Experiment 5

Student Name: Jain Aman **UID: 22BCS14831**
Branch: CSE **Section/Group: 637-B**
Semester: 6th **Date of Performance:20/2/25**
Subject Name: Advanced Programming - 2 **Subject Code: 22CSH-351**

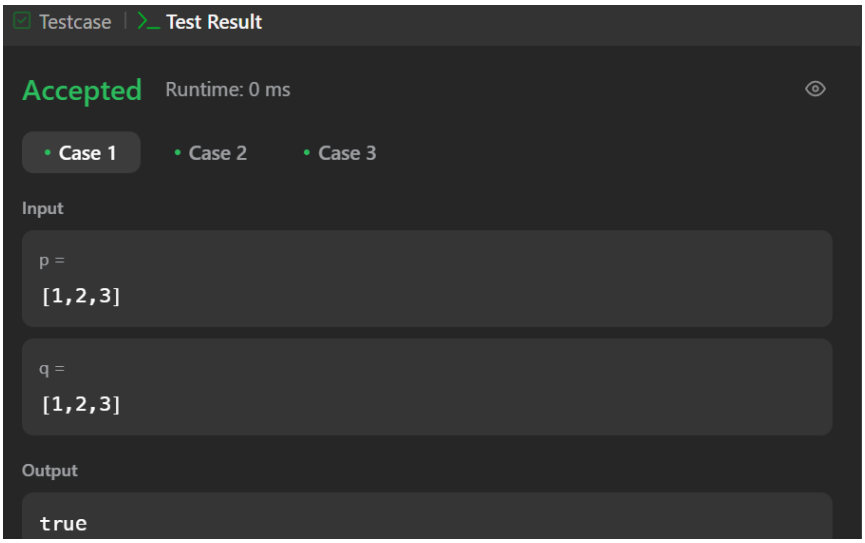
Ques 1:

Aim: Same Tree

Code:

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

Submission Screenshot:



Ques 2:

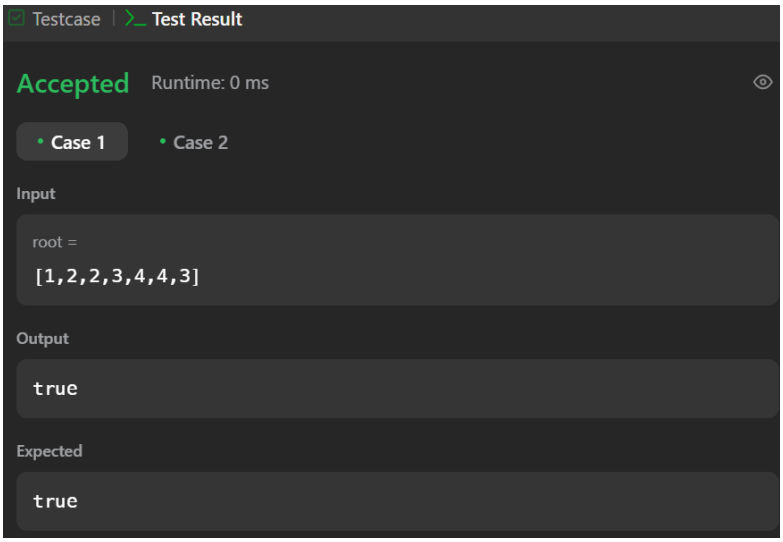
Aim: Symmetric Tree

Code:

```
#include <queue>

class Solution {
public:
    // Recursive approach
    bool isSymmetric(TreeNode* root) {
        return root ? isMirror(root->left, root->right) : true;
    }

    bool isMirror(TreeNode* t1, TreeNode* t2) {
        if (!t1 && !t2) return true;
        if (!t1 || !t2 || t1->val != t2->val) return false;
        return isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);
    }

    // Iterative approach using queue
    bool isSymmetricIterative(TreeNode* root) {
        if (!root) return true;
        std::queue<TreeNode*> q;
        q.push(root->left);
        q.push(root->right);

        while (!q.empty()) {
            TreeNode* t1 = q.front(); q.pop();
            TreeNode* t2 = q.front(); q.pop();

            if (!t1 && !t2) continue;
            if (!t1 || !t2 || t1->val != t2->val) return false;

            q.push(t1->left);
            q.push(t2->right);
            q.push(t1->right);
```

```
                q.push(t2->left);
            }
            return true;
        }
};
```

Submission Screenshot:



Ques 3:

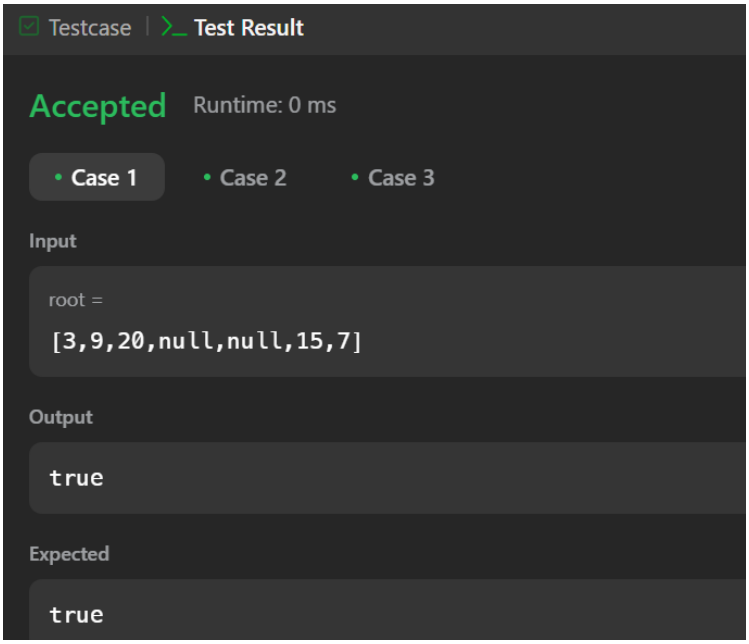
Aim: Balanced Binary Tree

Code:

```
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        return height(root) != -1;
    }

    int height(TreeNode* node) {
        if (!node) return 0;
        int left = height(node->left);
        int right = height(node->right);
        if (abs(left - right) > 1 || left == -1 || right == -1) return -1;
        return max(left, right) + 1;
```

```
    }
};
```

Submission Screenshot:



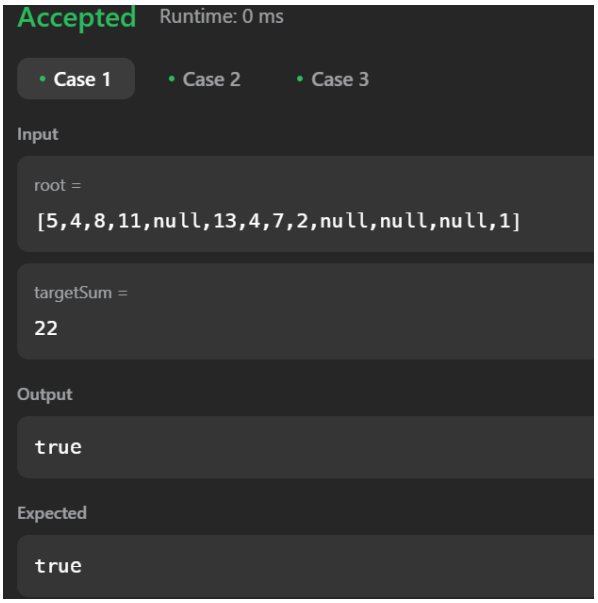
Ques 4:

Aim: Path Sum

Code:

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

Submission Screenshot:



Ques 5:

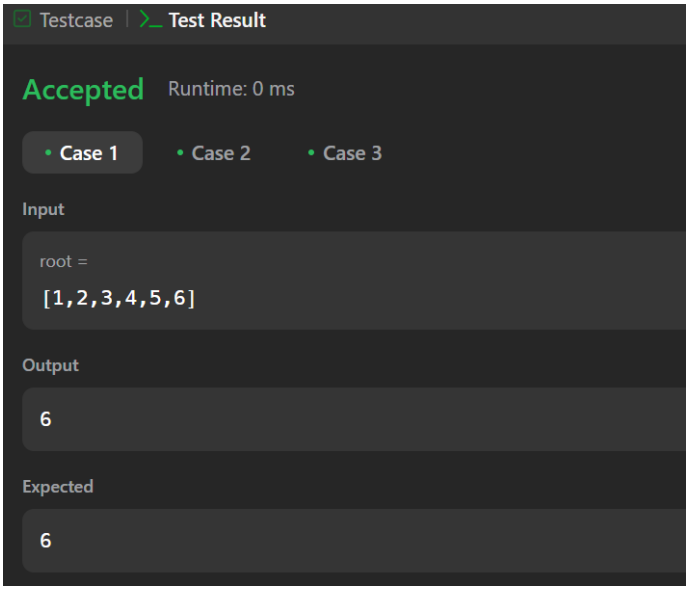
Aim: Count Complete Tree Nodes

Code:

```
class Solution {
public:
    int countNodes(TreeNode* root) {
        if (!root) return 0;

        int leftHeight = getHeight(root->left);
        int rightHeight = getHeight(root->right);

        if (leftHeight == rightHeight) {
            return (1 << leftHeight) + countNodes(root->right);
        } else {
            return (1 << rightHeight) + countNodes(root->left);
        }
    }

private:
```

```
        int getHeight(TreeNode* node) {
            int height = 0;
            while (node) {
                height++;
                node = node->left;
            }
            return height;
        }
};
```

Submission Screenshot:



Ques 6:

Aim: Delete node in a BST

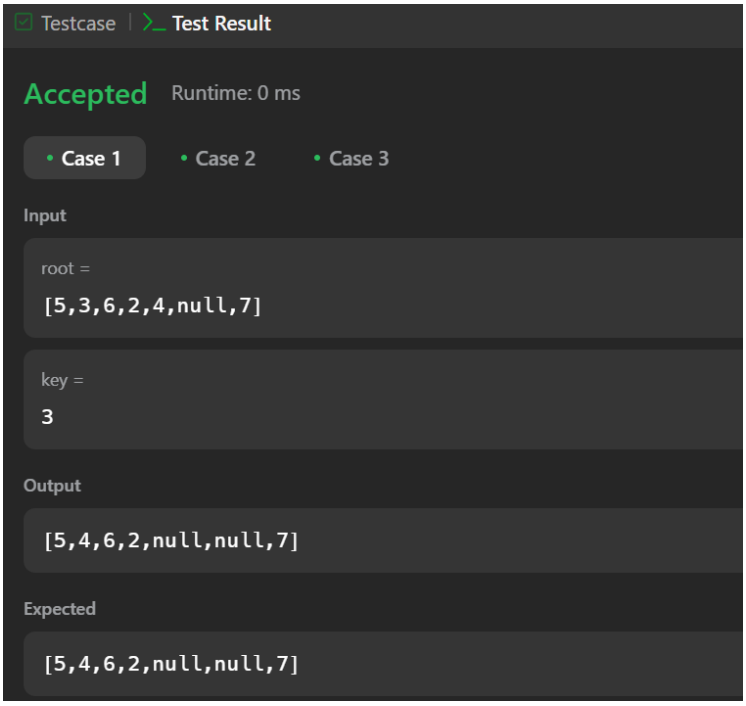
Code:

```
class Solution {
public:
    TreeNode* deleteNode(TreeNode* root, int key) {
        if (!root) return nullptr;
```

```
        if (key < root->val) {
            root->left = deleteNode(root->left, key);
        } else if (key > root->val) {
            root->right = deleteNode(root->right, key);
        } else {
            // Case 1: Node has no child
            if (!root->left && !root->right) {
                delete root;
                return nullptr;
            }
            // Case 2: Node has only one child
            if (!root->left) {
                TreeNode* temp = root->right;
                delete root;
                return temp;
            }
            if (!root->right) {
                TreeNode* temp = root->left;
                delete root;
                return temp;
            }
            // Case 3: Node has two children
            TreeNode* minNode = getMin(root->right);
            root->val = minNode->val;
            root->right = deleteNode(root->right, minNode->val);
        }
        return root;
    }

private:
    TreeNode* getMin(TreeNode* node) {
        while (node->left) node = node->left;
```

```
            return node;
        }
};
```

Submission Screenshot:



Ques 7:

Aim: Diameter of Binary Tree

Code:

```
class Solution {
public:
    int diameterOfBinaryTree(TreeNode* root) {
        int diameter = 0;
        depth(root, diameter);
        return diameter;
    }
```

```
    private:
        int depth(TreeNode* node, int& diameter) {
            if (!node) return 0;

            int leftDepth = depth(node->left, diameter);
            int rightDepth = depth(node->right, diameter);

            // Update the diameter with the longest path through the root
            diameter = max(diameter, leftDepth + rightDepth);

            // Return the height of the subtree
            return 1 + max(leftDepth, rightDepth);
        }
};
```

Submission Screenshot: