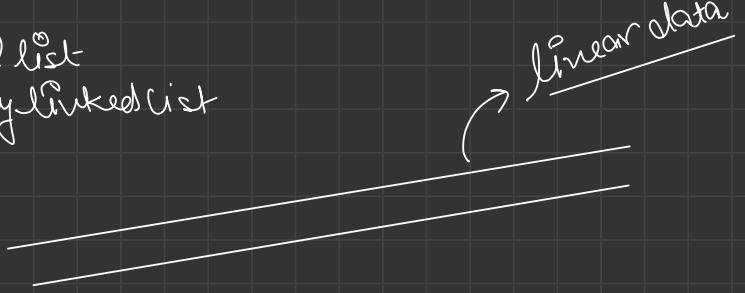
Binary Trees

linear ds

- arrays
- array list
- queues
- stacks
- linked list
- doubly linked list

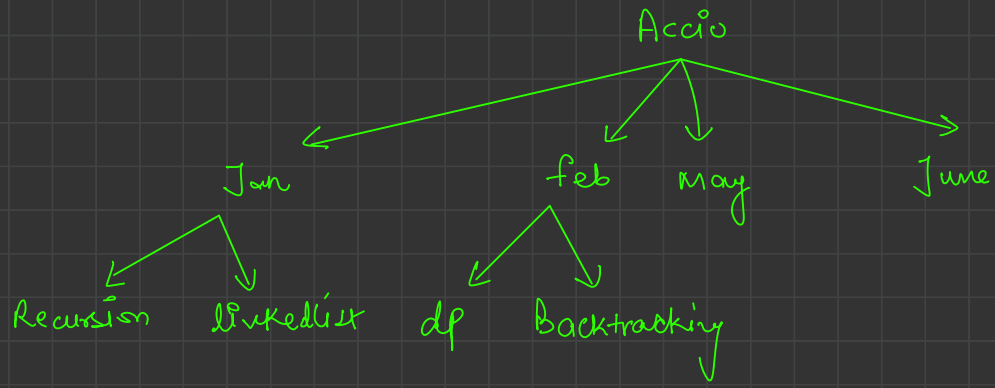


data ⟶ family tree

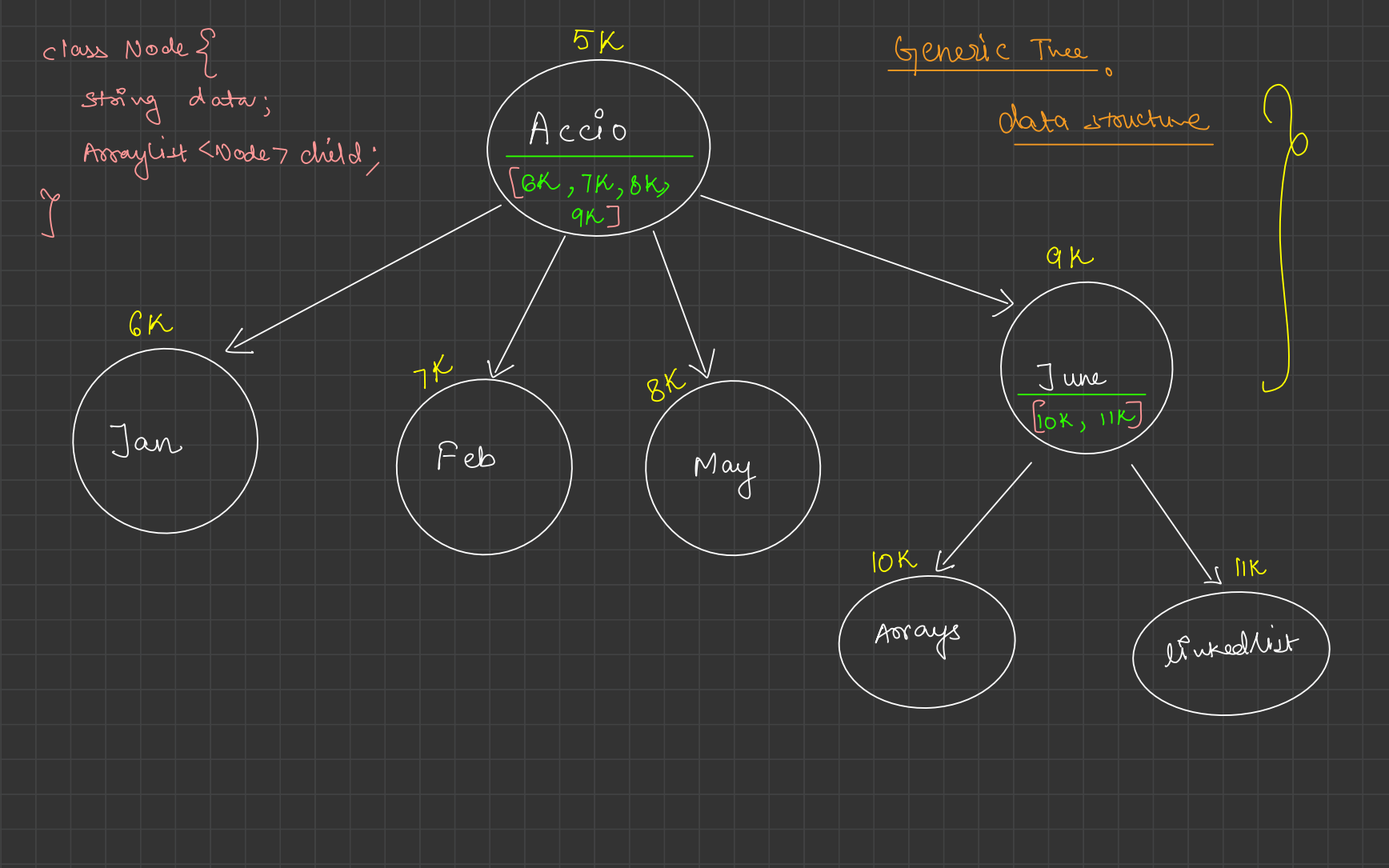
⟶ organisation chart

⟶ file system

→ work on this kind of data



can't be stored in a linear data structure


class Node {
string data;
ArrayList <Node> child;
}
5K
Accio
[6K, 7K, 8K, 9K]
6K
Jan
7K
Feb
8K
May
9K
June
[10K, 11K]
10K
Arrays
11K
LinkedList
Generic Tree
data structure

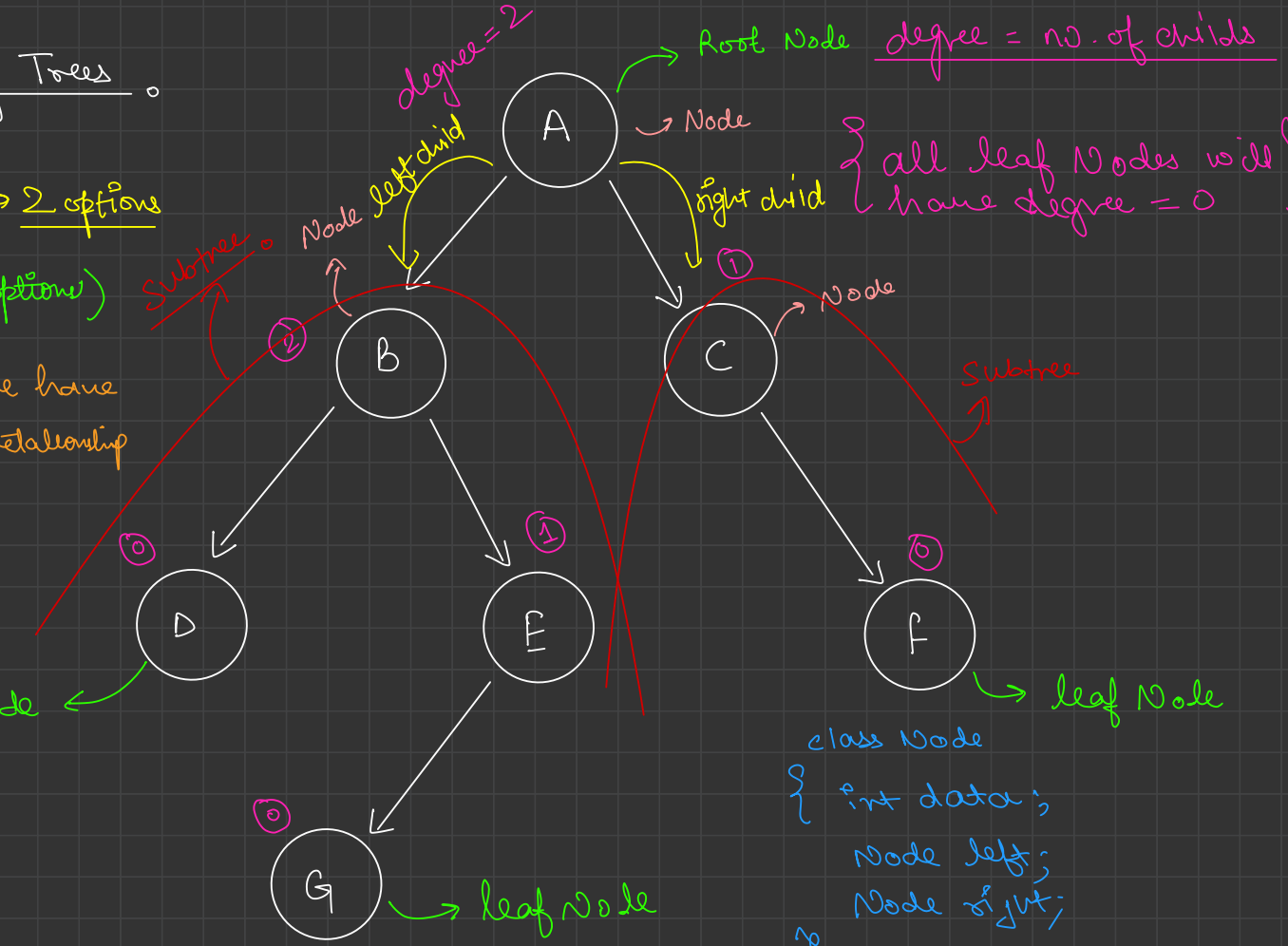
Binary Trees

0/1 → 2 options

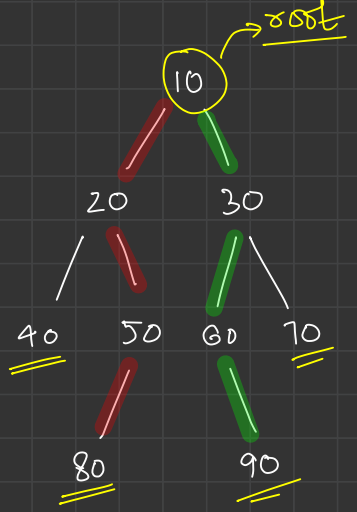
(Atmost 2 options)

* In trees we have parent child relationship

Siblings
- D, E
- B, C




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

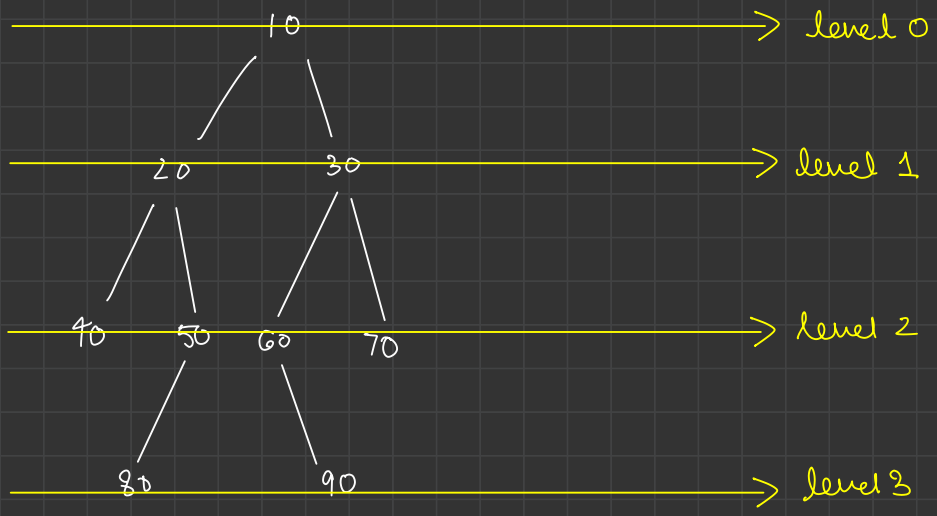


height of a Binary Tree

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

in term of Edges = 3

in term of Nodes = 4


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

Perfect Binary Trees { No. of Nodes in each level $h = 2^h$ }

{ 1 Node } → 2^0

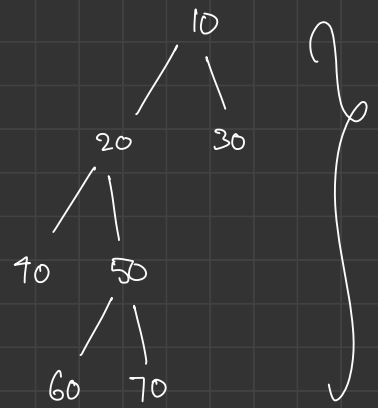
{ 2 Nodes } → 2^1

{ 4 Nodes } → 2^2

{ 8 Nodes } → 2^3

