Data Structure and Algorithms

Session-34

Dr. Subhra Rani Patra
SCOPE, VIT Chennai

Heap

Priority Queue

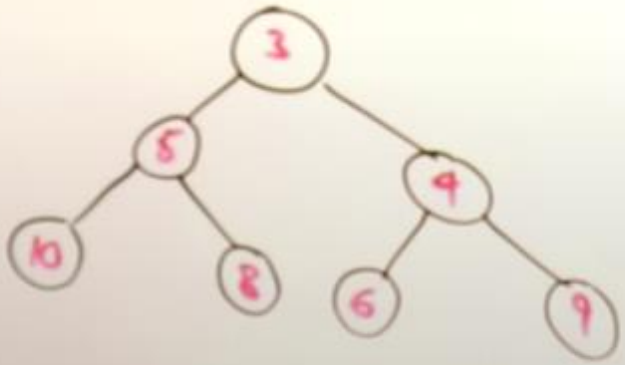
smaller number
Higher Priority

Large number
higher priority

A

1	2	3	4	5	6	7
8	6	3	10	5	4	9

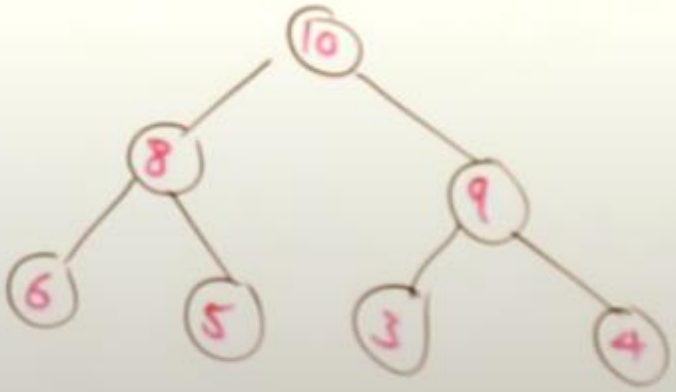
Min Heap



B

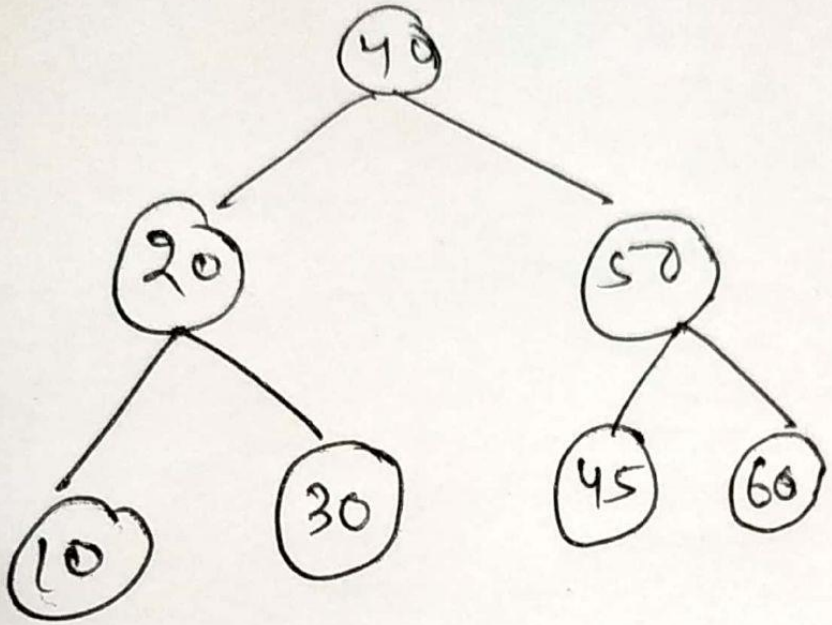
1	2	3	4	5	6	7
8	6	3	10	5	4	9

Max Heap



AVL Tree

BST

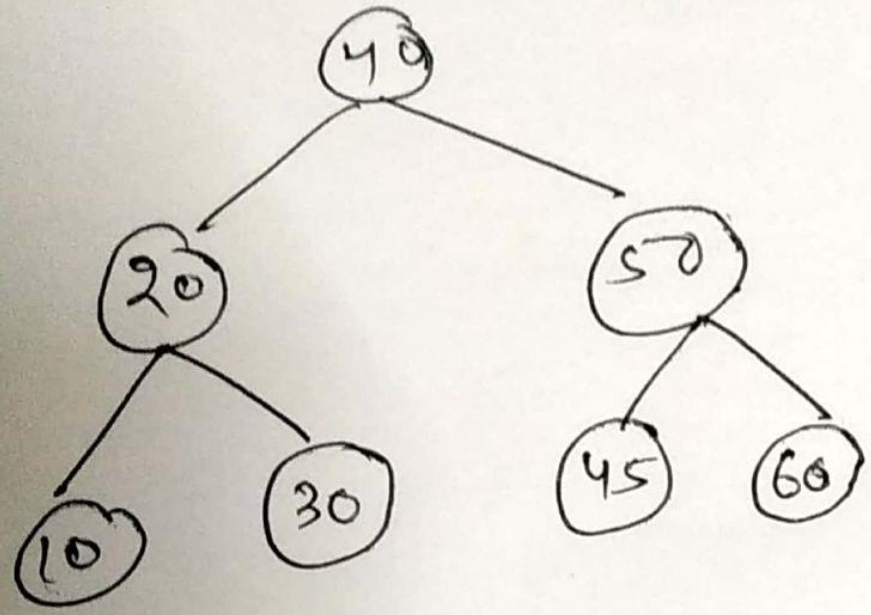


Key = 30

Key = 60

Key = 32

BST



min log n

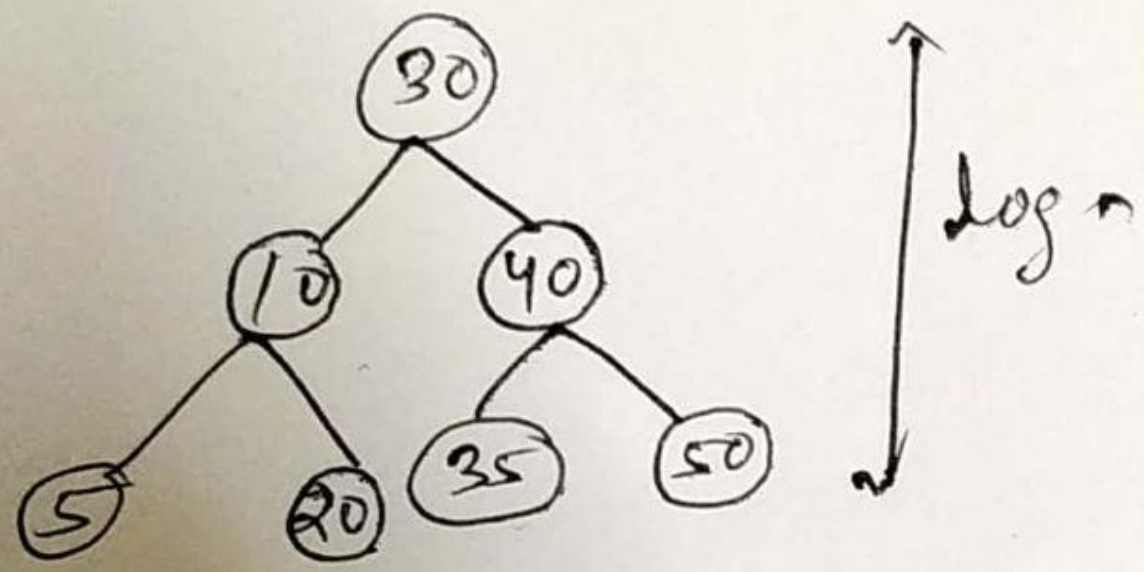
max n

key = 30

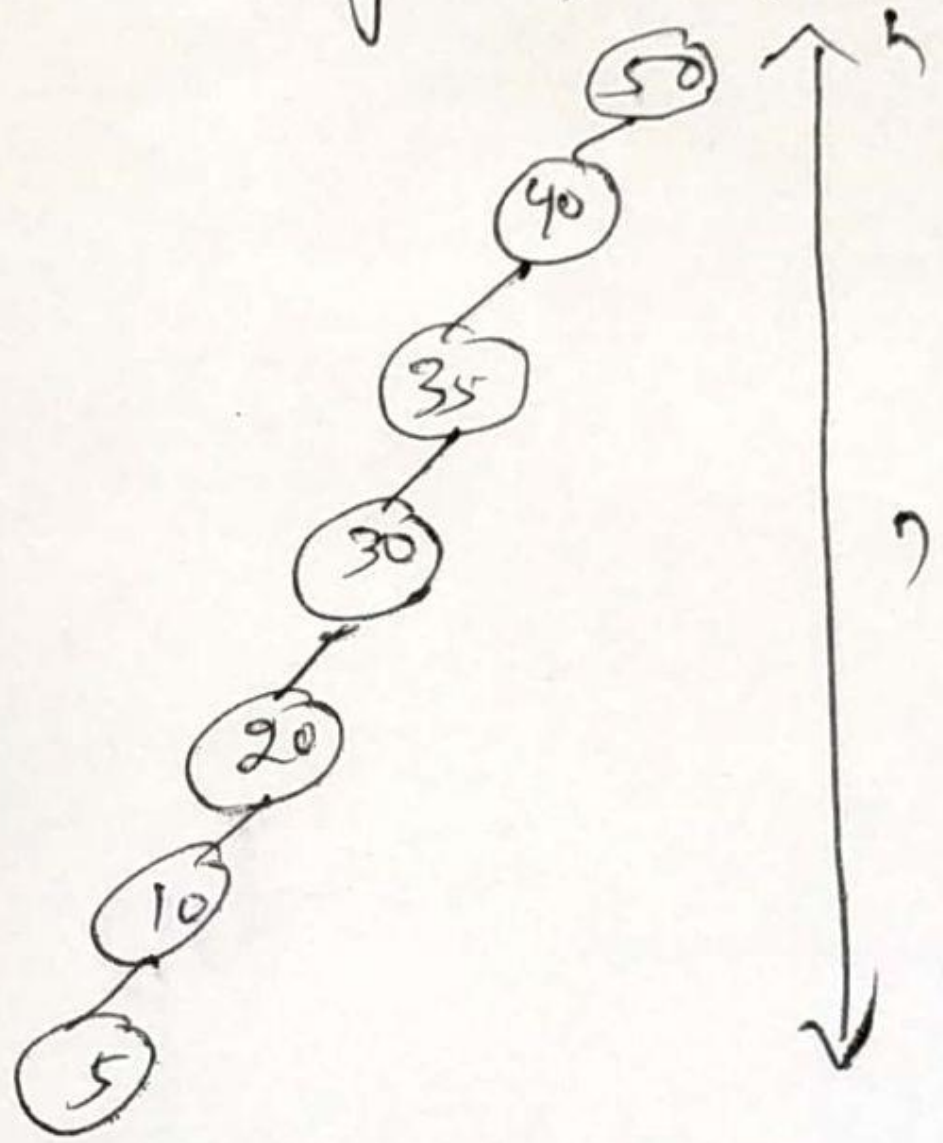
key = 60

key = 32

key - 30, 40, 10, 50, 20, 5, 35



