Experiment 6

Student Name: Prince Kumar
UID: 22BET10035
Branch: BE- IT
Section/Group: 22BET_IOT_703/B
Semester: 6th
Date of Performance: 25/01/25
Subject Name: AP LAB-II
Subject Code: 22ITP-351

1. Aim:

Problem1: Maximum Depth of Binary Tree
Problem2: Validate Binary Search Tree
Problem3: Symmetric Tree
Problem4: Binary Tree Level Order Traversal
Problem5: Convert Sorted Array to Binary Search Tree
Problem6: Binary Tree Inorder Traversal
Problem7: Binary Zigzag Level Order Traversal
Problem8: Construct Binary Tree from Inorder and Postorder Traversal

2. Objective:

1. The objective of this repository is to provide optimized C++ solutions for various Binary Tree-related problems from LeetCode.
2. These solutions help in understanding tree traversal techniques, recursion, iterative approaches, and other key concepts required for solving tree-based problems efficiently.

3. Code:

Problem 1

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

Problem 2

```
class Solution {
public:
    bool isValidBST(TreeNode* root, long minVal = LONG_MIN, long maxVal = LONG_MAX) {
        if (!root) return true;
        if (root->val <= minVal || root->val >= maxVal) return false;
```

```
        return isValidBST(root->left, minVal, root->val) && isValidBST(root->right, root->val, maxVal);
    }
};
```

Problem 3

```
class Solution {
public:
    bool isMirror(TreeNode* t1, TreeNode* t2) {
        if (!t1 && !t2) return true;
        if (!t1 || !t2) return false;
        return (t1->val == t2->val) && isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);
    }
    bool isSymmetric(TreeNode* root) {
        return isMirror(root, root);
    }
};
```

Problem 4

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

- **Problem 5**

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

Problem 6

```
class Solution {
public:
    vector<int>
inorderTraversal(TreeNode* root)
{
        vector<int> res;
        stack<TreeNode*> st;
        TreeNode* curr = root;
        while (curr || !st.empty()) {
            while (curr) {
                st.push(curr);
                curr = curr->left;
            }
            curr = st.top(); st.pop();
            res.push_back(curr->val);
            curr = curr->right;
        }
        return res;
    }
};
```

Problem 7

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);
        bool leftToRight = true;
```

```
        while (!q.empty()) {
            int size = q.size();
            vector<int> level(size);
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front(); q.pop();
                int index = leftToRight ? i : (size - 1 - i);
                level[index] = node->val;
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            res.push_back(level);
            leftToRight = !leftToRight;
        }
        return res;
    }
};
```

Problem 8

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

    TreeNode* build(vector<int>& post, vector<int>& in, int& postIndex, int inStart, int inEnd,
unordered_map<int, int>& inMap) {
        if (inStart > inEnd) return nullptr;

        TreeNode* root = new TreeNode(post[postIndex--]);

        root->right = build(post, in, postIndex, inMap[root->val] + 1, inEnd, inMap);
        root->left = build(post, in, postIndex, inStart, inMap[root->val] - 1, inMap);

        return root;
    }
};
```