# DS UNIT 3

[ 4-marks ]

1. **Define queue. Explain FIFO principle.**

   A queue is a linear data structure used to store data elements one after another. It allows data to be inserted and removed in an organized manner.

   Queue follows the FIFO (First In First Out) principle. This means the element that is inserted first is removed first.

   The insertion of an element is done at the rear end of the queue. This ensures that new elements are added at the end.

   The deletion of an element is done from the front end. This helps maintain the correct order of elements.

   A common example of a queue is people waiting in a line at a ticket counter. The person who comes first is served first.

2. **Write enqueue and dequeue operations for a linear queue.**

Enqueue is the operation used to insert an element into the queue. It always inserts the element at the rear end of the queue.

Before performing enqueue, the system checks whether the queue is full. If the queue is full, insertion is not allowed.

Dequeue is the operation used to remove an element from the queue. It always removes the element from the front end.

Before performing dequeue, the system checks whether the queue is empty. If the queue is empty, deletion cannot be performed.

These operations ensure that the queue follows the FIFO rule properly.

3. **What is queue overflow and underflow?**

Queue overflow occurs when we try to insert an element into a queue that is already full. In this situation, the rear pointer has reached the maximum limit.

Overflow causes an error because no more space is available in the queue. This problem usually occurs in array-based queues.

Queue underflow occurs when we try to delete an element from an empty queue. In this case, there is no element to remove.

Underflow also causes an error condition in the program. It indicates that deletion is attempted without any data present.

Proper condition checks help in avoiding overflow and underflow problems.

4. **List different types of queues.**

**A Linear Queue** is the simplest form of queue where insertion is done at the rear and deletion at the front. It may cause wastage of space after deletions.

**A Circular Queue** connects the last position of the queue to the first position. It efficiently uses the available memory space.

**A Priority Queue** stores elements along with their priority. Elements with higher priority are removed before others.

**A Double Ended Queue (Deque)** allows insertion and deletion from both ends. It provides more flexibility compared to a normal queue.

Different types of queues are used based on the requirement of the application.

5. **What is a circular queue? Why is it needed?**

A circular queue is a type of queue where the last position is connected back to the first position. This forms a circular structure instead of a straight line.

In a circular queue, when the rear reaches the end, it moves to the beginning if space is available. This allows better use of memory.

Circular queue is needed to avoid wastage of space that happens in a linear queue. In linear queues, empty spaces created by deletion cannot be reused.

It is commonly used in CPU scheduling and buffering systems. This improves performance and efficiency.

6. **Define priority queue. Give one example.**

A priority queue is a type of queue where each element is associated with a priority value. Elements are removed based on priority, not arrival time.

The element with the highest priority is deleted first from the queue. If two elements have the same priority, FIFO order is followed.

Priority queues are useful in situations where urgent tasks must be handled first. This helps in proper task management.

Example: In a hospital, patients with serious conditions are treated before normal patients. This is a real-life example of a priority queue.

**7. What is dequeue? Mention its types.**

A deque (Double Ended Queue) is a special type of queue where insertion and deletion can be done at both ends. It provides more flexibility than a normal queue.

In a deque, elements can be added or removed from the front as well as the rear. This makes it useful for many applications.

Input Restricted Deque allows insertion at only one end but deletion at both ends. This limits how elements are added.

Output Restricted Deque allows deletion at only one end but insertion at both ends. This limits how elements are removed.

**8. Write operations of queue with example.**

Insertion (Enqueue) is the operation used to add an element to the queue. The element is always inserted at the rear end.

Deletion (Dequeue) is the operation used to remove an element from the queue. The element is always removed from the front end.

Peek operation is used to view the front element without removing it. It helps in checking the next element to be processed.

Example: If we insert 10, 20, and 30 into a queue, 10 will be removed first. This shows the FIFO working of the queue.

**[ 5-marks ]**

**1. Explain representation of queue using array.**

A queue can be represented using an array where elements are stored in a linear order. Two variables called front and rear are used to manage the queue.

Front points to the first element of the queue, while rear points to the last element. This helps in tracking where deletion and insertion happen.

When an element is inserted (enqueue), it is added at the position pointed by the rear, and then rear is increased by one.

When an element is deleted (dequeue), it is removed from the position pointed by front, and front is increased by one.

