Experiment 1.2

Student Name: Nikhil Sharma
UID:22BCS15209
Branch:CSE
Section/Group:640/B
Semester: 6
Date of Performance: 16/01/25
Subject Name:Advance Programming -2
Subject Code: 22CSH-351

Aim 1: : You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n

Objective : Return indices of the two numbers such that they add up to target.

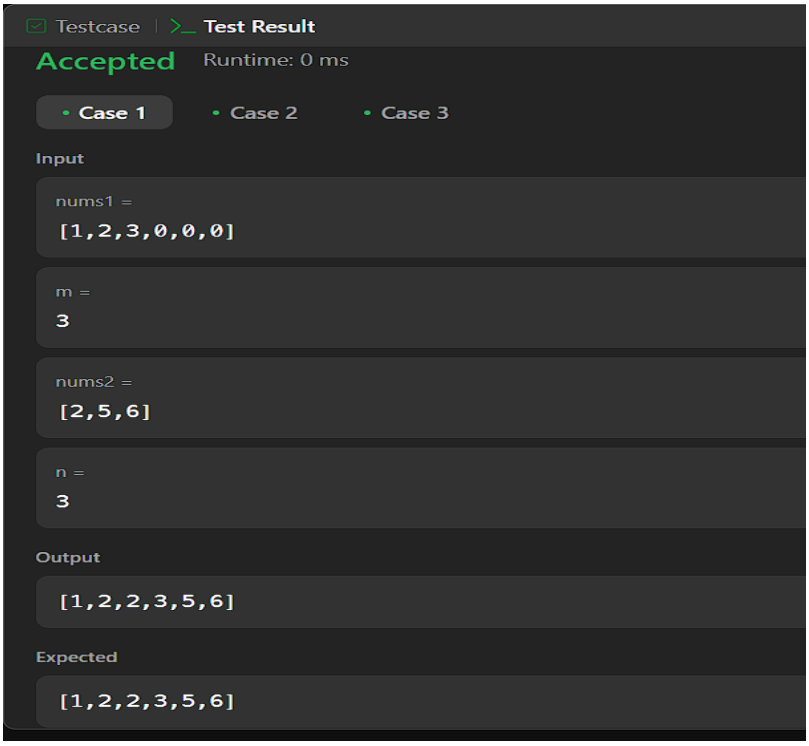
Code:

```
class Solution {
public:
   void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
      vector<int>arr;
      int i=0,j=0;
      while(i<m && j<n){
         if(nums1[i]<nums2[j]){
            arr.push_back(nums1[i]);
            i++;
         }
         else{
            arr.push_back(nums2[j]);
            j++;
         }
      }
      if(i<m){
      while(i<m){
       arr.push_back(nums1[i]);
            i++;
      }
      }
      else{
         while(j<n){
            arr.push_back(nums2[j]);
            j++;
```

```
        }
      }
      for(int i=0;i<m+n;i++){
         nums1[i]=arr[i];
      }
      arr.clear();

   }
};
```

OUTPUT:



Aim 2: A peak element is an element that is strictly greater than its neighbors.

Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.

You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in O(log n) time.

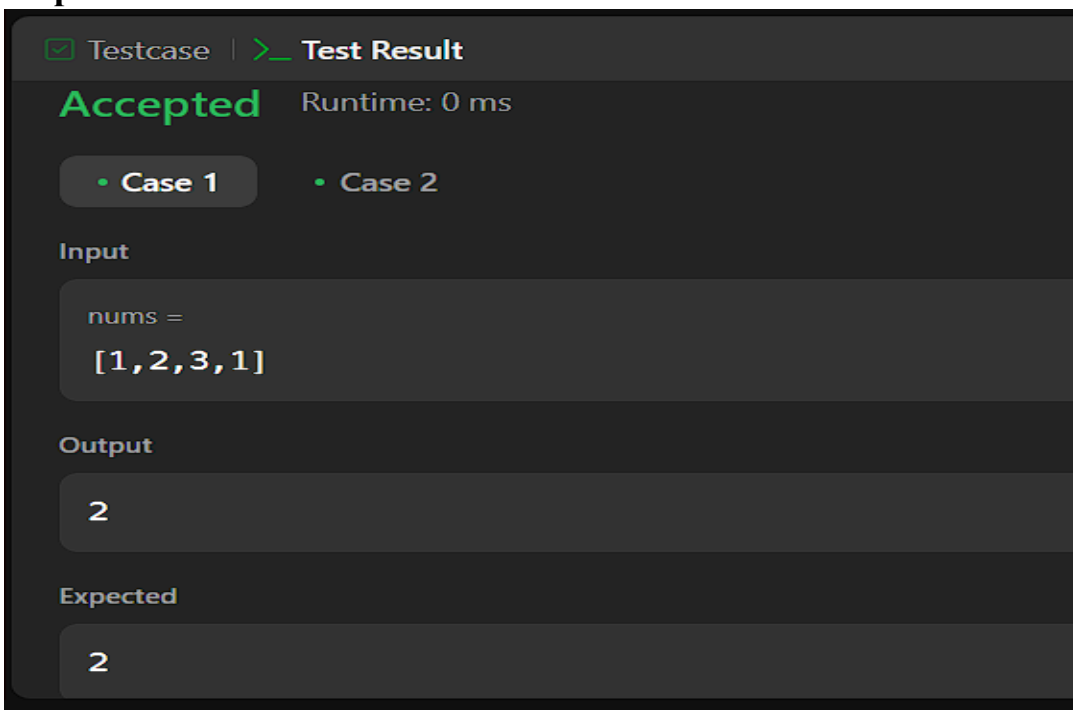
Code:

```
class Solution {
public:
   int findPeakElement(vector<int>& nums) {
      int n=nums.size();
```

```
        if(n==1) return 0;
        if(nums[0]>nums[1]) return 0;
        if(nums[n-1]>nums[n-2]) return n-1;
        int low=1,high=n-2;
        while(low<=high){
            int mid=(low+high)/2;
            if(nums[mid]>nums[mid-1] && nums[mid]>nums[mid+1]) return mid;
            else if(nums[mid]>=nums[mid-1] && nums[mid]<=nums[mid+1]) low=mid+1;
            else if(nums[mid]<=nums[mid-1] && nums[mid]>=nums[mid+1]) high=mid-1;

            // FOR MULTIPLE PEAKS
            else low=mid+1;
        }
        return -1;
    }
};
```

Output:



Aim - Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.
Integers in each column are sorted in ascending from top to bottom.

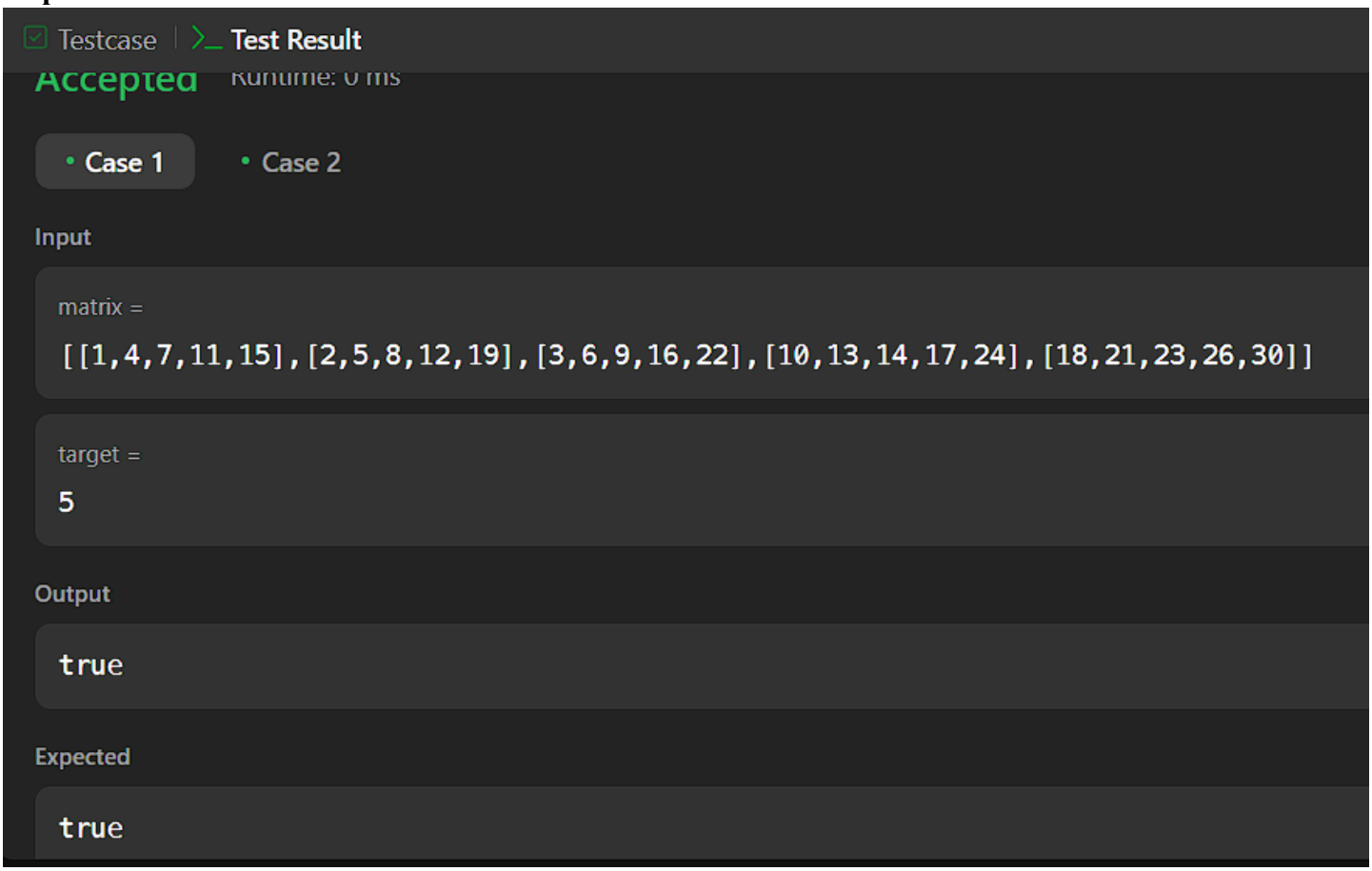
code -

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m= matrix.size();
```

```
        int n= matrix[0].size();
        int i=0,j=n-1;
        while(i<m &&j>=0){

            if(matrix[i][j]>target){
                j--;
            }
            else if(matrix[i][j]<target){
                i++;
            }
            else{
                return true;
            }
        }
        return false;
    }
};
```

output :



Learning Outcomes:

1. Understanding Algorithms – Learn the principles and mechanics behind various searching and sorting algorithms.
2. Time and Space Complexity – Analyze and compare the efficiency of different algorithms.
3. Implementation Skills – Gain hands-on experience in coding and optimizing search and sort functions.
4. Real-World Applications – Apply searching and sorting techniques to practical problems in computing.