DATA STRUCTURES AND ALGORITHMS

BY

A.ROGHYA PARVEEN

ASSISTANT PROFESSOR IN DEPARTMENT OF COMPUTER SCIENCE & IT

JAMAL MOHAMED COLLEGE (AUTONOMOUS),TRICHY.

QUEUE

QUEUE

- Overview of Queue
- Insertion and deletion in a Linear queue
- Types of Queue
  - Circular Queue
  - Dequeue
  - Priority Queue

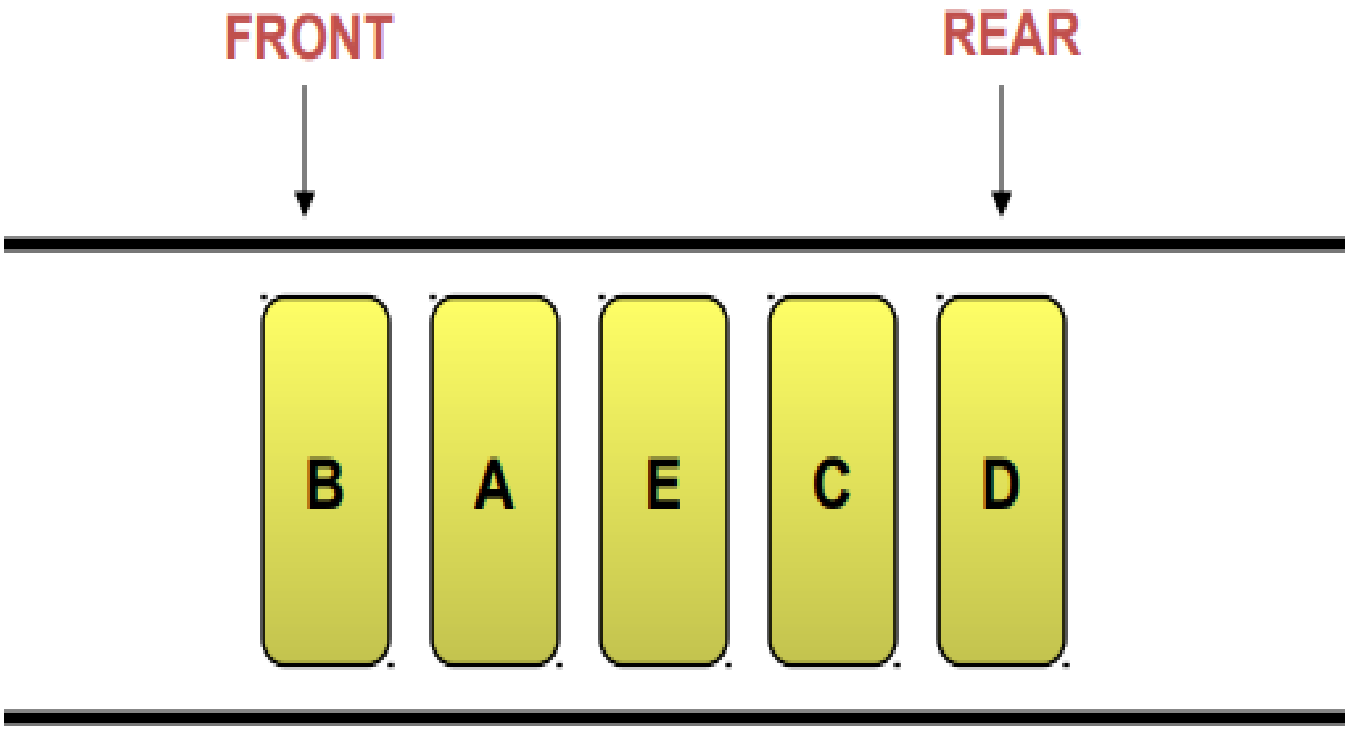
QUEUE -Example

QUEUE-DEFINITION

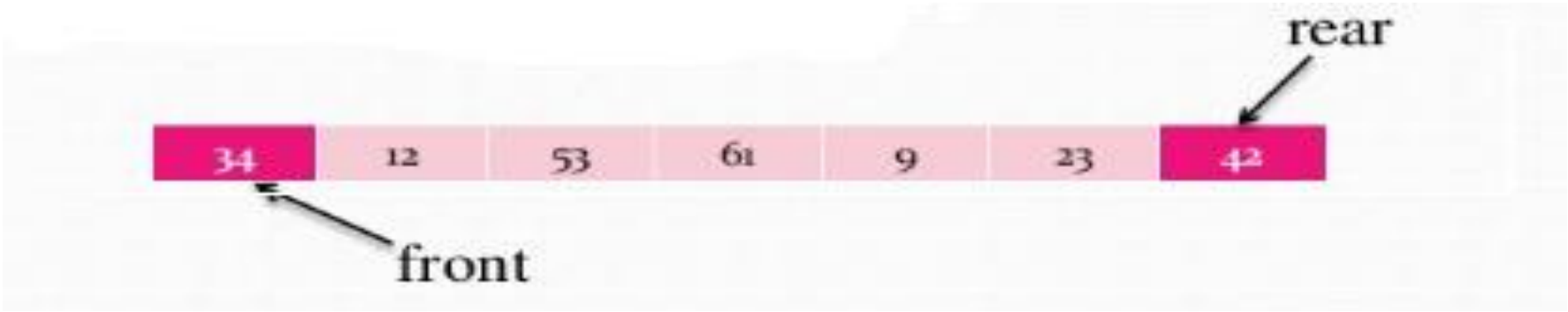
A queue is a linear list of elements in which deletion of an element can take place only at one end called the front and insertion can take place on the other end called as the rear.

