Experiment 5

Student Name: Aksh Dubey **UID:**22BET10271
Branch: BE -IT **Section/Group:**22BET/IOT/702/A
Semester: 6th **Date of Performance:**21/02/2025
Subject Name: Advanced Programming Lab-2 **Subject Code:** 22ITP-351

1. Aim 1 : Kth Largest Element in an Array

Given an integer array nums and an integer k, return *the* k^th^ *largest element in the array*.

Note that it is the k^th^ largest element in the sorted order, not the k^th^ distinct element.Can you solve it without sorting?

2. Merge Intervals :

Given an array of intervals where intervals[i] = [$start_i$, end_i], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

3. 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 <= k < 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].

Given the array nums after the possible rotation and an integer target, return *the index of* target *if it is in* nums*, or* -1 *if it is not in* nums.

4. 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:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

5. Kth Smallest Element in a Sorted Matrix:

Given an n x 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 $O(n^2)$.

6. Median of Two Sorted Arrays:

Given two sorted arrays nums1 and nums2 of size m and n respectively, return **the median** of the two sorted arrays.The overall run time complexity should be O(log (m+n)).

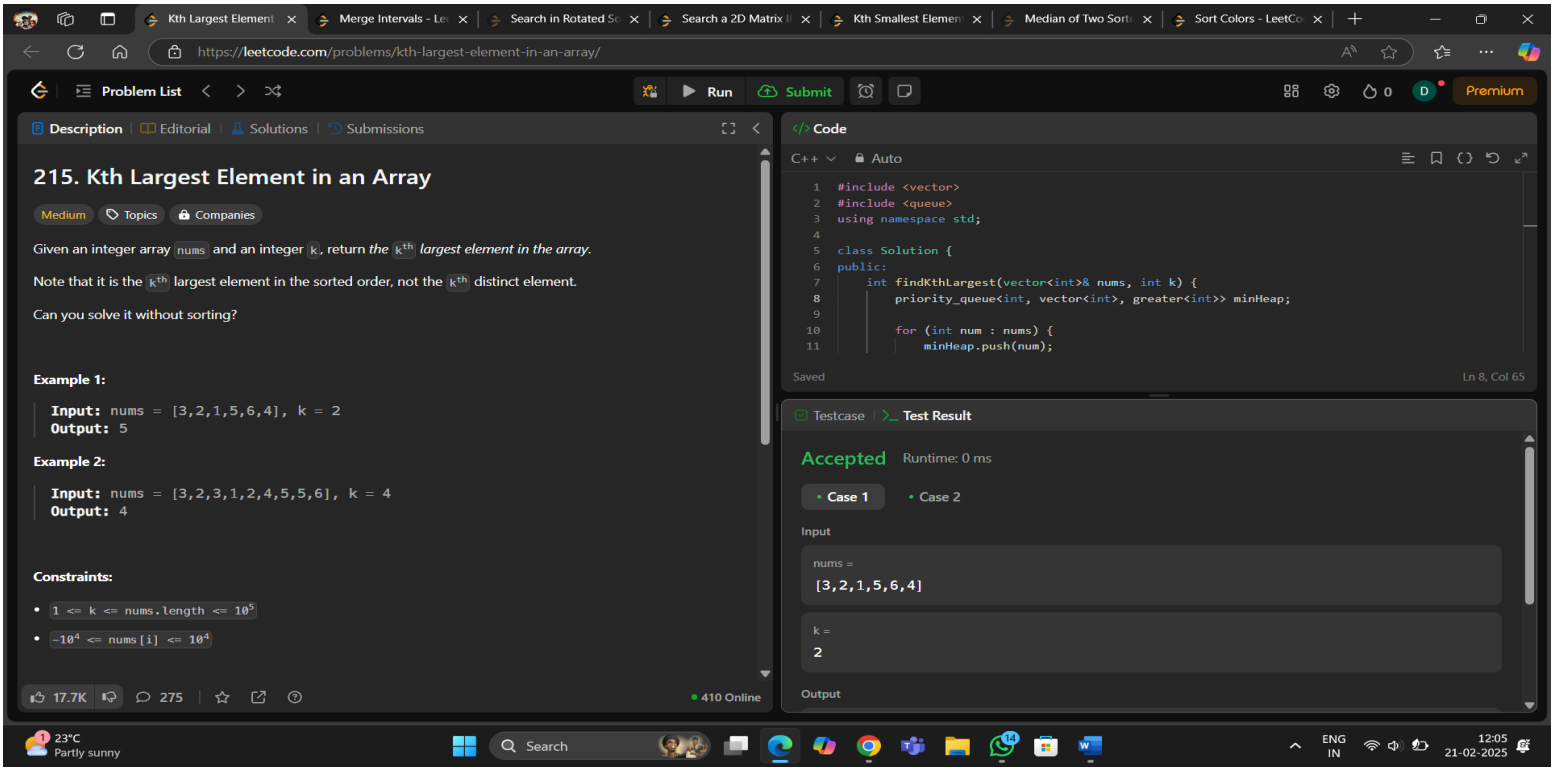
7. Sort Colors :

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.We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.You must solve thisproblem without using the library's sort function.

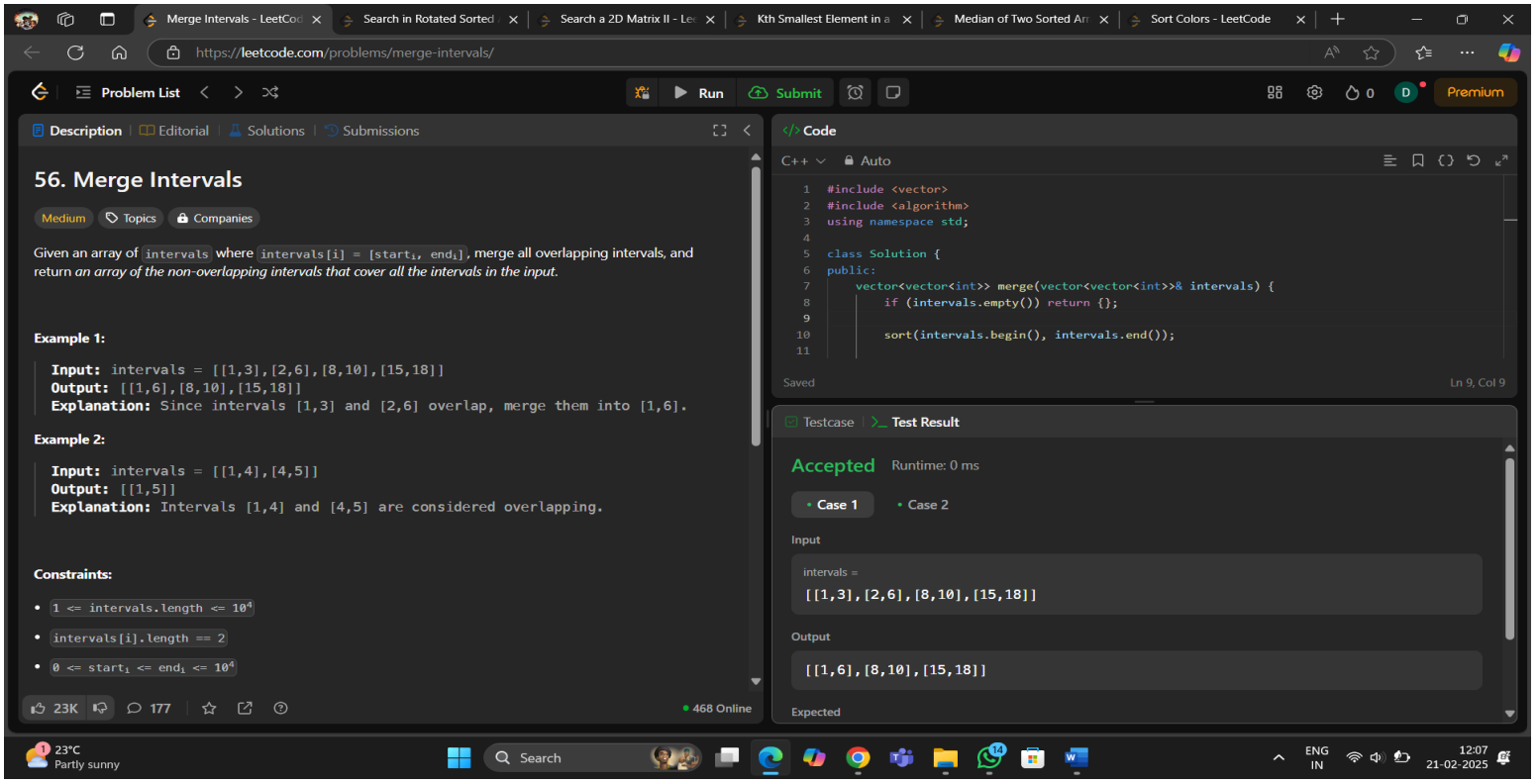
8. Objective:

