

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

WORKSHEET 5

Student Name: Paras Bhatt

UID: 22BCS15683

Branch: BE-CSE

Section/Group: 22BCS_NTPP-602-A

Semester: 6th

Date of Performance: 17/02/2025

Subject Name: AP LAB - II

Subject Code: 22CSP-351

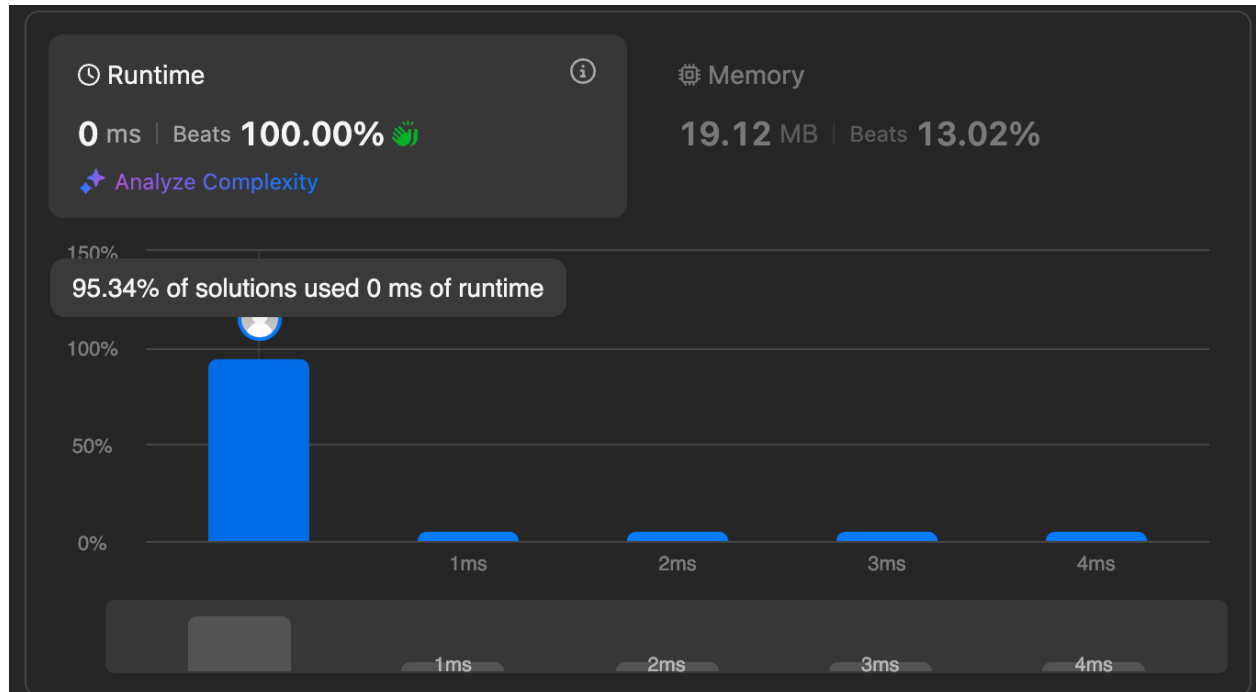
1. Aim: Given the root of a binary tree, return *its maximum depth*.

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. Source Code:

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

3. Screenshots of outputs:



2.

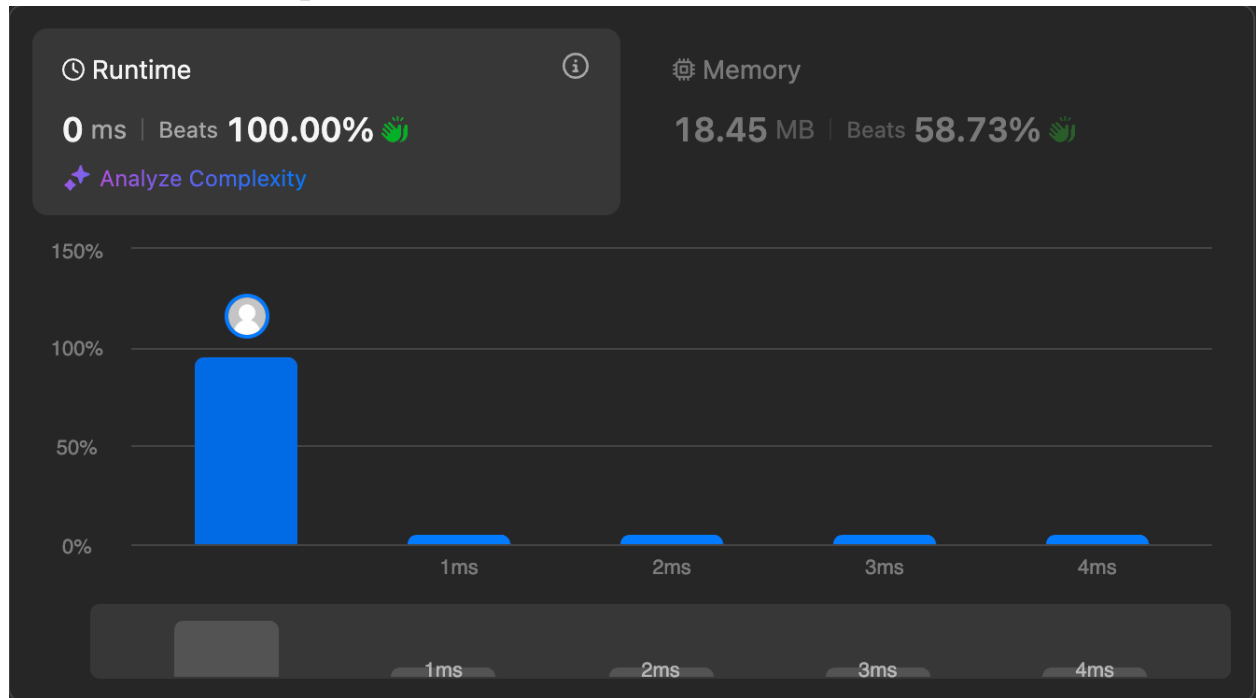
Aim: Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Source Code:

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if (root == nullptr) {
            return true;
        }
        return isMirror(root->left, root->right);
    }
private:
    bool isMirror(TreeNode* left, TreeNode* right) {
        if (left == nullptr && right == nullptr) {
            return true;
        }
        if (left == nullptr || right == nullptr || left->val != right->val) {
            return false;
        }
    }
}
```

```
        return isMirror(left->left, right->right) && isMirror(left->right,
right->left);
    }
};
```

Screenshots of outputs:



3.

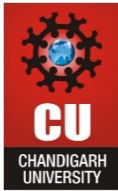
Aim: Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Source Code:

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        unordered_map<int, int> inorderMap;
        for (int i = 0; i < inorder.size(); ++i) {
            inorderMap[inorder[i]] = i;
        }
        return buildTreeHelper(preorder, 0, preorder.size() - 1, inorder, 0,
inorder.size() - 1, inorderMap);
    }
};
```

```
    }  
private:  
    TreeNode* buildTreeHelper(vector<int>& preorder, int preStart, int  
preEnd,  
        vector<int>& inorder, int inStart, int inEnd,  
        unordered_map<int, int>& inorderMap) {  
        if (preStart > preEnd || inStart > inEnd) {  
            return nullptr;  
        }  
        int rootVal = preorder[preStart];  
        TreeNode* root = new TreeNode(rootVal);  
        int rootIndexInorder = inorderMap[rootVal];  
        int leftSubtreeSize = rootIndexInorder - inStart;  
        root->left = buildTreeHelper(preorder, preStart + 1, preStart +  
leftSubtreeSize,  
            inorder, inStart, rootIndexInorder - 1, inorderMap);  
        root->right = buildTreeHelper(preorder, preStart + leftSubtreeSize +  
1, preEnd,  
            inorder, rootIndexInorder + 1, inEnd, inorderMap);  
        return root;  
    }  
};
```

4. Screenshots of outputs:



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

