1. Implement a Queue using Array

Implement a queue using arrays supporting:

- `enqueue(x)`
- `dequeue()`
- `front()`
- `isEmpty()` and `isFull()`

Handle **overflow** and **underflow** cases.

2. Implement a Queue using Linked List

Implement a queue using a linked list.

Each node contains `data` and `next`.

Maintain **front** and **rear** pointers for efficient enqueue/dequeue.

3. Implement a Priority Queue using Array

Implement a queue where each element has a priority.
The highest-priority element is dequeued first.

Hint:

Store elements with `(data, priority)` pairs.
On dequeue, find and remove the element with the highest priority.

4. Reverse First K Elements of a Queue

Reverse the first `k` elements of a queue while leaving the rest in the same order.

Example:
Input: `Queue = [1, 2, 3, 4, 5]`, `k = 3`
Output: `[3, 2, 1, 4, 5]`

5. Check if a Queue is a Palindrome

Given a queue of integers, check if it reads the same forward and backward. You can use a stack or a temporary queue.

Example:
Input: `[1, 2, 3, 2, 1]`
Output: `Palindrome`