# Experiment 6

**Student Name: Sujal Nag**

**Branch: CSE** **Semester: 6th Subject: AP Lab**

**Aim:**

**Problem 1.2.1:**

**Binary\_Tree\_Inorder\_Traversal**

**UID: 22BCS16984**

**Section: 22BCS\_IOT-641(A) DOP: 15.01.2025**

**Subject Code:22CSP-351**

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

**Problem 1.2.2:** [**Kth Smallest Element in a BST**](https://leetcode.com/problems/kth-smallest-element-in-a-bst/)

* **Problem Statement:** 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*.

**Problem 1.2.3:** [**Maximum Depth of Binary Tree**](https://leetcode.com/problems/maximum-depth-of-binary-tree/)

* **Problem Statement**: 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.

**Problem 1.2.4:** [**Symmetric Tree**](https://leetcode.com/problems/symmetric-tree/)

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

# Algorithm:

1. Initialize an empty hash map (dict).
2. Iterate through the nums array:
   * For each element num, calculate the complement: complement = target - num.
   * Check if the complement exists in the hash map:
     + If it does, return the indices of the complement and the current number.
     + If it doesn't, add the current number and its index to the hash map.
3. Return the indices of the two numbers that add up to the target.

**Code: 1.2.1**

**import java.util.\*;**

**// Definition for a binary tree node**

**class TreeNode {**

**int val;**

**TreeNode left, right;**

**TreeNode(int val) {**

**this.val = val;**

**this.left = null;**

**this.right = null;**

**}**

**}**

**class Solution {**

**public List<Integer> inorderTraversal(TreeNode root) {**

**List<Integer> result = new ArrayList<>();**

**Stack<TreeNode> stack = new Stack<>();**

**TreeNode current = root;**

**while (current != null || !stack.isEmpty()) {**

**while (current != null) {**

**stack.push(current);**

**current = current.left;**

**}**

**current = stack.pop();**

**result.add(current.val);**

**current = current.right;**

**}**

**return result;**

**}**

**}**

**// Main function to test the solution**

**public class Main {**

**public static void main(String[] args) {**

**TreeNode root = new TreeNode(1);**

**root.right = new TreeNode(2);**

**root.right.left = new TreeNode(3);**

**Solution sol = new Solution();**

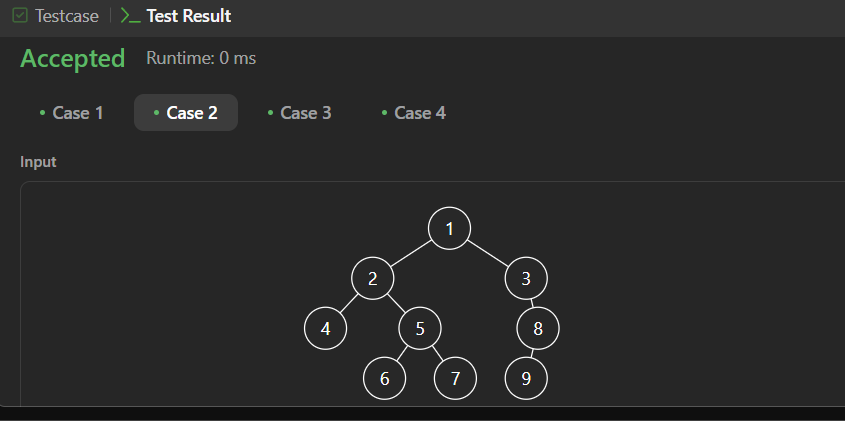
**List<Integer> result = sol.inorderTraversal(root);**

**System.out.println(result); // Output: [1, 3, 2]**

**}**

**}**

# Output:



**CODE: 1.2.2**

**import java.util.\*;**

**class Solution {**

**public int kthSmallest(int[][] matrix, int k) {**

**PriorityQueue<int[]> minHeap = new PriorityQueue<>(Comparator.comparingInt(a -> a[0]));**

**int n = matrix.length;**

**// Insert the first element of each row into the heap**

**for (int i = 0; i < n; i++) {**

**minHeap.offer(new int[]{matrix[i][0], i, 0});**

**}**

**int result = 0;**

**while (k-- > 0) {**

**int[] smallest = minHeap.poll(); // Get the smallest element**

**result = smallest[0];**

**int row = smallest[1], col = smallest[2];**

**// If the next column exists in the same row, push it into the heap**

**if (col + 1 < n) {**

**minHeap.offer(new int[]{matrix[row][col + 1], row, col + 1});**

**}**

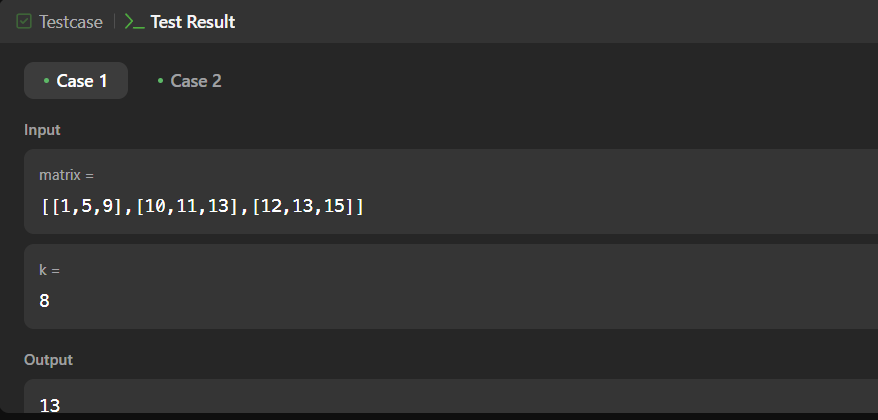
**}**

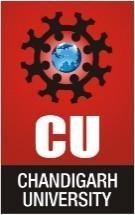
**return result;**

**}**

**}**

**OUTPUT:**

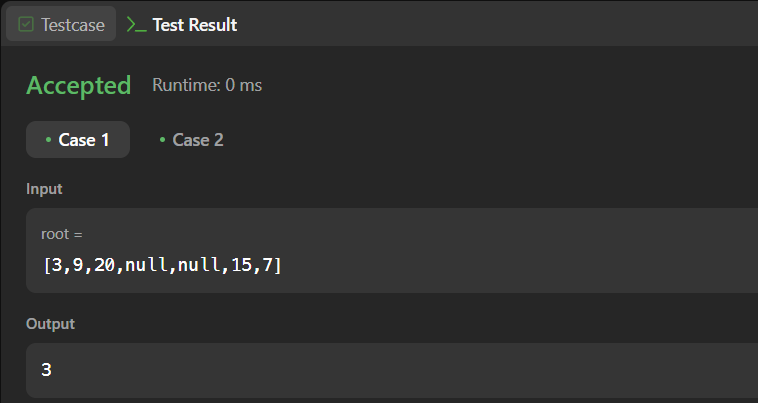
****



**CODE: 1.2.3**



**OUTPUT**

****