Experiment 5:

Student Name: Nitin Singh Rawat   **UID:** 22BCS14491
Branch: BE-CSE   **Section/Group:** KPIT_901/B
Semester: 6   **Date of Performance:** 26/02/25
Subject Name: Advanced Programming Lab-2   **Subject Code:** 22CSP-351

1. **Aim(a):**
   Given the root of a binary tree, return *its maximum depth*.
   A binary tree's maximum depth is the number of nodes along the longest path
   from the root node down to the farthest leaf node.

2. **Objective:** The objective of this program is to determine the maximum depth
   of a given binary tree by finding the longest path from the root node to the
   farthest leaf node.

3. **Algorithm:**
   - If the root is nullptr, return 0 (empty tree has depth 0).
   - Compute the depth of the left and right subtrees using maxDepth(root-
     >left) and maxDepth(root->right).
   - Take the maximum of both subtree depths.
   - Add 1 to account for the current node.
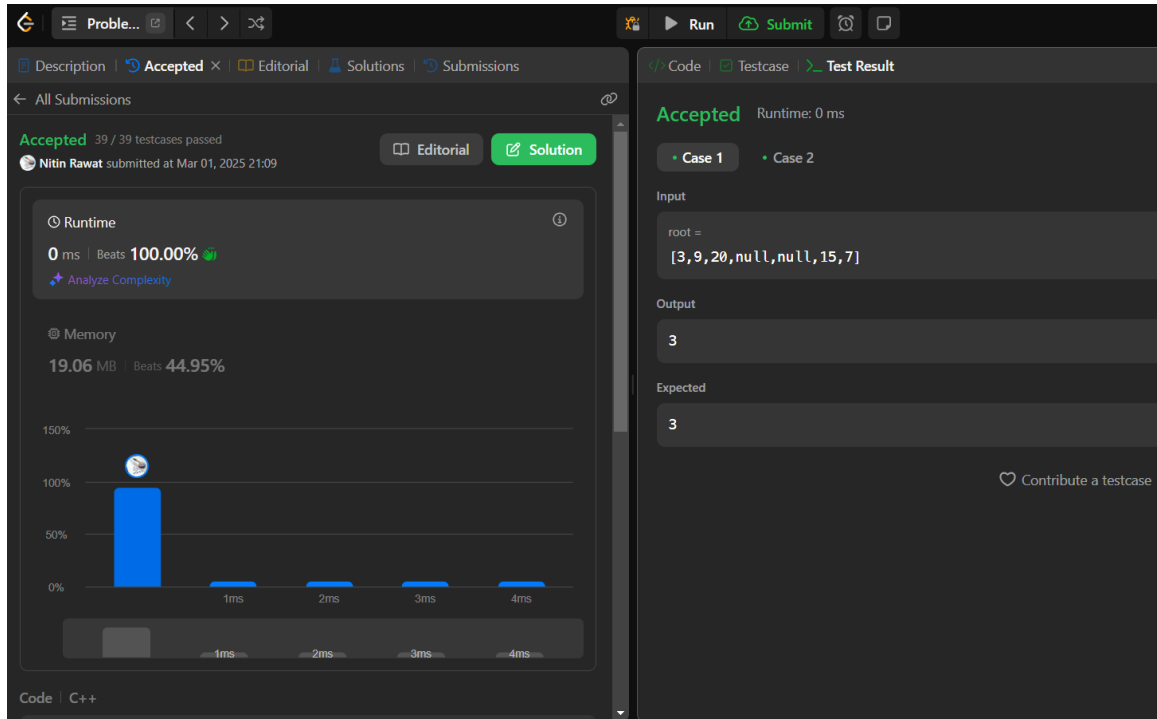   - Return the computed maximum depth.

4. **Code:**
   ```cpp
   class Solution {
   public:
    int maxDepth(TreeNode* root) {
      if (root == nullptr)
        return 0;
      return 1 + max(maxDepth(root->left), maxDepth(root->right));
    }
   };
   ```

Leetcode link:

https://leetcode.com/problems/maximum-depth-of-binary-tree/submissions/1559353479/

5. Output:

6. Time Complexity:

The time complexity is O(n), where n is the number of nodes in the tree.

7. Learning Outcomes:

- Learnt how to calculate the maximum depth of a binary tree using recursion.
- Learnt how to apply the divide and conquer approach in tree traversal.
- Learnt how recursion helps in solving tree-based problems efficiently.
- Learnt the time complexity analysis of O(n) for tree traversal.

PROBLEM-2

1. **Aim(b):** Given the root of a binary tree, *determine if it is a valid binary search tree (BST).* A **valid BST** is defined as follows:
   - The left subtree of a node contains only nodes with keys **less than** the node's key.
   - The right subtree of a node contains only nodes with keys **greater than** the node's key.
   - Both the left and right subtrees must also be binary search trees.

