DSA Assessment – 2

Q1.  
A hospital uses a queue system for patient appointments. Patients register with a priority number (higher is urgent). If the queue reaches full capacity N, an urgent patient (priority number > previous maximum) arriving causes the patient with the lowest priority to be removed in favor of the new patient. Otherwise, regular patients join until the queue is full.

Input Format:

* The first line denotes seats N available at the hospital.
* The next line contains total number of registered patients M, followed by M space-separated priority numbers.

Output Format:  
Print the priority numbers in the queue separated by a space.  
If M ≤ 1 and N ≤ 1, print -1.

Constraints:  
1 ≤ N ≤ 100  
1 ≤ M ≤ 1000

Sample Input:

text

4

15 10 30 5 46 25 8

Sample Output:

text

15 10 30 46

Code:

#include <bits/stdc++.h>  
  
using namespace std;  
  
class PriorityQueue {  
public:  
 int\* arr;  
 int MAX\_SIZE;  
 int front;  
 int rear;  
  
 PriorityQueue(int SIZE) {  
 MAX\_SIZE = SIZE;  
 arr = new int[MAX\_SIZE];  
 front = rear = -1;  
 }  
  
 ~PriorityQueue() {  
 delete[] arr;  
 }  
  
 bool isEmpty() {  
 return front == -1;  
 }  
  
 bool isFull() {  
 return (rear + 1) % MAX\_SIZE == front;  
 }  
  
 int getMax() {  
 int maxElement = INT\_MIN;  
 int\* tempArray = new int[MAX\_SIZE];  
 int i = 0;  
 while (!isEmpty()) {  
 int element = peek();  
 pop();  
 tempArray[i++] = element;  
 maxElement = max(maxElement, element);  
 }  
 for (int j = 0; j < i; j++) {  
 push(tempArray[j]);  
 }  
  
 delete[] tempArray;  
 if (maxElement == INT\_MIN) return -1;  
 return maxElement;  
 }  
  
 int getMin() {  
 int minElement = INT\_MAX;  
 int\* tempArray = new int[MAX\_SIZE];  
 int i = 0;  
 while (!isEmpty()) {  
 int element = peek();  
 pop();  
 tempArray[i++] = element;  
 minElement = min(minElement, element);  
 }  
 for (int j = 0; j < i; j++) {  
 push(tempArray[j]);  
 }  
  
 delete[] tempArray;  
 if (minElement == INT\_MAX) return -1;  
 return minElement;  
 }  
  
 void removeLowestPriority() {  
 // we assume the array has size: MAX\_SIZE  
 // this function should only be called when the queue is full  
 int minElement = getMin();  
 int\* tempArray = new int[MAX\_SIZE];  
 int i = 0;  
 bool seen = false;  
 while (! isEmpty()) {  
 int element = peek();  
 pop();  
  
 if (minElement == element && ! seen) {  
 seen = true;  
 continue;  
 }  
 tempArray[i++] = element;  
 }  
 for (int j = 0; j < i; j++) push(tempArray[j]);  
 delete[] tempArray;  
 }  
  
 void push(int x) {  
 if (isFull()) {  
 if (x > getMax()) {  
 removeLowestPriority();  
 rear = (rear + 1) % MAX\_SIZE;  
 arr[rear] = x;  
 return;  
 }  
 // cout << "Queue is full. Can't enqueue more elements." << endl; // queue is full  
 return;  
  
 }  
 if (isEmpty()) {  
 front = 0;  
 }  
 rear = (rear + 1) % MAX\_SIZE;  
 arr[rear] = x;  
 }  
  
 void pop() {  
 if (isEmpty()) {  
 cout << "Queue is empty. Cannot pop elements" << endl;  
 return;  
 }  
 if (front == rear) {  
 front = -1;  
 rear = -1;  
 } else {  
 front = (front + 1) % MAX\_SIZE;  
 }  
 }  
  
 int peek() {  
 if (isEmpty()) {  
 cout << "Queue is empty. Cannot peek" << endl;  
 return -1;  
 }  
 return arr[front];  
 }  
  
 void display() {  
 int\* tempArray = new int[MAX\_SIZE];  
 int i = 0;  
 while (!isEmpty()) {  
 int element = peek();  
 pop();  
 tempArray[i++] = element;  
 cout << element << " ";  
 }  
 cout << endl;  
 for (int j = 0; j < i; j++) {  
 push(tempArray[j]);  
 }  
 delete[] tempArray;  
 }  
};  
  
int main() {  
 int n ;  
 cin >> n;  
 int m; // no of patieents  
 cin >> m;  
 if (n <= 1 && m <= 1) {  
 cout << -1 << endl;  
 return 0;  
 }  
 int\* arr = new int[m];  
 for (int i = 0; i < m; i++) cin >> arr[i];  
 PriorityQueue\* pq = new PriorityQueue(n);  
 for (int i = 0; i < m; i++) {  
 pq->push(arr[i]);  
 }  
 pq->display();  
 delete[] arr;  
 delete pq;  
 return 0;  
}

Test Case 1:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 2:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 3:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 4:

A screenshot of a computer

AI-generated content may be incorrect.

Q2

John is doing a transport business. As he had a lot of customers for registering the car, he wants to give the preference for the

person who comes first, if the car is filled with N number after that who comes with EVEN number is added and the first element is

removed .If ODD number means do nothing.

**Input Format**

The first line of input denotes seats N available in the car.

The next line of input contains total number of registered members M , followed by M space-separated integers.

**Output Format**

Print the members in the car separated by a space.

If M < 1 and N < 1, print -1.

**Constraints**

**1 < M < N**

**Sample Input Sample Output**

5

8 2 4 3 5 6 1 10 12

3 5 6 10 12

Code:

#include <bits/stdc++.h>  
  
using namespace std;  
  
  
class CircularQueue {  
private:  
 int rear;  
 int front;  
 int MAX\_SIZE;  
 int\* arr;  
public:  
 CircularQueue(int SIZE) {  
 arr = new int[SIZE];  
 MAX\_SIZE = SIZE;  
 rear = -1;  
 front = -1;  
 }  
  
 ~CircularQueue() {  
 delete[] arr;  
 }  
  
 void enqueue(int n) {  
 // recollect that enqueue operation takes place at the rear position as per FIFO  
 if (isFull()) {  
 cout << "Queue is full. Can't add more elements" << endl;  
 return;  
 }  
 if (isEmpty()) {  
 front = 0;  
 }  
 rear = (rear + 1) % MAX\_SIZE;  
 arr[rear] = n;  
 }  
  
 int dequeue() {  
 if (isEmpty()) {  
 cout << "Queue is empty" << endl;  
 return INT\_MAX;  
 }  
 int data = arr[front];  
 if (front == rear) {  
 // last element  
 front = -1;  
 rear = -1;  
 }  
 else  
 front = (front + 1) % MAX\_SIZE;  
 return data;  
 }  
  
 bool isEmpty() {  
 return front == -1;  
 }  
  
 bool isFull() {  
 return (rear + 1) % MAX\_SIZE == front;  
 }  
};  
  
int main() {  
 int n, m;  
 cin >> n;  
 cin >> m;  
 if (n < 1 && m < 1) {  
 cout << -1 << endl;  
 return 0;  
 }  
 int\* arr = new int[m];  
 CircularQueue\* q = new CircularQueue(n);  
 for (int i = 0; i < m; i++) {  
 cin >> arr[i];  
 }  
 if (m > n) {  
 for (int i = 0; i < n; i++) {  
 q->enqueue(arr[i]);  
 }  
 for (int i = n; i < m; i++) {  
 if (arr[i] % 2 == 0) {  
 // even number  
 q->dequeue();  
 q->enqueue(arr[i]);  
 }  
 }  
 }  
 else {  
 for (int i = 0; i < m; i++) {  
 q->enqueue(arr[i]);  
 }  
 }  
 while (! q->isEmpty()) {  
 int element = q->dequeue();  
 cout << element << " ";  
 }  
 cout << endl;  
  
 delete q;  
 delete[] arr;  
 return 0;  
}

Test Case 1:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 2:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 3:

A screen shot of a computer

AI-generated content may be incorrect.

Q3

Aarav, Bhavin and Divit are friends.In their hand, they have some N, M and P number of candies with the different cost. Implement the code to arrange their candy's cost in ascending order using the selection and counting sort concept.

**Input Format**

The first element denotes number N. After that N-space separated integers. Then next element denotes number M. After that M space separated integers. Then next element denotes number P. After that P space separated integers.

**Output Format**

Print the ascending order of the candy's cost.

**Constraints**

Integers only.

**Sample Input**

2 500 1000 3 900 100 300 1 100000

**Sample Output**

100 300 500 900 1000 100000

Code:

#include <bits/stdc++.h>  
  
  
using namespace std;  
  
int getMax(int\* arr, int size) {  
 if (size < 1) return INT\_MIN;  
 int maxElement = arr[0];  
 for (int i = 1; i < size; i++) {  
 maxElement = max(maxElement, arr[i]);  
 }  
 return maxElement;  
}  
  
void countingSort(int\* arr, int size) {  
 int n = size;  
 if (n == 0) return;  
 int maxVal = getMax(arr, size);  
  
  
 int\* count = new int[maxVal + 1];  
 for (int i = 0; i < maxVal + 1; i++) count[i] = 0;  
 int\* output = new int[n];  
  
 for (int i = 0; i < n; i++) count[arr[i]]++;  
  
 for (int i = 1; i < maxVal + 1; i++) {  
 // we add prefix sum in this loop  
 count[i] += count[i - 1];  
 }  
  
 for (int i = n - 1; i >= 0; i--) {  
 output[--count[arr[i]]] = arr[i];  
 }  
  
 for (int i = 0; i < n; i++) arr[i] = output[i];  
 delete[] count;  
 delete[] output;  
}  
  
void selectionSort(int\* array, int size) {  
 for (int i = 0; i < size - 1; i++) {  
 int minIndex = i;  
 for (int j = i + 1; j < size; j++) {  
 if (array[minIndex] > array[j]) {  
 minIndex = j;  
 }  
 }  
 int temp = array[i];  
 array[i] = array[minIndex];  
 array[minIndex] = temp;  
 }  
}  
  
int main() {  
 int n, m, p;  
 cin >> n;  
 int\* arr1 = new int[n];  
 for (int i = 0; i < n; i++) {  
 cin >> arr1[i];  
 }  
 cin >> m;  
 int\* arr2 = new int[m];  
 for (int i = 0; i < m; i++) {  
 cin >> arr2[i];  
 }  
 cin >> p;  
 int\* arr3 = new int[p];  
 for (int i = 0; i < p; i++) {  
 cin >> arr3[i];  
 }  
  
 int\* final = new int[n + m + p];  
 int index = 0;  
 for (int i = 0; i < n; i++) {  
 final[index++] = arr1[i];  
 }  
 for (int i = 0; i < m; i++) {  
 final[index++] = arr2[i];  
 }  
 for (int i = 0; i < p; i++) {  
 final[index++] = arr3[i];  
 }  
  
 countingSort(final, n + m + p);  
  
 for (int i = 0; i < n + m + p; i++) {  
 cout << final[i] << " ";  
 }  
 cout << endl;  
  
 selectionSort(final, n + m + p);  
 for (int i = 0; i < n + m + p; i++) {  
 cout << final[i] << " ";  
 }  
  
 delete[] arr1;  
 delete[] arr2;  
 delete[] arr3;  
 delete[] final;  
 return 0;  
}

Test Case 1:

A computer screen shot of numbers and letters

AI-generated content may be incorrect.

Test Case 2:

A computer screen with white text

AI-generated content may be incorrect.

Test Case 3:

A computer screen with white text

AI-generated content may be incorrect.

Test Case 4:

A computer screen shot of a black screen

AI-generated content may be incorrect.

Q. 4. Write a program to simulate a banking queue system

i) All the new customers must be added to the queue with their name, transaction type.

ii) Serve the customers in correct order

iii) Remove them from the queue once served

Code:

#include <iostream>  
#include <string>  
#include <utility> // Required for std::pair  
  
using namespace std;  
  
template <typename T>  
class CircularQueue {  
private:  
 int rear;  
 int front;  
 int MAX\_SIZE;  
 T\* arr;  
public:  
 CircularQueue(int SIZE) {  
 arr = new T[SIZE];  
 MAX\_SIZE = SIZE;  
 rear = -1;  
 front = -1;  
 }  
  
 ~CircularQueue() {  
 delete[] arr;  
 }  
  
 void enqueue(T n) {  
 // recollect that enqueue operation takes place at the rear position as per FIFO  
 if (isFull()) {  
 cout << "Queue is full. Can't add more elements" << endl;  
 return;  
 }  
 if (isEmpty()) {  
 front = 0;  
 }  
 rear = (rear + 1) % MAX\_SIZE;  
 arr[rear] = n;  
 }  
  
 T dequeue() {  
 if (isEmpty()) {  
 cout << "Queue is empty" << endl;  
 return T();  
 }  
 T data = arr[front];  
 if (front == rear) {  
 // last element  
 front = -1;  
 rear = -1;  
 }  
 else  
 front = (front + 1) % MAX\_SIZE;  
 return data;  
 }  
  
 bool isEmpty() {  
 return front == -1;  
 }  
  
 bool isFull() {  
 return (rear + 1) % MAX\_SIZE == front;  
 }  
};  
  
  
// Function to process customers in the queue  
void serveCustomers(CircularQueue<pair<string, int>>& q) {  
 while (!q.isEmpty()) {  
 pair<string, int> customer = q.dequeue();  
 cout << "\n--- Serving Customer: " << customer.first << " ---" << endl;  
  
 switch(customer.second) {  
 case 1:  
 cout << "Check Balance: A report will be sent to the registered mobile number." << endl;  
 break;  
 case 2:  
 cout << "Withdraw Money: Money has been withdrawn successfully." << endl;  
 break;  
 case 3:  
 cout << "Deposit Money: Cash has been deposited successfully." << endl;  
 break;  
 default:  
 cout << "Invalid transaction type found. Please try again later." << endl;  
 break;  
 }  
 }  
}  
  
int main() {  
 cout << "The Banking System has the following transaction types: " << endl;  
 cout << "1. Check Balance" << endl;  
 cout << "2. Withdraw money" << endl;  
 cout << "3. Deposit money" << endl;  
 cout << "------------------------------------------" << endl;  
  
 int n;  
  
 cout << "\nEnter the number of customers: ";  
 cin >> n;  
 CircularQueue<pair<string, int>>\* q = new CircularQueue<pair<string, int>>(n);  
 for (int i = 0; i < n; i++) {  
 pair<string, int> p;  
 cout << "\nEnter details for customer " << i + 1 << endl;  
 cout << "Name: ";  
 cin >> p.first;  
 cout << "Transaction type (1-3): ";  
 cin >> p.second;  
 q->enqueue(p);  
 }  
  
 serveCustomers(\*q);  
 delete q;  
 return 0;  
}

Test Case 1:

A screenshot of a computer program

AI-generated content may be incorrect.

Test Case 2:

A screenshot of a computer program

AI-generated content may be incorrect.

Test Case 3:

A screenshot of a computer

AI-generated content may be incorrect.

Q. 5. Write a program for hospital waiting room queue. Consultation must be served based on FCFS order (First Come First Serve).

Add a patient to the queue with their name and symptom. New patient entry must be sorted order based on priority.

Comparator

1 -> non-emergency

2 -> emergency

Code:

#include <iostream>  
#include <string>  
#include <limits>  
  
using namespace std;  
  
struct Patient {  
 string name;  
 string symptom;  
 int priority; // 2 for emergency, 1 for non-emergency  
  
 // default constructor  
 Patient() : name(""), symptom(""), priority(0) {}  
  
 // parametrized constructor  
 Patient(const string& n, const string& s, int p) : name(n), symptom(s), priority(p) {}  
};  
  
class PriorityLinearQueue {  
private:  
 int rear;  
 int MAX\_SIZE;  
 Patient\* arr;  
  
public:  
 PriorityLinearQueue(int SIZE) {  
 arr = new Patient[SIZE];  
 MAX\_SIZE = SIZE;  
 // In this simple linear queue, the queue is the segment from index 0 to rear.  
 rear = -1;  
 }  
  
 ~PriorityLinearQueue() {  
 delete[] arr;  
 }  
  
 void enqueue(const Patient& patient) {  
 if (isFull()) {  
 cout << "Queue is full. Can't add more patients." << endl;  
 return;  
 }  
 rear++;  
 arr[rear] = patient;  
  
 // Re-sort the entire queue  
 insertionSort();  
 }  
  
 void insertionSort() {  
 for (int i = 1; i <= rear; i++) {  
 Patient temp = arr[i];  
 int j = i - 1;  
  
 while (j >= 0 && arr[j].priority < temp.priority) {  
 arr[j + 1] = arr[j];  
 j--;  
 }  
 arr[j + 1] = temp;  
 }  
 }  
  
 Patient dequeue() {  
 if (isEmpty()) {  
 cout << "Queue is empty." << endl;  
 return Patient();  
 }  
  
 // The highest priority patient is always at the front (index 0) after sorting  
 Patient data = arr[0];  
  
 // Shift all subsequent elements one position to the left  
 for (int i = 0; i < rear; i++) {  
 arr[i] = arr[i + 1];  
 }  
  
 // Decrease the size of the queue  
 rear--;  
 return data;  
 }  
  
 bool isEmpty() {  
 return rear == -1;  
 }  
  
 bool isFull() {  
 return rear == MAX\_SIZE - 1;  
 }  
};  
  
  
void clearInputBuffer() {  
 cin.ignore(numeric\_limits<streamsize>::*max*(), '\n');  
}  
  
int main() {  
 int n;  
 cout << "Enter the number of patients: ";  
 cin >> n;  
  
 PriorityLinearQueue waitingRoom(n);  
  
 for (int i = 0; i < n; i++) {  
 string name, symptom;  
 int priority;  
  
 cout << "\nEnter patient's name: ";  
 clearInputBuffer();  
 getline(cin, name);  
 cout << "Enter symptom: ";  
 getline(cin, symptom);  
 cout << "Enter state of emergency (2 for Emergency, 1 for Non-Emergency): ";  
 cin >> priority;  
  
 Patient patient(name, symptom, priority);  
 waitingRoom.enqueue(patient);  
 }  
  
 cout << "--- Serving Patients from the Queue ---" << endl;  
 cout << "Patients will be served based on priority (Emergency cases first)." << endl;  
  
 int patientNumber = 1;  
 while (!waitingRoom.isEmpty()) {  
 Patient currentPatient = waitingRoom.dequeue();  
 cout << "\n(" << patientNumber++ << ") Now Serving:" << endl;  
 cout << " Name: " << currentPatient.name << endl;  
 cout << " Symptom: " << currentPatient.symptom << endl;  
 cout << " Status: " << (currentPatient.priority == 2 ? "EMERGENCY" : "Non-Emergency") << endl;  
 }  
  
 cout << "\nAll patients in the waiting room have been served." << endl;  
  
 return 0;  
}

Test Case 1:

A screenshot of a computer

AI-generated content may be incorrect.

Test Case 2:

A screenshot of a computer program

AI-generated content may be incorrect.

Test Case 3:

A screenshot of a computer program

AI-generated content may be incorrect.

The entire source code can be found on my GitHub:

https://github.com/qburi/DSA-LAB