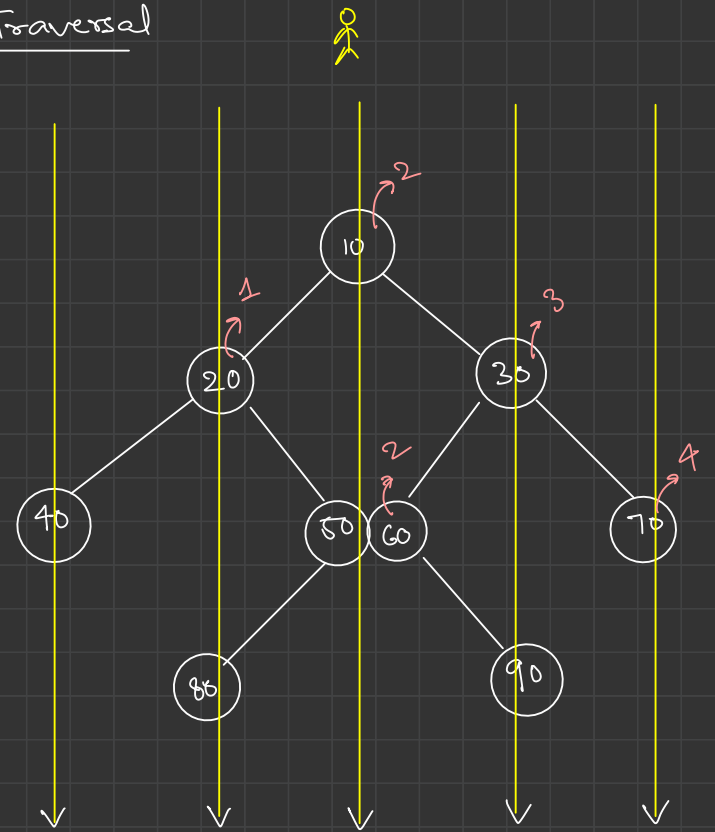
Vertical Order Traversal

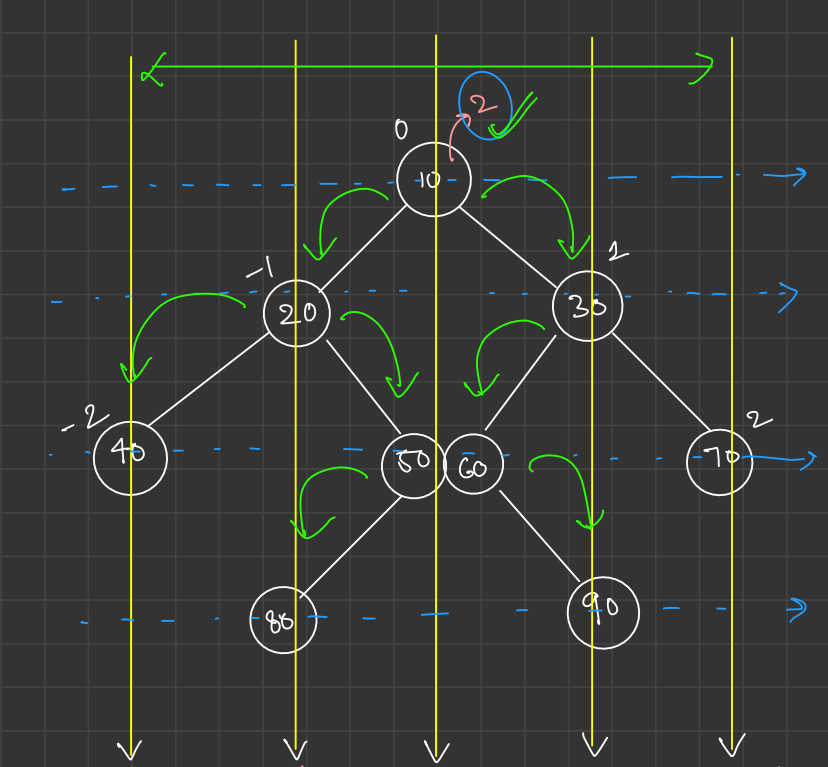


V.O. level
Root (l)

