Experiment 6

Student Name: Arun
UID: 22BET10320
Branch: Information Technology
Section/Group: 22BET_IOT-701/A
Semester: 6th
Subject Code: 22ITP-351

Problem 1

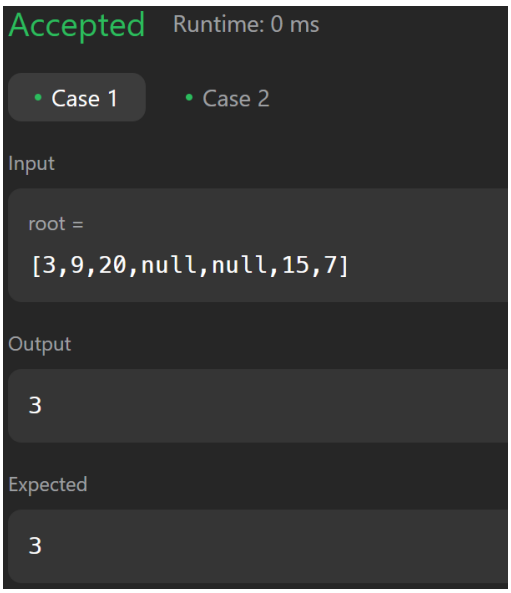
Aim: Maximum Depth of Binary Tree

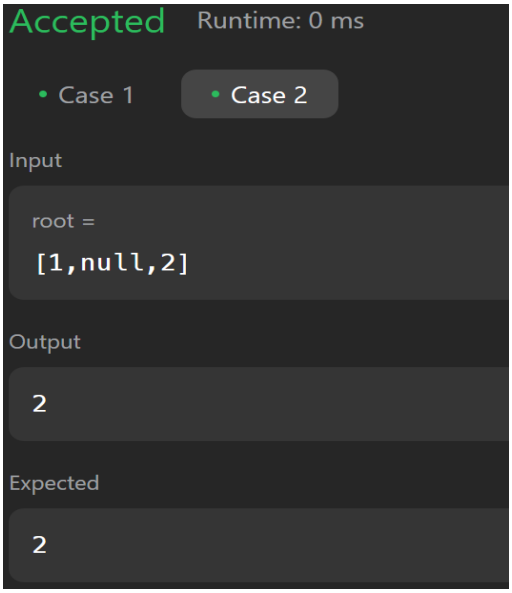
Code:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root)
    {
        if(root==NULL)
            return 0;
        return max(1+maxDepth(root->left), 1+maxDepth(root->right));

    }
};
```

Output:





Problem: 2

Aim: Validate Binary Search Tree

Code:

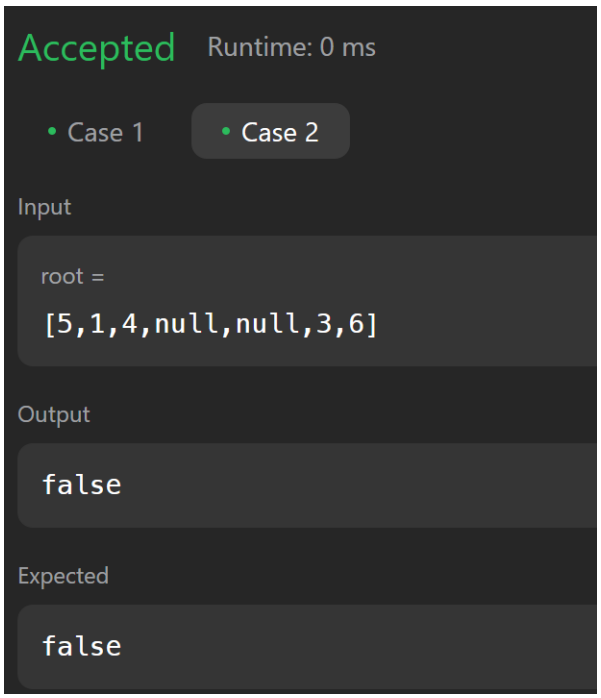
```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return valid(root, LONG_MIN, LONG_MAX);
    }

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:



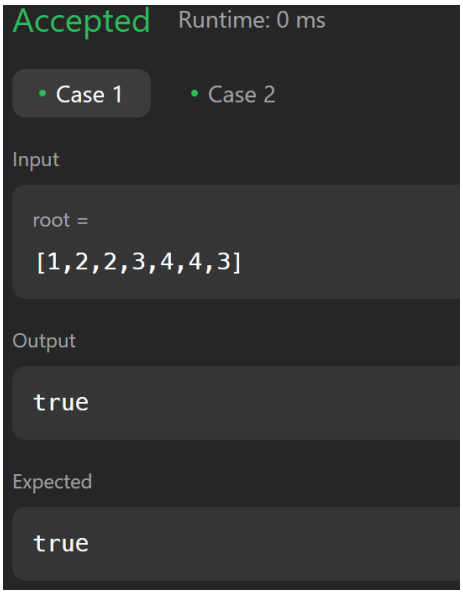
Problem: 3

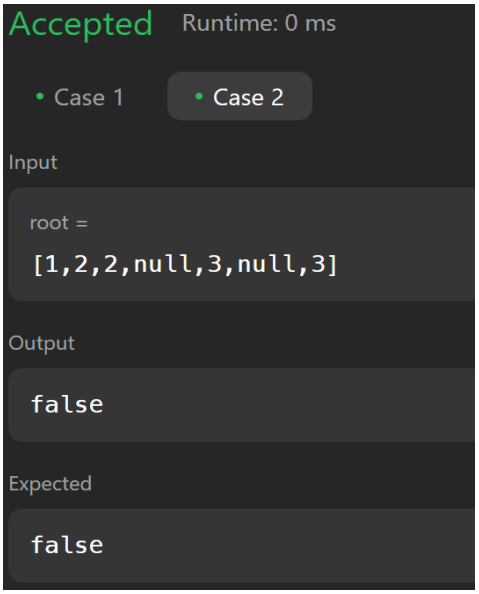
Aim: Symmetric Tree

Code:

```
class Solution {
public:
    bool checkForSubTrees(TreeNode* p, TreeNode* q) {
        if (p == NULL && q == NULL) {
            return true;
        }
        if (p == NULL || q == NULL) {
            return false;
        }
        if (p->val != q->val) {
            return false;
        }
        bool leftSide = checkForSubTrees(p->left, q->right);
        bool rightSide = checkForSubTrees(p->right, q->left);
        if (leftSide && rightSide) {
            return true;
        }
        return false;
    }
    bool isSymmetric(TreeNode* root) {
        if (root == NULL || (root->left == NULL && root->right == NULL)) {
            return true;
        } if ((root->left == NULL && root->right != NULL) ||
            (root->left != NULL && root->right == NULL)) {
            return false;
        } return checkForSubTrees(root->left, root->right);
    }
};
```

Output:





