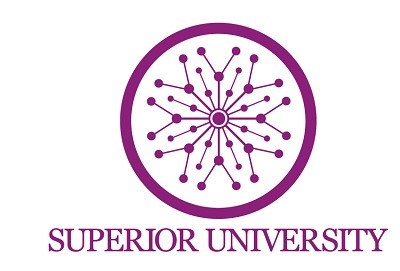
**Lab 12**



Submitted to

:

Sir Rasikh Ali

rR

Submitted by

:

Arooj Fatima

Roll No

:

SU92-BSSEM-S24-091

Subject:

DSA (Lab)

Class:

BS – Software Engineering

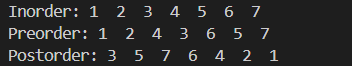
# **Lab # 12**

**BST and AVL**

**A-** **Insert and Traverse for BST**

This code implements a Binary Search Tree (BST) where each **Node** contains data and pointers to left/right children. The **Tree** class uses recursive insertion to maintain BST properties (left child < parent < right child) and provides three traversal methods: **inorder** (left-root-right, producing sorted output), **preorder** (root-left-right), and **postorder** (left-right-root). The recursive **insert** function navigates the tree to find the correct position for new values, while the traversal methods recursively visit each node in different orders. The example builds a BST with values 1-7, demonstrating how the tree automatically organizes them (with 4 as root, 2 as left child, etc.), and shows the different traversal outputs. This implementation efficiently handles dynamic data with O(log n) average-case insertion, though it could degenerate to O(n) if values are inserted in sorted order.

**Output:**



**B-** **Insert and Traverse for AVL**

This code implements a self-balancing AVL Tree, where each **Node** stores a value, height, and left/right child pointers. The tree maintains balance through rotations (single/double) whenever the balance factor (difference between left/right subtree heights) exceeds ±1. Key operations include: **insert** which recursively adds nodes while automatically rebalancing the tree using **rotateLeft**, **rotateRight**, and **balance** operations; **printInorder** for sorted output (left-root-right); and **printLevelOrder** for breadth-first traversal. The example inserts values 1-7 in a non-sequential order (1,2,4,3,6,5,7) and demonstrates how the AVL tree automatically reorganizes itself through rotations to maintain optimal **O(log n)** operation time complexity. The implementation ensures efficient search/insert/delete operations by keeping the tree height-balanced at all times.

**Output:**

