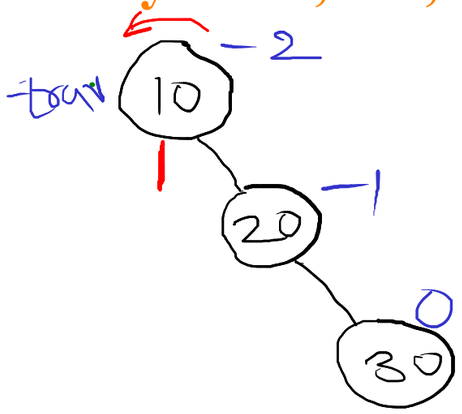
RR Imbalance

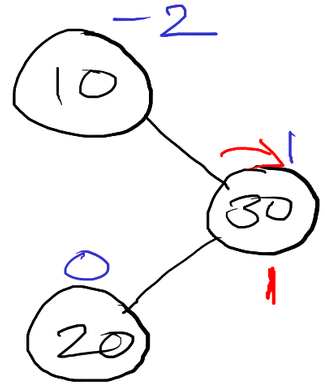
Keys : 10, 20 ,30



bf < -1 &&
value > trav.right.data

RL Imbalance

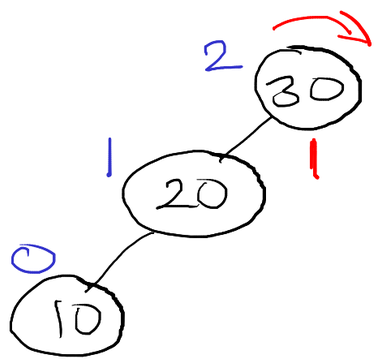
Keys : 10, 30, 20



bf < -1 &&
value < trav.right.data

LL Imbalance

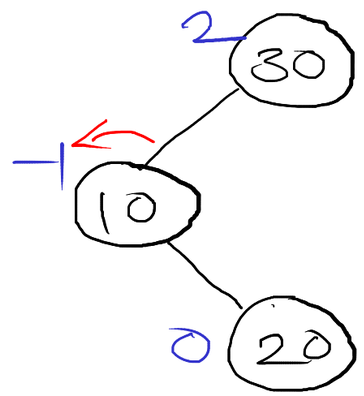
Keys : 30, 20, 10



bf > 1 &&
value < trav.left.data

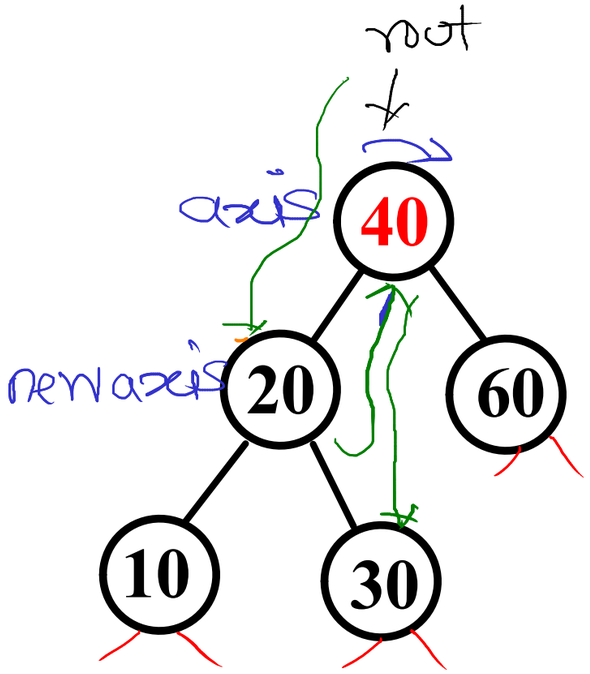
LR Imbalance

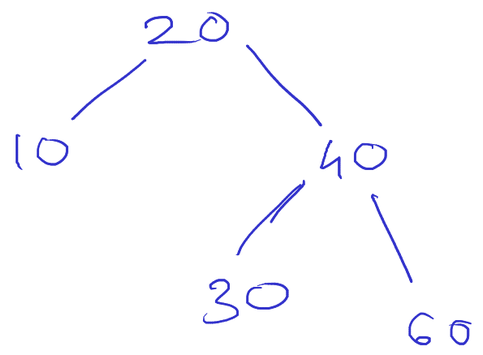
Keys : 30, 10, 20



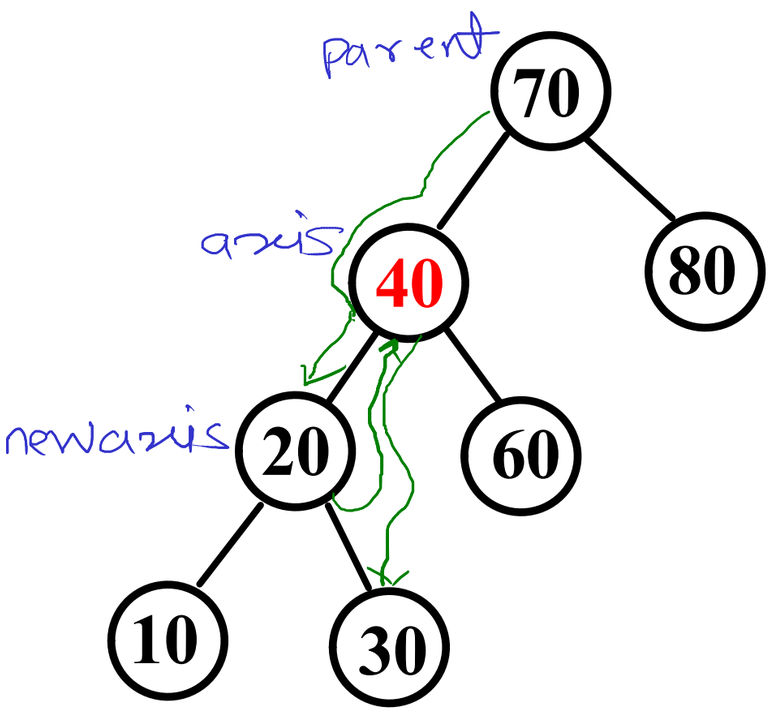
bf > 1 &&
value > trav.left.data

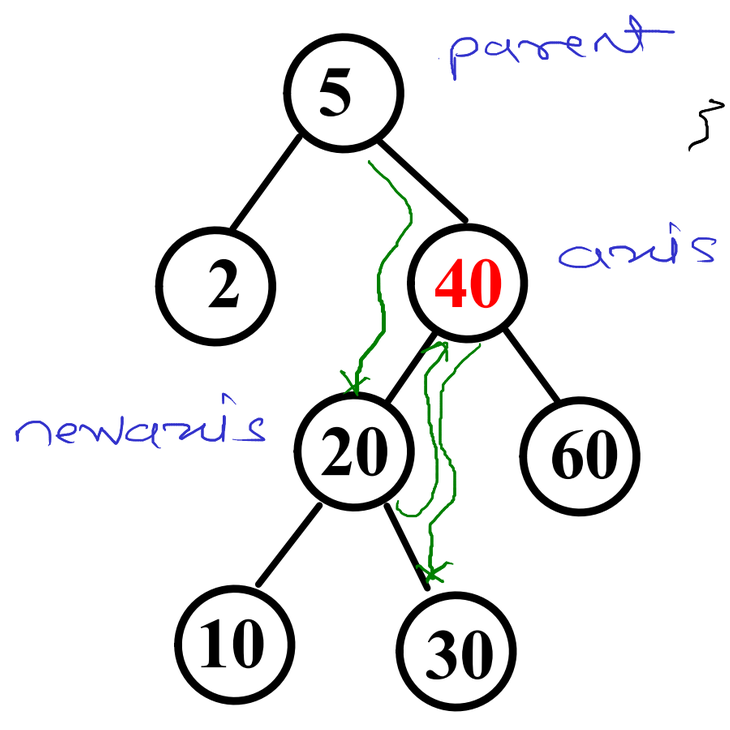
Right Rotation



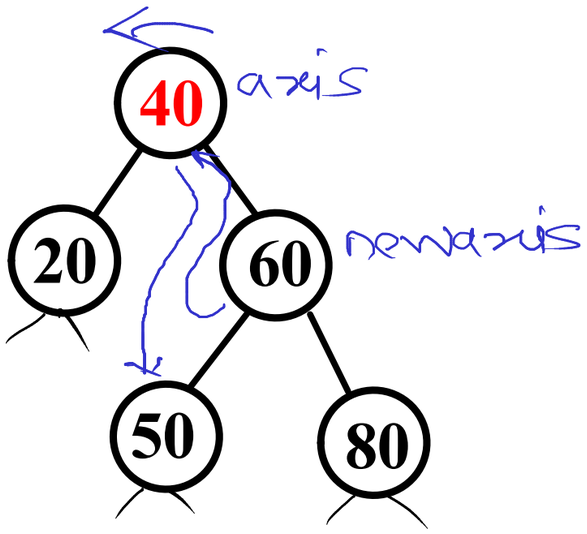


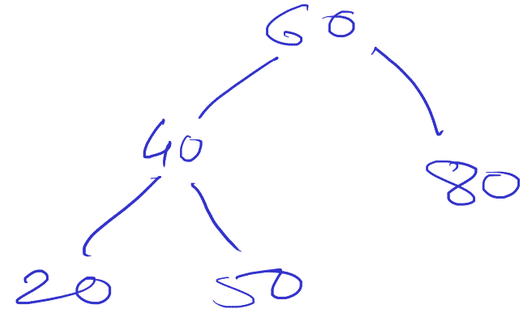

```
right_rotation(axis, parent){
newaxis=axis.left
axis.left=newaxis.right
newaxis.right=axis
if(axis==root)
    root=newaxis;
else if(axis== parent.left)
    parent.left=newaxis;
else
    parent.right=newaxis;
}
```



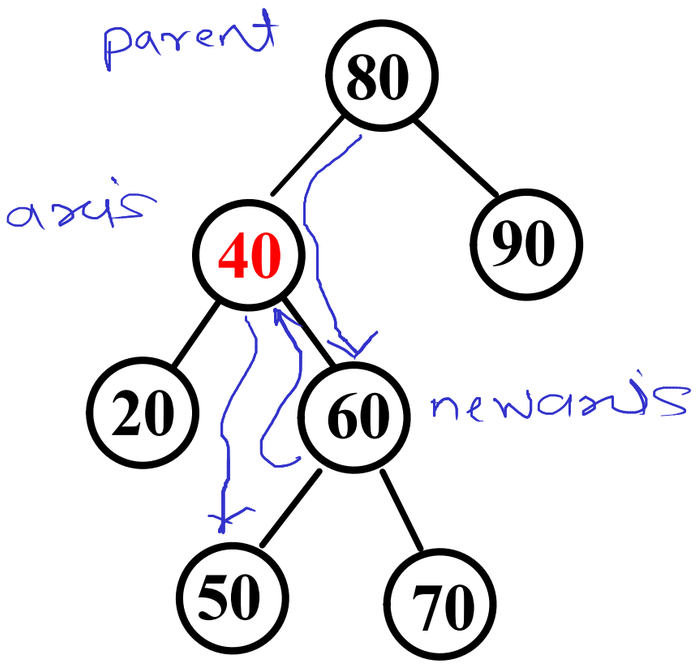


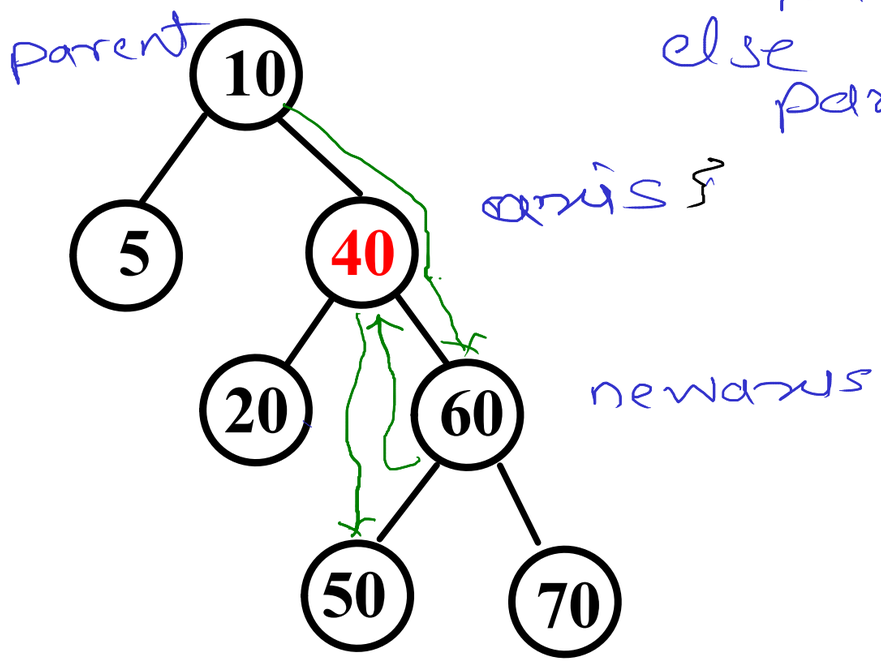
Left Rotation






```
left_rotation(axis, parent) {
newaxis = axis.right
axis.right = newaxis.left;
newaxis.left = axis ;
if(axis == root)
      root = newaxis ;
elseif(axis == parent.left)
      parent.left = newaxis;
else
      parent.right = newaxis;
```

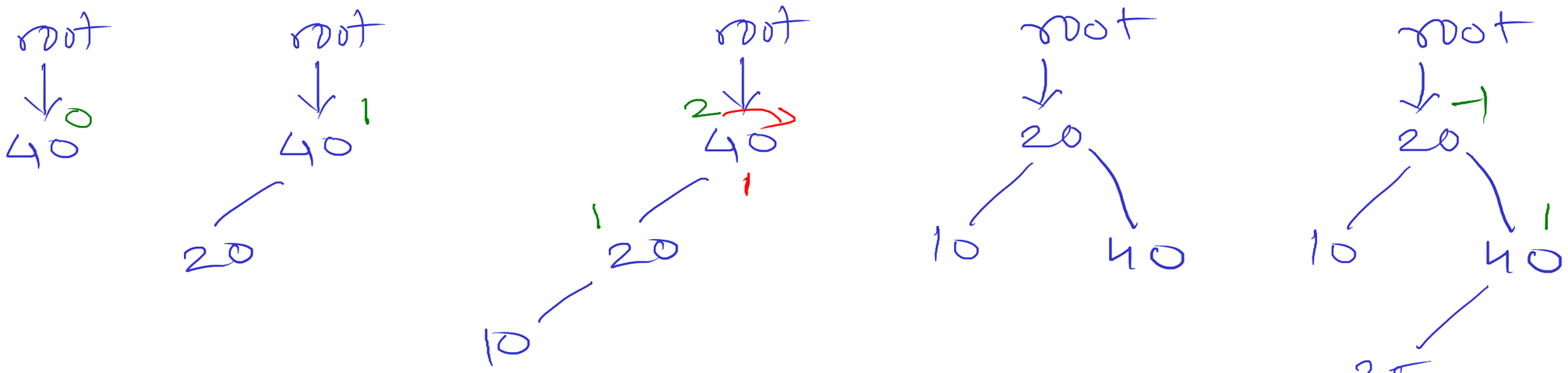


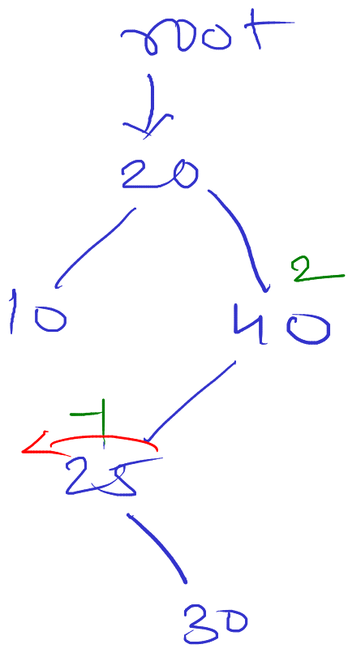


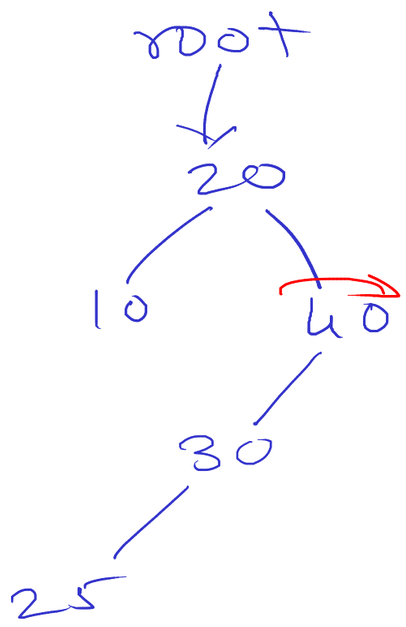
AVL Tree

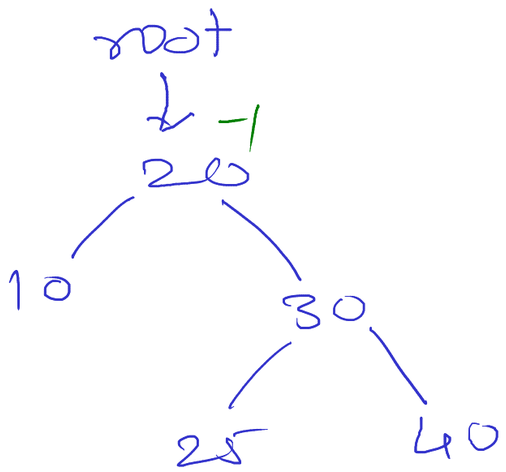
- Self balancing binary Search Tree
- on every insertion and deletion of node, tree is balanced
- All operation on AVL tree are perfromed in O(log n) time
- Balance factor of all nodes is either -1, 0 or +1

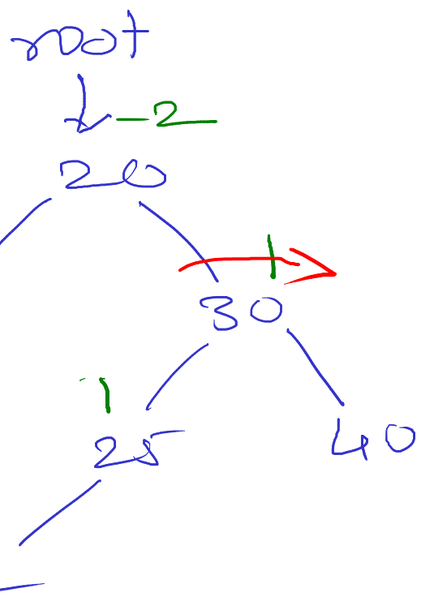
Keys : 40, 20, 10, 25, 30, 22, 50

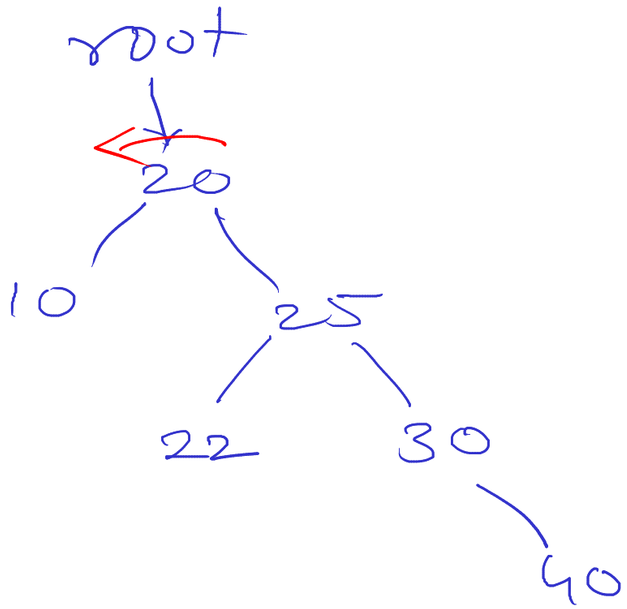


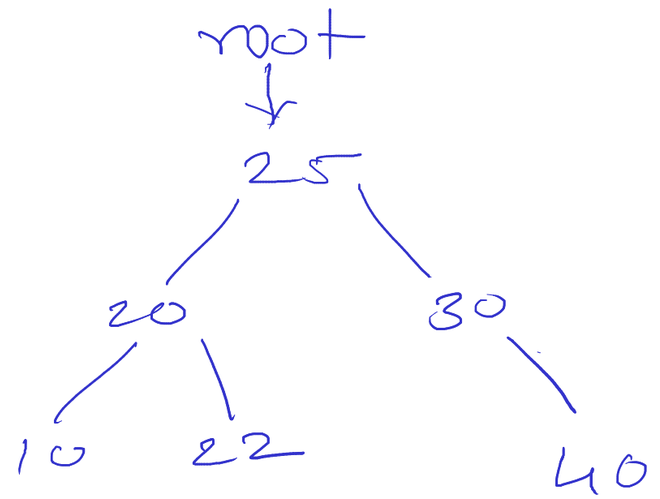

root
20
10
40
2
25
-1
30

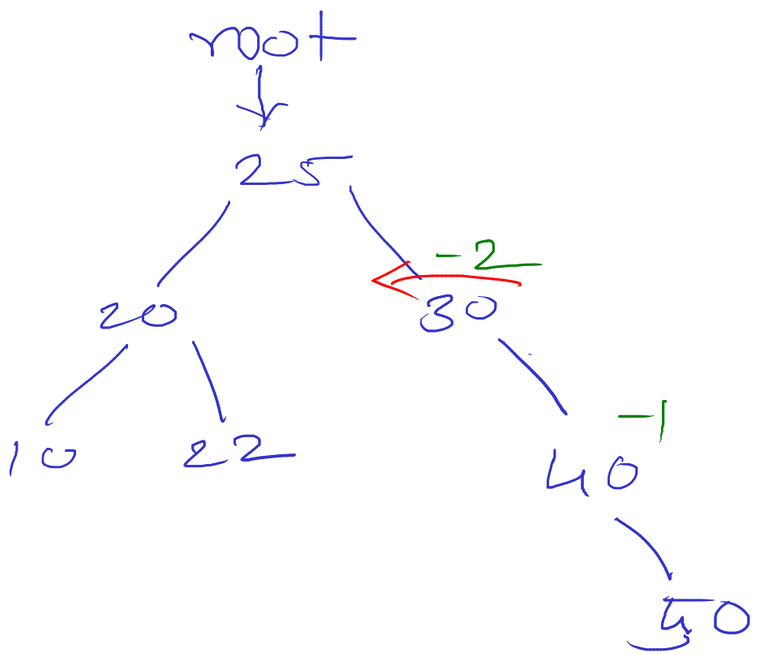

root
20
10
40
30
25

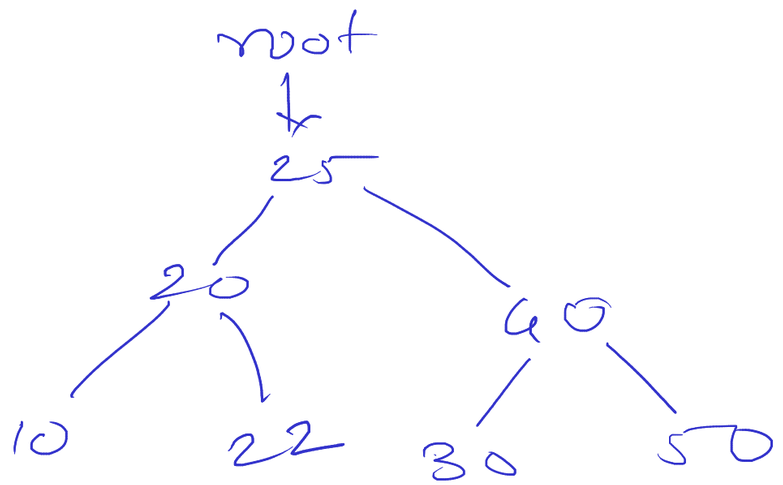

root
-1
20
10
30
25
40


root
-2
20
10
30
1
25
40
22


root
20
10
25
22
30
40


root
25
20
30
10
22
40


root
25
20
30
10
22
40
50
-2
-1


root
25
20
40
10
22
30
50