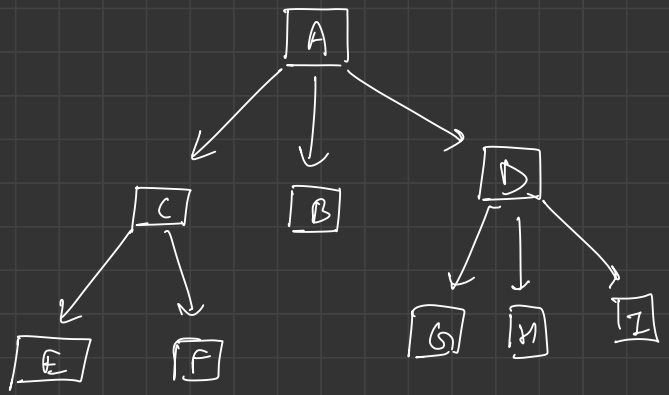
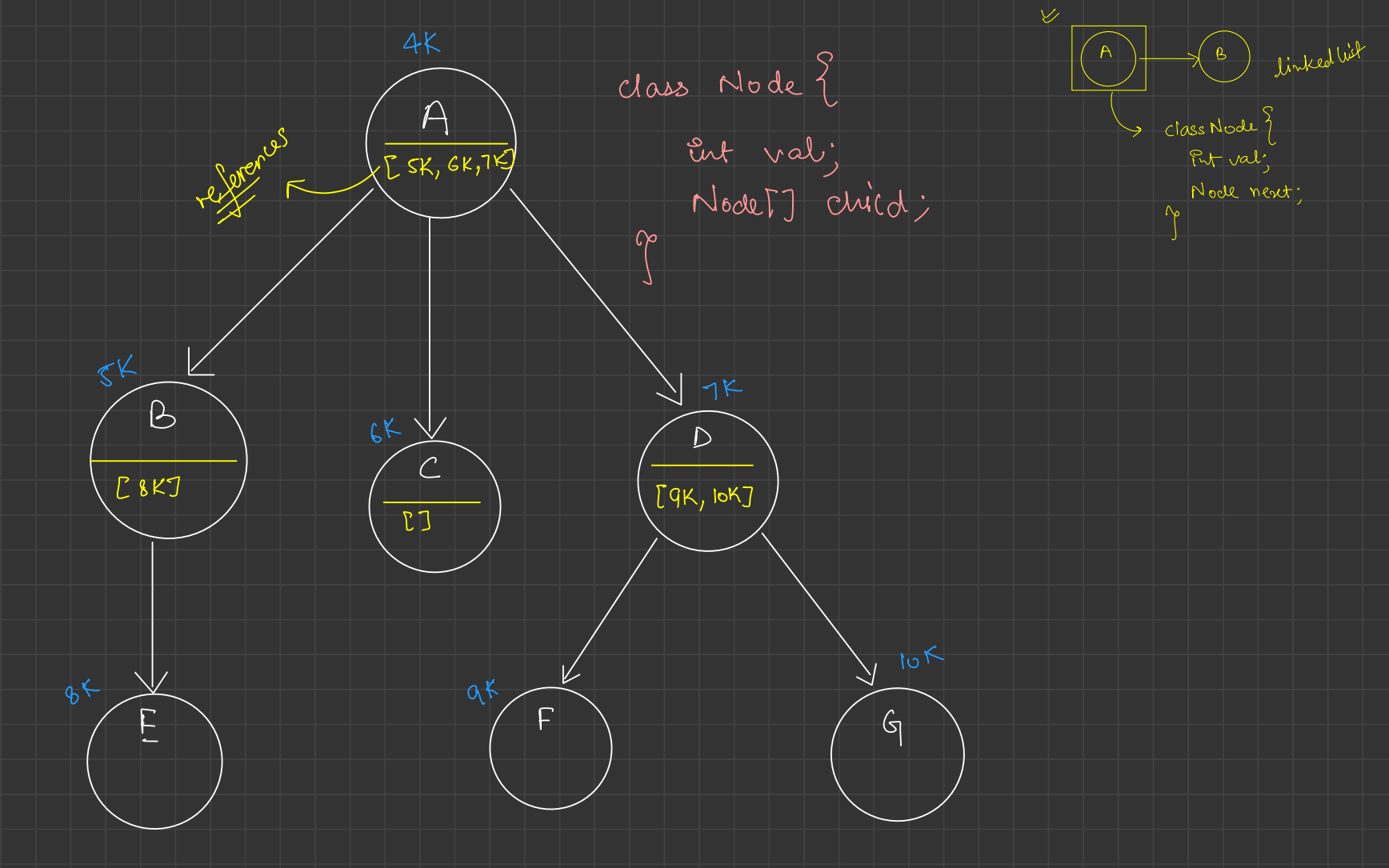
