Queue in Data Structures

1. Introduction

- A **Queue** is a **linear data structure** that follows the order:
  FIFO (First In, First Out).
- The element that is **inserted first is removed first.**
- Example:
  - People standing in a line → the one who comes first gets served first.
  - Printer queue → the first document added is printed first.

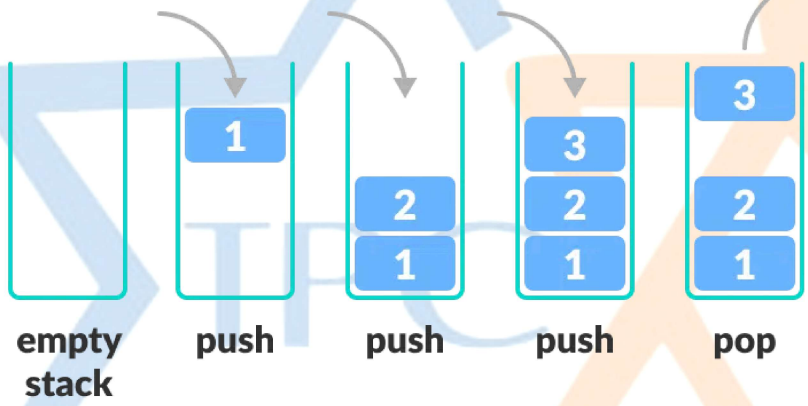
2. Characteristics of a Queue

1. Linear structure (elements arranged sequentially).
2. Insertion happens **only at the rear.**
3. Deletion happens **only at the front.**
4. Maintains strict FIFO order.
5. Can be implemented using **arrays** or **linked lists.**

3. Basic Queue Operations

1. **Enqueue (Insertion)** → Add an element at the **rear** of the queue.
2. **Dequeue (Deletion)** → Remove an element from the **front.**

3. **Peek/Front** → Get the first element without removing it.
4. **isEmpty()** → Checks if the queue is empty.
5. **isFull()** → Checks if the queue is full (in array-based implementation).



4. Real-Life Examples

- Ticket counter line.
- Call center waiting calls.
- Printer job scheduling.
- CPU scheduling (Round Robin).

5. Memory Representation of Queue

- Implemented using **array** or **linked list.**
- Example (Array implementation, SIZE = 5):

```
Index : 0    1    2    3    4
Value : 10   20   30   -    -
Front → 0
Rear  → 2
```

6. Time Complexity

Operation	Complexity
Enqueue	O(1)
Dequeue	O(1)
Peek	O(1)
Traversal	O(n)

Types of Queues

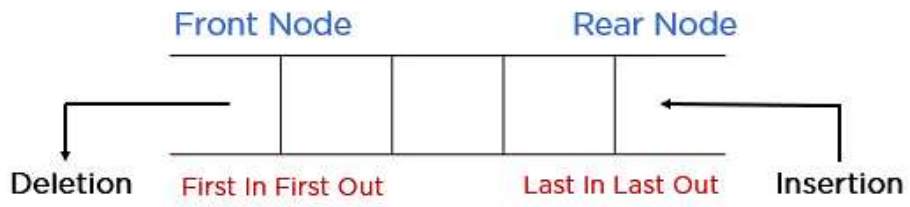
1. Simple Queue / Linear Queue

- Insertion → rear
- Deletion → front
- Maintains **FIFO.**

- Implemented using arrays or linked lists.
- **Problem:** Once rear reaches the end of the array, no new element can be inserted even if there is free space at the front.

Example:

```
Enqueue(10), Enqueue(20), Enqueue(30)
Front → 10  20  30 ← Rear
Dequeue → removes 10
```



Queue Representation

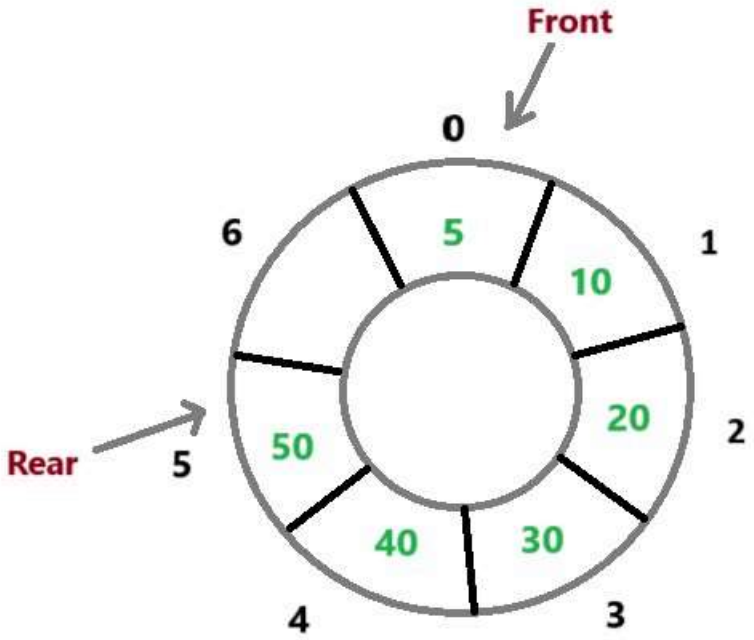
2. Circular Queue

- Solves the problem of wasted memory in **linear queue.**
- In circular queue, the **last position connects back to the first position.**
- Rear moves in a **circular fashion** using modulo (%).

Example (SIZE=5):

```
Initial: Front=-1, Rear=-1
```

```
Enqueue 10 → Front=0, Rear=0
Enqueue 20 → Rear=1
Enqueue 30 → Rear=2
Dequeue → Front=1
Enqueue 40 → Rear=3
Enqueue 50 → Rear=4
Enqueue 60 → Rear=0 (wraps around)
```



3. Double-Ended Queue (Deque)

- **Insertion and deletion can happen at both ends.**
- More flexible than simple/circular queue.
- Two types:

1. **Input-restricted deque:** Insertion allowed only at one end, deletion from both ends.

2. **Output-restricted deque:** Deletion allowed only at one end, insertion from both ends.

Example:

```
Front → 10  20  30  40 ← Rear
Insert at front: 5 → 5  10  20  30  40
Delete from rear → 5  10  20  30
```

4. Priority Queue

- Each element is associated with a **priority**.
- Deletion happens based on **priority** rather than position.
- Higher priority elements are dequeued first.
- If priorities are same → follow **FIFO**.

Example:

```
Insert(10, priority 2)
Insert(20, priority 1)
Insert(30, priority 3)

Queue (sorted by priority):
30 (high) → 10 → 20 (low)
```

Comparison of Queue Types

Type	Insertion	Deletion	Usage Example
Simple Queue	Rear	Front	Ticket counter
Circular Queue	Rear (mod N)	Front (mod N)	CPU scheduling
Deque	Both ends	Both ends	Browser history
Priority Queue	Based on value	Based on value	Hospital emergency