Experiment 6

Student Name: Armaan Siag
UID: 22BET10322
Branch: Information Technology
Section/Group: 22BET_IOT-703/A
Semester: 6th
Subject Code: 22ITP-351

Problem 1

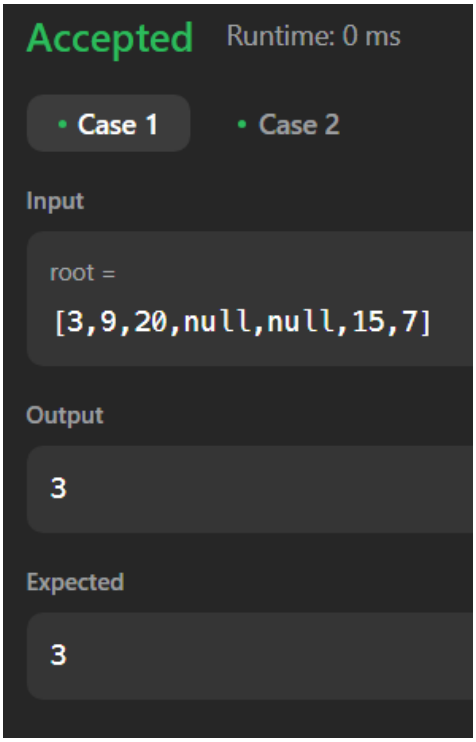
Aim:

Maximum Depth of Binary Tree

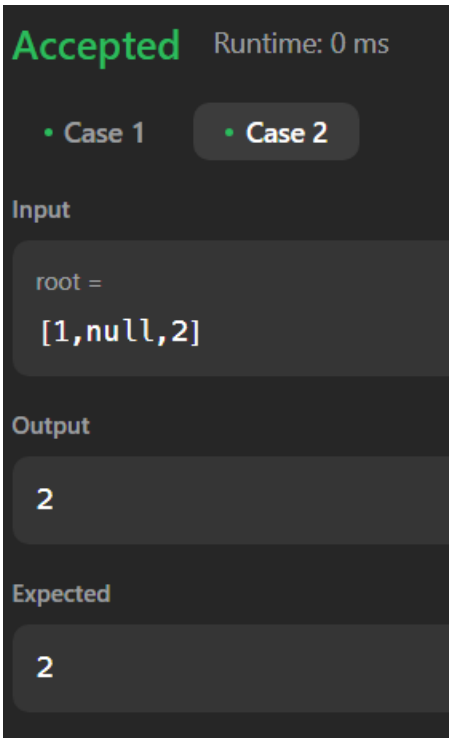
Code:

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

Output:



Case 1



Case 2

Problem 2

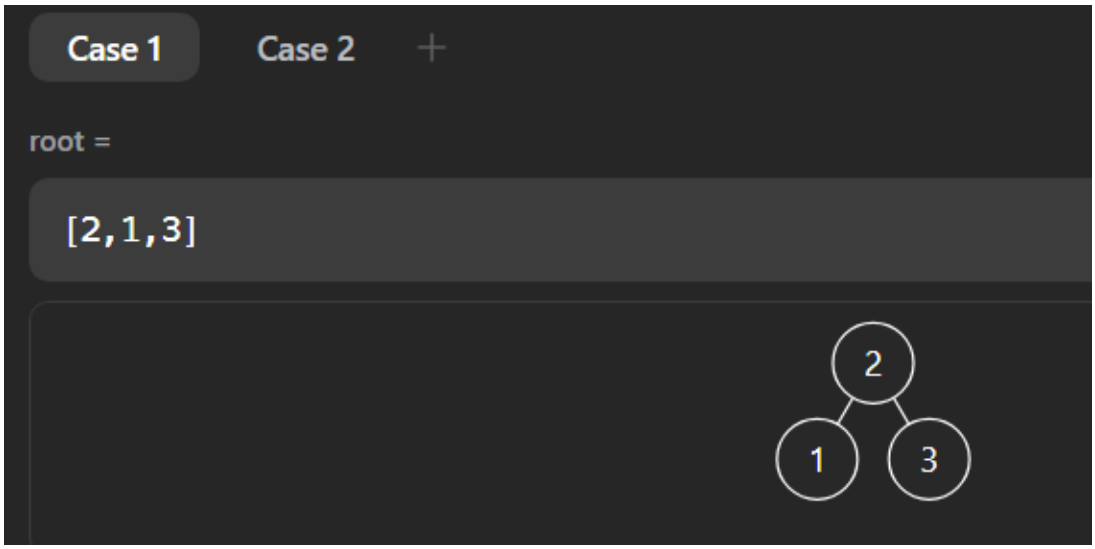
Aim:

Validate Binary Search Tree

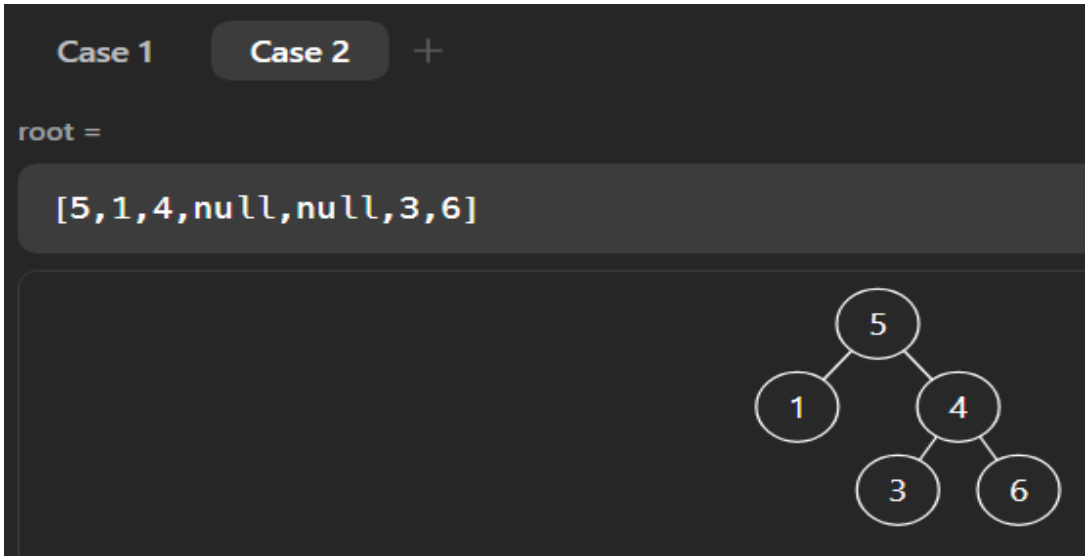
Code:

```
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:



Test Case 1



Test Case 2

Problem 3

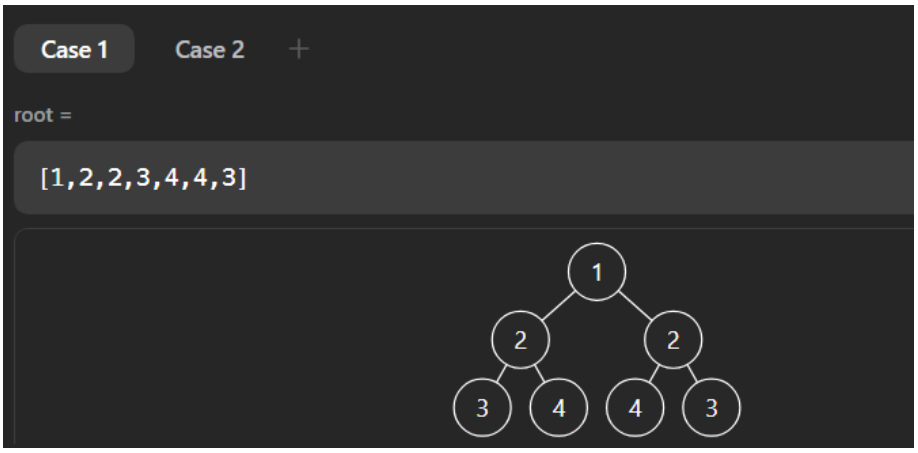
Aim:

Symmetric Tree

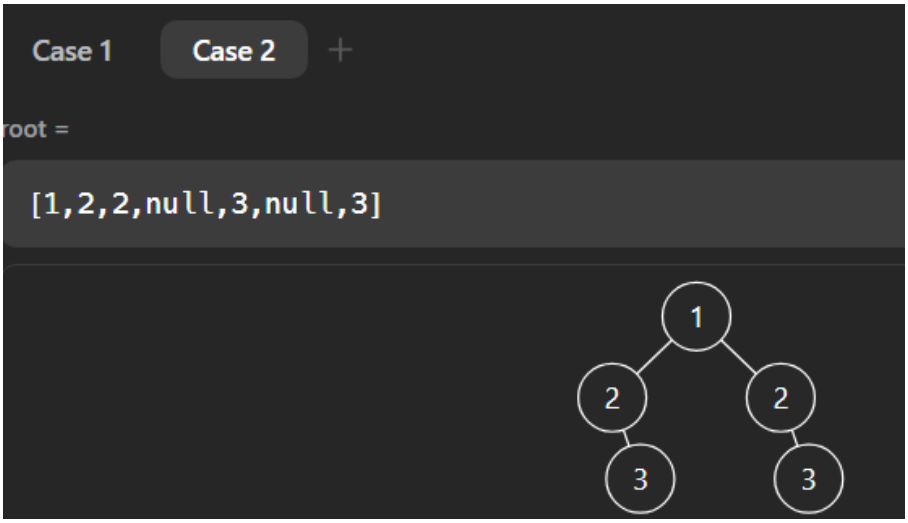
Code:

```
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:



Case 1



Case 2

Problem 4

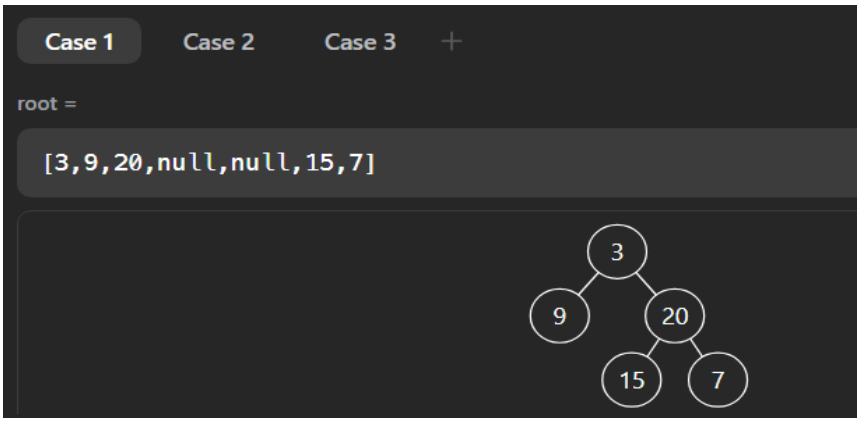
Aim:

Binary Tree Level Order Traversal

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

Output:



Case 1

Case 2

Problem 5

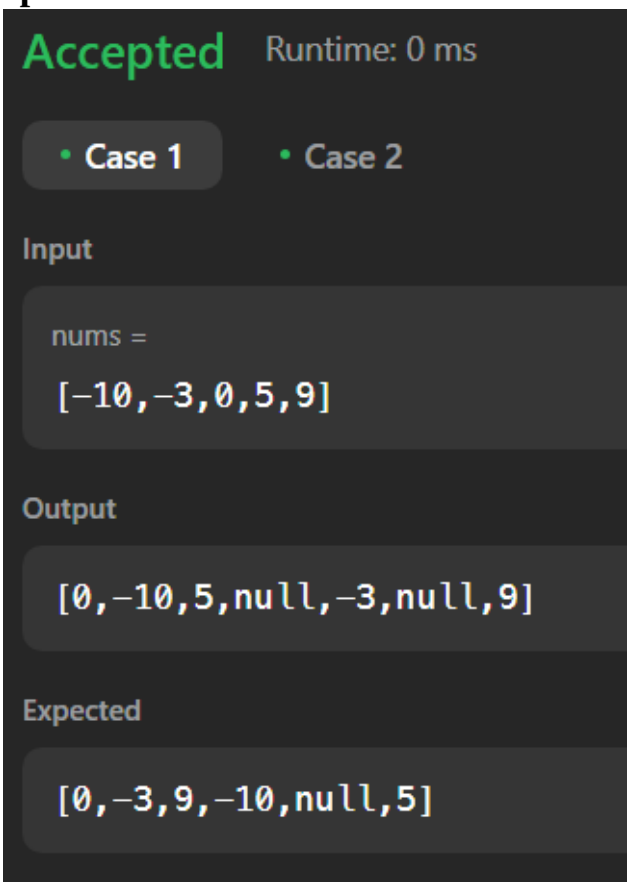
Aim:

Convert Sorted Array to Binary Search Tree

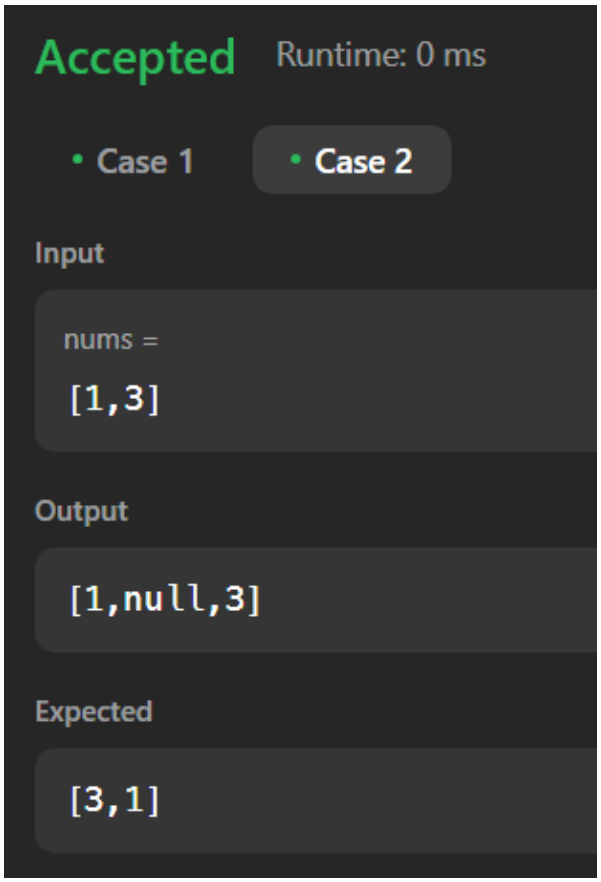
Code:

```cpp
#include <vector>
using namespace std;

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:



Case 1



Case 2

Problem 6

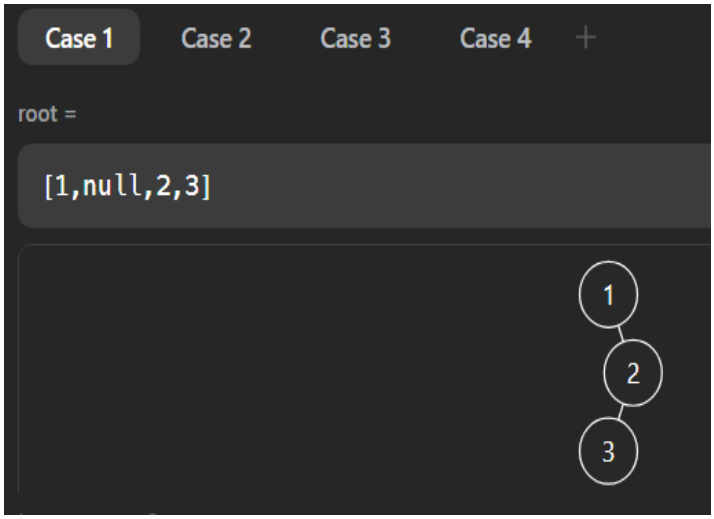
Aim:

Binary Tree Inorder Traversal

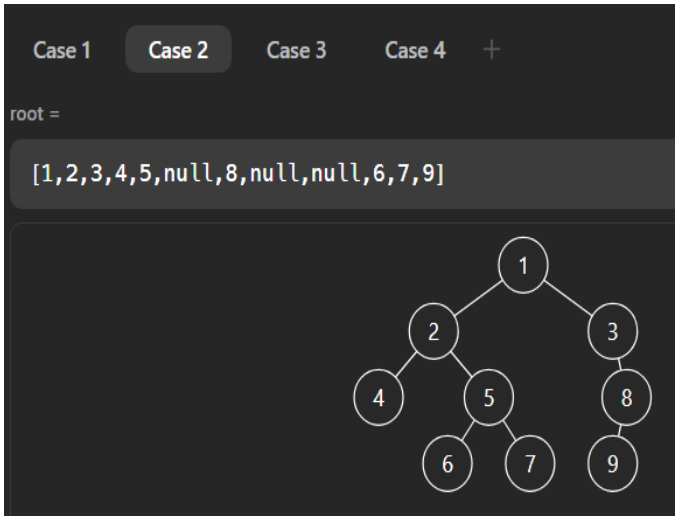
Code:

```cpp
class Solution {
public:
    void inorder(TreeNode* root, vector<int>& ans) {
        if (!root) return;
        inorder(root->left, ans);
        ans.push_back(root->val);
        inorder(root->right, ans);
    }

    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        inorder(root, ans);
        return ans;
    }
};
```

Output:



Case 1



Case 2

Case 3

Problem 7

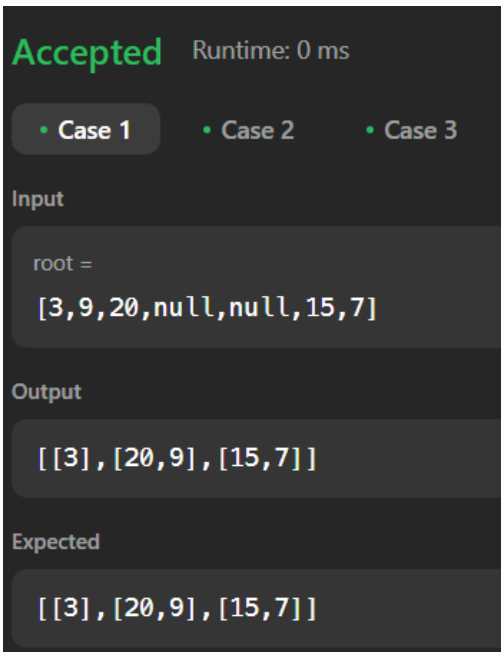
Aim:

Binary Zigzag Level Order Traversal

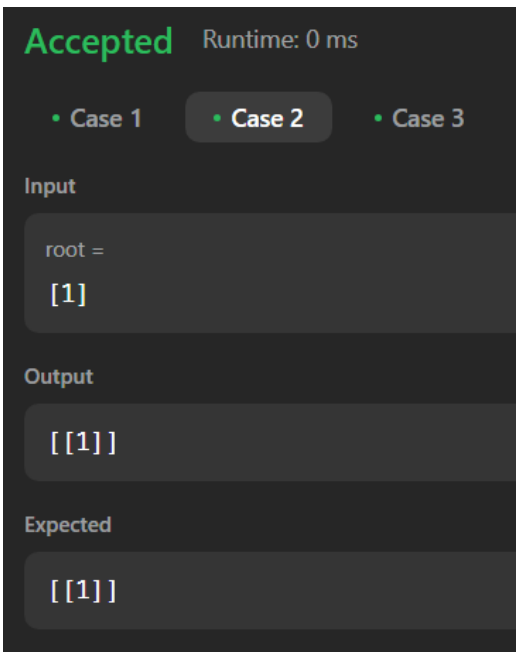
Code:

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

        while (!q.empty()) {
            int levelSize = q.size();
            vector<int> level(levelSize);
            for (int i = 0; i < levelSize; ++i) {
                TreeNode* node = q.front();
                q.pop();
                int index = leftToRight ? i : (levelSize - 1 - i);
                level[index] = node->val;

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            leftToRight = !leftToRight;
            result.push_back(level);
        }

        return result;
    }
};
```

Output:



Case 1



Case 2

Problem 8

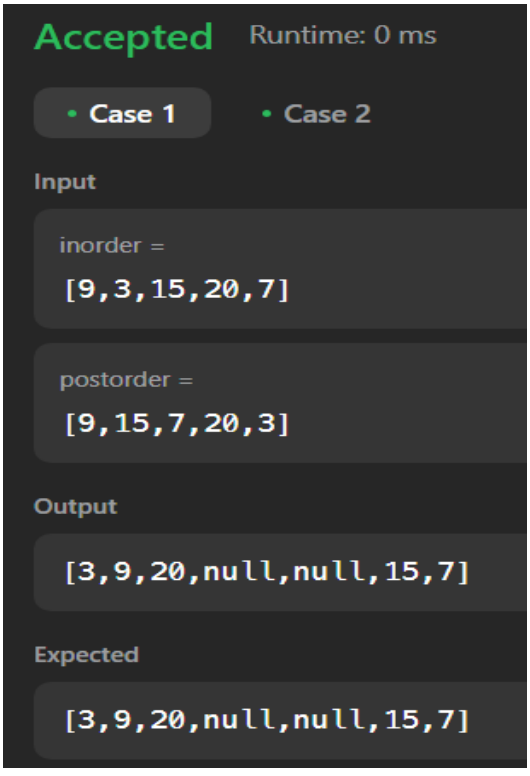
Aim:

Construct Binary Tree from Inorder and Postorder Traversal

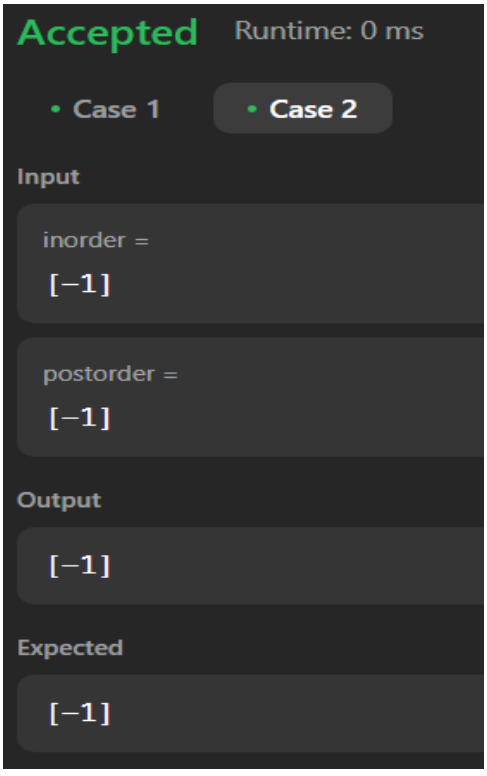
Code:

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder,int instart,int inend, vector<int>& postorder,int poststart,int postend,map<int,int>& inMap){
        if(instart>inend || poststart>postend){
            return NULL;
        }
        TreeNode* root=new TreeNode(postorder[postend]);
        int inRoot=inMap[root->val];
        int numsLeft=inRoot-instart;
        root->left=buildTree(inorder,instart,inRoot-1,postorder,poststart,poststart+numsLeft-1,inMap);
        root->right=buildTree(inorder,inRoot+1,inend,postorder,poststart+numsLeft,postend-1,inMap);
        return root;
    }
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        map<int,int> inMap;
        for(int i=0;i<inorder.size();i++){
            inMap[inorder[i]]=i;
        }
        TreeNode* root=buildTree(inorder,0,inorder.size()-1,postorder,0,postorder.size()-1,inMap);
        return root;
    }
};
```

Output:



Case 1



Case 2

Problem 9

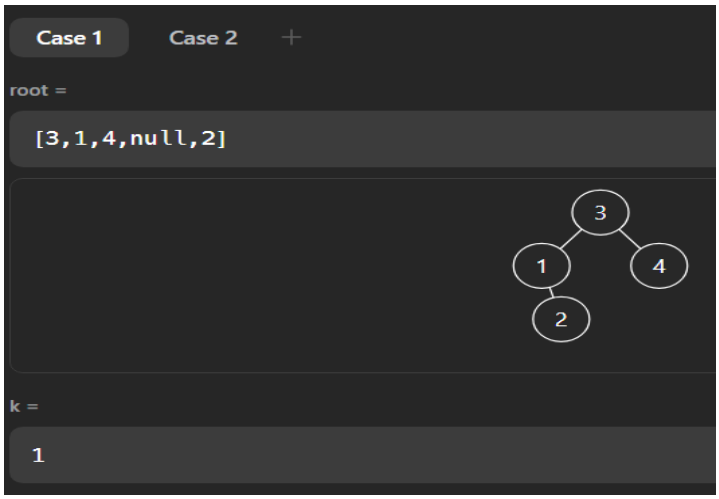
Aim:

Kth Smallest element in a BST

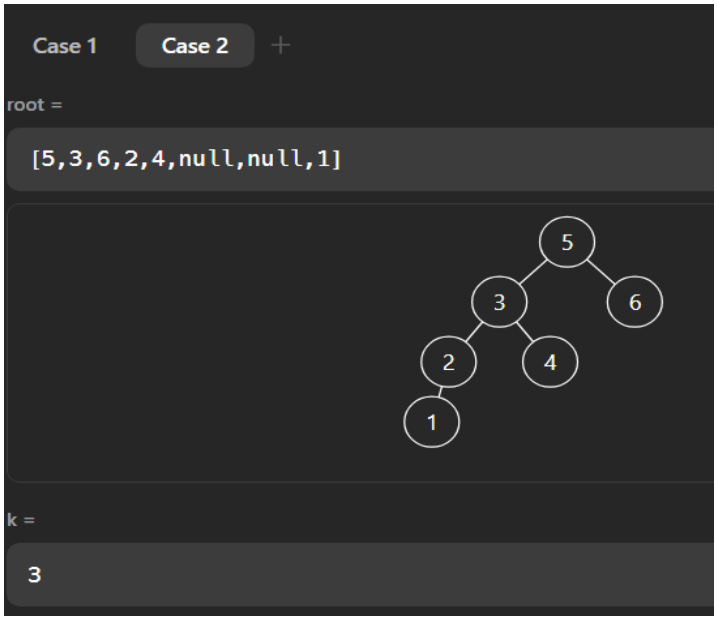
Code:

```
class Solution {
public:
    void inOrderTraversal(TreeNode* root, vector<int> &v){
        if(root == NULL)    return;

        //left, root, right
        inOrderTraversal(root->left, v);
        v.push_back(root->val);
        inOrderTraversal(root->right, v);
    }
    int kthSmallest(TreeNode* root, int k) {
        vector<int> v;
        inOrderTraversal(root, v);
        return v[k-1];
    }
};
```

Output:



Case 1



Case 2

Problem 10

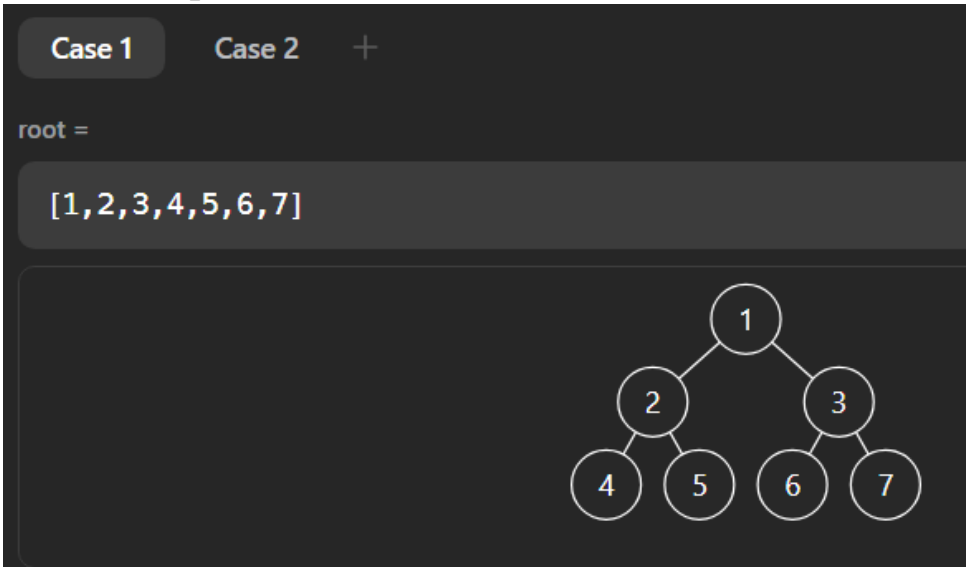
Aim:

Populating Next Right Pointers in Each Node

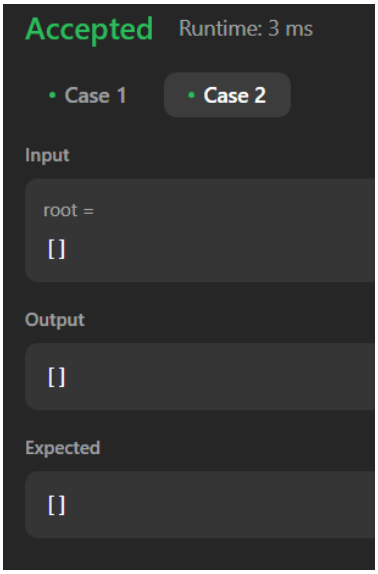
Code:

```
class Solution {
public:
    Node* connect(Node* root) {
        if(root==nullptr) return {};

        queue<Node*> q;
        q.push(root);

        while(!q.empty()){
            int n = q.size();

            for(int i=0;i<n;i++){
                Node* t = q.front();
                q.pop();

                if(i!=n-1){
                    t->next=q.front();
                }

                if(t->left) q.push(t->left);

                if(t->right) q.push(t->right);
            }
        }
        return root;
    }
};
```

Output:



Case 1



Case 2