Experiment-6

Name: Aayushi Sinha
UID: 22BET10268
Branch: BE-IT
Section/Group: 22BET_IOT-702/B
Semester: 6th
Date of Performance: 7/03/25
Subject Name: AP LAB-II
Subject Code: 22ITP-351

Problem-1

1.Aim:

Given the root of a binary tree, return its maximum depth.
A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

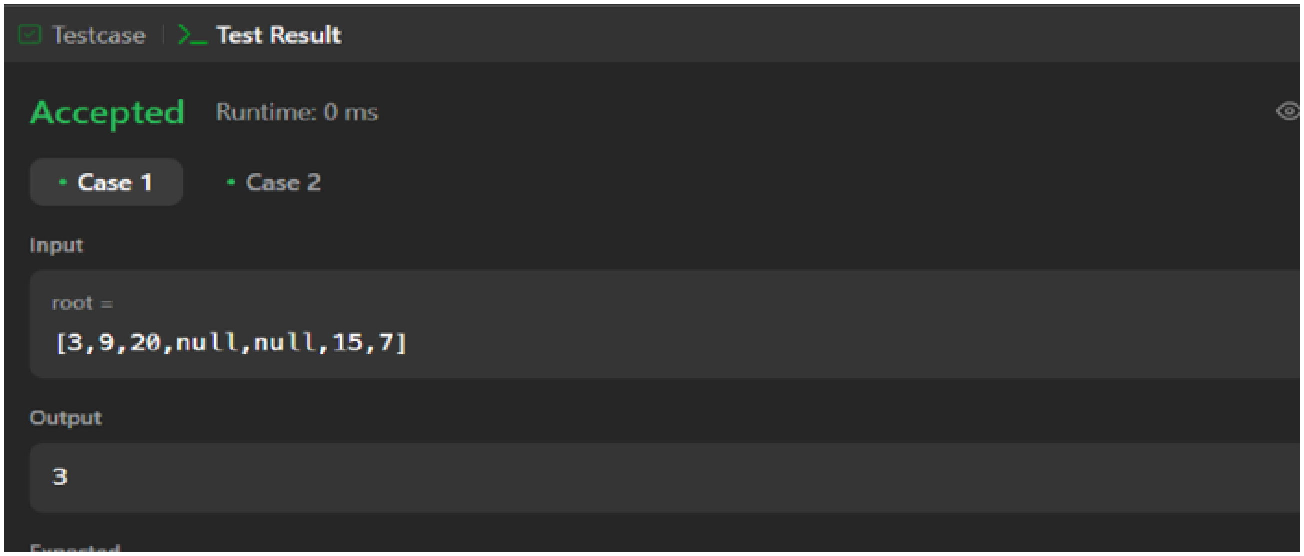
2.Objective:

- Learn how to recursively or iteratively compute the height (or depth) of a binary tree.

3.Code:

```
class Solution {

public:

int maxDepth (TreeNode* root) {

if (root == nullptr)

return 0;

int leftDepth = maxDepth (root->left);

int rightDepth = maxDepth(root->right);

return 1 + max(leftDepth, rightDepth);

}
```

4.Output:


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[3,9,20,null,null,15,7]
Output
3

Problem-2

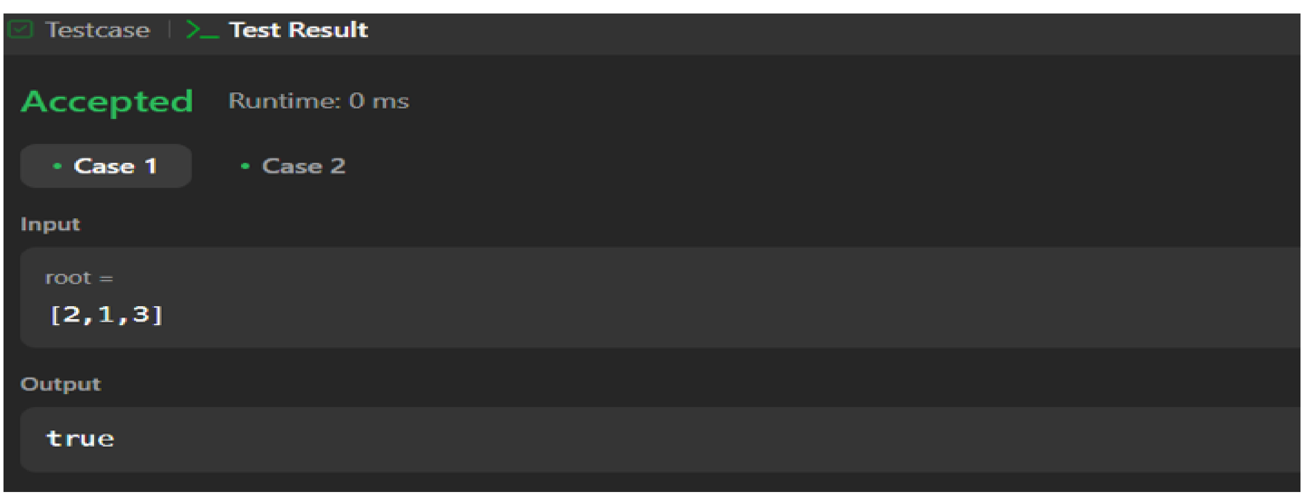
1.Aim:
Given the root of a binary tree, determine if it is a valid binary search tree (BST).

2.Objective:
- Understand the properties of a Binary Search Tree (BST).

3.Code:

```
class Solution {
public:
bool isValidBST (TreeNode root, long long minVal = LLONG MIN,
long long maxVal = LLONG 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);
```

4.Output:



Problem-3

1.Aim:

Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

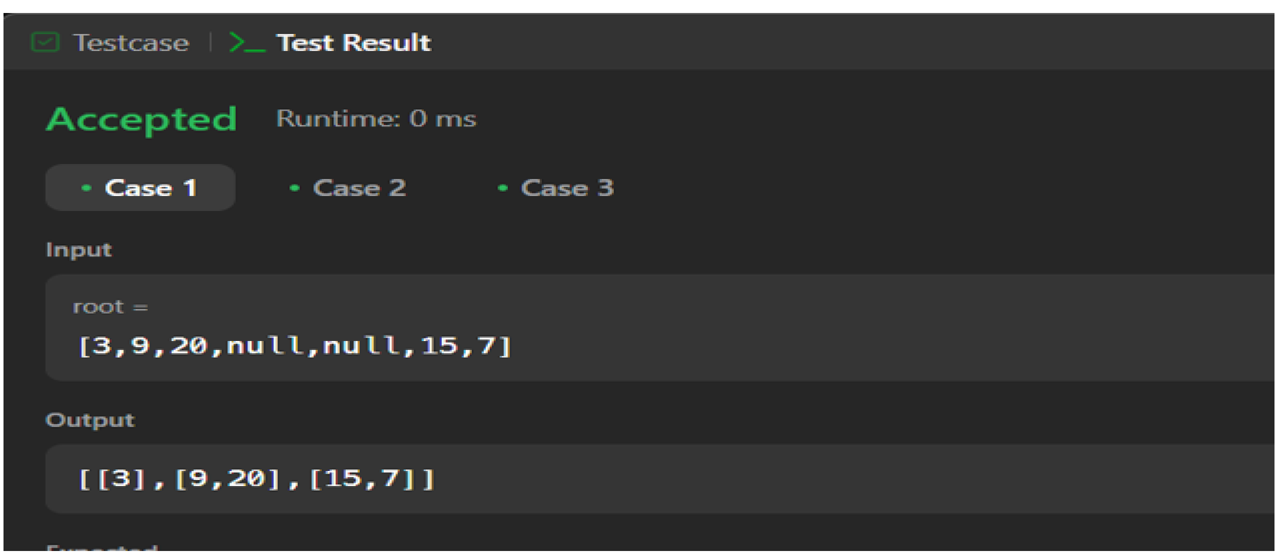
2.Objective:

- Understand mirror properties of a binary tree.

3.Code:

```
class Solution {

public:

bool isMirror (TreeNode t1, TreeNode t2) {

if (!t1 && !t2) return true;

if (!t1 || 1t2) return false;

return (t1->val == t2->val) && isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);
```

4.Output:



Problem-4

1.Aim:

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

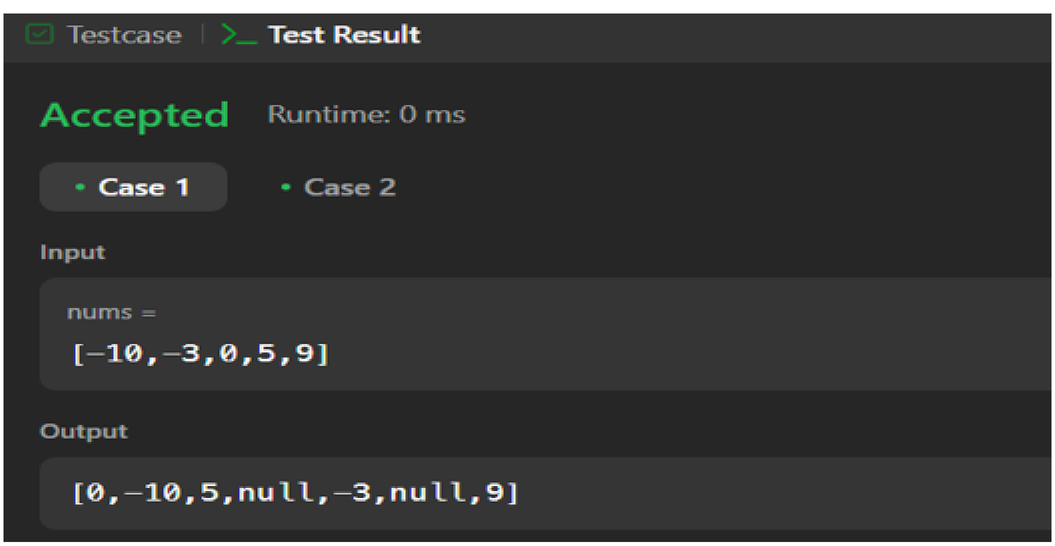
2.Objective:

- Learn Breadth-First Search (BFS) using queues.

3.Code:

```
#include <vector>

using namespace std;

class Solution {

public:

TreeNode sortedArrayToBST (vector<int>& nums) {

return buildBST (nums, 0, nums.size() - 1);

}
```

4.Output:



Problem-5

1.Aim:

Given an integer array nums where the elements are sorted in ascending order, convert it to a height balanced binary search tree.

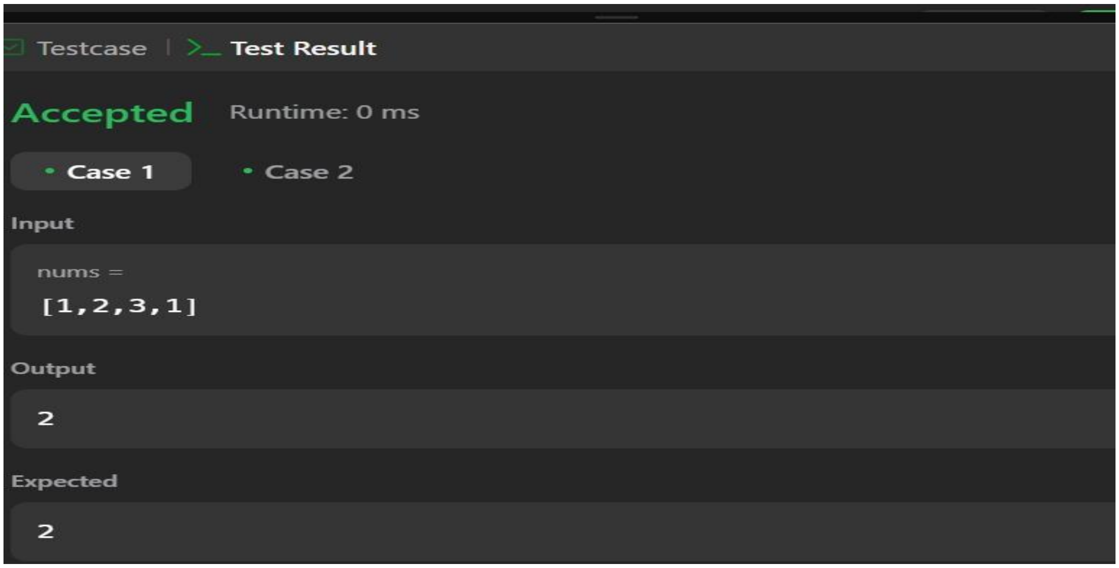
2.Objective:

- Learn how to construct a height-balanced BST from a sorted array.

3.Code:

```
class Solution {

public:

    int findPeakElement(vector<int>& nums) {

        int left = 0, right = nums.size() - 1;



        while (left < right) {

            int mid = left + (right - left) / 2;

            if (nums[mid] > nums[mid + 1]) {

                right = mid;

            } else {

                left = mid + 1;

            }

        }

        return left;

    }

};
```

4.Output:


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
nums =
[1,2,3,1]
Output
2
Expected
2

Problem-6

1.Aim:

Given the root of a binary tree, return the inorder traversal of its nodes' values.

2.Objective:

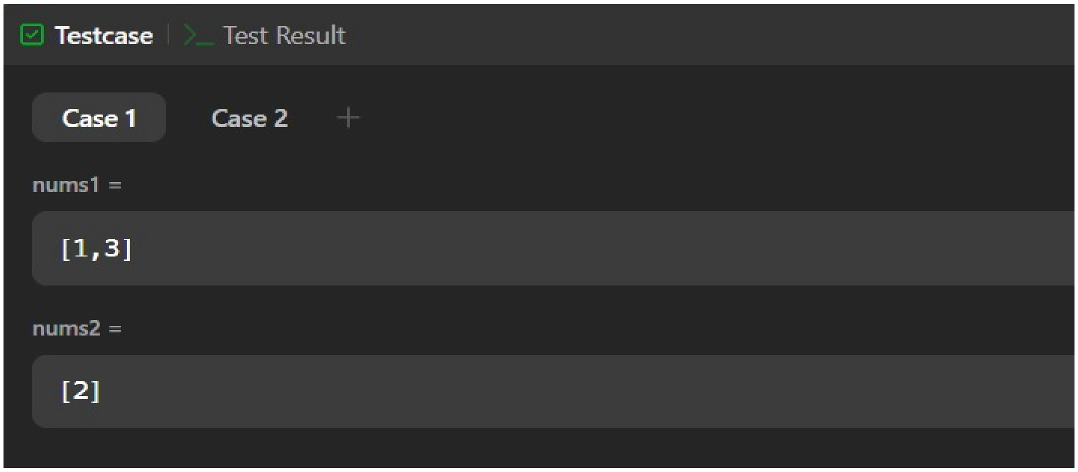
- Understand inorder traversal (left-root-right).

3.Code:

```
class Solution {

public:

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

      if (nums1.size() > nums2.size()) {

          return findMedianSortedArrays(nums2, nums1);

      }


      int x = nums1.size();

      int y = nums2.size();

      int low = 0, high = x;

      while (low <= high) {

          int partitionX = (low + high) / 2;

          int partitionY = (x + y + 1) / 2 - partitionX;


          int maxLeftX = (partitionX == 0) ? INT_MIN : nums1[partitionX - 1];

          int minRightX = (partitionX == x) ? INT_MAX : nums1[partitionX];


          int maxLeftY = (partitionY == 0) ? INT_MIN : nums2[partitionY - 1];
```

```
            int minRightY = (partitionY == y) ? INT_MAX : nums2[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                if ((x + y) % 2 == 0) {
                    return (max(maxLeftX, maxLeftY) + min(minRightX, minRightY)) / 2.0;
                } else {
                    return max(maxLeftX, maxLeftY);
                }
            } else if (maxLeftX > minRightY) {
                high = partitionX - 1;
            } else {
                low = partitionX + 1;
            }
        }
        return 0.0;
    }
};
```

4.Output:



Problem-7

1.Aim:

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

2.Objective:

.

