



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment-6

Name: Aayushi Sinha

Branch: BE-IT

Semester: 6th

Subject Name: AP LAB-II

UID: 22BET10268

Section/Group: 22BET_IOT-702/B

Date of Performance: 7/03/25

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:

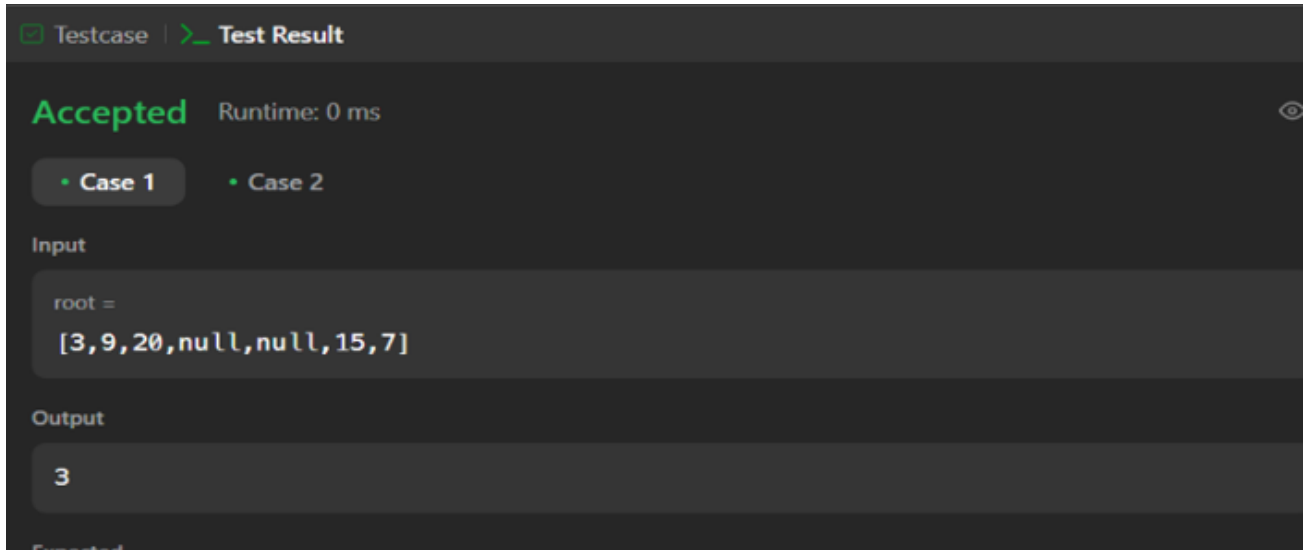
```
class Solution {  
  
public:  
  
    int maxDepth (TreeNode* root) {  
  
        if (root == nullptr)  
  
            return 0;  
  
        int leftDepth = maxDepth (root->left);  
  
        int rightDepth = maxDepth (root->right);  
  
        return 1 + max(leftDepth, rightDepth);  
  
 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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

4.Output:





Problem-2

1.Aim:

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

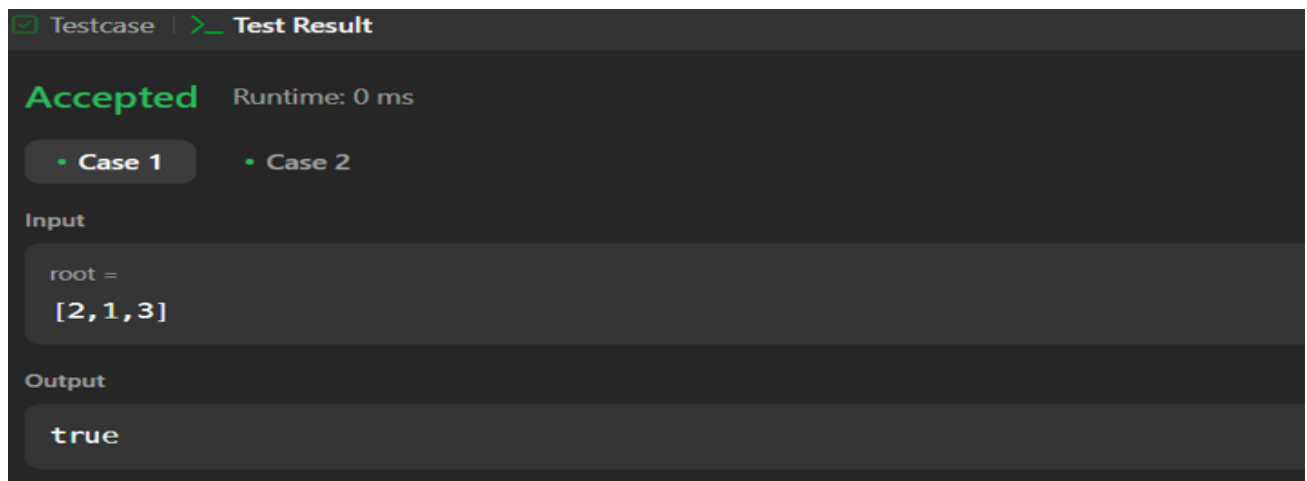
2.Objective:

- 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).

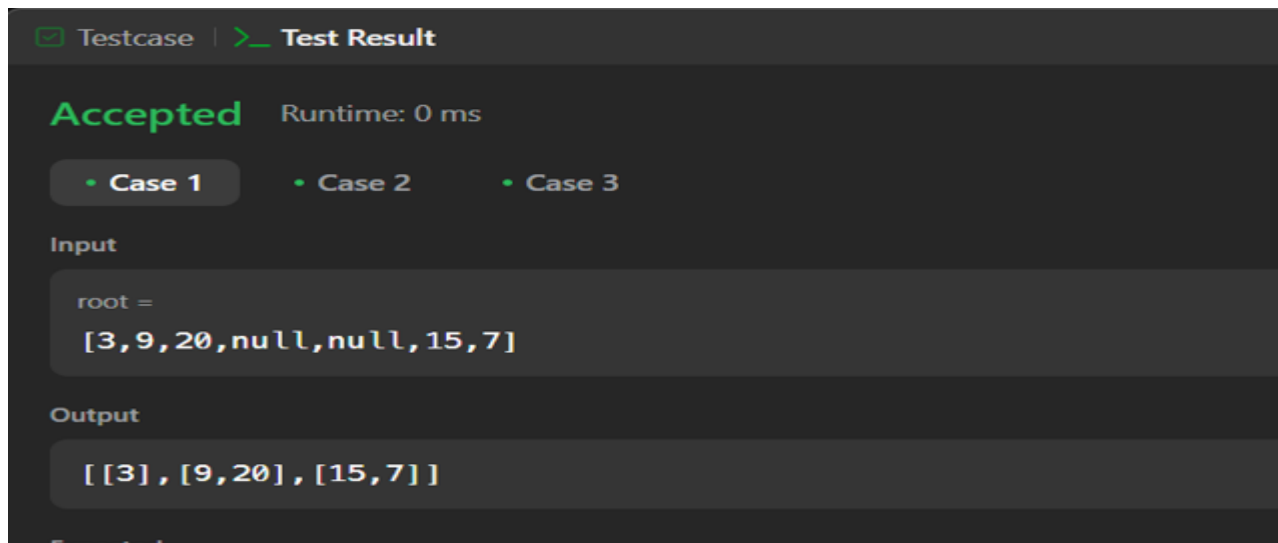
2.Objective:

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

4.Output:



The screenshot shows a code execution environment with a dark theme. At the top, there are tabs for 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the word 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are three buttons labeled 'Case 1', 'Case 2', and 'Case 3', with 'Case 1' being selected. Under the 'Input' section, the text 'root =' is followed by the array '[3,9,20,null,null,15,7]'. Under the 'Output' section, the array '[[3],[9,20],[15,7]]' is displayed.



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:

The screenshot shows a code execution interface with a dark theme. At the top, there are two tabs: 'Testcase' (checked) and 'Test Result'. Below the tabs, the word 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two buttons labeled 'Case 1' and 'Case 2'. Under the 'Input' section, the text 'nums =' is followed by the array '[-10, -3, 0, 5, 9]'. Under the 'Output' section, the text 'Output' is followed by the array '[0, -10, 5, null, -3, null, 9]'.

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;  
  
    }  
  
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

4.Output:

A screenshot of a test result interface. At the top, there are two tabs: 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the word 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two buttons labeled 'Case 1' and 'Case 2', with 'Case 1' being the selected one. Under the 'Input' section, the text 'nums =' is followed by the array '[1,2,3,1]'. Under the 'Output' section, the number '2' is displayed. Under the 'Expected' section, the number '2' is displayed.

Testcase | > Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =
[1,2,3,1]

Output

2

Expected

2



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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];
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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;
```

```
}
```

