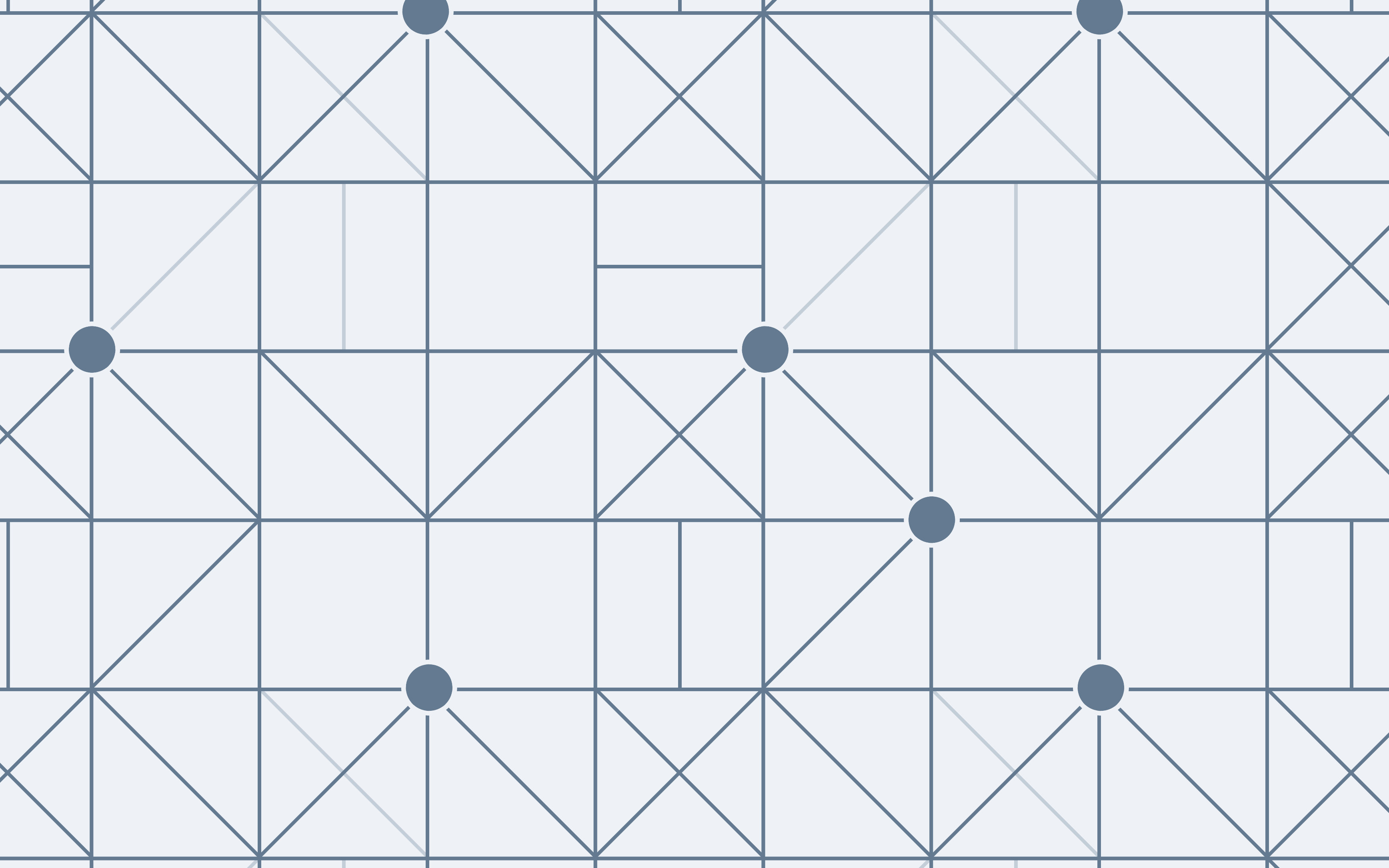
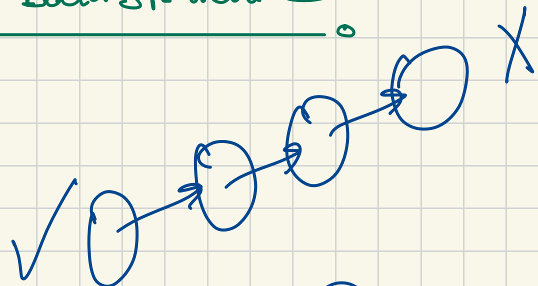


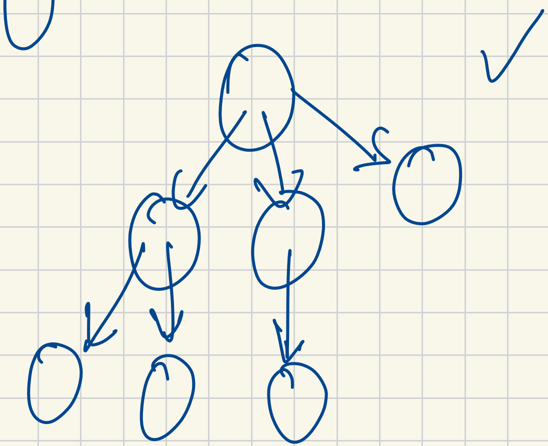
Binary Trees.

↳ Non Linear Data Structure.



data.

- ↳ Org Structure
- ↳ file system
- ↳ family tree



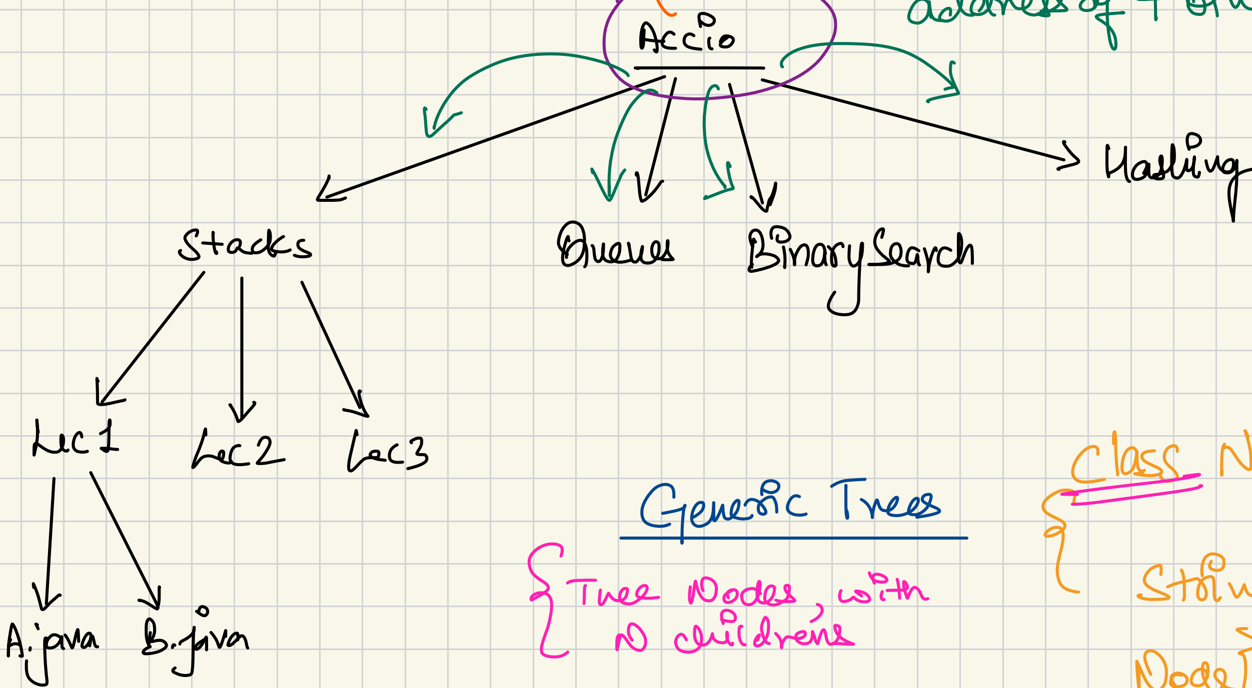
Heirarchy.

file system

String Name;

Tree Data Structure

address of 4 other files {Node[] child}



Generic Trees

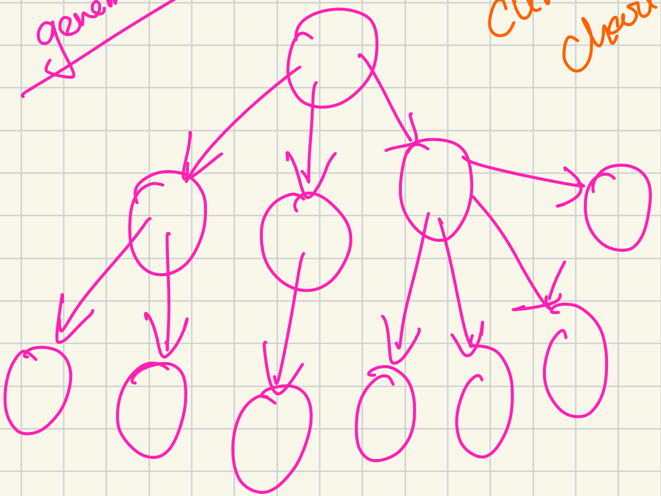
{Tree Nodes with 0 childrens}

```
class Node
{
    String data;
    Node[] child;
}
```

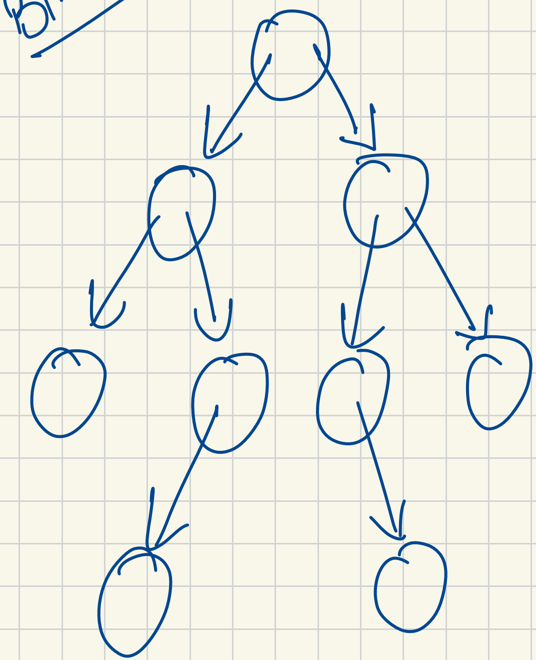
Binary Trees {Atmost 2 children}

{Current population chart}

generic tree



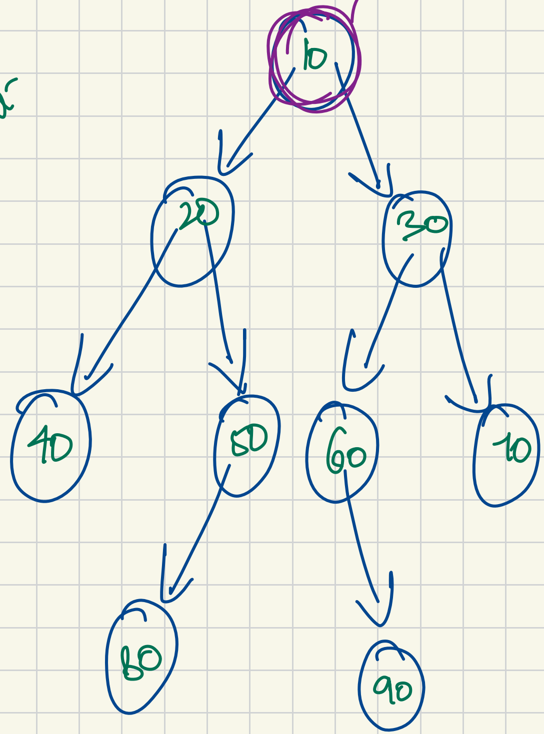
Binary Tree



Binary Tree.

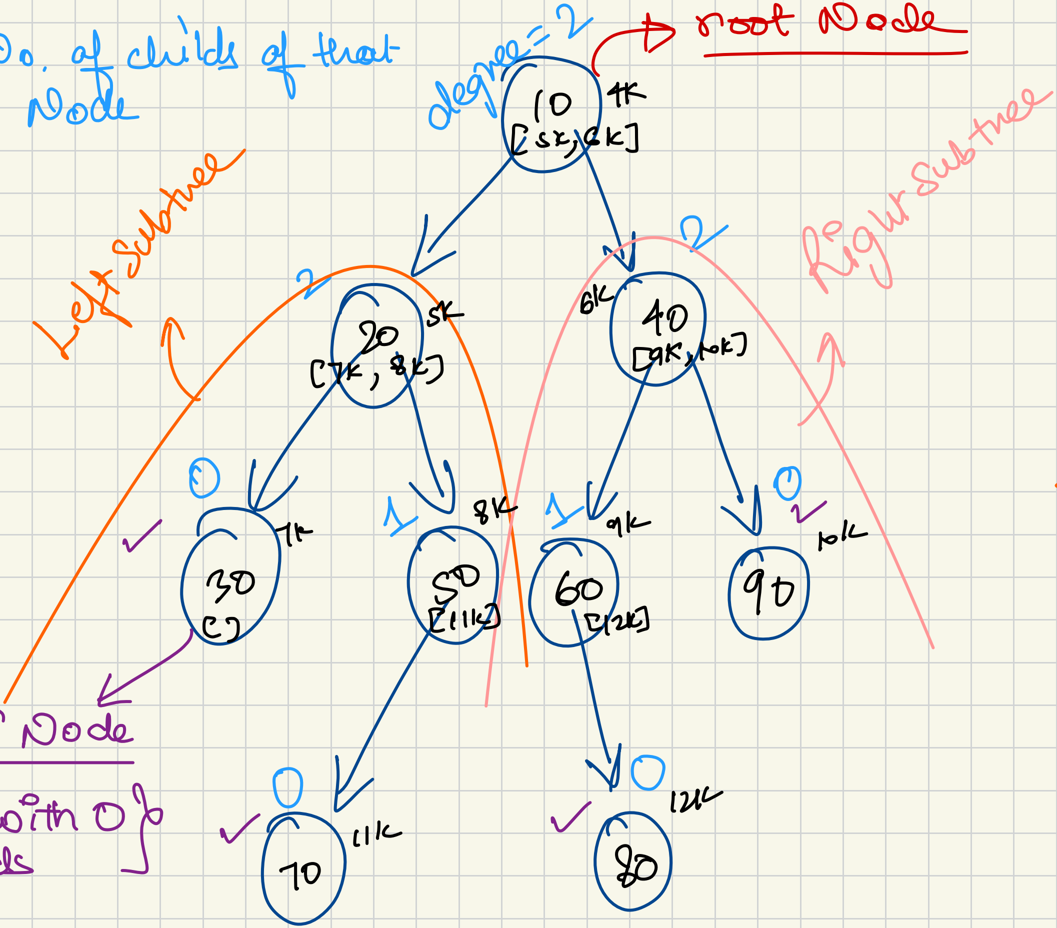
Atmost 2 childs

→ Named as Left & Right




```
class Node
{
    int data;
    Node Left;
    Node right;
}
```

degree: No. of childs of that Node



leaf Node

{Nodes with 0 childs}

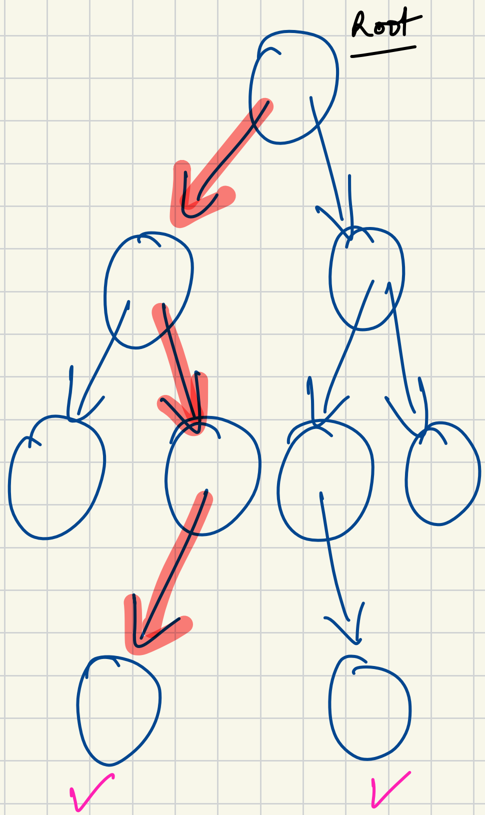
Siblings

20, 40
30, 50
60, 90

Subtrees

Height of the Binary Tree

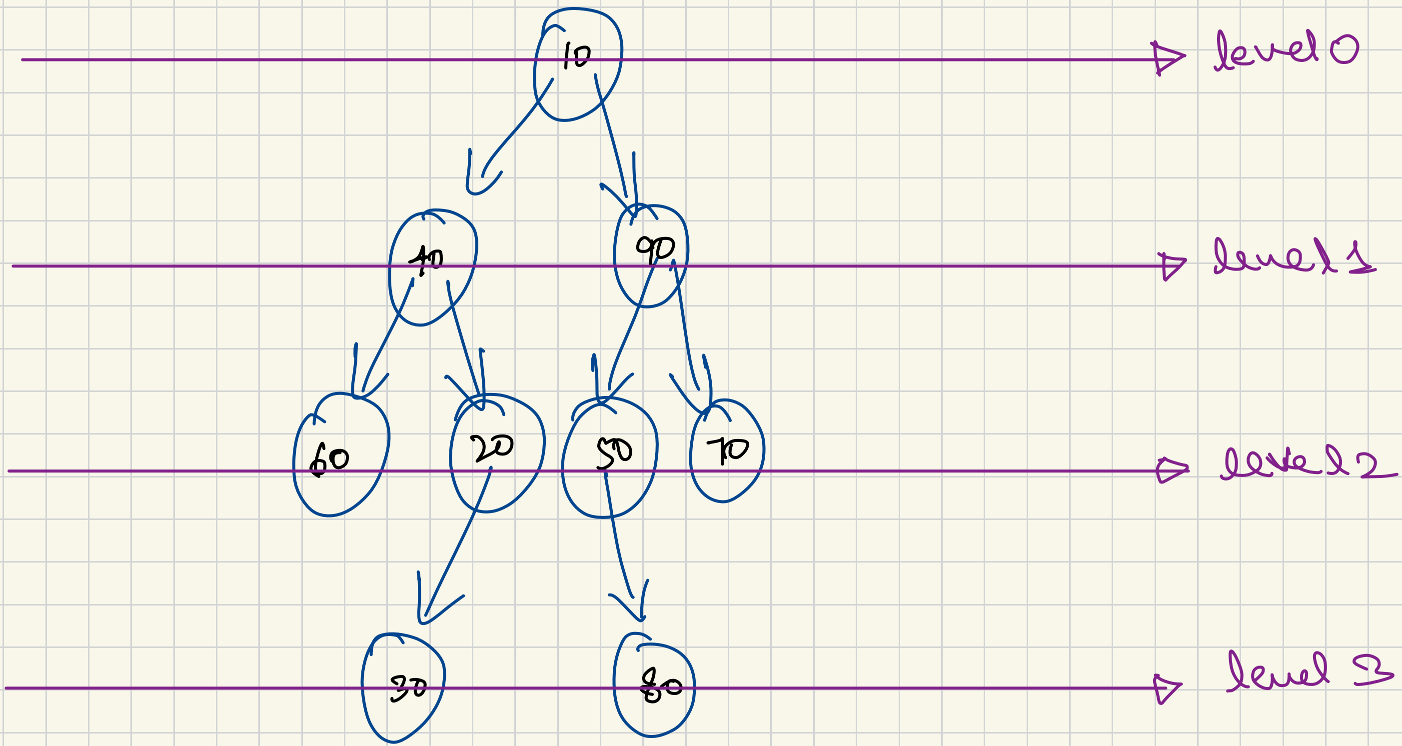
{Distance B/w Root Node & Deepest Leaf Node}



Height = 3 {in terms of Edges}

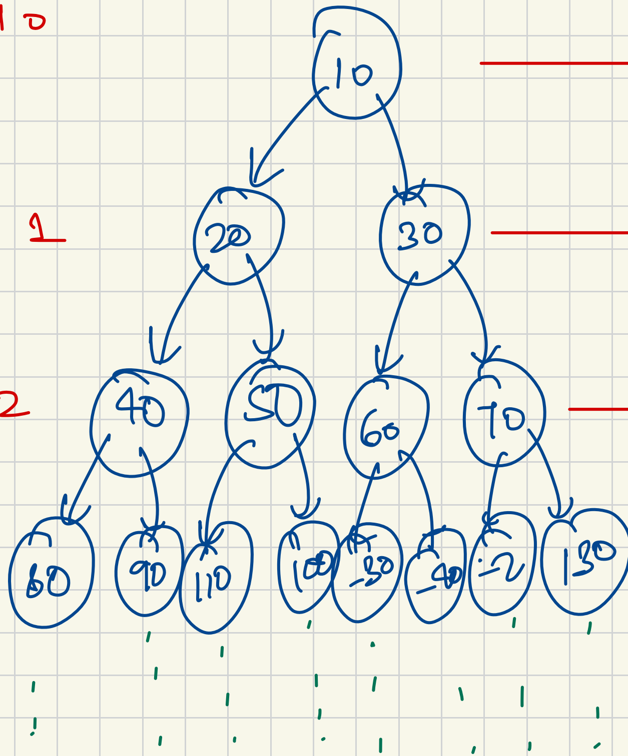
Height = 4 {in terms of Nodes}

Levels in a Binary Tree.



Perfect Binary Tree.

{where no. of people at each level (l) $= 2^l$}



level 0 → 1 → 2^0

level 1 → 2 → 2^1

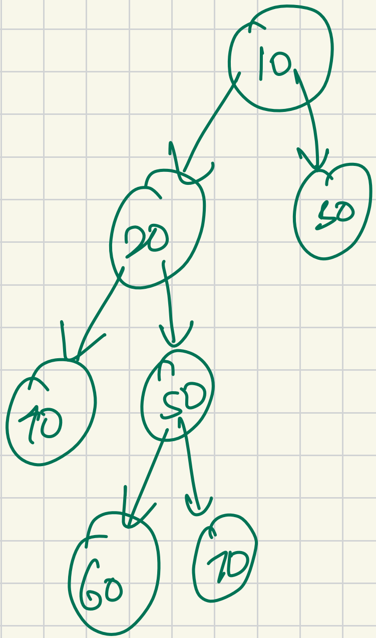
level 2 → 4 → 2^2

level 3 → 8 → 2^3

level K → 2^K

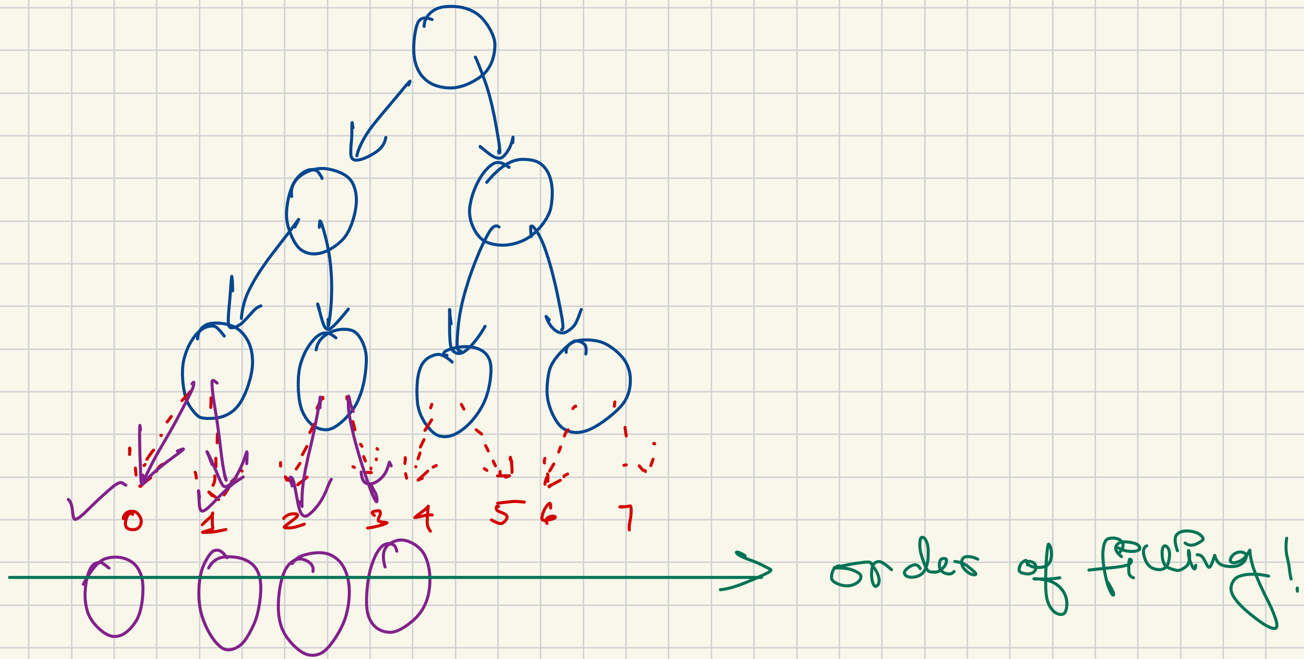
Full Binary Tree

↳ Where each node have either 0 or 2 children



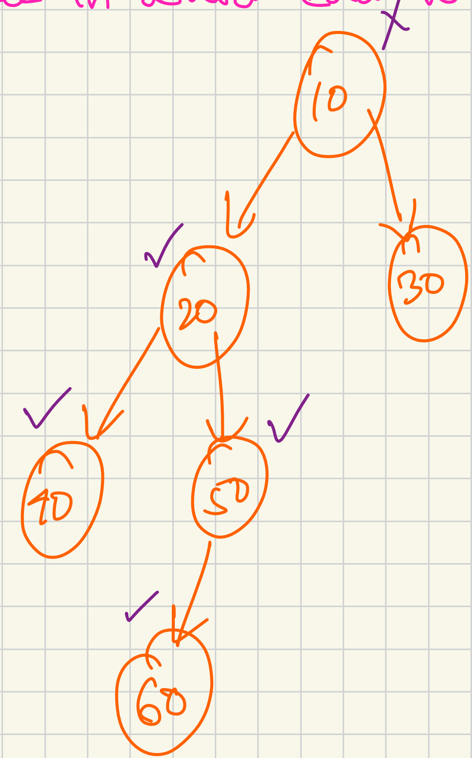
Complete Binary Tree.

where each level is completely filled, except last level; where nodes are as left positioned as possible.



Balanced Binary Trees

A tree in which each node is balanced!



Not Balanced!

Balanced Node

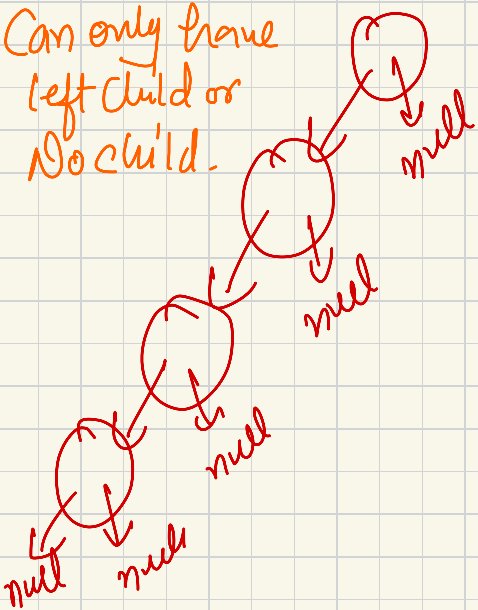
$$|\text{LST height} - \text{RST height}| \leq 1$$

{in terms of Nodes}

Skew tree

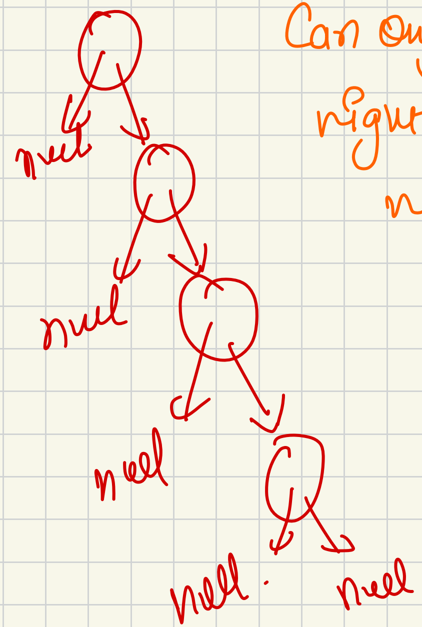
① Left Skew

Can only have left child or No child.



