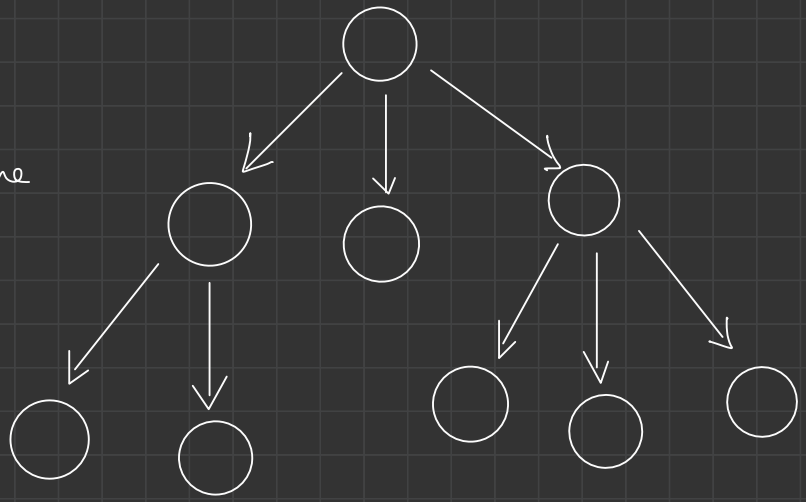
Binary Trees

(Non-linear data structure)

data

- → family tree
- → organisation structure
- → file system



File System

Class Node {

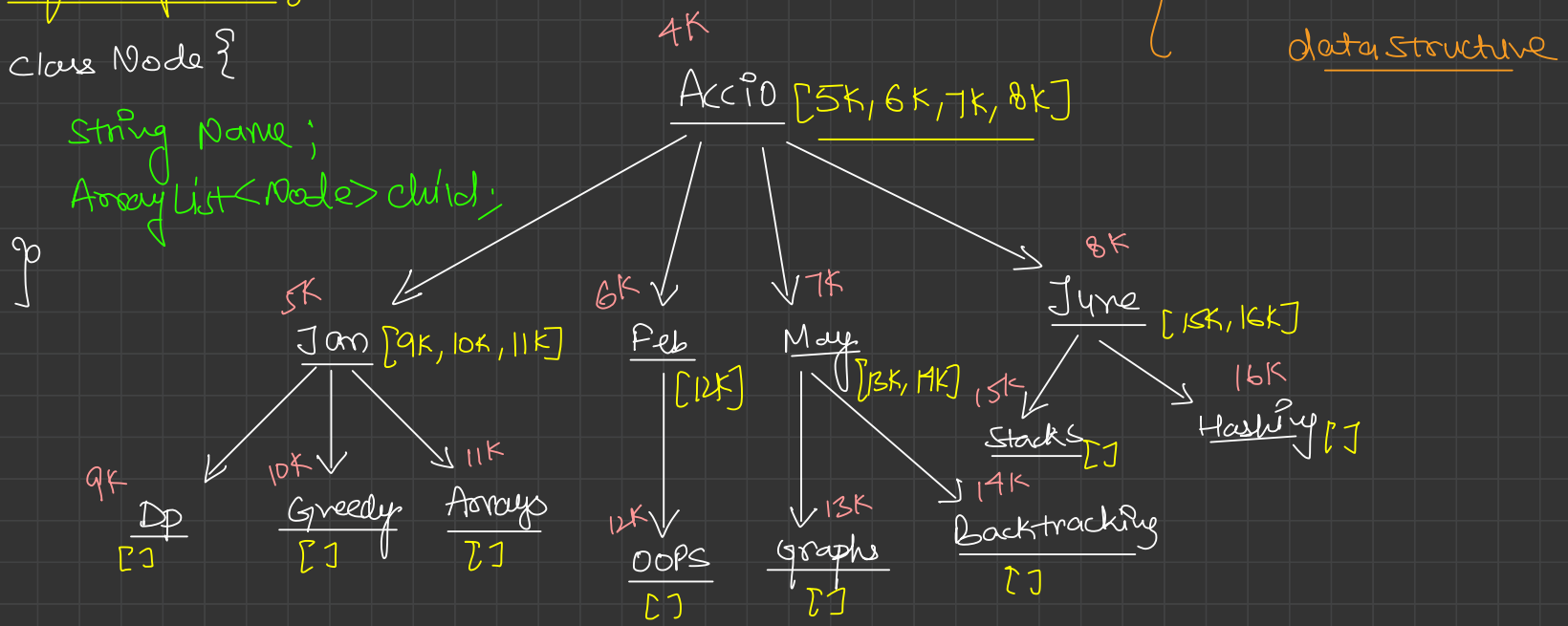
String Name;

ArrayList<Node> child;

}

Generic Tree

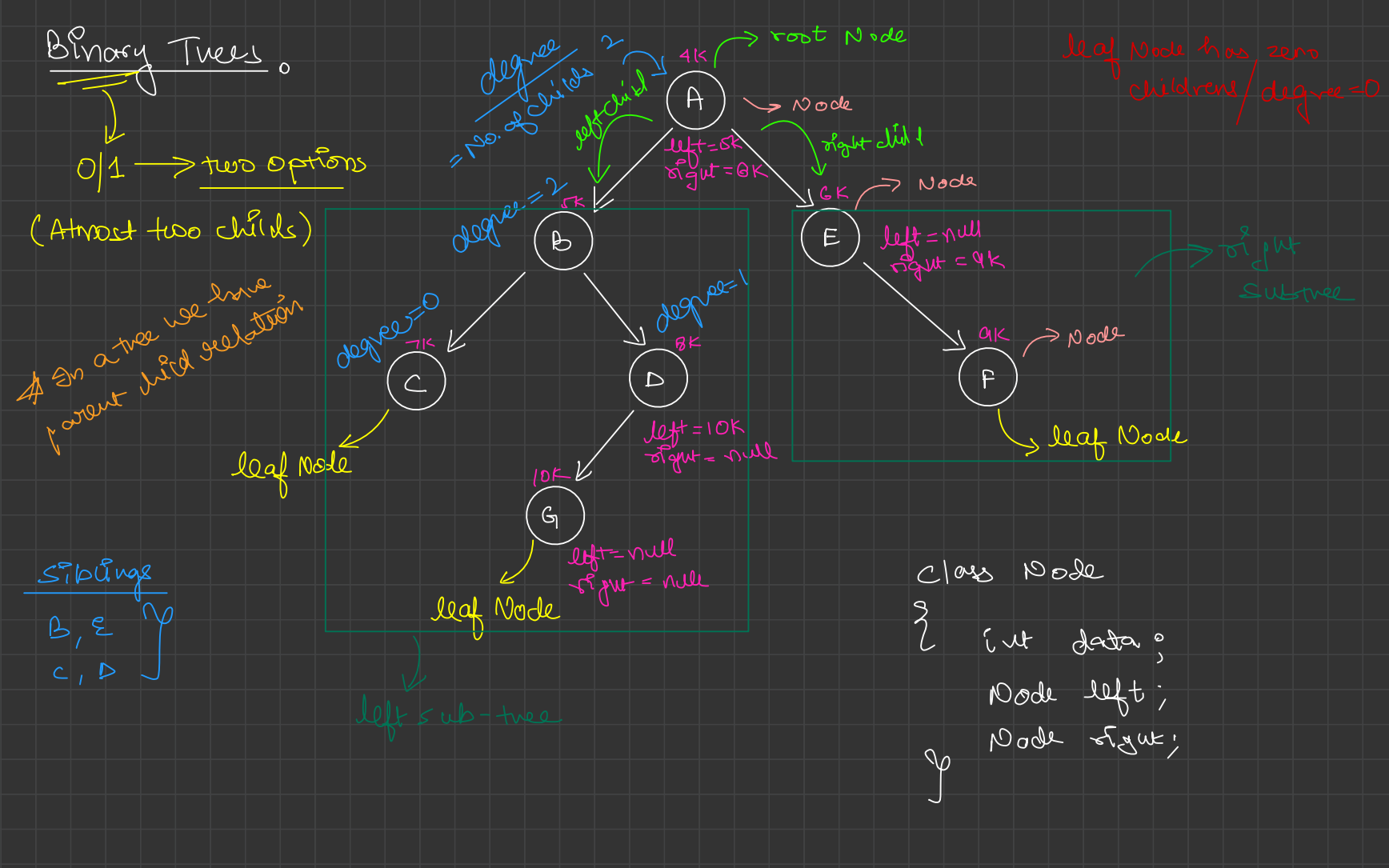
data structure



Binary Trees.

