Experiment 6

Student Name: Avil saini | **UID:** 22BET101380
Branch: IT | **Section/Group:** 22BET_IOT-702/B
Semester: 6th | **Date of Performance:** 28/02/25
Subject Name: Advance Programming-II | **Subject Code:** 22ITP-367

Problem: 1.6.1: Maximum Depth of Binary Tree

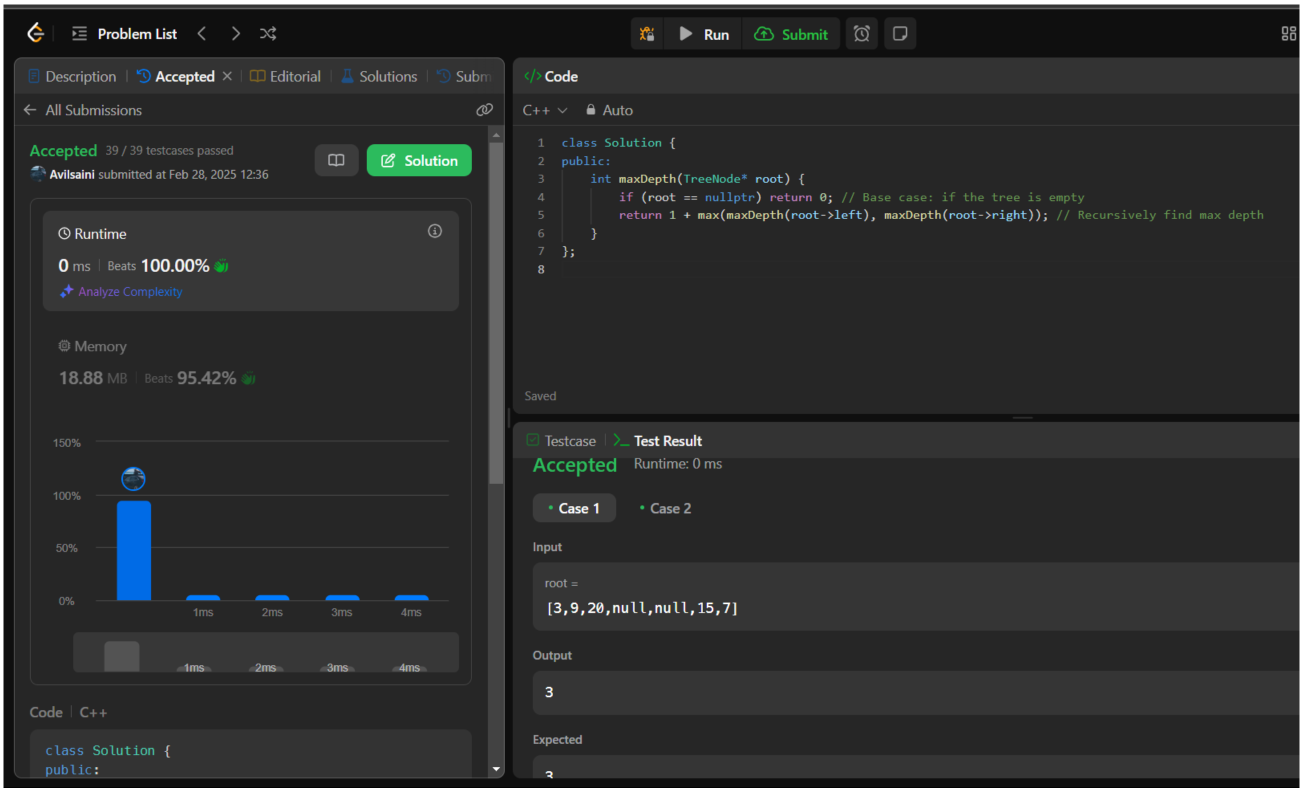
Problem Statement: Given the root of a **binary tree**, return its **maximum depth**. The **maximum depth** is the number of nodes along the **longest path** from the root down to the farthest leaf node.

1. **Objective:** Find the **maximum depth** of a binary tree using **recursion**.

