Experiment 5

Student Name: Arun **UID:** 22BET10320
Branch: Information Technology **Section/Group:** 22BET_IOT-701/A
Semester: 6th **Subject Code:** 22ITP-351

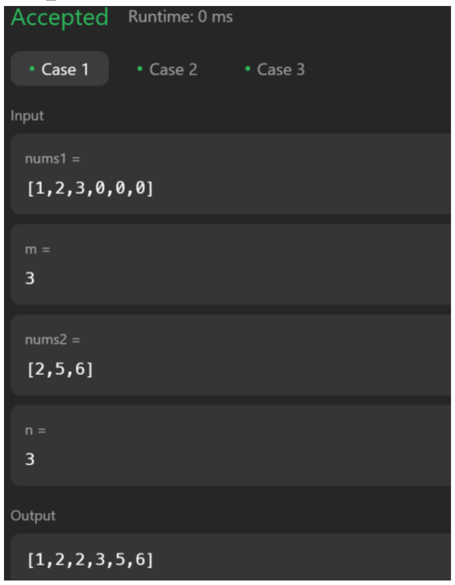
Problem 1

Aim: Merge Sorted Array

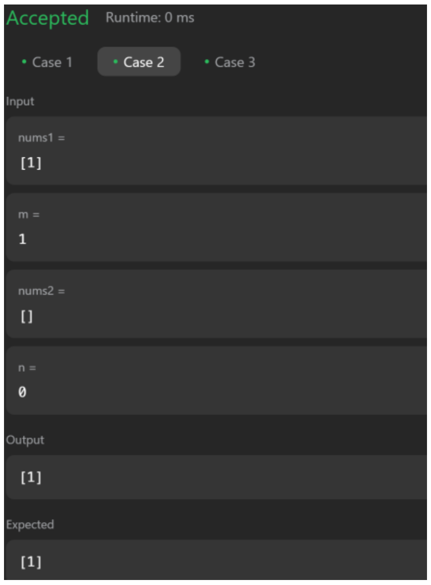
Code:

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int midx = m - 1;
        int nidx = n - 1;
        int right = m + n - 1;

        while (nidx >= 0) {
            if (midx >= 0 && nums1[midx] > nums2[nidx]) {
                nums1[right] = nums1[midx];
                midx--;
            } else {
                nums1[right] = nums2[nidx];
                nidx--;
            }
            right--;
        }
    }
};
```

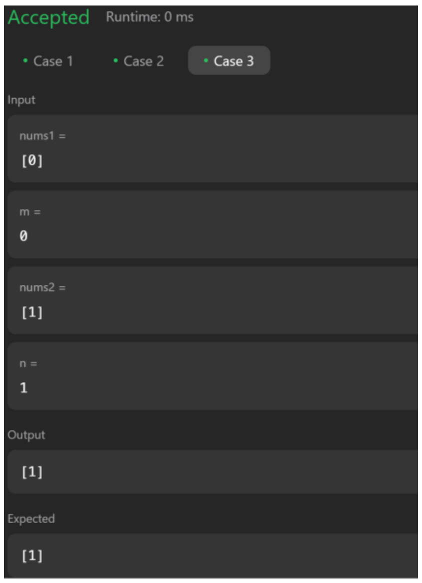
Output:



```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3
Input
nums1 =
[1,2,3,0,0,0]
m =
3
nums2 =
[2,5,6]
n =
3
Output
[1,2,2,3,5,6]
```



```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3
Input
nums1 =
[1]
m =
1
nums2 =
[]
n =
0
Output
[1]
Expected
[1]
```



```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3
Input
nums1 =
[0]
m =
0
nums2 =
[1]
n =
1
Output
[1]
Expected
[1]
```

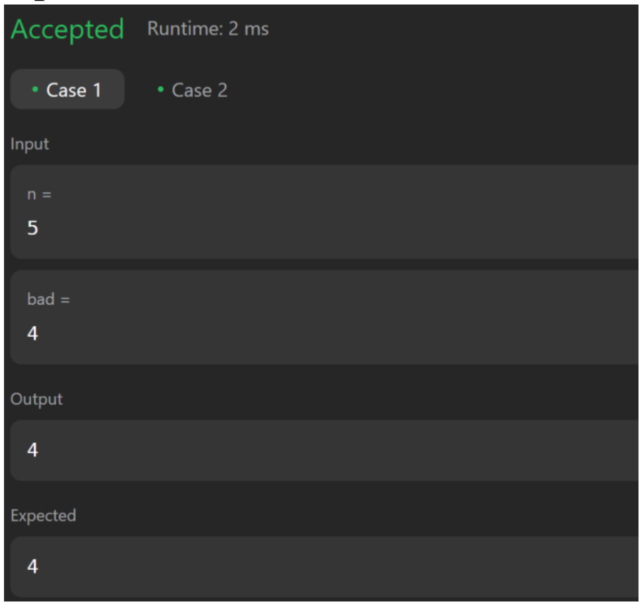
Problem: 2

Aim: First Bad Version

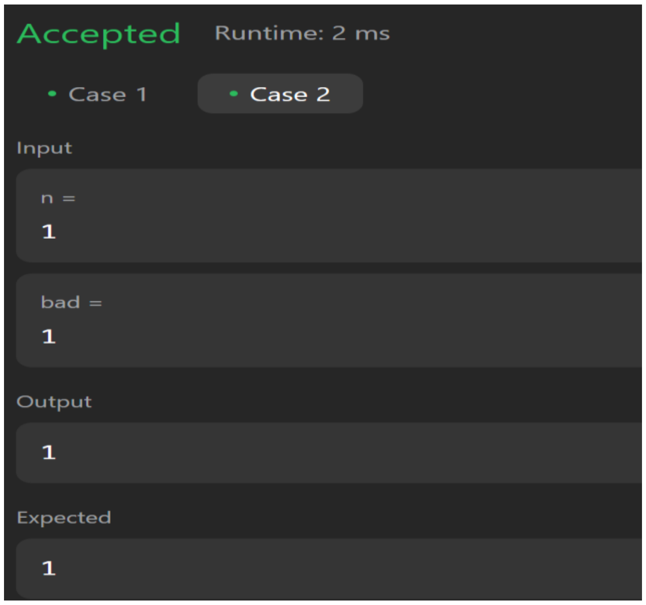
Code:

```
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

class Solution {
public:
    int firstBadVersion(int n){
        long long l = 1, r = n;
        long long m, res = n;
        while(l <= r){
            m = l + (r - l) / 2;
            if(isBadVersion(m) == 1){
                r = m - 1;
                res = min(res, m);
            } else {
                l = m + 1;
            }
        }
        return res;
    }
};
```

Output:



```
Accepted   Runtime: 2 ms
• Case 1   • Case 2
Input
n =
5
bad =
4
Output
4
Expected
4
```



```
Accepted   Runtime: 2 ms
• Case 1   • Case 2
Input
n =
1
bad =
1
Output
1
Expected
1
```

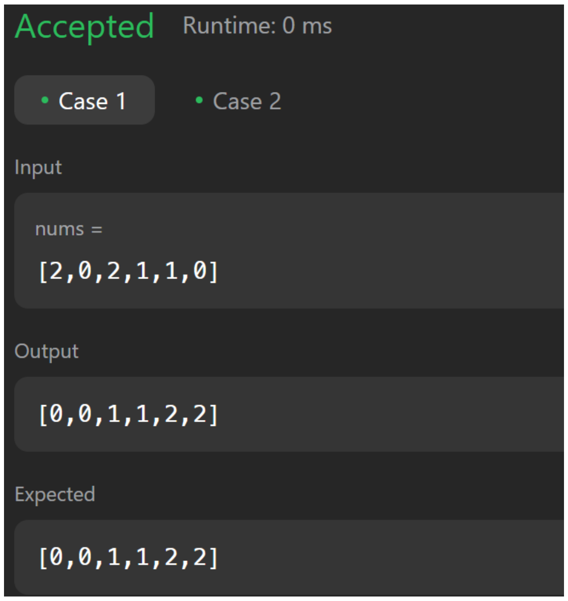
Problem: 3

Aim: Sort Colors

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

Input

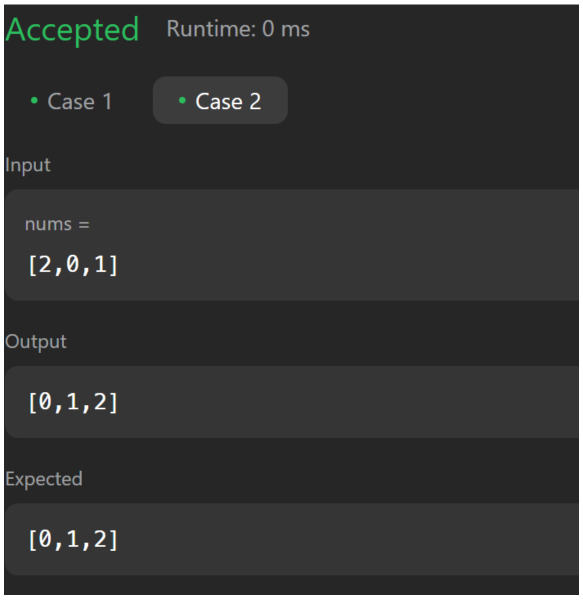
```
nums =
[2,0,2,1,1,0]
```

Output

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

Expected

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



Accepted Runtime: 0 ms

Input

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

Output

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

Expected

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

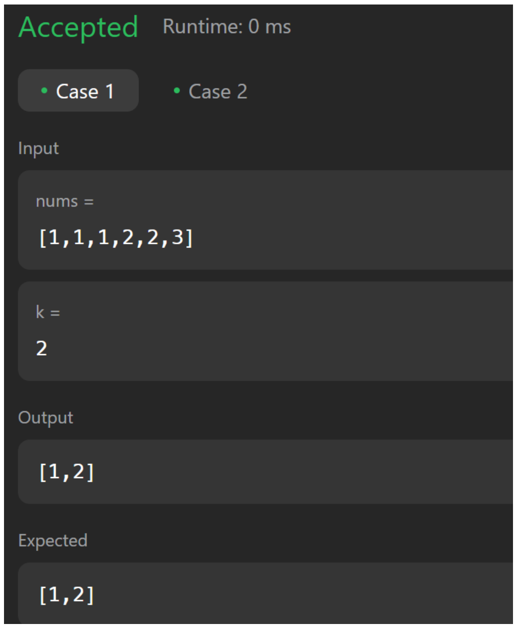
Problem: 4

Aim: Top K frequent elements

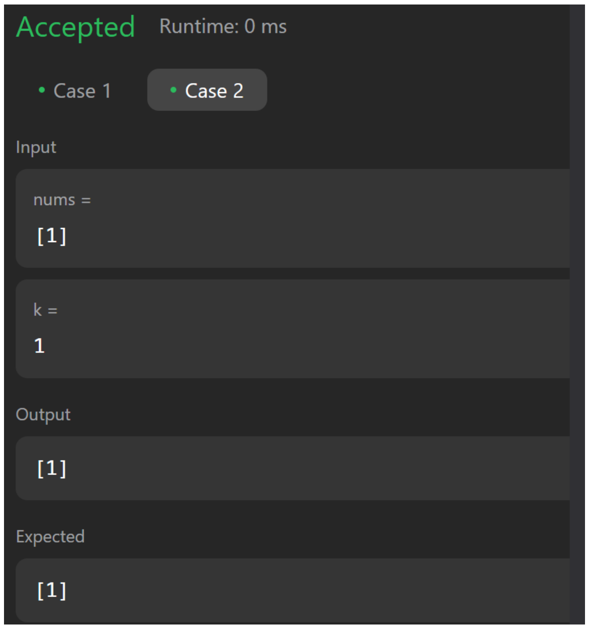
Code:

```
class Solution {
private:
    void fast(){
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
        cout.tie(NULL);
    }
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        fast();
        vector<int> res;
        unordered_map<int, int> mp;
        for(int num: nums){
            mp[num]++;
        }
        priority_queue<pair<int, int>> qu;
        for(auto num: mp){
            qu.push({num.second, num.first});
        }
        while(k--){
            res.push_back(qu.top().second);
            qu.pop();
        }
        return res;
    }
};
```

Output:



```
Accepted  Runtime: 0 ms
Case 1  Case 2
Input
nums =
[1,1,1,2,2,3]
k =
2
Output
[1,2]
Expected
[1,2]
```



```
Accepted  Runtime: 0 ms
Case 1  Case 2
Input
nums =
[1]
k =
1
Output
[1]
Expected
[1]
```

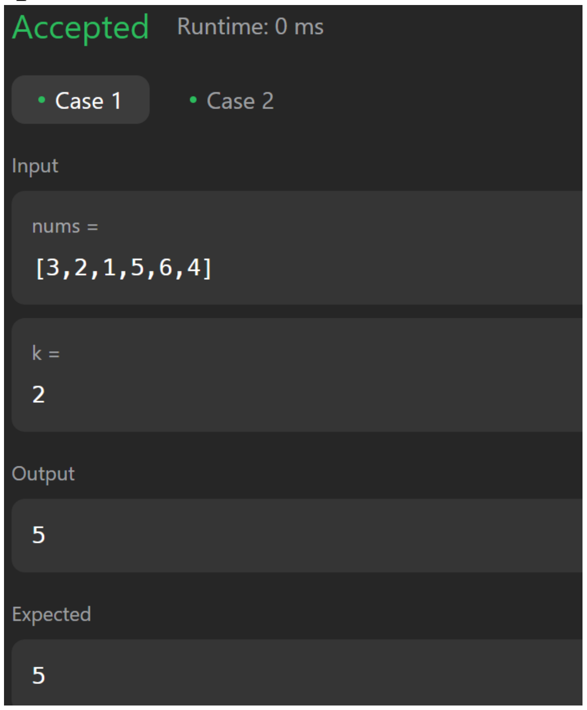
Problem: 5

Aim: Kth Largest Element in an Array

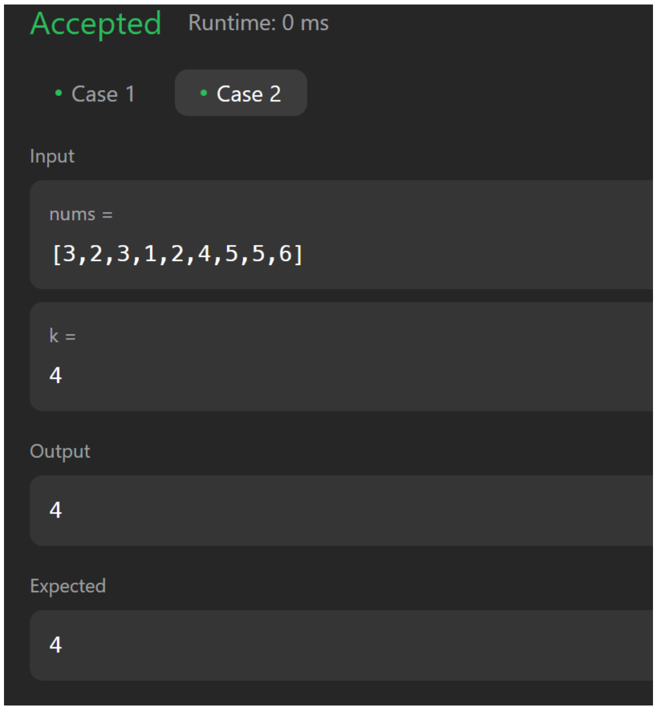
Code:

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

Output:



```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
Input
nums =
[3,2,1,5,6,4]
k =
2
Output
5
Expected
5
```



```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
Input
nums =
[3,2,3,1,2,4,5,5,6]
k =
4
Output
4
Expected
4
```

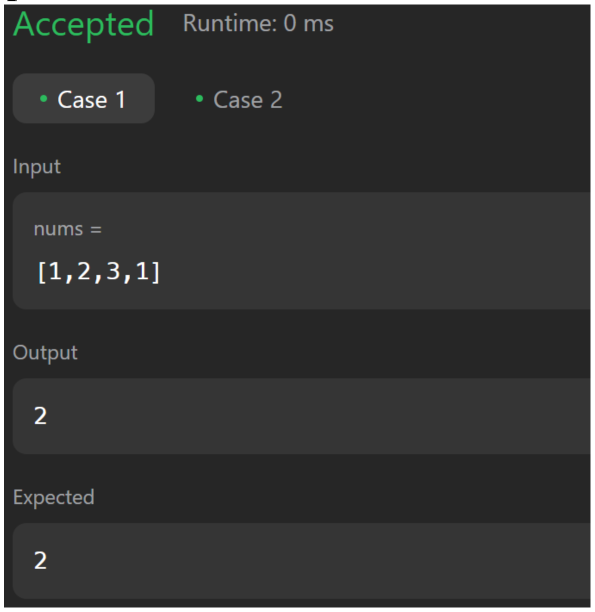
Problem: 6

Aim: Find Peak Element

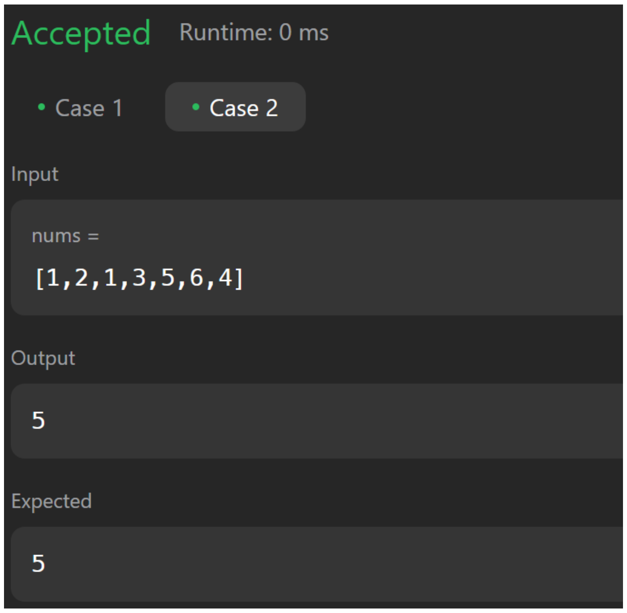
Code:

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

        while (left < right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] > nums[mid + 1]) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        return left;
    }
};
```

Output:



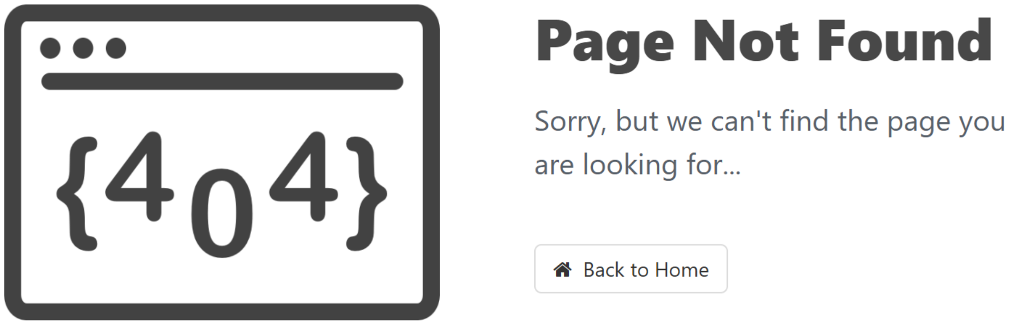
```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
Input
nums =
[1,2,3,1]
Output
2
Expected
2
```



```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
Input
nums =
[1,2,1,3,5,6,4]
Output
5
Expected
5
```

Problem: 7

Aim: Search For a range



Page Not Found

Sorry, but we can't find the page you are looking for...

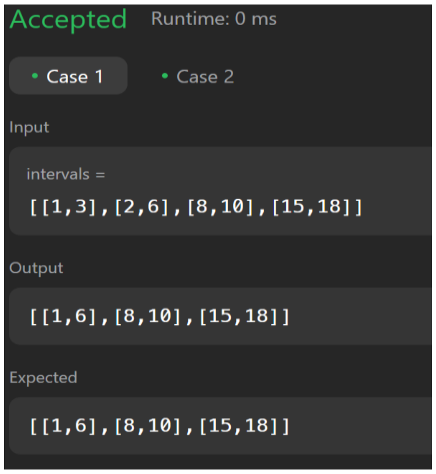
Problem: 8

Aim: Merge Intervals

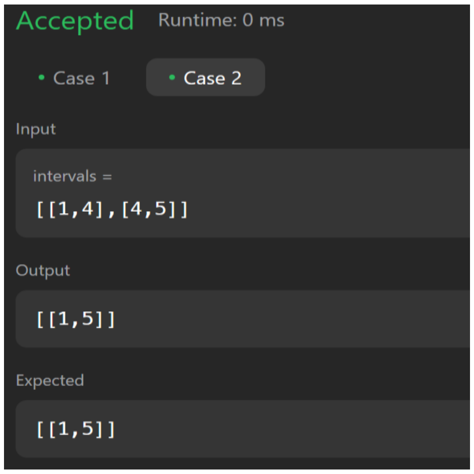
Code:

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {

        sort(intervals.begin(), intervals.end(), [](const vector<int>& a, const vector<int>& b) {
            return a[0] < b[0];
        });

        vector<vector<int>> merged;
        vector<int> prev = intervals[0];

        for (int i = 1; i < intervals.size(); ++i) {
            vector<int> interval = intervals[i];
            if (interval[0] <= prev[1]) {
                prev[1] = max(prev[1], interval[1]);
            } else {
                merged.push_back(prev);
                prev = interval;
            }
        }
        merged.push_back(prev);
        return merged;
    }
};
```

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



```
Accepted   Runtime: 0 ms
Case 1   Case 2
Input
intervals =
[[1,4],[4,5]]
Output
[[1,5]]
Expected
[[1,5]]
```

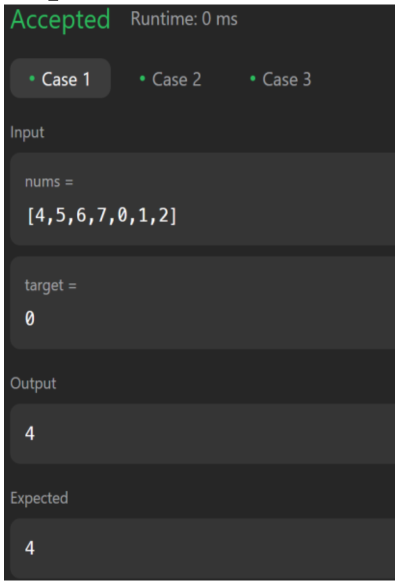
Problem: 9

Aim: Search in Rotated Sorted Array

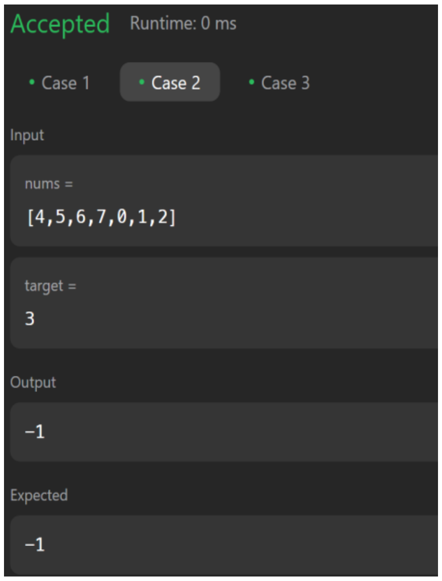
Code:

```
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;
    }
};
```

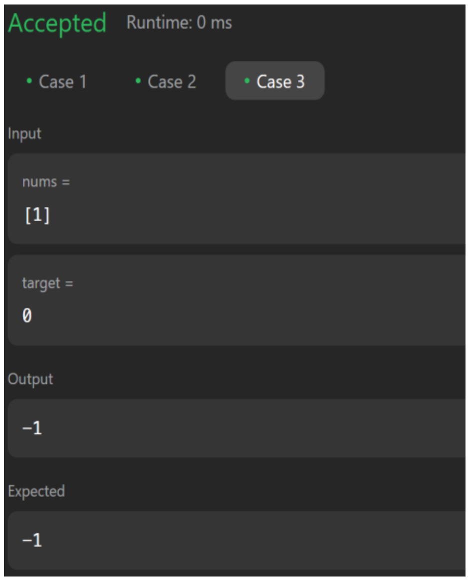
Output:



```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3
Input
nums =
[4,5,6,7,0,1,2]
target =
0
Output
4
Expected
4
```



```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3
Input
nums =
[4,5,6,7,0,1,2]
target =
3
Output
-1
Expected
-1
```



```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3
Input
nums =
[1]
target =
0
Output
-1
Expected
-1
```

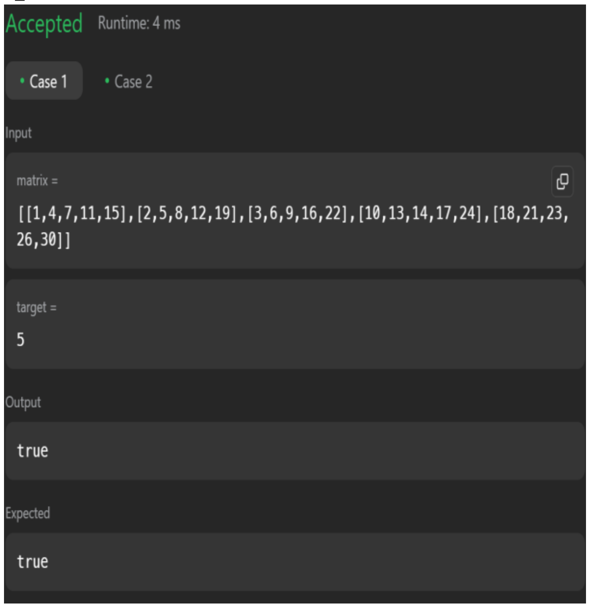
Problem: 10

Aim: Search a 2D Matrix II

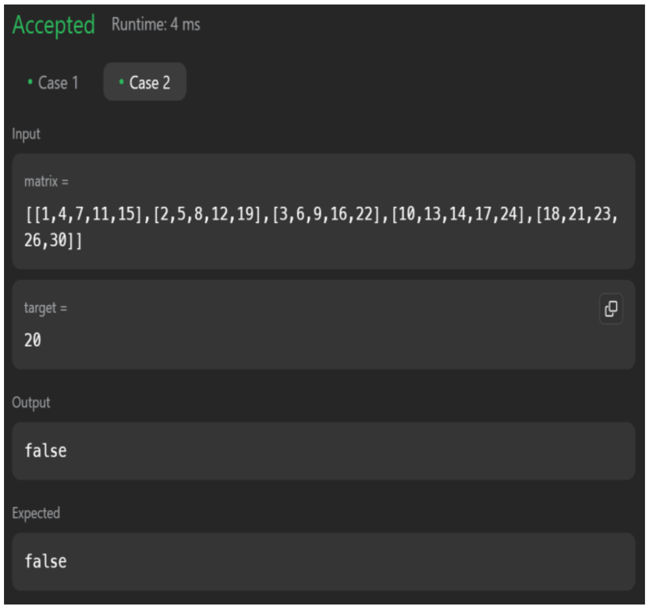
Code:

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int n = matrix.size(), m = matrix[0].size();
        int row = 0, col = m - 1;

        while (row < n && col >= 0) {
            if (matrix[row][col] == target) return true;
            else if (matrix[row][col] < target) row++;
            else col--;
        }
        return false;
    }
};
```

Output:



```
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
```



```
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 =
20
Output
false
Expected
false
```

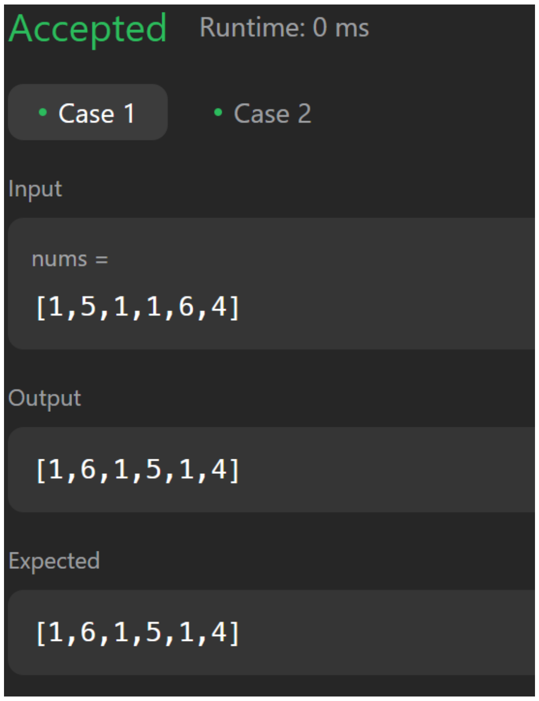
Problem: 11

Aim: Wiggle Sort 2

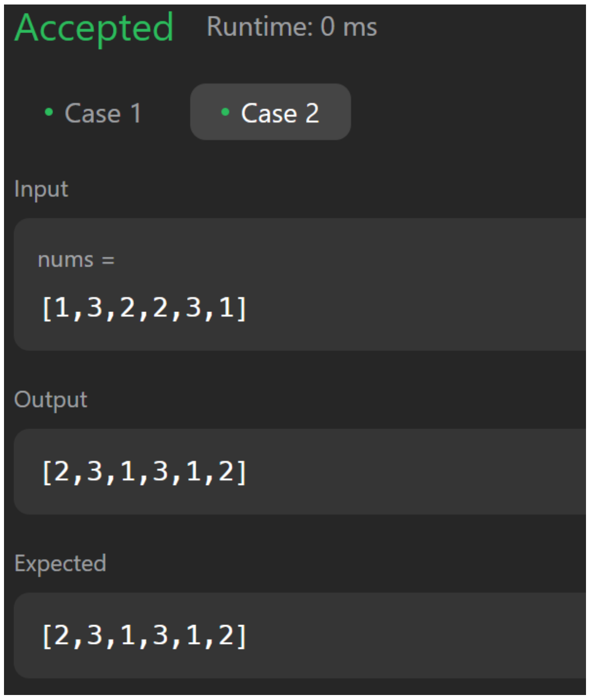
Code:

```
class Solution {
public:
    void wiggleSort(vector<int>& nums) {
        int n = nums.size();
        vector<int> nums1(nums);
        sort(nums1.begin(), nums1.end());
        int i = n-1;
        int j = 0;
        int k = i/2 + 1;
        while(i >= 0)
        {
            if(i % 2 == 1)
            {
                nums[i] = nums1[k];
                k++;
            }
            else
            {
                nums[i] = nums1[j];
                j++;
            }
            i--;
        }
    }
};
```

Output:



```
Accepted   Runtime: 0 ms

• Case 1   • Case 2

Input
nums =
[1,5,1,1,6,4]

Output
[1,6,1,5,1,4]

Expected
[1,6,1,5,1,4]
```



```
Accepted   Runtime: 0 ms

• Case 1   • Case 2

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

Output
[2,3,1,3,1,2]

Expected
[2,3,1,3,1,2]
```

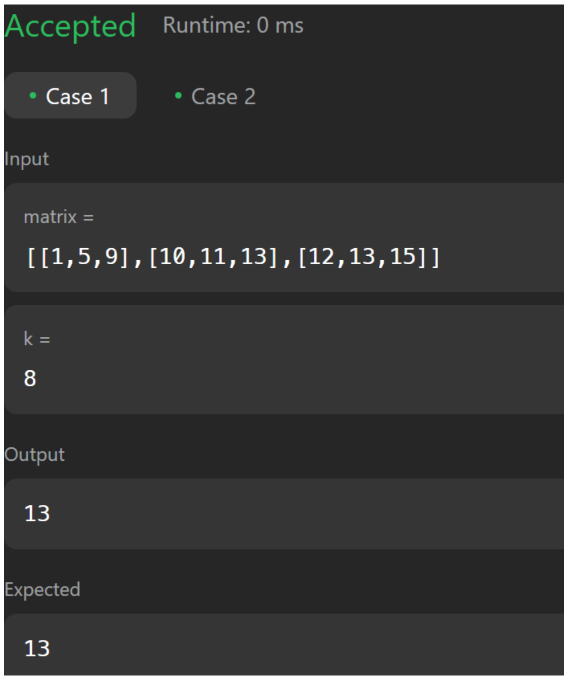
Problem:12

Aim: Kth smallest element in a sorted matrix

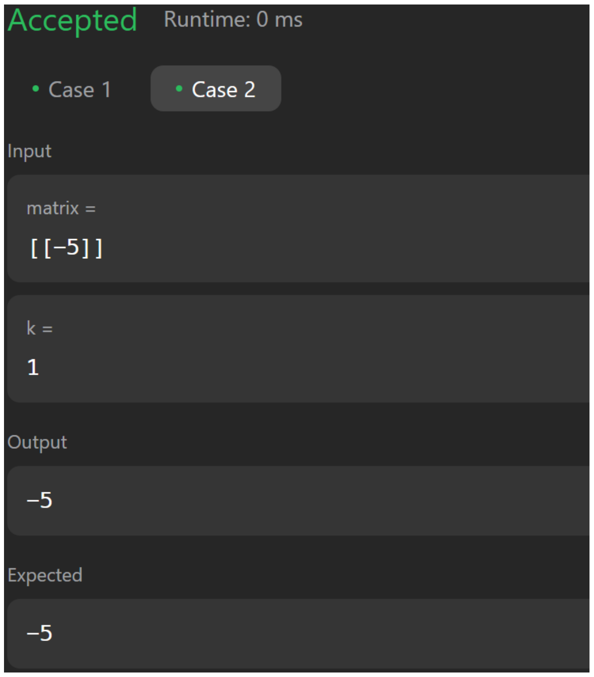
Code:

```
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];
    }
};
```

Output:



```
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
```



```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
Input
matrix =
[[-5]]
k =
1
Output
-5
Expected
-5
```

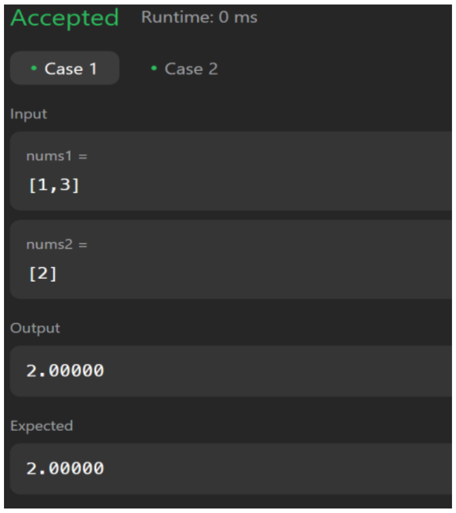
Problem: 13

Aim: Median of Two Sorted Arrays

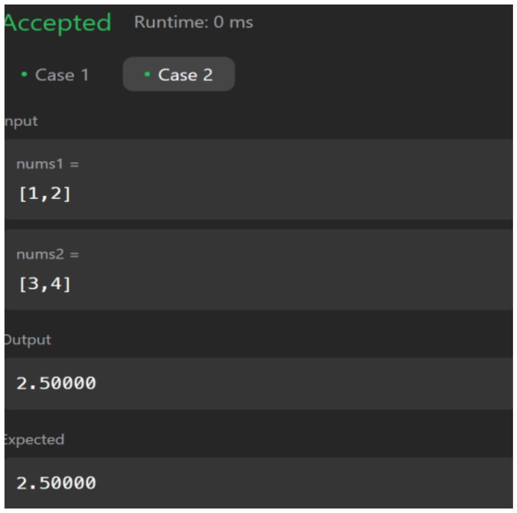
Code:

```
class Solution {
public:
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
// Get the sizes of both input arrays.
int n = nums1.size();
int m = nums2.size();

// Merge the arrays into a single sorted array.
vector<int> merged;
for (int i = 0; i < n; i++) {
merged.push_back(nums1[i]);
}
for (int i = 0; i < m; i++) {
merged.push_back(nums2[i]);
}

// Sort the merged array.
sort(merged.begin(), merged.end());

// Calculate the total number of elements in the merged array.
int total = merged.size();

if (total % 2 == 1) {
// If the total number of elements is odd, return the middle element as the median.
return static_cast<double>(merged[total / 2]);
} else {
// If the total number of elements is even, calculate the average of the two middle elements as the
median.
int middle1 = merged[total / 2 - 1];
int middle2 = merged[total / 2];
return (static_cast<double>(middle1) + static_cast<double>(middle2)) / 2.0;
}
}
};
```

Output:



```
Input
nums1 =
[1,3]
nums2 =
[2]
Output
2.00000
Expected
2.00000
```



```
nput
nums1 =
[1,2]
nums2 =
[3,4]
utput
2.50000
xpected
2.50000
```