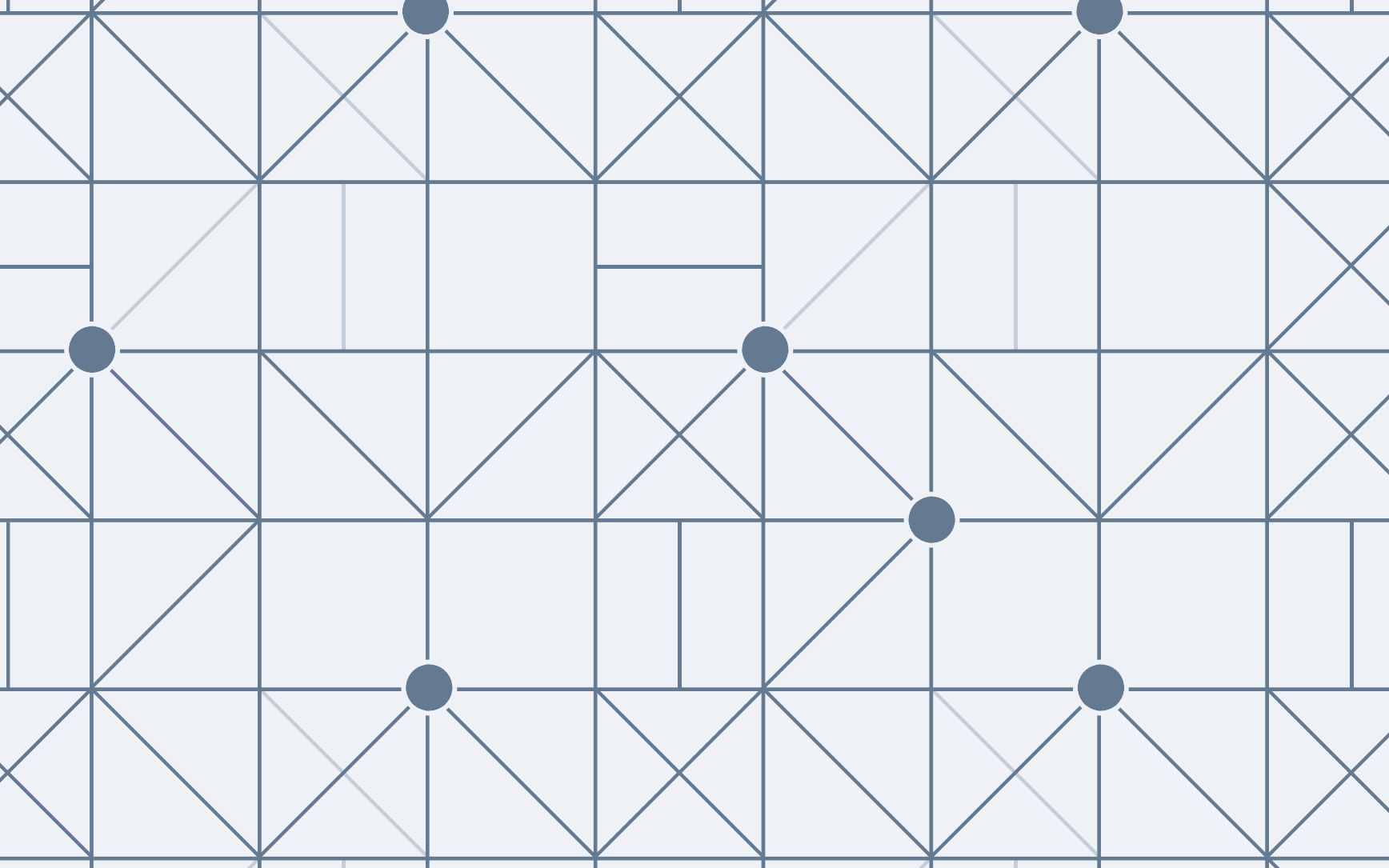
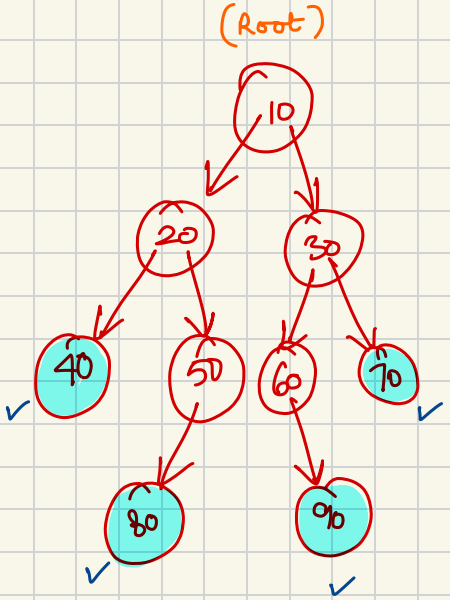


