Today's Content :-

Trees → Basic Idea

Terminologies

Binary Tree

Traversal → Pre
→ Post
→ In

Size

Height.

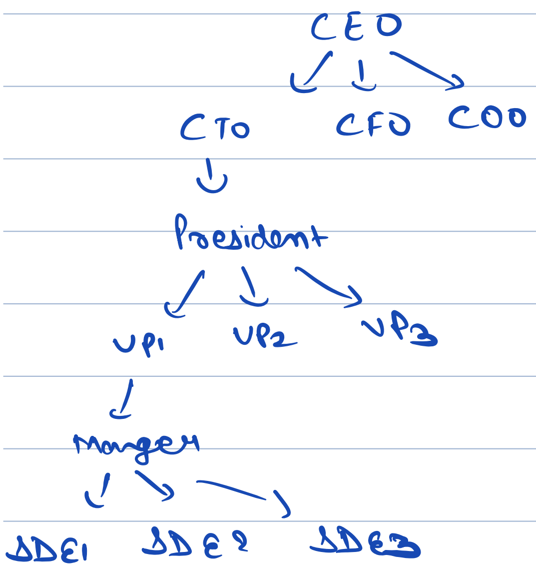
Today's Quote :-

If you can't be happy with a coffee, you won't be happy with a yatch.

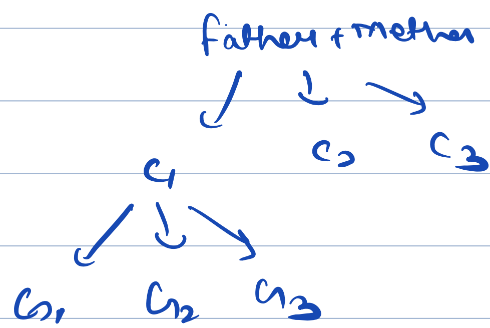
Linear

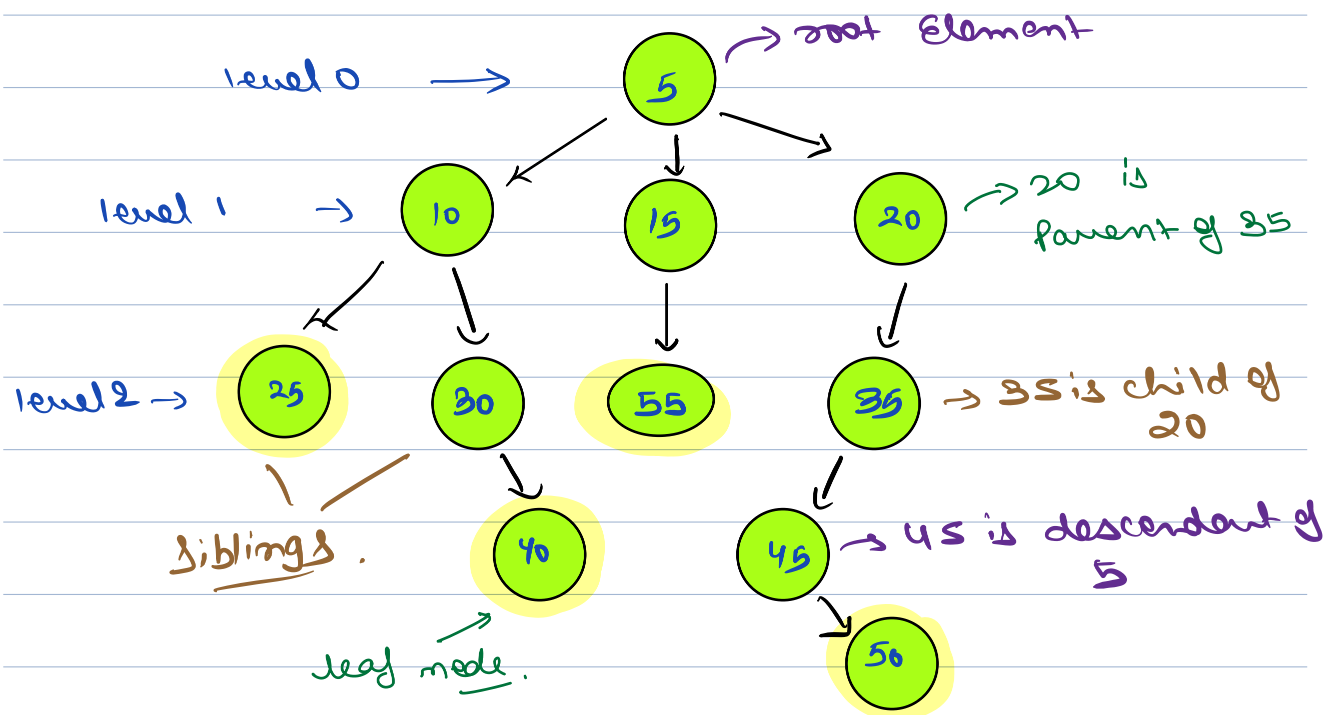
↳ arrays, LL, stacks, Queues.

