



## Experiment - 6

**Name:** Ranveer Singh

**Branch:** IT

**Semester:** 6<sup>th</sup>

**Subject:** AP LAB - 2

**UID:** 22BET10030

**Section:** IOT 703/A

**Date of Performance:** 25/02/25

**Subject Code:** 22ITP-351

### 1. Aim:

#### Problem 1.2.1: Maximum Depth of Binary Tree

**Problem Statement:** Maximum Depth of Binary Tree Find the maximum depth of a binary tree, defined as the longest root to-leaf path. Use a recursive depth-first search (DFS) or iterative breadth-first search (BFS). The DFS approach runs in  $O(n)$  time and  $O(h)$  space, where  $h$  is the tree height. Edge cases include an empty tree (depth 0) and a skewed tree (depth  $n$ ).

### 2. Objectives:

- Implement efficient algorithms for array and string manipulation.
- Develop skills in using data structures like array, stacks and queues.
- Learn and apply brute force and greedy techniques to optimize solutions.

### 3. Implementation of Code:

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root==nullptr){
            return 0;
        }
        int leftdepth=maxDepth(root->left);
        int rightdepth=maxDepth(root->right);

        return 1+max(leftdepth,rightdepth);
    }
};
```

## 4. Output:

☒ Testcase | [> Test Result](#)

**Accepted** Runtime: 0 ms

- Case 1
- Case 2

**Input**

nums =  
[-10,-3,0,5,9]

**Output**

[0,-10,5,null,-3,null,9]

**Expected**

[0,-3,9,-10,null,5]

### Problem 1.1.2: Validate Binary Search Tree

**Problem Statement:** Validate Binary Search Tree Check if a binary tree follows the BST property: left subtree < root < right subtree. Use DFS with a range (min, max) to validate nodes recursively in O(n) time. Alternatively, an in-order traversal should produce a strictly increasing sequence. Edge cases include a single-node tree and duplicate values.

### Implementation of Code:

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return valid(root, LONG_MIN, LONG_MAX);
    }

private:
    bool valid(TreeNode* node, long minimum, long maximum) {
        if (!node) return true;

        if (!(node->val > minimum && node->val < maximum)) return false;

        return valid(node->left, minimum, node->val) && valid(node->right, node->val, maximum);
    }
};
```



Output:

☒ Testcase | [>\\_ Test Result](#)

**Accepted** Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

root =  
[3,9,20,null,null,15,7]

Output

[ [3] , [9,20] , [15,7] ]

Expected

[ [3] , [9,20] , [15,7] ]

## 1. Problem 1.1.3: Symmetric Tree

**Problem Statement:** Check if a binary tree is a mirror of itself. Use recursive DFS or iterative BFS (queue) to compare corresponding nodes. The solution runs in  $O(n)$  time and  $O(h)$  space. Edge cases include an empty tree (symmetric) and a single-node tree.

## 2. Implementation of Code:

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

    bool isSymmetric(TreeNode* root) {
        if (!root) return true;
        return isMirror(root->left, root->right);
    }
};
```

### 3. Output:

☒ Testcase | >\_ Test Result

**Accepted** Runtime: 0 ms

- Case 1
- Case 2

Input

root =  
[1,2,2,3,4,4,3]

Output

true

Expected

true

#### Problem 1.1.4: Binary Tree Level Order Traversal

**Problem Statement:** Return nodes level by level from a binary tree using BFS (queue). This runs in  $O(n)$  time and  $O(n)$  space. Edge cases include an empty tree (return []) and a single-node tree.

#### Implementation of code:

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        if(root==NULL)return ans;
        queue<TreeNode*>q;
        q.push(root);
        while(!q.empty()){
            int s=q.size();
            vector<int>v;
            for(int i=0;i<s;i++){
                TreeNode *node=q.front();
                q.pop();
                if(node->left!=NULL)q.push(node->left);
                if(node->right!=NULL)q.push(node->right);
                v.push_back(node->val);
            }
        }
    }
};
```



# DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        ans.push_back(v);
    }
    return ans;
}
};
```

## Output:

☒ Testcase | [Test Result](#)

**Accepted** Runtime: 0 ms

- Case 1
- Case 2

Input

root =  
[2,1,3]

Output

true

Expected

true

### Problem 1.1.5: Convert Sorted Array to Binary Search Tree

**Problem Statement:** Convert a sorted array into a height-balanced BST using a divide-and conquer approach. Recursively select the middle element as the root and build left and right subtrees. This runs in  $O(n)$  time and  $O(\log n)$  space. Edge cases include arrays of length 1 and 2.

### Implementation of code:

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return helper(nums, 0, nums.size() - 1);
    }

private:
    TreeNode* helper(vector<int>& nums, int left, int right) {
```

```
        if (left > right) return nullptr;  
        int mid = left + (right - left) / 2;  
        TreeNode* root = new TreeNode(nums[mid]);  
        root->left = helper(nums, left, mid - 1);  
        root->right = helper(nums, mid + 1, right);  
        return root;  
    }  
};
```

## Output:

☒ Testcase | ☒ Test Result

**Accepted** Runtime: 0 ms

• Case 1

• Case 2

Input

root =  
[3,9,20,null,null,15,7]

Output

3

Expected

3

## Learning Outcomes:

- Understand binary trees and BSTs.
- Implement recursive (DFS) and iterative (BFS) approaches.
- Apply divide-and-conquer, brute force, and greedy techniques.
- Use data structures like arrays, stacks, queues, and trees.
- Handle edge cases like empty or skewed trees.



# **DEPARTMENT OF COMPUTERSCIENCE & ENGINEERING**

Discover. Learn. Empower.