2. **Code:**

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (root == nullptr) return 0; // Base case: if the tree is empty
        return 1 + max(maxDepth(root->left), maxDepth(root->right)); // Recursively find max depth
    }
};
```

3.Result


Problem List
Run
Submit
Description
Accepted
Editorial
Solutions
Code
All Submissions
C++
Auto
Accepted 39 / 39 testcases passed
Avilsaini submitted at Feb 28, 2025 12:36
Solution
class Solution {
public:
int maxDepth(TreeNode* root) {
if (root == nullptr) return 0; // Base case: if the tree is empty
return 1 + max(maxDepth(root->left), maxDepth(root->right)); // Recursively find max depth
}
};
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
18.88 MB | Beats 95.42%
150%
100%
50%
0%
1ms
2ms
3ms
4ms
Saved
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[3,9,20,null,null,15,7]
Output
3
Expected
Code | C++
class Solution {
public:

:

Problem 1.6.2: Validate Binary Search Tree

Problem Statement: Given the root of a binary tree, determine if it is a valid Binary Search Tree (BST).

A BST must satisfy the following properties:

The left subtree of a node contains only nodes with values less than the node's value.

The right subtree of a node contains only nodes with values greater than the node's value.

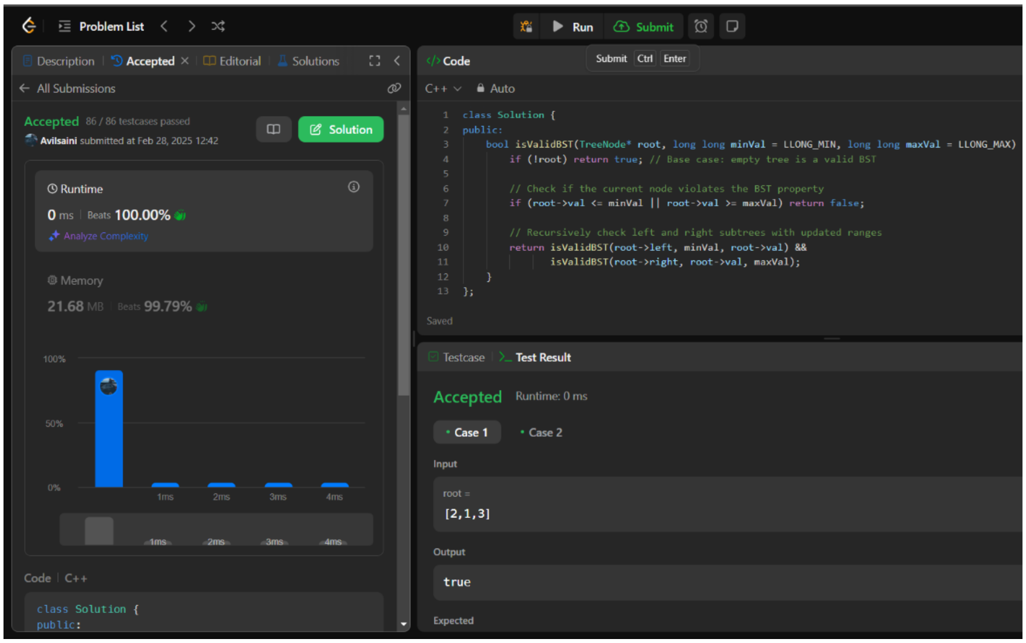
Both the left and right subtrees must also be BSTs.

1. **Objective:** Check if the given tree is a **valid BST** using **recursion and range validation**.

2. **Code:**

```
class Solution {
public:
   bool isValidBST(TreeNode* root, long long minVal = LLONG_MIN, long long maxVal = LLONG_MAX) {
        if (!root) return true; // Base case: empty tree is a valid BST

        // Check if the current node violates the BST property
        if (root->val <= minVal || root->val >= maxVal) return false;

        // Recursively check left and right subtrees with updated ranges
        return isValidBST(root->left, minVal, root->val) &&
               isValidBST(root->right, root->val, maxVal);
   }
};
```

Result:



Problem 1.6.3: Kth Largest Element in an Array

Problem Statement: Given an integer array nums and an integer k, return the kth largest element in the array.

Note: It is the kth largest element in sorted order, not the kth distinct element.

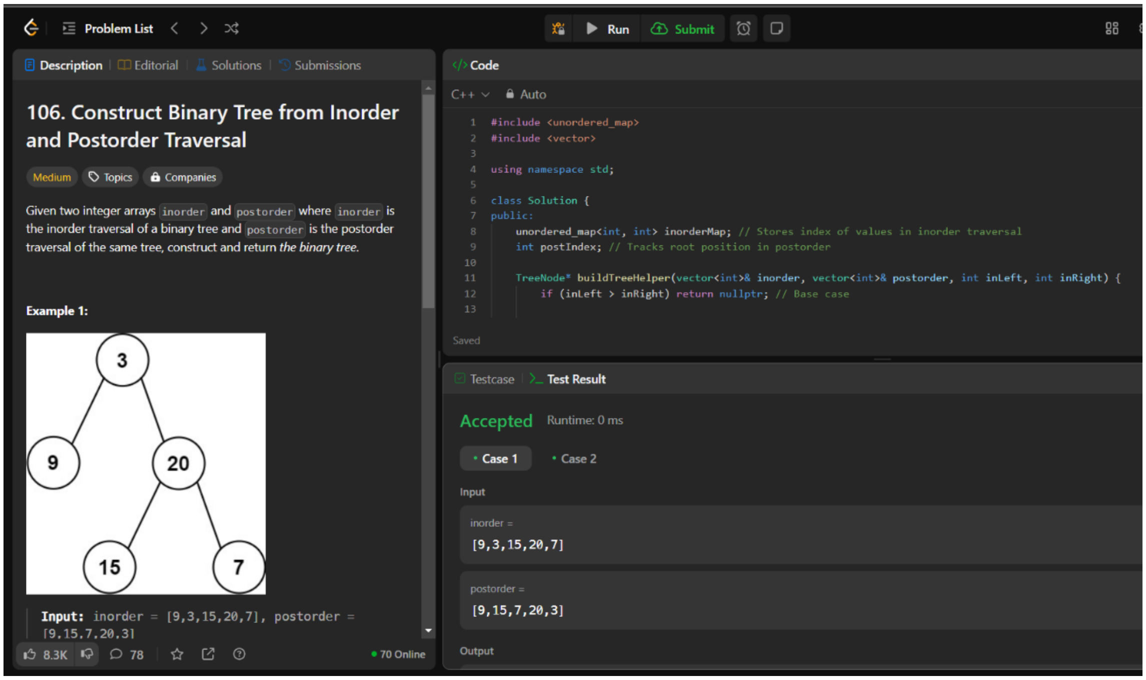
1. **Objective:** Find the **kth largest element** in the array using an efficient approach.

2. **Code:**

```
#include <unordered_map>
#include <vector>

using namespace std;

class Solution {
public:
   unordered_map<int, int> inorderMap; // Stores index of values in
inorder traversal
   int postIndex; // Tracks root position in postorder
```

```
    TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>&
postorder, int inLeft, int inRight) {
        if (inLeft > inRight) return nullptr; // Base case

        int rootVal = postorder[postIndex--]; // Get root from postorder
        TreeNode* root = new TreeNode(rootVal); // Create root node

        int inIndex = inorderMap[rootVal]; // Get index from inorder map

        // Build right subtree first (since postorder gives root -> right -> left)
        root->right = buildTreeHelper(inorder, postorder, inIndex + 1,
inRight);
        root->left = buildTreeHelper(inorder, postorder, inLeft, inIndex - 1);

        return root;
    }

    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        postIndex = postorder.size() - 1; // Start from the last element in
postorder

        // Store inorder indices in a map for O(1) lookup
        for (int i = 0; i < inorder.size(); i++) {
            inorderMap[inorder[i]] = i;
        }

        return buildTreeHelper(inorder, postorder, 0, inorder.size() - 1);
    }
};
```

3. Result:


Problem List
Run
Submit
Description
Editorial
Solutions
Submissions
106. Construct Binary Tree from Inorder and Postorder Traversal
Medium
Topics
Companies
Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree.
Example 1:
3
9
20
15
7
Input: inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]
8.3K
78
70 Online
Code
C++
Auto
#include <unordered_map>
#include <vector>
using namespace std;
class Solution {
public:
unordered_map<int, int> inorderMap; // Stores index of values in inorder traversal
int postIndex; // Tracks root position in postorder
TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int inLeft, int inRight) {
if (inLeft > inRight) return nullptr; // Base case
Saved
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
inorder =
[9,3,15,20,7]
postorder =
[9,15,7,20,3]
Output

Problem 1.6.4: Binary Tree Inorder Traversal

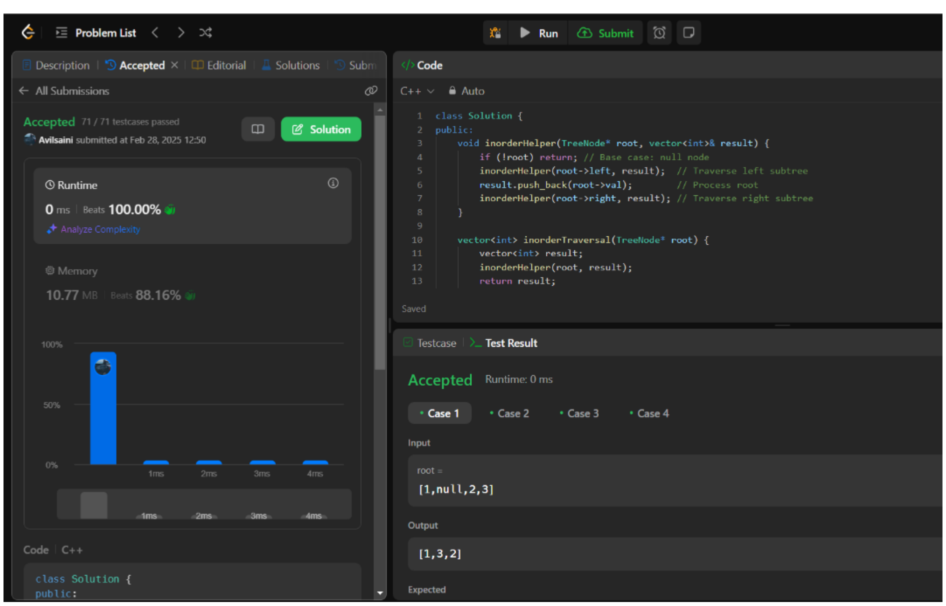
Problem Statement: Given the root of a **binary tree**, return its **inorder traversal** (**Left → Root → Right**).

Objective: Traverse a binary tree in inorder sequence (Left → Root → Right) and return the values in a list.

1. Code:

```
class Solution {
public:
    void inorderHelper(TreeNode* root, vector<int>& result) {
        if (!root) return; // Base case: null node
        inorderHelper(root->left, result);  // Traverse left subtree
        result.push_back(root->val);        // Process root
        inorderHelper(root->right, result); // Traverse right subtree
    }

    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorderHelper(root, result);
        return result;
    }
};
```

2. Result:



Problem 1.6.5: Populating Next Right Pointers in Each Node

1. **Problem Statement:** Given a perfect binary tree, connect each node's next pointer to its right adjacent node. If there is no right adjacent node, set the next pointer to NULL. The connections should be made in-place, without using extra memory for level-wise traversal.

2. **Objective:** Find the median of two sorted arrays **efficiently** using **binary search**.

3. **Code:**

```
#include <queue>
using namespace std;

class Solution {
public:
    Node* connect(Node* root) {
        if (!root) return nullptr; // Base case

        queue<Node*> q;
        q.push(root);

        while (!q.empty()) {
            int size = q.size();
            Node* prev = nullptr;

            for (int i = 0; i < size; i++) {
                Node* curr = q.front();
                q.pop();

                if (prev) prev->next = curr; // Connect previous node to current
                prev = curr;

                if (curr->left) q.push(curr->left);
                if (curr->right) q.push(curr->right);
            }
            prev->next = nullptr; // Last node in the level points to NULL
        }
```

```
        return root;
    }
};
        return 0;
```

4. Result: