Course Details

Course Title : Data Structure and Algorithms (3 Cr.)

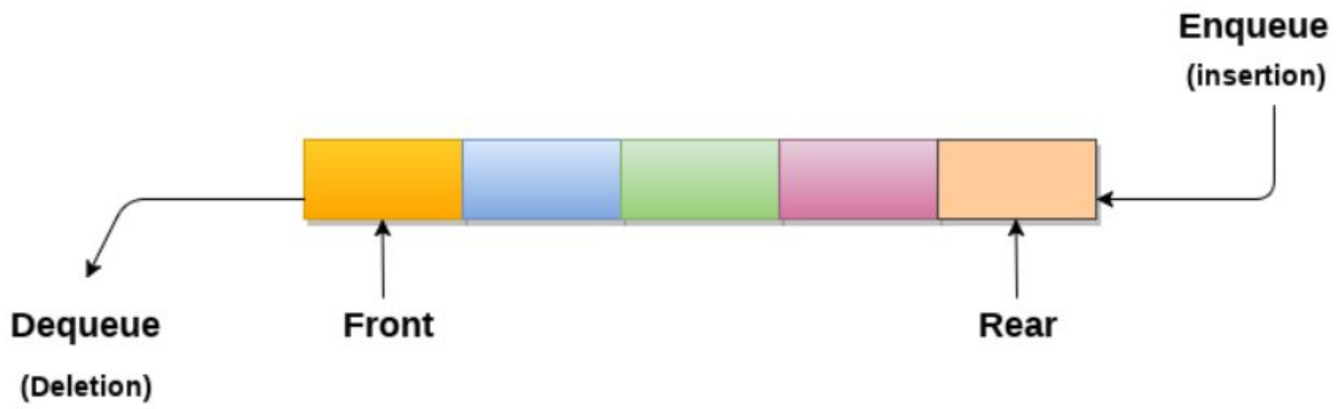
Course Code : CACS201

Why should we learn DSA?

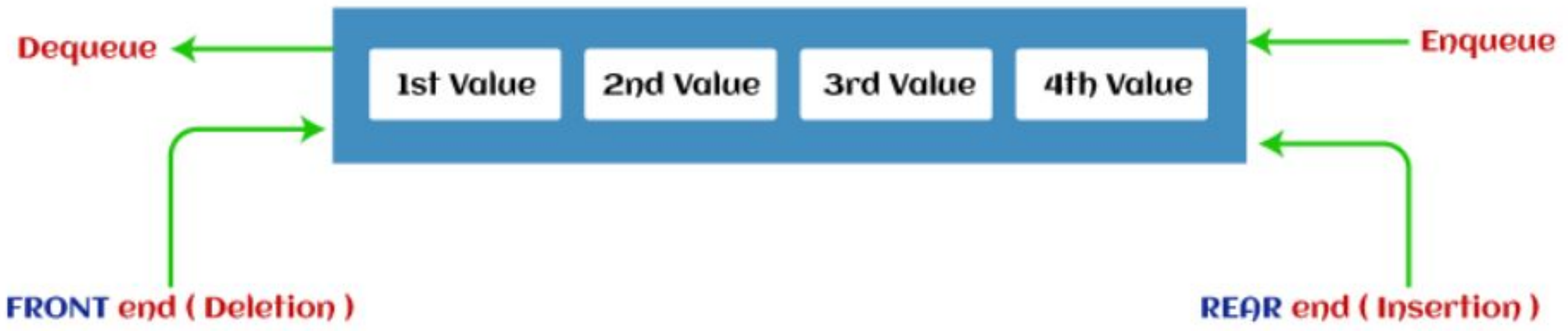
1. Data Structures allow us to organize and store data, whereas Algorithms allow us to process that data meaningfully.
2. Learning Data Structures and Algorithms will help us become better Programmers.
3. We will be able to write code that is more effective and reliable.
4. We will also be able to solve problems more quickly and efficiently.

<u>Unit 3</u>
The Queue

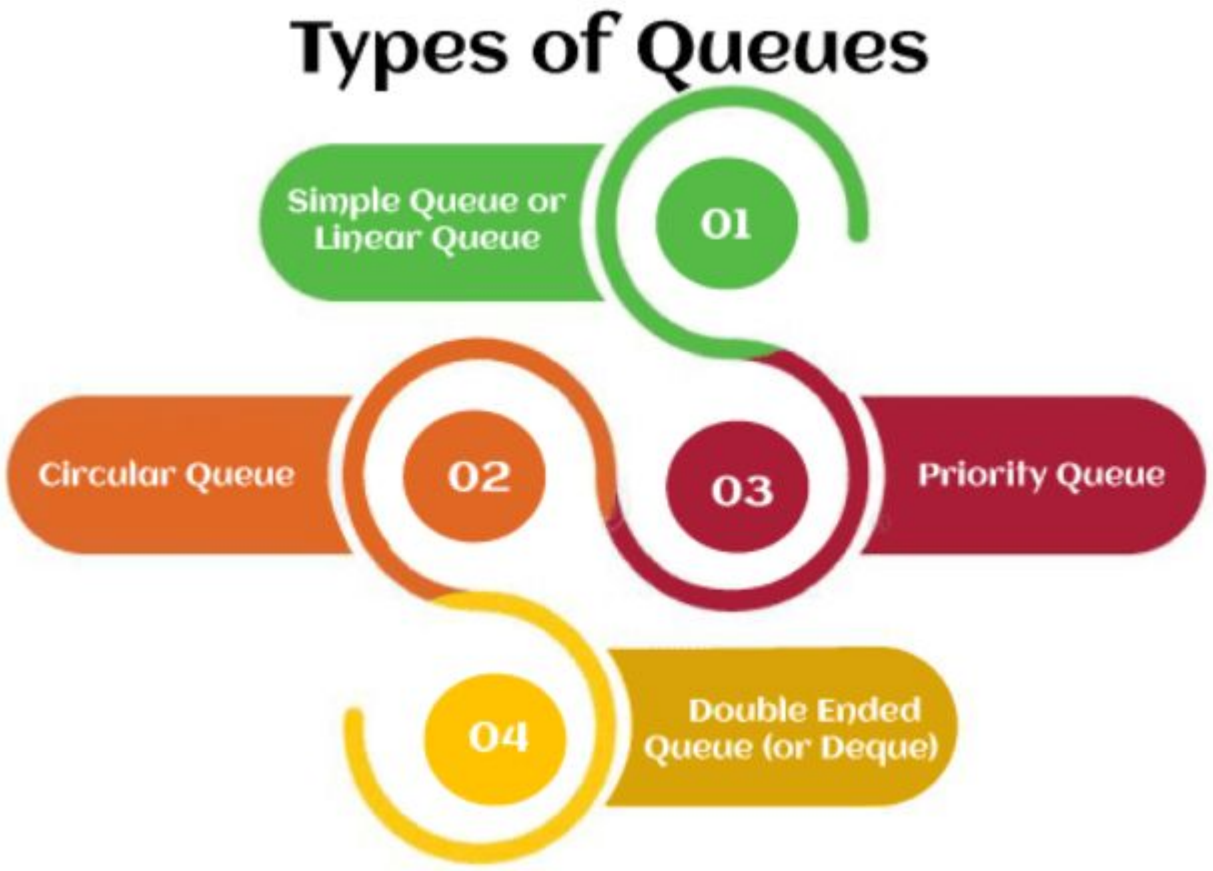
Queue



- A queue can be defined as an ordered list which enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT.
- Queue is referred to be as First In First Out list. For example, people waiting in line for a rail ticket form a queue.



Types of Queue



Operations performed on queue

- The most fundamental operations in the queue ADT include: enqueue(), dequeue(), peek(), isFull(), isEmpty(). These are all built-in operations to carry out data manipulation and to check the status of the queue.
- Queue uses two pointers − front and rear. The front pointer accesses the data from the front end (helping in enqueueing) while the rear pointer accesses data from the rear end (helping in dequeuing).

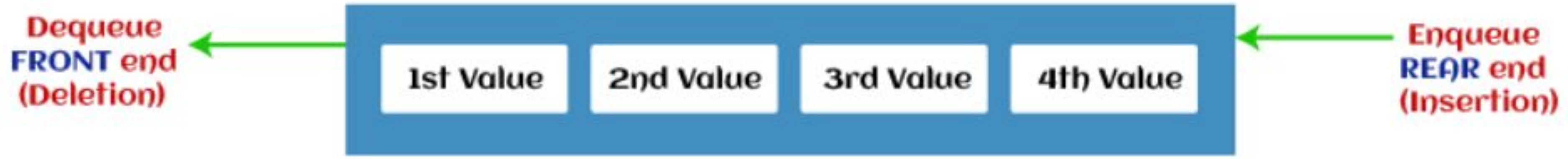
1. **Enqueue:** The Enqueue operation is used to insert the element at the rear end of the queue. It returns void.
2. **Dequeue:** It performs the deletion from the front-end of the queue. It also returns the element which has been removed from the front-end. It returns an integer value.
3. **Peek:** This is the third operation that returns the element, which is pointed by the front pointer in the queue but does not delete it.
4. **Queue overflow (isfull):** It shows the overflow condition when the queue is completely full.
5. **Queue underflow (isempty):** It shows the underflow condition when the Queue is empty, i.e., no elements are in the Queue.

Simple Queue or Linear Queue

- In Linear Queue, an insertion takes place from one end while the deletion occurs from another end. The end at which the insertion takes place is known as the rear end, and the end at which the deletion takes place is known as front end.
- It strictly follows the FIFO rule.

Disadvantages

- Linear Queue is that insertion is done only from the rear end.
- If the first three elements are deleted from the Queue, we cannot insert more elements even though the space is available in a Linear Queue. In this case, the linear Queue shows the overflow condition as the rear is pointing to the last element of the Queue.



Simple Queue or Linear Queue

Working of Queue

Queue operations work as follows:

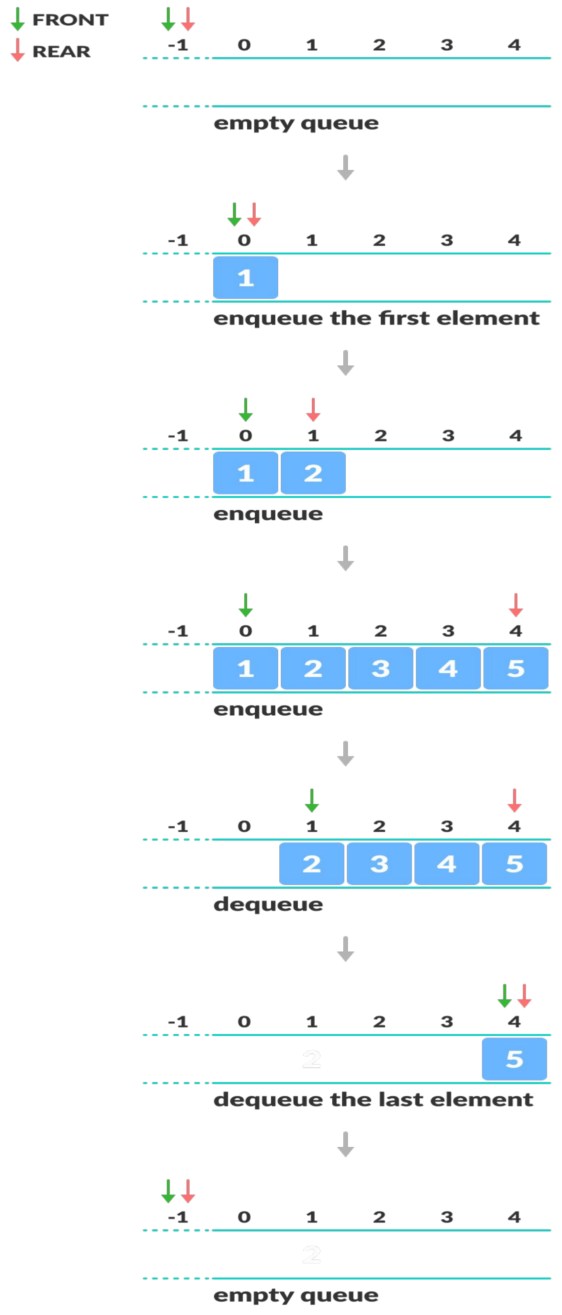
- Two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last element of the queue
- Initially, set value of FRONT and REAR to -1

Enqueue Operation

- Check if the queue is **full**
- For the first element, set the value of FRONT to 0
- **Increase** the REAR index by 1
- Add the new element in the **position pointed** to by REAR

Dequeue Operation

- Check if the queue is empty
- Return the value pointed by **FRONT**
- Increase the **FRONT** index by 1
- For the **last element**, reset the values of FRONT and REAR to -1



Simple Queue or Linear Queue

Working of Queue

Queue operations work as follows:

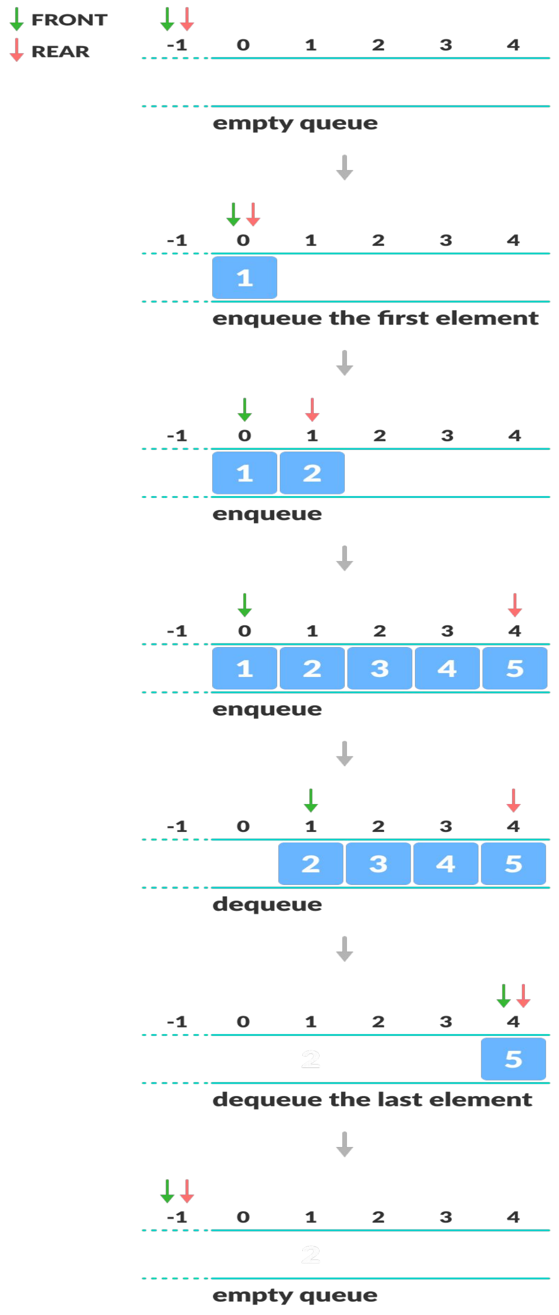
- Two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last element of the queue
- Initially, set value of FRONT and REAR to -1

Enqueue Operation

- Check if the queue is **full**
- For the first element, set the value of FRONT to 0
- **Increase** the REAR index by 1
- Add the new element in the **position pointed** to by REAR

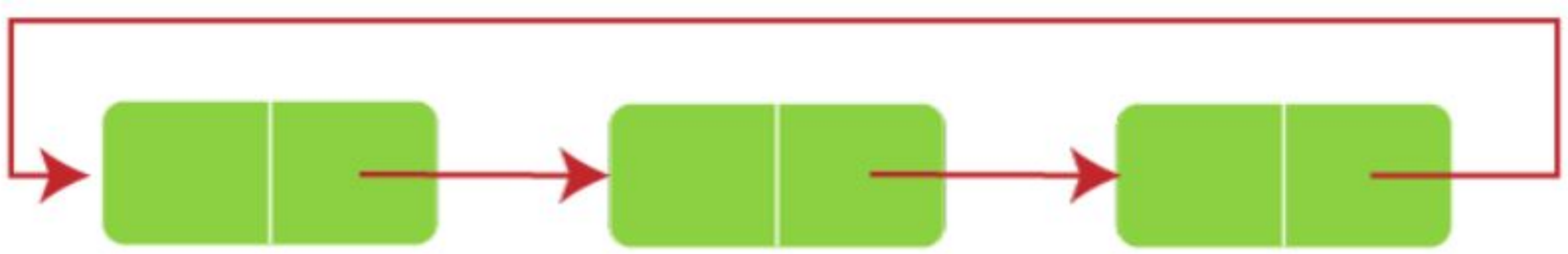
Dequeue Operation

- Check if the queue is empty
- Return the value pointed by **FRONT**
- Increase the **FRONT** index by 1
- For the **last element**, reset the values of FRONT and REAR to -1



Circular Queue

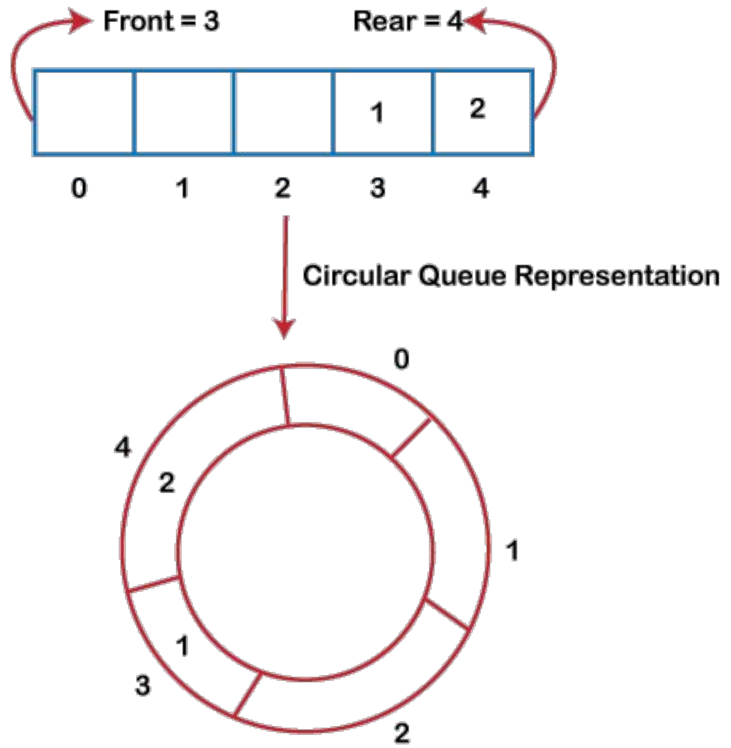
- In Circular Queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as Ring Buffer, as all the ends are connected to another end. It strictly follows the FIFO rule.



Circular Queue

Circular Queue

- A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a **Ring Buffer**.



Circular Queue

Circular Queue Operations:

- Two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last elements of the queue
- initially, set value of FRONT and REAR to -1

1. Enqueue Operation

- Check if the queue is full
- For the first element, set value of FRONT to 0
- Circularly increase the REAR index by 1 (i.e. if the rear reaches the end, next it would be at the start of the queue)
- Add the new element in the position pointed to by REAR

2. Dequeue Operation

- Check if the queue is empty
- Return the value pointed by FRONT
- Circularly increase the FRONT index by 1
- For the last element, reset the values of FRONT and REAR to -1
- However, the check for full queue has a new additional case:

  Case 1: FRONT = 0 && REAR == SIZE - 1

  Case 2: FRONT = REAR + 1

  The **second case** happens when REAR starts from 0 due to circular increment and when its value is just 1 less than FRONT, the queue is full.

Enqueue Operation

The steps of enqueue operation are given below:

- First, we will check whether the Queue is full or not.
- Initially the front and rear are set to -1. When we insert the first element in a Queue, front and rear both are set to 0.
- When we insert a new element, the rear gets incremented, i.e., **rear=rear+1**.

Scenarios for inserting an element

There are two scenarios in which queue is not full:

- **If rear != max - 1**, then rear will be incremented to **mod(maxsize)** and the new value will be inserted at the rear end of the queue.
- **If front != 0** and **rear = max - 1**, it means that queue is not full, then set the value of rear to 0 and insert the new element there.

There are two cases in which the element cannot be inserted:

- When **front ==0** && **rear = max-1**, which means that front is at the first position of the Queue and rear is at the last position of the Queue.
- front== rear + 1;

Algorithm to insert an element in a circular queue

```
Step 1: IF (REAR+1)%MAX = FRONT
        Write " OVERFLOW "
        Goto step 4
        [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
        SET FRONT = REAR = 0
           ELSE IF
        REAR = MAX - 1 and FRONT ! = 0
        SET REAR = 0
           ELSE
        SET REAR = (REAR + 1) % MAX
           [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```

Dequeue Operation

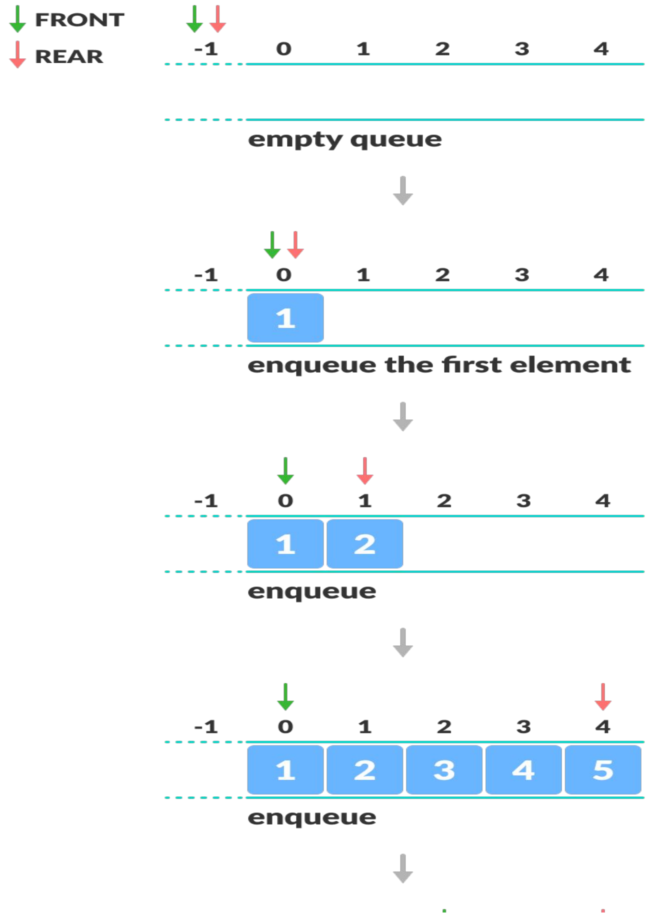
The steps of dequeue operation are given below:

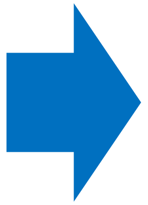
- First, we check whether the Queue is empty or not. If the queue is empty, we cannot perform the dequeue operation.
- When the **element is deleted**, the value of front gets incremented by 1.
- If there is **only one element left** which is to be deleted, then the **front and rear are reset to -1**.

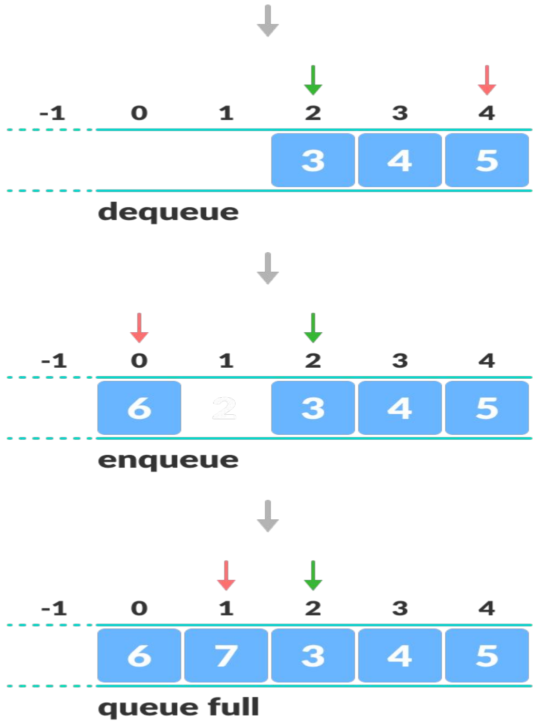
Algorithm to delete an element from the circular queue