0/1 → two options

(Almost two childs)



leaf Node has zero children / degree = 0

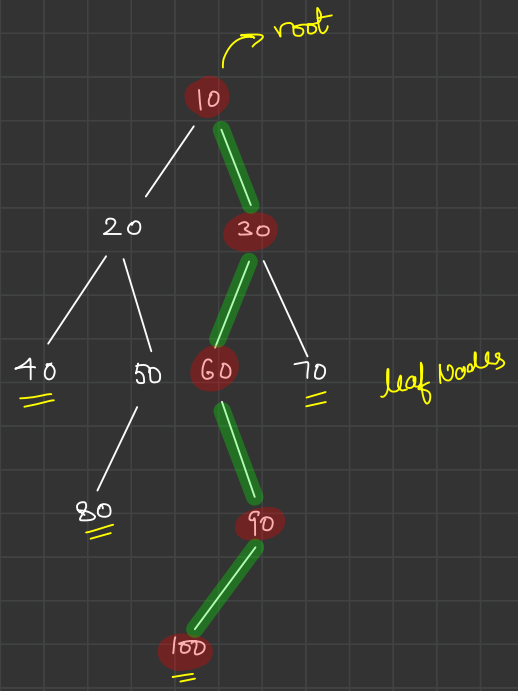
In a tree we have parent child relation

Siblings

B, E

C, D

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

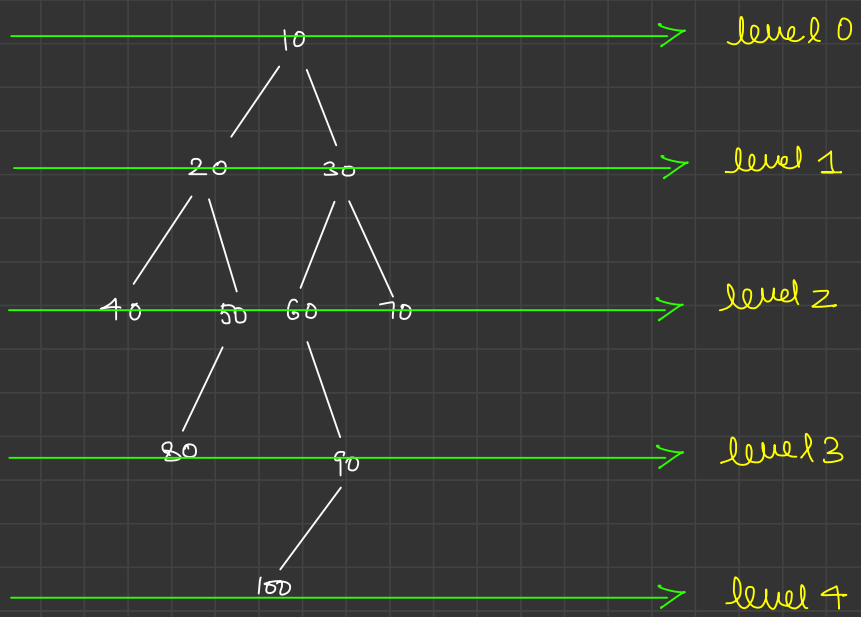


height of a Binary Tree

{ dist b/w root Node and the deepest leaf Node }

in terms of Nodes = 5

in terms of edges = 4


10
20
30
40
50
60
70
80
90
100
level 0
level 1
level 2
level 3
level 4

Height in terms of Nodes

Perfect Binary Tree

- {No. of Nodes at each level $l = 2^l$}

level 0 → {1 Node} → 2^0

level 1 → {2 Nodes} → 2^1

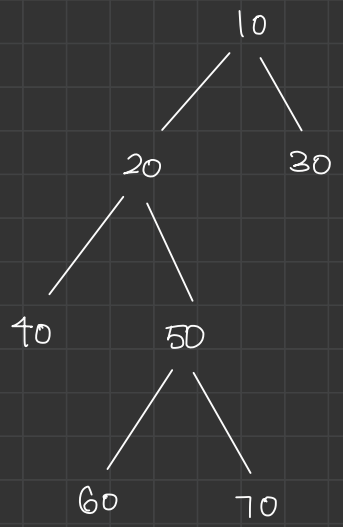
level 2 → {4 Nodes} → 2^2

level 3 → {8 Nodes} → 2^3

$\hookrightarrow 2^{h-1}$

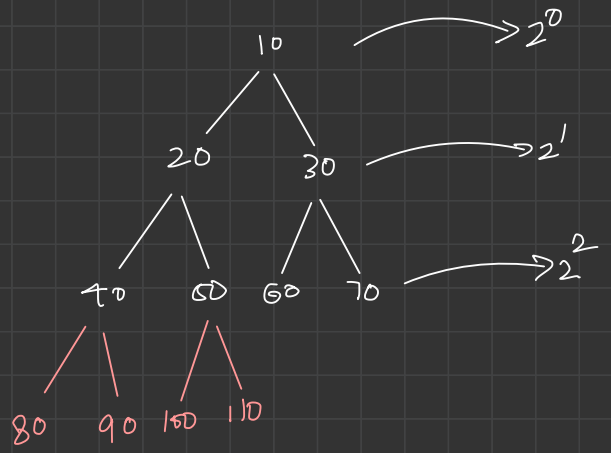
Full Binary Tree

→ Each Node either zero or two childrens



Complete Binary Tree

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

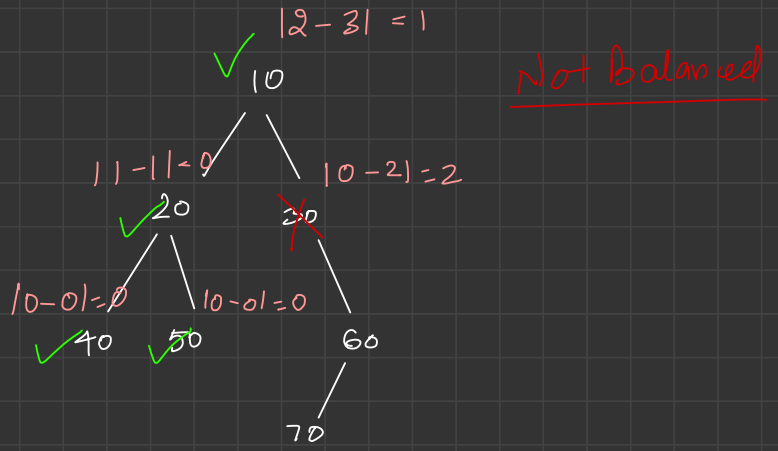


Balanced Binary Tree

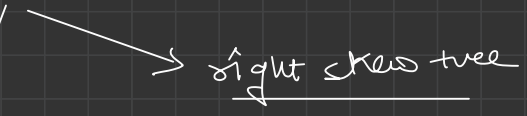
↳ when each node of tree is balanced.

Balanced Node ↳

$$|\text{height of lst} - \text{height of rst}| \leq 1$$



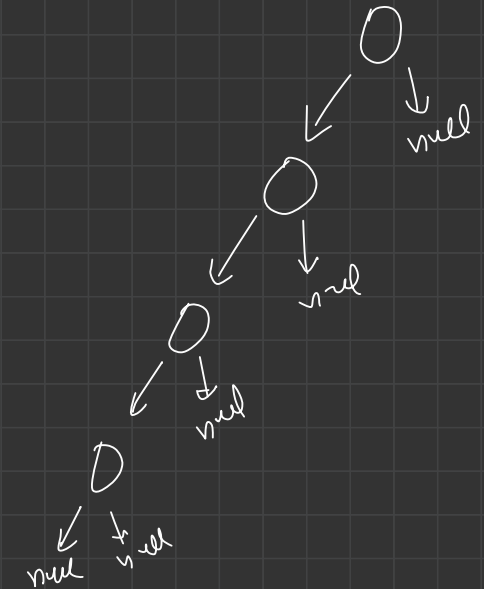
Skew Tree

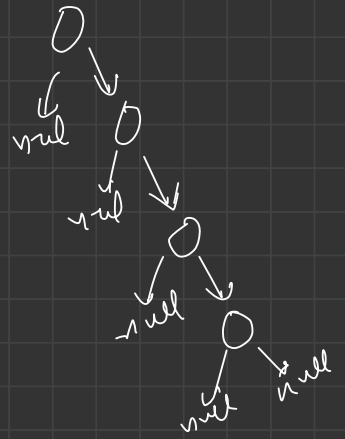


right skew tree

left - skew tree

each node has left child or no - child

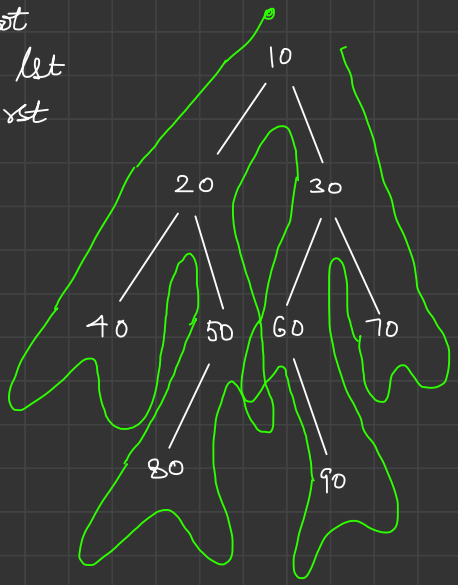




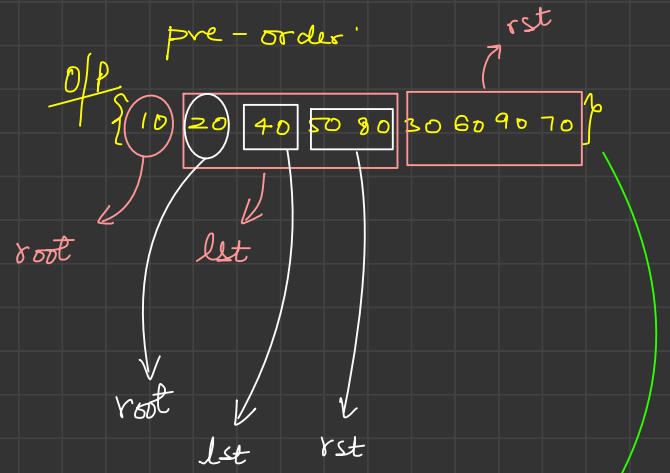
Traversal over Trees

1. pre-order traversal

- → print root
- → explore lst
- → explore rst

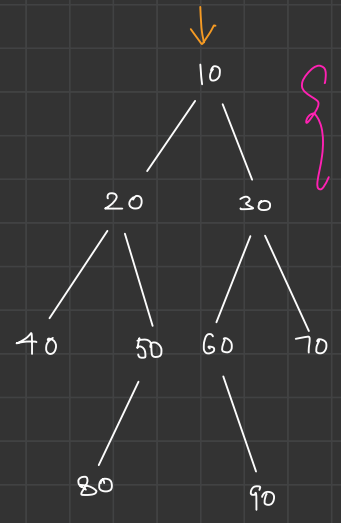


pre-order



Eular path

✓ 10 20 40 50 80 30 60 90 70



TC : O(N)
SC : O(h)

faith : print pre order from the root

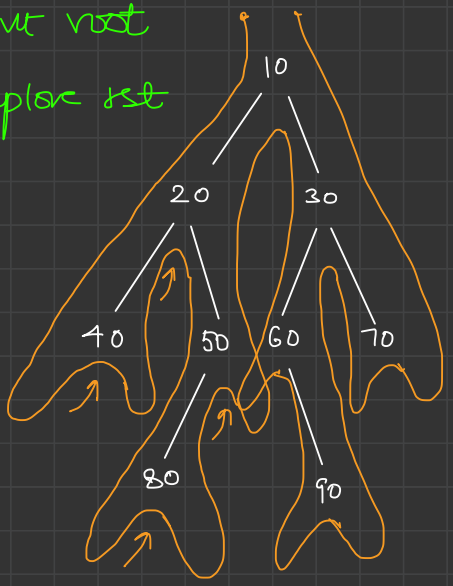
```
void printPreOrder(Node root)
{
    ① if(root == null) return;
    ② print(root.data);
    ③ printPreOrder(root.left);
    ④ printPreOrder(root.right);
}
```

o/p

10 20 40 50 80 30 60 90 70

In Order traversal

↳ explore lst
↳ print root
↳ explore rst



o/p lst root rst

{ 40 20 80 50 | 10 | 60 90 30 70 }

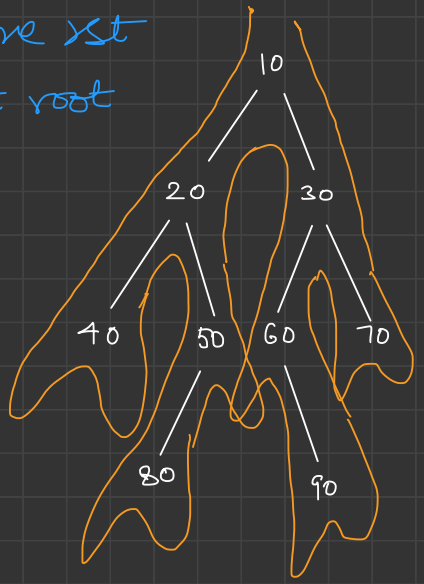
lst root rst

Euler Path

{ 40 20 80 50 10 60 90 30 70 }

Post Order Traversal

↳ explore lst
↳ explore rst
↳ print root



o/p

40 80 50 20 90 60 70 30 10

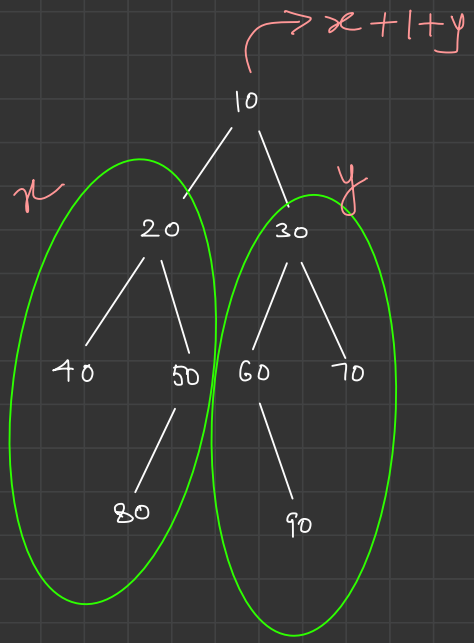
lst rst root

Euler path :

{ 40 80 50 20 90 60 70 30 10 }

Size of Binary Tree

{No. of Nodes}



faith: returns size of tree from root

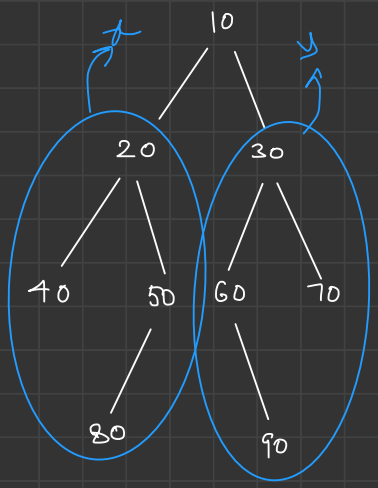
```
int size(Node root)
{

    int sizeOfLeft = size(root.left);

    int

}
```

Sum of tree : { sum of val of all the nodes of the tree }



sum = 450

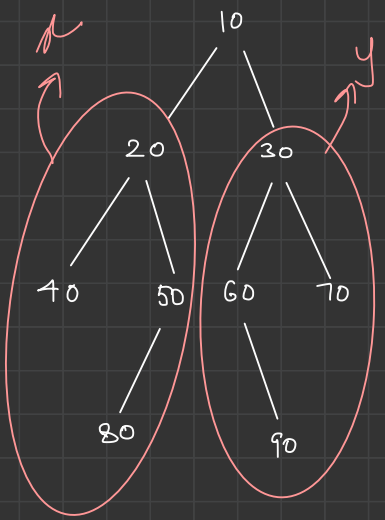
faith : return sum of a tree starting from root.

int sum (Node root)

Max of tree

→ Max^M value present in the tree

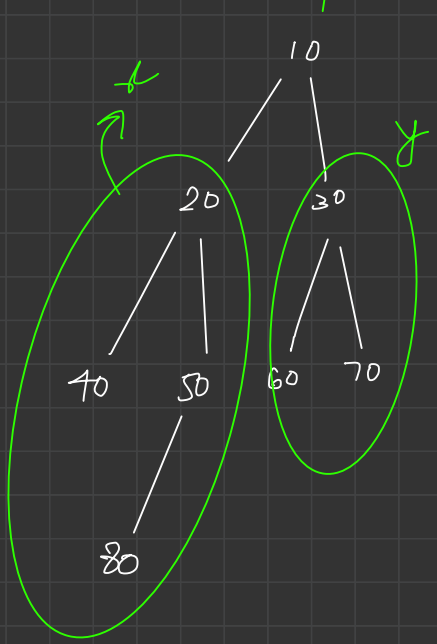
max(x, y, root, data)



faith: returns max value in the tree starting from root

int maxOfTree(Node root)

height of the tree (Nodes)



→ root Node

max(x, y) + 1

faith: height of tree from root

int height(Node root)