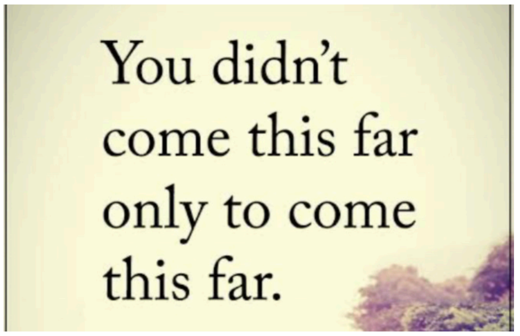
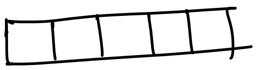


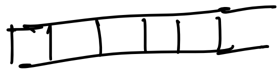
Today's Content

- → Trees intro
- → Naming Convention
- → Tree Traversal
- → Basic Tree problems.

Linear.

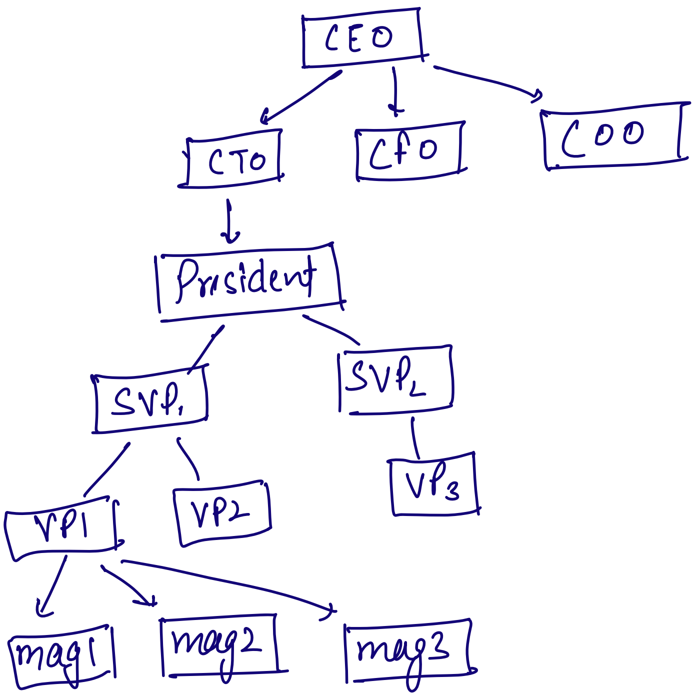
↳ arrays. , list<int> , stack & Queue.



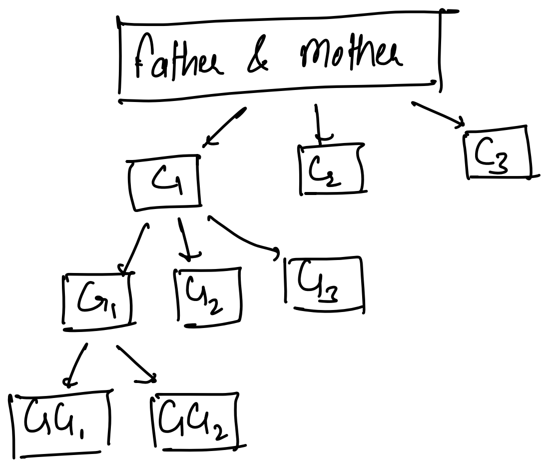


Linked-list.

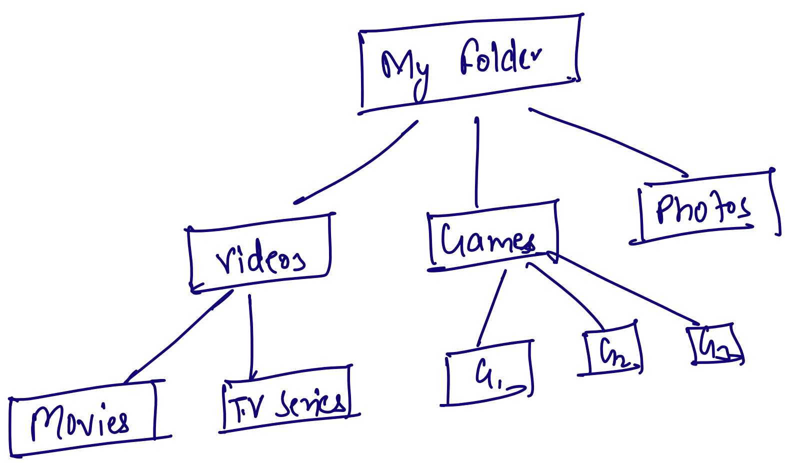
Hirarchical Data

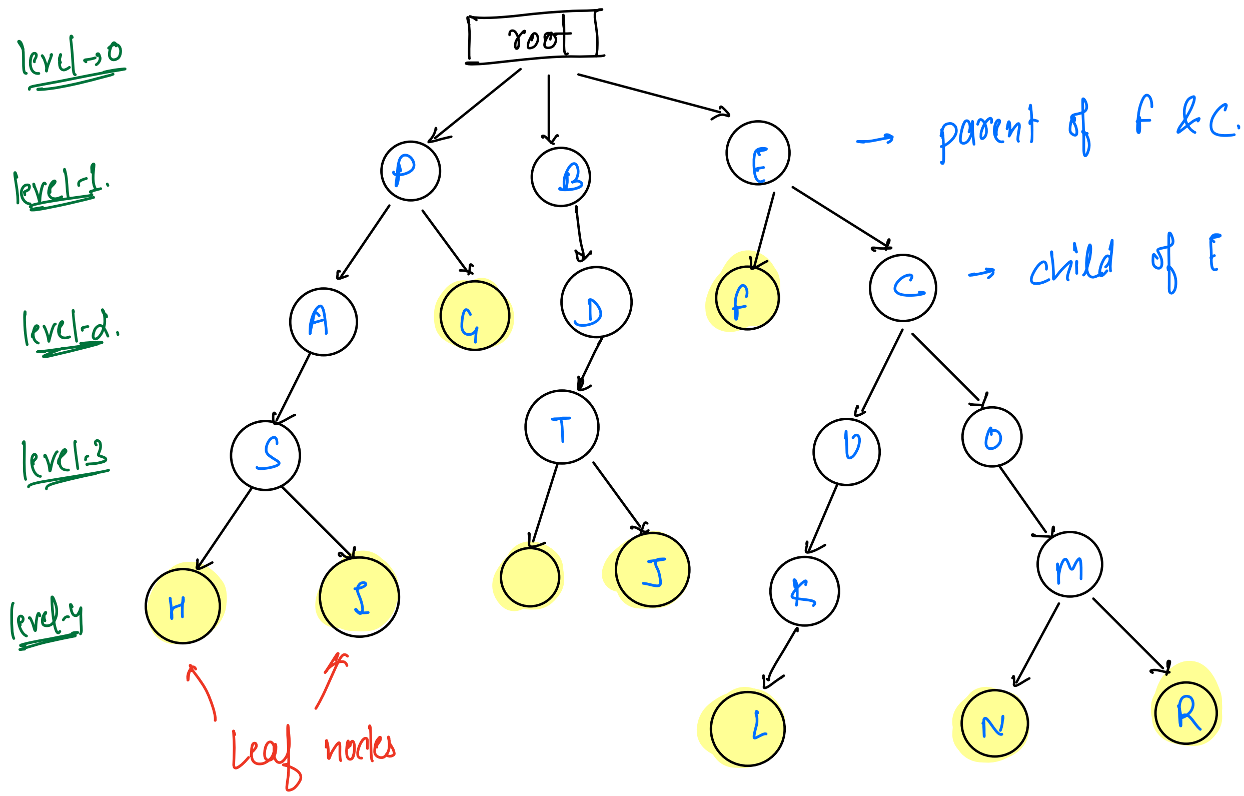


Family Tree.



Eg : Folder Structure





Naming.

$\left\{\begin{array}{l} E \to O \\ E \to M \\ E \to R \end{array}\right\}$: E is ancestor of O, M, R.

: O, M, R are descendents of E.

F, C : siblings

A, G, F, C : nodes at same level

leaf node : node with 0 no. of children.

