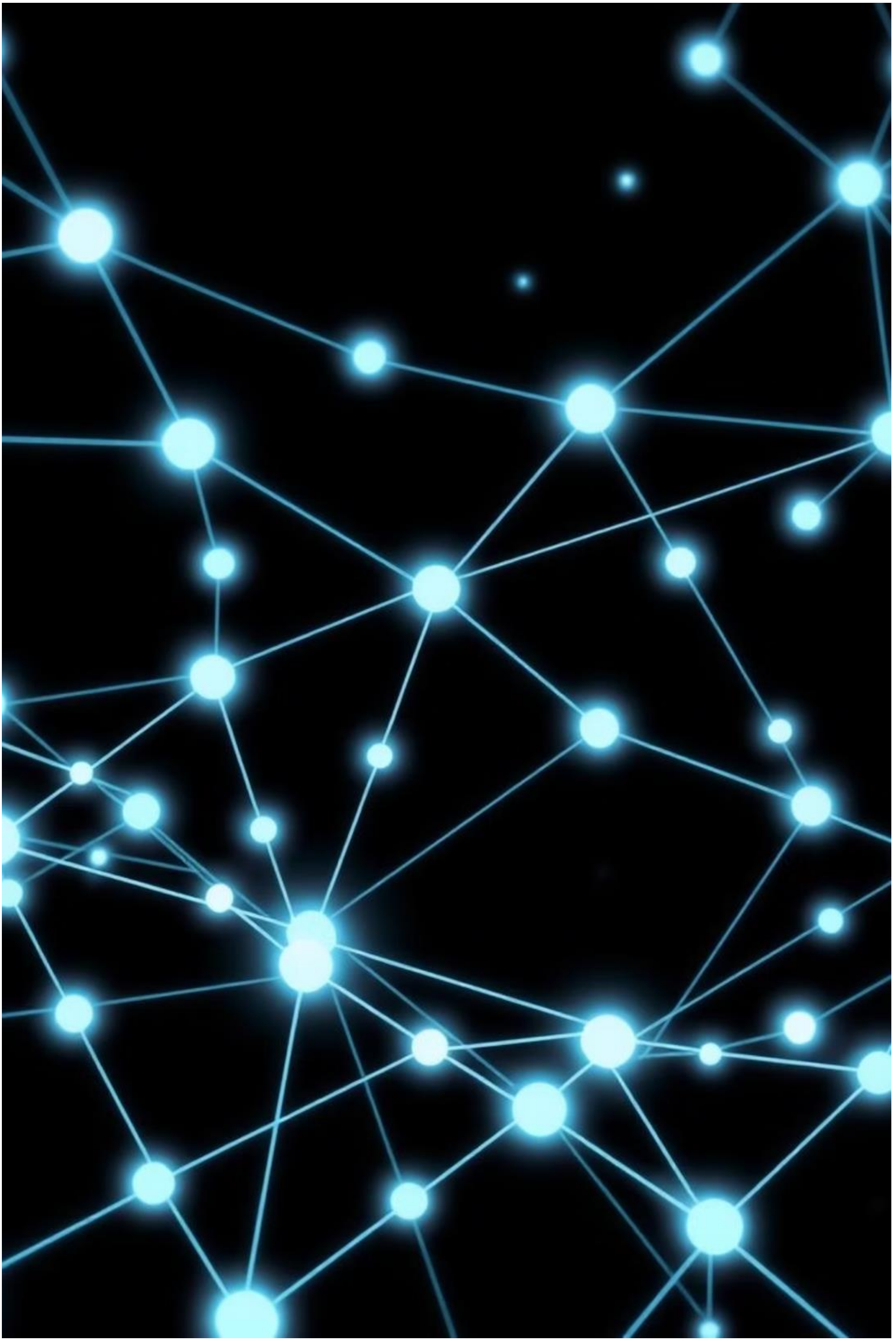


Data Structures

Lecture 5:Queues

Instructor:

Md Samsuddoha

Assistant Professor

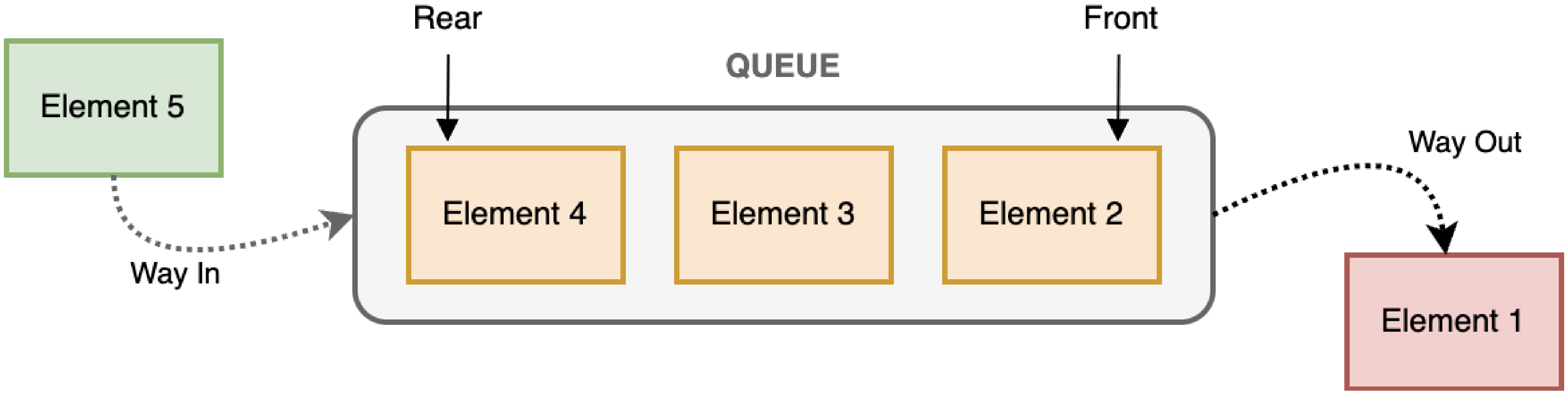
Dept of CSE, BU

Contents

- Concept of Queues
- Types of Queues
- Implementation

Queues: First In, First Out (FIFO) Lines

- **The Line Analogy**
  - Think of a queue like a traditional line at a store: the first person to enter the line is the first one to be served. This "First In, First Out" (FIFO) principle governs queue operations.
- **Fundamental Operations**
  - **Enqueue:** Adds an element to the rear of the queue.
  - **Dequeue:** Removes the element from the front of the queue.
- **Diverse Applications**
  - Queues are vital in various computing scenarios, including **breadth-first search (BFS)** algorithms, **managing tasks in operating systems**, handling **shared resources** (like printer queues), and buffering data in streams.


Rear
QUEUE
Front
Element 5
Element 4
Element 3
Element 2
Way Out
Way In
Element 1

Queue Operations

Operation	Description
Enqueue(x)	Insert an element x at the rear of the queue.
Dequeue()	Remove an element from the front of the queue.
Front() / Peek()	Get the element at the front without removing it.
isEmpty()	Check whether the queue is empty.
isFull()	Check whether the queue is full (for static arrays).

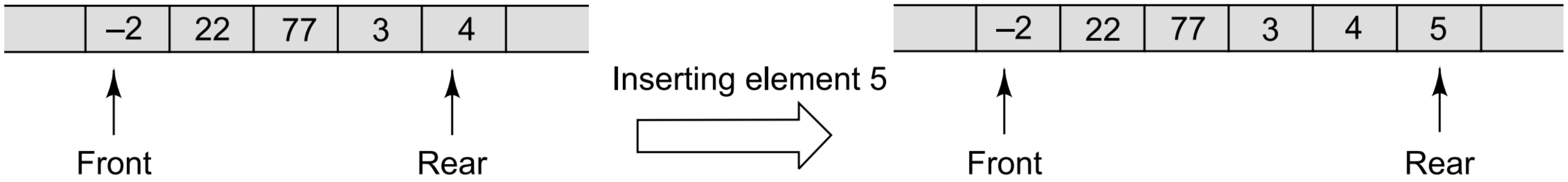
Implementations

1. **Array Implementation**
   - Fixed size
   - Easy to implement
   - Problem: Wastage of space after multiple deletions
2. **Linked List Implementation**
   - Dynamic size
   - No overflow unless memory is full
   - Slightly more complex

Types of Queue

- Simple Queue
  - Follows FIFO
  - Ticket Counter
- Circular Queue
  - Last position connects back to the first to reuse space
  - Memory Buffers
- Priority Queue
  - Elements are served based on priority, not order
  - CPU Scheduling
- Double-ended Queue
  - Insertion and deletion allowed at both ends
  - Task management systems

Simple Queue: Operations



	–2	22	77	3	4	

Front

Rear

Inserting element 5

	–2	22	77	3	4	5	

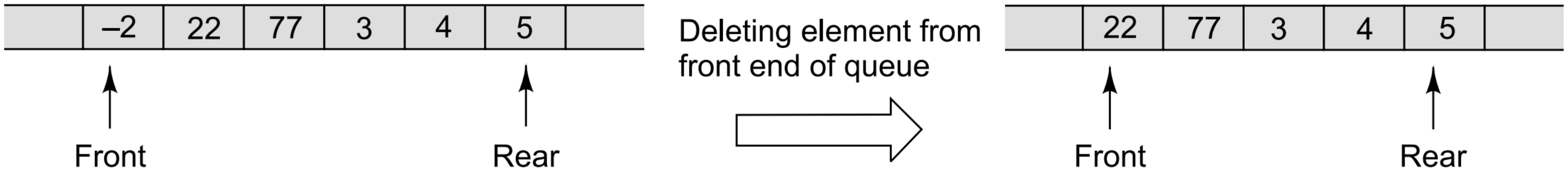
Front

Rear

Queue before insert

Queue after insert

Insert operation



	–2	22	77	3	4	5	

Front

Rear

Deleting element from front end of queue

	22	77	3	4	5	

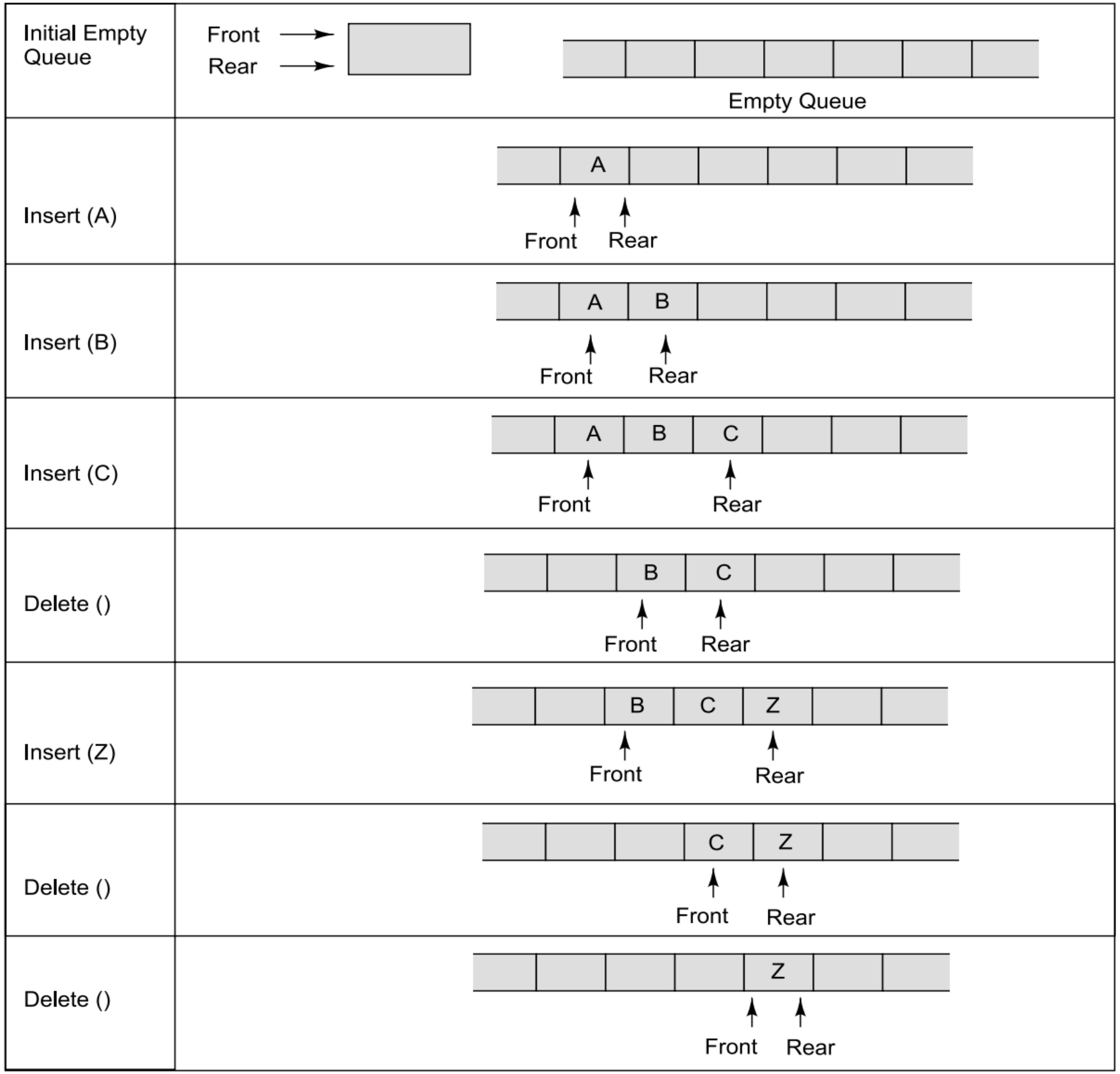
Front

Rear

Queue before delete

Queue after delete

Delete operation



Queue operations

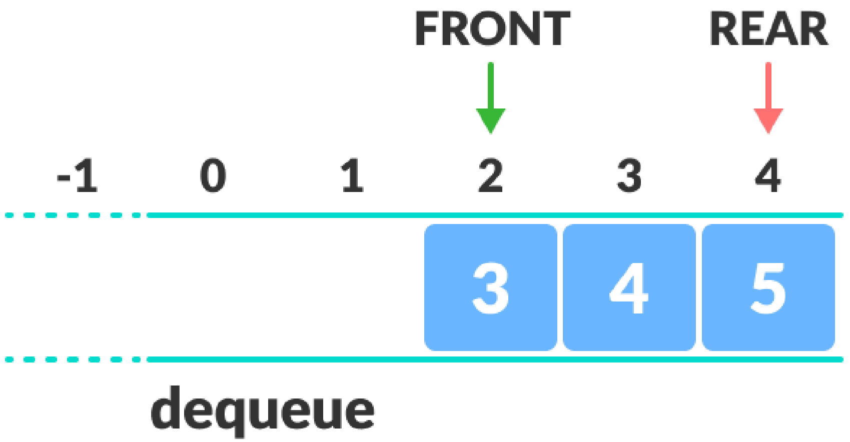
Implementation of Simple Queue : Array

- Initialization
  - Queue, front, rear
- Enqueue()
- Dequeue()
- isFull()
- isEmpty()
- Peek()/ front()
- Display()

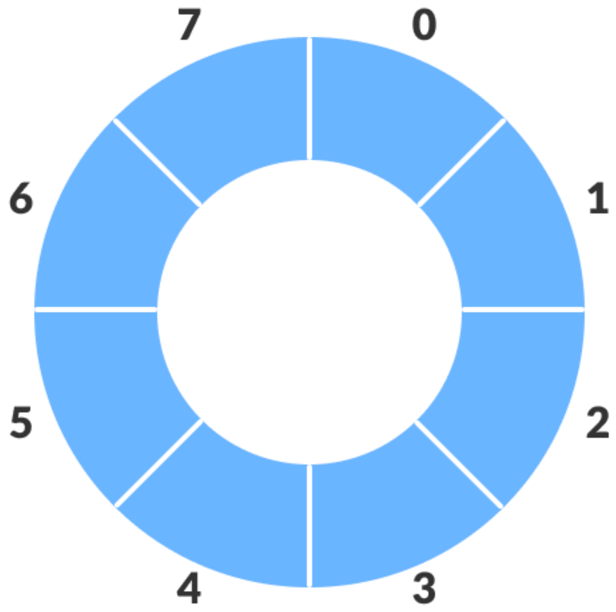
Circular Queue

- A Circular Queue is a linear data structure that follows the **FIFO (First In, First Out)** principle, but the ***last position is connected back to the first*** to form a circle
- It is an improved version of the **simple queue** that solves the problem of **unused space** in the array.

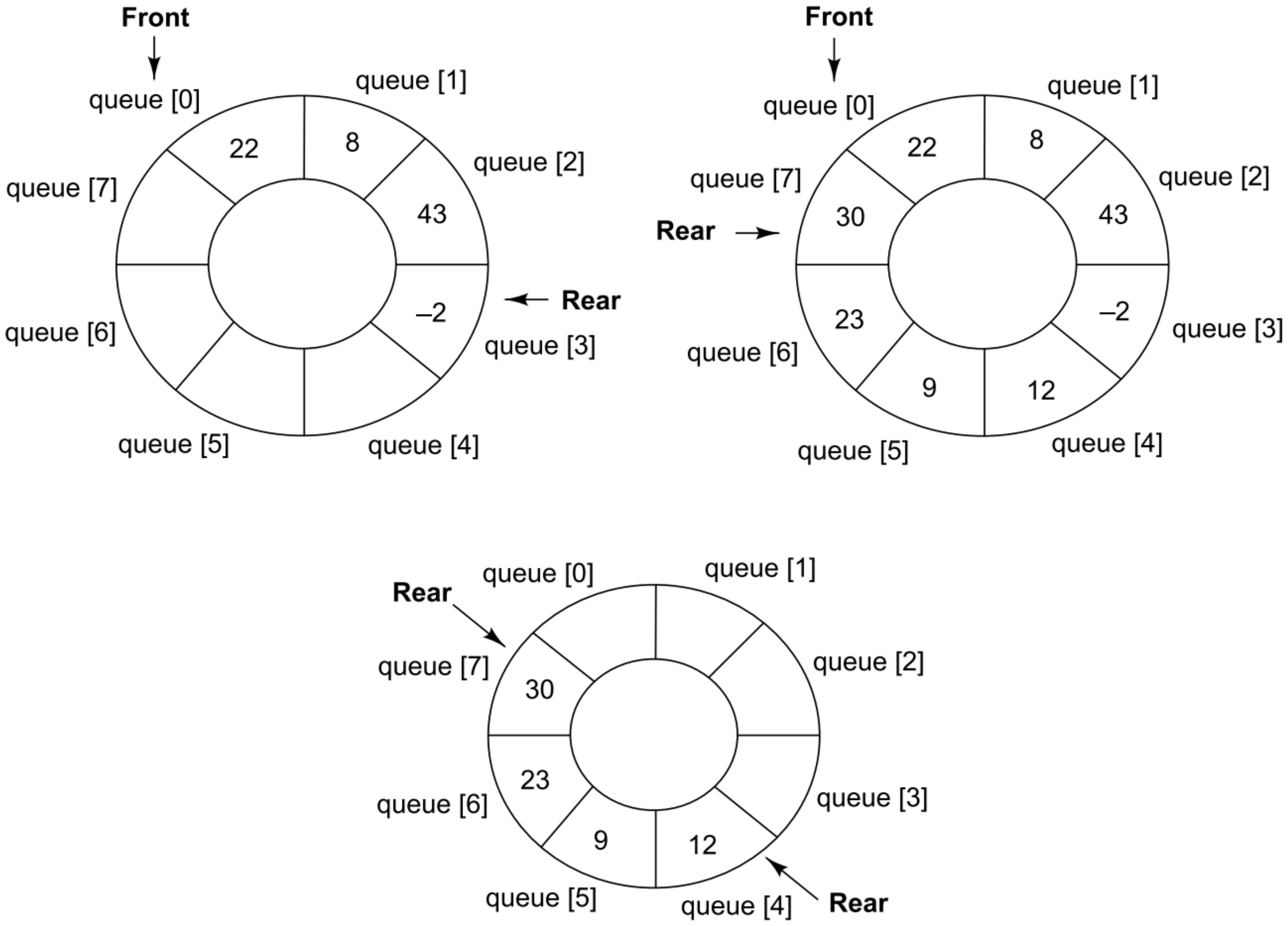
Circular Queue



Regular Queue



Circular Queue



Inserting and deleting elements in a circular queue

Circular Queue Conditions

Condition	**Formula / Description**
Empty	front == -1
Full	(rear + 1) % size == front
Front Move	front = (front + 1) % size
Rear Move	rear = (rear + 1) % size

Priority Queue

- A **Priority Queue** is an abstract data type similar to a regular queue, but **each element is associated with a priority**.
- In a PQ, **elements with higher priority are dequeued before elements with lower priority**, regardless of their insertion order.
- If two elements have the same priority, they may be served based on **FIFO order** (depending on implementation).

PQ

Index	0	1	2	3	4	5
Element	10	30	15	60	40	
Priority	2	10	5	10	12	

- Initialization: *priorityQ[], priority[], totalItem*
- isFull()
- isEmpty()
- Enqueue(element, priority)
- Dequeue()
- Display()

References

- **Chapter 7:**
  - **Data Structures using C** by E. Balagurusamy

Thank You