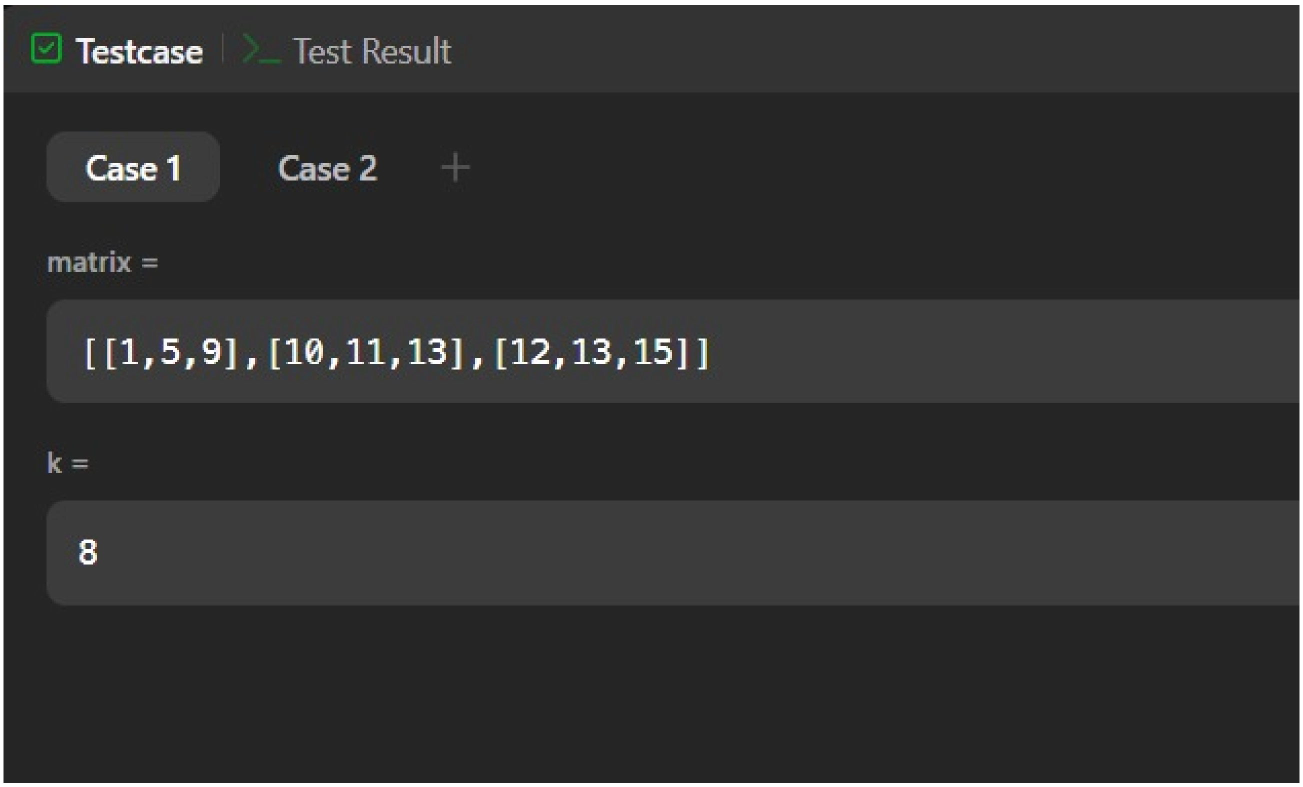
- Extend BFS level-order traversal to include zigzag ordering.

3.Code:

```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
        int n = matrix.size();
        int left = matrix[0][0], right = matrix[n - 1][n - 1];

        while (left < right) {
            int mid = left + (right - left) / 2;
            int count = countLessEqual(matrix, mid, n);

            if (count < k) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }
```

```
        return left;
    }

  private:
    int countLessEqual(vector<vector<int>>& matrix, int mid, int n) {
        int count = 0, row = n - 1, col = 0;

        while (row >= 0 && col < n) {
            if (matrix[row][col] <= mid) {
                count += (row + 1); // All elements in this column above are also ≤ mid
                col++; // Move right
            } else {
                row--; // Move up
            }
        }
        return count;


    }
};
```

4.Output:


Testcase
Test Result
Case 1
Case 2
matrix =
[[1,5,9],[10,11,13],[12,13,15]]
k =
8

Problem-8

1.Aim:

Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree.

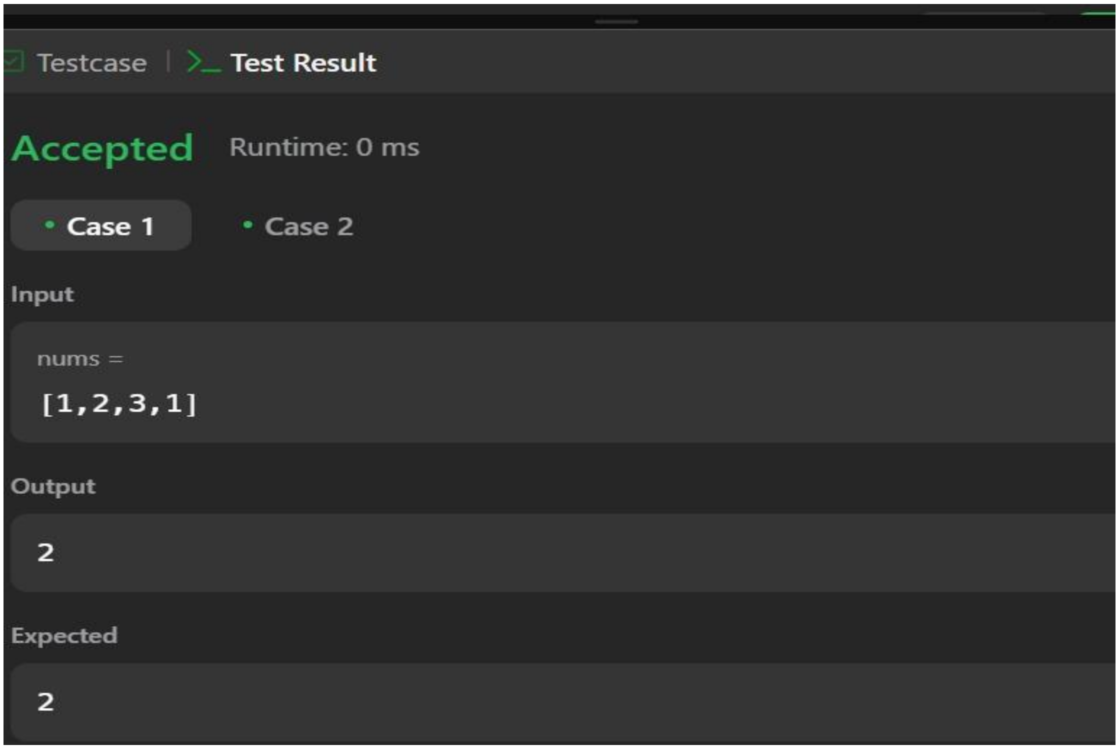
2.Objective:

- Understand tree reconstruction from traversal orders.

3.Code:

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int left = 0, right = nums.size() - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] > nums[mid + 1]) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }
        return left;
    }
};
```

4.Output:


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
nums =
[1,2,3,1]
Output
2
Expected
2

Problem-9

1.Aim:

Given the root of a binary search tree, and an integer k, return the k^{th} smallest value (1-indexed) of all the values of the nodes in the tree.

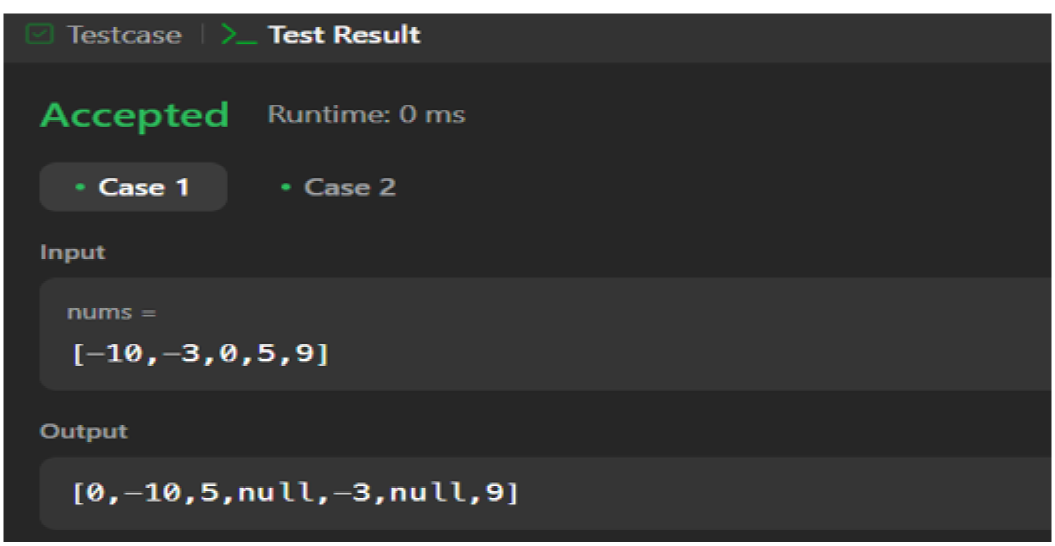
2.Objective:

- Use inorder traversal (sorted order) to efficiently find the kth smallest element.

3.Code:

```
#include <vector>

using namespace std;

class Solution {

public:

TreeNode sortedArrayToBST (vector<int>& nums) {

return buildBST (nums, 0, nums.size() - 1);

}
```

4.Output:



Problem-10

1.Aim:

You are given a perfect binary tree where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
struct Node {
int val;
Node *left;
Node *right;
Node *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.
Initially, all next pointers are set to NULL.

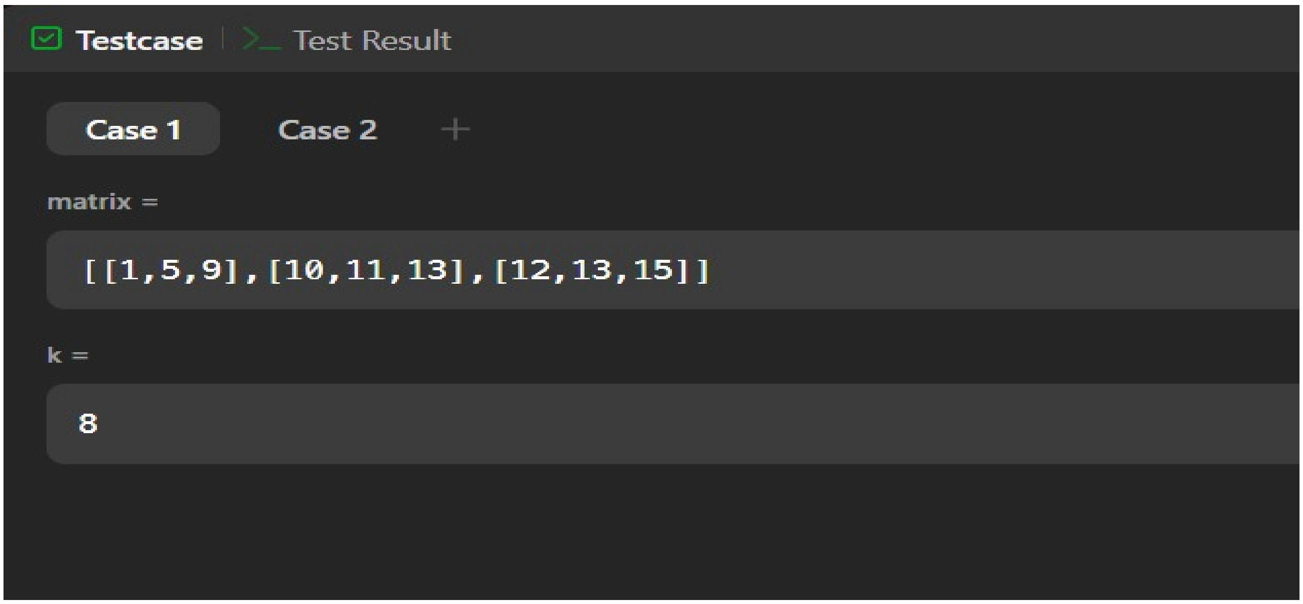
2.Objective:

- Understand tree linking using level pointers

3.Code:

```
class Solution {

public:

bool isMirror (TreeNode t1, TreeNode t2) {

if (!t1 && !t2) return true;

if (!t1 || 1t2) return false;

return (t1->val == t2->val) && isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);
```

4.Output:


Testcase
Test Result
Case 1
Case 2
matrix =
[[1,5,9],[10,11,13],[12,13,15]]
k =
8