# Queue

*Definition*

Queue is a linear list where all additions are made at one end, called rear, and all deletions (accesses) are made from another end called front of the list. So, in a queue there must be two indicators or pointers. One is rear to add elements and another is front used to delete (access) the elements from the queue.

Queue is a FIFO (First In First Out) structure. That means the element that is added first will be deleted (accessed) first. As, stack. Queue can be implemented using array and linked list.

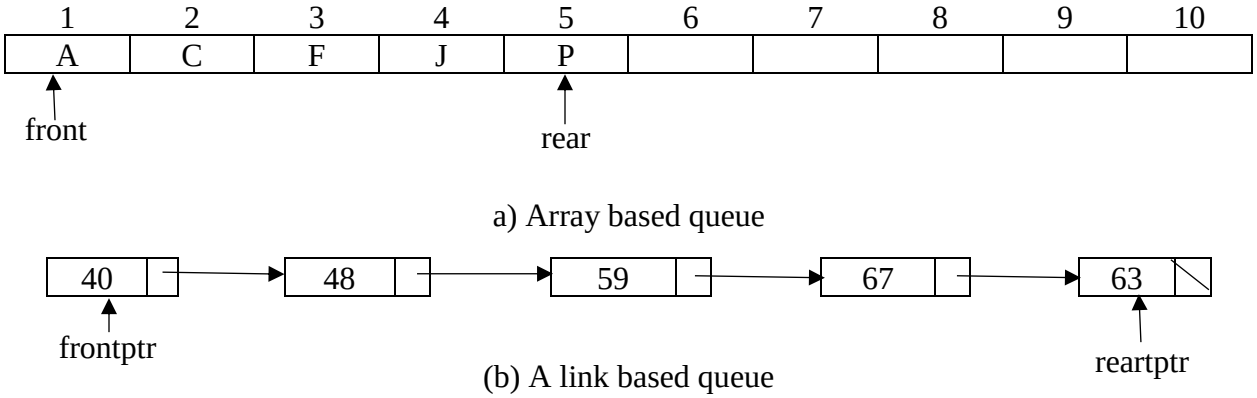



Fig. 1: Graphical representation of queue

**Array based queue**

The queue that will be created using an array is called array based queue. Here, we have to use two indicators (two index identifiers). One indicator will mark the front element and another will mark the rear element of the queue.

**Addition of an element in an array based queue**

We know in a queue element is added at the rear. So, to add an item at first we increase rear index and then place the element in the array based queue using the rear index.

**Algorithm to add an element to queue**

1. Declare array based queue and other variables:

```
que[1…..M], item, front, rear;
```

2. if(rear=0)

```
{
front=rear=1;
que[rear]=item;
}
```

3. if(rear<M)

{

rear=rear+1;

que[rear]=item;

}

else print “Over Flow message”

4. Output: Updated queue.

*Example*

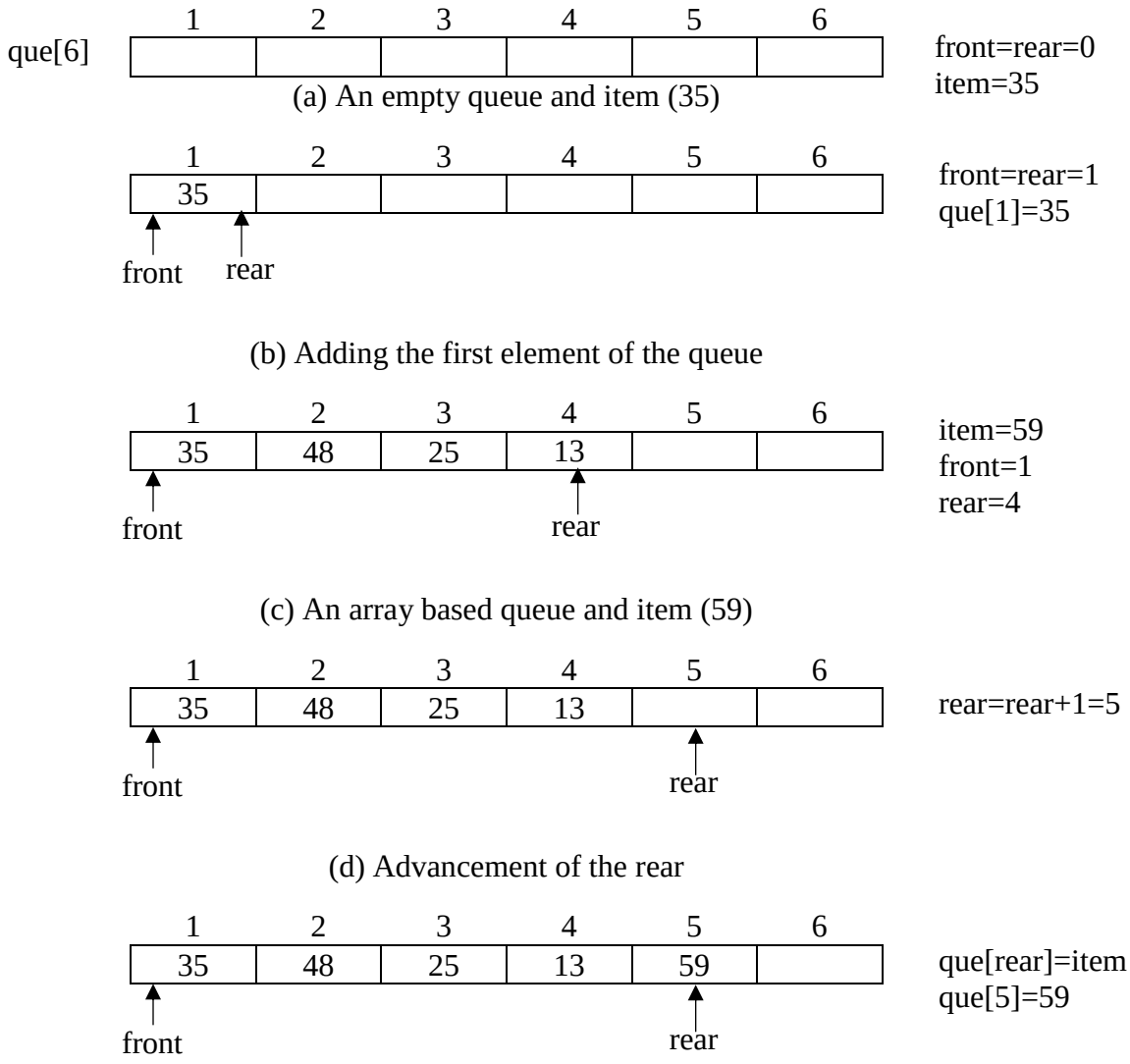



Fig.2: Addition of new items in an array based queue (Pictorial view)

**Deletion of an element from a queue**

We know that, the element is deleted from the front of the queue. So, at first we access the element, and then we increase the front index.

**Algorithm to delete an element from a queue**

1. Declare array based queue and other variables:

    que[1….M], item, rear, front;

2. if(front=rear)

    {

    item=que[front];

    front=rear=0;

    }

3. if(front !=0)

    {

    item=que[front];

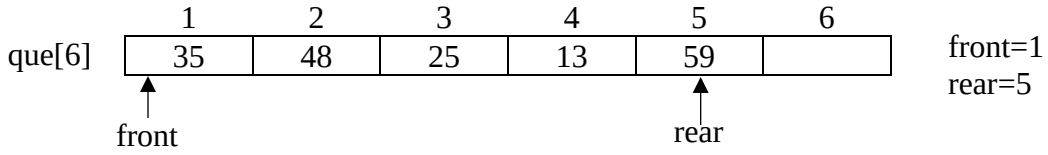
    front=front+1;

    }

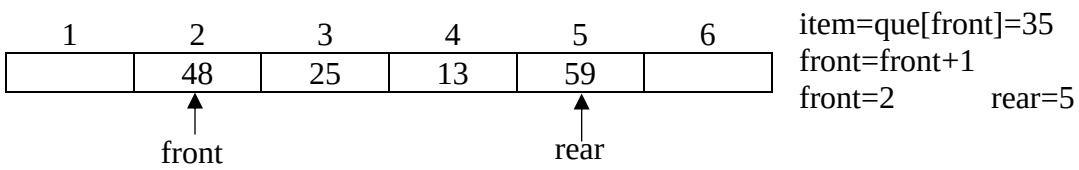
else print “Queue is empty”

4. Output: Updated queue

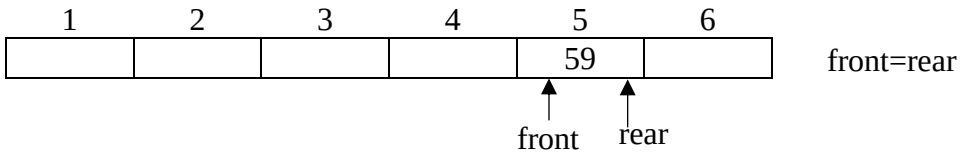
*Example*



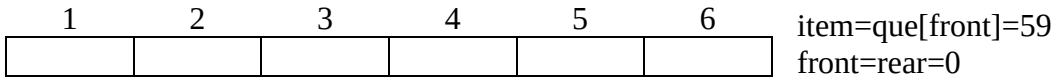

(a) An array based queue




(b) Deleting the first element of the queue




(c) The queue with only one element




(d) Deletion of the last element of the queue

Fig. 3: Deletion process of an array based queue (Pictorial view)

**Drawback of array implementation of queue**

Although, the technique of creating a queue is easy, but there are some drawbacks of using this technique to implement a queue.

- Memory wastage: The space of the array, which is used to store queue elements, can never be reused to store the elements of that queue because the elements can only be inserted at front end and the value of front might be so high so that, all the space before that, can never be filled.

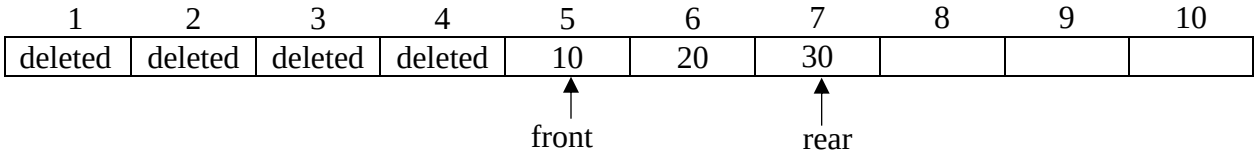



Fig. 4: Limitation of array implementation of queue

The above figure shows how the memory space is wasted in the array representation of queue. In the above figure, a queue of size 10 having 3 elements, is shown. The value of the front variable is 5, therefore, we cannot reinsert the values in the place of already deleted element before the position of front. That much space of the array is wasted and can not be used in the future (for this queue).

- Deciding the array size: On of the most common problem with array implementation is the size of the array which requires to be declared in advance. Due to the fact that, the queue can be extended at runtime depending upon the problem, the extension in the array size is a time taking process and almost impossible to be performed at runtime since a lot of reallocations take place. Due to this reason, we can declare the array large enough so that we can store queue elements as enough as possible but the main problem with this declaration is that, most of the array slots (nearly half) can never be reused. It will again lead to memory wastage.

**Priority Queue**

Priority Queue is an extension of queue with following properties.

1. Every item has a priority associated with it.
2. An element with high priority is dequeued before an element with low priority.
3. If two elements have the same priority, they are served according to their order in the queue.

In the below priority queue, element with maximum ASCII value will have the highest priority.

| Operation | Priority Queue<br>Initial Queue={}<br>Return Value | Queue Content |
|---|---|---|
| insert(C) | | C |
| insert(O) | | C O |
| insert(D) | | C O D |
| remove max | O | C D |
| insert(I) | | C D I |
| insert(N) | | C D I N |
| remove max | N | C D I |
| insert(G) | | C D I G |