UNIT - II.

Basic Terminology of Tree.

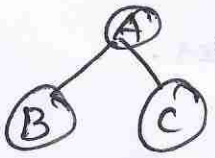
1) Root Node
2) Node "
3) Edge
4) Parent Node.
5) Child Node.
6) Leaf Node.
7) Siblings
8) Internal Nodes
9) Degree
10) Level of Tree
11) Height of Tree.
12) Depth of Node.
13) Path
14) Subtree.

UNIT - II

TREE TERMINOLOGY

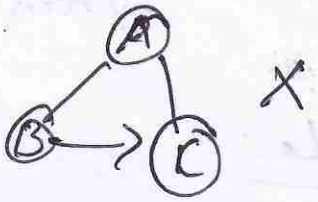
Tree

Data organized in Hierarchical manner without closed region is called as tree.

Ex. 

element → Node.

If B node and C node have connection. It is closed. So it is not a tree.

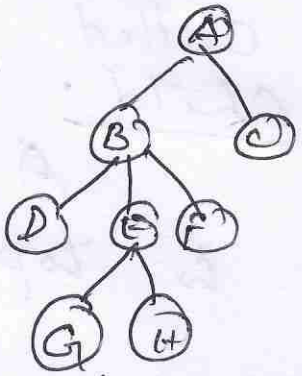


X

Basic Terminology of a Tree.

Node: Element of a tree is Called as Node.

Ex. A, B, C, D, E, F, G, H.



Root Node: A starting Node or First Node is called as Root Node.

Ex. A.

Tree will have only one root Node.

Edge: A Link or Connection between any two node is called as an Edge.

If the tree have N nodes then the tree have N-1 edges.

N = 8 Nodes

E = 7 Edges.

Parent:

A node which having branches is called as parent Node. (from Top to bottom)

Ex. A, B E

Leaf Node:

A node without branches is called as child node.

Ex. C, D, F, G, H.

Child:

A node with Edge from bottom to top or branches of parent.

Ex. B, C, D, E, F, G, H. — Child.

Siblings:

A child node of a same Parent is called as Siblings Node

B, C → Siblings.

D, E F → Siblings

G, H → Sibling.

Internal Nodes:

All nodes other than leaf node is called as internal Nodes.

Node with child Nodes.

A, B, E → Internal Nodes.

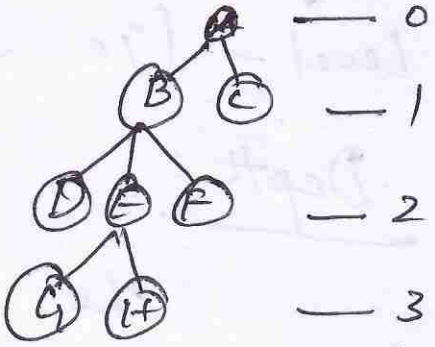
Degree:

Number of child Nodes represents degree of a Node.

degree of A = 2

B = 3

C = 2.



A maximum degree among all nodes is called → degree of tree.

Degree of Tree = 3.

Level of Tree

Every step / hierarchy in a tree is a level of tree.

Level starts from '0'

For every step / hierarchy level will be incremented by (1).

Level of a tree = 3.

Level of root Node = 0.

Height of Node

A longest path from a leaf node to that node is Height of node.

Height (B) → 2

Height (A) → 3

Height → (To Specify for Particular Node)

Level – (To Specify for tree)

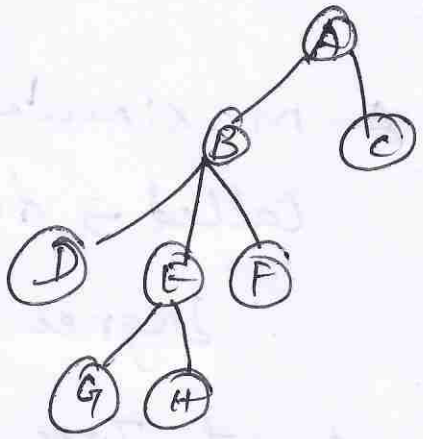
Depth:

The Longest path from the Root node that node.

Depth (E) – 2

Depth (G) – 3.

Depth (B) – 1



Path:

A Sequence of nodes from Root to Leaf (or) Source to destination.

Path (A to G) → A → B → E → G.

Subtree

Node with children forms Subtree.

Binary Tree

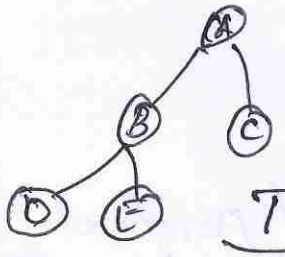
Every Node in a tree should have atmost 2 children is called binary tree.

A tree have.

At most 2 — 0 node

or (1) node

2 nodes.

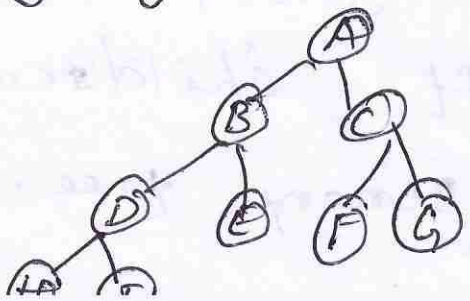


TYPes of Binary Tree.

1) Fully Binary Tree / Strictly Binary Tree.
2) Almost Complete Binary Tree / Incomplete Binary Tree.
3) Complete Binary Tree / Perfect Binary Tree.
4) Left Skewed Binary Tree
5) Right Skewed Binary Tree.

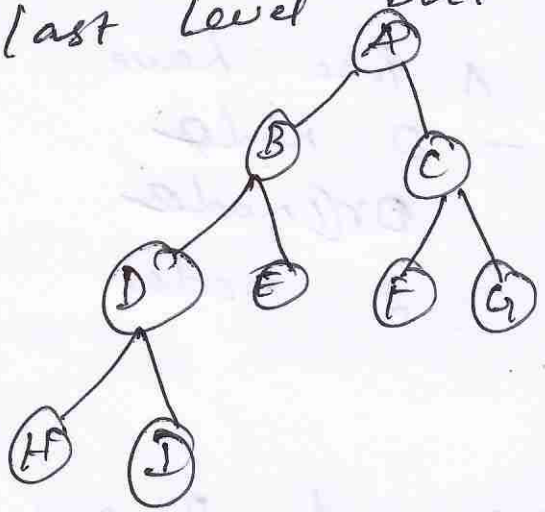
Fully Binary Tree.

Every Node must have two Childen except the leaf node is called fully binary Tree.



Incomplete BT / Almost Complete BT

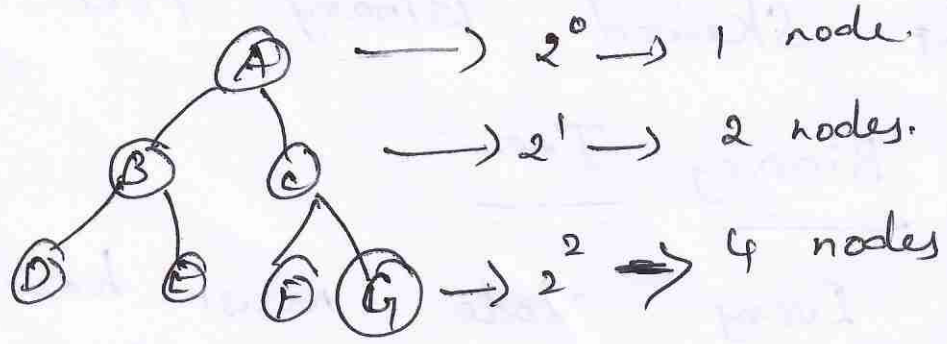
Every Node must have two Childreen in all levels except in last level but filled from left to Right.



Complete Binary Tree / Perfect Binary Tree.

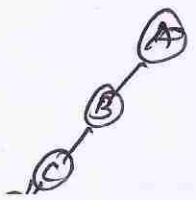
Every Node must have 2 Children in all the Level.

Each Level there must be 2^L nodes → L → Level.



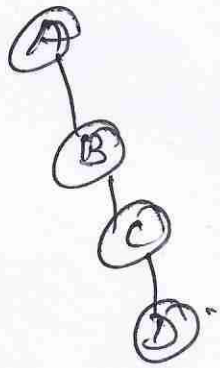
Left Skewed Binary Tree

A tree Should Contains only the left children is called Left Skewed Binary tree.

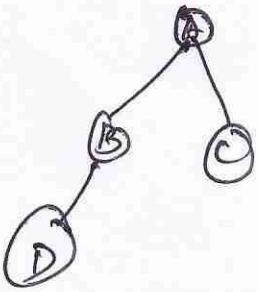


Right Skewed Binary Tree.

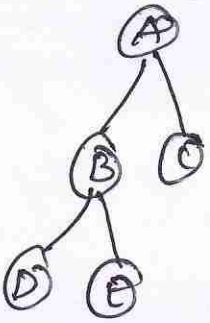
Every node should have only right children is called right Skewed binary Tree.



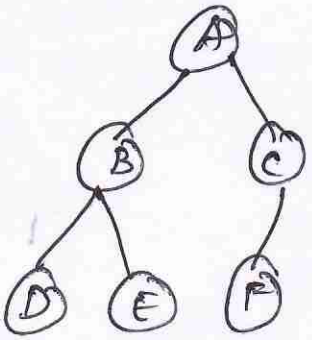
Binary Tree.



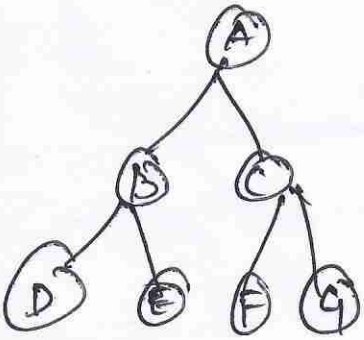
Strictly BT



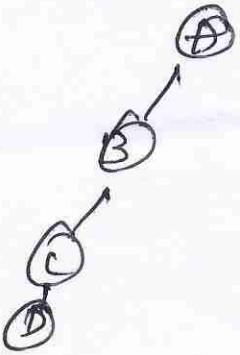
Incomplete/Almost BT



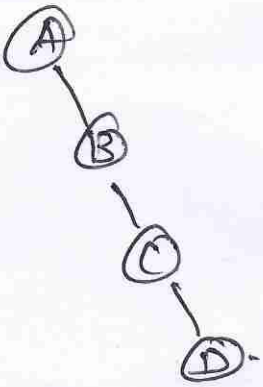
Complete/perfect



Left Skewed BT



Right Skewed BT



(8) Binary Tree Representation.

Represented in Two ways.

i) Linear or sequential way (Array)

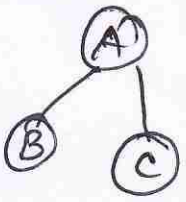
2) Using Linked list Concept.

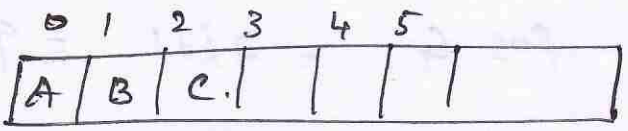
Using Array

Step 1: Consider the root node at index 0.

Step 2: Left child is placed at $2i+1$ where i is position of Parent.

Step 3: Right child is placed at $2i+2$ where i is position of Parent.



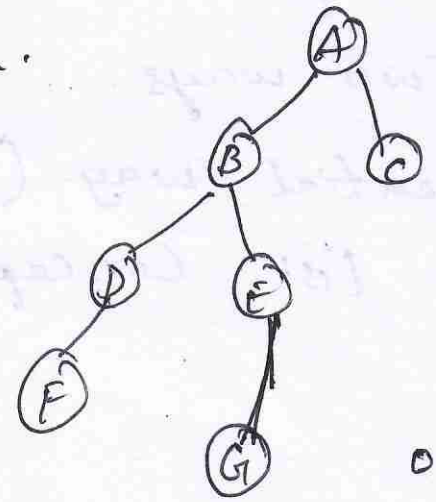


Pos. A = 0

Pos. B = $2i+1 = 1$

Pos C = $2i+2 = 2$

Ex. 2.



0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	–	–	F	–	G

Pos. A = 0

Pos B = 2i+1 = 1

Pos C = 2i+2 = 2

pos D = 2i+1 = 2+1 = 3

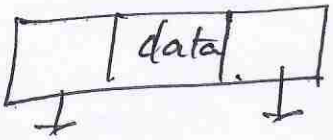
POS E = 2i+2 = 4

Pos F = 2i+1 = 7

Pos G = 2i+1 = 9

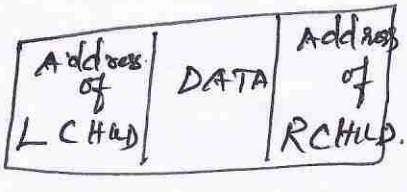
Using Linked List

Using Double Linked List for representing each node.

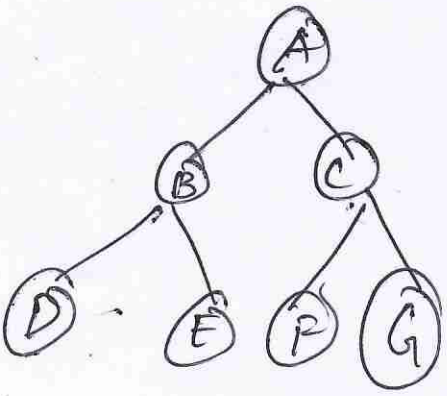


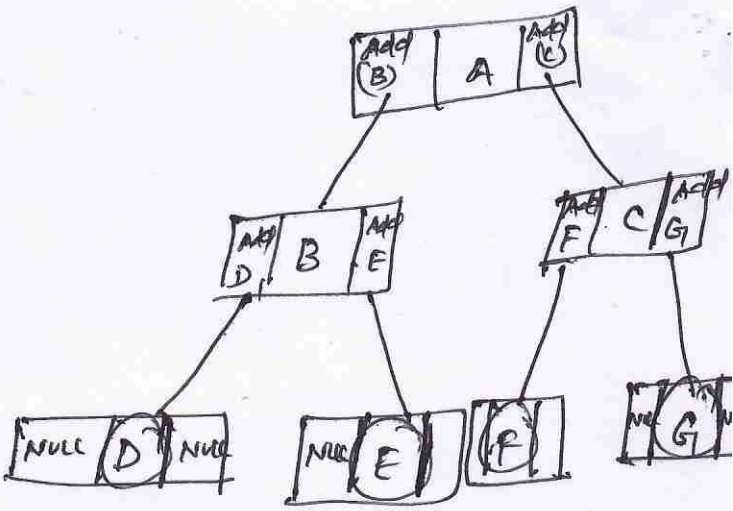
Address of previous

Address of Next.



10

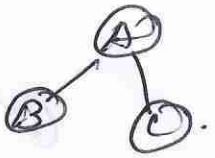

A
B
C
D
E
F
G


Add
(B)
A
Add
(C)
Add
D
B
Add
E
Add
F
C
Add
G
NULL
D
NULL
NULL
E
F
G

Unit II

Tree Traversals

1. Inorder Traversal
2. Pre order Traversal
3. Post order Traversal



Inorder Traversal:

In inorder traversal First we Process Left child and then process ROOT then process Right child.

LC → ROOT → RC.

BAC

Pre order Traversal

In Pre order traversal First we Process the ROOT then process Left child then only process Right child

ROOT → LC → RC.

ABC

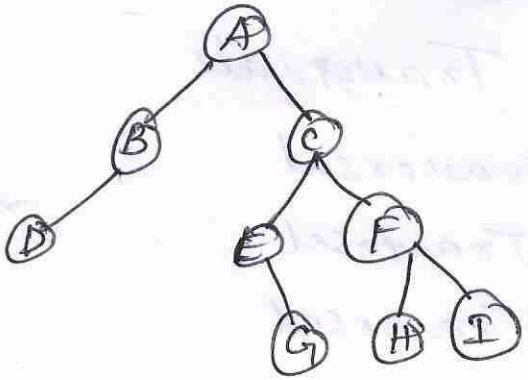
Post order Traversal

In post order traversal first we Process Left child and then Right child then ROOT.

BCA

LC → RC → ROOT

Ex. 1



Inorder ⇒ LC → ROOT → RC

DBAGECHFI

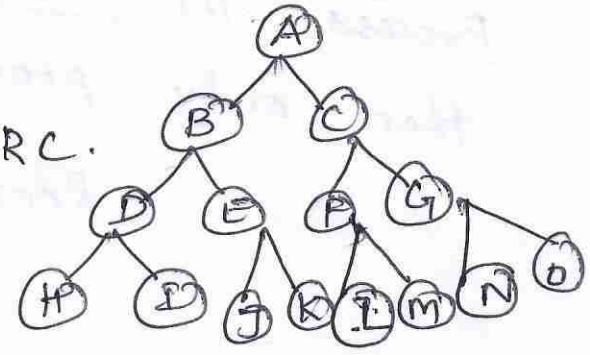
Preorder ⇒ ROOT → LC → RC

ABDEGCFHI

Post order ⇒ LC → RC → ROOT

DBGEHIFCA

Ex - 2



Inorder ⇒ LC → ROOT → RC.

HDIBJEKALFMCNGO

Preorder ⇒ ROOT → LC → RC.

ABDHIEJKCFLMGNO

Post order ⇒ LC → RC → ROOT

HIDJKEBLMFNOGCA

TREE TRAVERSALS

1. Inorder Traversals. LC – ROOT – RC

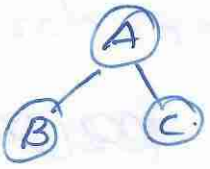
   BAC

2. Preorder Traversals. ROOT – LC – RC

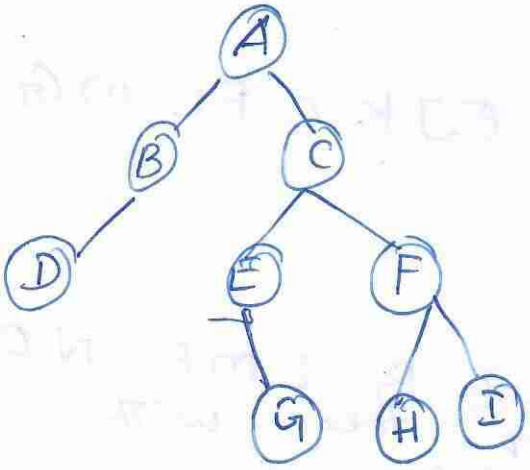
   ABC

3. Postorder Traversals. LC – RC – ROOT

   BCA



Ex.



Inorder: LC – ROOT – RC

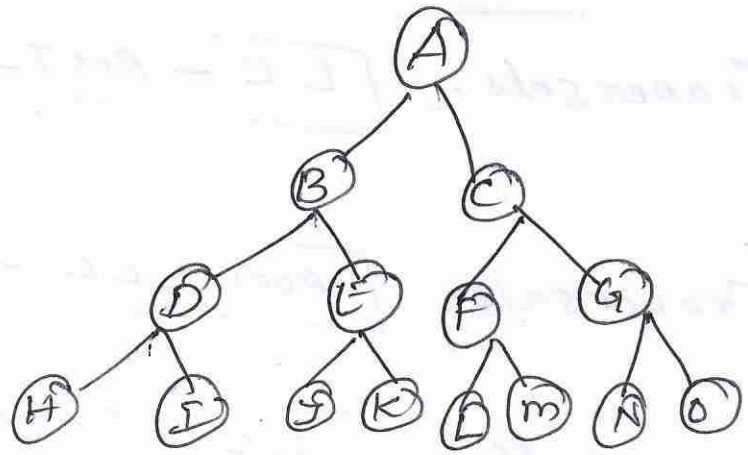
DBAEGCHFI

Preorder: — ROOT – LC – RC

~~ABDD~~ ABDCEGFHI

Post order: LC – RC – ROOT

DBGEHIFCA



Inorder — LC - Root - RC.

~~H D I B J E K L F M C N G O~~ A H D I B J E K L F M C N G O

Pre order

A B D H I E J K C F L M G N O.

Post order

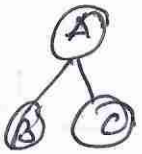
H I D J K E B L M F N O G C A

Construction of tree with inorder & preorder

root RC

Ex Pre order. A B C

Inorder B A C.

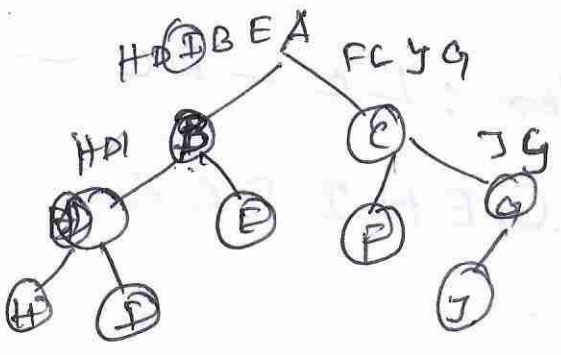


LRRC

Root Pre order A B D H I E C F G J

Inor Inorder H D I B E A F C J G.

RL



Construction of an Expression Tree.

operators — Internal Node
operands — Leaf Node.

Infix

(1) Convert infix expression into Postfix expression.

(2) Read the all characters from the Postfix expression.

(3) If the character is an operand then push into the stack.

(4) If the character is an operator

Step 1 : Pop the two top most elements from the Stack.

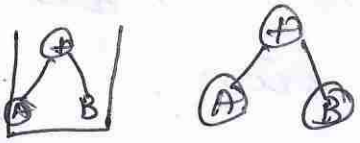
Step 2 : Consider the Operator as root node Construct the tree with operands as LC & RC.

Step 3 : Push the tree as element into the Stack.

(5) Repeat the process until reading all characters from the Postfix expression.

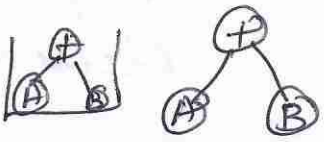
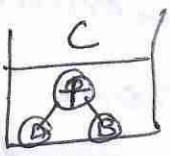
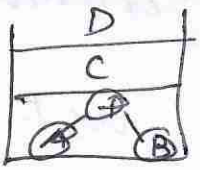
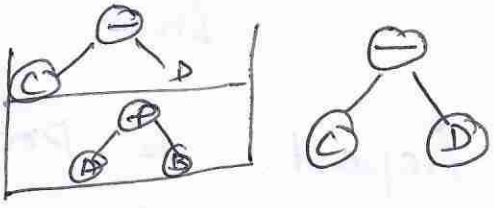
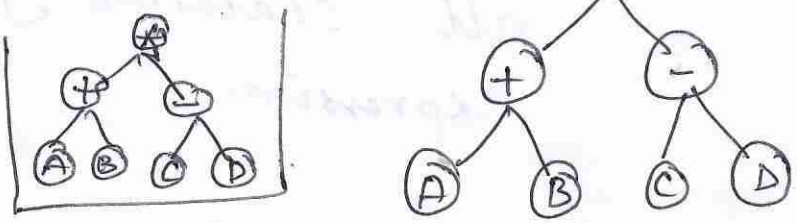
Ex 1. Expression A+B

Postfix. AB+

Char	Stack
A	
B	
+	

Ex 2. (A+B) * (C − D)

AB+ CD− *

Char	Stack
A	
B	
+	
C	
D	
−	
*	

Inorder & Post order Traversal Tree Construction.

POST ORDER → ROOT

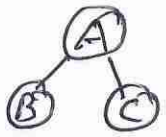
INORDER → LC & RC.

Inorder → LC → ROOT → RC

Post order → LC → RC → ROOT

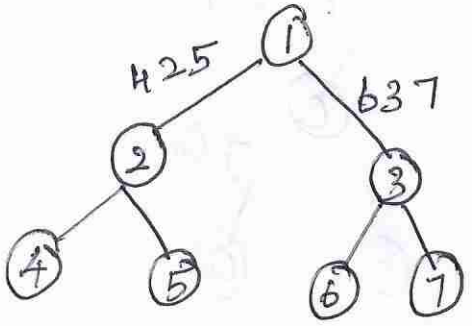
Inorder : BAC

Postorder : BCA



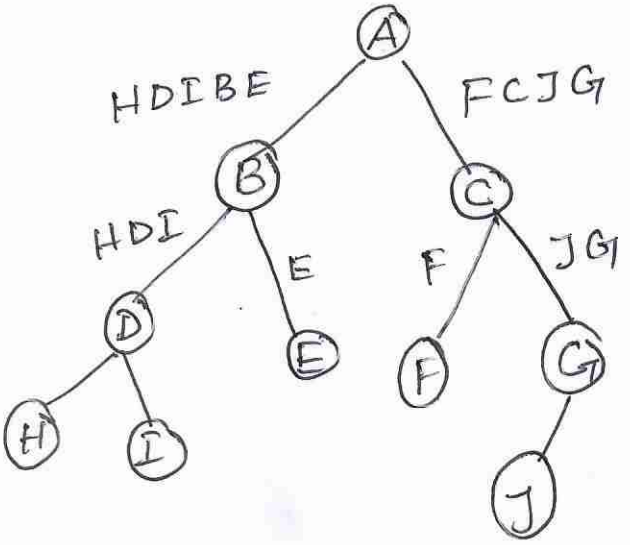
Ex.1. : 4 2 5 1 6 3 7 (INORDER)

7 6 3 5 4 2 1 (POSTORDER)



Ex. 2 : INORDER : H D I B E A F C J G

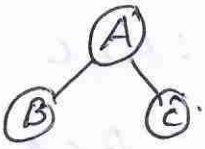
POSTORDER : H I D E B F J G C A



Construction of tree from Inorder and Preorder.

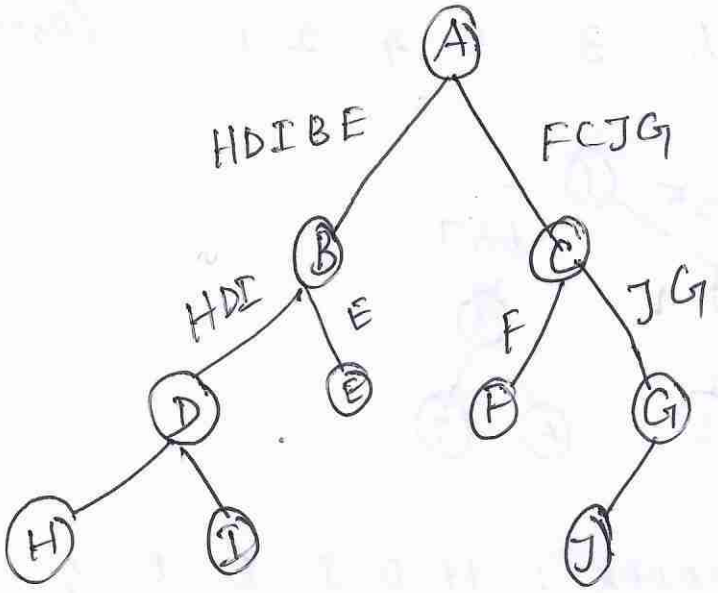
Pre order : A B C

Inorder : B A C



Ex: 1. Pre order : A B D H I E C F G J

Inorder : H D I B E A F C J G



Construction of expression Tree.

Operators - Internal Node.

Operands - Leaf Node.

Infix

1) Convert infix expression to postfix expression.

2) Read all the characters from postfix expression.

3) If the character is operand then Push it into STACK.

4) If the character is operator

Step 1 - Pop the top most two elements from STACK.

Step 2 - Consider operator as root and Construct the tree with elents as Right child and left child.

Step 3: Push the tree as element into stack.

5) ~~Push~~ the tree as element into ~~the stack~~ Repeat the process Until reading all characters from postfix expression.

Ex Expression A+B

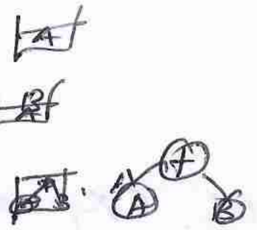
Postfix AB+

Char Stack

A

B

\+



Express (A+B) * (C - D)

AB + CD - *

Char Stack

A

B

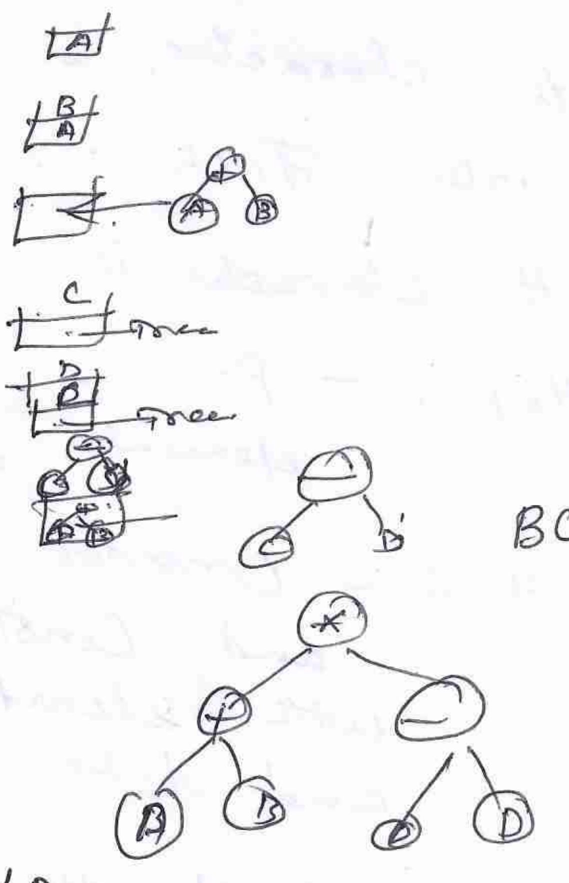
\+

C

D

\-

*



BC* D /

ABC * D / +

A

(Ex3) A+B * C / D.

AB C * D / +

AB C * D

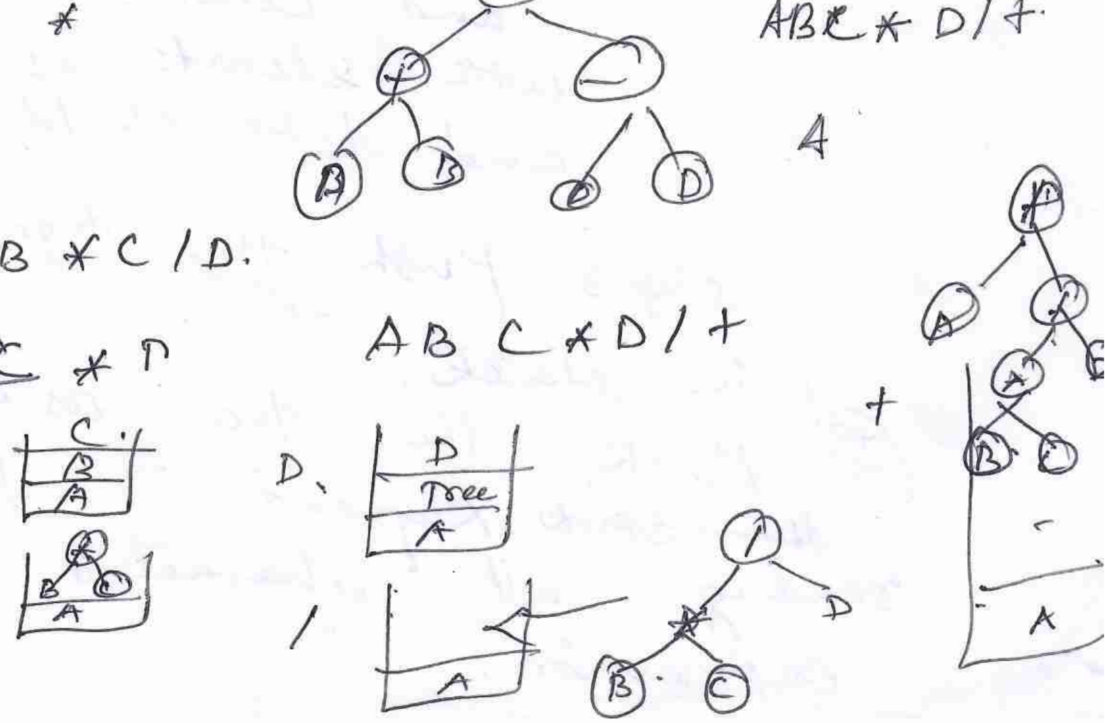
ABC

*

D.

/

\+



Unit II

GRAPH.

Graph is defined as set of Vertices and edges. (V, E)

where V - set of vertices

E - set of Edges.

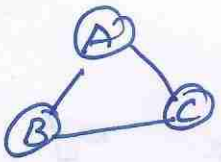
Node:

Vertices represented as circles and also known as nodes.

Edge:

Edge is represented as lines that is connecting two vertices / nodes.

Ex.



A, B, C → vertices / nodes.

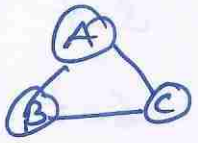
(A, B)
(A, C)
(B, C)
} → Edges.

Terminology.

1. Node.
2. Edge.
3. Adjacent nodes.

4. Degree of Node.

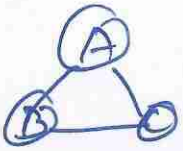
No of Edges connected to that node.



D(A) = 2
D(B) = 2
D(C) = 2

5. Size of the Graph.

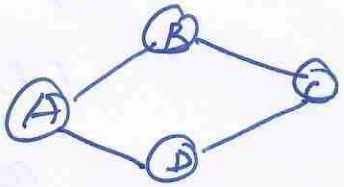
Total number of edges in the graph is the size of the graph.



Size = 3.

6. Path.

Sequence of vertices from Source node to destination node.



(S) A → C (D)

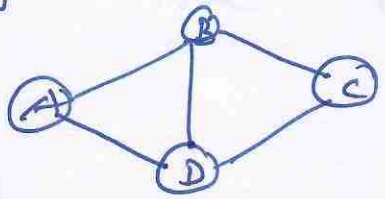
PATH ⇒ A → B → C | A → D → C.

Types of Graphs.

1. UNDirected Graph. (BIDIRECTIONAL)

There is no direction Specified in the edges.

Ex.



(AB) = (B, A)

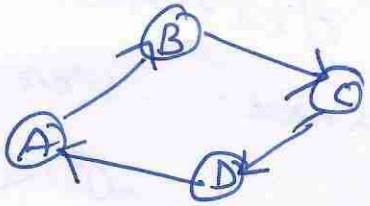
(B, C) = (C, B)

(A, D) = (D, A)

2. Directed Graph. (UNIDIRECTIONAL)

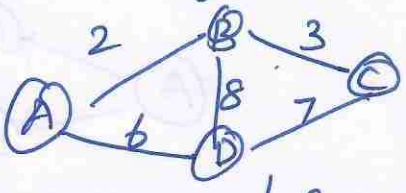
There is direction Specified in the edges.

Ex.



3. Weighted Graph.

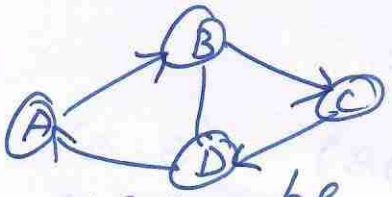
We have Specified some weight or cost in the edge is called weighted Graph.



Graph may be directed or undirected

4. Unweighted Graph.

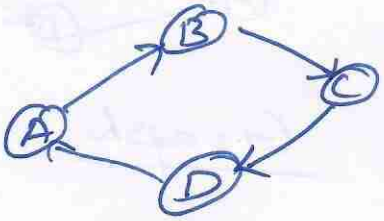
No weight or cost is specified in the edge is called Unweighted Graph.



Graph may be directed or undirected.

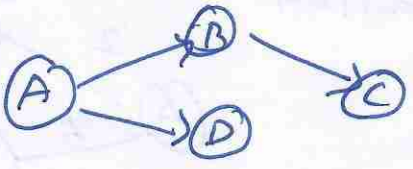
5. Cyclic Graph

It should Perform the loop. That is the source and destination in the is same vertices.



6. Acyclic Graph

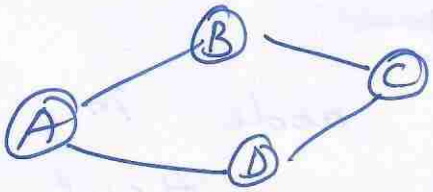
Any Graph does not have cycle that is Acyclic Graph.



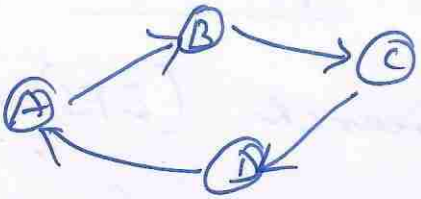
Representation of Graphs.

1. Using Multidimensional Array
2. Using Linked List.

Using Multidimensional Array

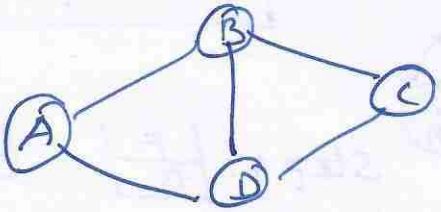


	A	B	C	D
A	0	1	0	1
B	1	0	1	0
C	0	1	0	1
D	1	0	1	0



	A	B	C	D
A	0	1	0	1
B	0	0	1	0
C	0	0	0	1
D	0	0	0	0

Using List

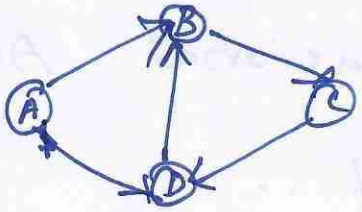


A : B → D

B : A → C

C : B → D

D : A → B → C



A : [B |] → [D | X]

B : [C | X] →

C : [D | X]

D : [B | X]

X → NULL.

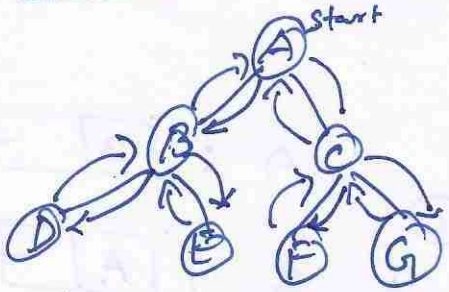
GRAPH TRAVERSALS

From Any starting node in the graph visit the all nodes without repeatation is Graph traversals.

Types of Graph Traversal

1. Depth First Search (DFS) → stack.
2. Breadth First Search (BFS) → Queue.

Depth First Search. (DFS)



DFS.

A C G F B D E

Step 1 : [A] [ ]

Step 2 : [C / B] [A]

Step 3 : [G / F / B] [A | C]

Step 4 : [F / B] [A | C | G]

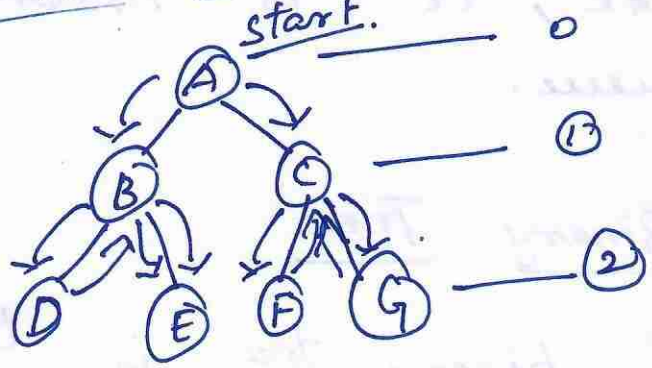
Step 5 : [B] [A | C | G | F]

Step 6 : [D / E] [A | C | G | F | B]

Step 7 : [E] [A | C | G | F | B | D]

Step 8 : [ ] [A | C | G | F | B | D | E]

Breadth First Search – Using Queue.



A B C D E F G

It is done as level wise.

Visit all node as level by level.

Step 1: Queue: [A] BFS: [ ]

Step 2: [B | C] BFS: [A]

Step 3: [C | D | E] [A | B]

Step 4: [D | E] [A | B | C]

Step 5: [D | E | F | G] [A | B | C | D]

Step 6: [E | F | G] [A | B | C | D | E]

Step 7: [F | G] [A | B | C | D | E | F]

Step 8: [G] [A | B | C | D | E | F | G]

BFS: Step 9: [ ] [A | B | C | D | E | F | G]

BFS: Insert the starting node in the Queue. Then insert into the

BFS and if the node have adjacent node, it is inserted into the Queue.

Threaded Binary Tree

Threaded binary tree is to make inorder traversal faster and do it without use of stack.

A binary tree is made threaded by making all Right child Pointers that would normally be NULL point to the inorder Successor of the node (if it exists)

Types of threaded binary trees.

i) Single threaded:

Right NULL pointers is made to point to the inorder Successor (If Successor exists)

ii) Double threaded:

Both Right NULL Pointers is made to point to the

inorder Successor (if Successor exists).

Left NULL pointer is made to point to the Inorder Predecessor respectively.

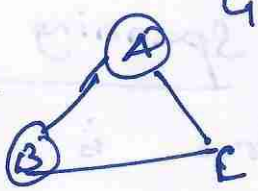
* The threads are also useful for fast accessing ancestors of a node.

Spanning Tree

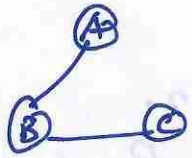
A Spanning tree is a Subset of Graph. G, which has all the vertices covered with minimum possible number of edges.

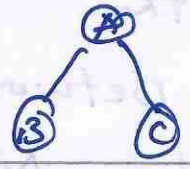
A Spanning tree does not have cycles. and it cannot be dis connected.

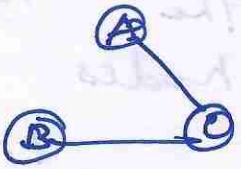
Ex. Graph G.



Spanning Trees

1.


2)


3)


* A Complete undirected graph G have n^{n-2} number of Spanning trees where n is number of nodes.

If $n = 3$ hence $3^{3-2} = 3$ Spanningtree

* Spanning tree has $n-1$ edges where n is vertices.

Minimum Spanning Tree:

In a weighted graph a Minimum Spanning tree is a Spanning tree that has minimum weight than all other Spanning trees of the same graph.

Minimum Spanning tree Algorithms:

1. Kruskal's Algorithm.
2. Prim's Algorithm.

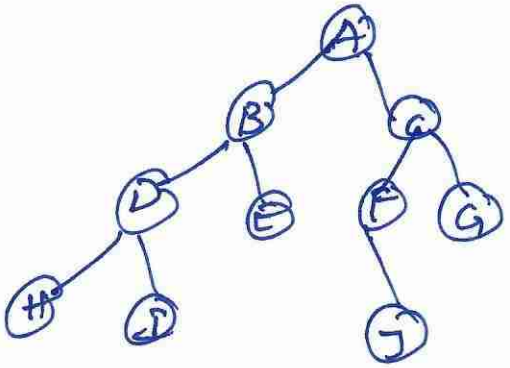
Application of Spanning Tree.

Spanning tree is used to find the minimum Path to connect all nodes in the graph.

1. Civil Network Plan.
2. Computer Network Routing Protocol.
3. Cluster Analysis.

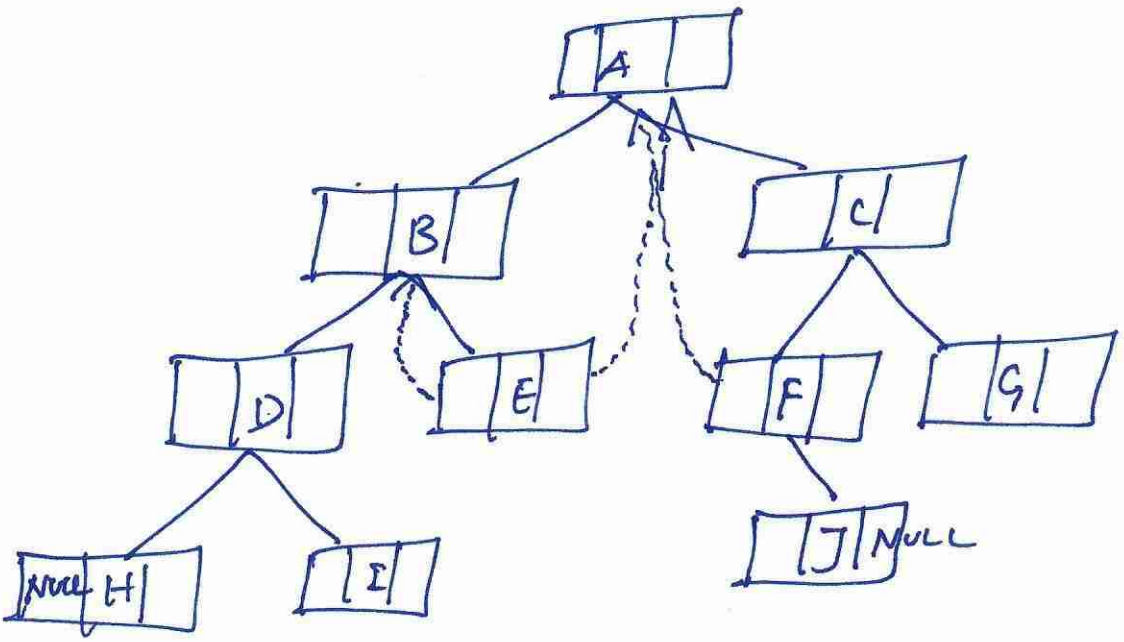
Threaded binary Tree

Threaded Binary Tree (TBT) is also a binary tree in which all left child pointers that are NULL (in Linked list representation) Points to its inorder predecessor, all right child pointers that are NULL in (Linked list representation) Points to its inorder Successor.



Inorder

H D I B E A F J C G.



UNIT III

Algorithm:

Algorithm is a step by step Procedure for solving a Computational Problem.

Applications of Priority Queue.

i) Dijkstra's Shortest path algorithm

2) Any time for fetch with next best, next worst elements.

3) Huffman coding. (Lossless data compression)

4. BFS (Breadth first search) Algorithm

5. Minimum Spanning tree (MST) Algorithms.

Heap:

A max (min) heap is a Complete binary tree with the Property that the value at each node is atleast as large as (as small as) +1. value at its children.

Unit - III

Priority Queue.

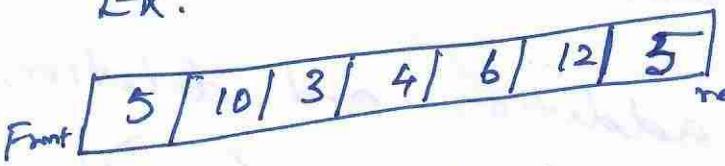
Priority Queue is a data structure that supports the operations of search min (or max), insert and delete min (or max).

Types of Priority Queue

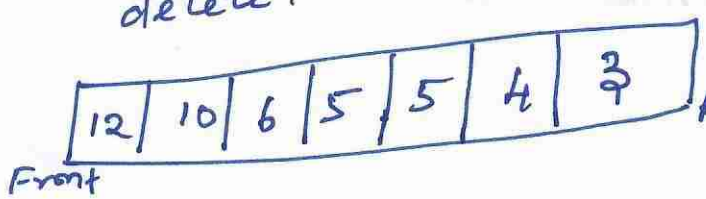
i) Max Priority Queue
2) Min Priority Queue

In max priority Queue the maximum element is deleted first. If there is more than one element have the same priority then the element is deleted as First in First Out method.

Ex.



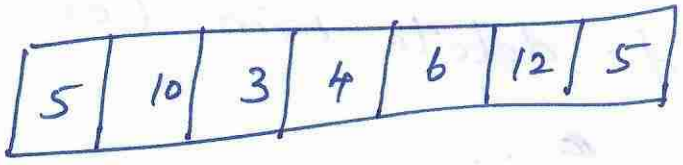
deleted order.



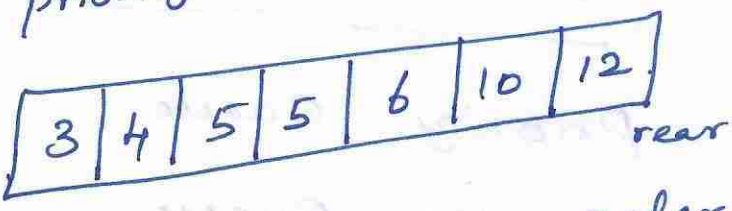
(Decending order)

In minimum priority Queue the minimum element is deleted first. If there is more than one element have the same value then First In First Out method is used.

Ex.



min. priority queue.



(Acending order)

In min priority and max priority Queue all elements have the key value. In max priority Queue the elements are deleted as decending order. and in min priority Queue the elements are deleted as acending order.

Both addition and deletion Can be Performed in $O(\log n)$ time.

Heap Sort

Complete Binary Tree.

Heap order { max heap
min heap

Complete Binary Tree

i) Root - Left - Right

2) Every node should have two children except the last level.

Max heap.

Parent node should be greater than it's children nodes.

Delete and replace root node with last children. Deleted node will be Palced at the last position in array

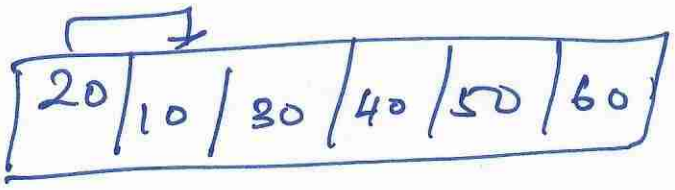
Max heap:

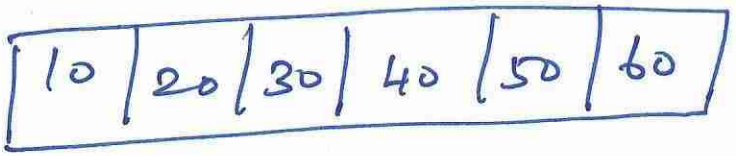
Parent node should be less than it's children node.

Delete and replace root node with last child node. Deleted node will be Placed at last Position in array.

(b)

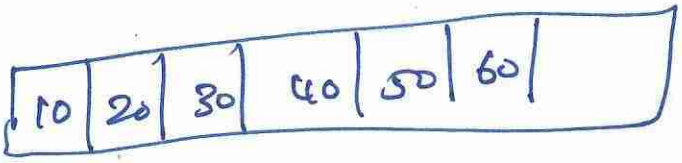
(5)





(6)

(10)



(6)

Merge Sort

Merge Sort using Divide and Conquer method.

Divide and Conquer Method

A large problem is divided into subproblems. After solution founded for subproblem then those subproblems combined and result is founded.

	①	②	③	④	⑤
Index=0	40	60	30	50	10	20
	↓ Low					↓ high

$$mid = \frac{low + high}{2}$$

$$= \frac{0+5}{2} = 2 \quad \text{(Floor division)}$$

Divide:

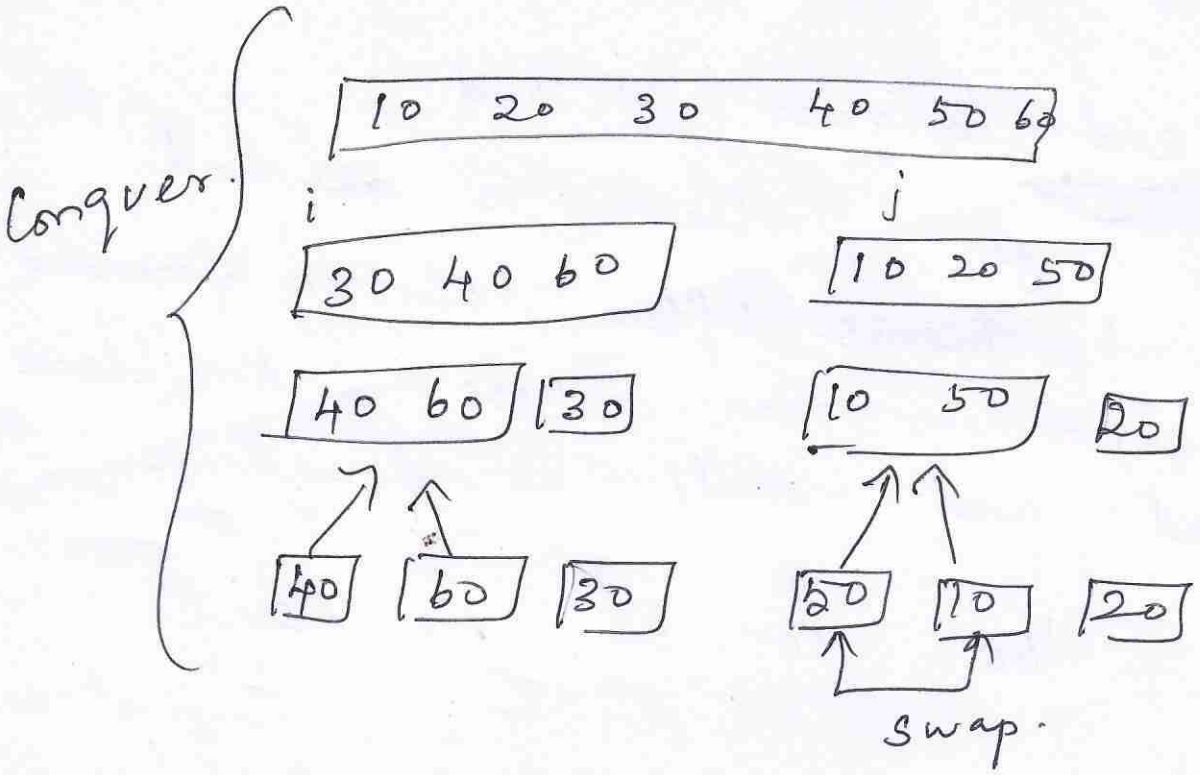
(low to mid) (mid+1 to high)

40	60	30		50	10	20
l		h				

$$mid = \frac{0+2}{2} = 1$$

40	60		30		50	10		20

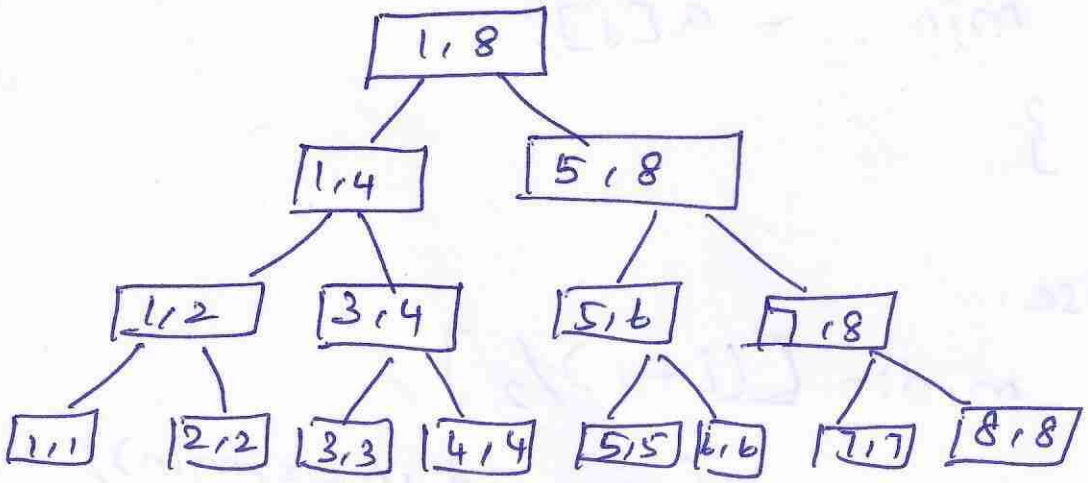
40		60		30		50		10		20


Conquer
10 20 30 40 50 60
i
j
30 40 60
10 20 50
40 60
30
10 50
20
40
60
30
20
10
20
swap.

Merge sort

1	2	3	4	5	6	7	8
9	3	7	5	6	4	8	2

```
Algorithm. mergesort ( l, h)
    {
      if (l<h)
      { mid = (l+h)/2 ;
      mergesort ( l, mid);
      mergesort ( mid+1 , h);
      merge ( l, mid, h);
      }
    }
```



Finding Maximum and minimum

```
Algorithm MaxMin (i, j, max, min)
{
  if (i=j) then max := a[i];
                min := a[i];
  else if (i=j-1) then
     { if (a[i] < a[j]) then
        {
          max := a[j];
          min := a[i];
        }
     else
        {
          max = a[i];
          min = a[j];
        }
     }
  else.
        mid := [(i+j)/2];
        Maxmin (i, mid, max, min);
        Maxmin (mid+1, j, max1, min1);
```

```
If (max < max1) then
    max := max1;
If (min > min1) then
    min := min1;
}
}.
```

① In given array a[i] = a[j] with one element then max = a[i] and min = a[i].

② If the given array has two element the first element is a[i] and the second element is a[j]. Compare a[i] with a[j].

If a[i] < a[j] then min = a[i] and max = a[j].

③ If the given array is more than 2 elements then the array is partitioned into two and so on recursively.

Find the max and min, and find max1 and min1 from the Partitioned.

Compare max and max1.

If max > max1 then

Max = max.

Compare min and min1

If min < min1 then

Min = min.

Binary Search

In Binary Search the given array is already sorted. Then only Search any element from the list.

The first Index is the low and last index is high. First we calculate the mid element = $\frac{low+high}{2}$.

Then the array is splited into two array. We check the mid element with the Search element (key element)

If the Search element is greater than the mid element we only Search the array from mid+1 to high. and low = low+1

If the Search element is less than the mid element then we Search only from low to mid. and high = mid − 1.

3	6	8	12	14	17	25	29	31	36	42	47	53	55	62	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

Key = 42

$$mid = \frac{1 + 15}{2} = 16/2 = 8$$

low	high	mid.
1	15	$16/2 = 8$
9	15	$24/2 = 12$
9	11	$20/2 = 10$
11	11	$11+11/2 = 11$

low	high	mid.
1	15	$16/2 = 8$
9	15	$24/2 = 12$
9	11	$20/2 = 10$
9	9	$18/2 = 9$
9	8	X.

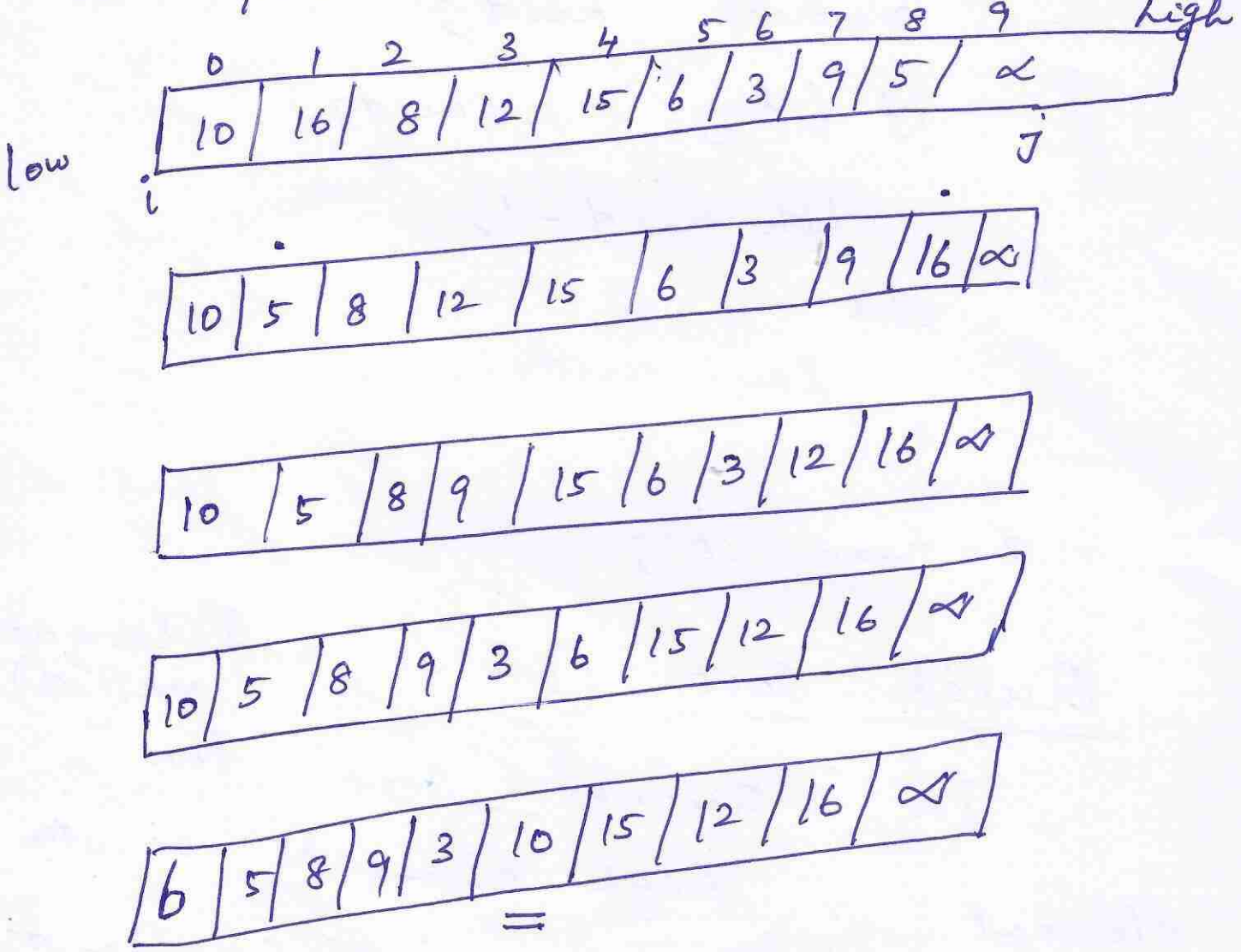
When the low is greater than high the search element is not in the array.

```
int Binsearch (A, n, key)
{
  low = 1, high = n;
  mid = (low + high)/2;
while (low < high)
  {
    if (key == A[mid])
    return mid;
    If (key < A[mid])
       high = mid - 1;
    else
       low = mid + 1;
  }
return 0;
```

Quick Sort (Divide and Conquer method)

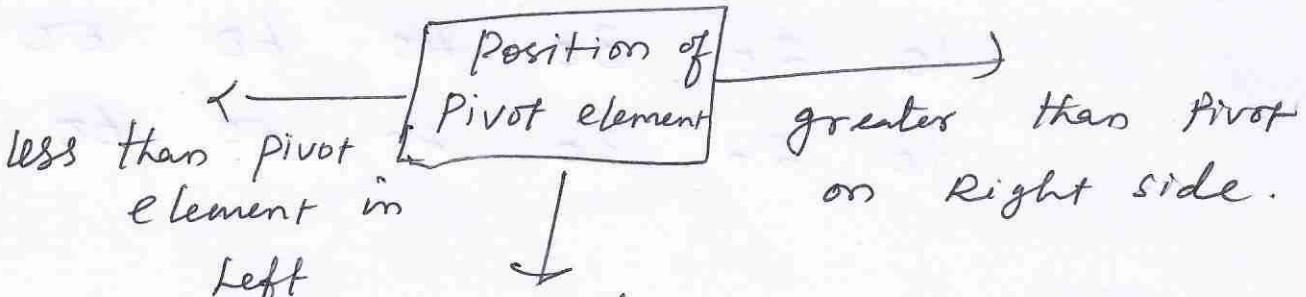
In Quick Sort low element or high element or the middle element is chosen as Pivot element. In first sequence

the pivot element positioned in its actual position. Then the left side of pivot element has smaller numbers. The right side of the pivot element is greater than the Pivot element. The given array is partitioned into two array.



Quick sort

Using divide and conquer method.



maybe
First element
last element
middle element.

Then find the position of Pivot element.

1) Consider the first element as pivot.
2) intialize "i" to low and "j" to high index
3) Repeat the steps until i<j
   a) → keep and incrementing "i" while a[i]≤pivot
   b) → keep an decrementing "j" while a[j]>Pivot
   c) → If i<j then Swap(a[i], a[j])
   d) If i>j then Swap (a[j], Pivot)

a[low]

J is the Position of Pivot

In Quick Sort usually the first element is chosen as pivot element. pivot is "i" and high is assumed (low) as "J."

increment "i" until A[i] ≤ Pivot; i

, and decrement J until A[j] > Pivot;

```
Partition ( l , h)
 {
   Pivot = A[l];
   i = l , J = h ;
 do
   {
     i++ ;
   } while ( A[i] ≤ Pivot);
 do
   {
     J-- ;
   } while (A[j] > Pivot);
 If ( i<j)
     Swap ( A[i], A[j])
   }
```

```
Quicksort (l,h)
{
  if (l<h)
  {
    j = Partition (l,h);
  Quicksort (l,j);
  Quicksort (j+1,h);
  }
}.
```

Strassen's Matrix multiplication:

When the no of columns in first matrix and no of rows in second matrix, then only we multiply those two matrix. In strassen's matrix multiplication formula is used instead of normal multiplication. In normal multiplication there are 8 multiplocation required. But in Strassen's matrix multiplication 7 multiplication only required. So the space requirement is reduced compare to maltilication and addition and subtraction require less memory.

Strassen's Matrix Multiplication

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad 2 \times 2 \qquad \times \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \quad 2 \times 2$$

$$C = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix} \quad 2 \times 2$$

Normal Multiplication

$$C_{ij} = \sum_{i=1}^{n} A_{ik} * B_{kj}$$

```
for (i=0 ; i<n ; i++)
{
    for (j=0 ; j<n ; j++)
    {
        c[i,j] = 0
        for (k=0 ; k<n ; k++)
        {
            c[i,j] = c[i,j] + A[i,k]
                     * B[j,k] ;
        }
    }
}
```

Time Complexity.

$O(n^3)$.

Strassen's Matrix Multiplication

It is using Divide and Conquer method.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} =$$

$$C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$

$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$

$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$

$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

Strassen's Matrix multiplication is applicable for $2^1, 2^2, 2^4 \ldots 2^n$.

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ & (A_{11}) & & (A_{12}) \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ & (A_{21}) & & (A_{22}) \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right]$$

$\frac{4}{2} \times \frac{4}{2}$

$$B = \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b \\ (B_{11}) & & & (B_1 \\ b_{21} & b_{22} & b_{23} & b \\ \hline b_{31} & b_{32} & b_{33} & \\ (B_{21}) & & & (B_2 \\ b_{41} & b_{42} & b_{43} & \end{array}\right]$$

$\frac{4}{2} \times \frac{4}{2}$

```
Algorithm mm (A, B, n)
{
    if (n ≤ 2)
    {
        c = 4 formulas.
    }
    else
    {
        mid = n/2 ;
        mm (A11, B11, n/2) + mm (A12, B2
                                 n/2)
        mm (A11, B12, n/2) + mm (A12, B2
                                 n/2)
        mm (A21, B11, n/2) + mm (A22, B1
                                 n/2)
        mm (A21, B12, n/2) + mm (A22,
                                 B22, n/2).
    }
}
```

Recurrence relation.

$$T(n) = \begin{cases} 1 & n < 2 \\ 8T(n/2) + n^2 & n > 2 \end{cases}$$

$2^8 = 3$.

Time complexity.

$O(n^3)$.

Strassen's Matrix multiplication.

$$P = (A11 + A22)(B11 + B22)$$

$$Q = (A21 + A22)\, B11$$

$$R = A11\,(B12 - B22)$$

$$S = A22\,(B21 - B11)$$

$$T = (A11 + A12)\, B22$$

$$U = (A21 - A11)(B11 + B12)$$

$$V = (A12 - A22)(B21 + B22)$$

$$C11 = P + S - T + V$$

$$C12 = R + T$$

$$C21 = Q + S$$

$$C22 = P + R - Q + U$$

For.

(23)

Strassen's Matrix Multiplication Recurrence relation.

$$T(n) = \begin{cases} 1 & n < 2 \\ 7T(n/2) + n^2 & n \geq 2 \end{cases}$$

$\log_2 7 = 2.81$ ($k = 2$)

$O(n^{\log_2 7}) = O(n^{2.81})$