



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment-5

Name: Manvi Chhabra

Branch: BE-IT

Semester: 6th

Subject Name: AP LAB-II

UID: 22BET10189

Section/Group: 22BET_IOT-702/A

Date of Performance: 07/03/25

Subject Code: 22ITP-351

Problem-1

1.Aim: Maximum Depth of Binary Tree 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.

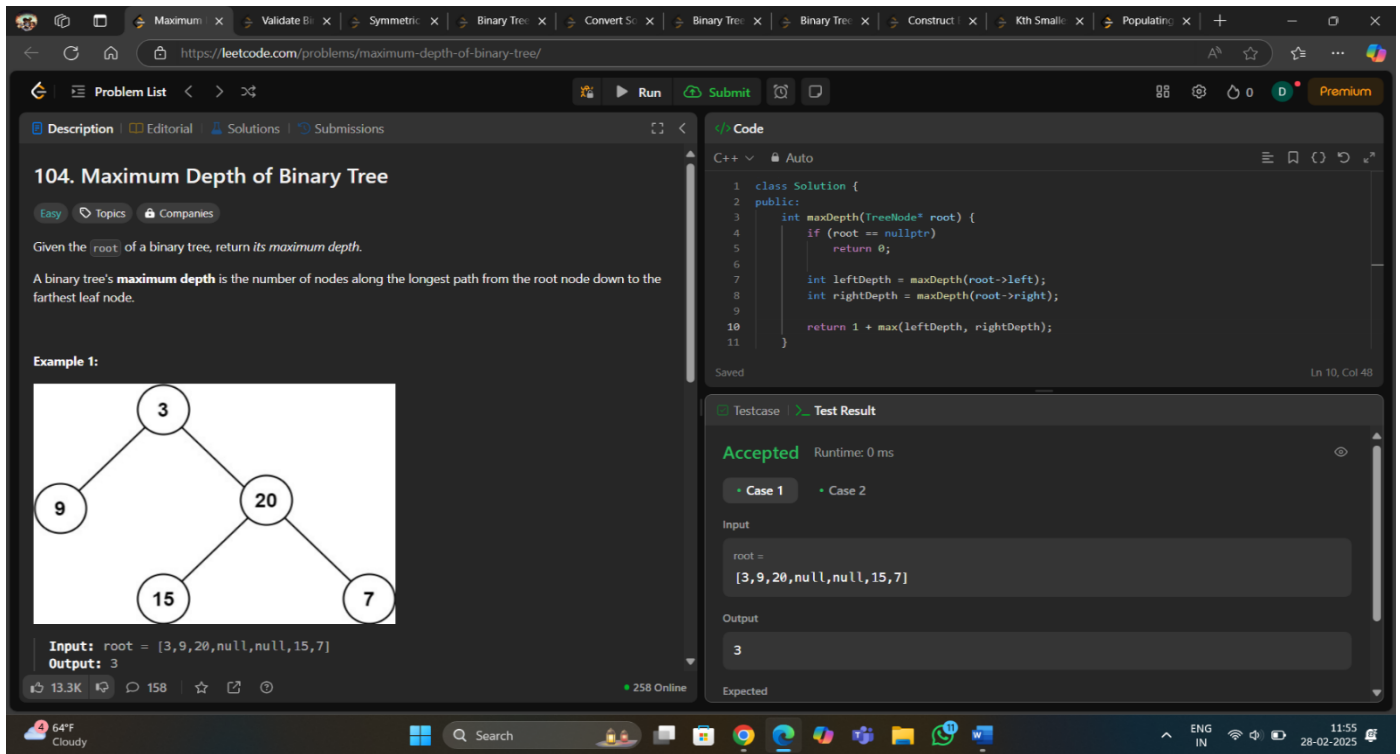
2.Objective:

1. Learn how to recursively or iteratively compute the height (or depth) of a binary tree.
2. Understand the properties of a Binary Search Tree (BST).

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:



104. Maximum Depth of Binary Tree

Easy Topics Companies

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.

Example 1:

```

graph TD
    3((3)) --- 9((9))
    3 --- 20((20))
    20 --- 15((15))
    20 --- 7((7))
  
```

Input: `root = [3,9,20,null,null,15,7]`
Output: `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);
    }
}
  
```

Test Result: Accepted Runtime: 0 ms

Case 1:

Input: `root = [3,9,20,null,null,15,7]`

Output: `3`

Expected: `3`

Problem-2

1. Aim:

- a) **Validate Binary Search Tree :** Given the root of a binary tree, determine if it is a valid binary search tree (BST). A valid BST is defined as follows:
- left subtree of a node contains only nodes with keys less than the node's key.
 - The right subtree of a node contains only nodes with keys greater than the node's key.
 - Both the left and right subtrees must also be binary search trees.

2. Objective:

- Understand mirror properties of a binary tree.
- Learn Breadth-First Search (BFS) using queues.

3. Code:

```

class Solution {

public:

    bool isValidBST(TreeNode* root) {
  
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
return isValidBSTHelper(root, LONG_MIN, LONG_MAX);
```

```
}
```

private:

```
bool isValidBSTHelper(TreeNode* root, long minVal, long maxVal) {
```

```
    if (!root) return true;
```

```
    if (root->val <= minVal || root->val >= maxVal) return false;
```

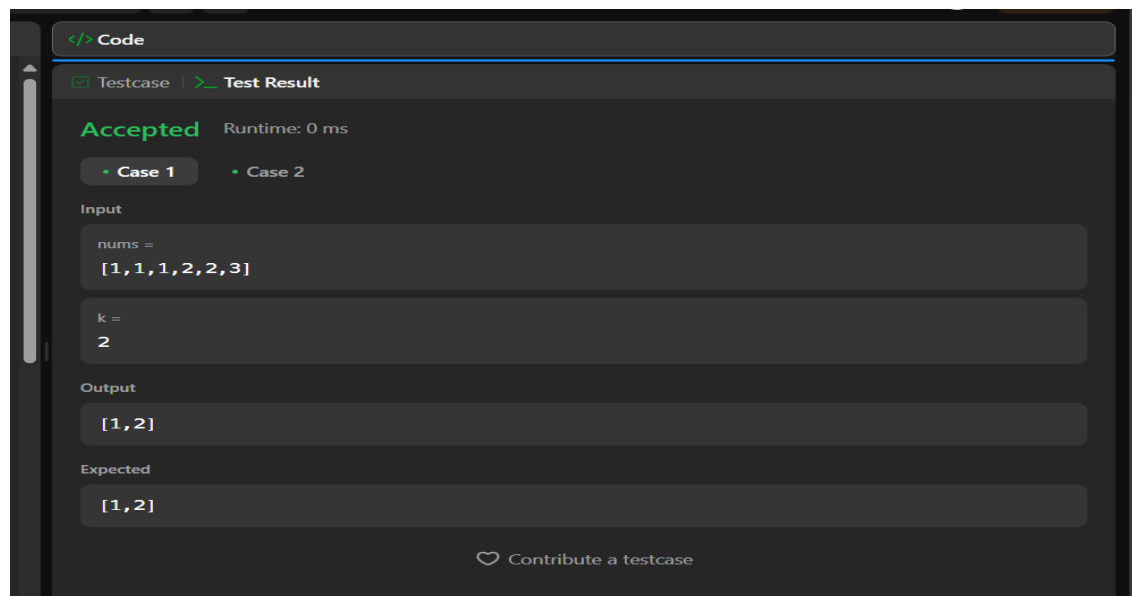
```
    return isValidBSTHelper(root->left, minVal, root->val) &&
```

```
        isValidBSTHelper(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:

- Learn Breadth-First Search (BFS) using queues.
- Learn how to construct a height-balanced BST from a sorted array.

3.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;
```

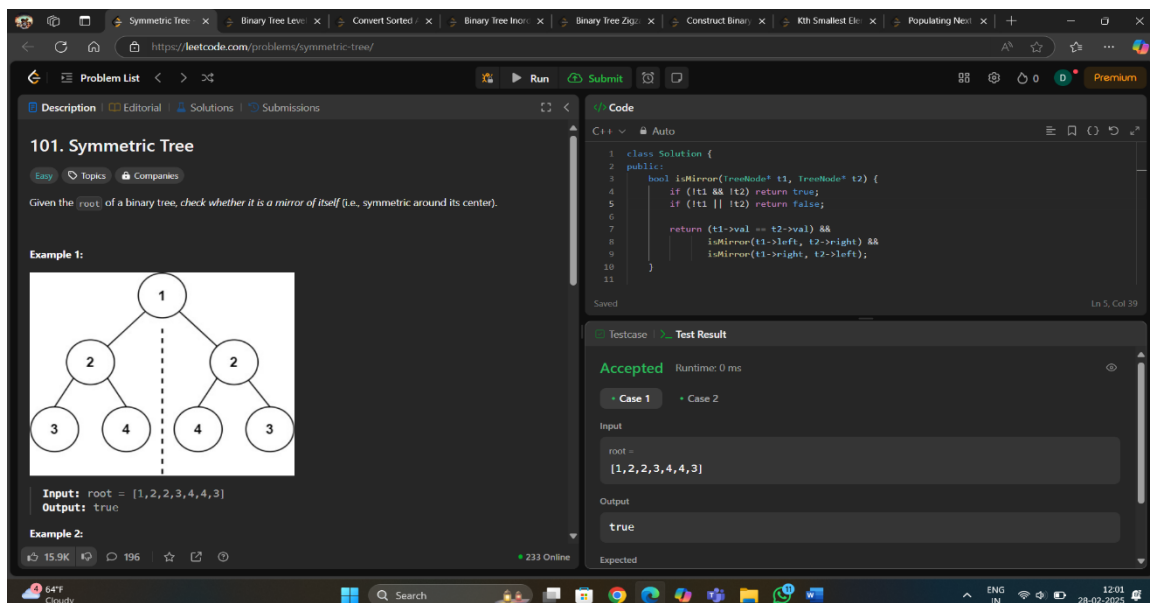
```
    }
```

```
    return checkForSubTrees(root->left, root->right);
```

```
}
```

```
};
```

4.Output:



101. Symmetric Tree

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

Example 1:

```

    1
   / \
  2   2
 / \ / \
3  4 4  3
  
```

Input: root = [1,2,2,3,4,4,3]
Output: true

Example 2:

```

    1
   / \
  2   2
 / \   \
3  4   4
  
```

Input: root = [1,2,2,3,4,4,4]
Output: false

C++ 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);
    }
};
  
```

Test Result: Accepted Runtime: 0 ms

Case 1:

Input: root = [1,2,2,3,4,4,3]
Output: true



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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:

- To find the median of two sorted array
- The overall run time complexity should be $O(\log (m+n))$.

3.Code:

```
class Solution {  
  
public:  
  
    double findMedianSortedArrays(vector<int> &nums1, vector<int> &nums2) {  
  
        int n1 = nums1.size(), n2 = nums2.size();  
  
  
        // Ensure nums1 is the smaller array for simplicity  
  
        if (n1 > n2)  
            return findMedianSortedArrays(nums2, nums1);  
  
  
        int n = n1 + n2;  
  
        int left = (n1 + n2 + 1) / 2; // Calculate the left partition size  
  
        int low = 0, high = n1;  
  
  
        while (low <= high) {  
  
            int mid1 = (low + high) >> 1; // Calculate mid index for nums1  
  
            int mid2 = left - mid1; // Calculate mid index for nums2  
  
  
  
            int l1 = INT_MIN, l2 = INT_MIN, r1 = INT_MAX, r2 = INT_MAX;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// Determine values of l1, l2, r1, and r2

if (mid1 < n1)

    r1 = nums1[mid1];

if (mid2 < n2)

    r2 = nums2[mid2];

if (mid1 - 1 >= 0)

    l1 = nums1[mid1 - 1];

if (mid2 - 1 >= 0)

    l2 = nums2[mid2 - 1];

if (l1 <= r2 && l2 <= r1) {

    // The partition is correct, we found the median

    if (n % 2 == 1)

        return max(l1, l2);

    else

        return ((double)(max(l1, l2) + min(r1, r2))) / 2.0;

}

else if (l1 > r2) {

    // Move towards the left side of nums1

    high = mid1 - 1;

}

else {

    // Move towards the right side of nums1

    low = mid1 + 1;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}  
  
}  
  
return 0; // If the code reaches here, the input arrays were not sorted.  
  
}  
  
};
```

4.Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

nums1 =
[1,3]

nums2 =
[2]

Output

2.00000

Expected

2.00000



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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:

- k^{th} smallest element in the sorted order, not the k^{th} distinct element.

3.Code:

```
class Solution {  
  
public:  
  
    int kthSmallest(vector<vector<int>>& matrix, int z) {  
  
        int n = matrix.size(), m = matrix[0].size();  
  
        int a[n*m], k=0;  
  
        for(int i=0; i<n; i++){  
  
            for(int j=0; j<m; j++){  
  
                a[k] = matrix[i][j];  
  
                k++;  
  
            }  
  
        }  
  
        sort(a, a+(n*m));  
  
        return a[z-1];  
  
    }  
  
};
```

4. Output:



Problem-6

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

2.Objective:

- k^{th} smallest element in the sorted order, not the k^{th} distinct element.

3.Code:

```
class Solution {  
  
public:  
  
    int kthSmallest(vector<vector<int>>& matrix, int z) {  
  
        int n = matrix.size(), m = matrix[0].size();  
  
        int a[n*m], k=0;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
for(int i=0; i<n; i++){  
  
    for(int j=0; j<m; j++){  
  
        a[k] = matrix[i][j];  
  
        k++;  
  
    }  
  
}  
  
sort(a, a+(n*m));  
  
return a[z-1];  
  
}  
  
};
```

4.Output:

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

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

k =
8

Output

13

Expected

13

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).

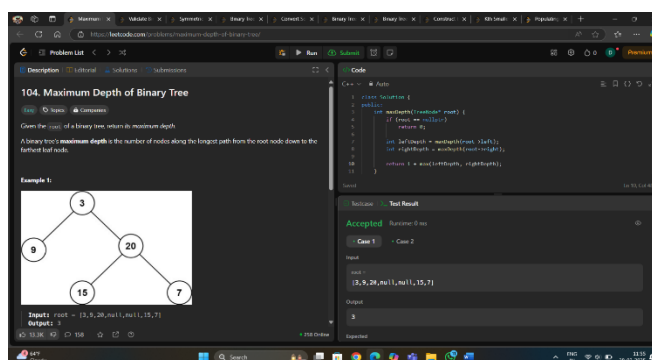
2.Objective:

- Learn how to recursively or iteratively compute the height (or depth) of a binary tree.
- Understand the properties of a Binary Search Tree (BST).

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:





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 mirror properties of a binary tree.
- Learn Breadth-First Search (BFS) using queues.

3.Code:

```
class Solution {

public:

    bool isValidBST(TreeNode* root) {

        return isValidBSTHelper(root, LONG_MIN, LONG_MAX);

    }

private:

    bool isValidBSTHelper(TreeNode* root, long minVal, long maxVal) {

        if (!root) return true;

        if (root->val <= minVal || root->val >= maxVal) return false;

        return isValidBSTHelper(root->left, minVal, root->val) &&

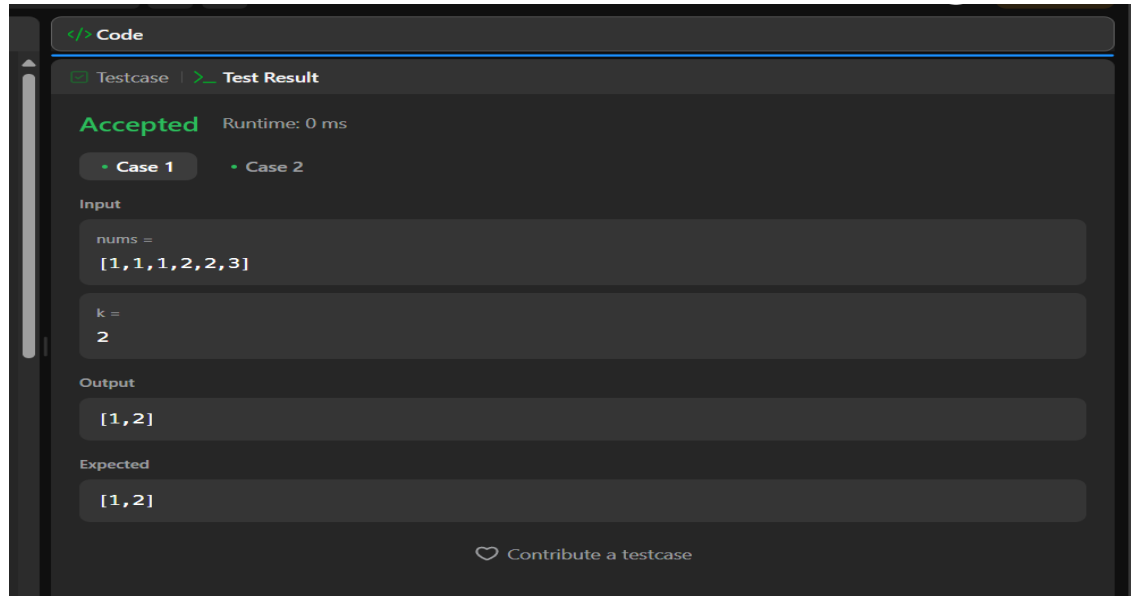
            isValidBSTHelper(root->right, root->val, maxVal);

    }

};
```