Diameter of a Binary Tree

{Max^m distance b/w any two leaf Nodes}

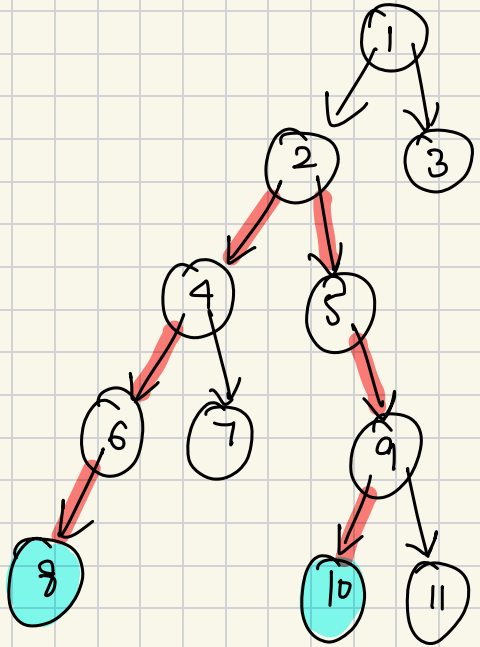


dist(40, 80) = 4
dist(40, 90) = 6
dist(40, 70) = 5
dist(80, 90) = 7
dist(80, 70) = 6
dist(90, 70) = 4

{diameter}

diameter = hLST + hRST + 1 → Wrong

NOTE: Actually this diameter passing root

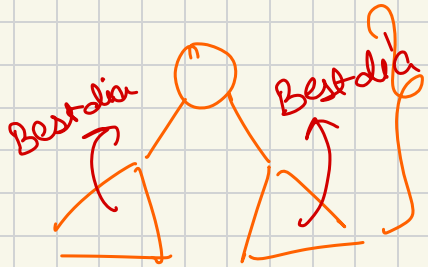


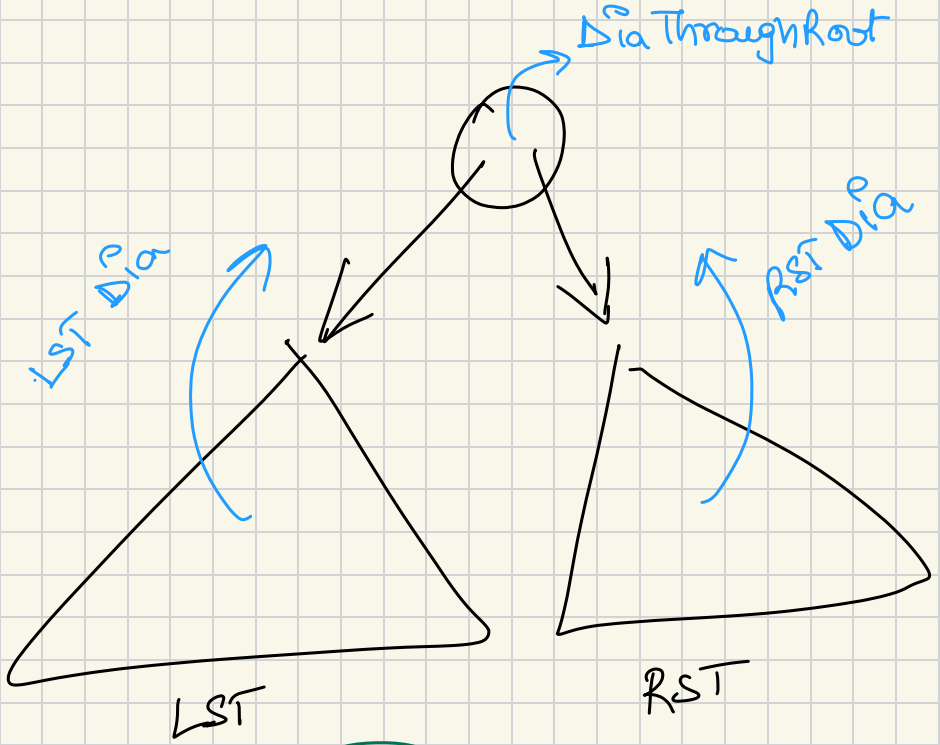
diameter = 7

NOTE: diameter not always passes through root

faith :- returns diameter starting from root

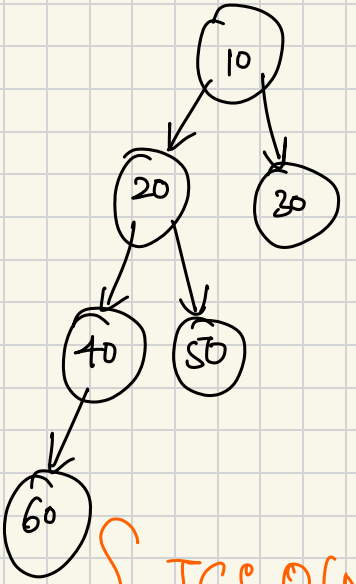
```
int diameter(Node root)
{
    Base Case !

    int LSTDia = diameter(root.left);
    int RSTDia = diameter(root.right);

    int diaThroughRoot = LSTh + RSTh + 1;
    return max{LSTDia, RSTDia, diaRoot};
}
```





$$dia = \max^{m}\{LST\ Dia, RST\ Dia, Dia\ Through\ Root\}$$

```java
// Faith: return diameter of the binary tree starting from root
public static int diameter(Node root) {
    // base case
    if (root == null) {
        return 0;
    }

    // get best diameter of left subtree
    int diaLST = diameter(root.left);

    // get best diameter of right subtree
    int diaRST = diameter(root.right);

    // get diameter passing through root
    int heightLST = heightOfTree(root.left);
    int heightRST = heightOfTree(root.right);
    int diaThroughRoot = heightLST + 1 + heightRST;

    // overall diameter of the tree
    return Math.max(diaThroughRoot, Math.max(diaRST, diaLST));
}
```



TC: O(N*N) ~ O(N^2)

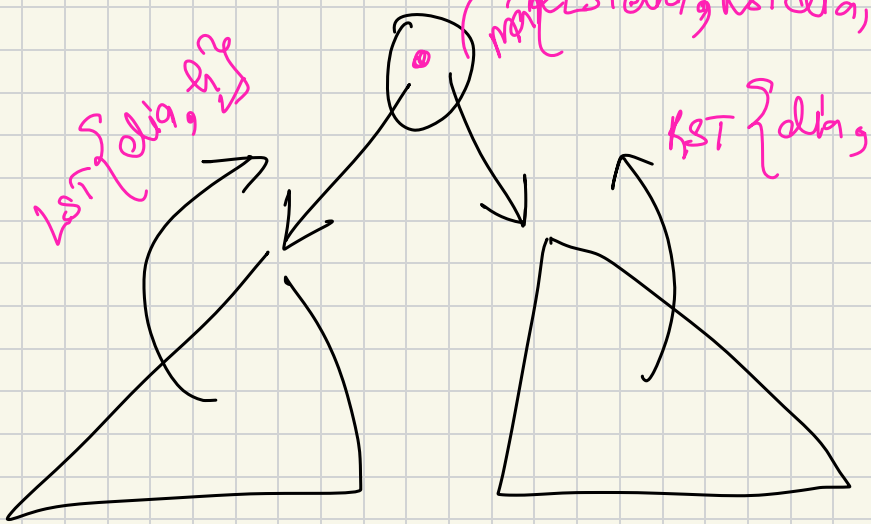
SC: O(H)

O(N^2)?

Better?

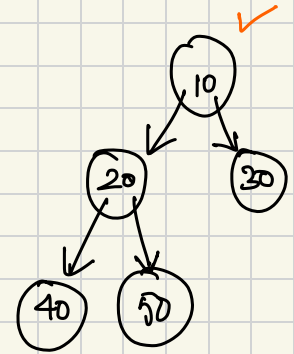
faith: return Dia and Heigut of the tree.

fun()



```java
// Faith: return diameter and height of the binary tree starting root
public static Pair helperFun(Node root) {
    // base case
    if (root == null) {
        return new Pair(0, 0);
    }

    // get the diameter and height of LST
    Pair LST = helperFun(root.left);

    // get the diameter and height of RST
    Pair RST = helperFun(root.right);

    // diameter of the tree -> Max(best in LST, best in RST, diameter through root)
    int diameterThroughRoot = LST.height + 1 + RST.height;
    int diameterOfTheTree = Math.max(diameterThroughRoot, Math.max(LST.diameter, RST.diameter));

    // height of the tree -> max (height of LST, height of RST) + 1
    int heighOfTheTree = Math.max(LST.height, RST.height) + 1;

    return new Pair(diameterOfTheTree, heighOfTheTree);
}
```

1 2 3 4 5



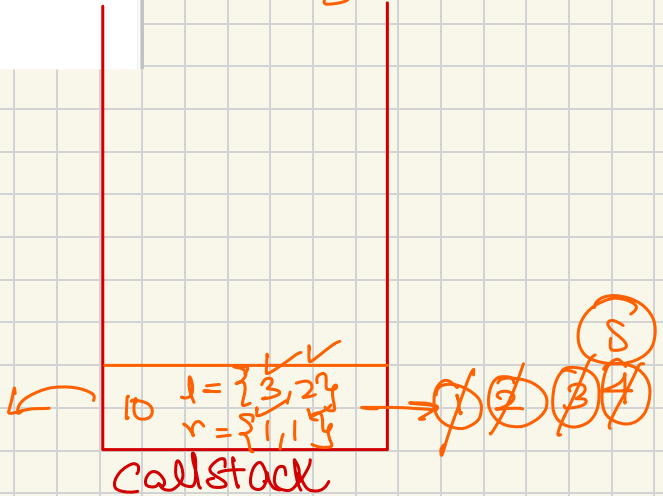
TC: O(N)
SC: O(H)

$x = 2 + 1 + 1 = 4$

$d = \{4, 3, 1\} = 4$

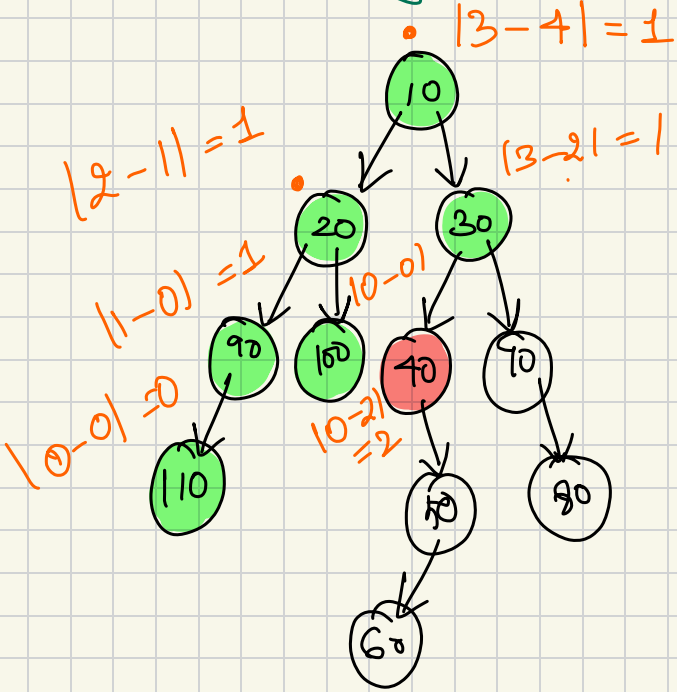
$h = \{2, 1\} + 1 = 3$

$\{4, 3\}$



Balanced Binary Tree

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



return false

NOT BALANCED!

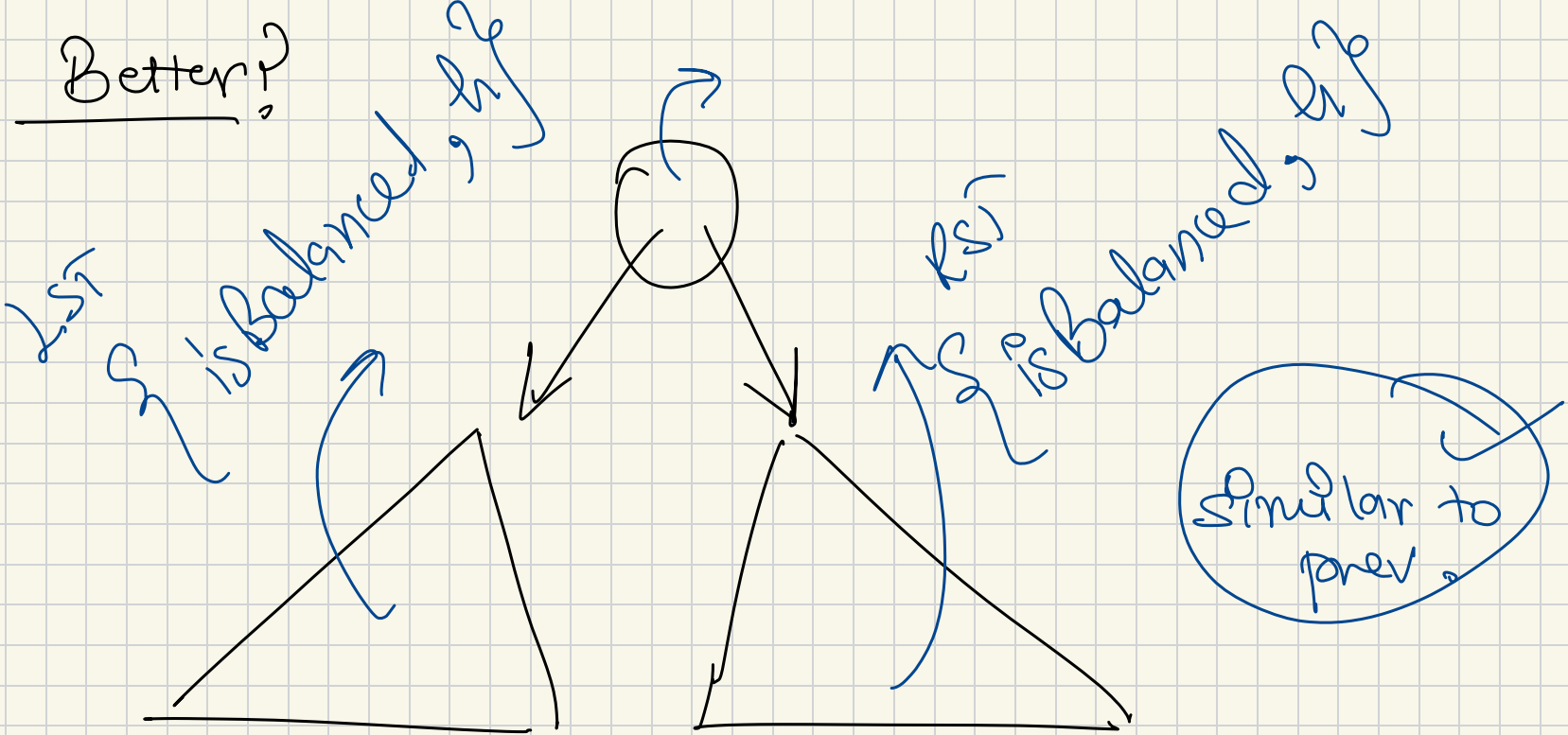
Faith: returns is Tree Balanced, from Root

```
Boolean isBalanced(Node Root)
{
    Boolean isLST = isBalanced(Root.left);
    Boolean isRST = isBalanced(Root.Right);

    int hLST = height(Root.left);
    int hRST = height(Root.right);
    int AbsDiff = |hLST - hRST|;
}
```

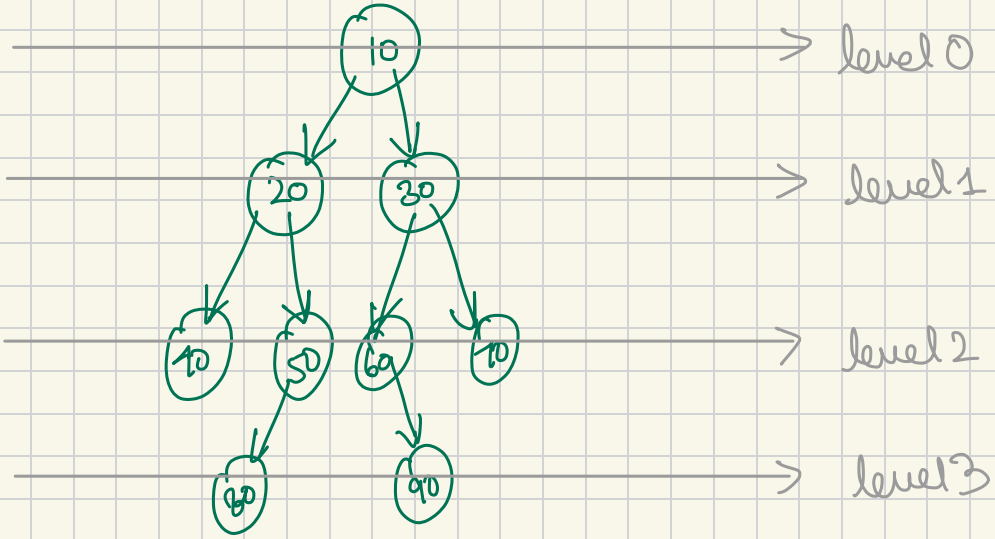
TC: $O(N^2)$
SC: $O(1)$

Check Root is Balanced OR not

Better?

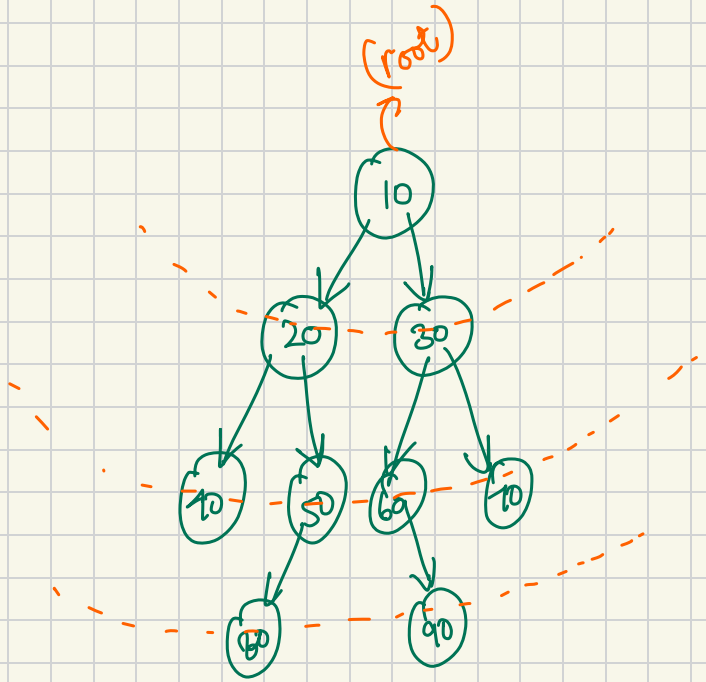


Level Order Traversal.



o/p

10
20 30
40 50 60 70
80 90



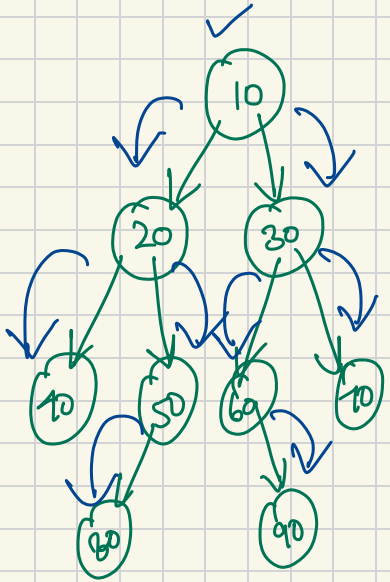
10

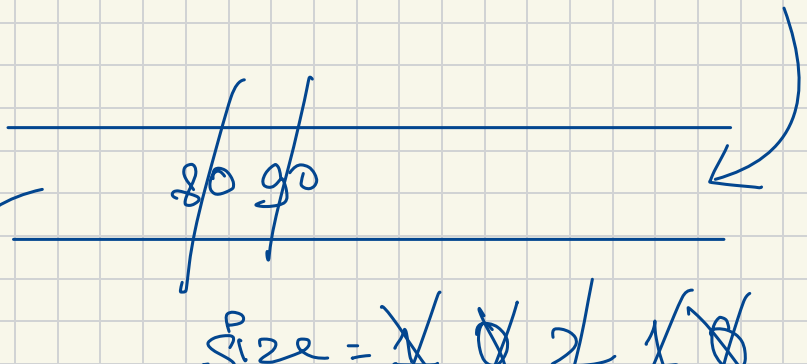
20 30

10 50 60 10

80 90

traverse vaidically : BFS





Size = 1 0 2 1 0

1 2 2 1 0

2 1 0

(Level Order Traversal)

10

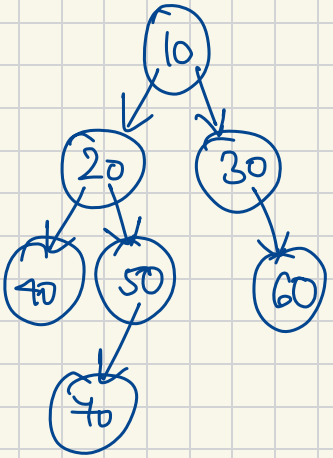
20 30

40 50 60 70

80 90

TC : O(N) SC : O(2^h)

```java
public void levelOrderTraversal(TreeNode root) {
    if (root == null) {
        return;
    }

    Queue<TreeNode> que = new ArrayDeque<>();
    que.add(root);

    while (que.size() != 0) {
        int size = que.size();
        while (size-->0) {
            TreeNode rnode = que.remove();

            System.out.print(rnode.val + " ");

            if (rnode.left != null) {
                que.add(rnode.left);
            }

            if (rnode.right != null) {
                que.add(rnode.right);
            }
        }
        System.out.println();
    }
}
```



o/p

10

20 30

40 50 60

70

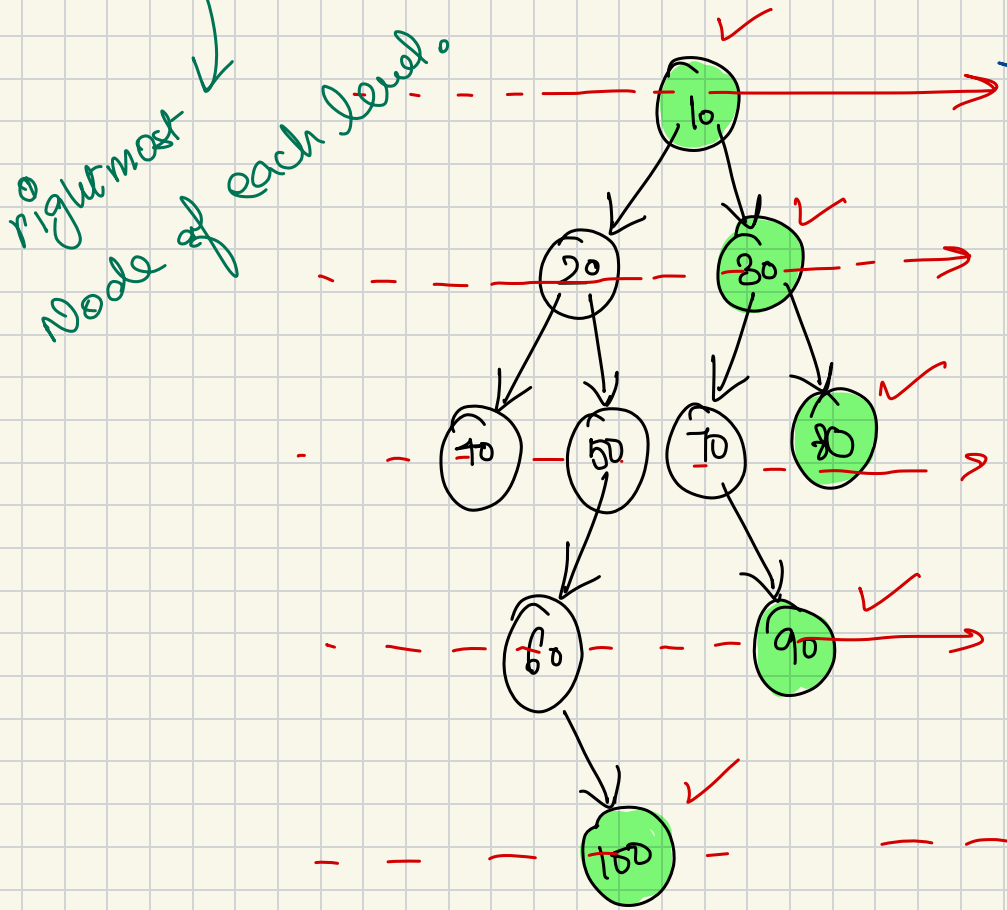
que

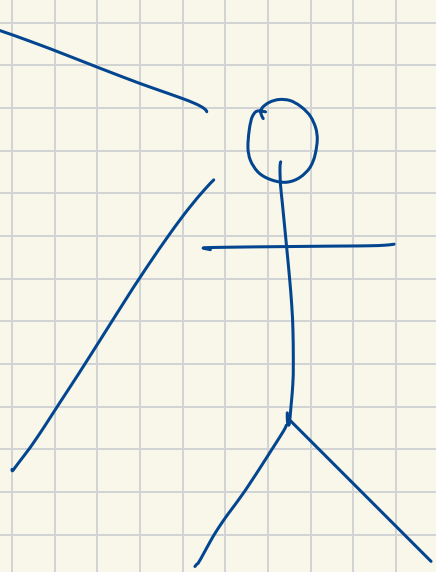
size = 0

Right View of Binary Tree

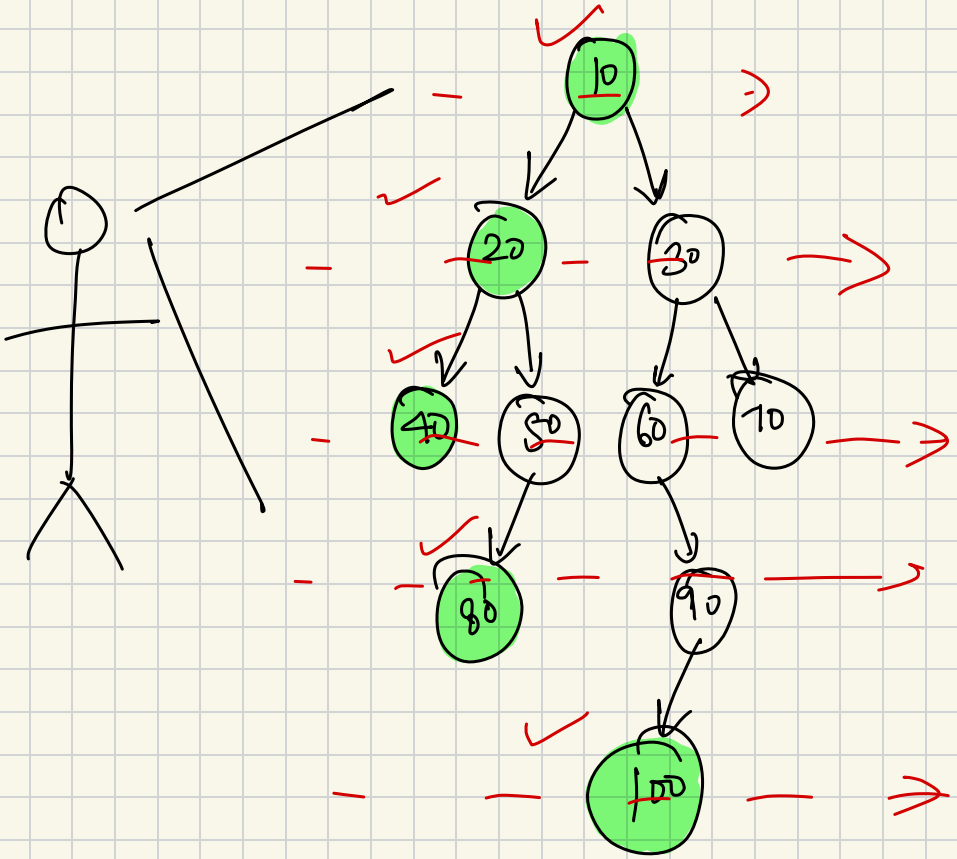
Right view: 10 30 80 90 100

rightmost Node of each level.

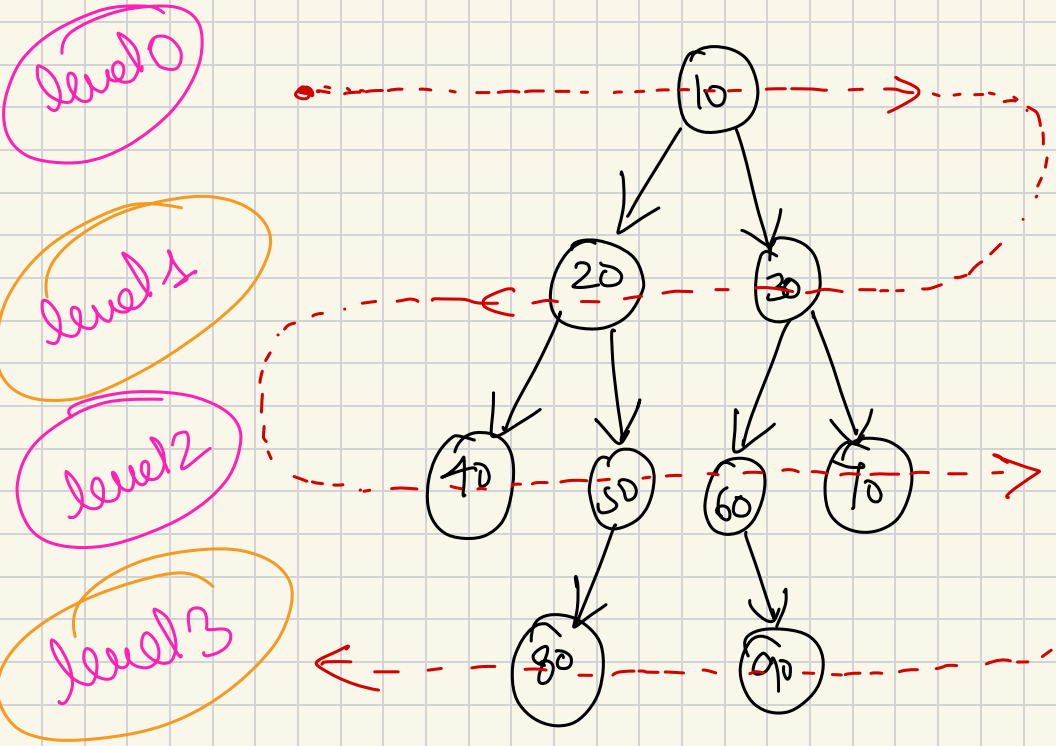




Left view



Zig Zag traversal



O/P

10

30 20

40 50 60 70

90 80

even levels : Left to Right

odd levels : Right to Left