Hierarchial Data :-



Family Tree



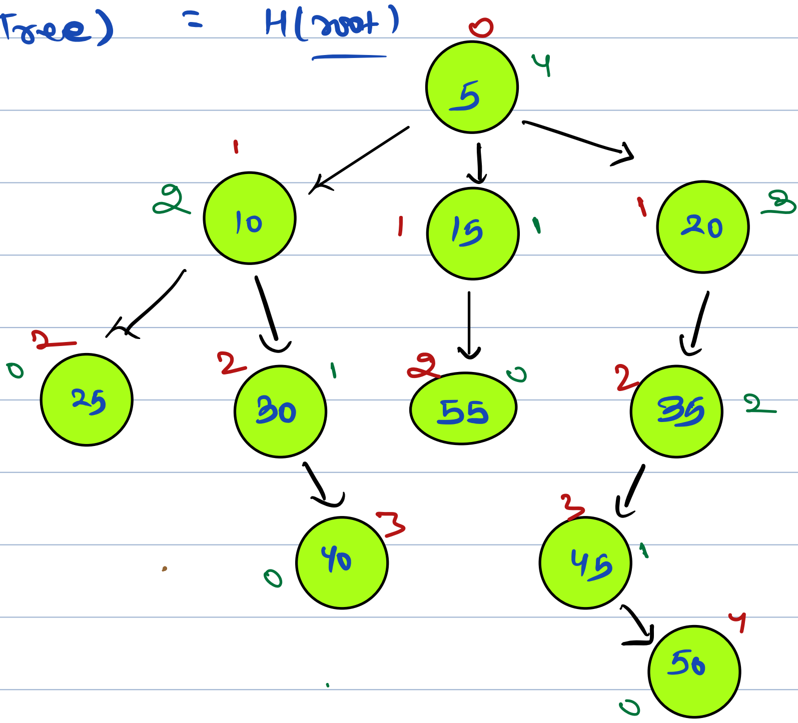


Leaf Node :- Node with no child.

Root Node :- Node without a parent.

H(node) = 1+ Max (Height of its children)

H(Tree) = H(root)

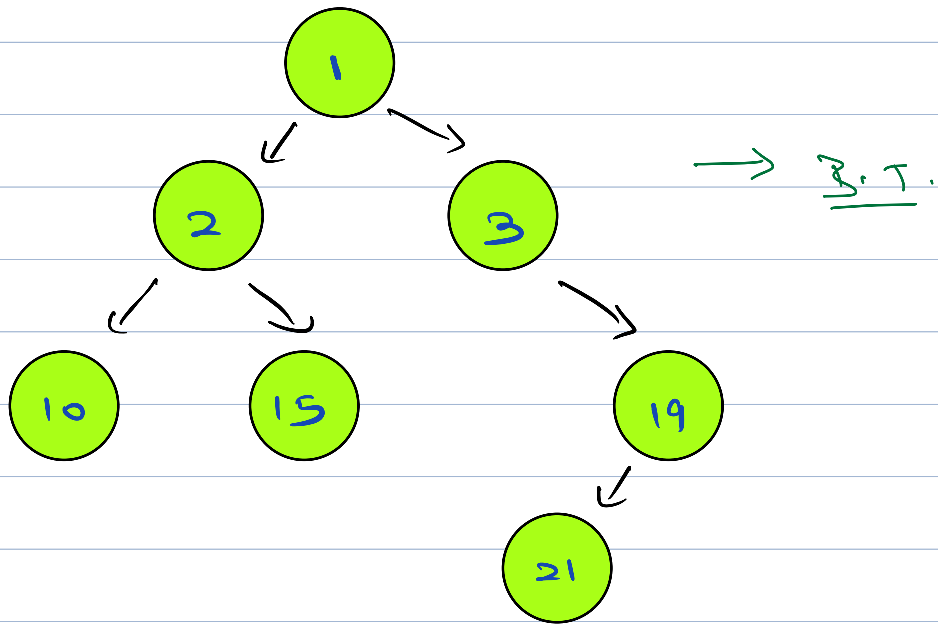


Height(node) :- Length of longest path from that node to any leaf node below it.

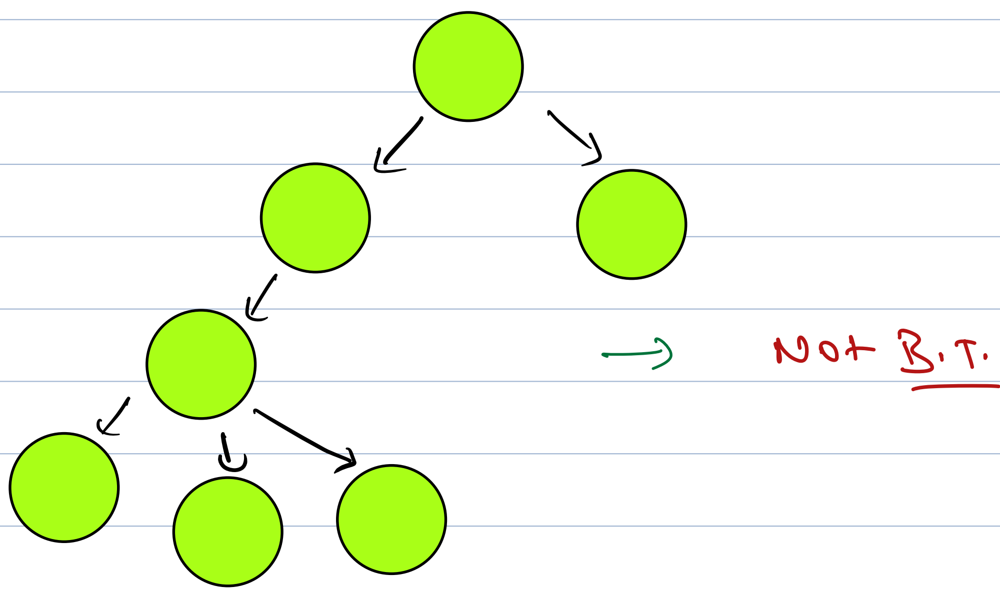
depth(node) :- Length of path from root to that node.

depth(node) = 1+ depth(its parent).

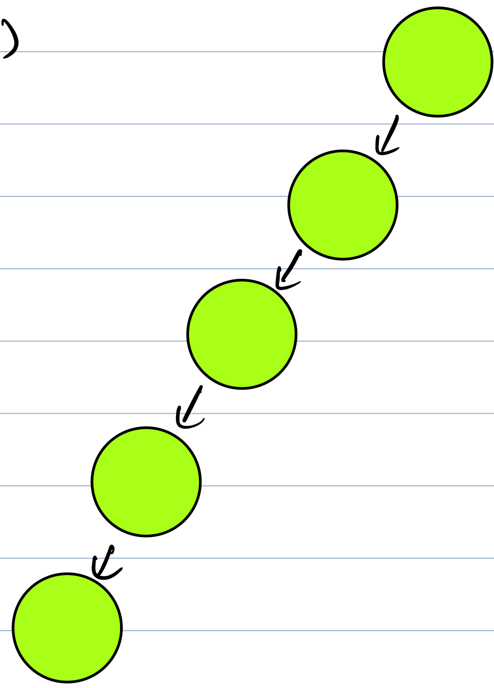
Binary Tree :- → Every node can at max have 2 children.



eg 2)

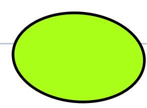


e.g 3)



→ B.T.

4)



→ B.T.

```
class Node {
    int data;
    Node left;
    Node right;
    Node (int x) {
        data = x;
        left = null;
        right = null;
    }
}
```

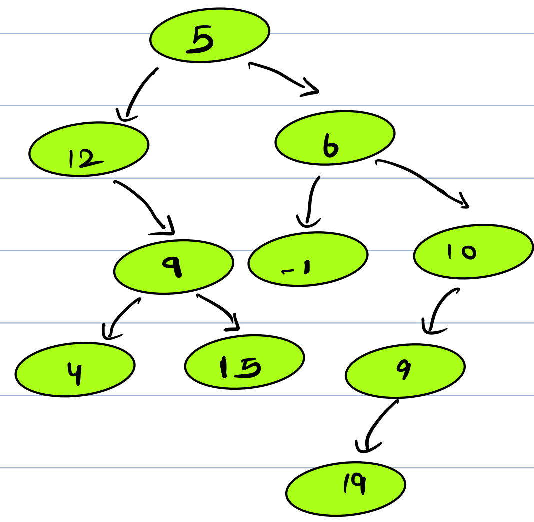
Tree Construction

↓

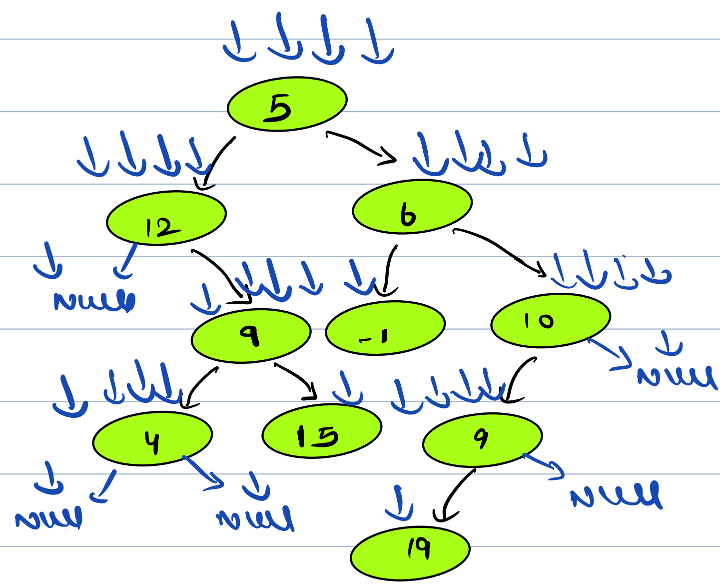
Advance.

Tree Traversals :-

1) Inorder
2) Preorder
3) Postorder.



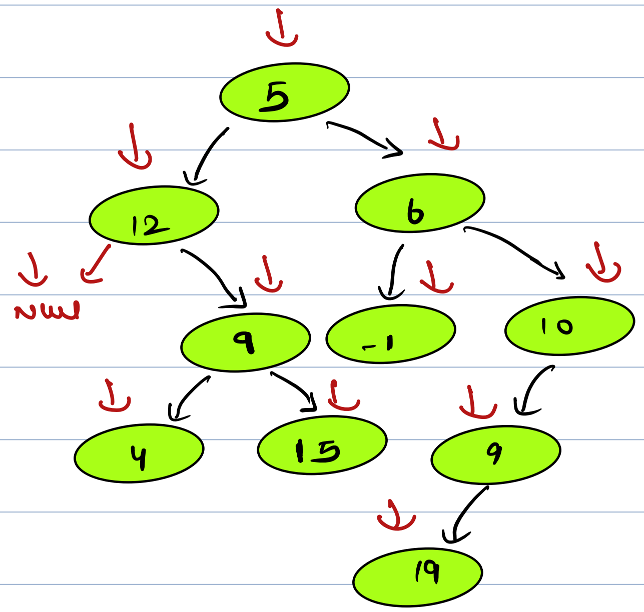
1) InOrder :-   Left Data Right.



12 4 9 15 5 -1 6 19 9 10

L   R

```
void InOrder(Node root){
    if(root==null){return}
    inorder(root.left);
    print(root.data);
    inorder(root.right);
}
```



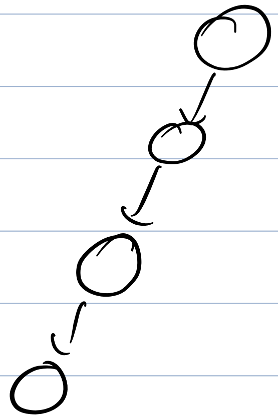
12 4 9 15 5 -1 6

19 9 10

S.C → O(H).

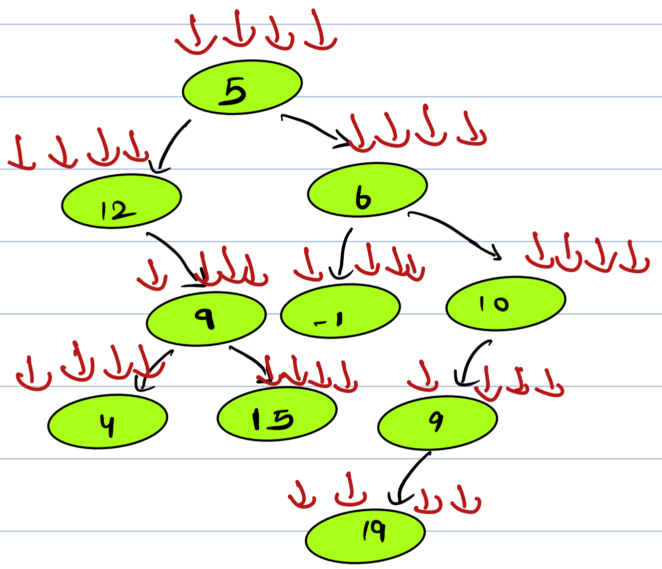
T.C → O(N)

* In a skewed Tree



→ Height = n.

2) Preorder :- Data Left Right



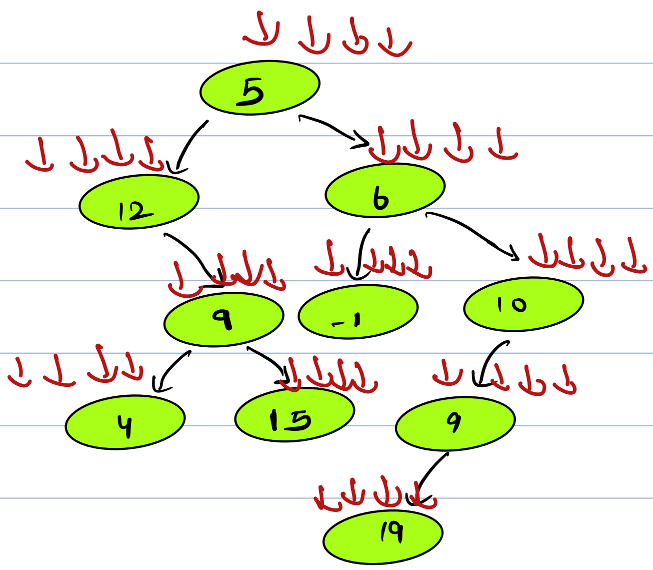
5 12 9 4 15 6 -1 10 9 19

```
void PreOrder (Node root) {
        if(root==null) {return}
        print (root.data);
        Preorder (root.left);
        Preorder (root.right);
}
```

T.C → O(N) , S.C → O(H).