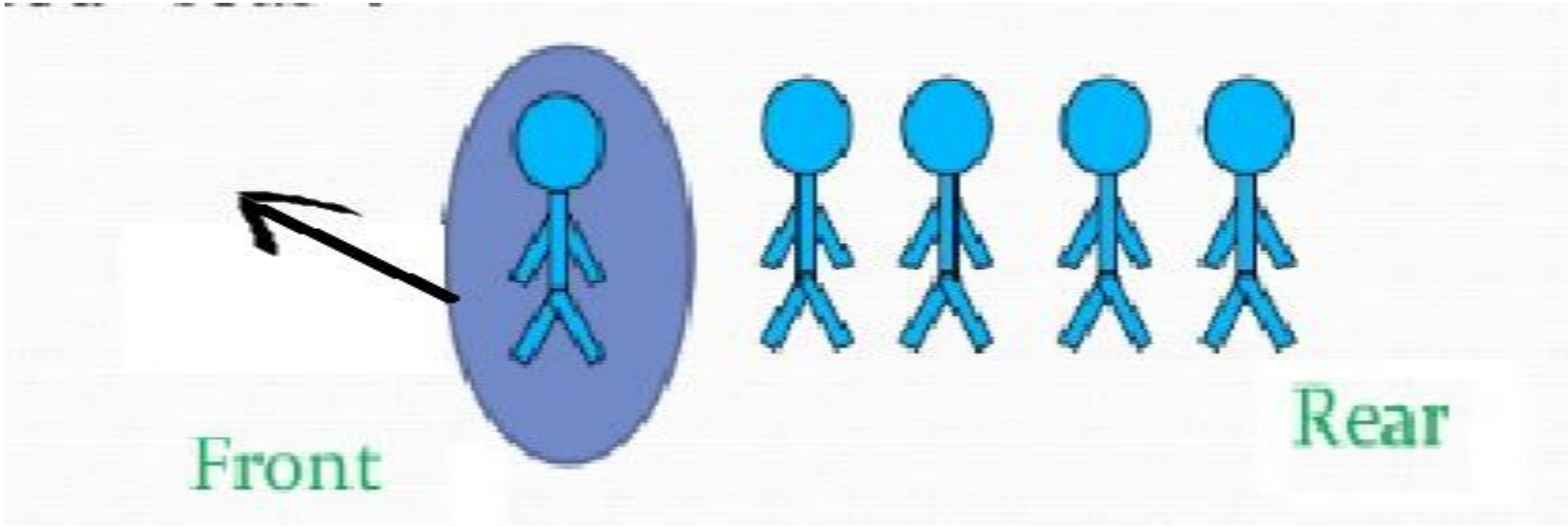
A queue is also called a FIRST IN FIRST OUT (FIFO) list.

QUEUE



QUEUE

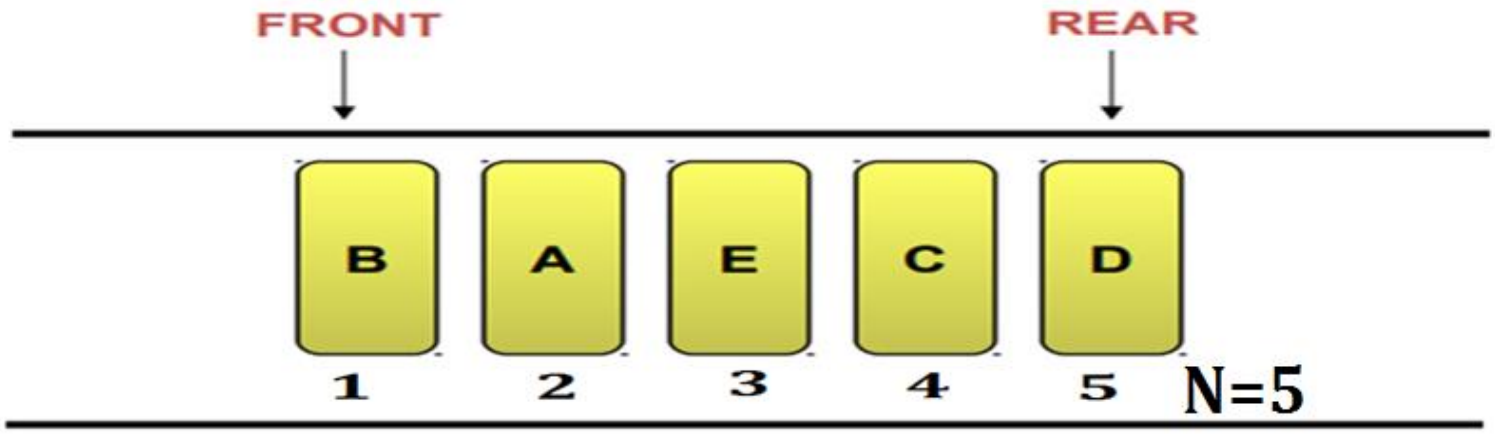




Components of Queue

- **Front** is a variable which refers to first position in queue.
- **Rear** is a variable which refers to last position in queue.
- **Element** is component which has data.
- **MaxQueue** is variable that describes maximum number of elements in a queue.

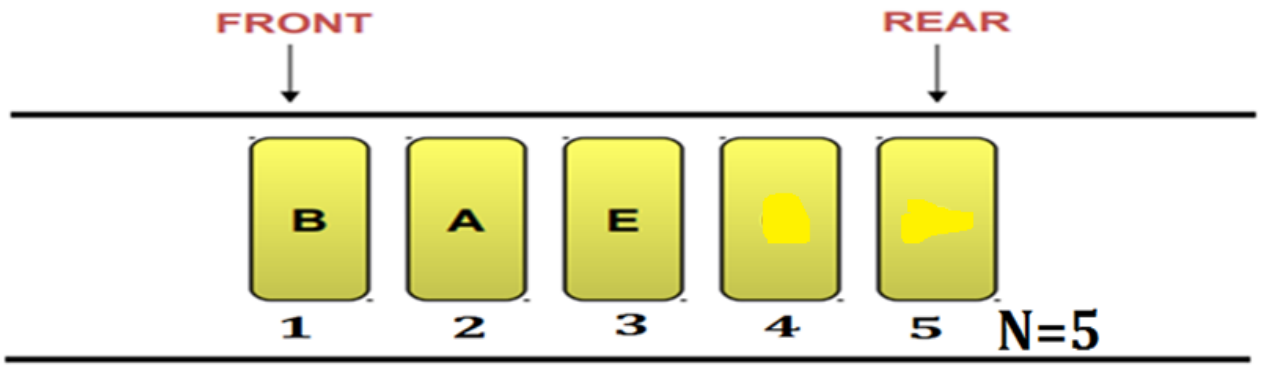
COMPONENTS OF QUEUE



FRONT = 1 **REAR = 5**

MAXQUEUE N = 5

COMPONENTS OF QUEUE



FRONT = 1 **REAR = 3**

MAXQUEUE N = 5

Operations on a queue

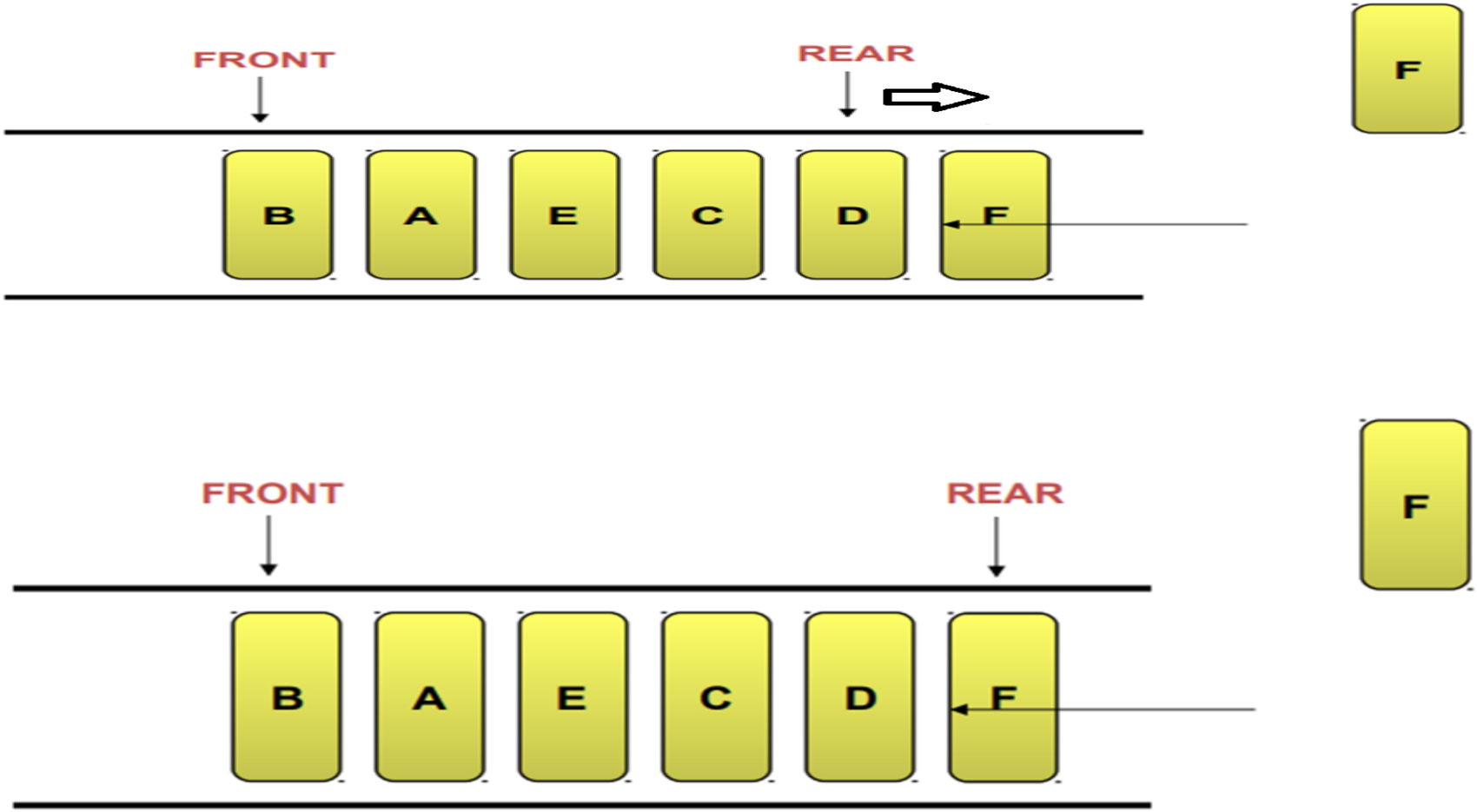
Various operations implemented on a queue are:

- ✓Insert
- ✓Delete

Insertion

- It refers to the addition of an item in the queue.
- Suppose you want to add an item F in the following queue.
- Since the items are inserted at the rear end, therefore, F is inserted after D.
- Now F becomes the rear end.

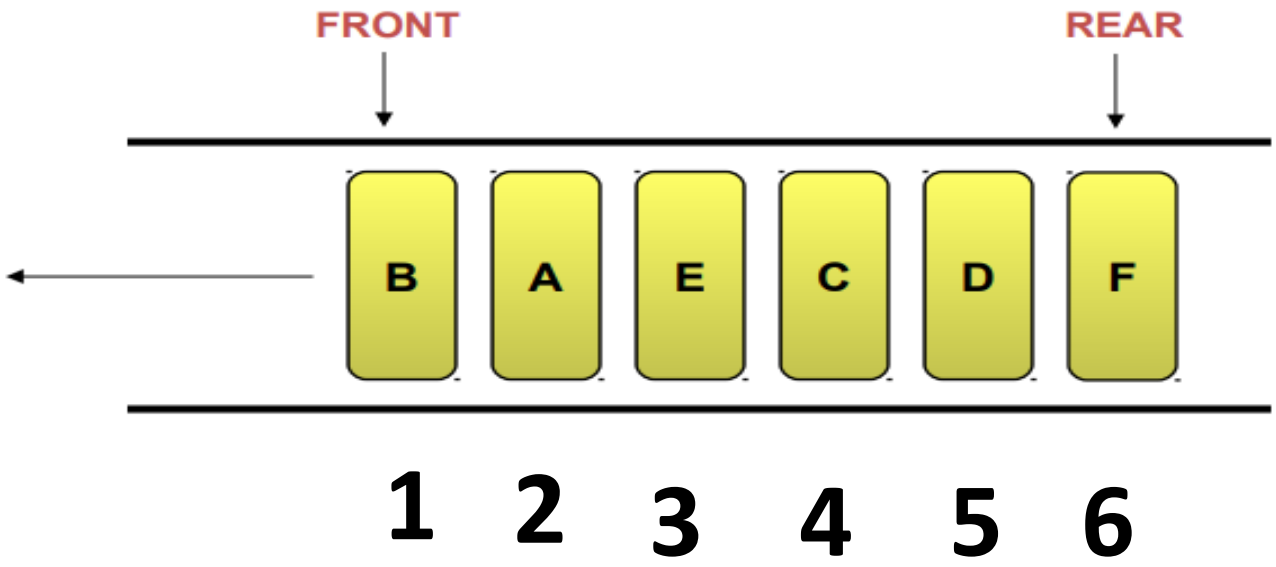
Insertion



Deletion

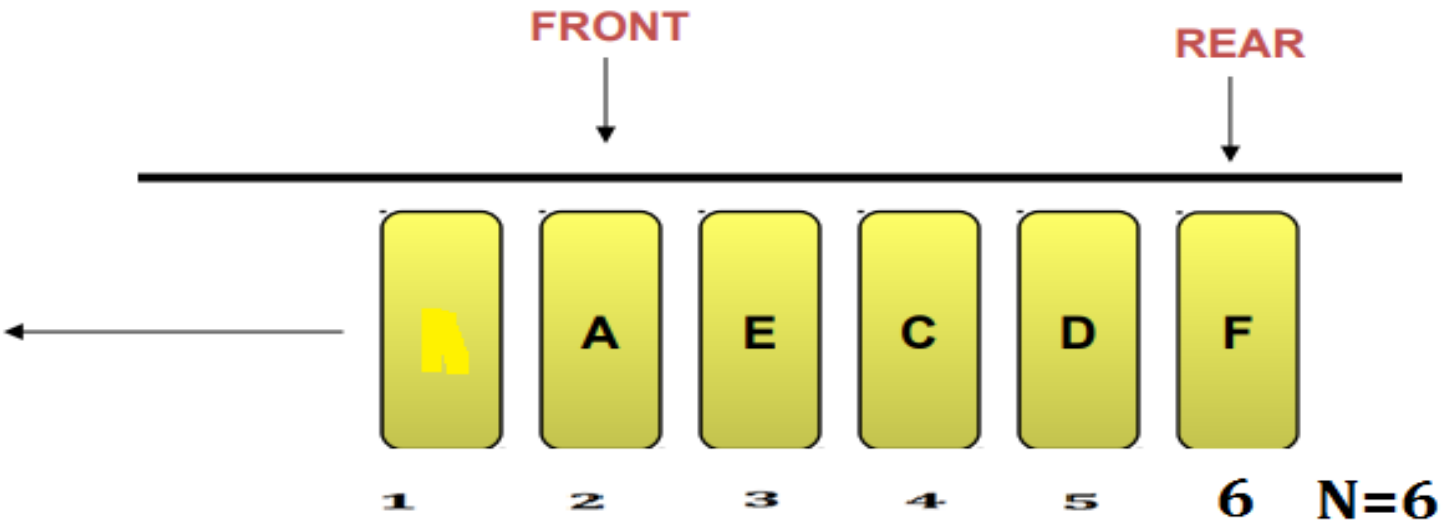
- It refers to the deletion of an item from the queue.
- Since the items are deleted from the front end, therefore, item B is removed from the queue.
- Now A becomes the front end of the queue

Deletion



FRONT = 1 **REAR = 6**

Deletion



FRONT = 2 **REAR = 6**

MAXQUEUE N = 6

Algorithm to insert data in a Queue

FRONT = -1, REAR = -1, TO REPRESENT QUEUE IS EMPTY

Algorithm INSERT(QUEUE[N],FRONT,REAR,ITEM)

{

//QUEUE is an array of size N ,ITEM is element to be inserted.

1. if (REAR == N-1) Print “OVERFLOW”

else

if (FRONT == -1)

1.1.1 FRONT = 0

1.2 REAR = REAR+1

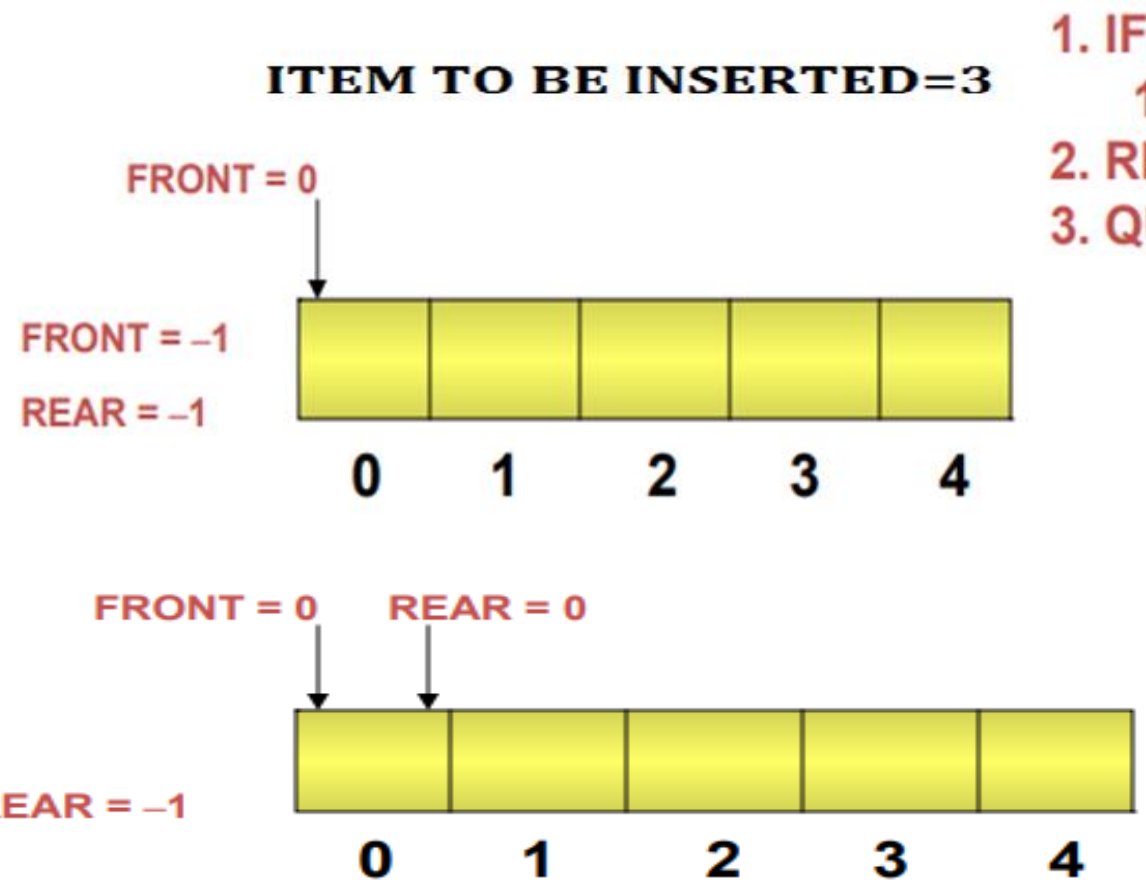
2. QUEUE[REAR] = ITEM

3. EXIT

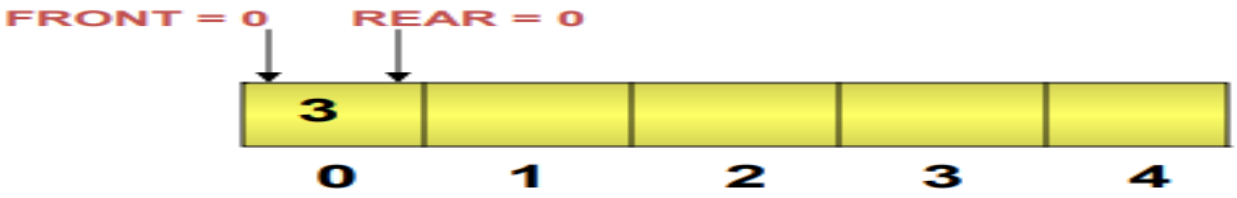
}

Insertion

1. IF(FRONT==-1)
   1.1 FRONT = 0
2. REAR = REAR+1
3. QUEUE[REAR] = ITEM



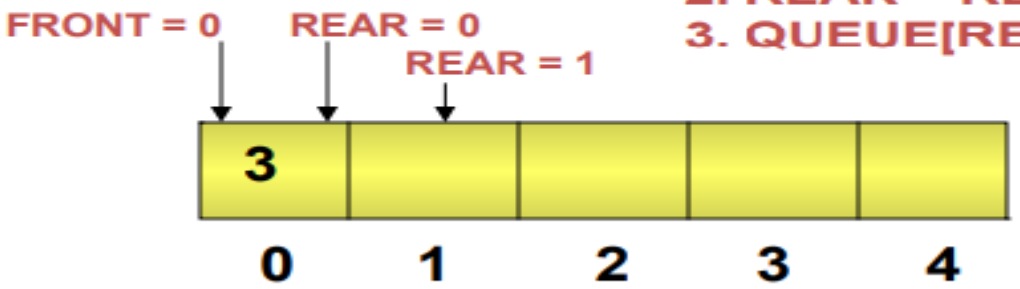
Insertion

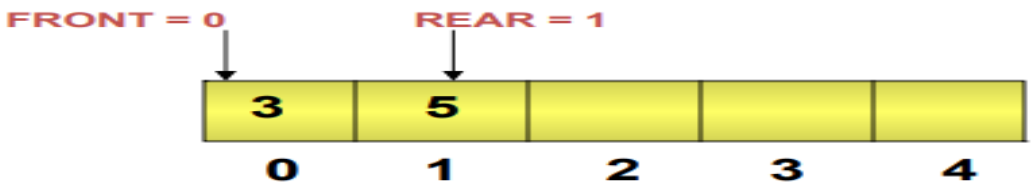


Insertion complete

ITEM=5

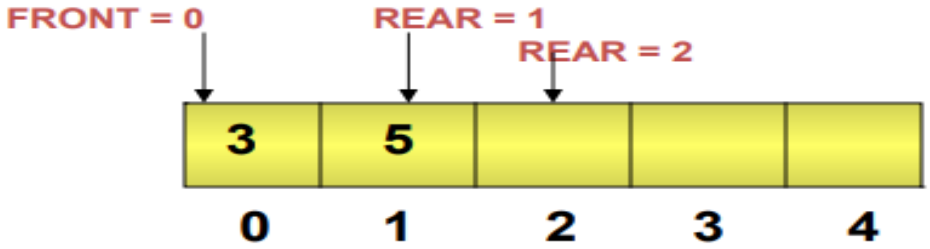
1. IF(FRONT==-1)
   1.1 FRONT = 0
2. REAR = REAR+1
3. QUEUE[REAR] = ITEM

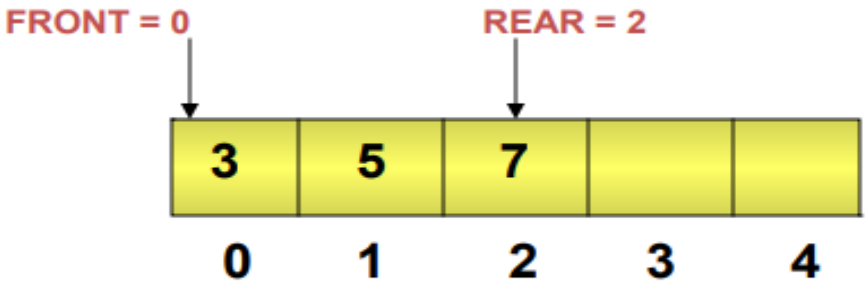




Insertion complete

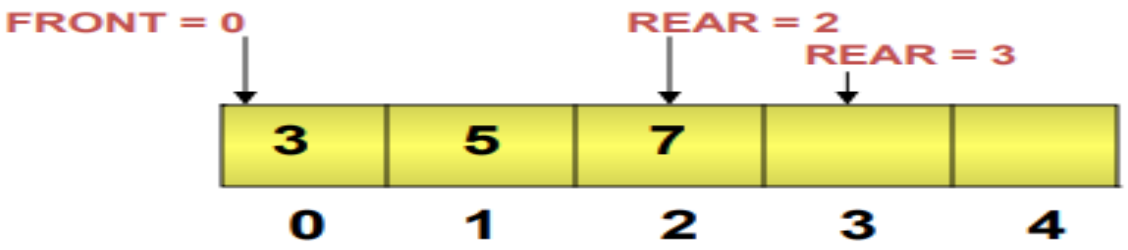
NEXT ITEM TO BE INSERTED=7

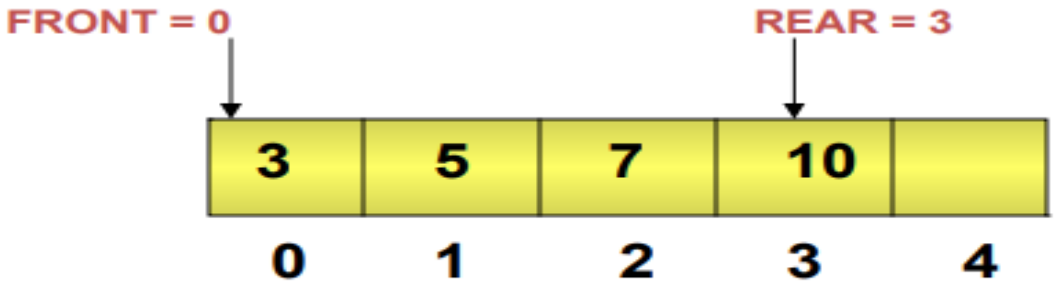




Insertion complete

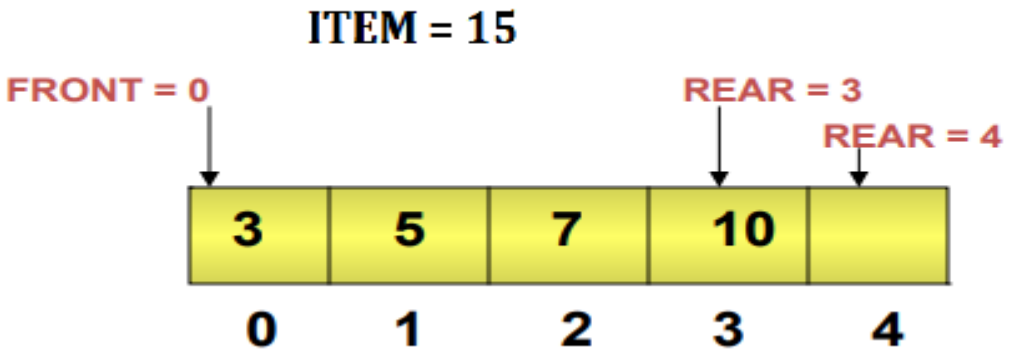
ITEM=10

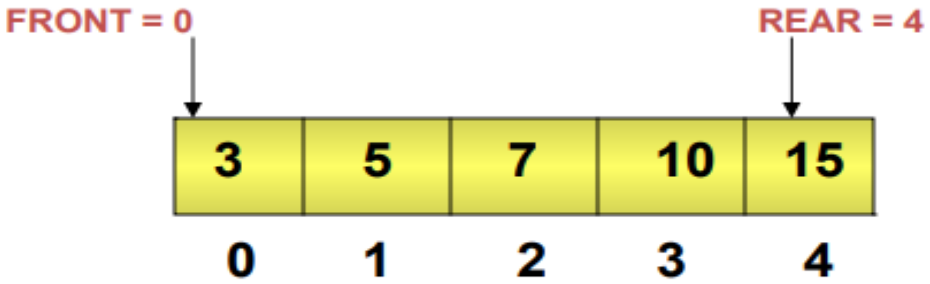




Insertion complete

Insertion





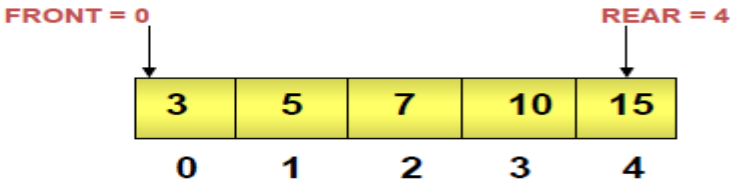
Insertion complete

Algorithm to delete an element from a queue

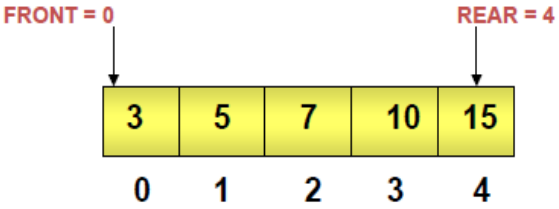
```
Algorithm DELETE(QUEUE[N],ITEM,FRONT,REAR)
{
1. if (( FRONT == – 1 ) || (FRONT == REAR+1))
     Print “QUEUE EMPTY”
        else
2.  ITEM = QUEUE[FRONT]
       FRONT = FRONT +1
}
```

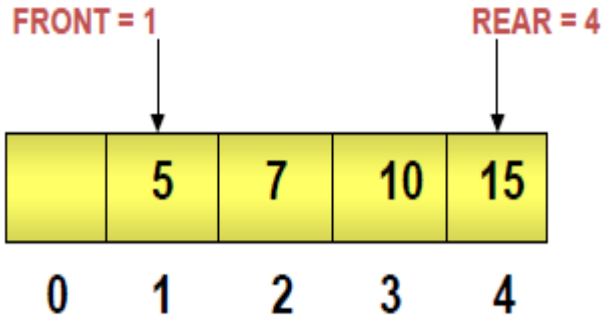
- Let us see how requests are deleted from the queue once they get processed.

```
1.If( FRONT == – 1 )
    1.1 print “QUEUE EMPTY”
2. else
    2.1  ITEM = QUEUE[FRONT]
    2.2  FRONT = FRONT +1
```

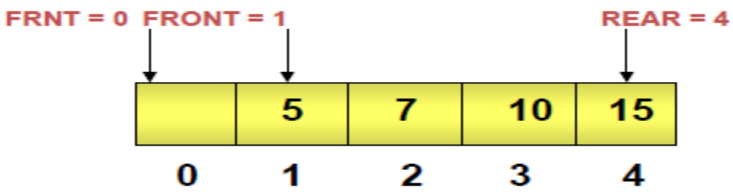


One request processed

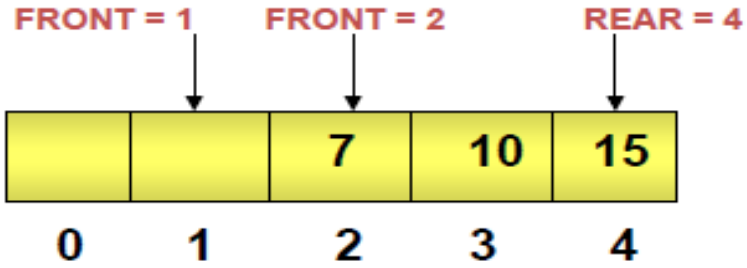




```
1.If( FRONT == – 1 )
    1.1 print “QUEUE EMPTY”
2. else
    2.1  ITEM = QUEUE[FRONT]
    2.2  FRONT = FRONT +1
```



Delete operation complete



Delete operation complete

TYPES OF QUEUES

1.CIRCULAR QUEUE

2. DEQUEUE

3.PRIORITY QUEUE

Drawback of Linear Queue

- Once the queue is full, even though few elements from the front are deleted and some occupied space is relieved, it is not possible to add anymore new elements, as the rear has already reached the Queue's rear most position.

Circular Queue

- This queue is not linear but circular.
- Its structure can be like the following figure:
- In circular queue, once the Queue is full the "First" element of the Queue becomes the "Rear" most element, if and only if the "Front" has moved forward. otherwise it will again be a "Queue overflow" state.

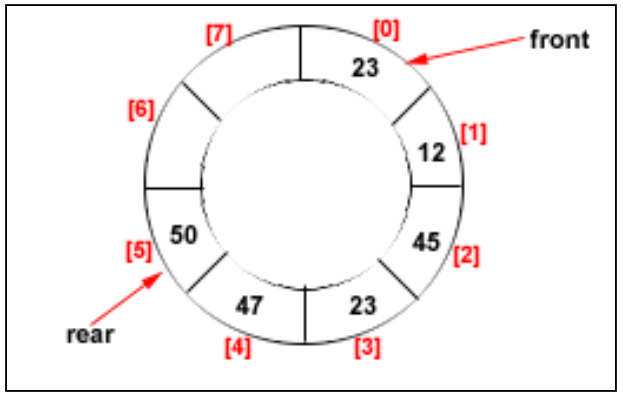


Figure: Circular Queue having Rear = 5 and Front = 0

Algorithms for Insert and Delete Operations in Circular Queue

For Insert Operation

Insert-Circular-Q(CQueue, Rear, Front, N, Item)

Here, CQueue is a circular queue where to store data. Here N is the maximum size of CQueue and finally, Item is the new item to be added. Initailly Rear = 0 and Front = 0.

1. If Front = 0 and Rear = 0 then Set Front := 1 and go to step 4.
2. If Front =1 and Rear = N or Front = Rear + 1
   then Print: “Circular Queue Overflow” and Return.
3. If Rear = N then Set Rear := 1 and go to step 5.
4. Set Rear := Rear + 1
5. Set CQueue [Rear] := Item.
6. Return

For Delete Operation

Delete-Circular-Q(CQueue, Front, Rear, Item)

Here, CQueue is the place where data are stored. Front element is assigned to Item. Initially, Front = 1.

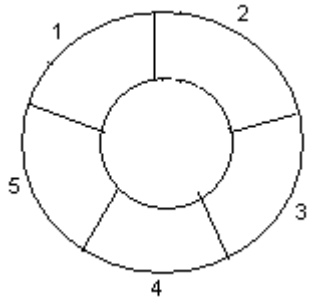
1. If Front = 0 then

   Print: “Circular Queue Underflow” and Return. /*..Delete without Insertion

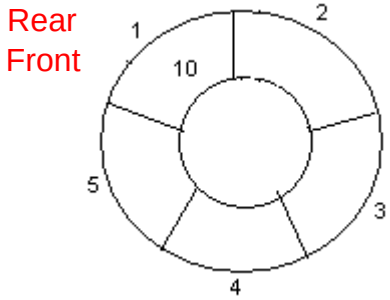
2. Set Item := CQueue [Front] GO TO STEP 5
3. If Front = N then Set Front = 1 and Return.
4. If Front = Rear then Set Front = 0 and Rear = 0 and Return.
5. Set Front := Front + 1
6. Return.

Example: Consider the following circular queue with N = 5.

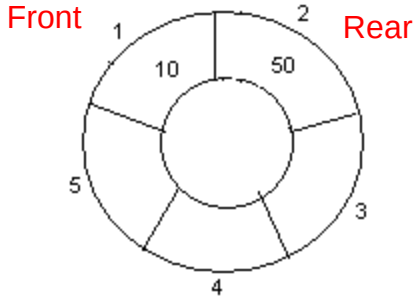
1. Initially, Rear = 0, Front = 0.



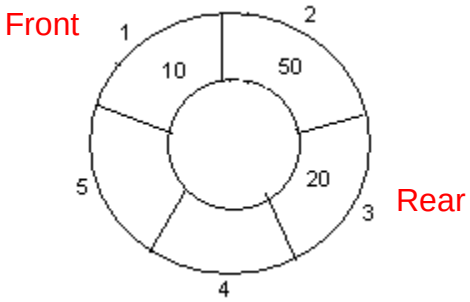
2. Insert 10, Rear = 1, Front = 1.



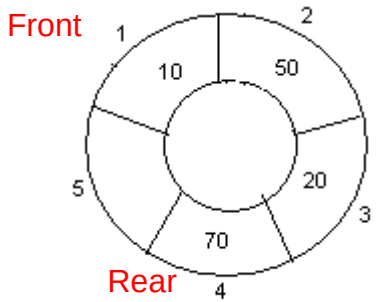
3. Insert 50, Rear = 2, Front = 1.



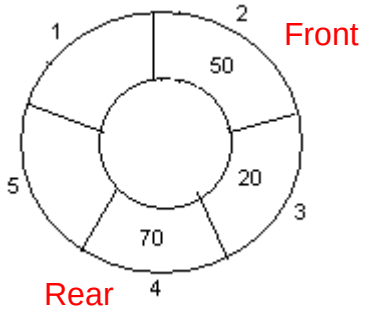
4. Insert 20, Rear = 3, Front = 1.



5. Insert 70, Rear = 4, Front = 1.



6. Delete front, Rear = 4, Front = 2.



DEQUEUE

This general data class has some possible sub-types:

- **An input-restricted deque is one where deletion can be** made from both ends, but insertion can only be made at one end.
- **An output-restricted deque is one where insertion can** be made at both ends, but deletion can be made from one end only.

PRIORITY QUEUE

- Is a collection of elements where the elements are stored according to their priority levels.
- Following rules are applied to maintain a priority queue:-

– The elements with a higher priority is processed before any element of lower priority.

– If there are elements with a same priority ,then the element added first in the queue would get Processed.

REFERENCES

Books

1. Fundamentals of Data Structure – Ellis Horowiz, SartajSahni, Galgotia

Publications, 2008.

2. Data Structures – Seymour Lipschutz, Tata Mcgraw Hill, Schaum's Outline

Series, 2014.

Websites

1.http://ds.nathanielgmartin.com/Queues.pdf

2. https://www.mlsu.ac.in/econtents/326_05Queues.pdf