**ADVANCED CODING- 2**

**ASSIGNMENT 3**

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

**PROGRAM:**

#include <vector>

#include <queue>

class Solution {

public:

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

std::priority\_queue<int, std::vector<int>, std::greater<int>> minHeap;

for (int num : nums) {

minHeap.push(num);

if (minHeap.size() > k) {

minHeap.pop();

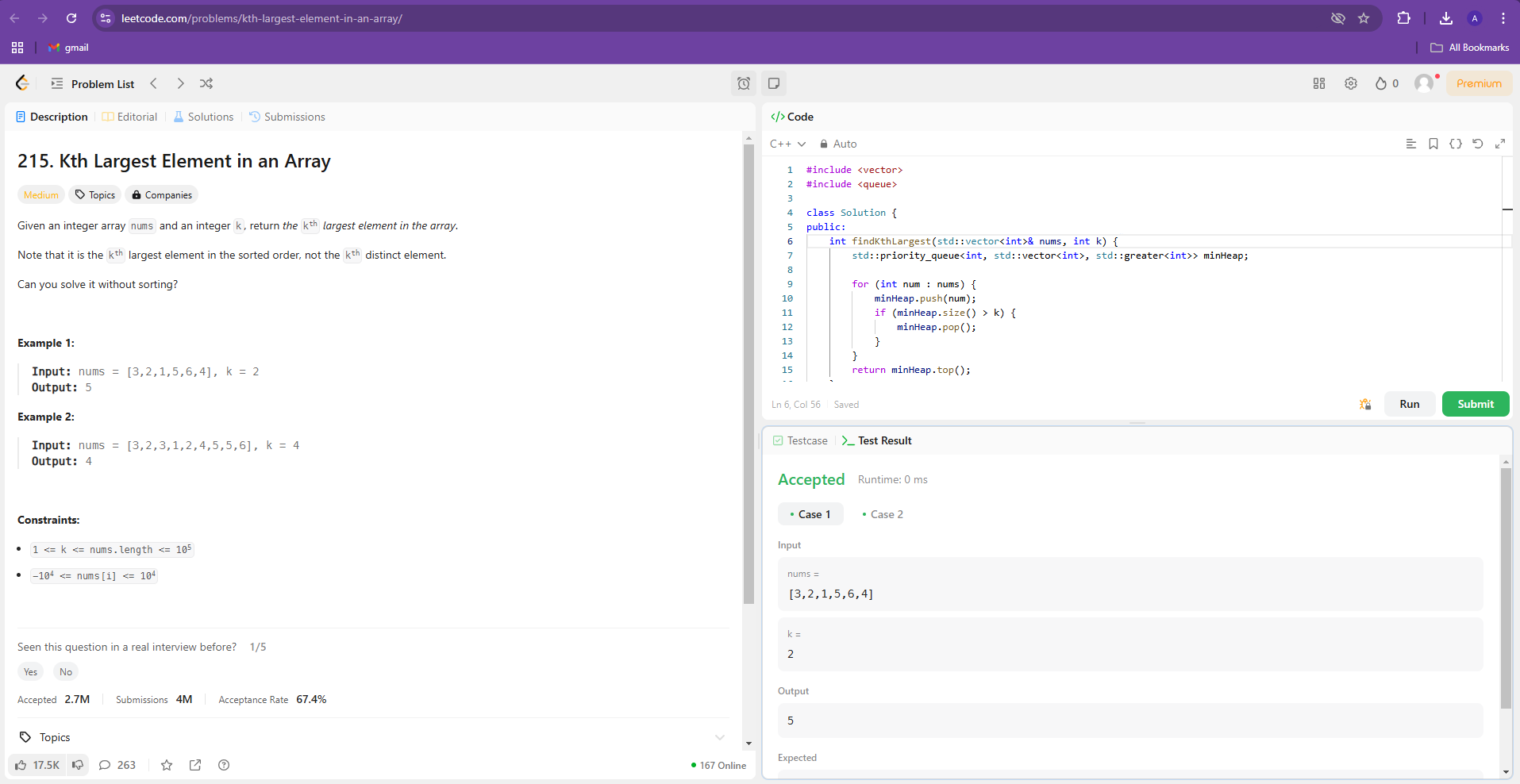
}

}

return minHeap.top();

}

};



1. You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.

**PROGRAM:**

#include <vector>

#include <queue>

using namespace std;

class Solution {

public:

    ListNode\* mergeKLists(vector<ListNode\*>& lists) {

        auto compare = [](ListNode\* a, ListNode\* b) {

            return a->val > b->val;

        };

        priority\_queue<ListNode\*, vector<ListNode\*>, decltype(compare)> minHeap(compare);

        for (ListNode\* list : lists) {

            if (list) {

                minHeap.push(list);

            }

        }

        ListNode\* dummy = new ListNode();

        ListNode\* current = dummy;

        while (!minHeap.empty()) {

            ListNode\* smallest = minHeap.top();

            minHeap.pop();

            current->next = smallest;

            current = current->next;

            if (smallest->next) {

                minHeap.push(smallest->next);

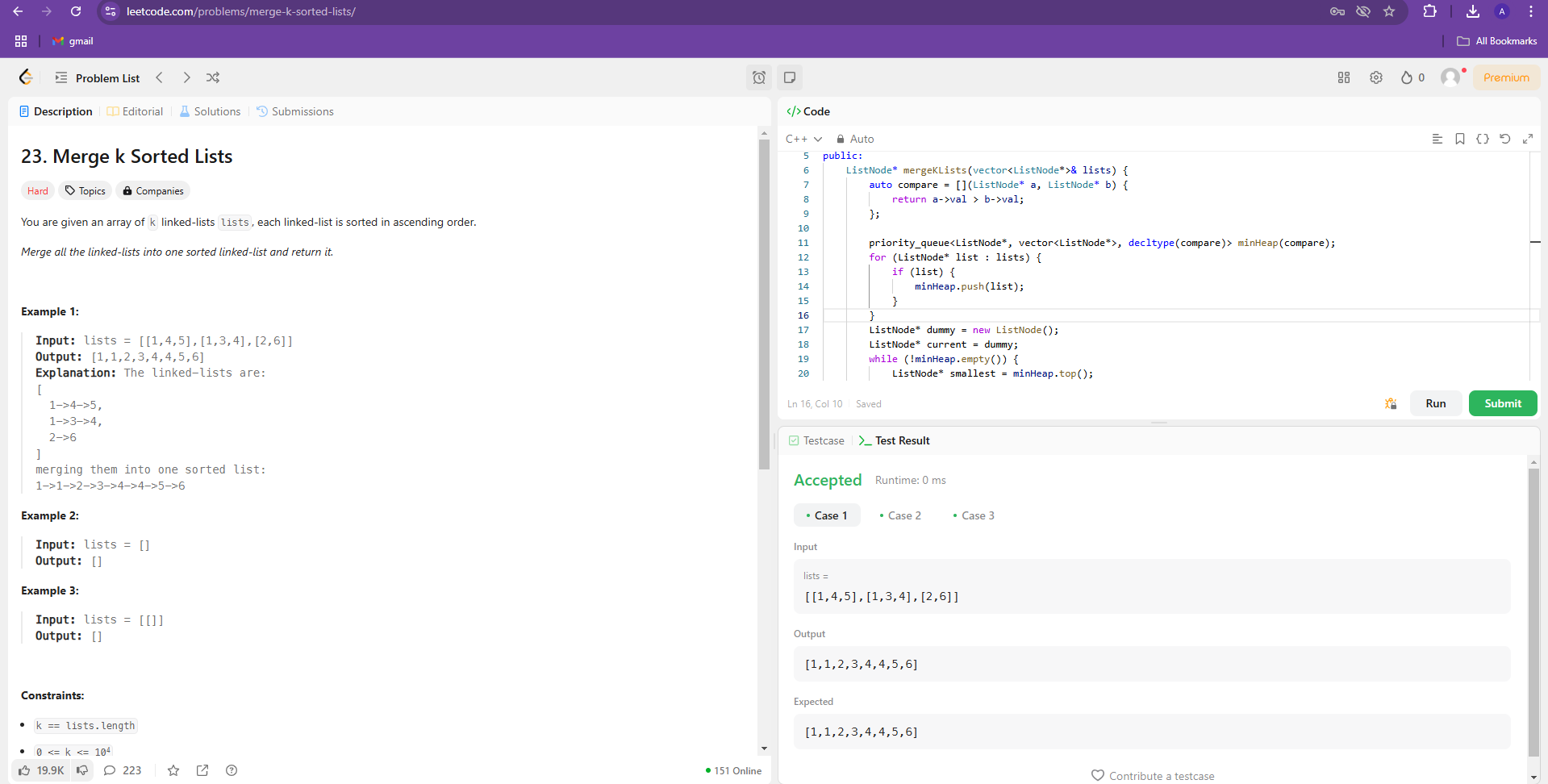
            }

        }

        return dummy->next;

    }

};



1. Design your implementation of the circular double-ended queue (deque).

**PROGRAM**

class MyCircularDeque {

private:

    int\* deque;

    int front;

    int rear;

    int size;

    int capacity;

public:

    MyCircularDeque(int k) {

        capacity = k;

        deque = new int[k];

        front = 0;

        rear = -1;

        size = 0;

    }

    bool insertFront(int value) {

        if (isFull()) return false;

        front = (front - 1 + capacity) % capacity;

        deque[front] = value;

        size++;

        if (rear == -1) rear = front;

        return true;

    }

    bool insertLast(int value) {

        if (isFull()) return false;

        rear = (rear + 1) % capacity;

        deque[rear] = value;

        size++;

        return true;

    }

    bool deleteFront() {

        if (isEmpty()) return false;

        front = (front + 1) % capacity;

        size--;

        if (isEmpty()) rear = -1;

        return true;

    }

    bool deleteLast() {

        if (isEmpty()) return false;

        rear = (rear - 1 + capacity) % capacity;

        size--;

        if (isEmpty()) rear = -1;

        return true;

    }

    int getFront() {

        if (isEmpty()) return -1;

        return deque[front];

    }

    int getRear() {

        if (isEmpty()) return -1;

        return deque[rear];

    }

    bool isEmpty() {

        return size == 0;

    }

    bool isFull() {

        return size == capacity;

    }

    ~MyCircularDeque() {

        delete[] deque;

    }

};