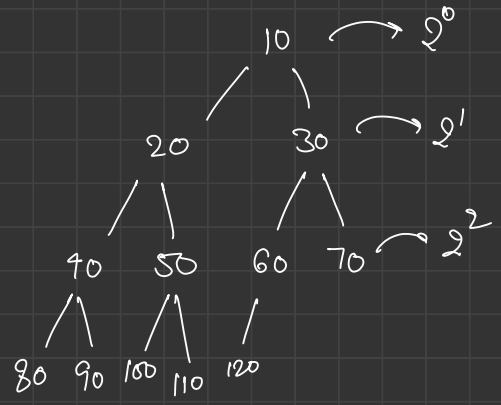
full Binary Tree

- each nodes has either zero or two children



Complete Binary Tree

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

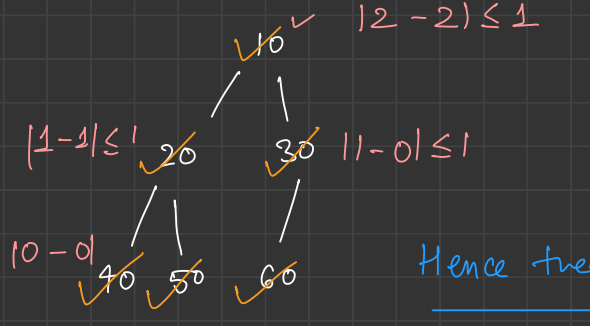


Balanced Binary Tree

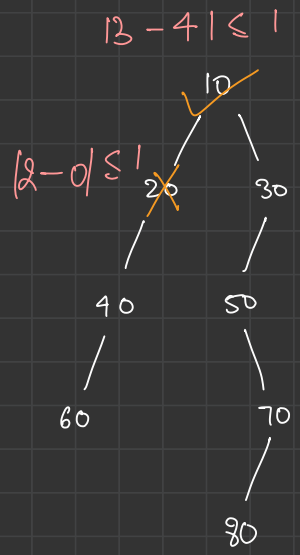
↳ when each node of a tree is balanced

Balanced Node

↳ $|\text{height of leftSubtree} - \text{height of rightSubtree}| \leq 1$

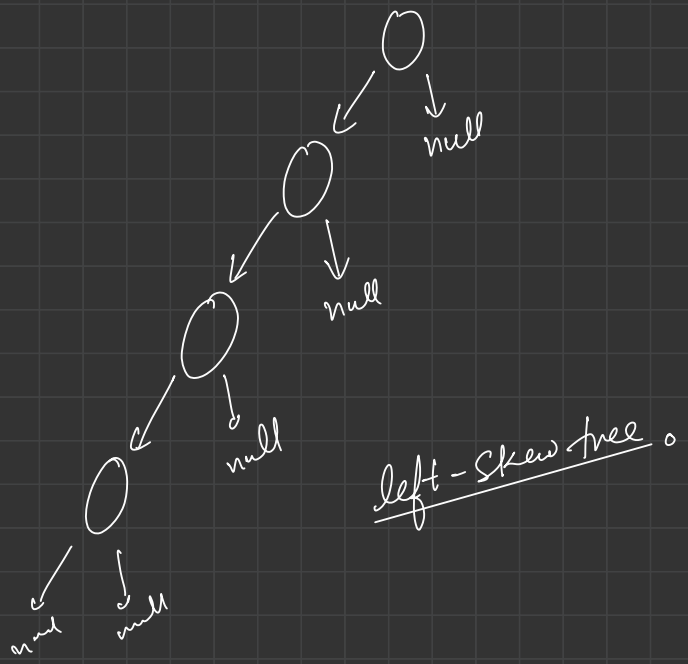


Hence tree, is balanced

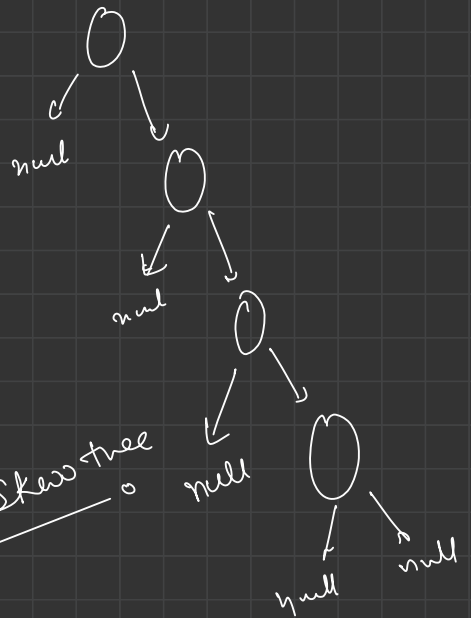


No, this tree is not Balanced !

Skew Tree



left-Skew tree

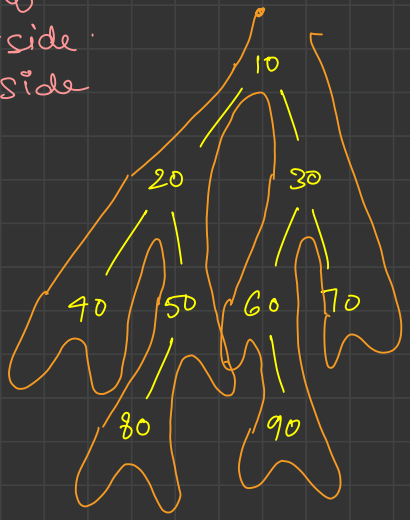


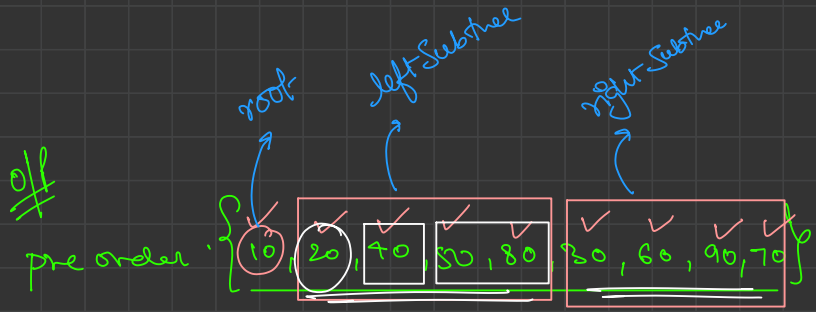
right-Skew tree

Traversal over Tree

① pre – order traversal

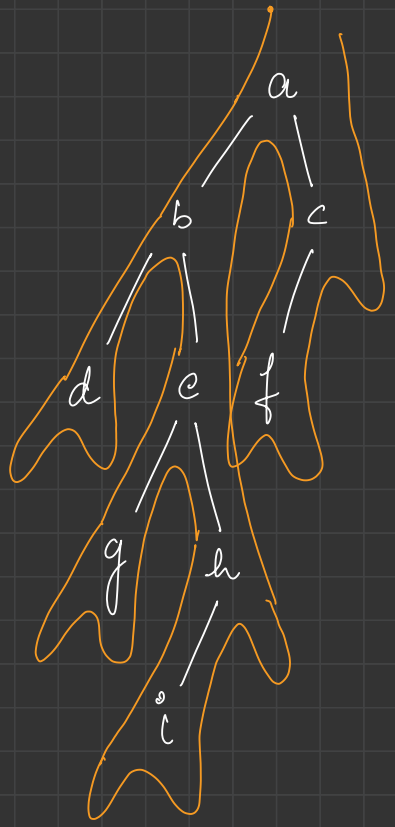
{ print yourself
explore left side
explore right side





pre order : { 10, 20, 40, 50, 80, 30, 60, 90, 70 }

Euler path → pre – order { 10, 20, 40, 50, 80, 30, 60, 90, 70 }

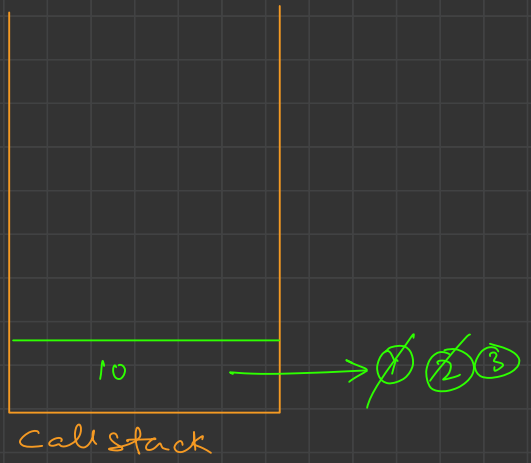


{ a, b, d, e, g, h, i, c, f }

↳ preorder.

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

4



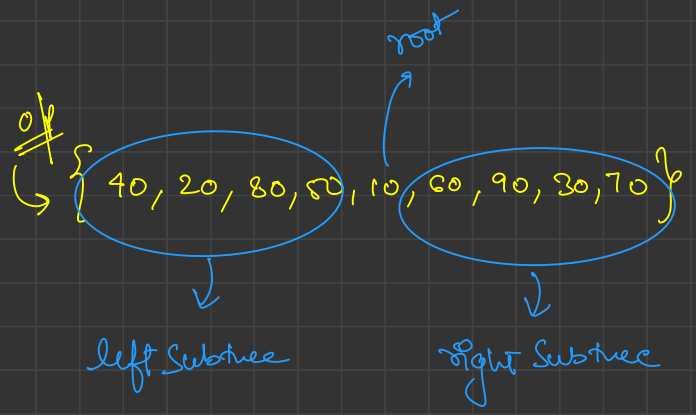
faith: It print the order from root

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

{ 10, 20, 40, 50, 80

(2) In-order traversal

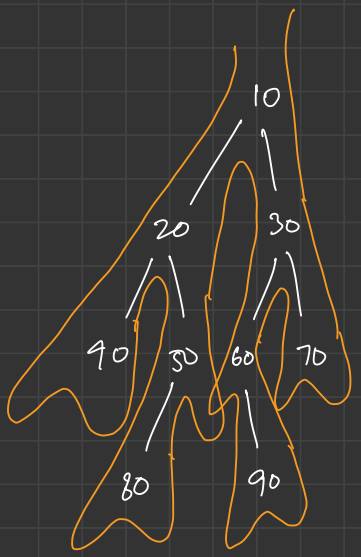
explore leftside
print yourself
explore rightside



Eular path : {40, 20, 80, 50, 10, 60, 90, 30, 70}

Post Order Traversal

{ explore left side
explore right side
print yourself



Ans

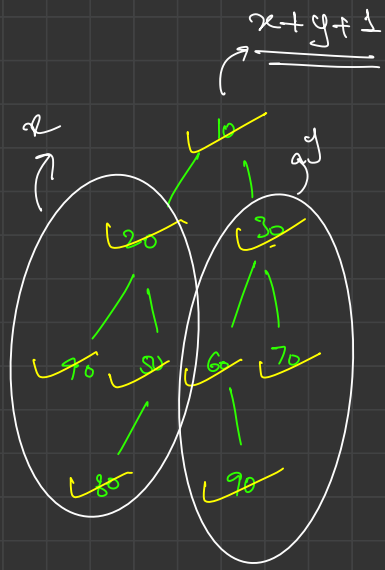
40, 80, 50, 20, 90, 60, 70, 30, 10

Eular path

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

Size of Binary Tree

↳ { No. of nodes in a Binary Tree }

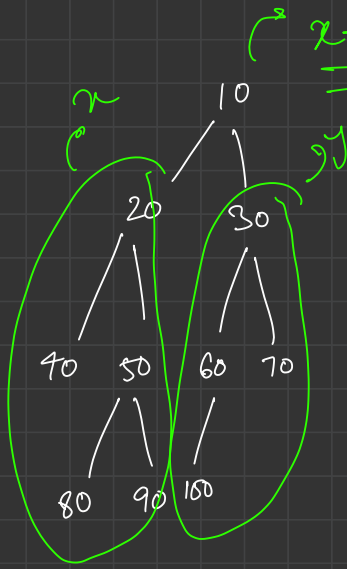


Size = 9

fxn: returns size of the tree

int size (Node *root)

Sum of tree { Sum of all the values of Nodes }



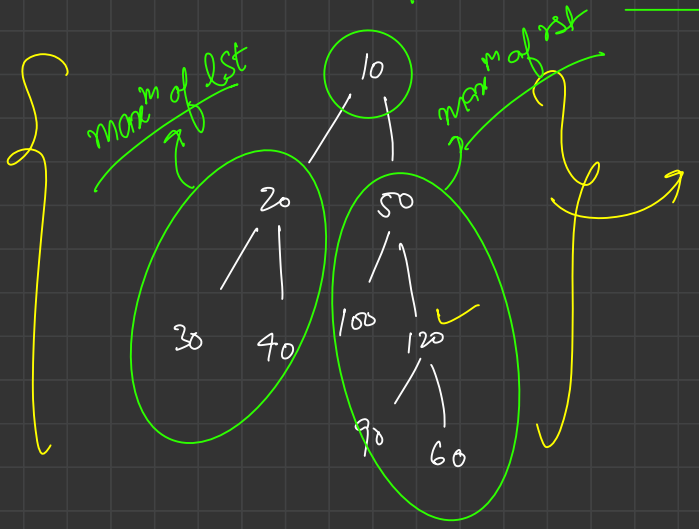
faith: Sum of tree

int Sum(Node root)

