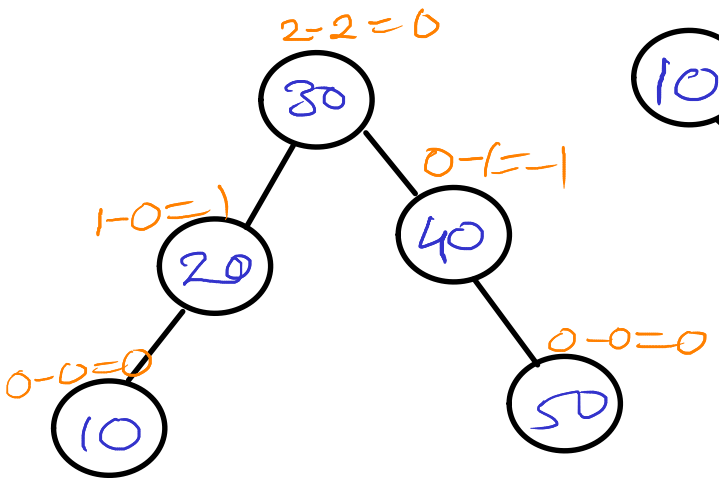
# Skewed BST

**Keys : 30, 40, 20, 50, 10**

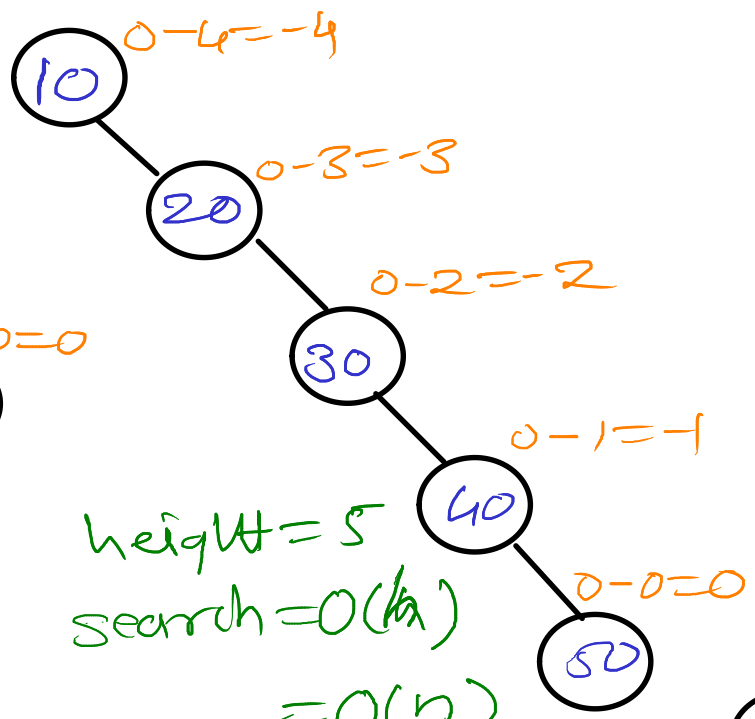



height = 3

search = O(h)

= O(log n)

**Keys : 10, 20, 30, 40, 50**

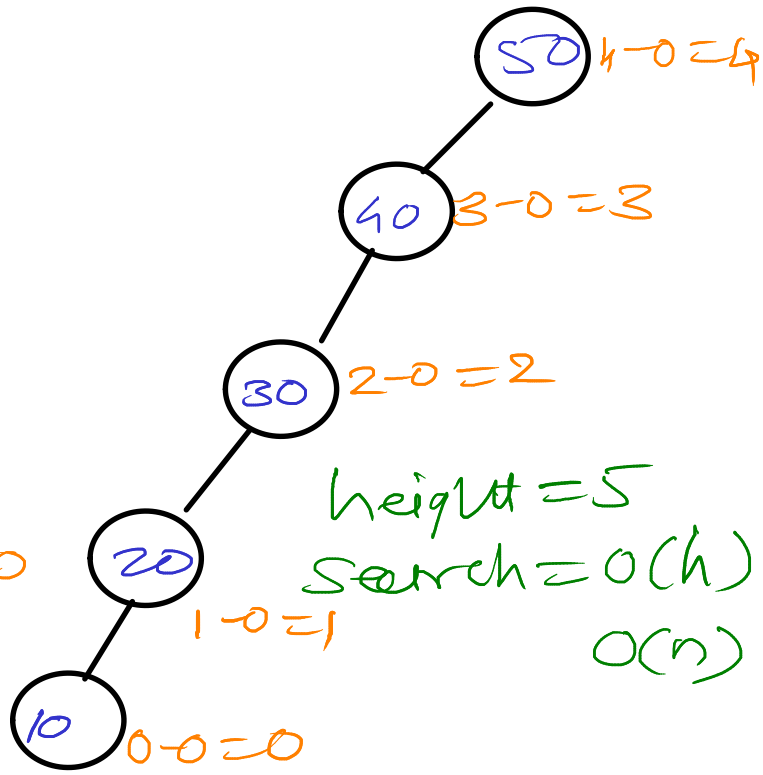



height = 5

search = O(h)

= O(n)

**Key : 50, 40, 30, 20, 10**




height = 5

search = O(h)

O(n)

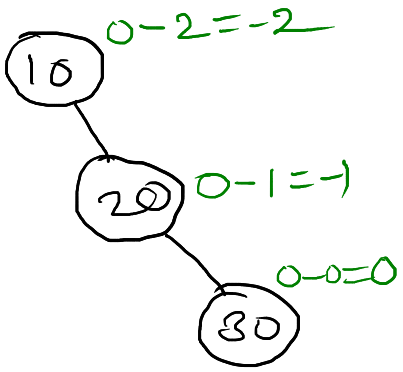
- **if BST is growing only in one direction, such tree is called as skewed BST**
- **if BST is growing in right direction only, such tree is called as Right skewed tree**
- **if BST is growing in left direction only, such tree is called as left skewed tree**

# Balanced BST

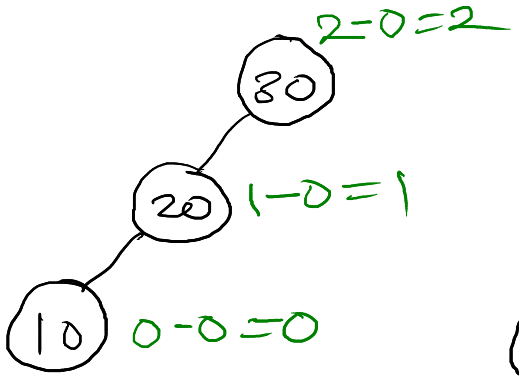
**Balance factor = height(left sub tree) - height(right sub tree)**

**if balance factors of each node is either -1, 0 or +1**
**then such BST is called as Balanced BST**

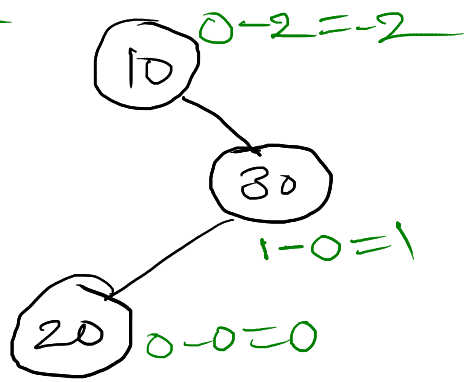
**Keys : 10, 20, 30**



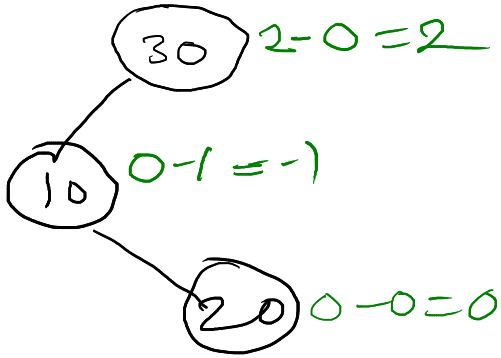

**Keys : 30, 20, 10**




**Keys : 10, 30, 20**

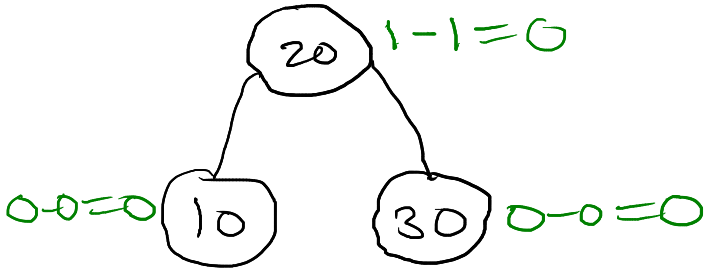



**Keys : 30, 10, 20**



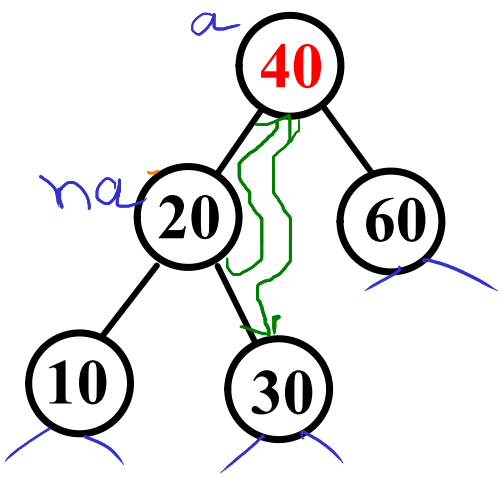

**Keys : 20, 10, 30**

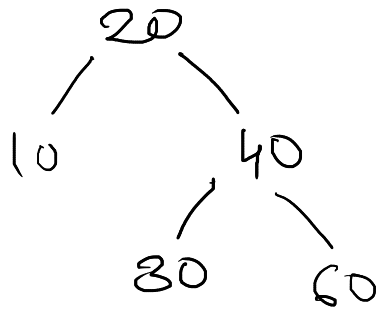
**Keys : 20, 30, 10**



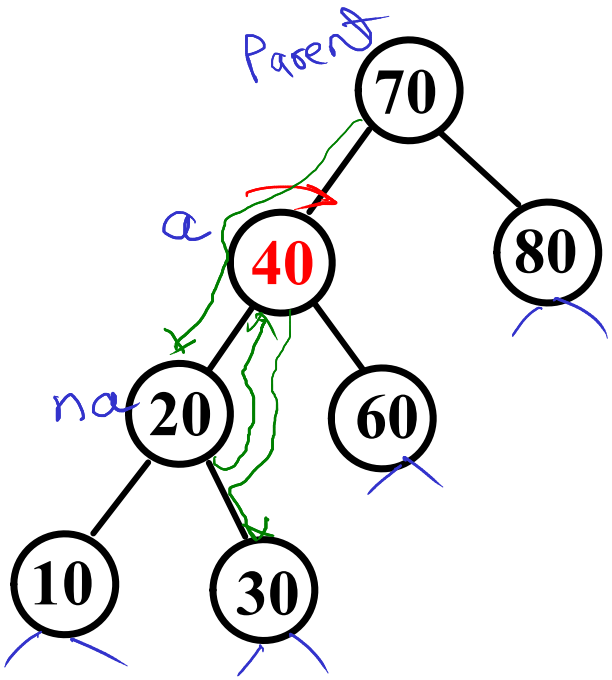

**Balanced BST**

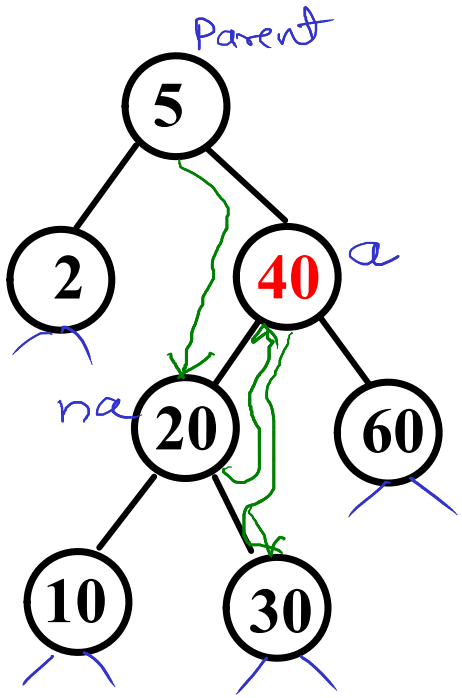
# Right Rotation



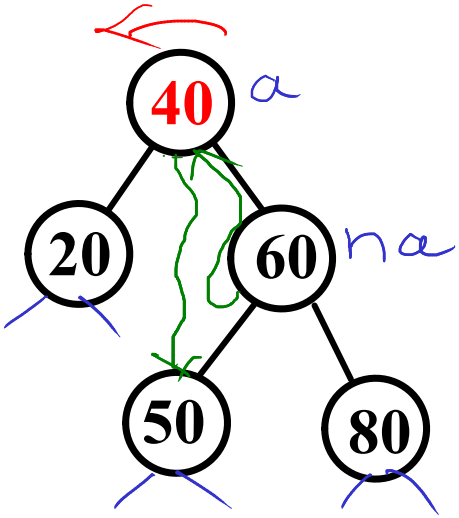



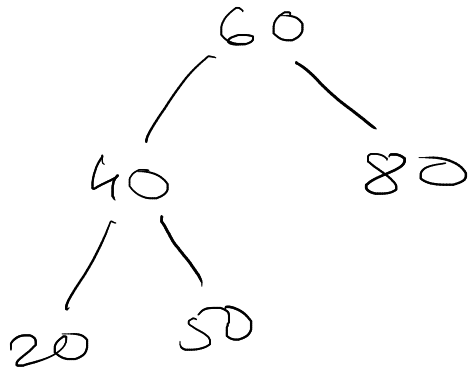

```
na=a.left;
a.left=na.right;
na.right=a;
if(a==root)
   root=na;
else if(a==parent.left)
   parent.left=na;
else if(a==parent.right)
   parent.right=na;
```



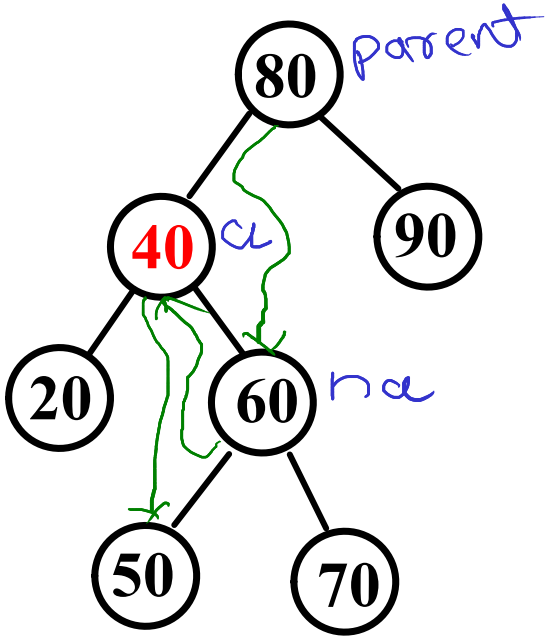



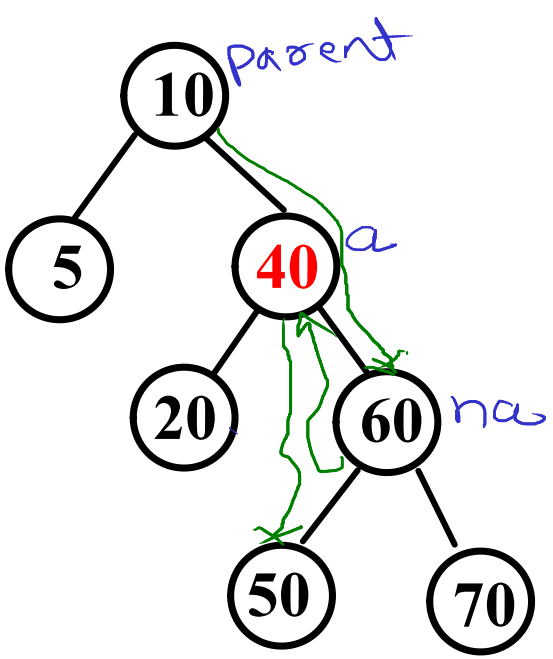
# Left Rotation




```
na=a.right;
a.right=na.left;
na.left=a;
if(a==root)
    root=a;
else if(a==parent.left)
    parent.left=na;
else if(a==parent.right)
    parent.right=na;
```

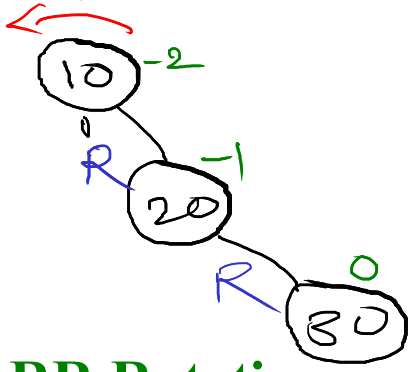


# Rotations

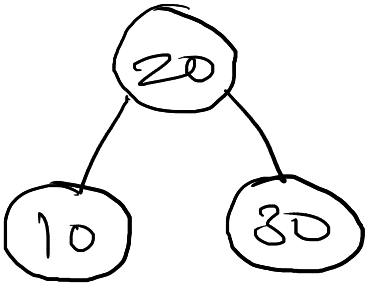
## RR Imbalance

Keys : 10, 20, 30



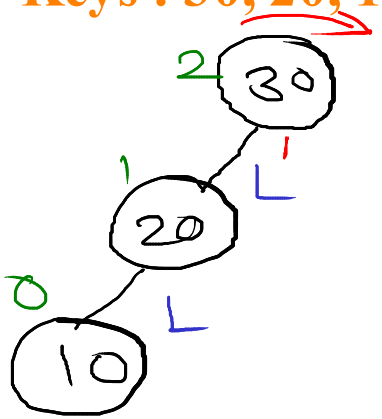

RR Rotation

Left Rotation



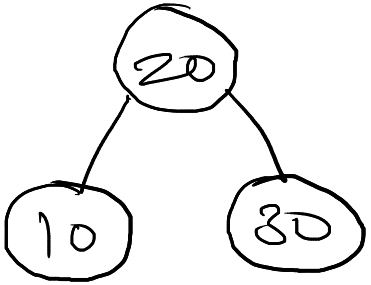

## LL Imbalance

Keys : 30, 20, 10



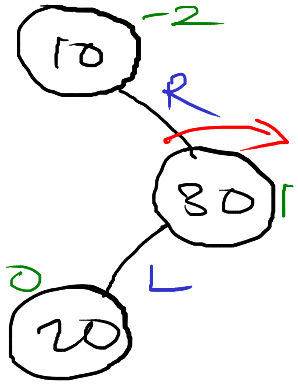

LL Rotation

Right Rotation

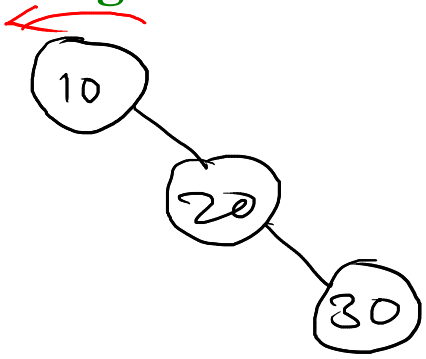



## RL Imbalance

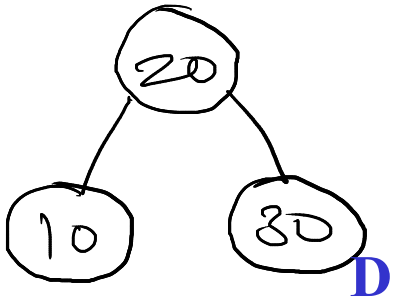
Keys : 10, 30, 20




Right Rotation

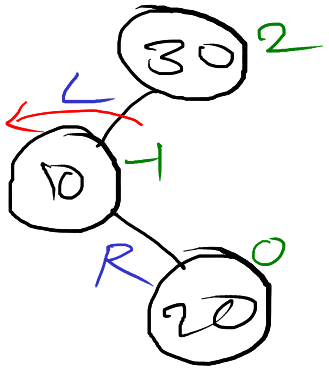



Left Rotation

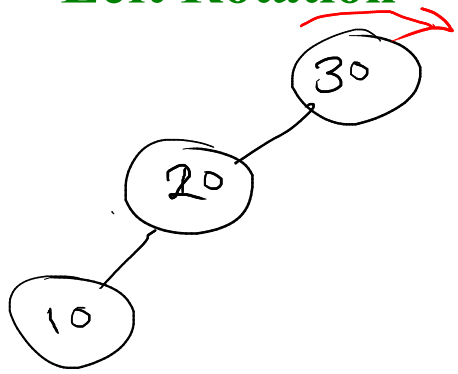



## LR Imbalance

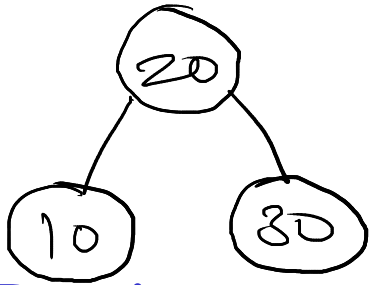
Keys : 30, 10, 20




Left Rotation




Right Rotation



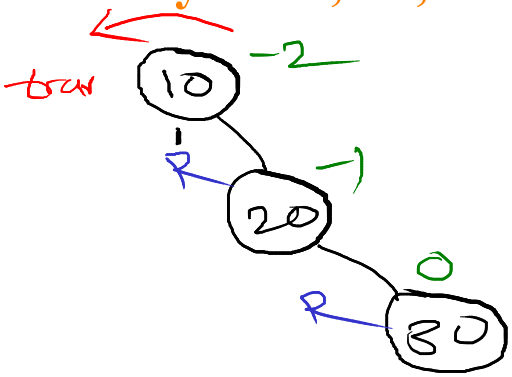

Single Rotations

Double Rotations

# Rotations

## RR Imbalance

**Keys : 10, 20, 30**



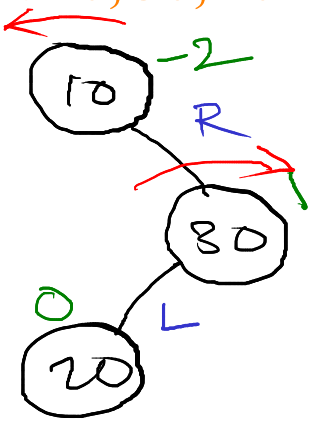

bf < -1

val > trav.right.data

30 > 20

## RL Imbalance

**Keys : 10, 30, 20**



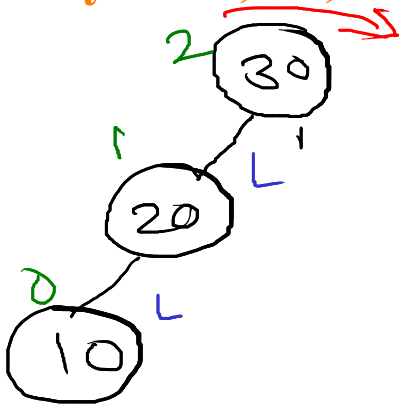

bf < -1

val < trav.right.data

20 < 30

## LL Imbalance

**Keys : 30, 20, 10**



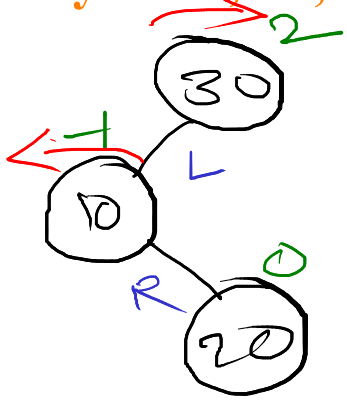

bf > 1

val < trav.left.data

10 < 20

## LR Imbalance

**Keys : 30, 10, 20**




bf > 1

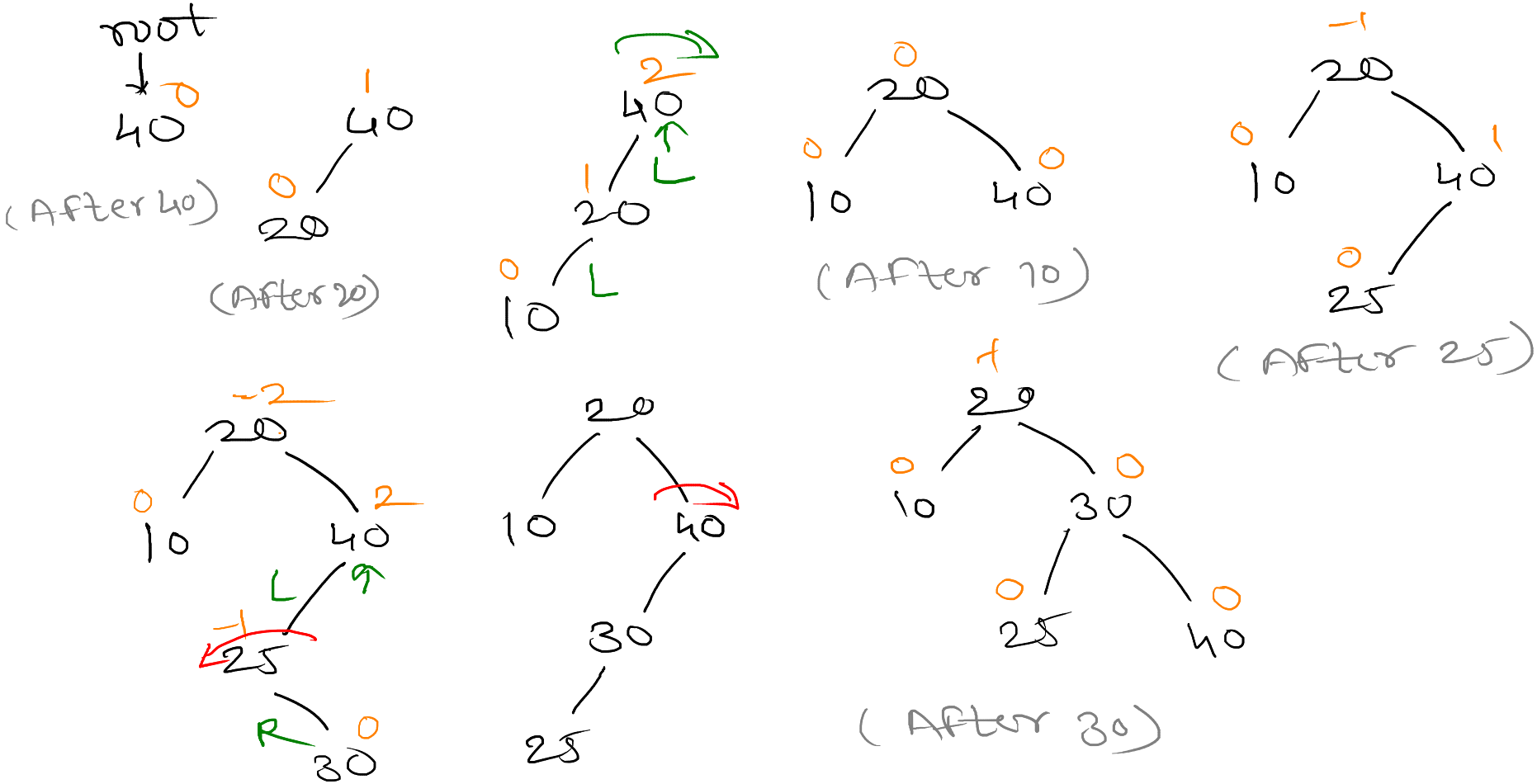
val > trav.left.data

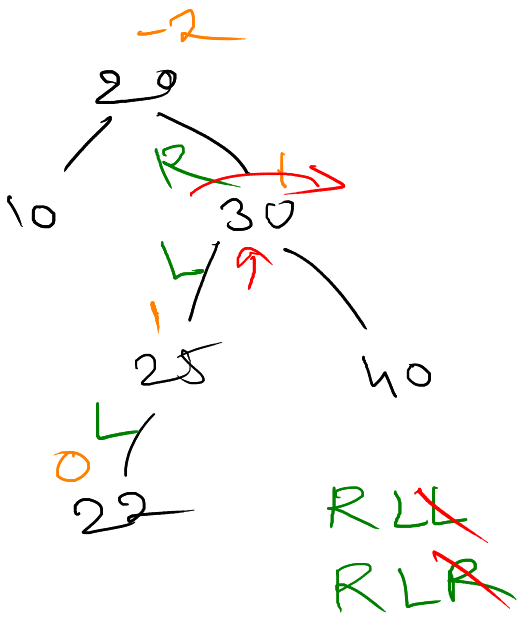
20 > 10

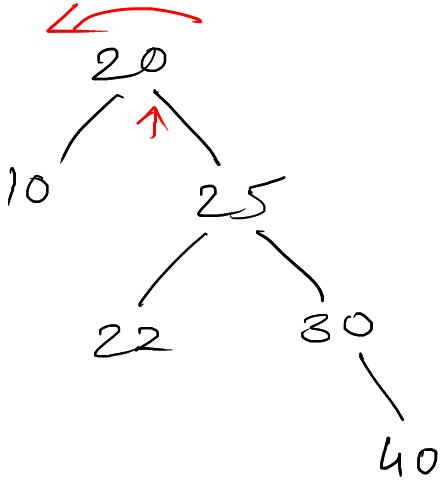
# AVL - Tree

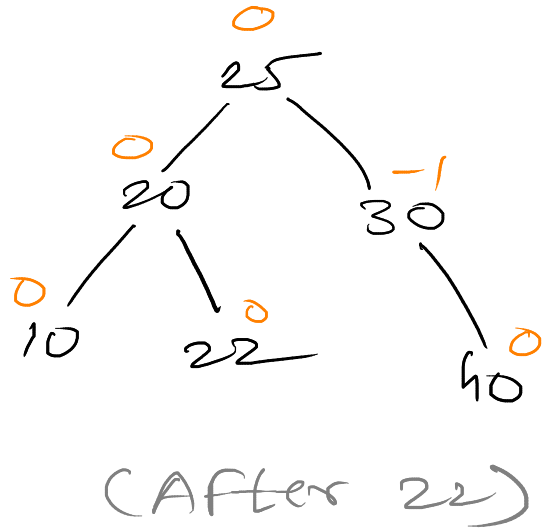
- AVL tree is a self balancing binary Search Tree
- on every insert and delete, tree is balanced(by performing rotations)
- most of the operations are performed into O(log n)
- balanced factore = {-1, 0, +1}

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

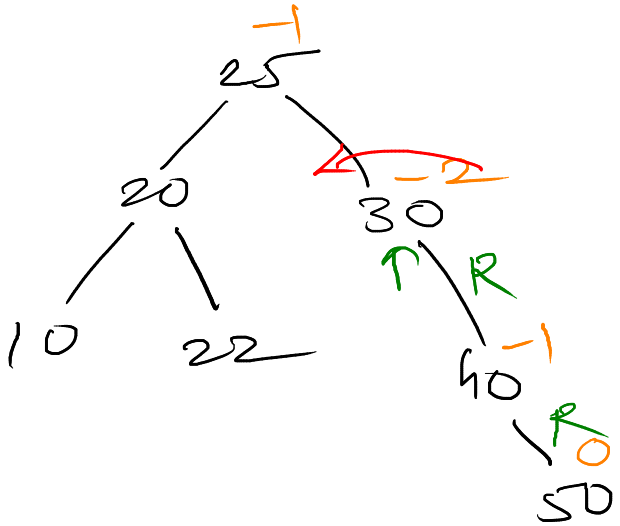


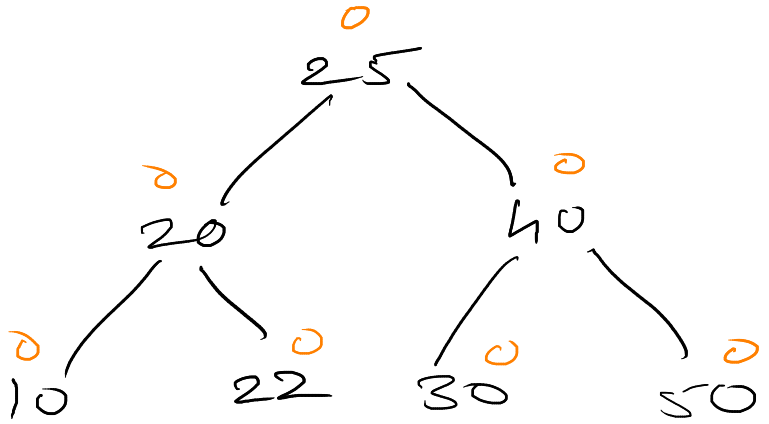

-2
20
10
R
30
L
25
40
L
0
22
RLL
RLR


20
10
25
22
30
40


0
25
0
20
-1
30
0
10
0
22
0
40

(After 22)


-1
25
20
-2
30
10
22
R
-1
40
R
0
50


0
25
0
20
0
40
0
10
0
22
0
30
0
50