Data Structures – CST 201
Module - 2

Syllabus

- Polynomial representation using Arrays
- Sparse matrix
- Stacks
  - Evaluation of Expressions
- Queues
  - Circular Queues
  - **Priority Queues**
  - Double Ended Queues
- Linear Search
- Binary Search

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.

PRIORITY QUEUE- Operations

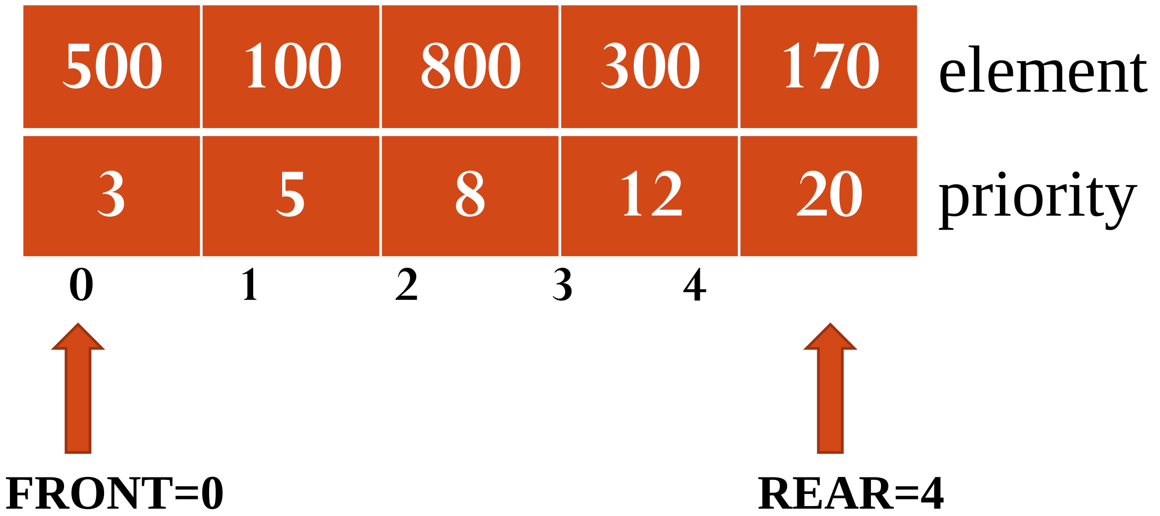
- **ENQUEUE**: Insert an element in the queue based on priority
- **DEQUEUE:** Delete highest priority element from the queue
- **DISPLAY**: Display the contents of the Queue

PRIORITY QUEUE – ENQUEUE

ENQUEUE_PQ(120,4)

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

Priority Queue is FULL



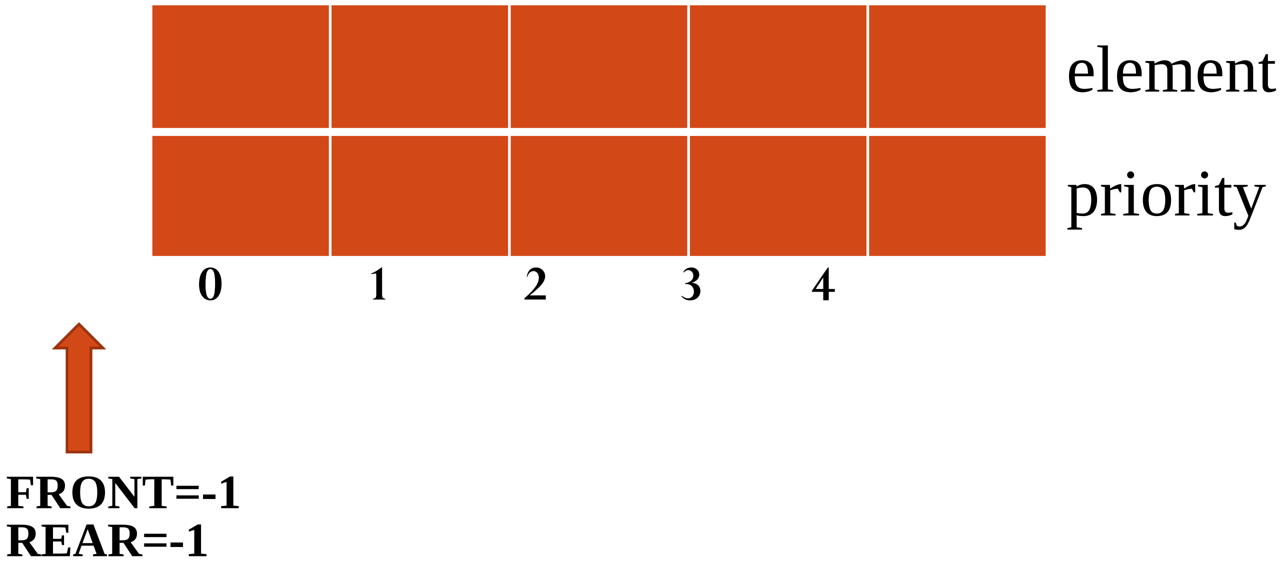
ENQUEUE_PQ(120,4)

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

FRONT=REAR=0

A[REAR].item=120

A[REAR].priority=4



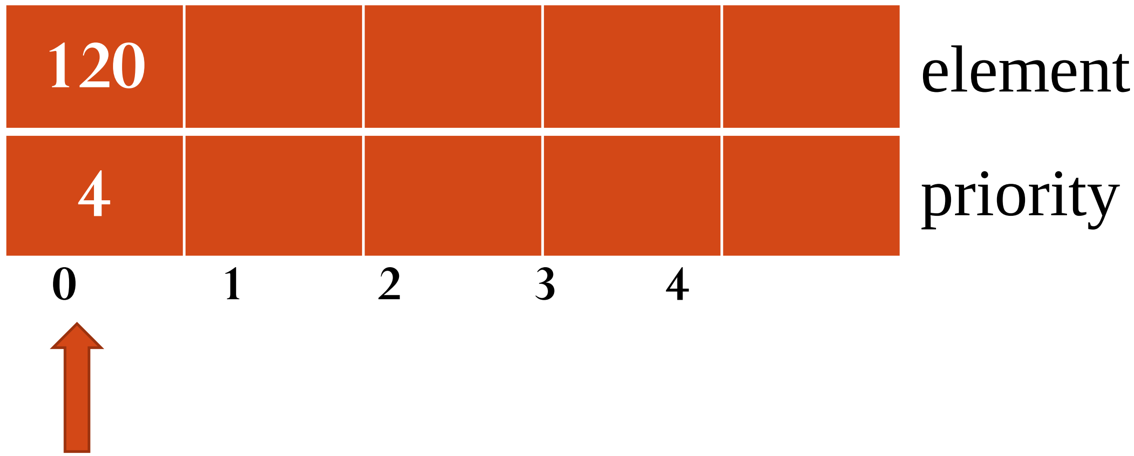
ENQUEUE_PQ(120,4)

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

FRONT=REAR=0

A[REAR].item=120

A[REAR].priority=4

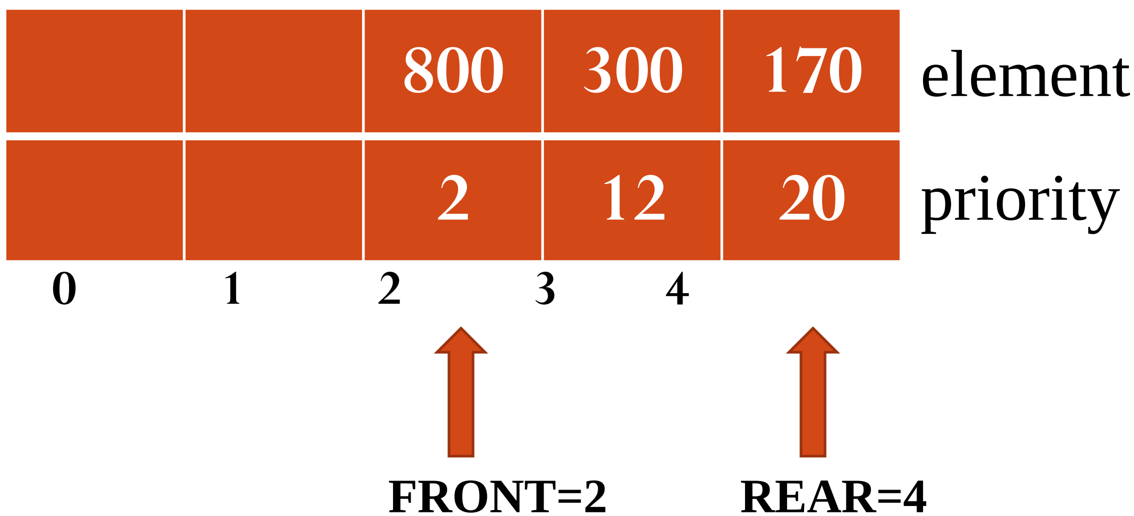


FRONT=0
REAR=0

ENQUEUE_PQ(120,4)

Case 3: **if REAR=SIZE-1**

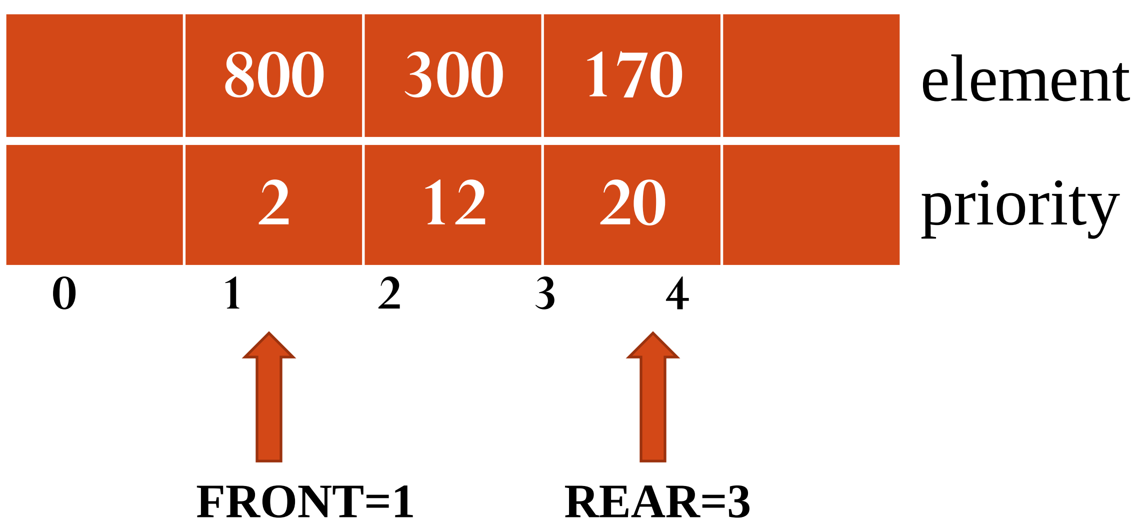
Shift all elements one position to left



ENQUEUE_PQ(120,4)

Case 3: if REAR=SIZE-1

Shift all elements one position to left

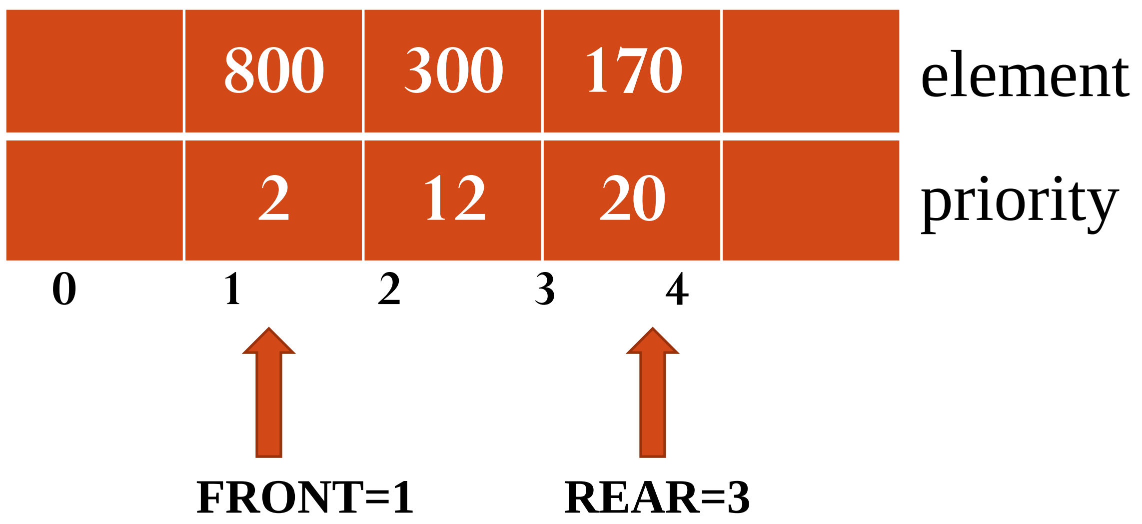


ENQUEUE_PQ(120,4)

Case 3: **if REAR=SIZE-1**

Shift all elements one position to left

Find the location where the new elmt is to be inserted

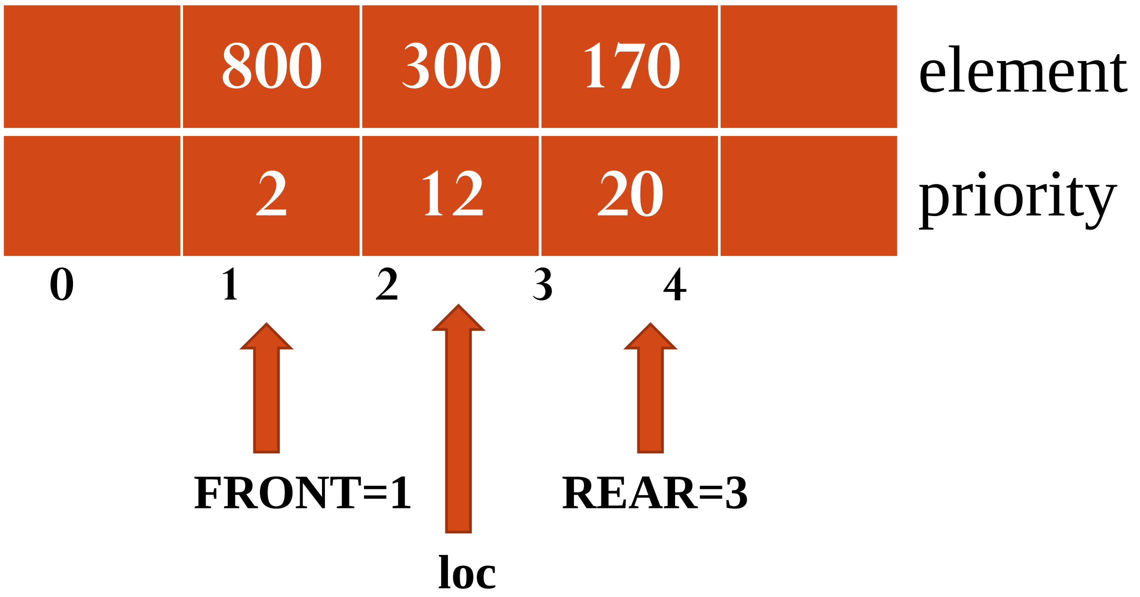


ENQUEUE_PQ(120,4)

Case 3: **if REAR=SIZE-1**

Shift all elements one position to left

Find the location where the new elmt is to be inserted



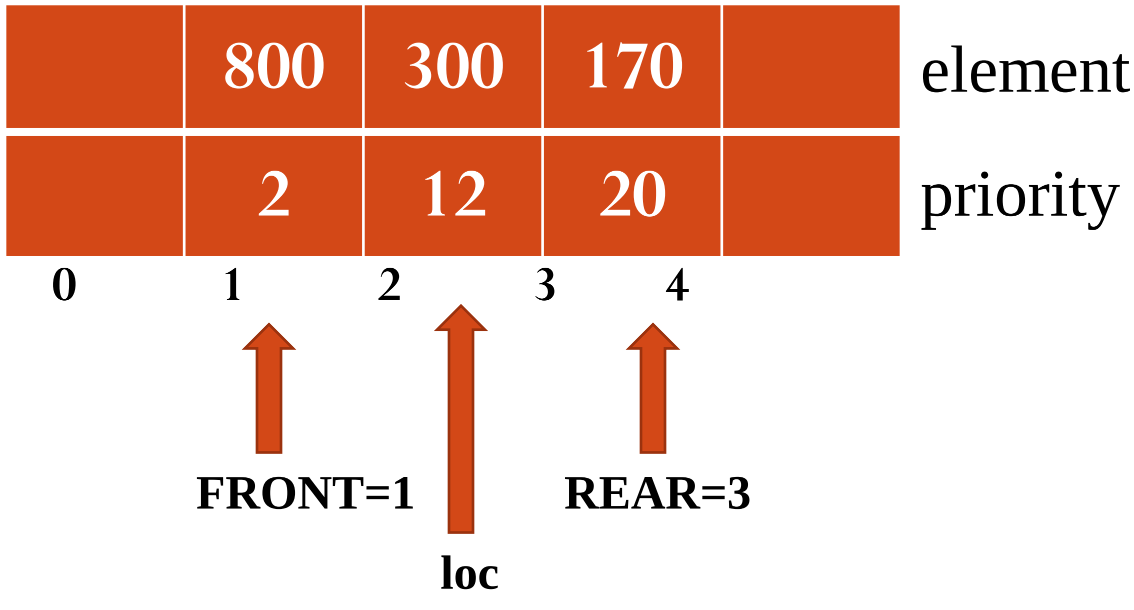
ENQUEUE_PQ(120,4)

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



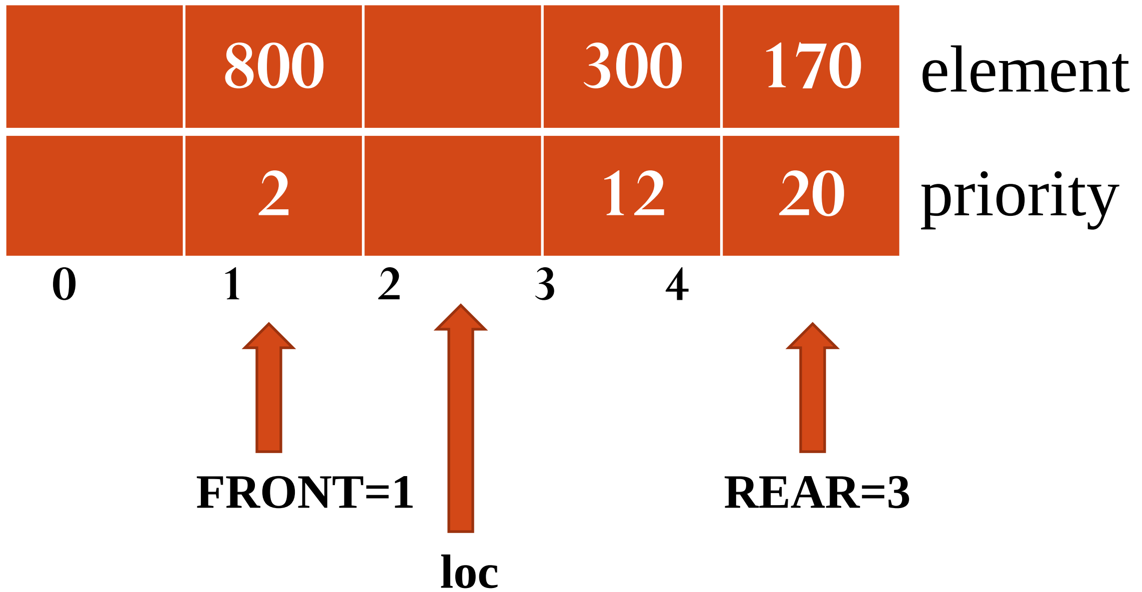
ENQUEUE_PQ(120,4)

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



ENQUEUE_PQ(120,4)

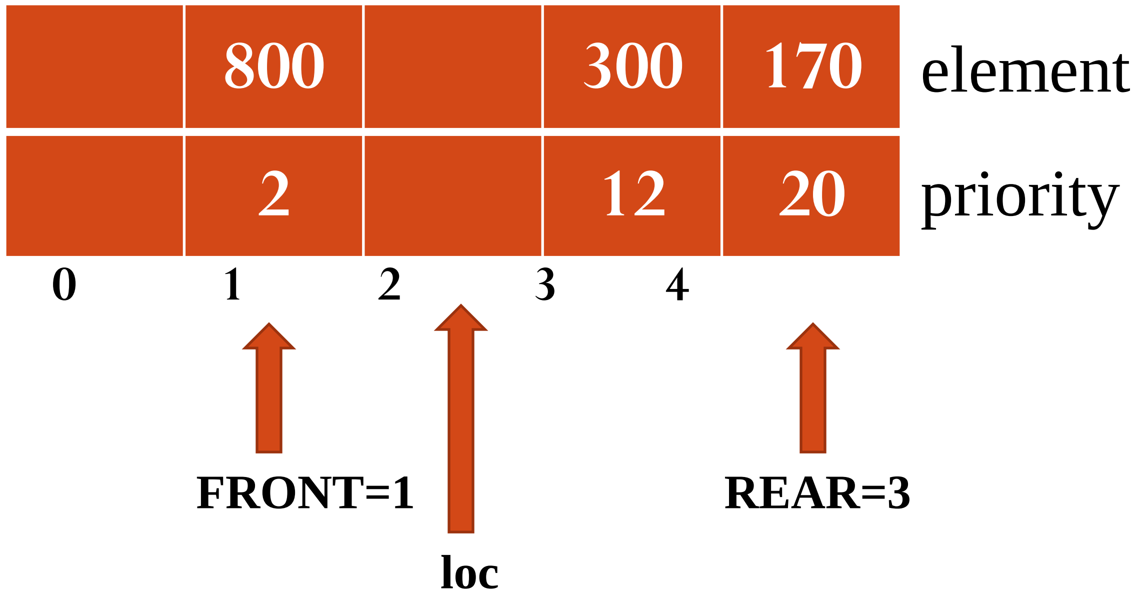
Case 3: if REAR=SIZE-1

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)

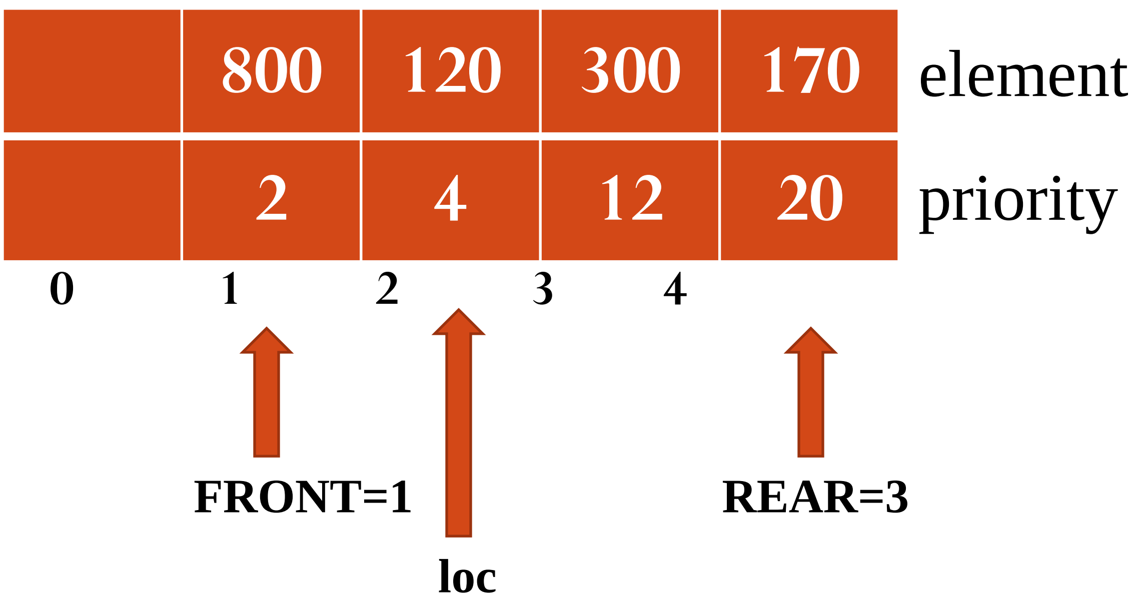
Case 3: **if REAR=SIZE-1**

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

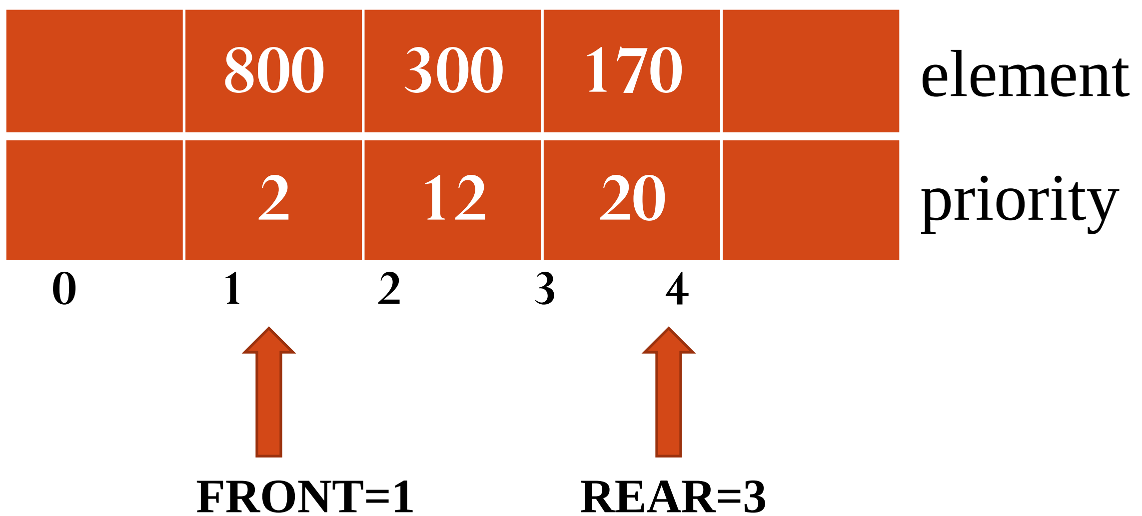
Insert the data at the index loc



ENQUEUE_PQ(120,4)

Case 4: **All other cases**

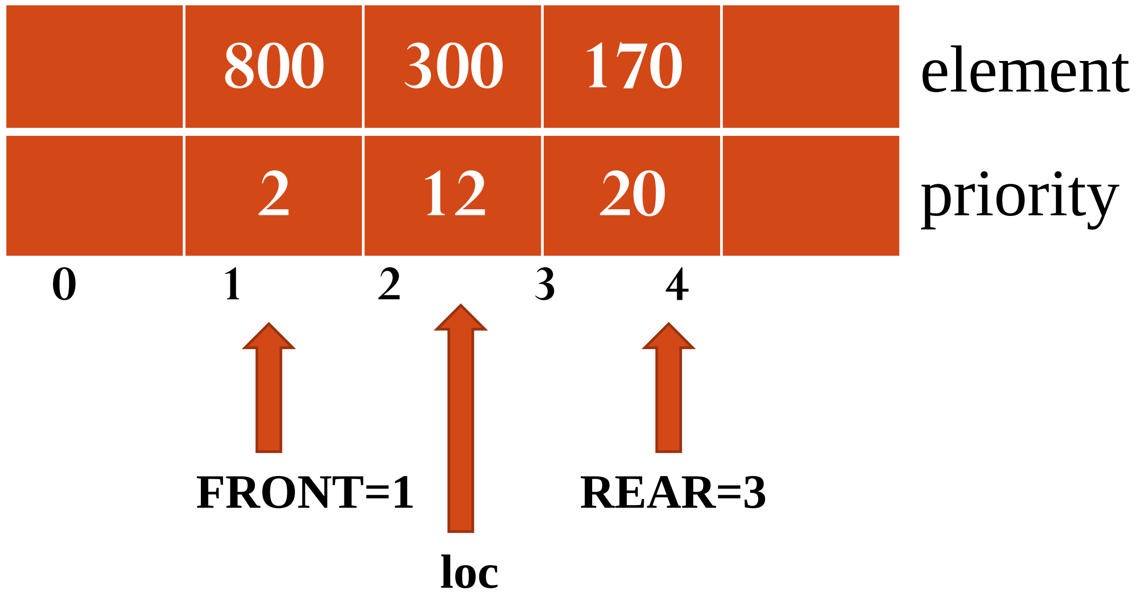
Find the location where the new elmt is to be inserted



ENQUEUE_PQ(120,4)

Case 4: All other cases

Find the location where the new elmt is to be inserted

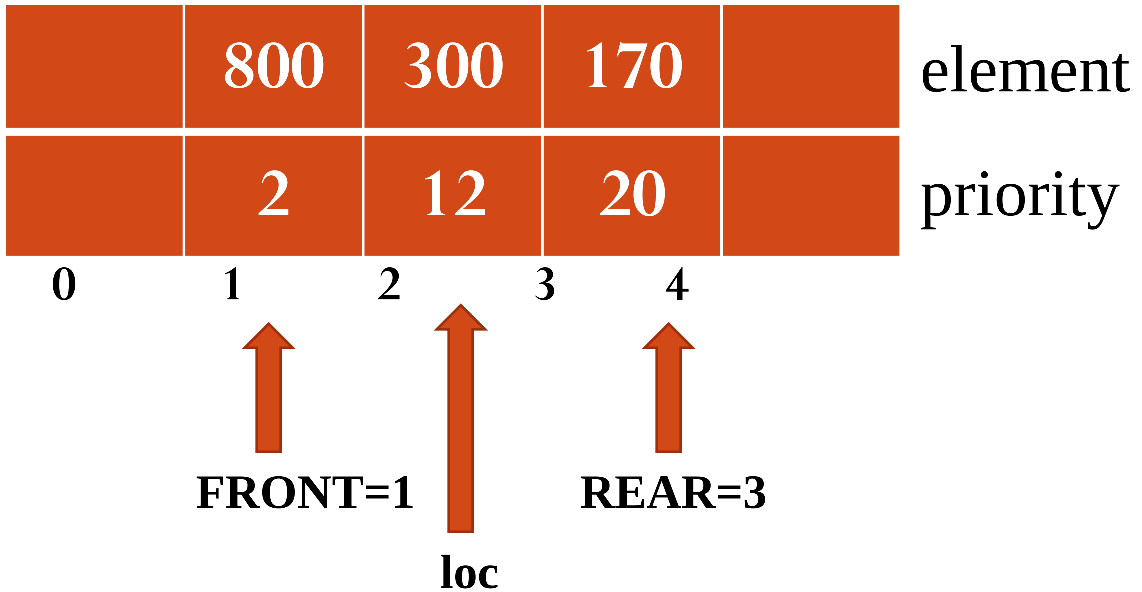


ENQUEUE_PQ(120,4)

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

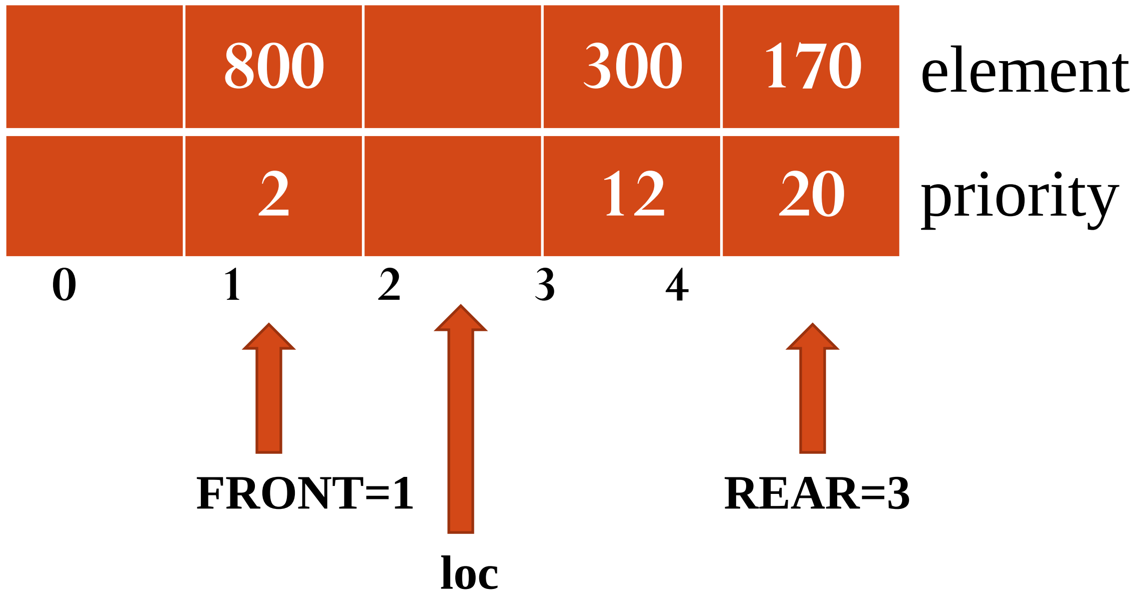


ENQUEUE_PQ(120,4)

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



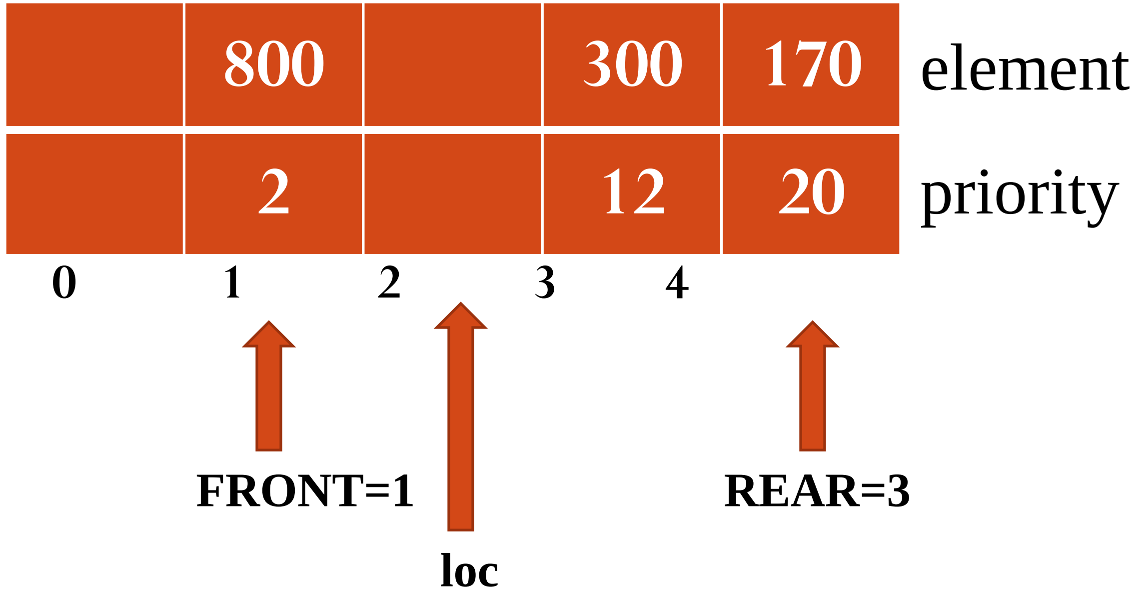
ENQUEUE_PQ(120,4)

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

Insert the data at the index loc



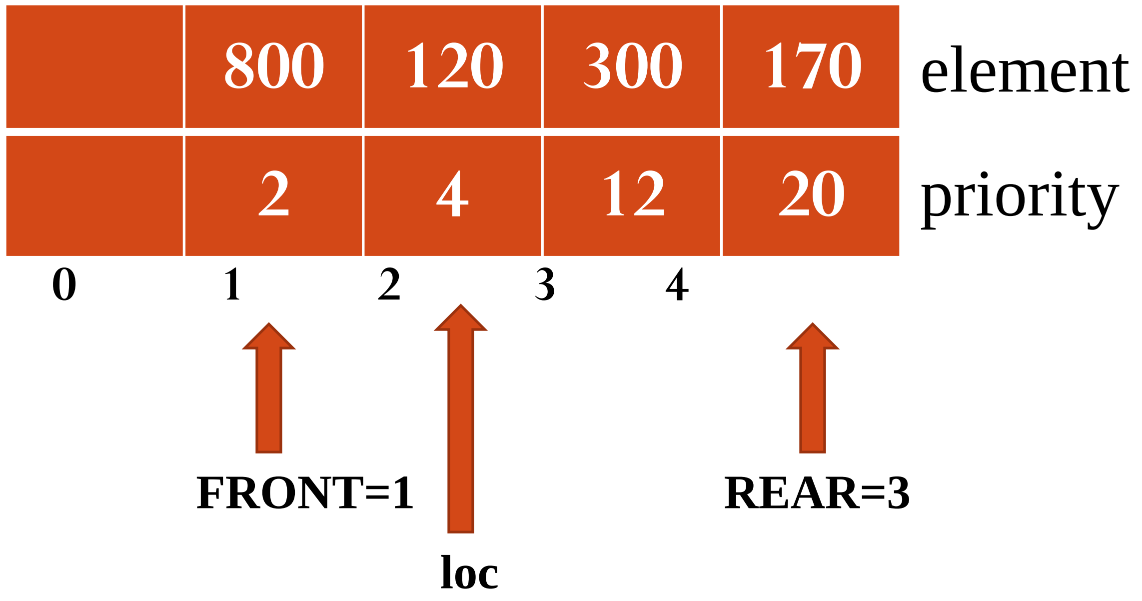
ENQUEUE_PQ(120,4)

Case 4: **All other cases**

Find the location where the new elmt is to be inserted

Shift elements from loc to REAR one position right

Insert the data at the index loc



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
}
```

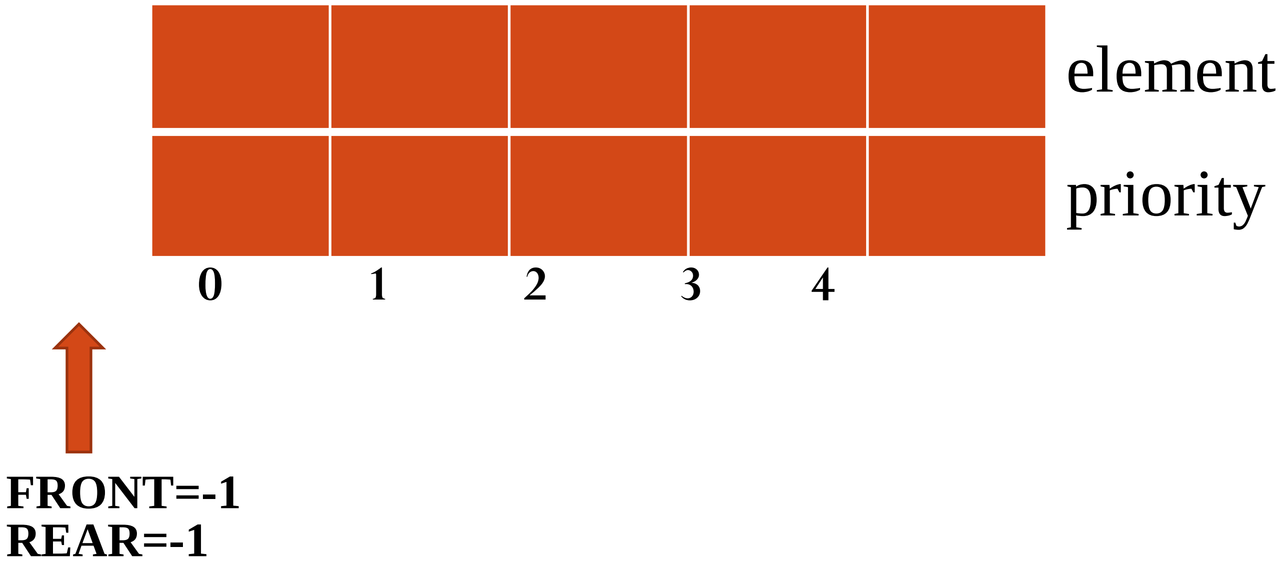
```
        else
        {
                for i=REAR to FRONT do
                {       if A[i].priority<PRIORITY then
                                break;
                }
                loc=i+1
                for i=REAR to loc do
                        A[i+1]=A[i]
                A[loc].item=ITEM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

PRIORITY QUEUE – DEQUEUE

DEQUEUE_PQ()

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

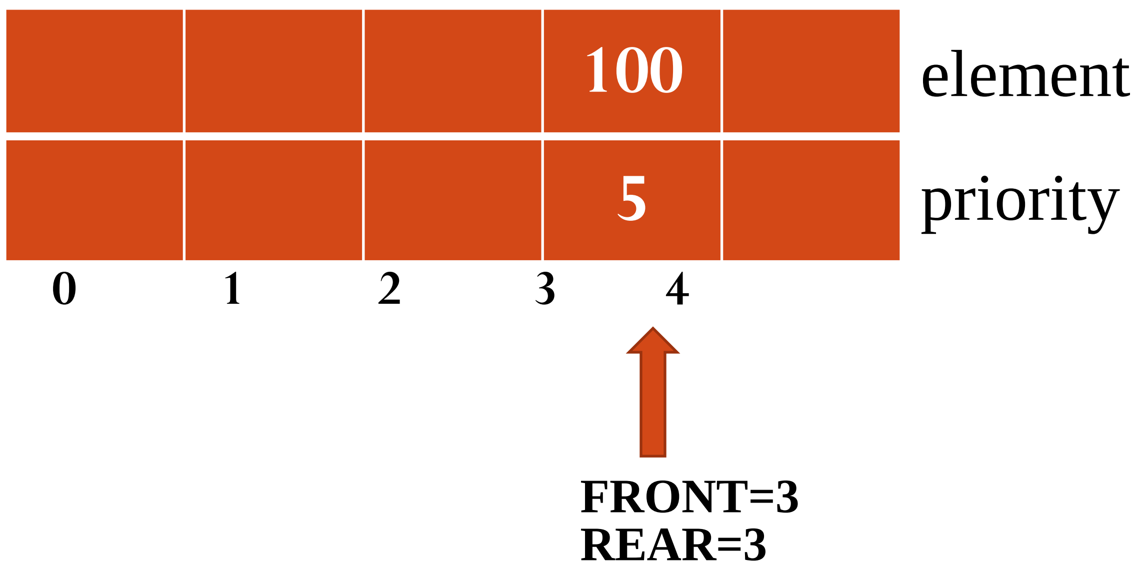
Print "Priority Queue is EMPTY"



DEQUEUE_PQ()

Case 2: **FRONT=REAR**

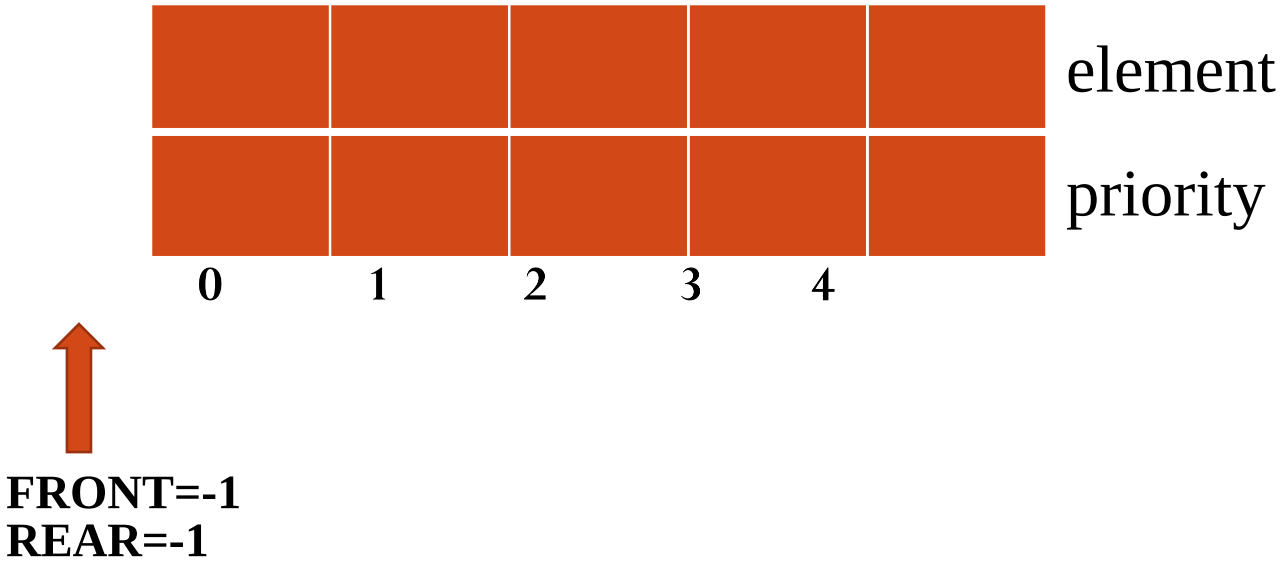
FRONT=REAR=-1



DEQUEUE_PQ()

Case 2: FRONT=REAR

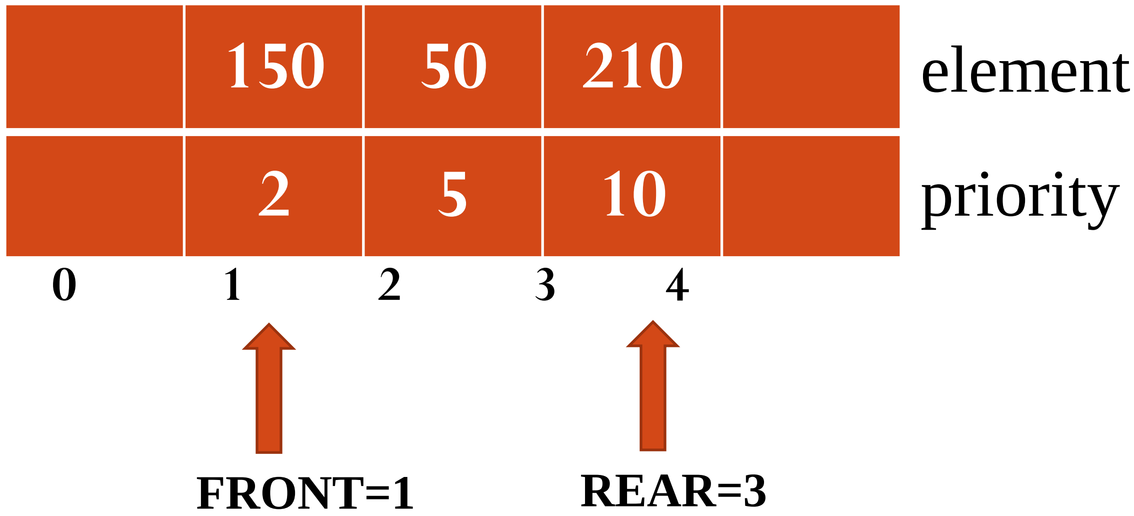
FRONT=REAR=-1



DEQUEUE_PQ()

