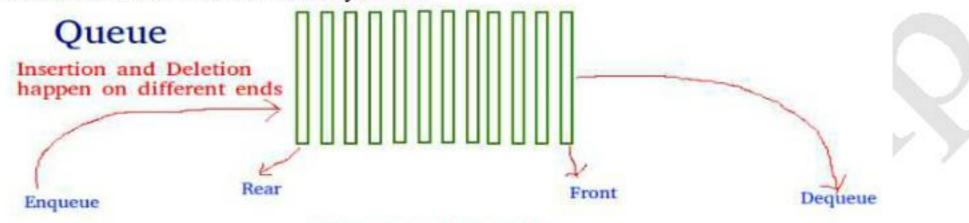
#### 3 Queue

- ➤ A Queue is an ordered collection of items from which items may be deleted at one end (called the front of the queue) and into which items may be inserted at the other end (the rear of the queue).
- > The first element inserted into a queue is the first element to be removed. For this reason a queue is sometimes called FIFO (First In First Out).
- A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



First in first out

## 3.1 The Queue ADT

- Queues are containers, and they hold values of some type. We must therefore speak of the ADT queue of T, where T is the type of the elements held in the queue. The carrier set of this type is the set of all queues holding elements of type T. The carrier set thus includes the empty queue, the queues with one element of type T, the queues with two elements of type T, and so forth.
- > The basic operations (Primitive Operations) also called as Queue ADT that can be performed on queue are
  - MakeEmpty(q): To make q as an empty queue
  - IsEmpty(q): To check whether the queue q is empty. Return true if q is empty, return false otherwise.
  - IsFull(q): To check whether the queue q is full. Return true in q is full, return false otherwise.
  - Enqueue(q, x) or Insert(q,x) (or add): To insert an item x at the rear of the queue, if and only if q is not full.
  - Dequeue(q) or Delete(q) (or remove): To delete an item from the front of the queue q. if and only if q is not empty.
  - Traverse (q): To read entire queue that is display the content of the queue.

## Applications of Queue:

- · Task waiting for the printing
- Time sharing system for use of CPU
- For access to disk storage
- · Task scheduling in operating system

## Types of Queue

- Linear Queue or Simple queue
- Circular queue
- Double ended queue (de-queue)
- Priority queue: Priority queue is generally implemented using linked list

#### 3.2 Implementation of queue:

There are two techniques for implementing the queue:

- Array implementation of queue(static memory allocation)
- Linked list implementation of queue(dynamic memory allocation)

If Queue is implemented using arrays, we must be sure about the exact number of elements we want to store in the queue, because we have to declare the size of the array at design time or before the processing starts. In this case, the beginning of the array will become the front for the queue and the last location of the array will act as rear for the queue.

## 3.2.1 Linear Queue (Algorithm for Queue Operations)

## Algorithm for insertion (or Enqueue )

```
QINSERT(Queue[MaxSize], data)
```

```
Step 1: if Rear = MaxSize -1

print "Overflow" and Exit

Step 2: if Front = -1 and Rear = -1

SET Front = Rear = 0

else

SET Rear = Rear + 1

[END OF IF]

Step 3: SET Queue[Rear] = data

Step 4: Exit
```

## Algorithm for deletion (or Dequeue ) QDELETE (Queue[MaxSize])

```
Step 1: if Front = -1 OR Front > Rear

print "Underflow" and Exit

else

SET data = Queue[Front]

SET Front = Front + 1

[END OF IF]

Step 2: EXIT
```

## Declaration of a Queue:

```
# define MaxSize 50  /* size of the queue items*/
struct QUEUE
{
    int front;
    int rear;
    int items[MaxSize];
};
typedef struct QUEUE queue;
```

## Defining the operations of linear queue:

• The MakeEmpty function:

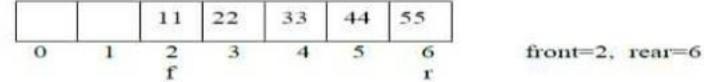
```
void initQueue(queue *q)
{
     q->rear = -1;
     q->front = -1;
}
```