```
Step 1: IF FRONT = -1
        Write " UNDERFLOW "
        Goto Step 4
        [END of IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT = REAR
        SET FRONT = REAR = -1
           ELSE
        IF FRONT = MAX -1
                SET FRONT = 0
        ELSE
                SET FRONT = FRONT + 1
        [END of IF]
            [END OF IF]
Step 4: EXIT
```

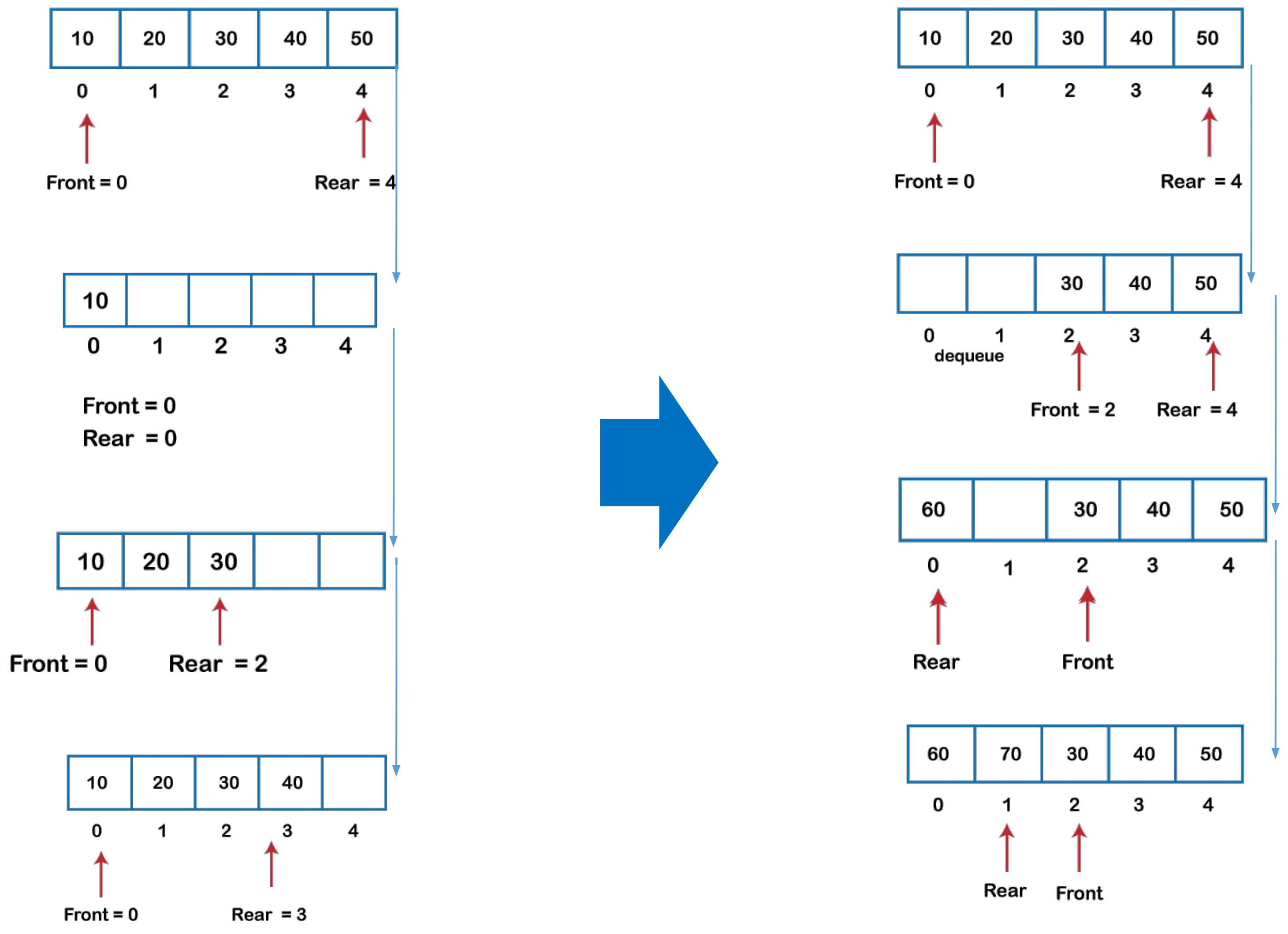
Circular Queue







Circular Queue

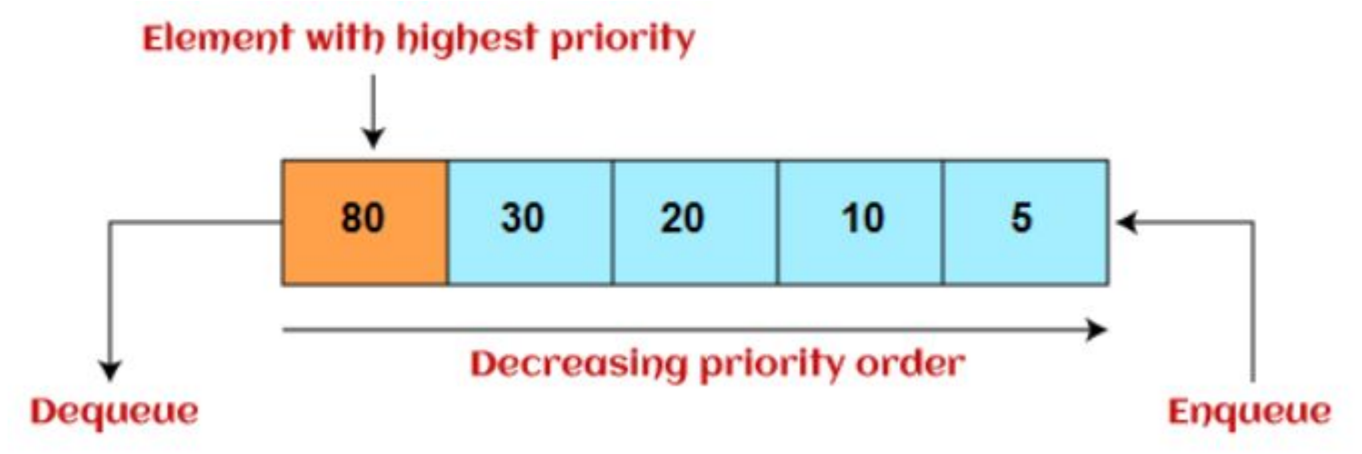


Priority Queue

- It is a special type of queue in which the elements are arranged based on the priority. It is a special type of queue data structure in which every element has a priority associated with it. Suppose some elements occur with the same priority, they will be arranged according to the FIFO principle.
- Insertion in priority queue takes place based on the arrival, while deletion in the priority queue occurs based on the priority. Priority queue is mainly used to implement the CPU scheduling algorithms.

Types of priority queue:

1. *Ascending priority queue:* Elements can be inserted in arbitrary order, but only smallest can be deleted first.
2. *Descending priority queue:* Elements can be inserted in arbitrary order, but only the largest element can be deleted first.

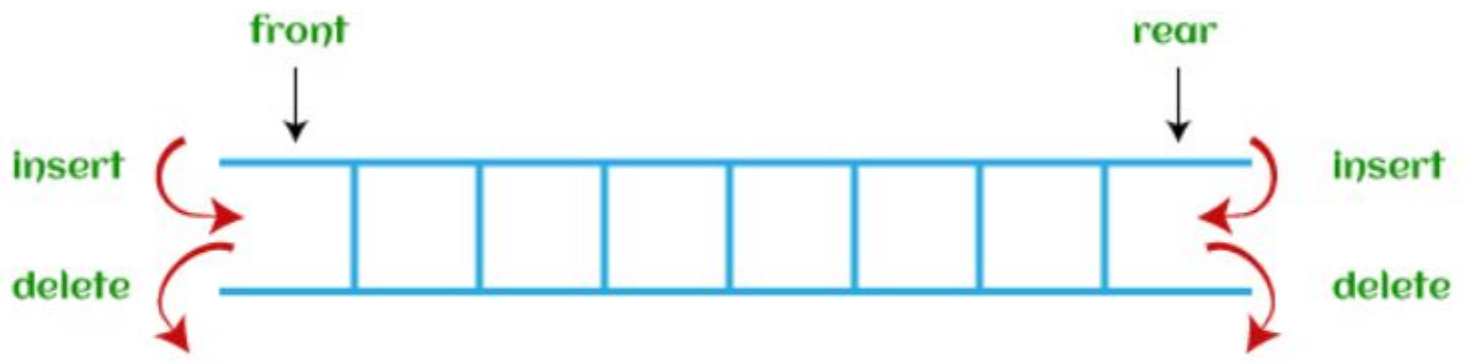


Deque (or, Double Ended Queue)

- In Deque or Double Ended Queue, insertion and deletion can be done from both ends of the queue either from the front or rear.
- It means that we can insert and delete elements from both front and rear ends of the queue. Deque can be used as a palindrome checker means that if we read the string from both ends, then the string would be the same.

Types of priority dequeue:

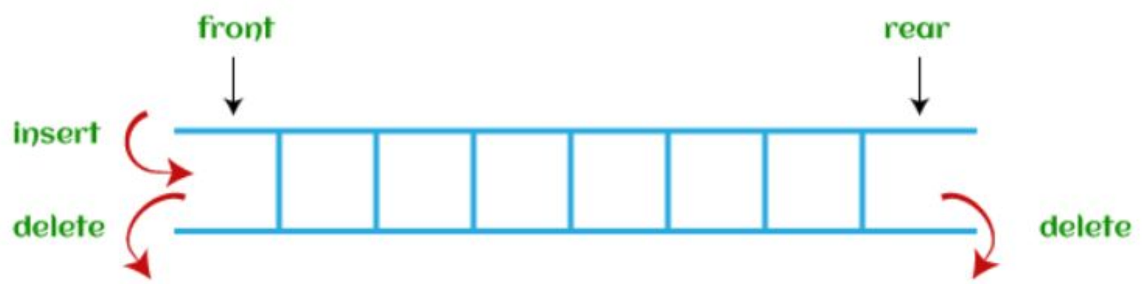
1. *Input restricted deque :* Insertion operation can be performed at only one end, while deletion can be performed from both ends.
2. *Output restricted deque:* Deletion operation can be performed at only one end, while insertion can be performed from both ends.



double ended queue

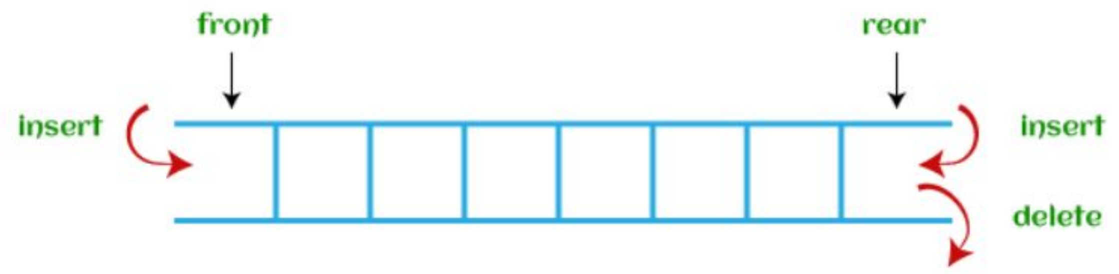
Types of priority dequeue:

1. *Input restricted deque : I*nsertion operation can be performed at only one end, while deletion can be performed from both ends.



input restricted double ended queue

2. *Output restricted deque:* Deletion operation can be performed at only one end, while insertion can be performed from both ends.



Output restricted double ended queue

Applications of Queue

1. Task based on FIFO principle.
2. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
3. Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg: pipes, file IO, sockets.
4. Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
5. Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.
6. Queues are used in operating systems for handling interrupts.

Complexity

Data Structure	Time Complexity								Space Compleity
	Average				**Worst**				**Worst**
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Queue	θ(n)	θ(n)	θ(1)	θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

Thank You