E- Lecture
Data Structures and Algorithms

By
H R Choudhary (Asstt. Professor)
Department of CSE
Engineering College Ajmer

Topics to be covered
Stack

What do you mean by Queue ?

Opertaions on Queue

What do you mean by Queue ?

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Queue..

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Queue..

Queue Representation:

As we now understand that in queue, we access both ends for different reasons. The following diagram given below tries to explain queue representation as data structure −

Queue..

As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures. For the sake of simplicity, we shall implement queues using one-dimensional array.

Basic Operations

Basic Operations:

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues −

- **Insertion** − add (enqueue) an item to the queue.
- **Deletion** − remove (dequeue) an item from the queue.

Basic Operations..

Few more functions are required to make the above-mentioned queue operation efficient. These are −

- **peek()** − Gets the element at the front of the queue without removing it.
- **isfull()** − Checks if the queue is full.
- **isempty()** − Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by front pointer and while enqueing (or storing) data in the queue we take help of rear pointer.

Basic Operations..

First learn about supportive functions of a queue −

Peek():

This function helps to see the data at the front of the queue. The algorithm of peek() function is as follows −

```
begin procedure peek
   return queue[front]
end procedure
```

Basic Operations..

Implementation of peek() function in C programming language −

```
int peek() {
   return queue[front];
}
```

Basic Operations..

Isfull():

As we are using single dimension array to implement queue, we just check for the rear pointer to reach at MAXSIZE to determine that the queue is full. In case we maintain the queue in a circular linked-list, the algorithm will differ. Algorithm of isfull() function −

```
begin procedure isfull

   if rear equals to MAXSIZE
      return true
   else
      return false
   endif

end procedure
```

Basic Operations..

Implementation of isfull() function in C programming language −

```
bool isfull() {
   if(rear == MAXSIZE - 1)
      return true;
   else
      return false;
}
```

Basic Operations..

Isempty():

Algorithm of isempty() function −

```
begin procedure isempty

   if front is less than MIN  OR front is greater than rear
      return true
   else
      return false
   endif

end procedure
```

Basic Operations..

If the value of front is less than MIN or 0, it tells that the queue is not yet initialized, hence empty.

Here's the C programming code −

```
bool isempty() {
   if(front < 0 || front > rear)
      return true;
   else
      return false;
}
```

Basic Operations..

Insertion Operation:

Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue −

Step 1 − Check if the queue is full.

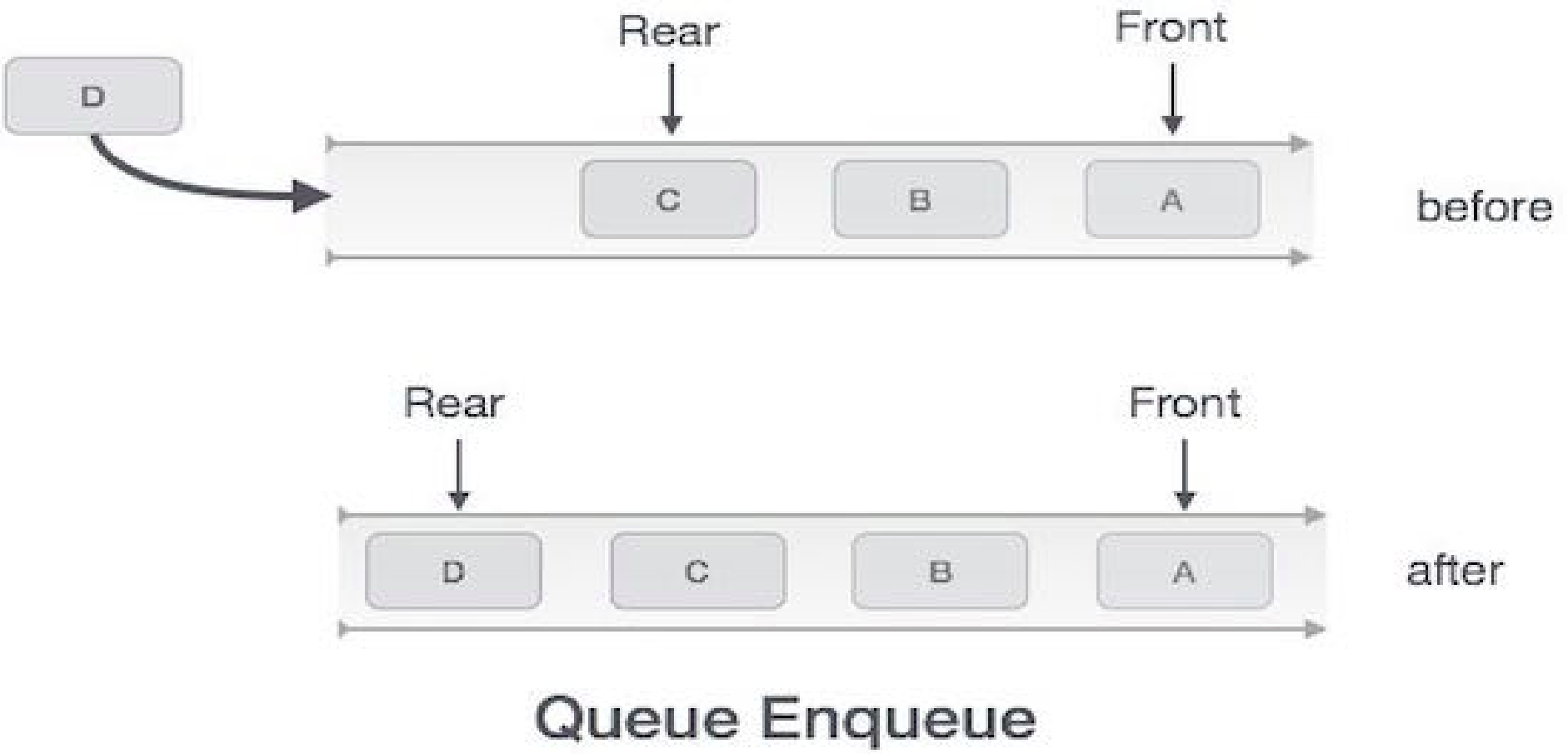
Step 2 − If the queue is full, produce overflow error and exit.

Step 3 − If the queue is not full, increment rear pointer to point the next empty space.

Step 4 − Add data element to the queue location, where the rear is pointing.

Step 5 − return success.

Basic Operations..



Queue Enqueue

Basic Operations..

Algorithm for enqueue(Insertion) operation:

```
procedure enqueue(data)
   if queue is full
      return overflow
   endif

   rear ← rear + 1
   queue[rear] ← data
   return true

end procedure
```

Basic Operations..

Implementation of enqueue() in C programming language −

```
int enqueue(int data)
   if(isfull())
      return 0;

   rear = rear + 1;
   queue[rear] = data;

   return 1;
end procedure
```

Basic Operations..

Deletion Operation:

Accessing data from the queue is a process of two tasks − access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation −

Step 1 − Check if the queue is empty.

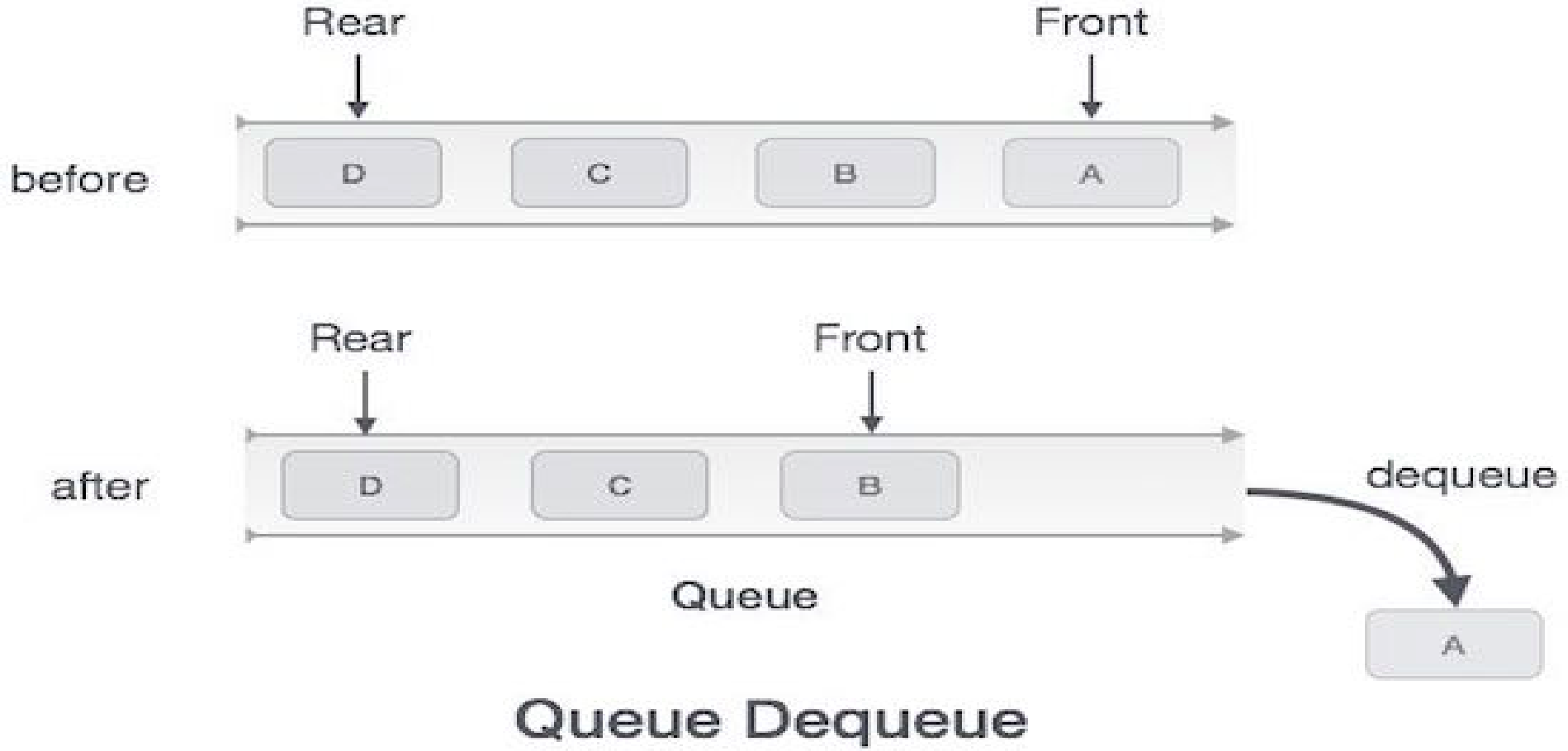
Step 2 − If the queue is empty, produce underflow error and exit.

Step 3 − If the queue is not empty, access the data where front is pointing.

Step 4 − Increment front pointer to point to the next available data element.

Step 5 − Return success.

Basic Operations..



Queue Dequeue

Basic Operations..

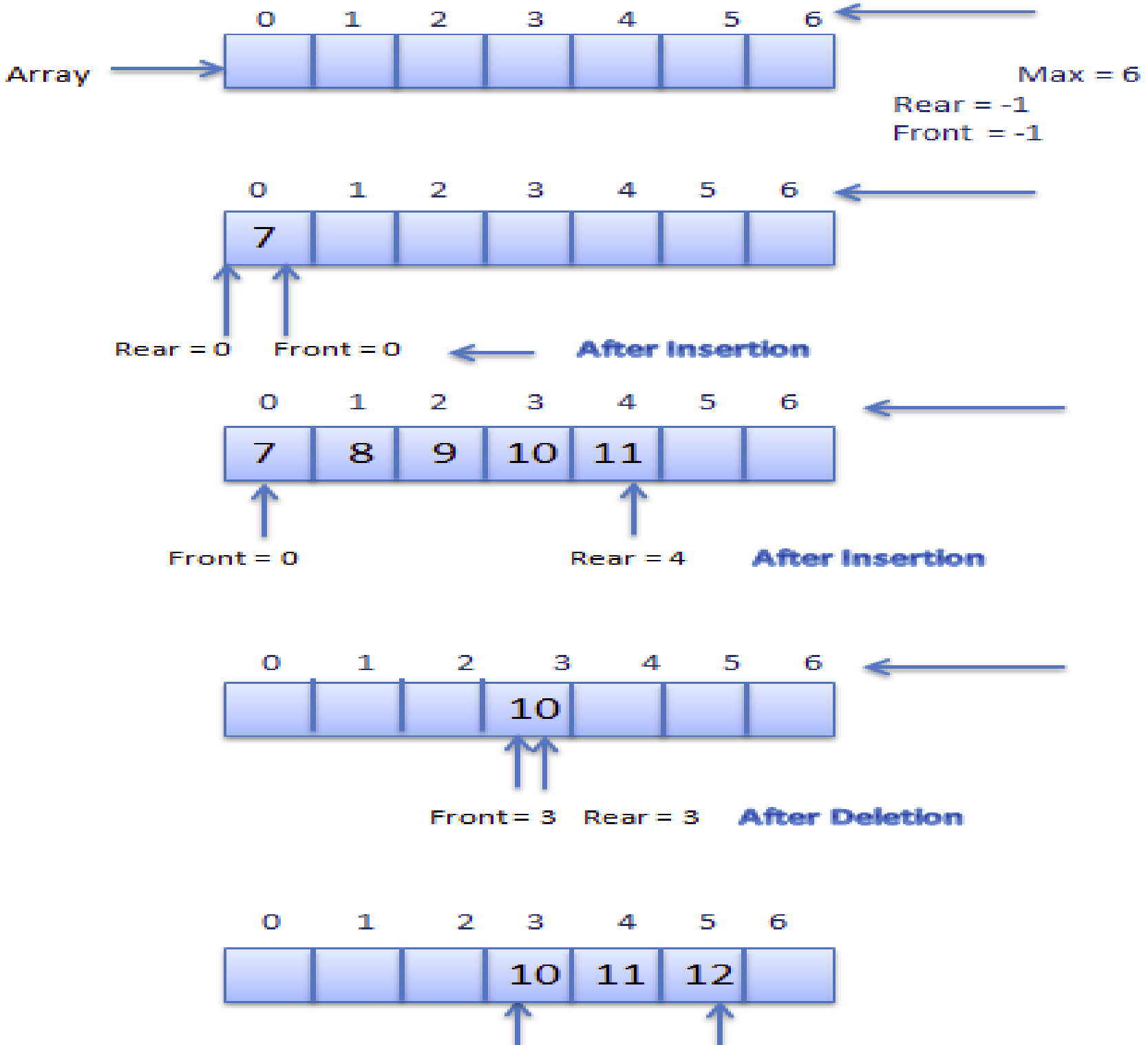
Algorithm for dequeue operation:

```
procedure dequeue

   if queue is empty
      return underflow
   end if

   data = queue[front]
   front ← front + 1
   return true

end procedure
```

Basic Operations..

Implementation of dequeue() in C programming language −

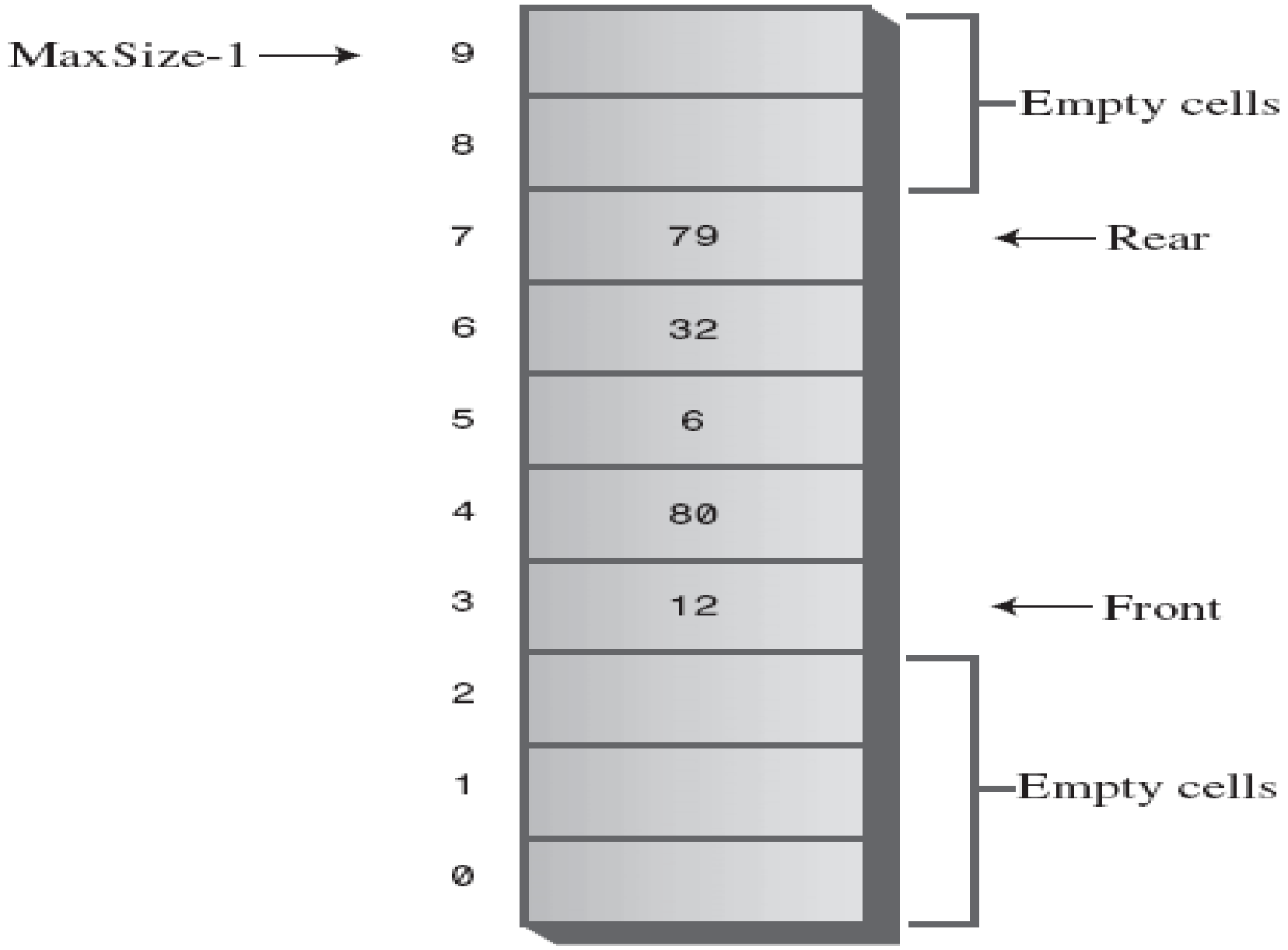
```
int dequeue() {
   if(isempty())
      return 0;

   int data = queue[front];
   front = front + 1;

   return data;
}
```

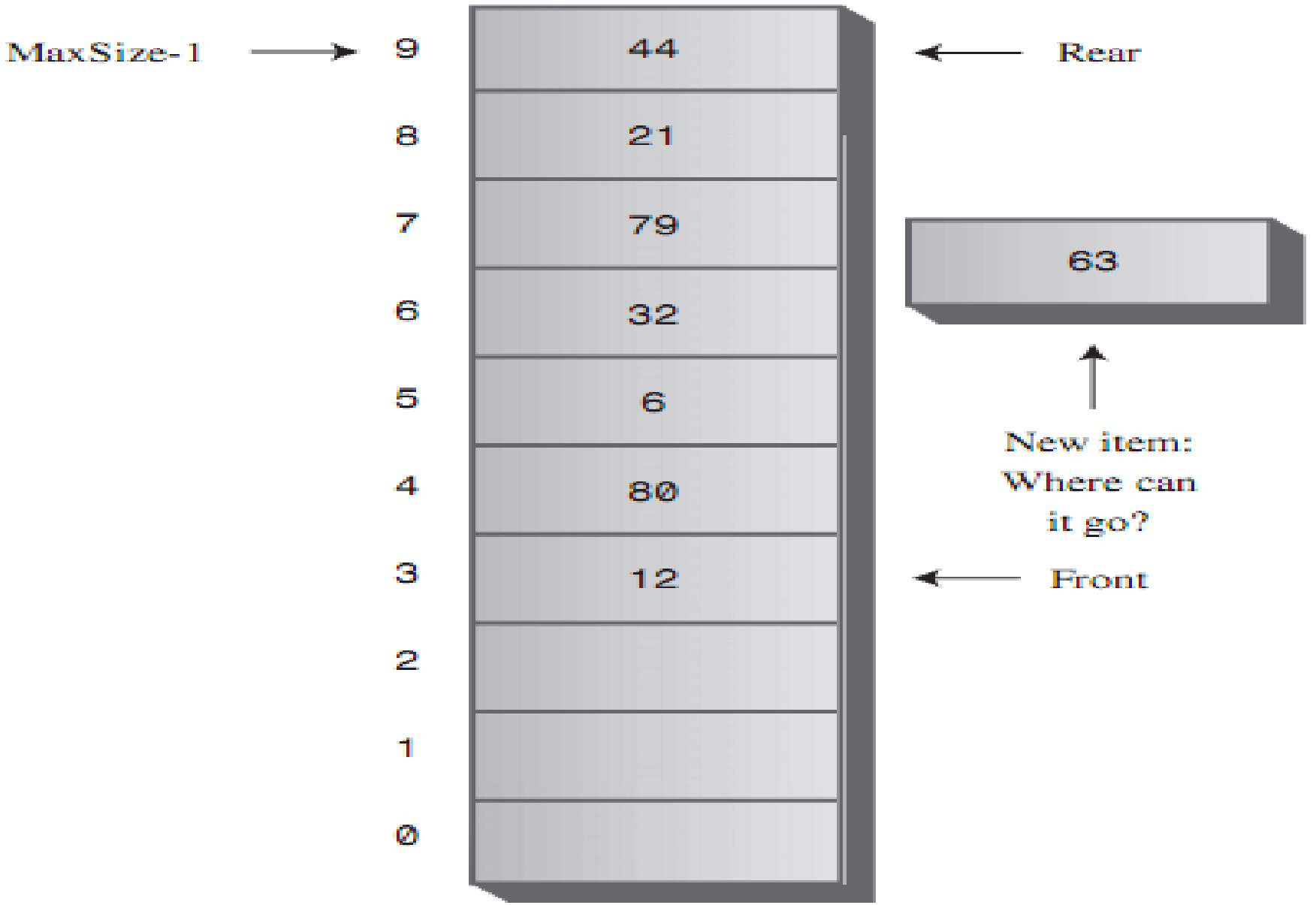

0 1 2 3 4 5 6
Array
Max = 6
Rear = -1
Front = -1
0 1 2 3 4 5 6
7
Rear = 0
Front = 0
After Insertion
0 1 2 3 4 5 6
7 8 9 10 11
Front = 0
Rear = 4
After Insertion
0 1 2 3 4 5 6
10
Front = 3
Rear = 3
After Deletion
0 1 2 3 4 5 6
10 11 12

Drawback of Linear Queue

The linear queue suffers from serious drawback that performing some operations, we can not insert items into queue, even if there is space in the queue. Suppose we have queue of 5 elements and we insert 5 items into queue, and then delete some items, then queue has space, but at that condition we can not insert items into queue.

The trouble with this arrangement is that pretty soon the rear of the queue is at the end of the array (the highest index). Even if there are empty cells at the beginning of the array, because we have removed them, we still can't insert a new item because Rear can't go any further. This situation is shown below.

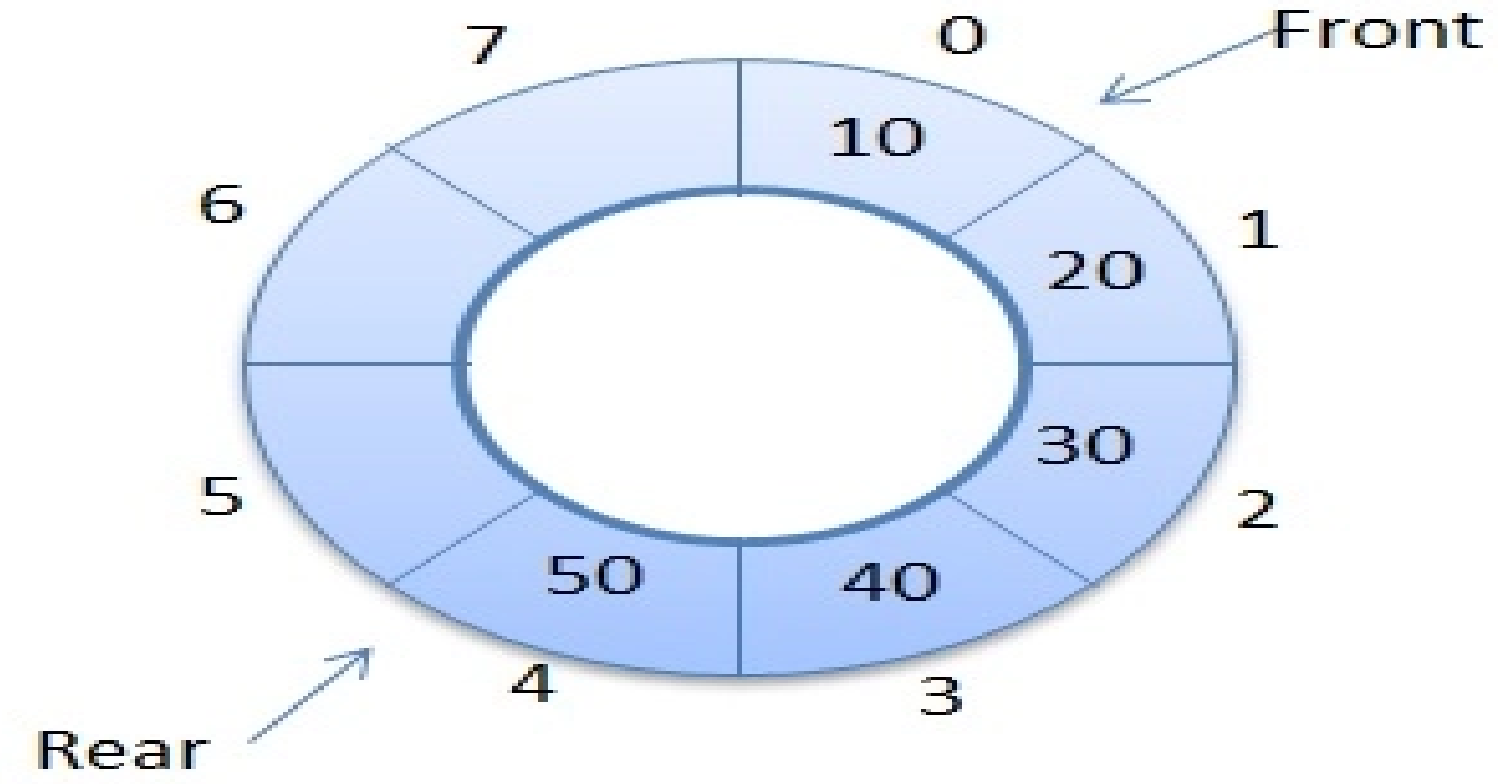

MaxSize-1
9
8
7
79
6
32
5
6
4
80
3
12
2
1
0
Empty cells
Rear
Front
Empty cells


MaxSize-1
9
8
7
6
5
4
3
2
1
0
44
21
79
32
6
80
12
Rear
63
New item:
Where can
it go?
Front

To avoid the problem of not being able to insert more items into the queue even when it's not full, the Front and Rear arrows wrap around to the beginning of the array. The results a circular queue(sometimes called a ring buffer).

What is Circular Queue?

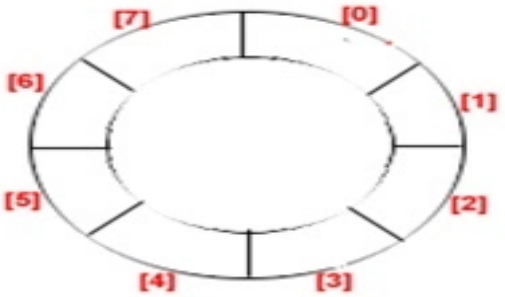
The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.Here Max=8



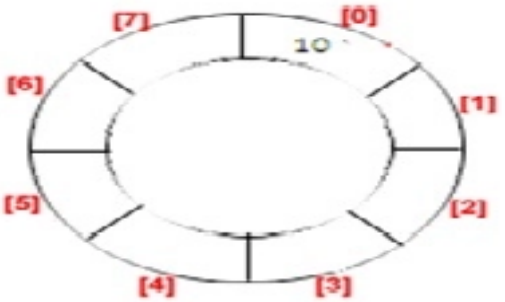
Note that the container of items is an array. Array is stored in main memory. Main memory is linear. So this circularity is only logical. There can not be physical circularity in main memory.

Consider the example with Circular Queue implementation:

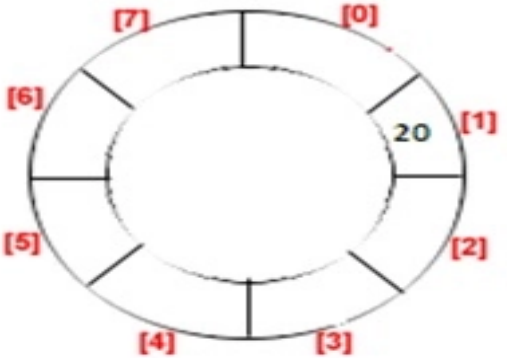
1) Initially: **Front = 0** and **rear = -1**



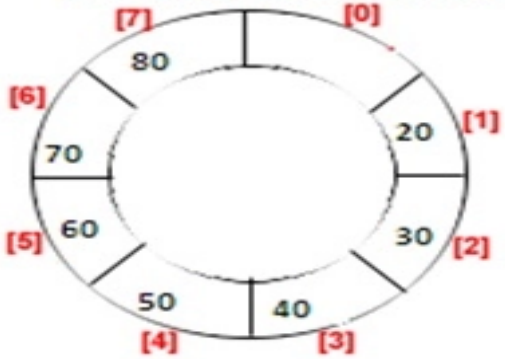
2) Add item 10 then **front = 0** and **rear =0**.



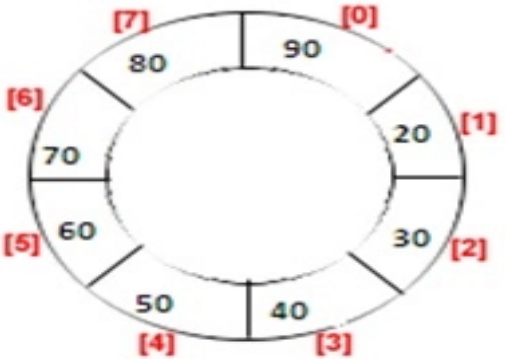
3) Now delete one item then **front = 1** and **rear =1**.



4) Like this now insert 30, 40, and 50,50,70,80 respectability then **front =1** and **rear = 7.**



5) Now in case of linear queue, we can not access 0 block for insertion but in circular queue next item will be inserted of 0 block then **front =0** and **rear = 0**.



See the logical circularity of the queue. Addition causes the increment in REAR. It means that when REAR reaches MAX-1 position then Increment in REAR causes REAR to reach at first position that is 0.

```
if( rear  == MAX -1 )
   rear  = 0;
else
   rear = rear + 1;
```

The short-hand equivalent representation may be

rear = (rear + 1) % MAX;

As we know that, Deletion causes the increment in FRONT. It means that when FRONT reaches the MAX-1 position, then increment in FRONT, causes FRONT to reach at first position that is 0.

```
if( front  == MAX -1 )
   front  = 0;
else
   front = front + 1;
```

The short-hand equivalent representation may be

front = (front + 1) % MAX;

Drawback of Circular Queue:

The drawback of circular queue is , difficult to distinguished the full and empty cases. It is also known as boundary case problem.

In circular queue it is necessary that:

Before insertion, fullness of Queue must be checked (for overflow).

Before deletion, emptiness of Queue must be checked (for underflow).

Condition to check FULL Circular Queue:

If **((rear == Max-1 && front == 0) || (front==Rear+1))**

Condition to check EMPTY Circular Queue:

if((front ==0 && Rear==-1)

Solution of the drawback of circular Queue:

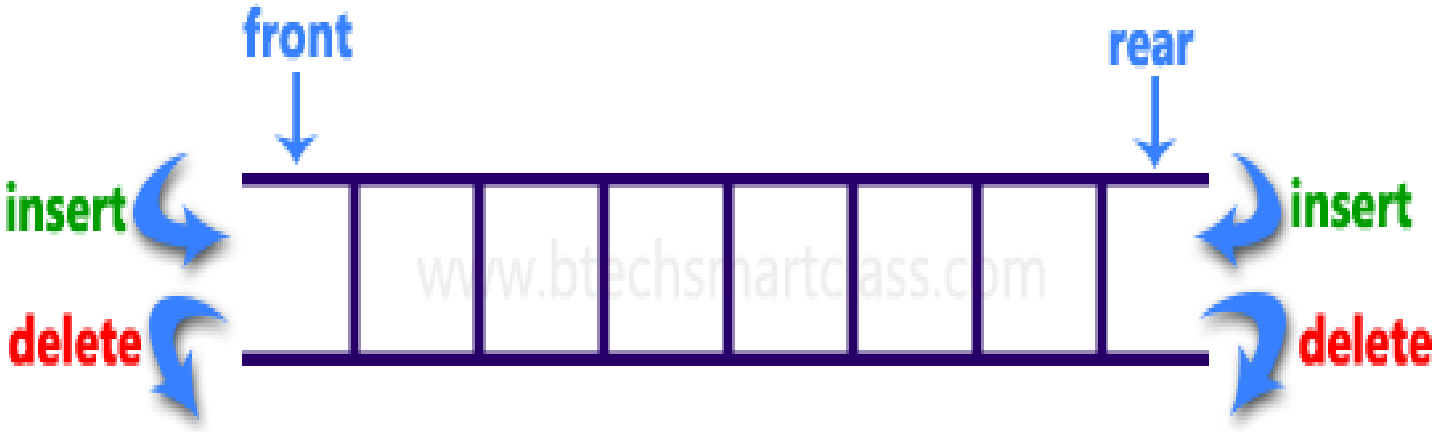
Use **count variable** to hold the current position (in case of insertion or deletion).

Operation of Circular Queue using count:

- Initialize Operation.
- Is_Full check.
- Is_Empty check.
- Addition or Insertion operation.
- Deletion operation.

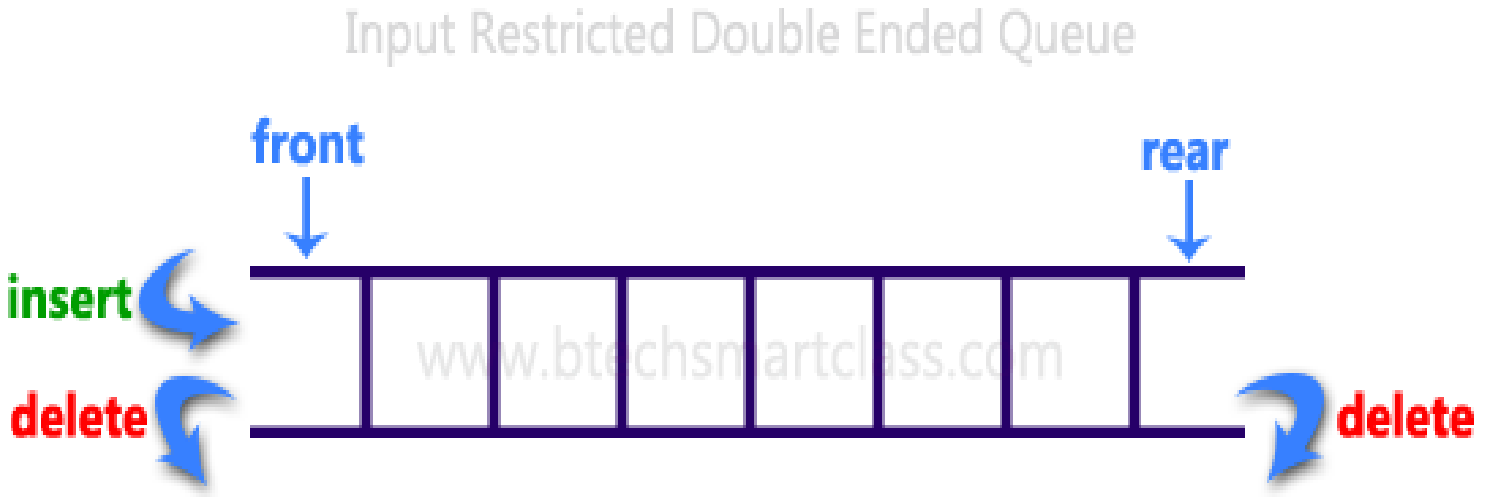
Double Ended Queue(Deque)

Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (front and rear). That means, we can insert at both front and rear positions and can delete from both front and rear positions.



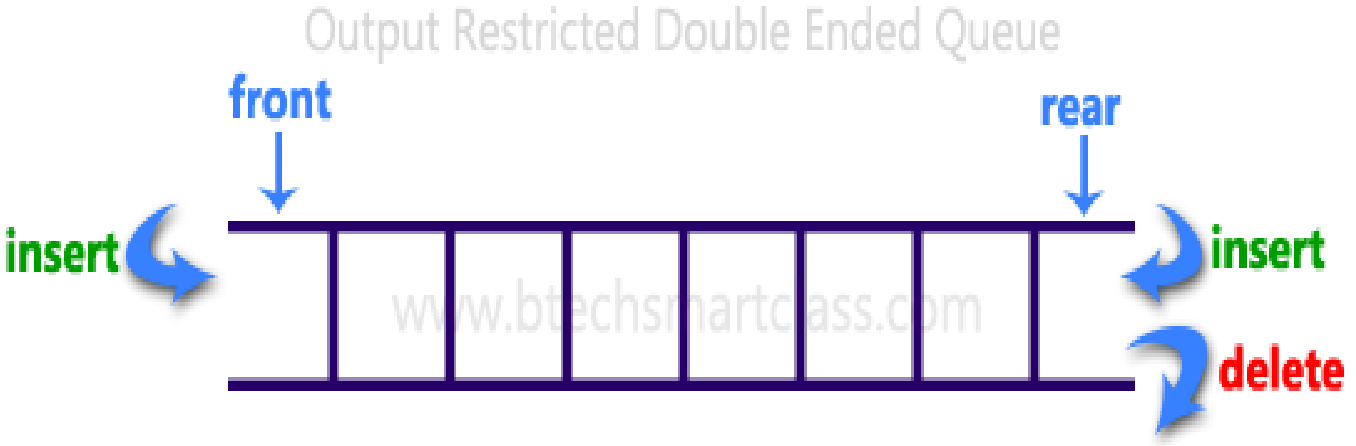
Input Restricted Double Ended Queue:

In input restricted double-ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.



Output Restricted Double Ended Queue:

In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.



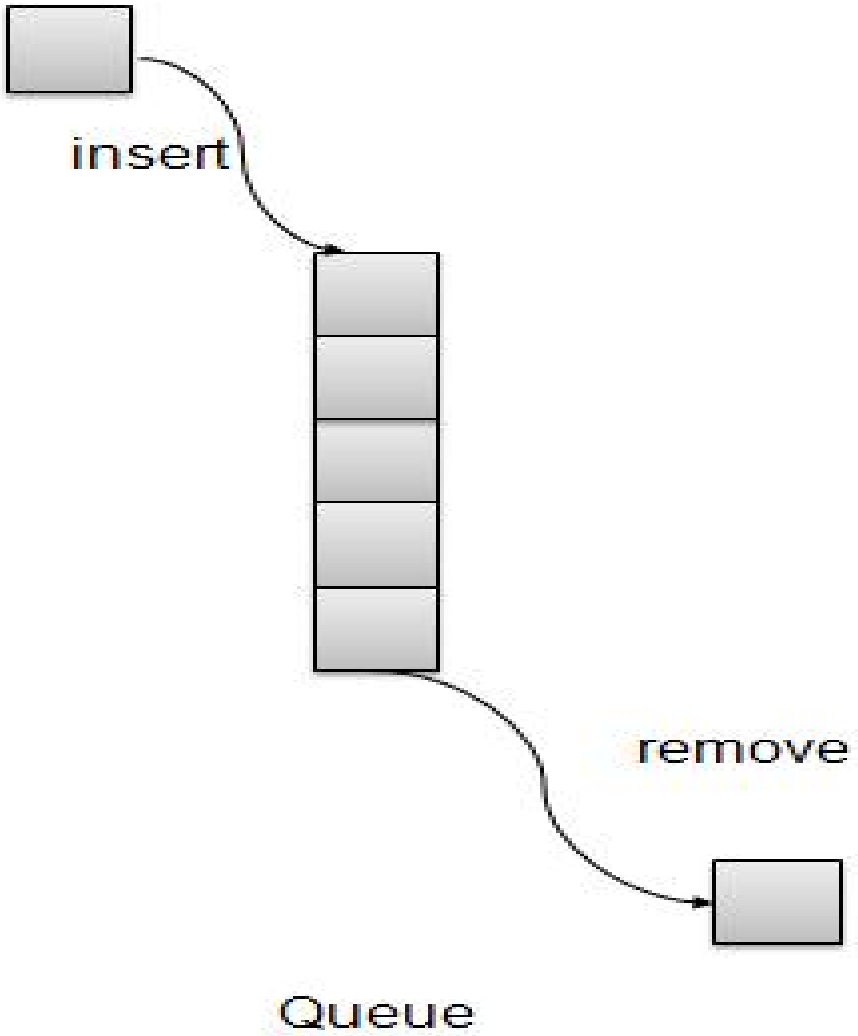
Priority Queue

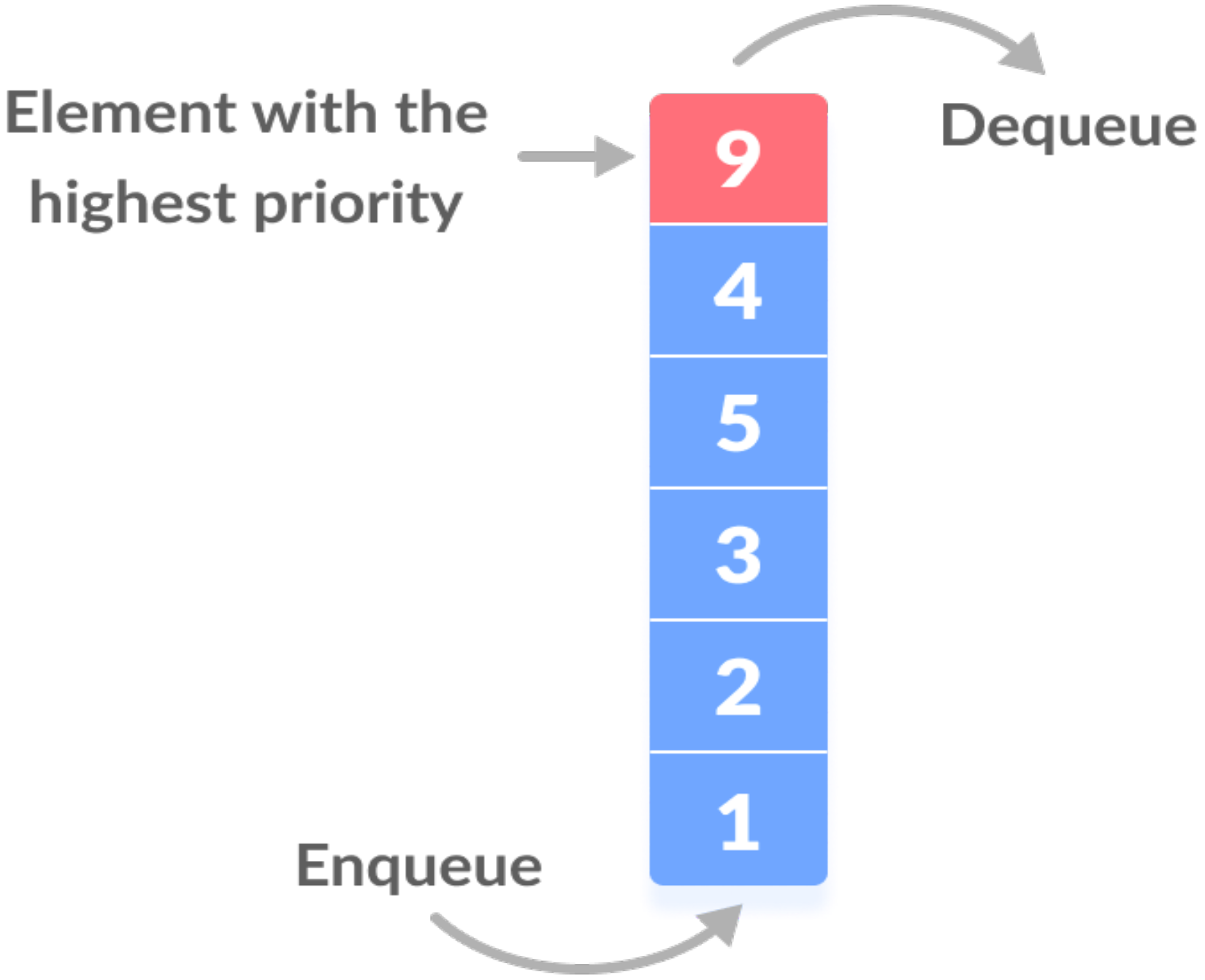
Priority Queue is more specialized data structure than Queue. Like ordinary queue, priority queue has same method but with a major difference. In Priority queue items are ordered by key value so that item with the lowest value of key is at front and item with the highest value of key is at rear or vice versa. So we're assigned priority to item based on its key value. Lower the value, higher the priority. Following are the principal methods of a Priority Queue.

Basic Operations

- insert / enqueue − add an item to the rear of the queue.
- remove / dequeue − remove an item from the front of the queue.

Priority Queue Representation:




Element with the highest priority
9
4
5
3
2
1
Dequeue
Enqueue

Applications of Queue

Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios:

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.

Round Robin scheduling using Queue

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

1. It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
2. One of the most commonly used technique in CPU scheduling as a core.
3. It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
4. The disadvantage of it is more overhead of context switching.

Round Robin Example:

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.

P1	P2	P3	P1	P2	P3	P1	P2	P3	P2

0 10

P1 waiting time : 4

P2 waiting time: 6

P3 waiting time: 6

The average waiting time(AWT) : (4+6+6)/3=5.33

Thank You