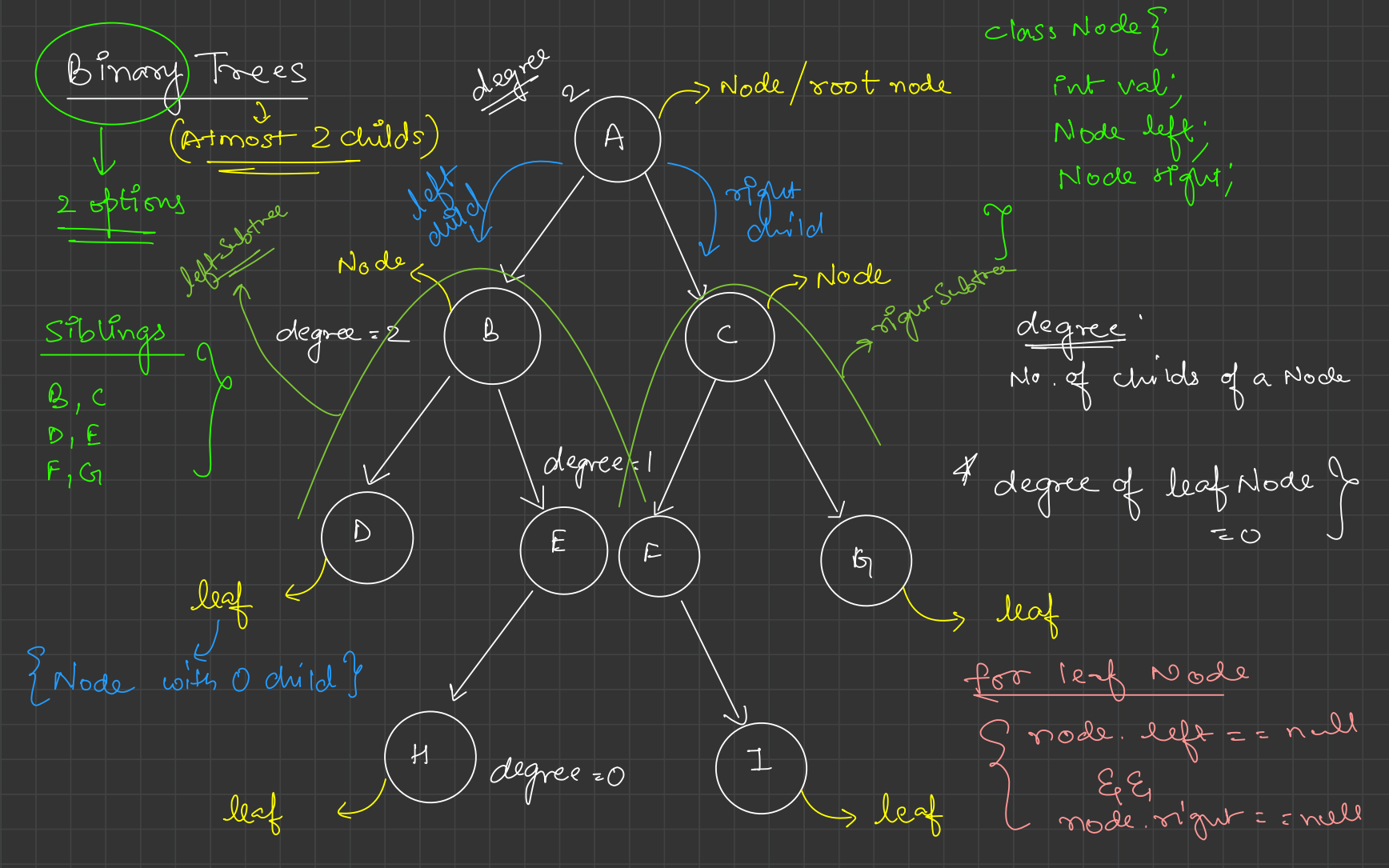
Binary Trees