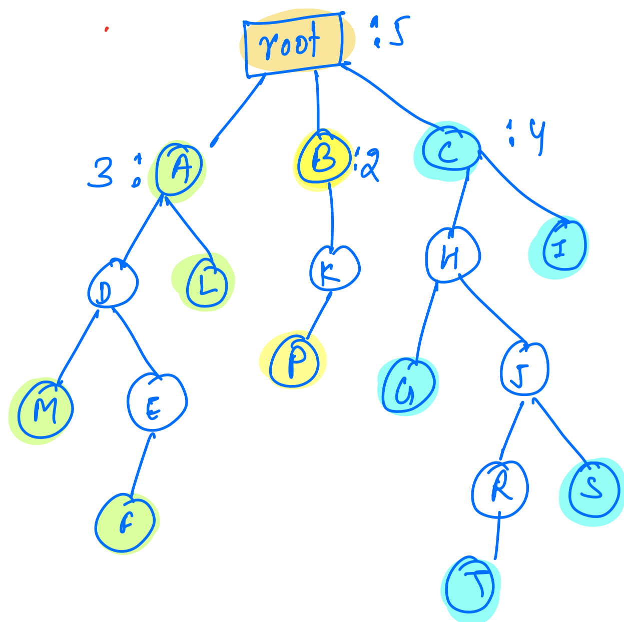
root node : node without parent node

Tree. : [→ We will have only 1 root node
→ For every node, there will be only 1 single parent.]

Height of a Node.

[length of the longest path from node to any of its descendent leaf nodes.]

** [length is defined in terms of no. of edges.]



Observation

① Ht. of any node = max(ht. of child nodes) + 1 .

② Ht. of leaf node = 0.

③ Ht. of tree = Ht. of root node.

Depth of a node.

length of path from root node to node.

$d(A) = d(B) = d(C) = 1.$

$d(F) = 4.$

Observation

① If depth of a node = d
depth of its child node = d+1.

② depth (root node) = 0

③ depth of a tree = Max depth of a leaf node

= no. of levels.

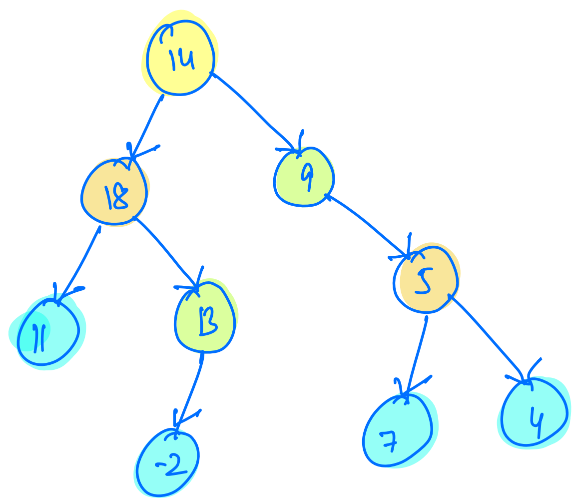
✓ [Height of a tree = Depth of a tree]

✗ [Height of a node = Depth of a node]

Binary Tree : Every node can have atmost 2 children.

0, 1, 2, ~~3, 4, 5, - -~~

Eg,



→ leaf nodes.

→ nodes with single child

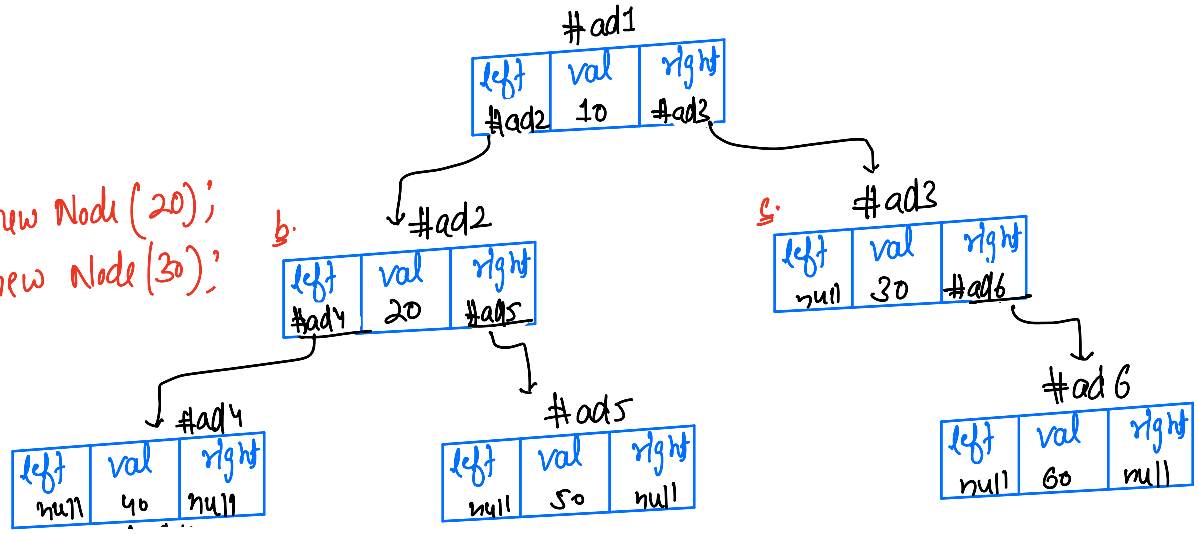
→ nodes with 2 children.

```
class Node {
    int val;
    Node left ; // object reference, it holds reference of Node object
    Node right ; // object reference, it holds reference of Node object
    Node(x) {
        val = x
        left = right = null
    }
}
```