② Right Skew

Can only have right child or no child.



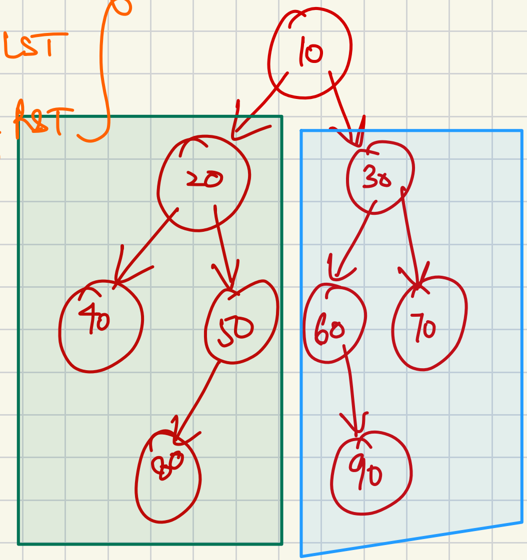
Traversal over Tree

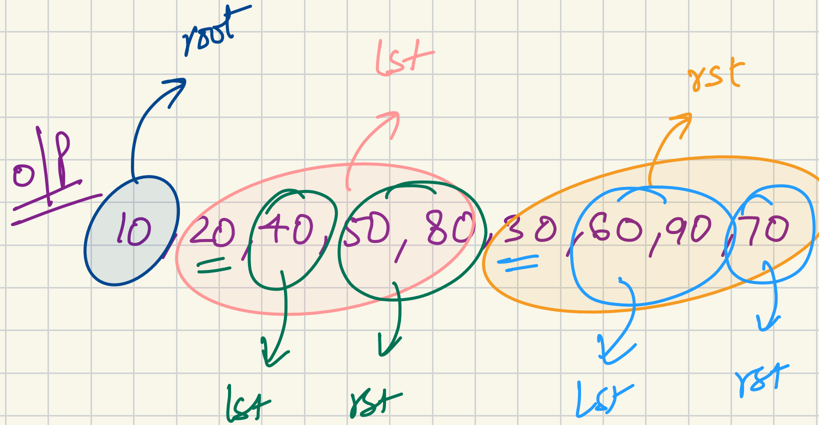
- Pre Order Traversal

~~print~~ root

~~pre~~ order of LST

~~pre~~ order of RST



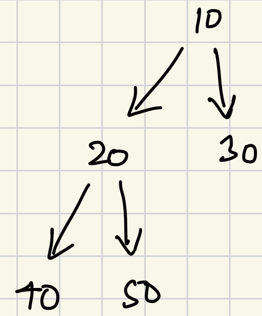


{ Recursion }

faith: print pre-order of the tree from the root

```
void printPreOrder(Node root)
{
    if (root == null) return;

    print(root.data);

    printPreOrder(root.left);

    printPreOrder(root.right);
}
```

```java
// PreOrder Traversal -> root, preorder (LST), preorder (RST)
// Faith: prints preorder traversal from the given root
public static void preorderTraversal(Node root) {
    // base case
(1) if (root == null) {
        return;
    }

    // print root's data
(2) System.out.print(root.data + " ");

    // print the preorder of left subtree
(3) preorderTraversal(root.left);

    // print the preorder of right subtree
(4) preorderTraversal(root.right);
}
```



o/p

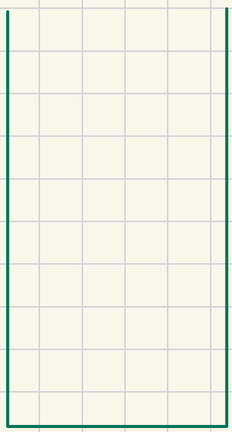
10, 20, 40, 50, 30

No. of Nodes

TC : O(N)

SC : O(H)

Height of tree

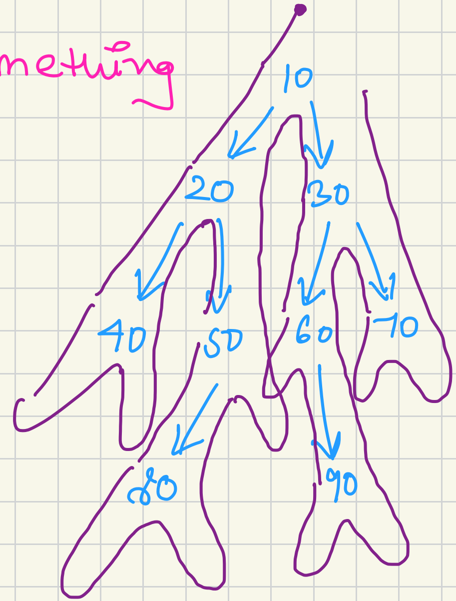


callstack

o/p 10 20 40 50 30

Eular's Path

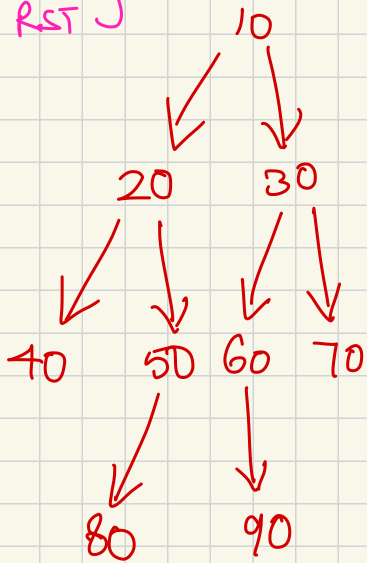
whenever see something new print it



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

In-order traversal.

→ inorder LST
→ root
→ inorder RST

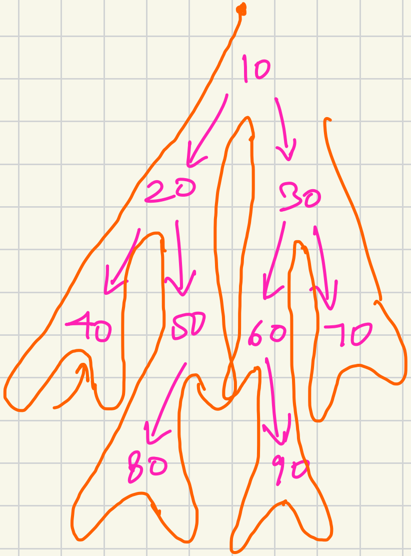


o/p

40 20 80 50 | 10 | 60 90 30 70

LST — root — RST

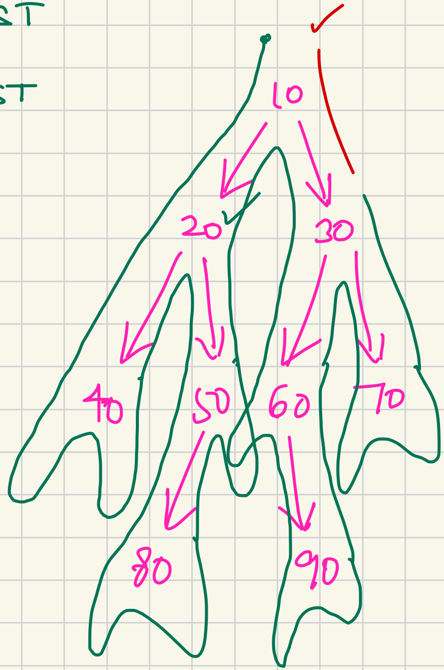
Euler Path



40 20 80 50 10 60 90 30 70

Post Order Traversal

- PostOrder LST
- PostOrder RST
- root



o/p

40 80 50 20 90 60 70 30 10

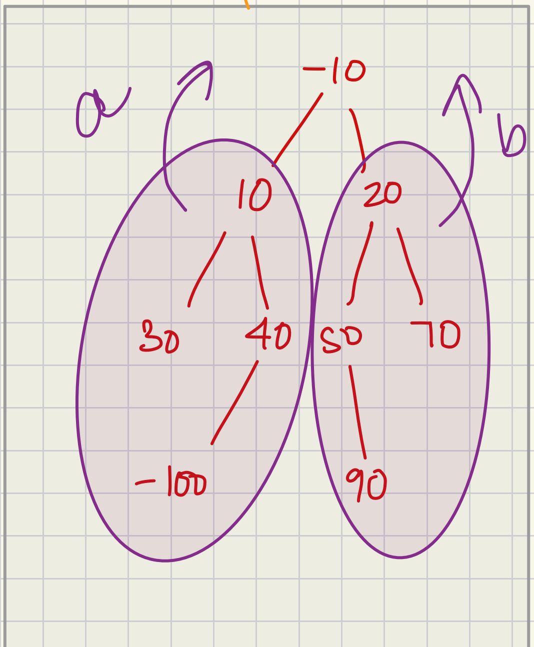
LST

RST

Root

Max^m in a tree. {Max^m value present in a tree}

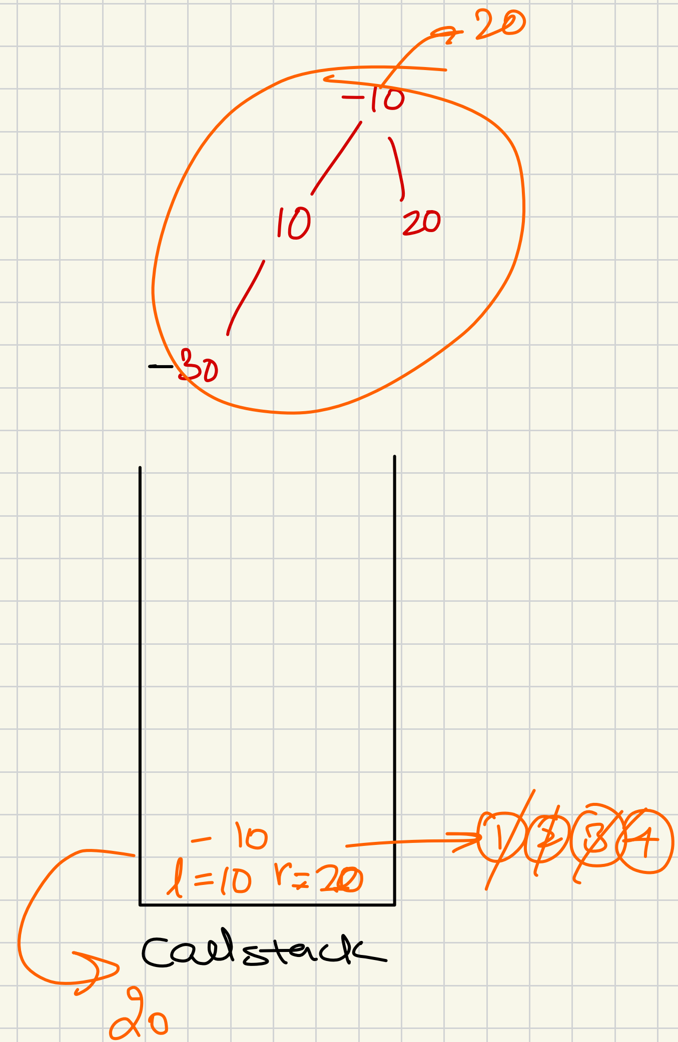
max^m {a, b, root}



faith: returns max^m value of tree {starting from root}

```
int maxOfTree(Node root)
{
    if (root == null) return MINValue;

    int a = maxOfTree(root.left);
    int b = maxOfTree(root.right);
    return max{a, b, root.data};
}
```

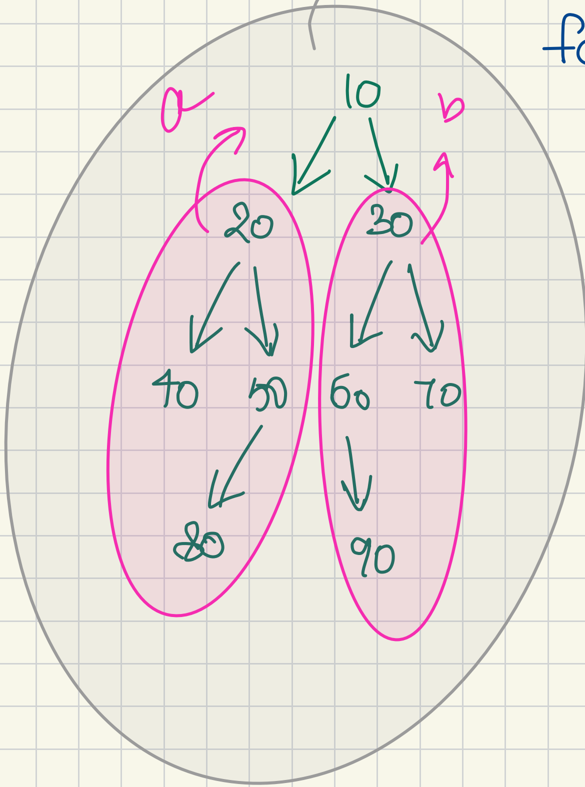
```java
// Faith: return maximum value of the tree starting from the root
public int maxOfTree(Node root) {
    // base case
    if (root == null) {
1       return Integer.MIN_VALUE;
    }

    // get maximum value in left subtree
2   int leftMax = maxOfTree(root.left);

    // get maximum value in right subtree
3   int rightMax = maxOfTree(root.right);

    // maximum value of the tree
    // Max of {leftMax, rightMax, root.data}
4   return Math.max(root.data, Math.max(leftMax, rightMax));
}
```



size of the tree

{No. of Nodes in a tree}

$\underline{\underline{a+1+b}}$



faith: returns size of the tree starting from root

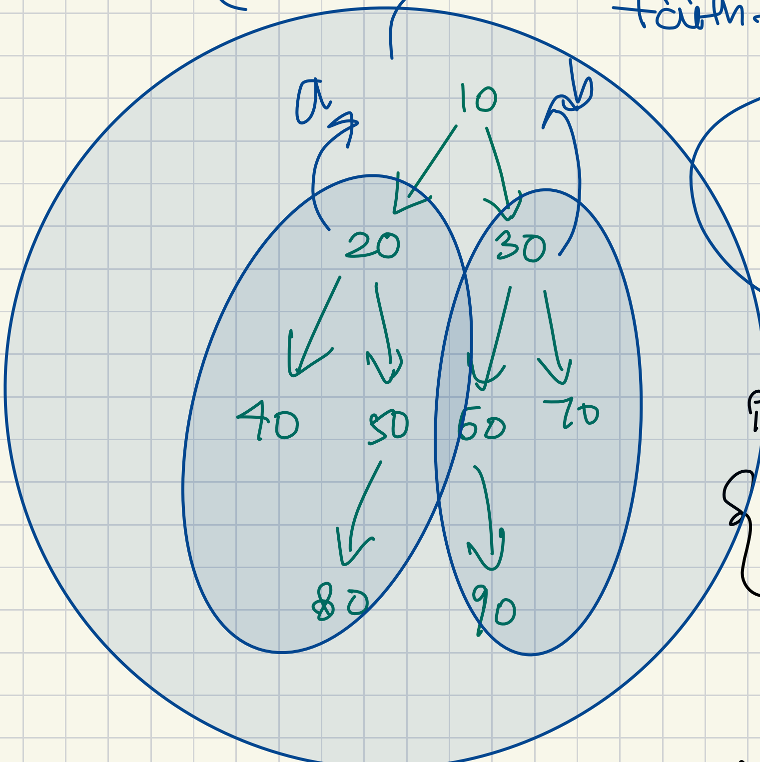
```
int size(Node root)
{

}
```

Sum of the Tree

{Sum of data of all the Nodes of a Tree}

$(a + 10 + b)$ Sum

faith: returns sum of the tree starting from root.



$\sum_i node.data = \boxed{450}$

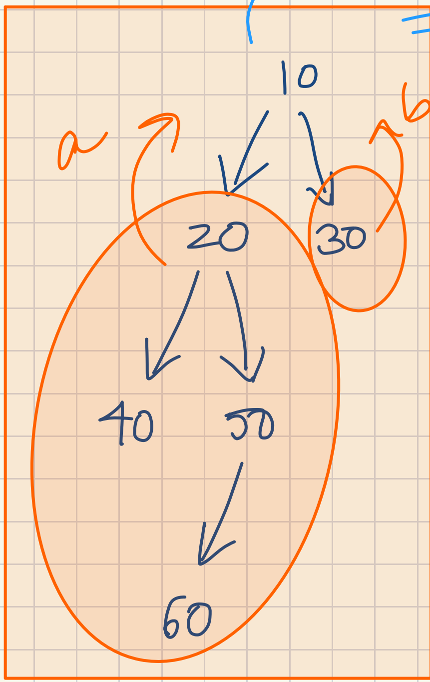
```
int sumOfNodes(Node root)
{

}
```

Height of the tree

$\max(a,b)+1$

height = 4

faith: returns height of the tree starting from root




```
int heightOfTree(Node root)
{

}
```