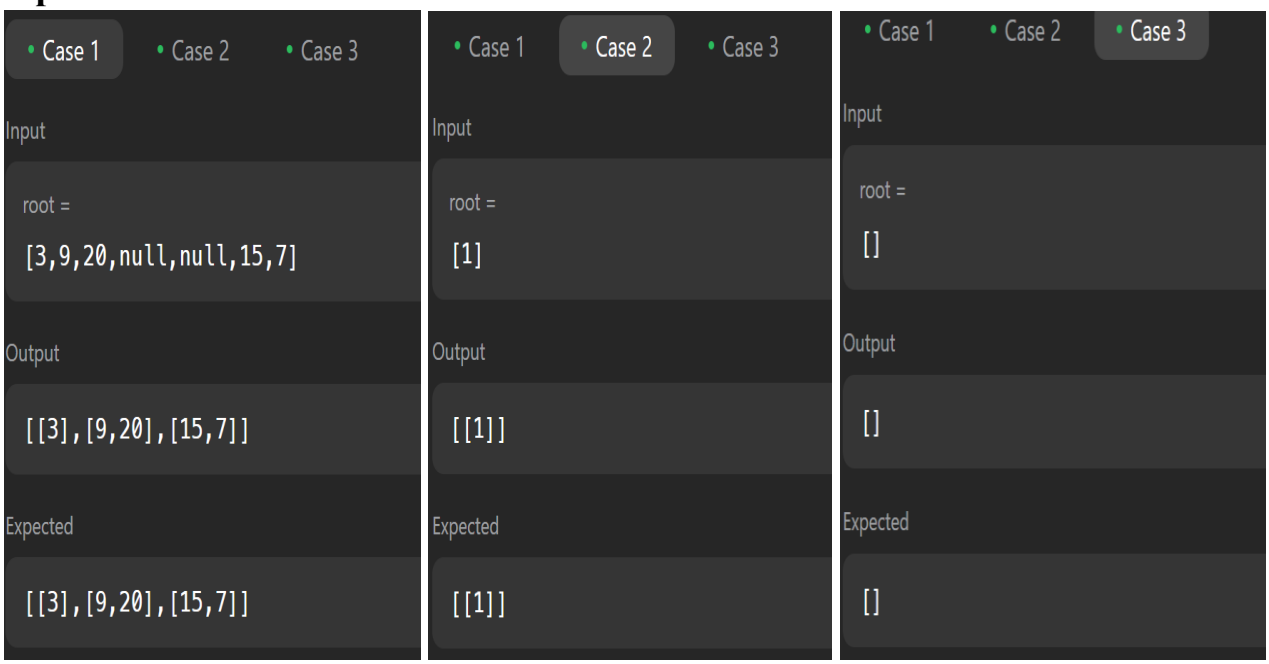
Problem: 4

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:



Problem :5

Aim: Convert Sorted Array to Binary Search Tree

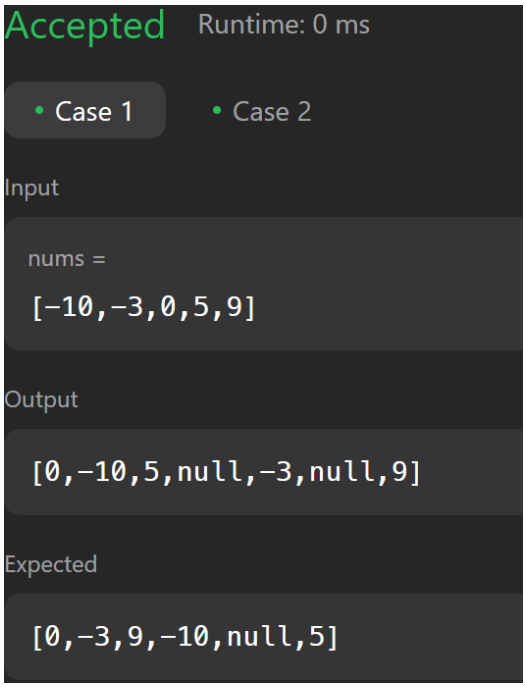
Code:

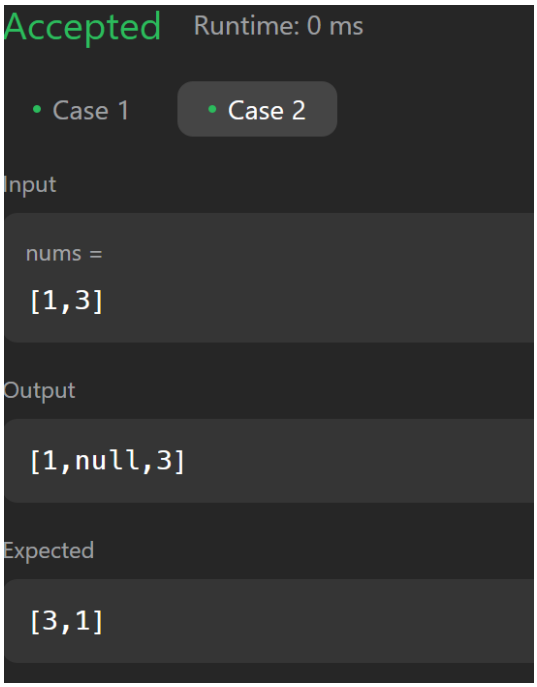
```
#include <vector>
using namespace std;

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:





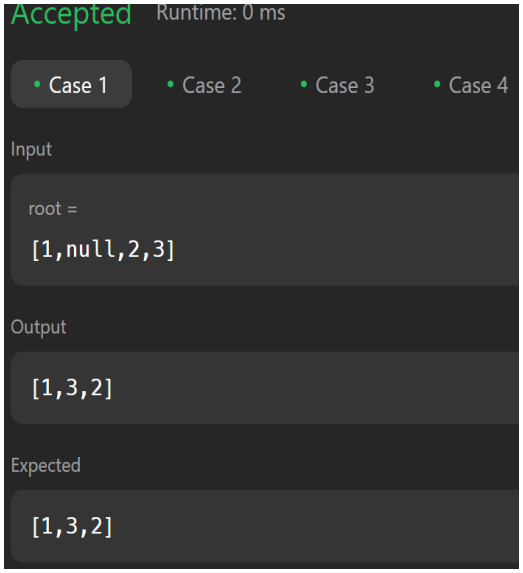
Problem:6

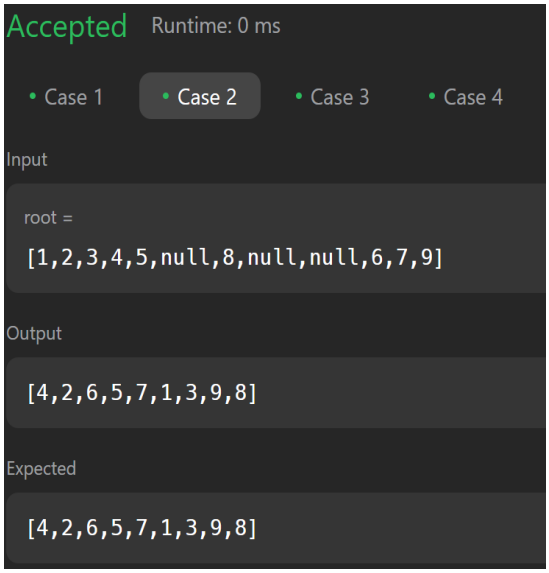
Aim: Binary Tree Inorder Traversal

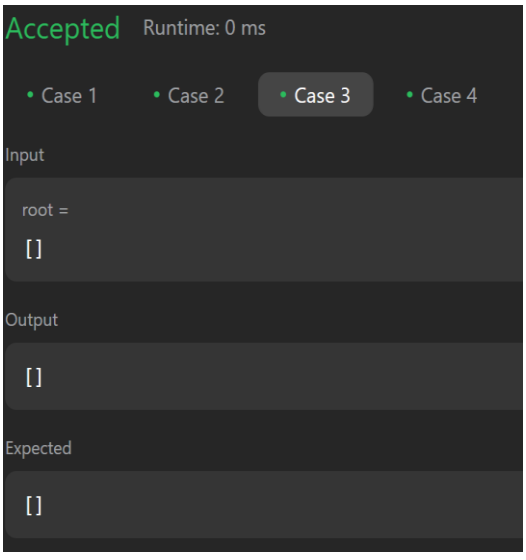
Code:

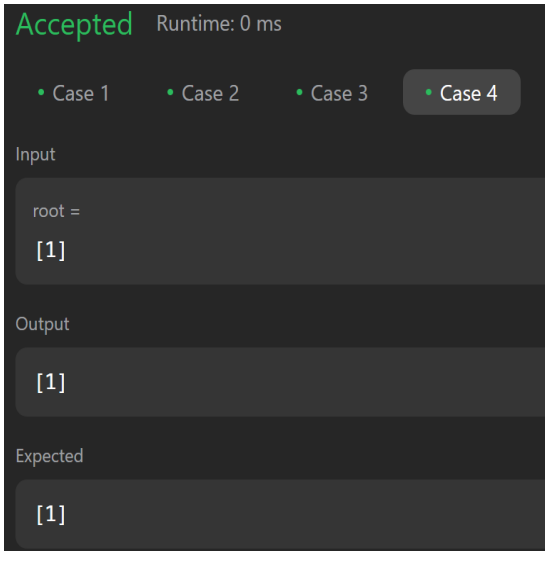
```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        inorder(root, res);
        return res;
    }
private:
    void inorder(TreeNode* node, vector<int>& res) {
        if (!node) {
            return;
        }
        inorder(node->left, res);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

Output:





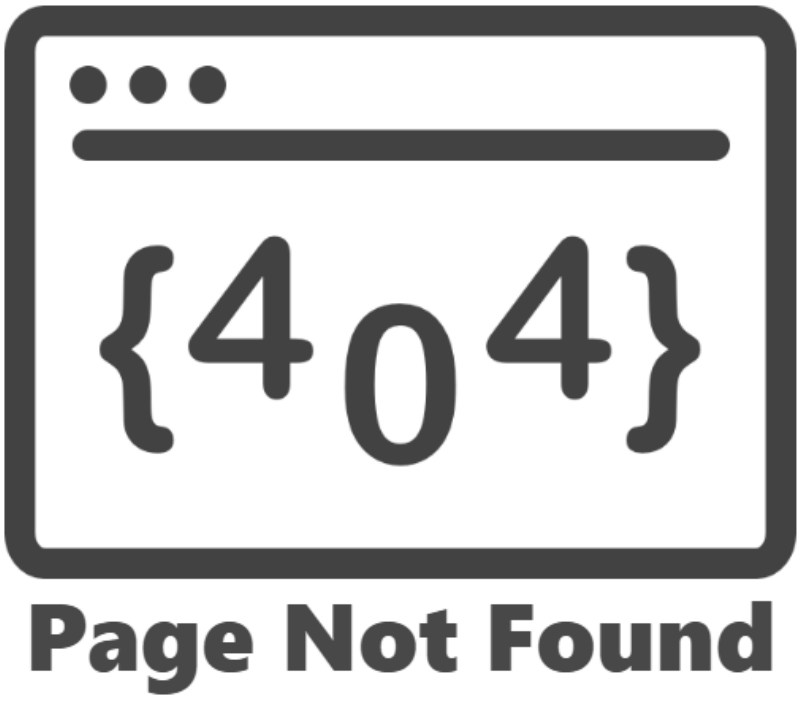




Problem:7

Aim: Binary Zigzag Level Order Traversal

Code:



Page Not Found

Sorry, but we can't find the page you are looking for...

Back to Home

Problem:8

Aim: Construct Binary Tree from Inorder and Postorder Traversal

Code:

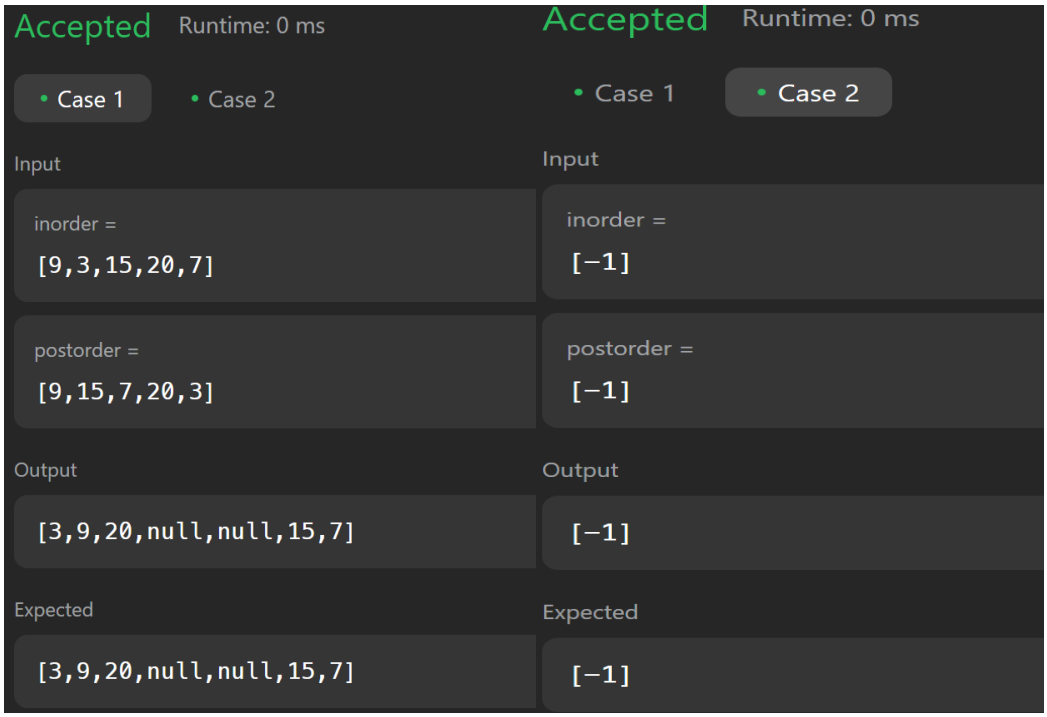
```
class Solution {

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:


Accepted
Runtime: 0 ms
Case 1
Case 2
Input
inorder =
[9,3,15,20,7]
postorder =
[9,15,7,20,3]
Output
[3,9,20,null,null,15,7]
Expected
[3,9,20,null,null,15,7]
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
inorder =
[-1]
postorder =
[-1]
Output
[-1]
Expected
[-1]

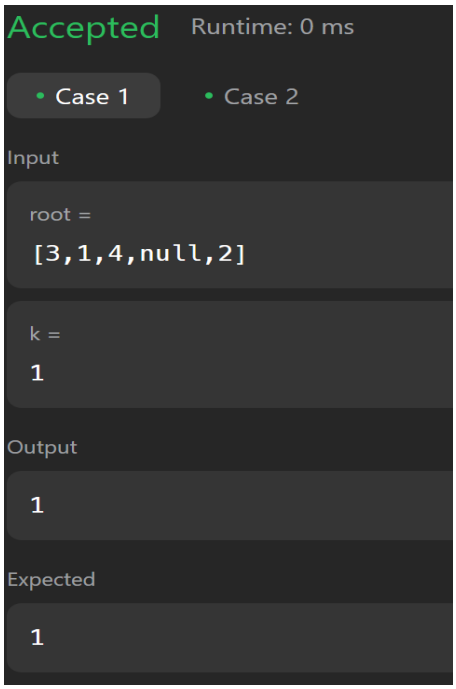
Problem:9

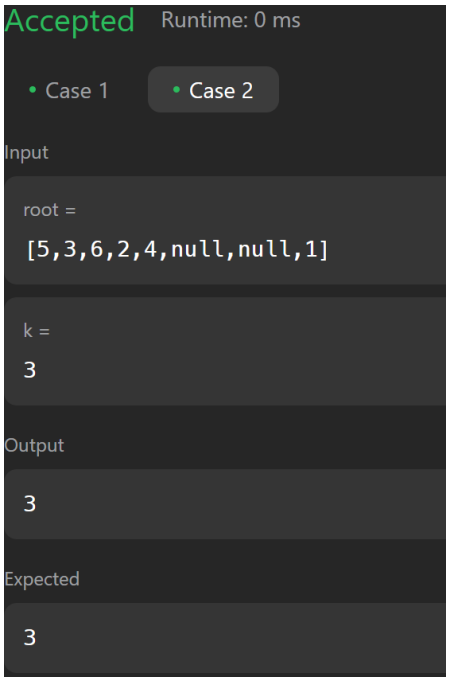
Aim: Kth Smallest element in a BST

Code:

```
class Solution {
 public:
  int kthSmallest(TreeNode* root, int k) {
    const int leftCount = countNodes(root->left);
    if (leftCount == k - 1)
      return root->val;
    if (leftCount >= k)
      return kthSmallest(root->left, k);
    return kthSmallest(root->right, k - 1 - leftCount);
  }
 private:
  int countNodes(TreeNode* root) {
    if (root == nullptr)
      return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
  }
};
```

Output:





Problem: 10

Aim: Populating Next Right ointers in Each Node

Code:

```
class Solution {
public:
    Node* connect(Node* root) {
        if(!root) return nullptr;
        queue<Node*> q;
        q.push(root);
        while(size(q)) {
            Node* rightNode = nullptr;              // set rightNode to null initially
            for(int i = size(q); i; i--) {          // traversing each level
                auto cur = q.front(); q.pop();      // pop a node from current level and,
                cur -> next = rightNode;            // set its next pointer to rightNode
                rightNode = cur;                    // update rightNode as cur for next iteration
                if(cur -> right)                    // if a child exists
                    q.push(cur -> right),           // IMP: push right first to do right-to-left BFS
                    q.push(cur -> left);            // then push left
            }
        }
        return root;
    }
};
```

Output: