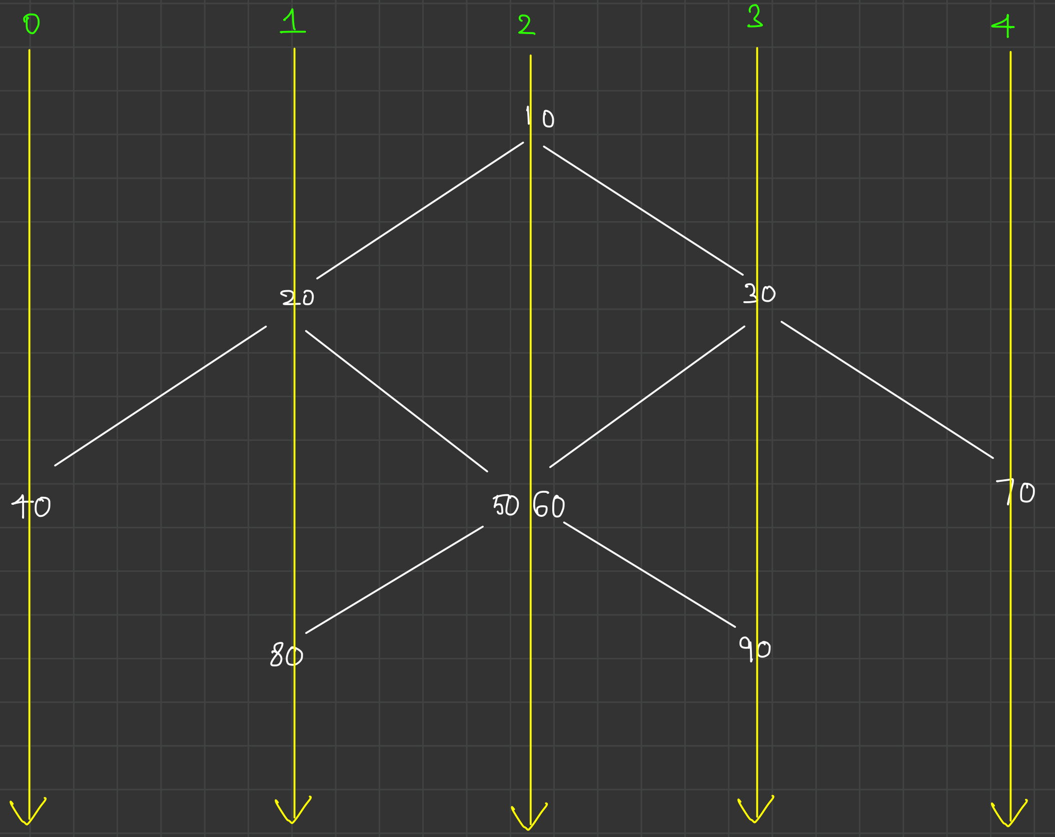
Vertical Order Traversal



o/p

40

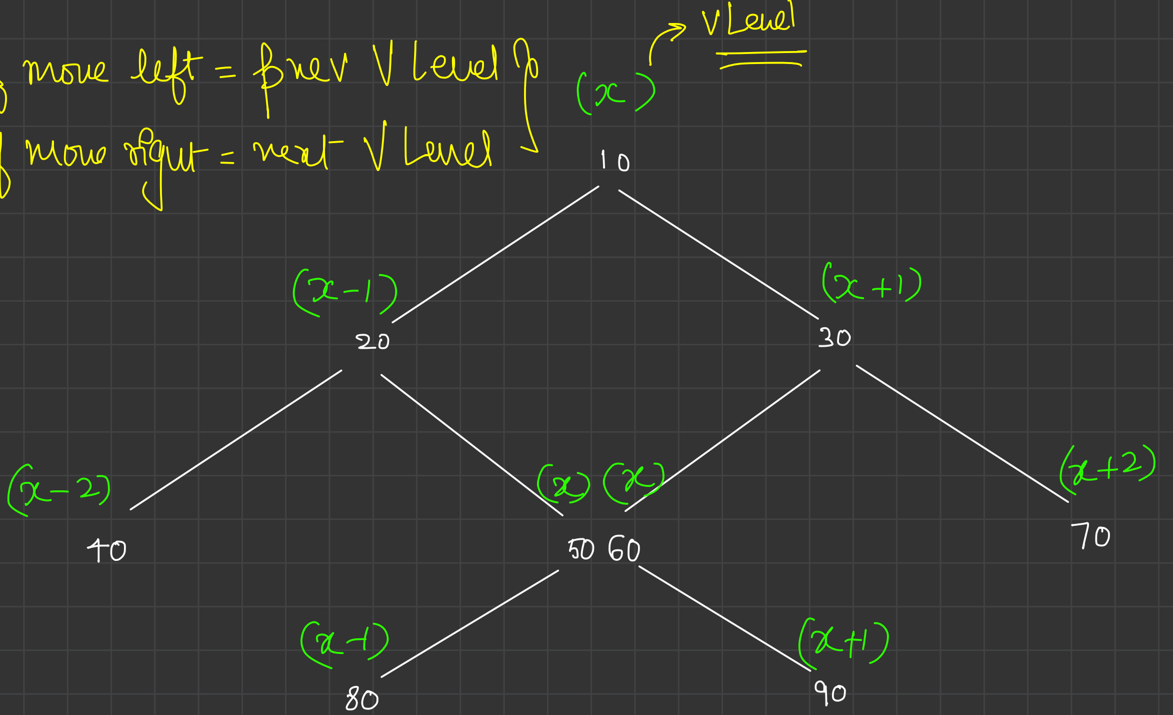
20 80

10 50 60