4.Output:



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:

- Learn Breadth-First Search (BFS) using queues.
- Learn how to construct a height-balanced BST from a sorted array.

3.Code:

```
class Solution {  
  
public:  
  
    bool checkForSubTrees(TreeNode* p, TreeNode* q) {  
  
        if (p == NULL && q == NULL) {  
  
            return true;  

```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}
```

```
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;
```

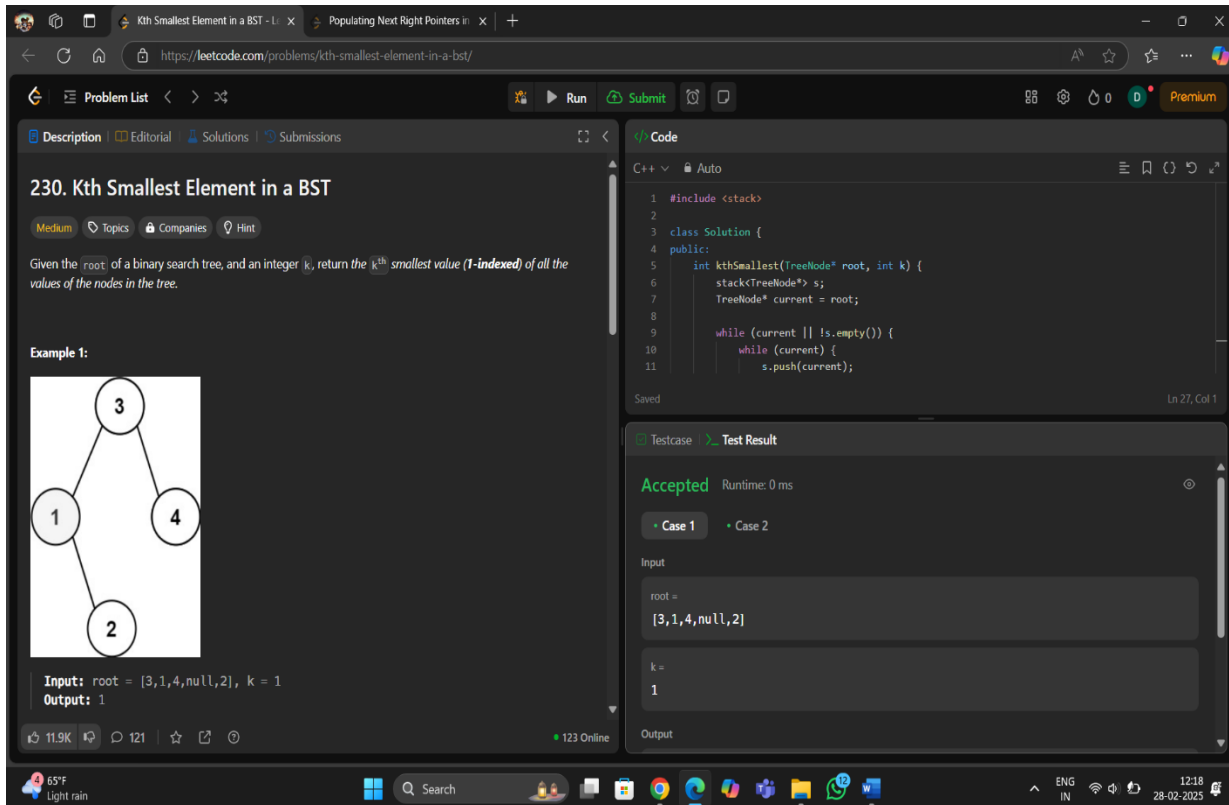
```
}
```

```
return checkForSubTrees(root->left, root->right);
```

```
}
```

```
};
```

4.Output:

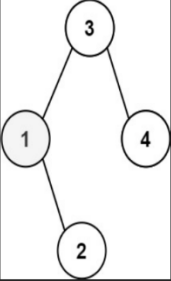


230. Kth Smallest Element in a BST

Medium Topics Companies Hint

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.

Example 1:



```
graph TD
    3((3)) --> 1((1))
    3((3)) --> 4((4))
    1((1)) --> 2((2))
```

Input: root = [3,1,4,null,2], k = 1
Output: 1

Code:

```
1 #include <stack>
2
3 class Solution {
4 public:
5     int kthSmallest(TreeNode* root, int k) {
6         stack<TreeNode*> s;
7         TreeNode* current = root;
8
9         while (current || !s.empty()) {
10             while (current) {
11                 s.push(current);
```

Testcase: Accepted Runtime: 0 ms

Case 1: Case 2

Input:

```
root =
[3,1,4,null,2]
```

k =

```
1
```

Output:

```
1
```

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:



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

- To find the median of two sorted array
- The overall run time complexity should be $O(\log (m+n))$.

3.Code:

```
class Solution {  
  
public:  
  
double findMedianSortedArrays(vector<int> &nums1, vector<int> &nums2) {  
  
    int n1 = nums1.size(), n2 = nums2.size();  
  
    // Ensure nums1 is the smaller array for simplicity  
  
    if (n1 > n2)  
        return findMedianSortedArrays(nums2, nums1);  
  
    int n = n1 + n2;  
  
    int left = (n1 + n2 + 1) / 2; // Calculate the left partition size  
  
    int low = 0, high = n1;  
  
    while (low <= high) {  
  
        int mid1 = (low + high) >> 1; // Calculate mid index for nums1  
  
        int mid2 = left - mid1; // Calculate mid index for nums2  
  
        int l1 = INT_MIN, l2 = INT_MIN, r1 = INT_MAX, r2 = INT_MAX;  
  
        // Determine values of l1, l2, r1, and r2
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
if (mid1 < n1)

    r1 = nums1[mid1];

if (mid2 < n2)

    r2 = nums2[mid2];

if (mid1 - 1 >= 0)

    l1 = nums1[mid1 - 1];

if (mid2 - 1 >= 0)

    l2 = nums2[mid2 - 1];


if (l1 <= r2 && l2 <= r1) {

    // The partition is correct, we found the median

    if (n % 2 == 1)

        return max(l1, l2);

    else

        return ((double)(max(l1, l2) + min(r1, r2))) / 2.0;

}

else if (l1 > r2) {

    // Move towards the left side of nums1

    high = mid1 - 1;

}

else {

    // Move towards the right side of nums1

    low = mid1 + 1;

}

}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
return 0; // If the code reaches here, the input arrays were not sorted.
```

```
}
```

```
};
```

4.Output:

The screenshot shows a web browser displaying the LeetCode problem "116. Populating Next Right Pointers in Each Node". The problem description states: "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`." Example 1 shows a binary tree with root 1, children 2 and 3, and grandchildren 4, 5, 6, 7. Red arrows indicate the next pointers: 1 to NULL, 2 to 3, 3 to NULL, 4 to 5, 5 to 6, 6 to 7, and 7 to NULL.

The code editor shows the following C++ code:

```
1 #include <queue>
2
3 class Solution {
4 public:
5     Node* connect(Node* root) {
6         if (!root) return nullptr;
7         queue<Node*> q;
8         q.push(root);
9         while (!q.empty()) {
10             Node* cur = q.front();
11             if (cur->left) q.push(cur->left);
12             if (cur->right) q.push(cur->right);
13             cur->next = q.front();
14             q.pop();
15         }
16     }
17 }
```

The test result shows "Accepted" with a runtime of 0 ms. The input is `root = [1,2,3,4,5,6,7]` and the output is `[1, #, 2, 3, #, 4, 5, 6, 7, #]`.