Node r = new Node(10);

r.left = new Node(20);

r.right = new Node(30);



→ [Given the root node of a binary tree, we can traverse the entire tree.]

→ We will be given root node for every question.

→ Construction of tree using arrays uses serialization & de-serialization. {adv.}

Tree traversals.

→ Pre-order

→ In-order

→ Post-order

→ level-order

→ vertical level-order

→ Diagonal -order

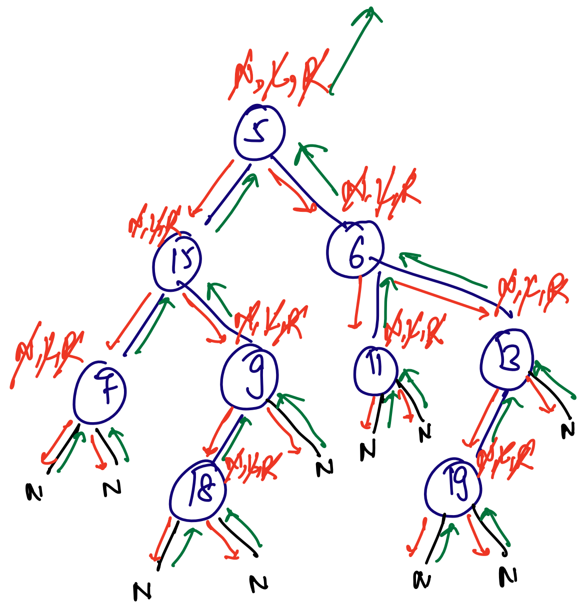
Adv.

Pre-order → N, L, R.

Step 1. print (root.val)

Step 2. Go to left-subtree and print entire left subtree in pre-order.

Step 3 Go to right-subtree and print entire right subtree in pre-order.



O/P → 5, 15, 7, 9, 18, 6, 11, 13, 19

In-order → L, N, R

Post-order → L, R, N

{# todo}

[**Assigments** → use recursion]

Pseudo-code.

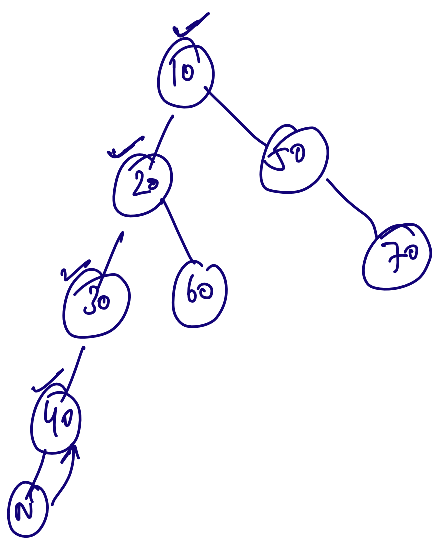
Assm: Given root node, it should print all nodes of tree in pre-order

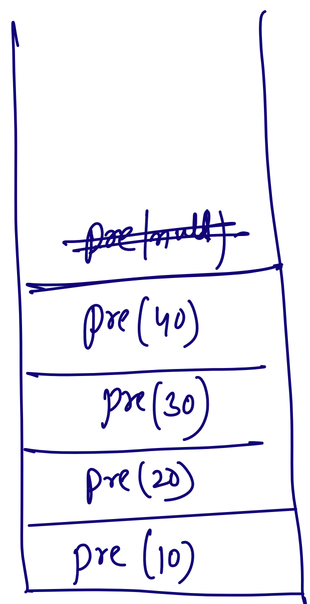
```
void pre-Order ( Node r ) {
    if ( r == NULL ) { return }
        print ( r.val );
        pre-Order ( r.left );
        pre-Order ( r.right );
}
```

T.C → O(N)

S.C → max size of stack at any given point of time

→ O(height of tree)



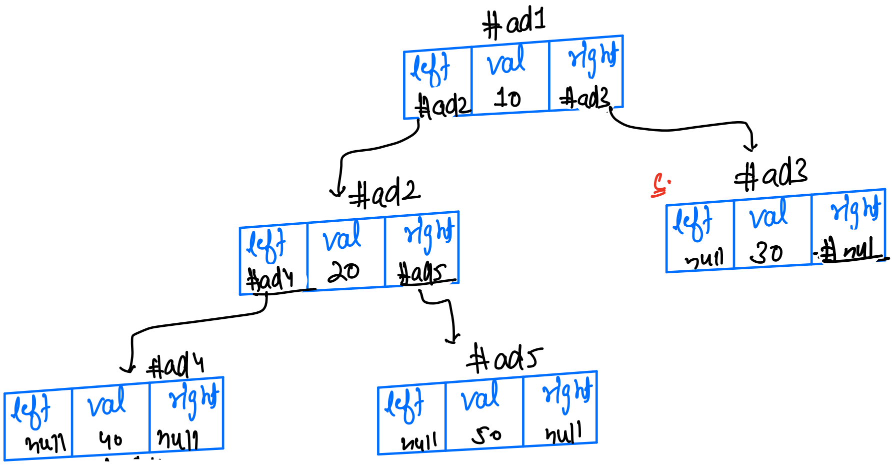


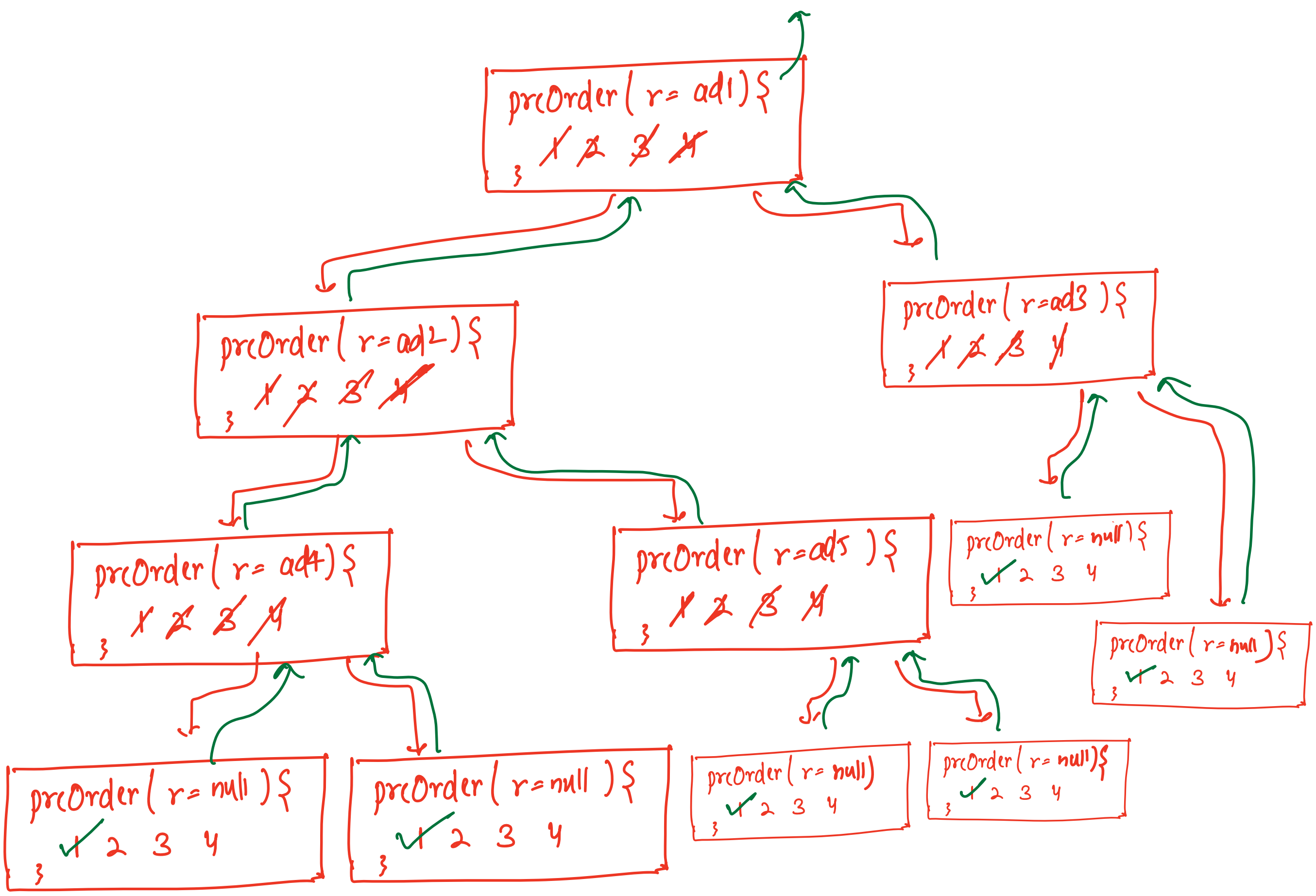
y. S-

Tracing.

```
void pre-Order ( Node r ) {
① if ( r == NULL ) { return }
②   print ( r.val );
③   pre-Order ( r.left );
④   pre-Order ( r.right );
}
```

{ o/p → 10, 20, 40, 50, 30 }



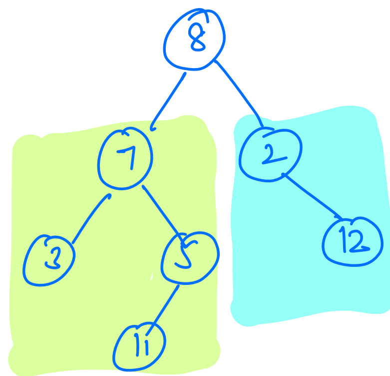


Tree problems.

1. Size (Node root)
2. Sum (Node root)
3. Height (Node root)

Recursive codes only without any Global variables.

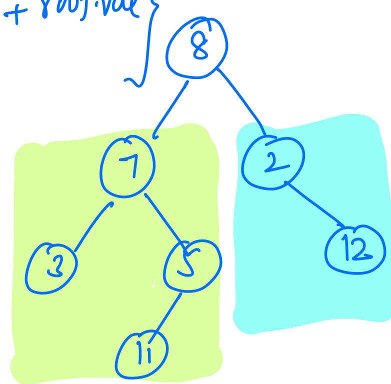
```
int size (Node root) { //Asm → Give root, return count of all nodes in tree.
    if (root == null) {return 0}
    l = size ( root.left);
    r = size ( root.right);
    return l + r + 1 ;
}
```



{size of tree = size of l.s.t + size of r.s.t + 1. }

//Asm → Given root, calculate & return sum of values of all the nodes.

{Sum of all nodes in tree = Sum of nodes in l.s.t + Sum of nodes in r.s.t + root.val}

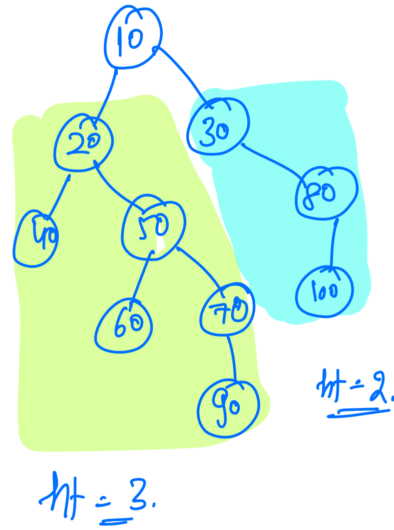


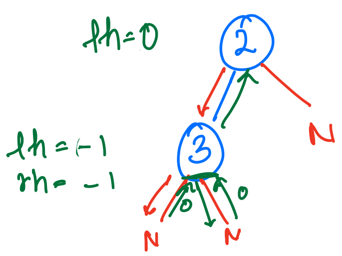

```
int sum( Node root) {
    if (root == null) { return 0 }
    int l = sum( root.left)
    int r = sum( root.right)
    return l+r+ root.val ;
}
```

③ Height

ht of tree = Max(ht of l.s.t, ht of r.s.t) + 1

```
int height ( Node root) {
    if (root == null) {return -1}
    int lh = height( root.left)
    int rh = height (root.right)
    return Max(lh, rh) + 1 ;
}
```





① if you want to calculate height of tree in terms of edges.
in base condition return -1

② if you want to calculate height of tree in terms of nodes,
in base condition return 0 .