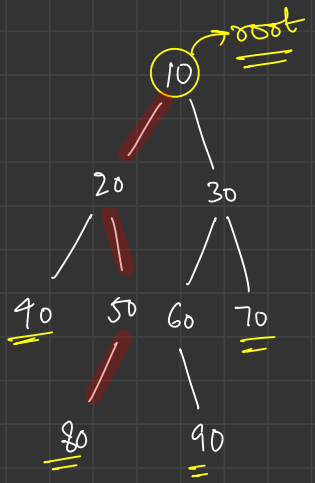
Store your office employees data!

Trees!




4K
A
[5K, 6K, 7K]
references
class Node {
int val;
Node[] child;
}
5K
B
[8K]
6K
C
[]
7K
D
[9K, 10K]
8K
E
9K
F
10K
G
A
B
linked list
class Node {
int val;
Node next;
}

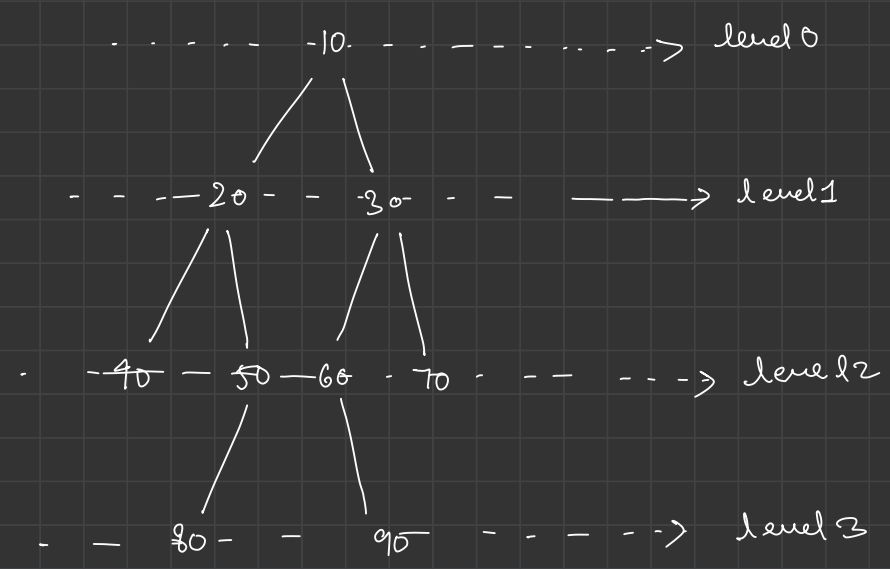

Binary Trees
(Atmost 2 childs)
2 options
degree
2
A
Node / root node
left child
right child
left subtree
right subtree
Node
Node
B
C
degree = 2
Siblings
B, C
D, E
F, G
degree = 1
D
E
F
G
leaf
leaf
{Node with 0 child}
H
I
degree = 0
leaf
leaf
class Node {
int val;
Node left;
Node right;
degree :
No. of childs of a Node
* degree of leaf Node = 0
for leaf Node
node.left == null
&&
node.right == null



height of a Binary tree

{ dist b/w root node and deepest leaf Node in term of edges

height = 3

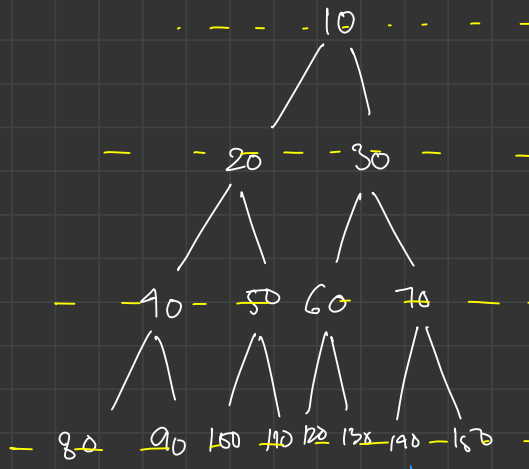


{Think of level as generation}

Perfect Binary Tree

{ A tree where no. of nodes at h level is 2^h }

height : 3



level 0 { 1 Node } → 2^0

level 1 { 2 Nodes } → 2^1

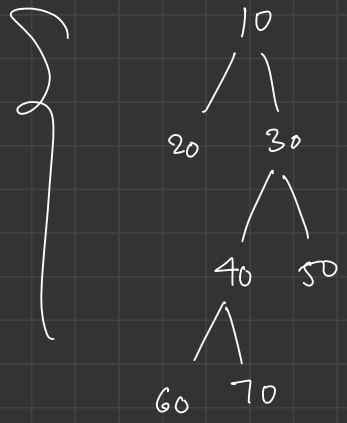
level 2 { 4 Nodes } → 2^2

level 3 { 8 Nodes } → 2^3

h level h → { 2^h Nodes }

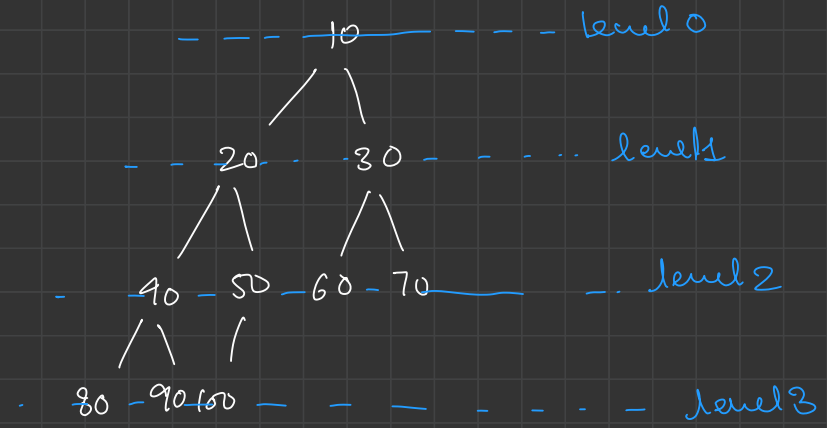
Full Binary Tree

↳ A BT where each Node has either zero or two childs.



Complete Binary Tree

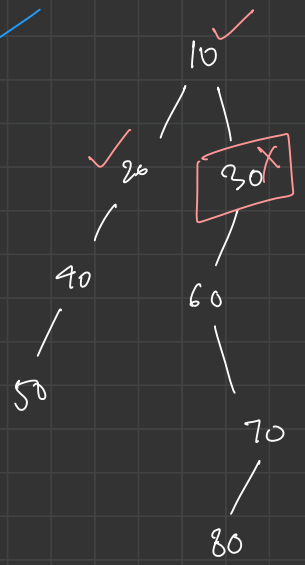
where each level is completely filled, except the last level, and the nodes in last level are as left as possible



