# 191.Number of 1 bits:

**CODE:**

class Solution {

public:

int hammingWeight(uint32\_t n) {

int res = 0;

for (int i = 0; i < 32; i++) {

if ((n >> i) & 1) {

res += 1;

}

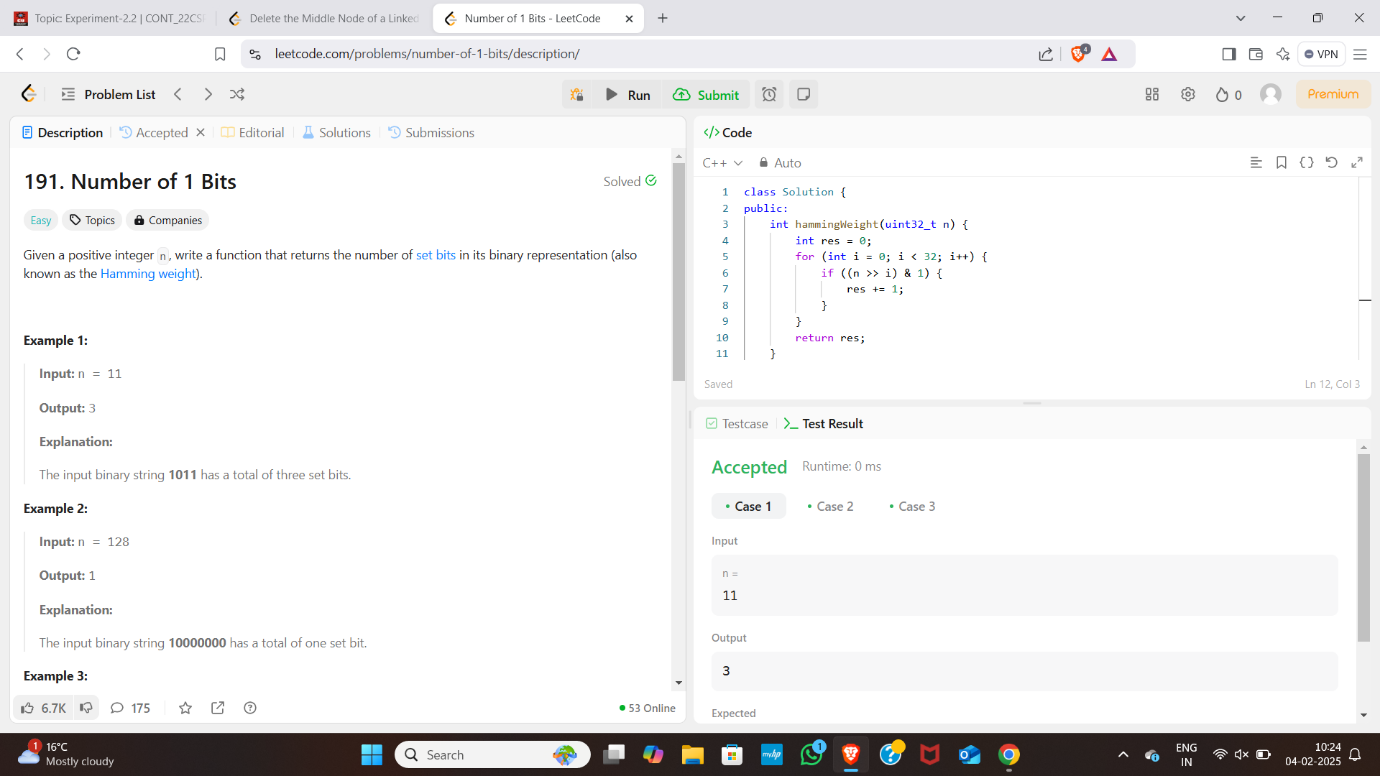
}

return res;

}

};

**OUTPUT:**

****

# 240. Search a 2D Matrix II:

**CODE:**

class Solution {

public:

bool searchMatrix(vector<vector<int>>& matrix, int target) {

int cols = matrix[0].size() - 1;

int n = matrix.size() - 1;

int rows = 0;

while(rows <= n && cols >= 0){

int toCompare = matrix[rows][cols];

if(toCompare > target){

cols--;

}else if(toCompare < target){

rows++;

}else{

return true;

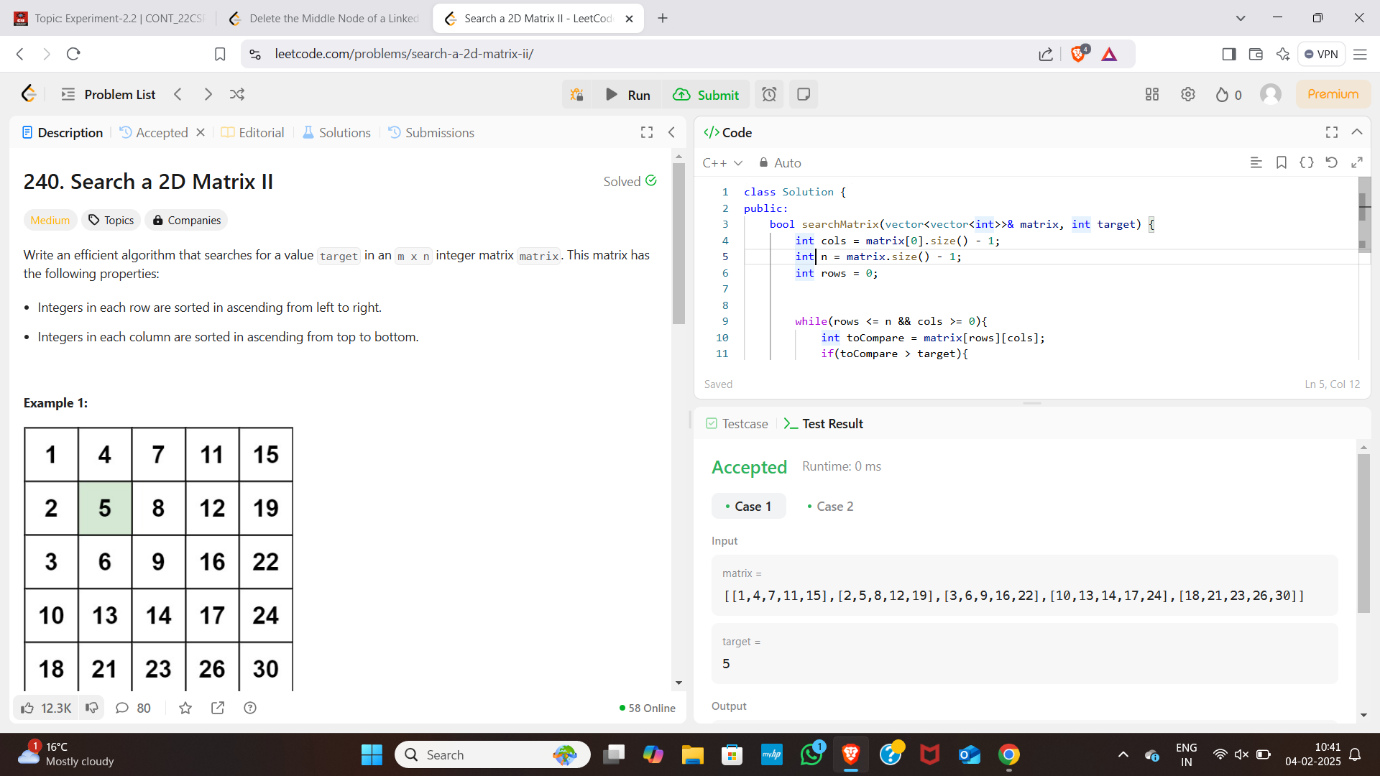
}

}

return false;

}

};

**OUTPUT:** 

# 53.Max Subarray:

**CODE:**

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int maxSum = nums[0];

int currentSum = nums[0];

for (int i = 1; i < nums.size(); i++) {

currentSum = max(nums[i], currentSum + nums[i]);

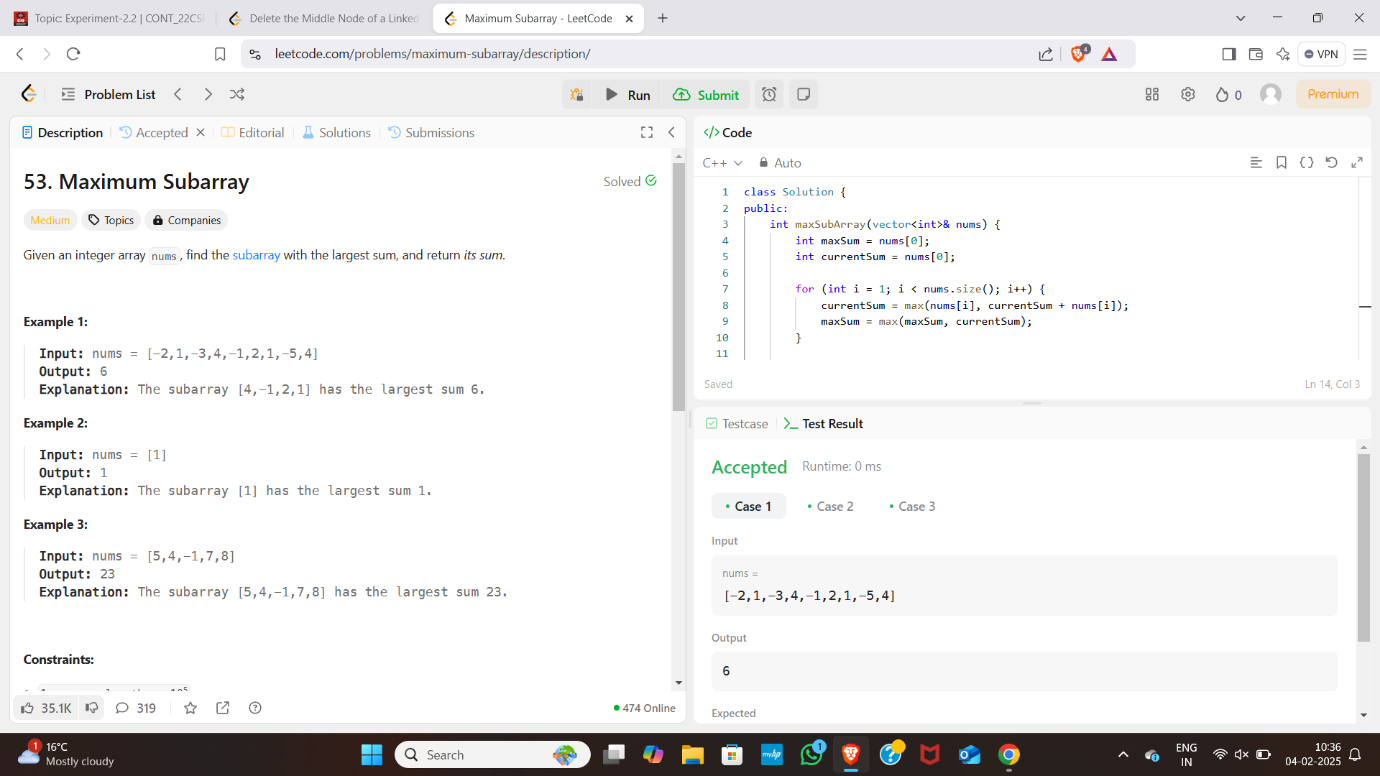
maxSum = max(maxSum, currentSum);

}

return maxSum;

}

};

**OUTPUT:**

# 56. Merge Intervals:

**CODE:**

class Solution {

public:

    vector<vector<int>> merge(vector<vector<int>>& intervals) {

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

        int k = 0;

        for (int i = 1; i < intervals.size(); i++) {

            if (intervals[k][1] >= intervals[i][0]) {

                intervals[k][1] = max(intervals[k][1], intervals[i][1]);

            } else {

                k++;

                intervals[k] = intervals[i];

            }

        }

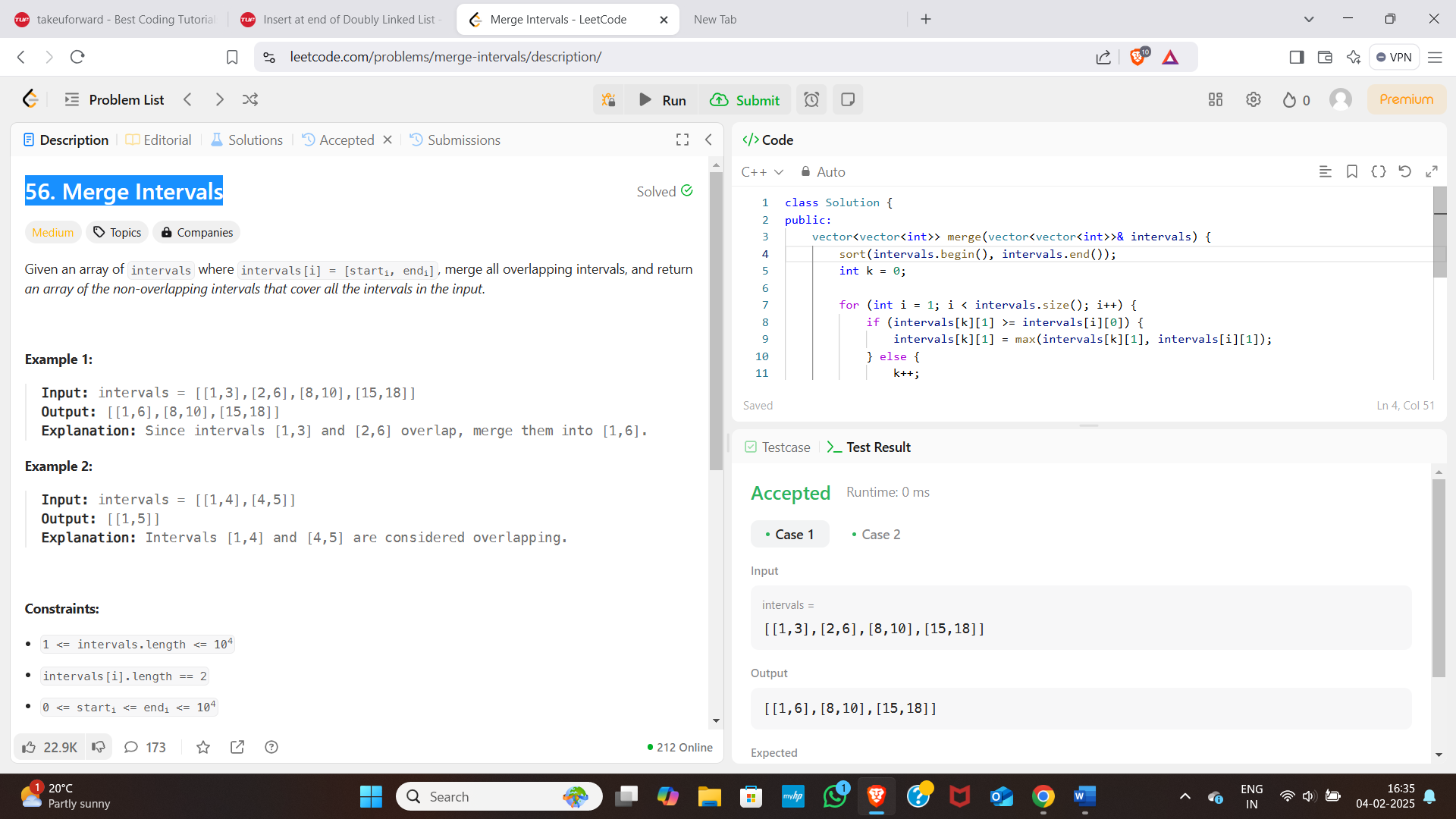
        intervals.resize(k + 1);

        return intervals;

    }

};

**OUTPUT**



# 493.[Reverse Pairs](https://leetcode.com/problems/reverse-pairs/description/)

**CODE:**

class Solution {

public:

    int reversePairs(vector<int>& nums) {

        int n = nums.size();

        long long reversePairsCount = 0;

        for(int i=0; i<n-1; i++){

            for(int j=i+1; j<n; j++){

                if(nums[i] > 2\*(long long)nums[j]){

                    reversePairsCount++

}

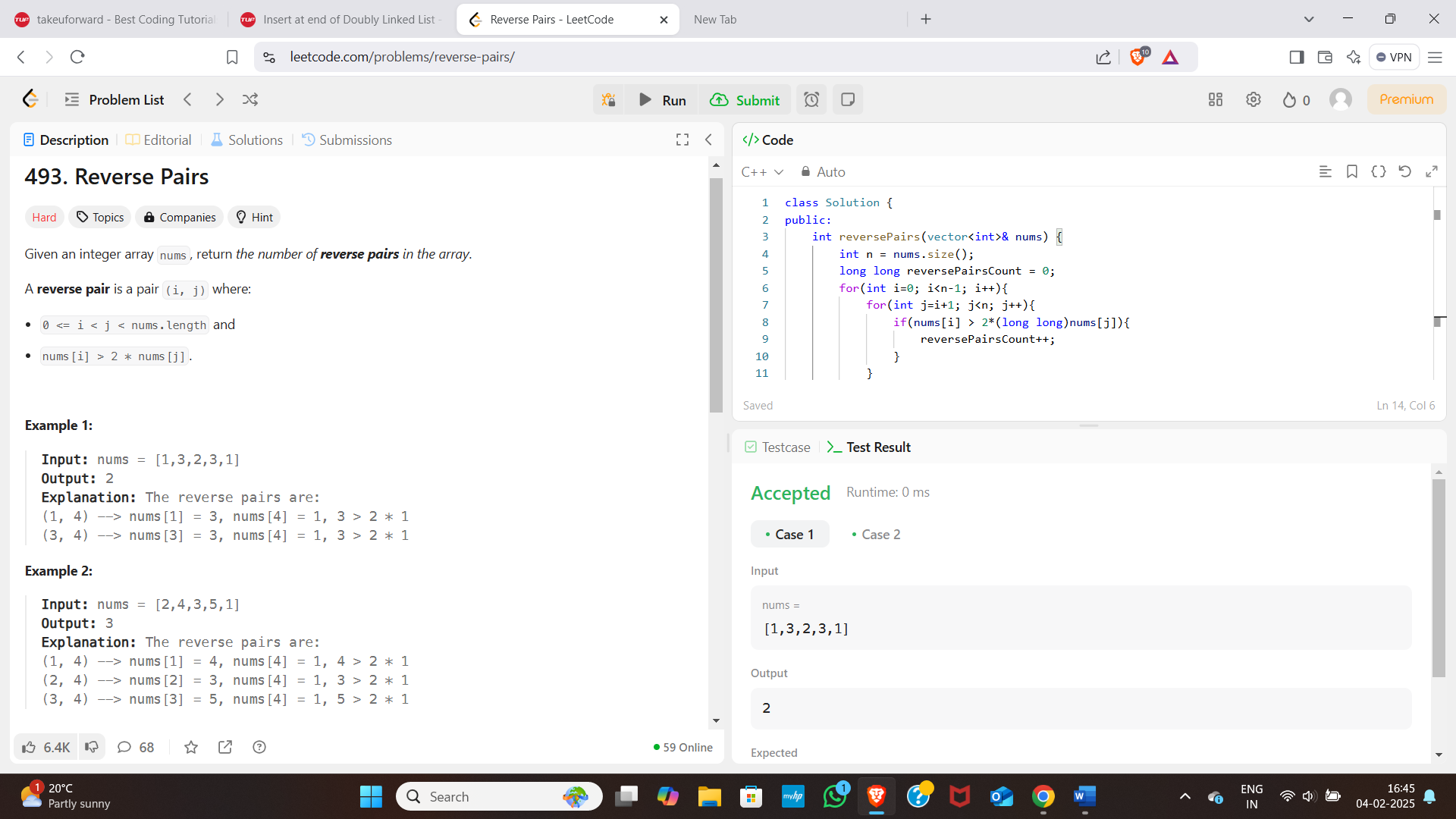
            }

        }

        return reversePairsCount;

    }

};

**OUTPUT:** 

# [**88. Merge Sorted Array**](https://leetcode.com/problems/merge-sorted-array/)

**CODE:**

class Solution {

public:

    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

        for(int i=0; i<n; i++){

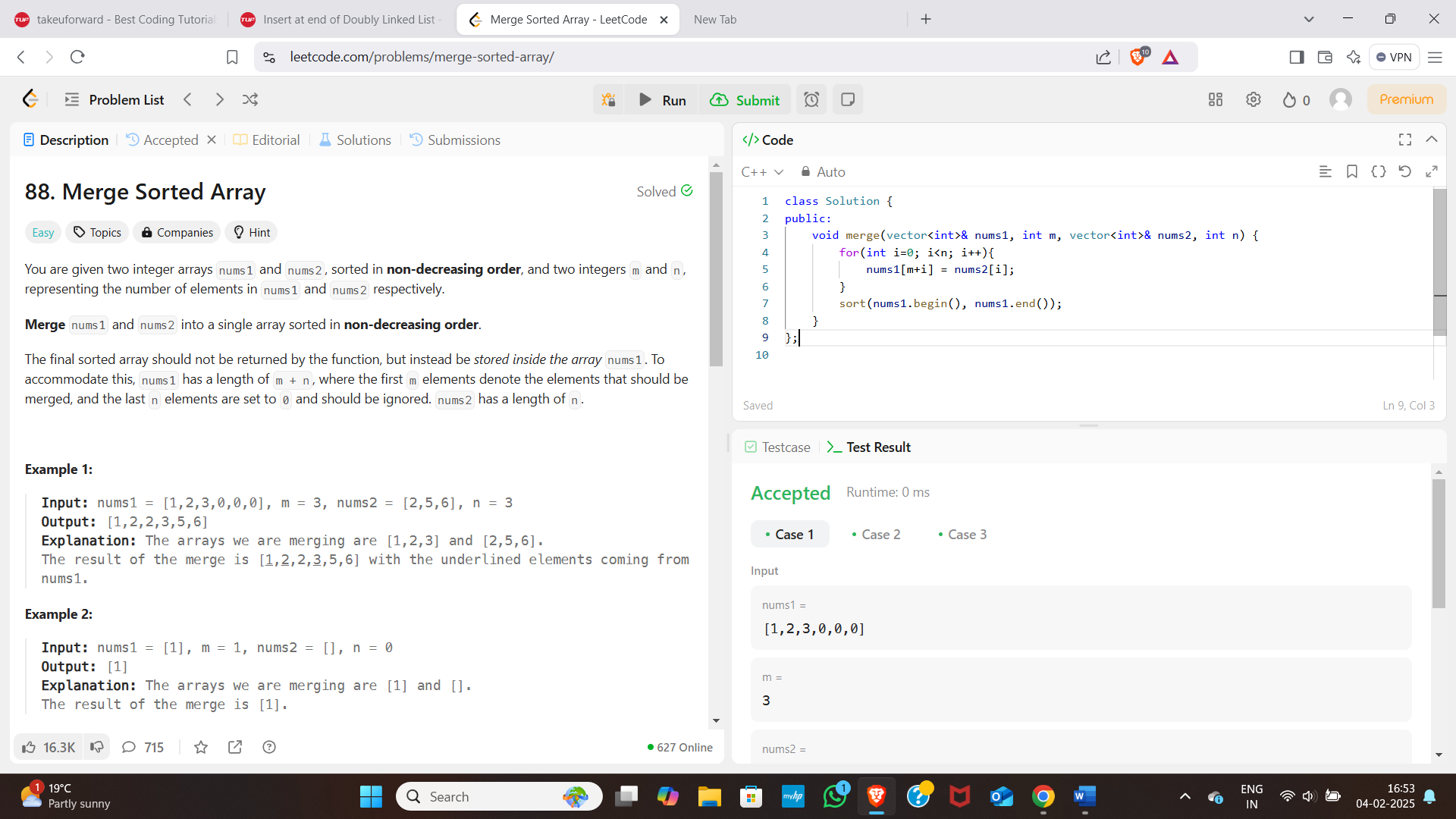
            nums1[m+i] = nums2[i];

