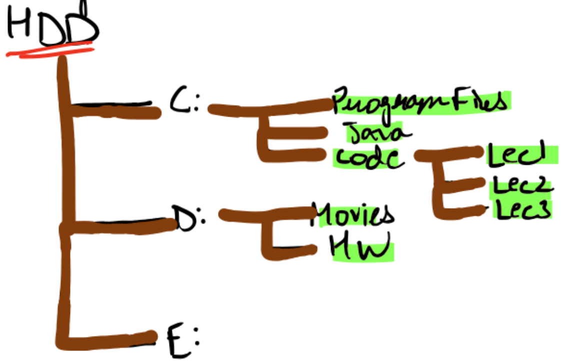
Binary Trees

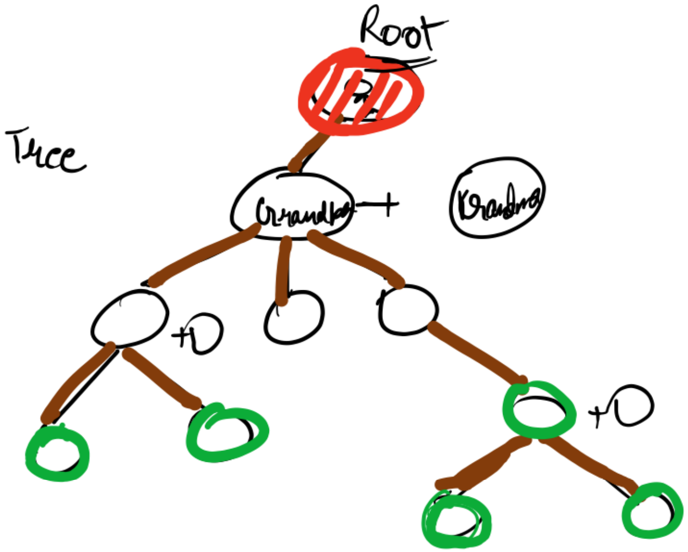
Wednesday, 31 July 2024 :50 PM

File System

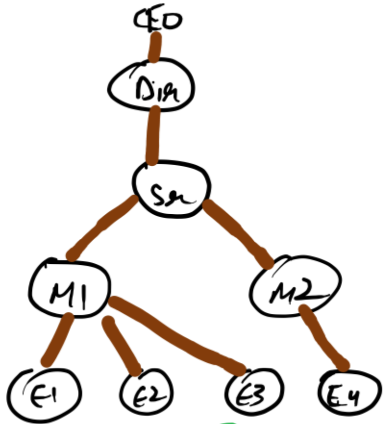


→ root
→ branches
→ leaves

Family Tree

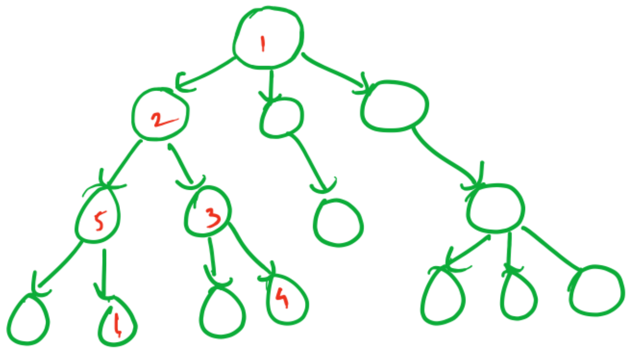


Organisation Structure



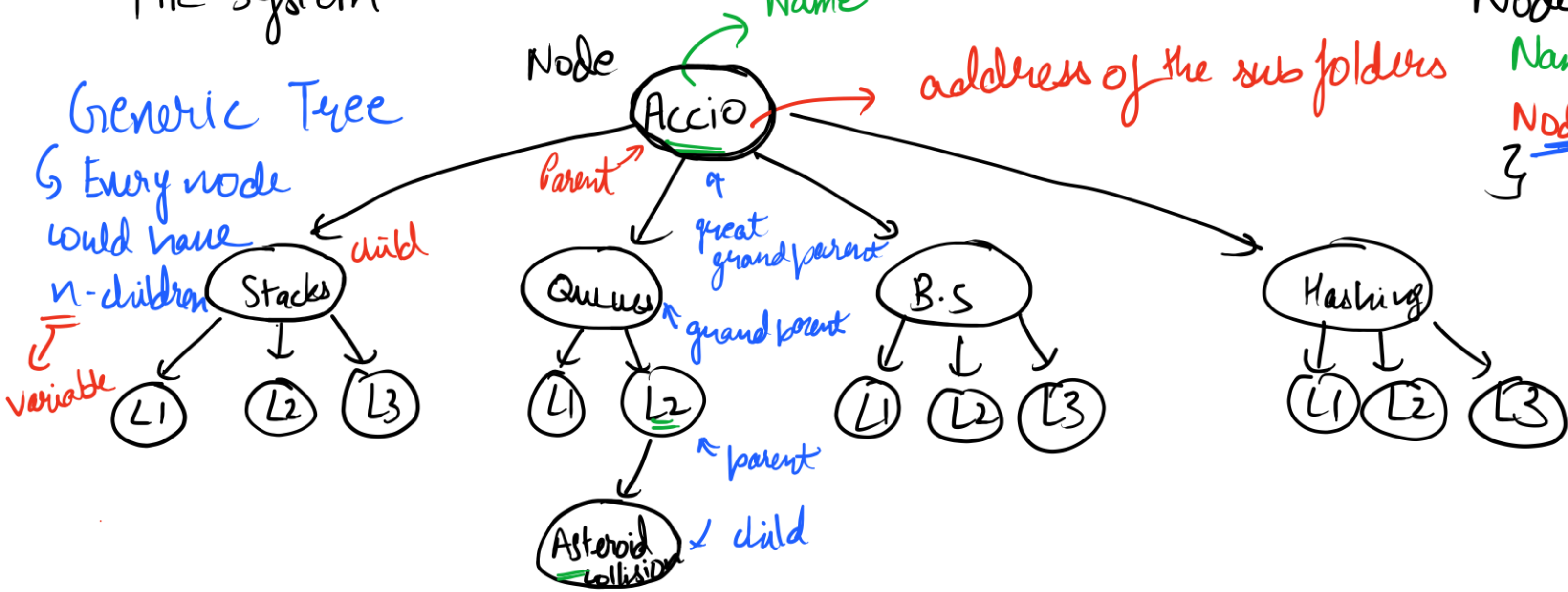
Tree

↳ Non Linear Data Structure



LL → Linear Data Structure

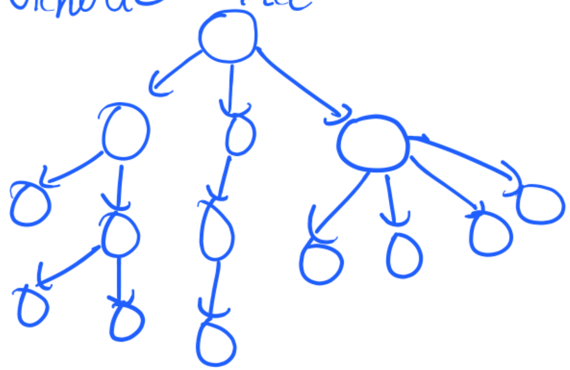
File System



Node{
Name/dat
Node[] chi
}

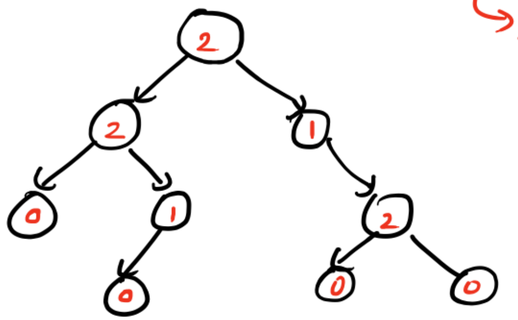

```
Node {
  data
  Node next
}
```

Generic Tree



Binary Tree {Atmost 2 children}

↳ {0, 1, 2}




```
Node{
  int data;
  Node left;  } child
  Node right; }
}
```

Binary Tree {Atmost 2 children}

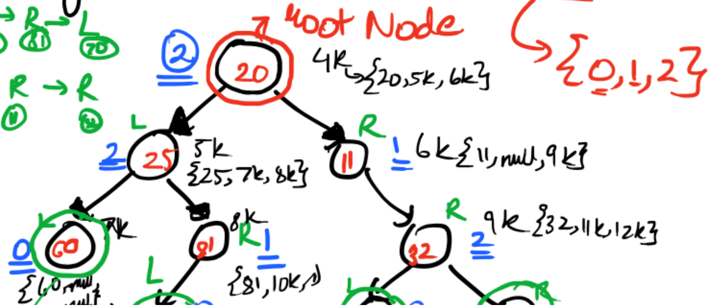
↳ {0, 1, 2}

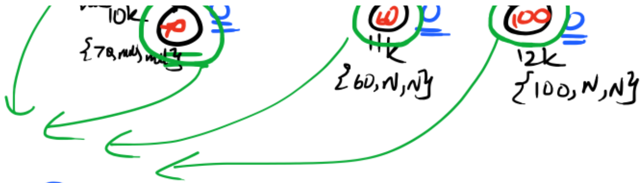
Named

Left   Right

70 : root → L → R → L

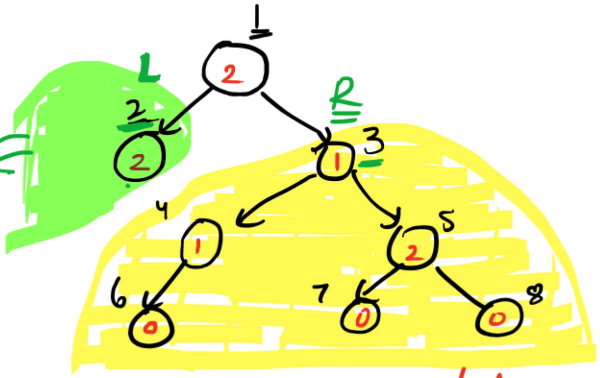
32 : root → R → R





degree : 0
↳ Nodes with 0 children
⇒ Leaf Node
↳ Not having any child

degree : No. of child per every node



Left Subtree

Leaf ⇒ 2, 6, 7, 8 ✓

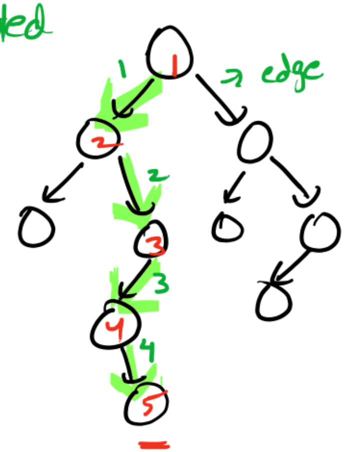
Subtrees ✓

Right Subtree
↳ Tree formed by right child and subsequent children.

Height of the Binary Tree

{ Distance b/w root node and the deepest leaf node }

Usually denoted by edges

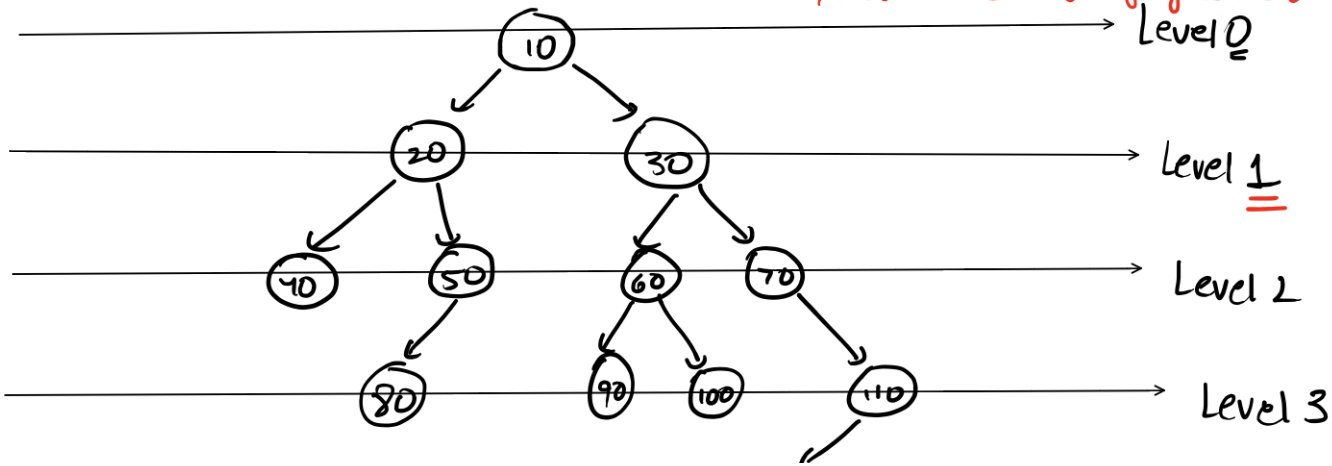


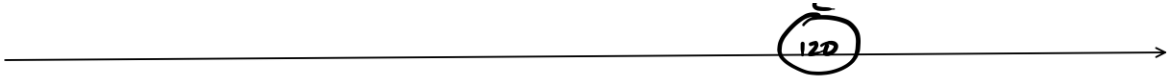
★ Ht of the tree :
↳ 4 { in terms of edges }

Ht of the tree { in terms of Nodes }
↳ 5

Levels of a Binary Tree

→ Level X contains all the nodes X distance away from a node





Level 4

Perfect Binary Tree

{ where no. of nodes at each level (l) = 2^l } ✓

level 0

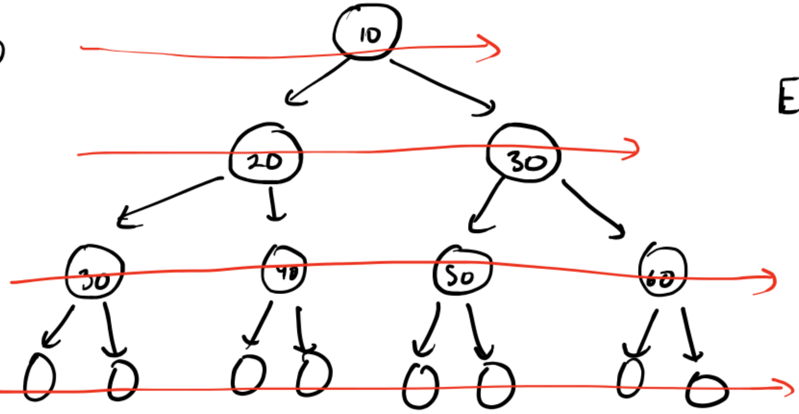
Level 1 ⇒ 2^1

Level 2 ⇒ 2^2

Level 3 ⇒ 2^3

⋮

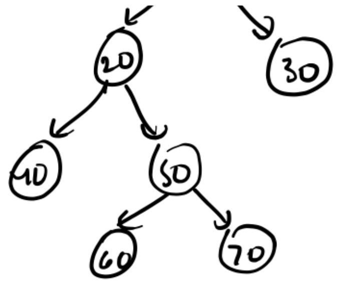
Level k ⇒ 2^k



Every node other than leaf node is having 2 children.

Full Binary Tree

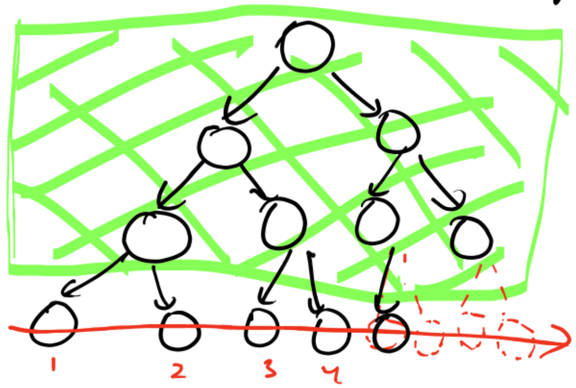
↳ where each node is having 0 or 2 children



Complete Binary Tree

↳ { where every level other than last level is completely filled and the nodes are left position in last level }

$2^3 \Rightarrow 8$



2^1

✓ Complete Binary Tree

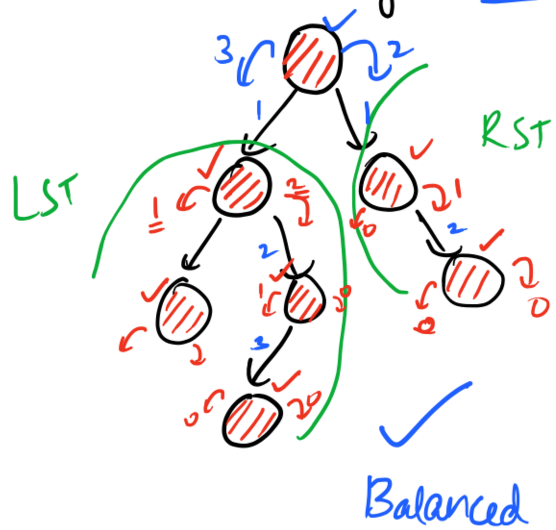
Order of filling in last level

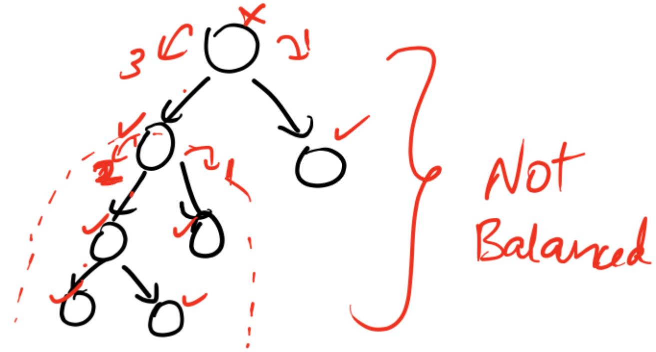
Balanced Binary Tree

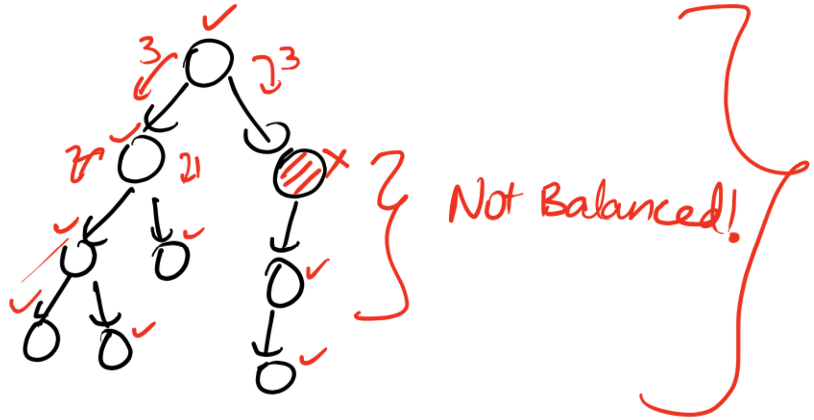
A tree in which every node is balanced

Balanced Note

$$|\text{left subtree} - \text{right subtree}| \leq 1$$

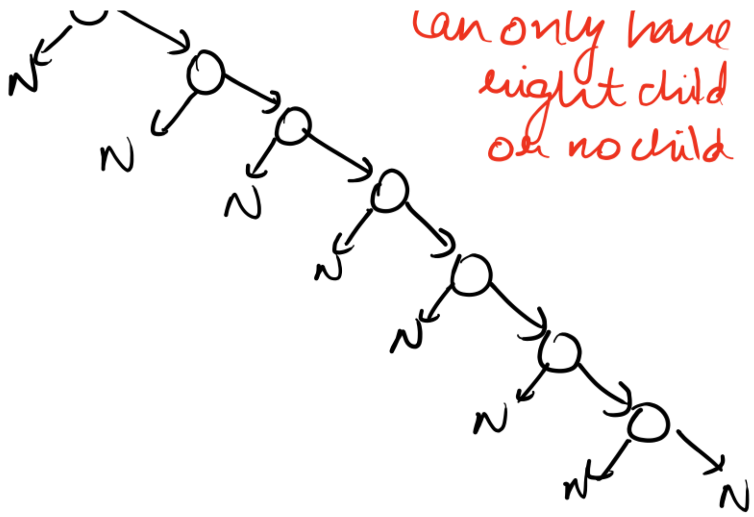






Skew Tree →

① Left Skewed Tree | ② Right Skewed Tree



* Tree Traversals

↳ Ways in which we can traverse over a tree

DFS

↳ recursion and go in depth

- → PreOrder Traversal
- → InOrder Traversal
- → Post Order Traversal

BFS

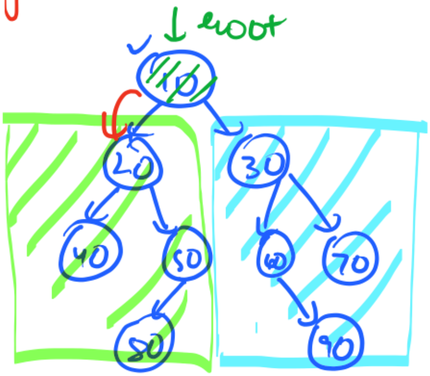
↳ queue and traverse level by level

- → level order traversal

Pre Order Traversal

- Node ✓
- Left Subtree
- Right Subtree

Print each node following pre order traversal



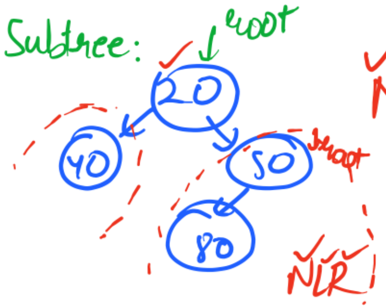
NLR

O/P:

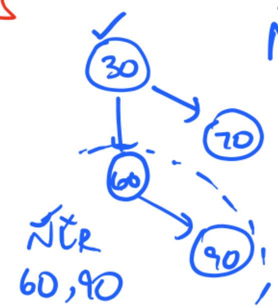
10, 20,40,50,80, 30,60,90,70

→ {Recursion}

Left Subtree:



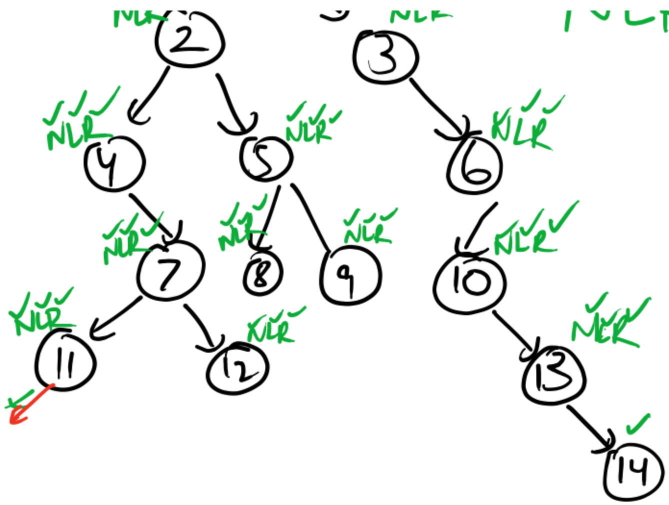
NLR : 20,40,50,80



NLR : 30,60,90,70

Pre Order Traversal

NLR : 1

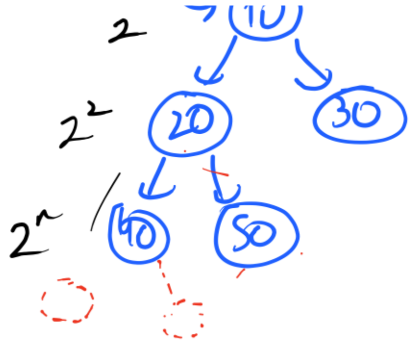


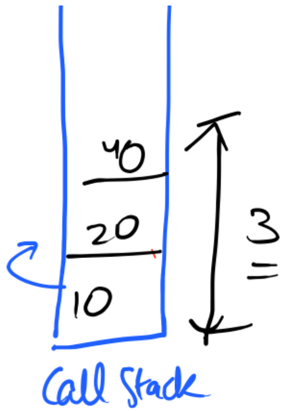
NLR → ,2,4,7,11,12,5,8,9,

3, 6, 10, 13, 14

```
void preOrderTraversal (Node root) {        //NLR
    if( root == null){ return;}             //Base case

    print ( root.data);
    preOrderTraversal (root.left);
                       └──> left subtree

    preOrderTraversal (root.right);
                       └──> right subtree
}
```

```java
class Solution {
  // preorderTraversal -> N L R
  // faith: preorderTraversal will print the ans for LST and RST
    public static void preorderTraversal(Node root) {
      if(root == null){ // Base case
        return;
      }

      System.out.print(root.data+" "); // N is complete

      preorderTraversal(root.left); // LST Answer is provided by recursion

      preorderTraversal(root.right); // RST Answer is provided by recursion
    }
}
```





O/P : 10 20 40 50 30

TC: O(N) ↪ No. of Nodes

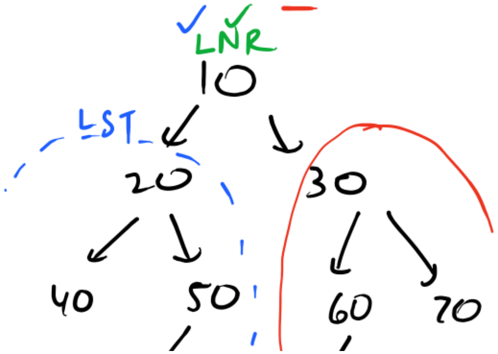
SC: O(H) ↪ Ht of the tree

$2 + 2^2 + 2^3 + \ldots + 2^{H+1}$

$\approx O(2^H)$ → He

In Order Traversal (L N R)

→ LST
Node
RST



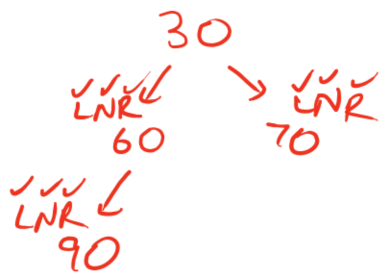
InOrder : LNR

$\boxed{40, 20, 80, 50}$, 10, $\boxed{90, 60, 30, 7}$

LST Root RST

LNR

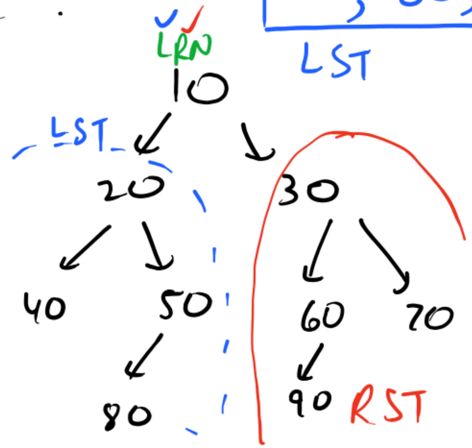
40, 20, 80, 50



90, 60, 30, 70

Post Order Traversal :

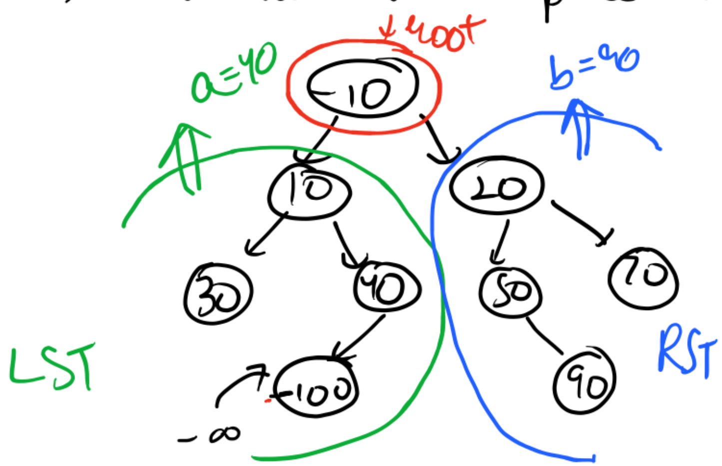
L.S.T
R.S.T } LRN
Node



$\boxed{40, 80, 50, 20}_{LST}$ $\boxed{90, 60, 70, 30}_{RST}$, $\underset{\text{Node}}{10}$

40, 80, 50, 20 10, 60, 70, 30

1) Maximum value present in a tree



recⁿ will find the max in subtree

```
int maxofTree( Node root){
LST→ int a = maxofTree(root.left)
RST→ int b = maxofTree(root.right);
    int curr = root.data;
    return Max{a, b, curr}
}
```

Max{root.data, a, b}

a → Max of L.ST

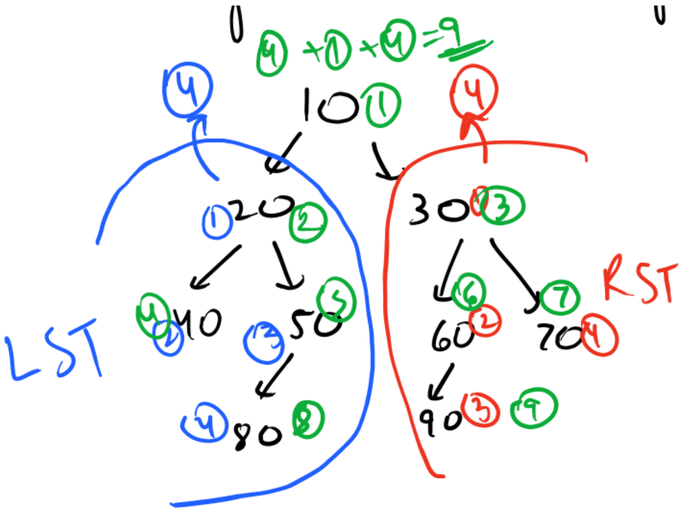
b → Max R.ST

L R N ⇒ Post Order Traversal

TC: O(N) → No. of nodes

SC: O(H) → Ht of tree

2) Size of a tree {No. of nodes in a Tree}




```
sizeofTree(Node node){
if(node==null) => return 0;
 int LST -> size(node.left);
 int RST -> size(node.right);
 return LST + RST + 1;
}
```

curr.node