CHAPTER 3: QUEUE

BY: Er. Krishna Khadka

CONTENT

- Introduction
- Queue as an ADT
- Primitive Operation in Queue
- Linear and Circular Queue and their Application
- Enqueue and Dequeue, Priority Queue

QUEUE - INTRODUCTION

- A QUEUE is logically a First In First Out (FIFO) linear data structure.
- It is a homogeneous collection of elements in which new elements are added at **one end** called **rear**, and the existing elements are deleted from **other end** called **front**.

QUEUE AS AN ADT/ OPERATION ON QUEUE

There are two basic operations/ primitive operation that can be performed on queue.

Enqueue ():

- It refers to the addition of an item in the queue.
- Items are always inserted at the **rear** end of queue
- Whenever we insert a data items the value of rear is increased by 1 i.e. rear = rear+1

Dequeue ():

- It refers to the deletion of an item from the queue
- Item are always deleted from the **front** end of queue
- Whenever an item is deleted from the queue the value of **front** is increased by 1 i.e. front = front+1

QUEUE AS AN ADT/ OPERATION ON QUEUE

- However, some more additional operations that can be performed on queue are:
- **Make Empty (Q) :** Create an empty queue, Q
- **Isempty (Q):** Returns true if the queue, Q, is empty otherwise false.
- **Isfull (Q) :** Returns true if the queue, Q, is full otherwise false.
- **Size (Q) :** Returns the number of items in the queue, Q
- **Front (Q) :** Return the object that is at the front of the queue without removing it.
- **Traverse (Q) :** Visit all the elements stored in the queue, Q
- **Search (K,Q) :** Search for the location of K in queue, Q

IMPLEMENTATION OF QUEUE

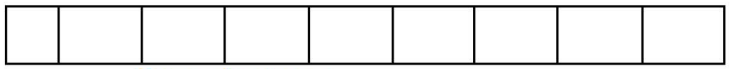
- Static Implementation (Array Implementation)
- Dynamic Implementation (Linked List Implementation)

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.

LINEAR QUEUE

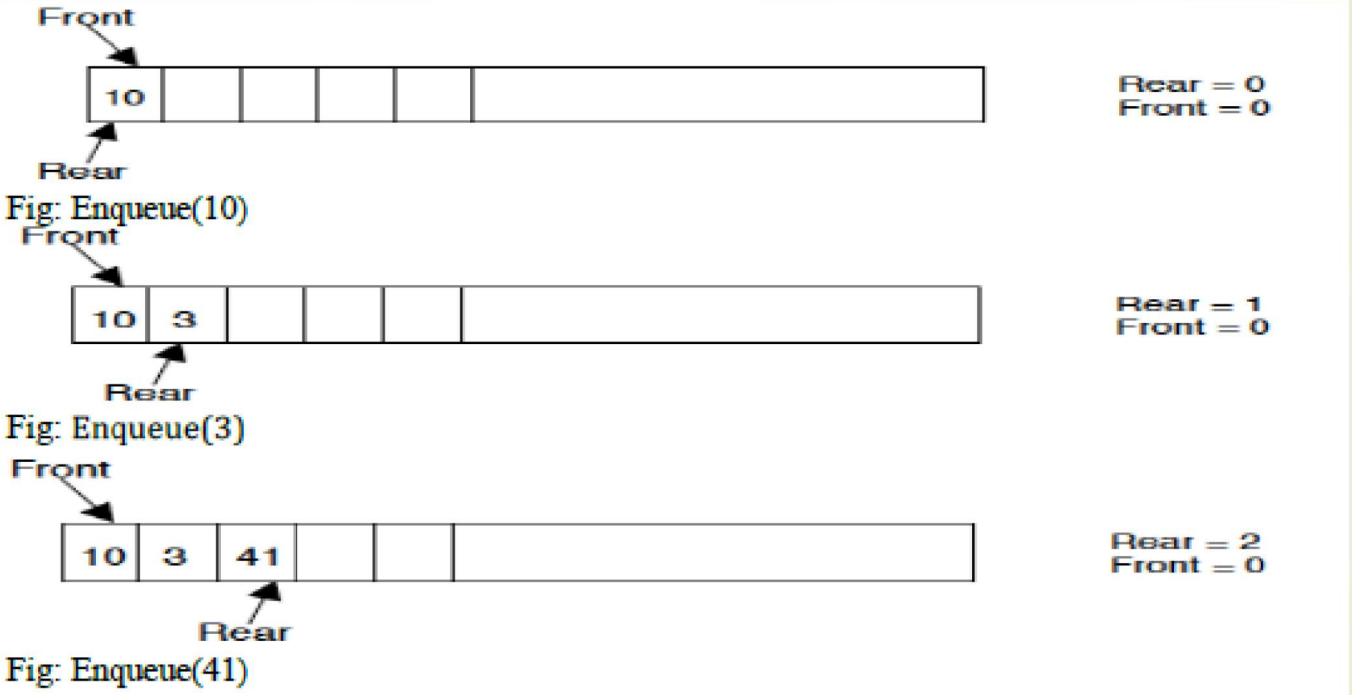
- **Enqueue** operation will insert an element to queue, at the rear end, by incrementing the array index.
- **Dequeue** operation will delete from the front end by decrementing the array index and will assign the deleted value to a variable.
- Initially front and rear is set to -1.
- The queue is empty whenever **rear < front** or both the rear and front is equal to -1.
- Total number of elements in the queue at any time is equal to **rear-front+1,** when implemented using arrays.
- Below are the few operations in the queue.



Queue is empty

rear=-1,front=-1

LINEAR QUEUE



LINEAR QUEUE

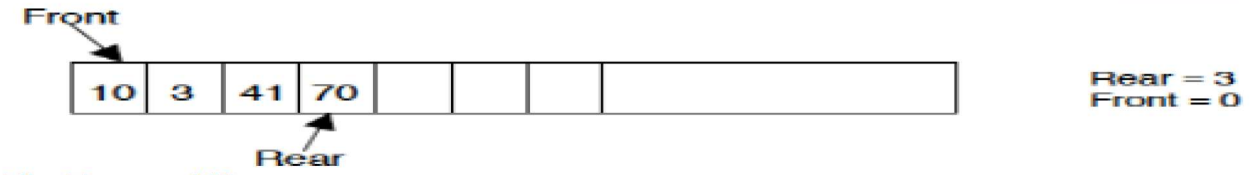


Fig: Enqueue(70)

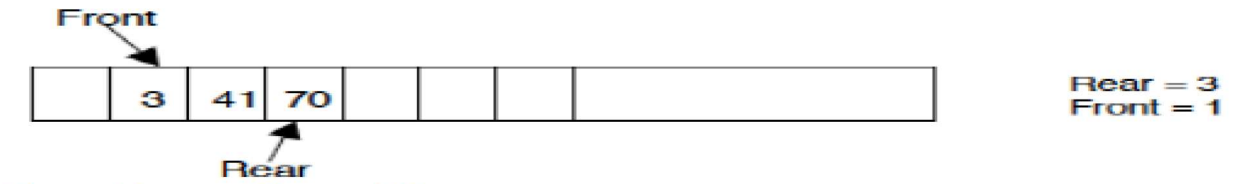


Fig: x = Dequeue()[i.e. x = 10]

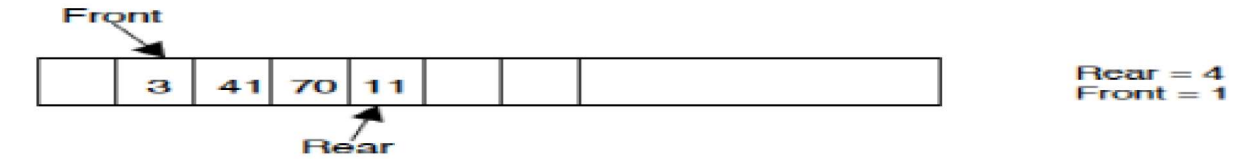


Fig: Enqueue(11)

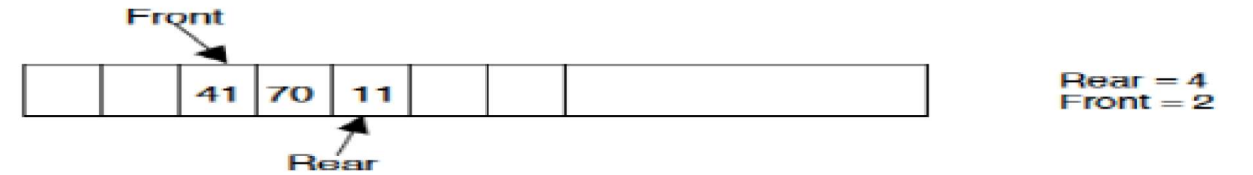


Fig: x = Dequeue()[i.e. x = 3]

LINEAR QUEUE

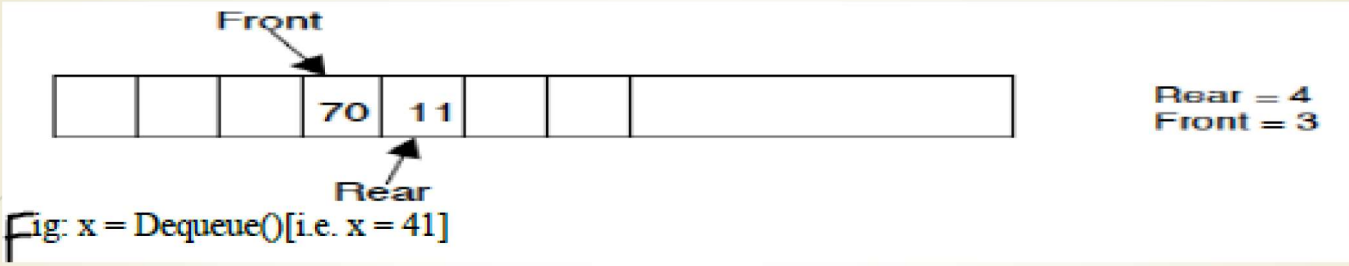


Fig: x = Dequeue()[i.e. x = 41]

- ***Note: During The insertion of first element in the queue, we always increment the front by one.***
- If we try to **dequeue** an element from queue **when it is empty**, underflow occurs.
- If we try to **enqueue** an element to queue , overflow occurs when the **queue is full**.

Overflow Condition: If Rear = MAX-1
Underflow Condition: If front = -1 and rear = -1 (Initial Condition) or rear < front
One Element: If rear = front.
Number of Elements present in a Queue : rear – front +1

LINEAR QUEUE- ALGORITHM FOR QUEUE OPERATIONS

- Let Q be the arrays of some specified size say MAX. **rear** and **front** are two points for element insertion and deletion.

Inserting an element into QUEUE (Enqueue)

1. Input the value to be inserted and assign to variable "data".
2. If (rear>= MAX-1)
   a. Display "Queue Overflow"
   b. Exit
3. Else
   a. If (front == -1 && rear ==-1) [first time insertion]
   b. Front = 0
4. rear =rear +1
5. Q[rear] = data
6. Exit

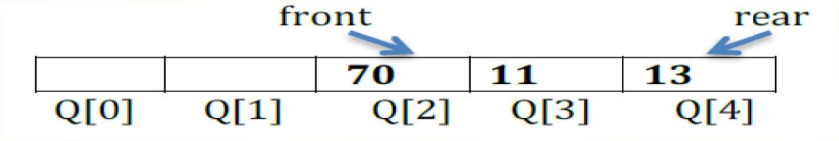
LINEAR QUEUE- ALGORITHM FOR QUEUE OPERATIONS

Deleting an element from QUEUE (Dequeue)

1. If (rear < front or (front == -1 && rear == -1))
   a. Display “Queue is empty”
   b. Exit
2. Else
   a. Data = Q [front]
3. front = front +1
4. Exit

CIRCULAR QUEUE

- Suppose a queue has maximum size 5, say 5 elements pushed and 2 elements popped.



- Now if we attempt to add more elements, even though 2 queue cells are free, the elements cannot be pushed.
- Because in a queue, elements are always inserted at the rear end and hence rear points to last location of the queue which indicates queue full.
- This limitation can be overcome if we use circular queue.
- In circular queues the elements Q[0], Q[1], Q[2],……, Q[n-1] is represented in a **circular fashion**.
- 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 at the queue is full.

CIRCULAR QUEUE

- Suppose Q is a queue array of 6 elements. Enqueue() and Dequeue() operation can be performed on circular. The following figure will illustrate the same.

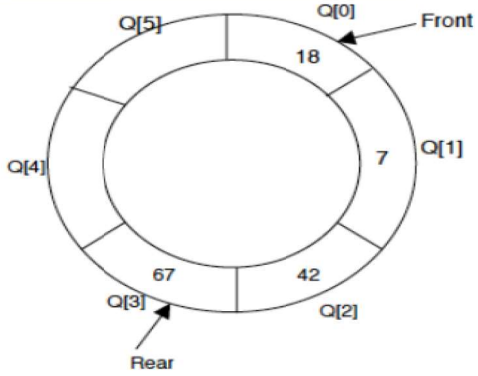


Fig1: A Circular Queue After inserting 18,7,42,67.

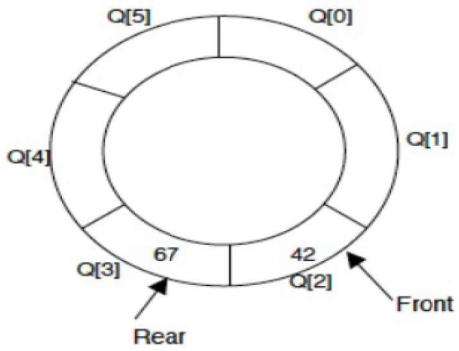


Fig2: Circular Queue after popping 18,7.

- After inserting an element at last location Q[5], the next element will be inserted at the very first location (i.e., Q[0]) that is circular queue is one in which the first element comes just after the last element.

CIRCULAR QUEUE

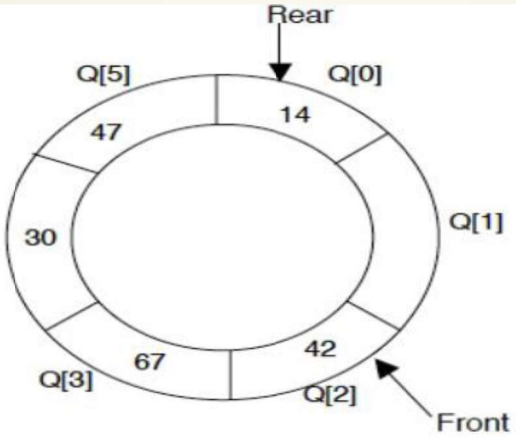


Fig3: Circular Queue after pushing 30, 47, 14

- At any time the relation will calculate the position of the element to be inserted.
  - **rear = (rear+1) % MAX** **[MAX = size]**
- After deleting an element from circular queue the position of the front end is calculated by the relation
  - **front = (front + 1) % MAX**

ALGORITHM FOR CIRCULAR QUEUE

- Let Q be the array of some specified size say **MAX**. **front** and **rear** are two pointers where the elements are deleted and inserted. **DATA** is the element to be inserted. Initially front=-1 and rear=-1.

Inserting an element to circular queue:

1. if ((front ==0 && rear ==MAX-1)

   OR front =rear+1)

   a. Display " Queue is Full"

   b. Exit

2. If (front == -1 && rear == -1)

   a. front =0

   b. rear = 0

3. else

   a. **Rear = (rear+1) % MAX**

4. Input the value to be inserted and assign to variable "DATA"

5. Q[rear] = DATA

6. Repeat steps 2 to 5 if we want to insert more elements

7. Exit.

ALGORITHM FOR CIRCULAR QUEUE

Deleting an element from a circular queue:

1. if (front ==-1 && rear ==-1)
   a. Display " Queue is Empty"
   b. Exit
2. Else
   a. DATA = Q[front]
3. If (rear==front)
   a. front = -1
   b. rear = -1
4. Else
   a. **front = (front + 1) % MAX**
5. Repeat steps 1 t0 4 if we want to delete more elements
6. Exit.

DEQUES

- A deque is a homogeneous list in which elements can be added or inserted (called enqueue operation) and deleted or removed (which is called dequeue operation) from both the ends.
- That is, 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**.

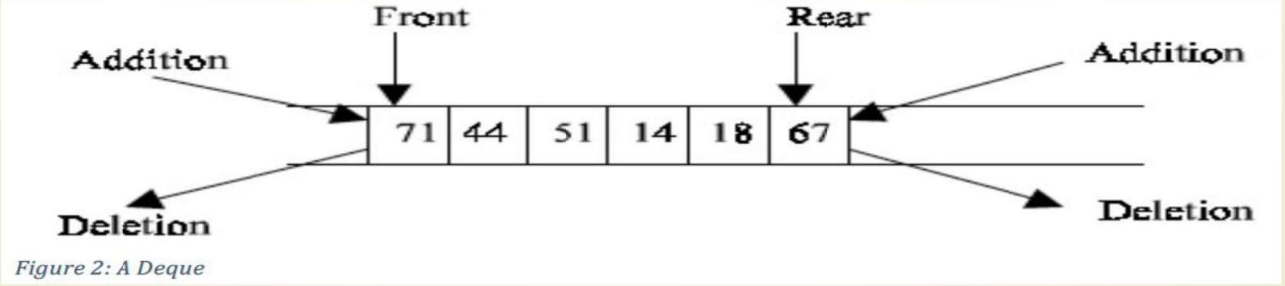


Figure 2: A Deque

DEQUES

- 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 one 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 end of the lists.

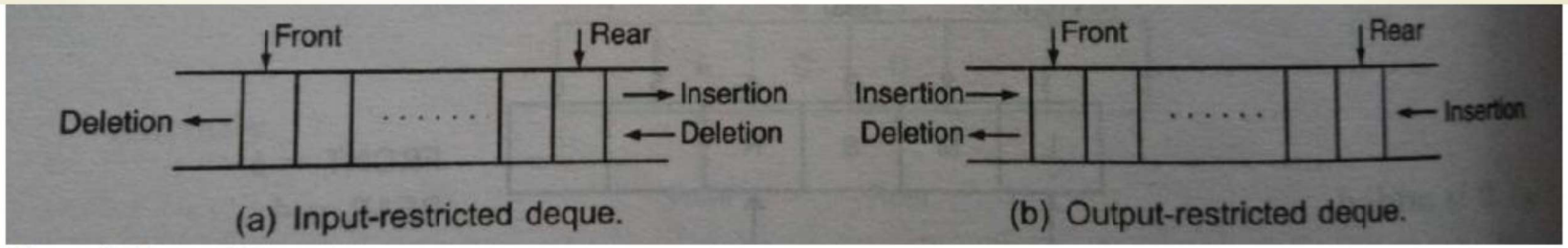


Figure 3: Types of Deques:

DEQUES

- The possible operation performed on deque is : (Deque ADT)
  - Add an element at the rear end (insert_rear)
  - Add an element at the front end (insert_front)
  - Delete an element from the front end (delete_front)
  - Delete an element from the rear end (delete_rear)
- Only 1st, 3rd and 4th operations are performed by input-restricted deque and 1st, 2nd and 3rd operations are performed by output – restricted deque.

ALGORITHM FOR INSERTING AN ELEMENT IN DEQUES

- Let Q be the queue of size **MAX** . **front** and **rear** are two pointers where the addition and deletion of elements occurred. Let **DATA** be the element to be inserted. Initially front == -1 and rear == -1.

Insert an element at the rear end of the deque:

1. Input DATA to be inserted
2. If ((front ==0 && rear==MAX-1)

   OR (front==rear+1)

   a. Display "Queue Full"

   b. Exit
3. If (front == -1 && rear == -1)

   a. Front =0

   b. Rear =0
4. Else

   a. If (rear !=MAX-1)

      i. rear=rear+1

   b. else

      i. rear=0
5. Q[rear] = DATA
6. Exit

ALGORITHM FOR INSERTING AN ELEMENT IN DEQUES

Insert an element at the front end of the deque:

1. Input DATA to be inserted
2. If ((front ==0 && rear==MAX-1) OR (front==rear+1)
   a. Display "Queue Full"
   b. Exit
3. If (front == -1 && rear == -1)
   a. Front =0
   b. Rear =0
4. Else
   a. **If (front ==0)**
      i. front=MAX-1
   a. else
      i. front=front-1
5. Q[front] = DATA
6. Exit

ALGORITHM FOR DELETING AN ELEMENT IN DEQUES

- Let Q be the queue of size **MAX** . **front** and **rear** are two pointers where the addition and deletion of elements occurred. Let **DATA** will contain the element just deleted. Initially front == -1 and rear == -1.

Delete an element from the rear end of the deque:

1. If (front == -1 && rear == -1)
   a. Display "Queue Underflow"
   b. Exit
2. DATA = Q[rear]
3. **If (rear ==0)**
   a. **rear = MAX-1**
4. Else
   a. **rear = rear-1**
5. Exit

ALGORITHM FOR DELETING AN ELEMENT IN DEQUES

Delete an element from the front end of the deque:

1. If (front == -1 && rear == -1)
    a. Display "Queue Underflow"
    b. Exit
2. DATA = Q[rear]
3. If (front ==MAX-1)
    a. front = 0
4. Else
    a. front = front + 1
5. Exit

PRIORITY QUEUES

- Priority queue is a queue where each element is assigned a priority.
- In priority queue, the elements are deleted and processed by following rules.
  - An element of higher priority is processed before any element of lower priority
  - Two elements with the same priority are processed according to the order in which they were inserted to the queue.
- For example, Consider a manager who is in process of checking and approving files in a first come first basis. In between, if any urgent file (with a high priority) comes, he will process the urgent file next and continue with the other low urgent files.

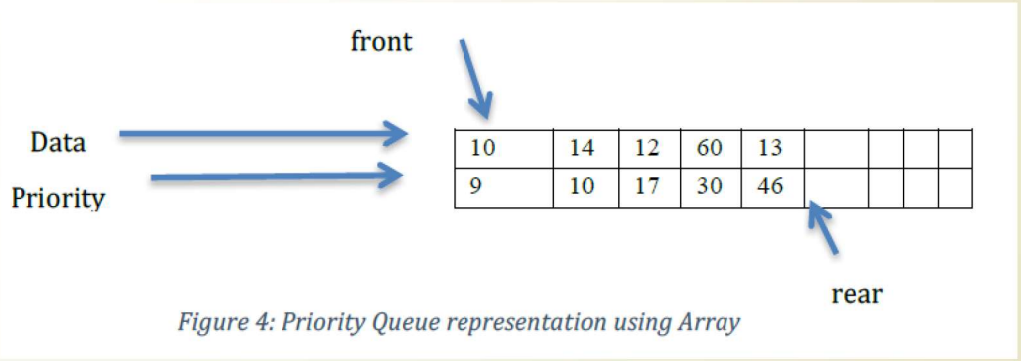


Figure 4: Priority Queue representation using Array

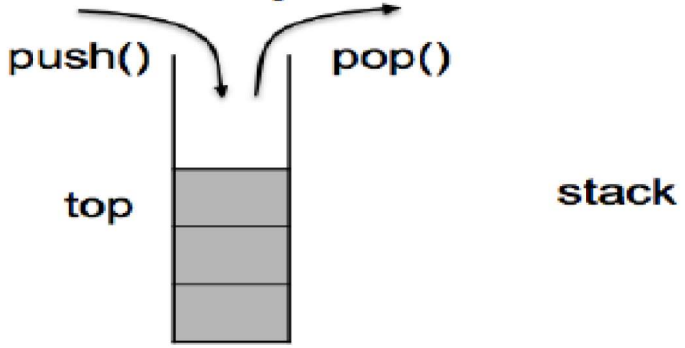
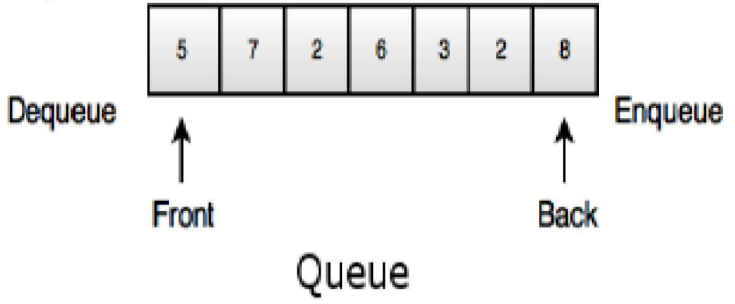
PRIORITY QUEUES

- Above figure gives the pictorial representation of priority queue using arrays after adding 5 elements with its corresponding priorities.
- Here the priorities of data are in ascending order.
- Always we may not be pushing the data in an ascending order.
- From the mixed priority list it is difficult to find the highest priority element if the priority queue is implemented using arrays.
- It is better to implement the priority queue using linked list where a node can be inserted at anywhere in the list.

APPLICATION OF QUEUES

- Round robin techniques for processor, scheduling is implemented using queue.
- Printer server routines (in drivers) are designed using queues.
- All type of customer service type software (e.g. Ticket reservation) are designed using queue to give proper service to the customers.
- When a resource is shared among multiple consumers. Example includes CPU scheduling, Disk Scheduling
- Scheduler (e.g. in operating system): maintains a queue of processes awaiting a slice of machine time

STACK VS QUEUES

SN	Stack	Queue
1	Stack is an ordered list where in all insertions and deletions are performed at the one end called top. 	Queue is an ordered list where in insertions are performed at one end called rear and deletions are performed at another end called front. 
2	Stacks follow Last In First Out (LIFO) order.	Queues following First In First Out (FIFO) order.
3	Stack operations are called push and pop.	Queue operations are called enqueue and dequeue.
4	Associated with stack there is one variable called top.	Associated with queues there are two variables called front and rear.
5	Stack is full can be represented by the condition, Top = MAX-1	Queue is full can be represented by the condition, rear = MAX-1.
6	Stack is empty is represented by the condition, Top = -1	Queue is Empty is represented by the condition, front = -1 and rear = -1
7	To insert an element into the stack top is incremented by 1.	To insert an element into the queue rear is incremented by 1.
8	To delete an element from the stack top is decremented by 1.	To delete an element from the queue front is incremented by 1.
9	Collection of dinner plates at a wedding reception is an example of stack.	People standing in a file to board a bus is an example of queue.

Thank you