The isEmpty function:

```
The isfull function:
        int isFull(queue *q)
                if(q->rear==MaxSize-1)
                        return 1;
                else
                        return 0;
The Enqueue function:
        void Enqueue(queue *q, int data)
                        if(isFull(q))
                                printf("queue is full");
                                exit(1);
                        if(q - rear = -1 & q - rear = -1)
                                q->rear = 0;
                                q->front = 0;
                        else
                             q->rear++;
                q->items[q->rear] = data;
The Dequeue function:
        int Dequeue(queue *q)
            int data;
                if(isEmpty(q))
                        printf("queue is Empty");
                        exit(1);
                else
                        data = q->items[q->front];
                        q->front++;
                        return data;
```

## 3.2.2 Problems with Linear queue implementation:

- Both rear and front indices are increased but never decreased.
- As items are removed from the queue, the storage space at the beginning of the array is discarded and never used again. Wastage of the space is the main problem with linear queue which is illustrated by the following example.



This queue is considered full, even though the space at beginning is vacant.

➤ To remove this problem, the first and an obvious solution which comes into our mind is whenever an element is deleted, shift all afterward elements to the left by one position. But if the queue is too large of say 5000 elements it will be difficult job to do and time consuming too. To remove this problem we use circular queue.

## Representation of Queue in C

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MaxSize 50
struct QUEUE
        int items[MaxSize];
        int rear;
        int front;
typedef struct QUEUE queue;
void initQueue(queue *q)
        q->rear = -1;
        q->front = -1;
int isEmpty(queue *q)
        if(q \rightarrow front == -1 || q \rightarrow rear < q \rightarrow front)
                return 1;
        else
                return 0;
int isFull(queue *q)
        if(q->rear==MaxSize-1)
                return 1;
        else
                return 0;
void Enqueue (queue*,int);
int Dequeue (queue*);
void display(queue*);
int main()
        int ch,data,flag=1;
        queue lq;
        queue *q;
        q= &lq;
        initQueue(q);
        printf("Menu for program:\n");
        printf("1:insert\n2:delete\n3:display\n4:exit\n");
        do
                printf("\nEnter youer choice\n");
                scanf("%d",&ch);
                switch(ch)
                         case 1:
                                 printf("Enter data to be inserted\n");
                                 scanf("%d",&data);
```

```
insert(q,data);
                               break;
                       case 2:
                               data= Dequeue(q);
                               printf("Deleted item is:");
                               printf("%d\n",data);
                               break;
                       case 3:
                               display(q);
                               break;
                       case 4:
                               flag=0;
                               break;
                       default:
                               printf("Your choice is wrong\n");
        }while(flag);
       getch();
       return 0;
/********insert function*********/
void Enqueue(queue *q,int data)
               if(isFull(q))
                       printf("Queue is full\n");
                       exit(1);
               if(q - rear == -1 \&\& q - rear == -1)
                       q->rear = 0;
                       q->front = 0;
               else
                       q->rear++;
       q->items[q->rear] = data;
/**********delete function************/
int Dequeue(queue *q)
       int d;
       if(isEmpty(q))
               printf("Queue is empty\n");
               exit(1);
       else
               d=q->items[q->front];
               q->front++;
               return d;
```

## Note:

- Wastage of the space is the main problem with linear queue,
  - To remove this problem, the first and an obvious solution which comes into our mind is whenever an element
    is deleted, shift all afterward elements to the left by one position.

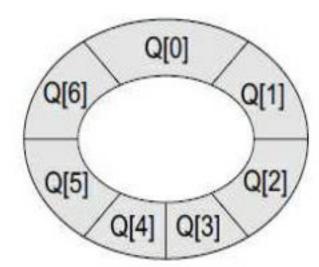
 To remove this problem. Another solution is don't shift elements at time of deletion do normal deletion by simply increasing front value but whenever an element need to be inserted, check if rear is equal to maxsize-1 and front is not equal to zero then shift all elements so that front start from zero position.

```
/***********Modified insert function**********/
                void Enqueue(queue *q,int data)
                       int i,j;
                               if(isFull(q))
                                       printf("Queue is full\n");
                                       exit(1);
                               if(q->rear == -1 && q->front == -1)
                                       q->rear = 0;
                                       q->front = 0;
                               else if(q->rear == MaxSize-1 && q->front != 0)
                                       for(i=q->front, j=0; i <= q->rear; i++, j++)
                                               q->items[j]=q->items[i];
                                       q->rear = q->rear- q->front;
                                       q->front = 0;
                                else
                                       q->rear++;
                       q->items[q->rear] = data;
```

But if the queue is too large it will be difficult job to do shifting and time consuming too (Time Complexity high).
To remove this problem we use circular queue.

#### 3.3 Circular Queue

- Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.
- ➤ A circular queue is one in which the insertion of a new element is done at 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 queue implementation as array.

## 3.3.1 Algorithm to insert an element in a circular queue

```
This algorithm is assume that rear and front are initially set to MaxSize-1.

1. if (front==(rear+1)% MaxSize)
    print Queue is full and exit
    else
        rear=(rear+1)% MaxSize; [increment rear by 1]

2. cqueue[rear]=data;

3. end
```

## 3.3.2 Algorithm to delete an element from a circular queue

This algorithm is assume that rear and front are initially set to MaxSize -1.

```
    if (rear==front) [checking empty condition]
        print Queue is empty and exit
    front=(front+1)% MaxSize; [increment front by 1]
    data=cqueue[front];
    return data;
    end
```

#### **Declaration of a Circular Queue**

```
# define MaxSize 50 /* size of the circular queue items*/
struct circularQueue
{
    int front;
    int rear;
    int items[MaxSize];
};
typedef struct circularQueue cqueue;
```

## Operations on Circular Queue:

```
The MakeEmpty function:

void initQueue(cqueue *q)
{

q->rear= MaxSize -1;

q->front= MaxSize -1;
```

## **The IsEmpty function:**

```
int isEmpty(cqueue *q)
               if(q - rear = q - front)
                       return 1;
               else
                       return 0;
The Isfull function:
        int isFull(cqueue *q)
               if(q->front==(q->rear+1)% MaxSize)
                       return 1;
               else
                       return 0;
The Enqueue function:
        void Enqueue(cqueue *q, int data)
               if(IsFull(q))
                       printf("queue is full");
                       exit(1);
               else
                       q->rear=(q->rear+1)% MaxSize;
                       q->items[q->rear]=data;
The Dequeue function:
        int Dequeue(cqueue *q)
          int data;
               if(IsEmpty(q))
                       printf("queue is Empty");
                       exit(1);
               else
                       q->front=(q->front+1)% MaxSize;
                       data = q->items[q->front];
                       return data;
```

## Representation of circular queue in C, with secrifying one cell (One Position Vacant)

```
#include<stdio.h>
#include<conio.h>
#define MaxSize 50
struct circularQueue
       int items[MaxSize];
       int rear;
       int front;
typedef struct circularQueue cqueue;
void initQueue(cqueue *q)
       q->rear= MaxSize -1;
       q->front= MaxSize -1;
int isEmpty(cqueue *q)
       if(q->rear==q->front)
               return 1;
       else
               return 0;
int isFull(cqueue *q)
       if(q->front==(q->rear+1)\% MaxSize)
               return 1;
       else
               return 0;
void Enqueue(cqueue*);
void Dequeue(cqueue*);
void Display(cqueue*);
void main()
       int ch,flag=1;
       cqueue *q;
       cqueue cq;
       q = &cq;
       initQueue(q);
       printf("Menu for program:\n");
       printf("1:insert\n2:delete\n3:display\n4:exit\n");
               do
                       printf("\nEnter youer choice\n");
                       scanf("%d",&ch);
                       switch(ch)
                               case 1:
                                       Enqueue(q);
                                       break;
                               case 2:
```

```
Dequeue(q);
                                      break;
                              case 3:
                                     Display(q);
                                      break;
                              case 4:
                                      flag = 0;
                                      break;
                              default:
                                      printf("Your choice is wrong\n");
               }while(flag);
       getch();
/********insert function*********/
void Enqueue(cqueue *q)
       int d;
               if(isFull(q))
                      printf("Queue is full\n");
               else
                      q->rear=(q->rear+1)%MaxSize;
                      printf ("Enter data to be inserted\n");
                      scanf("%d",&d);
                      q->items[q->rear]=d;
/********delete function***********/
void Dequeue(cqueue *q)
       int data;
       if(isEmpty(q))
               printf("Queue is empty\n");
       else
               q->front=(q->front+1)%MaxSize;
               data = q->items[q->front];
               printf("Deleted item is:");
               printf("%d\n",data);
/************display function*********/
void Display(cqueue *q)
       int i;
       if(isEmpty(q))
               printf("Queue is empty\n");
       else
               printf("Items of queue are:\n");
               for(i=(q->front+1)%MaxSize;i!=q->rear;i=(i+1)%MaxSize)
```

```
printf("%d\t",q->items[i]);
}
printf("%d\t",q->items[q->rear]);
}
```

**Representation of circular queue in C, without secrifying one cell by using a count variable** 

```
#include<stdio.h>
#include<conio.h>
#define MaxSize 5
struct circularQueue
       int items[MaxSize];
       int rear;
       int front;
       int count;
typedef struct circularQueue cqueue;
void initQueue(cqueue *q)
       q->rear= MaxSize -1;
       q->front= MaxSize -1;
       q->count = 0;
int isEmpty(cqueue *q)
       if(q>count==0)
               return 1;
       else
               return 0;
int isFull(cqueue *q)
       if(q>count == MaxSize)
               return 1;
       else
               return 0;
void Enqueue(cqueue*);
void Dequeue(cqueue*);
void Display(cqueue*);
void main()
       int ch,flag=1;
       cqueue *q;
       cqueue cq;
       q = &cq;
       initQueue(q);
       printf("Menu for program:\n");
       printf("1:insert\n2:delete\n3:display\n4:exit\n");
```

```
do
                      printf("\nEnter youer choice\n");
                      scanf("%d",&ch);
                      switch(ch)
                              case 1:
                                      Enqueue(q);
                                      break;
                              case 2:
                                      Dequeue(q);
                                      break;
                              case 3:
                                      Display(q);
                                      break;
                              case 4:
                                      flag = 0;
                                      break;
                              default:
                                      printf("Your choice is wrong\n");
               }while(flag);
       getch();
/********insert function********/
void Enqueue(cqueue *q)
       int d;
               if(isFull(q))
                      printf("Queue is full\n");
               else
                      q->rear=(q->rear+1)%MaxSize;
                      printf ("Enter data to be inserted\n");
                      scanf("%d",&d);
                      q->items[q->rear]=d;
                      q->count++;
/********delete function**********/
void Dequeue(cqueue *q)
       int data;
       if(isEmpty(q))
               printf("Queue is empty\n");
       else
               q->front=(q->front+1)%MaxSize;
               data = q->items[q->front];
               printf("Deleted item is:");
               printf("%d\n",data);
               q->count--;
```

```
/*******************************

void Display(cqueue *q)
{

    int i;
    if(isEmpty(q))
        printf("Queue is empty\n");
    else
    {

        printf("Items of queue are:\n");
        for(i=(q->front+1)%MaxSize;i!=q->rear;i=(i+1)%MaxSize)
        {

            printf("%d\t",q->items[i]);
        }
        printf("%d\t",q->items[q->rear]);
    }
}
```

## Representation of circular queue in C

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MaxSize 10
struct CircularQueue
  int items[MaxSize];
  int rear, front;
};
typedef struct CircularQueue cqueue;
void initialize(cqueue *q);
int isEmpty(cqueue *q);
int isFull(cqueue *q);
void Insert(cqueue *q);
void Delete(cqueue *q);
void display(cqueue *q);
int main()
  int ch,flag=1;
  cqueue cq;
  cqueue *q;
  q= &cq;
  initialize(q);
  printf("\n1.Insert(Rear)\n2.Delet(Front)\n5.Print\n6.Exit\n");
  do
    printf("\nEnter your choice:");
    scanf("%d",&ch);
    switch(ch)
       case 1:
            Insert(q);
            break;
       case 2:
            Delete(q);
            break;
```

```
case 3:
           display(q);
            break;
       case 4:
            flag = 0;
            break;
       default:
              printf("Your choice is wrong\n");
  }while(flag);
  getch();
  return 0;
void initialize(cqueue *q)
  q->rear=-1;
  q->front=-1;
int isEmpty(cqueue *q)
  if(q->rear==-1)
    return 1;
  else
      return 0;
int isFull(cqueue *q)
  if((q->rear+1)%MaxSize==q->front)
    return 1;
  else
      return 0;
void Insert(cqueue *q)
       int d;
  if(isFull(q))
               printf("Queue is Full");
               return;
  else if(isEmpty(q))
       q->rear=0;
       q->front=0;
  else
    q->rear=(q->rear+1)%MaxSize;
  printf ("Enter data to be inserted\n");
       scanf("%d",&d);
       q->items[q->rear]=d;
```

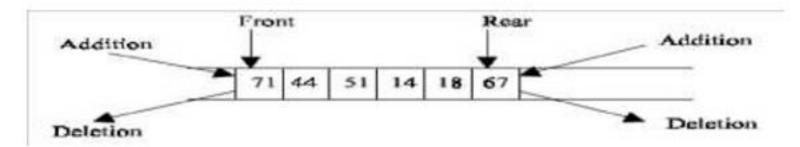
```
void Delete(cqueue *q)
  int data;
       if(isEmpty(q))
               printf("Queue is Empty");
               return;
  data = q->items[q->front];
       if(q \rightarrow front == q \rightarrow rear)
                                                          //delete the last element
               initialize(q);
       else
               q->front=(q->front+1)%MaxSize;
  printf("Deleted item is:%d",data);
void display(cqueue *q)
       int i;
       if(isEmpty(q))
               printf("Queue is empty\n");
       else
               printf("Items of queue are:\n");
               i=q->front;
               while(i!=q->rear)
                        printf("\n%d",q->items[i]);
                        i=(i+1)% MaxSize;
               printf("\n\%d\n",q->items[q->rear]);
```

## 3.3.3 Application of Circular Queue

- Memory Management: The unused memory locations in the case of ordinary queues can be utilized in circular queues.
- Traffic system: In computer controlled traffic system, circular queues are used to switch on the traffic lights one
  by one repeatedly as per the time set.
- CPU Scheduling: Operating systems often maintain a queue of processes that are ready to execute or that are
  waiting for a particular event to occur.

#### 3.4 DEQUE

A deque is a homogeneous list in which elements can be added or inserted (called Enqueue operation) and deleted or removed (called Dequeue operation) from both the ends. ie; we can add a new element at the rear or front end and also we can remove an element from both front and rear end. Hence it is called Double Ended Queue.



There are two types of deque depending upon the restriction to perform insertion or deletion operations at the two ends. They are

- Input restricted deque: An input restricted deque is a deque, which allows insertion at only 1 end, rear end, but allows deletion at both ends, rear and front end of the lists.
- Output restricted deque: An output-restricted deque is a deque, which allows deletion at only one end, front
  end, but allows insertion at both ends, rear and front ends, of the lists.

## **Declaration of a Queue:**

```
# define MaxSize 50  /* size of the queue items*/
struct QUEUE
{
    int front;
    int rear;
    int items[MaxSize];
};
```

# typedef struct QUEUE queue;

# The possible operation (Deque ADT) performed on deque is

- Insertion of an element at the REAR end of the queue
- Deletion of an element from the FRONT end of the queue
- Insertion of an element at the FRONT end of the queue
- Deletion of an element from the REAR end of the queue

#### Functions to carry out these four operations using linear array are:

Insertion of an element at the Rear end of the queue

```
void dqinsert_rear(queue *q, int data)
{
    if(q->rear == (MaxSize - 1))
    {
        printf("Queue is full");
        exit(1);
    }
    else
    {
        q->rear = q->rear + 1;
        q->items[q->rear] = data;
    }
}
```

Deletion of an element from the FRONT end of the queue

```
int dqdelete_front(queue *q)
{
    int data;
    if(q->front == q->rear)
    {
        printf("Queue is empty");
        exit(1);
```

```
}
else
{
     data = q->items[q->front];
     q->front = q->front + 1;
     return data;
}
```

Insertion of an element at the FRONT end of the queue

Deletion of an element from the REAR end of the queue

```
int dqdelete_rear(queue *q)
{
    int data;
    if(q->front == q->rear)
    {
        printf("Queue is empty");
        exit(1);
    }
    else
    {
        data = q->items[q->rear];
        q->rear = q->rear - 1;
        return data;
    }
}
```

Function to display the contents (Status) of a queue

```
void dq_display(queue *q)
{
    int i;
    if(q->front <= q->rear)
```

```
printf("Status of the Queue");
        for(i=q->front;i<=q->rear;i++)
                printf("%d",q->items[i]);
else
        printf("Queue is Empty")
```

```
Implementation of Deque or Double Ended Queue using circular array
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MaxSize 5
struct Deque
  int items[MaxSize];
  int rear, front;
typedef struct Deque deque;
void initialize(deque *q);
int isEmpty(deque *q);
int isFull(deque *q);
void insert_rear(deque *q);
void insert_front(deque *q);
void delete_front(deque *q);
void delete_rear(deque *q);
void display(deque *q);
int main()
   int ch,flag=1;
  deque dq;
  deque *q;
  q = &dq;
  initialize(q);
  printf("\n1.Insert(Rear)\n2.Insert(Front)\n3.Delete(Rear)\n4.Delet(Front)\n5.Print\n6.Exit\n");
     printf("\nEnter your choice:");
     scanf("%d",&ch);
     switch(ch)
       case 1:
            insert_rear(q);
            break;
       case 2:
            insert_front(q);
            break;
       case 3:
            delete_rear(q);
            break;
       case 4:
            delete_front(q);
```

```
break;
       case 5:
           display(q);
            break;
       case 6:
           flag = 0;
           break;
       default:
           printf("Your choice is wrong\n");
  }while(flag);
  getch();
  return 0;
void initialize(deque *q)
  q->rear=-1;
  q->front=-1;
int isEmpty(deque *q)
  if(q->rear==-1)
    return 1;
  else
        return 0;
int isFull(deque *q)
  if((q->rear+1)\%MaxSize==q->front)
    return 1;
  else
        return 0;
void insert_rear(deque *q)
        int d;
  if(isFull(q))
    printf("Queue is Full");
    return;
  else if(isEmpty(q))
        q->rear=0;
        q->front=0;
   else
      q->rear=(q->rear+1)%MaxSize;
  printf ("Enter data to be inserted\n");
        scanf("%d",&d);
        q->items[q->rear]=d;
void insert_front(deque *q)
```

```
int d;
        if(isFull(q))
                printf("Queue is Full");
                return;
        else if(isEmpty(q))
                 q->rear=0;
                 q->front=0;
        else
             q->front=(q->front-1+MaxSize)%MaxSize;
        printf ("Enter data to be inserted\n");
        scanf("%d",&d);
        q->items[q->front]=d;
void delete_front(deque *q)
  int data;
  if(isEmpty(q))
     printf("Queue is Empty");
     return;
  data = q->items[q->front];
                                                          //delete the last element
  if(q->front == q->rear)
        initialize(q);
  else
        q->front=(q->front+1)%MaxSize;
  printf("Deleted item is:%d",data);
void delete_rear(deque *q)
  int data;
        if(isEmpty(q))
                printf("Queue is Empty");
                return;
        data=q->items[q->rear];
        if(q \rightarrow front == q \rightarrow rear)
                                                          //delete the last element
                initialize(q);
        else
            q->rear=(q->rear-1+MaxSize)%MaxSize;
  printf("Deleted item is:%d",data);
void display(deque *q)
        int i;
        if(isEmpty(q))
                printf("Queue is empty\n");
        else
```

```
printf("Items of queue are:\n");
i=q->front;
while(i!=q->rear)
{
    printf("\n%d",q->items[i]);
    i=(i+1)%MaxSize;
}
printf("\n%d\n",q->items[q->rear]);
}
```

## 3.5 Priority Queues

- ➤ 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:.
  - An element of higher priority is processed before any element of lower priority.
  - If two elements has same priority then they are processed according to the order in which they were added to the queue.
- The best application of priority queue is observed in CPU scheduling.
  - The jobs which have higher priority are processed first.
  - If the priority of two jobs is same this jobs are processed according to their position in queue.
  - A short job is given higher priority over the longer one.

## Types of priority queues:

- Ascending priority queue(min priority queue): An ascending priority queue is a collection of items into
  which items can be inserted arbitrarily but from which only the smallest item can be removed.
- Descending priority queue(max priority queue): An descending priority queue is a collection of items into
  which items can be inserted arbitrarily but from which only the largest item can be removed
- The implementation of priority queue using array will yield n comparisons (in liner search), so the time complexity is much higher than the other queue for inserting an element. So it is always better to implement the priority queue using linked list - where a node can be inserted at anywhere in the list.

#### 3.6 APPLICATION OF QUEUES

- Round robin techniques for processor scheduling is implemented using queue.
- Printer server routines (in drivers) are designed using queues.
- All types of customer service software (like Railway/Air ticket reservation) are designed using queue to give proper service to the customers.
- Disk Driver: maintains a queue od disk input/output requests
- Scheduler (e.g, in operating system): maintains a queue of processes awaiting a slice of machine time.
- Call center phone systems will use a queue to hold people in line until a service representative is free.
- Buffers on MP3 players and portable CD players, iPod playlist. Playlist for jukebox add songs to the end, play from the front of the list.
- ➤ When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
- When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.