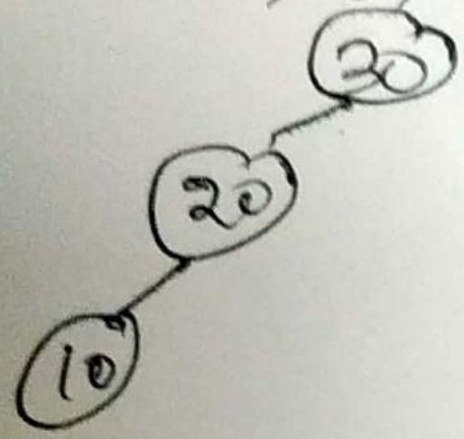
key 50, 40, 35, 30, 20, 10, 5



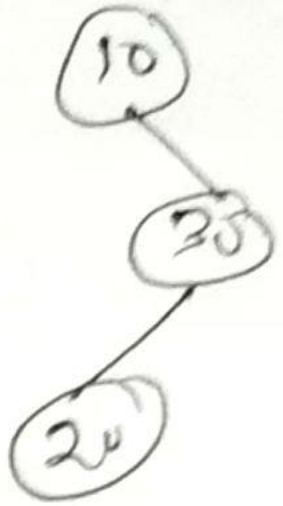
key — 30, 20, 10

10, 30, 20

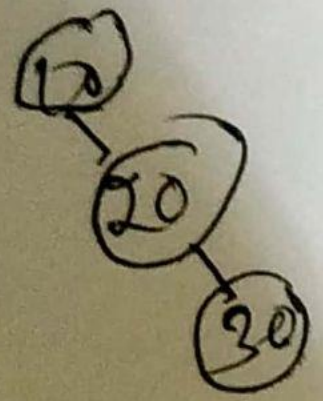
30, 20, 10



30, 10, 20

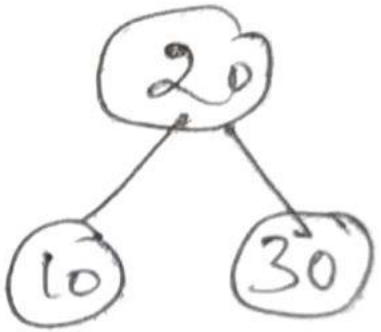


10, 20, 30



20, 10, 30

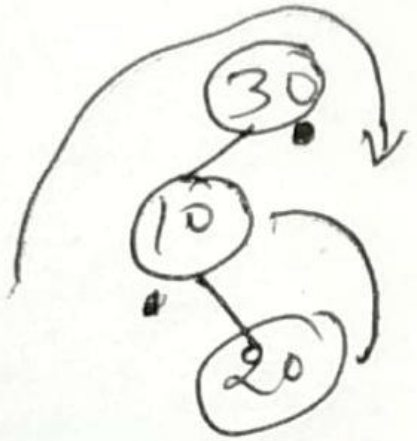
20, 30, 10



key — 30, 20, 10

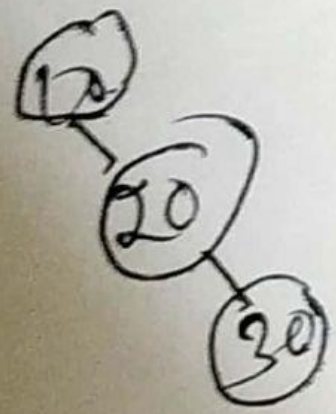
30, 20, 10

30, 10, 20



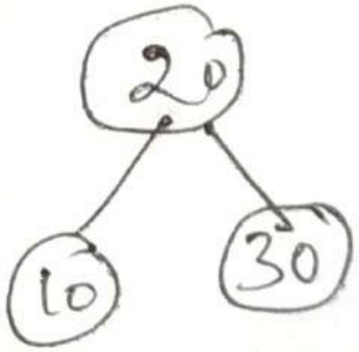
10, 30, 20

10, 20, 30



20, 10, 30

20, 30, 10

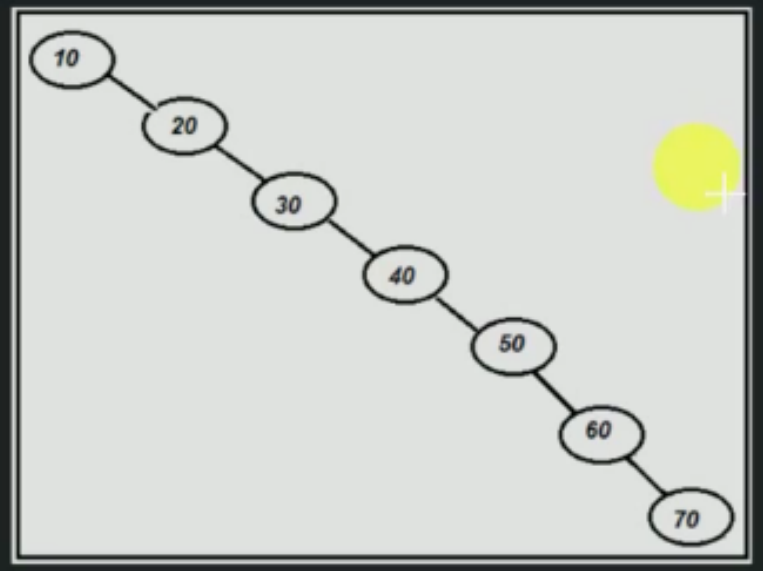


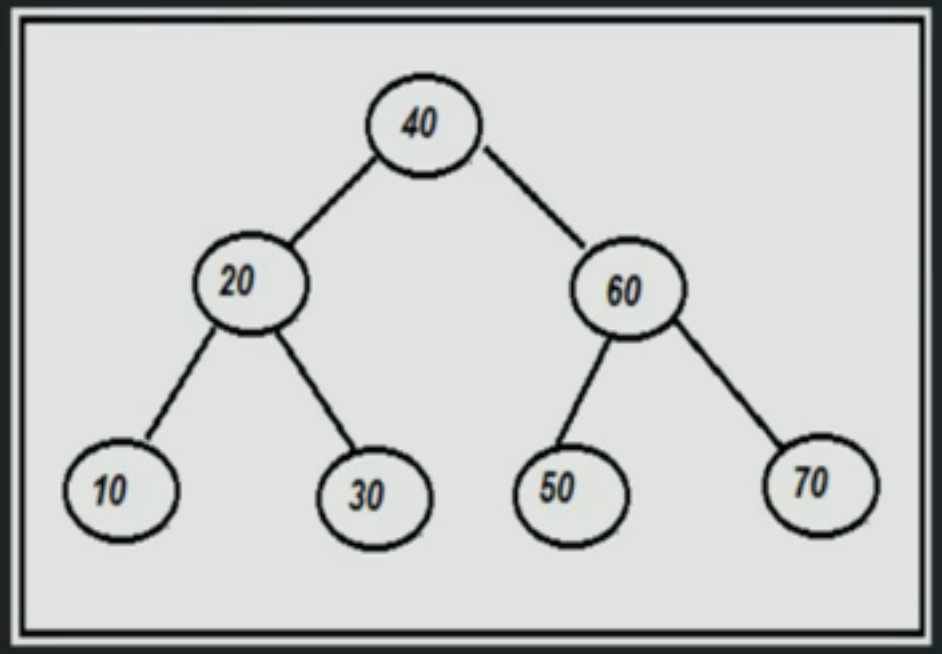
Why AVL Tree ?

- ✓ Depending on Incoming data, A Binary Search tree can get skewed and hence its performance starts going down. Instead of O(log n) for insertion/searching/deleting it can go Upto O(n).

- ✓ AVL tree attempts to solve this problem of 'skewing' by introducing concept called 'Rotation'.

- ✓ Question:
  - ✓ Insert 10,20,30,40,50,60,70 in BST





What is AVL Tree ?

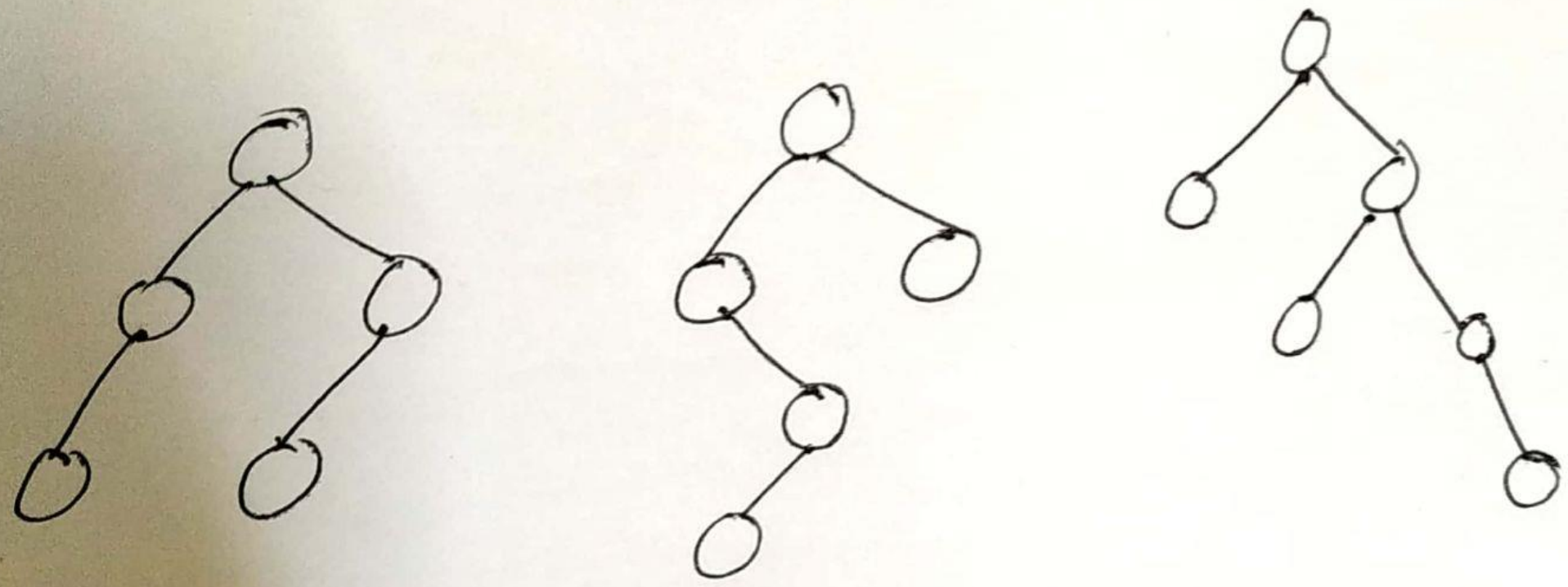
- ✓ An AVL tree is a balanced 'Binary Search Tree' where the height of immediate subtrees of any node differs by at most one (also called balance factor).

- ✓ If at any time heights differ by more than one, rebalancing is done to restore this property (called **rotation**).

AVL Tree

Balance factor = height of left subtree - height of right subtree

$$bf = h_l - h_r = \{-1, 0, 1\}$$

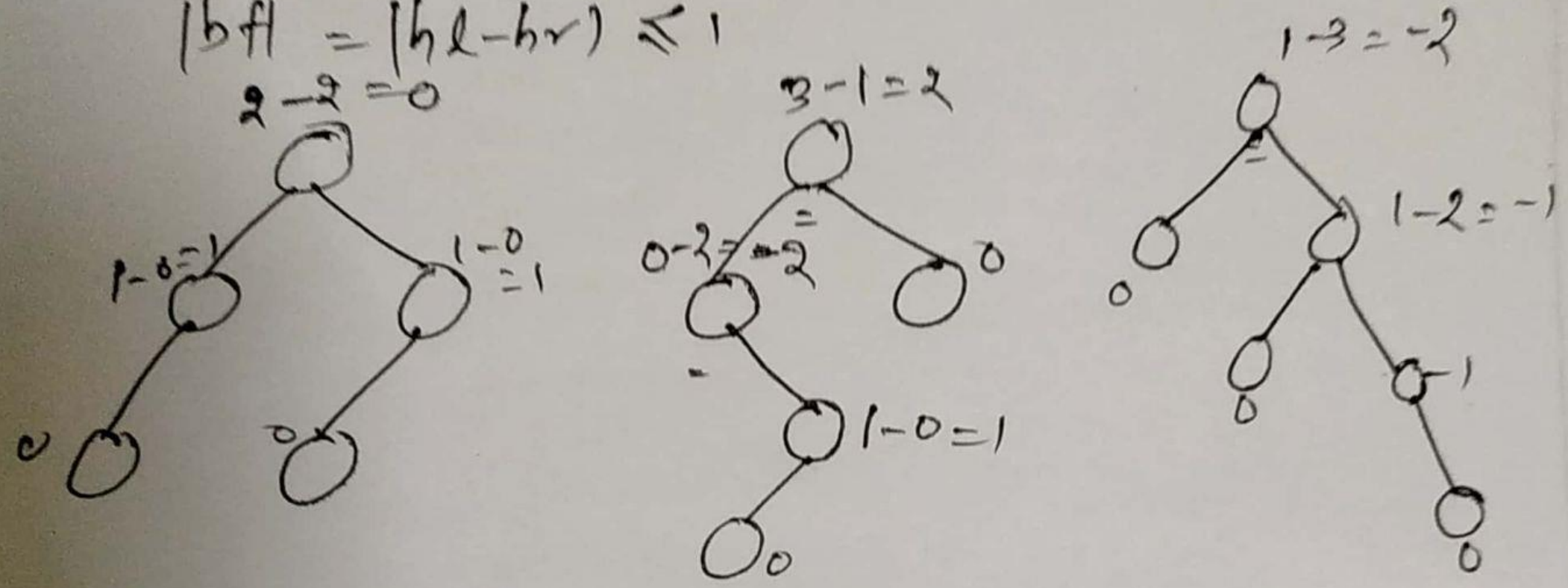


AVL Tree

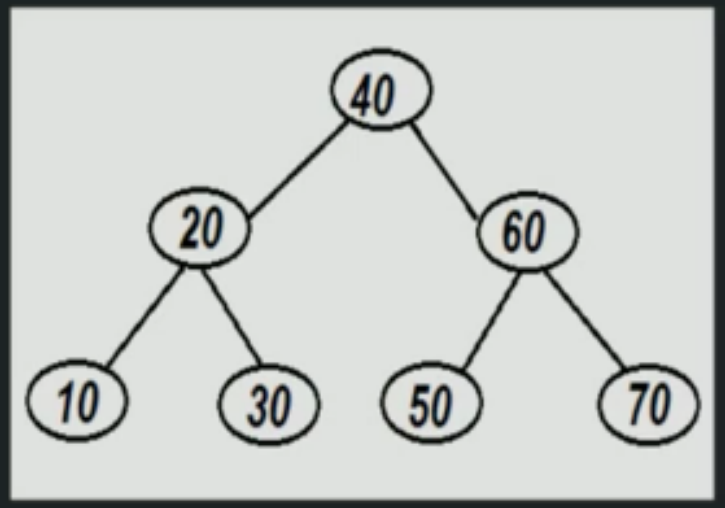
Balance factor = height of left subtree - height of right subtree

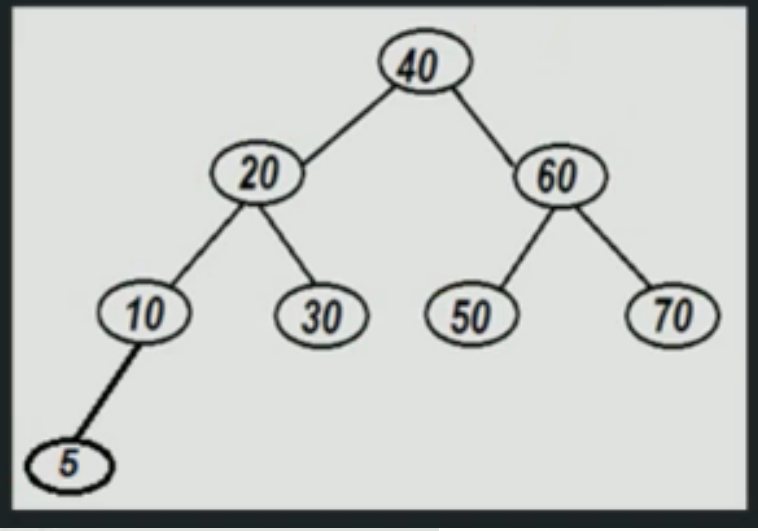
$$bf = h_l - h_r = \{-1, 0, 1\}$$

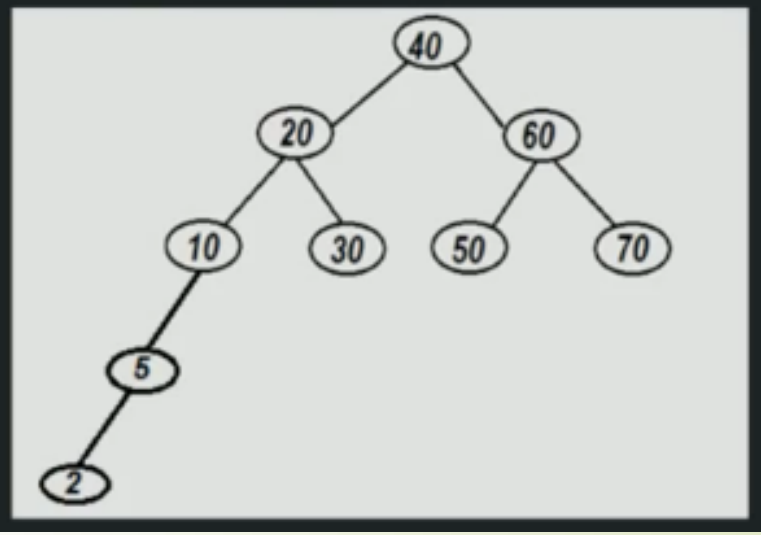
$$|bf| = |h_l - h_r| \leq 1$$

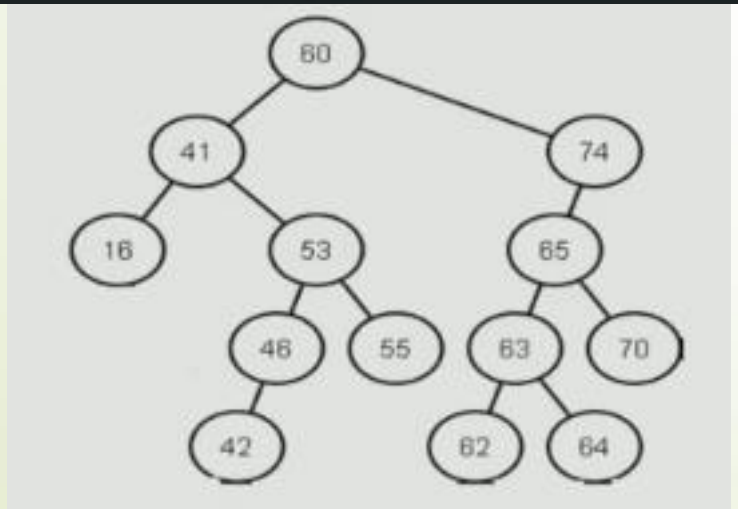


Examples of AVL Tree:









Common operations of AVL Tree:

- ✓ Create a AVL Tree
- ✓ Search a node
- ✓ Traverse all nodes
- ✓ Insert a node
- ✓ Delete a node
- ✓ Delete the AVL Tree

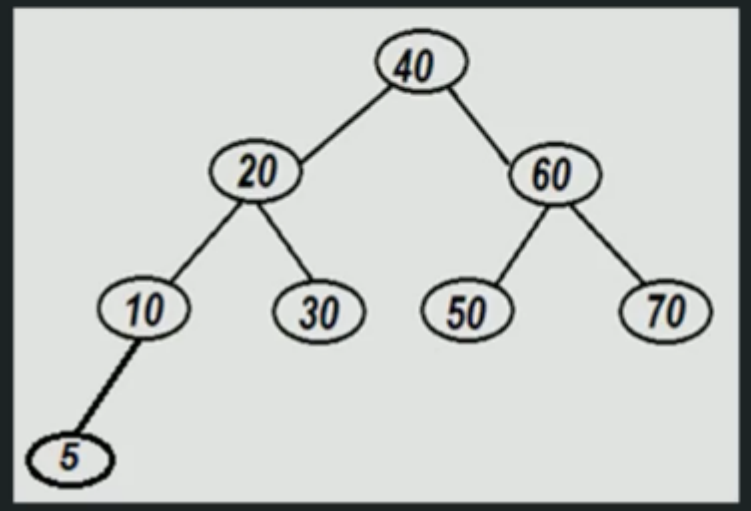
Algorithm - Creation of blank AVL Tree:

createAVL()

Initialize Root with null

Algorithm - Searching a node in AVL Tree:

```
AVL_Search (root, value):
    if (root is null)
        return null
    else if (root == value)
        return root
    else if (value < root)
        AVL_Search (root.left, value)
    else if(value > root)
        AVL_Search (root.right, value)
```

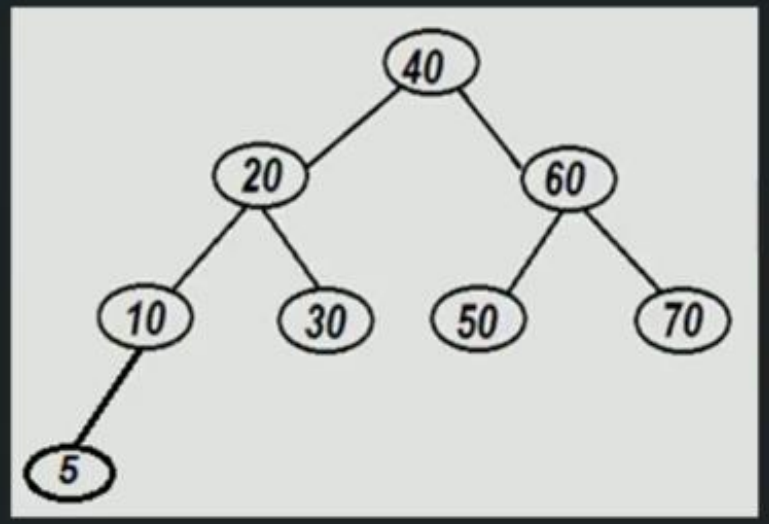


Traversing all nodes of AVL Tree:

- ✓ Depth First Search:
  - ✓ PreOrder Traversal
  - ✓ InOrder Traversal
  - ✓ PostOrder Traversal
- ✓ Breadth First Search:
  - ✓ LevelOrder Traversal

Insertion of node in AVL Tree:

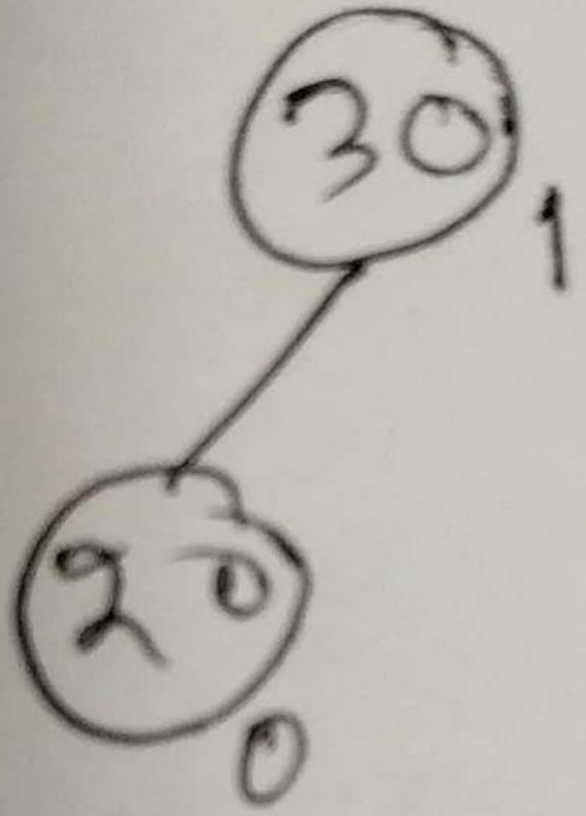
- ✓ Insertion of a node:
  - ✓ Case#1 - When 'rotation' is not required.
  - ✓ Case#2 - When 'rotation' is required (LL, LR, RR, RL).



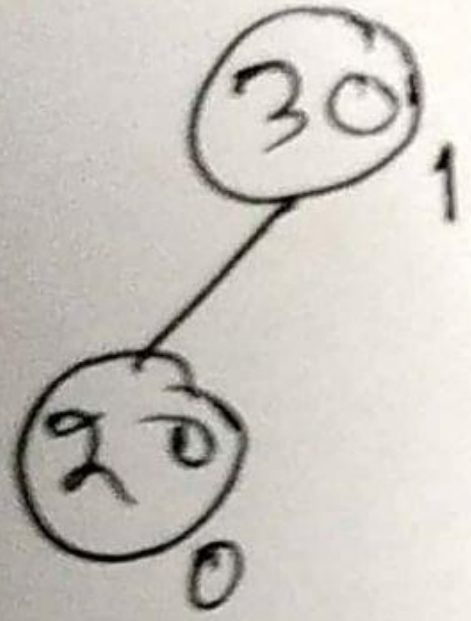
‘Rotation’ Conditions:

- ✓Left Left condition (LL)
- ✓Left Right condition (LR)
- ✓Right Right condition (RR)
- ✓Right Left condition (RL)

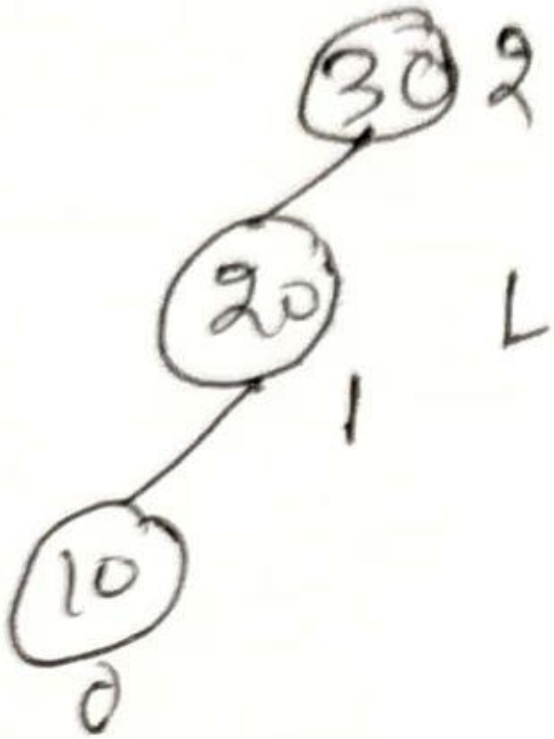
Initially



Initially



Insert 10

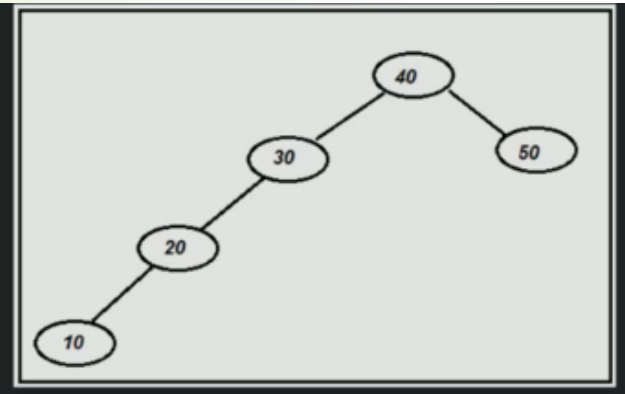


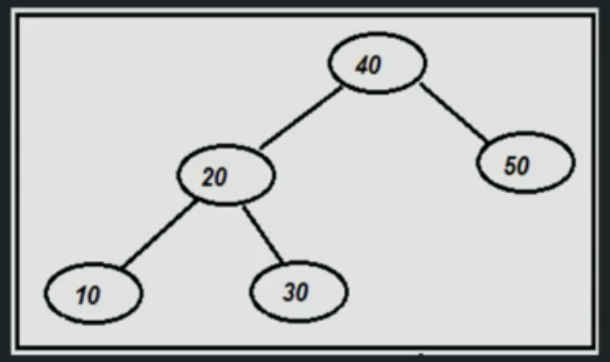
LL-Imbalance

✓What is Left-Left condition ?

✓Left-Left Node from currentNode is causing disbalance.