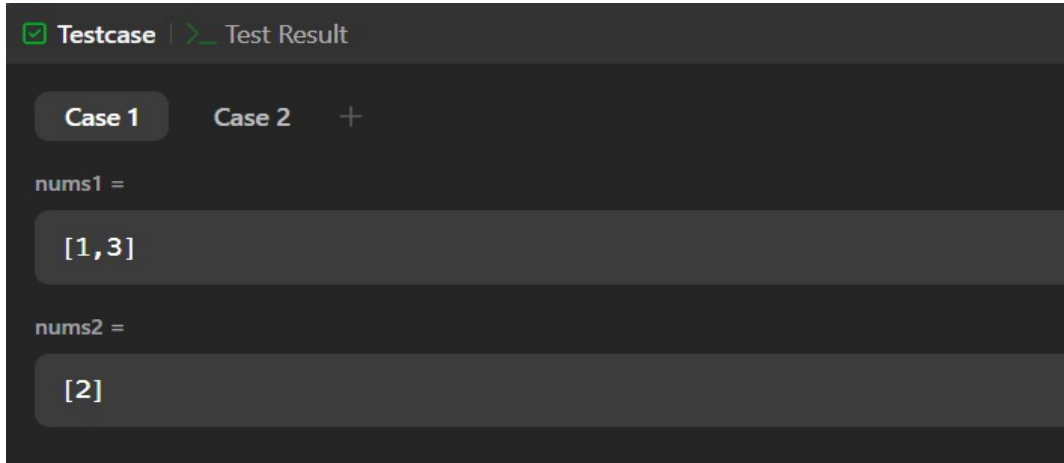
```
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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;  
  
            }  
  
        }  
  
    }  
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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  $\leq$  mid
```

```
            col++; // Move right
```

```
        } else {
```

```
            row--; // Move up
```

```
        }
```

```
    }
```

```
    return count;
```

```
}
```

```
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

4.Output:

☒ Testcase | > Test Result

Case 1 Case 2 +

matrix =

[[1,5,9] , [10,11,13] , [12,13,15]]

k =

8



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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;  
  
    }  
  
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

4.Output:

A screenshot of a web-based test result interface. At the top, there is a navigation bar with a green checkmark icon and the text 'Testcase', followed by a green right-pointing arrow and the text 'Test Result'. Below this, the word 'Accepted' is displayed in large green font, followed by 'Runtime: 0 ms' in a smaller grey font. There are two tabs: 'Case 1' (selected, indicated by a green dot) and 'Case 2' (indicated by a grey dot). Under the 'Input' section, the text 'nums =' is followed by a code block containing '[1,2,3,1]'. The 'Output' section shows the number '2'. The 'Expected' section also shows the number '2'.

Testcase | > Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

```
nums =  
[1,2,3,1]
```

Output

```
2
```

Expected

```
2
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem-9

1.Aim:

Given the root of a binary search tree, and an integer k, return the kth 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:

The screenshot shows a code execution interface with a dark theme. At the top, there are tabs for 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the word 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two buttons labeled 'Case 1' and 'Case 2', both with a small green dot to their left. Below these buttons, the 'Input' section shows 'nums =' followed by the array '[-10, -3, 0, 5, 9]'. The 'Output' section shows the array '[0, -10, 5, null, -3, null, 9]'. The 'null' values in the output are represented by the text 'null'.

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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

4.Output:

```
✓ Testcase | > Test Result

Case 1 Case 2 +

matrix =
[[1,5,9],[10,11,13],[12,13,15]]

k =
8
```