↳ Dry Run → Todo.

3) PostOrder :- Left Right Data.

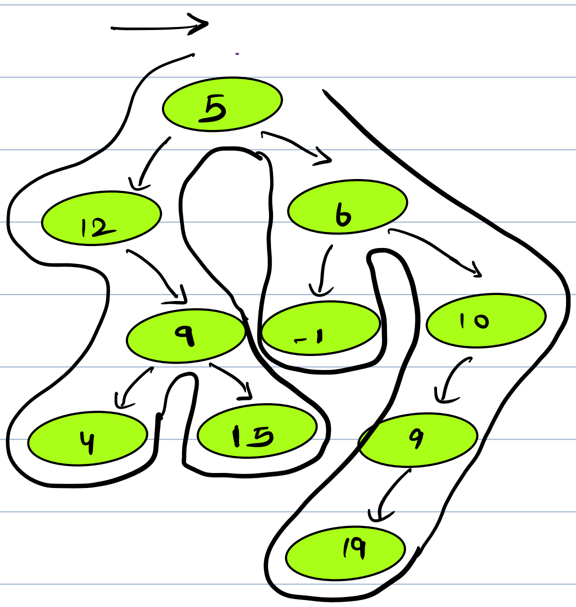


4 15 9 12 -1 19 9 10 6 5

```
void PostOrder (Node root) {
    if(root==null) {return}
    PostOrder (root.left);
    PostOrder (root.right);
    print (root.data);
}
```

$T.C \to O(N)$

$S.C \to O(H)$



Inorder :- LDR

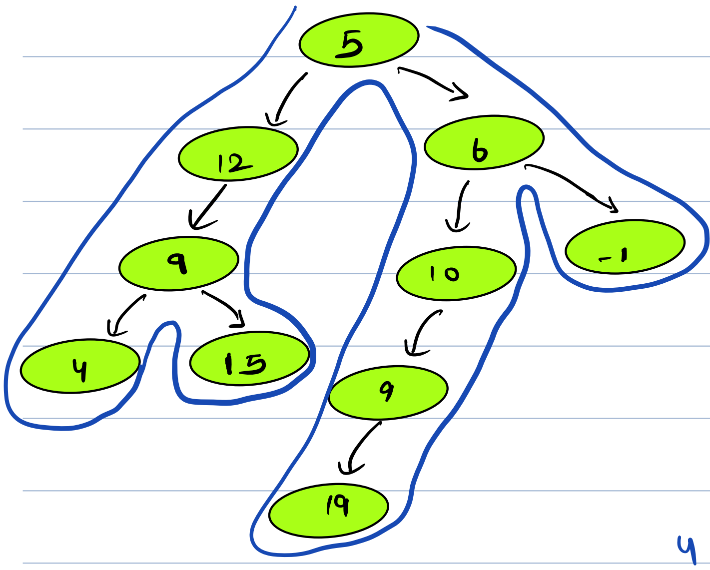
12 4 9 15 5 -1 6 19 9 10

Preorder :- DLR

5 12 9 4 15 6 -1 10 9 19

PostOrder :- LRD

4 15 9 12 -1 19 9 10 6 5



Inorder :-

4 9 15 12 5 19 9 10 6 -1

Preorder :-

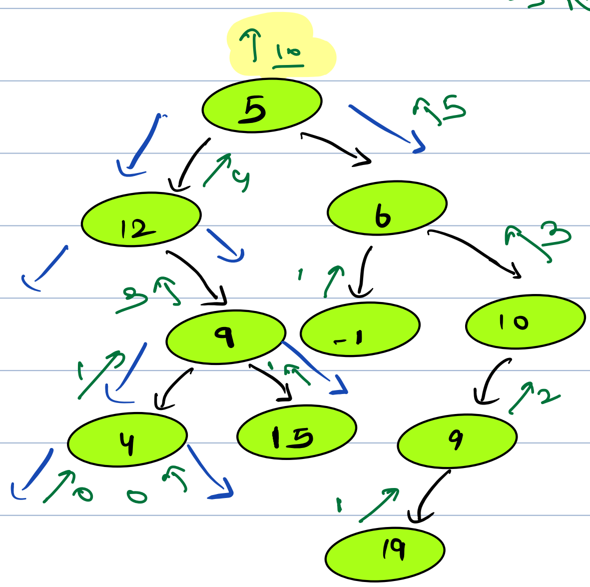
5 12 9 4 15 6 10 9 19 -1

PostOrder :-

4 15 9 12 19 9 10 -1 6 5

* Calculate size of tree.

↳ No. of nodes in Tree.



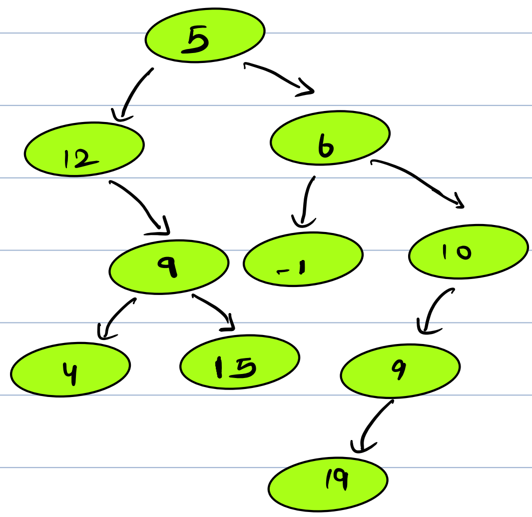
→ 10

T.C → O(N)

S.C → O(H).

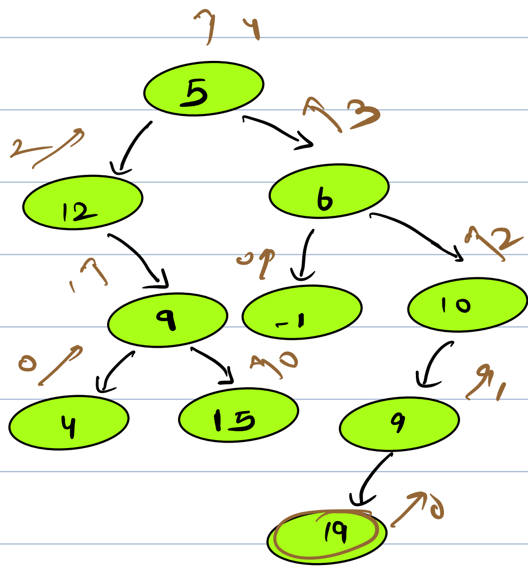
```
int size(node root) {
    if (root == null) { return 0 }
    int l = size(root.left);
    int r = size(root.right);
    return l + r + 1;
}
```

Ques) Sum of Nodes :-




```
int sum(node root) {
    if (root == null) { return 0 }
    int l = sum(root.left);
    int r = sum(root.right);
    return l + r + root.data;
}
```

Ques) Calculate height of tree :-




```
int height( node root) {
    if (root = = null) {return -1}
    int l = height(root. left);
    int r = height (root. right);
    return  Max(l, r) + 1
}
```

Uses of trees :-

Folder Structure

B-Tree, B+Tree → indexing in dbs.

Syntax Tree :-

Red Black Tree, AVL trees.

we'll have 1 P.S.

wednesday.

P.S → Sunday → 9 pm

6 pm or 5pm.