Queue – Queue Using Array

1. Introduction

A **Queue Using Array** is an implementation of the queue data structure where elements are stored in a **linear array**.

It follows the **FIFO (First In, First Out)** principle, where insertion happens at the **rear** and deletion happens at the **front**.

This is one of the **simplest queue implementations** and is commonly taught in DSA fundamentals.

---

2. What is Queue Using Array?

In this implementation:

- A fixed-size array is used to store queue elements
- Two pointers are maintained:
  - **Front** → points to the first element
  - **Rear** → points to the last element
- Initially:

```
Front = -1
Rear = -1
```

---

3. Queue Structure (Conceptual)

```
Array: [ 10 | 20 | 30 |   |   ]
Index:   0    1    2
Front → 0
Rear  → 2
```

Elements are accessed **only from the front or rear**, not randomly.

4. Components of Queue Using Array

- **Array** → Stores queue elements
- **Front pointer** → Tracks the first element
- **Rear pointer** → Tracks the last element
- **Size** → Maximum capacity of the queue

5. Enqueue Operation (Array Queue)

Logic (Plain English)

1. Check if rear == size - 1
2. If true → Queue Overflow
3. If queue is empty:
    - Set front = 0
4. Increase rear by 1
5. Insert element at array[rear]

Example

```
Enqueue 40
Rear moves from 2 → 3
```

6. Dequeue Operation (Array Queue)

Logic (Plain English)

1. Check if front == -1 or front > rear
2. If true → Queue Underflow
3. Remove the element at array[front]

4. Increase front by 1

Example

```
Dequeue → removes 10
Front moves from 0 → 1
```

---

7. Peek / Front Operation

Logic (Plain English)

1. Check if queue is empty
2. If not empty, return array[front]
3. Queue remains unchanged

---

8. isEmpty Operation

The queue is empty when:

```
front == -1 OR front > rear
```

Used before dequeue and peek operations.

---

9. isFull Operation

The queue is full when:

```
rear == size - 1
```

Used before enqueue operation.

---

10. Queue Overflow in Array Queue

Queue Overflow occurs when:

- Attempting to enqueue an element when rear == size - 1

Reason:

- Fixed size of the array

---

11. Queue Underflow in Array Queue

Queue Underflow occurs when:

- Attempting to dequeue an element when the queue is empty

---

12. Time and Space Complexity

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

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

  (where n is the size of the queue)

---

13. Advantages of Queue Using Array

- Simple and easy to implement
- Fast operations
- Suitable for fixed-size problems
- Low overhead

---

14. Limitations of Queue Using Array

- Fixed size
- Space wastage after dequeue operations
- Overflow even when free space exists

- Less flexible than circular queue

---

15. Comparison with Circular Queue

Feature	Array Queue	Circular Queue
Memory Usage	Inefficient	Efficient
Overflow Issue	Common	Rare
Implementation	Simple	Moderate
Performance	Moderate	Better

---

16. Real-World Applications

- Basic task scheduling
- Printer queue simulation
- Educational demonstrations
- Small buffering systems

---

17. Summary

- Queue using array uses fixed-size storage
- Uses front and rear pointers
- Enqueue at rear, dequeue at front
- Simple but has space limitations
- Time complexity is O(1)
