

**Name: Syed Muhammad Fahad Fiaz**

**Roll No: BSSEM-S24-036**

**Section: SE 3A**

**Subject: Data Structure and Algorithms**

**Submitted To: Sir Rasikh Ali**

**Submission Date:**

**Assignment 9**

**DSA LAB TASK’S**

**LAB 11 :**

BST and AVL

1. Insert and Traverse for BST

2. Insert and Traverse for AVL

**Explanation:**

**How Binary Search Trees (BST) Work**

Basic Structure

* Each node contains a value and has up to two children (left and right)
* Left child contains values smaller than parent
* Right child contains values larger than parent

Key Operations

**Insertion:**

* Start at root node
* Compare new value with current node
* Move left if smaller, right if larger
* Repeat until empty spot is found

**Search:**

* Start at root node
* Compare target value with current node
* Move left if smaller, right if larger
* Repeat until value is found or reach null

**Problems:**

* Can become unbalanced with sorted input
* Worst-case performance degrades to O(n)

**How AVL Trees Work**

**Self-Balancing Mechanism**

* Maintains height balance property:
* Heights of left/right subtrees differ by at most 1
* Automatically rebalances after insertions/deletions

Key Operations

**Insertion:**

**Perform standard BST insertion**

* Update heights of ancestor nodes
* Check balance factor (left height - right height)

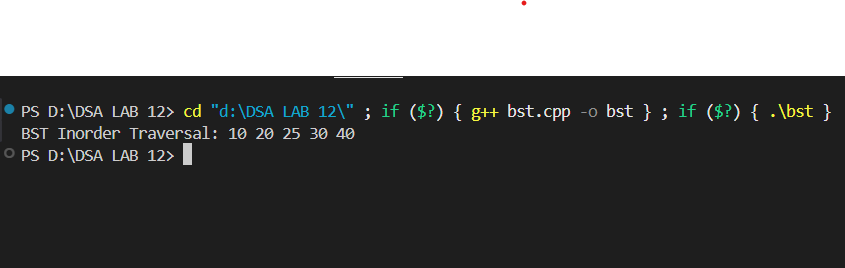
**Perform rotations if unbalanced (4 cases)**

* Left-Left: Single right rotation
* Right-Right: Single left rotation
* Left-Right: Left then right rotation
* Right-Left: Right then left rotation

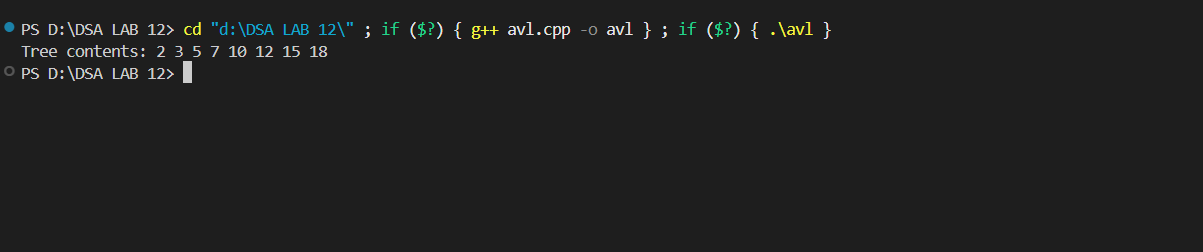
**Advantages**

* Guaranteed O(log n) time for all operations
* Never becomes severely unbalanced
* Efficient for frequent insertions/deletions

**BST OUTPUT**

****

**AVL OUTPUT**

****