(l-1) left child

(l+1) Right child

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



Node v.s. level

(10,2) (20,1) (30,3) (40,0) (50,2)

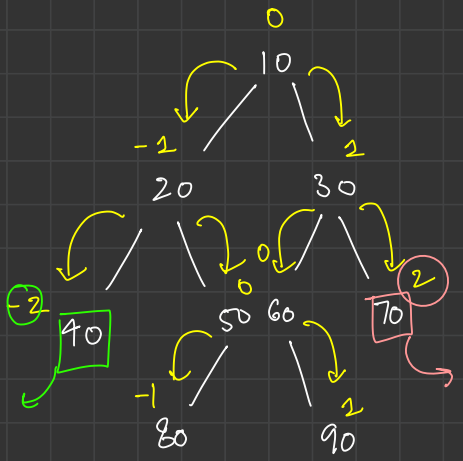
queue

(60,2) (70,4) (80,1) (90,3)

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

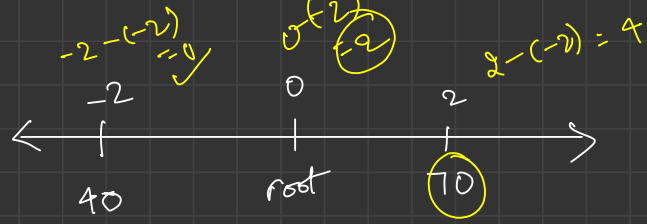
for the left Most Node in tree

for rigut most Node in a tree



left Most Node

right Most Node



No. of v. levels = rightMost Idx − leftMost Idx + 1

= 2 − (−2) + 1 = 5 levels

↳ {0, 1, 2, 3, 4}

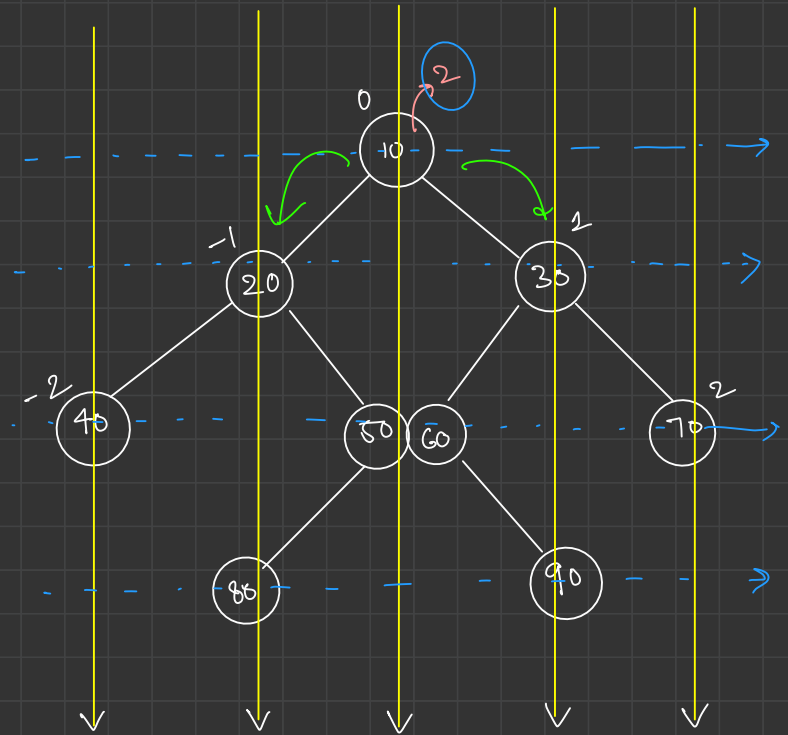
V.O. level of root = 0 − (leftMost Idx)

= − leftMost Idx

{ we want some our own set of rules, applied to que, so min value exist first when two level, and level is same for 2 nodes

priority queue

↳ Our own custom priority !