Max of tree

↳ Maxᵐ value present in a tree.

maxᵐ(maxᵐ lst, maxᵐ rst, root.val)



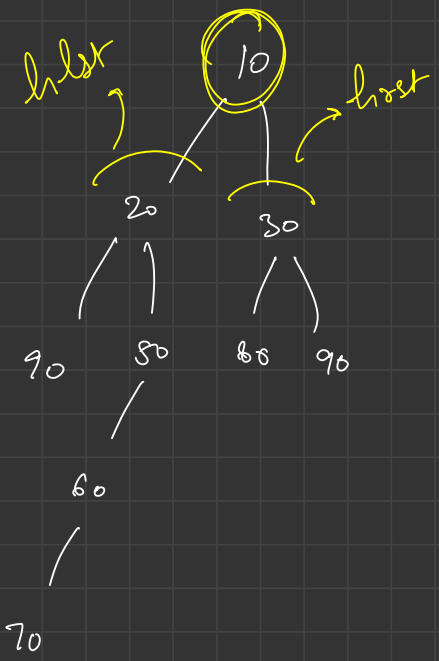
120.

faith: maxᵐ of a tree.

int maxOfTree(Node root)

height of tree (Nodes)

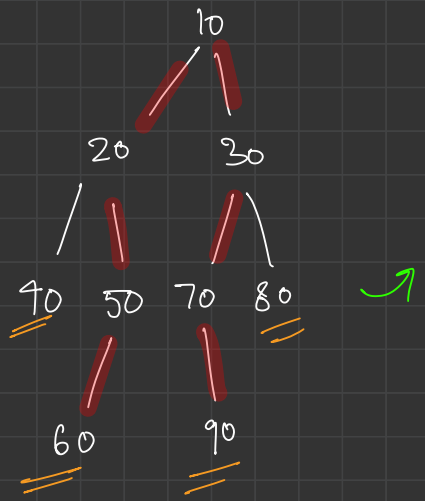
$\max(lh, rh) + 1$ → root node



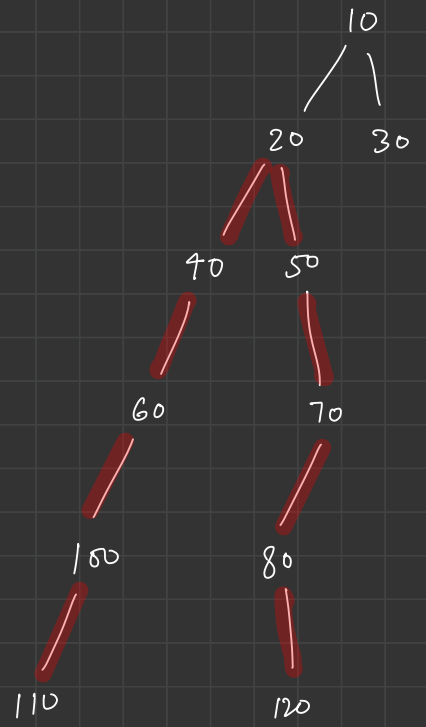
fair height of tree

```
int height(Node node)
```

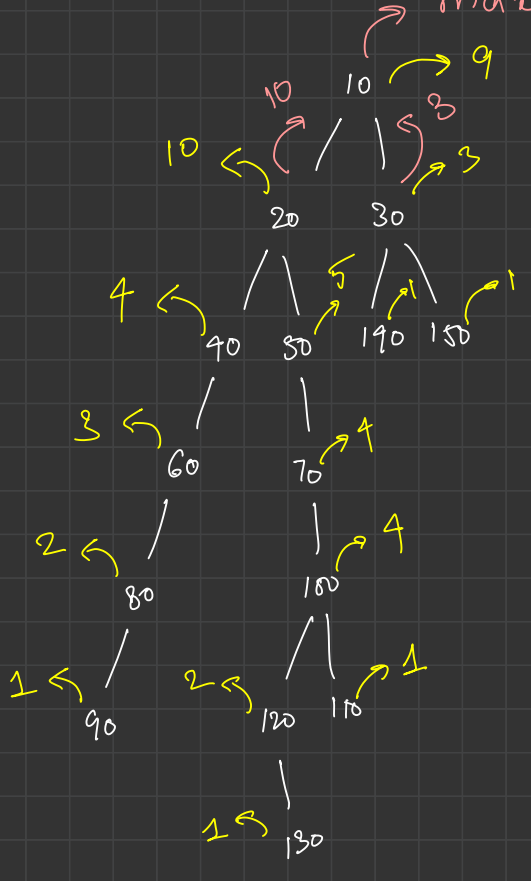
diameter of tree. { max^m dist b/w any two leaf Nodes }



diameter = 7



diameter = 9



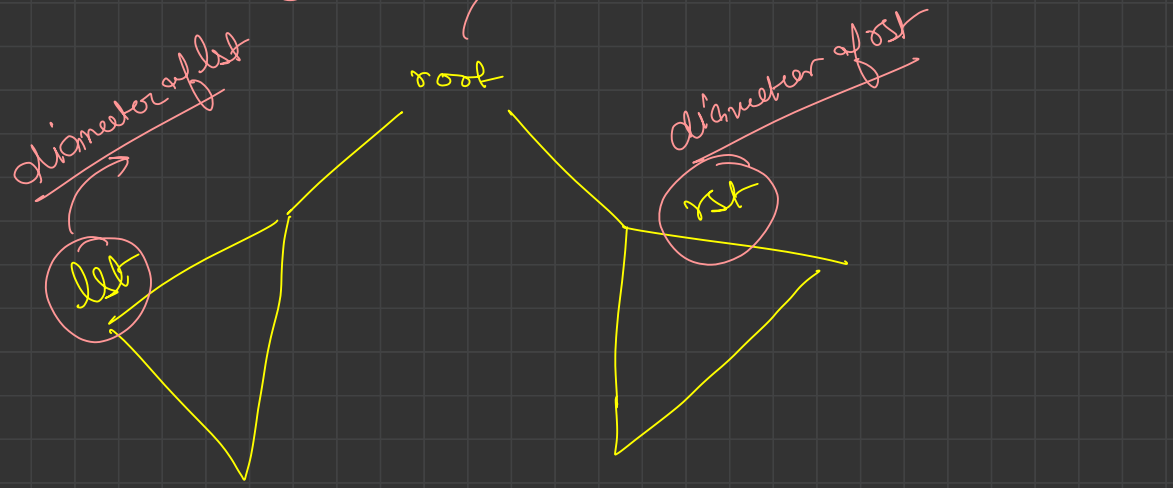
Diameter of tree

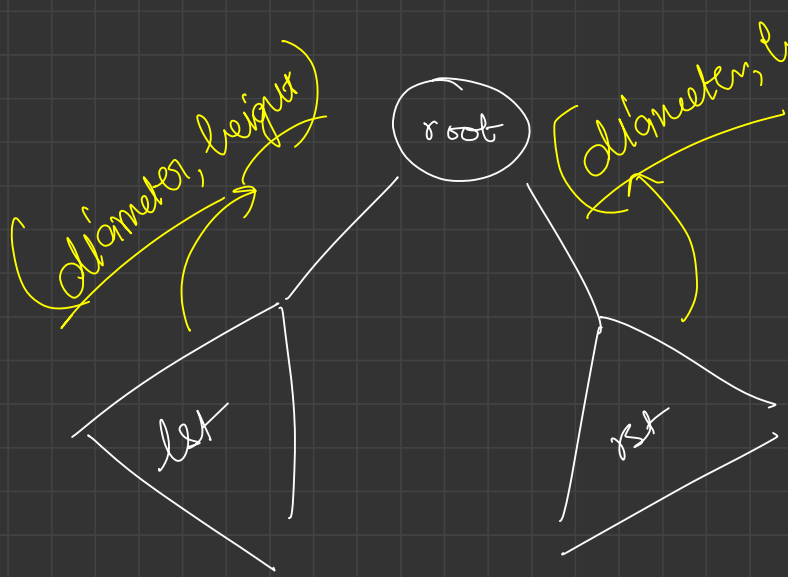
{ max^m of dia of each node }

int diameter(Node root)

faith: returns diameter of tree!

max(lst, rst, root's dia)?!





TC: O(N)

{ fair → diameter + height }