- Find the kth largest element efficiently without sorting the entire array.
- Given overlapping intervals, merge them into a minimal set of non-overlapping intervals.
- Find the target element in a rotated sorted array in O(log n) time.
- Search for a target efficiently in a row-wise and column-wise sorted matrix.
- Find the kth smallest element in a sorted matrix efficiently.
- Find the median of two sorted arrays in O(log (m+n)) time.
- Sort an array containing 0s, 1s, and 2s in-place without using built-in sort functions.

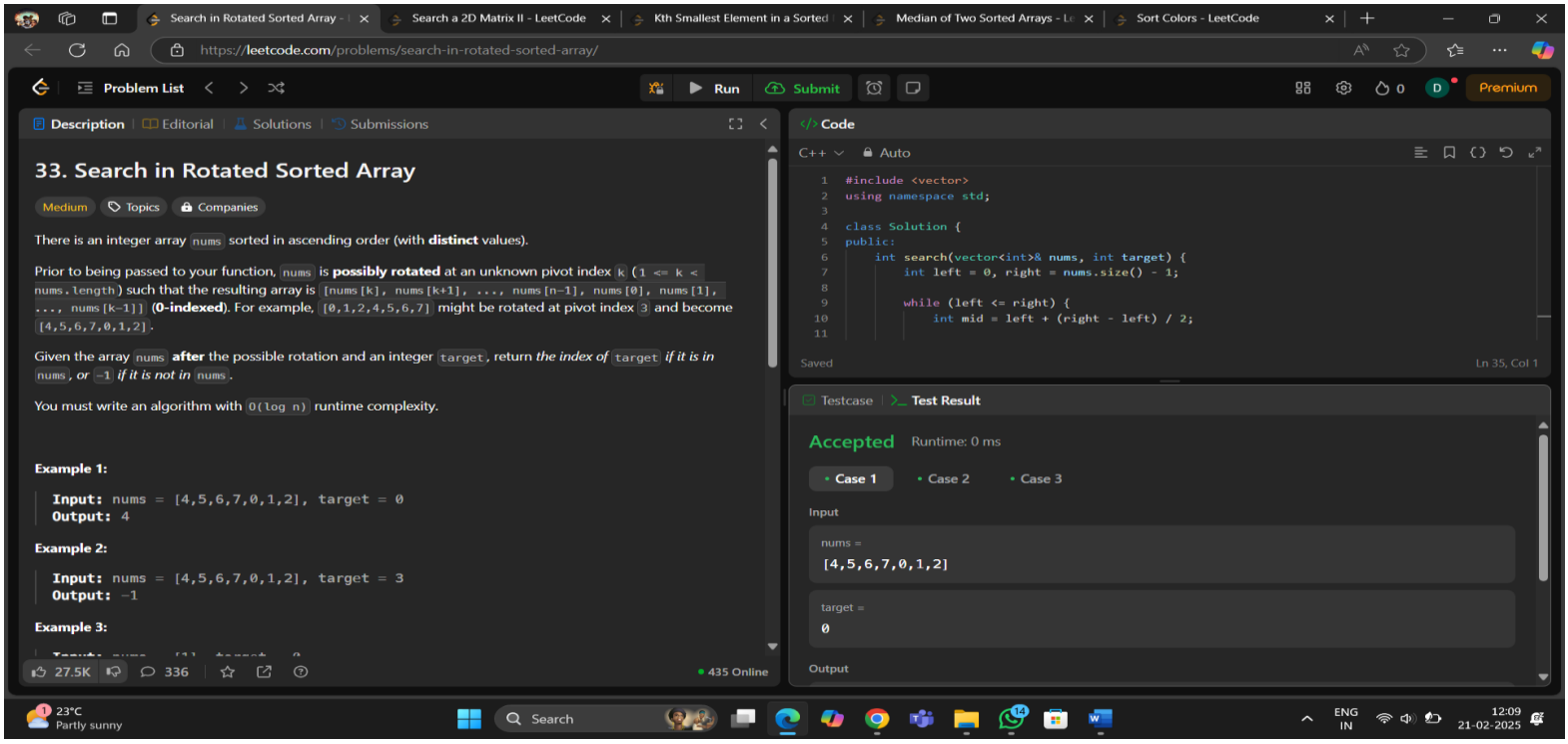
9. Implementation of Code/Output 1 :



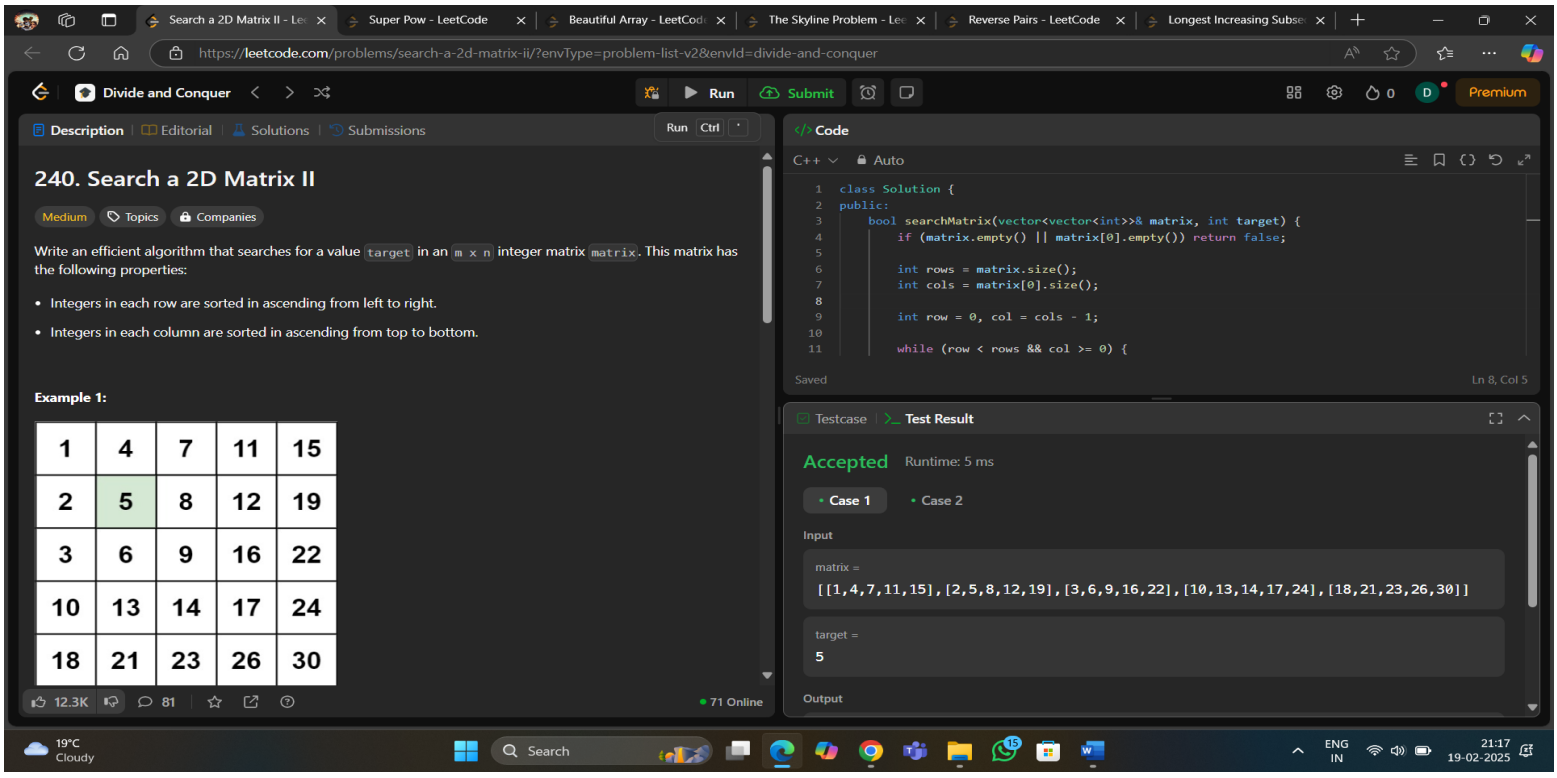
10.Code 2 :



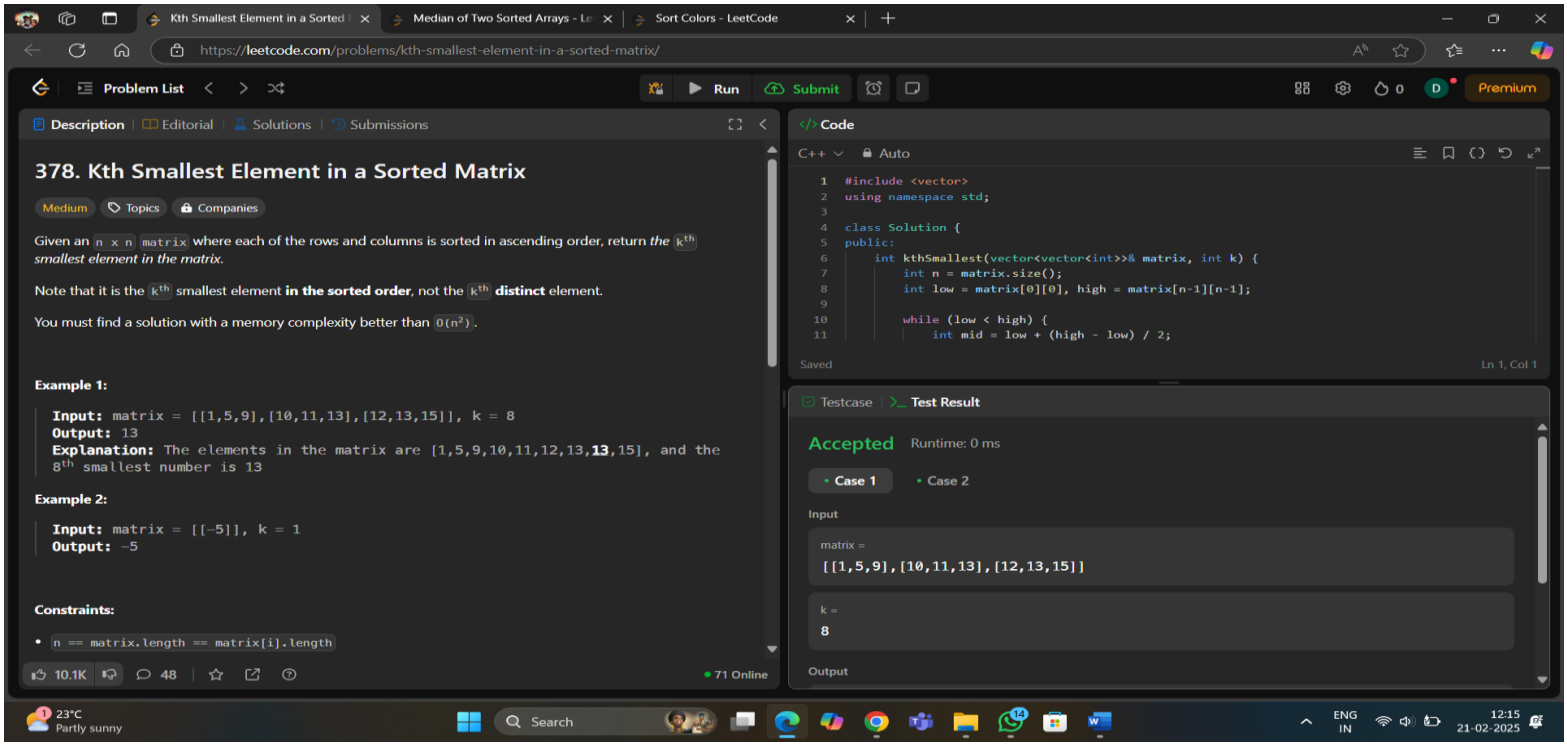
11.Code 3 :