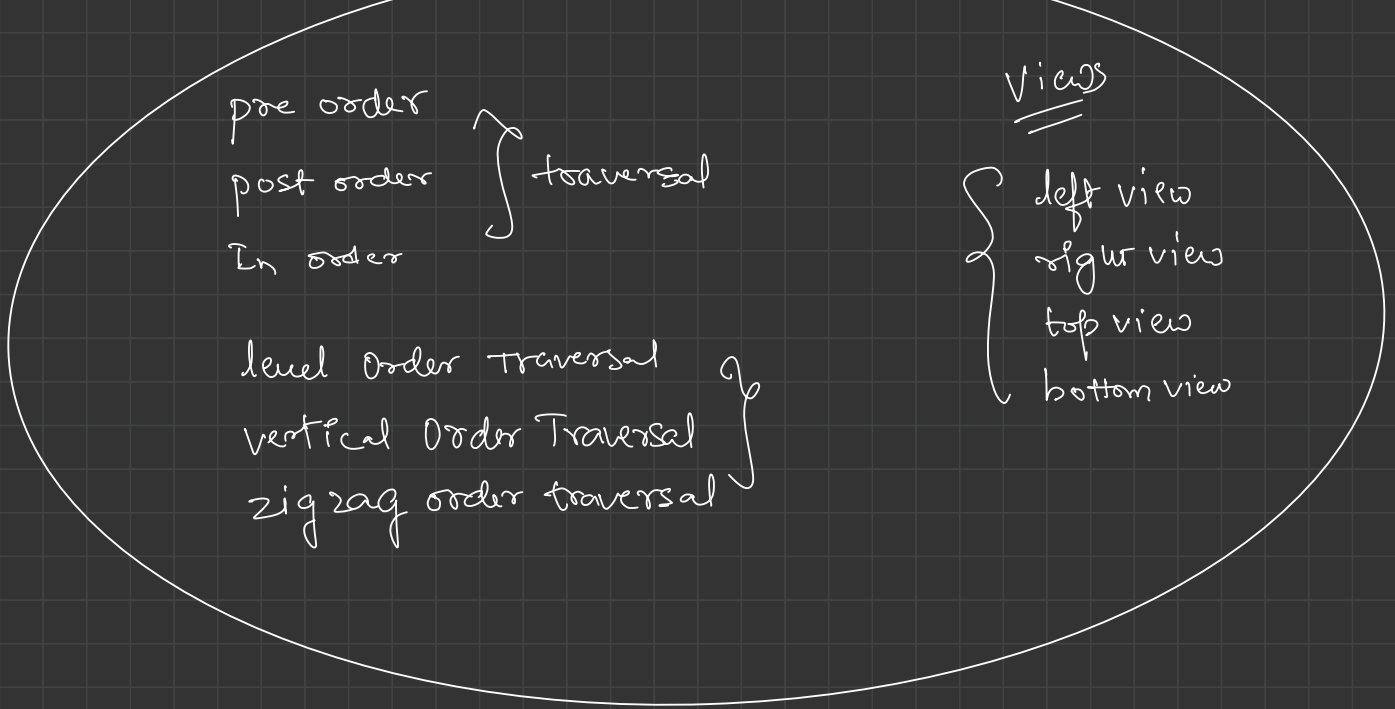
(80,1) (90,3)

pq

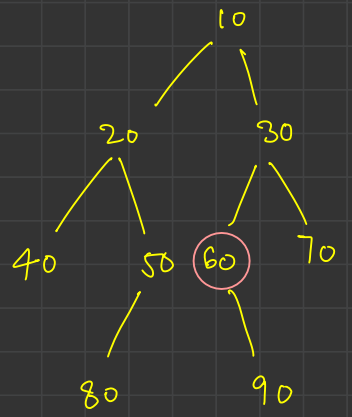
temp

Rules

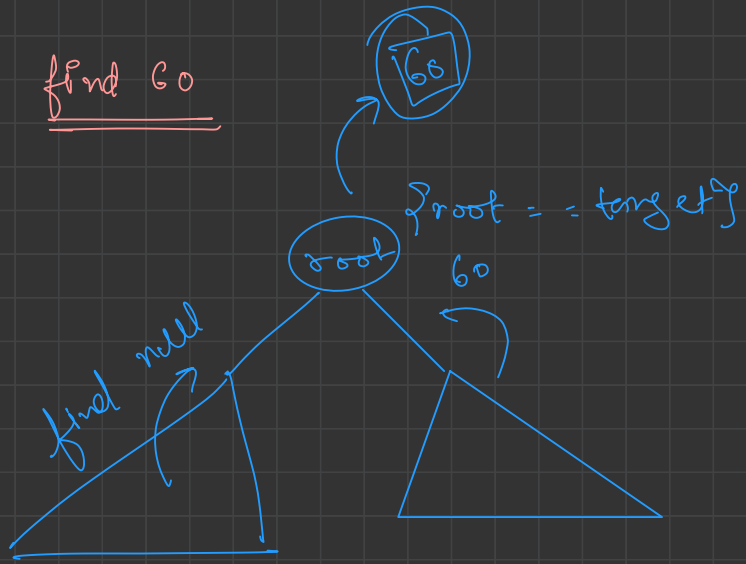
- Remove people with less v.o. level first.
- If two nodes have same v. level remove person with smaller value


pre order
post order
In order
traversal
level Order Traversal
vertical Order Traversal
zig zag order traversal
Views
left view
right view
top view
bottom view

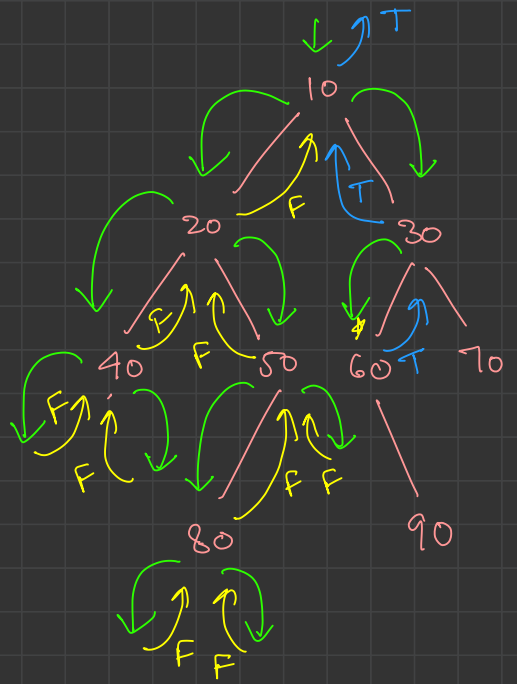
find a given node in a tree



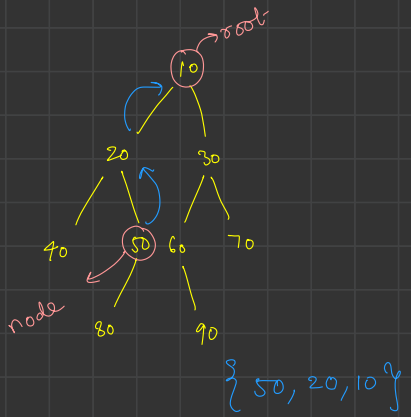
find 60



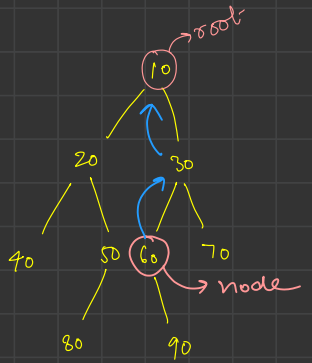
```
boolean find (Node root, int target)
{
① if (root == null) return false;
② if (root.data == target) return true;
③ boolean fldc = find (root.left, target);
④ if (fldc == true) return true;
⑤ boolean frgt = find (root.right, target);
⑥ if (frgt == true) return true;
⑦ return false;
}
```



