AVL Tree

BST → Binary Search Tree

(Skewed BST)

↳ Height BST ≈ n

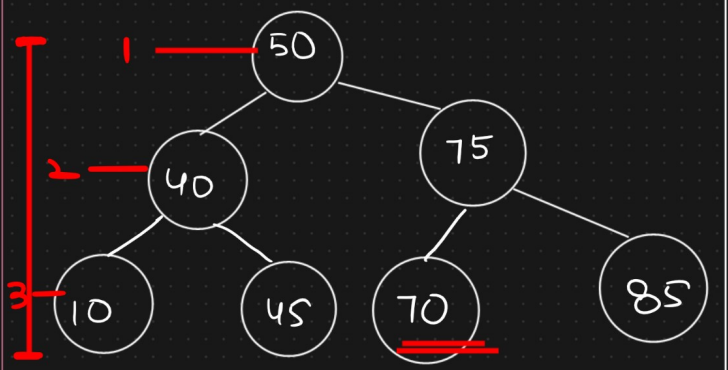
(Search time)

AVL Tree \ / Red Black Tree

Self Balancing BST

Balanced BST

50, 40, 10, 45, 75, 70, 85



Search = 70

50, 70

75, 70

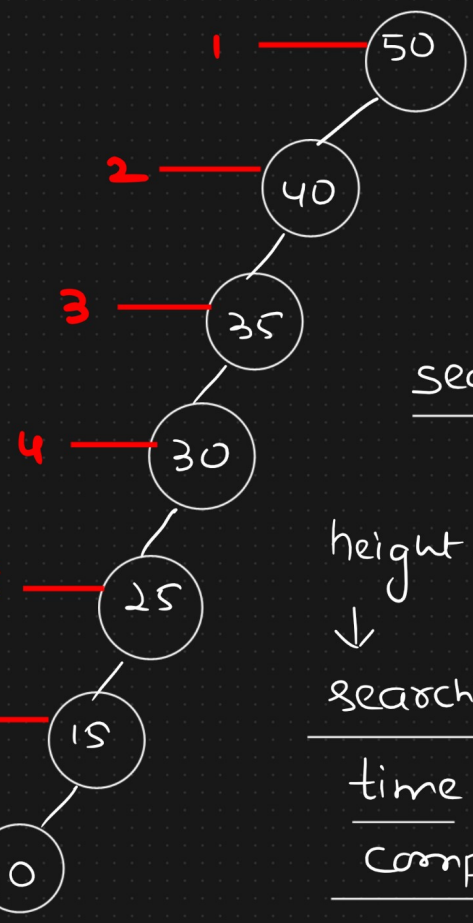
70, 70

height = $\Theta(\log n)$

Searching time complexity

Imbalanced BST

50, 40, 35, 30, 25, 15, 10



Search = 10

height = $\Theta(n)$

↓

Searching time complexity

AVL Tree

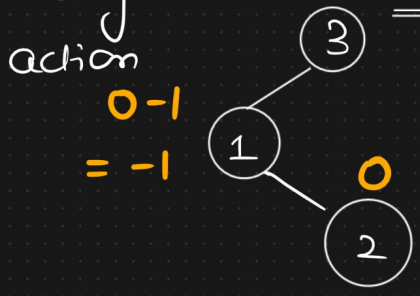
↳ Georgy Adelson-Velsky & Evgenii Landis

Balanced factor = Height of left subtree − Height of Right subtree

(0, +1, −1)

↳ Not need to take any action

2 − 0 = 2



0 − 1 = −1

Rotations

(+2, −2, +3, ---)

Balanced factor

> +1

< −1

1) LL Rotation = Right Rotation

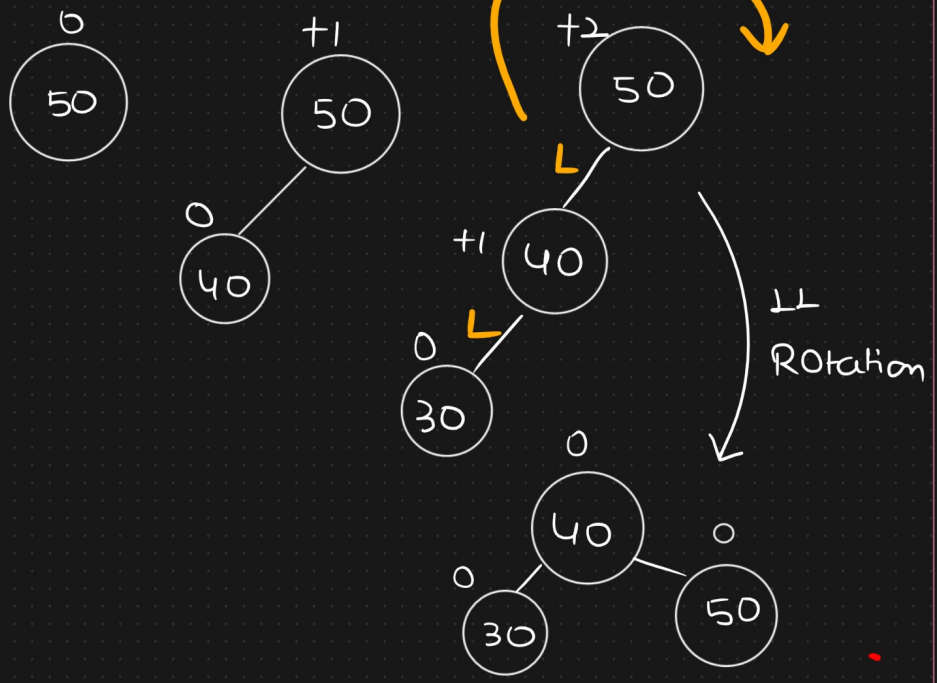
2) RR Rotation = Left Rotation

3) LR Rotation

4) RL Rotation

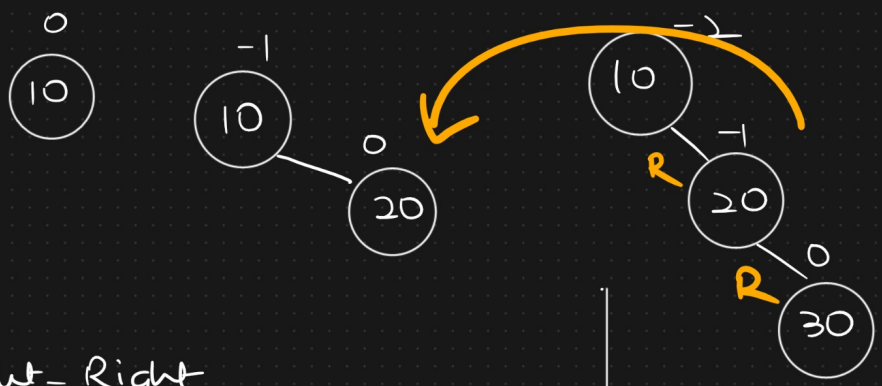
LL Rotation

50, 40, 30



RR Rotation

10, 20, 30

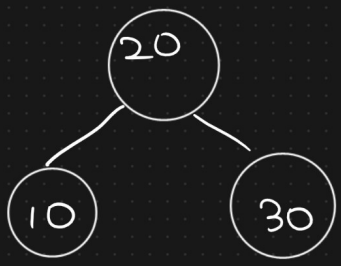


Left-Left & RR — Right-Right

↳ Middle node

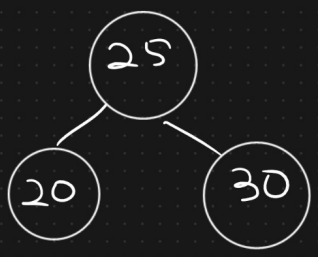
↳ Root Node after the rotation

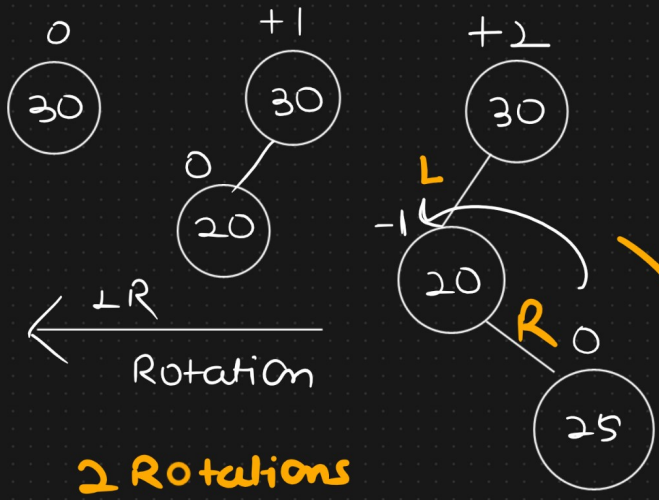
RR Rotation



LR Rotation

30, 20, 25

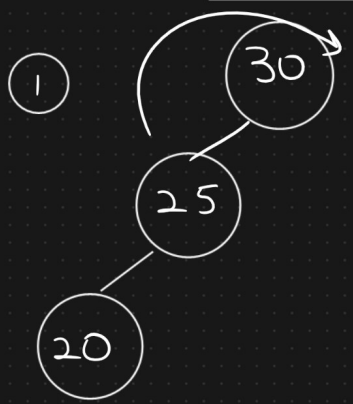




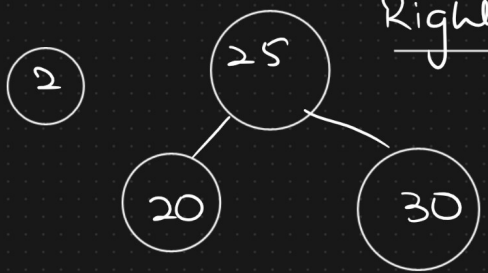
2 Rotations

LR Rotation

Left Rotation

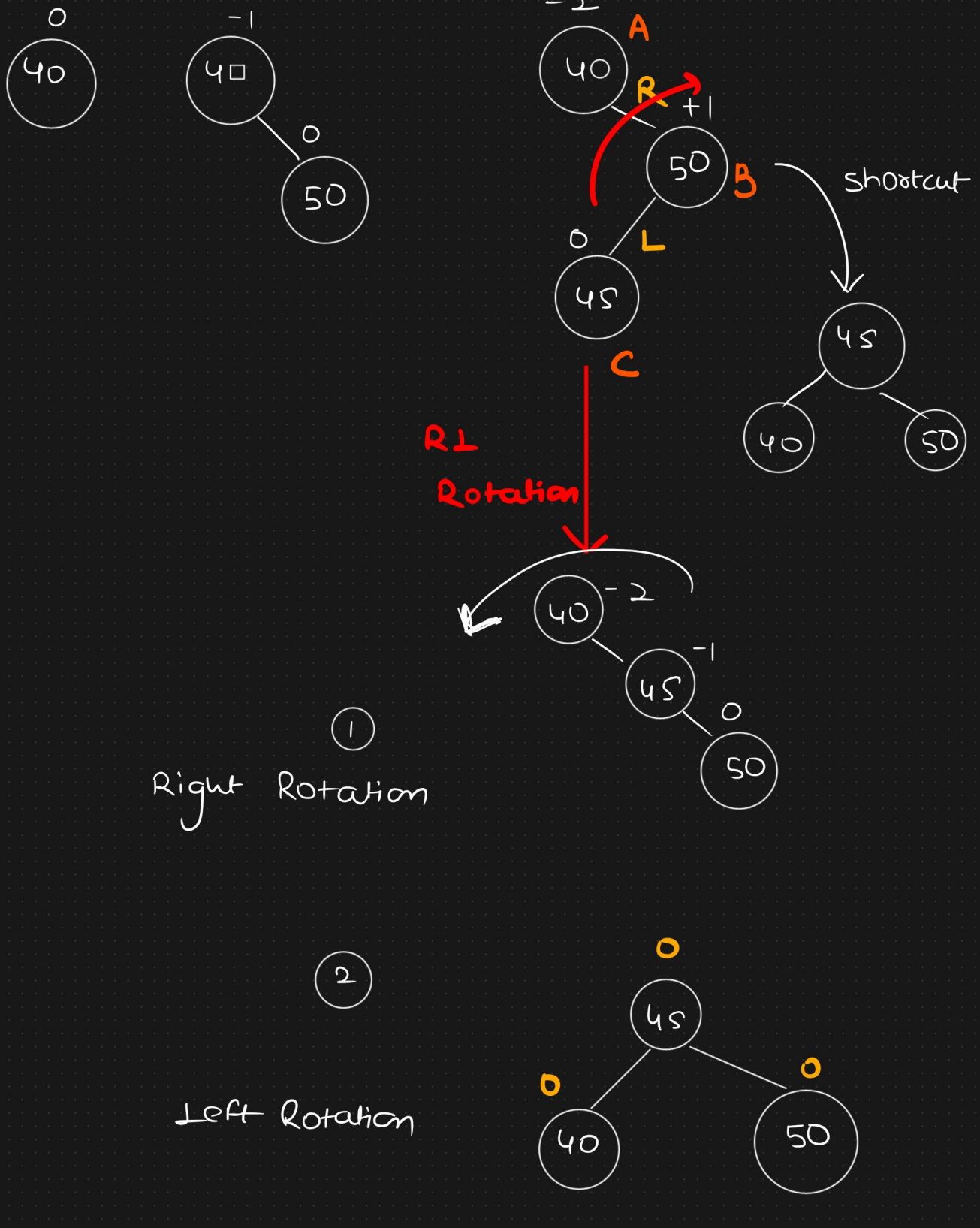


Right Rotation



RL Rotation

40, 50, 45

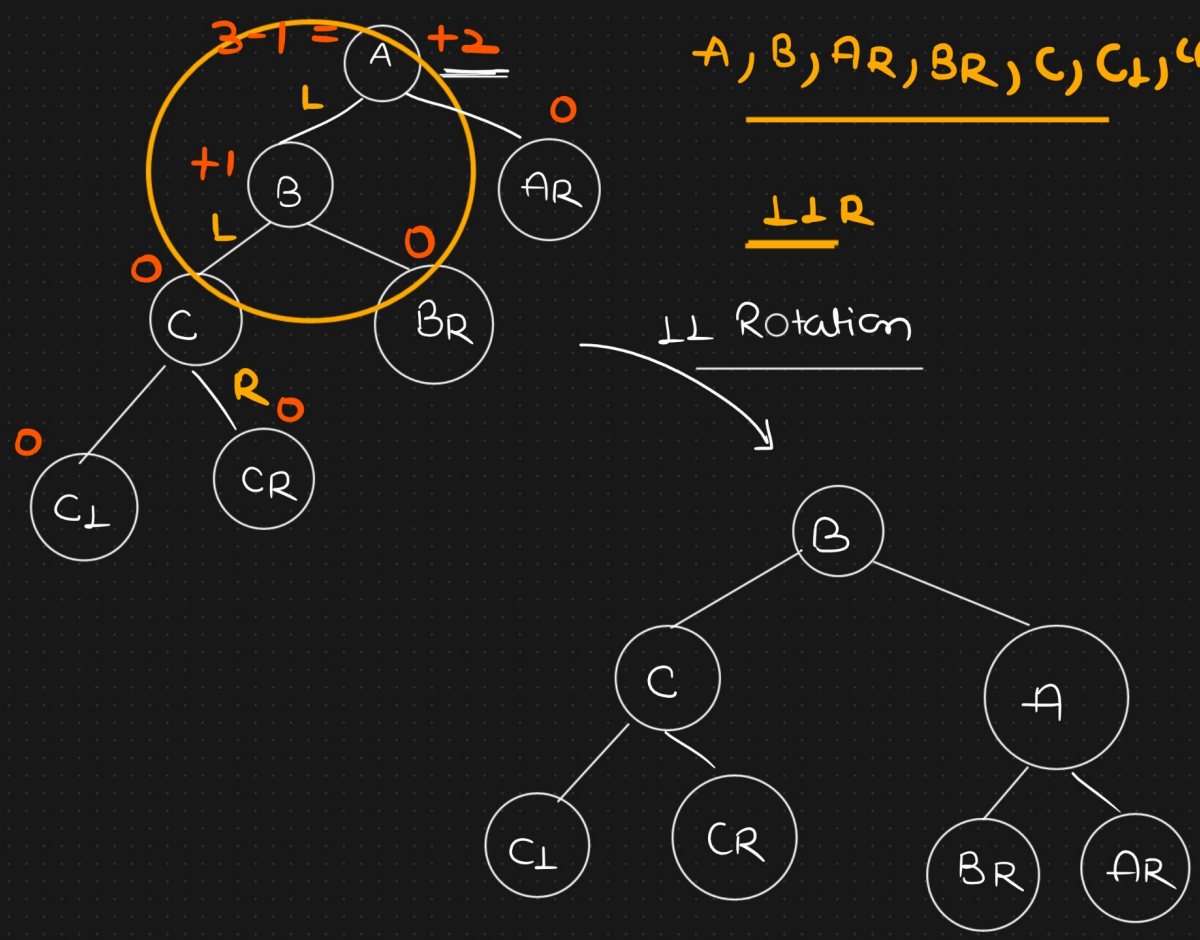


1

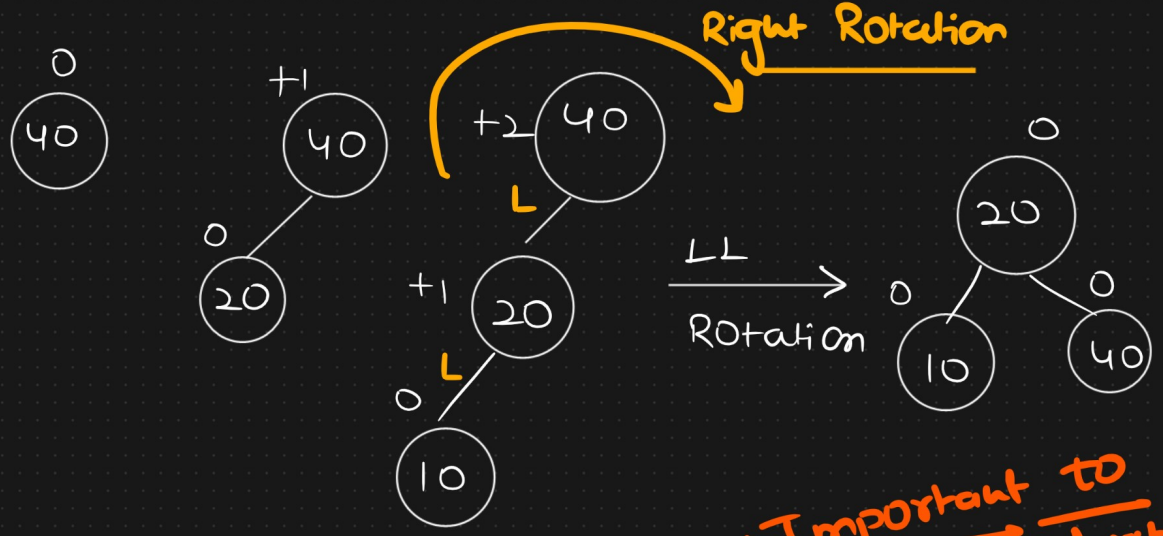
Right Rotation

2

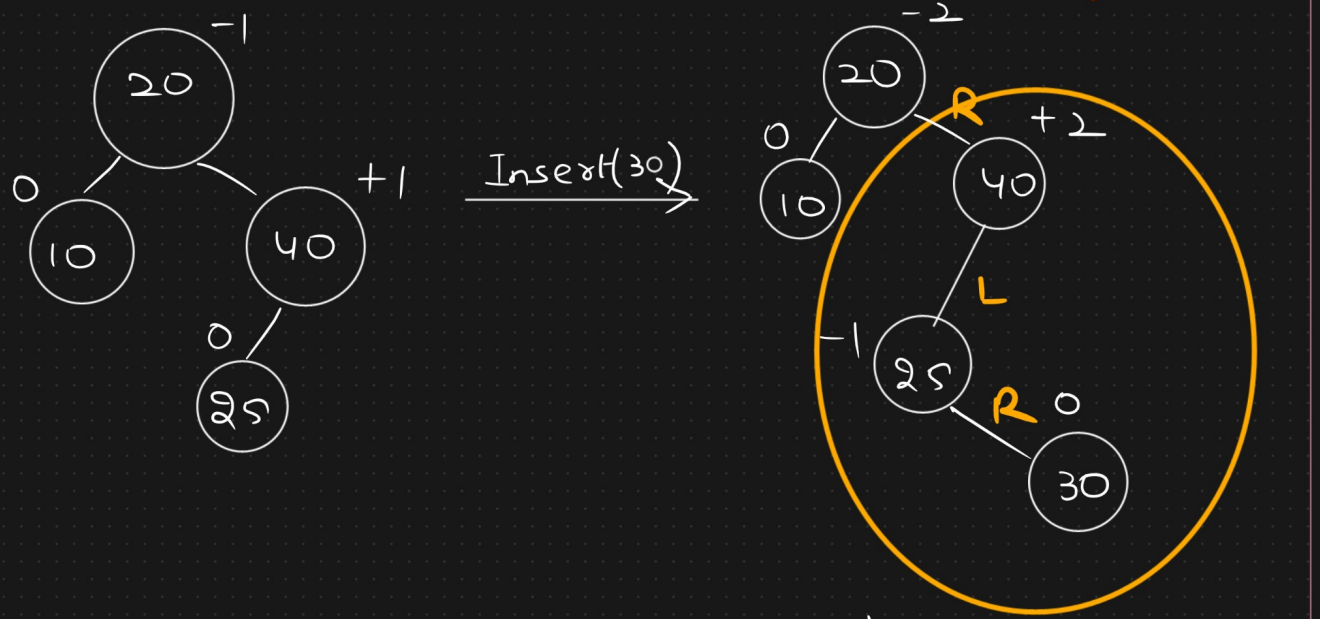
Left Rotation


3-1 = 2
A
+2
L
0
+1
B
AR
L
0
0
C
BR
R
0
0
CL
CR
A, B, AR, BR, C, CL, CR
LL R
LL Rotation
B
C
A
CL
CR
BR
AR

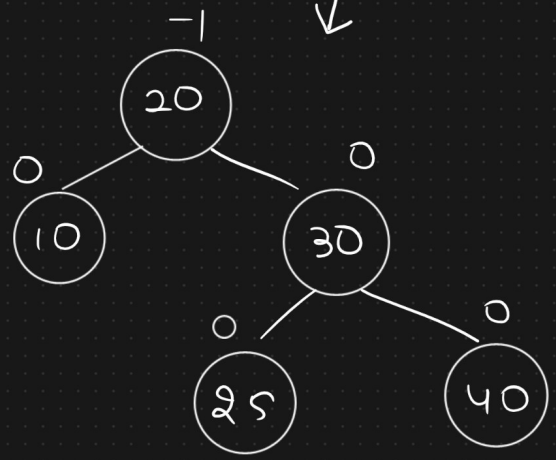
Keys = 40, 20, 10, 25, 30, 22, 50



*Important to Understand

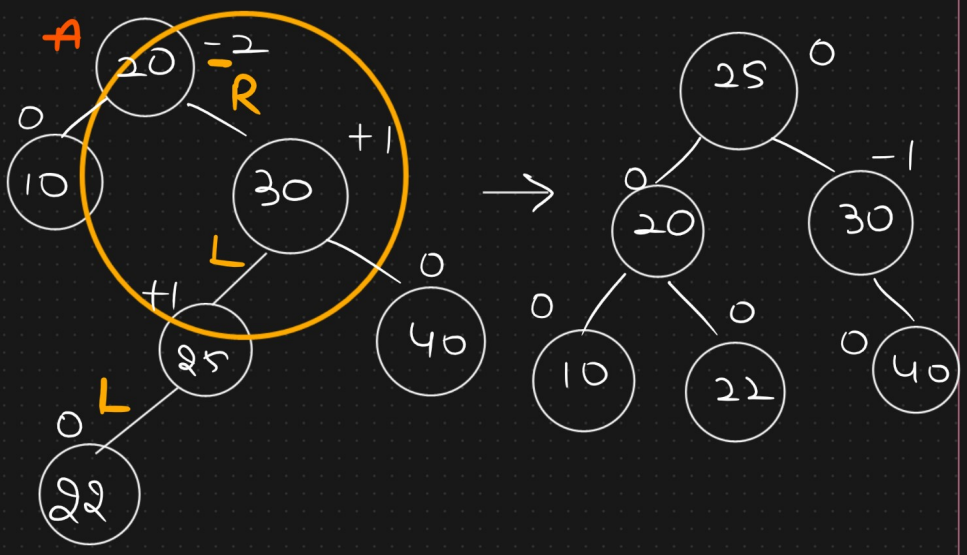


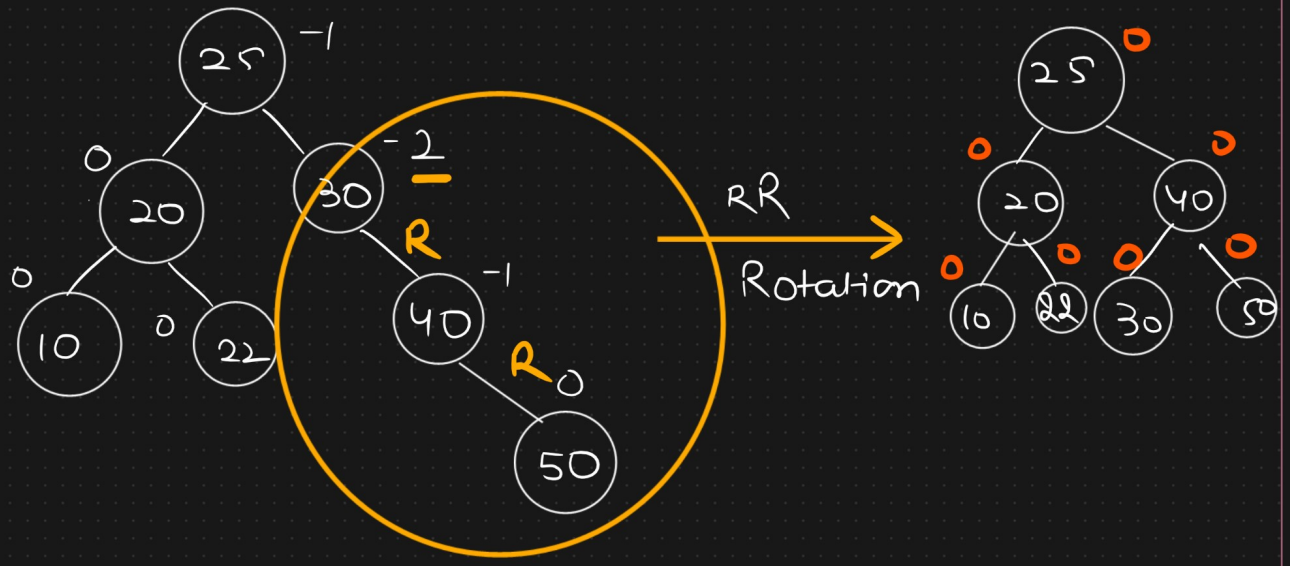
LR Rotation



20 30 25

A, B, C ⟶ RL Rotation



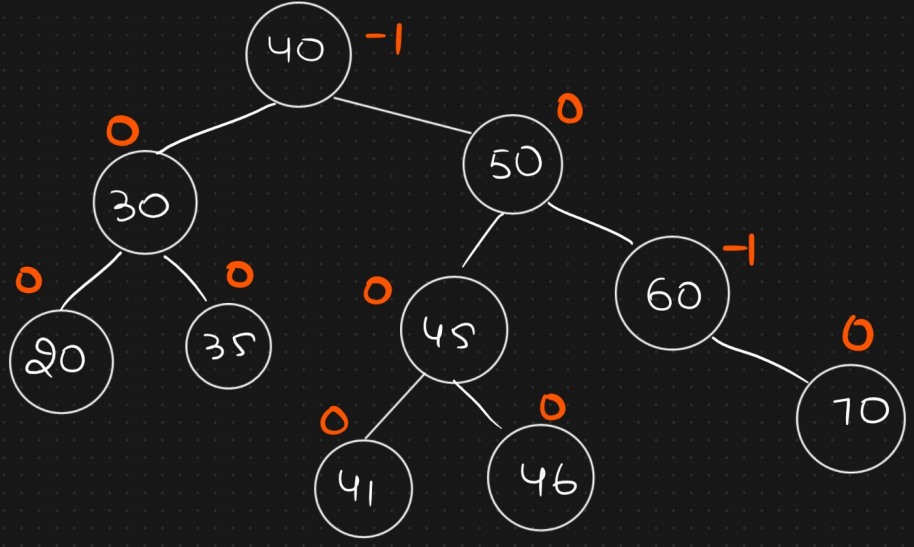


Note:

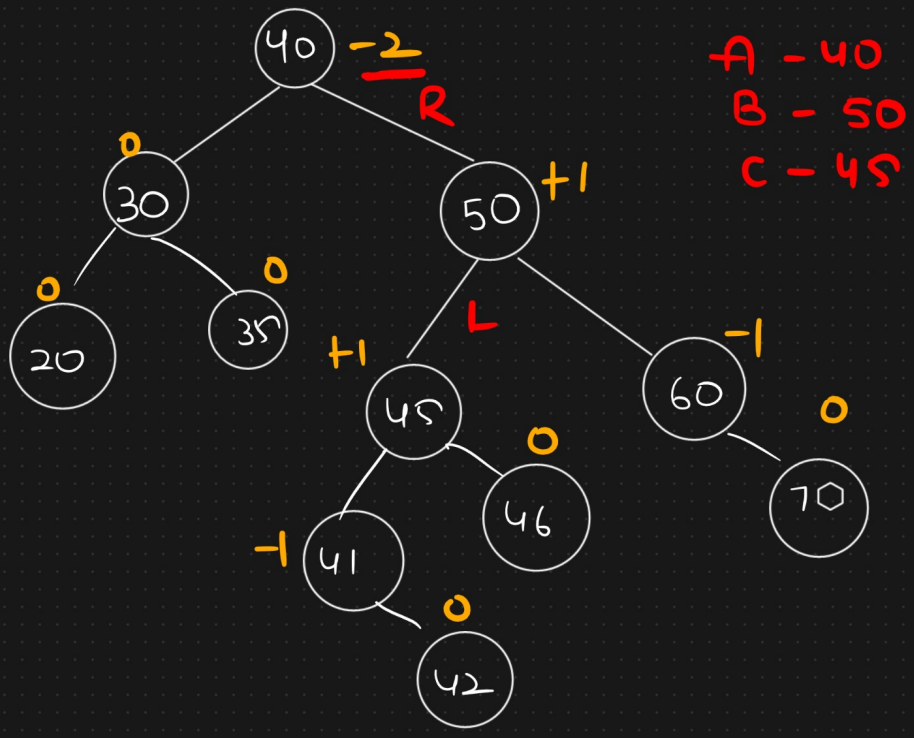
1) (+2, -2) only once side:
Pick three nodes (New node is inserted)

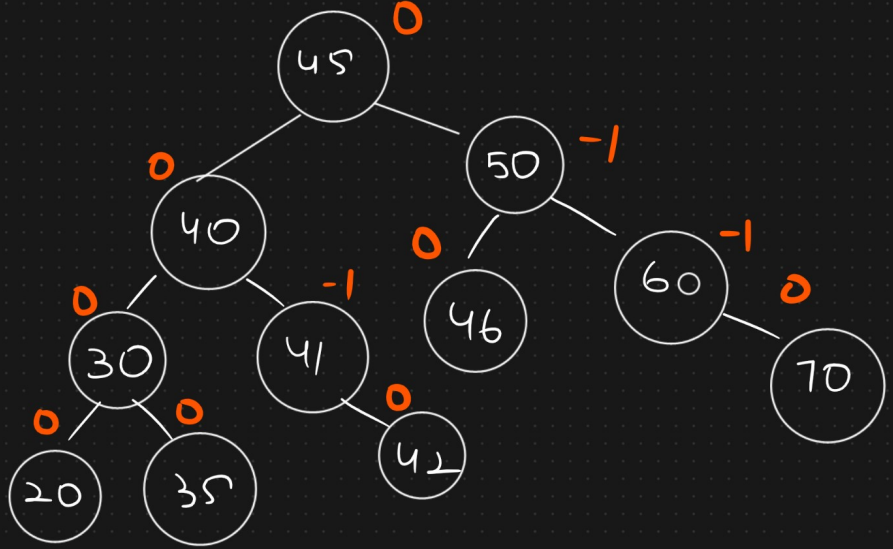
2) (+2, -2) twice
↳ focus on where actual insertion happened

3) Insertion always happened at leaf node Δ
after every insertion → update all nodes BF.


40
-1
30
0
50
0
20
0
35
0
45
0
60
-1
41
0
46
0
70
0

Insert → 42


40
-2
R
A - 40
B - 50
C - 45
30
0
50
+1
20
0
35
0
+1
L
45
60
-1
46
0
70
0
-1
41
0
42


45
0
40
0
50
-1
30
0
-1
41
46
0
60
-1
0
20
0
35
0
42
70

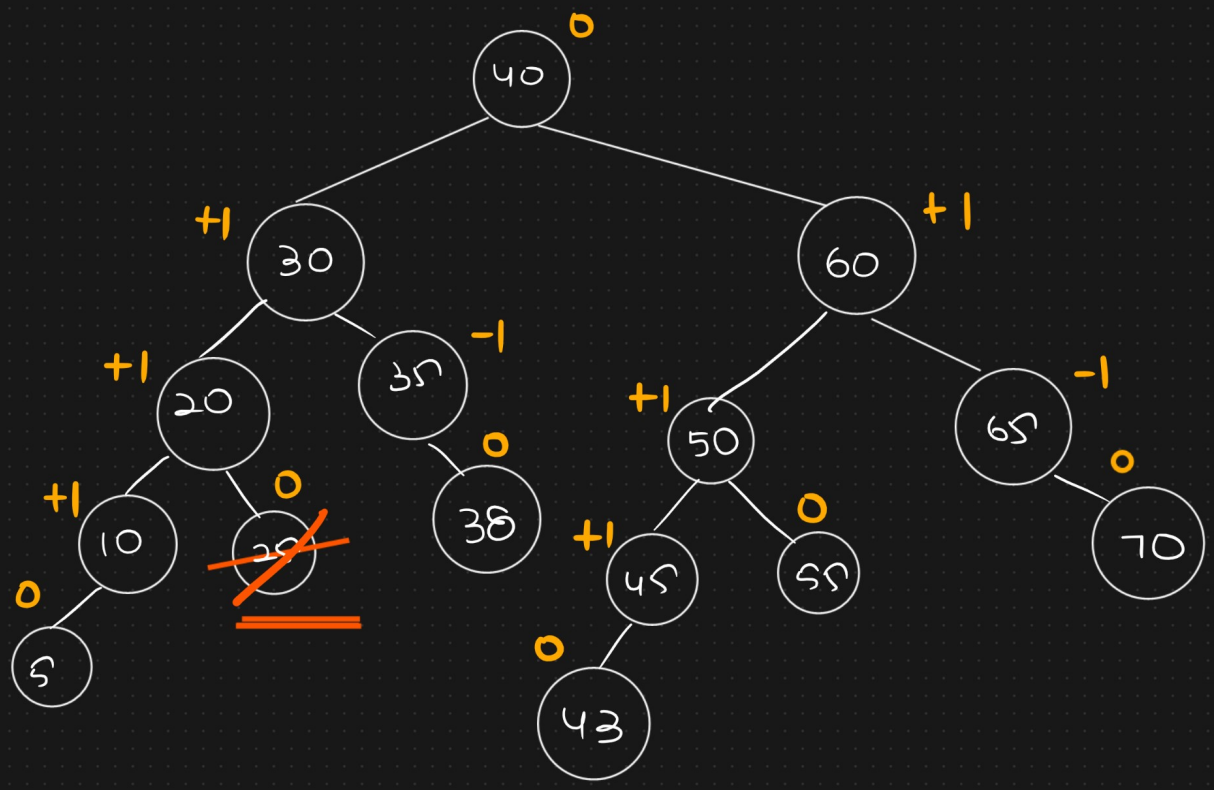
Deletion

1) No child (Leaf Node)

2) One child

3) Two child

- Inorder Successor
- Pred.

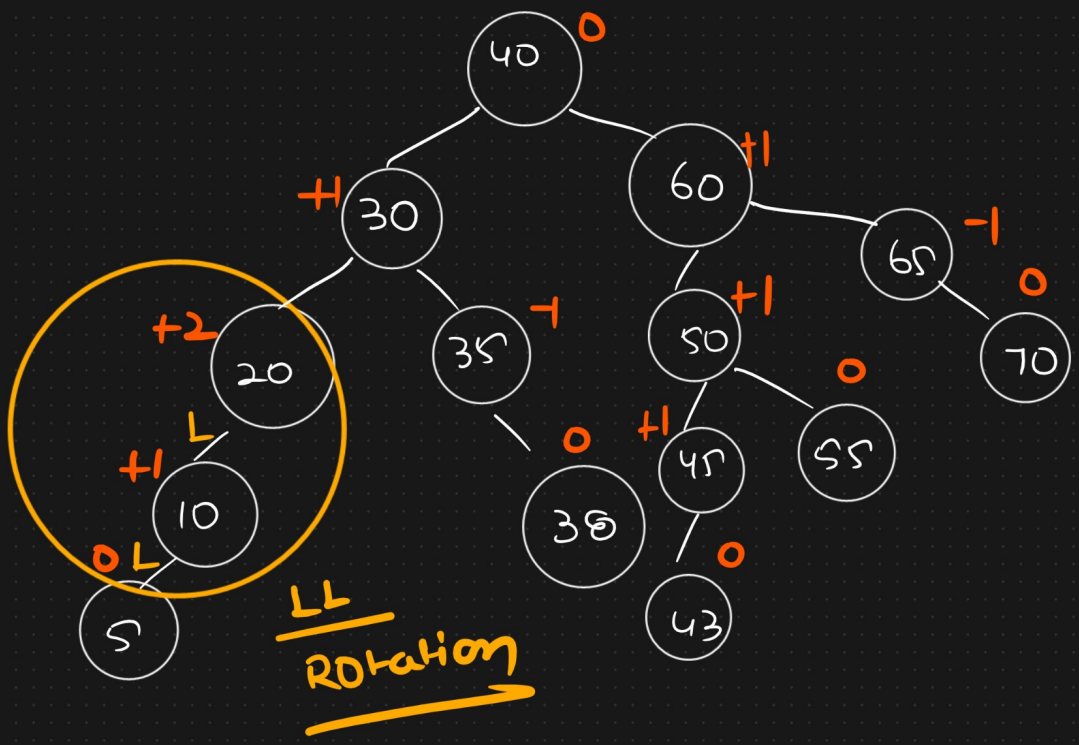


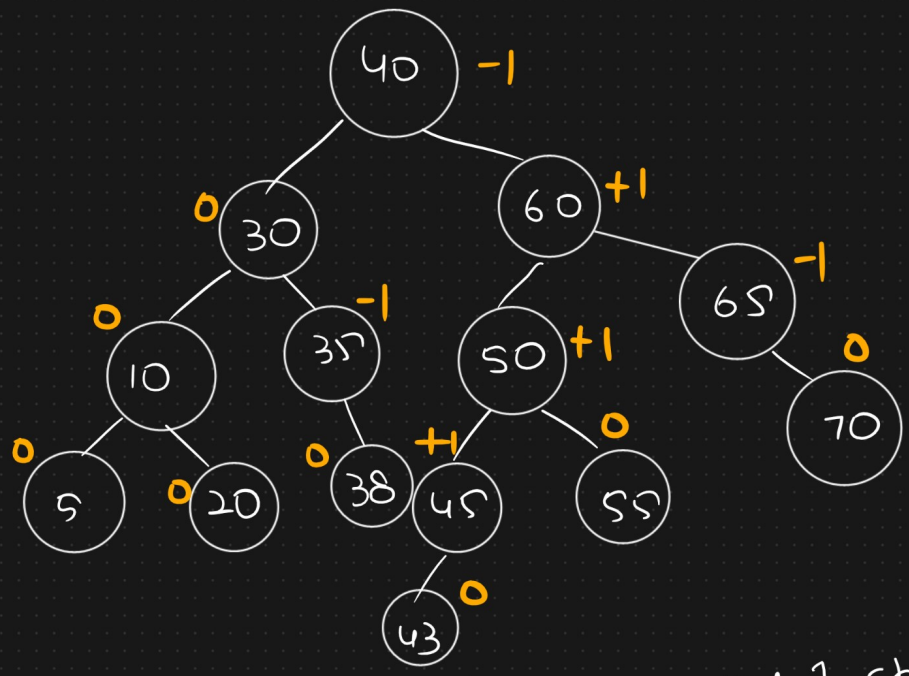
Search (25)

Delete 25

No child

Case 1

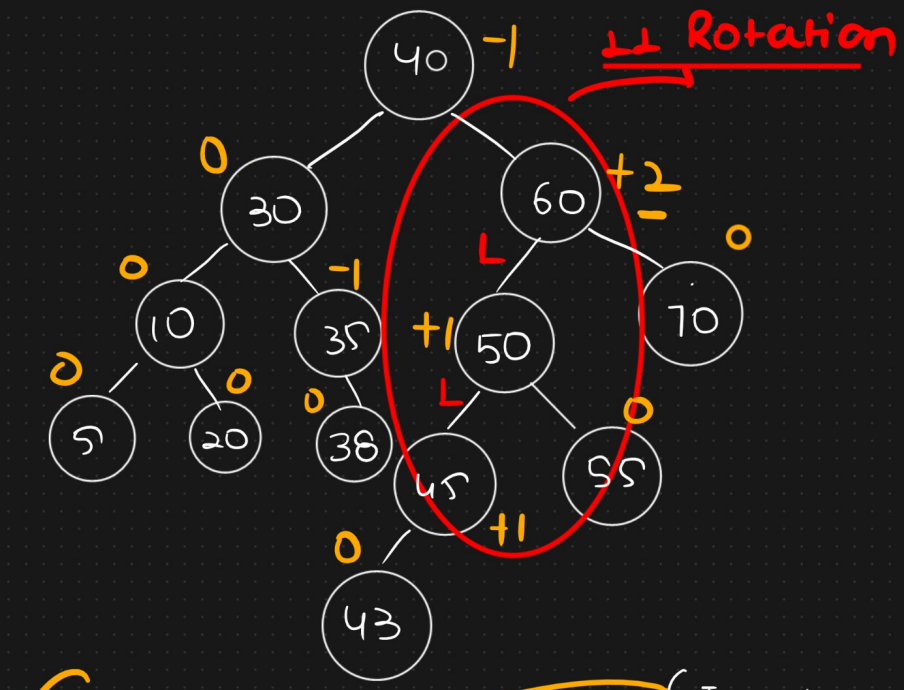




1 child

case 2

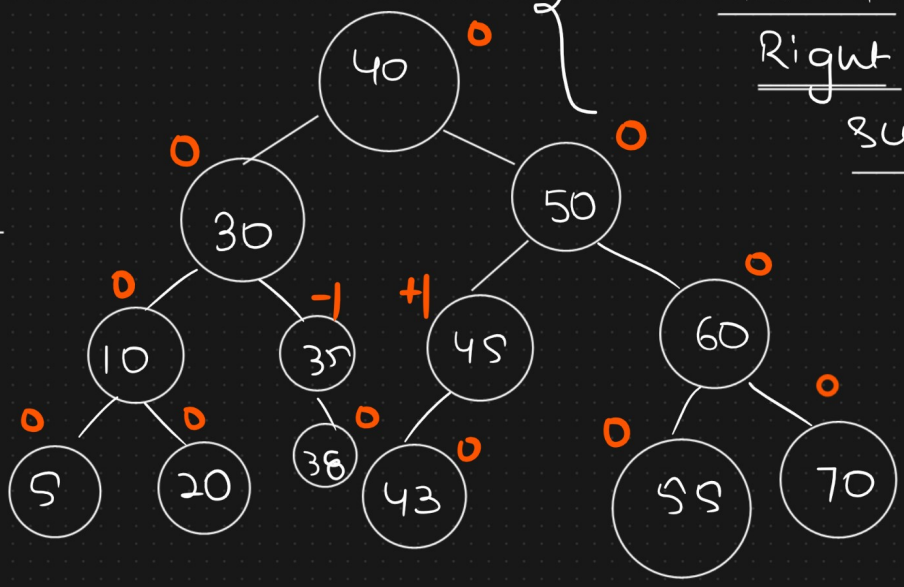
Delete -65



38 ← 40 → 43

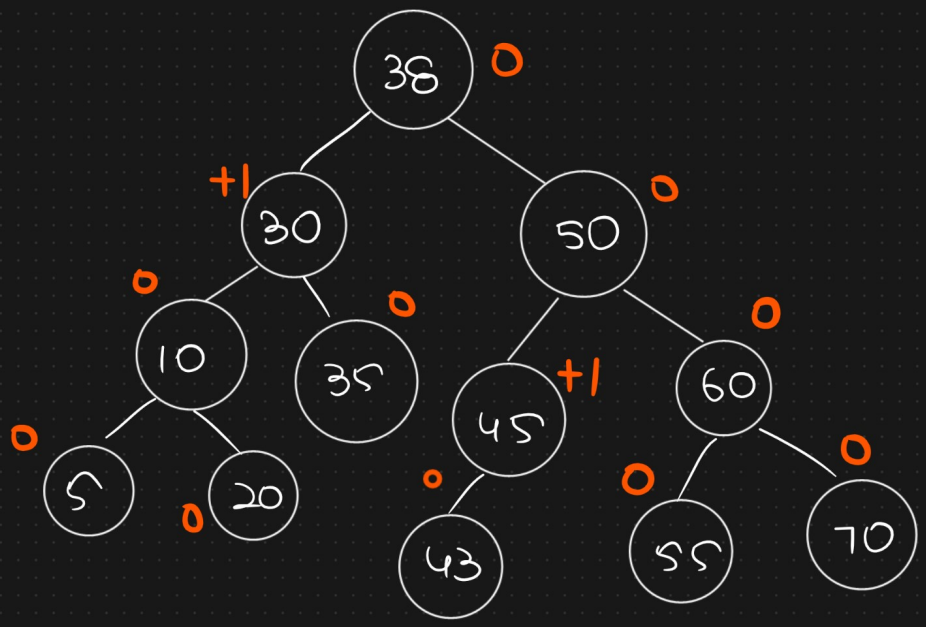
Inorder Pred.
↳ Max in Left Subtree

Inorder Successor
↳ Min in Right Subtree



Delete(40)

Case 3
↳ 2 -child



Time complexity → $\Theta(\log n)$

(Every case)

Drawback

↳ { Huge number of Rotations
Strict version of Balanced BST

↳ Greater amount of effort/time to generate result

↳ Two types of nodes

Red Black Tree — Red, Black

↳ Less rotations as compared to AVL Tree

↳ Result in much faster pace (AVL Tree)

↳ Red - Red node can't be adjacent

Insertion

↳ Black node → Root node

utility of every Data Structure

- Insertion & Deletion (frequent Operations) → Red Black Tree
- Search (frequent Operation) → AVL Tree