WORKSHEET 3

Student Name: Hardik Jhamb **UID:** 22BCS16504

Branch: BE-CSE **Section/Group:** 22BCS_NTPP-602-A

Semester: 6th **Date of Performance:**

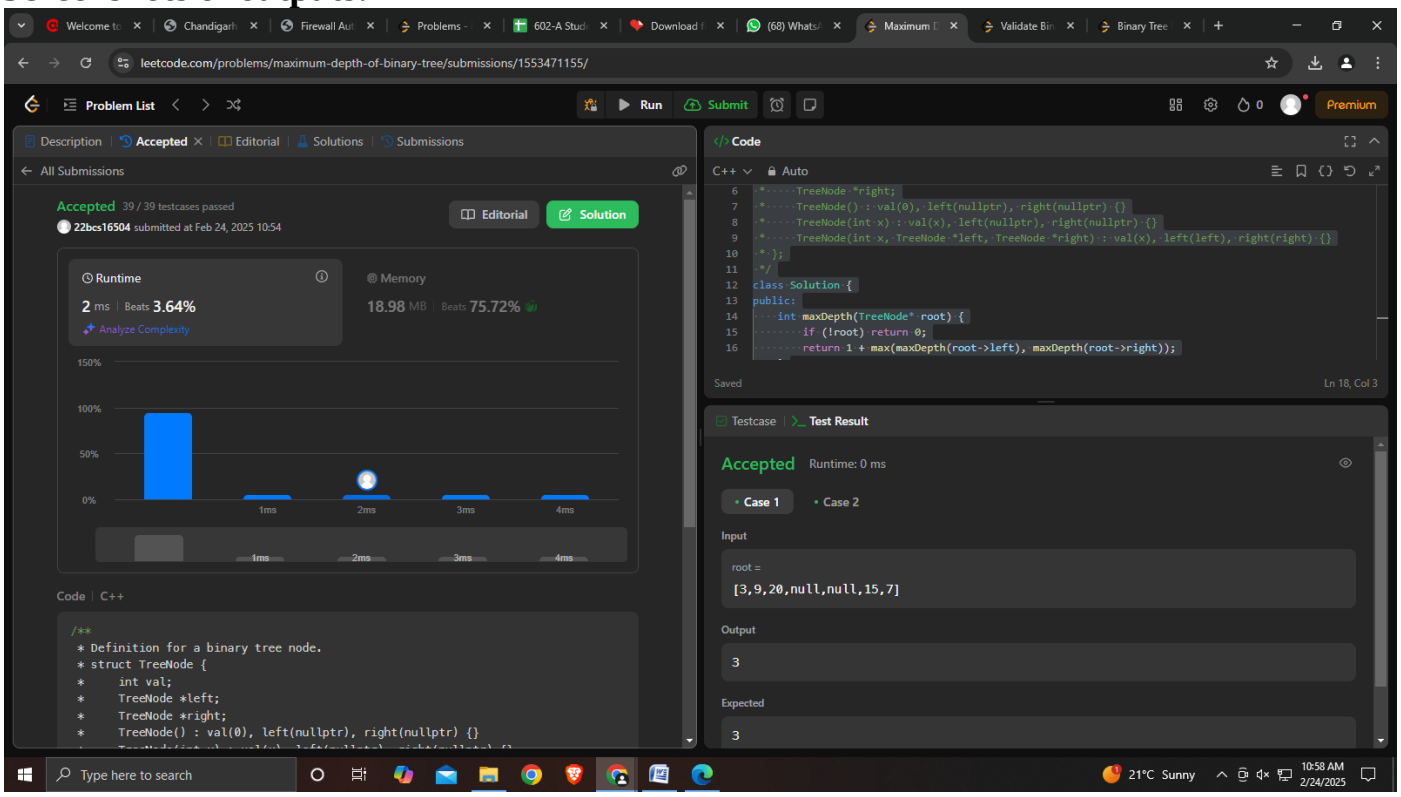
Subject Name: AP LAB - II **Subject Code:** 22CSP-351

1. **Aim:** Longest Nice Substring , Reverse Bits , Number of 1 bits

2. **Source Code:**

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (!root) return 0;
        return 1 + max(maxDepth(root->left), maxDepth(root->right));
```

3. Screenshots of outputs:



2. Aim: 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

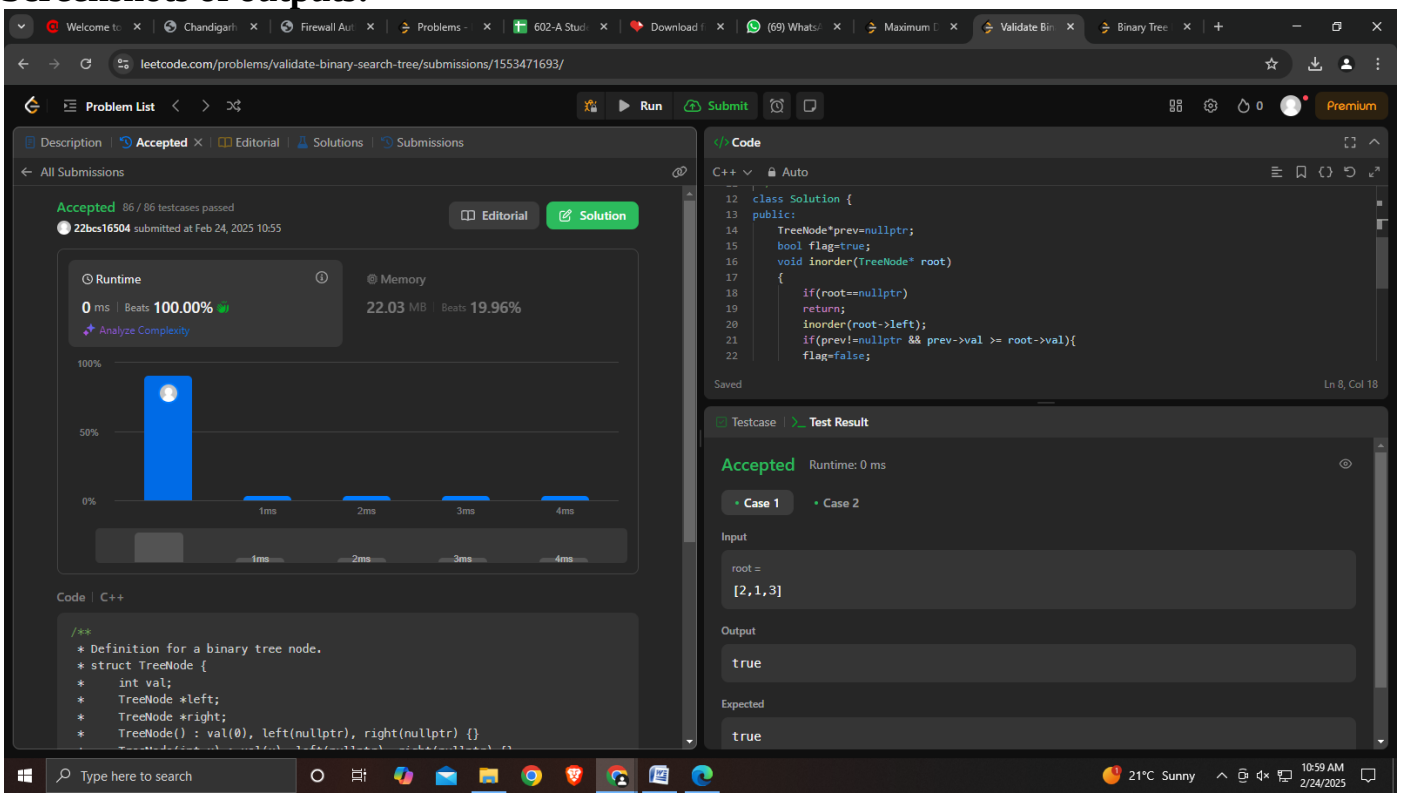
of a node contains only nodes with keys **less than** the node's key.

- 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

Source Code:

```
class Solution {
public:
    TreeNode*prev=nullptr;
    bool flag=true;
    void inorder(TreeNode* root)
    {
        if(root==nullptr)
        return;
        inorder(root->left);
        if(prev!=nullptr && prev->val >= root->val){
        flag=false;
        return;
        }
```

Screenshots of outputs:



Aim: Given the root of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

Source Code:

```
class Solution {
 public:
  vector<vector<int>> levelOrder(TreeNode* root) {
    if (root == nullptr)
      return {};

    vector<vector<int>> ans;
    queue<TreeNode*> q{{root}};

    while (!q.empty()) {
      vector<int> currLevel;
      for (int sz = q.size(); sz > 0; --sz) {
        TreeNode* node = q.front();
        q.pop();
```

4. Screenshots of outputs: