then:
 Set LEFT[PAR]:=CHILD.
      Else:
      Set RIGHT[PAR]:CHILD = CHILD
      [End of If structure.]
      Else:
      Set ROOT : = CHILD.
      [End of If structure.]
3. Return.
Inorder traversal of the tree is
4 6 10 11 12 14 15 20
To delete 10
PAR = Parent of 10 ie 15
SUC = inorder succ of 10 ie. 11
PARSUC = Parent of inorder succ ie 12
PTR = RIGHT [LOC]
```

```
 Address of 12     SAVE: = address of 10
SAVE: = address of 12
PTR = address of 11
SUC = ADDRESS OF 11
PAR SUCC:= ADDRESS OF 12
CHILD = NULL
LEFT [PARSUC] = CHILD= NULL
LEFT [PAR]= ADDRESS OF 11
LEFT [SUC] = LEFT [LOC] = ADDRESS OF 6
RIGHT [SUC] = RIGHT[LOC] = ADDRESS OF 12
```

Q107. Draw a picture of the directed graph specified below:

G = (V, E)

V(G) = {1, 2, 3, 4, 5, 6}

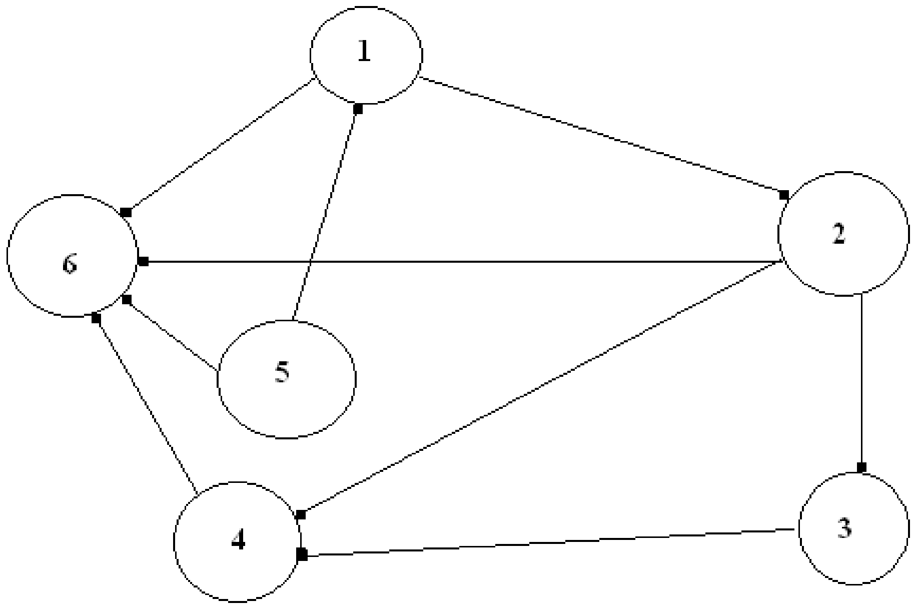
E(G) = {(1,2), (2, 3), (3, 4), (5,1), (5, 6), (2, 6), (1, 6), (4, 6), (2, 4)}

Obtain the following for the above graph:

(i) Adjacency matrix.

(ii) React ability matrix. **(7)**

Ans.



(i) Adjacency matrix

$$\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 0 & 0 & 1 \\
2 & 0 & 0 & 1 & 1 & 0 & 1 \\
3 & 0 & 0 & 0 & 1 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 1 \\
5 & 1 & 0 & 0 & 0 & 0 & 1 \\
6 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

(ii) **Reachability Matrix (Path Matrix)**

$$\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 1 \\
2 & 0 & 0 & 1 & 1 & 0 & 1 \\
3 & 0 & 0 & 0 & 1 & 0 & 1 \\
4 & 0 & 0 & 0 & 0 & 0 & 1 \\
5 & 1 & 1 & 1 & 1 & 0 & 1 \\
6 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Q108. Write an algorithm for binary search. What are the conditions under which sequential search of a list is preferred over binary search? **(7)**

Ans.

Algorithm for Binary Search

```
1.   if (low> high)
2.   return (-1)
3.   Mid = (low + high)/2
4.   if ( X = = a[mid])
5.   return (mid);
6.   if (X < a[mid])
7.   search for X in  a[low] to a[mid-1]
8.   else
9.   search for X in a[mid+1] to a[high]
```

Sequential Search is a preferred over binary search when the list is unordered and haphazardly constructed. When searching is to be performed on unsorted list then linear search is the only option.

Q109. Write an algorithm for selection sort. Describe the behaviours of selection sort when the input is already sorted. **(7)**

Ans.

Algorithm for Selection Sort

```
SELECTION (A, N)
This algorithm sorts the array A with N elements
1.  Repeat steps 2  & 3 for K = 1,2…….N-1
2.  Call MIN (A,K, N, LOC)
3.  [Interchange A[K] and A[LOC].]
Set TEMP : = A[K], A[K]:=A[LOC] and A[LOC]: = TEMP,
[END of Step 1 loop]
```

```
4.  EXIT
MIN (A, K, N, LOC)
An array A is in memory. This procedure finds the
location LOC of the smallest element among A[K],
A[K+1]……….A[N],
1.  Set MIN : =A[K] and LOC:=K  [Initialize pointer]
2.  Repeat for J= K+1, K+2, …..N;
If MIN > A[J], then : set MIN: = A[J] and LOC:=J
[End of loop]
3.  Return
```

The complexities of Selection sort are

Worst Case	Average Case	Best Case
$O(n^2)$	$O(n^2)$	$O(n^2)$

So if the list is already sorted the it is the best case, then also the complexity of algorithm is $O(n^2)$. Therefore, there is no change in behaviors of selection sort if the list is already sorted.

Q110. Define the following terms:
(i) Abstract data type.
(ii) Column major ordering for arrays.
(iii) Adjacency multilist.
(iv) Game trees.

(14)

Ans.

(i) Abstract Data Type:- A useful tool for specifying the logical properties of data type is the abstract data type or ADT. A data type is a collection of values and a set of operation on those values. The term "ADT" refer to the basic mathematical concept that defined the data types. ADT is not concerned with implementation details at all. By specifying the mathematical and logical properties of data type or structures, the ADT is a useful guideline to implementation or a useful tool to programmers who wish to use the data type correctly.
An ADT consists of 2 parts: a value definition and an operation definition
The **value definition** defines the collection of values for the ADT and consists of 2 parts: a definition clause and a condition clause.
Operation definition: - Each operation is defined as an abstract form with 3 parts: a leader, the operational pre-condition and the post condition. Eg.

/* value definition */

```
abstract type def <integer, integer>RATIONAL;
condition RATIONAL [1]! = 0
/* operator definition */
abstract RATIONAL Makerational (a, b)
int a, b;
precondition b! = 0;
postcondition makerational[0] = = a;
      maKerational[1] = =b;
```

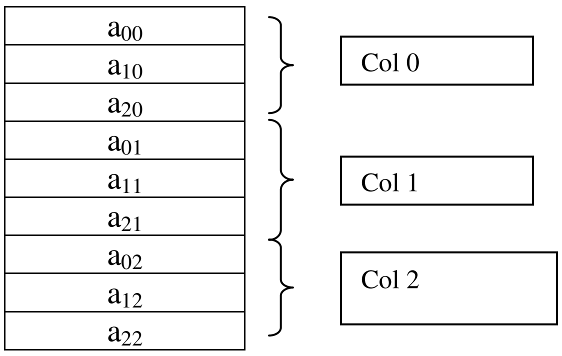
(ii) Column Major Ordering for Array

Let a be a 2 dimensional m x n array

	Col 0	Col 1	Col 2
Row 0	a_{00}	a_{01}	a_{02}
Row 1	a_{10}	a_{11}	a_{12}
Row 2	a_{20}	a_{21}	a_{22}

Though a is pictured as a rectangular pattern with in row and n column it is represented in memory by a Row of m x n elements .

Column major order:- the elements are stored column by column ie m elements of the 1st column is stored in first m locations, elements of the 2nd column are stored in next m location and so on.



Column major order:-

Loc (a[i][j]) = base (a) + w [m (j – lbc) + (i – lbr)]

Where w is number of bytes per storage location for any one element of the array.

M is the number of rows in an array.

lbr is lower bound for row index

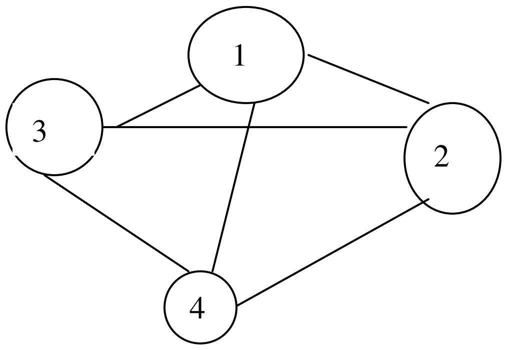
lbc is the lower bound for column index.

(iii) Adjacency Multilist

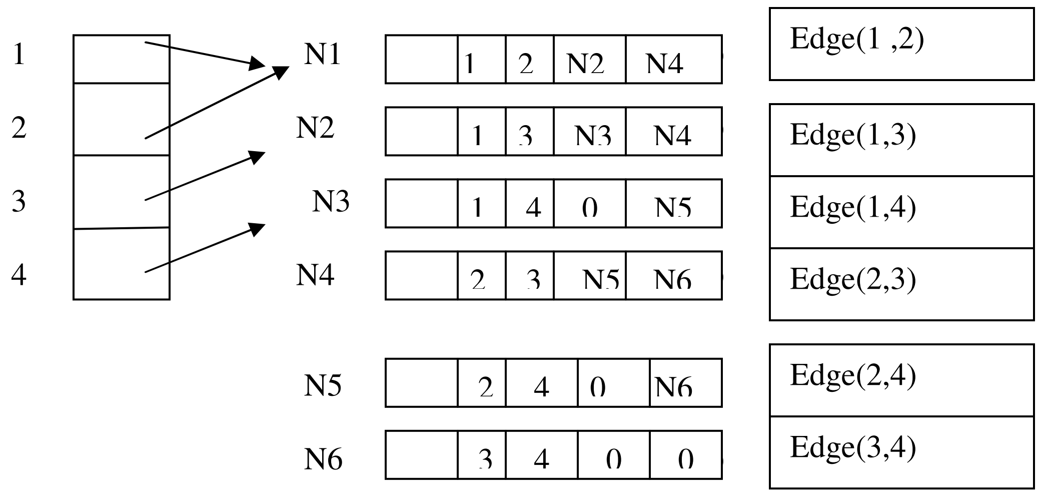
In the adjacency list representation of an undirected graph each edge (v_i , v_j) is represented by 2 entries, one on the list for v_i and the other list for v_j. In some situations it is necessary to be able to determine the 2nd entry for a particular edge and mark that edge as already having been examined. This can be accomplished easily if the adjacency lists are actually maintained as multilist (ie. List in which nodes may be shared among several lists). For each edge there will be exactly one node, but this node will be in 2 lists, i.e. the adjacency list for each of the 2 nodes it is incident. The node structure now becomes

M	V_1	V_2	Link for V_1	Link for V_2

Where M is a one bit mark field to indicate whether or no edge has been examined.
Let the graph be



Vertex



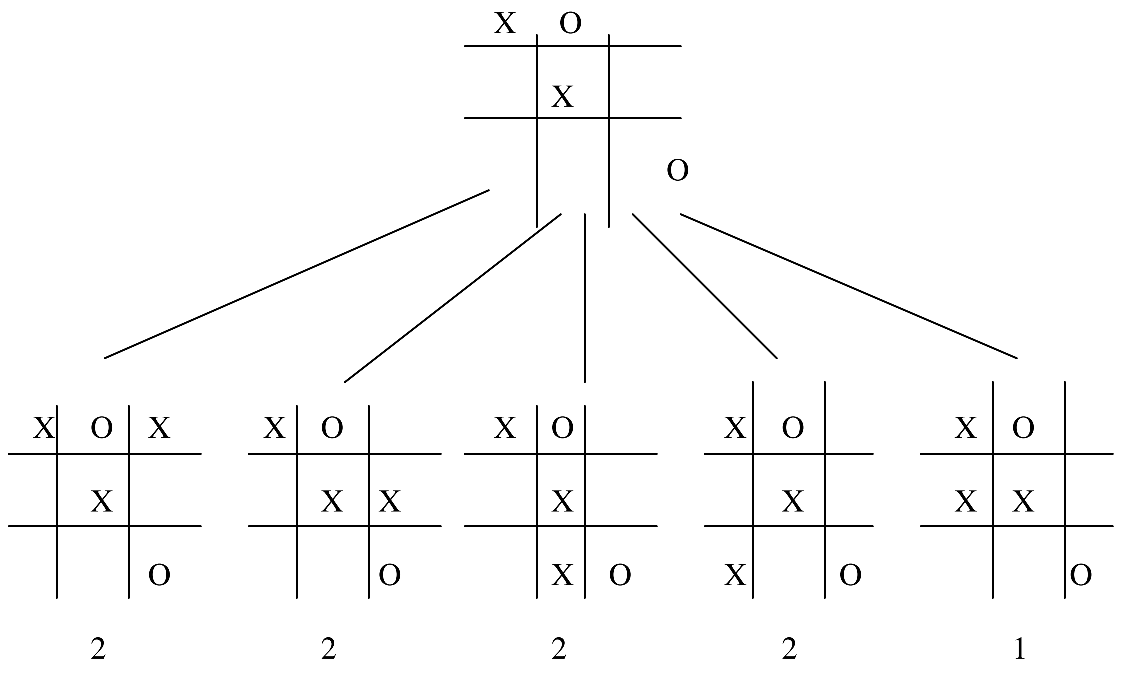
The lists are
Vertex 1 : N1→ N2 →N3
Vertex 2 : N1→N4 →N5
Vertex 3 : N2 →N4 →N6
Vertex 4 : N3 →N5 →N6

(iv) Game trees

An interesting application of trees is the playing of games such as tie-tac-toe, chess, nim, kalam, chess, go etc.

We can picture the sequence of possible moves by mean of a game tree in which the root denotes the initial situation and the branches from the root denote the legal moves that the first player could make. At next level down, the branches corresponds to the legal move by the second player in situation and so on ,with branches from vertices at even levels denoting moves by the 1st player and from vertices at odd levels denoting moves by he 2nd player.

Eg. Tic-tac-toe



Q111. Describe various memory allocation strategies. **(8)**

Ans.

Memory Allocation Strategies

If it is not desirable to move blocks of allocated storage from one area of memory to another, it must be possible to relocate memory blocks that have been freed dynamically. Each time a request is made for storage, a free area large enough to accommodate the size requested must be allocated. The most obvious methods for keeping track of the free blocks is to use linear linked list. Each free block contains a field containing the size of the blocks and a field containing a pointer to be next free block. A global P for free block points to the 1st free block on this list. There are several methods of selecting the free block to use at when requesting storage.

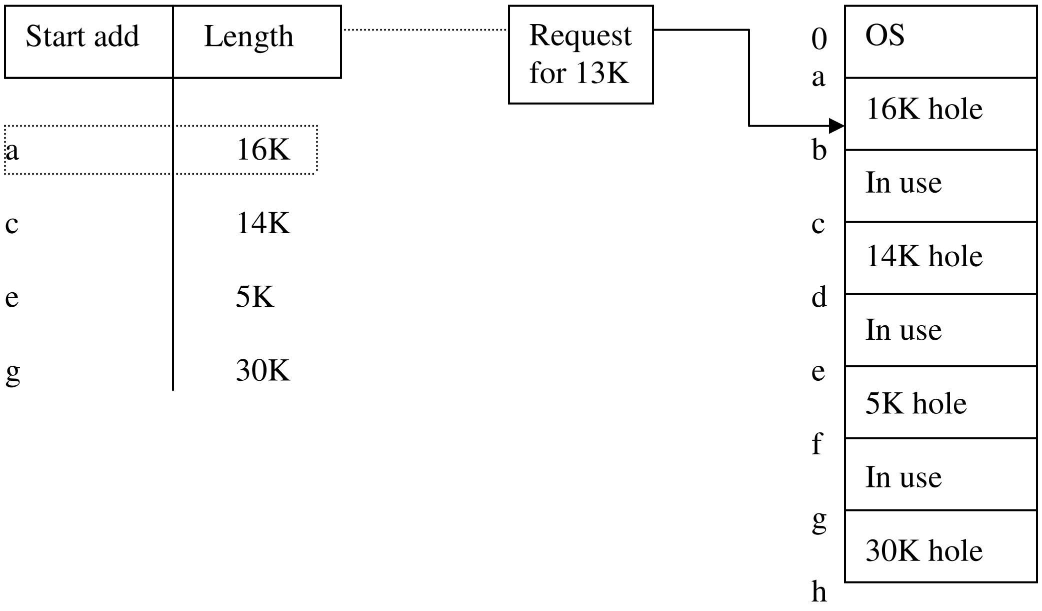
First – Fit Method:- The free list is traversed sequentially to find the 1st free block whose size is larger than or equal to the amount requested. Once the block is found it is removed from the list (if it is greater than the amount requested). The 1st of these portions remains on the list and the 2nd is allocated.

Best – Fit Method:- This method obtains the smallest free block whose size is greater than or equal to obtain such a block by traversing the entire free list follows.

Worst Fit method:- In this method the system always allocate a portion of the largest free block in memory. The philosophy behind this method is that by using small number of a very large block repeatedly to satisfy the majority of requested, many moderately sized blocks will be left un-fragmented.

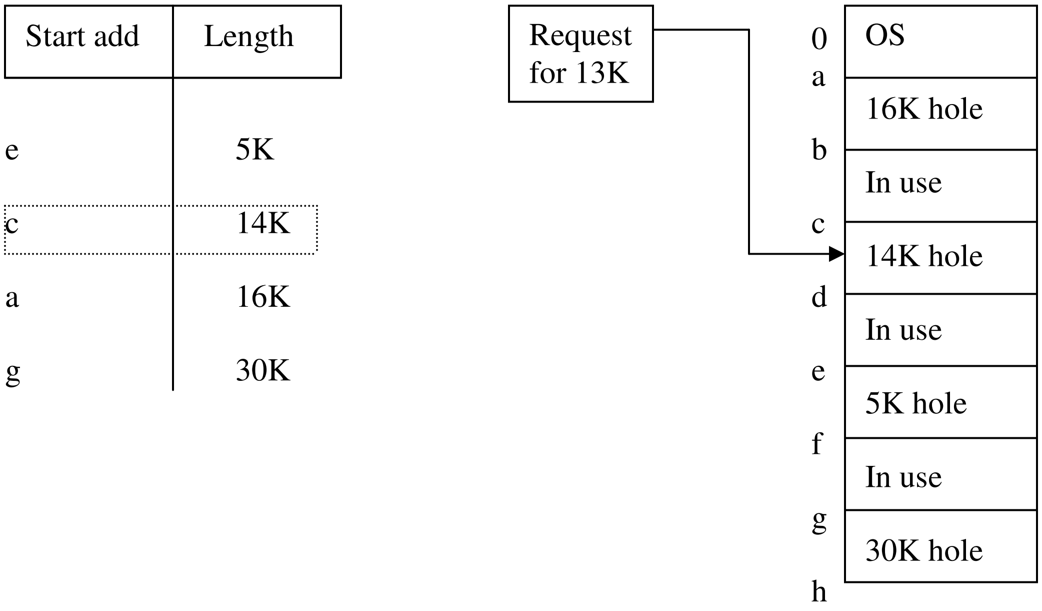
Free Storage List

First fit method



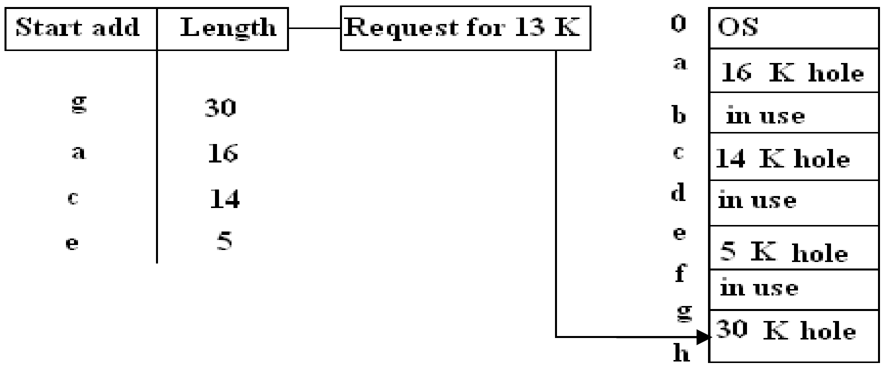
Start add	Length
a	16K
c	14K
e	5K
g	30K

Best Fit method



Start add	Length
e	5K
c	14K
a	16K
g	30K

Worst Fit Strategy



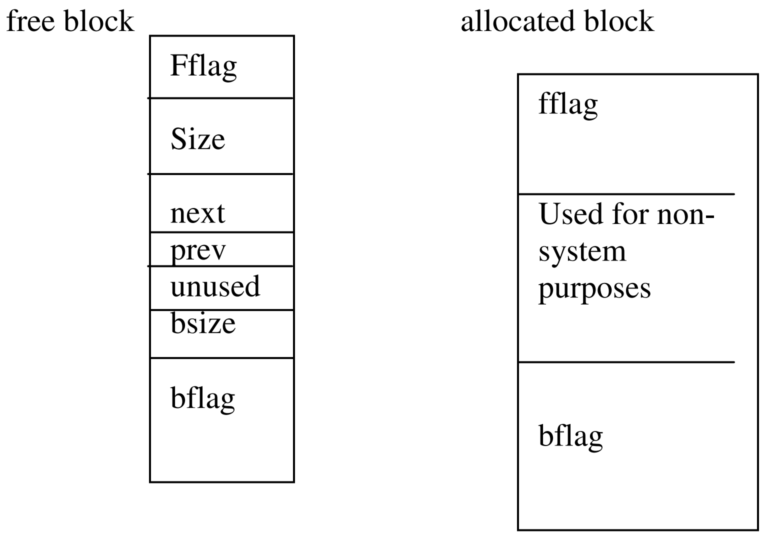
Start add	Length
g	30
a	16
c	14
e	5

Q112. How memory is freed using Boundary tag method in the context of Dynamic memory management? **(6)**

Ans.

Boundary Tag Method to free Memory

To remove an arbitrary block from the free list (to combine it with a newly freed block) without traversing entire list, the free list must be doubly linked. Thus each free block must contain 2 pointers, **next** and **prev** to the next and previous free block on the free block. (This is needed when combining a newly freed block with a free block that immediately precedes in memory). Thus the front of the free block must be accessible from its rear. One way to do this is to introduce a **bsize** field at a given offset from the last location of each free block. This field contains the same value as the size filed at the front of the block. This field contains the some value as the **size** field at the front of the block. The figure below illustrates the structure of free and allocated blocks under this method which is called the boundary tag method.



Each of the contract fields fflag, size, next, prev, bsize and bflag is shown as occupying a complete word, although in practice they may be packed together, several fields to a word. We assume that fflag or bflag are logically flags and that true indicates an allocated block and false indicates a free block. Using the information in the free block and the newly freed block, the system merges the two blocks into a bigger hole.

Q 113. Define a method for keeping two stacks within a single linear array S in such a way that neither stack overflows until entire array is used and an entire stack is never shifted to a different location within the array. Write routines for pushing and poping elements in two stacks. **(8)**

Ans.

```
/* Two Stacks Into One Array */
#include<stdio.h>
#define MAX 50
int mainarray[MAX];
int top1=-1,top2=MAX;
void push1(int elm)
 {
  if((top2-top1)==1)
    {
     clrscr();
     printf("\n Array has been Filled !!!");
     return;
    }
 mainarray[++top1]=elm;
       }
void push2(int elm)
 {
 if((top2-top1)==1)
    {
       clrscr();
       printf("\n Array has been Filled !!!");
       return;
    }
 mainarray[--top2]=elm;
 }
int pop1()
 {
    int temp;
    if(top1<0)
       {
          clrscr();
          printf(" \n This Stack Is Empty !!!");
          return -1;
       }
    temp=mainarray[top1];
    top1--;
    return temp;
 }
int pop2()
 {
    int temp;
    if(top2>MAX-1)
       {
          clrscr();
          printf(" \n This Stack Is Empty !!!");
          return -1;
       }
    temp=mainarray[top2];
    top2++;
    return temp;
 }
```

Q114. Suppose a queue is housed in an array in circular fashion. It is desired to add new items to the queue. Write down a procedure ENQ to achieve this also checking whether the queue is full. Write another procedure DQ to delete an element after checking queue empty status. **(6)**

Ans.
Add an element in Circular Queue
```
# define  MAXQUEUE 100
struct queue{
   int items[MAXQUEUE];
   int front, rear;
}
struct queue q;
q.front=q.rear=MAXQUEUE -1;
void ENQ(struct queue *pq, int x)
   {
      /* make room for new element*/
      if(pq ->rear = MAXQUEUE - 1)
         pq-> rear = 0;
      else
         (pq->rear)++;
      /* check for overflow */
      if(pq ->rear==pq->front)
         {
            printf(“queue overflow);
            exit(1);
         }
      pq->items[pq->rear]=x;
      return;
   }/* end of ENQ*/
```
Delete an element from Circular Queue
```
int DQ(struct queue *pq)
   {
      if(pq-> rear == pq-> front)
         {
            printf(“queue underflow”);
            exit(1);
         }/*end if*/
      if(pq->front = = MAXQUEUE-1)
         pq->front=0;
      else
         (pq->front)++;
      return(pq->items[pq->front]);
   }/*end DQ*/
```

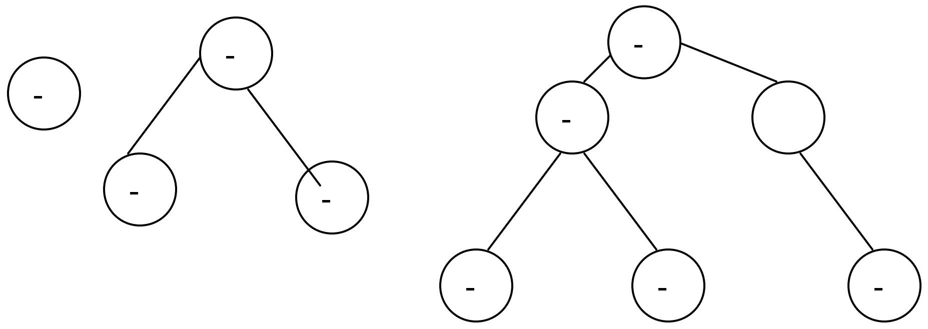
Q115. What is a height balanced tree? Explain how the height is balanced after addition/deletion of nodes in it? **(7)**

Ans.
Height Balanced Tree (AVL Tree)
An AVL tree is a binary search tree in which the height of the left and right subtree of the root differ by at most 1 and in which the left and right subtrees are again AVL trees. With each node of an AVL tree is associated with a balanced factor that is left high, equal or right high, according, respectively, as

the left subtrees has height greater than, equal to, or less than that of the right subtree.

The definition does not require that all leaves be on the same or adjacent levels.

for example:

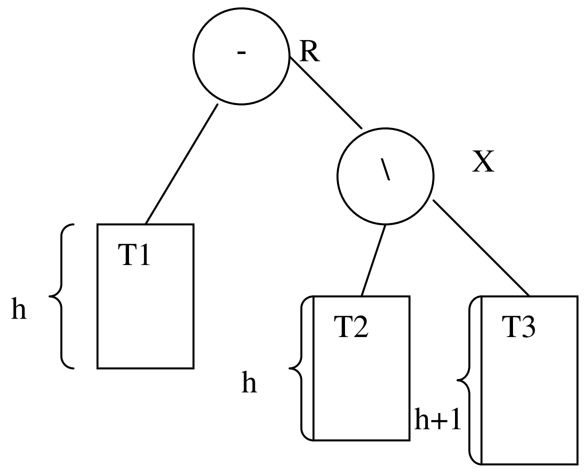


To achieve balance in an AVL tree after insertions and deletions, rotations are carried out.

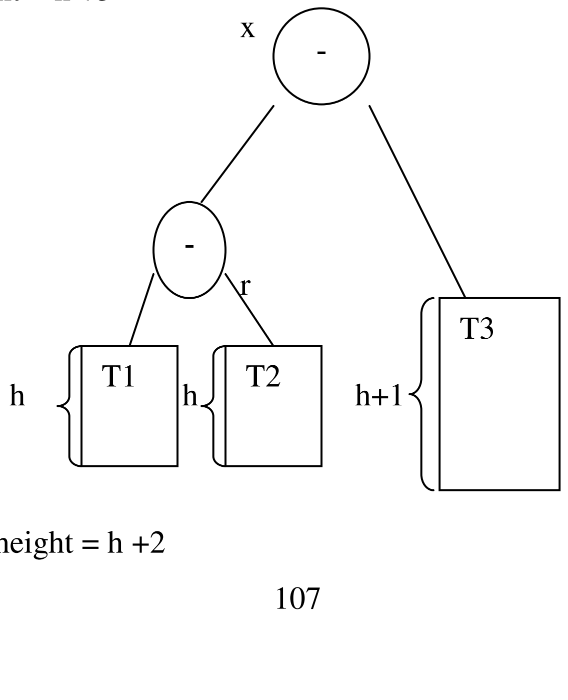
Rotation:-

Let r be the root of the tree and x be the root of its right subtree.

Case 1: right High



Total height = h +3

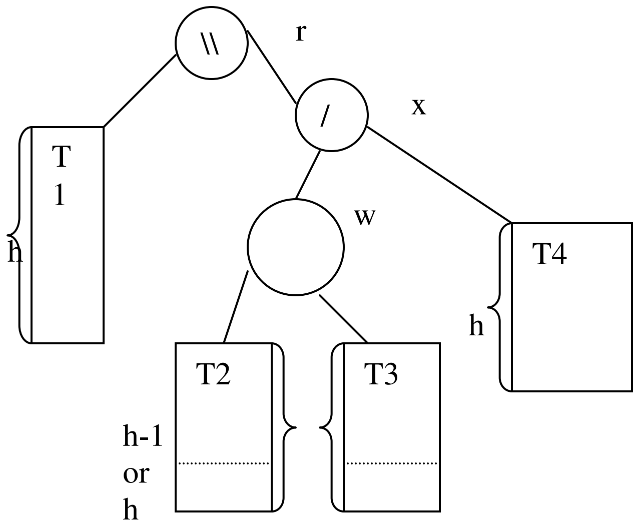


Total height = h +2

Item x is right high. The action needed in this case is called a **left rotation**; we have rotated the node x upward to the root, dropping r down into the left subtree of x; the subtree T2 of nodes with keys between those of r and x now becomes the right subtree of r rather than the left subtree of x.

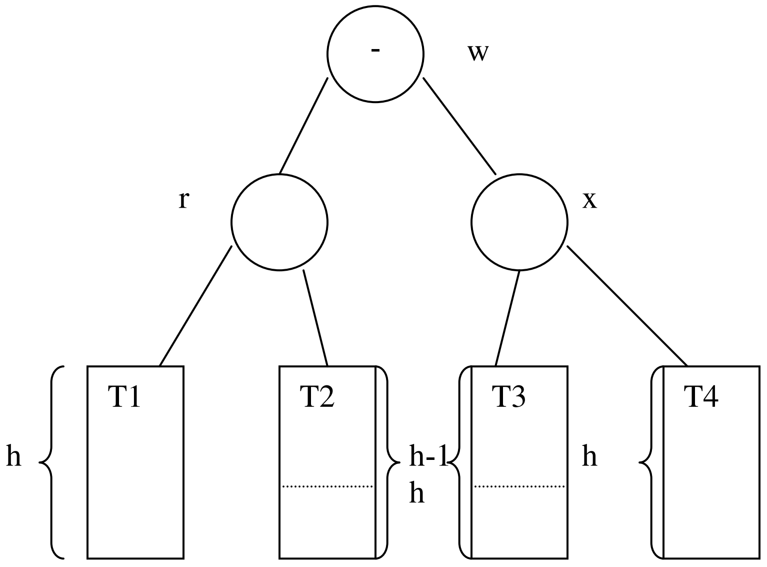
Case 2: left high: it is carried out similar to right high case

Case 3:double rotation



one of the T2 or 3 has height h

Total height = h + 3



Total height = h + 2

In this case it is necessary to move 2 levels to the node w that roots the left subtree of x , to find the new root. This process is known as **double rotation** because the transformation can be obtained in 2 steps by 1st rotating the subtree within root x to the right (so that w becomes the root) and then rotating the tree with root r to the left (moving w up do becomes the new root).

The new balance factors for r and x depends on the previous balance factor from w.

Q116. Show the linked representation of the following two polynomials.

$$7x^{80}+5x^{50}+3x^{30}+1=0$$

$$9x^{90}+6x^{60}+2x^{20}-1=0$$

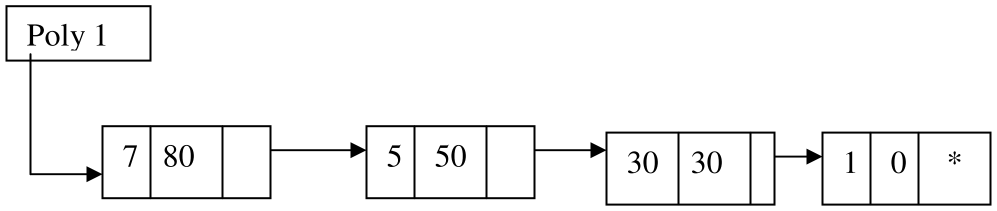
Build a procedure for adding two polynomials stored in linked lists. Verify steps of your procedure for the above two polynomials. **(7)**

Ans.

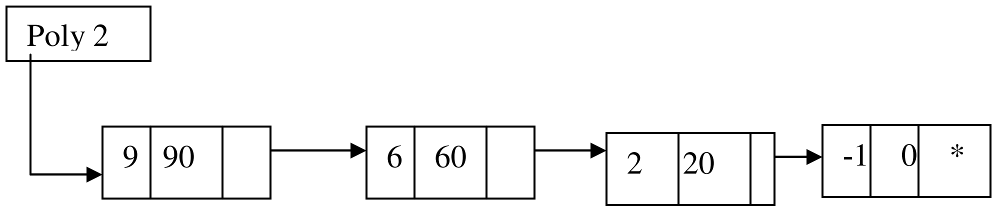
$7x^{80} + 5x^{50} + 30x^{30} + 1$ ----→1

$9x^{90} + 6x^{60} + 2x^{20} - 1$ -----→2

Poly 1 can be represented as



Poly 2 can be represented as

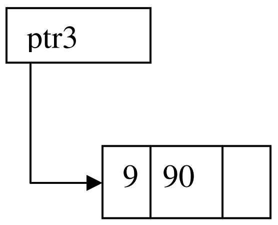


The node of the polynomial is of the form

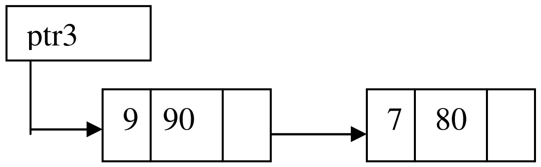
Coefficient of the term	Power of x	Link to next node

Procedure for Adding two Polynomials

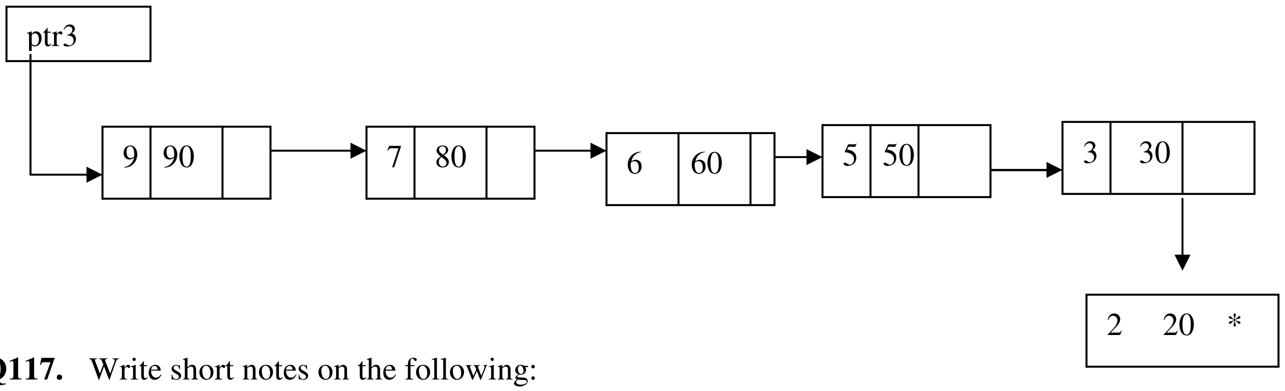
```
 typedef  struct node_type
{
   int coeff;
   int power;
   struct node_type *next;
}node;
node *poly;
void AddPolynomial(node *ptr1 , node  *ptr2, node **ptr3)
{
   int powc;
   int coef;
   while((ptr1 !=NULL) && (ptr2! =NULL))
   {
      if(ptr1->power > ptr2->power)
      {
         coef = ptr1 -> coeff;
         powe = ptr1 -> power;
         ptr1 = ptr1-> next;
      }
      else if(ptr1 -> power < ptr2-> power)
      {
         coef = ptr2 -> coeff;
         powe = ptr2 -> power;
         ptr2 = ptr2-> next;
      }
         else
         {
            coef = ptr1 -> coeff + ptr2 -> coeff;
            powe = ptr1 -> power;

            ptr1 = ptr1-> next;
            ptr2 = ptr2-> next;
         }
      if(coef !=0)
         add_node(ptr3, coef , powe);
   }
   if(ptr1 = = NULL)
   {
      for(; ptr2 ! = (node *)NULL; ptr2 = ptr2-> next)
         add_node(ptr3, ptr2->coef , ptr2->powe);
   }
   else if(ptr2 = = NULL)
   {
      for(; ptr1 ! = NULL; ptr1 = ptr1-> next)
         add_node(ptr3, ptr1->coef , ptr1->powe);
   }
   coef=9
   powe=90
   add(ptr3,9,90)
```



coef = 7
powe=80
add(ptr3,7,80)



Like this we keep continuing and finally we get



Q117. Write short notes on the following:

(i) Threaded binary trees.
(ii) Graph traversal.
(iii) Conversion of forest into tree.
(iv) Doubly linked list. **$(3.5 \times 4 = 14)$**

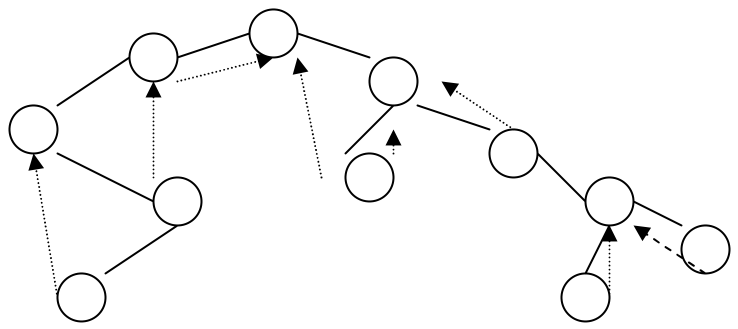
Ans.

(i) Threaded Binary Tree:-

By changing the NULL lines in a binary tree to special links called threads, it is possible to perform traversal, insertion and deletion without using either a stack or recursion.

In **a right in threaded binary tree** each NULL link is replaced by a special link to the successor of that node under inorder traversal called right threaded. Using right threads we shall find it easy to do an inorder traversal of the tree, since we need only follow either an ordinary link or a threaded to find the next node to visit.

If we replace each NULL left link by a special link to the predecessor of the node known as left threaded under inorder traversal the tree is known as **left in threaded binary tree**. If both the left and right threads are present in tree then it is known **as fully threaded binary tree** for example:

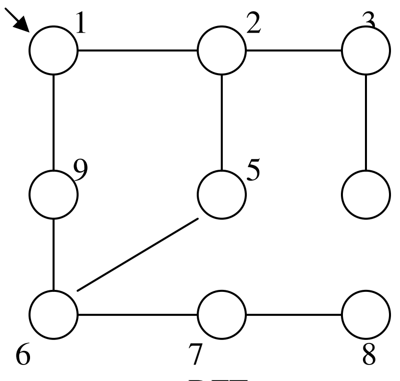


(ii) Graph Traversal

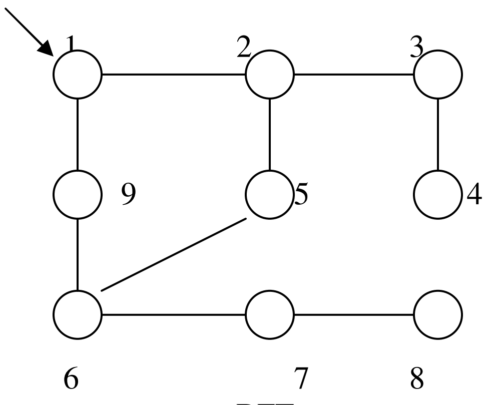
In many problems we wish to investigate all the vertices in a graph in some systematic order. In graph we often do not have any one vertex singled out as special and therefore the traversal may start at an arbitary vertex. The two famous methods for traversing are:-

a) **Depth first traversal**: of a graph is roughly analogous to pre-order traversal of an ordered tree. Suppose that the traversal has just visited a vertex or, and let W_0, W_1,……W_k be the vertices adjacent to V. Then we shall next visit W_0 and keep W_1…..W_k waiting. After visiting W_0 we traverse all the vertices to which it is adjacent before returning to traverse W_1, W_2, ……..W_k.

b) **Breadth first**: of a graph is roughly analogous to level by level traversal of ordered tree. If the traversal has just visited a vertex V, then it next visits all the vertices adjacent to V. Puting the vertices adjacent to these is a waiting list to be traversed after all the vertices adjacent to V have been visited.



DFT



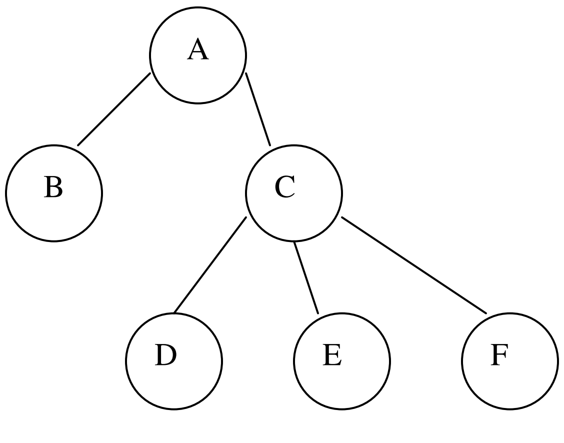
BFT

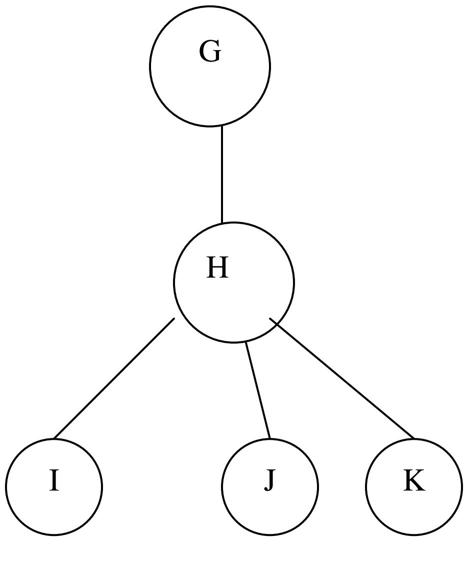
DFT= 1 2 3 4 5 6 7 8 9
BFT = 1 2 9 3 5 6 4 7 8

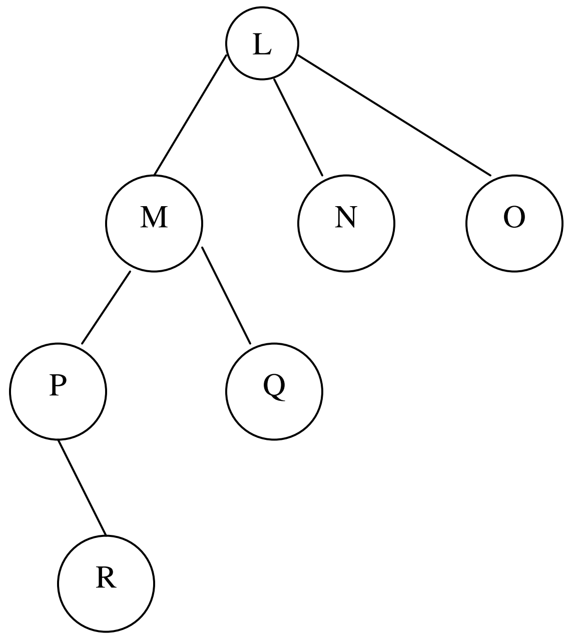
(iii) Conversion of Forest into Tree

A binary tree may be used to represent an entire forest, since the next pointer in the root of a tree can be used to point to the next tree of the forest. The following figure illustrates a forest and its corresponding binary tree.

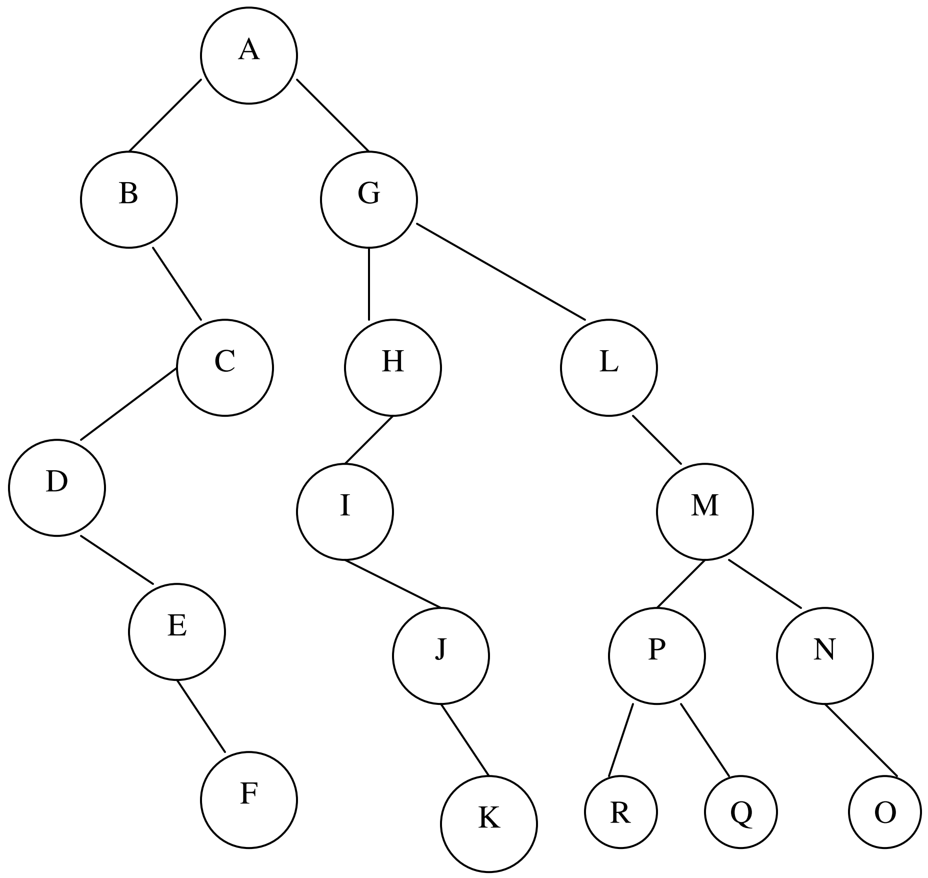
Initially the left tree is represented as a binary tree then the 2nd tree is made the right child of the root node of the first tree and the 3rd tree is made the right child of the root node of the 2nd tree.







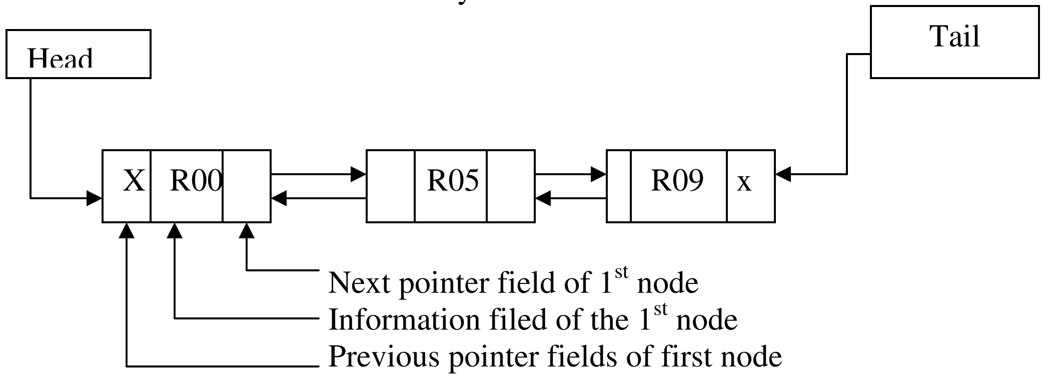
FOREST



BINARY TREE

(iv) Double Linked List

In a doubly linked list, also called as 2 way list, each node is divided into 3 parts. The first part is called previous pointer field. It contains the address of the preceding elements in the list. The second part contains the information of the element and the third part is called next pointer field. It contains the address of the succeeding elements in the list. In addition 2 pointer variable HEAD and TAIL are used that contains the address of the 1st element or the address of the last element of the list. Doubly linked list can be traversed in both directions.


Head
Tail
X
R00
R05
R09
x
Next pointer field of 1st node
Information filed of the 1st node
Previous pointer fields of first node

Q118. Differentiate between system defined data types and Abstract data types with suitable examples. **(8)**

Ans.

Abstract Data Type:- A useful tool for specifying the logical properties of data type is the abstract data type or ADT. A data type is a collection of values and a set of operation on those values. The term "ADT" refer to the basic mathematical concept that defined the data types. ADT is not concerned with implementation details at all. By specifying the mathematical and logical properties of data type or structures, the ADT is a useful guideline to implementation or a useful tool to programmers who wish to use the data type correctly.

An ADT consists of 2 parts: a value definition and an operation definition

The **value definition** defines the collection of values for the ADT and consists of 2 parts: a definition clause and a condition clause.

Operation definition: - Each operation is defined as an abstract form with 3 parts: a leader, the operational pre-condition and the post condition. Eg.

/* value definition */
```
abstract type def <integer, integer>RATIONAL;
condition RATIONAL [1]! = 0
/* operator definition */
abstract RATIONAL Makerational (a, b)
int a, b;
precondition b! = 0;
postcondition makerational[0] = = a;
      maherational[1] = =b;
```

System defined data types:-

These are data types that have been defined by the compiler of any program. The C language contains 4 basic data types:- int, float, char and double. There are 4 qualifier that can be applied to ints:- short, long, unsigned.

A variable declaration in C specifies 2 things:-

First it specifies the amount of storage that must be set aside for object declared with that type.

Second it specifies how data represented by string of bits are to be interpreted. The some bits at a specific location can be interpreted as an int of a floating-point number, yielding 2 completely different numeric values.

In ADT's we are not concerned with space or time efficiency.

It is not possible to implement a particular ADT on a particular piece of hardware or using a particular software system.

Q119. Explain the following:

(i) Complexity of an Algorithm.

(ii) The space-time trade off algorithm. **(6)**

Ans.

(i) Complexity of an Algorithm

An algorithm is a sequence of steps to solve a problem; there may be more than one algorithm to solve a problem. The choice of a particular algorithm depends upon following consideration:-

1) Time Complexity
2) Space Complexity

Time Complexity:- The time complexity of an algorithm is the amount of time it needs to run to completion. Some of the reasons for studying time complexity are:-

➢ We may be interested to know in advance whether the program will provide a satisfactory real time response.

➢ There may be several possible solutions with different time requirement.

Space Complexity:- The space complexity of an algorithm is the amount of memory it needs to run to completion. Some of the reasons to study space complexity are: -

➢ There may be several possible solutions with in different space requirement.

➢ To estimate the size of the largest problem that a program can solve.

(ii) The Space – Time Trade Off

The best algorithm to solve a given problem is one that requires less space in memory and takes less time to complete its execution. But in practice it is not always possible to achieve both of these objectives. There may be more than one approach to solve a problem. One approach may require more space but less time to complete its execution. The 2^{nd} approach may require less space but takes more time to complete execution. We choose 1^{st} approach if time is a constraint and 2^{nd} approach if space is a constraint. Thus we may have to sacrifice one at cost of the other. That is what we can say that there exists a time space trade among algorithm.

Q120. Define a sparse matrix. Explain different types of sparse matrices? Show how a triangular array is stored in memory. Evaluate the method to calculate address of any element a_{jk} of a matrix stored in memory. **(10)**

Ans.

Sparse Matrix

A m x n matrix A is said to be sparse if MOST of its elements are zero. A matrix that is not sparse is called dense matrix.

Types of Sparse matrix

1) Diagonal Matrix

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

This is the square matrix where the non zero elements are only where *row = col* ie at *diagonal.*

2) Tridiagonal Matrix

$$\begin{pmatrix} X & x & & & & \\ X & x & x & & & \\ & X & x & x & & \\ & & X & x & x & \\ & & & - & - & - \\ & & & & x & x \end{pmatrix}$$

In this square matrix all elements other than those on and on the diagonals immediately above and below this one are zero.

Triangular Matrices

Tiangular Matrices is of 2 types:

a) Lower triangular
b) Upper triangular

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 3 & 5 & 0 & 0 \\ 8 & 6 & 7 & 0 \\ 4 & 1 & 4 & 9 \end{pmatrix} \qquad \begin{pmatrix} 2 & 1 & 8 & 6 \\ 0 & 5 & 3 & 4 \\ 0 & 0 & 7 & 2 \\ 0 & 0 & 0 & 9 \end{pmatrix}$$

Lower Upper

In an n*n lower triangular matrix A, row 1 has one non zero element, row 2 has 2, ….., and row n has n. whereas, in an n*n upper triangular matrix A, row 1 has n non zero elements, row 2 has n-1 ,…. , and row n has 1. In both the cases, the total number of non-zero elements is n(n+1)/2.

Both of these matrices can be represented using an one dimensional array *la* of size n(n+1)/2.

Consider lower triangular matrix L. the elements can be mapped by rows or by columns.

In a row-wise mapping, the element L[i,j], i>=j, is preceded by ∑k for k=1 to i-1, elements that are in row 1 through i-1, and j-1 such elements from row i. the total number of elements that precede it in a row-wise mapping is $\frac{i(i-1)}{2}+j-1$. This expression also gives the position l[i,j] in *la.*

Method to calculate address of any element a_{jk} of a matrix stored in memory.
Let us consider 2 dimensional array a of size m*n further consider that the lower bound for the row index is *lbr* and for column index is *lbc.*
Like linear array, system keeps track of the first element only i.e. , the base address of the array.
Using this base address, the computer computes the address of the element in the ith row and jth column i.e. loc(a[i][j]), using the following formulae:
Column major order:-
Loc (a[i][j]) = base (a) + w [m (j – lbc) + (i – lbr)] in general
Row major order:-
Loc (a[i][j]) = base (a) + w [n(i – lbr) + (j – lbc)] in general
Where w is number of bytes per storage location for any one element of the array.

Q121. A linear array A is given with lower bound as 1. If address of A[25] is 375 and A[30] is 390, then find address of A[16]. **(4)**

Ans.
Loc (a[k]) = base (a) + w (k-lb)
375 = base (a) + w(25 – 1)
390 = base (a) + w(30 – 1)
375 = x + 24w
390 = x + 29w

---

15 = 5w
$\therefore$ w = 3
$\therefore$ x = 375 – 24 *3
x = 375 – 72
x = 303
$\therefore$ base address is 303 and w = 3.
$\therefore$ Address of A[16] is
loc = 303+ 3(16-1) = 348.

Q122. Let a binary tree 'T' be in memory. Write a procedure to delete all terminal nodes of the tree. **(8)**

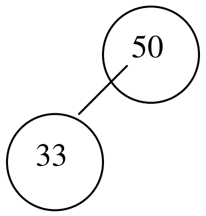
Ans.
function to Delete Terminal Nodes from Binary Tree
```
void deleteleaves(struct node* root,struct node* prev)
 {
if(root)
{
if(root->left==NULL&&root->right==NULL)
   {
    if(prev->item>=root->item)
       prev->left=NULL;
    else
       prev->right=NULL;
```

```
          printf("\n\n  %d Is Being Deleted ...",root->item);
          free(root);
          return;
         }
      deleteleaves(root->left,root);
      deleteleaves(root->right,root);
     }
}
```

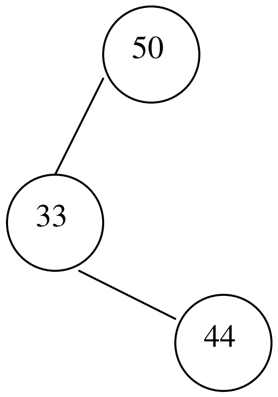
the call to this function will deleteleaves (root, root).

Q123. Consider the following eight numbers 50, 33, 44, 22, 77, 35, 60 and 40. Display the construction of the binary by inserting the above numbers in the given order. **(6)**

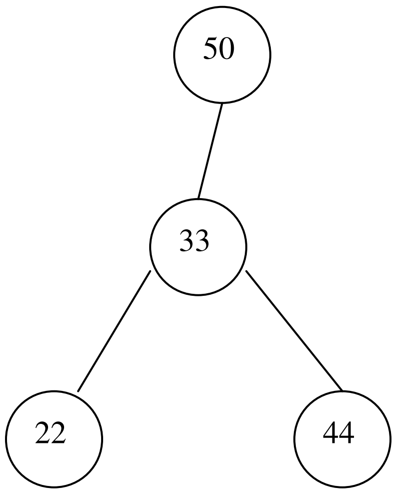
Ans. 50 33 44 22 77 35 60 40



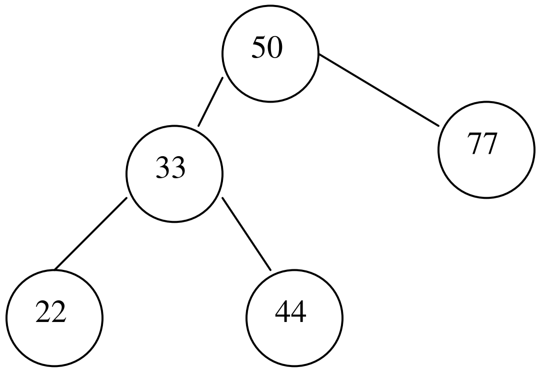
44



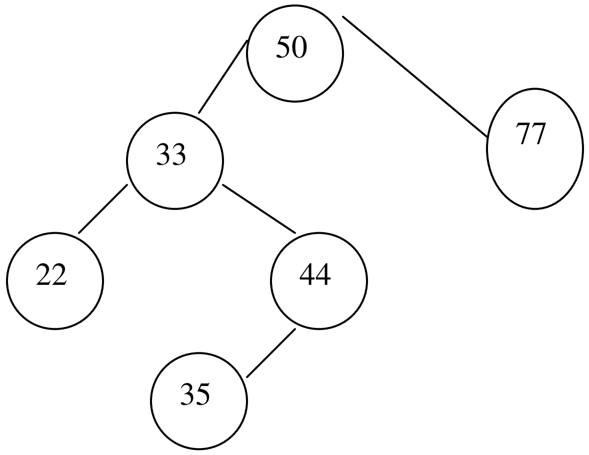
22

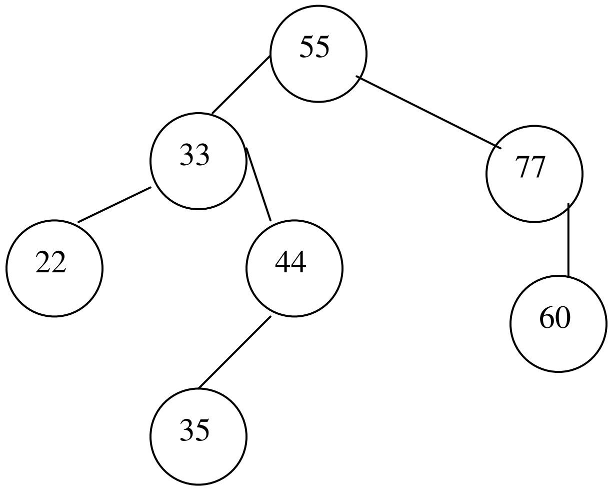


72

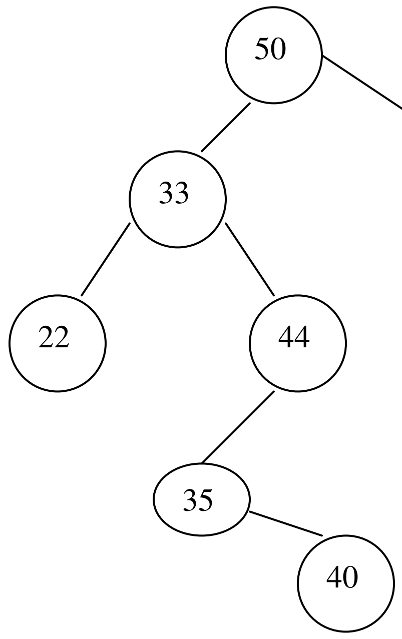


35



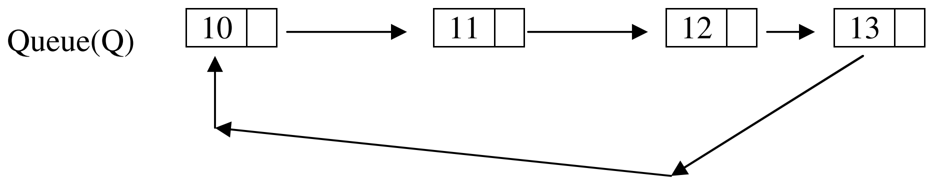


40



Q124. Can a Queue be represented by circular linked list with only one pointer pointing to the tail of the queue? Substantiate your answer using an example. **(5)**

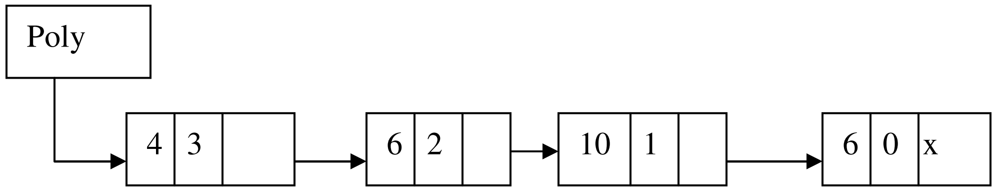
Ans.
Yes a Queue can be represented by a circular linked list with only one pointer pointing to the tail of the queue.
for example:



Q has a pointer pointing to the tail of 'Q'.
This pointer represents real of the Q and ptr->next is the front of Q.

Q125. Establish the usage of linked lists for polynomial manipulation. **(5)**

Ans.
Usage of Linked List for Polynomial Manipulation.
linked lists are frequently used for maintaining polynomial in memory. for example:
A polynomial of type $4x^3 + 6x^2 + 10x + 6$
can be represented using following linked list.



Where a node of the form

Coefficient of the term	Power of x	Link to next node

A linked list can be used for adding the two polynomial

Q126. Convert the following infix expressions to postfix notation

(i) A+((B+C)*(D+E)+F/G)

(ii) A ↑ B ↑ C ∗ D **(4)**

Ans.

	In Fix	Post Fix
(i)	A + (((B + C) * (D +E)) + (F/G))	ABC +DE+*FG/++
(i)	(A ↑ B ↑ C)) * D	ABC↑↑D*

Q127. Explain Hash Tables, Hash function and Hashing Techniques? **(8)**

Ans.

Hash Table:

A hash table is a data structure in which the location of a data item is determined directly as a function of data item itself rather than by a sequence of comparison. Under ideal condition, the time required to locate a data item in a hash table is 0(1) ie. It is constant and DOES not depend on the number of data items stored. When the set of K of keys stored is much smaller then the universe U of all possible keys, a hash table require much less storage spare than a direct address table.

Hash Function

A hash function h is simply a mathematical formula that manipulates the key in some form to compute the index for this key in the hash table. Eg a hash function can divide the key by some number, usually the size of the hash table and return remainder as the index for the key. In general, we say that a hash function h maps the universe U of keys into the slots of a hash table T[0….n-1]. This process of mapping keys to appropriate slots in a hash table is known as **hashing.**

The main reconsideration for choosing hash function is :-

1) It should be possible to compute it efficiently.

2) It should distribute the keys uniformly across the hash table i.e. it should keep the number. of collisions as minimum as possible.

Hashing Techniques:

There is variety of hashing techniques. Some of them are:-

a) Division Method:- In this method, key K to be mapped into one of the m states in the hash table is divided by m and the remainder of this division taken as index into the hash table. That is the hash function is

h(k) = k mod m

where mod is the modules operation.

b) **Multiplication Method**: The multiplication method operates in 2 steps. In the 1st step the key value K is multiplied by a constant A in the range O<A<1 and the fractional part of the value obtained above is multiplied by m and the floor of the result is taken as the hash values. That is the hash function is

h(k) = [m (K A mod 1)]

where "K A mod 1" means the fractional part KA of KA-[KA].A

A= (5-1/2)=0.6180334887

(c) **Midsquare Method**:- this operates in 2 steps. In the first step the square of the key value K is taken. In the 2nd step, the hash value is obtained by deleting digits from ends of the squared values ie. K^2. The hash function is

h(k) = s

where s is obtained by deleting digits from both sides of K^2.

Q128. Define a linked-list? How are these stored in the memory? Suppose the linked list in the memory consisting of numerical values. Write a procedure for each of the following:

(i) To find the maximum MAX of the values in the list.
(ii) To find the average MEAN of the values in the list.
(iii) To find the product PROD of the values in the list. **(14)**

Ans.

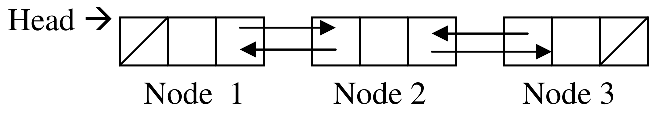
Linked List :-

A linked list is a linear collection of data elements called nodes. The linear order is given by pointer. Each node is divided into 2 or more parts. A linked list can be of the following types:-

- Linear linked list or one way list
- Doubly linked list or two way list.
- Circular linked list
- Header linked list

Representation of Linked list in Memory:-

Every node has an info part and a pointer to the next node also called as link. The number of pointers is two in case of doubly linked list. for example :



An external pointer points to the beginning of the list and last node in list has NULL. The space is allocated for nodes in the list using malloc or calloc functions. The nodes of the list are scattered in the memory with links to give linear order to the list.

```
(i)
float MAX_OF(node * start)
  {
  n=start;
  max=n->k;
  while(n!= NULL)
  {
  if(max<=n->k)
  max=n->k;
   }
  return max;
 (ii)   float MEAN_OF(struct node * start)
  {
  int h=0; struct node *n;
  n=start;
  while (n!= NULL)
  {
  s+=n->k;h++;
  n=n->next;
  }
  return s/h;
  }
(iii)   float PROD_OF(node *start)
  {
float p; struct node *n;
  n=start;
  while(n!= NULL)
  {
  p*=n->k;
  n=n->next;
  }
return p;
  }
```

Q129. Using array to implement the queue structure, write an algorithm/program to
(i) Insert an element in the queue.
(ii) Delete an element from the queue. **(9)**

Ans.

(i) **Algorithm to Insert an element in the queue.**

```
QINSERT( QUEUE, N, FRONT, REAR, ITEM)
This procedure inserts an elements ITEM into a queue.
1. [QUEUE already filled ?]
if FRONT =1 and REAR =N or if FRONT = REAR +1 then
write:OVERFLOW, and Return.
2. [Find new value of REAR]
If FRONT : = NULL, then : [QUEUE initially empty]
```

```
Set FRONT : = 1 and REAR : = 1,
Else if REAR = N then;
Set REAR : = 1
Else:
 Set REAR : = REAR + 1
[End of if structure]
3.Set QUEUE [REAR]: = ITEM [This inserts new elements]
4. Return
```

(ii) Algorithm to delete an element from the Queue

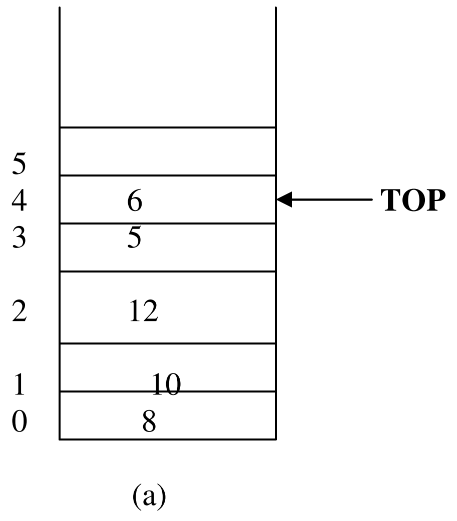
```
QDELETE (QUEUE, N, FRONT, REAR, ITEM)
This procedure deletes an element from a Queue and
assign it to the variable ITEM.
1.  [Queue already Empty?]
If  FRONT:= NULL, then write UNDERFLOW and Return
2.  Set ITEM:=QUEUE[FRONT]
3.  [Find new value of FRONT]
If FRONT=REAR, THEN[Queue has only one element to
start.]
 Set FRONT:=NULL and REAR:=NULL
Else  if    FRONT=N then:
 Set FRONT:=1
          Else:
  Set FRONT:=FRONT+1
[End of If structure]
4.  Return.
```

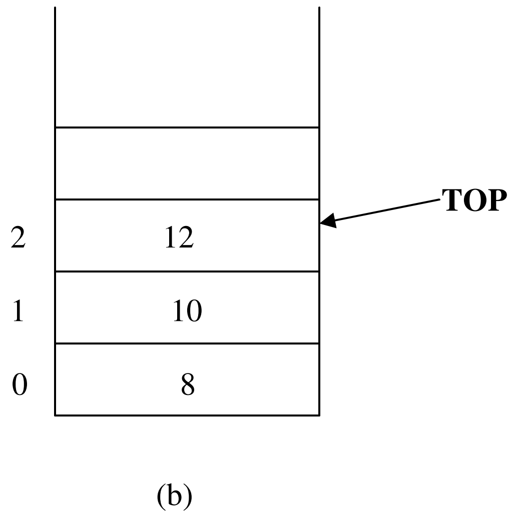
Q130. Define a stack. Describe ways to implement stack. **(5)**

Ans.

STACK:

A stack is one of the most commonly used data structure. A stack is also called a Last-In-First-Out (LIFO) system, is a linear list in which insertion or deletion can take place only at one end called the top. This structure operates in the same way as the stack of trays If we want to place another tray, it can be placed only at the top. Similarly, if we want to remove a tray from stack of trays, it can only be removed from the top. The insertion and deletion operations in stack terminology are known as PUSH and POP operation.





(a) Stack after pushing elements 8,10,12,5,6
(b) Stack after popping 2 elements.

Implementation of Stack

Stacks can be implemented in the following 2 ways:

a) Arrays
b) Linked List

a) **Arrays**:- To implement a stack we need a variable called top, that holds the index of the top element of stack and an array to hold the element of the stack.

For example:-
```
#define MAX 100
Typedef struct
{ int top;
int elements [MAX]
} stack;
```

b) **Linked List**:- A stack represented using a linked list is known as linked stack.
The array based representation of stack suffers from following limitations.

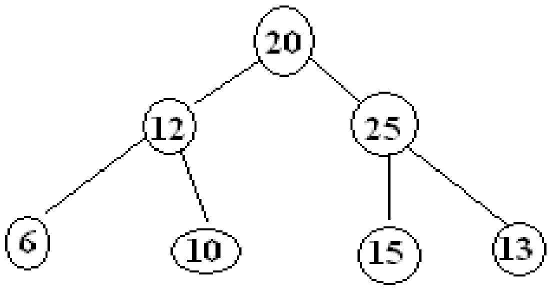
- ➢ Size of stack must be known in advance
- ➢ Overflow condition might occur.

The linked representation allows a stack to grow to a limit of the computer's memory.

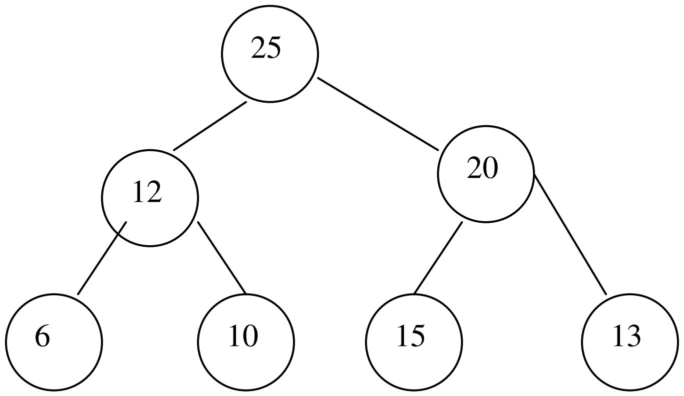
for example:-
```
typedef struct node
{
    int info;
    struct node * next;
} stack;
stack * top;
```

Q131. Sort the following list using Heap Sort technique, displaying each step.
20, 12, 25 6, 10, 15, 13 **(7)**

Ans.
HEAP SORT
Let the following numbers be stored in array.
20, 12, 25, 6, 10, 15, 13



First we create the heap from the above binary tree:-

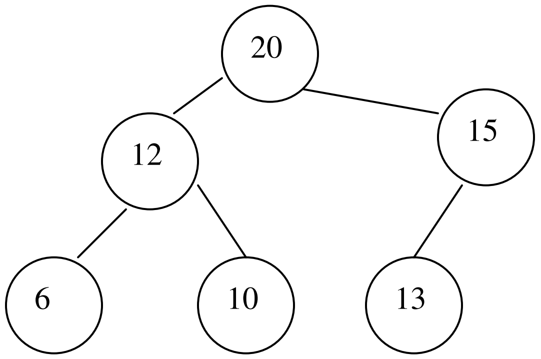


Now the above tree is a heap. Therefore, we store the above tree in an implicit representation of tree in an array.

25	12	20	6	10	15	13

Therefore 25 is target we place 25 in last of array.

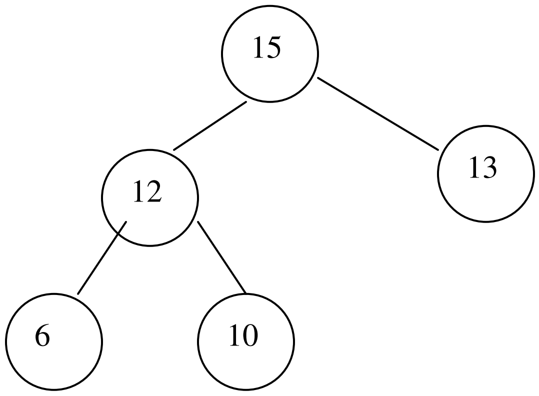
20	12	15	6	10	13	25



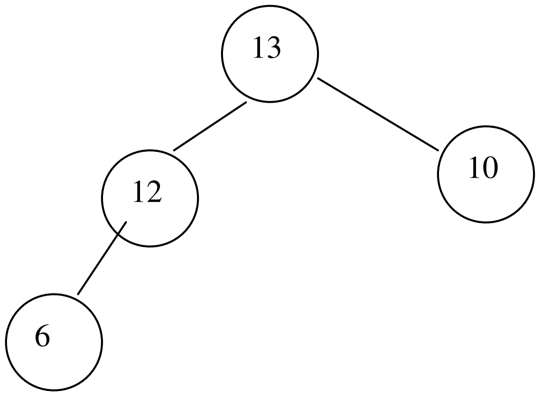
Now put 20 in its proper position by replacing 13

15	12	13	6	10	20	25

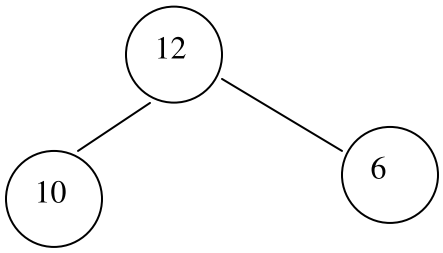
After reheapifying



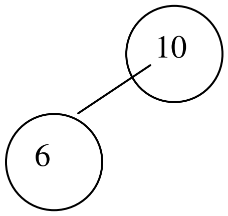
13	12	6	10	15	20	25



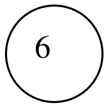
12	10	6	13	15	20	25



10	6	12	13	15	20	25



6	10	12	13	15	20	25



Q132. Give the binary search algorithm. **(7)**

Ans.

Binary Search Algorithm

```
1. if (low > high)
2.      return (-1)
3. mid = (low +high)/2;
4. if ( X = = a [mid])
5.       return (mid);
6. if ( X < a [mid])
7.     search for X in a (low) to [mid -1];
8.     else
9.      search for X in a [mid + 1] to a [high];
```

Q133. What do you understand by structured programming? Explain. **(5)**

Ans.

Structured Programming

This term is used for programming design that emphasizes:-

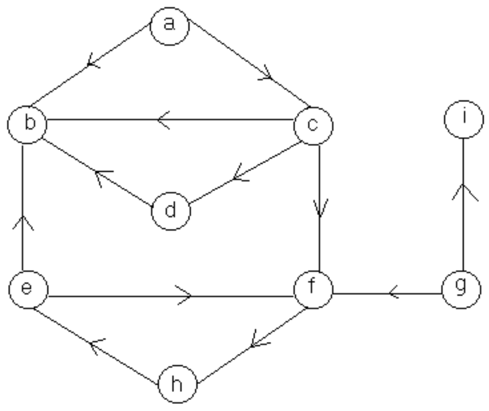
(1) Hierarchical design of programming structure using only the simple control forms of comparison, alteration and iteration.

(2) Representation of the hierarchical design directly in the program text, using the structured control statement.

(3) Program text in which the textured sequence if statements corresponds to the execution sequence.

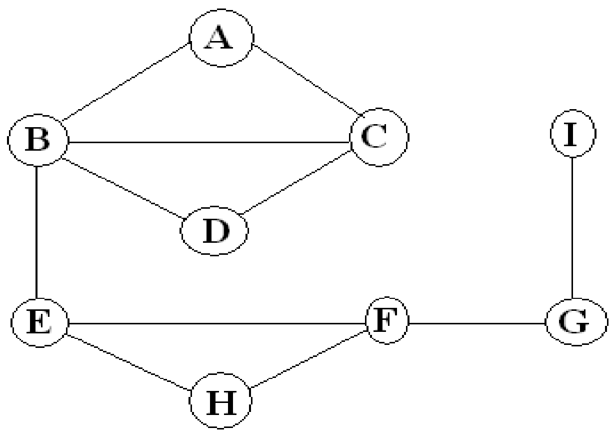
(4) Use of single purpose groups of stack even if statement must be copied when a program is written by following these tenets of structural programming, it is much easier to understand, debug, verify to be correct and take modify and re-verify. Eg. C, Java, C ++

Q.134. Give the adjacency matrix and adjacency list of the following graphs.



(6)

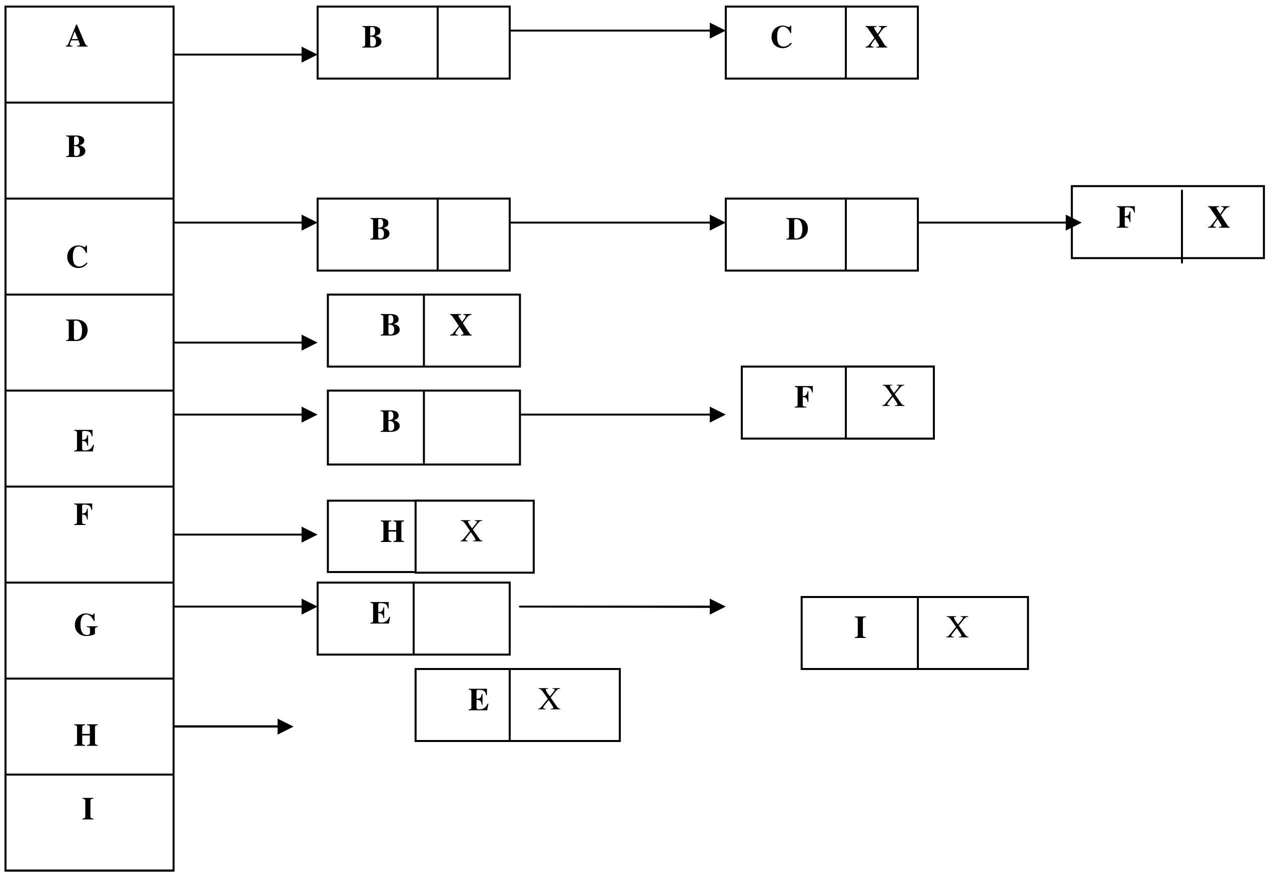
Ans.



	A	B	C	D	E	F	G	H	I
A	0	1	1	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0
C	0	1	0	1	0	1	0	0	0
D	0	1	0	0	0	0	0	0	0
E	0	1	0	0	0	1	0	0	0
F	0	0	0	0	0	0	0	1	0
G	0	0	0	0	0	1	0	0	1
H	0	0	0	0	1	0	0	0	0
I	0	0	0	0	0	0	0	0	0

Adjacency Matrix

Adjacency List



Q135. Consider the algebraic expression

$E = (5x+z)\ (3a-b)^2$

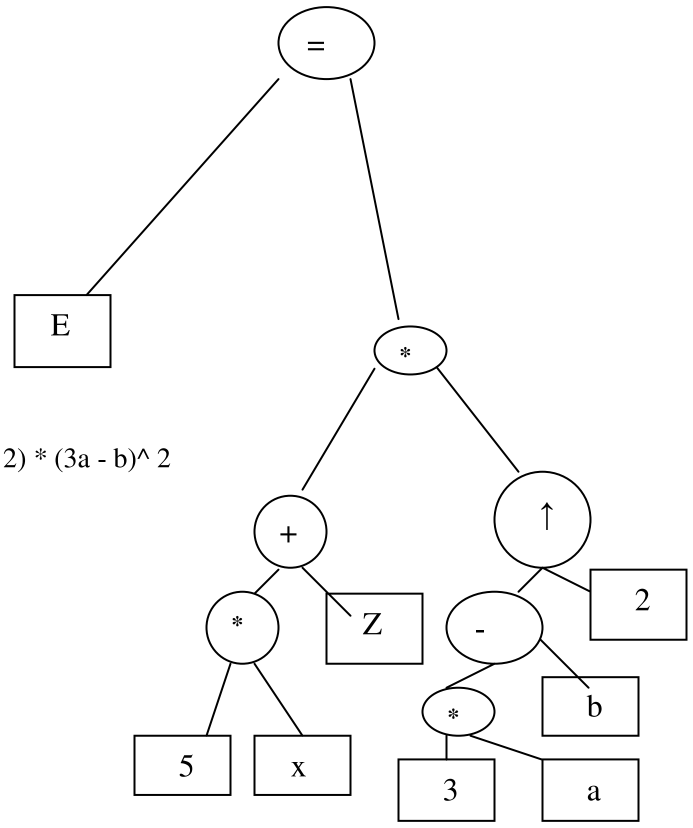
(i) Draw the expression tree corresponding to E

(ii) Find the scope of exponential operator i.e. the subtree rooted at the exponential operator. (7)

Ans.

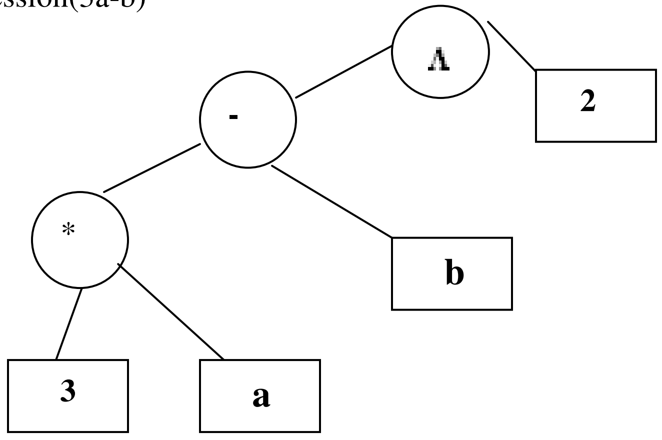
$E = (5x + z) * (3a - b)^2$

$= (5x + z) * (3a - b) * (3a - b)$



(i) (5x - 2) * (3a - b)^ 2

ii) the scope of exponential(^) is the tree shown below in the figure. It corresponds to the subexpression(3a-b)^2



Q136. Write an algorithm to evaluate an expression given in postfix notation. Show the execution of your algorithm for the following expression.
AB^CD-EF/GH+/+* **(7)**

Ans.
Algorithm to evaluate Post fix Expression

```
Opndstk = the empty stack;
/*scan the input string reading one */
/*element at a time into symb */
while ( not end of input){
        symb  = next input character;
        if (symb is an operand)
        push (opndstk, symb);
        else
        {
        /* symb is an operator */
        opnd 2 = Pop (opnd stk);
        opnd 1 = Pop (opnd stk);
        value = result of applying symb to opnd 1 and opnd 2;
        push (opndstk,value);
        }/*end else */
       }/*end while */
       return (pop (opnd stk));
```

AB^CD-EF/GH+/+*

Symb	Opnd1	Opnd2	Value	Opndstk
A				A
B				A,B
^	A	B	A^B	A^B
C	A	B	A^B	A^B,C
D	A	B	A^B	A^B,C,D
-	C	D	C-D	A^B,C-D
E	C	D	C-D	A^B,C-D,E
F	C	D	C-D	A^B,C-D,E,F
/	E	F	E/F	A^B,C-D,E/F
G	E	F	E/F	A^B,C-D,E/F,G
H	E	F	E/F	A^B,C-D,E/F,G,H
+	G	H	G+H	A^B,C-D,E/F,G+H
/	E/F	G+H	(E/F)/(G+H)	A^B,C-D, (E/F) /(G+H)
+	C-D	(E/F)/(G+H)	(C-D)+(E/F)/(G+H)	A^B,(C-D)+ (E/F)/(G+H)
*	A^B	(C-D)+(E/F)/(G+H)	A^B*((C-D)+ (E/F)/(G+H))	A^B*((C-D)+ (E/F)/(G+H))

Q137. Define an array. How does an array differ from an ordinary variable? How are arrays represented in the memory? **(5)**

Ans.

Array Vs. Ordinary Variable

Array is made up of similar data structure that exists in any language. Array is set of similar data types. Array is the collection of similar elements. These similar elements could be all int or all float or all char etc. Array of char is known as string. All elements of the given array must be of same type. Array is finite ordered set of homogeneous elements. The number of elements in the array is pre-specified.

For example. *Ordinary variable*: - int a

Array: - int a[10]

An ordinary variable ofa simple data type can store a single element only

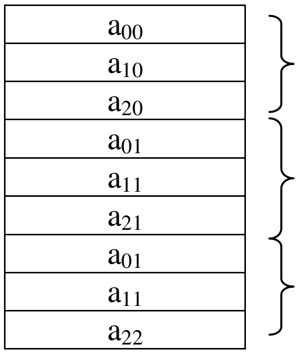
Representation of Array in memory

Let a be a 2 dimensional m x n array

	Col 0	Col 1	Col 2
Row 0	a_{00}	a_{01}	a_{02}
Row 1	a_{10}	a_{11}	a_{12}
Row 2	a_{20}	a_{21}	a_{22}

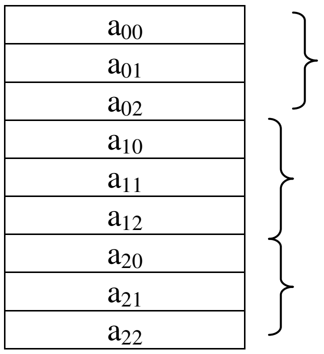
Though a is pictured as a rectangular pattern with in row and n column it is represented in memory by a row of m x n elements. Sequential memory locations the element can be stored in row major order.

Column major order:- the elements are stored column by column ie m elements of the 1st column is stored in first m locations, elements of the 2nd column are stored in next m location and so on..



Row major order:-

The elements are stored row by row ie. N elements of the 1st row are stored in 1st n location, elements of 2nd row are stored in next n location, and so on



Q138. Consider an array A[20, 10]. Assume 4 words per memory cell and the base address of array A is 100. Find the address of A[11, 5] assuming row major storage. **(5)**

Ans.

A[20, 10]
W = 4 words
Base (A) = 100
A [11, 5] $\rightarrow$ in row major m x n
loc [a [i] [j] = base (a) + w [n [i – lbr] + (j – lbc)]
= 100 + 4 [10 (11) + 5]
= 100 + 4[110+5]
=100 + 115 *4
= 560

Q139. Write an algorithm to convert an infix expression into postfix expression. **(8)**

Ans.

Algorithm to convert infix expression to post fix expression.

```
1. opstk = the empty stack;
2. while (not end of input) {
3. symb = next input character;
4. if (symb is an operand) add symb to postfix string
5. else {
6. while (!empty (opstk) and prcd (top (opstk),
       symb)>0){
7. topsymb = pop(opstk);
8. add topsymb to the postfix string; }/*end while*/
9. if (empty (opstk) || symb! = ‘)’ ) push (opstk,
       symb); else /*pop the open parenthesis and
       discard it */
         topsymb = pop(opstk);
       } /* end else */
     }/* end while */
     /* output any remaining operator */
10.while (!empty (opstk)){
11. top symb = pop (opstk);
12. add topsymb to the postfix string;
    } /* end while * /
/*output any remaining operator*/
```

Q140. Suggest a way of implementing two stacks in one array such that as long as space is there in an array, you should be able to add an element in either stack. Using proposed method, write algorithms for push and pop operations for both the stacks. **(6)**

Ans.

```
    /* Two Stacks Into One Array */
#include<stdio.h>
#define MAX 50

int mainarray[MAX];
int top1=-1,top2=MAX;
void push1(int elm)
 {
  if((top2-top1)==1)
    {
     clrscr();
     printf("\n Array has been Filled !!!");
     return;
    }
 mainarray[++top1]=elm;
       }
void push2(int elm)
 {
 if((top2-top1)==1)
    {
       clrscr();
       printf("\n Array has been Filled !!!");
       return;
    }
 mainarray[--top2]=elm;
 }
int pop1()
 {
    int temp;
    if(top1<0)
       {
          clrscr();
          printf(" \n This Stack is Empty !!!");
          return -1;
       }
    temp=mainarray[top1];
    top1--;
    return temp;
 }
int pop2()
 {
    int temp;
    if(top2>MAX-1)
       {
          clrscr();
          printf(" \n This Stack Is Empty !!!");
          return -1;
       }
    temp=mainarray[top2];
    top2++;
    return temp;
 }
```

Q141. Write an algorithm to insert a node p at the end of a linked list. **(5)**

Ans.

Algorithm to Insert a Node p at the End of a Linked List

```
Step1:   [check for space]
      If new1= NULL output "OVERFLOW"
         And exit
Step2:   [Allocate free space]
         New1 = create new node.
Step3:   [Read value of information part of a new
node]
         Info[new1]=value
Step4:   [Move the pointer to the end of the list]
node = previous=strt
      Repeat while Node != NULL
      Previous=node
Node = Next[Node]
Step5:   [Link currently created node with the
last node of the list]
      Next[New1] = Node
      Next[Previous] = New1
Step6:   Exit.
```

Q142. Write down any four applications of queues. **(4)**

Ans.

Application of Queue

(i) Queue is used in time sharing system in which programs with the same priority form a queue while waiting to be executed.

(ii) Queue is used for performing level order traversal of a binary tree and for performing breadth first search at a graph.

(ii) Used in simulation related problem.

(iii) When jobs are submitted to a networked printer, they are arranged in order of arrival, ie. Jobs are placed in a queue.

Q.143 Draw a binary tree from its inorder and preorder traversal sequences given as follows:

Inorder : d b g e h a c n f

Preorder : a b d e g h c f n **(7)**

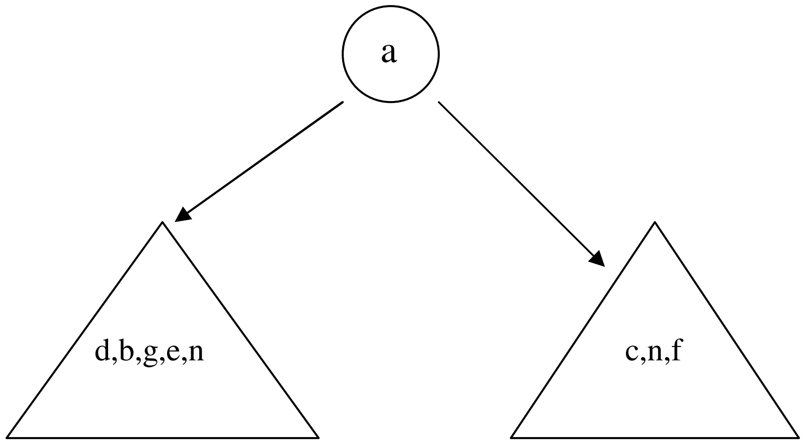
Ans.

Inorder :

	LTA	**Root**	**RTA**
→	d,b,g,e,h	a	c,n,f

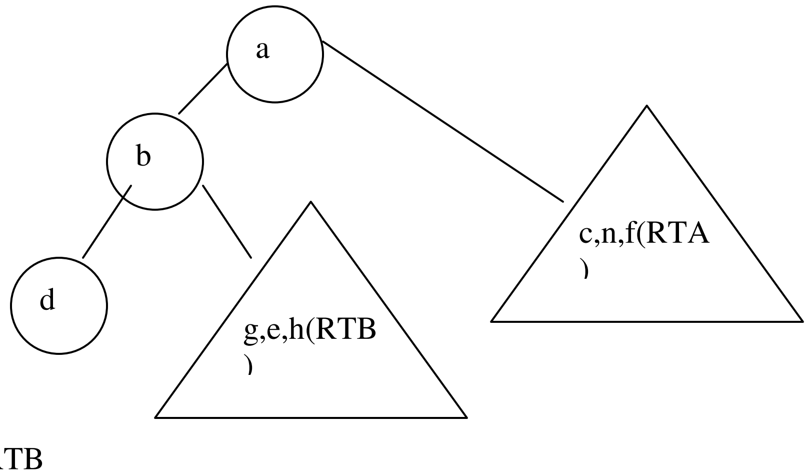
Preorder :

	Root	**LTA**	**RTA**
→	a	b,d,e,g,h	c,f,n

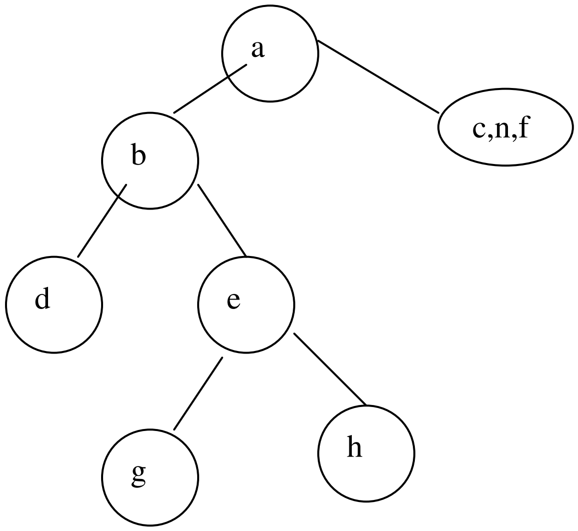


Let us consider LTA.

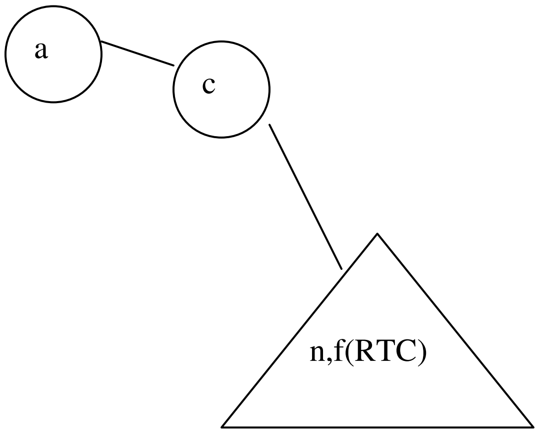
In order	: d, b, g, e, h
Preorder :	b,d, e, g, h



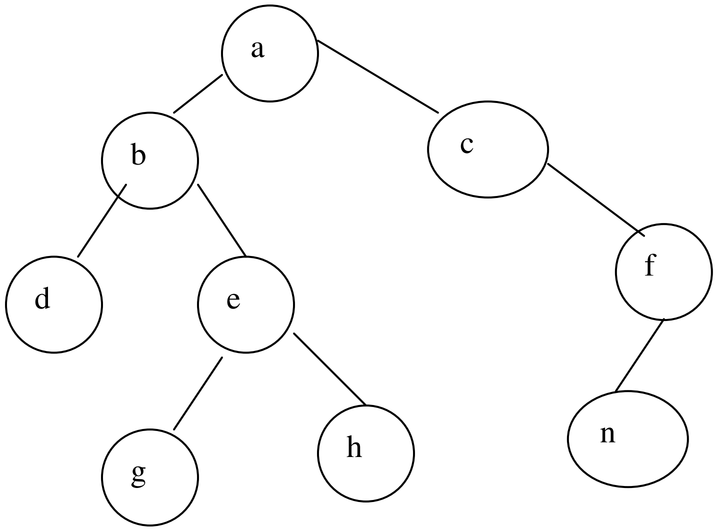
Let us consider RTB
In order : g , e , h
Pre order : e, g, h



Let us consider RTA
In order : c,n,f,RTC
Pre order : c, F , n
Hence the tree is



Let us consider RTC
In order : f, n
Pre order : n , f (root)



Final tree

Q144. Write an algorithm that counts number of nodes in a linked list. **(5)**

Ans.

Algorithm to Count No. of Nodes in Linked List

COUNT (INFO, LINK, START, NUM)

```
1. Set NUM: = 0 [initializes counter]
2. Set PTR=START [ initializes pointer]
3. Repeat steps 4 & 5 while PTR! = NULL
4. Set NUM:= NUM + 1 [ increases NUM by 1]
5. Set PTR: =LINK [PTR] . [updates pointer to
   point to next node]
   [End of step 3 loop]
6. Return
```

Q145. Write an algorithm to add an element at the end of circular linked list. **(5)**

Ans.

Algorithm to Add the Element at the End of Circular Linked List.

```
IINSENDCLL( INFO, LINK, START, AVAIL, ITEM)
The algorithm deletes last element from the
 circular linked list.
1.  [OVERFLOW?] if AVAIL = NULL, then Write:
       OVERFLOW, and Exit.
2.  [Remove first node from the AVAIL: =
       LIN[AVAIL].
a.  Set NEW:= AVAIL and AVAIL:=LINK[AVAIL].
3.  Set INFO[NEW]:=ITEM. [copies new data into new
       node.]
4.  Set  PTR:= LINK[START] and
       SAVE:=START.[initializes popinters]
5.  Repeat while LINK[PTR]!=START: [ Traverses list
       seeking last node.]
a.  Set PTR:=LINK[PTR]. [Updates PTR]
            [ End of loop]
6.  Set LINK [PTR]:= NEW. [ Attaches new node to
       the last node of the list]
7.  Set LINK[NEW]:= START [ New node now points to
       the original first node.]
8.  Exit
```

Q146. Delete a given node from a doubly linked list. **(4)**

Ans.

Delete a Node from Double Linked List

```
DELETEDBL(INFO, FORW, BACK, START, AVAIL,LOC)
     1. [Delete Node]
        Set FORW [ BACK [LOC]]:= FORW[LOC]&
        BACK [FORW[LOC]]:=BACK[LOC].
        2. [Return node to AVAJL list]
           Set FORW[LOC]:=AVAIL & AVAIL:=LOC
        3. Exit.
```

Q147. Write an algorithm to sort a given list using Quick sort method. Describe the behaviour of Quick sort when input is already sorted. **(7)**

Ans.

Algorithm for Quick Sort

```
 QUICK(A, N, BEG, END, LOC)
 Here A is an array with N element. Parameter BEG
and END contain the      boundary value of the sub
list of A to which this procedure applies. LOC
keeps track of the position of the first element
A[BEG] of the sublist during the procedure. The
local varrible LEFT   and  RIGHT will contain the
boundary value of the list elements that have not
been scanned.
  1. [Initialize]    Set LEFT:=BEG, RIGHT;=END and
      LOC:=BEG.
 2. [Scan from left to right]
    (a) Repeat while A[LOC] <=A[RIGHT]  and
    LOC!=RIGHT;
     RIGHT:=RIGHT-1;
     [End  of loop]
     (b)If LOC= RIGHT, then Return;
                 (c)If A[LOC ] > A[RIGHT],then:
                  [Interchange  A[LOC] and A[RIGHT]]
      TEMP:=  A[LOC]  ,A[LOC] =  A[RIGHT] ,
                   A[RIGHT] :=TEMP;
      (i)  Set LOC =RIGHT
      (ii)  Go to step 3
 3.[Scan from left to right]
   repeat while A[LEFT] <=A[LOC]  and  LEFT!= LOC;
   LEFT := LEFT +1;
                  [End of loop]
    (a) If LOC  =LEFT, then Return;
    (b) If  A[LEFT]  > A[LOC] ,then
   (i) [Interchange  A[LEFT]  and A[LOC]]
          TEMP:=A[LOC],A[LOC]:=A[LEFT] .
          A[LEFT]:= TEMP
    (ii) Set LOC :=LEFT
    (iii) Go to Step 2;
    [End of if structure]
 (Quicksort)  This algorithm sorts an array A with
 N elements.
 1. [Intialize.] TOP := NULL
 2. [Push boundary values of A onto stacks when A
 has 2 or more elements.]
```

```
If  N>1, then: TOP+1,LOWER [1]:=1, UPPER [1]: =N
3. Repeat steps 4 to 7 while TOP != NULL.
4. [Pop sublist from stacks.]
Set BEG: =LOWER[TOP], END:=UPPER[TOP],
TOP:=TOP-1.
5. Call QUICK (A, N, BEG, END, LOC).  [ Push left
sublist onto stacks when it has 2 or more
elements.]
If BEG < LOC -1, then:
      TOP:= TOP+1, LOWER[TOP] := BEG,
        UPPER[TOP]= LOC -1.
[End of If structure.]
6. [Push right sublist onto stacks when it has 2
or more elements.]
If  LOC +1< END , then:
      TOP := TOP+1, LOWER[TOP] := LOC +1,
       UPPER[TOP] := END.
[End of If structure .]
[End of Step 3 loop.]
7. Exit.
```
The behaviour of quick sort when the list is sorted is of order $O(n^2)$ as this is the worst case for quicksort

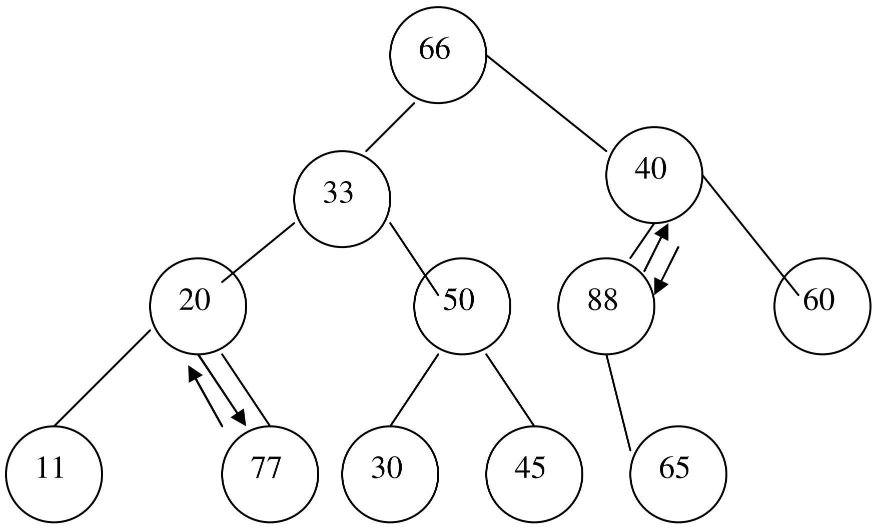
Q148. Sort the following list using Heap Sort
66, 33, 40, 20, 50, 88, 60, 11, 77, 30, 45, 65. **(7)**

Ans.

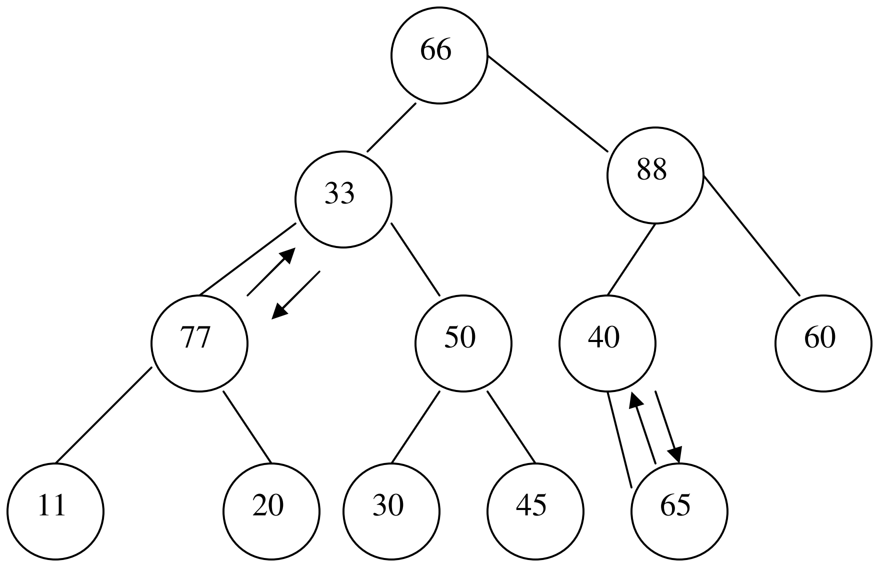
Heap Sort

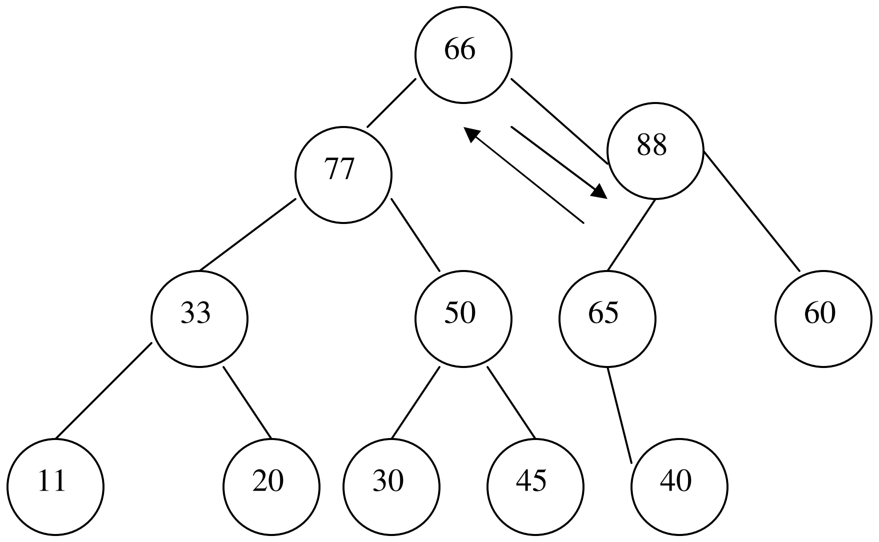
Let the array represents implicit representation:-

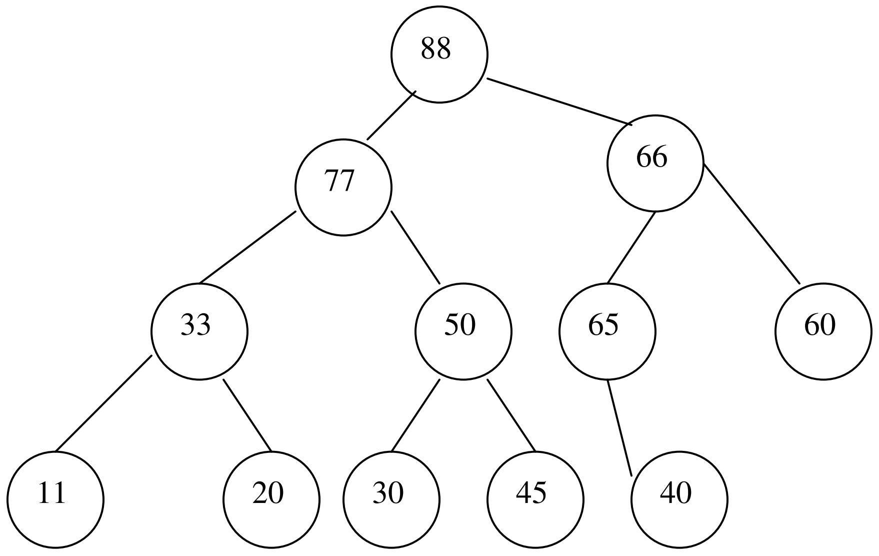
66	33	40	20	50	88	60	11	77	30	45	65



First let us make the heap.





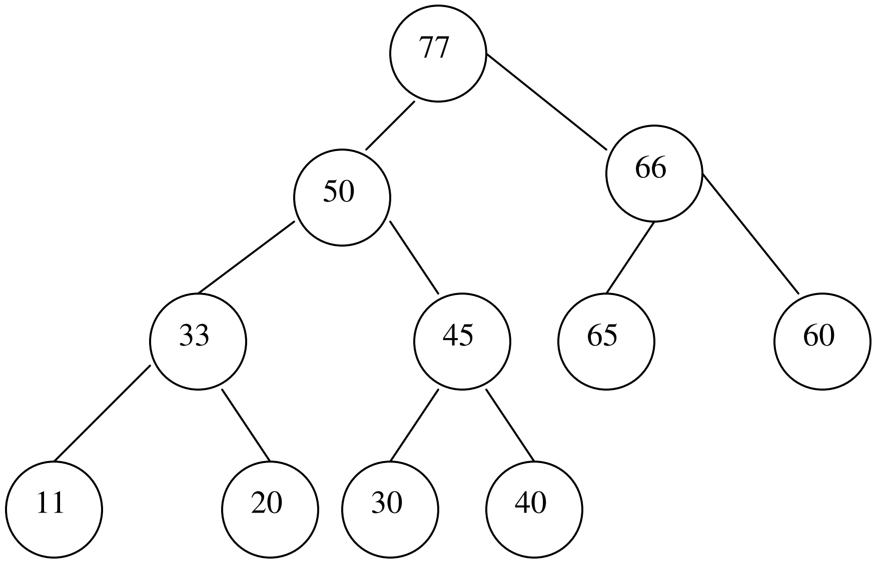


The above is new heap satisfying all conditions of heap.

88	77	66	33	50	65	60	11	20	30	45	40	

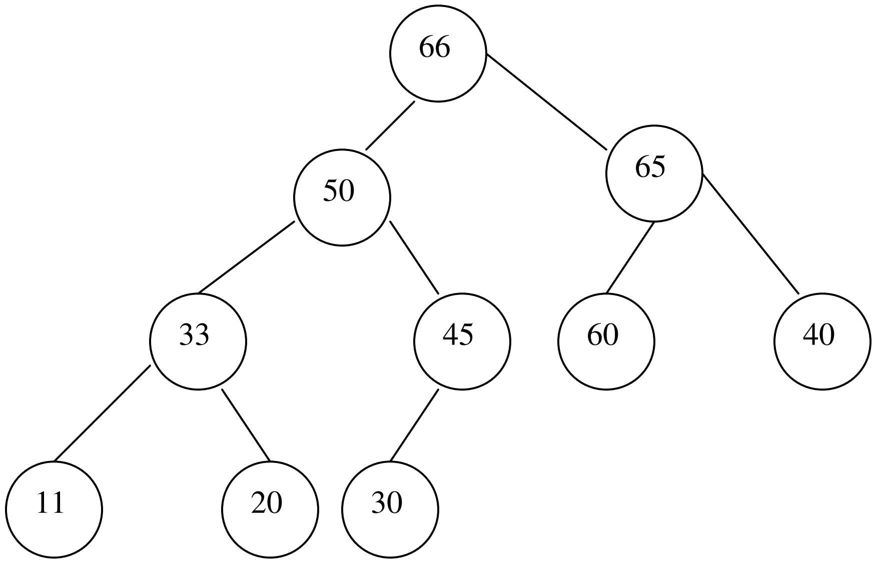
Place 88 at its proper position and replace 40

88



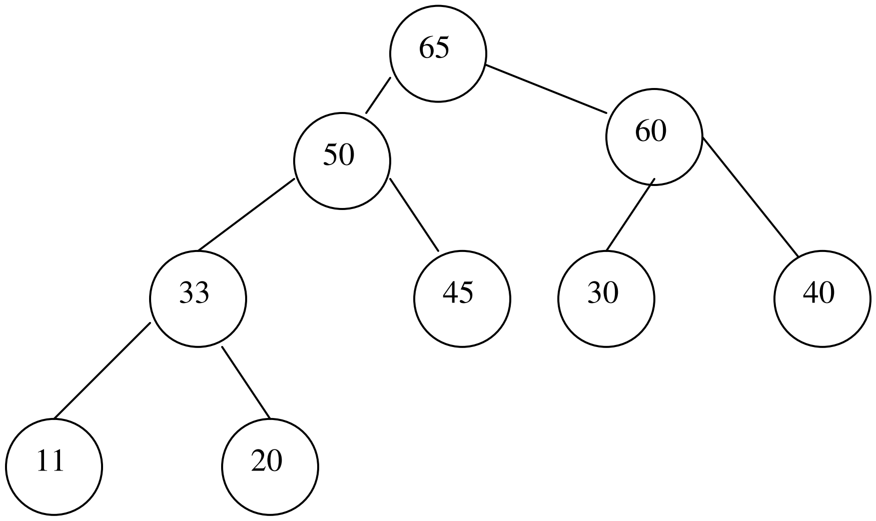
77	50	66	33	45	60	65	11	20	30	40	88	

77



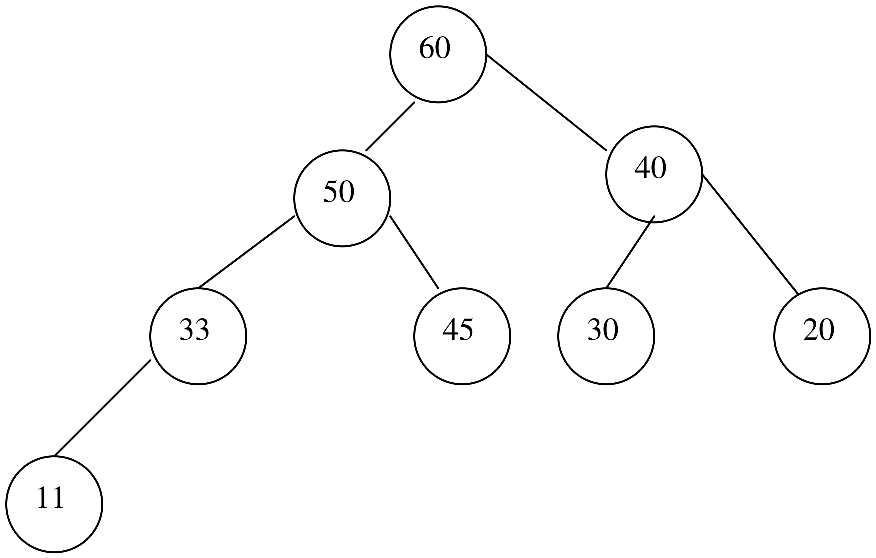
66	50	65	33	45	60	40	11	20	30	77	88	

66



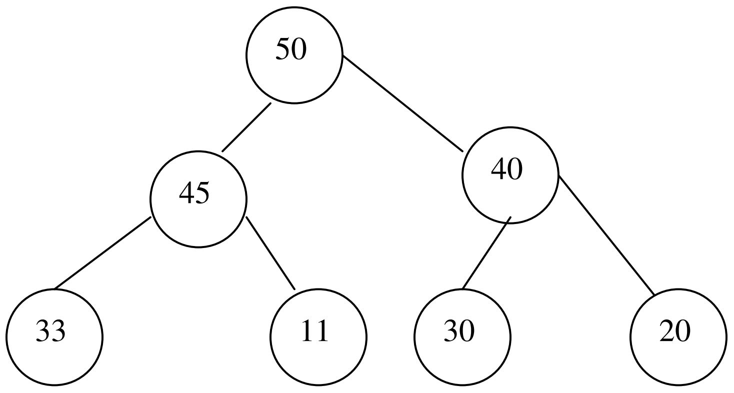
65	50	60	33	45	30	40	11	20	66	77	88

65



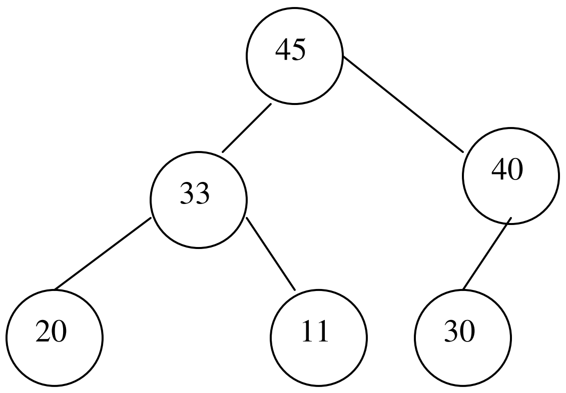
60	50	40	33	45	30	20	11	65	66	77	88

60



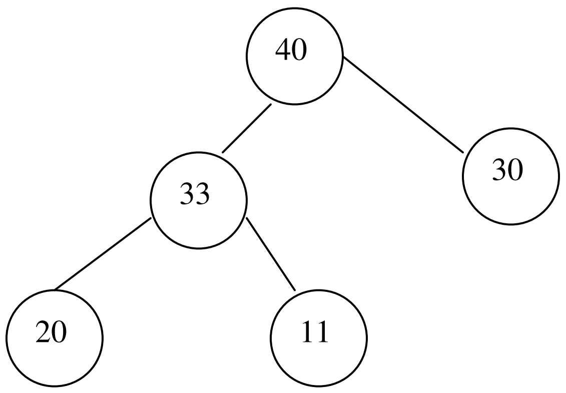
50	45	40	33	11	30	20	60	65	66	77	88

50



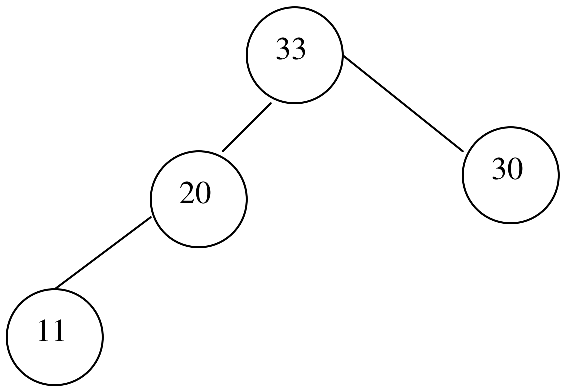
45	33	40	20	11	30	50	60	65	66	77	88

45



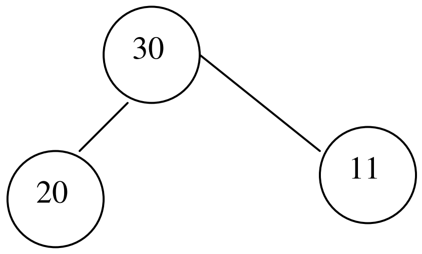
40	33	30	20	11	45	50	60	65	66	77	88

40



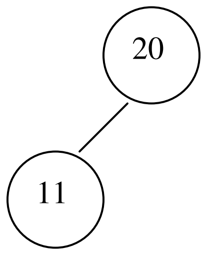
33	20	30	11	40	45	50	60	65	66	77	88

33



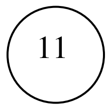
30	20	11	40	45	50	60	65	66	77	88

30



20	11	30	40	45	50	60	65	66	77	88

20



11	20	30	40	45	50	60	65	66	77	88

Finally we have the sorted list.

Q149. Write a recursive function to count the number of nodes in a binary tree. **(7)**

Ans.

Recursive Function to count no. of Nodes in Binary Tree

The number NUM of nodes in T is 1 more than the number NULL of nodes in the left subtree of T plus the number NUMR of nodes in the right subtree of T.

```
Count( LEFT , RIGHT , ROOT , NUM.)
 This procedure finds the number NUM of nodes in a
binary tree T in memory.
1.   If ROOT = NULL, then : Set NUM := 0, & Return.
2.   Call COUNT( LEFT , RIGHT , LEFT[ ROOT], NUML).
3.   Call COUNT ( LEFT , RIGHT, RIGHT[ ROOT], NUMR).
4.   Set NUM := NUML +NUMR+1.
5.   Return.
```

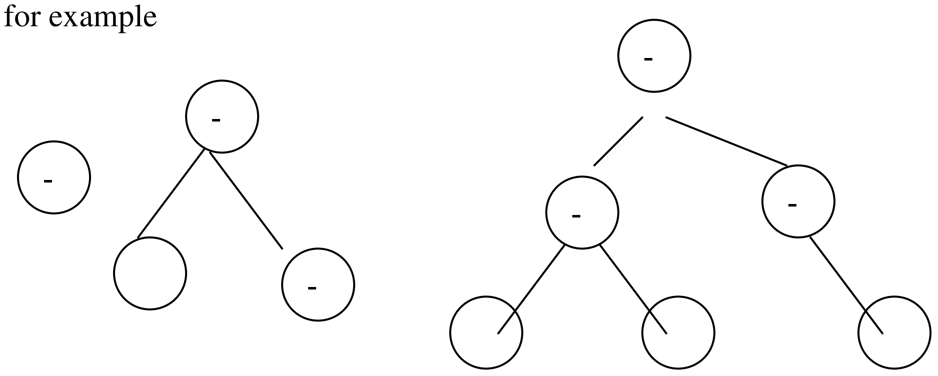
Q150. Define the following :

(i) AVL tree.
(ii) Thread.
(iii) Heap.
(iv) Binary Search Tree. **(8)**

Ans

(i) Height Balanced Tree (AVL Tree)

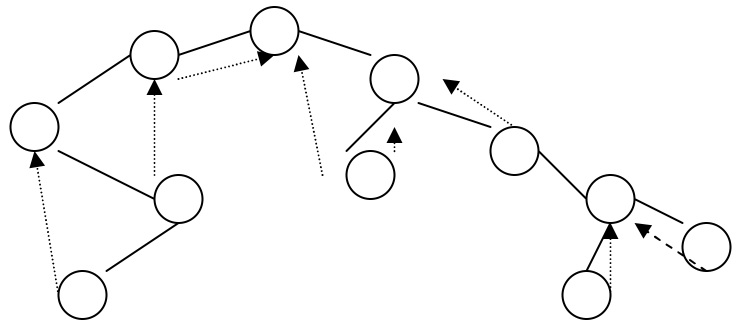
An **AVL tree** is a binary search tree in which the height of the left and right subtree of the root differ by at most 1 and in which the left and right subtrees are again AVL trees. With each node of an AVL tree is associated with a balanced factor that is left high, equal or right high, according, respectively, as the left subtrees has height greater than, equal to, or less than that of the right subtree.
The definition does not require that all leaves be on the same or adjacent levels.
for example



(ii) Thread

By changing the NULL lines in a binary tree to special links called **threads,** it is possible to perform traversal, insertion and deletion without using either a stack or recursion.
In **a right in threaded binary tree** each NULL link is replaced by a special link to the successor of that node under inorder traversal called right threaded. Using right

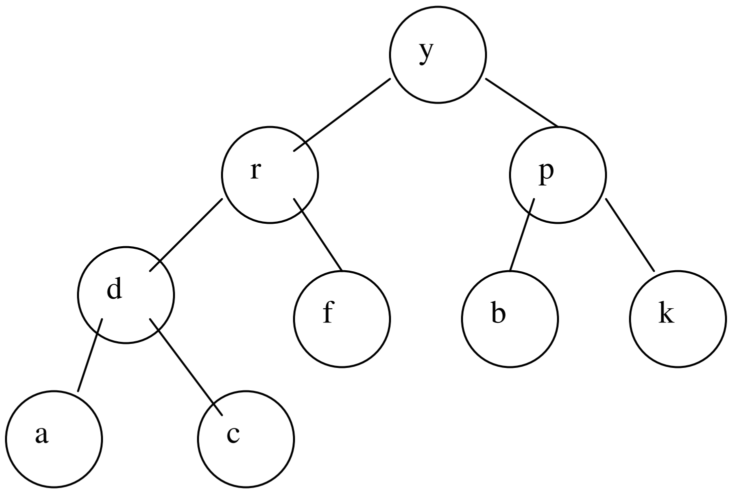
threads we shall find it easy to do an inorder traversal of the tree, since we need only follow either an ordinary link or a threaded to find the next node to visit.
If we replace each NULL left link by a special link to the predecessor of the node known as left threaded under inorder traversal the tree is known as **left in threaded binary tree**. If both the left and right threads are present in tree then it is known **as fully threaded binary tree.**



Fully threaded binary tree

(iii) HEAP

A heap is defined to be a binary tree with a key in each node, such that
1-All the leaves of the tree are on 2 adjacent levels.
2- All leaves on the lowest level occur to the left & all levels except possibly the lowest are filled.
3- The information the root is at least as large as the keys in its children (if any) & the left & right sub trees (if they exist) are again heap.
for example



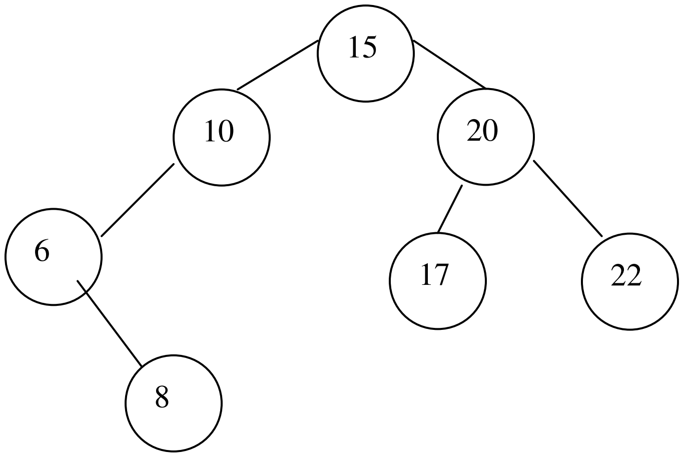
y	r	p	d	f	b	k	a	c

(iv) Binary search tree.

A binary search tree is a binary tree that is either empty or in which each node contains a key that satisfies the following conditions: -

- All keys (if any) in the left sub tree of the root precede the key in the root.

- The key in the root precedes all keys (if any) in its right sub tree.
- The left and right sub trees of the root are again search trees.



Q151. Write an algorithm for searching a key from a sorted list using binary search technique. **(6)**

Ans.

Binary Search Algorithm

```
1.   if (low > high)
2.     return (-1)
3.    mid = (low +high)/2;
4    .if ( X = = a [mid])
5      return (mid);
6     if ( X < a [mid])
7    search for X in a (low) to [mid -1];
8    else
9     search for X in a [mid + 1] to a [high];
```

Q152. Define graph, adjacency matrix, adjacency list, hash function, sparse matrix, reachability matrix. **(6)**

Ans.

Graph

A graph G, consists of 2 sets V & E. V is a finite non-empty set of vertices. E is a set of pairs of vertices, these pairs are called edges.

Adjacent Matrix:-

Let G=(V,E) be graph with n vertices, n>=1.

The Adjacency Matrix of G is a 2- dimensional n*n array , say A, with the property that A(i,j) = 1 iff the edge (v_i,v_j) (< $_{vi}$, $_{vj}$> for a directed graph) is in E(G). A(i,j)= 0 if there is no such edge in G.

Adjacency Lists

In this representation the n rows of the adjacency matrix are represented as n linked list. There is one list for each vertex in G. The nodes in list represents the vertices that are adjacent from vertex i.

Hash Function
A hash function h is simply a mathematical formula that manipulates the key in some form to compute the index for this key in the hash table In general, we say that a hash function h maps the universe U of keys into the slots of a hash table T[0….n-1]. This process of mapping keys to appropriate slots in a hash table is known as **hashing.**

Sparse Matrix
A m x n matrix A is said to be sparse if most of its elements are zero. A matrix that is not sparse is called dense matrix.
for example:

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

Reachability Matrix /Path Matrix:-
Let G be a simple directed graph with m nodes, V_1, V_2, V_3……..V_m. The reachability matrix of G is the m –square matrix $P=(P_{ij})$ defined as follows

$$Pij = \begin{cases} 1 & \text{if there is a path from } V_i \text{ to } V_j \\ 0 & \text{otherwise} \end{cases}$$

Q153. Explain various graph traversal schemes and write their merits and demerits. **(8)**

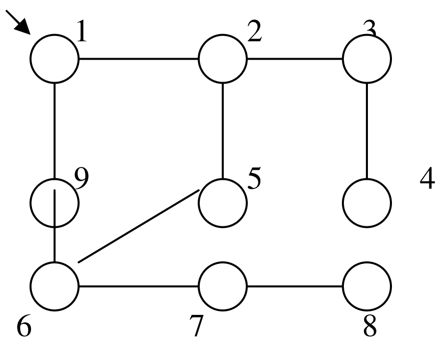
Ans.
Graph Traversal Scheme.

In many problems we wish to investigate all the vertices in a graph in some systematic order. In graph we often do not have any one vertex singled out as special and therefore the traversal may start at an arbitrary vertex. The two famous methods for traversing are:-

a) **Depth first traversal**: of a graph is roughly analogous to pre-order traversal of an ordered tree. Suppose that the traversal has just visited a vertex or, and let W_0, W_1,……W_k be the vertices adjacent to V. Then we shall next visit W_0 and keep W_1…..W_k waiting. After visiting W_0 we traverse all the vertices to which it is adjacent before returning to traverse W_1, W_2, ……..W_k.

b) **Breadth first traversal**: of a graph is roughly analogous to level by level traversal of ordered tree. If the traversal has just visited a vertex V, then it next visits all the vertices adjacent to V. Putting the vertices adjacent to these is a waiting list to be traversed after all the vertices adjacent to V have been visited.

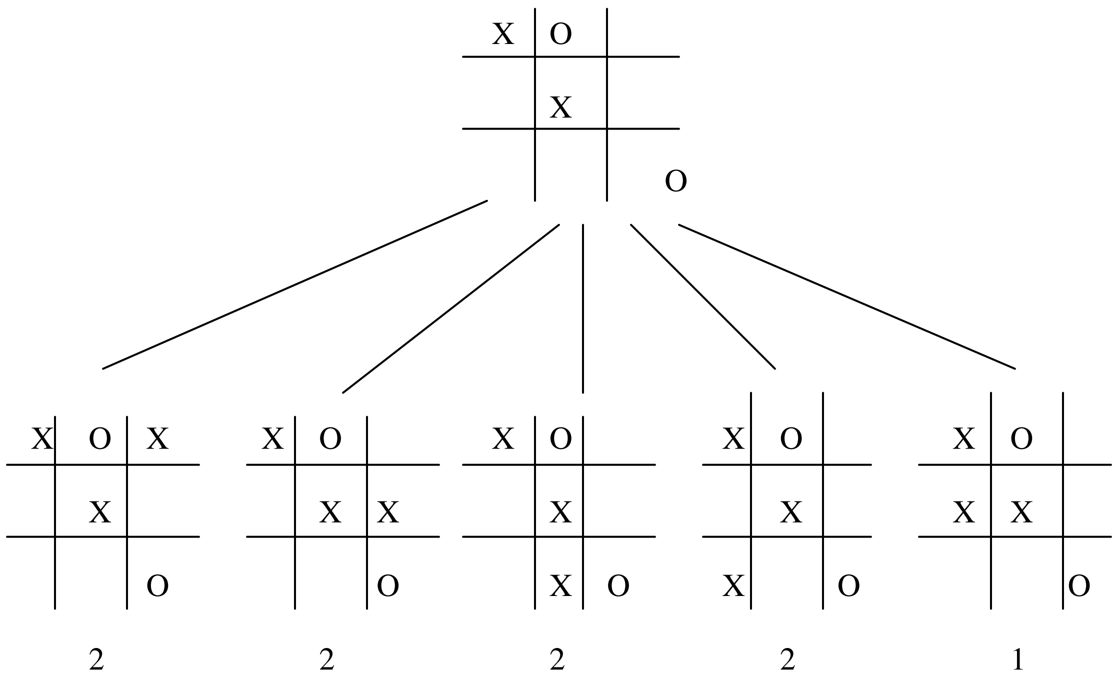
The figure below shows the order of visiting the vertices of one graph under both DFS and BFS.



DFT = 1 2 3 4 5 6 7 8 9
BFT= 1 2 9 3 5 6 4 7 8

Q154. Write short notes on the following:
(i) Decision and game trees.
(ii) Polynomial representation and manipulation using linked lists.
(iii) Analysis of algorithm.
(iv) Circular queues. **(3 ½ x 4 = 14)**

Ans.
(i) Decision Tree and game Tree
Decision Tree
A decision tree is a diagram that represents conditions and actions sequentially and thus shows which condition is to be considered first, second and so on. It is also a method of showing the relationship of each condition and its permissible actions. In the decision tree, the root of the tree is the starting point of the decision sequence. The particular branch to be followed depends on the conditions that exists and decisions to be made progressing along a particular branch is the result of making a series of decisions. The node of a tree represent conditions.
GAME TREES
An interesting application of trees is the playing of games such as tie-tac-toe, chess, chess, etc.
We can picture the sequence of possible moves by mean of a game tree in which the root denotes the initial situation and the branches from the root denote the legal moves that the first player could make. At next level down, the branches corresponds to the legal move by the second player in situation and so on ,with branches from vertices at even levels denoting moves by the 1st player and from vertices at odd levels denoting moves by he 2nd player.
for example. Tic-tac-toe



(ii) Circular Queues:-

A more efficient queue representation is obtained by regarding the array Q(1:n) as circular. It becomes more convenient to declare the array as Q(O: n-1), when
rear = n-1, the next element is extended at Q(O) in the case that spot in free. Front will always point one position counterclockwise from the first element in the queue. Again, front = rear if and only if the queue is empty. Initially we have
front = rear = 1. In order to add an element it will be necessary to move rear one position clockwise i.e.
If rear = n-1 then rear ← O
Else rear ← rear + 1
Using the modulo operator which completes remainder. This is first rear ← (rear +1) mod n. Similarly, it will be necessary to move front one position clockwise each time a deletion is made. This can be accomplished by front ← (front +1) mod n.

Q155. What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine. **(4)**

Ans.

F (n) = $100\ n^2$
G(n) = 2^n
Value of n so that $100\ n^2 < 2^n$
The smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine is 15.

Q156. Explain any three methods of representing polynomials using arrays. Write which method is most efficient for representing the following polynomials.

$8x^{100}+10x+6$
$8x^3-7x^2+5x+15$

Ans.
3 methods of representing polynomials using arrays are as follows
(1) if maximum value of exponent of a polynomial is m then define an array of size m+1 and store coefficient in corresponding index position as exponent. Ex: $2x^2 +1$ is stored as

1	0	2

(2) 1 one-dimensional array is used to store exponent and coefficient alternatively. Ex: $2x^2 +1$ is stored as

2	2	1	0

The size of array required is 2*n where n is the number of elements in polynomial.
(3) Use 2 arrays or one-dimensional array of structures one for storing exponents and other for co-efficient.
Ex: $2x^2 +1$ is stored as

2	2
1	0

The size of arrays is 2*n where n is the number of elements in polynomial.
(i) The second and third methods are the efficient methods.
For storing $8x^{100}+10x+6$, as in method 1 there is a requirement of 101 integer locations.
(ii) $8x^3-7x^2+5x+15$ for this polynomial any one of the representations can be used, but method 1 will be best as there is only coefficients need to be stored. There are no gaps in the exponents, hence the entire array will be filled with the coefficients.

Q157. Let X = (X1, X2, X3,….Xn) and Y= (Y1, Y2, Y3,….Xm) be two linked lists. Write an algorithm to merge the lists together to obtain the linked list Z such that
Z = (X1, Y1, X2, Y2,….Xm, Ym,Xm+1….Xn) if m<=n or
Z = (X1, Y1,X2,Y2….Xn,Yn,Yn+1….Ym) if m>n. **(7)**

Ans.
```
void near (*head1,*head2,*head3)
{
  p1 = head1;
  p2 = head2;
     p3 = head3;
if (p1 != NULL)
{
    p3 = head3 = p1 ;
   p1=p1 ⟶ next ;
```

```
       }
       while (p1!= NULL && p2 != NULL)
         {
            p3  →  next = p2 ;
            p2=p2  →  next;
            p3=p3   →  next;
            p3    next = p1;
            p1 = p1    next ;
            p3 = p3    next ;
          }
       for ( ; p1 != NULL; p1 =  p1 □.next)
          {
            p3      next = p1;
            p3 = p3        next ;
          }
       for ( ; p2 != NULL ; p2 = p2 □ next)
         {
          P2       next = p2 ;
          p2 = p2   next ;
         }
```

Q158. Devise a representation for a list where insertions and deletions can be made at either end. Such a structure is called a Deque (Double ended queue). Write functions for inserting and deleting at either end. **(7)**

Ans.

There are various ways of representing dequeue in a computer. We will assume that dequeue is maintained by a circular array DEQUE with pointer FRONT and REAR, which point to the 2 ends of the dequeue. We assume that the element extend from the left end to the right end in the array. The condition FRONT = NULL will be used to indicate that dequeue is empty.

Algorithm for inserting in dequeue

```
Step 1:  [check overflow condition]
   If rear ≥ n and front = 0
   Output :overflow and return"
Step 2:  [check front pointer value]
   If front > 0
   Front = front -1
   Else
   Return
Step 3:  [insert element at the front end]
   Q[front ] = value
Step 4:  [check rear pointer value]
   If rear < n
   Rear = rear +1
   Else
   Return
Step 5:  [insert element at the rear end]
   Q [rear] = value
```

```
Step 6:  Return
```

Deletion Algorithm for dequeue

```
Step 1:  [check for underflow]
      If front = 0 and rear = 0
      Output "underflow" and return
Step 2:  [delete element at front end]
      If front > 0
      Value = q [front]
      Return [value]
Step 3:  [check queue for empty]
      If front = rear
      Front = rear = 0
      Else
      Front = front +1
Step 4:  [delete element at the rear end]
      If rear > 0
      Value = Q [rear]
      Return (rear)
Step 5:  [check queue for empty]
      If front = rear
      Front = rear = 0
      Else
      Rear = rear - 1
Step 6:   Return
```

Q159. Using stacks, write an algorithm to determine whether the infix expression has balanced parenthesis or not. **(7)**

Ans.

Algorithm parseparens

This algorithm reads a source program and parses it to make sure all opening – closing parenthesis are paired

```
1. loop (more data)
read (character)
if (character is not a  closing parenthesis)
pushstack ( stack, character)
else
if (character is closing parenthesis)
popstack (stack, token)
while ( token is not  an opening paranthesis)
popstack (stack, token)
endwhile
end if

end if
end loop
 if (not emptystack (stack))
   print (Error: opening parenthesis not matched)
end parseparens.
```

Q160. Implement a stack using linked list. Show both the PUSH and POP operations. **(7)**

Ans.

```
Stack implemantation using linked list
# include<stdio.h>
# include<malloc.h>
struct link
{
         int info ;
         struct link *next;
}   *start;
struct link * push (struct link * rec)
{
        struct link *new_rec;
        printf ("\n Input the new value for next
location of the stack:") ;
        new_rec = (struct link *) malloc (size of
(struct link)) ;
        scanf ("%d", &new_rec->info) ;
        new_rec->next = rec;
        rec = new_rec;
        return (rec);
}
struct link * pop (struct link * rec)
{
        struct link * temp ;
           if (rec == NULL)
        {
                printf ('\n Stack is empty") ;
        }
        else
        {
                temp = rec; rec= temp->next;
printf("the popped element  %d", temp->.info);
   free(temp) ;
      return(rec);
}
```

Q161. Write binary search algorithm and trace to search element 91 in following list:

13 30 62 73 81 88 91

What are the limitations of Binary Search? **(7)**

Ans.

13 30 62 73 88 91
let us store the numbers in an array

13	30	62	73	88	91
0	1	2	3	4	5

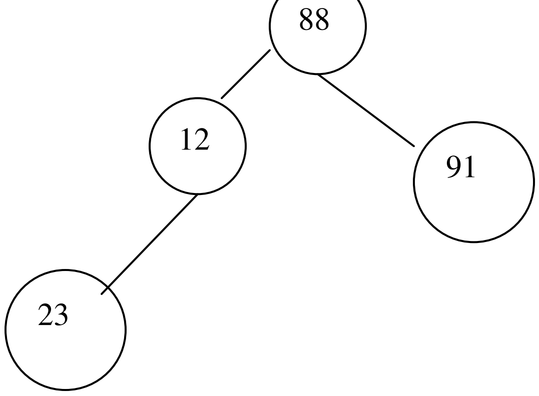
To begin with we take low = 0 and high = 5
therefore mid = (low+high)/2 = (0+5) /2 = 2
since a[mid] = a[2] = 62 ≠ 91 and low < high we go to next iteration as 62 < 91
∴we take low = mid+1 = 3 and high =
now we find mid =(3+5)/2 = 4
∴a [mid] = a[4] = 88.
88 is not equal to 91
again 88< 91
∴ we take low = mid +1 = 4 + 1 = 5 and high = 5
∴mid = (5+5)/2 = 5
∴ a[mid] = a[5] = 91 which is the search element.

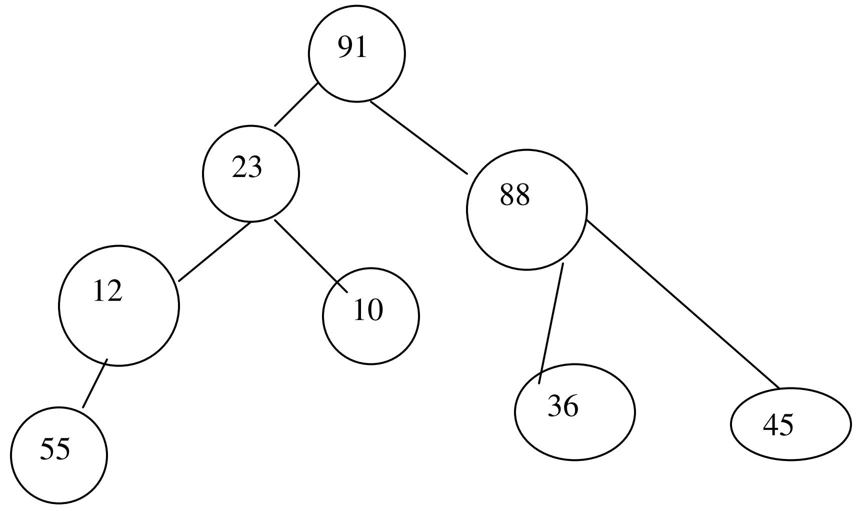
Limitation of Binary Search: -

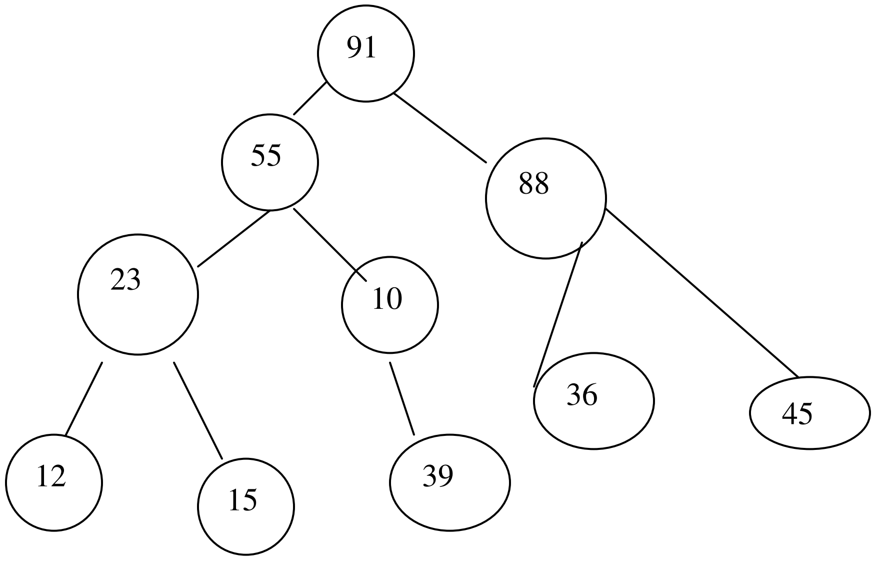
(i) The complexity of Binary search is O($\log_2 n$). the complexity is same irrespective of the the position of the element, even if it is not present in the array.
(ii) The algorithm assumes that one has direct access to middle element in the list on a sub list. This means that the list must be stored in some type of array. Unfortunately inserting an element in an array requires element to be moved down the list and deleting an element from an array requires element to be moved up the list.
(iii) The list must be sorted.

Q162. What are the two phases in heap sort algorithm? Sort the following data using heap sort and show all the intermediate steps.
88, 12, 91, 23, 10, 36, 45, 55, 15, 39, 81

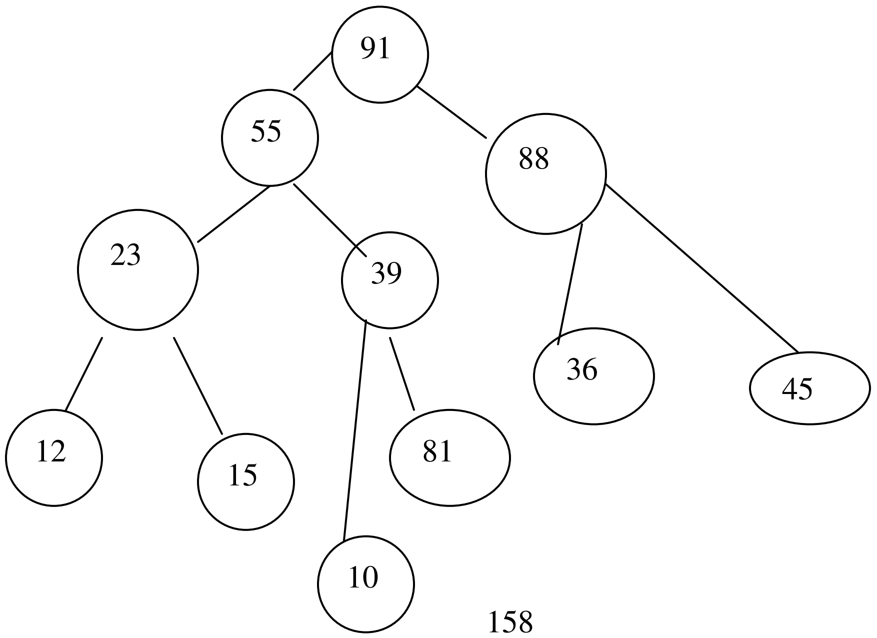
Ans.
The two phases of heapsort are : (1) Heap Creation (2) Sorting via deletion.
Sorting 88,12,91,23,10,36,45,55,15,39,8 using heap sort creating heap
88,12,91,23,10, 36, 45,55,15,39



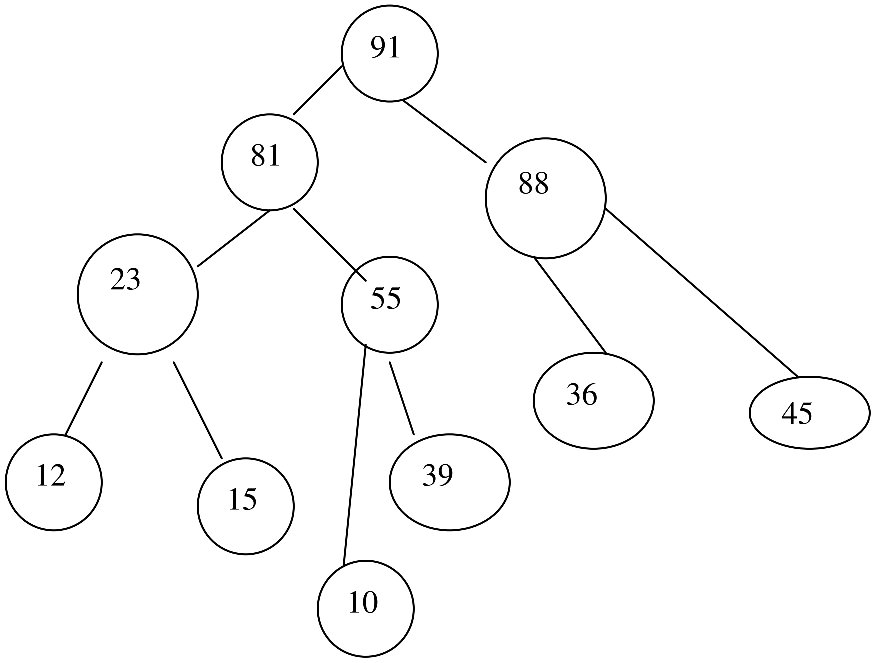

91
23
88
12
10
36
45
55


91
55
88
23
10
36
45
12
15
39

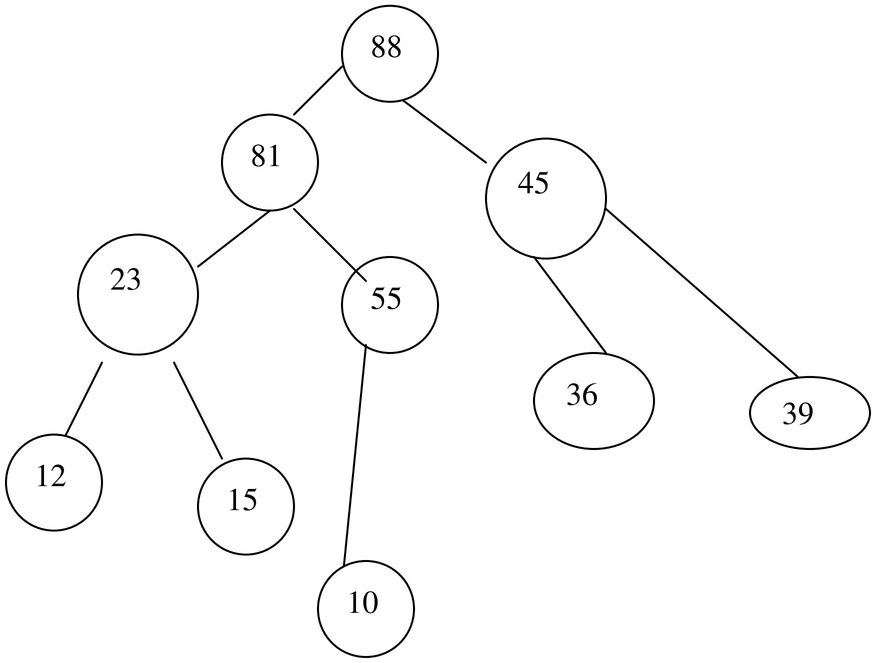
81:


91
55
88
23
39
36
45
12
15
81
10

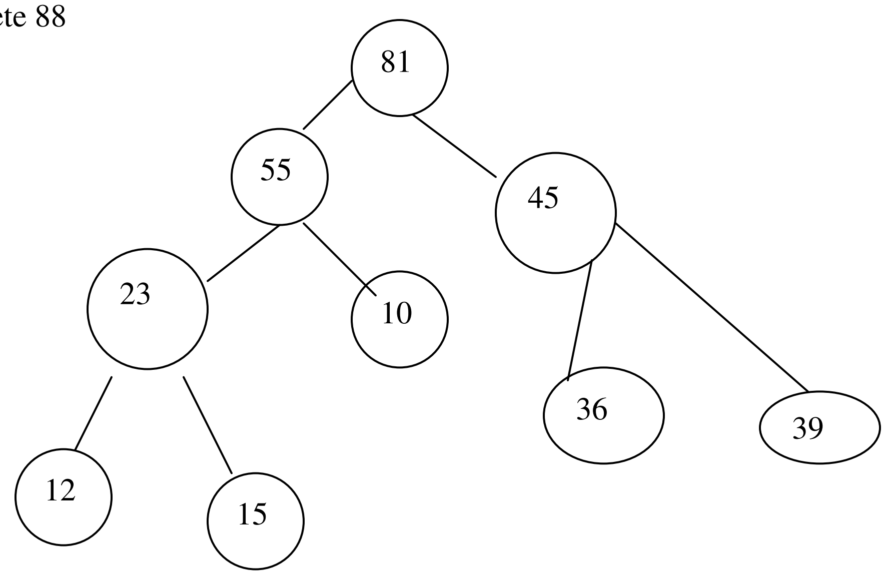
Heap:



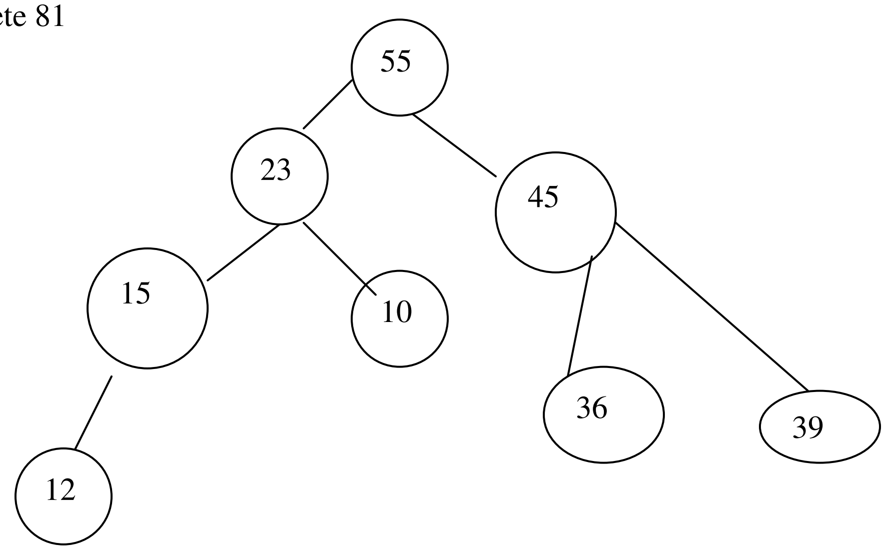
Delete 91



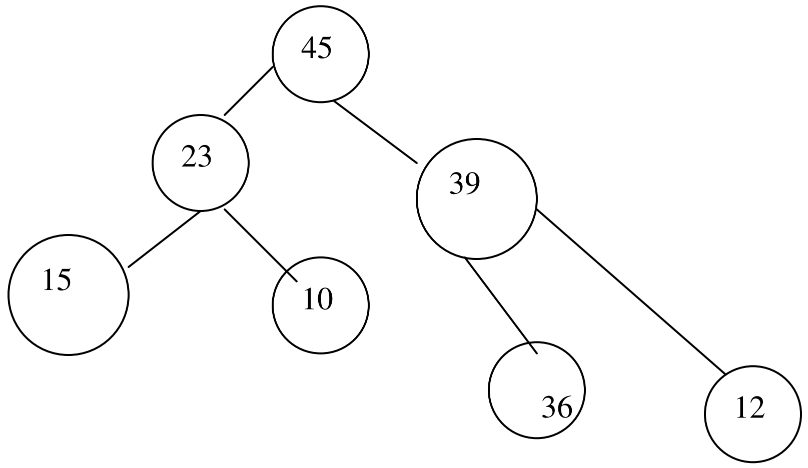
Delete 88



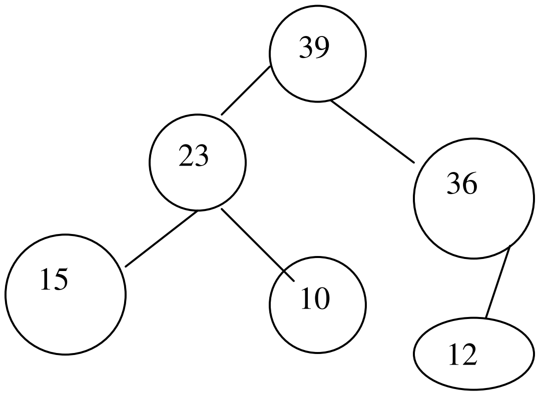
Delete 81



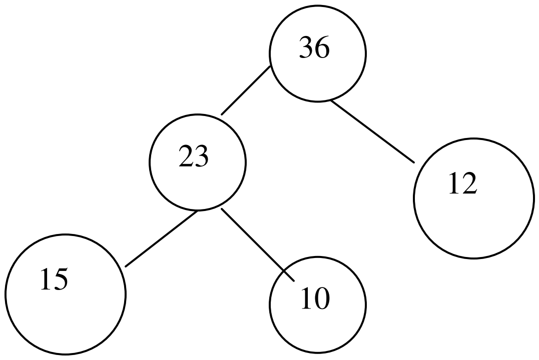
Delete 55



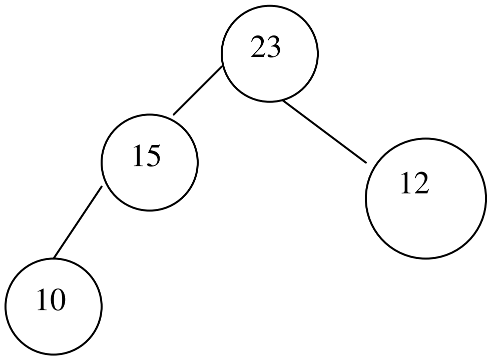
Delete 45



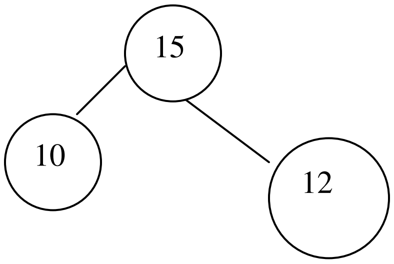
Delete 39



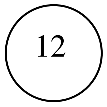
Delete 36

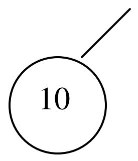


Delete 23

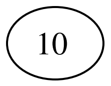


Delete 15





Delete 12



Delete 10
List is 10,12,15,23,36,39,45,55,81,88,91.

Q163. What is a Binary Search Tree (BST)? Make a BST for the following sequence of numbers.
45, 32, 90, 21, 78, 65, 87, 132, 90, 96, 41, 74, 92 **(7)**

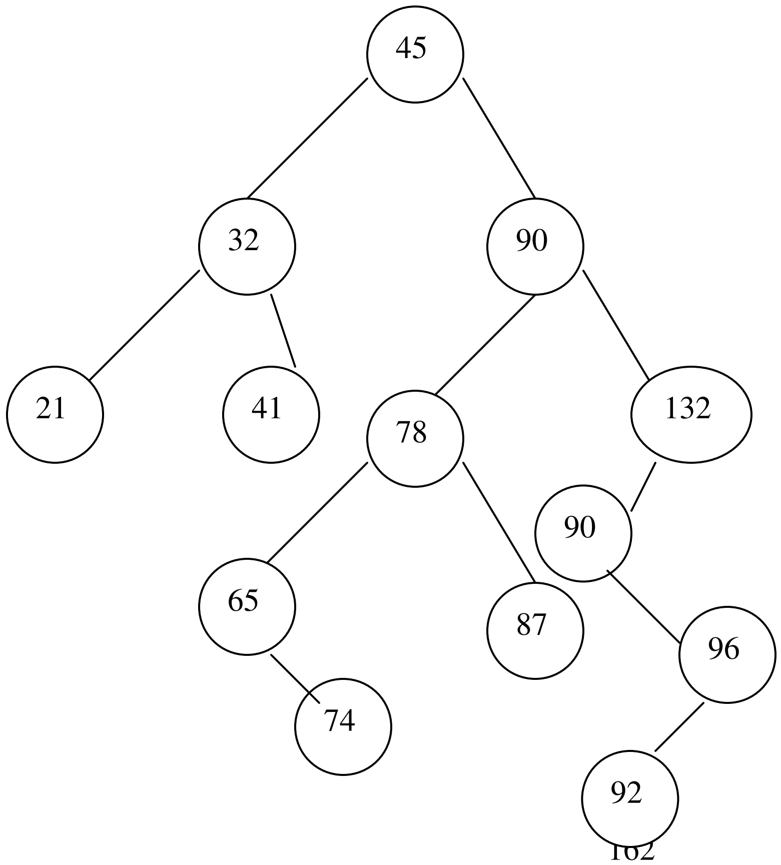
Ans.

Binary search tree.

A binary search tree is a binary tree that is either empty or in which each node contains a key that satisfies the following conditions: -

- All keys (if any) in the left sub tree of the root precede the key in the root.
- The key in the root precedes all keys (if any) in its right sub tree.
- The left and right sub trees of the root are again search trees.

The tree for the following list of numbers is
45 32 90 21 78 65 87 132 90 96 41 74 92



Q164. Traverse the Binary Search Tree created above in Preorder, Inorder and Postorder. **(7)**

Ans.

Preorder: VLR
45 32 21 41 90 78 65 74 87 132 90 96 92
Inorder: LVR
21 32 41 45 65 74 78 87 90 90 92 96 132
PostOrder: LRV
21 41 32 74 65 87 78 92 96 90 132 90 45

Q165. What is a sparse matrix? How is it stored in the memory of a computer? Write a function to find the transpose of a sparse matrix using this representation. **(8)**

Ans.

Sparse Matrix
A m x n matrix A is said to be sparse if most of its elements are zero. A matrix that is not sparse is called dense matrix.
Representing Sparse Matrix in Memory Using Array
In array representation an array of triplets of type < row, col, element> is used to store non-zero elements, where 1st field of the triplet is used to record row position second to record column and the 3rd to record the non zero elements of the sparse matrix.
In addition, we need to record the size of the matrix (ie number of rows and number of columns)and non zero elements of array of triplets is used for this purpose where the 1st filed stores the number of rows and the 2nd field stores the number of columns and the third field stores the number of non zero elements. The remaining elements of the array stores matrix on row major order. The array representation will use
[2 * (n+1) * size of (int) + n*size of(T)] bytes of memory where n is the number of non-zero elements and T is the data type of element.
Ex: consider a 5*6 sparse matrix below

$$\begin{pmatrix} 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 5 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 \end{pmatrix}$$

Array Representation of Sparse Matrix

$$\begin{pmatrix} 5 & 6 & 5 \\ 1 & 4 & 2 \\ 2 & 3 & 1 \\ 2 & 6 & 5 \\ 3 & 2 & 4 \\ 5 & 5 & 7 \end{pmatrix}$$

here n = 5 but size of array is 6 as first row stores the order of array along with number non-zero elements.

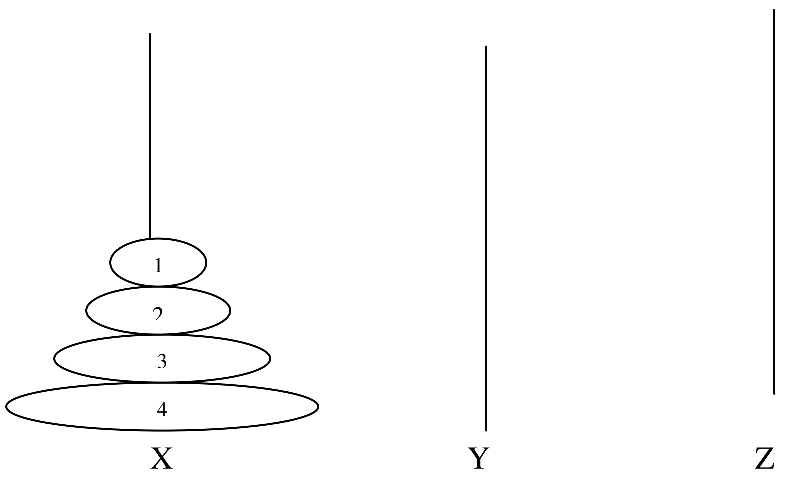
Memory declaration will be

```
# define Max 50
    struct triplet
    {   int row;
     int col;
     float element;
  }
  struct triplet sparse_mat [MAX];
    sparse matrix represented as above
    [n is the number of non zero elements in array]
    for I= 1,2,…n+1
    temp = a[I].row
    a[I].row= a[I].col
    a[I].col = temp
    endfor.
```

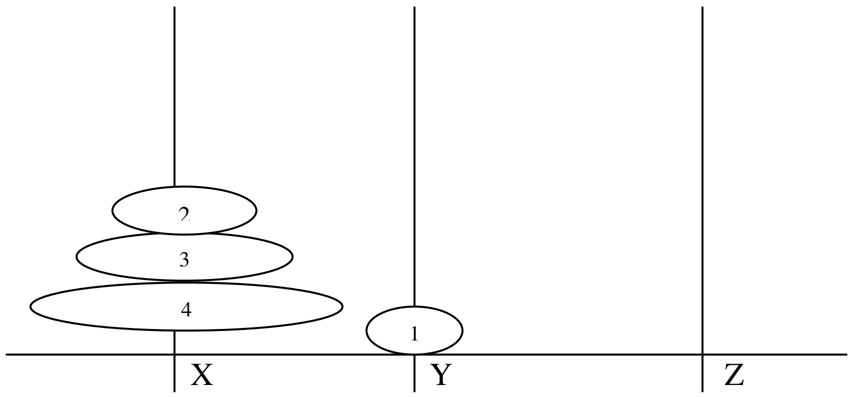
Q166. Write an algorithm for finding solution to the Towers of Hanoi problem. Explain the working of your algorithm (with 4 disks) with diagrams. **(7)**

Ans.

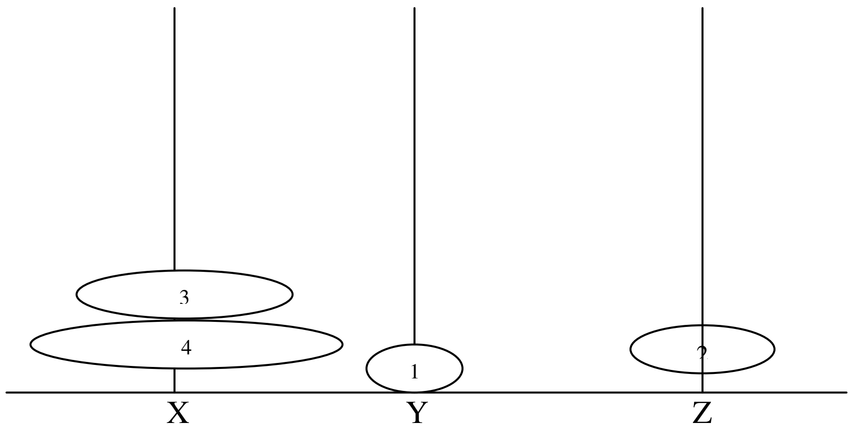
```
 Algorithm for Tower of Hanoi
TOWER ( N, BEG, AUX, END)
This procedure gives a recruitment solution to the
tower of Hanoi problem for N discuss:-
1.  If N = 1 then
(a) Write BEG → END
(b) Return
[end of if structure]
2.  {move N-1 disks  from peg BEG to peg AUX]
call TOWER (N-1, BEG, END, AUX)
3.  Write : BEG →END
4.  [Move N-1 disks from peg AUX to peg END]
Call Tower (N-1, AUX, BEG, END)
5.  Return
```



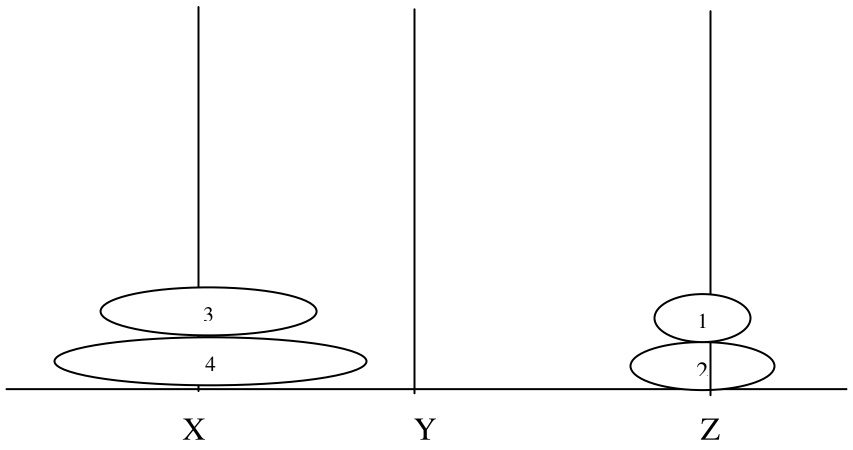
In the beginning let X= BEG Y=AUX and Z= END



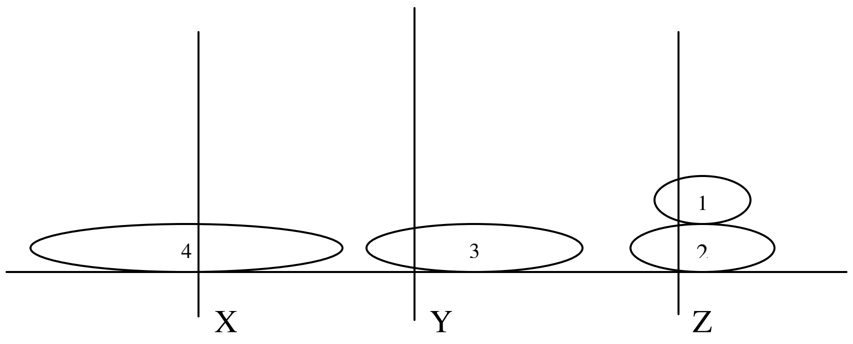
Move from X to Y



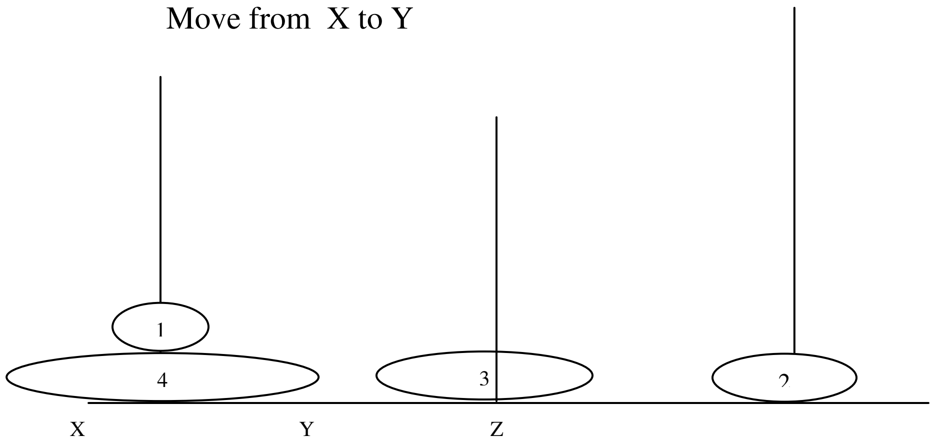
Move from X to Z



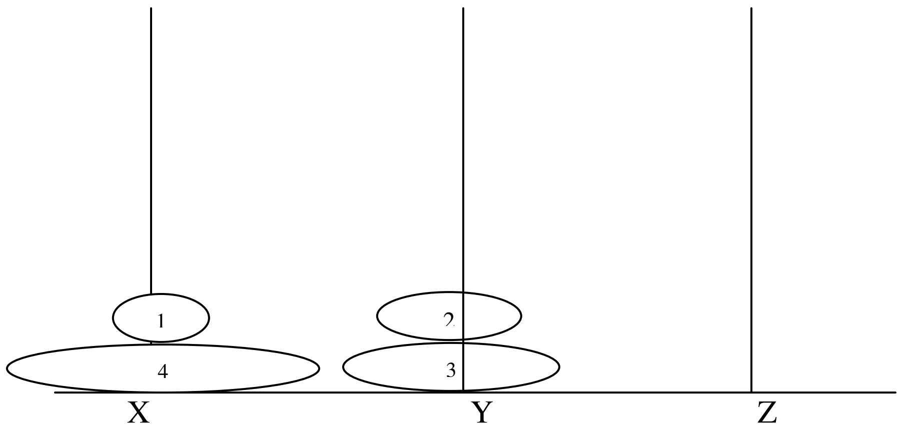
Move from Y to Z



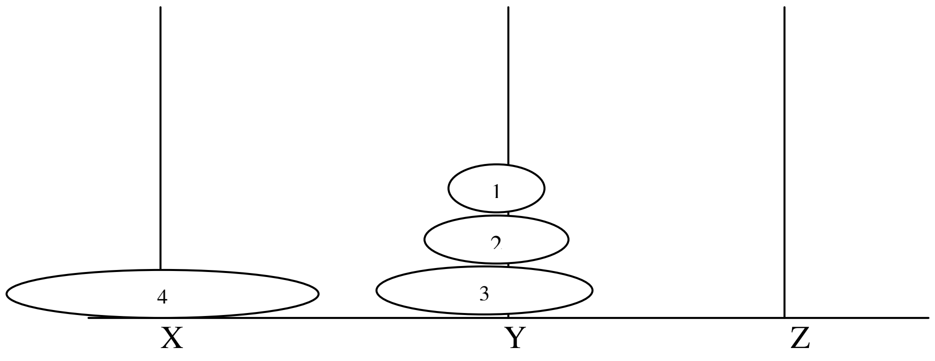
Move from X to Y



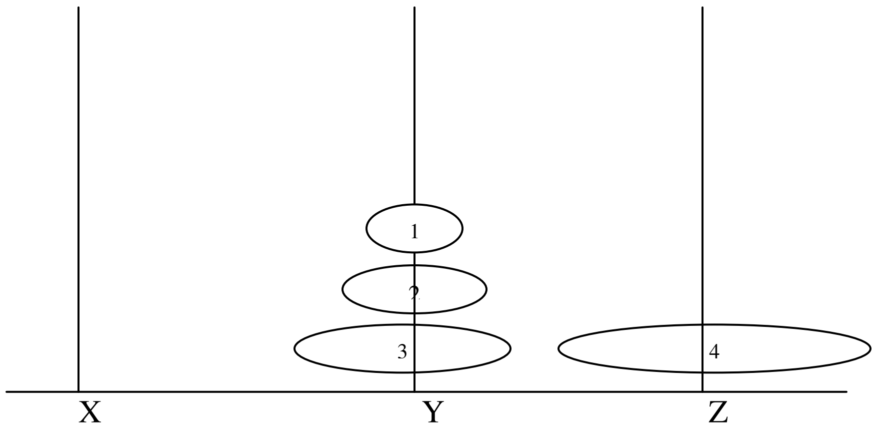
Move from Z to X



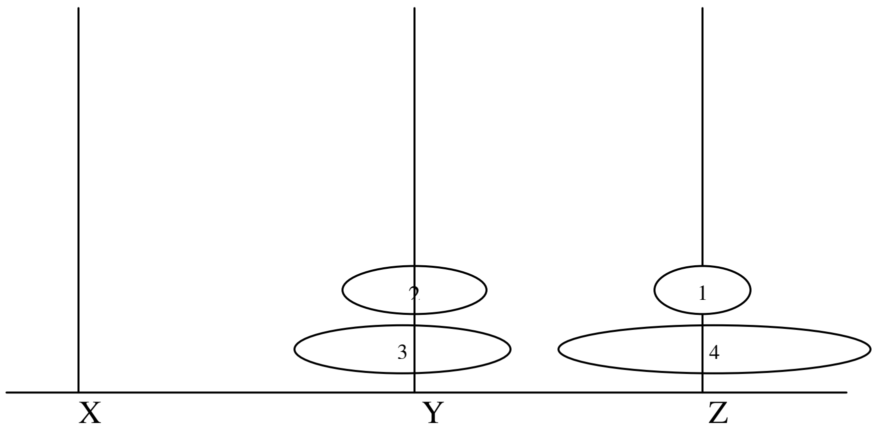
Move from Z to Y



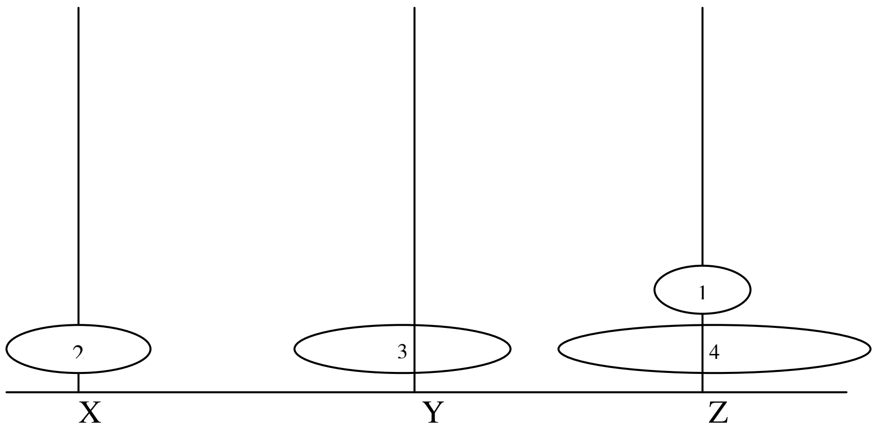
Move from X to Y



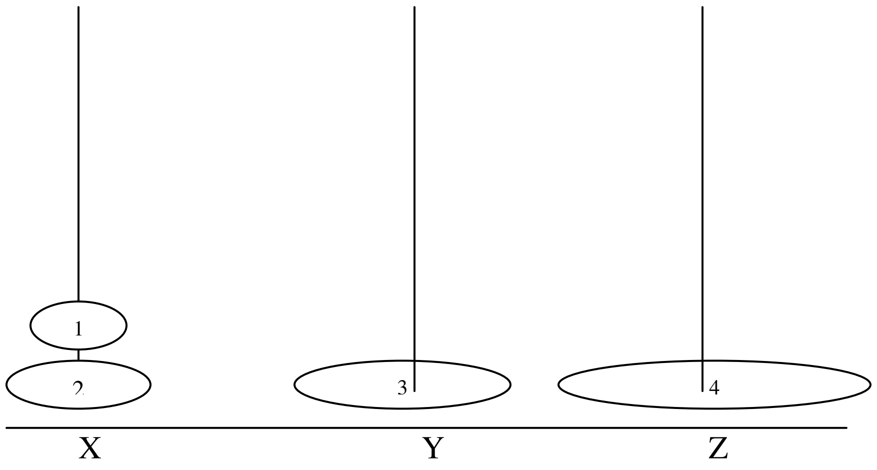
Move from X to Z



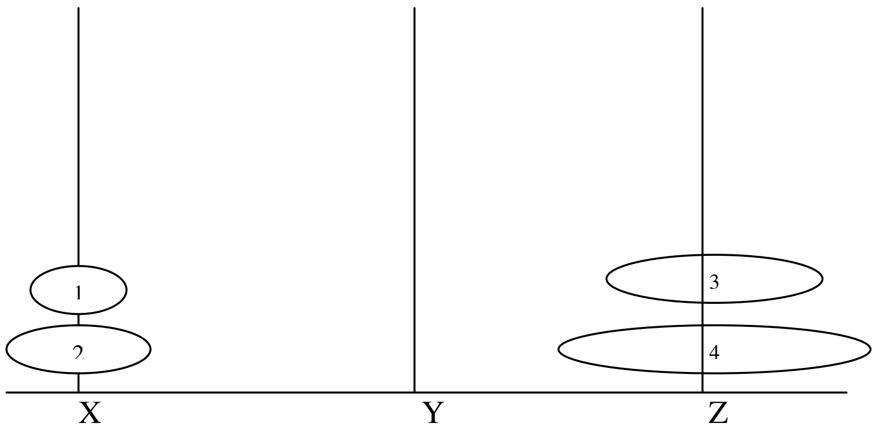
Move from Y to Z



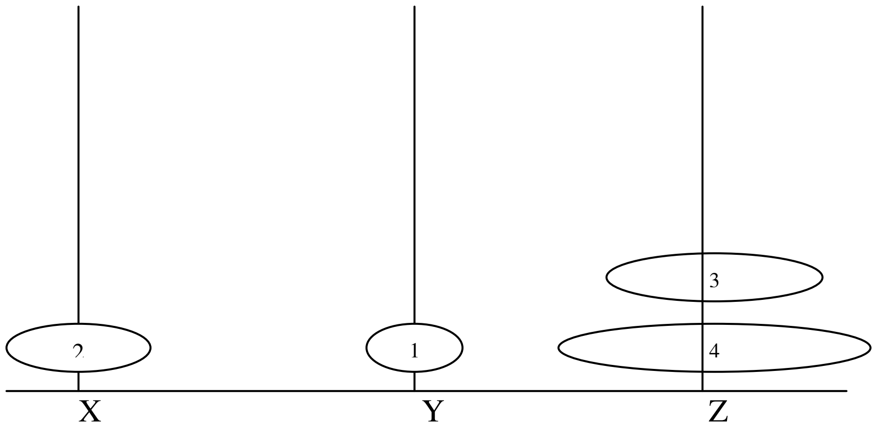
Move from Y to X



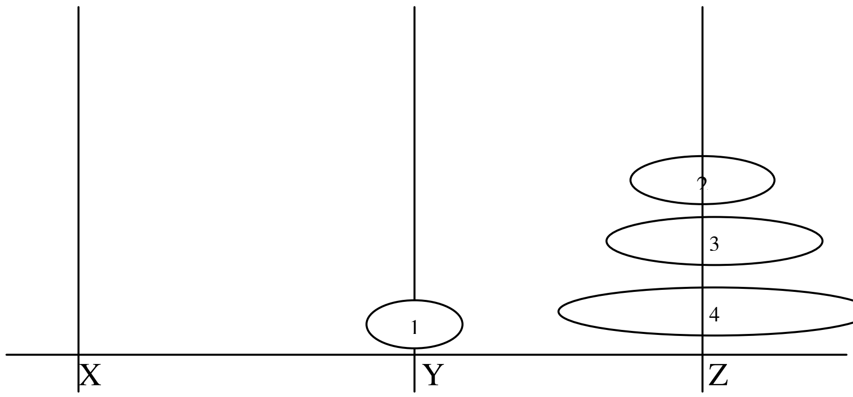
Move from Z to X



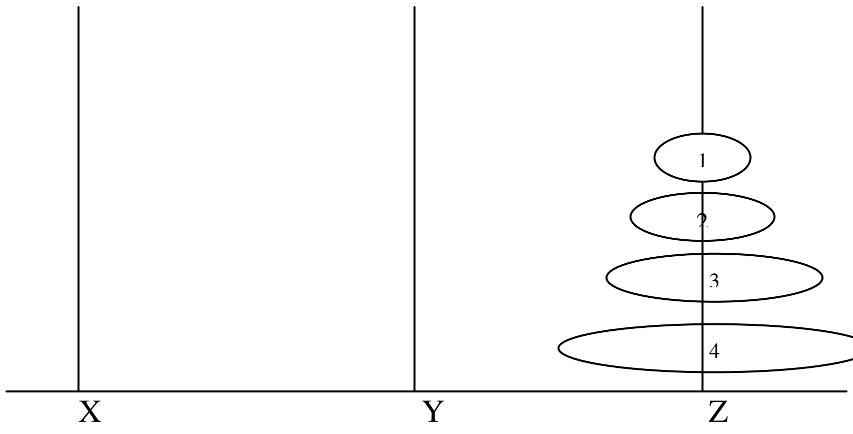
Move from Y to Z



Move from X to Y



Move from X to Z



Move from Y to Z

Q167. Suppose we have divided n elements in to m sorted lists. Explain how to produce a single sorted list of all n elements in time O (n log m)? **(7)**

Ans.

The list can be produced using Merge sort. Following is the procedure for it. Suppose A is a sorted list with r elements and B is a sorted list with s elements. The operation that combines the elements of A and B into a single sorted list C with n = r +s elements is called merging.

Procedure 1

```
MERGING(A, R, B, S, C)
```

Let A and B be sorted arrays with R and S elements, respectively. This algorithm merges A and B into an array C with N= R + S elements.

```
1. [Initialize.] Set NA := 1, NB := 1 and PTR := 1.
2. [Compare.] Repeat while NA <= R and NB <= S :
             If A[NA] < B[NB], then ;
     (a)  [Assign element from A to C.] Set C[PTR] :=
     A[NA].
     (b)  [Update pointers.] Set PTR :=  PTR + 1 and
     NA := NA + 1.
     Else:
```

```
    (a)    [Assign element from B to C.] Set C[PTR]
    := B[NB].
    (b)    [Update pointers.] Set PTR :=  PTR + 1 and
    NB := NB + 1.
         [End of If structure.]
               [End of loop.]
 3. [Assign remaining elements to C.]
    If NA > R, then:
          Repeat for K = 0, 1, 2,…,S-NB:
        Set C[PTR + K] := B[NB + K].
           [End of loop.]
    Else:
           Repeat for K = 0, 1, 2, …, R - NA:
        Set C[PTR + K] := A[NA + K].
            [End of loop.]
    [End of If structure.]
 4. Exit.
```

Procedure 2:

```
MERGE(A, R, LBA, S, LBB, C, LBC)
         This procedure merges the sorted arrays A
and B into the array C.
1. Set NA := LBA, NB := LBB, PTR := LBC, UBA:= LBA
    + R - 1, UBB :=      LBB + S - 1.
2. call merging (A,UBA,B,UBB,C)
3. Return.
```

Procedure 3:

MERGEPASS(A, N, L, B)

The N-element array A is composed of sorted subarrays where each subarray has L elements except possibly the last subarray, which may have fewer than L elements. The procedure merges the pairs of subarrays of A and assigns them to the array B.

```
1.   Set Q := INT(N/(2*L)), S:= 2*L*Q and R := N
      - S.
2.  [Use procedure2  to merge the Q pairs of
    subarrays.]
    Repeat for J = 1, 2, . . ., Q:
    (a) Set LB := 1 + (2*J - 2) * L. [Finds lower
    bound of first array.]
    (b) Call MERGE(A, L, LB, A, L, LB + L, B,
    LB).
       [End of loop.]
3.   [Only one subarray left ?]
    If R □ L, then:
           Repeat for J = 1, 2, . . ., R:
       Set B(S + J) := A(S+J).
           [End of loop.]
    Else :
    CALL MERGE(A, L, S + 1, A, R, L + S + 1, B, S
    + 1).
```

```
          [End of If structure.]
     4.   Return.
     Procedure 4 MERGESORT( A, N)
```
This algorithm sorts the Nth element array A using an auxiliary array B.
```
     1.   Set L:=1 . [ Initiliazes the number of
          elements in the subarrays.]
     2.   Repeat Steps 3 to 6 while L<N:
     3.           Call MERGEPASS(A,N,L,B)
     4.           Call MERGEPASS(B,N,2*L,A).
     5.            Set L:= 4*L.
          [End of Step 2 loop].
     6.   Exit.
```

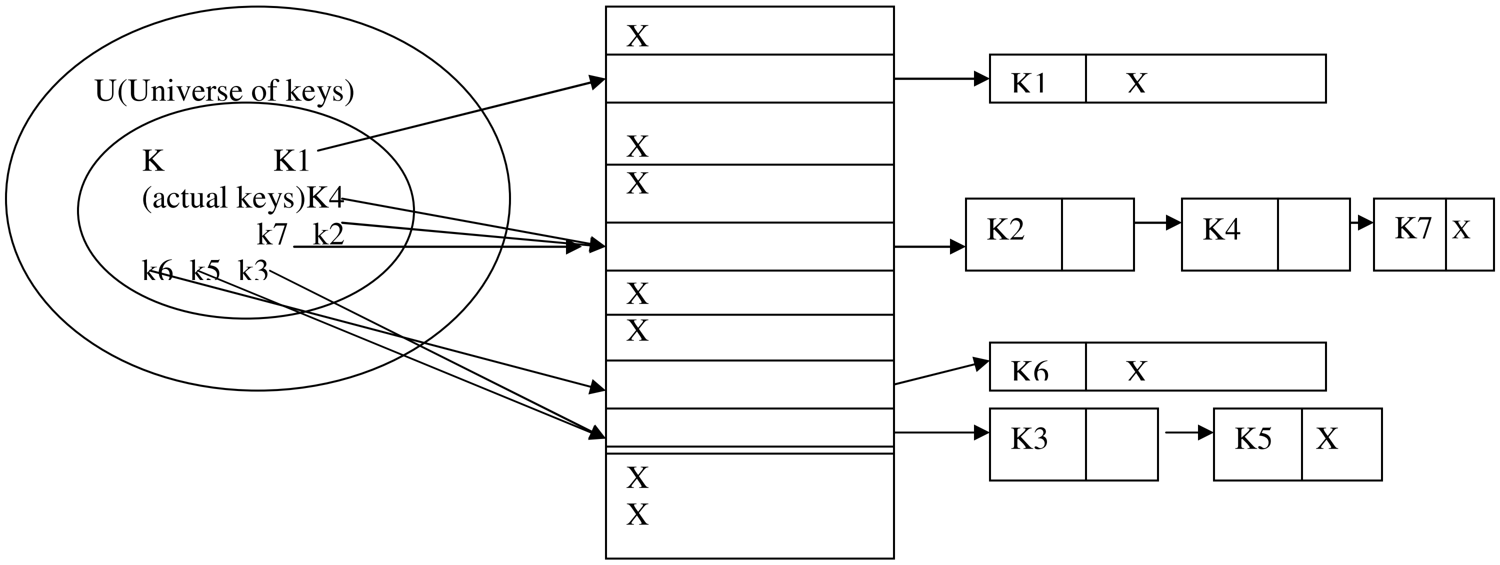
Q168. Define hashing. Describe any two commonly used hash functions. Describe one method of collision resolution. **(7)**

Ans.
A hash table is a data structure in which the location of a data item is determined directly as a function of data item itself rather than by a sequence of comparison. Under ideal condition, the time required to locate a data item in a hash table is 0(1) ie. it is constant and documents not depend on the number of data items stored.
When the set of K of keys stored is much smaller then the universe U of all possible keys, a hash table require much less storage spare than a direct address table.
Hash Function
A hash function h is simply a mathematical formula that manipulates the key in some form to compute the index for this key in the hash table. Eg a hash function can divide the key by some number, usually the size of the hash table and return remainder as the index for the key. In general, we say that a hash function h maps the universe U of keys into the slots of a hash table T[0….n-1]. This process of mapping keys to appropriate slots in a hash table is known as **hashing.**
Some of methods of hashing are:-
a) Division method
b) Multiplication method
c) Midsquare method
d) Folding method
(a) Division Method:- In this method, key K to be mapped into one of the m states in the hash table is divided by m state in the hash table is divided by m and the remainder of this division taken as index into the hash table. That is the hash function is
h(k) = k mod m
where mod is the modules operations.
(b) Multiplication Method: The multiplication method operates in 2 steps. In the 1st step the key value K is multiplied by a constant A in the range O<A<1 and the fractional part of the value obtained above is multiplied by m and the floor of the result is taken as the hash values. That is the hash function is
h(k) = [m (K A mod 1)]

Methods of Collision Resolution

1) Collision Resolution by separate chaining
2) Collision Resolution by open addressing

1) In this scheme all elements whose key hash to same hash table slot are put in a linked list. Thus the slot in the hash table contains a pointer to the head of the linked list of all the elements that hash to value i. If there is no such element that hash to value I , the slot I contains NULL value (pictures as X)



Q169. A Binary tree has 9 nodes. The inorder and preorder traversals of the tree yields the following sequence of nodes:

Inorder :	E	A	C	K	F	H	D	B	G
Preorder:	F	A	E	K	C	D	H	G	B

Draw the tree. Explain your algorithm. **(7)**

Ans.

Inorder:	E	A	C	K	F	H	D	B	G
Preorder:	F	A	E	K	C	D	H	G	B

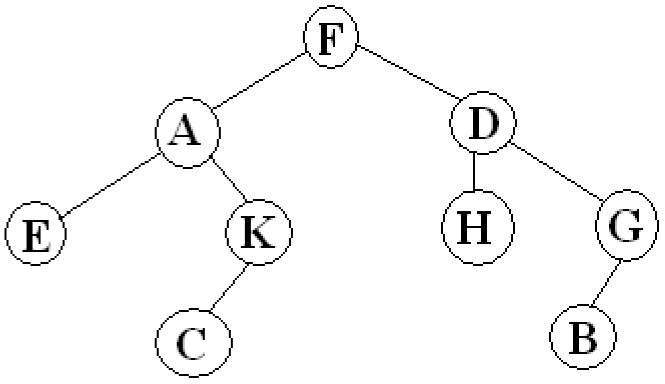
The tree T is drawn from its root downward as follows.

a) The root T is obtained by choosing the first node in its preorder. Thus F is the root of T.

b) The left child of the node F is obtained as follows. First use the inorder of T to find the nodes the in the left subtree T_1 of F. Thus T_1 consists of the nodes E, A, C and K. Then the left child of F is obtained by choosing the first node in the preorder of T_1 (which appears in the preorder of T). Thus A is the left son of F.

c) Similarly, the right subtree T_2 of F consists of the nodes H, D, B and G, and D is the root of T_2, that is, D is the right child of F.

Repeating the above process with each new node, we finally obtain the required tree.



Q170. What are B-trees? Construct a B-Tree of order 3 for the following set of Input data:

69, 19, 43, 16, 25, 40, 132, 100, 145, 7, 15, 18 **(7)**

Ans.

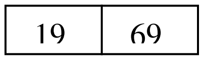
B- Tree

A B-tree of order m is an m-way true in which

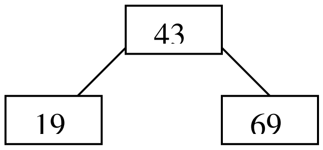
1) All leaves are on the same level
2) All internal nodes except the root have at most m-1(non-empty) children and at least [m/2] (non empty children)
3) The number of keys in each internal node is one less than the number of its children and these keys partition the keys in the children in the fashion of a search tree.
4) The root has at most m children, but may have as few as 2 if it is not a leaf, or none if the tree consists of the root alone.

69 19 43 16 25 40 132 100 145 7 15 18

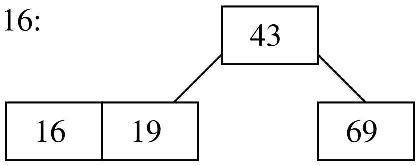
69,19 :



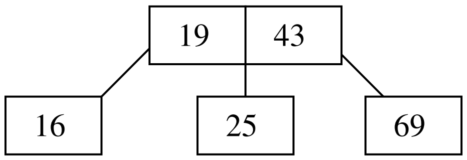
43:



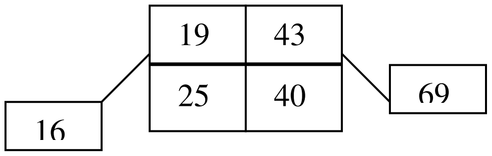
16:



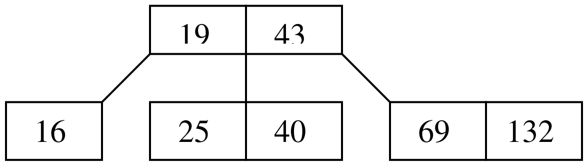
25:



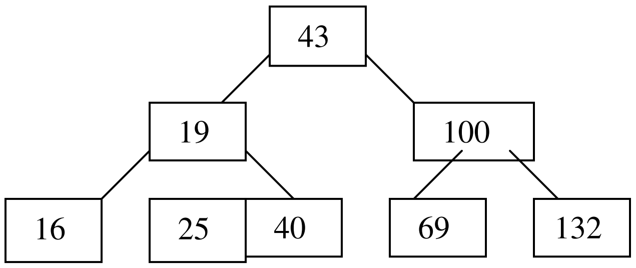
40:



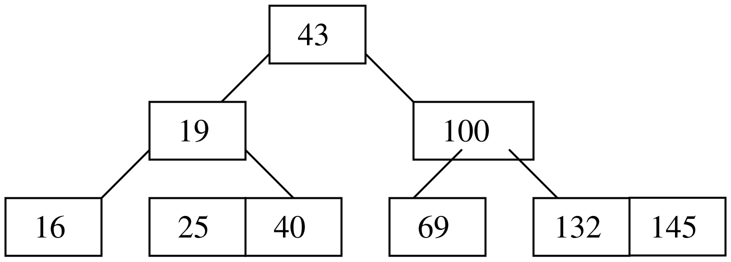
132:



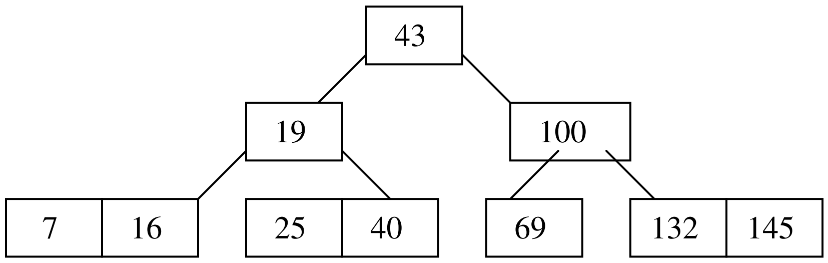
100:



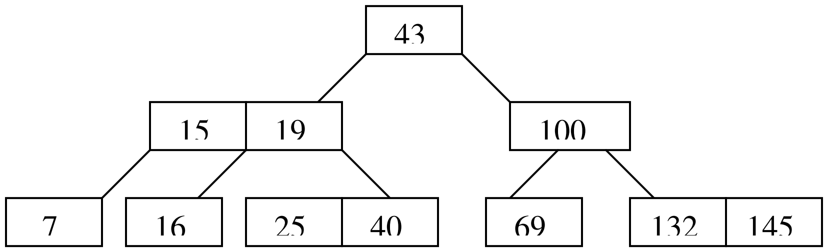
145:



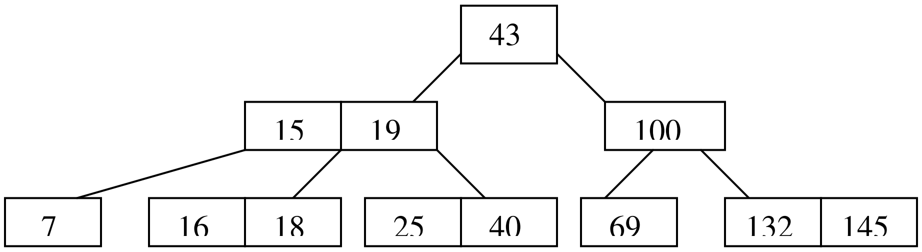
7:



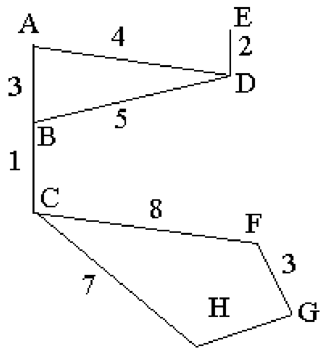
15:



18:



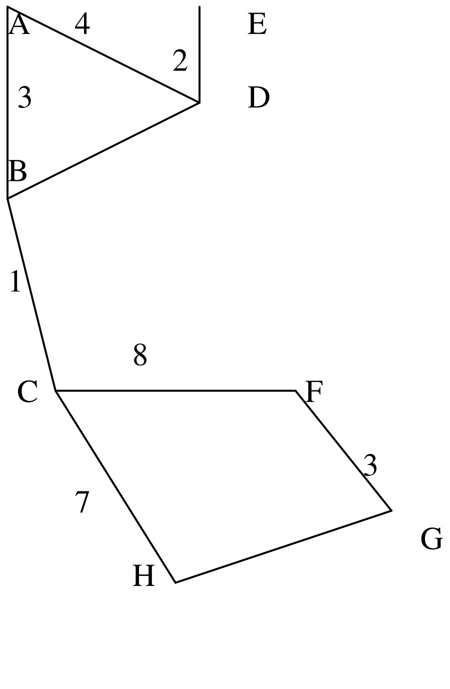
Q171. What is the difference between Prims algorithm and Kruskals algorithm for finding the minimum-spanning tree of a graph? Execute both Prim's and Kruskal's algorithms on the following graph. **(8)**



Ans.

Difference Between Prism's and Kruskal's Algorithm

In Kruskal's algorithm, the set A is a forest. The safe edge added to A is always a least-weight edge in the paragraph that connects two distinct components. In Prim's algorithm, the set A forms a single tree. The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

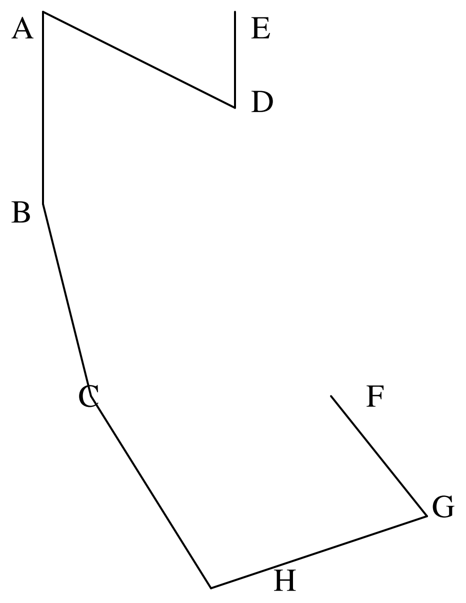


Let us assume distance between H and G to be 0 since distance between them is not given.

Using Kruskal's Method

Edge(u,v)	Weight	Included
(A,D)	4	√
(A,B)	3	√
(D,E)	2	√
(B,C)	1	√
(C,F)	8	X
(C,H)	7	√
(H,G)	0	√
(F,H)	3	√
(B,D)	5	X

The minimum spanning tree is:-



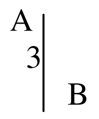
min cost = 2+4+3+1+7+0+3 = 20 units.

Using Prim's Algorithm

Step 1, 2, 3 are

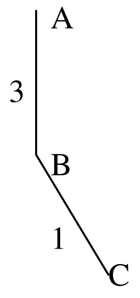
	A	B	C	D	E	F	G	H
A	_	**<u>3</u>**	∝	4	∝	∝	∝	∝
B	3	_	1	5	∝	∝	∝	∝
C	∝	1	_	∝	∝	8	∝	7
D	4	5	∝	_	2	∝	∝	∝
E	∝	∝	∝	2	_	∝	∝	∝
F	∝	∝	8	∝	∝	_	3	∝
G	∝	∝	∝	∝	∝	3	_	0
H	∝	∝	7	∝	∝	∝	0	_

Step 4
Start with 1 and pick the smallest entry in row 1 which is (A,B)



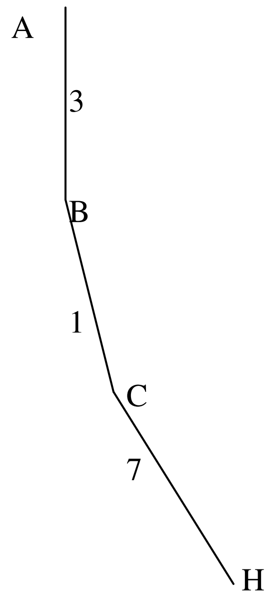
	A	B	C	D	E	F	G	H
A	_	3	∝	4	∝	∝	∝	∝
B	3	_	**<u>1</u>**	5	∝	∝	∝	∝

Now select vertex C and edge (B, C)



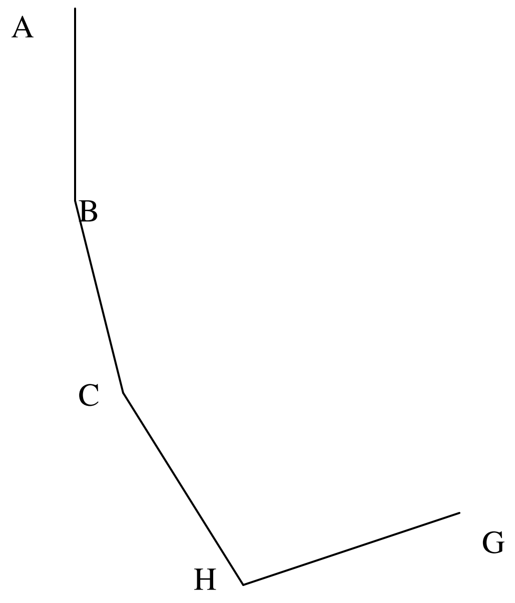
	A	B	C	D	E	F	G	H
A	_	3	∝	4	∝	∝	∝	∝
B	3	_	1	5	∝	∝	∝	∝
C	∝	1	_	∝	∝	8	∝	**<u>7</u>**

Now Select (C, H) edge



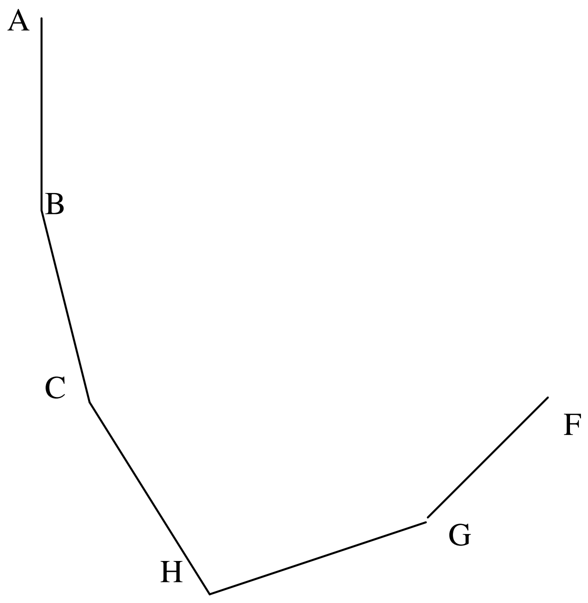
	A	B	C	D	E	F	G	H
A	_	3	∝	4	∝	∝	∝	∝
B	3	_	1	5	∝	∝	∝	∝
C	∝	1	_	∝	∝	8	∝	7
H	∝	∝	7	∝	∝	∝	**0**	_

Now select edge (H, G)



Now select the edge (G,F)

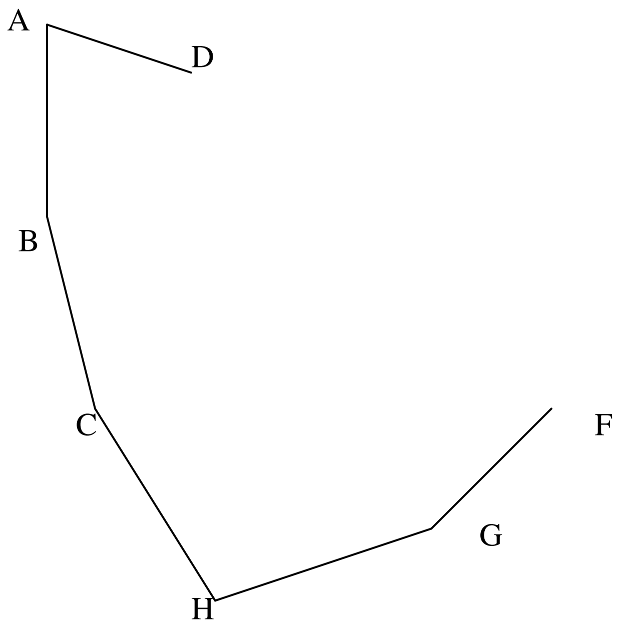
	A	B	C	D	E	F	G	H
A	_	3	∝	4	∝	∝	∝	∝
B	3	_	1	5	∝	∝	∝	∝
C	∝	1	_	∝	∝	8	∝	7
H	∝	∝	7	∝	∝	∝	0	_
G	∝	∝	∝	∝	∝	**3**	_	0



	A	B	C	D	E	F	G	H
A	_	3	∝	**4**	∝	∝	∝	∝
B	3	_	1	5	∝	∝	∝	∝
C	∝	1	_	∝	∝	8	∝	7
H	∝	∝	7	∝	∝	∝	0	_
G	∝	∝	∝	∝	∝	3	_	0
F	∝	∝	8	∝	∝	_	3	∝

We cannot consider edge (F, C) as it will make a cycle. so we consider the edge AD.

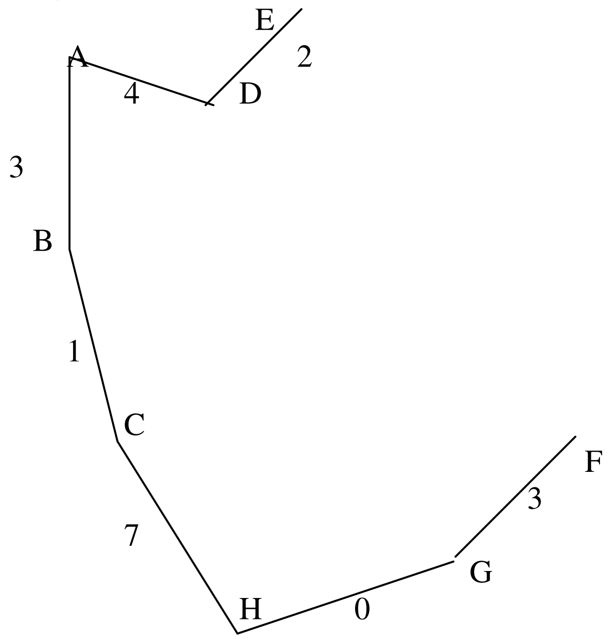
	A	B	C	D	E	F	G	H
A	_	3	∝	4	∝	∝	∝	∝
B	3	_	1	5	∝	∝	∝	∝
C	∝	1	_	∝	∝	8	∝	7
H	∝	∝	7	∝	∝	∝	0	_
G	∝	∝	∝	∝	∝	3	_	0
F	∝	∝	8	∝	∝	_	3	∝
D	4	5	∝	_	**2**	∝	∝	∝



Now select the edge (D, E)

	A	B	C	D	E	F	G	H
A	_	3	∝	4	∝	∝	∝	∝
B	3	_	1	5	∝	∝	∝	∝
C	∝	1	_	∝	∝	8	∝	7
H	∝	∝	7	∝	∝	∝	0	_
G	∝	∝	∝	∝	∝	3	_	0
F	∝	∝	8	∝	∝	_	3	∝
D	4	5	∝	_	**2**	∝	∝	∝
E	∝	∝	∝	2	_	∝	∝	∝

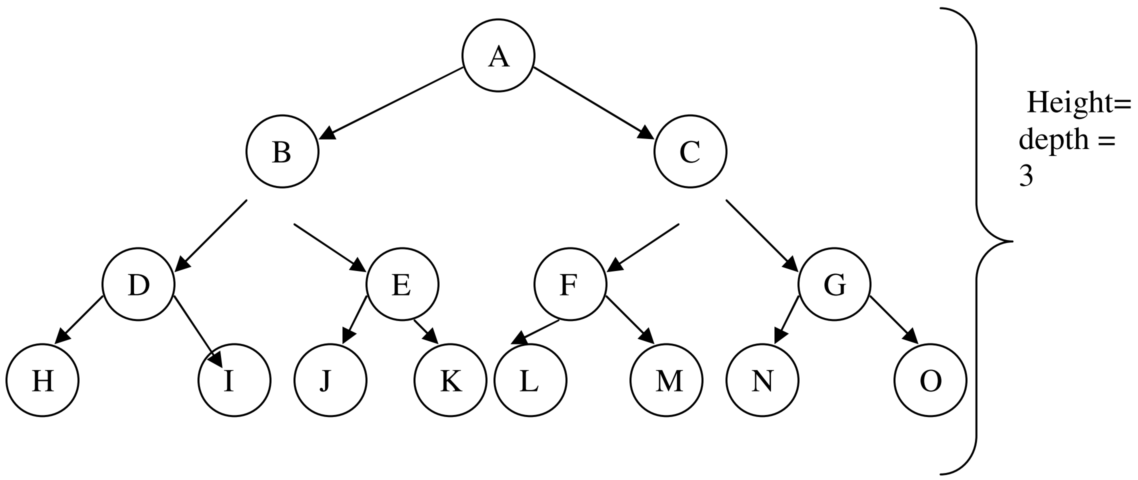
Our spanning tree is now



Min cost = 2+4+3+1+7+0+3= 20 units

Q172. Show the result of running BFS on a complete Binary Tree of depth 3. Show the status of the data-structure used at each stage. **(6)**

Ans.
Complete Binary Tree of Depth 3



BFS uses a queue data structure.

A	

Delete A from queue and add to the queue nodes adjacent to it i.e. B, C

B	C	

Delete node B from queue and add nodes adjacent to B i.e Q

C	D	E	

Delete node C from queue and add nodes adjacent to C i.e F,G

D	E	F	G

Delete node D from queue and add nodes adjacent to D i.e H, I

E	F	G	H	I

Delete node E from queue and add nodes adjacent to E i.e J, K

F	G	H	I	J	K

Delete node F from queue and add nodes adjacent to F i.e L, M

G	H	I	J	K	L	M

Delete node G from queue and add nodes adjacent to G i.e N, O

H	I	J	K	L	M	N	O

Delete node H from queue. Since no nodes are adjacent to H no more nodes will be added to the queue.

H	I	J	K	L	M	N	O

Therefore the nodes visited till now are
A B C D E F G H
The queue after deletion of H is

I	J	K	L	M	N	O

The queue after deletion of I is

J	K	L	M	N	O

The queue after deletion of J is

K L	M	N	O	

The queue after deletion of K is

L	M	N	O

The queue after deletion of L is

M	N	O

The queue after deletion of M is

N	O

The queue after deletion of N is

O

The queue after deletion of O is

Hence the BFS traversal of tree yields the result..
A B C D E F G H I J K L M N O

Q173. Write an algorithm to test whether a Binary Tree is a Binary Search Tree. **(4)**

Ans.

The algorithm to test whether a Binary tree is as Binary Search tree is as follows:

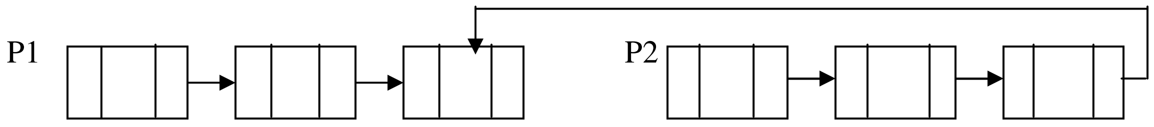
```
     bstree(*tree)
     {
     while((tree->left !=null)&& (tree->right !=null))
       {
          if(tree->left < tree->root)
             bstree(tree->left);
          else
             return(1);
          if(tree->right > tree->root)
             bstree(tree->right);
          else
       return(1);
       }
       return(0);
       }
```

Q174. Define a linked list with a loop as a linked list in which the tail element points to one of the list's elements and not to NULL. Assume that you are given a linked list *L*, and two pointers *P1*, *P2* to the head. Write an algorithm that decides whether the list has a loop without modifying the original list. The algorithm should run in time *O(n)* and additional memory *O(1)*, where *n* is the number of elements in the list. **(10)**

Ans

```
 struct node
     {
      int info;
      int node_count;
      struct node *next;
      }*P1, *P2;
```

Let the given linked list L at some stage is:



Every node in list contains its position from head, i.e. 1,2,3…. In its node_count field. The following algorithm tests whether the list has a loop in it or not.

```
void test_loop()
{
  struct node *temp;
  P2 = P1 -> next;
  if(P1 == P2) {
   printf("There exists a loop in list");
   return;  }
  temp = P2->next;
  while(P2->node_count < temp->node_count) {
   P2 = P2->next;
   temp = P2->next;  }
  printf("Loop  is  from  %d  to  %d", P2->node_count,
  temp->node_count);
}
```

Q175. Execute quick algorithm on the following data till two key values are placed in their position 12,34,45,15,4,11,7,8,5,14,35,89,43,21. **(8)**

Ans:

The given data is:- 12,34,45,15,4,11,7,8,5,14,35,89,43,21

Pass 1:-

```
 d                                                 u
 12   34   45  15  4   11  7   8   5   14  35  89  43  21
      d                            u
 12   34   45  15  4   11  7   8   5   14  35  89  43  21
swap x[down] and x[up]
           d                       u
 12   5    45  15  4   11  7   8   34  14  35  89  43  21
           d                   u
 12   5    45  15  4   11  7   8   34  14  35  89  43  21
swap x[down] and x[up]
           d                   u
 12   5    8   15  4   11  7   45  34  14  35  89  43  21
               d           u
 12   5    8   15  4   11  7   45  34  14  35  89  43  21
swap x[down] and x[up]
                           du
 12   5    8   7   4   11  15  45  34  14  35  89  43  21
                       u   d
 12   5    8   7   4   11  15  45  34  14  35  89  43  21
swap x[up] and a
                   u   d
(11 5   8   7   4)  12  (15 45  34  14  35  89  43    21)
```

Pass 2:-

```
 d               u
(11 5   8   7   4)  12  (15 45  34  14  35  89  43    21)
 d               u
(11 5   8   7   4)  12  (15 45  34  14  35  89  43    21)
swap a and x[up]
        d   u
 (4 5   8   7   11) 12  (15 45  34  14  35  89  43    21)
swap a and x[up]
                         d                             u
4   5   7   8   11  12  (15 45  34  14  35  89  43    21)
```

Thus , two key values i.e. , 11 and 12 are now placed in their right positions.

Similarly sort 2nd array also, such that we get after few steps as a sorted list
4 5 7 8 11 12 14 15 21 34 35 43 45 89

Q 176 Sort the following array of elements using quick sort *{3 1 4 1 5 9 2 6 5 3 5 8}* **(8)**

Ans

The given data is :-
3 1 4 1 5 9 2 6 5 3 5 8

Pass 1:-

```
a=x[lb]=3
d                                           u
3   1   4   1   5   9   2   6   5   3   5   8
        d                           u
3   1   4   1   5   9   2   6   5   3   5   8
swap x[down] and x[up]
        d                           u
3   1   3   1   5   9   2   6   5   4   5   8
                d       u
3   1   3   1   5   9   2   6   5   4   5   8
swap x[down] and x[up]
                d       u
3   1   3   1   2   9   5   6   5   4   5   8
                u   d
3   1   3   1   2   9   5   6   5   4   5   8
swap a and x[up]
 (2   1   3   1)  3   (9   5   6   5   4   5   8)
```

Pass 2:-
(1 1) 2 (3) 3 (9 5 6 5 4 5 8)

Pass 3:-
(1) 1 2 (3) 3 (9 5 6 5 4 5 8)

Pass 4:-
1 1 2 (3) 3 (5 6 5 4 5 8) 9

Pass 5:-
1 1 2 3 3 (5 6 5 4 5 8) 9

Pass 6:-
1 1 2 3 3 (4 5 5) 5 (6 8) 9

Pass 7:-
1 1 2 3 3 4 (5 5) 5 (6 8) 9

Pass 8:-

```
1 1   2   3   3   4   (5)  5   5   (6   8)  9
```

Pass 9:-

```
1 1   2   3   3   4   5    5   5   (6   8)  9
```

Pass 10:-

```
1 1   2   3   3   4   5    5   5   (6   8)  9
```

Pass 11:-

```
1 1   2   3   3   4   5    5   5   6   (8)  9
```

Pass 12:-

```
1 1   2   3   3   4   5    5   5   6    8   9.
```

Q.177 Execute your algorithm for two passes using the following list as input:
66, 33, 40, 20, 50, 88, 60, 11, 77, 30, 45, 65
Describe the behaviour of Quick sort when the input is already sorted. **(10)**

Ans:
The given data is as follows:-66, 33, 40, 20, 50, 88, 60, 11, 77, 30, 45, 65

Pass 1:-

```
66   33  40  20  50  88  60  11  77  30  45  65
   a=x[lb]=66
d→                                           u
66   33  40  20  50  88  60  11  77  30  45  65
             d                       u
66   33  40  20  50  88  60  11  77  30  45  65
                     d                       u
66   33  40  20  50  88  60  11  77  30  45  65
swap x[down] and x[up]
                     d                       u
66   33  40  20  50  65  60  11  77  30  45  88
                                 d       u
66   33  40  20  50  65  60  11  77  30  45  88
swap x[down] and x[up]           d       u
66   33  40  20  50  65  60  11  45  30  77  88
u                                    d
66   33  40  20  50  65  60  11  45  30  77  88
swap a and x[up]
 u                               d
(30  33  40  20  50  65  60  11  45) 66  (77    88)
End of pass 1.
```

Pass 2:-

Taking the left partition:-

a=x[lb]=30

```
(30  33   40   20   50   65   60   11   45)  66   (77     88)
d                                  u
(30  33   40   20   50   65   60   11   45)  66   (77     88)
     d                             u
 (30 33   40   20   50   65   60   11   45)  66   (77     88)

     d                             u
 (30 33   40   20   50   65   60   11   45)  66   (77     88)
swap x[down] and x[up];
     d                             u
(30  11   40   20   50   65   60   33   45)  66   (77     88)
          d    u
(30  11   40   20   50   65   60   33   45)  66   (77     88)
swap x[down] and x[up];
          d    u
(30  11   20   40   50   65   60   33   45)  66   (77     88)
 u        d
(30  11   20   40   50   65   60   33   45)  66   (77     88)
swap a and x[up];
(20  11)  30  (40   50   65   60   33   45)  66   (77     88)
End of pass 2.
```

When the array will be already sorted:-In the beginning itself a condition is set that

```
        if(lower_bound >= upperbound)

        return;
```

therefore, once the partition function is executed, the value of down shall increment only once and the value up shall decrement till it becomes zero as all the elements in its way shall be greater than pivot(array being sorted in ascending order.). Thus it will return the value in j= up as zero.
Now, again recursively when the program is called and upperbound is passed the value as j-1 which is equal to -1. Thus the condition now checked will terminate the program without going inside the partition function.