Balanced Binary Trees

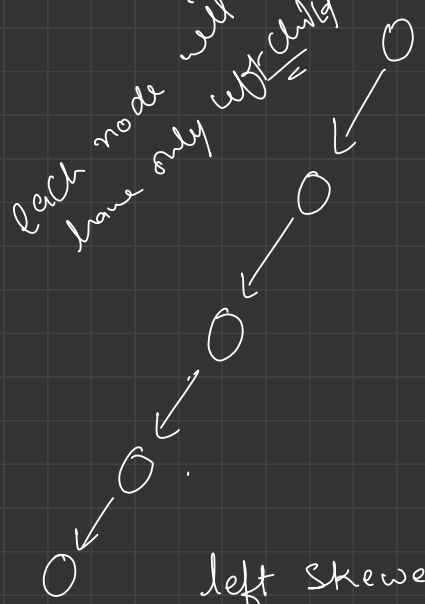
$$|\text{height of left Subtree} - \text{height of right Subtree}| \leq 1$$

for Every Node

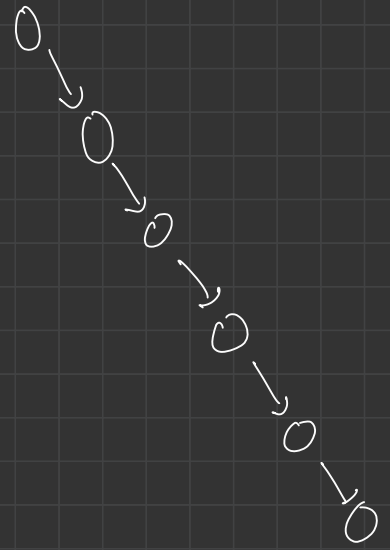


→ Not Balanced

Skew tree.



left skewed tree



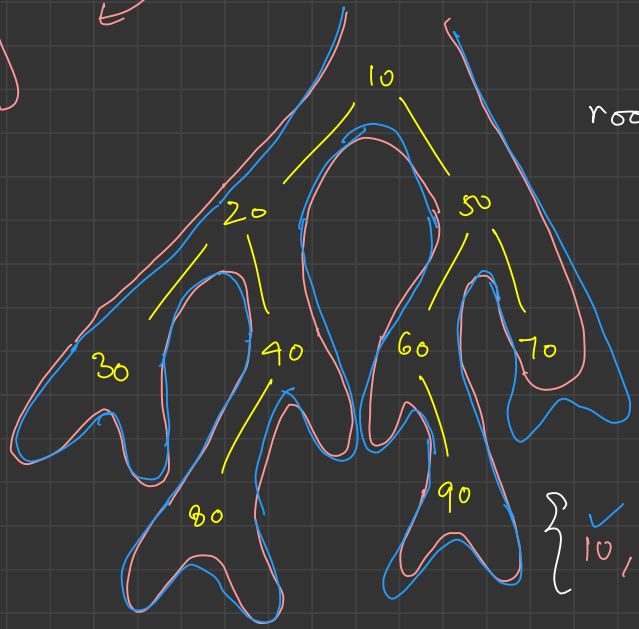
right skewed tree

Traversal over tree

Q.

① pre order traversal

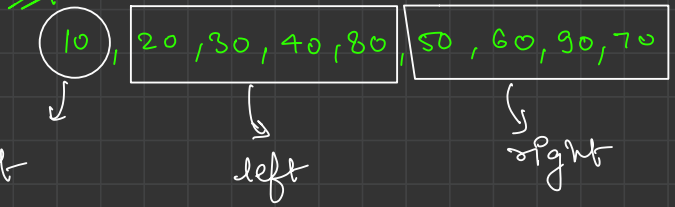
print (node)
left subtree
right subtree



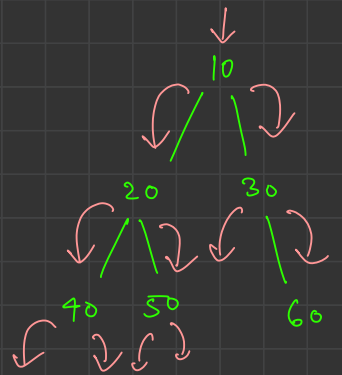
Eular path

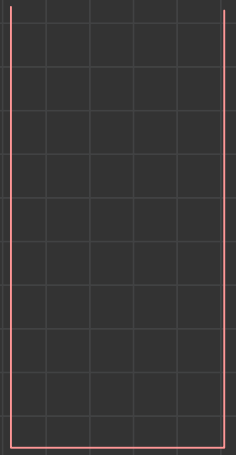
{10, 20, 30, 40, 80, 50, 60, 90, 70}

Or



```
void preOrder(TreeNode root)
{
    (1) if (root == null)
            return;
    (2) print(root.val);
    (3) preOrder(root.left);
    (4) preOrder(root.right);
}
```



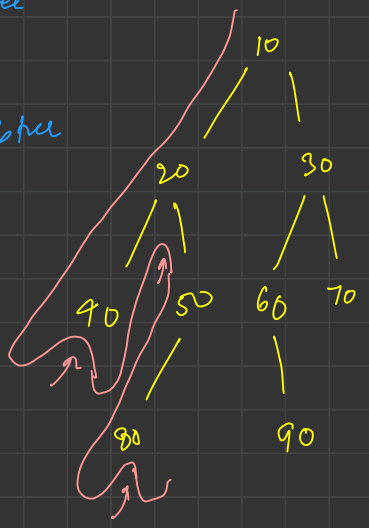


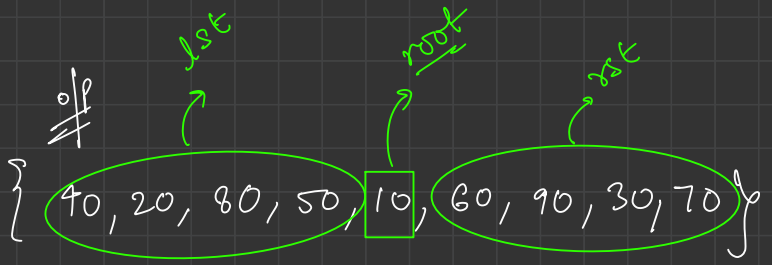
callstack

{ 10, 20, 40, 50, 30, 60 }

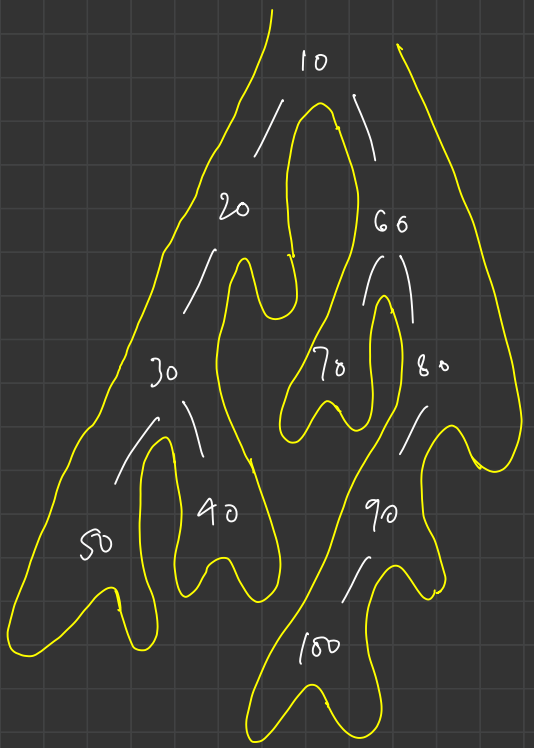
InOrder Traversal

- leftSubtree
- root
- rightSubtree





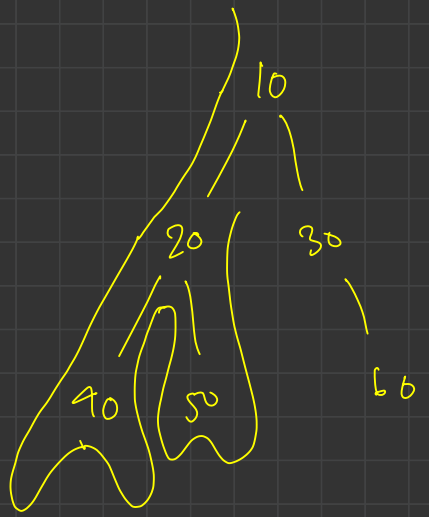
{ 40, 20, 80, 50, 10, 60, 90, 30, 70 }



Inorder

50, 30, 40, 20, 10, 70, 60, 100, 90, 80

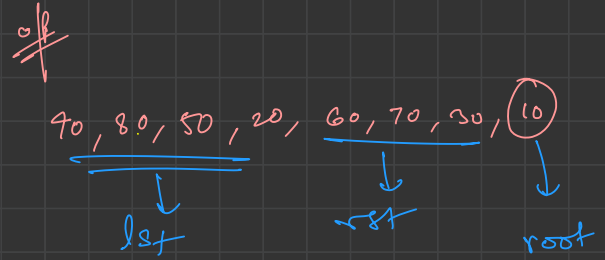
```
void InOrder(TreeNode root)
{
    if (root == null)
        return;

    InOrder(root.left);
    print(root.val);
    InOrder(root.right);
}
```

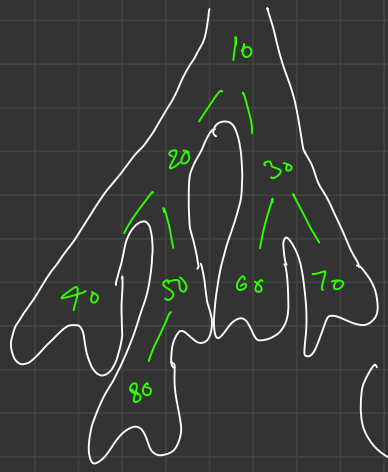


40, 20, 50, 10, 30, 60

post Order Traversal

- left Subtree
- right Subtree
- print ()

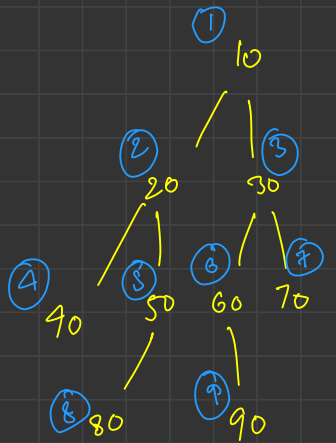




(40, 80, 50, 20, 60, 70, 30, 10)

Size of a Binary Tree

(No. of Nodes in Binary Tree)