Case 3: **Queue contains more than one elements**

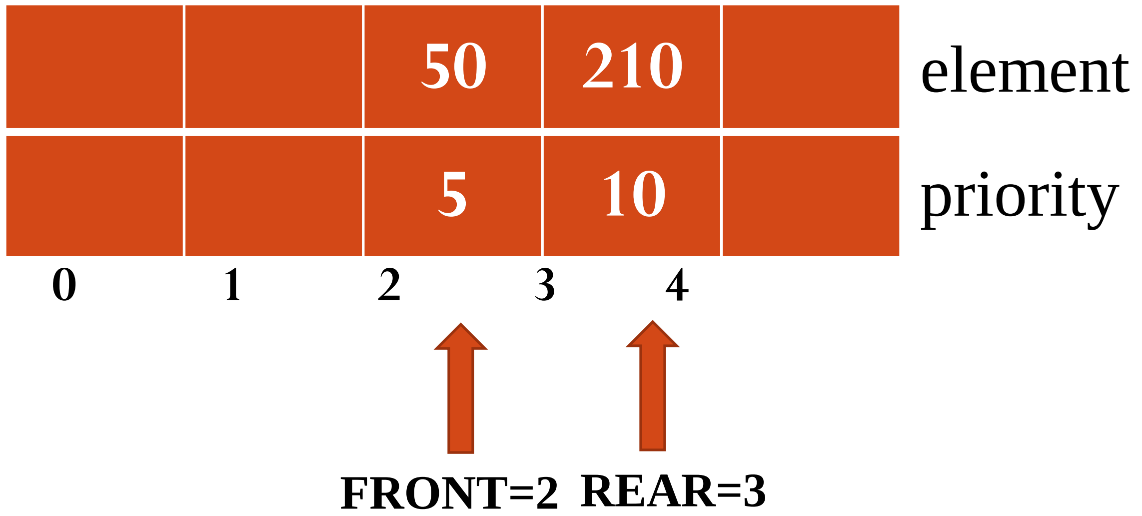
FRONT=FRONT+1



DEQUEUE_PQ()

Case 3: **Queue contains more than one elements**

FRONT=FRONT+1



PRIORITY QUEUE – DEQUEUE Algorithm

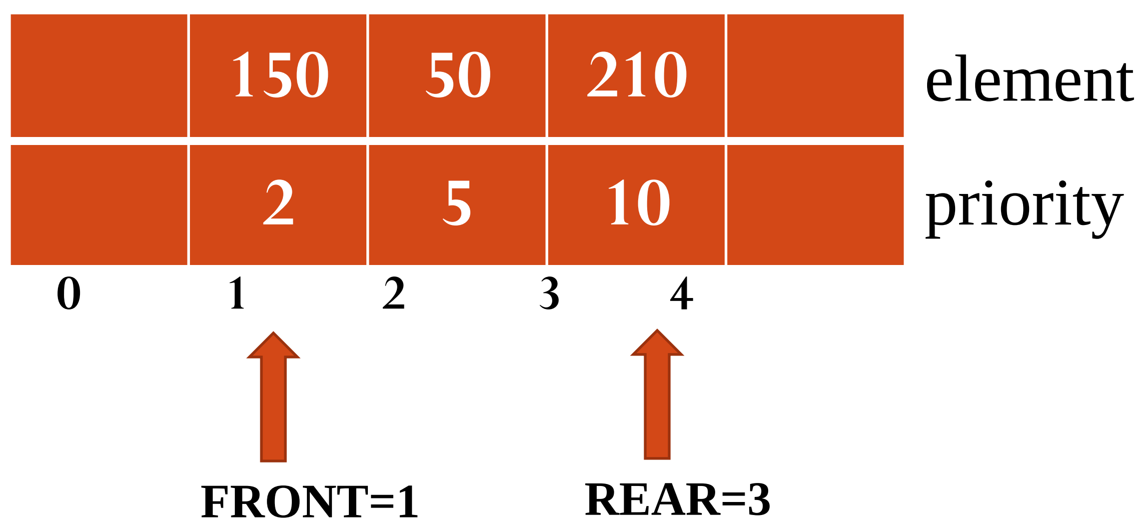
Algorithm DEQUEUE_PQ()

```
{       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 – 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- Various States

1. Priority Queue is Empty: FRONT=-1 & REAR=-1
2. Priority Queue is Full: FRONT = 0 and REAR=SIZE-1
3. Priority Queue contains only one element: FRONT=REAR
4. Total elements in the Priority Queue : REAR-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)

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

PRIORITY QUEUE USING ARRAY-PROGRAM

```
#include<stdio.h>
int size,front,rear;
struct PQ
{       int item,priority;
}A[20];
void display()
{       int i;
        if(front==-1)
                printf("queue is EMPTY");
        else
        {       for(i=front;i<=rear;i++)
                        printf("%d\t",A[i].item);
        }
}
```

```
void enqueue(int ITEM,int PRIORITY)
{
        int i,loc;
        if(front==0 && rear==size-1)
                printf("queue is FULL");
        else if(front==-1)
        {       front=0;
                rear=0;
                A[rear].item=ITEM;
                A[rear].priority=PRIORITY;
        }
```

```
else
{   if(rear==size-1)
    {   for(i=front;i<=rear;i++)
                A[i-1]=A[i];
        front--;
        rear--;
    }
    for(i=rear;i>=front;i--)
    {   if(A[i].priority<PRIORITY)
        {       break;
        }
    }
    loc=i+1;

        for(i=rear;i>=loc;i--)
        {       A[i+1]=A[i];
        }
        A[loc].item=ITEM;
        A[loc].priority=PRIORITY;
        rear++;
    }
}
```

```c
void dequeue()
{       if(front==-1)
                printf("Queue is empty");
        else if(front==rear)
        {
                printf("deleted item is %d",A[front].item);
                front=-1;
                rear=-1;
        }
        else
        {       printf("deleted item is %d",A[front].item);
                front++;
        }
}
```

```
void main()
{
   int opt,item,prio;
   front=-1;
   rear=-1;
   printf("Enter the size of the queue");
   scanf("%d",&size);
   do
   {
          printf("\nEnter the option\n”);
          printf(“1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
          scanf("%d",&opt);
```

```
        switch(opt)
        {       case 1:printf("Enter the item and priority");
                       scanf("%d%d",&item,&prio);
                       enqueue(item,prio);
                       break;
                case 2:dequeue();
                       break;
                case 3:display();
                       break;
                case 4:break;
                default:printf("Enter a valid option");
        }
    }while(opt!=4);
}
```