2. **Objective**: The objective of this program is to determine whether a given binary tree is a valid Binary Search Tree (BST) by ensuring that all nodes satisfy the BST properties, where the left subtree contains smaller values, the right subtree contains larger values, and both subtrees are valid BSTs.

3. **Algorithm:**
   - Initialize Pointers: Use a prev pointer to track the previous node in an inorder traversal.
   - Perform Inorder Traversal: Recursively traverse the left subtree first.
   - Check BST Condition: If the prev node exists and its value is greater than or equal to the current node's value, return false.
   - Update prev: Set prev = node and continue traversal to the right subtree.
   - Return Result: If all nodes satisfy BST conditions, return true; otherwise, return false.
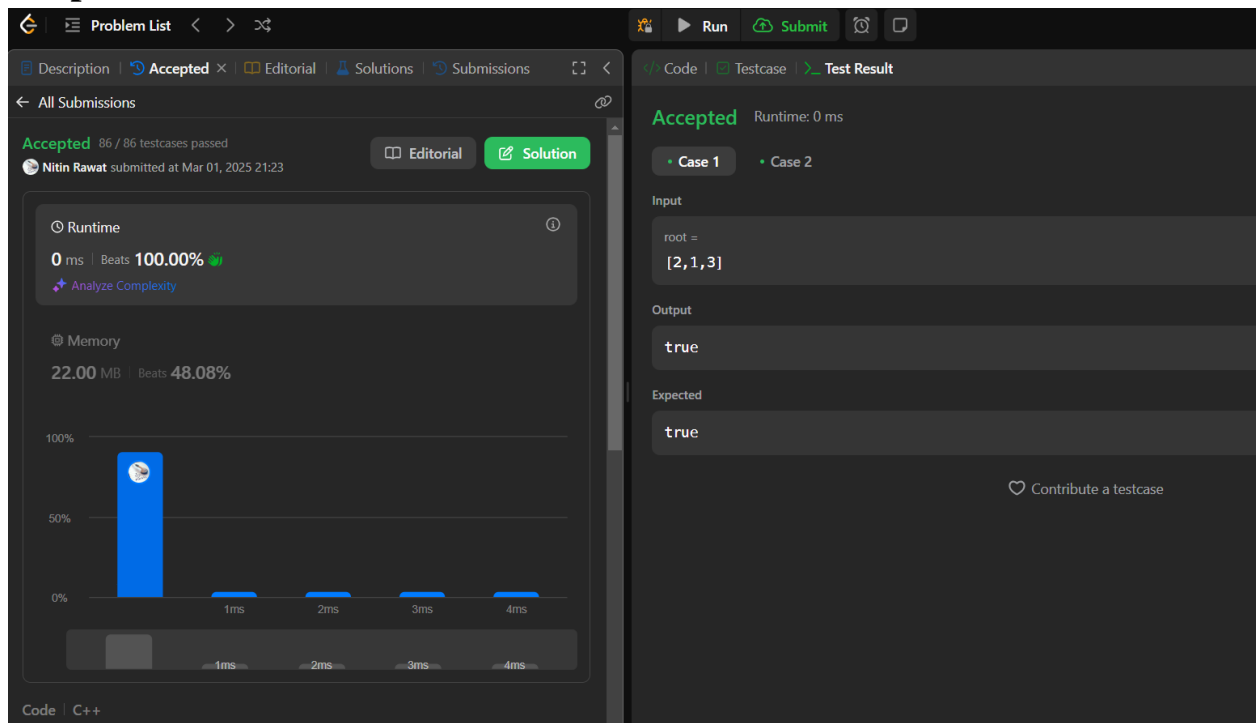
4. **Code:**

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        TreeNode* prev = NULL;
        return validate(root,prev);
```

```cpp
    }
    bool validate(TreeNode* node, TreeNode* &prev)
    {
        if (node==NULL) return true;
        if (!validate(node->left,prev)) return false;
        if(prev!=NULL && prev->val >= node->val) return false;
        prev=node;
        return validate(node->right,prev);
    }
};
```

LeetCode Link:

https://leetcode.com/problems/validate-binary-search-tree/submissions/1559368278/

5. Output:

6. **Time Complexity:**

The time complexity is O(n), where n is the number of nodes in the tree.

7. **Learning outcomes:**
   - Learnt how to validate a Binary Search Tree (BST) using inorder traversal.
   - Learnt the importance of maintaining a prev pointer to track the last visited node in inorder traversal.
   - Learnt how to implement recursive tree traversal efficiently.
   - Learnt how inorder traversal of a BST produces a sorted sequence, which helps in validation.