Basic Data Structures

Unit 4 : Linear Data Structures II

Prepared by :
Prof. Khushbu Chauhan
Computer Engg. Dept.

MPSTME, NMIMS

Outlines

- Introduction to Queues
- Array representation of Queues
- Queue operations
- Types of Queues: Linear Queues, Circular Queues, Priority Queues
- Applications of Queues

Introduction of Queue

- A Queue is a FIFO (First-In, First-Out) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the REAR and removed from the other end called the FRONT.
- Queues can be implemented by using either arrays or linked lists. In this section, we will see how queues are implemented using each of these data structures

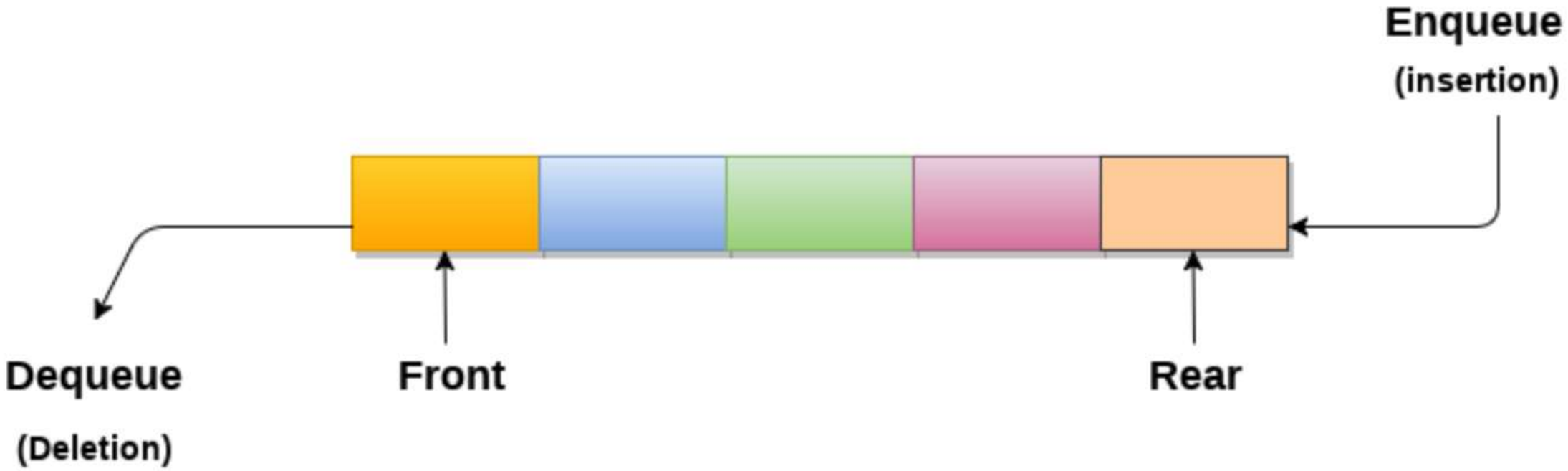
- People moving on an escalator. The people who got on the escalator first will be the first one to step out of it.
- People waiting for a bus. The first person standing in the line will be the first one to get into the bus.
- People standing outside the ticketing window of a cinema hall. The first person in the line will get the ticket first and thus will be the first one to move out of it.
- Luggage kept on conveyor belts. The bag which was placed first will be the first to come out at the other end.
- Cars lined at a toll bridge. The first car to reach the bridge will be the first to leave

Array representation of Queues

- Queues can be easily represented using linear arrays. As stated earlier, every queue has front and rear variables that point to the position from where deletions and insertions can be done, respectively.

Operations on queue

- **Enqueue (Insert)**: Adds an element to the rear of the queue.
- **Dequeue (Delete)**: Removes and returns the element from the front of the queue.
- **Peek**: Returns the element at the front of the queue without removing it.
- **Empty**: Checks if the queue is empty.
- **Full**: Checks if the queue is full.