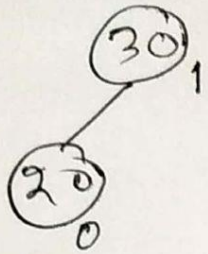
✓In this case we do a 'Right Rotation'



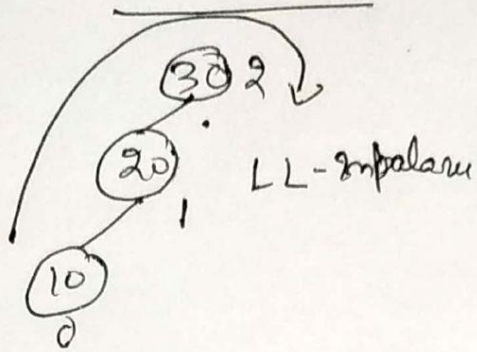


LL

Initially

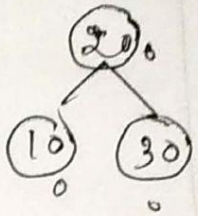


Insert 10

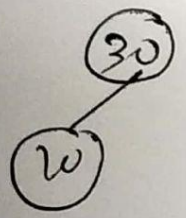


LL-imbalance

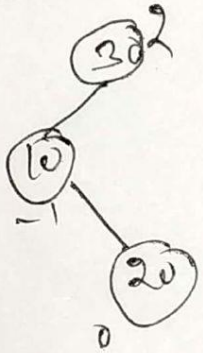
After Rotation



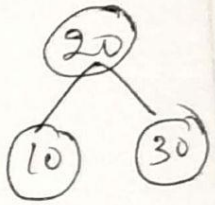
Initially



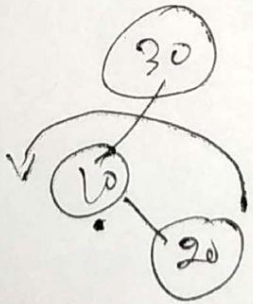
Insert 20

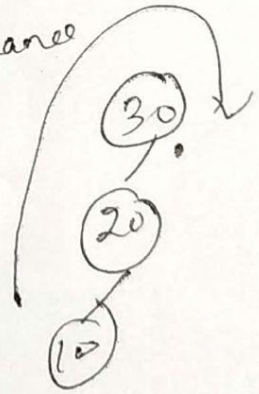


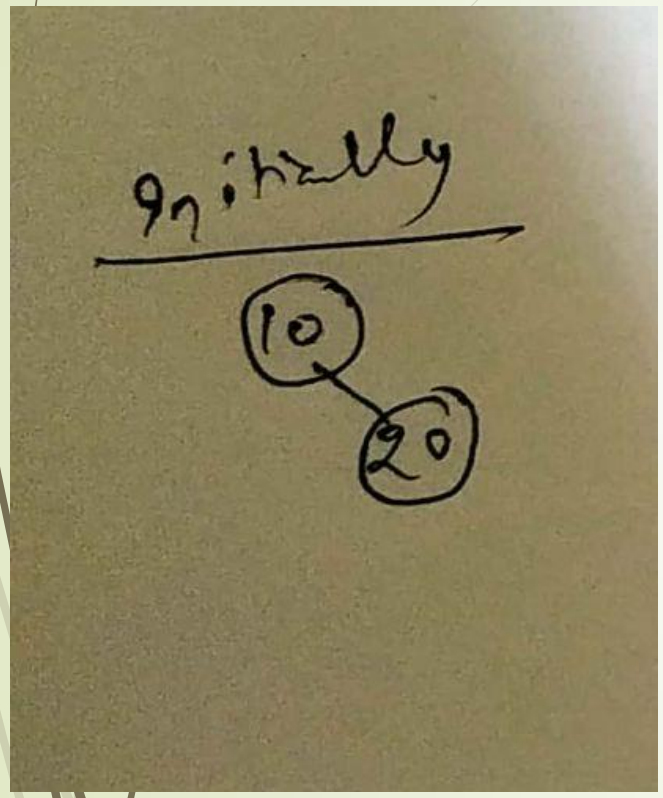
After LR Rotation

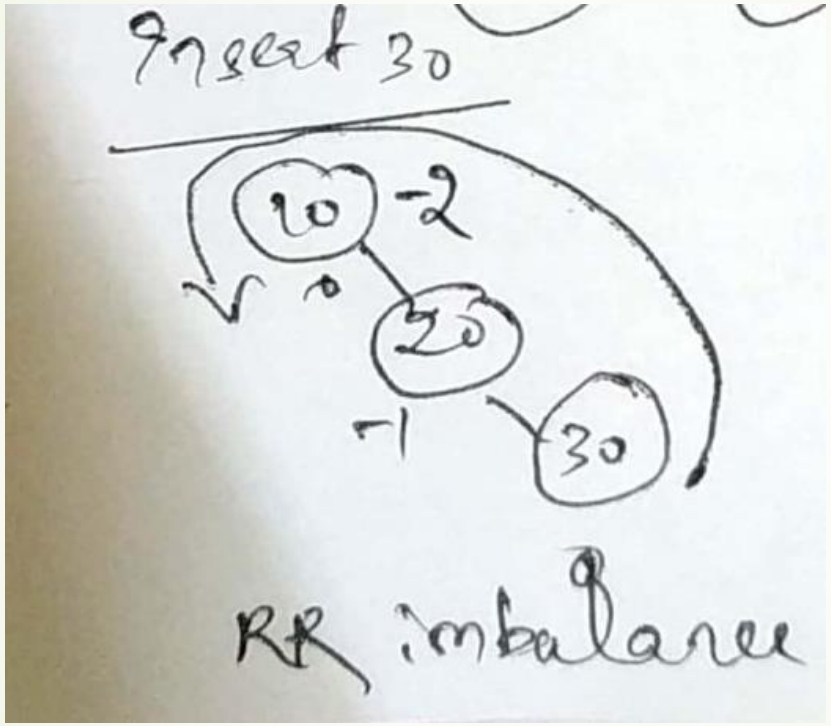


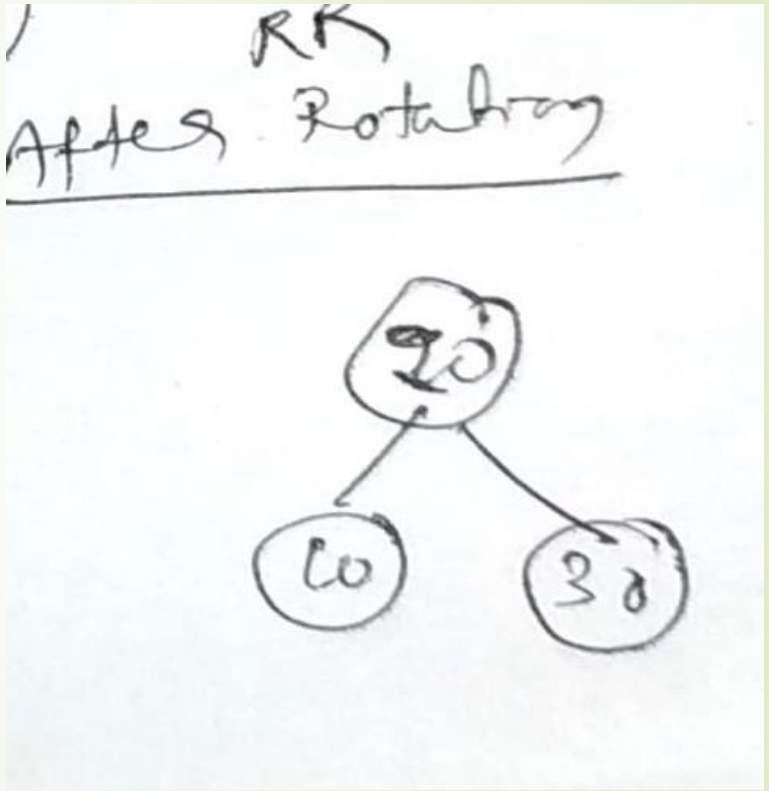
LR imbalance



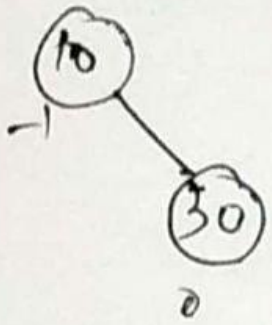



Initially
10
20

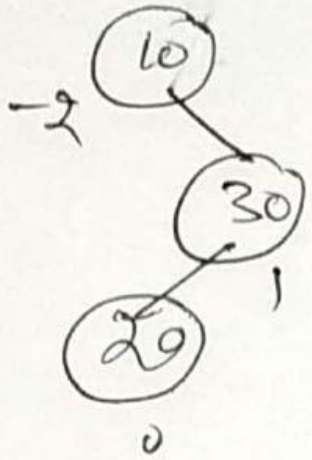

Insert 30
10
-2
0
20
-1
30
RR imbalance


RR
After Rotation
20
10
30

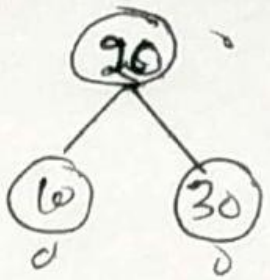
Initially



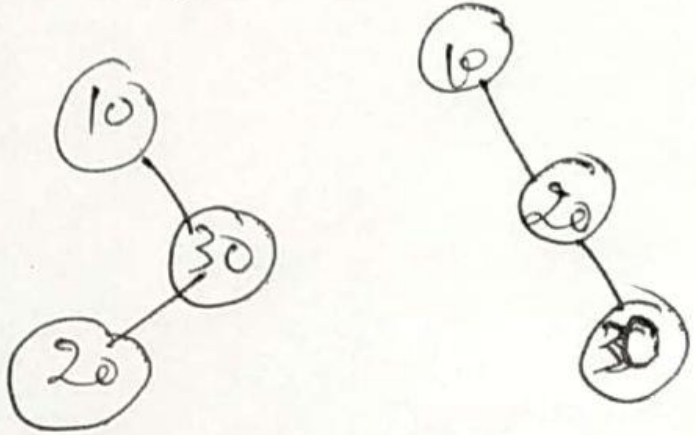
Insert 20



After RL Rotation

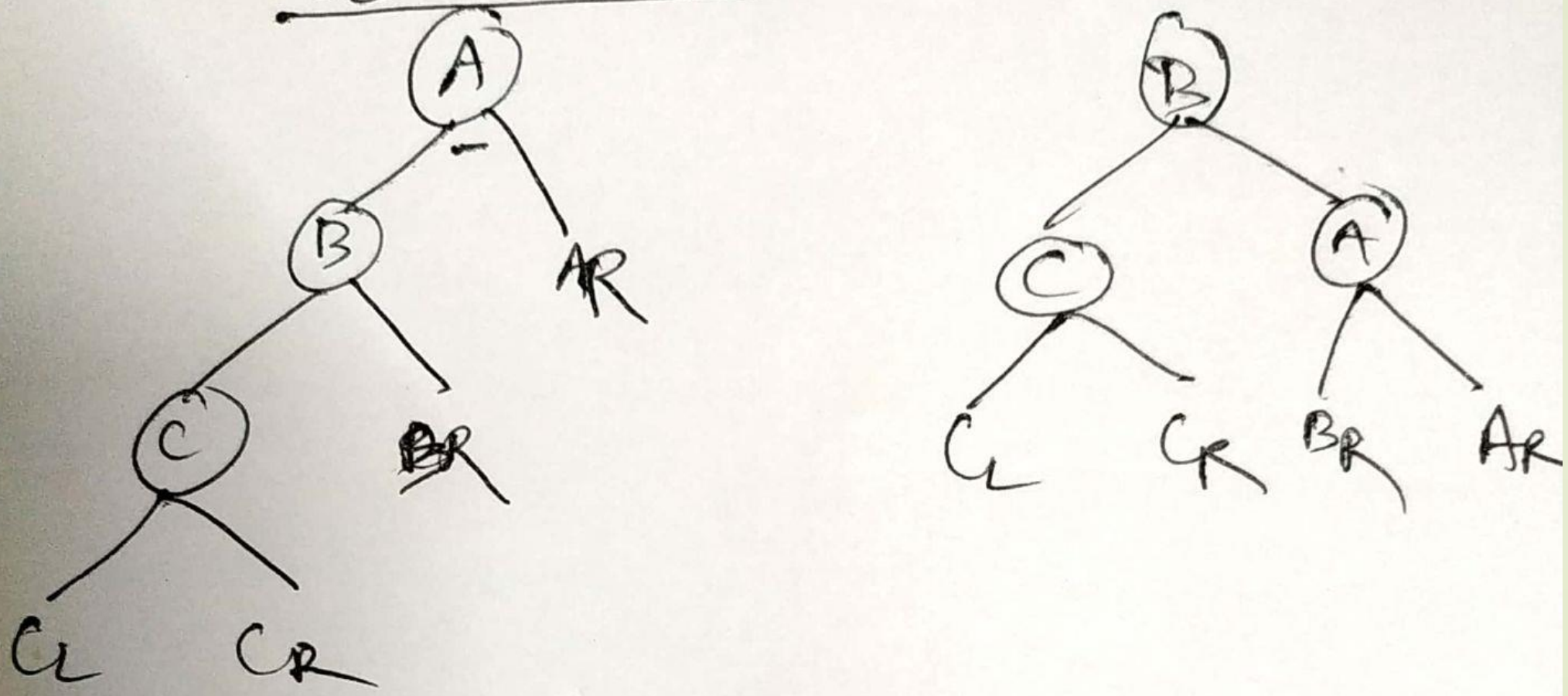


RL imbalance

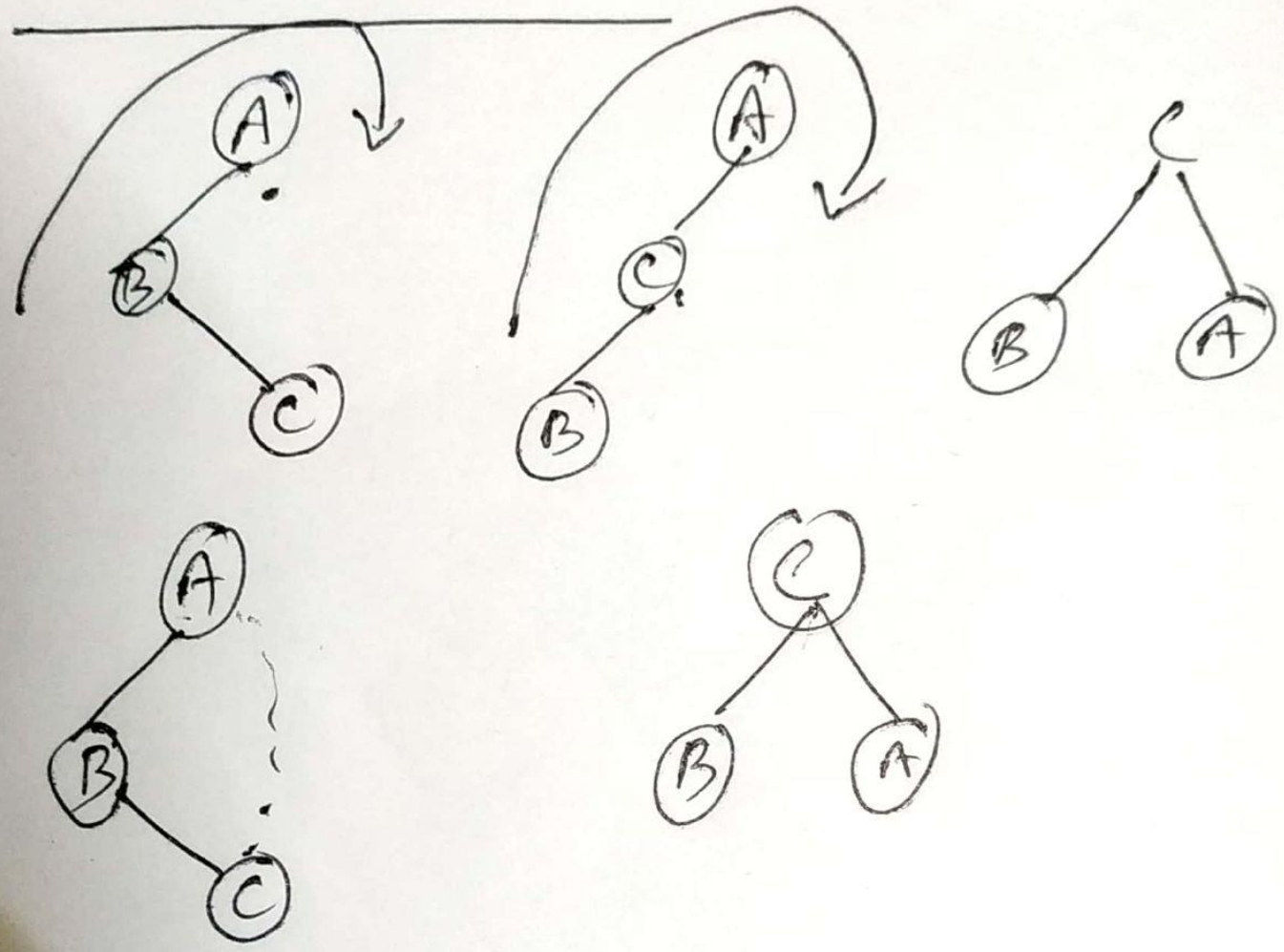


Special cases in these rotations

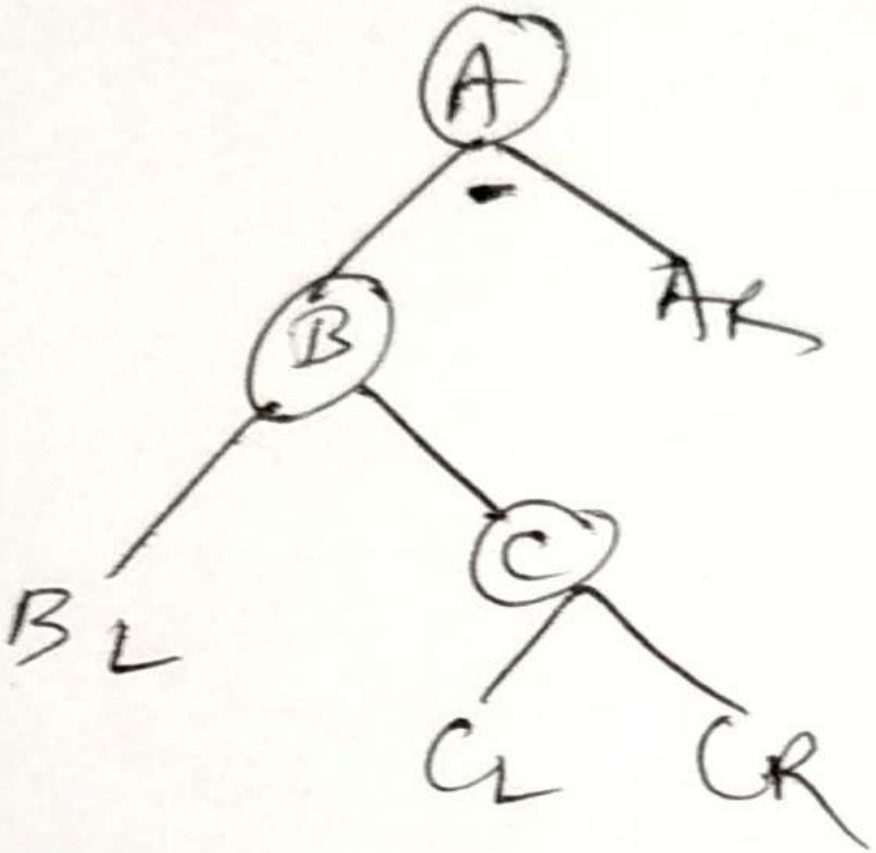
LL-Rotation



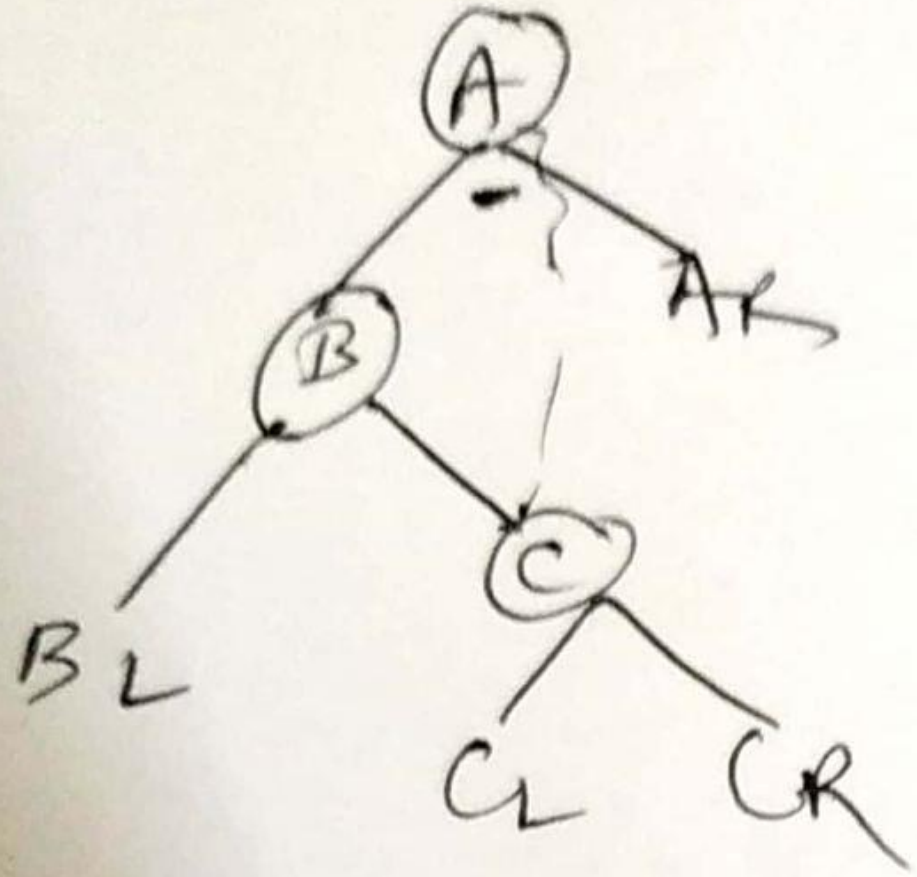
LR - Rotation

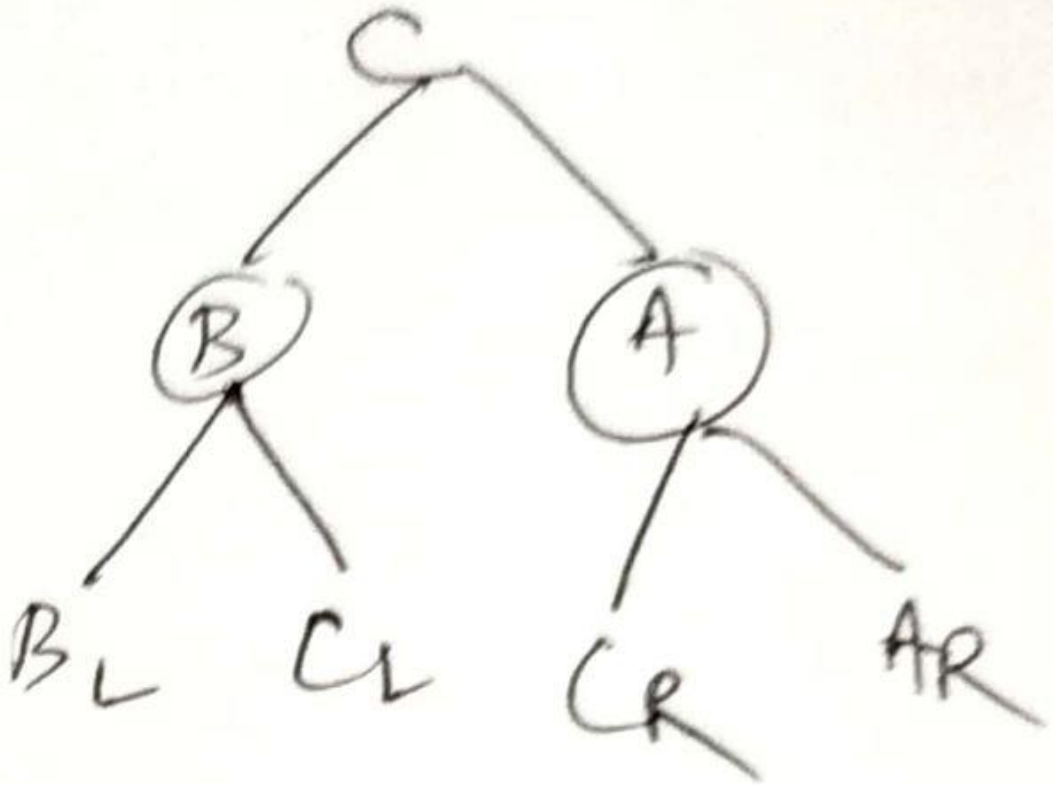


LR Rotation



LR Rotation






Thank you