Initially, both front and rear are set to -1, which shows that the queue is empty. After the first insertion, front becomes 0.

A major problem with array representation is queue overflow, which occurs when rear reaches the maximum size of the array.


**2. Explain representation of queue using linked list.**

A queue can also be represented using a linked list, where each element is stored in a separate node. Each node contains data and a link to the next node.

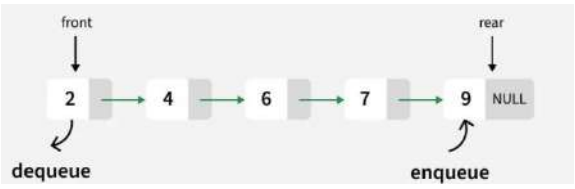
Two pointers are used: front points to the first node, and rear points to the last node in the linked list.

When an element is inserted, a new node is created and added at the end of the list. The rear pointer is updated to point to this new node.

When an element is deleted, the front pointer moves to the next node, and the first node is removed from memory.

If the queue becomes empty after deletion, both front and rear are set to NULL.

Linked list representation does not suffer from overflow, as memory is allocated dynamically when needed.



**3. Explain circular queue with suitable example.**

A circular queue is an improved version of a simple queue where the last position is connected back to the first position. This makes the queue work in a circular manner.

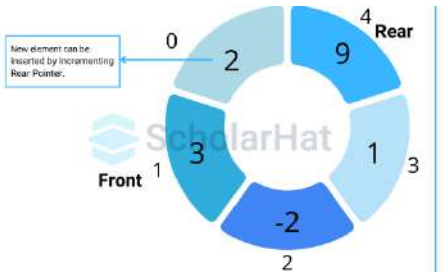
In a circular queue, the front and rear move circularly instead of moving only in one direction. This helps in reusing the empty spaces created after deletion.

When the rear reaches the last index of the array, it wraps around to the first index if there is free space available.

This structure avoids the problem of wasted memory, which occurs in a simple linear queue.

Insertion (enqueue) is done at the rear position, and deletion (dequeue) is done from the front position, just like a normal queue.

A circular queue is said to be full when the next position of rear is equal to front. This condition helps detect overflow correctly.




**4. Explain operations on dequeue with example.**

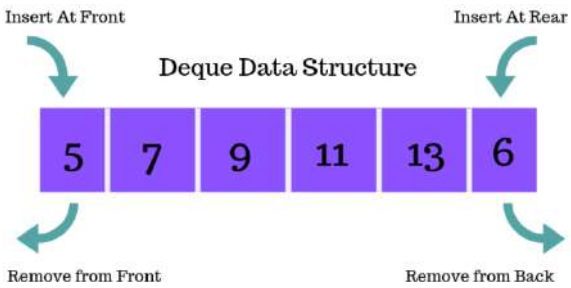
A deque (double ended queue) is a special type of queue in which insertion and deletion can be done from both ends.

The insertFront operation adds an element at the front end of the deque. For example, if deque contains 10, 20 and we insert 5 at front, it becomes 5, 10, 20.

The insertRear operation adds an element at the rear end of the deque. For example, inserting 30 at rear makes it 10, 20, 30.

The deleteFront operation removes the element from the front. For example, deleting from 10, 20, 30 removes 10.

The deleteRear operation removes the element from the rear. For example, deleting from 10, 20, 30 removes 30.



**5. Explain priority queue and its applications.**

A priority queue is a special type of queue in which each element is assigned a priority. Elements are processed based on priority instead of arrival time.

In a priority queue, the element with higher priority is deleted before elements with lower priority. If two elements have the same priority, they are processed in FIFO order.

Priority queues can be implemented using arrays, linked lists, or heaps. Heap implementation is most commonly used because it is efficient.

There are two types of priority queues: max-priority queue and min-priority queue. In a max-priority queue, the element with the highest priority is served first.

In a min-priority queue, the element with the lowest priority value is served first. Both types are used depending on the application.

Example: In a hospital system, emergency patients are treated before normal patients based on priority level.

**6. Differentiate between linear queue and circular queue.**

| **Linear Queue** | **Circular Queue** |
|---|---|
| Linear queue follows a simple straight-line structure. It does not reuse freed spaces easily. | Circular queue connects the last position of the queue back to the first position, forming a circle. |
| In linear queue, the rear moves only in the forward direction. Once it reaches the end, insertion stops. | In circular queue, both front and rear move in a circular manner using modulo operation. |
| Deletion from a linear queue creates unused memory space at the beginning. This leads to memory wastage. | Circular queue efficiently utilizes memory by reusing the empty spaces created after deletion. |
| Linear queue suffers from false overflow even when free space is available in the array. | Circular queue avoids the problem of false overflow completely. |
| Implementation of a linear queue is simple and easy to understand. | Implementation of a circular queue is slightly more complex than a linear queue. |
| Linear queues are suitable for small and less complex applications. | Circular queues are commonly used in real-time systems and scheduling applications. |

**7. Explain applications of queue in real-world scenarios.**

A queue is widely used in real-world situations where tasks are processed in the order they arrive. This follows the First In First Out (FIFO) principle.

In CPU scheduling, processes are stored in a queue and executed one by one based on arrival time. This ensures fair use of the CPU.

In printer systems, multiple print jobs are placed in a queue and printed in sequence. This avoids conflicts when many users send print commands.

Customer service systems like banks, ticket counters, and call centers use queues to serve customers in order. This maintains discipline and fairness.

In traffic management, vehicles waiting at signals are handled in a queue to control smooth movement.

Data buffering in devices such as keyboards and network routers uses queues to temporarily store data before processing.

In operating systems, queues are used in disk scheduling, task scheduling, and interrupt handling.

Queues are also used in breadth-first search (BFS) in graphs and level-order traversal in trees.

**8. Explain how queue can be used for finding a solution to the Josephus problem.**

The Josephus problem is a famous problem where people stand in a circle and every k-th person is eliminated repeatedly.

A queue is used to simulate this circular elimination process in a simple way. All people are first inserted into the queue.

The process starts by repeatedly removing the front element and inserting it back at the rear. This simulates moving in a circle.

When the count reaches the given number (k), the person at the front is removed permanently from the queue.

This elimination continues until only one person remains in the queue. That remaining person is the winner or survivor.

Example: If there are 5 people and every 2nd person is eliminated, people are rotated in the queue and every second one is removed.

Using a queue makes the Josephus problem easy to understand and implement. It clearly represents circular movement and elimination.

**[ 10 -marks ]**

1. **Explain queue data structure. Describe its representation using array and linked list with algorithms.**

A queue is a linear data structure that follows the principle of FIFO (First In First Out). This means the element inserted first is removed first.

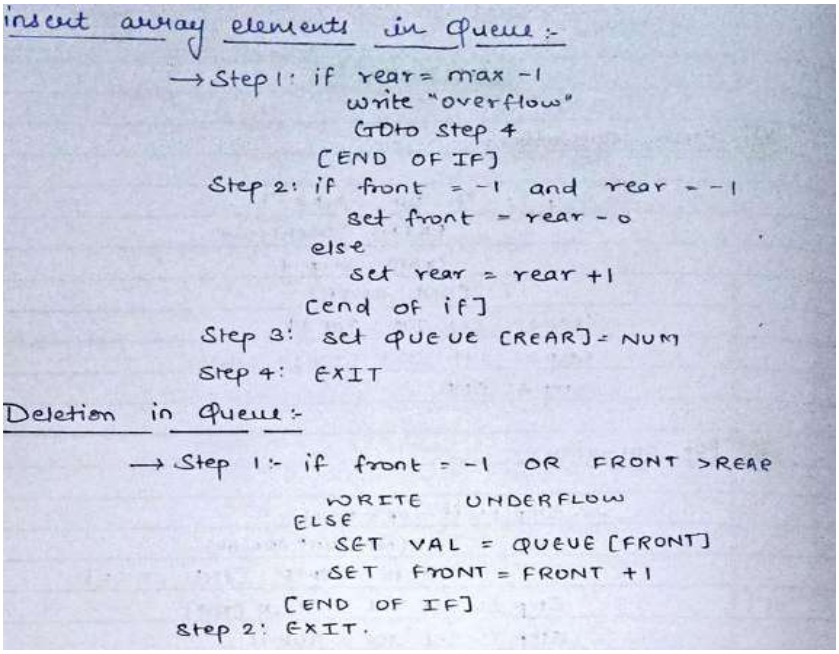
In a queue, insertion is done at one end called the rear, and deletion is done at the other end called the front.

Queues are commonly used in real-life situations such as waiting lines, task scheduling, and printer job management.

The two basic operations of a queue are enqueue (insert) and dequeue (delete).

**Queue Representation Using Array**

- In array representation, a fixed-size array is used to store queue elements in sequential order.
- Two variables, **front** and **rear**, are used to keep track of deletion and insertion positions.
- Initially, both front and rear are set to **-1**, indicating that the queue is empty.




**Queue Representation Using Linked List**

- In linked list representation, each element is stored in a node containing data and a link to the next node.
- Two pointers, **front** and **rear**, are used to point to the first and last nodes respectively.

**( no algorithm , write linked list algo of insetion at end , and deletion at front for this )**

**2. Explain circular queue in detail with insertion and deletion algorithms.**

A circular queue is a linear data structure that works on the FIFO (First In First Out) principle, just like a simple queue. The main difference is that the last position of the queue is connected back to the first position, forming a circle.

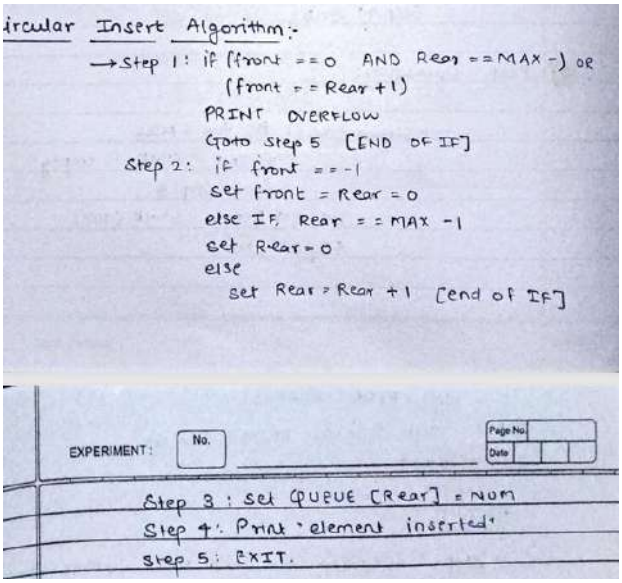
In a circular queue, the unused spaces created after deletion can be reused efficiently. This overcomes the major drawback of a linear queue, which suffers from memory wastage.

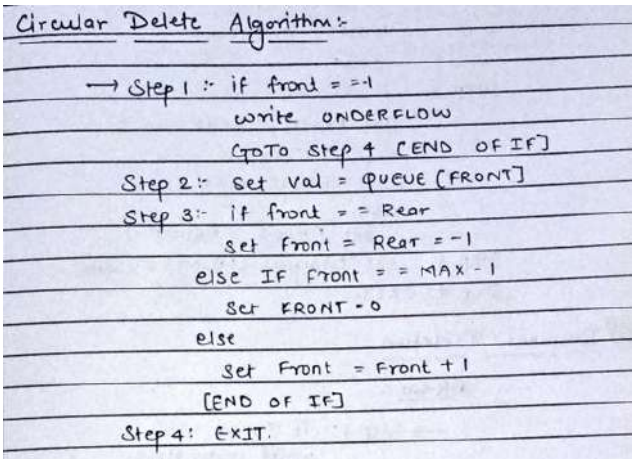
Circular queues are generally implemented using arrays along with two variables called front and rear.

Initially, both front and rear are set to -1, indicating that the queue is empty.

The queue is considered full when the next position of rear is equal to front. This condition is checked using the modulo operation.

The queue is considered empty when front is equal to -1.



**3. Explain priority queue in detail. Discuss its types, operations, and applications.**

A priority queue is a special type of queue in which each element is associated with a priority value. Elements are not processed strictly in the order of arrival.

In a priority queue, the element with the highest priority is removed first. If two elements have the same priority, they are served according to FIFO order.

Priority queues are widely used when some tasks are more important than others and must be handled earlier.

A priority queue can be implemented using arrays, linked lists, or heaps. Heap-based implementation is the most efficient and commonly used.

**Types of Priority Queue**

**Max Priority Queue:** In a max priority queue, the element with the highest priority value is removed first. It is commonly implemented using a max heap structure.

**Min Priority Queue:** In a min priority queue, the element with the lowest priority value is removed first. It is implemented using a min heap.

**Ascending Priority Queue:** Elements are arranged in ascending order of priority. Deletion happens from the front.

**Descending Priority Queue:** Elements are arranged in descending order of priority. Deletion also occurs from the front.

**Operations on Priority Queue**

**Insertion (Enqueue):** A new element is inserted according to its priority rather than at the rear. The queue is rearranged to maintain the correct priority order.

**Deletion (Dequeue**): The element with the highest (or lowest) priority is removed first. If multiple elements have the same priority, the one inserted earlier is removed first.

**Peek:** This operation returns the element with the highest priority without removing it.

**isEmpty / isFull**: These operations check whether the priority queue is empty or full.

Applications of Priority Queue

- Priority queues are used in CPU scheduling, where processes with higher priority get CPU time first.
- They are used in operating systems for job scheduling and interrupt handling.
- Network routing algorithms, such as shortest path algorithms, use priority queues for efficient processing.
- Priority queues are used in hospital management systems to handle emergency patients first.
- They are also used in event-driven simulations and task management systems.

4. **Explain dequeue with its types and operations using suitable examples.**

- A deque (double ended queue) is a special type of queue in which insertion and deletion can be performed at both ends.
- Unlike a normal queue where insertion happens at the rear and deletion at the front, a deque allows more flexibility in operations.
- Deque follows the basic concept of a queue but does not strictly follow FIFO because operations can occur at both ends.
- Deques can be implemented using arrays or linked lists, depending on memory and application needs.

Types of Deque

- Input-Restricted Deque:
    - In this type, insertion is allowed at only one end (usually the rear).
    - Deletion is allowed at both front and rear ends.
    - Example: Insert elements 10, 20, 30 from the rear. Deleting from front removes 10, and deleting from rear removes 30.
- Output-Restricted Deque:
    - In this type, deletion is allowed at only one end (usually the front).
    - Insertion is allowed at both front and rear ends.
    - Example: Insert 5 at front and 40 at rear. Deleting removes only the front element 5.

Operations on Deque with Examples

- Insert Front:
    - Adds an element at the front end of the deque.
    - Example: If deque is 20, 30, inserting 10 at front makes it 10, 20, 30.
- Insert Rear:
    - Adds an element at the rear end of the deque.
    - Example: If deque is 10, 20, inserting 30 at rear makes it 10, 20, 30.
- Delete Front:
    - Removes the element from the front end.
    - Example: From deque 10, 20, 30, deleting front removes 10.
- Delete Rear:
    - Removes the element from the rear end.
    - Example: From deque 10, 20, 30, deleting rear removes 30.

**5. Discuss various applications of queues in operating systems and real-time systems.**

1] Queues play a very important role in operating systems because many processes and resources must be managed in an orderly manner. They help maintain fairness and smooth execution.

2] In CPU scheduling, processes that are ready to execute are stored in a ready queue. The CPU selects processes from this queue based on scheduling algorithms like FCFS or Round Robin.

3] In printer management, print requests from multiple users are placed in a printer queue. Jobs are printed one by one in the order they arrive, avoiding conflicts.

4] Disk scheduling also uses queues to manage read and write requests efficiently. This helps reduce seek time and improves system performance.

5] Interrupt handling in operating systems uses queues to store interrupt requests until the CPU is ready to process them.

6] In real-time systems, queues are used to ensure tasks are executed within strict time limits. Tasks are often placed in priority queues based on urgency.

7] Message queues are used for communication between processes. One process sends data to the queue, and another process receives it safely.

8] In network systems, queues manage data packets waiting to be transmitted. This helps control traffic and prevent data loss.

9] Real-time systems such as traffic control systems, industrial automation, and embedded systems rely on queues to manage events and sensor data reliably.

6. **Explain the Linear Queue as Data structure. Write algorithms for adding and removing element from Queue.**

A **linear queue** is a linear data structure that follows the **FIFO (First In First Out)** principle. This means the element that is inserted first is removed first.

In a linear queue, insertion is done at one end called the **rear**, and deletion is done at the other end called the **front**.

Linear queues are usually implemented using an **array** with a fixed size.

Two variables, **front** and **rear**, are used to keep track of the queue positions.

Initially, both front and rear are set to **-1**, which indicates that the queue is empty.