- Zig Zag traversal
- find a node
- path to given Node
- LCA
- Vertical order traversal

Zig zag traversal

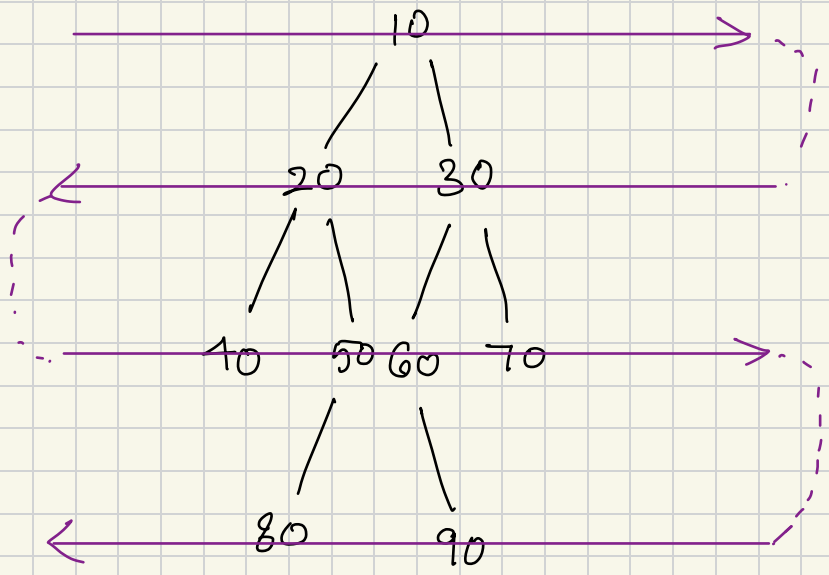
Level

✓ 0

1

✓ 2

3



10

30 20

40 50 60 70

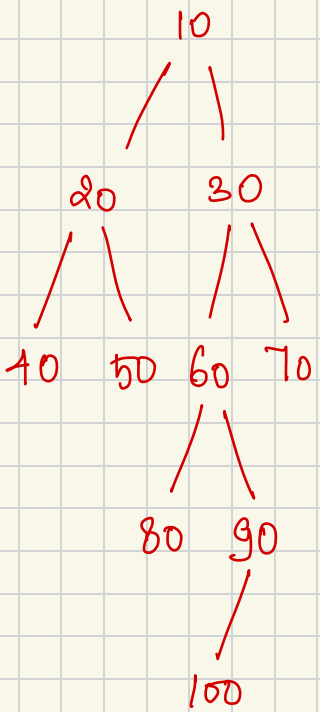
90 80

even level : Left to right

odd level : right to left

```
int level = 0;
while (que.size() != 0)
{
    ArrayList<Integer> currLevel;
    int size = que.size();
    while (size > 0)
    {
        Node node = que.remove()
        currLevel.add(node)
    }
    if (level) ⟶ reverse(0 to k)
}
levelPe;
```

find a Node



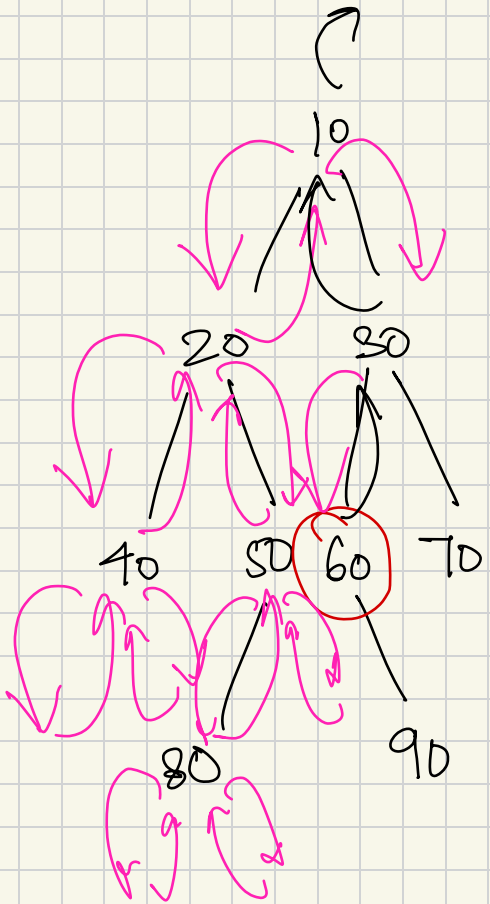
int target = 90 ✓

faith: findNode, with val == target

```
Node find(Node root, int target)
{
    Base case
    if (root.data == target) return root;

    Node file = find(root.left, target);
    if (file != null)
        return file;

    Node fire = find(root.right, target);
    if (fire != null)
        return fire;
```


```
Node find(Node root, int tar)
{
  ① if (root == null)  return null;
  ② if (root.data == tar) return root;
  ③ Node filc = find(root.left, tar);
    if(filc != null) return filc;
  ④ Node firc = find(root.right, tar);
    if(firc != null) return firc;

  return null;
}
```

Path to a given Node



Path: 10 → 30 → 60

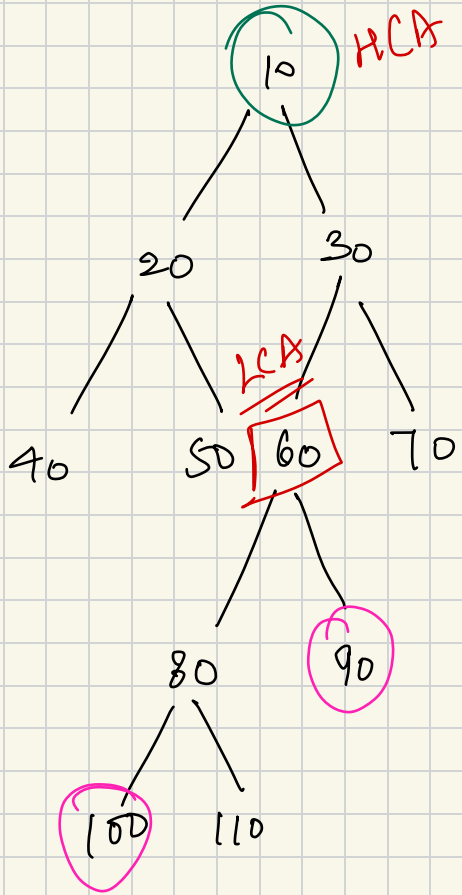
{60, 30, 10}

reverse

10, 30, 60

root to Node path.

Least Common Ancestor (LCA)

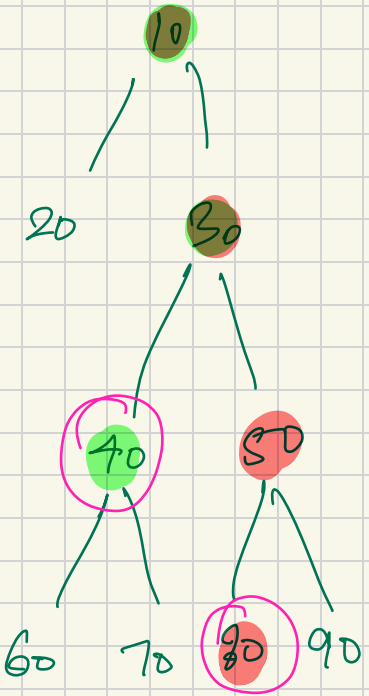


100 : { 100, 80, 60, 30, 10 }

90 : { 90, 60, 30, 10 }

LCA : 60 ✓

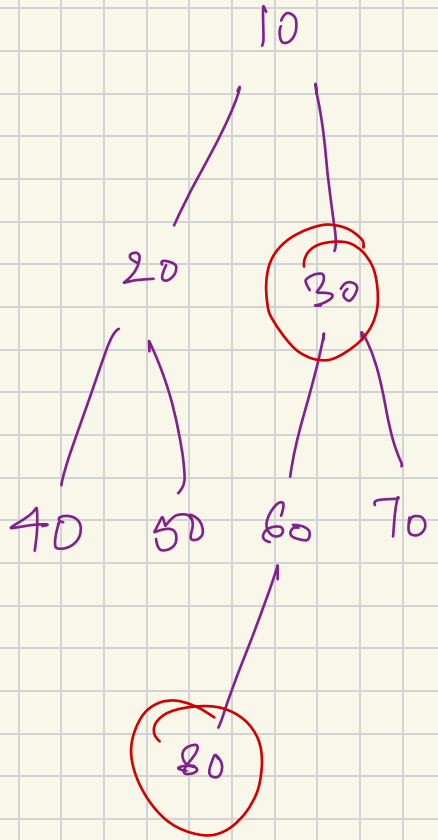
Node findLCA(Node root, int a, int b)



40: {40, 30, 10}

80: {80, 50, 30, 10}

30

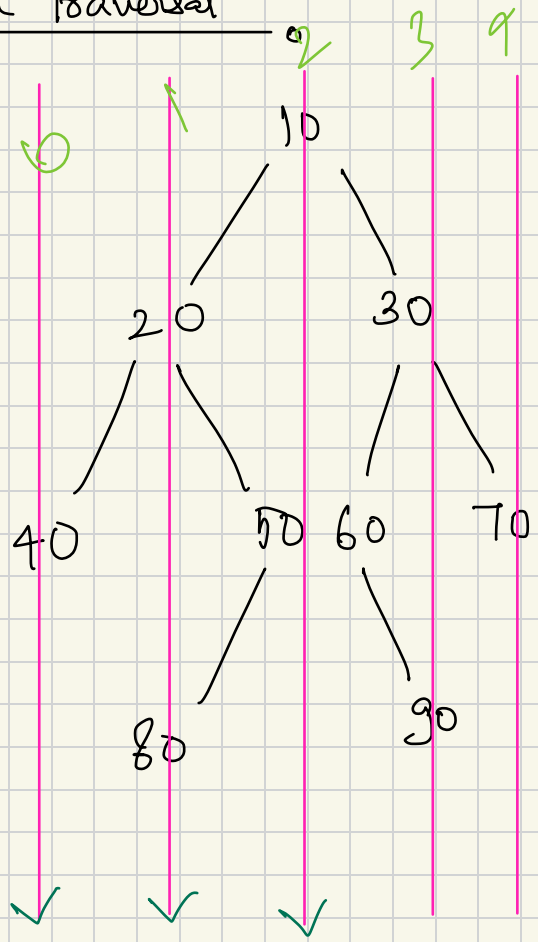


80 : {80, 60, 30, 10}

30 : {30, 10}

LCA : 30

Vertical Order Traversal



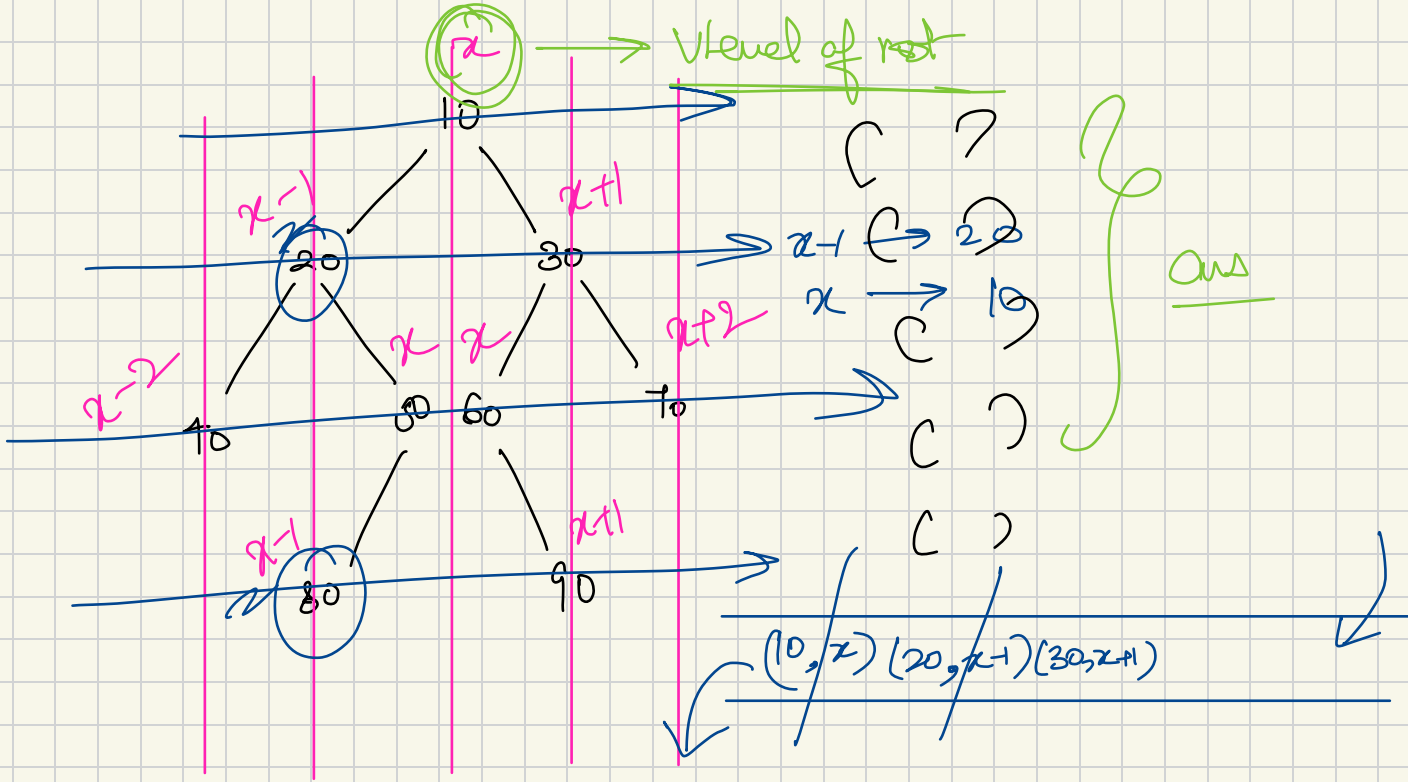
0 → 40

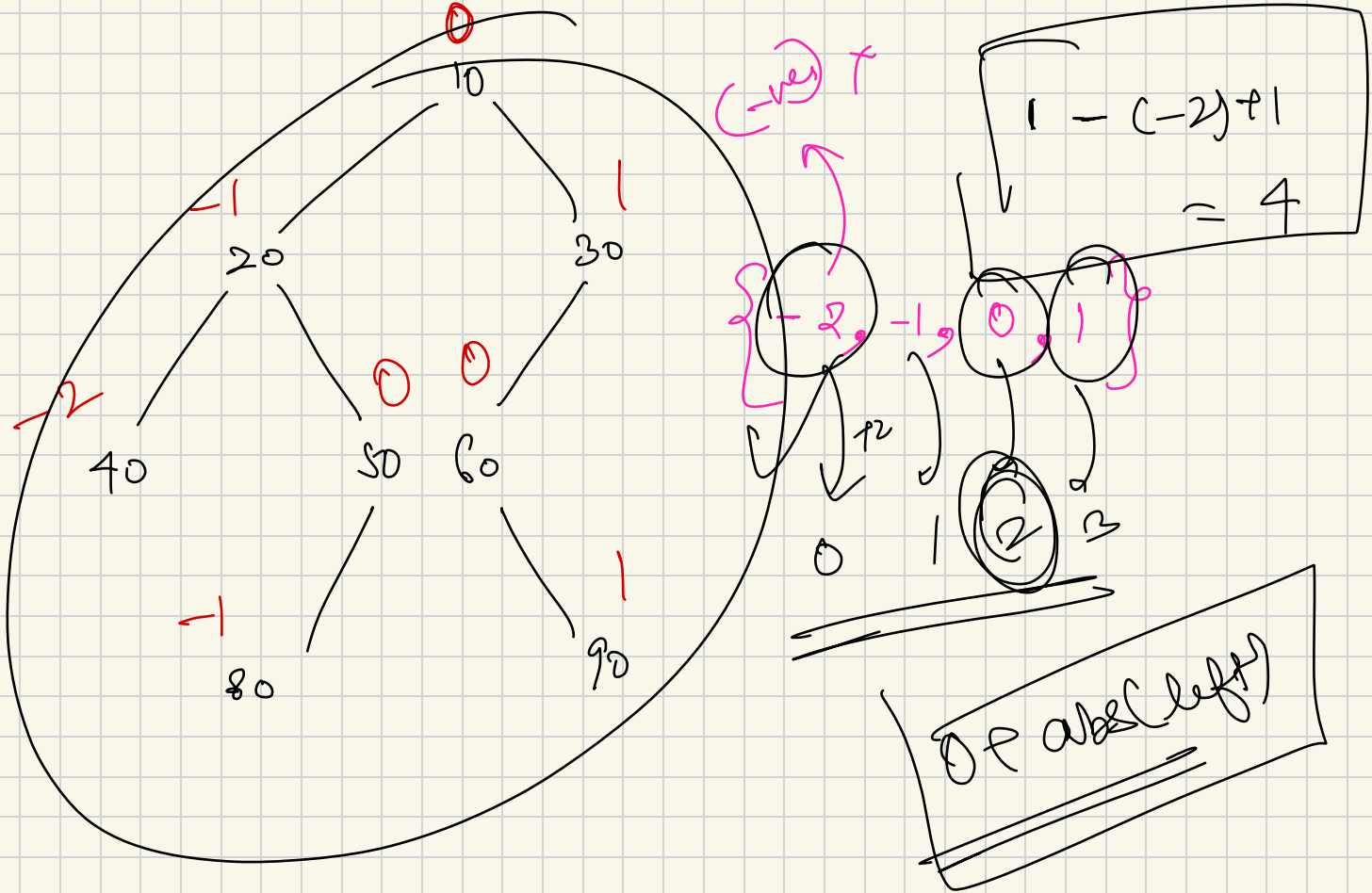
1 → 20, 80

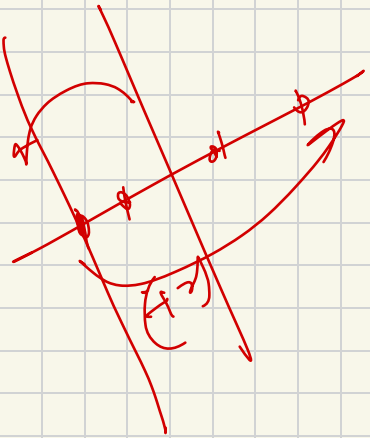
2 → 10, 50, 60

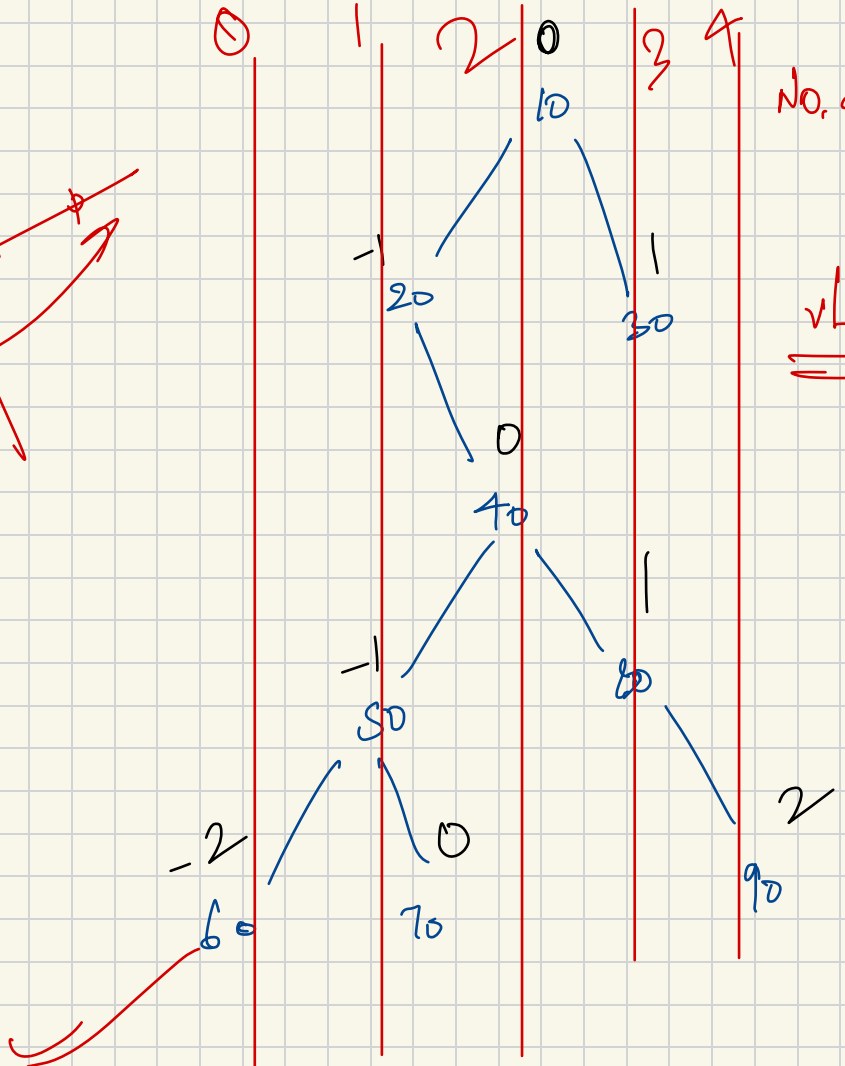
3 → 30, 90

4 → 70


x
level of root
10
x-1
x+1
20
30
x-1
20
x
10
x-2
x
x
x+2
40
50
60
70
x-1
x+1
80
90
(10,x) (20,x-1) (30,x+1)
ans


0
10
-1
20
1
30
-2
40
0
50
0
60
-1
80
1
90
(-ve)
1 - (-2) + 1
= 4
-2
-1
0
1
12
0
1
2
3
0 + abs(left)

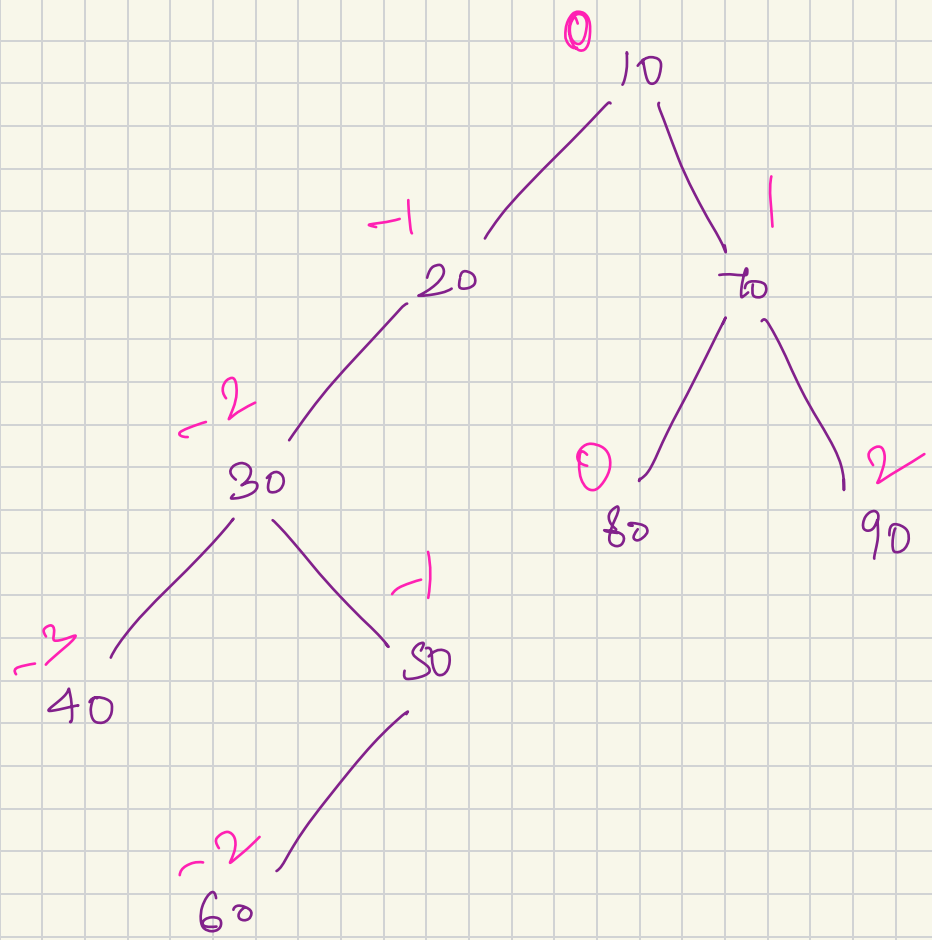




No. of nodes $= 2 - (-2) + 1$

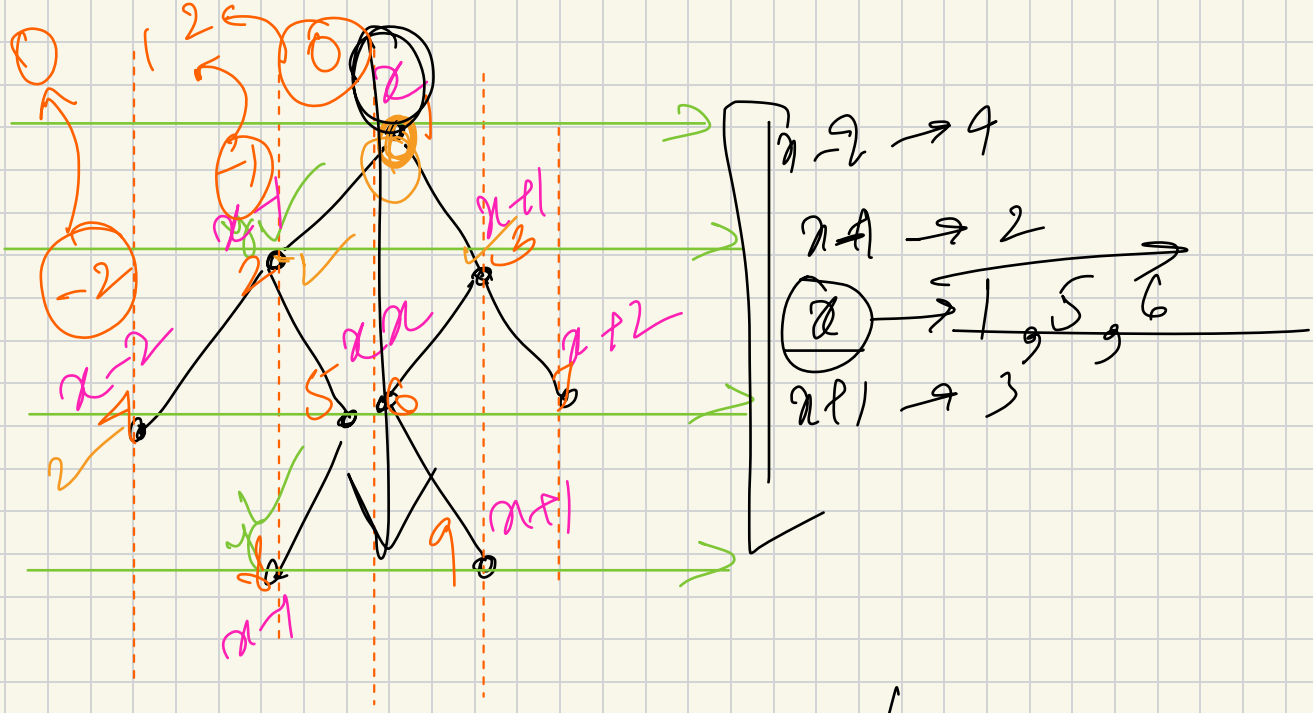
$= 5$ ✓

$|\text{Level (root)} - 2| = 2$ ✓



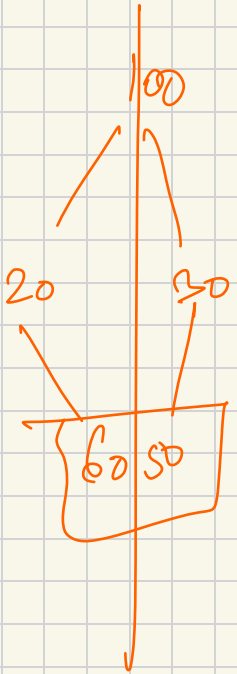
No. = 2 - (-3) + 1 = 6

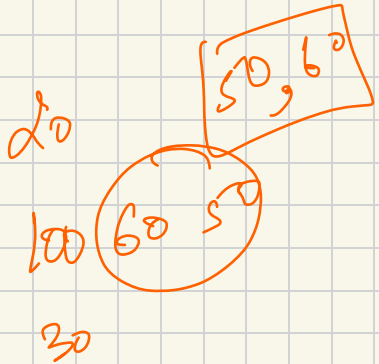
Vlevel = 0 + |-3| = 3



(1, n) (2, n-1) (3, n+1) (4, n-2) (5, n) (6, n)

(-1, n+2)


100
20
30
60
50


50, 6°
20
100
60
30