Queue – Circular Queue

1. Introduction

A **Circular Queue** is an advanced version of the simple queue that **overcomes the limitation of unused space** in array-based queue implementations.

In a circular queue, the **last position is connected back to the first**, forming a circular structure.

This allows **efficient memory utilization**.

---

2. What is a Circular Queue?

In a circular queue:

- The queue positions are treated as **circular**
- When the rear reaches the end of the array, it wraps around to the beginning
- Both **front and rear pointers move circularly**

This avoids space wastage caused in a linear queue.

---

3. Problem with Linear Queue (Why Circular Queue?)

In a linear queue:

- After multiple dequeue operations, empty spaces appear at the beginning
- Even if space is available, new elements cannot be added once rear reaches the end

Circular queue **reuses this empty space**.

---

4. Structure of Circular Queue

Conceptually:

```
Rear → [ ] [ ] [ ] [ ] ← Front
   ↑________________↓
```

- Last index connects back to first index
- Front and Rear move in a circular manner

---

5. Key Idea Behind Circular Queue

The main idea:

- Use modulo (%) operation to wrap around indices
- Treat the array as circular
- Efficiently utilize all available space

---

6. Logic for Enqueue in Circular Queue (Plain English)

1. Check if the queue is full
2. If full, report overflow
3. If queue is empty:
    - Set both front and rear to 0
4. Else:
    - Move rear to (rear + 1) % size
5. Insert element at the new rear position

---

7. Logic for Dequeue in Circular Queue (Plain English)

1. Check if the queue is empty
2. If empty, report underflow
3. Remove the element at front
4. If front == rear:

- Reset both front and rear

5. Else:
    - Move front to (front + 1) % size

---

8. Full and Empty Conditions

Queue is Full When:

```
(front == (rear + 1) % size)
```

Queue is Empty When:

```
front == -1
```

---

9. Visualization Example

Queue size = 5

Enqueue operations:

```
Enqueue 10 → Enqueue 20 → Enqueue 30 → Enqueue 40
```

Dequeue twice:

```
Remove 10, Remove 20
```

Rear wraps around:

```
Enqueue 50 → Enqueue 60
```

Queue uses all positions efficiently.

---

10. Time and Space Complexity

Operation	Time Complexity
Enqueue	O(1)
Dequeue	O(1)
Peek	O(1)

- **Space Complexity:** O(n)

---

11. Advantages of Circular Queue

- Efficient memory utilization
- No space wastage
- Faster operations
- Suitable for fixed-size buffers

---

12. Limitations of Circular Queue

- Slightly complex logic
- Difficult to implement compared to linear queue
- Fixed size in array implementation

---

13. Real-World Applications

- CPU scheduling
- Memory buffering
- Traffic signal systems
- Streaming data handling
- Producer-consumer problems

---

14. Comparison with Linear Queue

Feature	Linear Queue	Circular Queue
Space Utilization	Poor	Efficient
Overflow Issue	Common	Rare
Implementation	Simple	Moderate
Performance	Moderate	Better

---

15. Summary

- Circular queue connects end to start
- Solves memory wastage problem
- Uses modulo arithmetic
- All operations are O(1)
- Widely used in real systems
