LABSHEET - 7

CS F211 - Data Structures and Algorithms
Academic Year: 2025-2026 (2nd Semester)

General Instructions for this Labsheet:

- Use of any built-in function like qsort(), bsearch(), sqrt(), etc. is not allowed.
- All the questions should be done by implementing queues using arrays or linked lists, unless mentioned specifically.
- All the basic operations such as push(x), pop(), top(), size() while using stack and push(x), pop(), front(), size() while using queue should be implemented.
- Any other function should be implemented, if specified.
- Consider **0-based** indexing whenever needed.

Problem 1: Queue Implementation using Dual Stacks

Problem Description: Implement a FIFO (First-In-First-Out) Queue using exactly **two stacks.** All elements added to the queue must be stored within these stacks. You must ensure that the basic queue operations—Enqueue, Dequeue, and Peek—behave exactly as they would in a standard queue, maintaining the correct order of elements. The queue should be implemented such that either Enqueue or Dequeue (depending on the chosen approach) runs in amortized O(1) time, since each element is transferred between the two stacks at most once overall.

Logic & Boundary Conditions:

- **Enqueue (X):** Adds an integer X to the queue.
- **Dequeue:** Removes and returns the oldest element in the queue. If the queue is empty, return **-1**.
- **Peek:** Returns the oldest element in the queue without removing it. If the queue is empty, return **-1**.

Input Format:

- The first line contains an integer T, the number of operations to perform.
- Each of the following T lines contains an operation:
    - 1 X: Enqueue the integer X.
    - 2: Dequeue and print the removed element.
    - 3: Peek and print the oldest element.

Output Format:

- For every operation of type 2 (Dequeue) and 3 (Peek), print the resulting integer on a new line.

Sample Test Case 1:
Input:

```
9
1 10
1 20
1 30
1 40
3
2
2
1 50
2
```

Output:

```
10
10
20
30
```

Sample Test Case 2:
Input:

```
10
1 100
3
1 200
1 300
2
2
2
2
1 400
3
```

Output:

```
100
100
200
300
-1
400
```

Problem 2: First Non-Repeating Character in a Stream

Problem Description:

You are processing a live stream of lowercase characters. As each character arrives, you must determine the earliest character in the stream that has appeared exactly once so far. If at any point every character received has appeared at least twice, print -1. You must implement this efficiently using a **Queue** to maintain the order of arrival and a frequency tracking mechanism.

Input Format:

- A single string S consisting of lowercase English letters.

Output Format:

- A sequence of characters (or -1) separated by spaces.

Sample Test Case 1:

Input: `racecar`

Output: `r r r r r r e`

Explanation: 'r' remains the oldest unique character through the first six letters because, although 'a' and 'c' repeat, 'r' does not repeat until the very last character. When the final 'r' arrives, all previous characters ('r', 'a', 'c') have repeated, leaving 'e' as the only and earliest unique character.

Sample Test Case 2:

Input: `aabbcdefgh`

Output: `a -1 b -1 c c c c c c`

Explanation: The first two 'a's cancel each other out to output -1, then the arrival of 'b's does the same, leaving the queue empty again. Once 'c' arrives, it becomes the oldest unique character and remains so for the rest of the stream, as 'd' through 'h' are all "younger" than 'c'.

Problem 3: Interleave Queue Elements

Problem Description:

Given a queue of N integers, where N is always an even number, you must rearrange the elements by interleaving the first half of the queue with the second half. You must read the input values and insert them directly into a Queue. You are strictly allowed to use exactly one Stack as auxiliary storage. Implement using functions like enqueue(), dequeue(). No other data structures (no second queue, no arrays) are permitted.

Interleaving Pattern:

If the queue is [1, 2, 3, 4, 5, 6], the first half is [1, 2, 3] and the second half is [4, 5, 6]. The final interleaved queue must be [1, 4, 2, 5, 3, 6].

Input Format

- **Line 1:** An even integer N, representing the total number of elements.
- **Line 2:** N space-separated integers representing the queue elements.

Output Format

- The modified queue elements separated by spaces.

Sample Test Case 1

Input:

```
4
11 12 13 14
```

Output:

```
11 13 12 14
```

Explanation: The first half is [11, 12], second half is [13, 14]. Interleaving them gives 11 (from 1st half), then 13 (from 2nd half), then 12, then 14.

Sample Test Case 2

Input:

```
8
1 2 3 4 5 6 7 8
```

Output:

```
1 5 2 6 3 7 4 8
```

Explanation: First half: [1, 2, 3, 4], Second half: [5, 6, 7, 8]. Interleaving yields [1 5 2 6 3 7 4 8].

Problem 4: Hit the Target

Problem Description:

Given an M x N grid containing three numbers: 0, 1, 2. You want to travel from the top-left corner (0,0) to the target cell containing 2. You have to follow the following rules while travelling:

- Cells marked with 0 are empty cells and can be visited.
- Cells marked with 1 are blocked cells and can **NOT** be visited.
- Cells marked with 2 are the **Target Cells**.
- You can move Up, Down, Left, or Right.
- You cannot move beyond boundaries.

Find the minimum number of points to move upon to reach the target cell containing 2 from (0,0) (including start and target). It is guaranteed that such a path will always exist.

Note: Starting Point (0,0) will always be marked with 0.
Hint: Use queues to search for the element (breadth-first on matrix)

Input Format:
Line 1: Two integer M, N (Number of rows and columns of the grid; **M, N ≤ 10**)
Next M Lines: N numbers (Representing a row in the grid)

Output Format:
Minimum points to cover to reach T.

Sample Test Case 1:

Input:
```
3 4
0 0 0 0
0 1 1 0
0 1 0 2
```

Output:
6

Explanation:
```
Followed path is shown:
P P P P
0 1 1 P
0 1 0 P
```

Sample Test Case 2:

Input:
```
5 5
0 0 0 0 0
0 1 0 0 0
1 0 0 1 1
0 0 1 2 0
0 0 0 0 0
```

Output:
11

Explanation:
```
Followed path is shown:
P P P 0 0
0 1 P 0 0
1 P P 1 1
0 P 1 P 0
0 P P P 0
```

Problem 5: Maximum Sliding Window

Problem Description:

Given an array of integers **arr** of size **N** and an integer **K**. Find the maximum element in each window of size K from the left to right. More formally,

`ans[i] = max(arr[i], arr[i+1], … , arr[i+K-1]) where 0 ≤ i ≤ N-K`

Return the array **ans** consisting of the maximum elements in each window.

Note: Consider **0-based indexing** for the question.
Note: The question should be done by implementing queues only in O(N) time. Solution with time complexity O(K*N) will **NOT** be accepted.
Hint: Standard queue will not work in this scenario. Try to use a double-ended queue.

Input Format:
Line 1: Two integers N, K (Size of the array and an integer K, N ≥ 3, K ≥ 2)
Line 2: N integers (Array arr elements)

Output Format:
Array ans containing maximum elements in each window

Sample Test Case 1:

Input:
```
6 2
4 2 5 6 1 3
```

Output:
```
4 5 6 6 3
```

Sample Test Case 2:

Input:
```
8 3
1 3 -1 -3 5 3 6 7
```

Output:
```
3 3 5 5 6 7
```

Problem 6: Time-To-Ticket

Problem Description:

There are N people in a line queuing to buy tickets, where the 0th person is at the front of the line and the (N - 1)th person is at the back.

You are given a 0-indexed integer array **tickets** of length N where the number of tickets that the **ith** person would like to buy is **tickets[i]**. Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to **go back to the end** of the line (if they need more tickets).

If a person does not need more tickets, they leave the line.
Return the time taken for the person at position K (initially) to finish buying tickets.

Note: Consider **0-based indexing** for the question.
Note: The question should be done by implementing queues only.

Input Format:
Line 1: Two integers N, K (Size of the array and an integer K, $N \geq 3$, $0 \leq K < N$)
Line 2: N integers (Array tickets elements)

Output Format:
Time taken for the person at position K to finish buying tickets

Sample Test Case 1:

Input:
```
3 2
2 3 2
```

Output:
```
6
```
Explanation:
```
      - In the first pass, everyone buys 1 ticket. Queue: [1,2,1].
Time = 3

      - The Person 0 buys a ticket and leaves the queue.
Queue: [2,1]. Time = 4

      - The Person 1 buys a ticket and goes to the end of the queue.
Queue: [1,1]. Time = 5

      - The Person 2 buys a ticket and leaves the queue. Time = 6

Person 2 finishes and leaves at Time = 6.
```

Sample Test Case 2:

Input:

```
4 0
5 1 1 1
```

Output:

```
8
```

Explanation:

```
- Pass 1: Everyone buys 1 ticket. Queue: [4]. Time = 4
- Persons 1, 2, and 3 leave. Only Person 0 remains.
- Pass 2: Person 0 buys 1 ticket. Queue: [3]. Time = 5
- Pass 3: Person 0 buys 1 ticket. Queue: [2]. Time = 6
- Pass 4: Person 0 buys 1 ticket. Queue: [1]. Time = 7
- Pass 5: Person 0 buys 1 ticket. Queue: [0]. Time = 8
- Person 0 finishes and leaves at Time = 8.
```

Problem 7: Circular Tour (Gas Station)

Problem Description:

Suppose there is a circle with N gas stations. You are given two sets of data:

1. The amount of **gas** that every gas station has.
2. The **distance** from that gas station to the next gas station.

You have a car with an unlimited fuel tank and it costs 1 unit of gas to travel 1 unit of distance. You must find the **starting point** from where the car can complete a full circle without running out of gas. If multiple such points exist, return the earliest one. If no such point exists, return -1.

Logic & Boundary Conditions:

- You must implement this using a **Queue** to track the current tour.
- The tour is complete if the car returns to the starting station with greater than equal to 0 gas.

Input Format:

- **Line 1:** An integer N, the number of gas stations.
- **Line 2:** N space-separated integers representing the amount of gas at each station.
- **Line 3:** N space-separated integers representing the distance to the next station.

Output Format:

- The index of the first starting point (0-indexed). If no solution exists, print -1.

Sample Test Case 1:

Input:

```
4
4 6 7 4
6 5 3 5
```

Output:

```
1
```

Explanation:

```
If we start at index 1 (gas=6, dist=5), we have 1 unit left. At index
2 (gas=7, dist=3), we have 1+7-3=5 units left. At index 3 (gas=4,
dist=5), we have 5+4-5=4 units left. Finally, at index 0 (gas=4,
dist=6), we have 4+4-6=2 units, returning to index 1.
```

Problem 8: Binary Number Generator

Problem Description:

Given an integer N, generate and print all binary numbers from **1 to N** in increasing order. To ensure optimal performance, you must use a **Queue** data structure to generate these numbers as strings sequentially rather than using built-in mathematical conversions for every individual number.

Input Format:

- A single integer N.

Output Format:

- A sequence of binary strings separated by spaces.

Sample Test Case 1:

Input:
```
5
```

Output:
```
1 10 11 100 101
```

Sample Test Case 2:

Input:
```
7
```

Output:
```
1 10 11 100 101 110 111
```

Problem 9: Reverse First K Elements

Problem Description:

Given a queue of N integers and an integer K, you must reverse the order of the **first K elements** of the queue, while leaving the remaining elements in their original relative order. You must read the input values into a Queue first.

Logic & Boundary Conditions:

- You are strictly allowed to use exactly **one Stack** as auxiliary storage.
- No other data structures (no second queue, no arrays) are permitted.
- After reversing the first K elements using the stack, the remaining N-K elements must be moved to the back of the queue to maintain their original order.

Input Format:

- **Line 1:** Two integers N and K.
- **Line 2:** N space-separated integers representing the queue elements.

Output Format:

- The modified queue elements separated by spaces.

Sample Test Case 1:

Input:

```
5 3
10 20 30 40 50
```

Output:

```
30 20 10 40 50
```

Sample Test Case 2:

Input:

```
7 4
58 61 103 2 8 12 76
```

Output:

```
2 103 61 58 8 12 76
```

Problem 10: Elevator Boarding Simulation

Problem Description

You are managing a specialized service elevator that can only transport **one person at a time**. People arrive at the elevator lobby to either enter the elevator to go up or exit the elevator to leave. Because the space is narrow, if multiple people are waiting, the system follows specific priority rules based on the elevator's "momentum" to determine who moves next.

Priority Rules

- **Momentum Rule:** If the elevator was used in the previous second, the person moving in the **same direction** as the last user gets preference.
- **Idle Rule:** If the elevator was not used in the previous second (it was idle), the **exiting (1)** direction gets preference over the entering (0) direction.
- **Solitary Rule:** If only one direction has people waiting at the current time, that person goes immediately, regardless of the previous state or the idle rule.

This is a simulation problem. To solve it efficiently, maintain two separate queues—one for **entering (0)** and one for **exiting (1)**—containing (original_index, arrival_time) pairs.

Input Format

- An array time[i] representing the time (in seconds) when the i-th person arrives at the elevator (sorted in non-decreasing order).
- An array direction[i] representing the intent of the i-th person:
    - **0**: Entering the elevator.
    - **1**: Exiting the elevator.

Output Format

- An array of integers where the i-th element is the **exact time** the i-th person (from the original input order) successfully uses the elevator.

Sample Test Case 1

Input:
```
0 0 1 5
0 1 1 0
```
Output:
```
2 0 1 5
```

Explanation for test case 1:

- **At t=0:** Person 0 (Enter) and Person 1 (Exit) arrive. Elevator was idle, so **Exit** gets priority. Person 1 goes at **t=0**.
- **At t=1:** Person 2 (Exit) arrives. Since the last move was an Exit, the **Momentum Rule** gives Person 2 priority. Person 2 goes at **t=1**.
- **At t=2:** Only Person 0 (Enter) is left. They go at **t=2**.
- **At t=5:** Person 3 (Enter) arrives and goes immediately at **t=5**

Sample Test Case 2

Input:
```
0 1 1 3 3
0 1 0 0 1
```
Output:
```
0 2 1 4 3
```