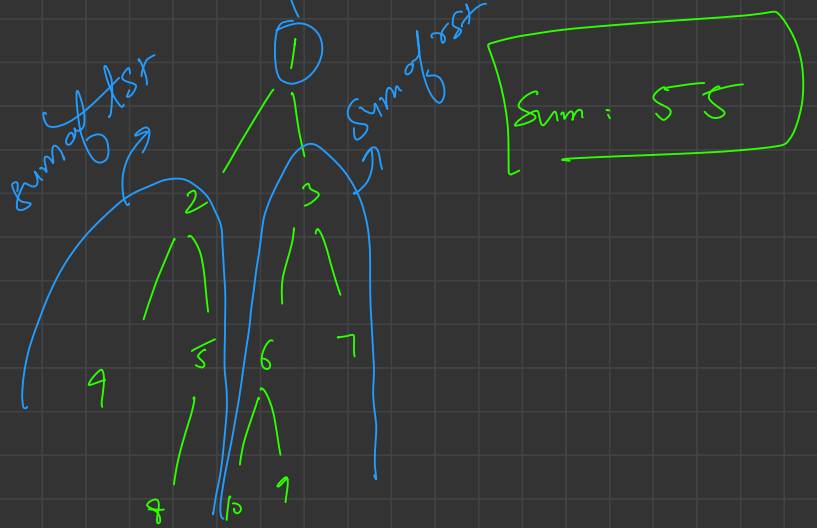
Size = 9

```
int size (TreeNode root)
{ if(root == null) return 0;
  int lst = size(root.left);
  int rst = size(root.right);
  return lst + 1 + rst;
}
```

Sum of tree

↳ Sum of all nodes value in a tree

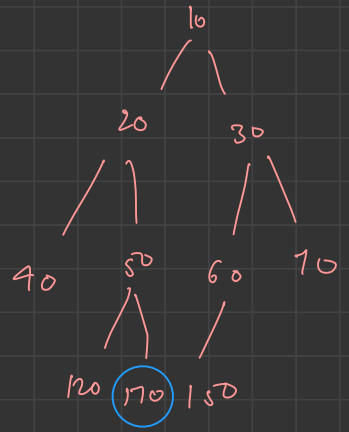
Sum of lst + root.val + Sum of rst

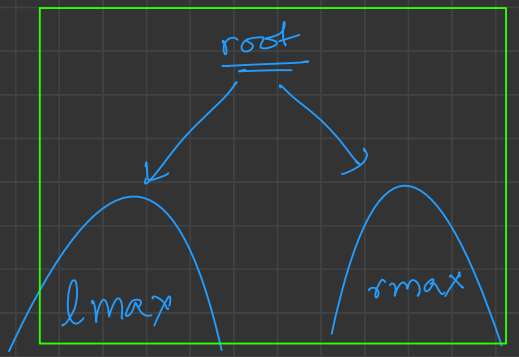


Max of tree

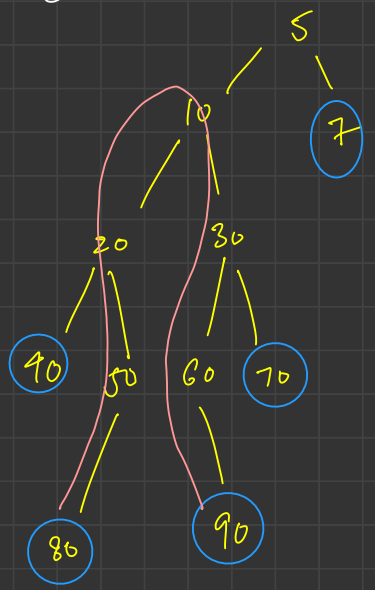
↳ Max^{m} node value present in a tree.

max(lmax, root.val, rmax)





diameter of tree { max^m dist. betw any two leaf Nodes }



40 - 80 → 3

40 - 90 → 5

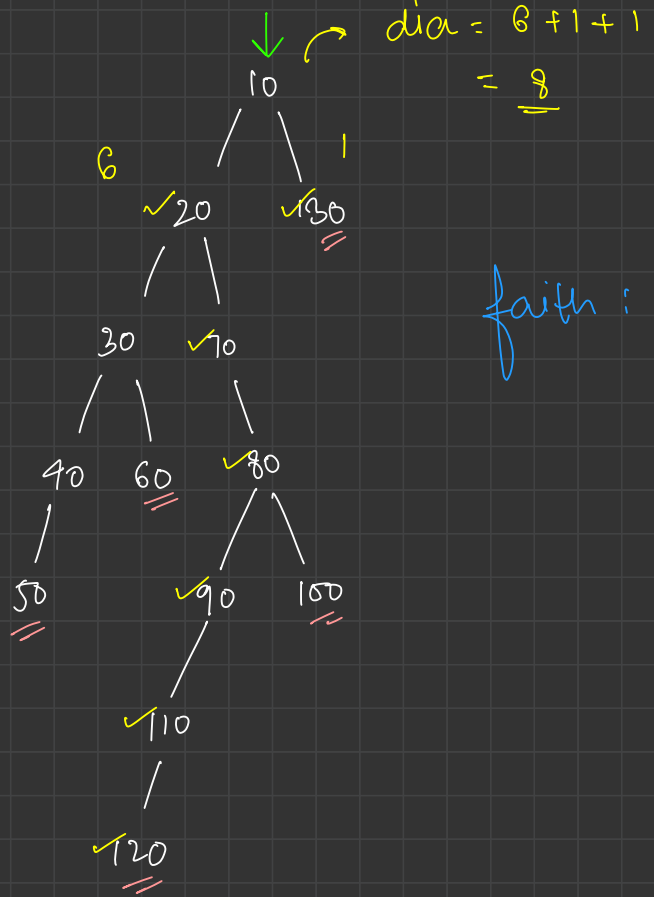
40 - 70 → 4

80 - 90 → 6

80 - 70 → 5

90 - 70 → 3

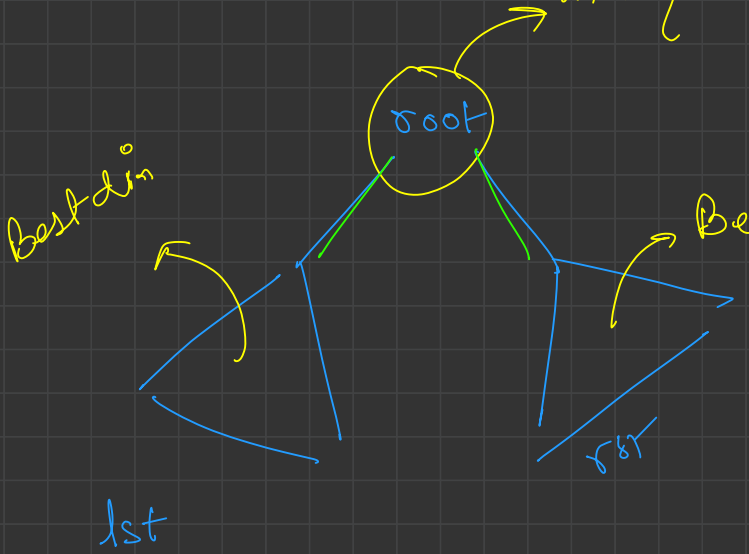
→ diameter



dia = 6 + 1 + 1

= 8

faith : subtree ji's best ✓ diameter



max { Best from Left, Best from Right, dia passing through me }

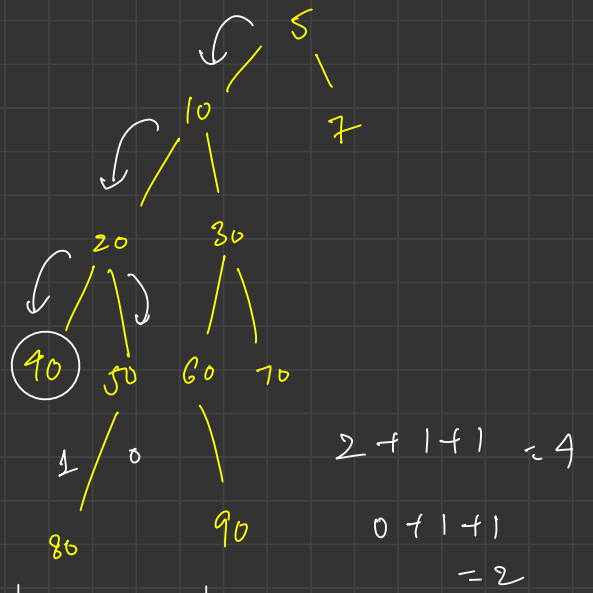
dia = (lh + 1 + rh)

(In terms of nodes)

```java
public static int diameter(Node root) {
    // Base Case
    if (root == null) {
        return 0;
    }

    int bestDiaInLeft = diameter(root.left);
    int bestDiaInRight = diameter(root.right);

    // diameter passing through root
    int lh = height(root.left);
    int rh = height(root.right);
    int diaThroughRoot = lh + 1 + rh;

    // diameter of the tree -> max {bestDiaInLeft, bestDiaInRight, diaThroughRoot}
    return Math.max(diaThroughRoot, Math.max(bestDiaInLeft, bestDiaInRight));
}
```

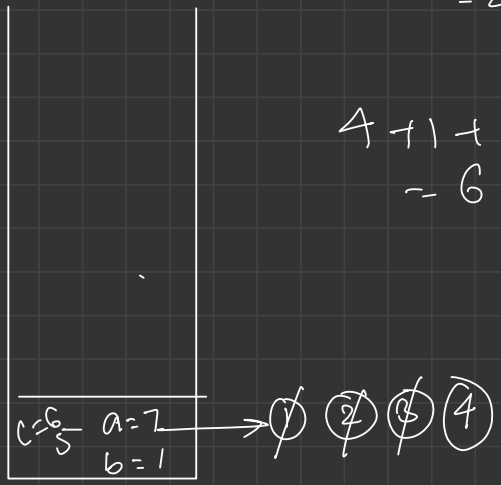
1 2 3 4 5

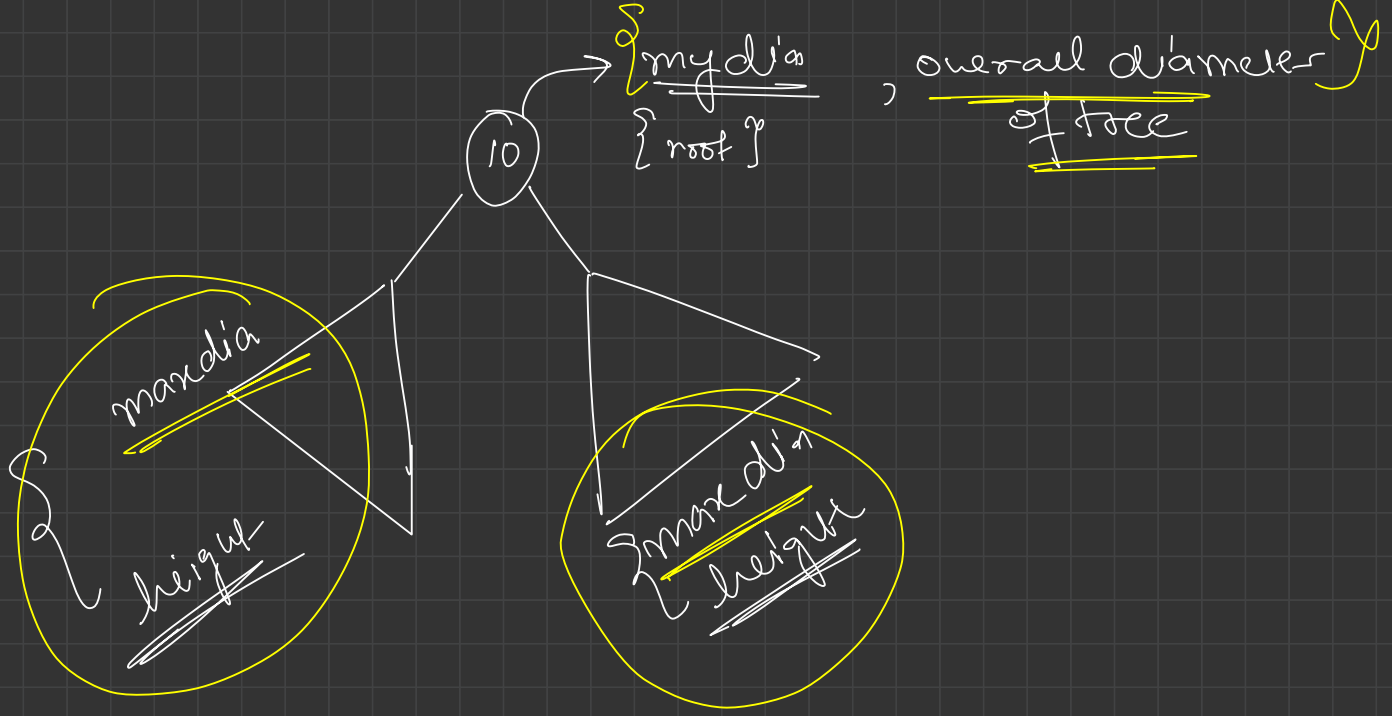
a b c



1 + 1 + 2 = 4

4 + 1 + 1 = 6




mydia
overall diameter of tree
{ root }
10
maxdia
height
max dia
height

Balanced Binary tree

↓

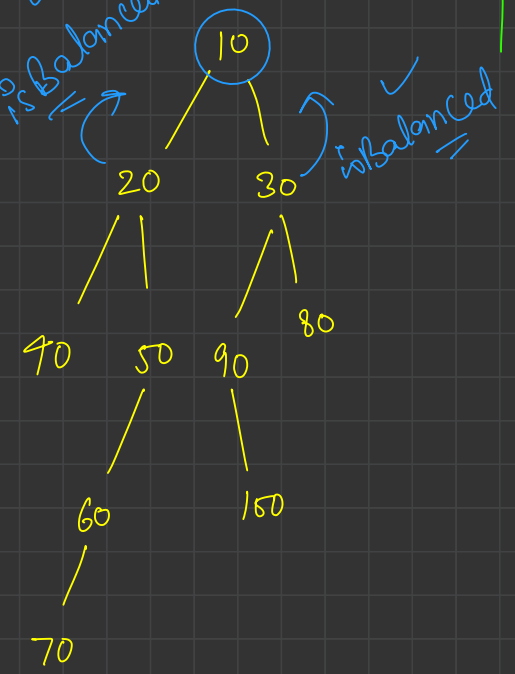
{when all node are balanced}

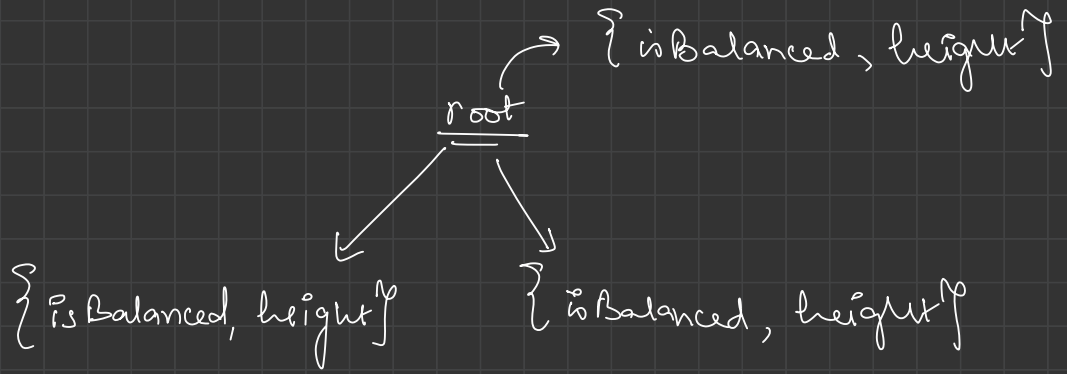
$|lh - rh| \leq 1$

{Node is Balanced

↓

$|lh - rh| \leq 1$




{ isBalanced , height }
root
{ isBalanced, height }
{ isBalanced , height }