Experiment-5

Student Name: Gaurav Saini	**UID:** 22BCS11085
Branch: BE-CSE	**Section/Group:** NTPP_IOT_603_B
Semester:06	**Date of Performance:**21-02-2025
Subject Name: AP LAB-II	**Subject Code:** 22CSP-351

1. **Aim:**
   a. Merge sorted array
   b. Kth largest Element in an Array
   c. Top K Frequent Elements
2. **Introduction to Binary Trees Problem:**

An array is a linear data structure that stores elements of the same type in contiguous memory locations. Arrays allow fast access to elements using an index and are widely used in programming for their simplicity and efficiency.

Key Features of Arrays:

- Fixed Size: The size of an array is determined at the time of declaration (in static arrays).
- Zero-Based Indexing: The first element is at index 0, the second at 1, and so on.
- Efficient Access: Any element can be accessed in O(1) time using its index.

3. **Implementation/Code:**

A. Merge sorted array

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m - 1; // Last valid element in nums1
        int j = n - 1; // Last element in nums2
        int k = m + n - 1; // Last position in nums1

        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k--] = nums1[i--];
            } else {
```

```
                     nums1[k--] = nums2[j--];
                 }
             }

             // If there are remaining elements in nums2, copy them
             while (j >= 0) {
                 nums1[k--] = nums2[j--];
             }
         }
};
```

B. Kth largest element in an Array

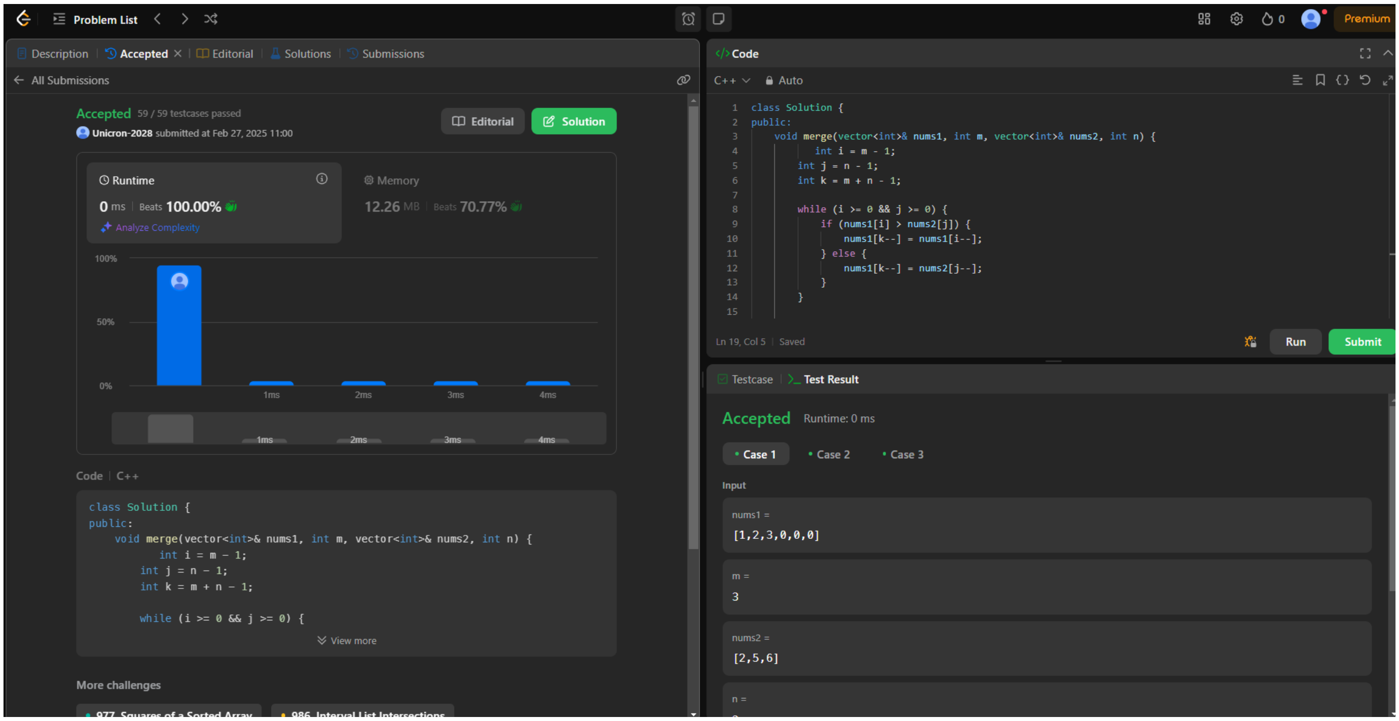
```
     #include <queue>
     #include <vector>

     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();
                 }
             }

             return minHeap.top(); // Root of the heap contains the kth largest
            element
         }
};
```

C. Top K Frequent elements

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

using namespace std;

class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int, int> freqMap;
        for (int num : nums) {
            freqMap[num]++;
        }

        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>> minHeap;

        for (auto& [num, freq] : freqMap) {
            minHeap.push({freq, num});
            if (minHeap.size() > k) {
                minHeap.pop();
            }
        }

        vector<int> result;
        while (!minHeap.empty()) {
            result.push_back(minHeap.top().second);
            minHeap.pop();
        }

        return result;
    }
};
```

4. Output

A. Merge sorted array



Accepted 59 / 59 testcases passed
Unicron-2028 submitted at Feb 27, 2025 11:00

Runtime
0 ms | Beats 100.00%

Memory
12.26 MB | Beats 70.77%

Code | C++

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

        while (i >= 0 && j >= 0) {
```

View more

More challenges

977. Squares of a Sorted Array
986. Interval List Intersections

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

        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k--] = nums1[i--];
            } else {
                nums1[k--] = nums2[j--];
            }
        }

```

Accepted Runtime: 0 ms

Input

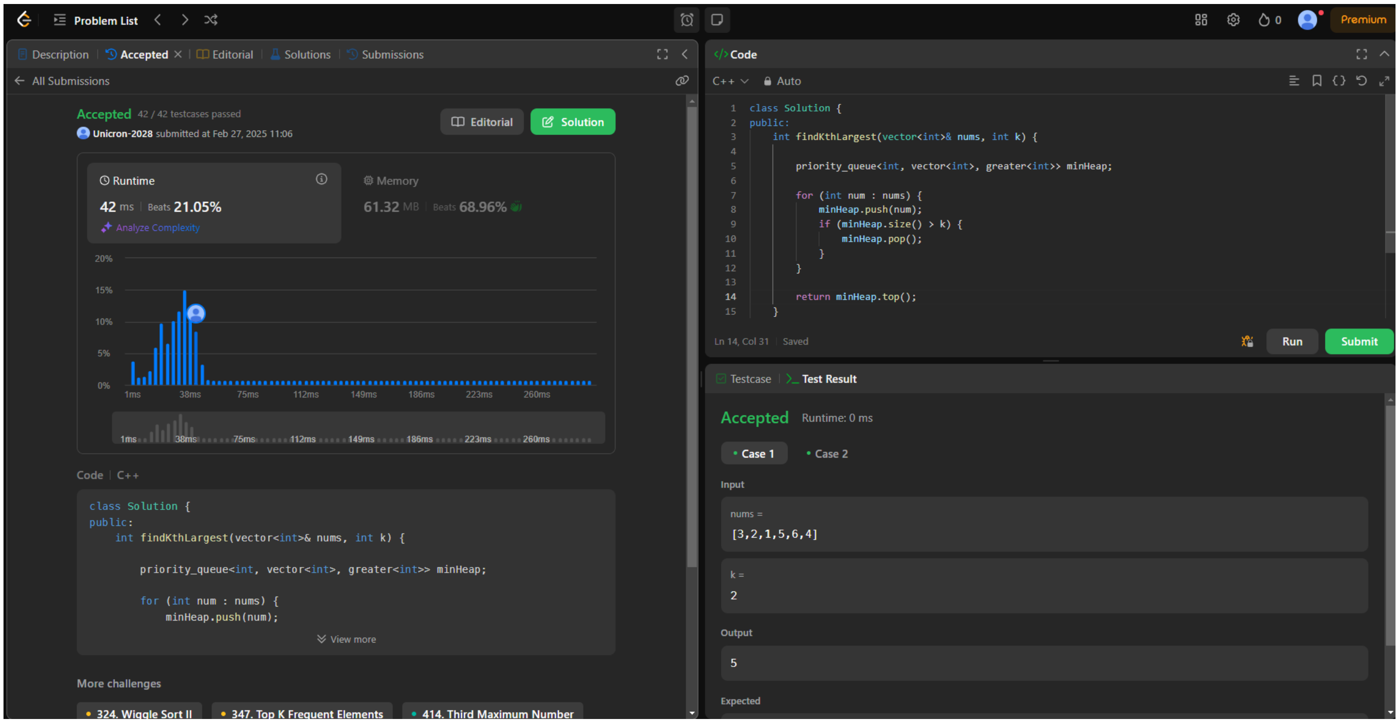
nums1 =
[1,2,3,0,0,0]

m =
3

nums2 =
[2,5,6]

n =

B. Kth largest element in an Array



Accepted 42 / 42 testcases passed
Unicron-2028 submitted at Feb 27, 2025 11:06

Runtime
42 ms | Beats 21.05%

Memory
61.32 MB | Beats 68.96%

Code | C++

```
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);
```

View more

More challenges

324. Wiggle Sort II
347. Top K Frequent Elements
414. Third Maximum Number

```
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();
            }
        }

        return minHeap.top();
    }
```

Accepted Runtime: 0 ms

Input

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

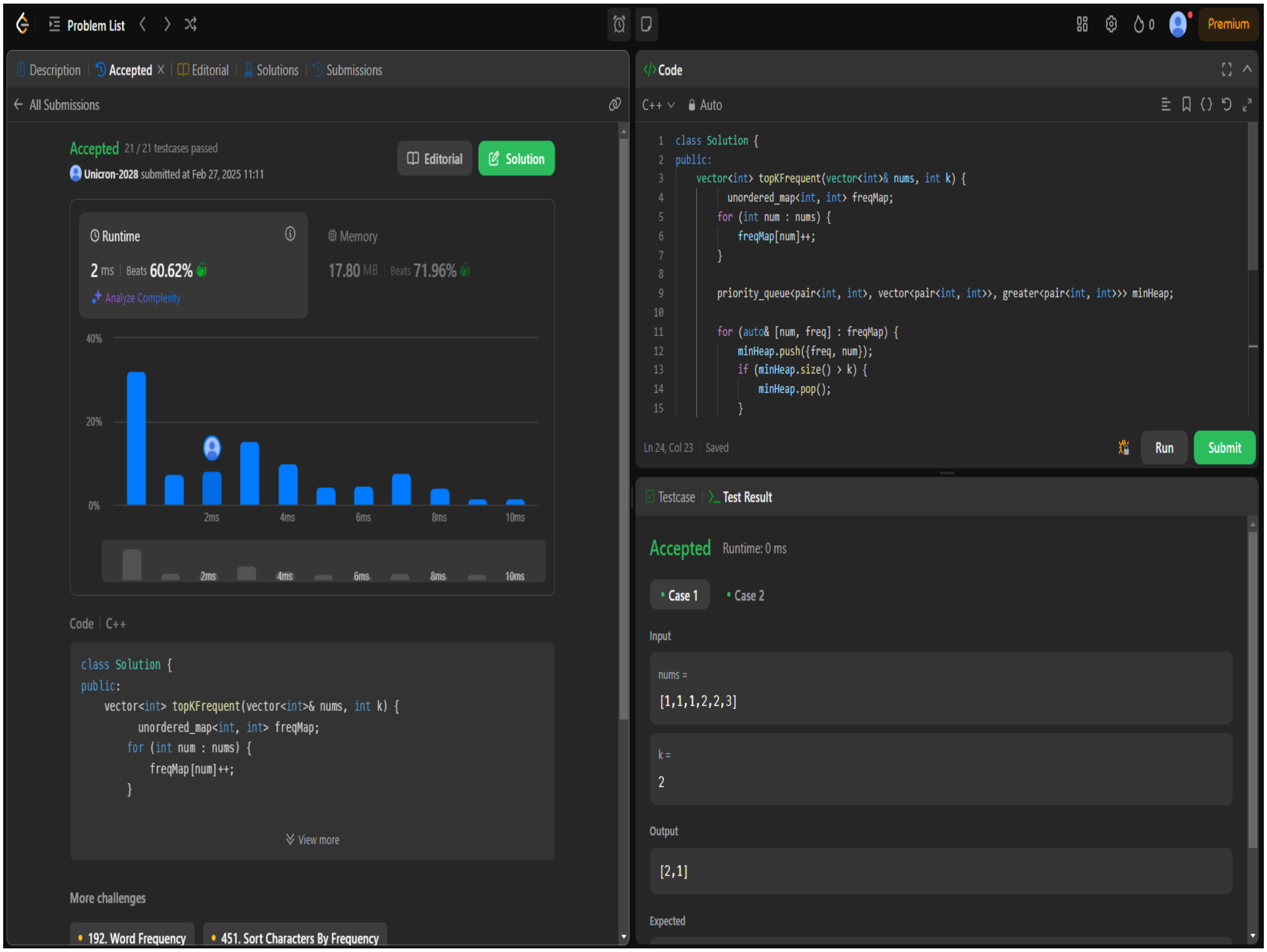
k =
2

Output

5

Expected

A. Top K Frequent elements



Accepted 21 / 21 testcases passed

Unicron-2028 submitted at Feb 27, 2025 11:11

Runtime: 2 ms | Beats 60.62%

Memory: 17.80 MB | Beats 71.96%

Code | C++

```cpp
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
         unordered_map<int, int> freqMap;
        for (int num : nums) {
            freqMap[num]++;
        }
```

```cpp
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
         unordered_map<int, int> freqMap;
        for (int num : nums) {
            freqMap[num]++;
        }

        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> minHeap;

        for (auto& [num, freq] : freqMap) {
            minHeap.push({freq, num});
            if (minHeap.size() > k) {
                minHeap.pop();
            }
```

Accepted Runtime: 0 ms

Input

nums =

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

k =

2

Output

[2,1]

Expected

5. Learning Outcomes:

❖ Efficient Sorting & Merging → Using two-pointer techniques instead of naive sorting.
❖ Heap-Based Optimizations → Using min-heaps for optimal element selection.
❖ Hash Maps for Counting Frequencies → Useful for problems involving frequent elements.
❖ Time & Space Complexity Trade-offs → Choosing the right approach based on constraints.