



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

## Experiment-5

**Name: Yashraj Yadav**

**Branch: BE-IT**

**Semester: 6**

**Subject Name: Advanced Programming Lab-2**

**UID: 22BET10063**

**Section/Group: 22BET\_702-B**

**Date of Performance: 21-02-25**

**Subject Code: 22ITP-351**

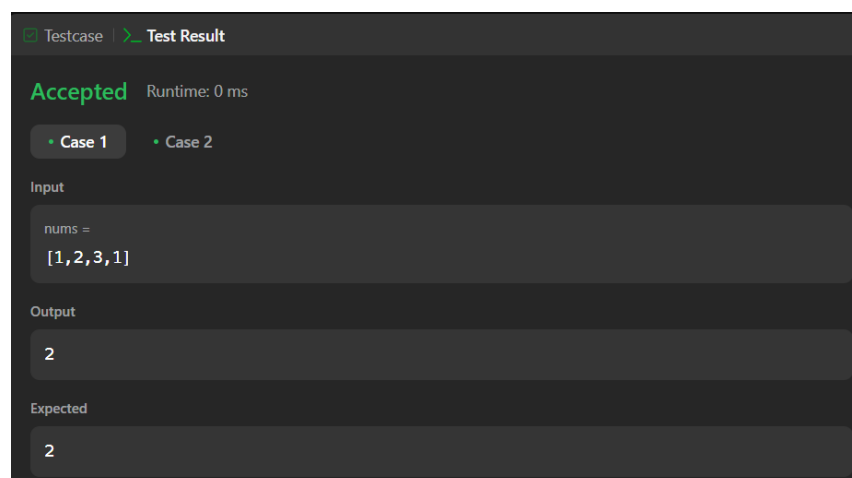
### **Problem 1.** Merge Sorted Arrays.

#### **Code:**

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int n= nums.size();
        int s=0;
        int e=n-1;

        while(s<e){
            int m = s + (e-s) / 2;
            if(nums[m] > nums[m+1]){
                e = m;
            }
            else{
                s = m +1;
            }
        }
        return s;
    }
};
```

#### **Output:**



**Problem 2.** Given an array of intervals where  $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$ , merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

### Code:

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        int n = intervals.size();
        sort(intervals.begin(), intervals.end());
        vector<vector<int>> output;
        for(auto interval : intervals){
            if(output.empty() || output.back()[1] < interval[0]){
                output.push_back(interval);
            }
            else{
                output.back()[1] = max(output.back()[1], interval[1]);
            }
        }
        return output;
    }
};
```

### Output:

☒ Testcase | ☒ Test Result

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

intervals =  
[[1,3],[2,6],[8,10],[15,18]]

Output

[[1,6],[8,10],[15,18]]

Expected

[[1,6],[8,10],[15,18]]

**Problem 3.** There is an integer array `nums` sorted in ascending order (with distinct values).

**Code:**

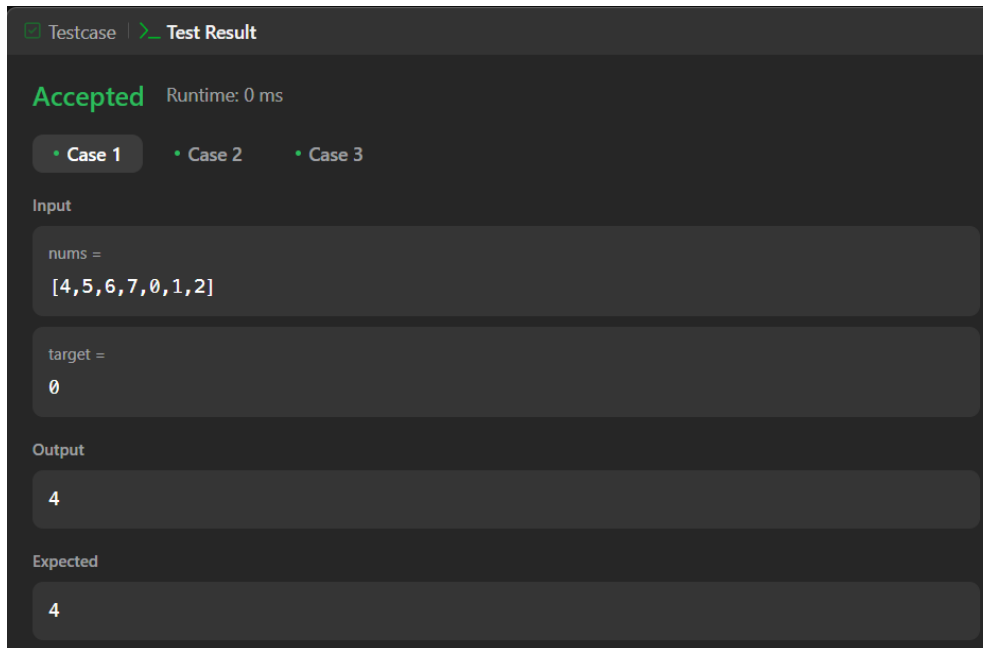
```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left = 0;
        int right = nums.size() - 1;

        while (left <= right) {
            int mid = (left + right) / 2;

            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] >= nums[left]) {
                if (nums[left] <= target && target <= nums[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            } else {
                if (nums[mid] <= target && target <= nums[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }

        return -1;
    }
};
```

## Output:



**Problem 4.** 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(), n = m ? matrix[0].size() : 0, r = 0, c = n - 1;
        while (r < m && c >= 0) {
            if (matrix[r][c] == target) {
                return true;
            }
            matrix[r][c] > target ? c-- : r++;
        }
        return false;
    }
};
```

## Output:

Testcase
Test Result

Accepted Runtime: 4 ms

• Case 1
• Case 2

Input

matrix =  
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]

target =  
5

Output

true

Expected

true

**Problem 5.** Given an integer array nums and an integer k, return the kth largest element in the array.

## Code:

```
class Solution {
public:
    int findKthLargest(std::vector<int>& nums, int k) {
        std::sort(nums.begin(), nums.end(), std::greater<int>());
        return nums[k-1];
    }
};
```

## Output:

Testcase
Test Result

Accepted Runtime: 0 ms

• Case 1
• Case 2

Input

nums =  
[3,2,1,5,6,4]

k =  
2

Output

5

Expected

5

**Problem 6.** Given an  $n \times n$  matrix where each of the rows and columns is sorted in ascending order, return the  $k$ th smallest element in the matrix.

**Code:**

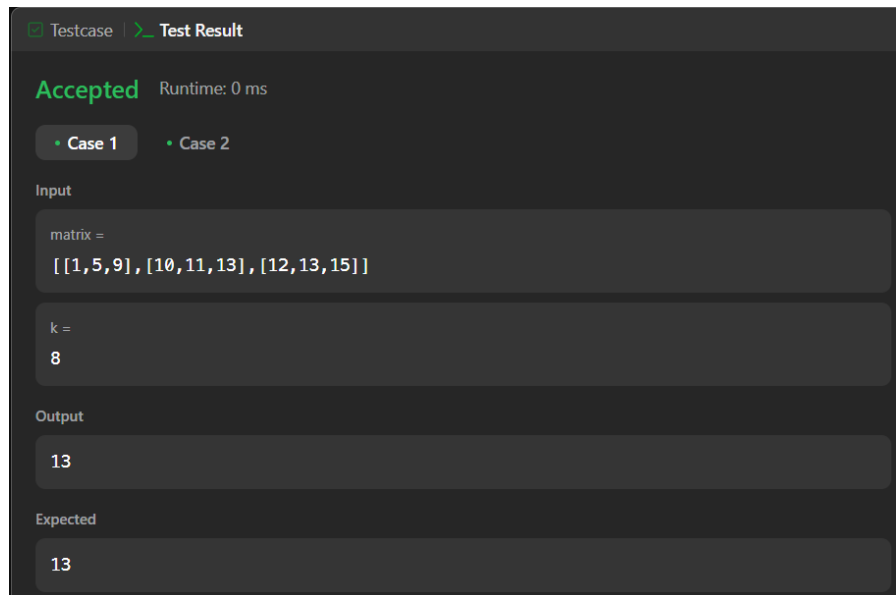
```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {

        int n= matrix.size();
        int l=matrix[0][0];
        int h = matrix[n-1][n-1];
        int mid;
        int count;
        while(l<h)
        {
            count=0;
            mid = l + (h-l)/2;

            for(int i=0;i<n;i++)
            {
                count += upper_bound(matrix[i].begin() , matrix[i].end(),mid) - matrix[i].begin();
            }
            if(count<k)
            {
                l =mid+1;
            }
            else
            {
                h=mid;
            }
        }

        return l;
    }
};
```

## Output:



**Problem 7.** Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return the median of the two sorted arrays.

## Code:

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

        int n = nums1.size();
        int m = nums2.size();

        vector<int> num(n+m);
        int i=0, j=0;
        int k=0;
        while(i<n && j<m){
            if(nums1[i] < nums2[j]){
                num[k++] = nums1[i++];
            }
            else{
                num[k++] = nums2[j++];
            }
        }
        while(i<n){num[k++] = nums1[i++];}
        while(j<m){num[k++] = nums2[j++];}

        int median = (n+m)/2;
```



# DEPARTMENT OF

## COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// cout<<median<<endl;
double ans = 0;
if((n+m)%2==0){
    ans = (num[median]+num[median-1])/2.0;
    // cout<<num[median]<< num[median-1]<<endl;
}
else{
    ans=num[median];
}
return ans;
}
};
```

**Output:**

☒ Testcase | ☒ Test Result

**Accepted** Runtime: 0 ms

• Case 1

• Case 2

Input

nums1 =  
[1,3]

nums2 =  
[2]

Output

2.00000

Expected

2.00000