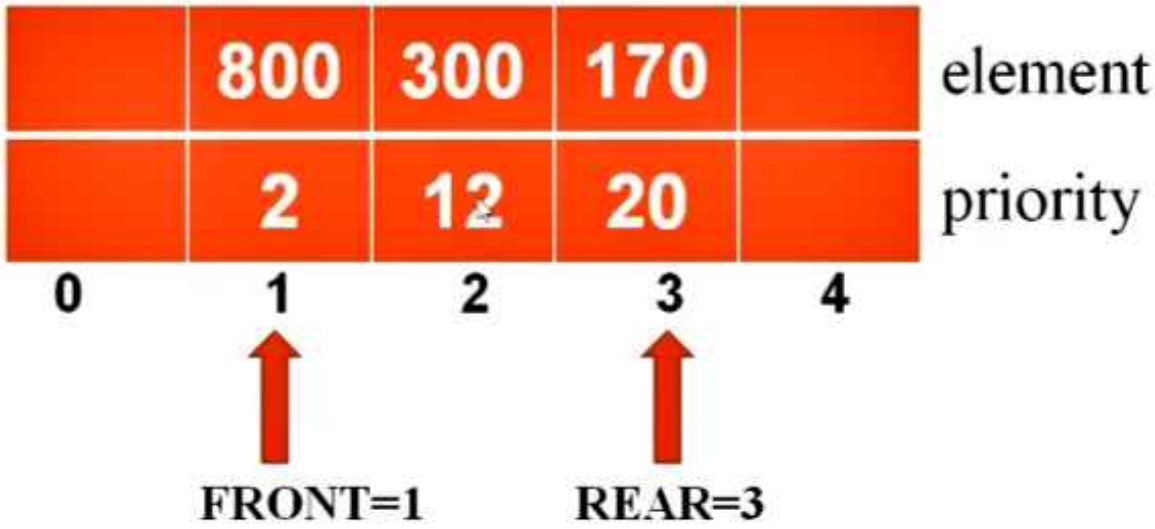
PRIORITY QUEUE

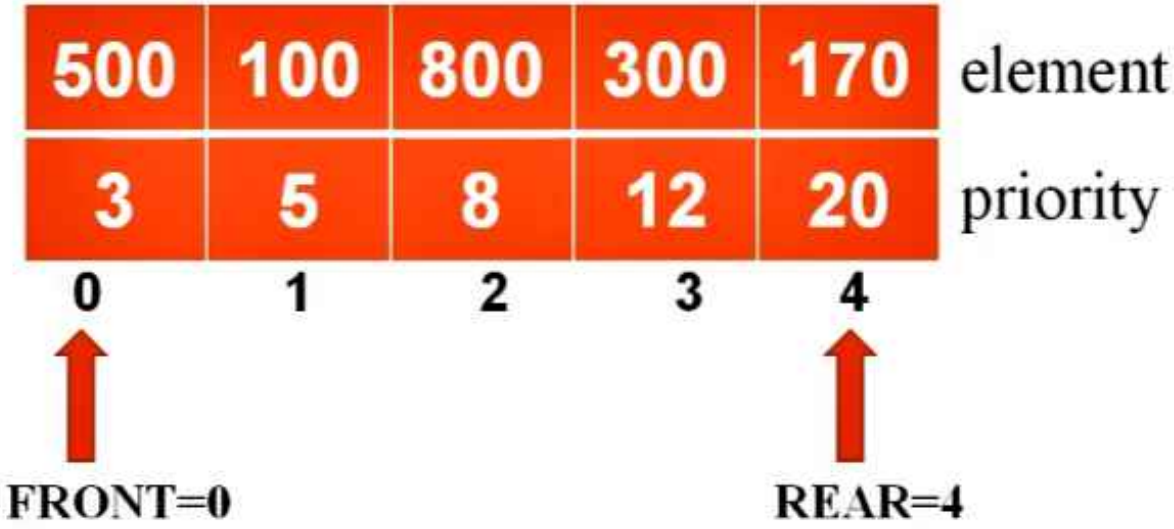
- Priority Queue is an extension of queue with following properties.
  - Every item has a priority associated with it.
  - An element with high priority is dequeued before an element with low priority.
  - If two elements have the same priority, they are served according to their order in the queue.


800
300
170
element
2
12
20
priority
0
1
2
3
4
FRONT=1
REAR=3

ENQUEUE_PQ(120,4)

Case 1: FRONT=0 and REAR=SIZE-1

Priority Queue is FULL



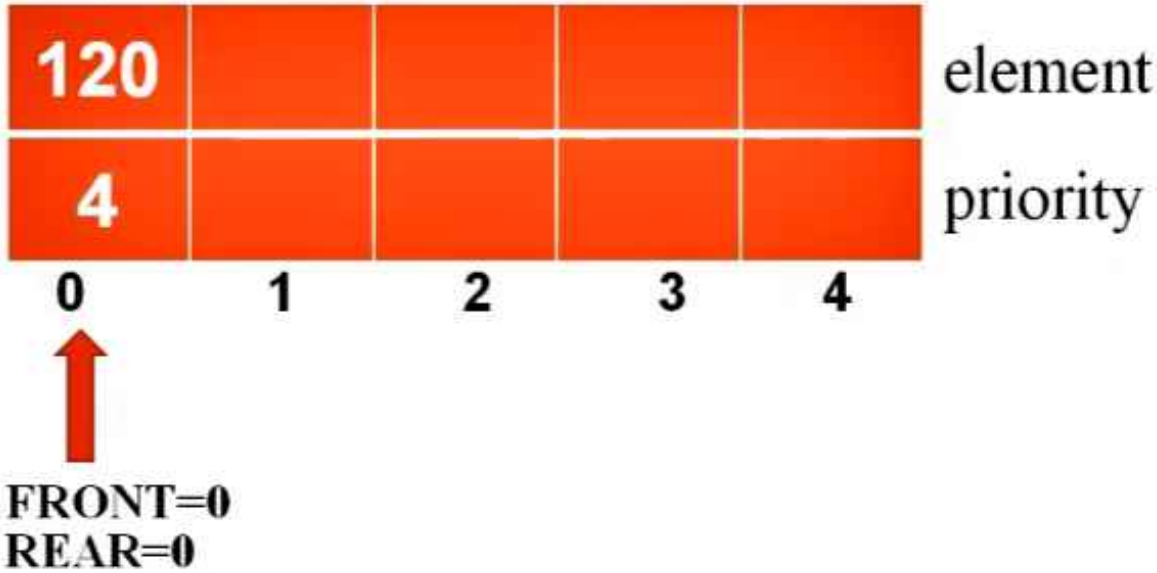
ENQUEUE_PQ(120,4)

Case 2: **FRONT=-1 and REAR=-1**

FRONT=REAR=0

A[REAR].item=120

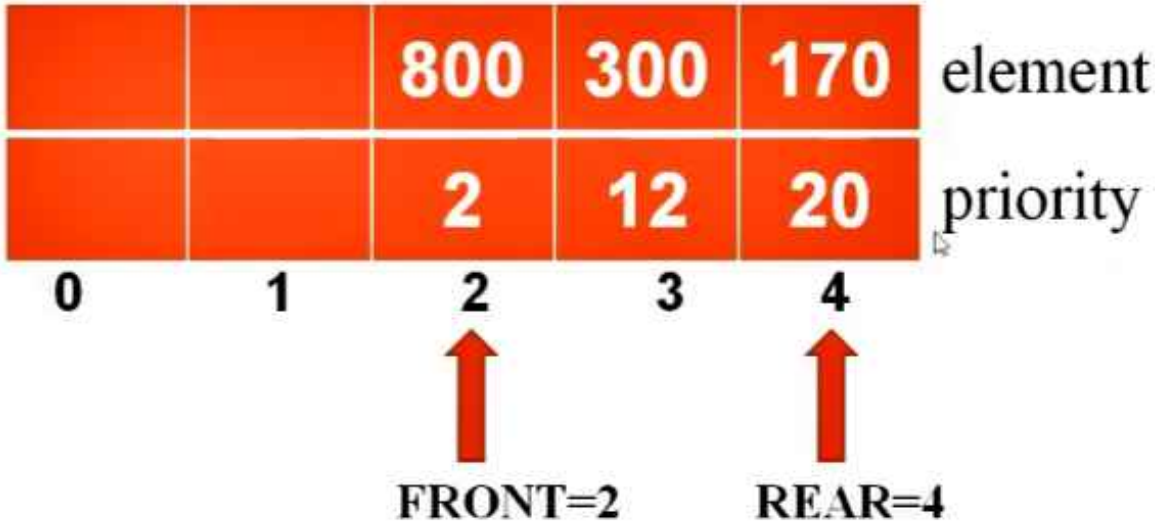
A[REAR].priority=4



ENQUEUE_PQ(120,4)

Case 3: if REAR=SIZE-1

Shift all elements one position to left



ENQUEUE_PQ(120,4)

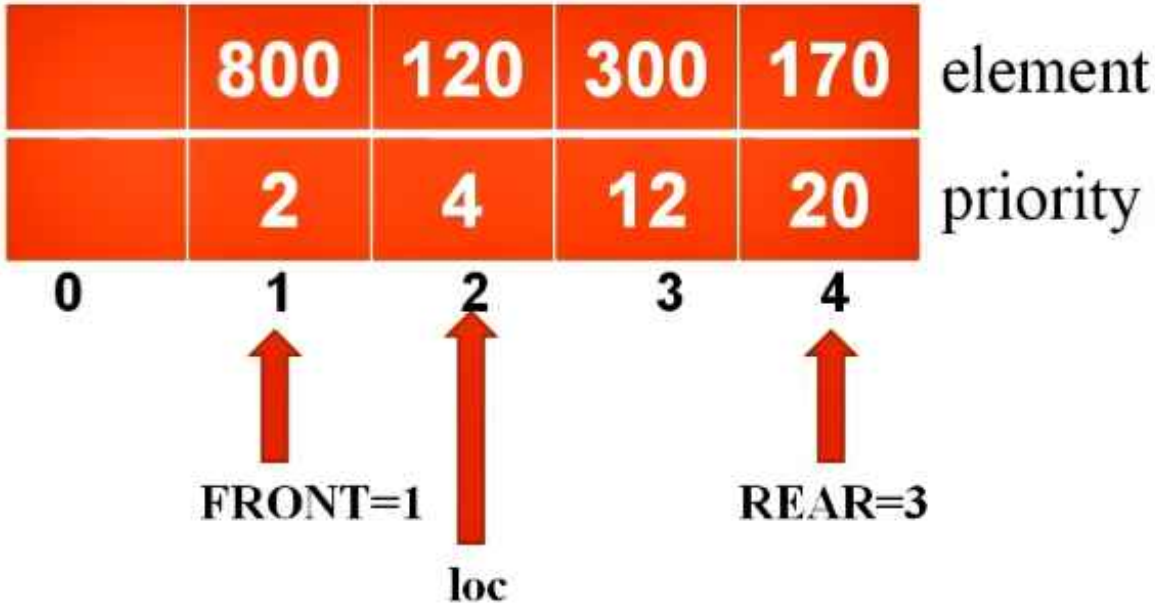
Case 3: if REAR=SIZE-1

Shift all elements one position to left

Find the location where the new elmt is to be inserted

Shift loc to REAR elements one position to right

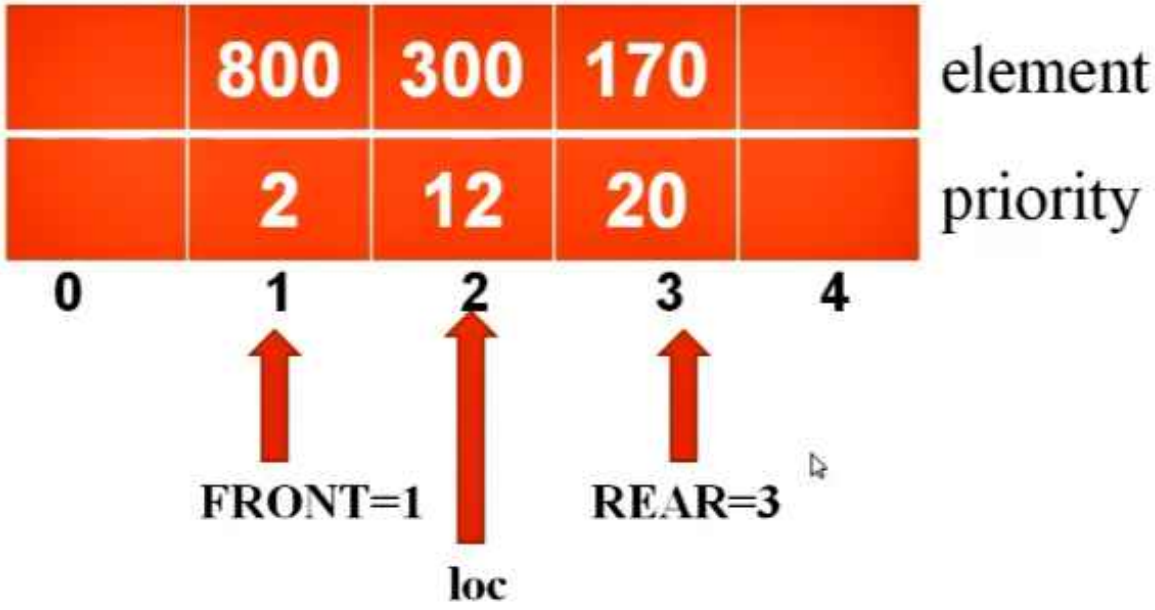
Insert the data at the index loc



ENQUEUE_PQ(120,4)

Case 4: All other cases

Find the location where the new elmt is to be inserted

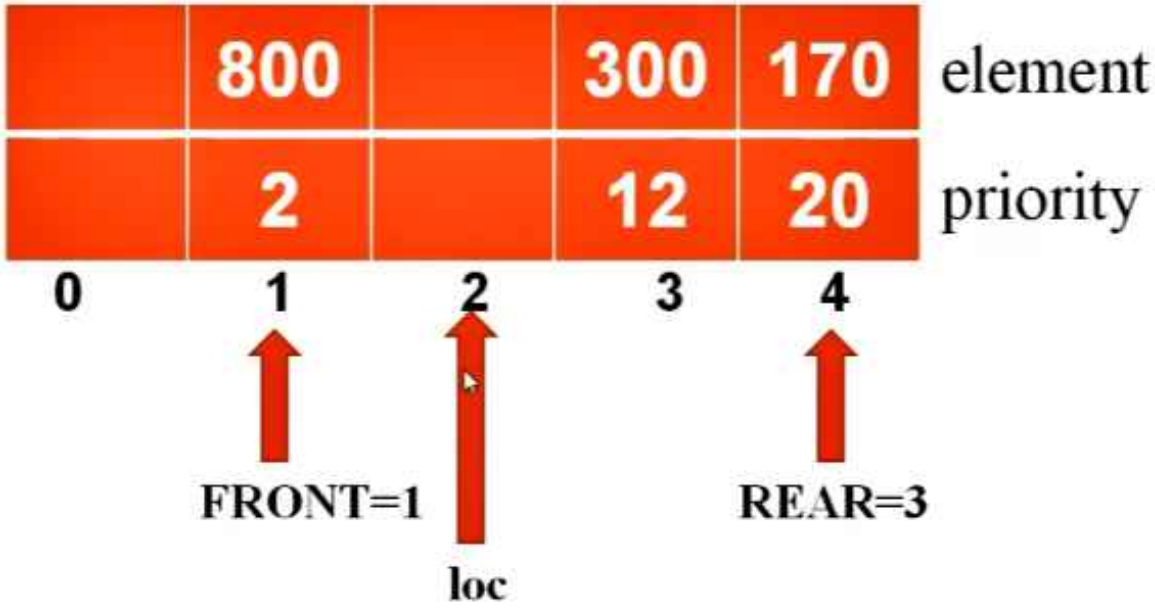


ENQUEUE_PQ(120,4)

Case 4: All other cases

Find the location where the new elmt is to be inserted

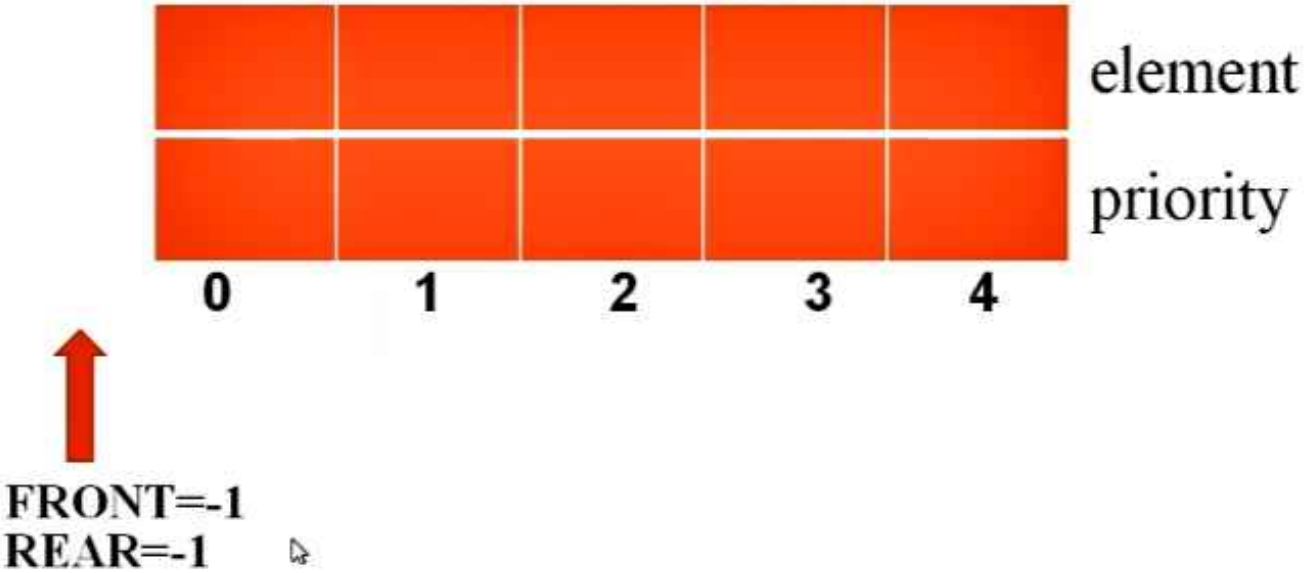
Shift elements from loc to REAR one position right



DEQUEUE_PQ()

Case 1: FRONT=-1 and REAR=-1

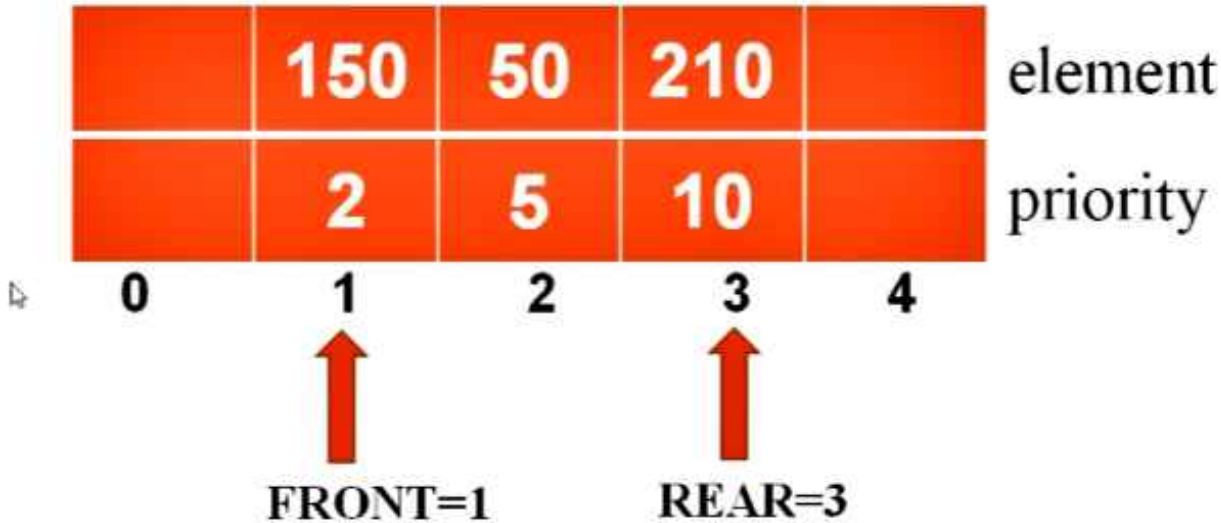
Print "Priority Queue is EMPTY"



PRIORITY QUEUE – DEQUEUE Algorithm

Algorithm DISPLAY_PQ()

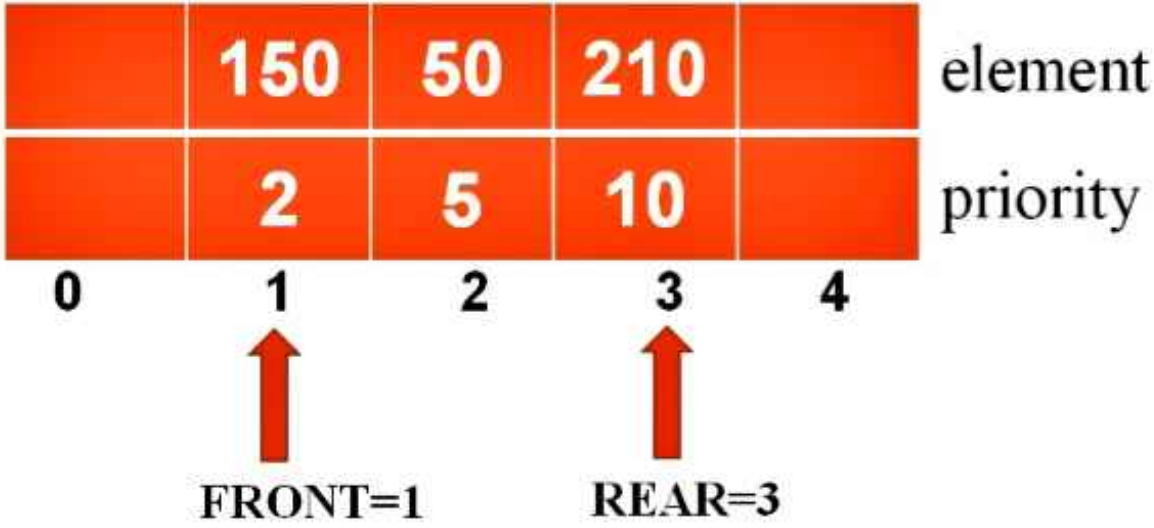
```
{       if  FRONT=-1then
                Print "Priority Queue is EMPTY"
        else
        {       for i=FRONT to REAR do
                        Print  A[i].item
        }
}
```



PRIORITY QUEUE – DEQUEUE Algorithm

Algorithm DISPLAY_PQ()

```
{       if  FRONT=-1then
                Print "Priority Queue is EMPTY"
        else
        {       for i=FRONT to REAR do
                        Print  A[i].item
        }
}
```



PRIORITY QUEUE – ENQUEUE Algorithm

Algorithm ENQUEUE_PQ(ITEM,PRIORITY)

```
{
        if FRONT=0 and REAR=SIZE-1 then
                Print "Priority Queue is FULL"
        else if FRONT=-1 then
        {
            FRONT=REAR=0
            A[FRONT].item=ITEM
            A[FRONT].priority=PRIORITY
        }
```

```
else if REAR=SIZE-1 then
{
        for i=FRONT to REAR do
                A[i-1]=A[i]
        FRONT=FRONT-1
        REAR=REAR-1

        for i=REAR to FRONT do
        {       if A[i].priority<PRIORITY then
                        break;
        }
        loc=i+1
        for i=REAR to loc do
                A[i+1]=A[i]
        A[loc].item=ITEM
        A[loc].priority=PRIORITY
        REAR=REAR+1
}
```

```
else
{
        for i=REAR to FRONT do
        {       if A[i].priority<PRIORITY then
                        break;
        }
        loc=i+1
        for i=REAR to loc do
                A[i+1]=A[i]
        A[loc].item=ITEM
        A[loc].priority=PRIORITY
        REAR=REAR+1
}
}
```

PRIORITY QUEUE – DEQUEUE Algorithm

Algorithm DEQUEUE_PQ()

```
{       if  FRONT=-1then
                Print "Priority Queue is EMPTY"
        else if FRONT=REAR then
        {       Print "Dequeued item is " A[FRONT].item
                FRONT=REAR=-1
        }
        else
        {       Print "Dequeued item is " A[FRONT].item
                FRONT=FRONT+1
        }
}
```

PRIORITY QUEUE – DEQUEUE Algorithm

Algorithm DEQUEUE_PQ()

```
{       if  FRONT=-1then
                Print "Priority Queue is EMPTY"
        else if FRONT=REAR then
        {       Print "Dequeued item is " A[FRONT].item
                FRONT=REAR=-1
        }
        else
        {       Print "Dequeued item is " A[FRONT].item
                FRONT=FRONT+1
        }
}
```

PRIORITY QUEUE IMPLEMENTATION

- **Using ordered Array:**
  - Elements are inserted in the sorted order of their priority. The time complexity = O(n)
  - Deletion operation is performed from the front end. The time complexity= O(1)

- **Using unordered Array:**
  - Elements are inserted at any end. The time complexity = O(1)
  - For deletion, search an element in the Queue with highest priority. The time complexity = O(n)

PRIORITY QUEUE IMPLEMENTATION

- **Using ordered Array:**
  - Elements are inserted in the sorted order of their priority. The time complexity = O(n)
  - Deletion operation is performed from the front end. The time complexity= O(1)

- **Using unordered Array:**
  - Elements are inserted at any end. The time complexity = O(1)
  - For deletion, search an element in the Queue with highest priority. The time complexity = O(n)

PRIORITY QUEUE IMPLEMENTATION

- **Using ordered Array:**
  - Elements are inserted in the sorted order of their priority. The time complexity = O(n)
  - Deletion operation is performed from the front end. The time complexity= O(1)

- **Using unordered Array:**
  - Elements are inserted at any end. The time complexity = O(1)
  - For deletion, search an element in the Queue with highest priority. The time complexity = O(n)

DIFFERENT PRIORITY QUEUES

- **Max-Priority Queue:** Element with highest priority is served first
- **Min-Priority Queue:** Element with lowest priority is served first.

APPLICATIONS OF PRIORITY QUEUE

- CPU Scheduling
- Graph algorithms like Dijkstra's shortest path algorithm, Prim's Minimum Spanning Tree, etc
- All queue applications where priority is involved