AVL Tree:

An **AVL Tree** (Adelson-Velsky and Landis Tree) is a **self-balancing binary search tree (BST)** where the height difference between the left and right subtrees of any node (called the **Balance Factor**) is at most **1**.

This balance ensures that operations like **search, insertion, and deletion** remain efficient, with a worst-case time complexity of **O(log n)**.

◆ **Key Properties of AVL Tree**

1) **Binary Search Tree (BST) Rules Apply** → Every node follows the standard **BST property**:

- Left subtree contains values **smaller** than the root.

- Right subtree contains values **greater** than the root.

2) **Balance Factor Condition** → The **balance factor** of every node is calculated as:

Balance Factor=Height of Left Subtree−Height of Right Subtree

- If the balance factor is **-1, 0, or 1** → **Tree is balanced** ✅

- If the balance factor is **outside** this range → **Tree needs balancing** ❌

Examples:

Balanced   Not balanced   Not balanced
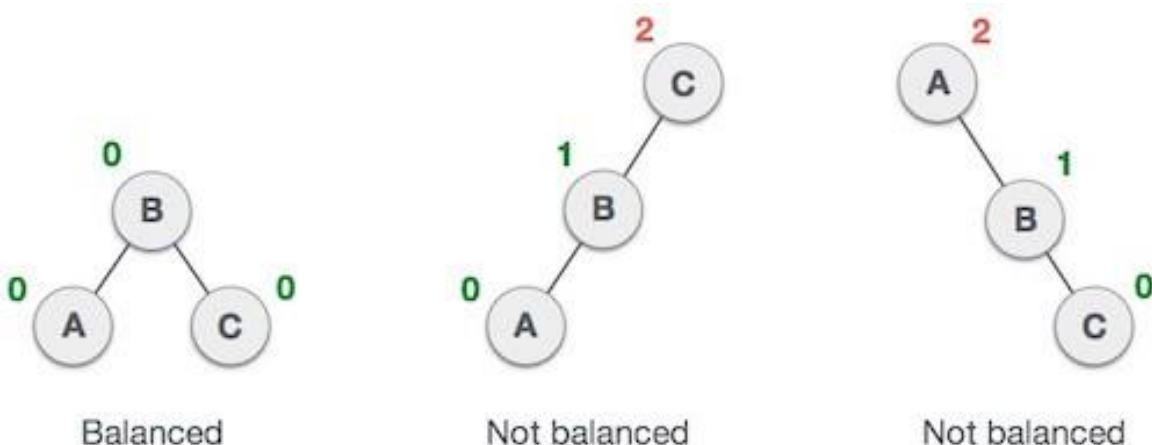
In the second tree, the left subtree of **C** has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of **A** has height 2 and the left is missing, so it is 0, and the difference is 2 again. AVL tree permits difference (balance factor) to be only 1.

> **_BalanceFactor_** = height(left-sutree) – height(right-sutree)

If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using some rotation techniques.

AVL Rotations

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.
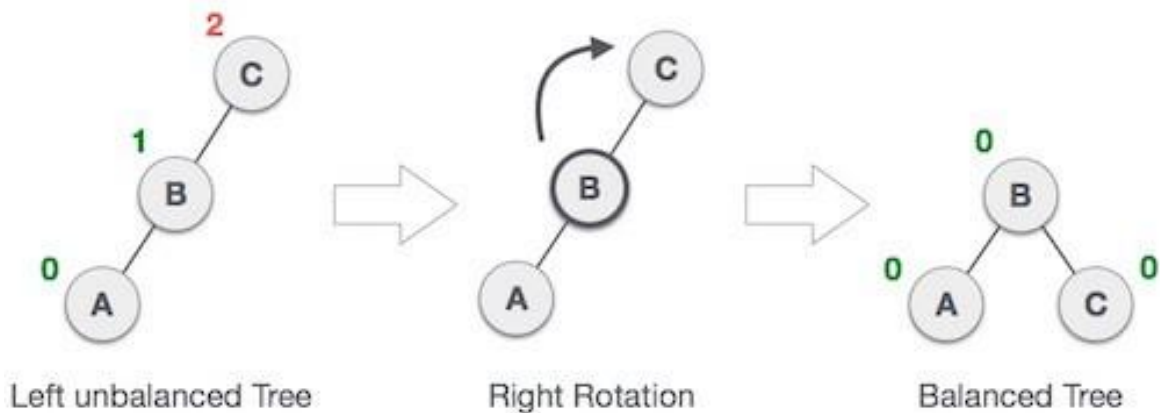
Left Rotation

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation

Right unbalanced tree   Left Rotation   Balanced
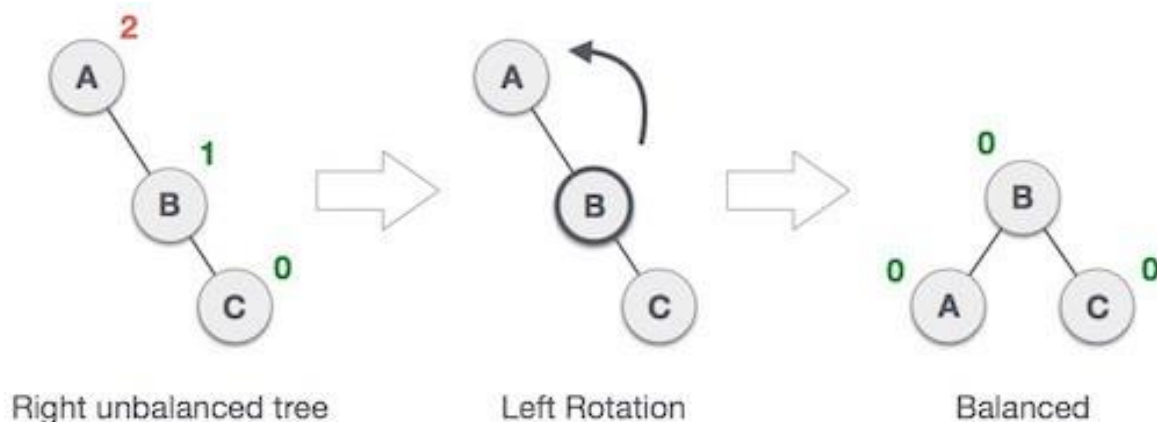
In our example, node **A** has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making **A** the left-subtree of B.
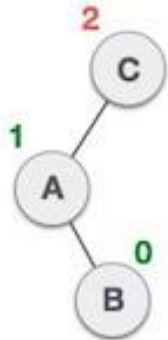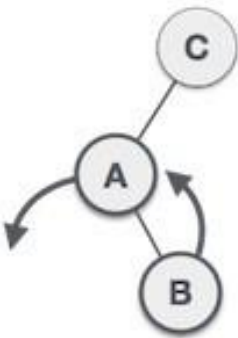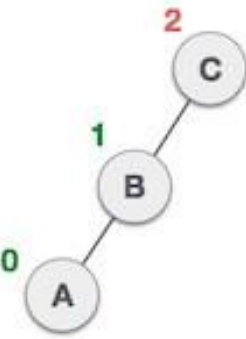
Right Rotation

AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.


As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.
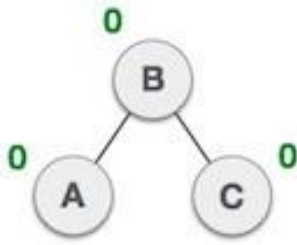
Left-Right Rotation

Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

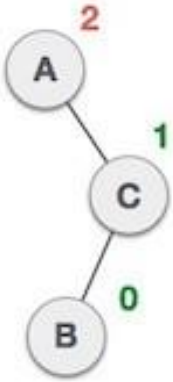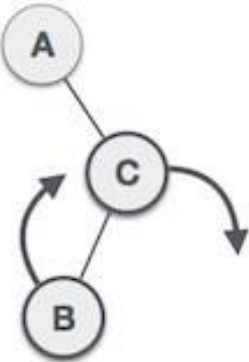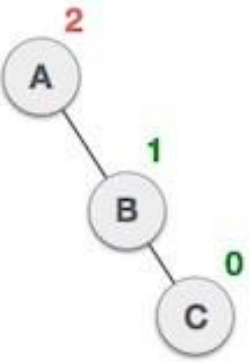| State	Action
	A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**.
	Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree.
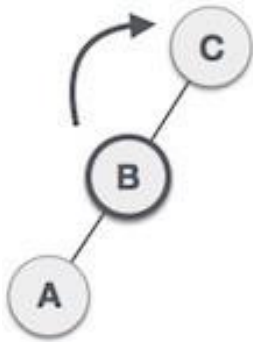
We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree.

The tree is now balanced.

Right-Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

State	Action
	A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2.
	First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**.
	Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation.
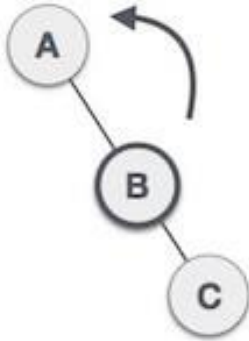
A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**.

H.W:

1) 40   20   10   25   30   22   50
2) 40   45   50   10   5   25
3) 4   6   10   2   1   3   7   11   8   5   9
4) 33   13   11   21   8   9   53   61
5)