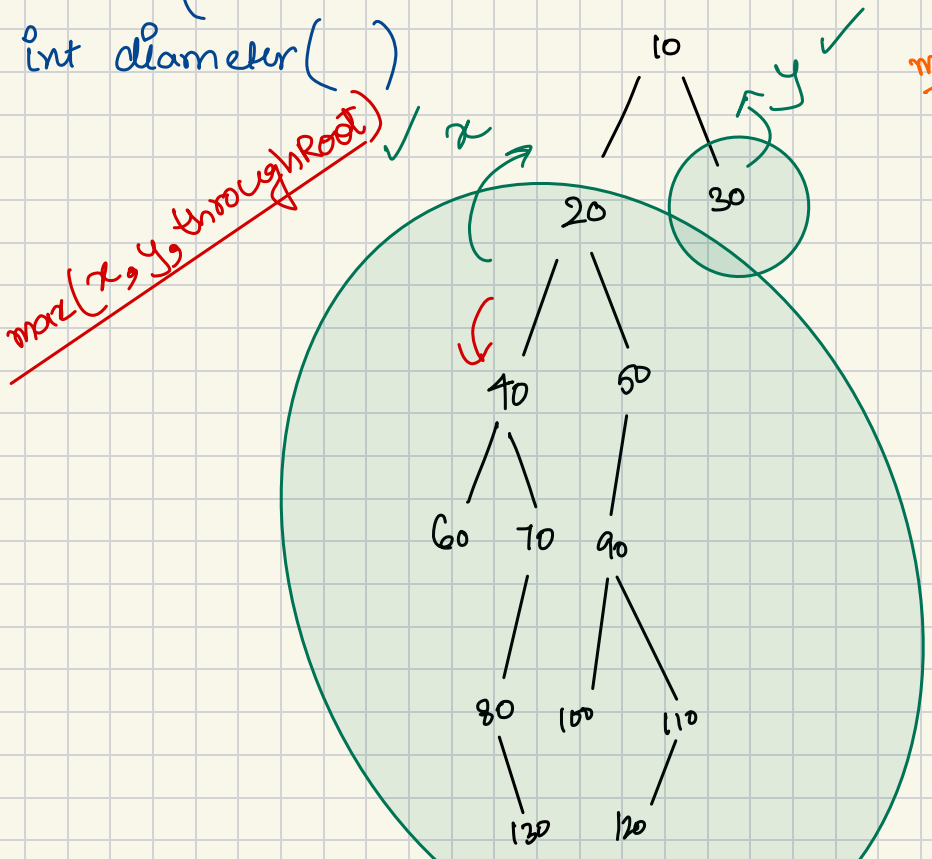
Diameter of a Binary Tree.

int diameter( ) → faith: returns diameter of the tree



max dist. b/w any two leaf Nodes

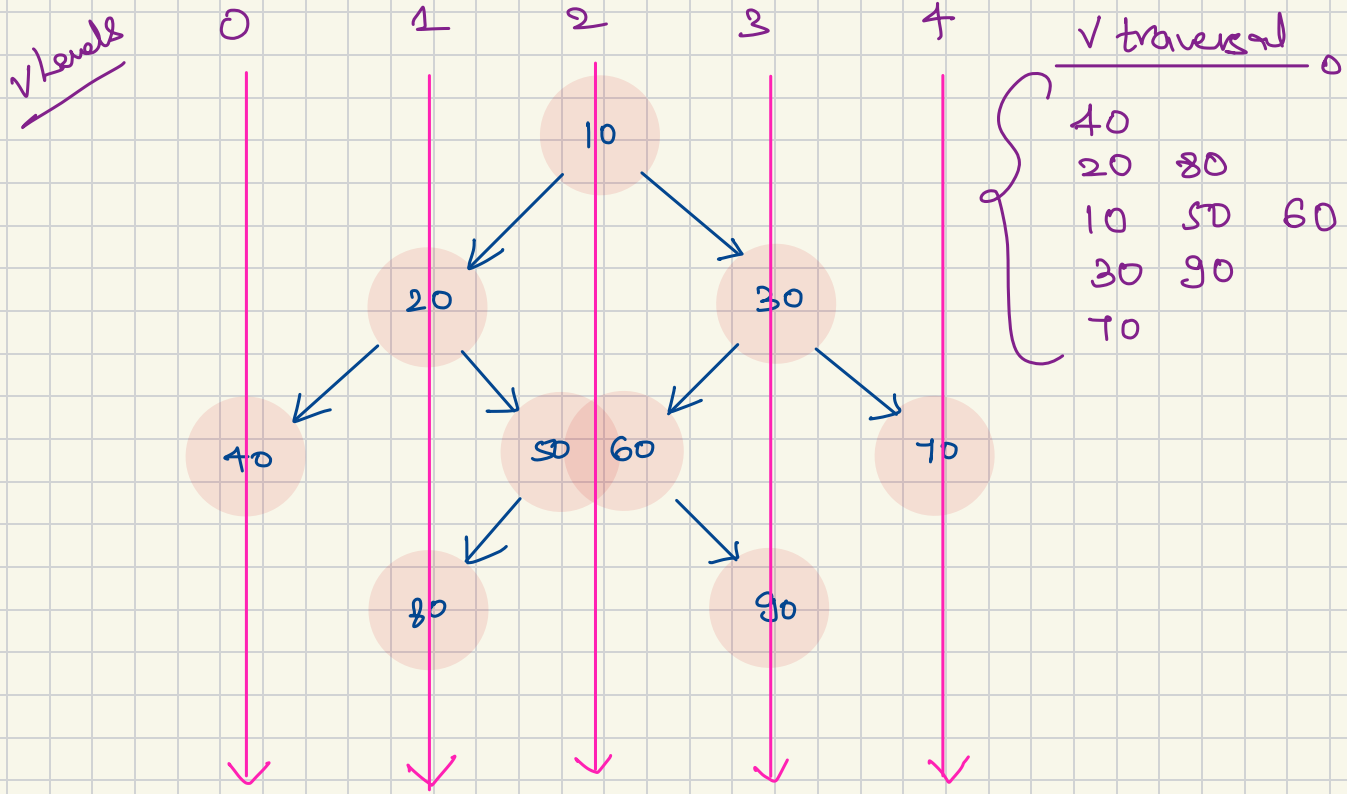
↓

diameter may not pass through root.

Agenda

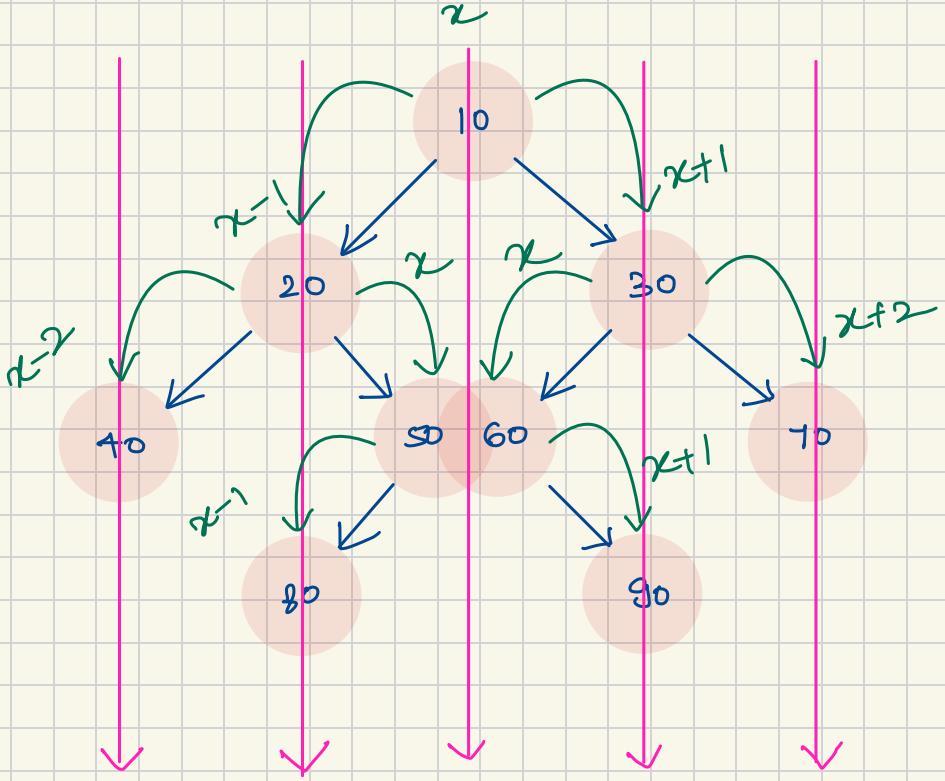
- Vertical Order traversal
- find a node
- Path to a given Node
- LCA

Vertical Order traversal



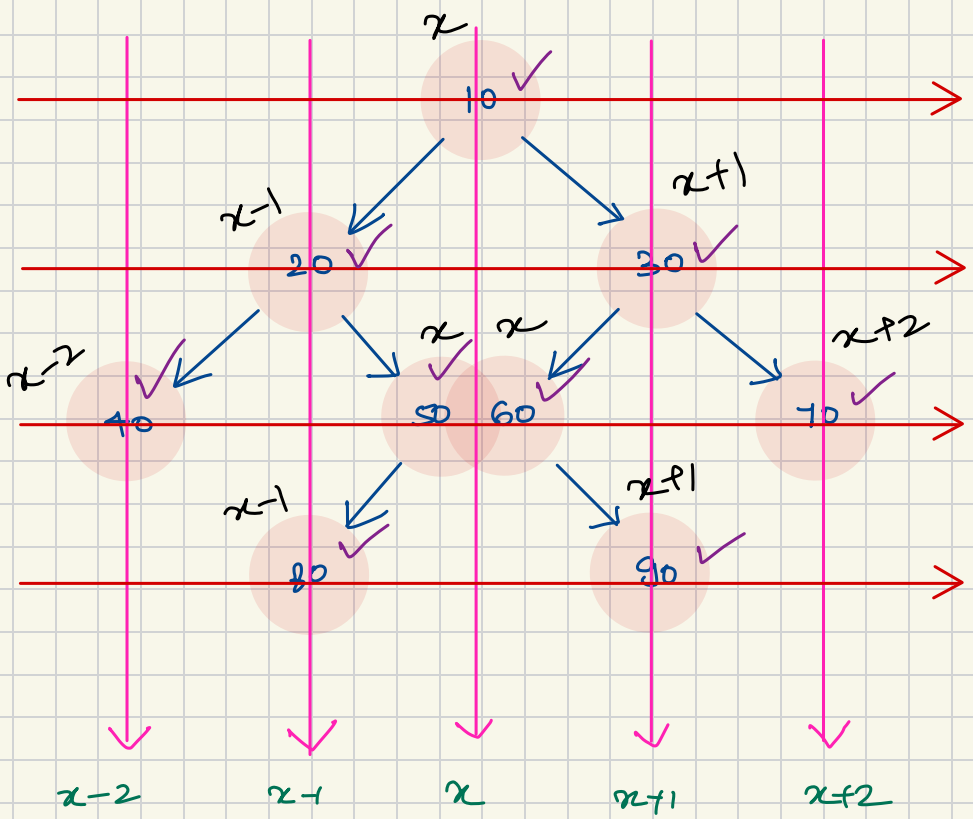
V traversal

40
20 80
10 50 60
30 90
70



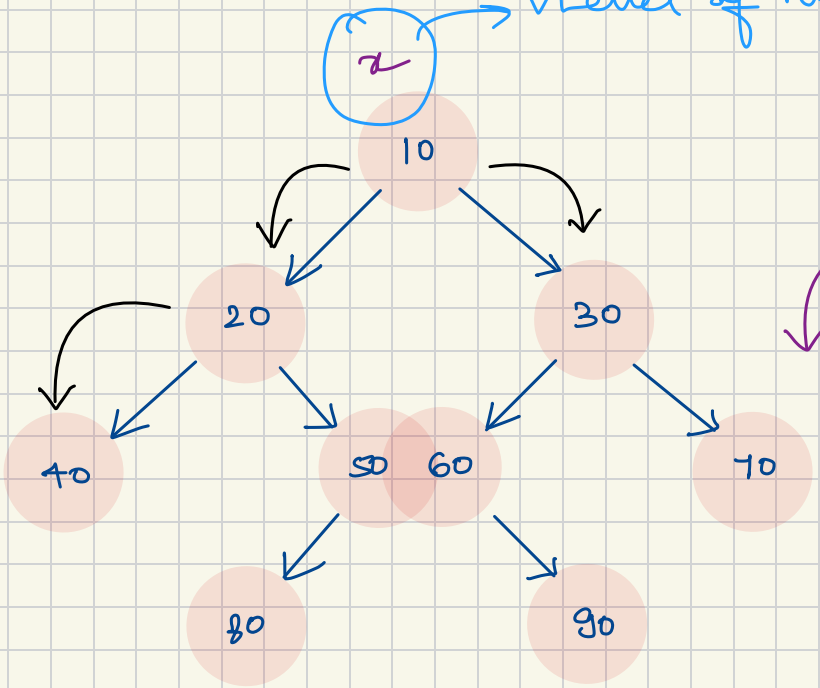
- when moving left
  - level decreases by 1
- when moving right
  - level increases by 1

cost to parent node



Vlevels

x-2	40
x-1	20, 80
x	10, 50, 60
x+1	30, 90
x+2	70



→ Level of the Root Node.

que

(10, x) (20, x-1) (30, x+1)

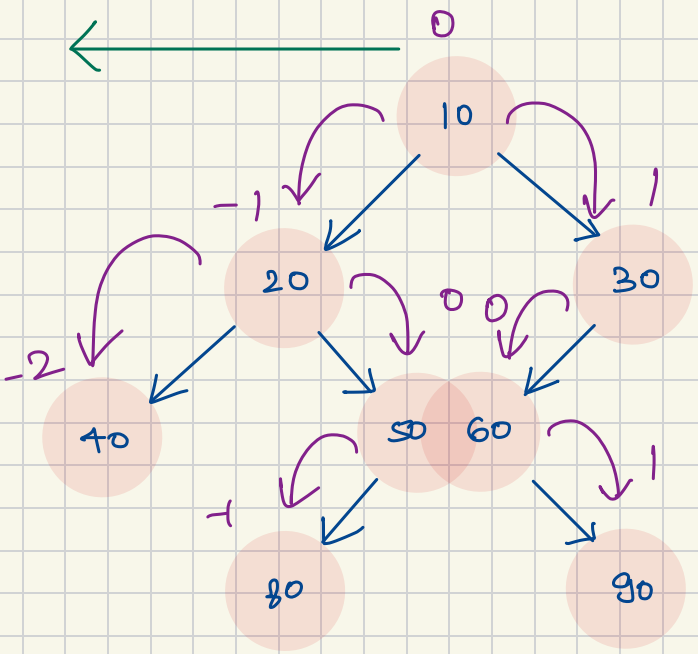
(40, x-2) (50, x)

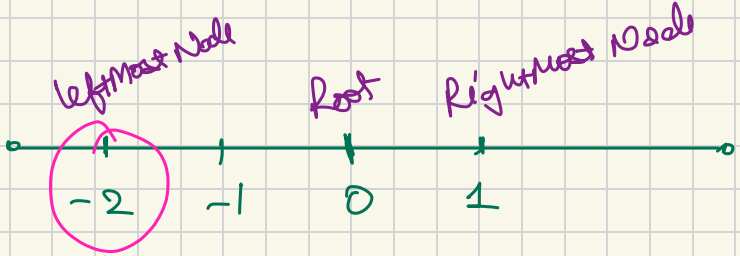
x+1 ⟶ 20

x ⟶ 10

- Level RootNode
- No. of levels

Calc. vLevel of Root Node



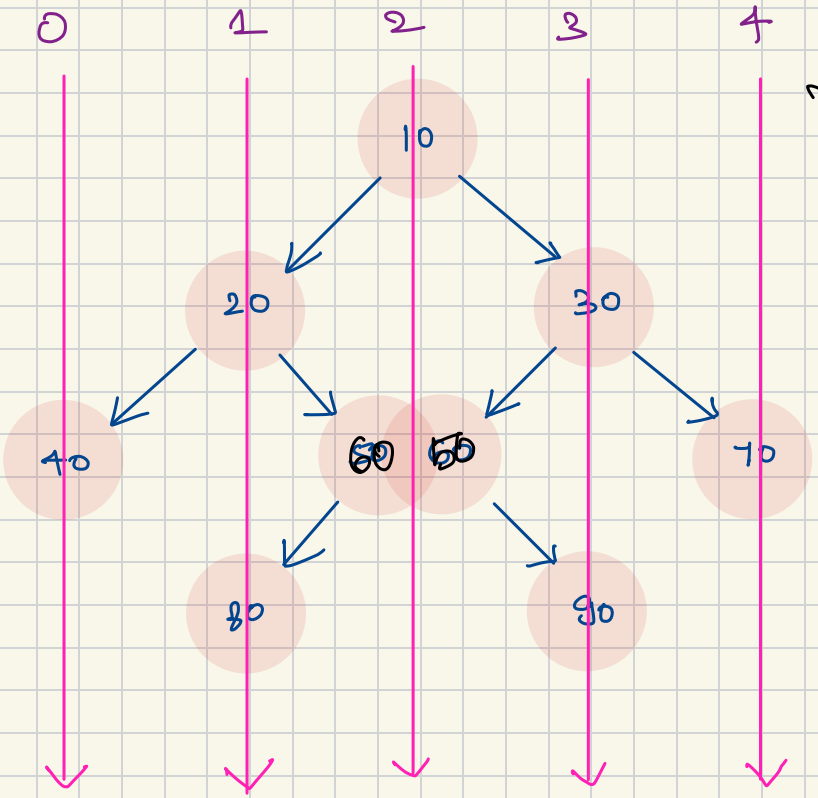


total vLevels = (right - left + 1) vLevels

Should be at 0

Shifting of Origin

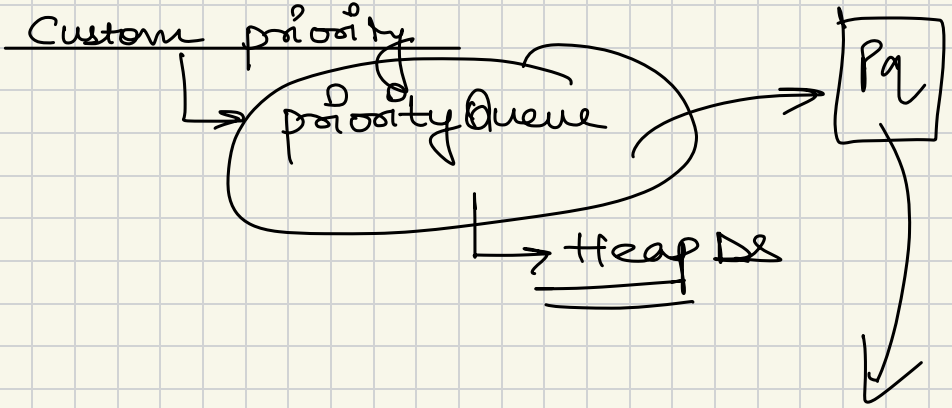
vLevel of Root = abs(min Level)



Conditions

- When vLevel and Level are same for nodes, then add nodes in sorted order in VL.

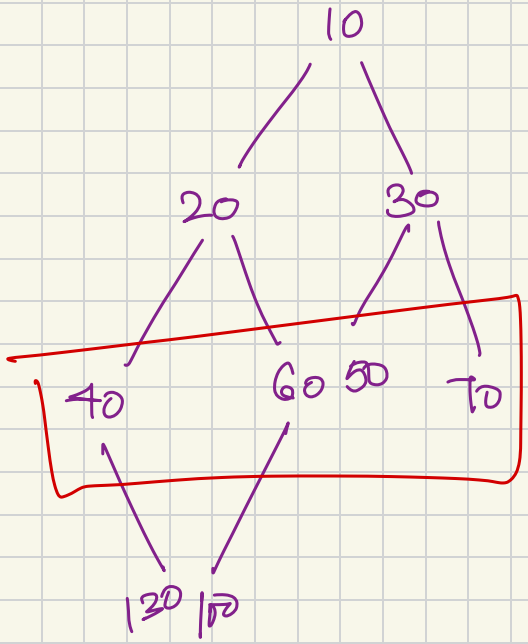
Q: 10, 50, 60

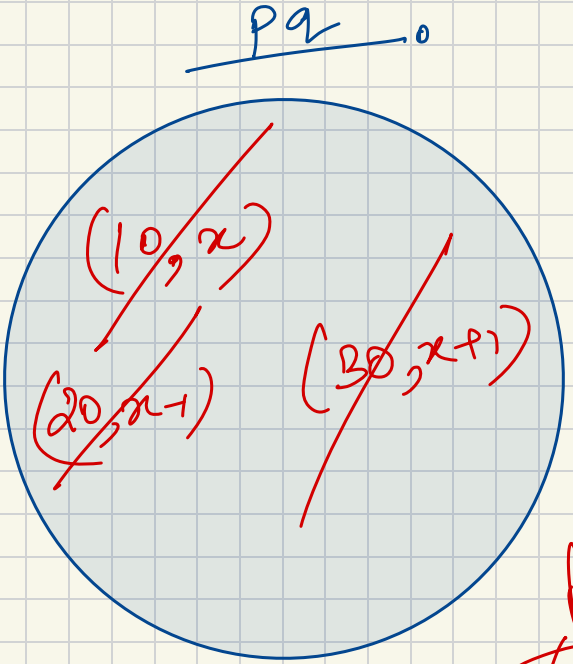


Pq

Set some Rules:

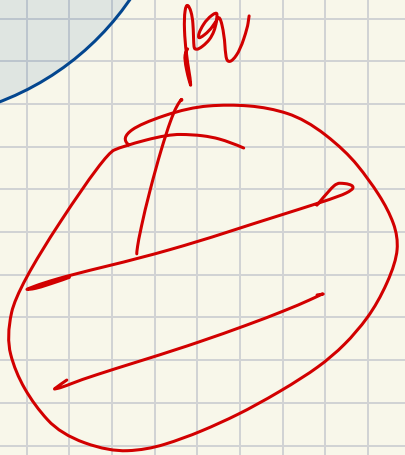
- If val1 and th level are same value should be sorted
- If only th level is same, returns smaller val1 first.



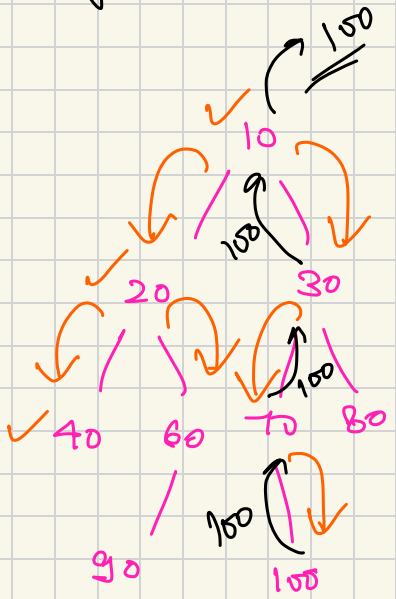


x+1 ! 20

x ! 10



find the given Node in a BT

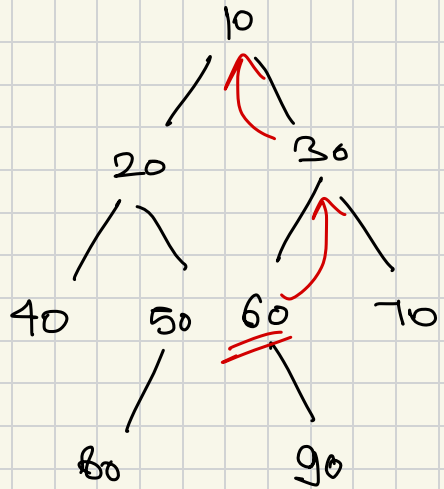


target - 100

fun() returns add. of Node if found

```
Node find (Node root, int target)
{
  if (root == null) return null;
  if (root == target) return root;
  Node filc = find(root.left)
  if(filc) return filc;
  Node fir = find (root.left)
  if(fire) return fir;
  return null;
}
```

Node to Root Path

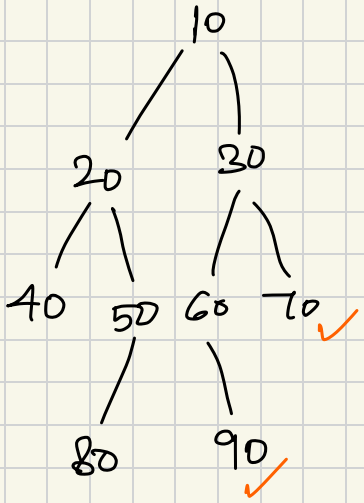


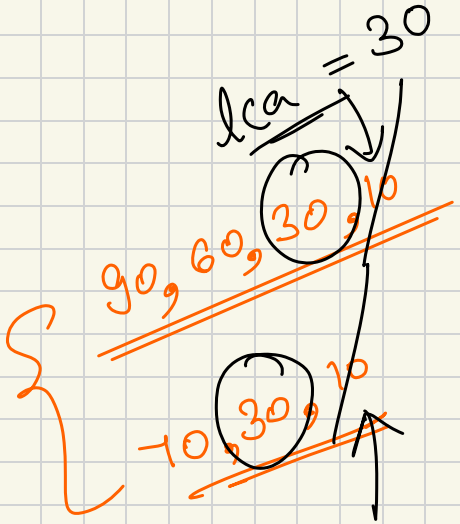
N2R : 60, 30, 10

```
void printN2R(Root, Node)
{

}
```

LCA { Lowest Common Ancestor }



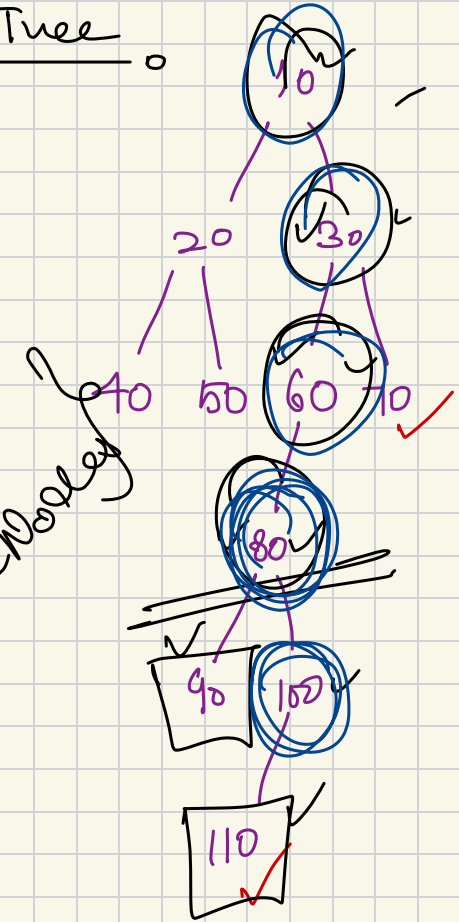


Binary Tree

lca

two Node

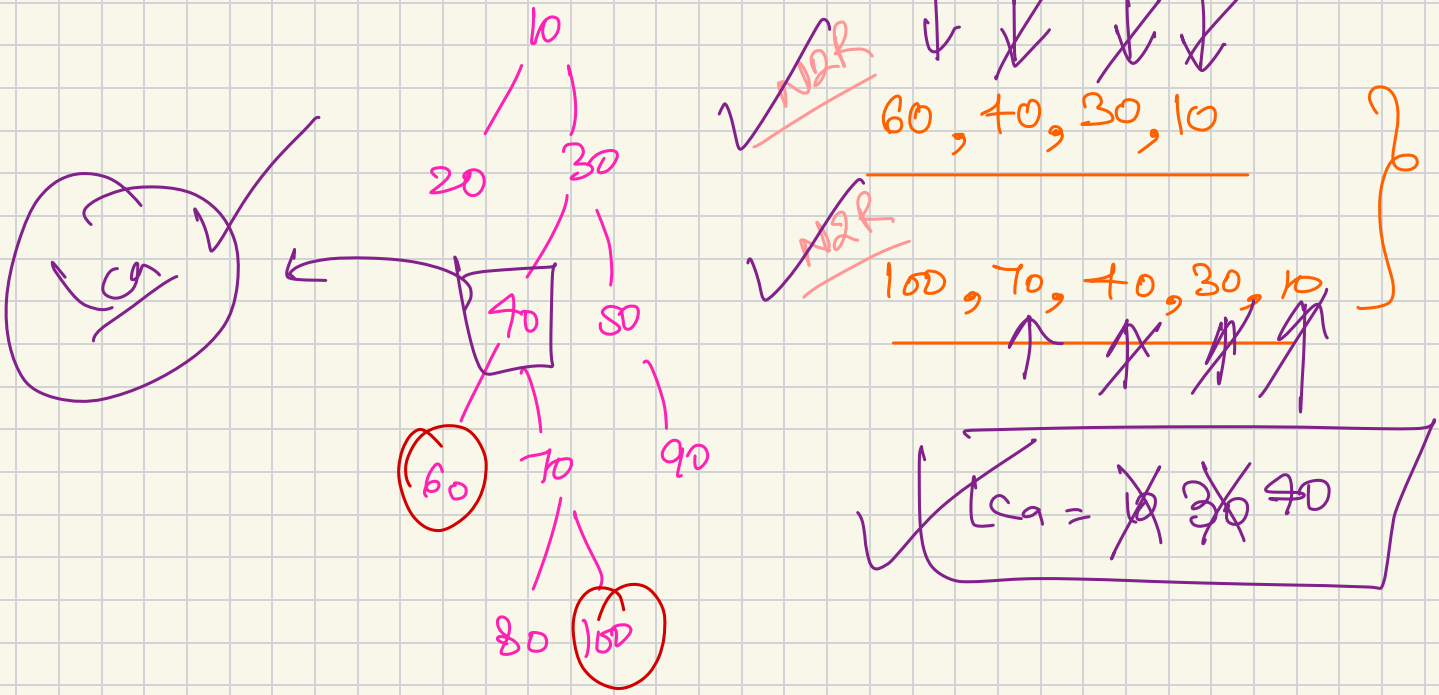
lca of three Nodes



110, 100, 80, 60, 30, 10

70, 30, 10

lca = 30


10
20
30
40
50
60
70
80
90
100
N2R
N2R
60, 40, 30, 10
100, 70, 40, 30, 10
Ca = 10 30 40