Node To Root Path

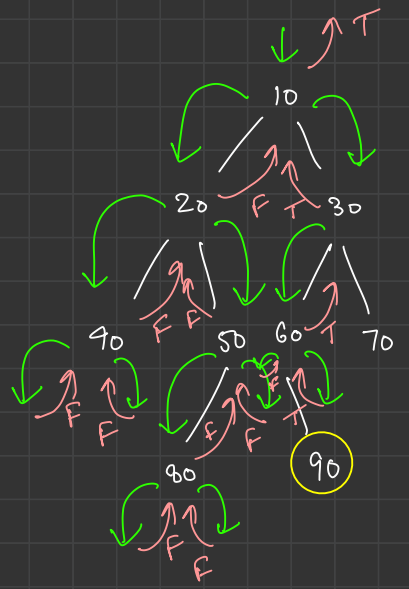


{50, 20, 10}



{60, 30, 10}

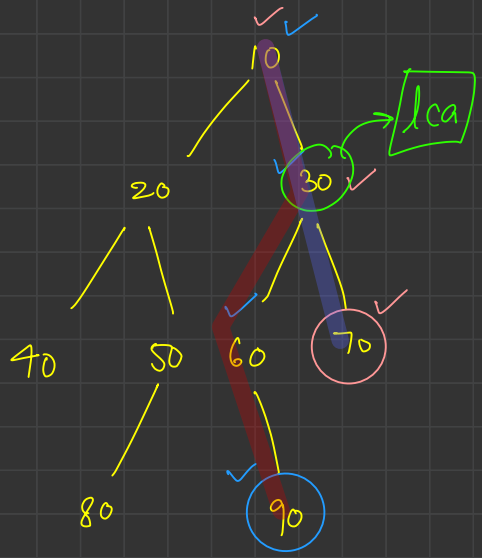
```java
ArrayList<Integer> n2r = new ArrayList<>();

boolean node2rootPath (Node root, int target) {
    if (root == null) {
        return false;
    }

    if (root.data == target) {
        n2r.add(root.data);
        return true;
    }

    boolean filc = node2rootPath(root.left, target);
    if (filc == true) {
        n2r.add(root.data);
        return true;
    }

    boolean firc = node2rootPath(root.right, target);
    if (firc == true) {
        n2r.add(root.data);
        return true;
    }

    return false;
}
```



{ 90, 60, 30, 10 }

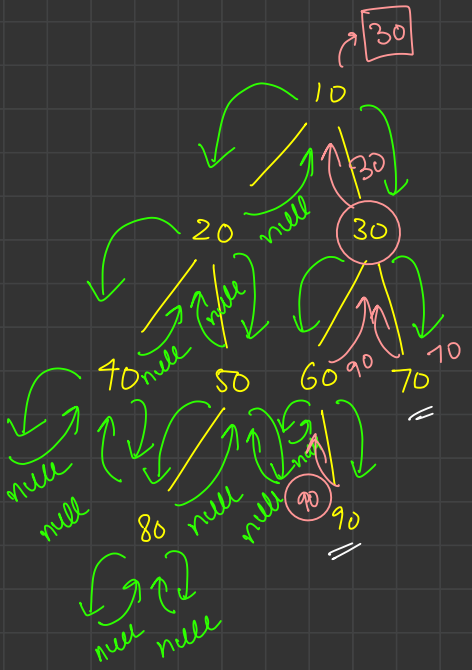
N2R

lca (lowest Common ancestor)



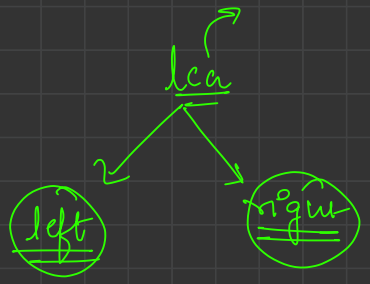
90 → {90, 60, 30, 10}

70 → {70, 30, 10}



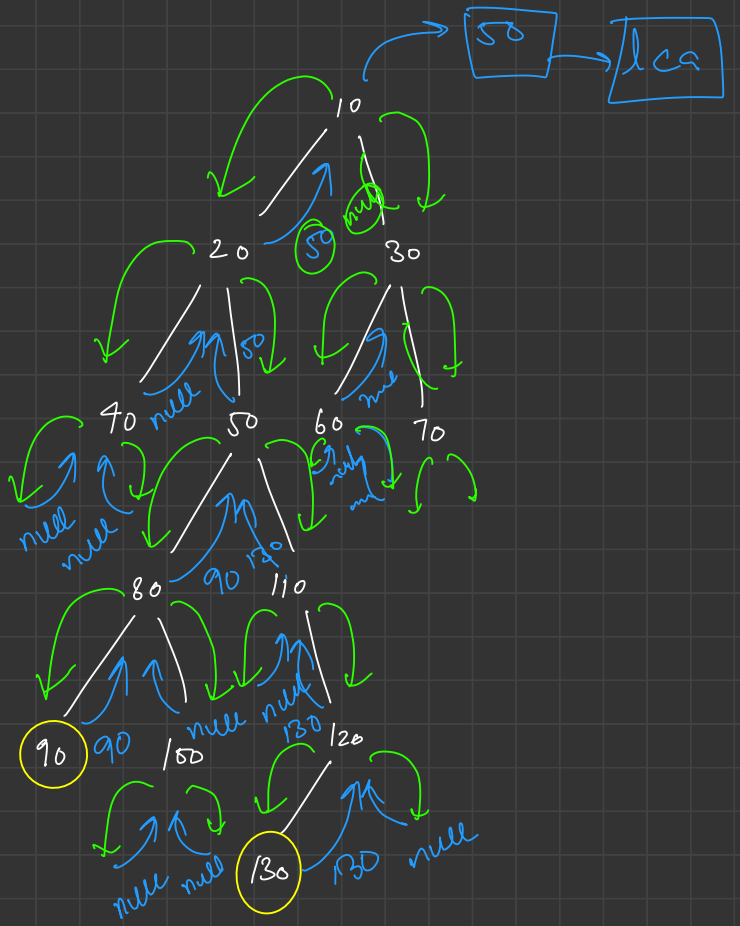
(n1 = 90, n2 = 70)

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

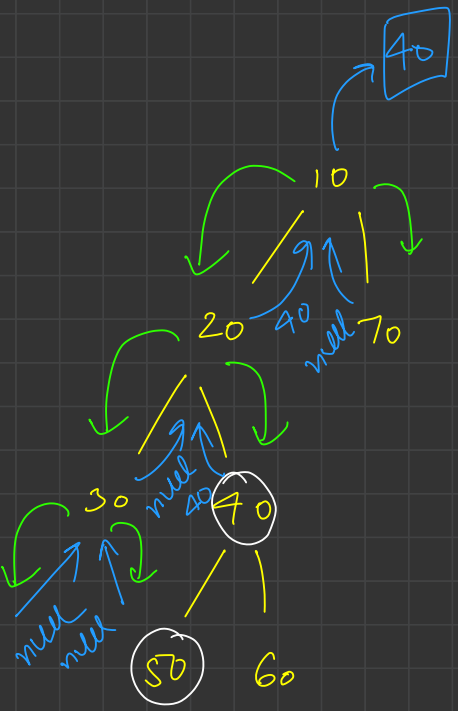


4 types of code

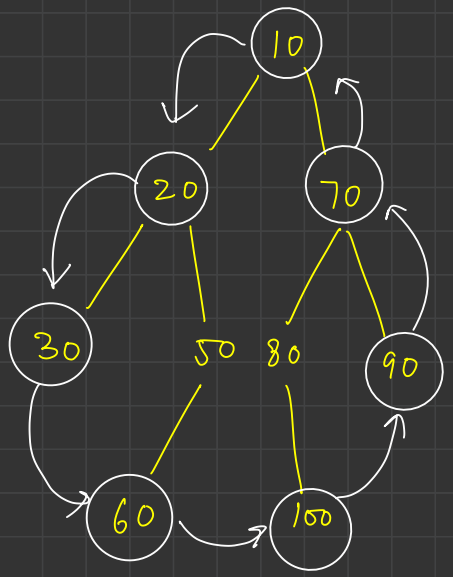
```java
public static Node findLCA(Node node,int n1,int n2) {
    if (node == null) {
        return null;
    }

    if (node.data == n1 || node.data == n2) {
        return node;
    }

    Node lc = findLCA(node.left, n1, n2);
    Node rc = findLCA(node.right, n1, n2);

    if (lc != null && rc != null) {
        return node;
    }

    if (lc != null) {
        return lc;
    }

    if (rc != null) {
        return rc;
    }

    return null;
}
```



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

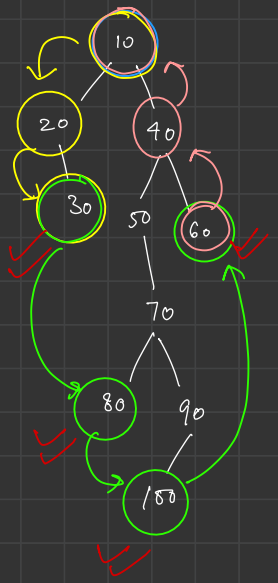

40
10
20
40
null
70
30
null
40
40
null
null
50
60

Boundry Traversal



{10, 20, 30, 60, 100, 90, 70, 10}

Boundry



{10, 20, 30, 80, 100, 60, 40, 10}

Boundry of a tree

{ root + left Wall + Bottom wall + Right wall }

leaf Nodes from left 2 right

go (left)
if left Not there they only go (right)

before call

go (right)
if right is not there then only go (left)

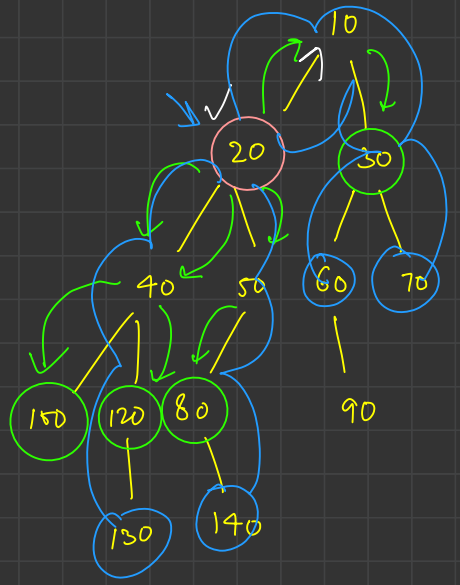
add after call

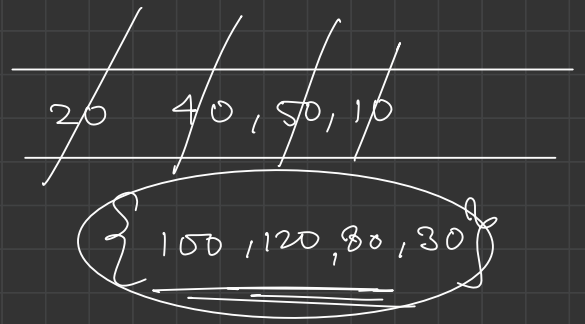
All Nodes Distance K Away

Node, Node

20 → 10

K = 2





130, 140, 60, 70

{100, 120, 80, 30}