# Assignment 3

## Student Name: Diksha Pandey UID:22BCS10359

## Branch: BE-CSE (General) Section/Group: FL\_IOT-602-A

## Semester:6th Date of Performance: 14-02-25

## Subject Name: Advanced Programming Lab-2 Subject Code: 22CSP-351

# **Aim:** [94. Binary Tree Inorder Traversal](https://leetcode.com/problems/binary-tree-inorder-traversal/)

# Implementation/ Code:

# class Solution {

# public:

# vector<int> inorderTraversal(TreeNode\* root) {

# vector<int> res;

# inorder(root, res);

# return res;

# }

# private:

# void inorder(TreeNode\* node, vector<int>& res) {

# if (!node) {

# return;

# }

# inorder(node->left, res);

# res.push\_back(node->val);

# inorder(node->right, res);

# }

# };

# Output:Screenshot 2025-02-14 at 2.00.30 PM.png

# **Aim:** [101. Symmetric Tree](https://leetcode.com/problems/symmetric-tree/)

# Implementation/ Code:

# class Solution {

# public:

# bool isSymmetric(TreeNode\* root) {

# return isMirror(root->left, root->right);

# }

# private:

# bool isMirror(TreeNode\* n1, TreeNode\* n2) {

# if (n1 == nullptr && n2 == nullptr) {

# return true;

# }

# 

# if (n1 == nullptr || n2 == nullptr) {

# return false;

# }

# 

# return n1->val == n2->val && isMirror(n1->left, n2->right) && isMirror(n1->right, n2->left);

# }

# };

# Output:Screenshot 2025-02-14 at 2.05.39 PM.png

# **Aim:** [104. Maximum Depth of Binary Tree](https://leetcode.com/problems/maximum-depth-of-binary-tree/)

# Implementation/ Code:

# class Solution {

# public:

# int maxDepth(TreeNode\* root) {

# if (!root) {

# return 0;

# }

# return 1 + max(maxDepth(root->left), maxDepth(root->right));

# }

# };

# Output:Screenshot 2025-02-14 at 2.07.26 PM.png

# **Aim:** [98. Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree/)

# Implementation/ 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:Screenshot 2025-02-14 at 2.09.48 PM.png

# **Aim:**[230. Kth Smallest Element in a BST](https://leetcode.com/problems/kth-smallest-element-in-a-bst/)

# Implementation/ Code:

# class Solution {

# public:

# void preOrderTraversal(TreeNode\* root, vector<int> &v){

# if(root == NULL) return;

# 

# //root, left, right

# v.push\_back(root->val);

# preOrderTraversal(root->left, v);

# preOrderTraversal(root->right, v);

# }

# int kthSmallest(TreeNode\* root, int k) {

# vector<int> v;

# preOrderTraversal(root, v);

# sort(v.begin(), v.end());

# return v[k-1];

# }

# };

# Output:Screenshot 2025-02-14 at 2.12.28 PM.png

# **Aim:** [102. Binary Tree Level Order Traversal](https://leetcode.com/problems/binary-tree-level-order-traversal/)

# Implementation/ 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();

# currLevel.push\_back(node->val);

# if (node->left)

# q.push(node->left);

# if (node->right)

# q.push(node->right);

# }

# ans.push\_back(currLevel);

# }

# return ans;

# }

# };

# Output:Screenshot 2025-02-14 at 2.15.42 PM.png

# **Aim:** [107. Binary Tree Level Order Traversal II](https://leetcode.com/problems/binary-tree-level-order-traversal-ii/)