        }

        sort(nums1.begin(), nums1.end());

    }

};

**OUTPUT:** 

# [**347. Top K Frequent Element**](https://leetcode.com/problems/top-k-frequent-elements/):

**CODE:**

class Solution {

public:

    vector<int> topKFrequent(vector<int>& nums, int k) {

        int n = nums.size();

        unordered\_map<int, int> map;

        vector<int> ans;

        for (int &x : nums) map[x]++;

        vector<vector<int>> arr(n + 1);

        for (auto [a, b] : map) arr[b].push\_back(a);

        for (int i = n; i > 0; i--) {

            for (int &x : arr[i]) {

                if (ans.size() == k) return ans;

                ans.push\_back(x);

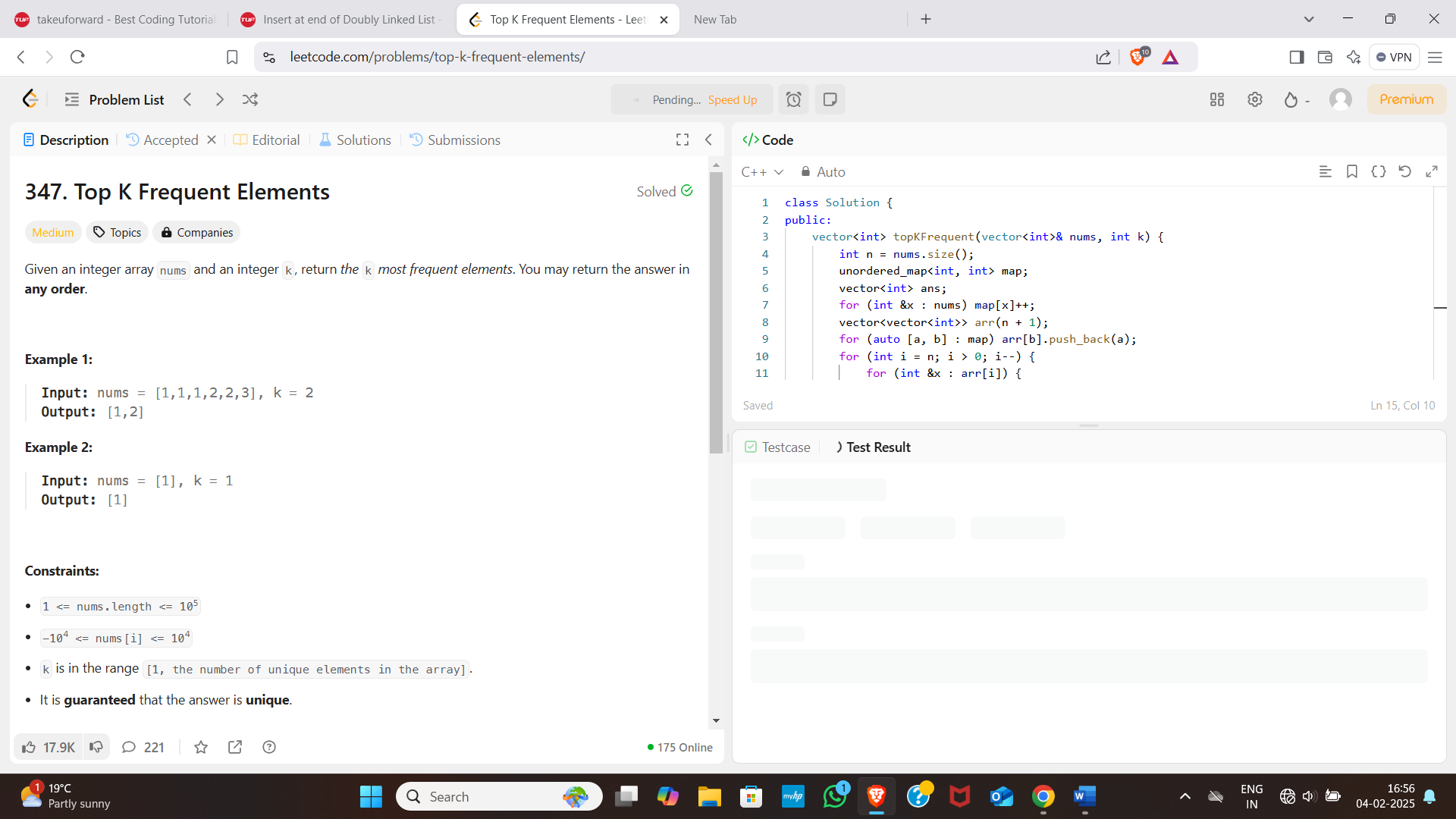
            }

        }

        return ans;

    }

};

**OUTPUT:** 

# [**215. Kth Largest Element in an Array**](https://leetcode.com/problems/kth-largest-element-in-an-array/):

**CODE:**

class Solution {

public:

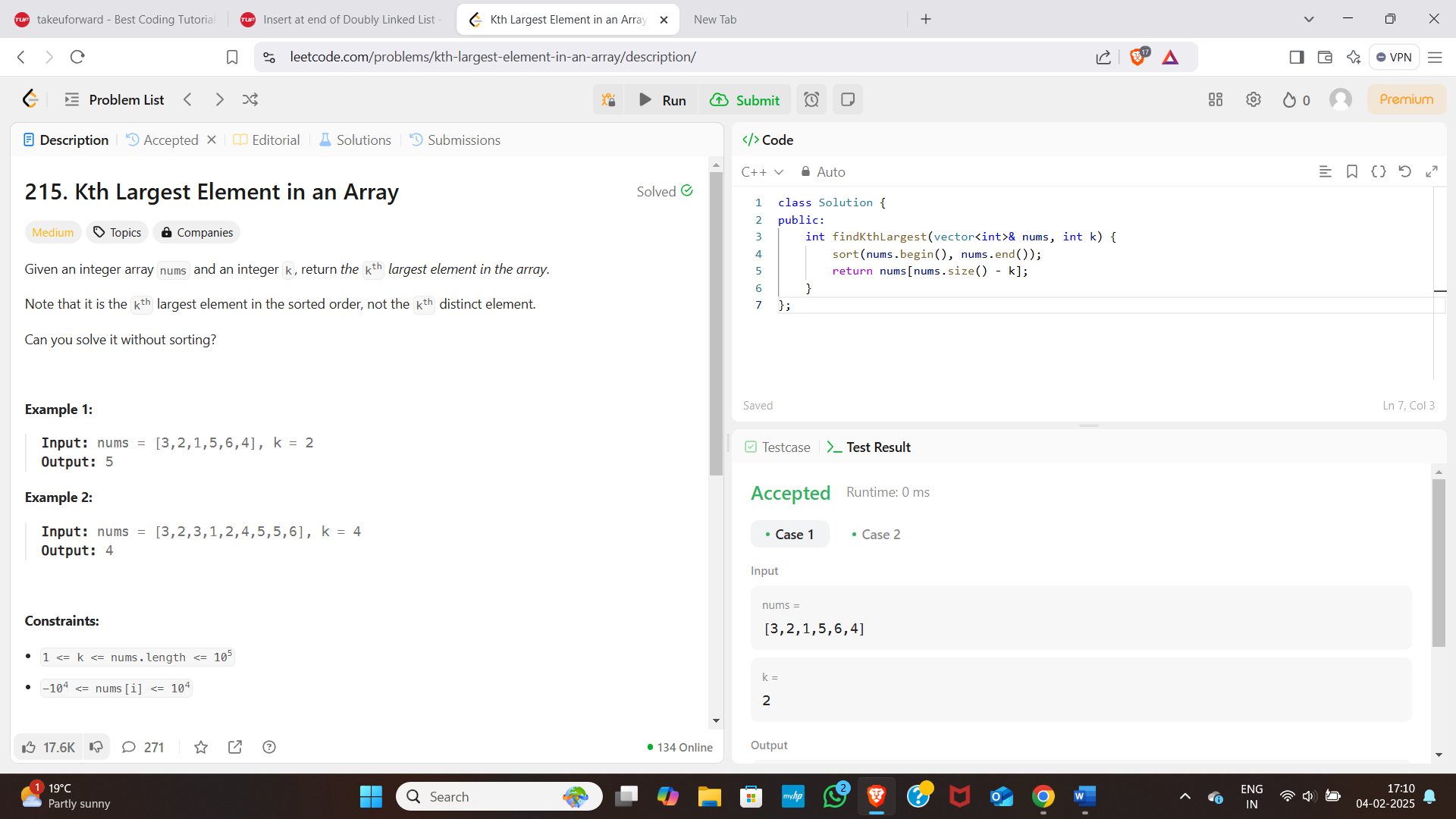
int findKthLargest(vector<int>& nums, int k) {

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

return nums[nums.size() - k];

}

};

**OUTPUT:** 

# [**33. Search in Rotated Sorted Array**](https://leetcode.com/problems/search-in-rotated-sorted-array/):

**CODE:**

class Solution {

public:

    int search(vector<int>& nums, int target) {

        int n = nums.size();

        int i = 0;

        while (i < n - 1 && nums[i] < nums[i + 1]) i++;

        int low, high;

        if (target >= nums[0]) low = 0, high = i;

        else low = i + 1, high = n - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            if (nums[mid] == target) return mid;

            else if (nums[mid] > target) high = mid - 1;

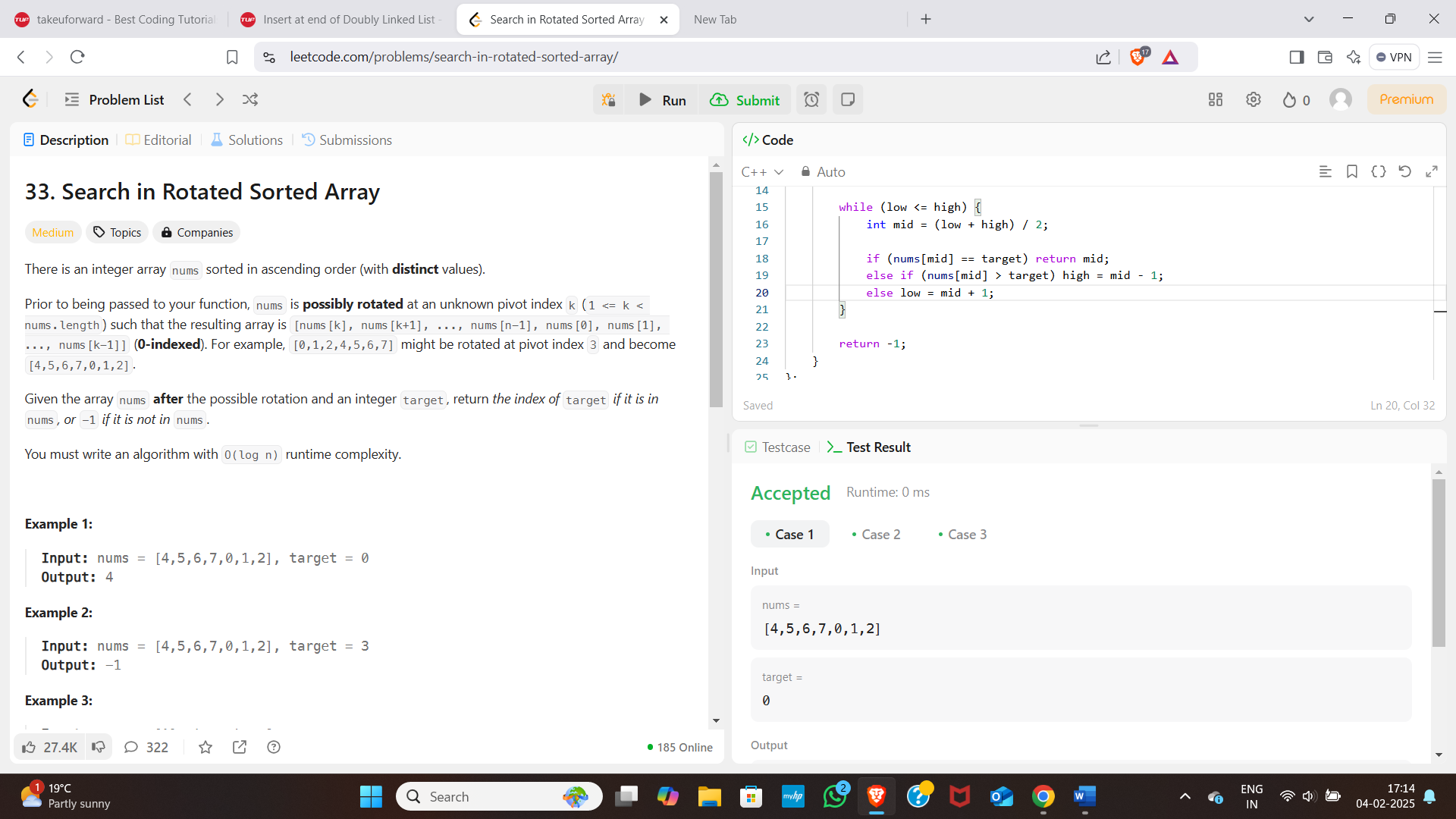
            else low = mid + 1;

        }

        return -1;

    }

};

**OUTPUT:** 

# [**162. Find Peak Element**](https://leetcode.com/problems/find-peak-element/):

**CODE:**

class Solution {

public:

    int findPeakElement(vector<int>& nums) {

        if(nums.size() == 1) return 0;

        int l = 1;

        int h = nums.size() - 2;

        if(nums[0] > nums[1]) return 0;

        if(nums[nums.size() - 1] > nums[nums.size() - 2]) return nums.size() - 1;

        while(l <= h){

            int mid = (l+h)/2;

            if(nums[mid] > nums[mid+1] && nums[mid] > nums[mid-1]){

                return mid;

            }

            else if(nums[mid] < nums[mid+1]){

                l = mid + 1;

            } else {

                h = mid - 1;

            }

        }

        return -1;

    }

};

**OUTPUT:** 