Unit 5: Queue

5.1 Introduction
5.2 Array Implementation of Queue
5.3 Linked List Implementation of Queue
5.4 Circular Queue
5.5 Priority Queue.

5.1 Introduction

Definition

A queue is an ordered collection of items from which items may be deleted at one end (FRONT) and into which item may be inserted at the other end (REAR). The first element inserted into the queue is the first element to be removed. So, it is also called FIFO (First In First Out)

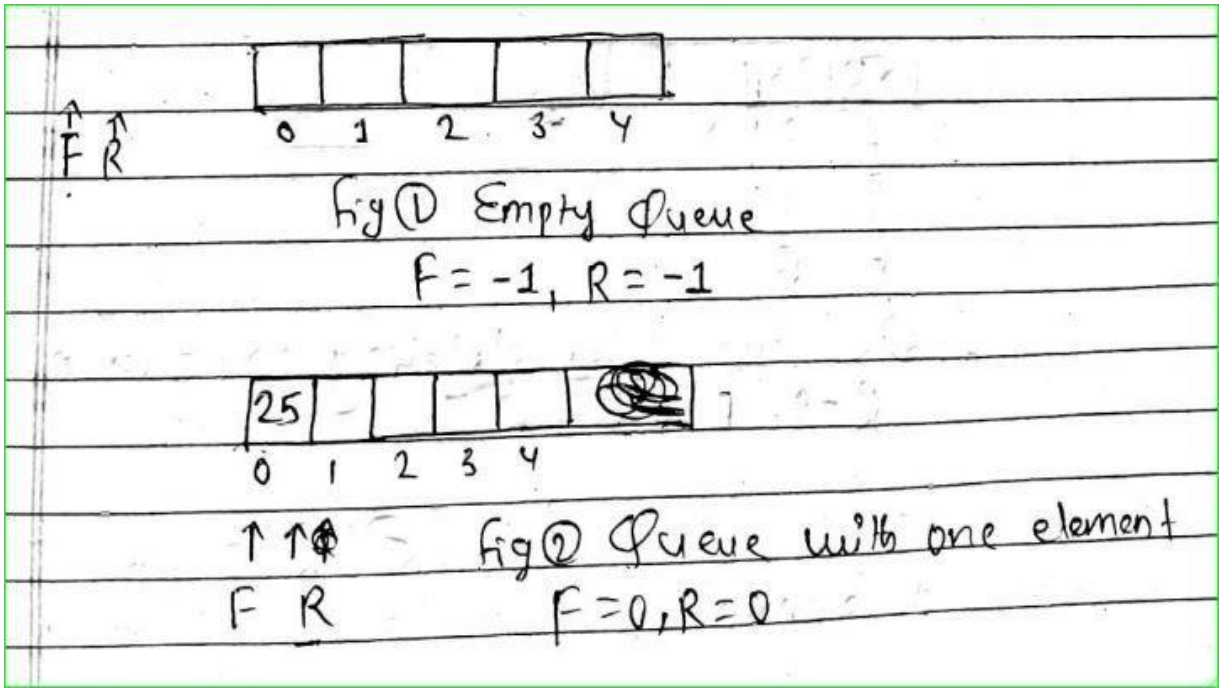
Queue as an ADT

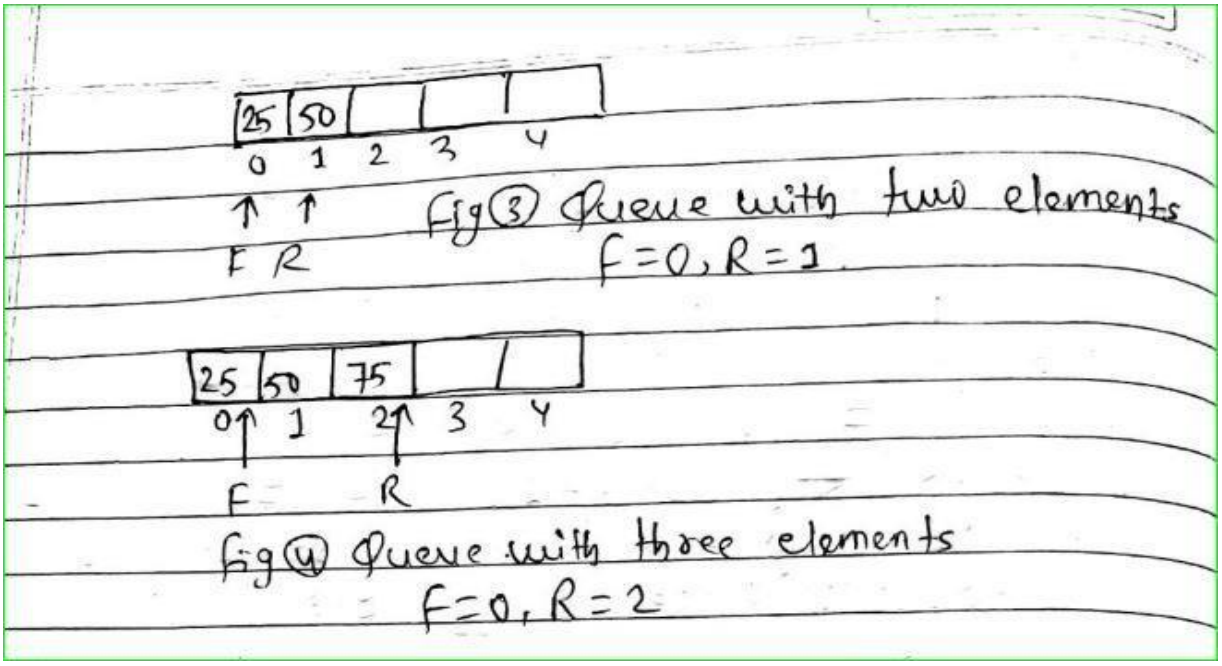
A queue 'Q' of type 'T' is a finite sequence of elements with the operation

1. Make Empty Queue(Q) ,
   Make queue 'Q' as an empty queue.
2. IsEmpty(Q)
   Check whether the queue 'Q' is empty or not. Return 'True' if 'Q' is empty. Return 'False' otherwise.
3. IsFull(Q)
   Check whether the queue 'Q' is full or not Return 'True' if
   'Q' is full Return 'False' otherwise.
4. Enqueue(Q,x) Insert the item 'x' at the REAR end of the queue 'Q' if and only if the queue 'Q' is not full.
5. Dequeue(Q)
   Delete the item from the FRONT end of queue 'Q' if and only if the queue 'Q' is not empty.
6. Traverse(Q)
   Display the content of the entire queue.

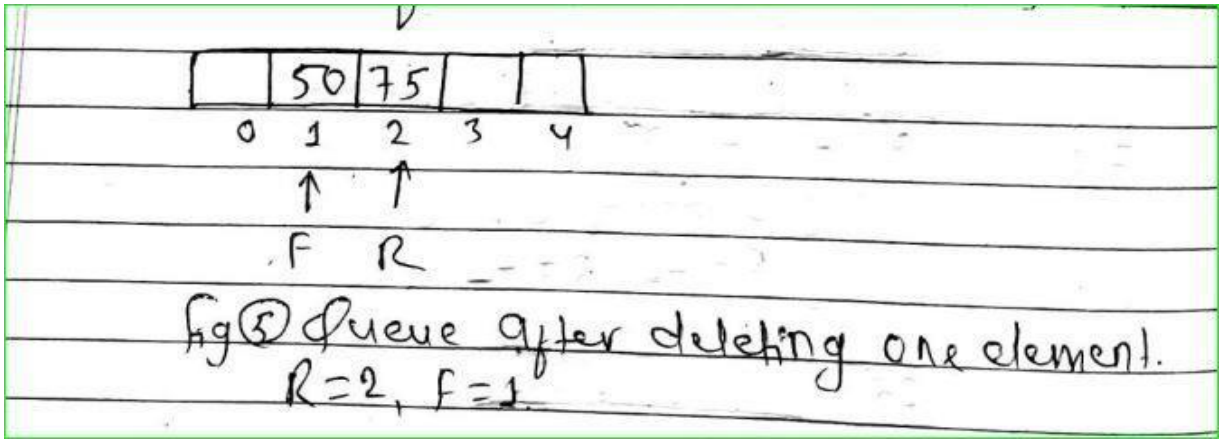
Linear Queue:

Linear queue is the queue where the value of FRONT and REAR can only be incremented.





From the above figures, we can say that R= -1 and F= -1 when queue is empty. R=0 and F=0 when there is one element in a queue. After that only 'R' is incremented by 1 with insertion of each element in the queue. 'F' will only increase for one time during the insertion of first element in the queue.



In the above fig 5, R=2 and F=1. When one element is deleted from the queue, the 'F' will get incremented by 1.

5.2 **Array Implementation of Queue (Static Implementation)**

Since a queue usually holds a bunch of items with the same type, we could implement by means of one way list or linear array. Here the queue in a linear array is maintained with two pointers called FRONT, containing the location of the front element of the queue and REAR, to hold the location which is at rear. Insertion and deletion of element is not handled as the normal array where we shift the elements forward or backward. Here, whenever the element is deleted from the queue the value of the FRONT is increased by 1 this can be implemented by FRONT: =FRONT+1. Similarly whenever an element is added to the queue, the value of REAR is increased by 1 by assigning REAR: =REAR +1 as referred in the figure below.

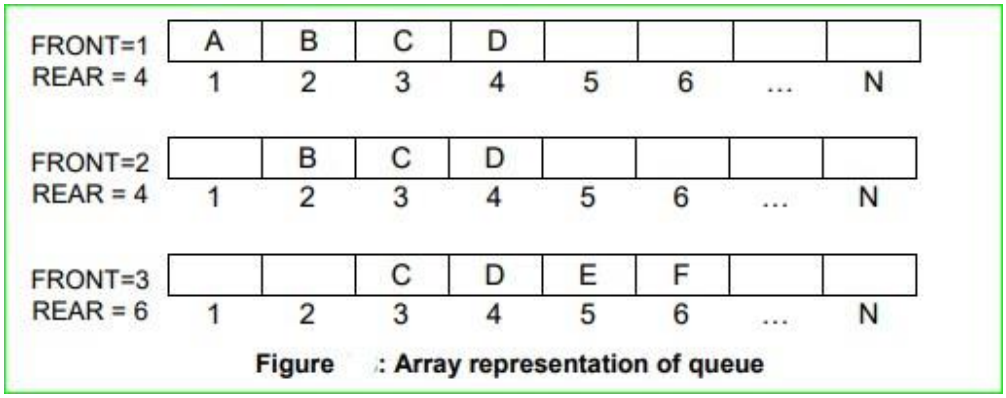


Figure : Array representation of queue

After N insertions, the rear element of the queue will occupy QUEUE[N] and this occurs even the array is not full. If we insert a new element when REAR =N, one way to handle this is to simply move the entire queue to the beginning of the array and change FRONT AND REAR accordingly an then insert an item. This procedure may be expensive, an alternate is consider the queue is circular that is QUEUE [1] comes after QUEUE [N] in the array. With this assumption we can insert ITEM into the queue by assigning ITEM to QUEUE[1], instead of increasing the REAR to N+1 we can reset REAR=1 then the assignment will be QUEUE[REAR]:=ITEM. Similarly if FRONT=N and an element of queue is deleted we can reset FRONT=1 instead of increasing FRONT to N+1. The following are the operations to insert (enqueue) an item into linear queue and delete (dequeue) an item from a linear queue.

1. Enqueue

Enqueue is used to insert an item at the REAR end of the queue. When the queue is full, the item cannot be inserted in the queue.

```
QINSERT (QUEUE,N,FRONT,REAR,ITEM)
1. If FRONT=1 and REAR =N, or FRONT=REAR+1 then
        Write OVERFLOW, and Return
2. [Find new value of REAR]
   If FRONT :=NULL, then: [Queue is empty initially]
        Set FRONT :=1 and REAR:=1
   Else if REAR=N then:
   Set REAR :=1.
   Else
   Set REAR:=REAR+1
3. Set QUEUE[REAR]:=ITEM
4. Return
```

2. Dequeue:

Dequeue is used to delete an item from the FRONT end of the queue. When the queue is empty, dequeue operation is not possible.

```
QDELETE (QUEUE, N, FRONT, REAR, ITEM)
    1. If FRONT: =NULL, then write : UNDERFLOW and Return.
    2. Set ITEM:=QUEUE[FRONT].
    3. If FRONT = REAR, then:          // only one element.
                Set FRONT: =NULL and REAR: =NULL.
       Else if FRONT := N then:
                Set FRONT: =1.
       Else:
                Set FRONT: = FRONT +1.
    4. Return.
```

5.3 Linked List Implementation of Queue (Dynamic Implementation)

Queues are very much like linked list except the ability to manipulate items on the lists, a linked queue is a queue implemented as a linked list with two pointer variables FRONT and REAR pointing to the nodes which is in the FRONT and REAR of the queue as referred in figure below.

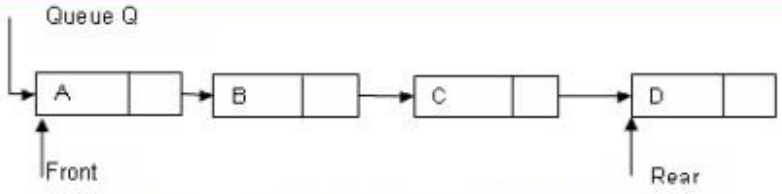


Figure : Linked representation of queue

In the linked list representation of queue, following are the operations to insert (enqueue) an item into linear queue and delete (dequeue) an item from a linear queue.

1. Enqueue:

The array representation of queue had a disadvantage of limited queue capacity that is, every time of insertion we need to check for OVERFLOW status and then insert a new element. But in linked queue representation while inserting an element a new node will be availed from the AVAIL list which holds the ITEM, will be inserted as the last node of the linked list representing queue. The rear point will be updated in order to point the node which is recently entered. Figure below is illustrating how to insert a new node into the queue, new node is availed from the AVAIL list and ITEM D is assigned with the new node. Before insertion the value of the REAR was REAR = 3 and having NULL pointer. As we know insertion can happen only with REAR, the NEW node is linked with the LINK [REAR] and the Value of the REAR is incremented to 4.

```
LINKQ_INSERT (INFO,LINK,FRONT,REAR,AVAIL,ITEM)
    1. If AVAIL = NULL, then write OVERFLOW and Exit
    2. Set NEW:=AVAIL and AVAIL:=LINK[AVAIL]
       [Remove first node from AVAIL list]
    3. Set INFO[NEW] :=ITEM and LINK[NEW]=NULL
       [copies ITEM to new node]
    4. If (FRONT=NULL) then FRONT=REAR=NEW
       [If queue is empty then ITEM is the first element in the
       queue Q]
       Else Set LINK[REAR] :=NEW and REAR = NEW
       [REAR points to the new node appended to the end of list]
    5. Exit.
```

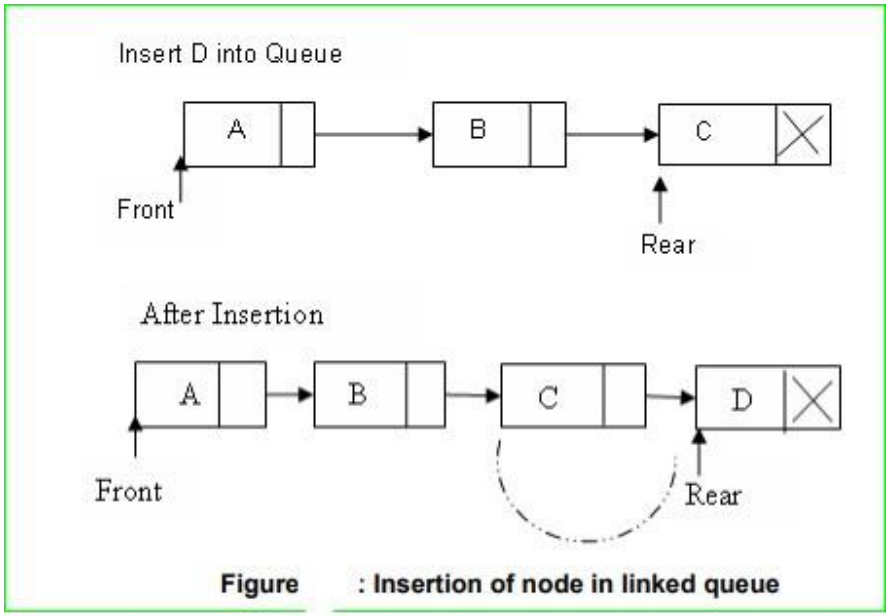


Figure : Insertion of node in linked queue

2. Dequeue:

In the linked queue, the deletion can happen only from the FRONT end. In case of deletion the first node of the list pointed to by FRONT is deleted and the FRONT pointer is updated to point to the next node in the list and the deleted node will be return to the AVAIL list. The figure below is showing the process of deletion, here the node which is in the FRONT carrying the value A is deleted and the FRONT pointer is updated to the next node carrying the value B.

```
LINKQ_DELETE (INFO, LINK, FRONT, REAR, AVAIL, ITEM)
1. If (FRONT=NULL) then Write: UNDERFLOW and Exit
2. Set TEMP=FRONT [if queue is not empty]
3. ITEM=INFO(TEMP)
4. FRONT=LINK(TEMP) [Reset FRONT to next element]
5. LINK(TEMP)=AVAIL and AVAIL=TEMP
   [Return deleted node to AVAIL list]
6. Exit.
```

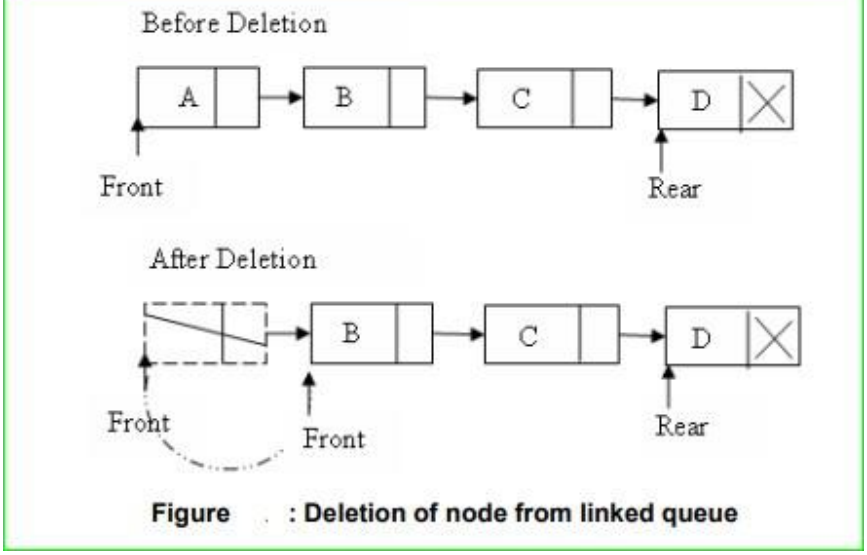


Figure : Deletion of node from linked queue

Problems with Linear queue implementation:

1. Both REAR and FRONT are increased but never decreased.
2. As items are removed from the queue, the storage space at the beginning of the queue is discarded and never used again.

3. Wastage of the space is the main problem with linear queue which is illustrated by the following example.

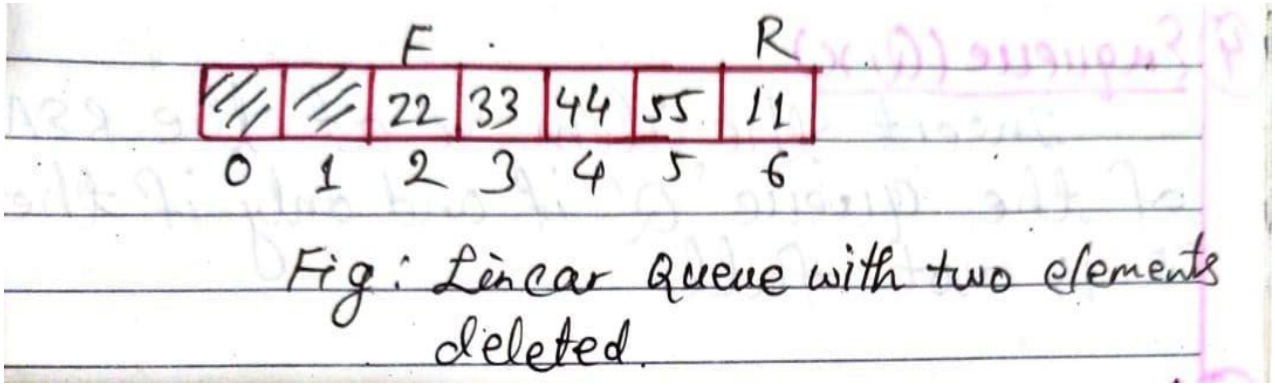


Fig: Linear Queue with two elements deleted.

In the above figure, queue is considered full, even though the space at beginning is vacant

5.4 Circular Queue A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full. A circular queue overcomes the problem of unutilized space in linear queues implemented as arrays. A circular queue also has a FRONT and REAR to keep the track of the elements to be deleted and inserted inorder to maintain the unique characteristics of the queue. In circular queue, the following assumptions are made.

a. FRONT will always to pointing to the first element (as in the linear queue).
b. If FRONT = REAR, then the queue will be empty.
c. Each time a new element is inserted into the queue, the REAR is incremented by 1 i.e. REAR = REAR + 1.
d. Each time an element is deleted from the queue, the value of FRONT is incremented by 1 i.e FRONT = FRONT + 1

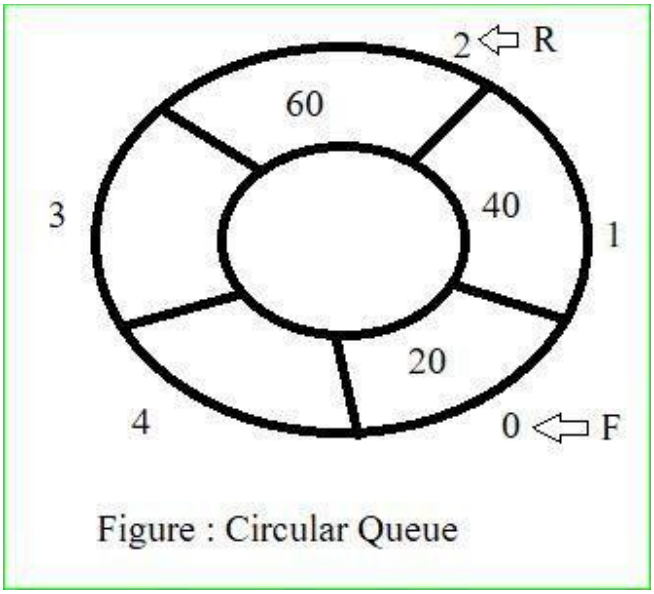
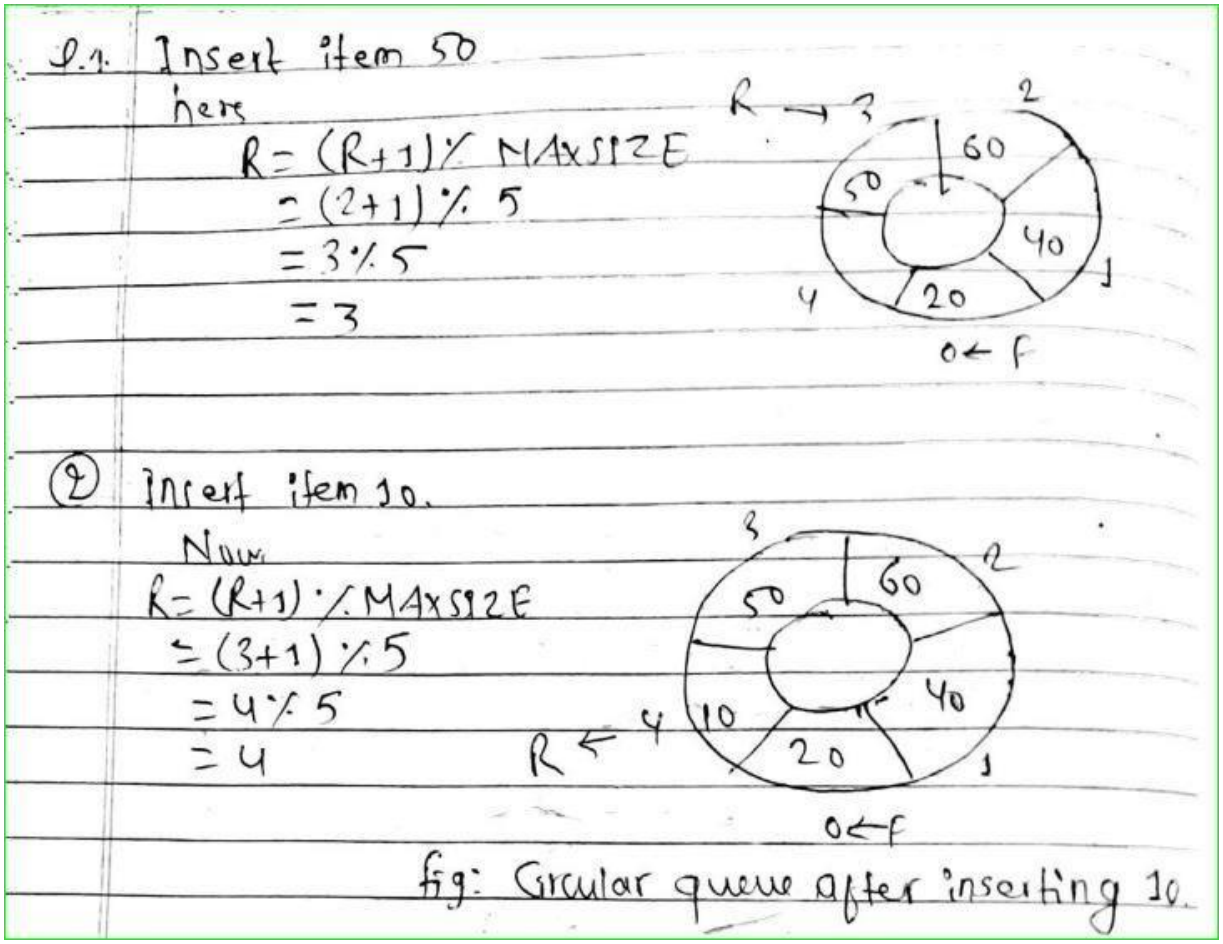
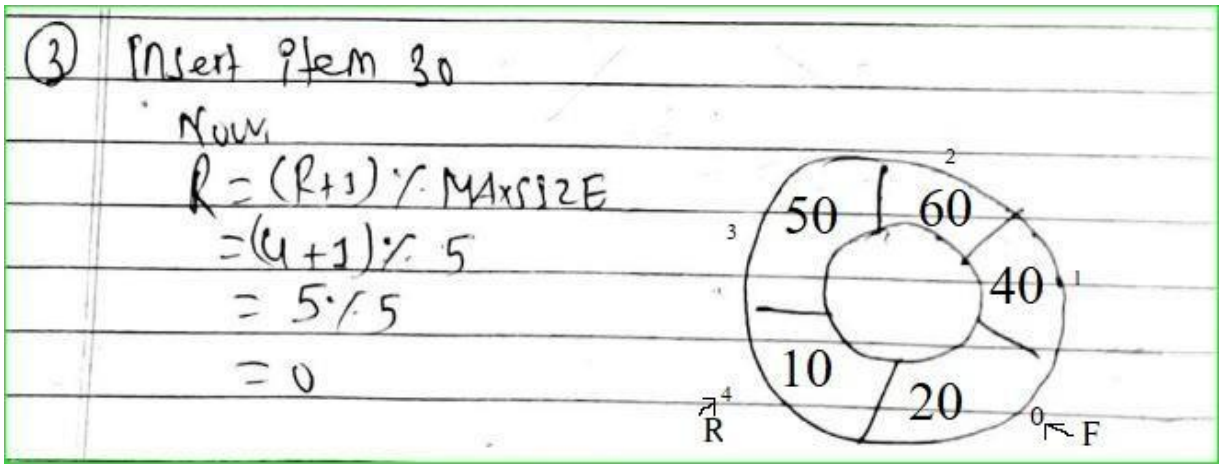


Figure : Circular Queue

In the above figure, a circular queue 'CQ' with a capacity of 5 elements is shown. Initially there are 3 elements with F=0 and R=2.





Here,
F=0 already. So if R=0, then we can say circular queue is full and no more elements can be inserted further.

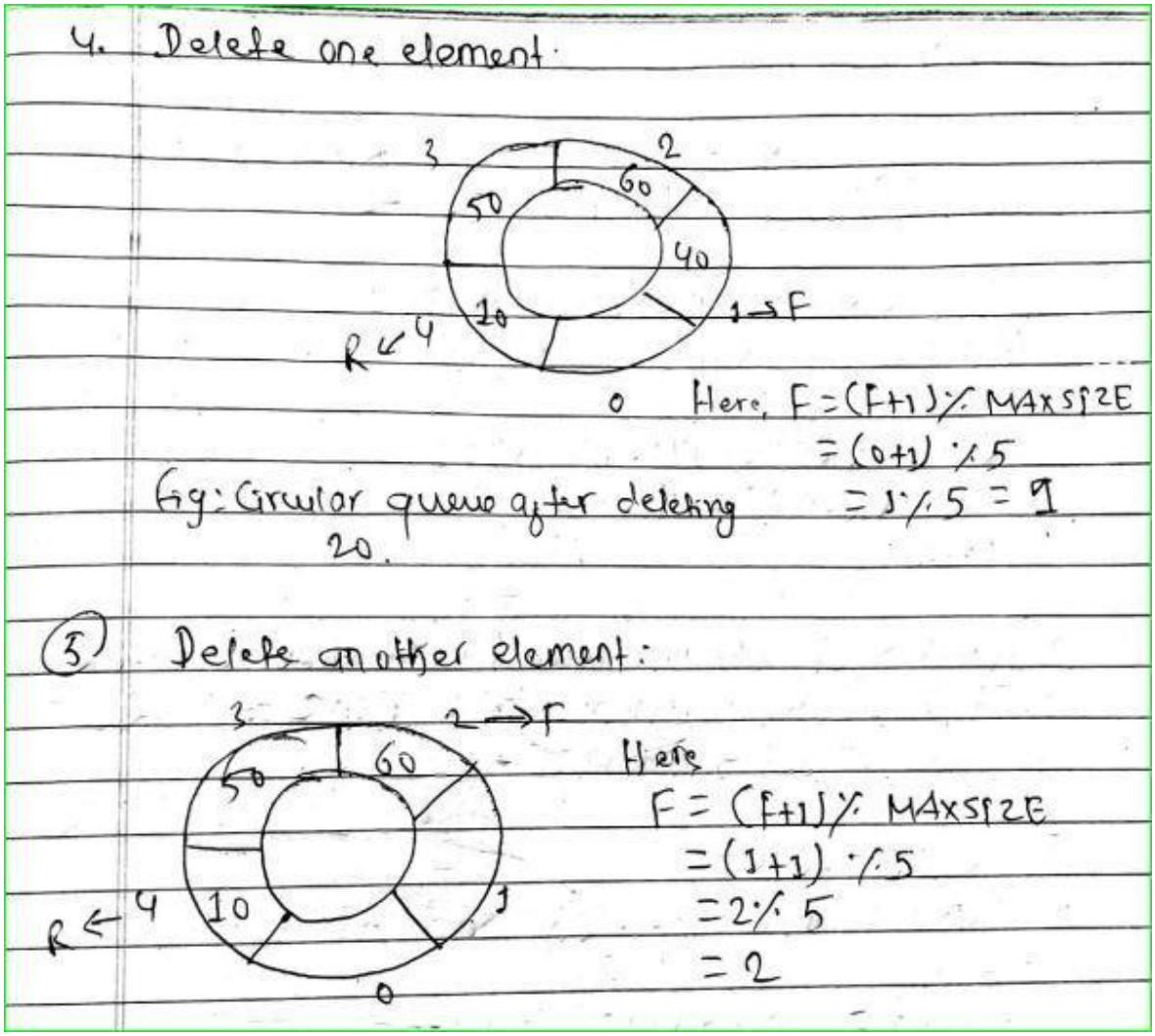


Figure: Circular queue after deleting 40

Figure: Circular queue after inserting 70

- Algorithm for enqueue operation in circular queue.

  This algorithm assumes that REAR and FRONT are initially set to MAXSIZE -1.

  1. If (FRONT = = (REAR + 1) % MAXSIZE)
     Print “Queue overflow” and Exit.
     Else
     REAR = (REAR+1) % MAXSIZE
  2. CQUEUE [REAR] = ITEM
  3. Exit

- Algorithm for dequeue operation in circular queue.

  This algorithm assumes that REAR and FRONT are initially set to MAXSIZE -1.

  1. If (REAR = = FRONT)
     Print “Queue underflow” and Exit. Else FRONT = (FRONT+1) % MAXSIZE
  2. ITEM = CQUEUE [FRONT]
  3. Return ITEM
  4. Exit

5.5 Priority Queue

A priority queue is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted and processed comes from the following rules.

a. An element of higher priority is processed before any element of lower priority
b. Two elements with the same priority are processed according to the order in which they were added to the queue.

- **Types of Priority Queue**
  - **a. Ascending Priority Queue:** collection of items into which items can be inserted arbitrarily and from which only the smallest item can be removed is called ascending priority queue.

  - **b. Descending Priority Queue:** collection of items into which items can be inserted arbitrarily and from which only the largest item can be removed is called descending priority queue.

- **Operations in Priority Queue**
  - **a. Insertion:** The insertion in priority queues is the same as in non-priority queues.

  - **b. Deletion:**
  - **c.** Deletion requires a search for the element of highest priority and deletes the element with highest priority. The following methods can be used for deletion/removal from a given priority queue.
  1. An empty indicator replaces deleted elements.
  2. After each deletion, elements can be moved up in the array decrementing the REAR.
  3. The array in the queue can be maintained as an ordered circular array.