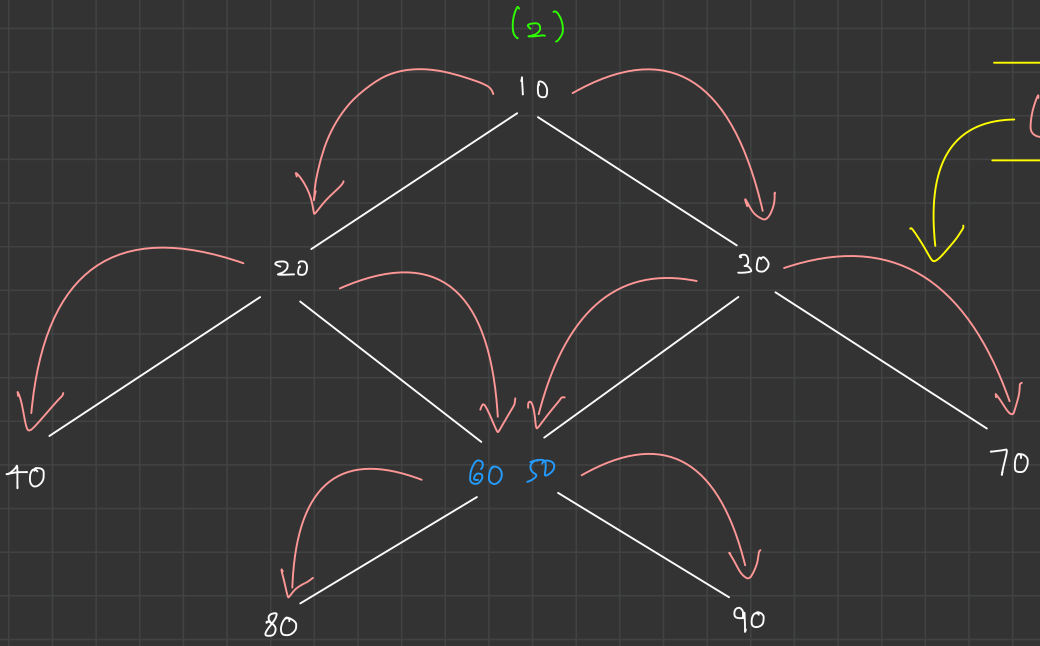
30 90

70

if move left = prev V Level }
if move right = next V Level



40
20 80
10 50 60
30 90
70



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

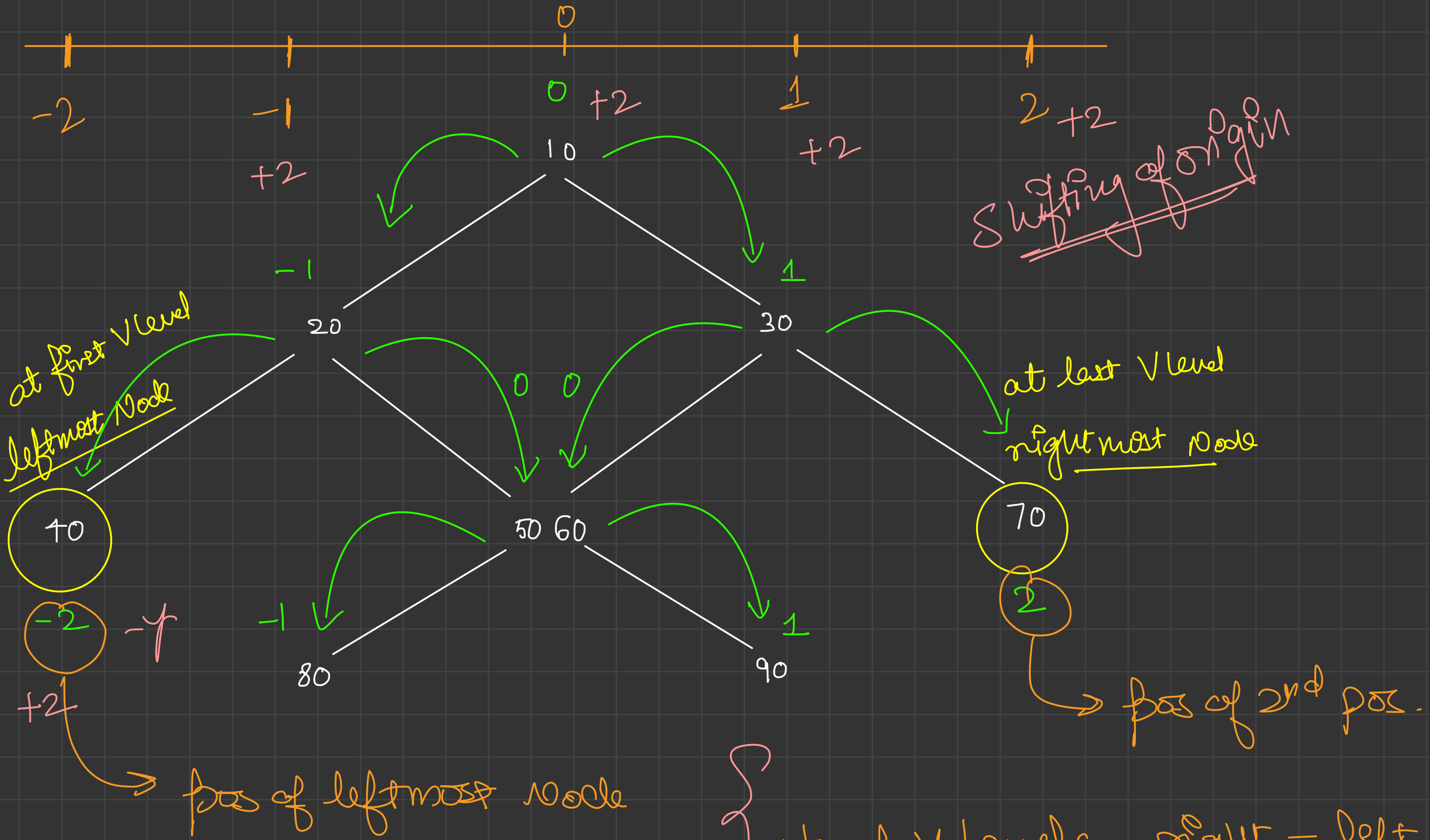
0 →	40
1 →	20, 80
2 →	10, 50, 60
3 →	30, 90
4 →	70

① remove leftmost Node first

② If same v level, then remove person with smaller level first

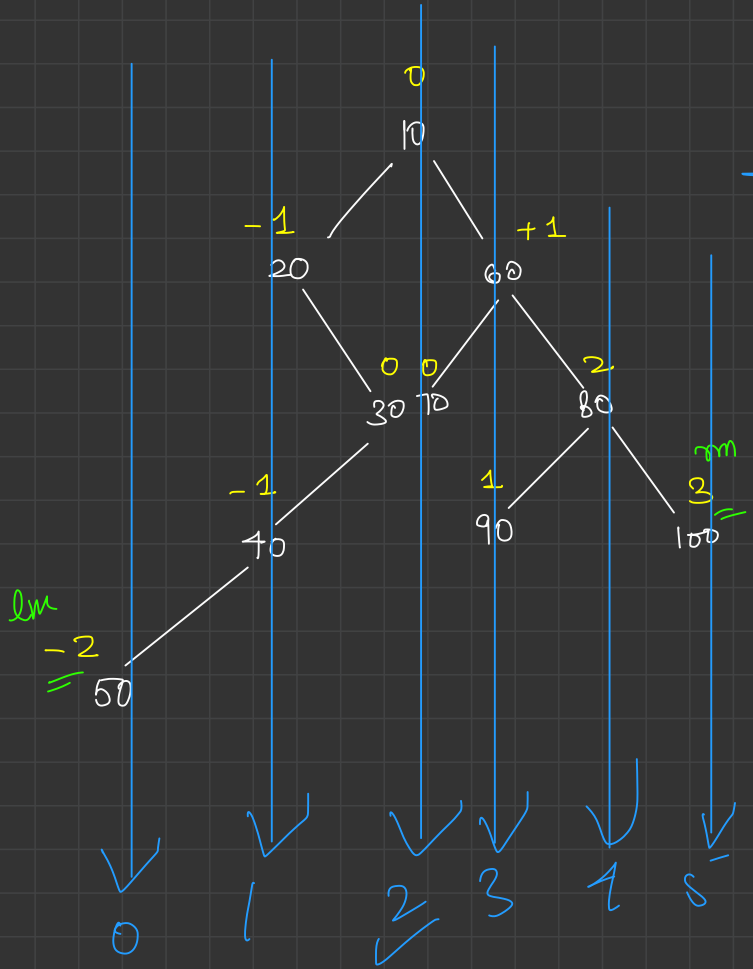
custom sorting rule

→ queue {Priority Queue}



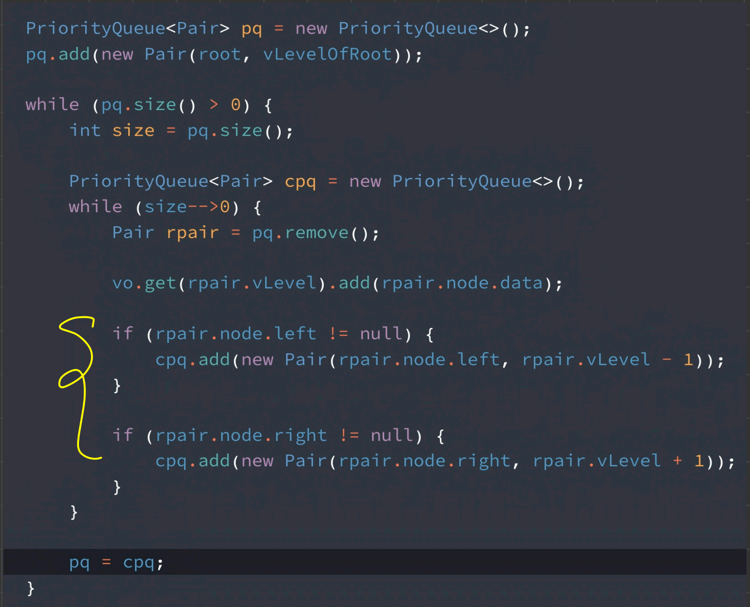
No. of V Levels = right - left + 1

Vlevel pos of root = - leftmost pos.

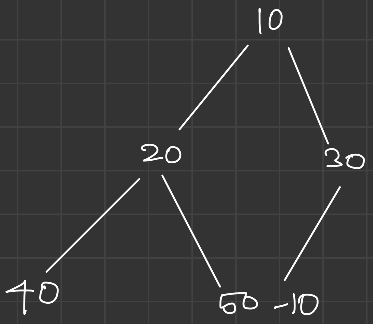


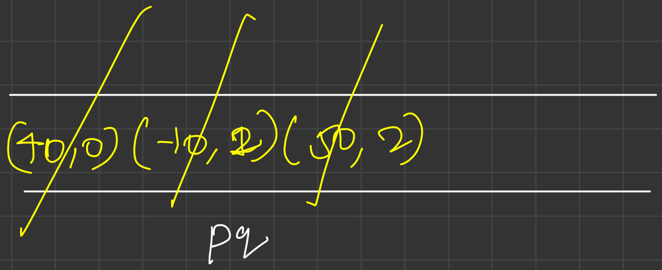
no. of Vlevels = $3 - (-2) + 1 = 6$

Vlevel of root $= -(-2) = 2$



```java
PriorityQueue<Pair> pq = new PriorityQueue<>();
pq.add(new Pair(root, vLevelOfRoot));

while (pq.size() > 0) {
    int size = pq.size();

    PriorityQueue<Pair> cpq = new PriorityQueue<>();
    while (size-->0) {
        Pair rpair = pq.remove();

        vo.get(rpair.vLevel).add(rpair.node.data);

        if (rpair.node.left != null) {
            cpq.add(new Pair(rpair.node.left, rpair.vLevel - 1));
        }

        if (rpair.node.right != null) {
            cpq.add(new Pair(rpair.node.right, rpair.vLevel + 1));
        }
    }

    pq = cpq;
}
```



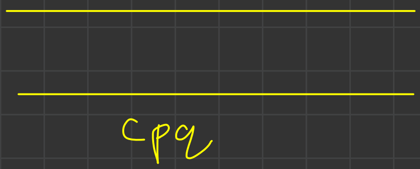


0 → 40

1 → 20

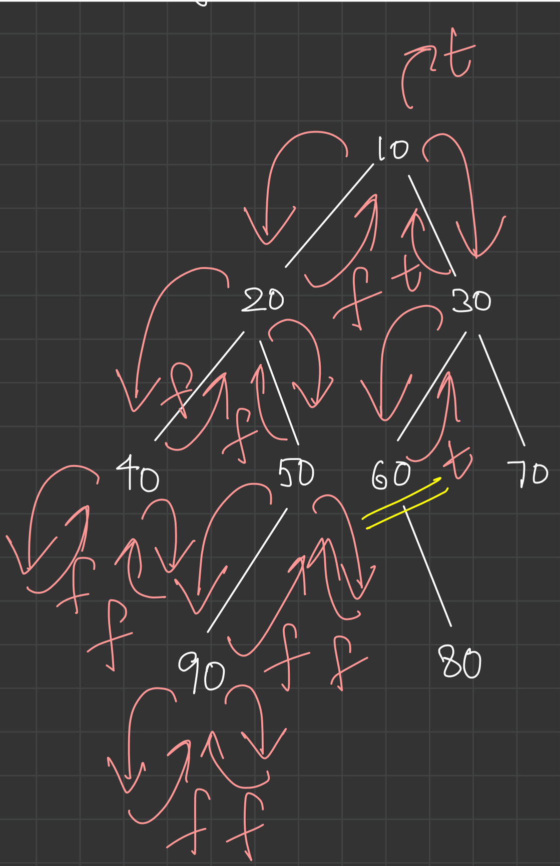
2 → 10, -10, 50

3 → 30



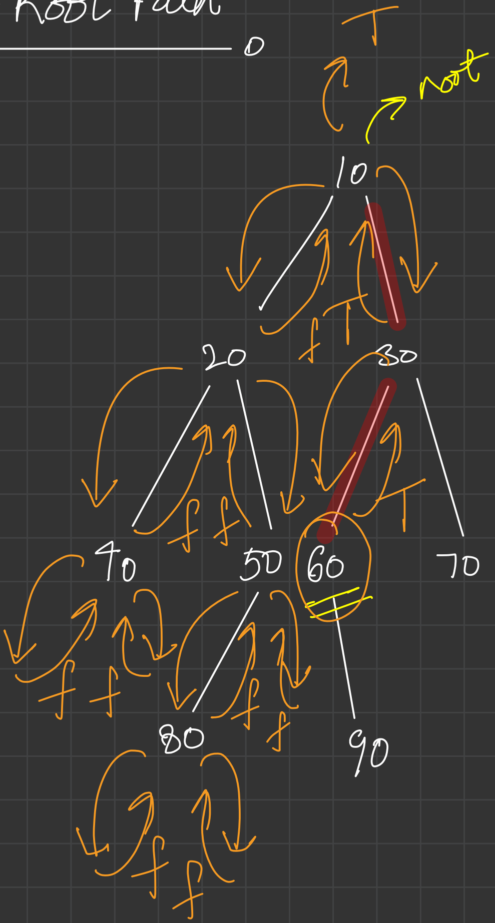
find a given node in a tree

faith : tells weather val is present in the tree starting from root .



```
Boolean find (Node root, Int val)
{
    if(root = = null)
            return false ;

    if (root.data = = val)
        return true;

    boolean filc = find (root.left, val);
    if (filc) return true;
    boolean firc = find(root.right,val)
    if(firc) return true,

return false;
}
```

Node to Root Path

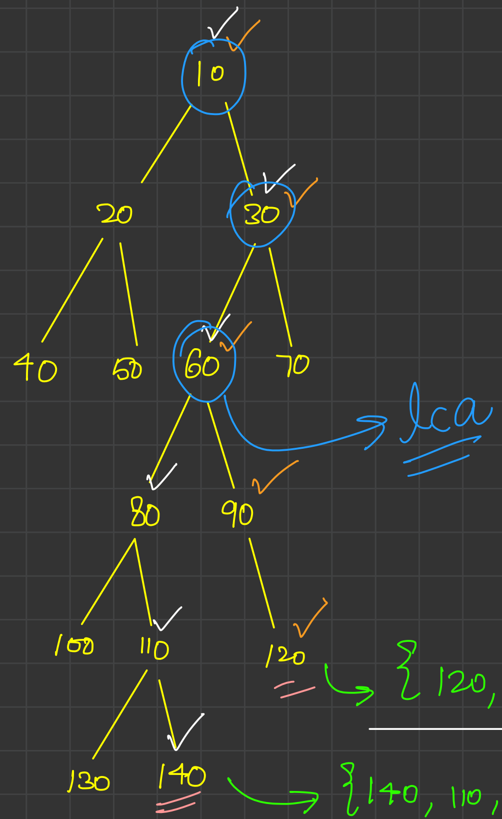


{60, 30, 10}

↳ path

{60, 30, 10}

LCA { lowest Common Anscestor }



lca = ~~10~~ ~~30~~ $\boxed{60}$ ✓

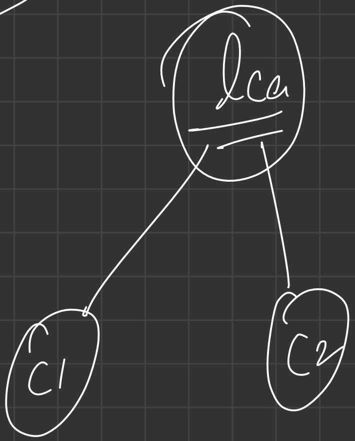
i

{120, 90, 60, 30, 10} → N2R

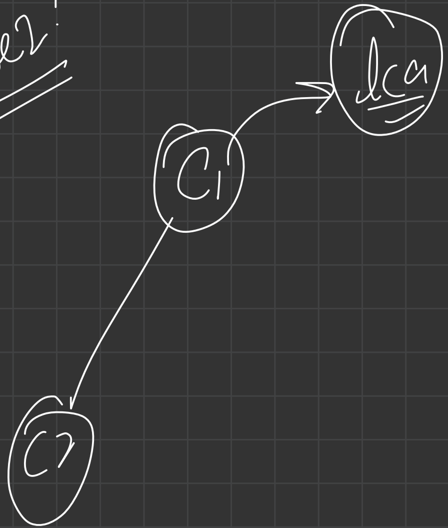
j

{140, 110, 80, 60, 30, 10} → N2R

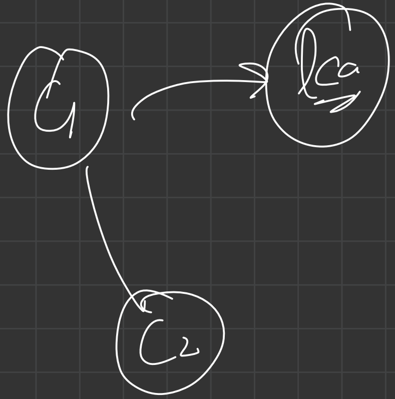
Case1:

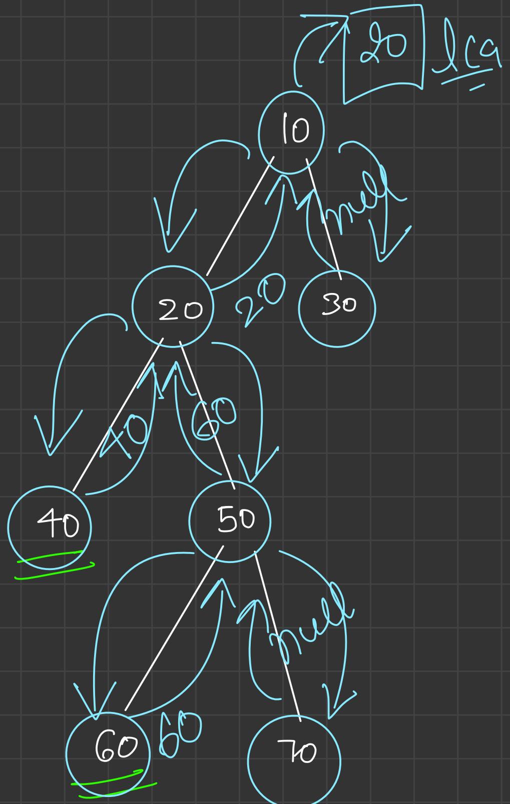


Case2:



Case3:

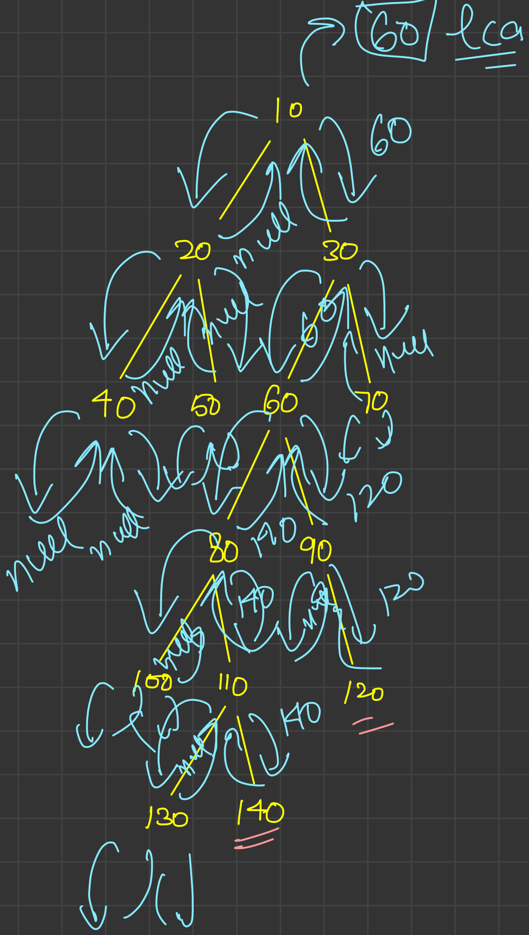




faith: it will return LCA of n1 & n2

Node findLCA(root, n1, n2)

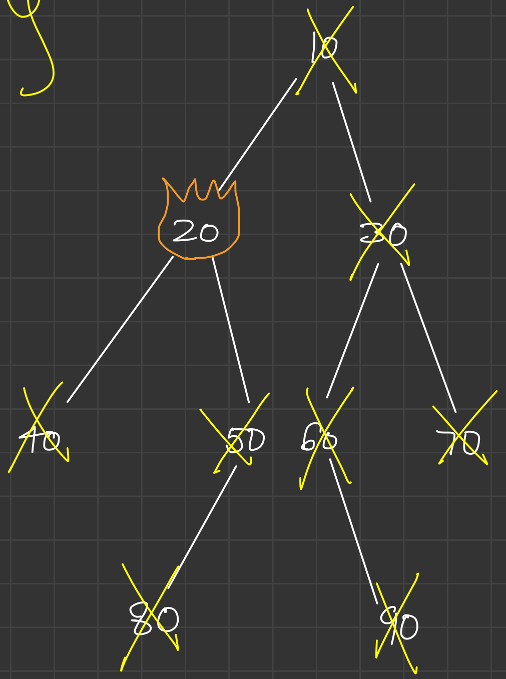
40 60


60
lca
10
60
20
null
30
null
null
60
null
40
null
50
60
70
null
null
120
80
140
90
140
null
120
100
110
120
140
null
130
140

Time to burn tree

BFS

{ min time to burn } = 4sec

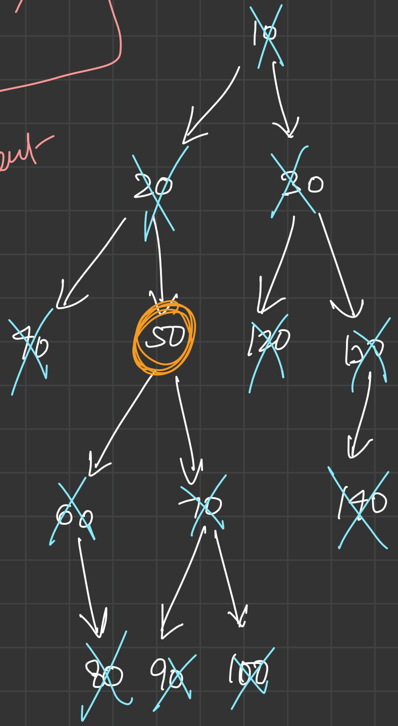


time = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

HashMap< Node, Node >

child parent

vis → HashSet



~~40~~, ~~10~~, ~~80~~, ~~90~~, ~~100~~, ~~30~~, ~~120~~, ~~130~~, ~~140~~

level = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

time = level − 1 = 5 sec