



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment-5

Name: Gurpreet Yadav
Branch: BE-IT
Semester: 6th
Subject Name: AP LAB-II

UID: 22BET10336
Section/Group: 22BET_IOT-702/A
Date of Performance: 21/02/25
Subject Code: 22ITP-351

Problem-1

1.Aim:

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.

2.Objective:

- Merge two sorted integer arrays, nums1 and nums2, into a single sorted array.
- Store the merged result in nums1, which has sufficient space (size m + n).
- Maintain non-decreasing order in the merged array.
- Perform the merge operation in-place without using extra space.
- Efficiently handle elements from both arrays while merging.

3.Code:

```
class Solution {  
  
public:  
  
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {  
  
        int i=m-1 , j=n-1 ;  
  
        while(i>=0 && j>=0 ){  
  
            if(nums1[i]>=nums2[j]){  
  
                nums1[i+j+1]=nums1[i] ;  
  
                i-- ;  
  
            }else{  
  
                nums1[i+j+1]=nums2[j];
```

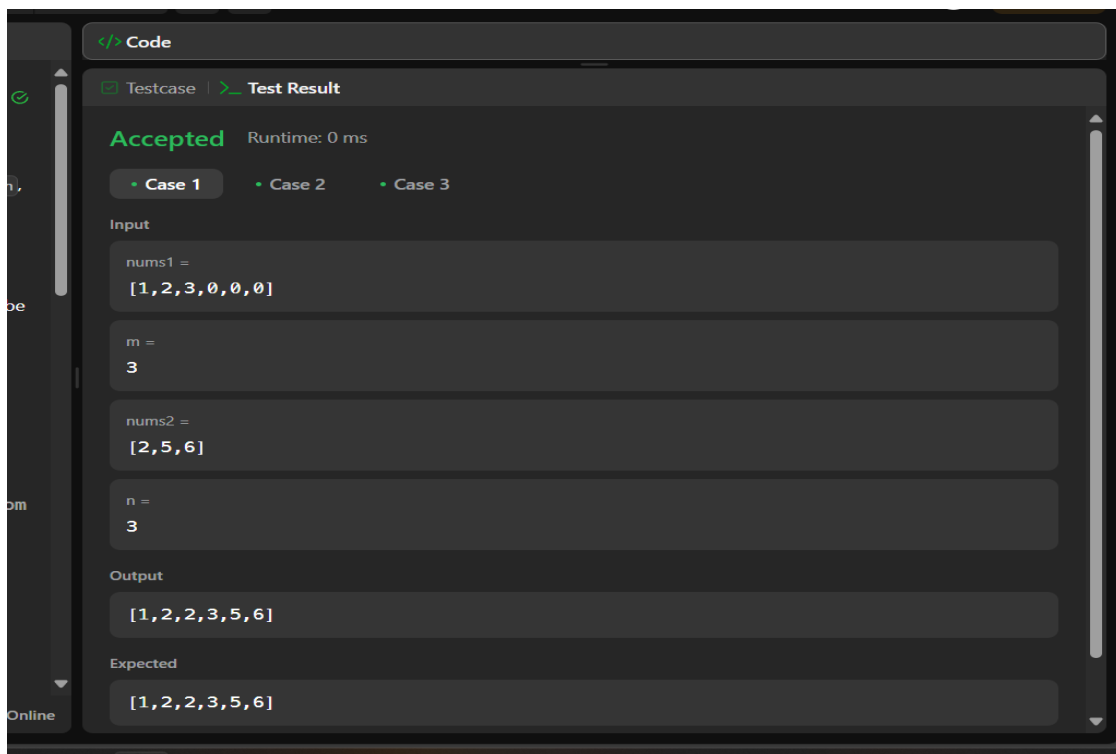


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        j-- ;  
    }  
}  
  
while(j>=0){  
    nums1[j]=nums2[j];  
    j-- ;  
}  
}  
};
```

4.Output:





Problem-2

1.Aim:

Given an integer array nums and an integer k, return *the k most frequent elements*. You may return the answer in any order.

2.Objective:

- Identify the k most frequent elements in the given integer array nums.
- Return these k elements in any order.
- Efficiently determine element frequencies and extract the top k frequent elements.
- Optimize for performance, ideally using a heap or other efficient data structures.

3.Code:

```
class Solution {  
  
public:  
  
    vector<int> topKFrequent(vector<int>& nums, int k) {  
  
        unordered_map<int, int> ump;  
  
  
        for(int i: nums){  
  
            ump[i]++;  
  
        }  
  
  
        priority_queue<pair<int, int>> pq;  
  
  
        for(auto i: ump){  
  
            pq.push({i.second,i.first});  
  
        }  
    }  
};
```

```
vector<int> res;  
  
while(k--){  
    auto [elem, count] = pq.top();  
    res.push_back(count);  
    pq.pop();  
}  
  
return res;  
}  
};
```

4.Output:

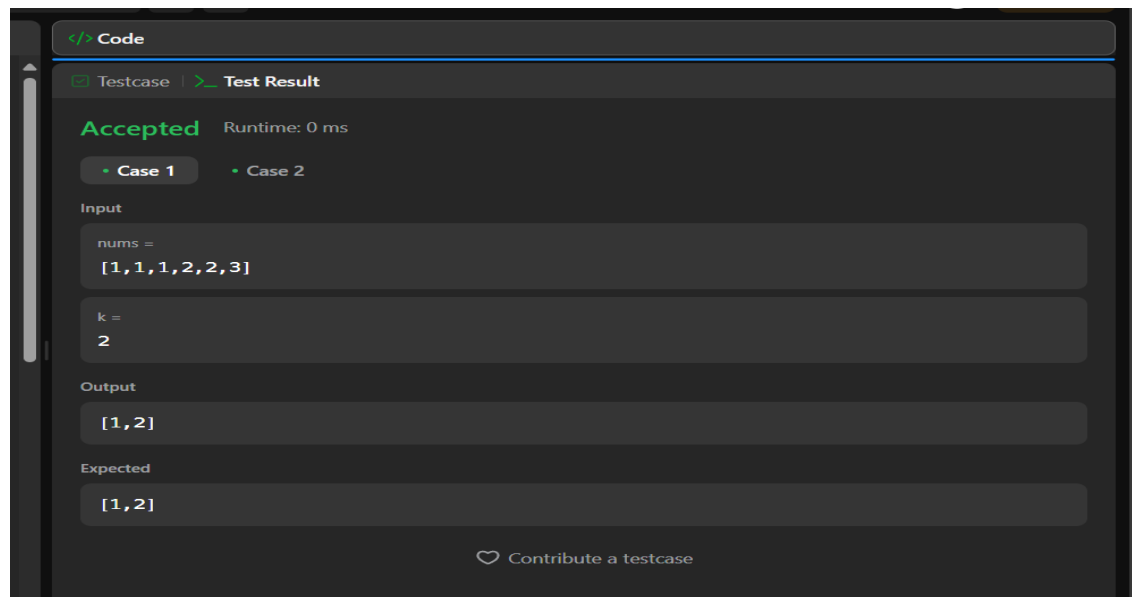


Fig.2: Top K frequent Elements



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem-3

1.Aim:

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.

2.Objective:

- Identify a peak element in the given 0-indexed integer array `nums`.
- A peak element is strictly greater than its neighbors.
- If multiple peaks exist, return the index of any one of them.
- Ensure an efficient approach, ideally using binary search instead of a linear scan.

3.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;  
  
            }  
        }  
    }  
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}  
  
}  
  
return s;  
  
}  
  
};
```

4.Output:

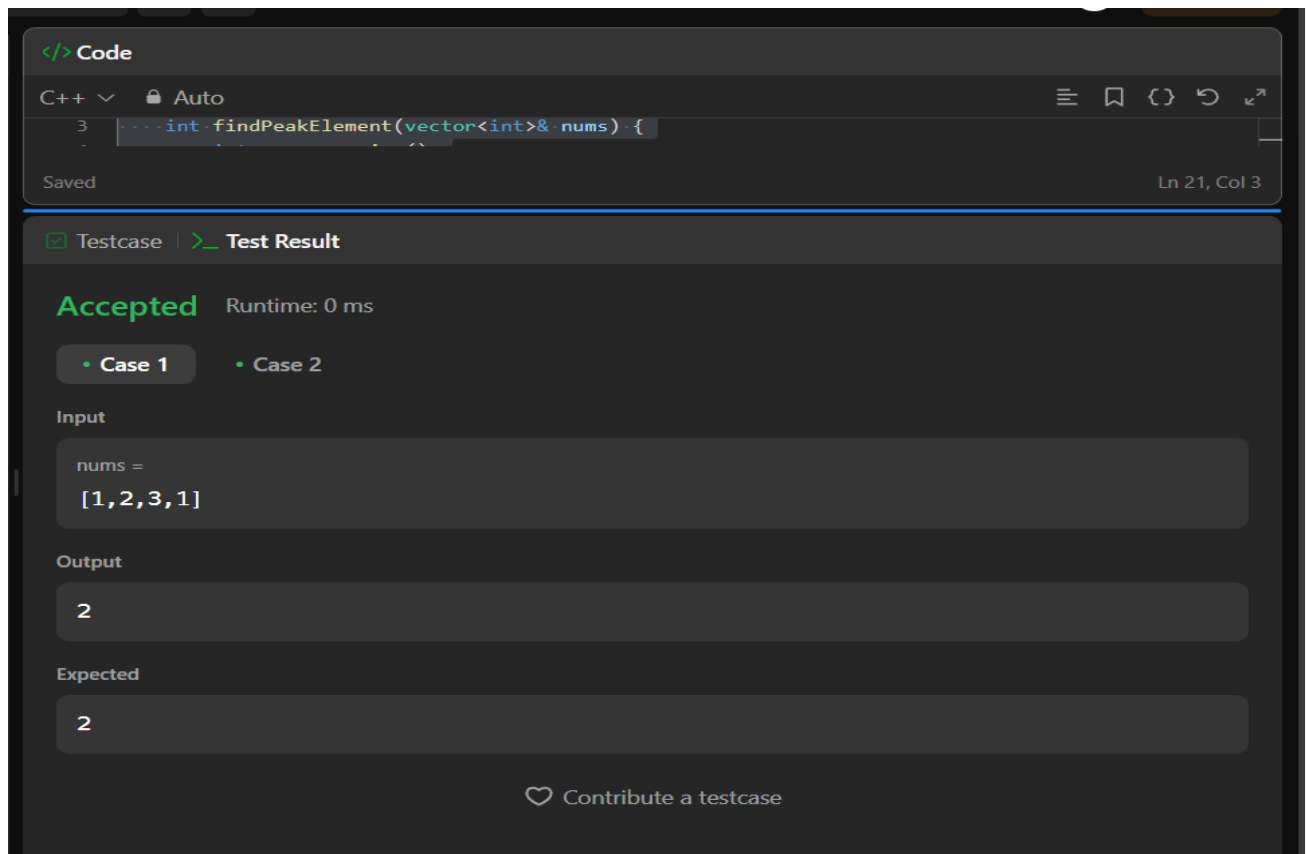


Fig.3:Find Peak Element



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem-4

1.Aim: Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

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:

The screenshot displays a test result interface with a dark theme. At the top, there are two tabs: 'Testcase' (selected) and 'Test Result'. Below the tabs, the status 'Accepted' is shown in green, followed by 'Runtime: 0 ms'. There are two buttons labeled 'Case 1' and 'Case 2', with 'Case 1' being the active one. Under the 'Input' section, there are two text boxes: the first contains 'nums1 =' followed by '[1,3]' on the next line, and the second contains 'nums2 =' followed by '[2]' on the next line. Under the 'Output' section, there is a text box containing '2.00000'. Under the 'Expected' section, there is a text box containing '2.00000'.

Fig.4:Median Of Two Sorted Array



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem-5

1.Aim: 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.

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];  
  
    }  
  
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

4. Output:

A screenshot of a coding problem solution interface. At the top, it says 'Accepted' in green and 'Runtime: 0 ms' in white. Below this, there are two tabs: 'Case 1' (selected) and 'Case 2'. Under the 'Input' section, there are two text boxes: the first contains 'matrix =' followed by a 3x3 array `[[1,5,9],[10,11,13],[12,13,15]]`, and the second contains 'k =' followed by the number '8'. Under the 'Output' section, there is a text box containing the number '13'. Under the 'Expected' section, there is a text box containing the number '13'.

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

Fig.5:kth smallest element