Enqueue
(insertion)
Dequeue
(Deletion)
Front
Rear

Queue operations

isEmpty Operation

```
// This function will check whether
// the queue is empty or not:
bool isEmpty()
{
    if (front == -1)
        return true;
    else
        return false;
}
```

isFull Operation

```
// This function will check
// whether the queue is full or not.
bool isFull()
{
    if (front == 0 && rear == MAX_SIZE - 1) {
        return true;
    }
    return false;
}
```

Queue operations

- **Enqueue Operation**

```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto step 4
        [END OF IF]
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] = NUM
Step 4: EXIT
```

Dequeue Operation

```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

- However, before inserting an element in a queue, we must check for overflow conditions. An overflow will occur when we try to insert an element into a queue that is already full. When REAR = MAX – 1, where MAX is the size of the queue, we have an overflow condition. Note that we have written MAX – 1 because the index starts from 0.
- Similarly, before deleting an element from a queue, we must check for underflow conditions. An underflow condition occurs when we try to delete an element from a queue that is already empty. If FRONT = –1 and REAR = –1, it means there is no element in the queue.

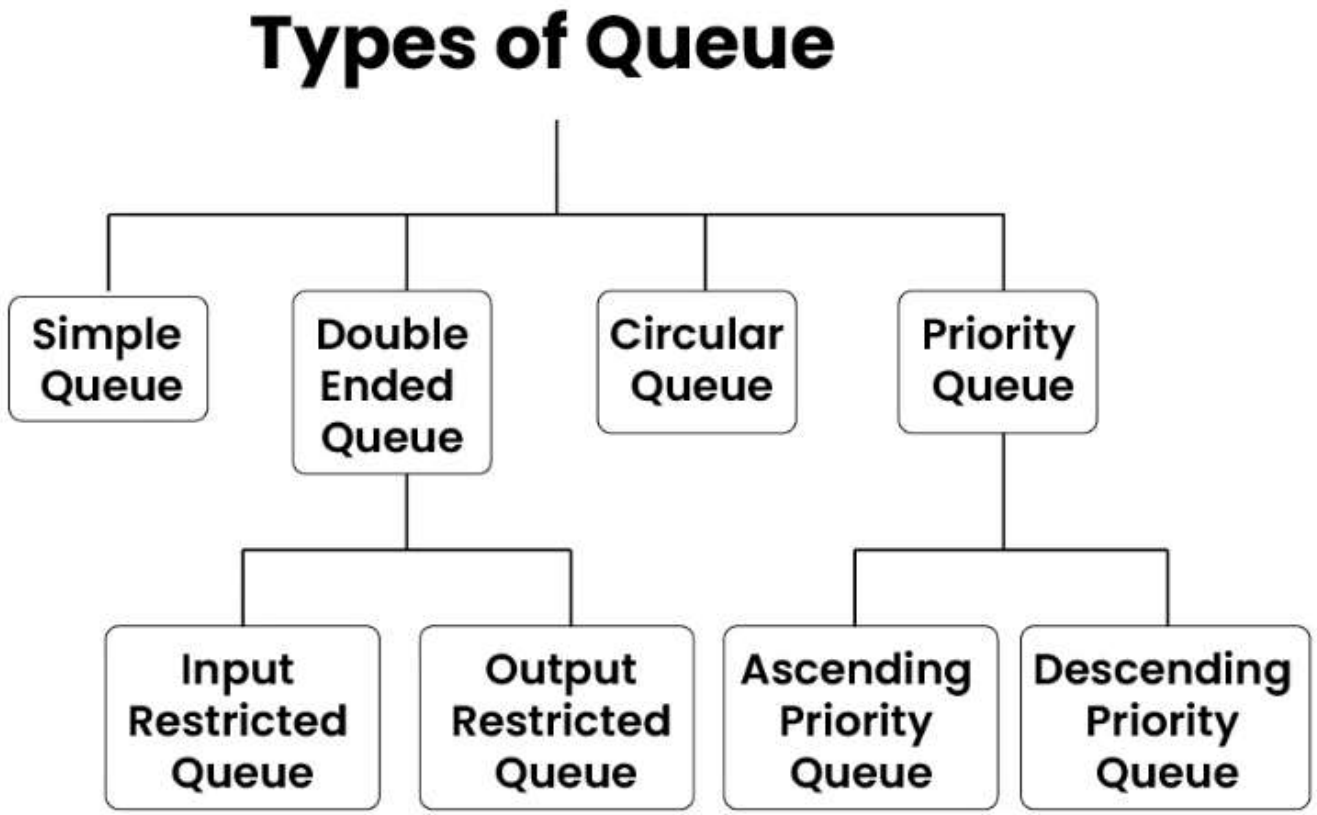
Example

Perform Queue operations to insert maximum 8 elements in an array.

1. Check underflow and overflow condition. Return positions of front and rear.
2. Perform enqueue operation. Return positions of front and rear.
3. Perform dequeue operation. Return positions of front and rear.
4. Display elements of queue in array.

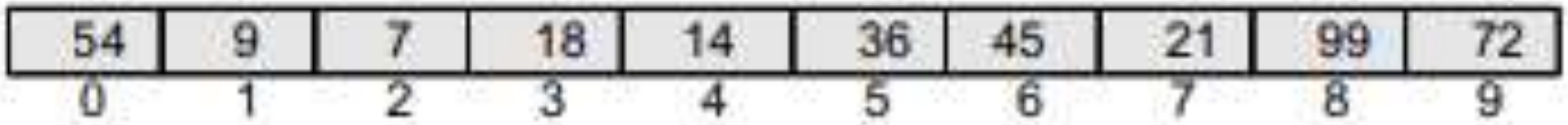
Types of Queues

- A queue data structure can be classified into the following types:



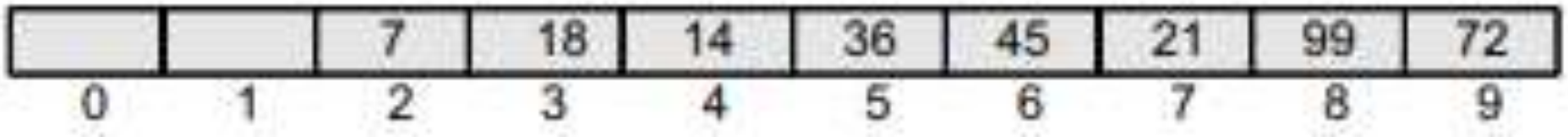
Circular Queue

- In linear queues, the insertions can be done only at one end called the REAR and deletions are always done from the other end called the FRONT. Look at the queue shown in below:



- Here, FRONT = 0 and REAR = 9.

- if you want to insert another value, it will not be possible because the queue is completely full. There is no empty space where the value can be inserted. Consider a scenario in which two successive deletions are made. The queue will then be given as below:



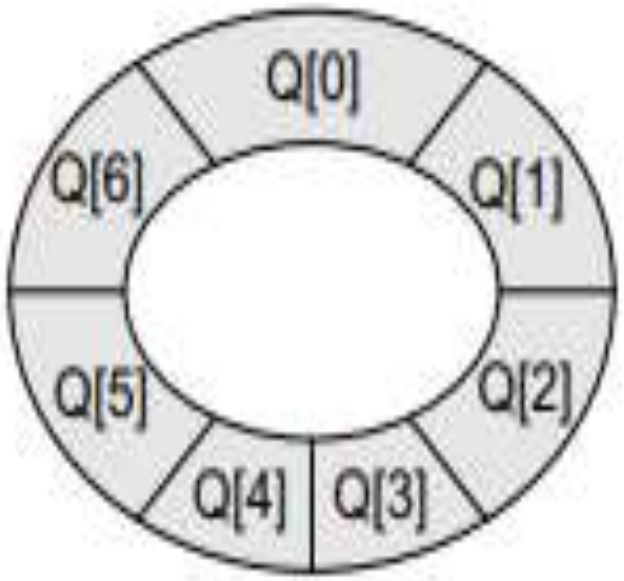
Queue after two successive deletions

front = 2 and REAR = 9

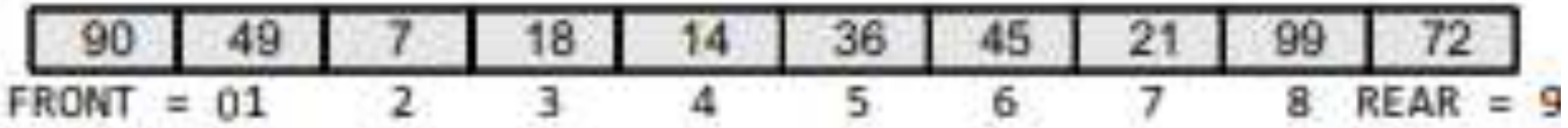
- Suppose we want to insert a new element in the queue. Even though there is space available, the overflow condition still exists because the condition rear = MAX – 1 still holds true. This is a major drawback of a linear queue.
- To resolve this problem, we have two solutions. First, shift the elements to the left so that the vacant space can be occupied and utilized efficiently. But this can be very time-consuming, especially when the queue is quite large. The second option is to use a circular queue. In the circular queue, the first index comes right after the last index.

Circular Queue

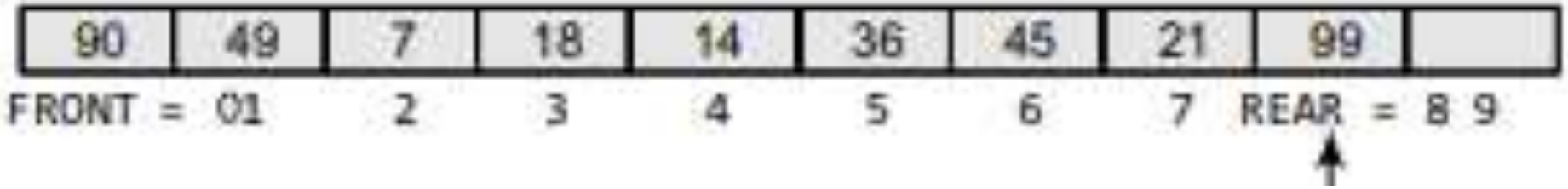
- The circular queue will be full only when front = 0 and rear = Max – 1. A circular queue is implemented in the same manner as a linear queue is implemented. The only difference will be in the code that performs insertion and deletion operations. For insertion, we now have to check for the following three conditions:



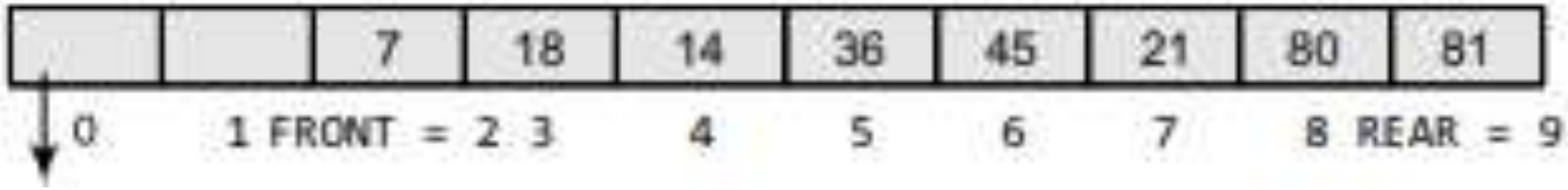
- If front = 0 and rear = MAX – 1, then the circular queue is full.



- If rear != MAX – 1, then rear will be incremented and the value will be inserted.



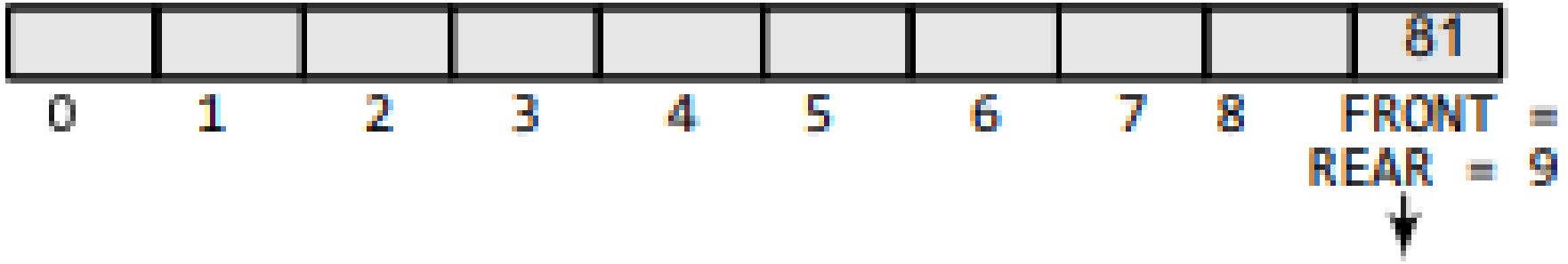
- If front != 0 and rear = MAX – 1, then it means that the queue is not full. So, set rear = 0 and insert the new element there.

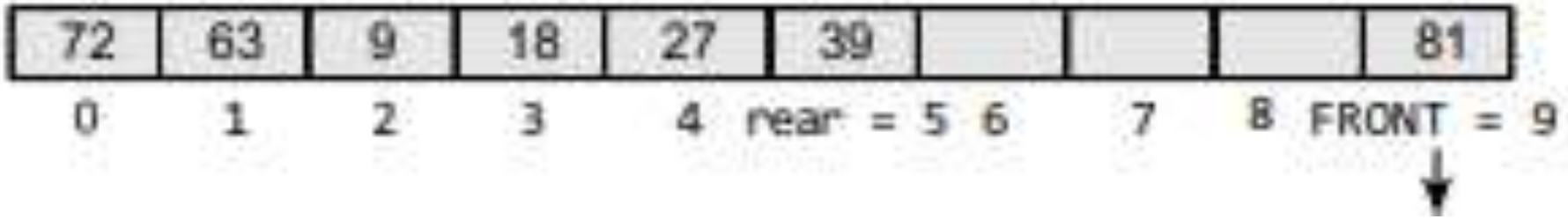


- Algorithm to insert an element in a circular queue

```
Step 1: IF FRONT = 0 and Rear = MAX - 1
            Write "OVERFLOW"
            Goto step 4
        [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1 and FRONT != 0
            SET REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```


0 1 2 3 4 5 6 7 8 9
FRONT = REAR = -1


81
0 1 2 3 4 5 6 7 8
FRONT =
REAR = 9


72 63 9 18 27 39 81
0 1 2 3 4 rear = 5 6 7 8 FRONT = 9

- Algorithm to delete an element from a circular queue

```
Step 1: IF FRONT = -1
            Write "UNDERFLOW"
            Goto Step 4
        [END of IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT = REAR
            SET FRONT = REAR = -1
        ELSE
            IF FRONT = MAX -1
                  SET FRONT = 0
            ELSE
                  SET FRONT = FRONT + 1
            [END of IF]
        [END OF IF]
Step 4: EXIT
```

Priority Queue

- A priority queue is a data structure in which each element is assigned a priority. The priority of the element will be used to determine the order in which the elements will be processed. The general rules of processing the elements of a priority queue are
- An element with higher priority is processed before an element with a lower priority.
- Two elements with the same priority are processed on a first-come-first-served (FCFS) basis.

- A priority queue can be thought of as a modified queue in which when an element has to be removed from the queue, the one with the highest-priority is retrieved first.
- The priority of the element can be set based on various factors. Priority queues are widely used in operating systems to execute the highest priority process first.
- The priority of the process may be set based on the CPU time it requires to get executed completely.

Operations on Priority Queue

1) Insertion in a Priority Queue

- When a new element is inserted in a priority queue, it moves to the empty slot from top to bottom and left to right. However, if the element is not in the correct place then it will be compared with the parent node. If the element is not in the correct order, the elements are swapped. The swapping process continues until all the elements are placed in the correct position.

Enqueue(priorityQueue[], element, priority)

1. Create a node (element, priority)
2. If the queue is empty,
   Insert the node at the start of the array
   return
3. Else,
   traverse the queue to find the correct insertion position:
   Set i = 0 (start of the array)
   While i < size of the queue AND priority of the current node in queue[i] ≥ priority of the new element:
   Increment i (Move to the next element)
4. Insert the new element at index i (shift all elements after i one position to the right)
5. return the updated priorityQueue[]

2) Deletion in a Priority Queue

- As you know that in a max heap, the maximum element is the root node. And it will remove the element which has maximum priority first. Thus, you remove the root node from the queue. This removal creates an empty slot, which will be further filled with new insertion. Then, it compares the newly inserted element with all the elements inside the queue to maintain the heap invariant.

3) Peek in a Priority Queue

- This operation helps to return the maximum element from Max Heap or the minimum element from Min Heap without deleting the node from the priority queue.

Dequeue(priorityQueue[])

1. If the queue is empty,

   return "Queue is empty“

2. Else,

   Remove and return the element at the front (index 0) of the array

3. Shift all elements left by one position to fill the gap

4. Return the updated priorityQueue[]

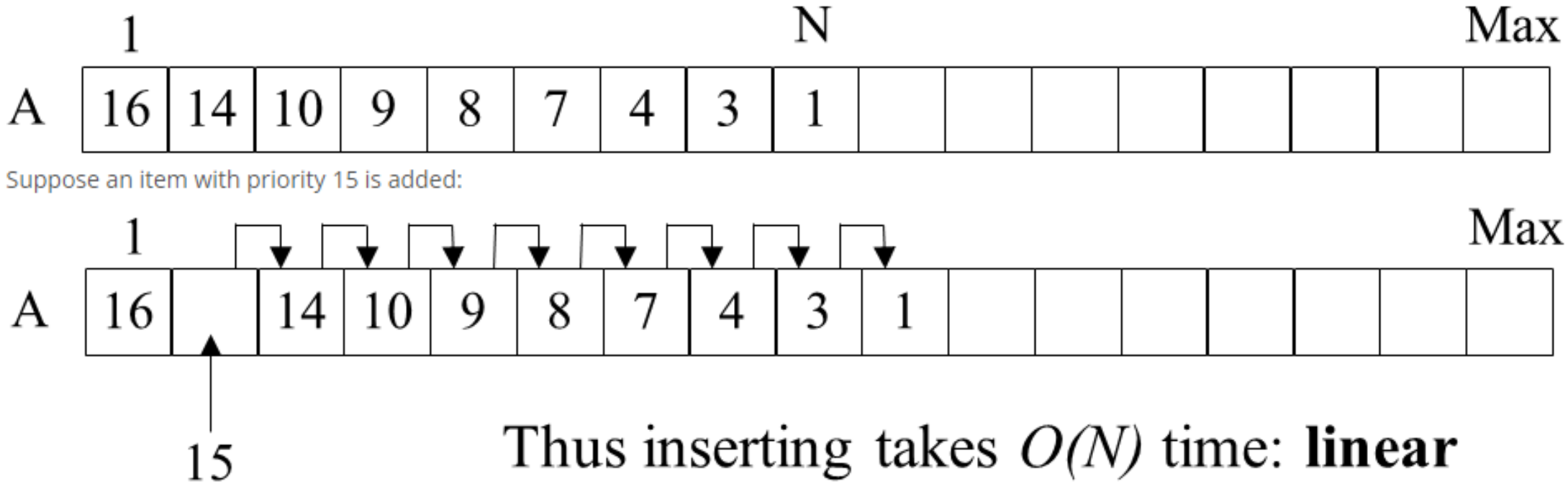
Difference between Priority and Linear Queue

- There is no priority attached to elements in a queue, the rule of first-in-first-out(FIFO) is implemented whereas, in a priority queue, the elements have a priority. The elements with higher priority are served first.

Array Representation of Priority Queue

- **enqueue():** This function is used to insert new data into the queue.
- **dequeue():** This function removes the element with the highest priority from the queue.
- **peek()/top():** This function is used to get the highest priority element in the queue without removing it from the queue.
- Thus the largest item is always at the end, and the code for remove the maximum in the priority queue is the same as for pop in the stack.

- Suppose items with priorities 16, 14, 10, 9, 8, 7, 4, 3, 1 are to be stored in a priority queue.



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.
- Data compression in Huffman code
- Event-driven simulation such as customers waiting in a queue.
- Finding Kth largest/smallest element.

Advantages

- It helps to access the elements in a faster way. This is because elements in a priority queue are ordered by priority, one can easily retrieve the highest priority element without having to search through the entire queue.
- The ordering of elements in a Priority Queue is done dynamically. Elements in a priority queue can have their priority values updated, which allows the queue to dynamically reorder itself as priorities change.
- Efficient algorithms can be implemented. Priority queues are used in many algorithms to improve their efficiency, such as Dijkstra's algorithm for finding the shortest path in a graph and the A* search algorithm for pathfinding.
- Included in real-time systems. This is because priority queues allow you to quickly retrieve the highest priority element, they are often used in real-time systems where time is of the essence.

Disadvantages

- High complexity. Priority queues are more complex than simple data structures like arrays and linked lists, and may be more difficult to implement and maintain.
- High consumption of memory. Storing the priority value for each element in a priority queue can take up additional memory, which may be a concern in systems with limited resources.
- It is not always the most efficient data structure. In some cases, other data structures like heaps or binary search trees may be more efficient for certain operations, such as finding the minimum or maximum element in the queue.
- At times it is less predictable:. This is because the order of elements in a priority queue is determined by their priority values, the order in which elements are retrieved may be less predictable than with other data structures like stacks or queues, which follow a first-in, first-out (FIFO) or last-in, first-out (LIFO) order.

Applications of Queues

- Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
- Queues are used to transfer data asynchronously (data not necessarily received at same rate as sent) between two processes (IO buffers), e.g., pipes, file IO, sockets.
- Queues are used as buffers on MP3 players and portable CD players, iPod playlist.

- Queues are used in Playlist for jukebox to add songs to the end, play from the front of the list.
- Queues are used in operating system for handling interrupts. When programming a real-time system that can be interrupted, for example, by a mouse click, it is necessary to process the interrupts immediately, before proceeding with the current job. If the interrupts have to be handled in the order of arrival, then a FIFO queue is the appropriate data structure.

Josephus Problem

- In Josephus problem, n people stand in a circle waiting to be executed. The counting starts at some point in the circle and proceeds in a specific direction around the circle. In each step, a certain number of people are skipped and the next person is executed (or eliminated). The elimination of people makes the circle smaller and smaller. At the last step, only one person remains who is declared the ‘winner’.
- Therefore, if there are n number of people and a number k which indicates that k–1 people are skipped and k–th person in the circle is eliminated, then the problem is to choose a position in the initial circle so that the given person becomes the winner.

- For example, if there are 5 (n) people and every second (k) person is eliminated, then first the person at position 2 is eliminated followed by the person at position 4 followed by person at position 1 and finally the person at position 5 is eliminated. Therefore, the person at position 3 becomes the winner

DISCUSSION…

SVKM'S NMIMS
Deemed to be UNIVERSITY

THANK YOU