

Experiment - 5

Student Name: Shivam Kumar

Branch: BE-CSE

Semester: 6th

Subject Name: AP LAB-II

UID: 23BCS80024

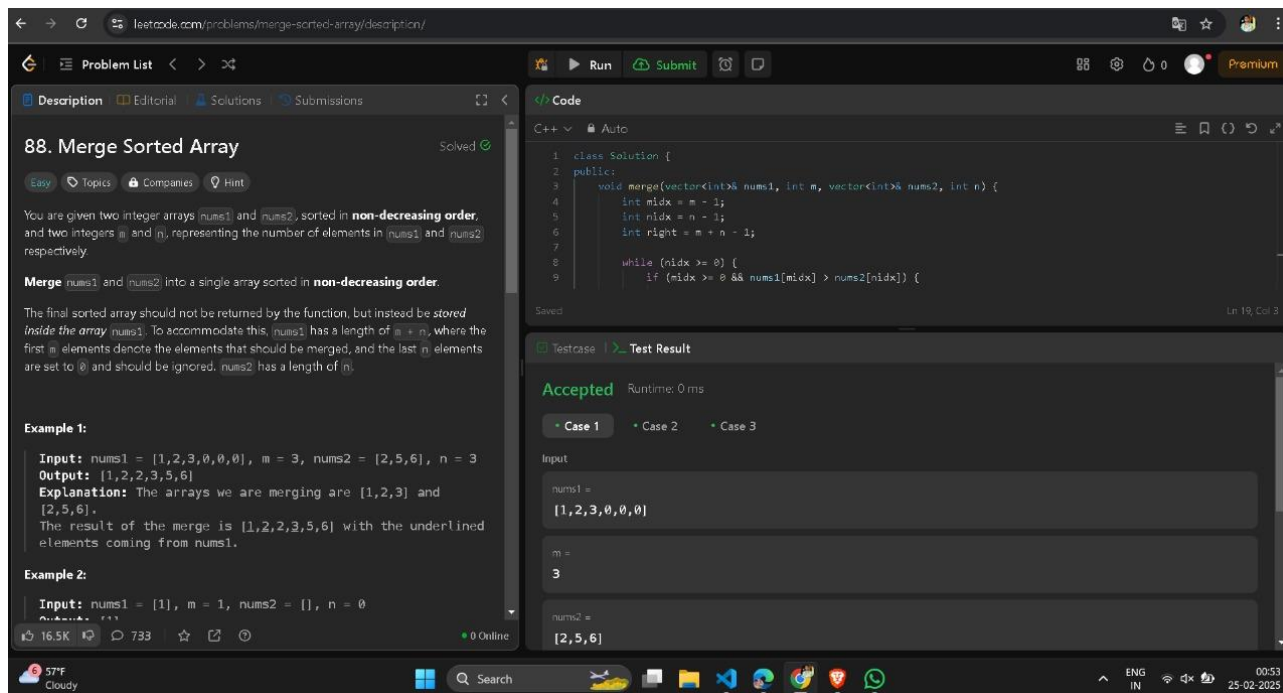
Section/Group: 634/A

Date of Performance: 20/02/25

Subject Code: 22CSP-351

Problem 5.1: Merge Sorted Array.

1. Output:



The screenshot displays the LeetCode interface for problem 88, "Merge Sorted Array". The problem description states: "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. Merge `nums1` and `nums2` into a single array sorted in non-decreasing order. The final sorted array should not be returned by the function, but instead be stored inside the array `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to 0 and should be ignored. `nums2` has a length of `n`." Example 1 shows `nums1 = [1,2,3,0,0,0]`, `m = 3`, `nums2 = [2,5,6]`, `n = 3`, resulting in `[1,2,2,3,5,6]`. Example 2 shows `nums1 = [1]`, `m = 1`, `nums2 = []`, `n = 0`, resulting in `[1]`.

The C++ solution provided is:

```

1 class Solution {
2 public:
3     void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
4         int midx = m - 1;
5         int nidx = n - 1;
6         int right = m + n - 1;
7
8         while (nidx >= 0) {
9             if (midx >= 0 && nums1[midx] > nums2[nidx]) {

```

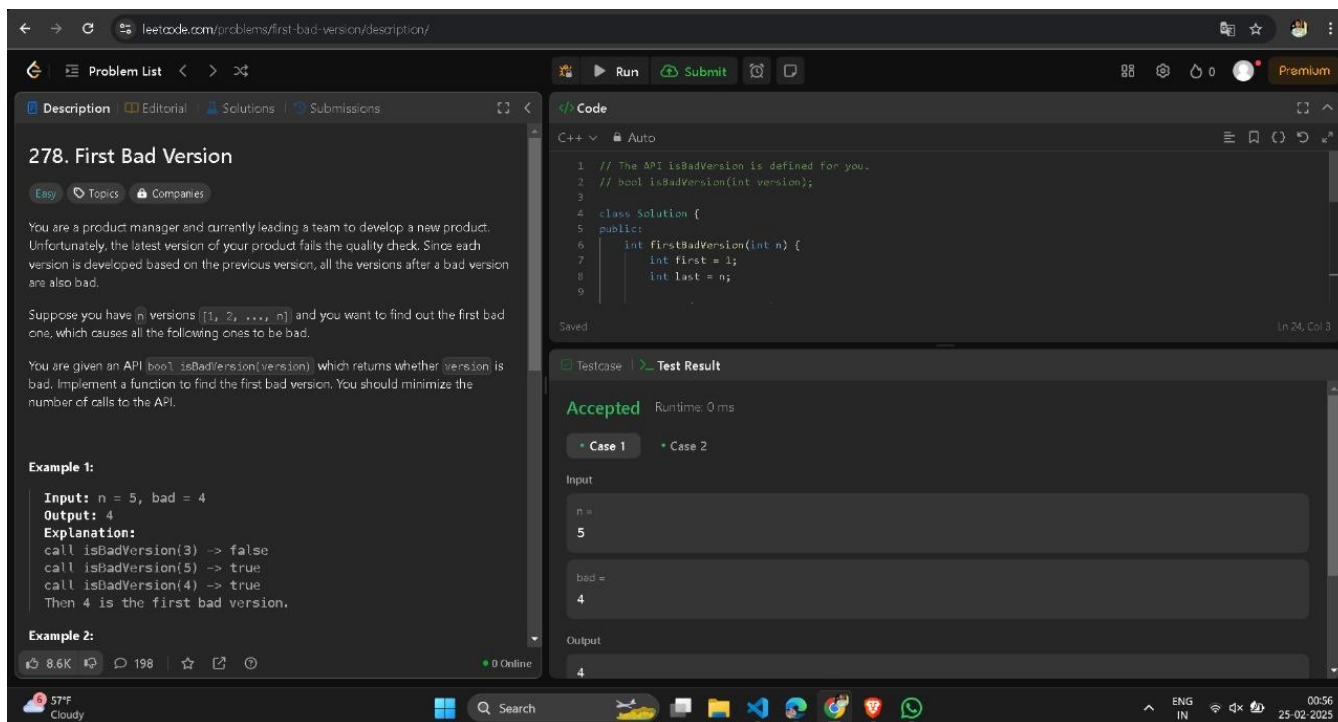
The test results show "Accepted" with a runtime of 0 ms. The input for Case 1 is `nums1 = [1,2,3,0,0,0]`, `m = 3`, and `nums2 = [2,5,6]`.

2. Learning outcomes:

1. Understanding the Two-Pointer Technique
2. Mastering In-Place Merging
3. Time Complexity Analysis
4. Handling Edge Cases

Problem 5.2: First Bad Version

1. Output:



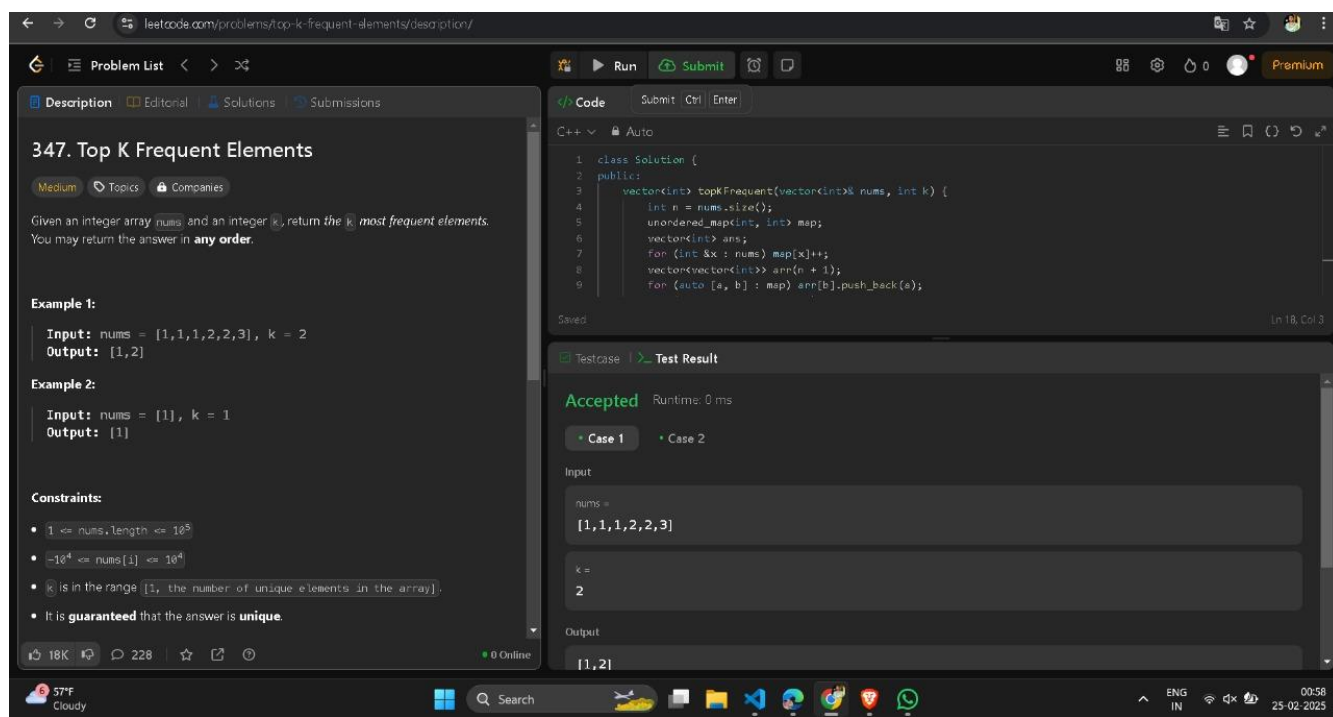
The screenshot displays the LeetCode interface for the problem "278. First Bad Version". The problem description on the left explains that a product manager is leading a team to develop a new product, but the latest version fails a quality check. Since each version is based on the previous one, all versions after a bad one are also bad. The goal is to find the first bad version among n versions (1 to n) by minimizing API calls to `isBadVersion(version)`. Example 1 shows $n=5$, $\text{bad}=4$, with `isBadVersion(3)` returning false and `isBadVersion(4)` returning true, identifying 4 as the first bad version. Example 2 is also present. The right panel shows a C++ solution using binary search, with a `class Solution` containing a `firstBadVersion` method. The test results section shows "Accepted" with a runtime of 0 ms for Case 1, where the input $n=5$ and $\text{bad}=4$ resulted in an output of 4.

2. Learning Outcomes:

- Understanding Binary Search
- Reducing Time Complexity.
- Working with a Monotonic Condition.
- Implementing a Search with a Condition

Problem 5.3: Top K frequent elements

1. Output:



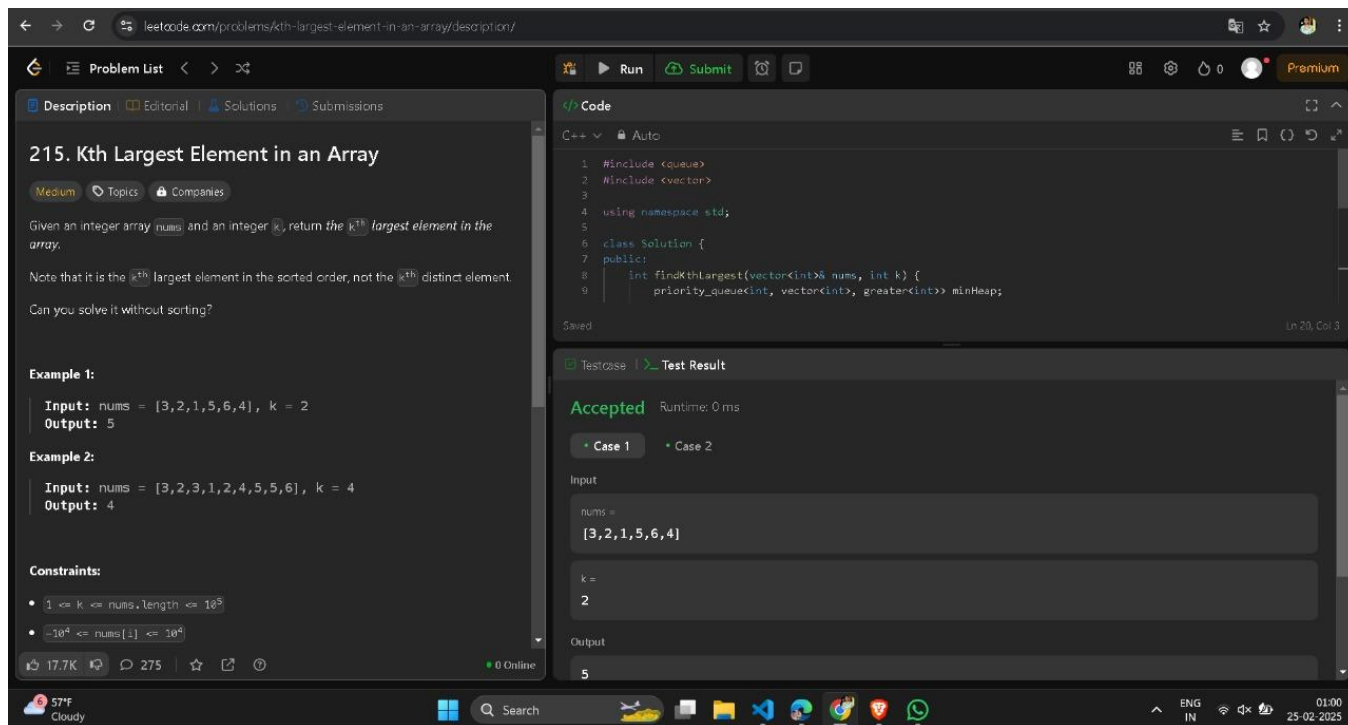
The screenshot shows the LeetCode interface for problem 347, "Top K Frequent Elements". The problem description states: "Given an integer array `nums` and an integer `k`, return the `k` most frequent elements. You may return the answer in **any order**." Example 1: Input: `nums = [1,1,1,2,2,3]`, `k = 2`; Output: `[1,2]`. Example 2: Input: `nums = [1]`, `k = 1`; Output: `[1]`. Constraints include `1 <= nums.length <= 10^5`, `-10^4 <= nums[i] <= 10^4`, `k` is in the range `[1, the number of unique elements in the array]`, and the answer is guaranteed to be unique. The code editor shows a C++ solution using a hash map and a priority queue. The test result shows "Accepted" with a runtime of 0 ms for Case 1, where the input is `nums = [1,1,1,2,2,3]` and `k = 2`, resulting in the output `[1,2]`.

2. Learning Outcomes:

- Understanding Frequency Counting (Hash Maps)
- Priority Queues (Heaps)
- Sorting and Bucket Sort.
- QuickSelect Algorithm (Partitioning)

Problem 5.4: Kth Largest element in an array

1. Output:



215. Kth Largest Element in an Array

Medium Topics Companies

Given an integer array `nums` and an integer `k`, return the `kth` largest element in the array.

Note that it is the `kth` largest element in the sorted order, not the `kth` distinct element.

Can you solve it without sorting?

Example 1:

Input: `nums = [3,2,1,5,6,4]`, `k = 2`
Output: `5`

Example 2:

Input: `nums = [3,2,3,1,2,4,5,5,6]`, `k = 4`
Output: `4`

Constraints:

- `1 <= k <= nums.length <= 105`
- `-104 <= nums[i] <= 104`

17.7K 275 0 Online

```
1 #include <queue>
2 #include <vector>
3
4 using namespace std;
5
6 class Solution {
7 public:
8     int findKthLargest(vector<int>& nums, int k) {
9         priority_queue<int, vector<int>, greater<int>> minHeap;
```

Accepted Runtime: 0ms

Case 1 Case 2

Input

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

k =
2

Output

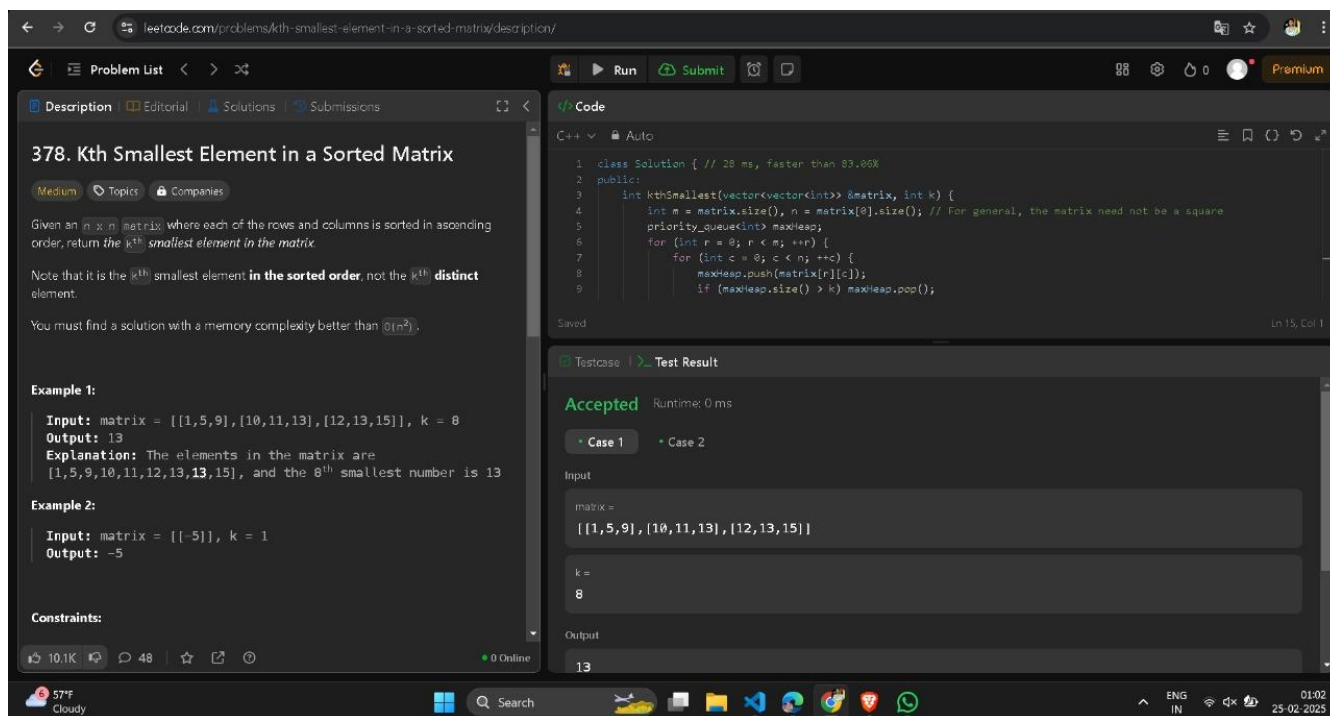
5

2. Learning Outcomes:

- Understanding Sorting and Its Complexity.
- Learning Efficient Selection Algorithms
- Working with Different Data Structures.
- Trade-offs Between Time and Space Complexity.

Problem 5.5: Kth smallest element in a sorted matrix

1. Output:



378. Kth Smallest Element in a Sorted Matrix

Medium Topics Companies

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.

Note that it is the k^{th} smallest element in the sorted order, not the k^{th} distinct element.

You must find a solution with a memory complexity better than $O(n^2)$.

Example 1:

Input: matrix = [[1,5,9],[10,11,13],[12,13,15]], k = 8
Output: 13
Explanation: The elements in the matrix are [1,5,9,10,11,12,13,13,15], and the 8th smallest number is 13

Example 2:

Input: matrix = [[-5]], k = 1
Output: -5

Constraints:

- 10.1K votes, 48 comments, 0 Online

```

1 class Solution { // 28 ms, faster than 83.86%
2 public:
3     int kthSmallest(vector<vector<int>> &matrix, int k) {
4         int m = matrix.size(), n = matrix[0].size(); // For general, the matrix need not be a square
5         priority_queue<int> maxHeap;
6         for (int r = 0; r < m; ++r) {
7             for (int c = 0; c < n; ++c) {
8                 maxHeap.push(matrix[r][c]);
9                 if (maxHeap.size() > k) maxHeap.pop();
10            }
11        }
12        return maxHeap.top();
13    }
14 };
  
```

Testcase: Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

matrix =

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

k =

8

Output

13

2. Learning Outcomes:

- Binary Search on Answer (Matrix Search Space)
- Heap (Priority Queue) Approach
- Matrix Traversal & Counting
- Time Complexity Trade-offs