

Experiment 5

Name: Prashant Dubey

Branch: IT

Semester: 6

Subject Name: Advanced Programming Lab-2

UID: 22BET10331

Section/Group: 22BET-701/A

Date of Performance: 14-02-25

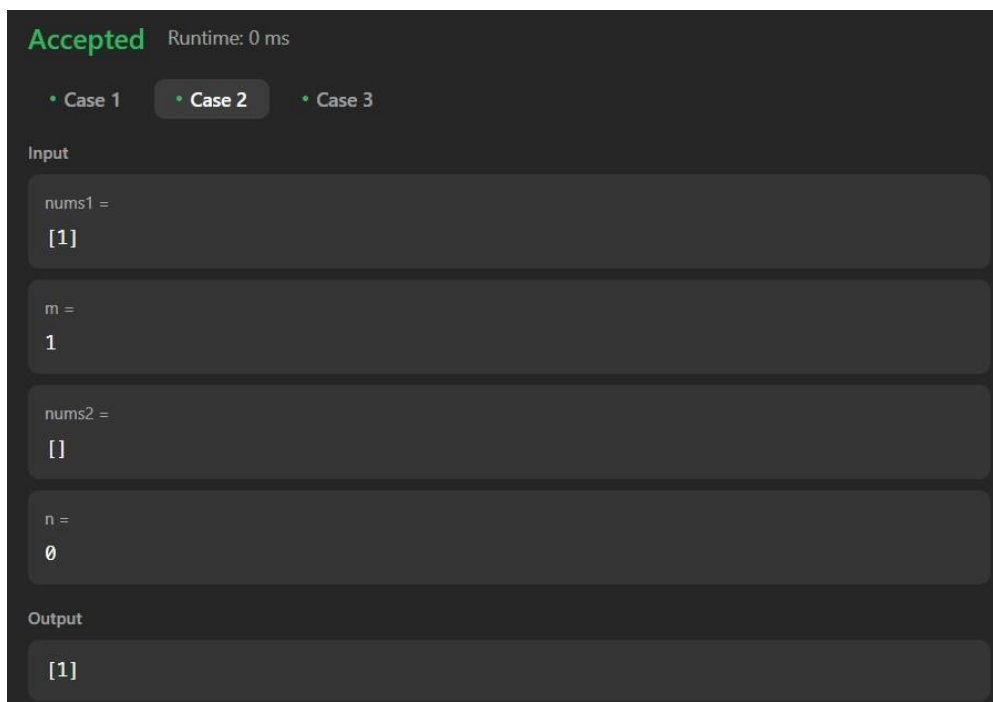
Subject Code: 22ITP-351

Problem 1. Merge Sorted Array - You are given two integer arrays nums1 and nums2, sorted in nondecreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Code:

```
class Solution { public:    void merge(vector<int>& nums1, int m,
vector<int>& nums2, int n) {        for (int j = 0, i = m; j<n; j++){
nums1[i] = nums2[j];            i++;
        }
        sort(nums1.begin(),nums1.end());
    } };
```

Output:



The screenshot shows a code execution interface with a dark theme. At the top, it says "Accepted" in green and "Runtime: 0 ms". Below this, there are three tabs: "Case 1", "Case 2" (which is selected), and "Case 3". Under the "Input" section, there are four input fields: "nums1 =" with the value "[1]", "m =" with the value "1", "nums2 =" with the value "[]", and "n =" with the value "0". Under the "Output" section, there is one output field showing the value "[1]".

First Bad Version - You are a product manager and currently leading a team to develop a

Problem 2.

new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Code:

```
class Solution { public:    int
firstBadVersion(int n) {
    int first = 1;
        int last = n;

    while (first < last) {
        int mid = first + (last - first) / 2;

        if (isBadVersion(mid)) {            last = mid; // Mid could be
the first bad version, so narrow the        // range to the left
half.
        } else {                first = mid + 1; // If mid is not bad, the
first bad version                // must be after mid.
        }
    }
    return first; // At the end, first will be the first bad version.
}
};
```

Output:

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

n =
5

bad =
4

Output

4

Problem 3.

Sort Colors. - Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

Code:

```
class Solution { public:    void
sortColors(vector<int>& nums) {        int low
= 0, mid = 0, high = nums.size()-1;
while(mid <= high){            if(nums[mid] ==
0){                swap(nums[low], nums[mid]);
low++;                mid++;
            }
            else if(nums[mid] == 1){
mid++;
            }
            else{
                swap(nums[mid], nums[high]);
high--;
            }
        }
    } };
```

Output:

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

nums =
[2,0,2,1,1,0]

Output

[0,0,1,1,2,2]

Top K frequent elements- Given an integer array nums and an integer k, return the k most

Problem 4.

frequent elements. You may return the answer in any order.

Code:

```
class Solution { public:    vector<int>
topKFrequent(vector<int>& nums, int k) {        int n =
nums.size();        unordered_map<int, int> map;
vector<int> ans;        for (int &x : nums) map[x]++;
vector<vector<int>> arr(n + 1);        for (auto [a, b] :
map) arr[b].push_back(a);        for (int i = n; i > 0; i--) {
for (int &x : arr[i]) {                if (ans.size() == k) return
ans;                ans.push_back(x);
        }
    }
    return ans;
} };
```

Output:

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

nums =
[1,1,1,2,2,3]

k =
2

Output

[1,2]

Kth Largest element in an array- Given an integer array nums and an integer k, return the kth largest element in the array.

Problem 5.

Code:

```
#include <queue>
#include <vector>

using namespace std;

class Solution { public:    int
findKthLargest(vector<int>& nums, int k) {
    priority_queue<int, vector<int>, greater<int>>> minHeap;

    for (int num : nums) {
minHeap.push(num);          if (minHeap.size() > k)
{          minHeap.pop(); // Remove smallest
element
        }
    }

    return minHeap.top();
} };
```

Output:

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

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

k =
2

Output

5

Problem 6.

Find Peak Element-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..

Code:

```
class Solution { public:
    int findPeakElement(vector<int>& nums) {        int
n=nums.size();    if(n==1)return 0;    int low=1, high=n-2;
if(nums[0]>nums[1])return 0;    if(nums[n-1]>nums[n-2])return
n-1;    while(low<=high){        int mid=low+(high-low)/2;
if(nums[mid]>nums[mid-1] && nums[mid]>nums[mid+1])
return mid;        else if(nums[mid]>nums[mid+1]){
high=mid-1;
        }
    else
        low=mid+1;
    }
    return -1;
} };
```

Output:

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

nums =
[1,2,3,1]

Output

2

Problem 7. Merge Intervals- 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) {    sort(intervals.begin(),
intervals.end()); // Sort intervals by start time    int k = 0; // Index for
merged intervals

    for (int i = 1; i < intervals.size(); i++) {        if (intervals[k][1]
>= intervals[i][0]) { // Overlap detected            intervals[k][1] =
max(intervals[k][1], intervals[i][1]); // Merge
        } else {            k++; // Move to the next
position        }
        intervals[k] = intervals[i]; // Replace
in-place
    }

    intervals.resize(k + 1); // Resize to include only merged intervals
return intervals;
}
};
```

Output:

```
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]]
```

Problem 8.

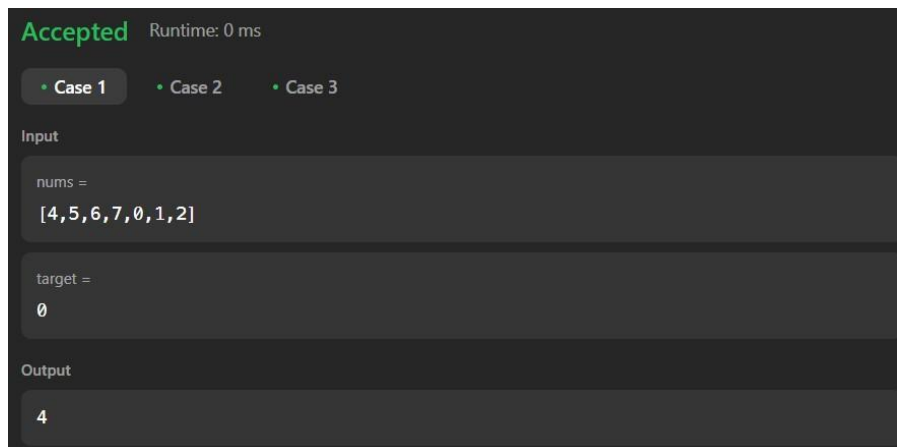
Search in Rotated Sorted Array - There is an integer array `nums` sorted in ascending order (with distinct values). Prior to being passed to your function, `nums` is possibly rotated at an unknown pivot index `k` ($1 \leq k < \text{nums.length}$) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (0-indexed). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index 3 and become `[4,5,6,7,0,1,2]`.

Code:

```
#include<vector>
using namespace std;
class Solution{
public:
    int search(vector<int>&nums,int target){        int
left=0,right=nums.size()-1;        while(left<=right){
int mid=left+(right-left)/2;
if(nums[mid]==target)return mid;
if(nums[left]<=nums[mid]){
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 9.



Search a 2D Matrix II - Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

Code:

```
class Solution { private:    bool
search(vector<int>& arr, int target) {        int
low = 0, high = arr.size() - 1;        while (low
<= high) {            int mid = low + (high - low)
/ 2;            if (arr[mid] == target) return true;
else if (arr[mid] < target) low = mid + 1;
else high = mid - 1;
        }
return false;
    }
public:    bool searchMatrix(vector<vector<int>>& matrix, int
target) {        int n = matrix.size(), m = matrix[0].size();
for (int i = 0; i < n; i++) {            if (search(matrix[i], target)) {
return true;
            }        }
return false;
    }
};
```

Output:

Problem 10.

Accepted Runtime: 0 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
```

Kth smallest element in a sorted matrix - Given an $n \times n$ matrix where each of the rows and columns is sorted in ascending order, return the kth smallest element in the matrix. Note that it is the kth smallest element in the sorted order, not the kth distinct element. You must find a solution with a memory complexity better than $O(n^2)$.

Code:

```
class Solution { public:    int kthSmallest(vector<vector<int>> &matrix, int k) {        int m =  
matrix.size(), n = matrix[0].size(); // For general, the matrix need not be a square  
priority_queue<int> maxHeap;        for (int r = 0; r < m; ++r) {            for (int c = 0; c < n;  
++c) {                maxHeap.push(matrix[r][c]);  
                if (maxHeap.size() > k) maxHeap.pop();  
            }  
        }  
        return maxHeap.top();  
    }  
};
```

Output:

Problem 11.

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

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

k =
8

Output

13