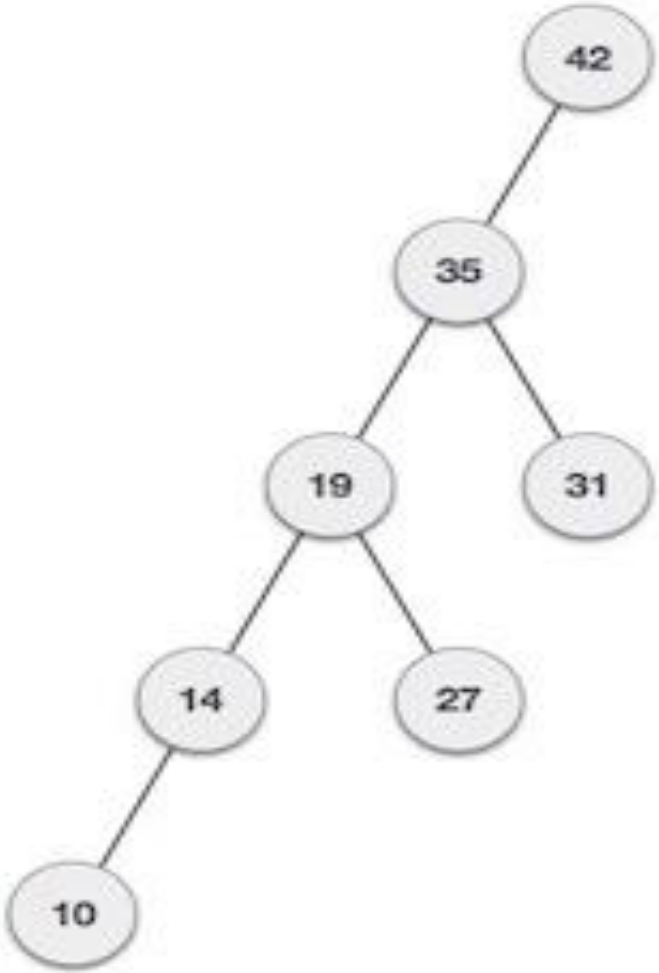
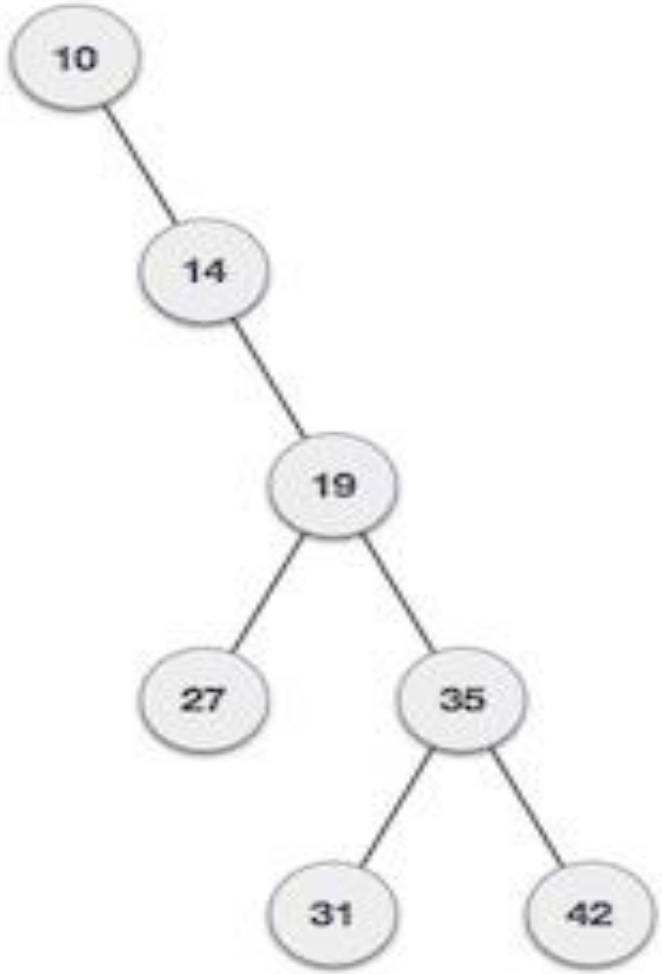
AVL Trees

By
Aditya Tiwari
Assistant Professor
CSVTU Bhilai

- What if the input to binary search tree comes in a sorted (ascending or descending) manner? It will then look like this



If input 'appears' non-increasing manner



If input 'appears' in non-decreasing manner

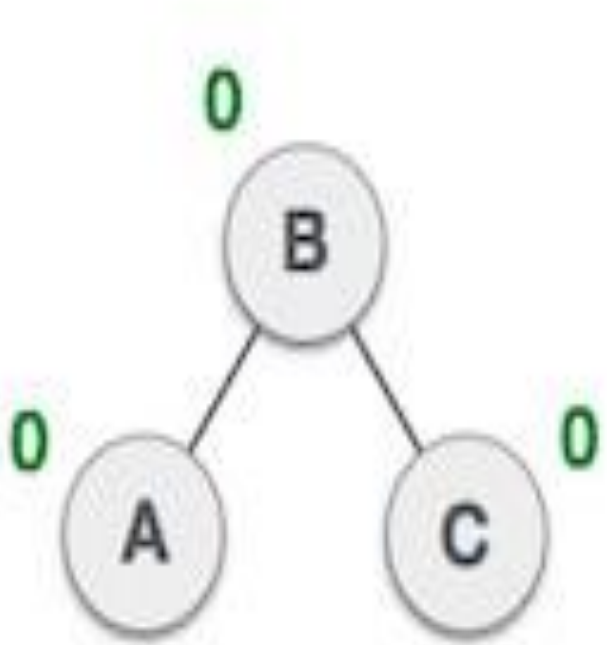
- It is observed that BST's worst-case performance is closest to linear search algorithms, that is O(n). In real-time data, we cannot predict data pattern and their frequencies. So, a need arises to balance out the existing BST.

- Named after their inventor **Adelson**, **Velski** & **Landis**, **AVL trees** are height balancing binary search tree. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the **Balance Factor**.

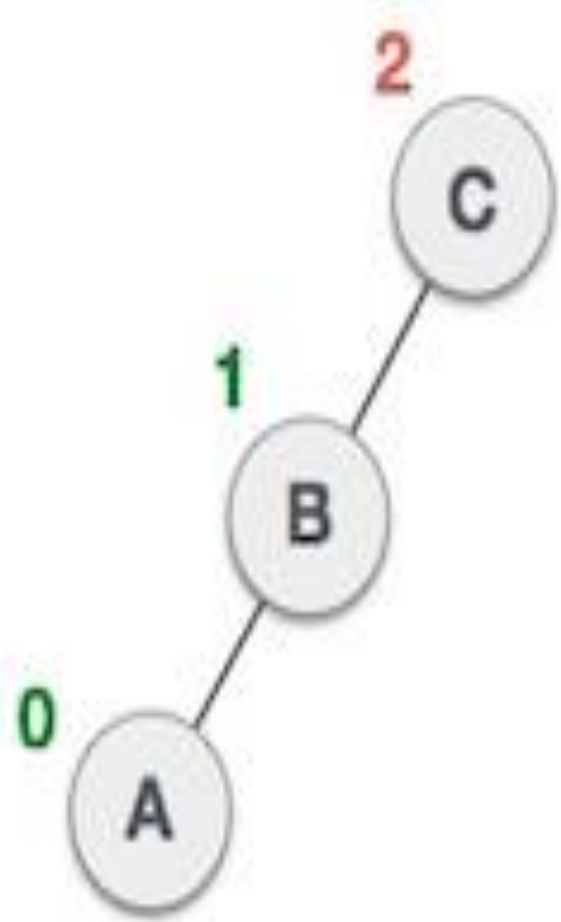
DEFINATION

- **AVL tree** is a height-balanced binary search tree. That means, an AVL tree is also a binary search tree but it is a balanced tree. A binary tree is said to be balanced if, the difference between the heights of left and right subtrees of every node in the tree is either -1, 0 or +1. In other words, a binary tree is said to be balanced if the height of left and right children of every node differ by either -1, 0 or +1. In an AVL tree, every node maintains an extra information known as **balance factor**.

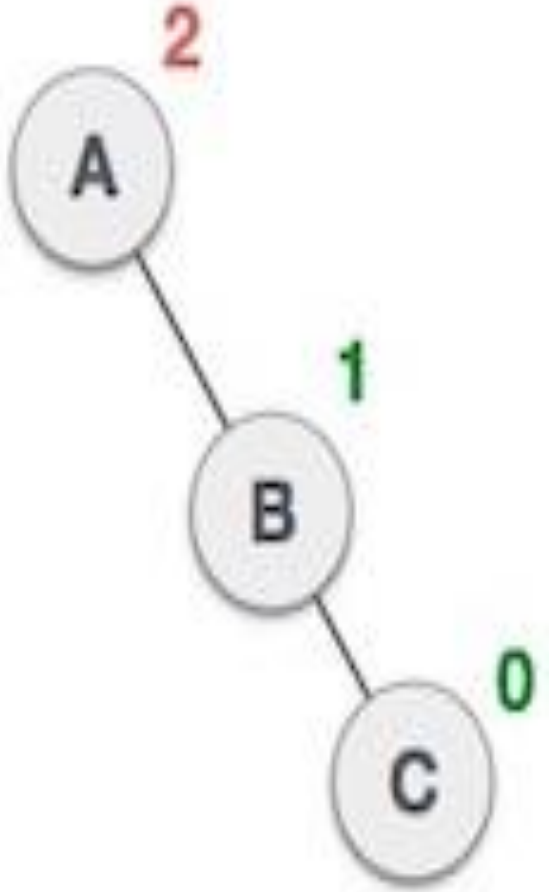
- Here we see that the first tree is balanced and the next two trees are not balanced −



Balanced



Not balanced



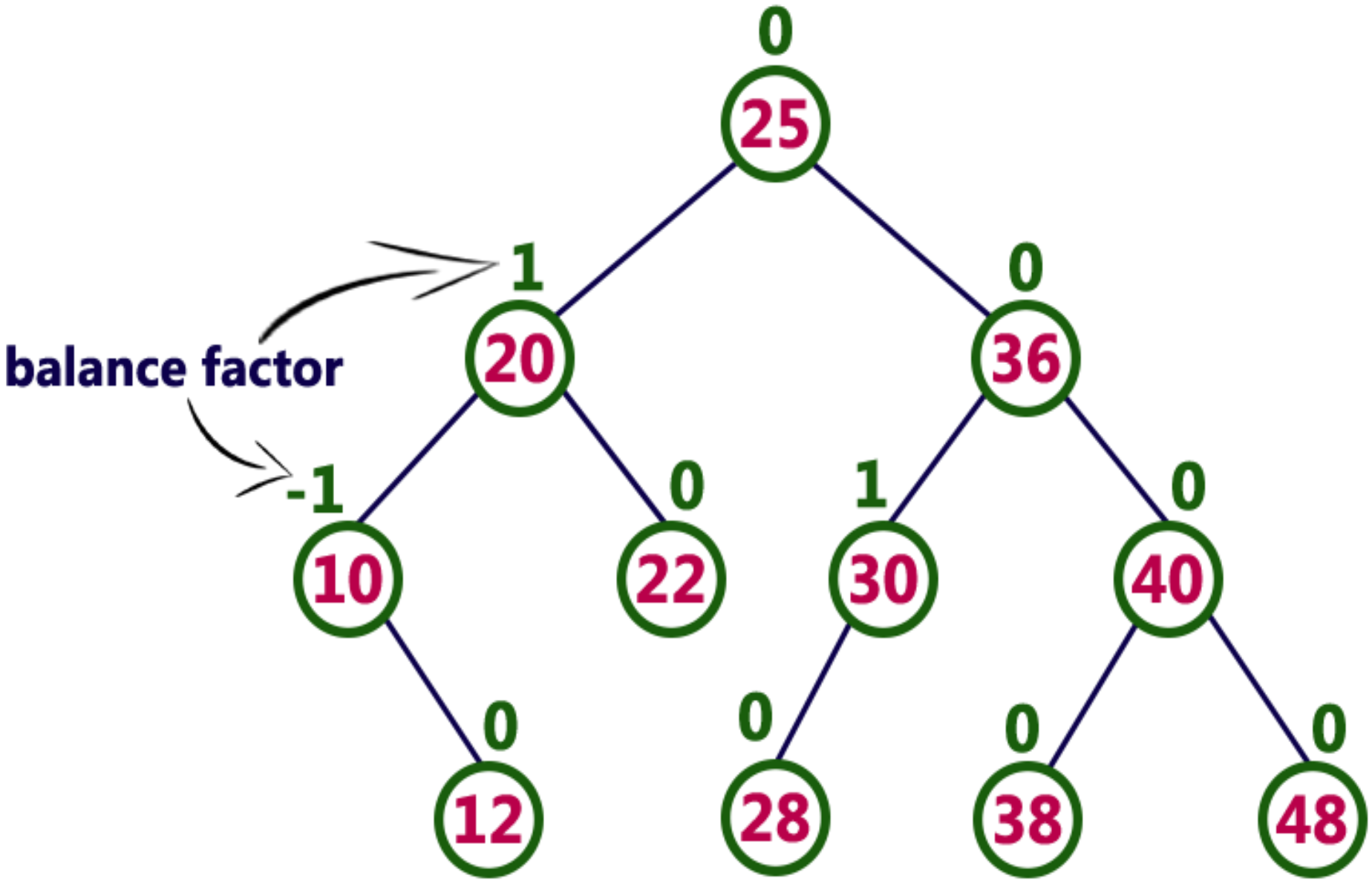
Not balanced

- In the second tree, the left sub tree of **C** has height 2 and the right sub tree has height 0, so the difference is 2. In the third tree, the right sub tree of **A** has height 2 and the left is missing, so it is 0, and the difference is 2 again. AVL tree permits difference (balance factor) to be only 1.

- Balance factor of a node is the difference between the heights of the left and right subtrees of that node. The balance factor of a node is calculated either **height of left subtree - height of right subtree** (OR) **height of right subtree - height of left subtree**. In the following explanation, we calculate as follows.

- ***Balance Factor*** = height(left-sub tree) − height(right-sub tree)
- of its two child sub-trees. A binary tree is defined to be an *AVL tree* if the invariant
- Balancing factor (N) = (-1,0,1)
- holds for every node N in the tree.
- A node N with Balancing factor (N)< 0 is called "left-heavy", one with Balancing factor (N)< 0 is called "right-heavy", and one with Balancing factor (N)= 0 is sometimes simply called "balanced".
- If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using some rotation techniques.

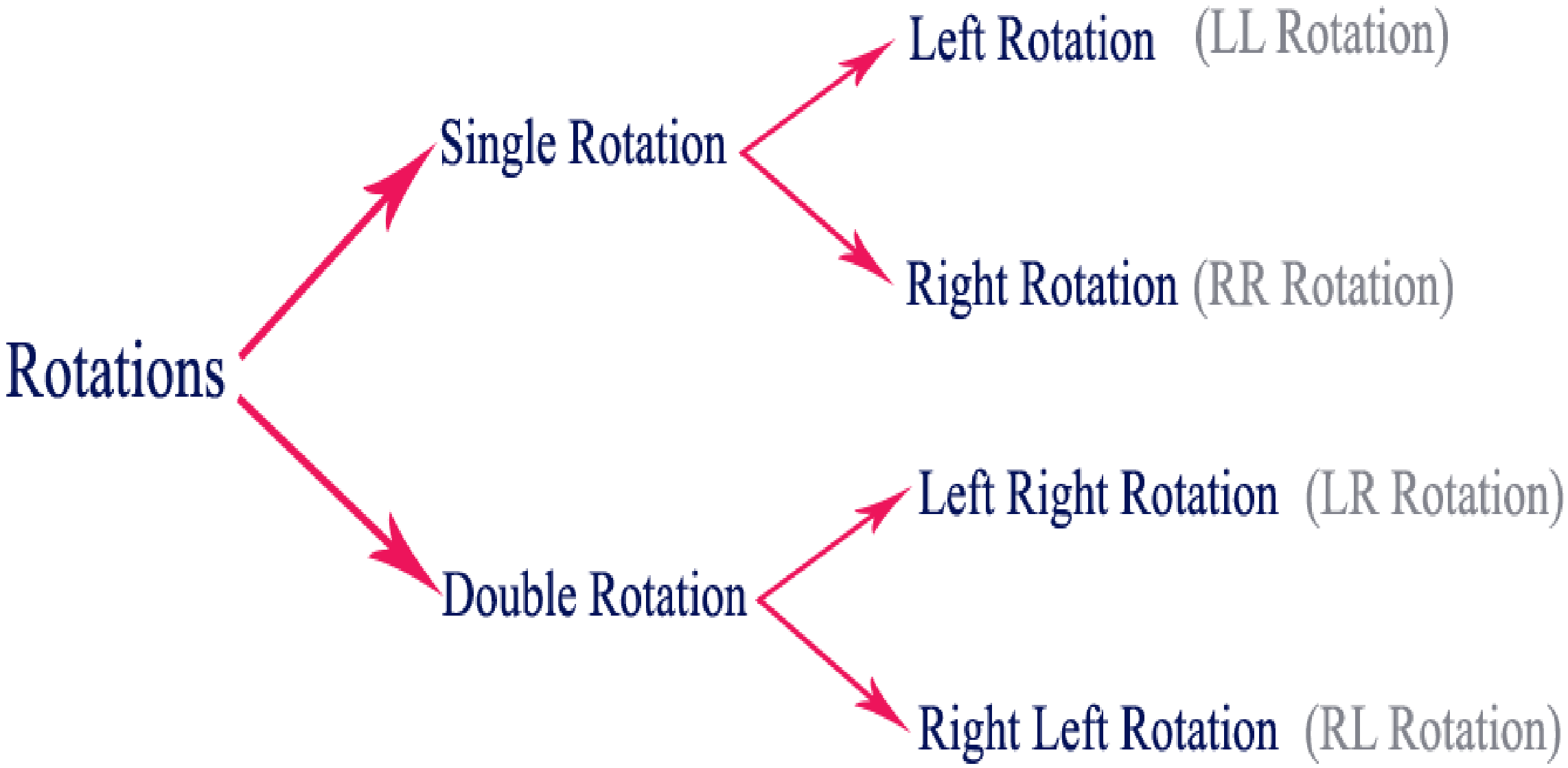
Example of AVL Tree



- The tree is a binary search tree and every node is satisfying balance factor condition. So this tree is said to be an AVL tree.

AVL Rotations

- In **AVL tree**, after performing operations like insertion and deletion we need to check the **balance factor** of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. Whenever the tree becomes imbalanced due to any operation we use **rotation** operations to make the tree balanced.


Rotations
Single Rotation
Left Rotation (LL Rotation)
Right Rotation (RR Rotation)
Double Rotation
Left Right Rotation (LR Rotation)
Right Left Rotation (RL Rotation)

- To balance itself, an AVL tree may perform the following four kinds of rotations −

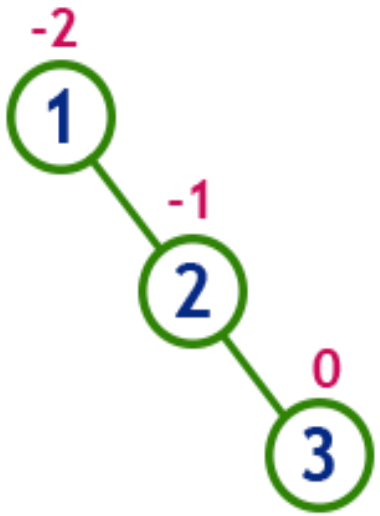
- Left rotation
- Right rotation
- Left-Right rotation
- Right-Left rotation

- The first two rotations are single rotations and the next two rotations are double rotations. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

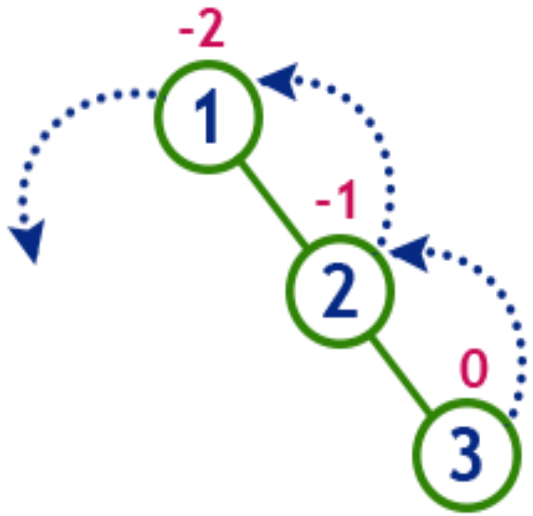
Single Left Rotation (LL Rotation)

- In LL Rotation, every node moves one position to left from the current position. To understand LL Rotation, let us consider the following insertion operation in AVL Tree...

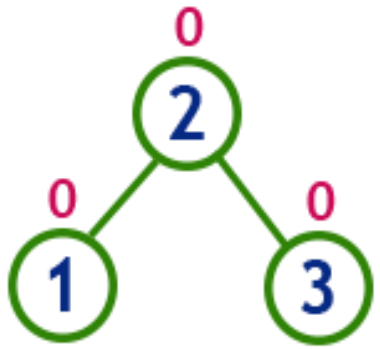
insert 1, 2 and 3



Tree is imbalanced



To make balanced we use LL Rotation which moves nodes one position to left

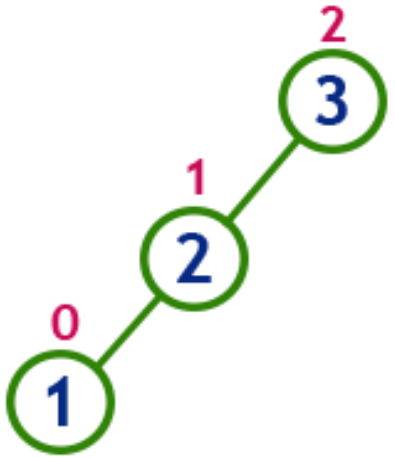


After LL Rotation Tree is Balanced

Single Right Rotation (RR Rotation)

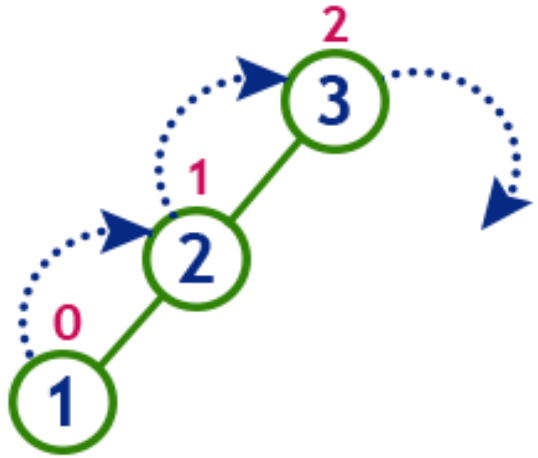
- In RR Rotation, every node moves one position to right from the current position. To understand RR Rotation, let us consider the following insertion operation in AVL Tree...

insert 3, 2 and 1

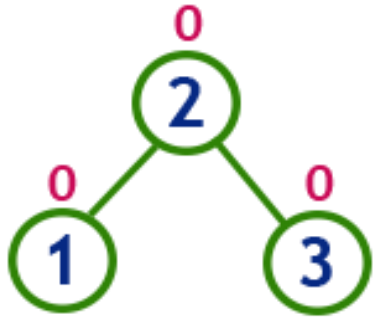


Tree is imbalanced

because node 3 has balance factor 2



To make balanced we use RR Rotation which moves nodes one position to right



After RR Rotation Tree is Balanced

- Thankyou