# Implementation/ Code:

# class Solution {

# public:

# vector<vector<int>> levelOrderBottom(TreeNode\* root) {

# if (!root) return {};

# vector<vector<int>> result;

# 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);

# }

# result.push\_back(level);

# }

# reverse(result.begin(), result.end());

# return result;

# }

# };

# Output:Screenshot 2025-02-14 at 2.20.27 PM.png

# **Aim:** [103. Binary Tree Zigzag Level Order Traversal](https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/)

# Implementation/ Code:

# class Solution {

# public:

# vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

# if (root == nullptr)

# return {};

# vector<vector<int>> ans;

# deque<TreeNode\*> dq{{root}};

# bool isLeftToRight = true;

# while (!dq.empty()) {

# vector<int> currLevel;

# for (int sz = dq.size(); sz > 0; --sz)

# if (isLeftToRight) {

# TreeNode\* node = dq.front();

# dq.pop\_front();

# currLevel.push\_back(node->val);

# if (node->left)

# dq.push\_back(node->left);

# if (node->right)

# dq.push\_back(node->right);

# } else {

# TreeNode\* node = dq.back();

# dq.pop\_back();

# currLevel.push\_back(node->val);

# if (node->right)

# dq.push\_front(node->right);

# if (node->left)

# dq.push\_front(node->left);

# }

# ans.push\_back(currLevel);

# isLeftToRight = !isLeftToRight;

# }

# return ans;

# }

# };

# Output:Screenshot 2025-02-14 at 2.23.21 PM.png

# **Aim:** [199. Binary Tree Right Side View](https://leetcode.com/problems/binary-tree-right-side-view/)

# Implementation/ Code:

# class Solution {

# public:

# vector<int> res;

# unordered\_map<int,int> mp;

# void check(TreeNode\* root,int n){

# if(!root){

# return;

# }

# if(!(mp.find(n) != mp.end())){

# res.push\_back(root->val);

# mp[n]++;

# }

# check(root->right,n+1);

# check(root->left,n+1);

# }

# vector<int> rightSideView(TreeNode\* root) {

# check(root,0);

# return res;

# }

# };

# Output:Screenshot 2025-02-14 at 2.27.05 PM.png

# **Aim:** [106. Construct Binary Tree from Inorder and Postorder Traversal](https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/)

# Implementation/ Code:

# class Solution {

# public:

# TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {

# unordered\_map<int, int> index;

# for (int i = 0; i < inorder.size(); i++) {

# index[inorder[i]] = i;

# }

# return buildTreeHelper(inorder, postorder, 0, inorder.size() - 1, 0, postorder.size() - 1, index);

# }

# 

# TreeNode\* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int inorderStart, int inorderEnd, int postorderStart, int postorderEnd, unordered\_map<int, int>& index) {

# if (inorderStart > inorderEnd || postorderStart > postorderEnd) {

# return nullptr;

# }

# int rootVal = postorder[postorderEnd];

# TreeNode\* root = new TreeNode(rootVal);

# int inorderRootIndex = index[rootVal];

# int leftSubtreeSize = inorderRootIndex - inorderStart;

# root->left = buildTreeHelper(inorder, postorder, inorderStart, inorderRootIndex - 1, postorderStart, postorderStart + leftSubtreeSize - 1, index);

# root->right = buildTreeHelper(inorder, postorder, inorderRootIndex + 1, inorderEnd, postorderStart + leftSubtreeSize, postorderEnd - 1, index);

# return root;

# }

# };

# Output:Screenshot 2025-02-14 at 2.29.44 PM.png

# **Aim:** [513. Find Bottom Left Tree Value](https://leetcode.com/problems/find-bottom-left-tree-value/)

# Implementation/ Code:

# class Solution {

# public:

# void tt(TreeNode\* root, int level, vector<vector<int>>&nums){

# if(root==NULL){

# return;

# }

# if(nums.size()<=level){

# nums.push\_back({});

# }

# nums[level].push\_back(root->val);

# tt(root->right,level+1,nums);

# tt(root->left,level+1,nums);

# }

# int findBottomLeftValue(TreeNode\* root) {

# vector<vector<int>>nums;

# tt(root,0,nums);

# return nums.back().back();

# }

# };

# Output:Screenshot 2025-02-14 at 2.32.50 PM.png

# **Aim:** [124. Binary Tree Maximum Path Sum](https://leetcode.com/problems/binary-tree-maximum-path-sum/)

# Implementation/ Code:

# class Solution {

# public:

# int maxSum = INT\_MIN;

# int maxPathSum(TreeNode\* root) {

# helper(root);

# return maxSum;

# }

# private:

# int helper(TreeNode\* node) {

# if (!node) return 0;

# int leftMaxPath = max(helper(node->left), 0);

# int rightMaxPath = max(helper(node->right), 0);

# int maxIfNodeIsRoot = node->val + leftMaxPath + rightMaxPath;

# maxSum = max(maxSum, maxIfNodeIsRoot);

# return node->val + max(leftMaxPath, rightMaxPath);

# }

# };

# Output:Screenshot 2025-02-14 at 2.34.24 PM.png

# **Aim:** [987. Vertical Order Traversal of a Binary Tree](https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/)

# Implementation/ Code:

# class Solution {

# public:

# vector<vector<int>> verticalTraversal(TreeNode\* root) {

# map<int,map<int,multiset<int>>>nodes;

# queue<pair<TreeNode\*,pair<int,int>>>q;

# q.push({root,{0,0}});

# while(!q.empty()){

# auto t = q.front();

# q.pop();

# TreeNode\* a = t.first;

# int x =t.second.first, y = t.second.second;

# nodes[x][y].insert(a->val);

# if(a->left){

# q.push({a->left,{x-1,y+1}});

# }

# if(a->right){

# q.push({a->right,{x+1,y+1}});

# }

# }

# vector<vector<int>>ans;

# for(auto p: nodes){

# vector<int>col;

# for(auto b:p.second){

# col.insert(col.end(),b.second.begin(),b.second.end());

# }

# ans.push\_back(col);

# }

# return ans;

# }

# };

# Output:Screenshot 2025-02-14 at 2.35.54 PM.png