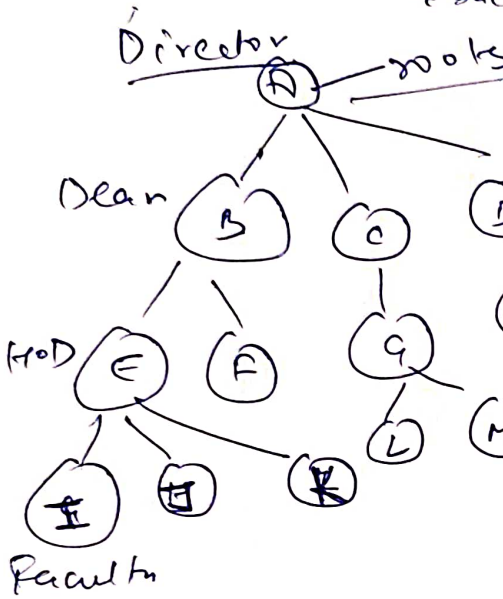
Introduction to Trees

{Array, LL
Stack, Queue}

{Tree Graph}



Tree is a non-linear DS to simulate hierarchical structure.

Unidirectional → it can grow from top to bottom

Nodes — Stores info + link to other node

"Tree is a non-linear DS, defined as collection of nodes linked together to simulate a hierarchy.

Root → node with no parent — (A)

nodes → A, B, C, D, E, F, G, H, I, J, K, L, M

Parent → A, B, C, D, E, F, G

Child ⇒ L, M are children of G

Leaf nodes → nodes with no child/children. (External node)

Eg: I, J, K, L, M

Path — Sequence of consecutive edges from source node to destination node.

Ancestor → any predecessor node on the path from root to that node.

Ex ancestor of L, → A, C, G
ancestor of H → D, A

Decedent → any successor node on the path from that node to ~~root~~ leaf node.

Ex decedant of C → G, H, M
decendant of B → E, F, I, J, K

Degree → Max. degree of tree = max. no. of children a node can have in that tree.

degree of tree → Max. degree among all nodes

Depth of node :- length of path from root to that node

Ex Depth of J = 4

depth of root node = 0

height of node ⇒ no. of edges in the longest path from that node to leaf.

Ex height of A ⇒ 4
height of G ⇒ 1
height of B ⇒ (2)

Note :- height of tree = height of root node

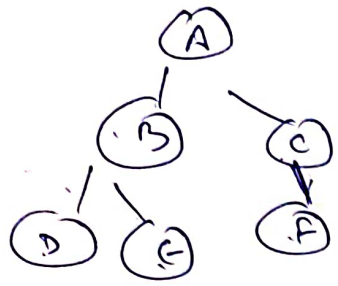
Siblings → children of same parent

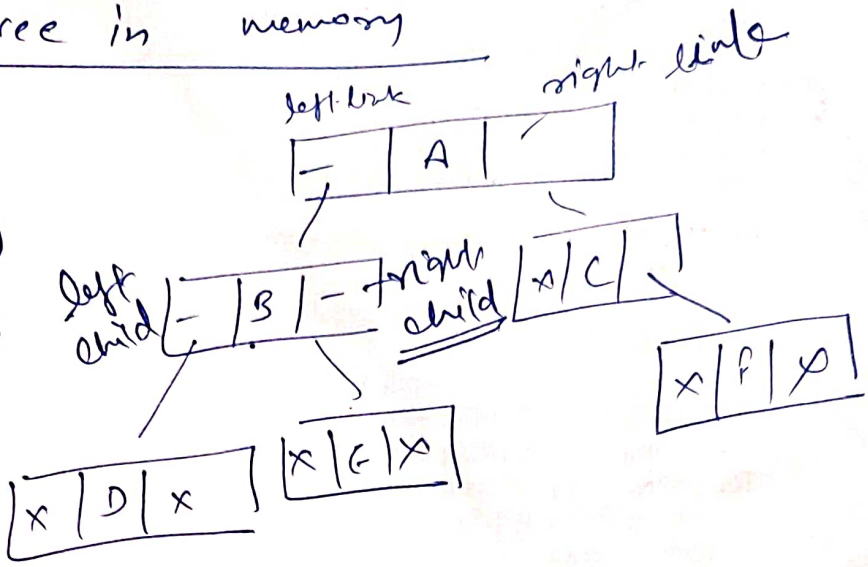
B, C, D are siblings

level of node :- the tree given has total 4 levels starting from 0 to 3.

In any tree if there are (n) nodes then no. of edges = (n-1)

How to represent tree in memory





Representation of tree node in C

```
Struct node
{
    int data;
    struct node *left;
    struct node *right;
};
```

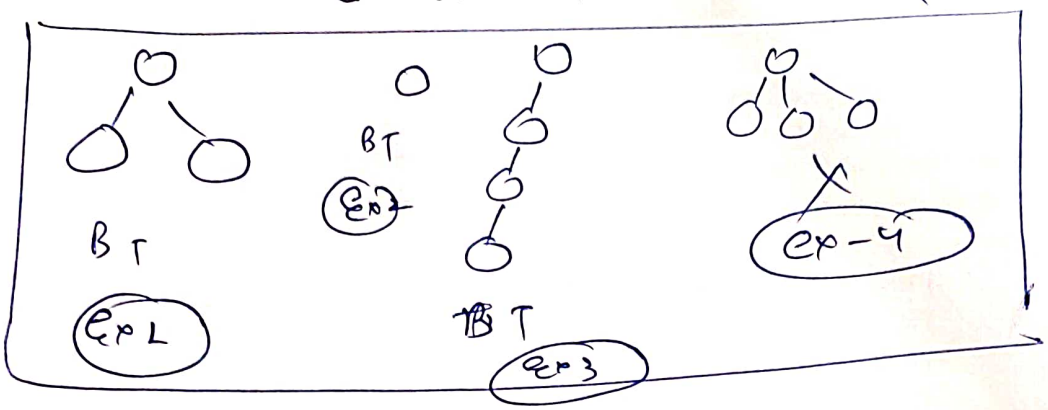
Applications of Tree

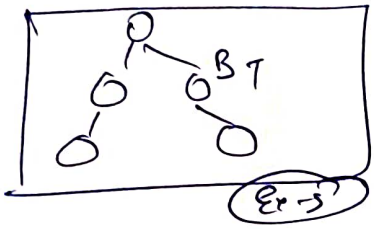
File System
Routing Protocols
Quick searches.

Binary Tree

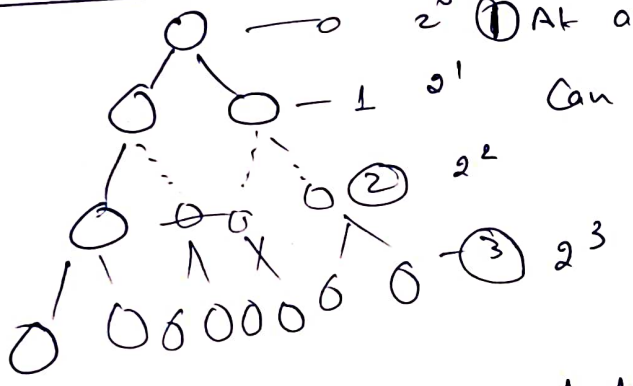
In a Binary Tree Each node can have atmost 2 children (either 1 or 2 or no child)

Ex





Properties of B-Tree



(1) At any level (i) 2^i nodes can be present in a Binary Tree.

(2) max. no. of nodes of height h = $2^{h+1} - 1$

Proof

at level 0 → 0
at level 1 → 2
at level 2 → 4

total nodes = $2^0 + 2^1 + 2^2 \dots 2^h$

$= (2^{h+1} - 1)$

(3) Min no. of ~~nodes~~ ~~of~~ at height h = $(h+1)$

Proof

total nodes = $(h+1)$

height 1 = 2 nodes
height 2 = 3 nodes
height 3 = 4 nodes

(4) Total no. of leaf nodes in Binary Tree
= total no. of degree 2 nodes + 1

(5) Max. no. of Binary Tree with n nodes

$$= \frac{^{2n}C_n}{n+1} \Rightarrow$$

$$^nC_r = \frac{n!}{r!(n-r)!}$$

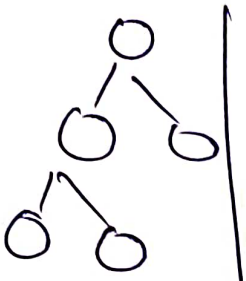
(6) Height of a Binary Tree = Total levels in BT − 1

$$h = L - 1$$

Types of Binary Tree

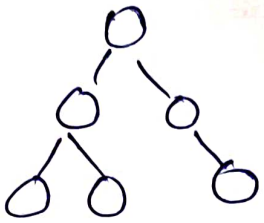
Full Binary Tree / Strict A BT with either 0 or 2 children.

Ex

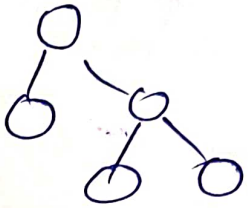


FBT

FBT



X FBT

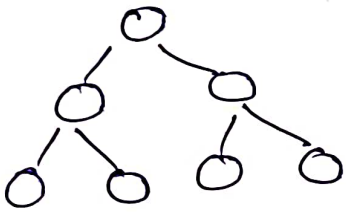


FBT

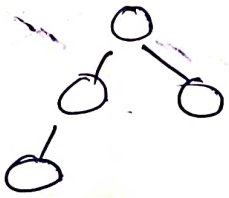
Almost Complete Binary Tree

→ A BT which always fill from left to right

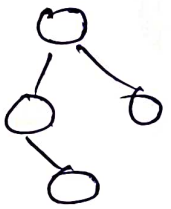
→ only after filling one level you will go to next level



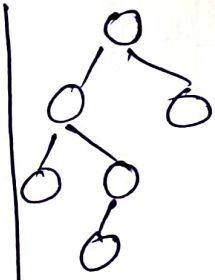
ACBT



ACBT



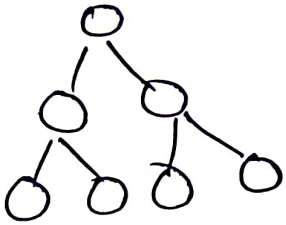
X ACBT



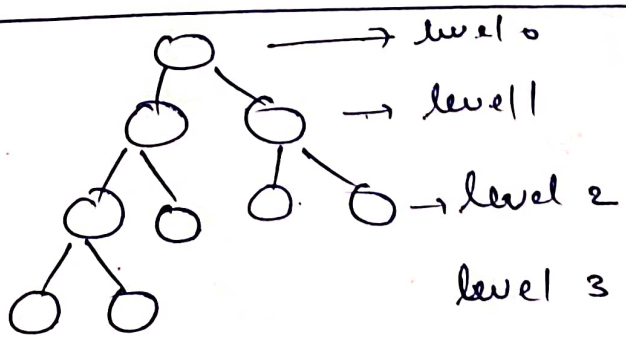
X ACBT

Complete Binary Tree

A BT where internal nodes always have 2 children except leaves, and no holes are allowed



complete Binary Tree



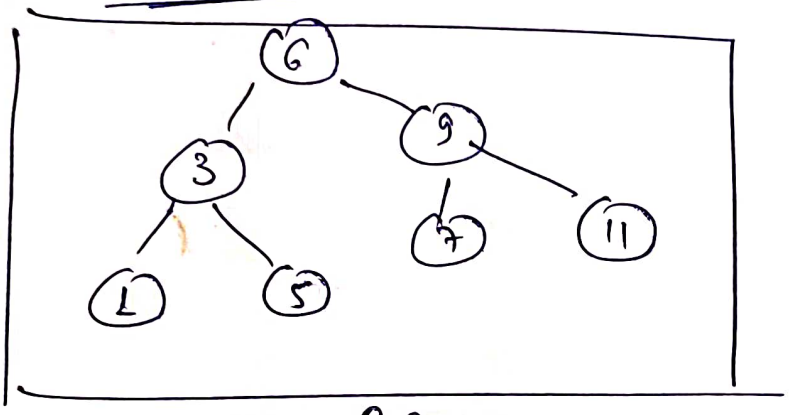
X complete Binary Tree (No holes allowed)

Pointing root node

Binary Search Tree

* Is a special kind of Binary Tree, which overcome the drawback of Binary Tree where nodes are inserted with no order.

* In BST, a node's key is greater than or equal to it's left child's key but less than or equal to right child's key.



BST

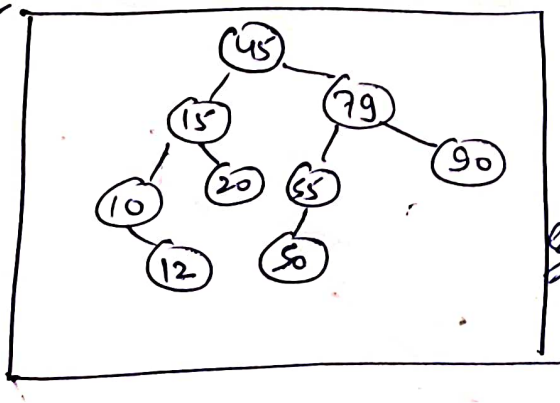
* BST inorder will always give an sorted array

in above tree inorder

| 1, 3, 5, 6, 7, 9, 11 |

* How to create A BST, given an array

45, 15, 79, 90, 10, 55, 12, 20, 50



Q. Generate a BST by inserting following keys

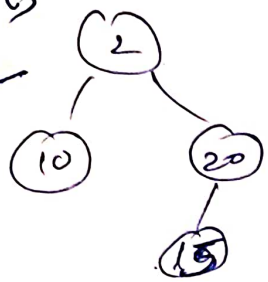
50, 15, 62, 5, 20, 58, 91, 3.

Process :- Select first value as node, & than add every key value in the left if smaller or right if greater than root.

Deletion in BST

- Deletion of leaf node
- Deletion of Non-leaf node.
  - Non-leaf node with one child
  - Non-leaf child with 2 children

Eg:

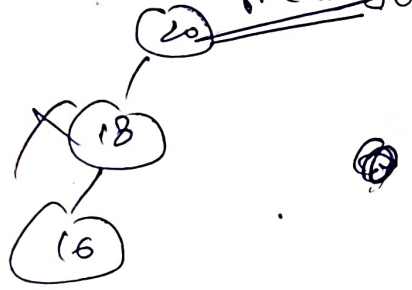


delete (15) (No change in BST after Deletion of leaf node)

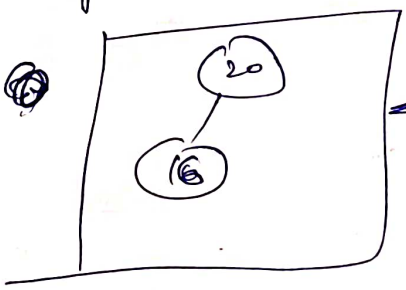
Removing Non-leaf Node with one child

Process: Join the leaf node with grand Parent node

Eg:



Solution

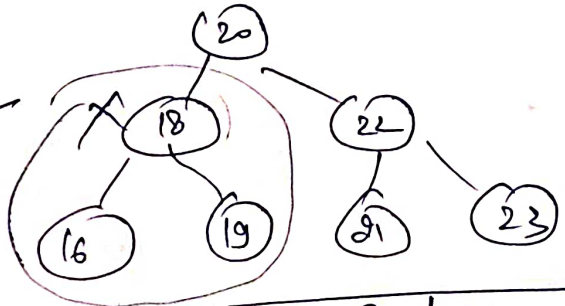


Removing Non-leaf node with 2 children

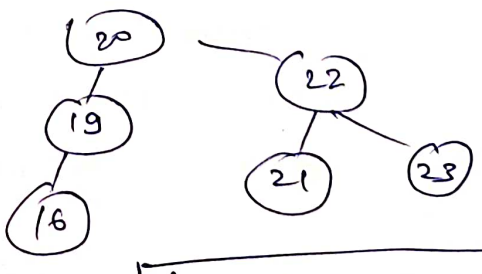
Method-II

(1) Pick the largest element from the left subtree & put it in the place of the node which you want to delete

Eg:



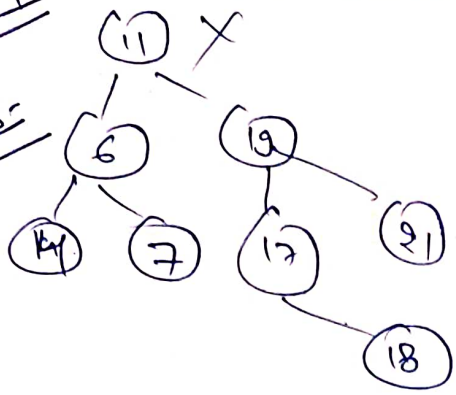
remove node 18 from above tree

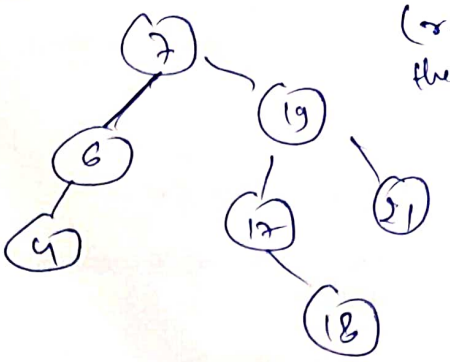


tree after removal of 18

Remove root node from left subtree (Method I)

Q.no:-1



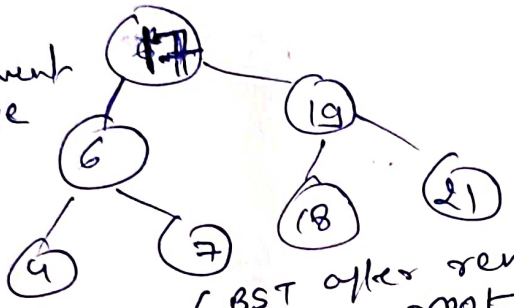


(replace the highest element of left sub tree with root

Hence, new root will be (7) :-

Method-II

Replace the smallest element of Right subtree with node that is being deleted



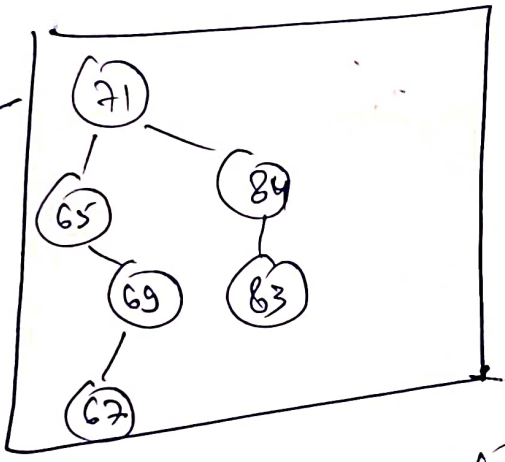
(BST after removal of root node (11))

(BST after removal of 11)

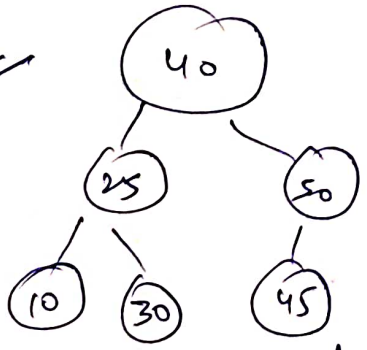
=

Q.: 71, 65, 84, 69, 67, 83 (Insert the given keys to a BST)

Solution



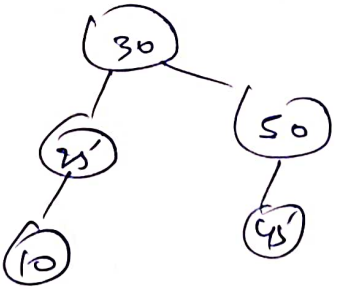
Q.:



(Remove root node 40 from above BST)

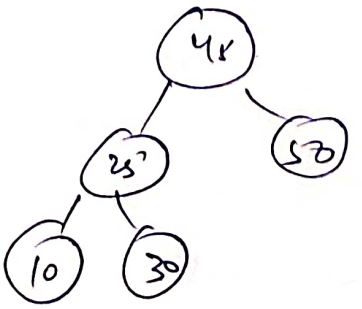
Method-1

removing 40, and replacing it with largest node of left subtree i.e 30

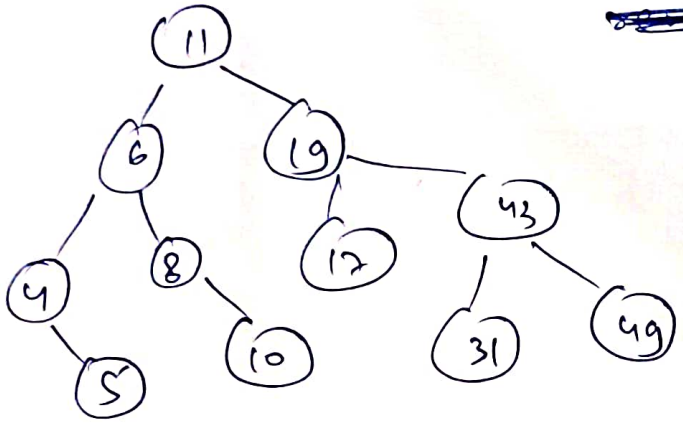


~~Method-2~~ ~~Replacing from Right subtree~~

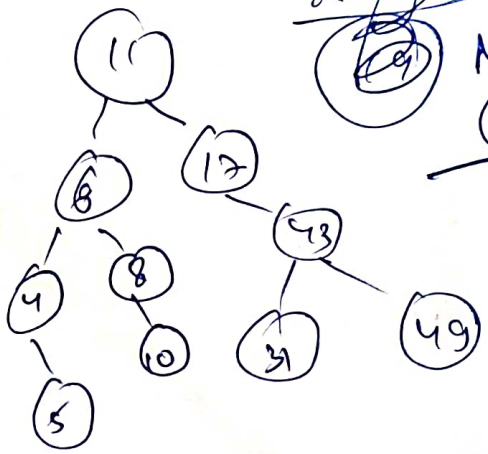
Removing 40, and replacing it with smallest node of Right subtree i.e, (45)



Remove (19) from the following BST.

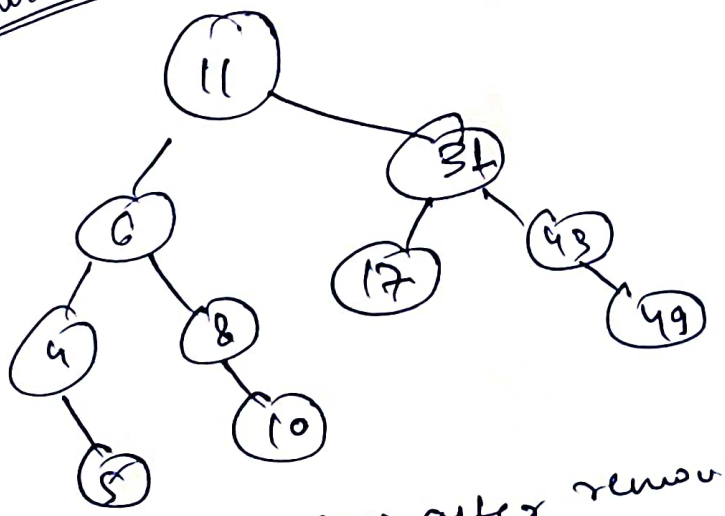


remove (11)



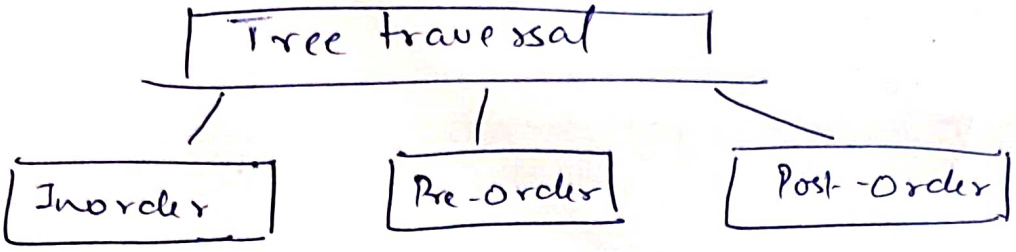
Method I

Method 2:



Solution (BST after removal of ~~BST~~ node (19) by replacing it with smallest node of right subtree node (31).

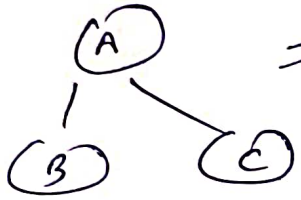
BINARY TREE TRAVERSAL



INORDER TRAVERSAL

LEFT - ROOT - RIGHT

(Here we recursively visit the left subtree, then root & then explore the right subtree) used to create a sorted list

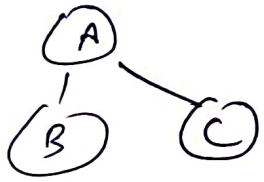
Ex: 

⇒ Inorder traversal

B A C

PRE-ORDER TRAVERSAL

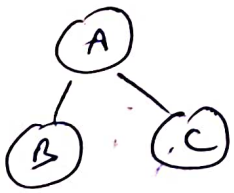
ROOT - LEFT - RIGHT [the order of traversal]



Ex ⇒ A B C

POST-ORDER TRAVERSAL

~~ROOT - RIGHT - LEFT~~ LEFT - RIGHT - ROOT [order of traversal]

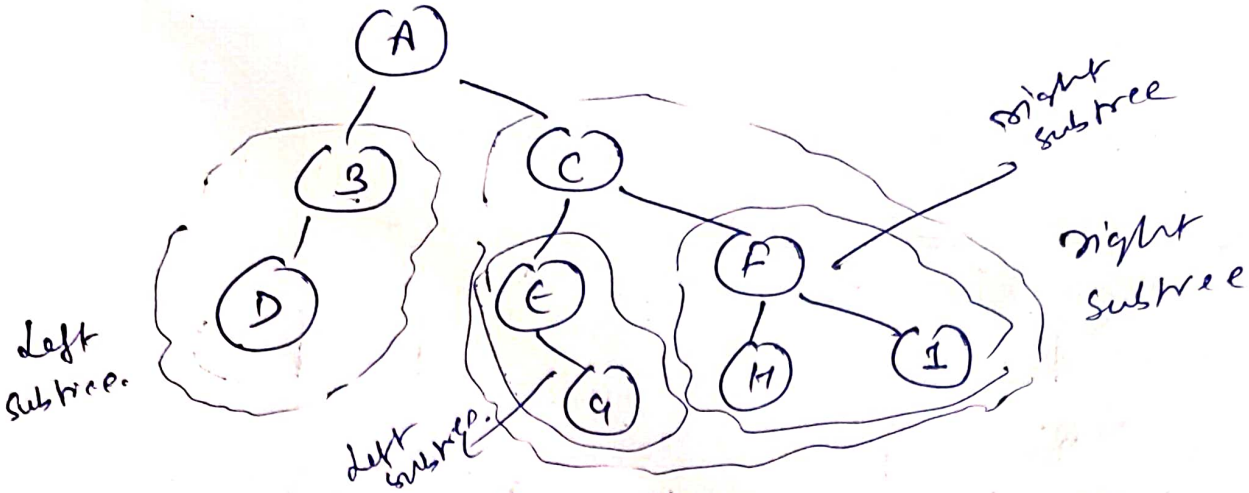
Ex 

⇒ ~~A C B~~ B C A

POST ORDER visits the left & Right subtree first, then ends up at root.

This is often used to delete trees.

Ex 1



INORDER (Left - Root - Right)

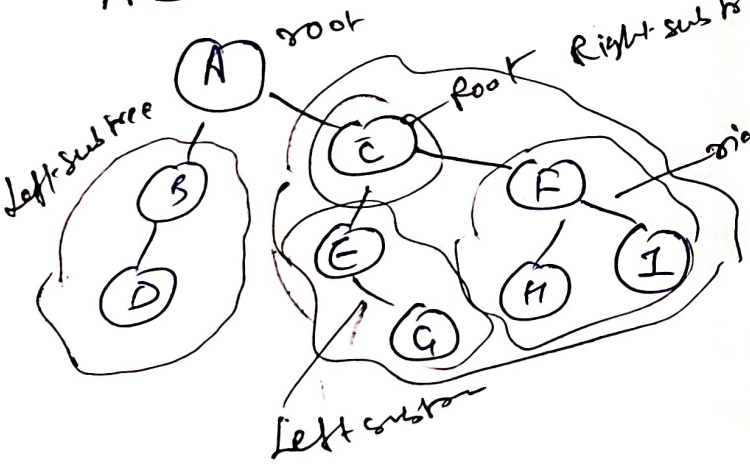
D B A E G C H F I

PRE-ORDER (Root - Left - Right)

A B D C E G F H I

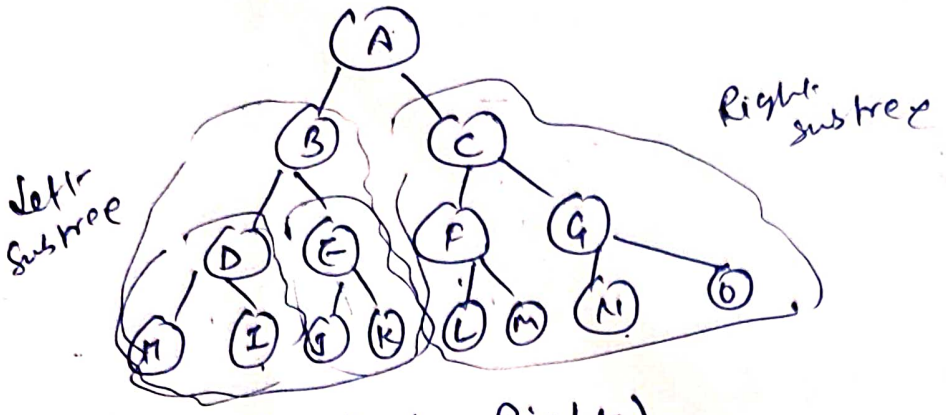
POST-ORDER (~~Root - Right - Left~~) Left Right Root

~~A C F I H E G B D~~



D B G E H I F C A

Eg-2 Find out the Pre-order, Postorder, Inorder traversal of Binary Tree.



INORDER: (Left - Root - Right)

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

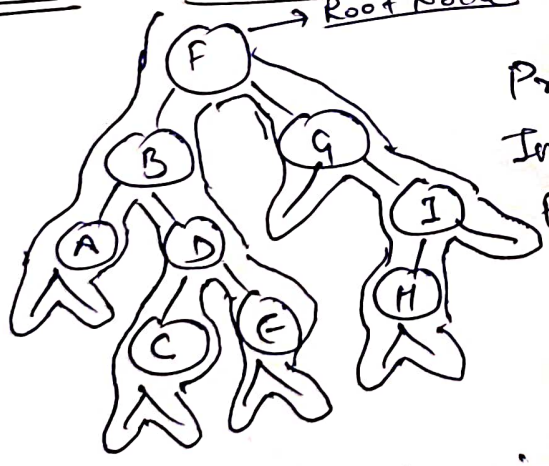
PREORDER: (Root - Left - Right)

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

POSTORDER: (Left - Right - Root)

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

TRICK to traverse a Binary Tree



Pre Order - F B A D C E G I H

In Order - A B C D E F G H I

Post. Order - A C E D B H I G F

Trick:- if node is visited once but it in Pre-order
" " " twice " In-order
" " thrice " Post.-Order