Experiment 5

Student Name: Mandeep Sharma **UID:**22BET10014

Branch: BE -IT **Section/Group:**22BET/IOT/702/A

Semester: 6^{th} **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)).

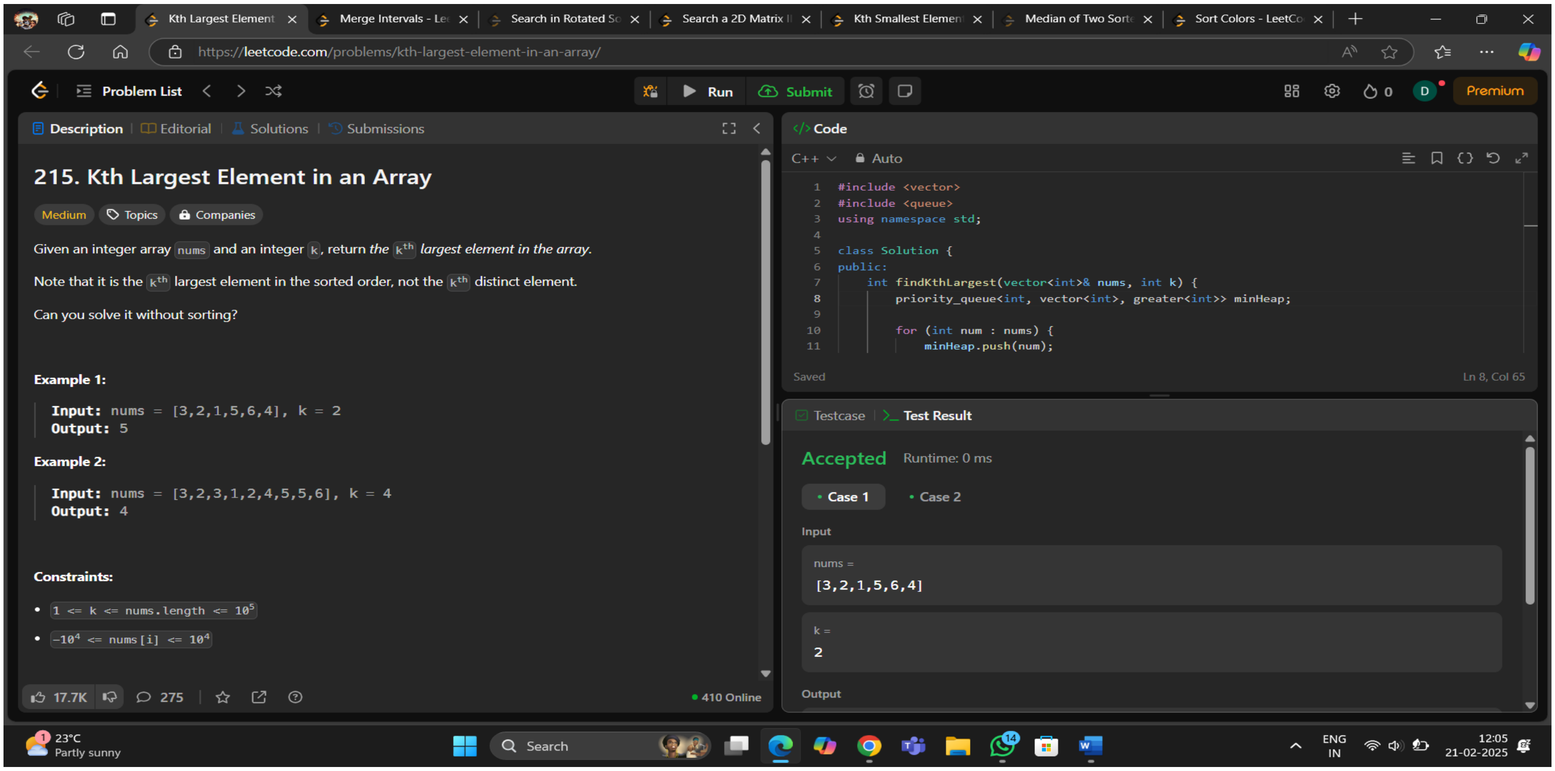
7. Sort Colors :

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 :



215. 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?

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:

- $1 <= k <= nums.length <= 10^5$
- $-10^4 <= nums[i] <= 10^4$

```cpp
#include <vector>
#include <queue>
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);
```

Saved

Ln 8, Col 65

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =

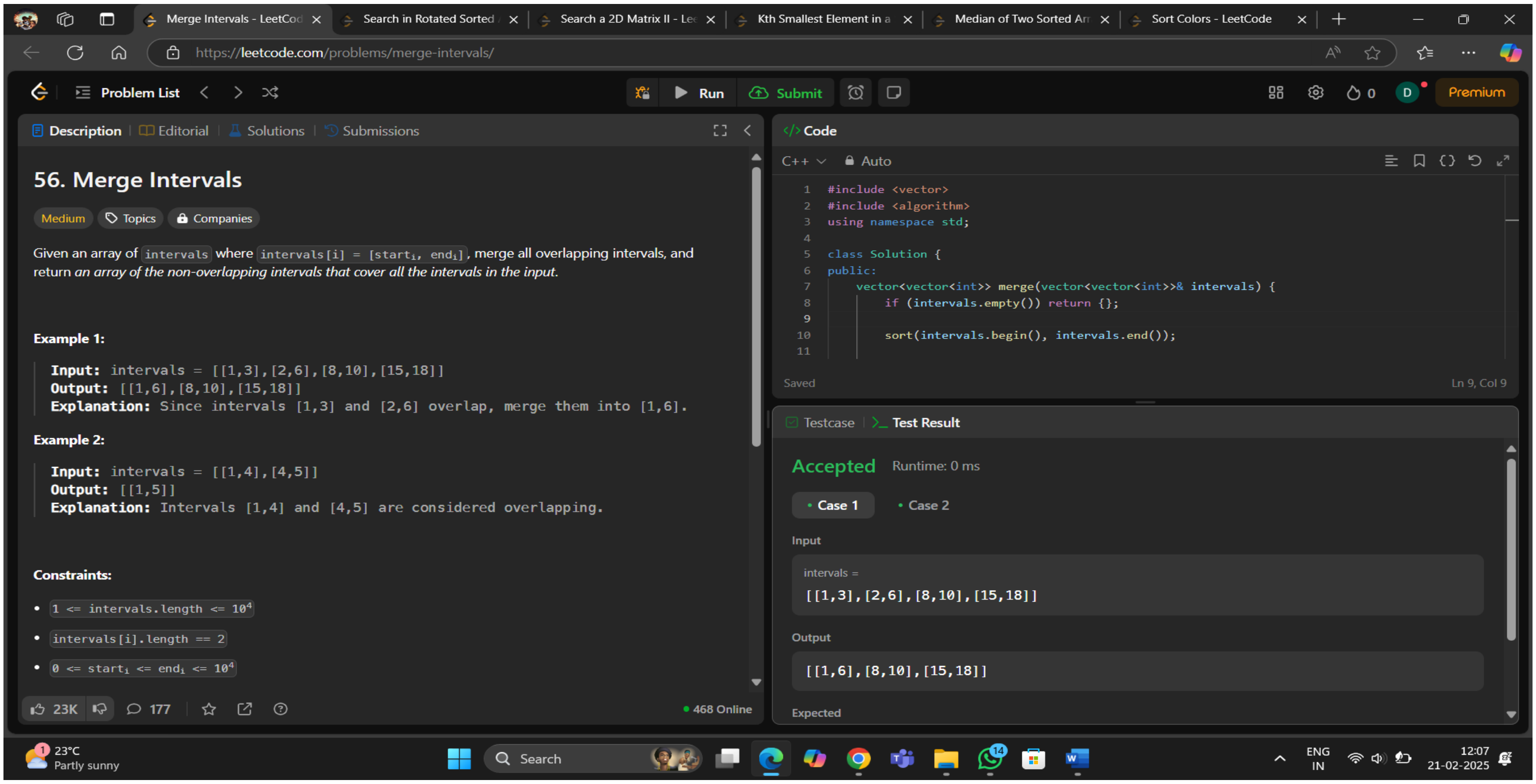
[3,2,1,5,6,4]

k =

2

Output

10.Code 2 :



56. 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.*

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

Constraints:

- $1 <= intervals.length <= 10^4$
- `intervals[i].length == 2`
- $0 <= start_i <= end_i <= 10^4$

```cpp
#include <vector>
#include <algorithm>
using namespace std;

class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        if (intervals.empty()) return {};

        sort(intervals.begin(), intervals.end());

```

Saved

Ln 9, Col 9

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

intervals =

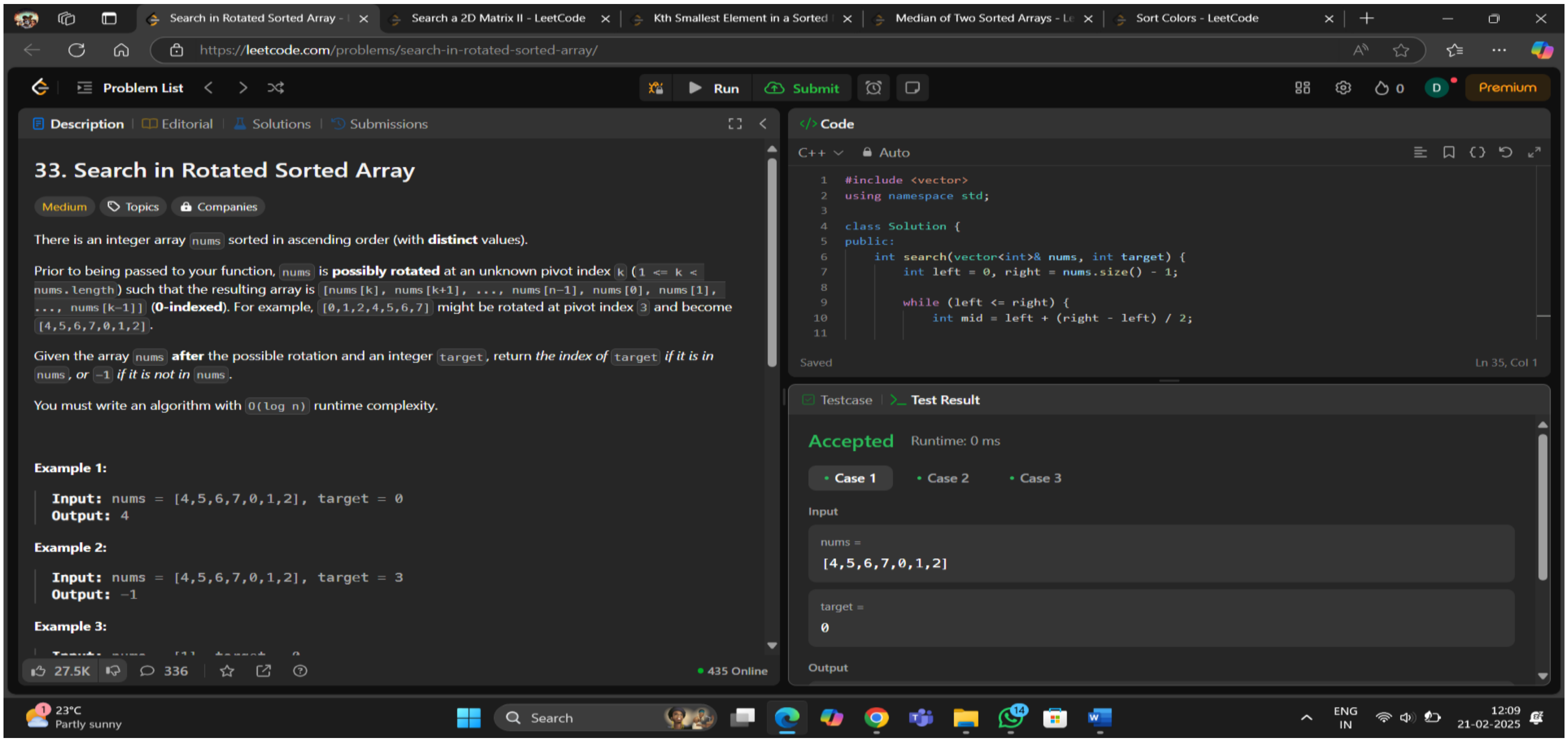
[[1,3],[2,6],[8,10],[15,18]]

Output

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

Expected

11.Code 3 :


Search in Rotated Sorted Array - L
Search a 2D Matrix II - LeetCode
Kth Smallest Element in a Sorted
Median of Two Sorted Arrays - Le
Sort Colors - LeetCode
https://leetcode.com/problems/search-in-rotated-sorted-array/
Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
Code
C++
Auto
Testcase
Test Result
27.5K
336
435 Online
23°C
Partly sunny
Search
ENG
IN
12:09
21-02-2025

33. Search in Rotated Sorted Array

Medium · Topics · Companies

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`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

Example 3:

```cpp
#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;

```

Saved · Ln 35, Col 1

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

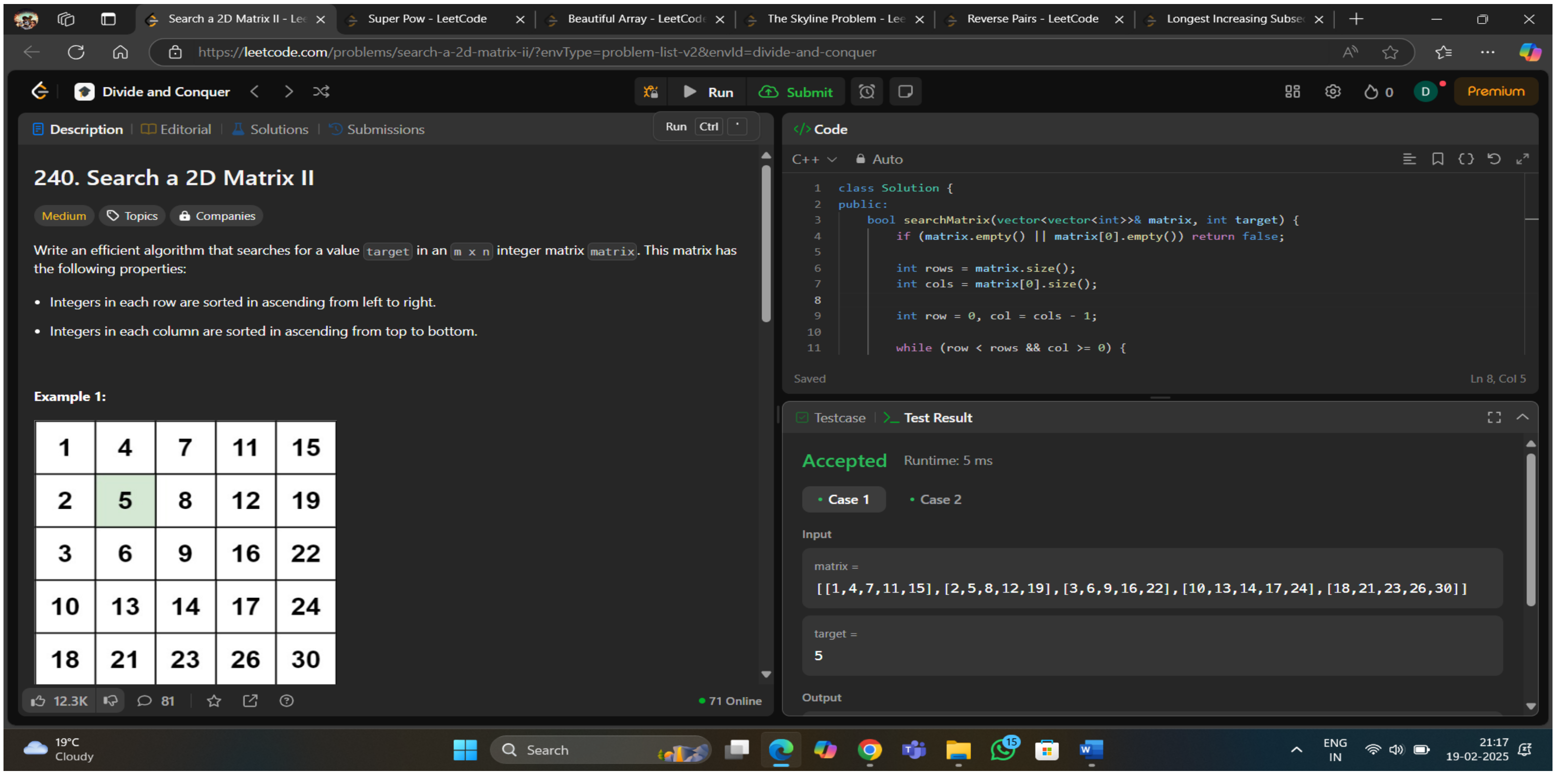
Input

nums =
```
[4,5,6,7,0,1,2]
```

target =
```
0
```

Output

12.Code 4 :



240. Search a 2D Matrix II

Medium · Topics · Companies

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.

Example 1:

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

```cpp
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty() || matrix[0].empty()) return false;

        int rows = matrix.size();
        int cols = matrix[0].size();

        int row = 0, col = cols - 1;

        while (row < rows && col >= 0) {
```

Saved · Ln 8, Col 5

Accepted Runtime: 5 ms

Case 1 · Case 2

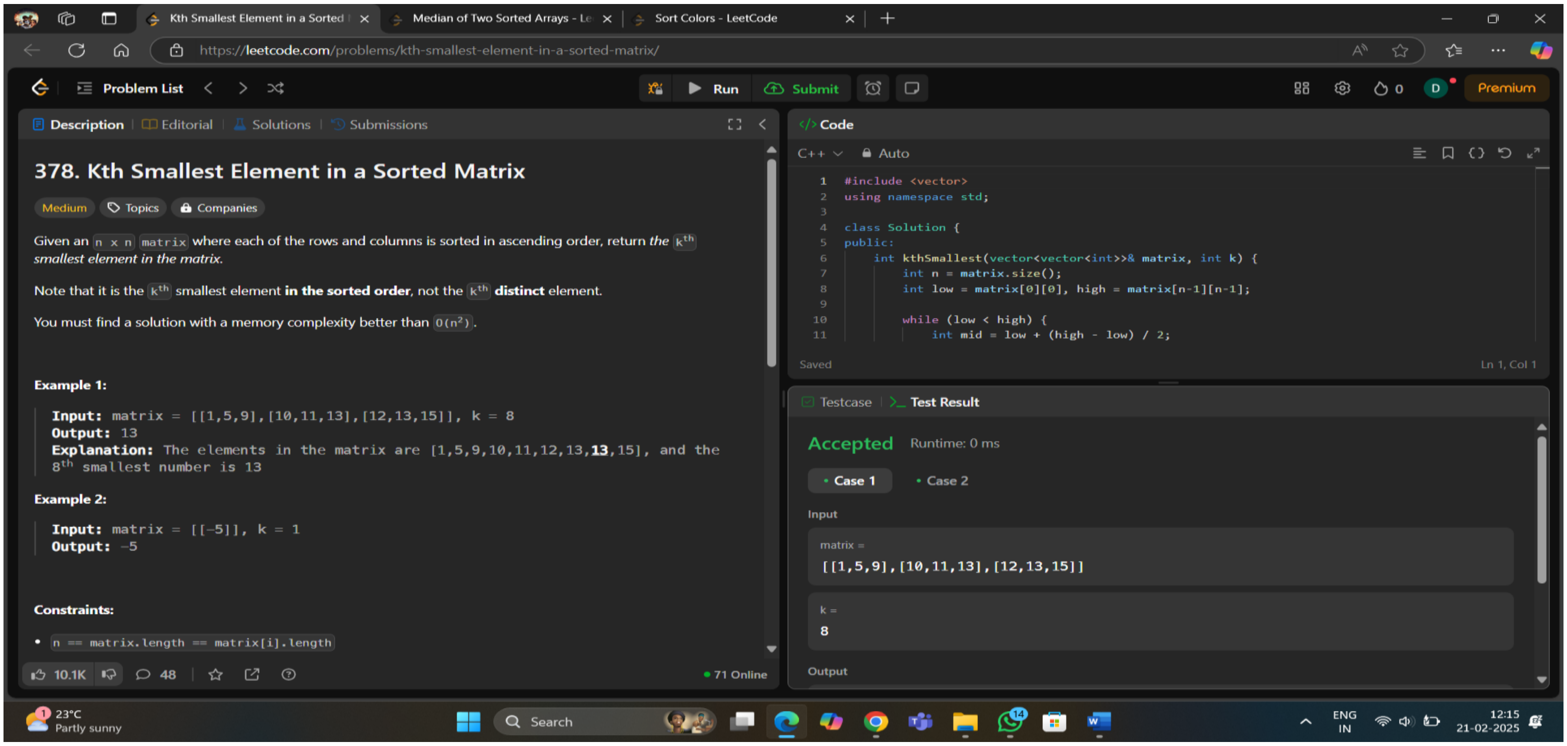
Input

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

13.Code 5 :



378. Kth Smallest Element in a Sorted Matrix

Medium · Topics · Companies

Given an `n x 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(n²)`.

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:

- `n == matrix.length == matrix[i].length`

```cpp
#include <vector>
using namespace std;

class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
        int n = matrix.size();
        int low = matrix[0][0], high = matrix[n-1][n-1];

        while (low < high) {
            int mid = low + (high - low) / 2;
```

Saved · Ln 1, Col 1

Accepted Runtime: 0 ms

• Case 1 • Case 2

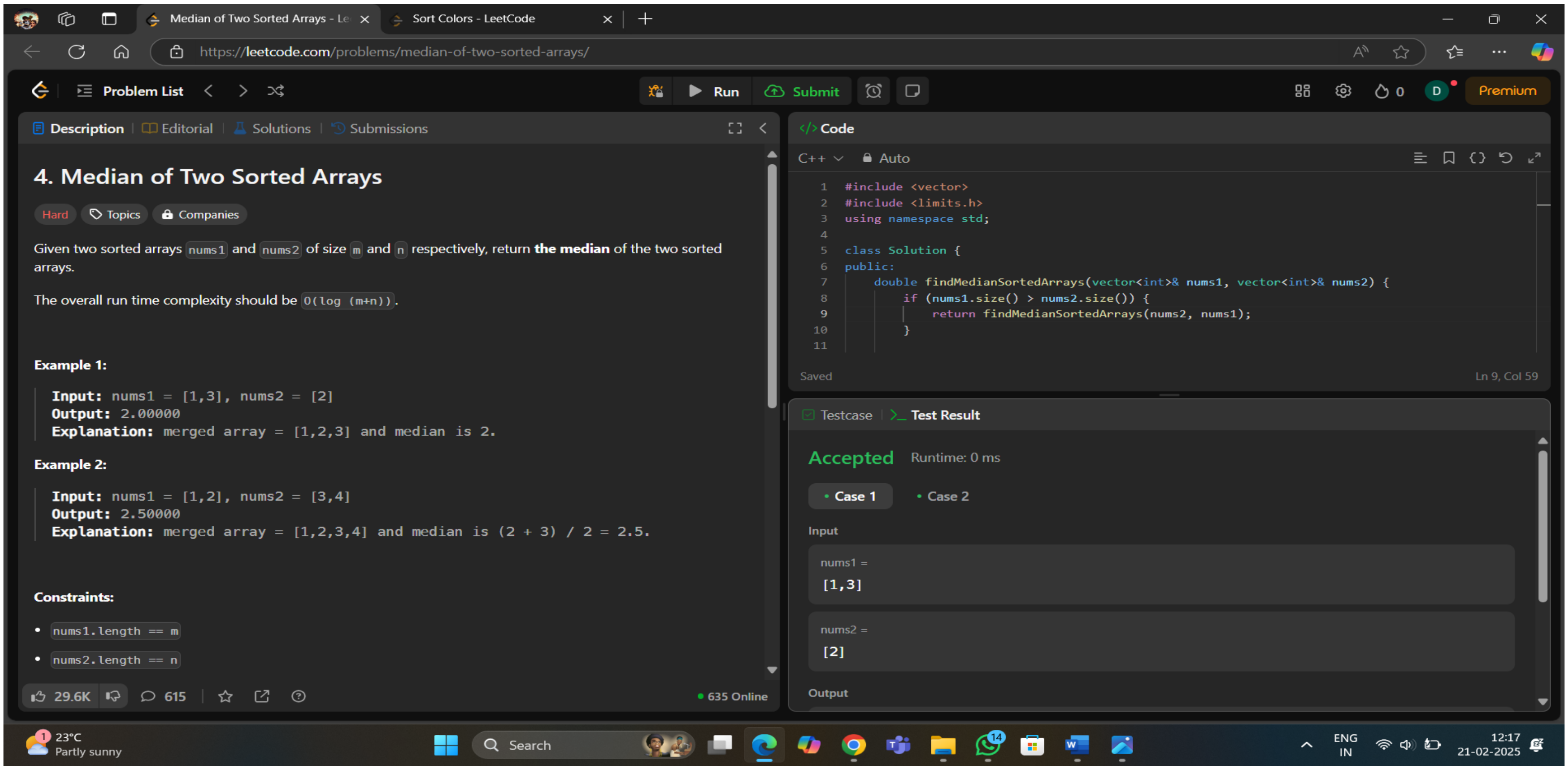
Input

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

k =
8

Output

14.Code 6 :



4. Median of Two Sorted Arrays

Hard · Topics · Companies

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))`.

Example 1:

```
Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.
```

Example 2:

```
Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.
```

Constraints:

- `nums1.length == m`
- `nums2.length == n`

```cpp
#include <vector>
#include <limits.h>
using namespace std;

class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        if (nums1.size() > nums2.size()) {
            return findMedianSortedArrays(nums2, nums1);
        }

```

Saved · Ln 9, Col 59

Accepted Runtime: 0 ms

• Case 1 • Case 2

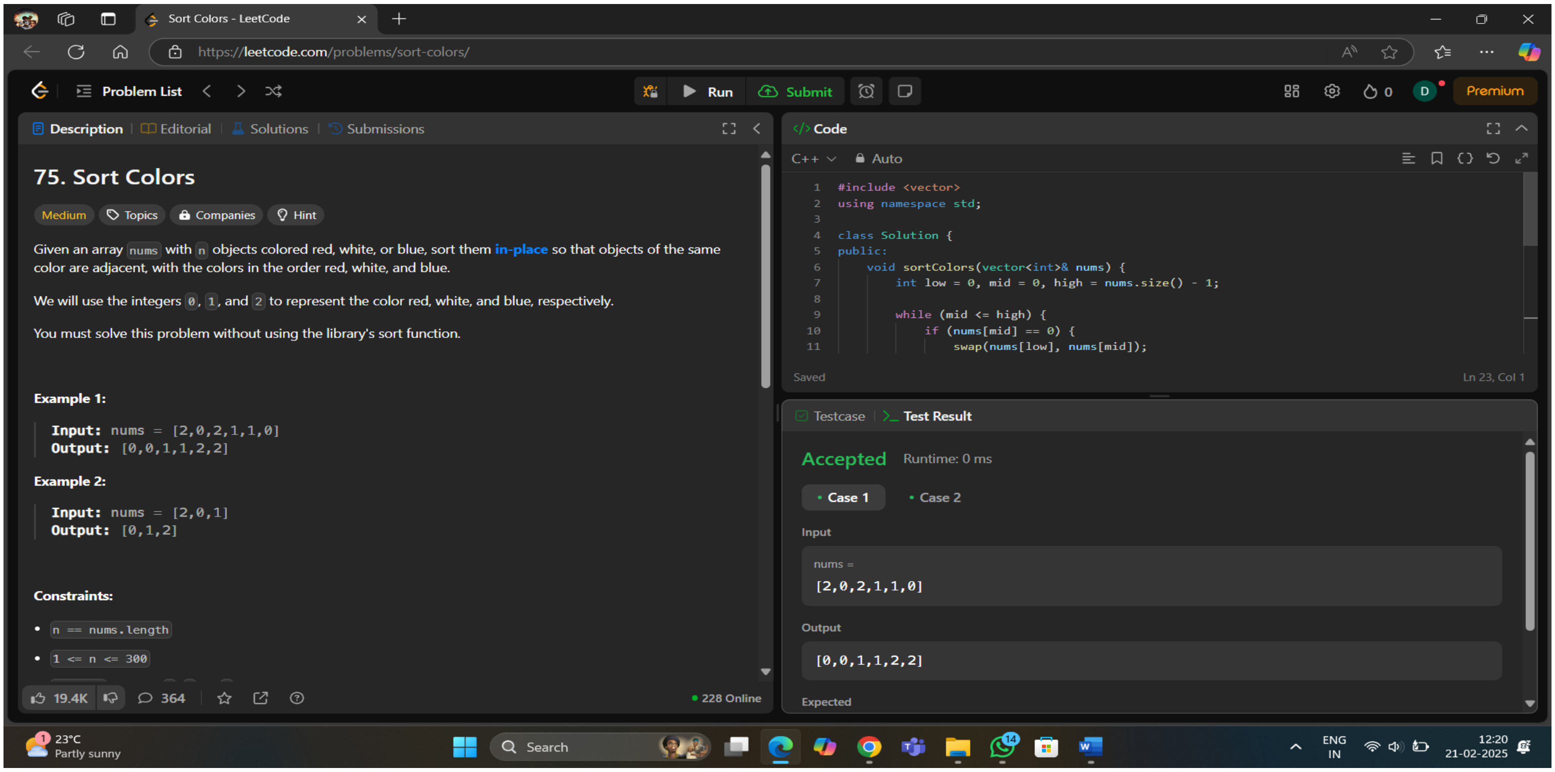
Input

nums1 =
[1,3]

nums2 =
[2]

Output

15.Code 7 :



75. 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 this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```

Constraints:

- `n == nums.length`
- `1 <= n <= 300`

```cpp
#include <vector>
using namespace std;

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]);
```

Saved

Ln 23, Col 1

Accepted Runtime: 0 ms

Input

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

Output

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

Expected

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).