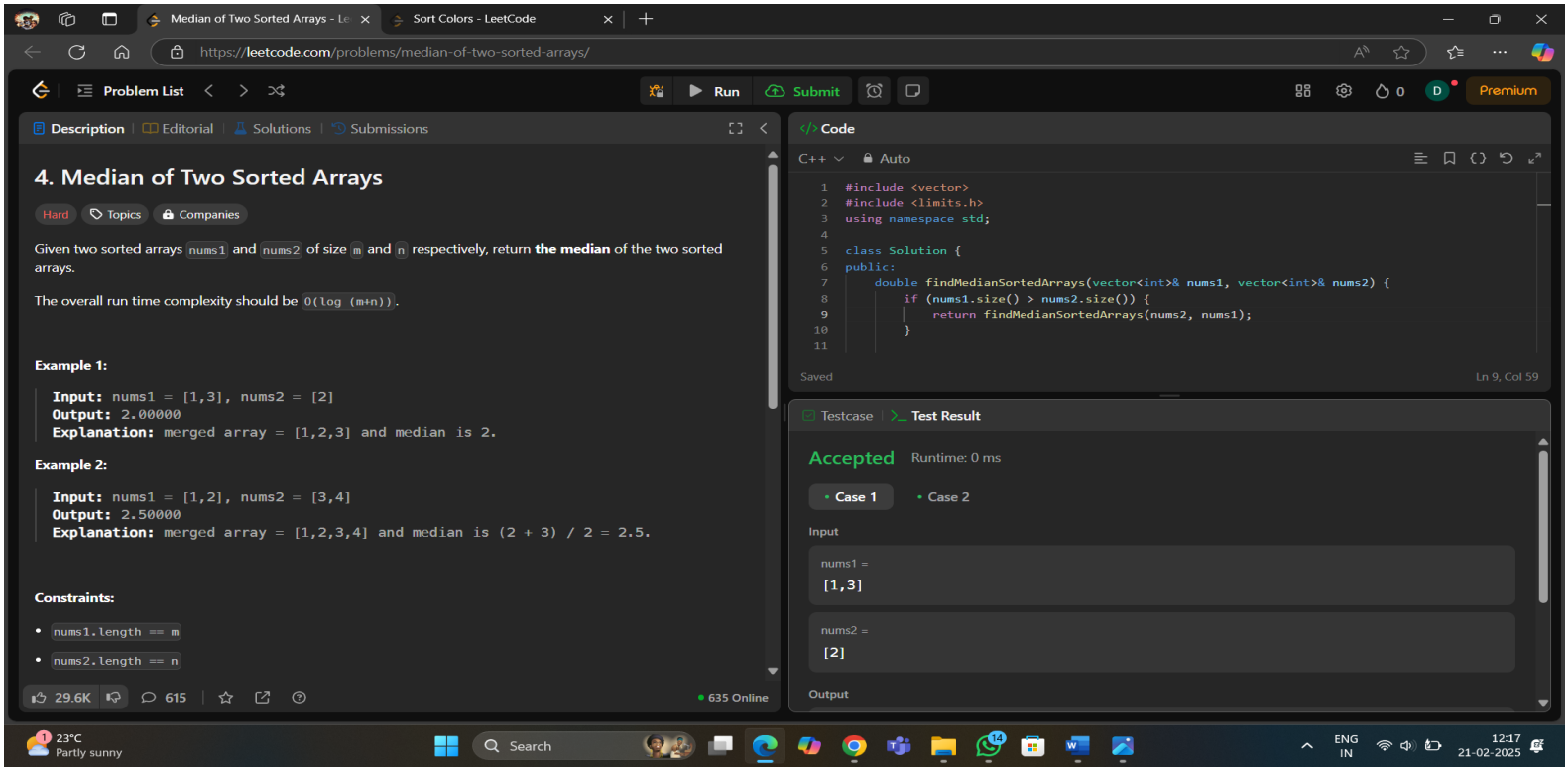
12.Code 4 :



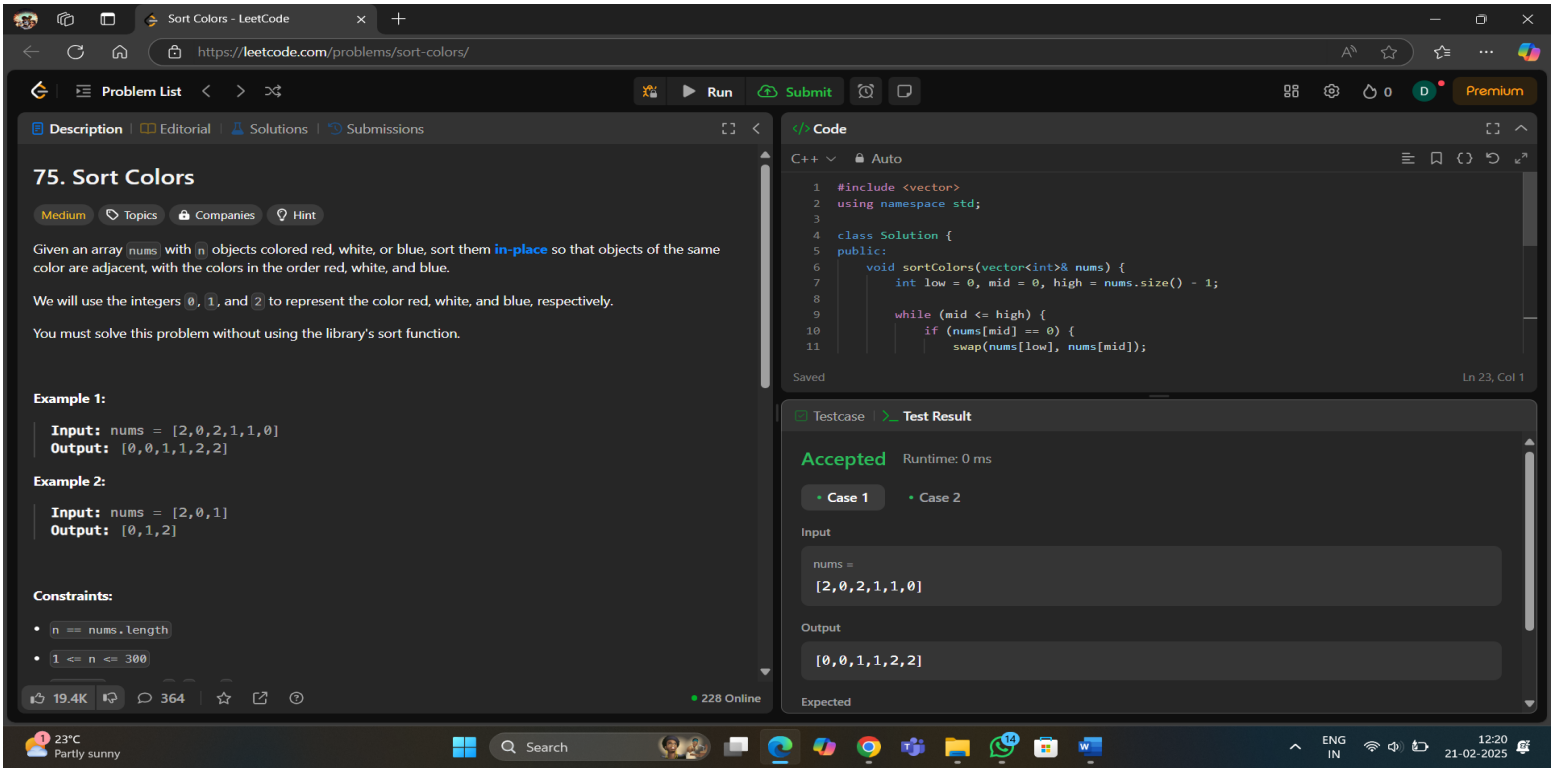
13.Code 5 :



14.Code 6 :



15.Code 7 :



16.Learning Outcome:

- Using a Min-Heap (Priority Queue) to maintain the k largest elements.
- QuickSelect (Hoare's Selection Algorithm) for finding the kth largest element in O(n) average time complexity.
- Sorting + Merging Technique to process overlapping intervals.
- inary Search in a Rotated Array.
- Matrix traversal from the top-right or bottom-left for O(m + n) complexity.
- Using a Min-Heap to extract the smallest k elements efficiently.
- Optimal O(log(min(m, n))) solution instead of naive merging (O(m+n)).
- Three-way partitioning using three pointers (low, mid, high).