BITS Pilani
K K Birla Goa Campus

innovate achieve lead

# 6. Queues

Dr. Swati Agarwal

# Agenda

1. Queues
2. Queue Operations
3. Queue Implementation using Arrays
4. Circular Arrays
5. Queue Implementation using Linked Lists

# Queues

- Queue is a linear data structure used for storing data
- Items are inserted and deleted as per the FIFO (first-in-first-out) principle.
- The first element inserted in the queue is the first element to be deleted
- Queue is open from both the ends unlike Stack (three sided container). Insertion takes place from one end called front and deletion takes place from another end called rear.

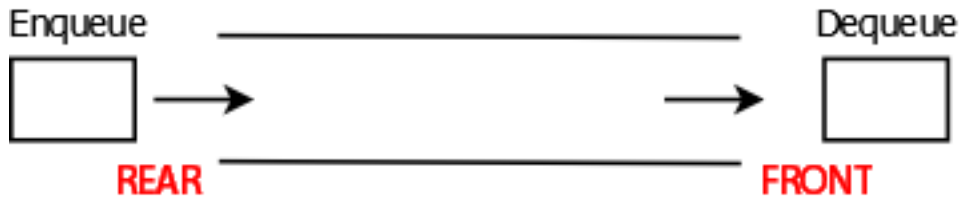
**Examples:** Queue formed for bus/ticket/elevator, CPU scheduling, Printer scheduling

# Queue Operations

- **Fundamental ADT operations:**
  - Enqueue(x) OR push(x)
    - inserts an element x at the rear of the queue
  - Dequeue() OR pop()
    - deletes an element from the front of the queue

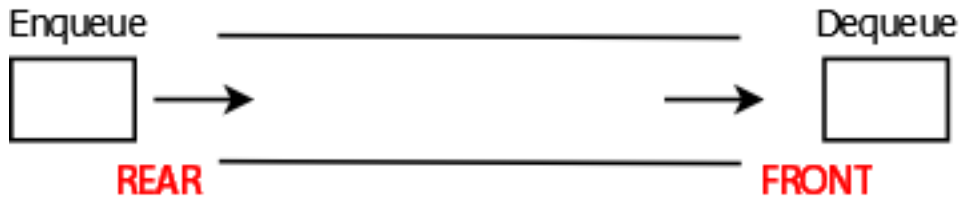
# Queue Operations

- **Fundamental ADT operations:**
  - Enqueue(x) OR push(x)
    - inserts an element x at the rear of the queue
  - Dequeue() OR pop()
    - deletes an element from the front of the queue



# Queue Operations

- **Fundamental ADT operations:**
  - Enqueue(x) OR push(x)
    - inserts an element x at the rear of the queue
  - Dequeue() OR pop()
    - deletes an element from the front of the queue




- **Additional Operations:**
  - isEmpty()
    - returns TRUE if the queue is empty otherwise FALSE
  - size() OR isFull()
    - returns the total number of elements in the queue
  - front() OR peek()
    - returns the value of the element at the front position

# Queues using Arrays

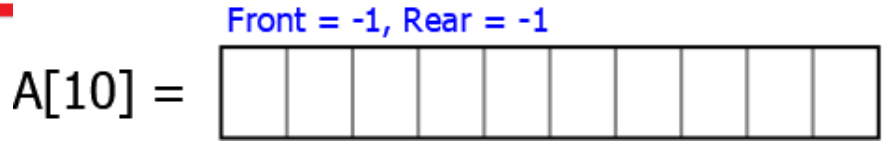
# Queue Implementation using Arrays

- Use an array to store a queue
- Two indexes are marked as FRONT and REAR pointers
- Queue is empty when $FRONT = REAR = -1$
- A queue is full when $REAR = MAX\_SIZE$

# Working Example: Enqueue Operation

```
function Enqueue(A, x)
    if isFull() then
        return
    else if isEmpty() then
        front = rear = 0
    else
        rear = rear + 1
    A[rear] = x
```

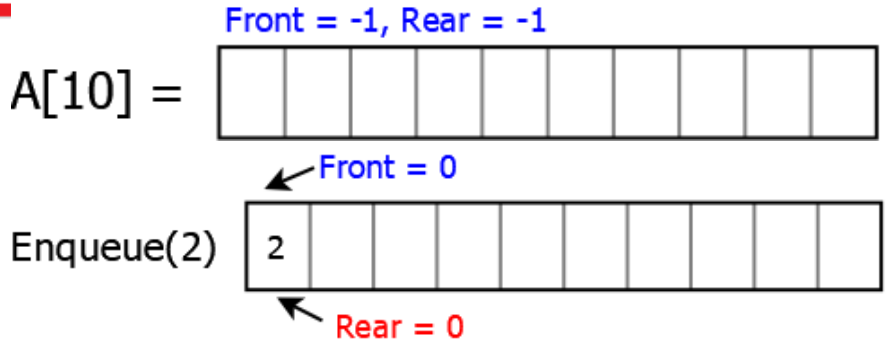
# Working Example: Enqueue Operation




**function** **Enqueue**($A$, $x$)
    **if** *isFull*() **then**
        **return**
    **else if** *isEmpty*() **then**
        $front = rear = 0$
    **else**
        $rear = rear + 1$
    $A[rear] = x$

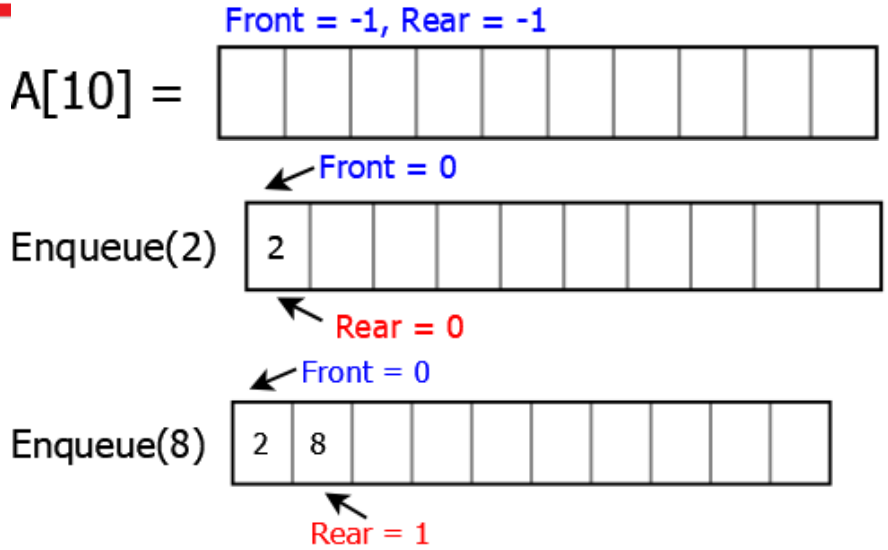
# Working Example: Enqueue Operation

**function** **Enqueue**($A$, $x$)
  **if** *isFull*() **then**
    **return**
  **else if** *isEmpty*() **then**
    $front = rear = 0$
  **else**
    $rear = rear + 1$
  $A[rear] = x$



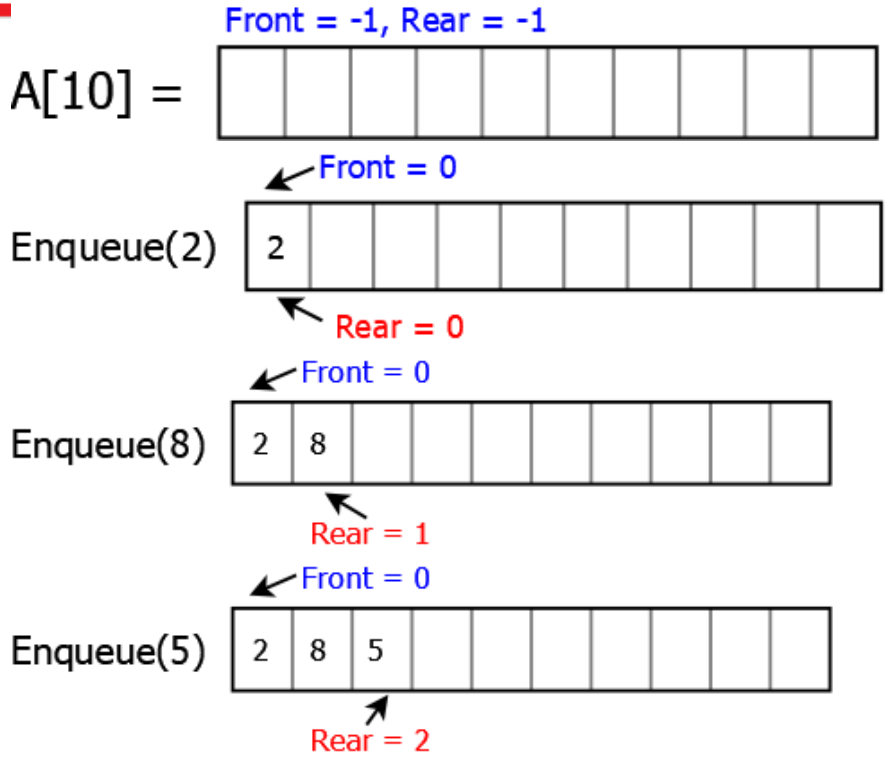
# Working Example: Enqueue Operation

**function** **Enqueue**($A$, $x$)
    **if** *isFull*() **then**
        **return**
    **else if** *isEmpty*() **then**
        $front = rear = 0$
    **else**
        $rear = rear + 1$
    $A[rear] = x$



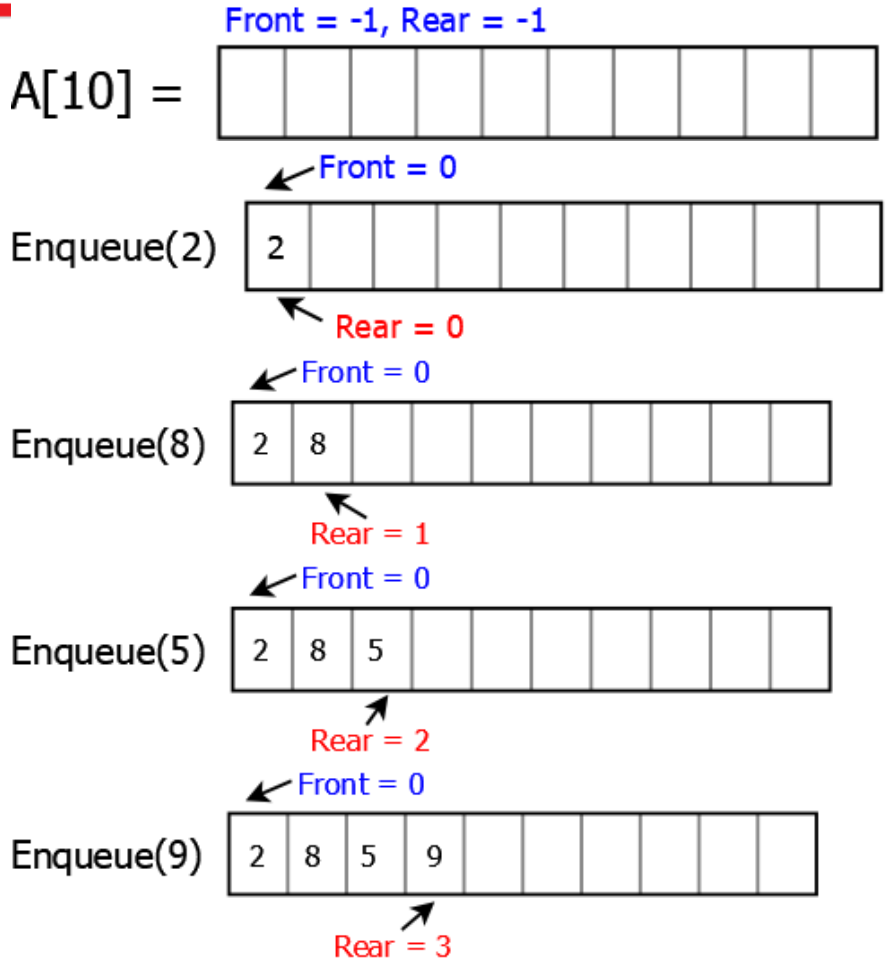
# Working Example: Enqueue Operation

**function** Enqueue($A$, $x$)
  **if** *isFull*() **then**
    **return**
  **else if** *isEmpty*() **then**
    $front = rear = 0$
  **else**
    $rear = rear + 1$
  $A[rear] = x$



# Working Example: Enqueue Operation

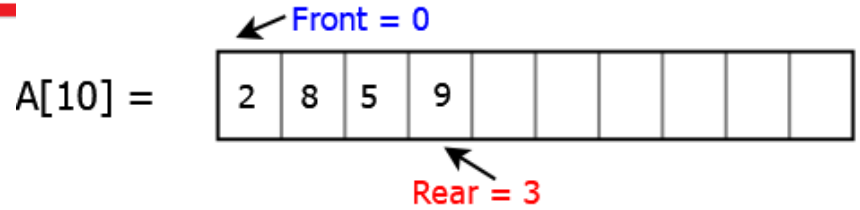
**function** **Enqueue**($A$, $x$)
  **if** *isFull*() **then**
    **return**
  **else if** *isEmpty*() **then**
    $front = rear = 0$
  **else**
    $rear = rear + 1$
  $A[rear] = x$



# Working Example: Dequeue Operation

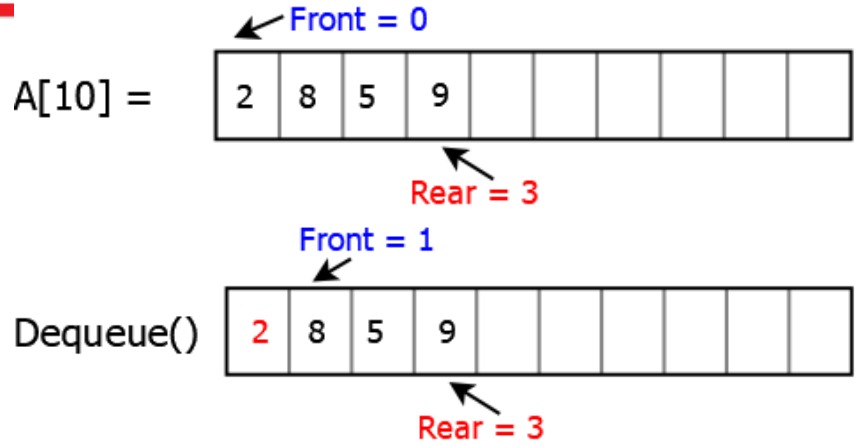
```
function Dequeue(A)
    if isEmpty() then
        return
    else if front == rear then
        front = rear = −1
    else
        front = front + 1
```

# Working Example: Dequeue Operation




**function** **Dequeue**($A$)
    **if** *isEmpty*() **then**
        **return**
    **else if** $front == rear$ **then**
        $front = rear = -1$
    **else**
        $front = front + 1$

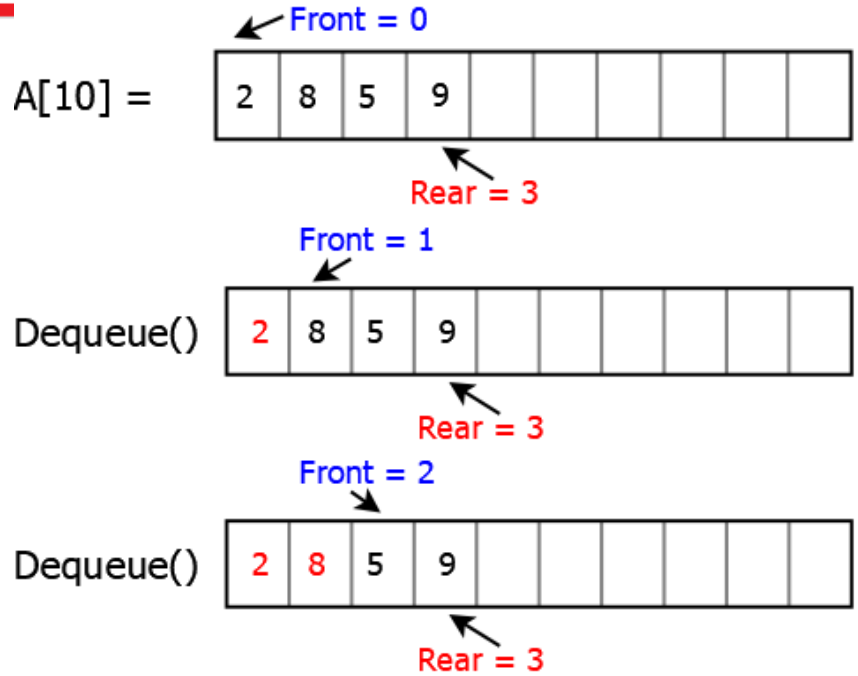
# Working Example: Dequeue Operation




**function** **Dequeue**($A$)
    **if** *isEmpty*() **then**
        **return**
    **else if** $front == rear$ **then**
        $front = rear = -1$
    **else**
        $front = front + 1$

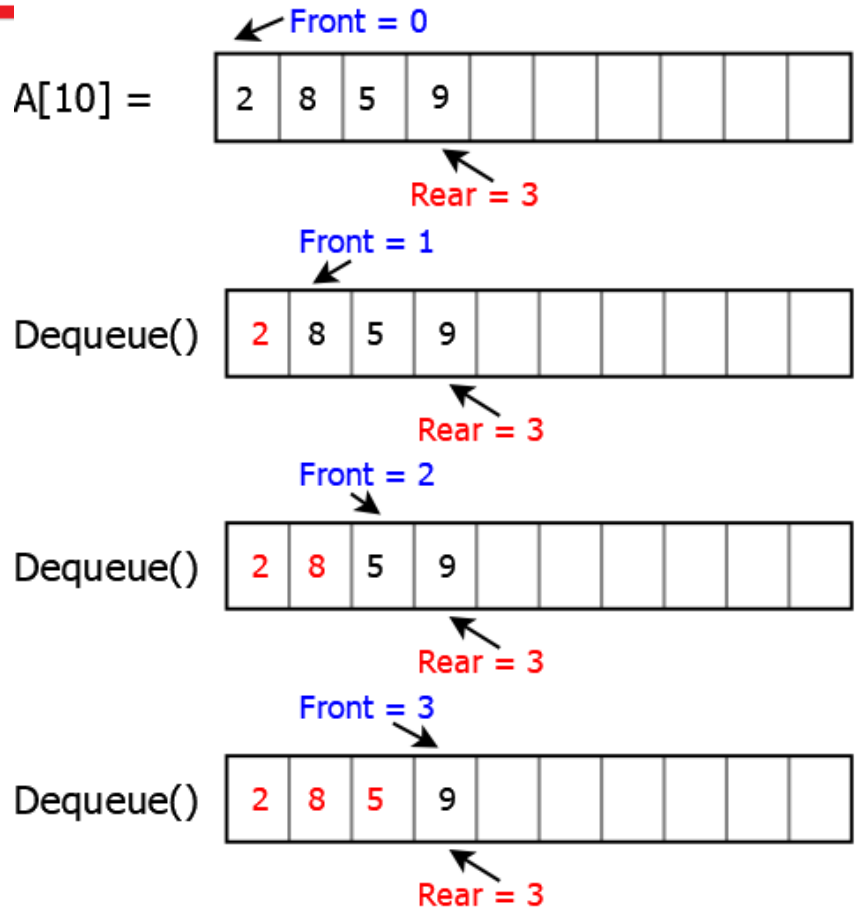
# Working Example: Dequeue Operation

```
function Dequeue(A)
    if isEmpty() then
        return
    else if front == rear then
        front = rear = −1
    else
        front = front + 1
```



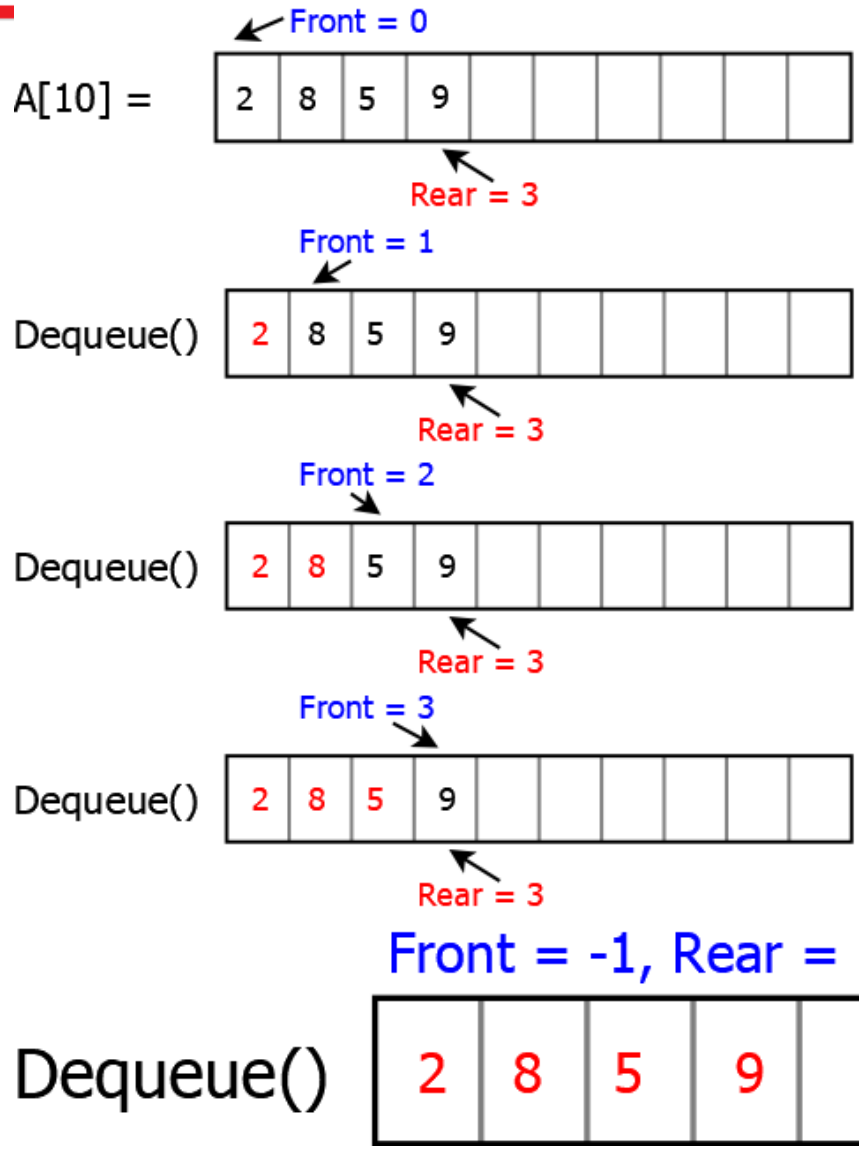
# Working Example: Dequeue Operation

```
function Dequeue(A)
    if isEmpty() then
        return
    else if front == rear then
        front = rear = −1
    else
        front = front + 1
```



# Working Example: Dequeue Operation

```
function Dequeue(A)
    if isEmpty() then
        return
    else if front == rear then
        front = rear = −1
    else
        front = front + 1
```



# Limitations

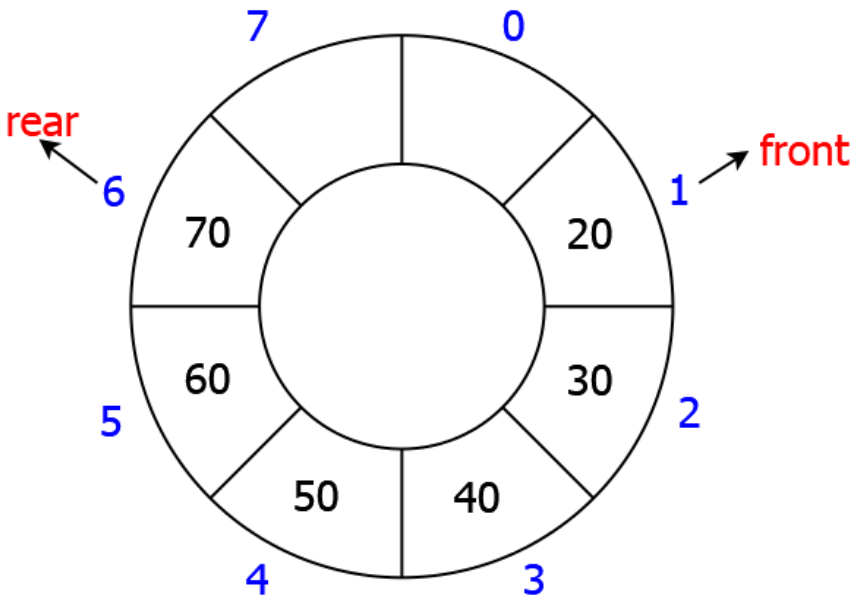
- Once we *Dequeue*() an element from the array, the associated cell space is wasted.
- The above space cannot be reused until the queue is empty ($front = rear = -1$) and *Enqueue*($x$) operations are performed from the beginning.

**Possible way to solve that?**

- using a circular array
- If the current *rear* value is $i$ then next element to be inserted at $(i + 1)$ % $N$. $N$= size of the array.

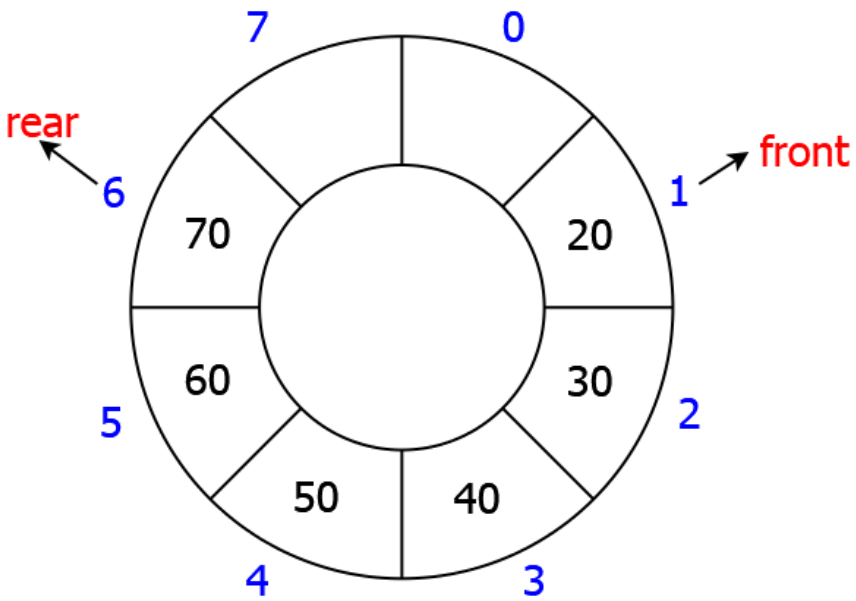
# Circular Array

# Queue Implementation using Circular Array




**function Enqueue**$(A, x)$
    **if** $(rear + 1)\%N == front$
**then**
        **return**
    **else if** $isEmpty()$ **then**
        $front = rear = 0$
    **else**
        $rear = (rear + 1)\%N$
    $A[rear] = x$

# Queue Implementation using Circular Array



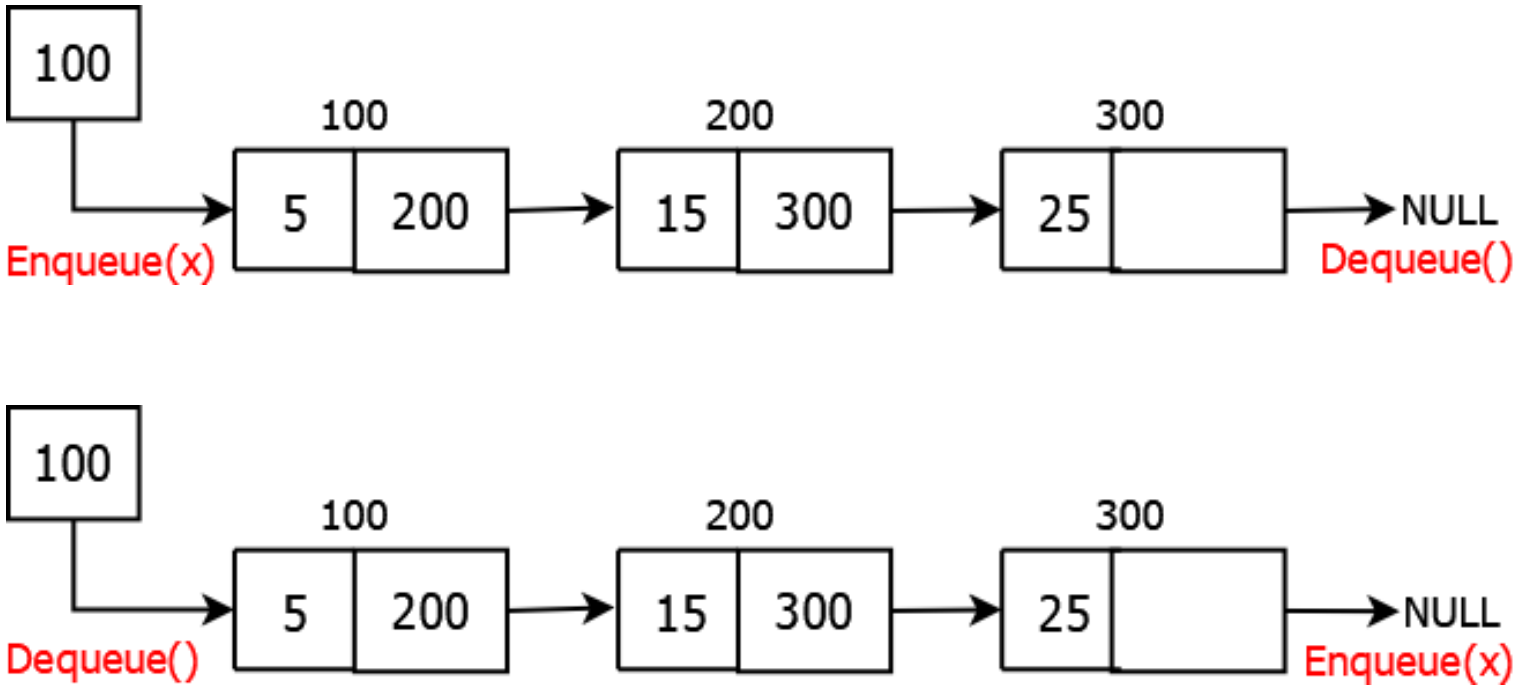

**function** **Dequeue**$(A)$
  **if** *isEmpty*() **then**
    **return**
  **else if** $front == rear$ **then**
    $front = rear = -1$
  **else**
    $front = (front + 1)\%N$

# Queues using Linked Lists

# Queue Implementation using Linked Lists

- An array can allow only $N$ number of elements to be inserted in the queues at once and highly likely to get exhausted even for circular arrays.
- Possible Solution is to use **Dynamic Arrays** but copying the elements from smaller arrays to larger array costs $\mathcal{O}(n)$ operations.
- We can use a dynamic array but a lot of memory might be unused by the queue.
- Another feasible solution is to implement the queues using Linked Lists.

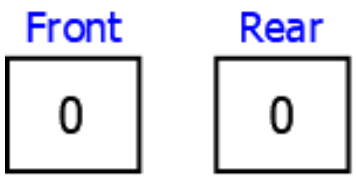
# Queue Implementation using Linked Lists



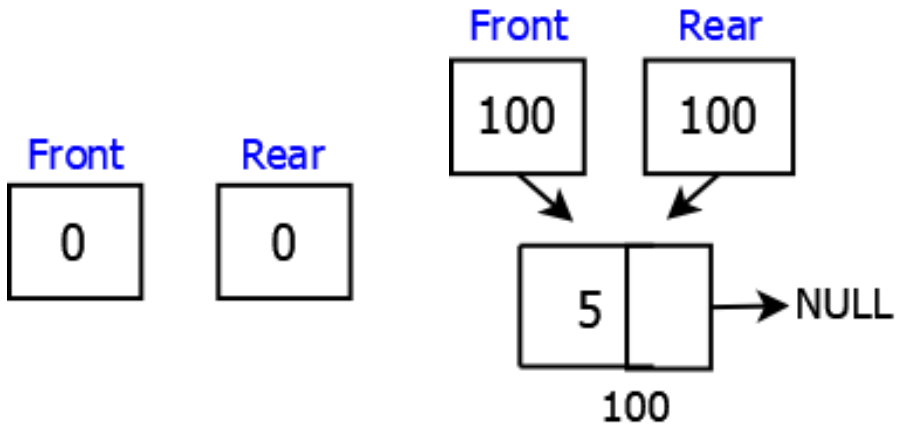

**Challenge:** insertion or deletion operation at the end of the queue takes $\mathcal{O}(n)$ time.

**Solution:** maintain two pointers *front* and *rear*.

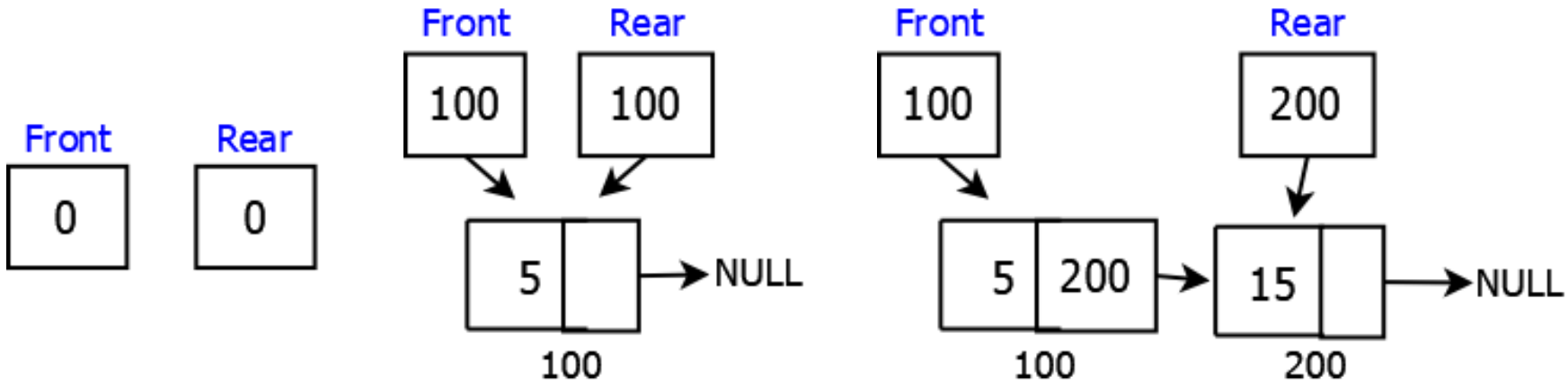
# Working Example: Enqueue Operation



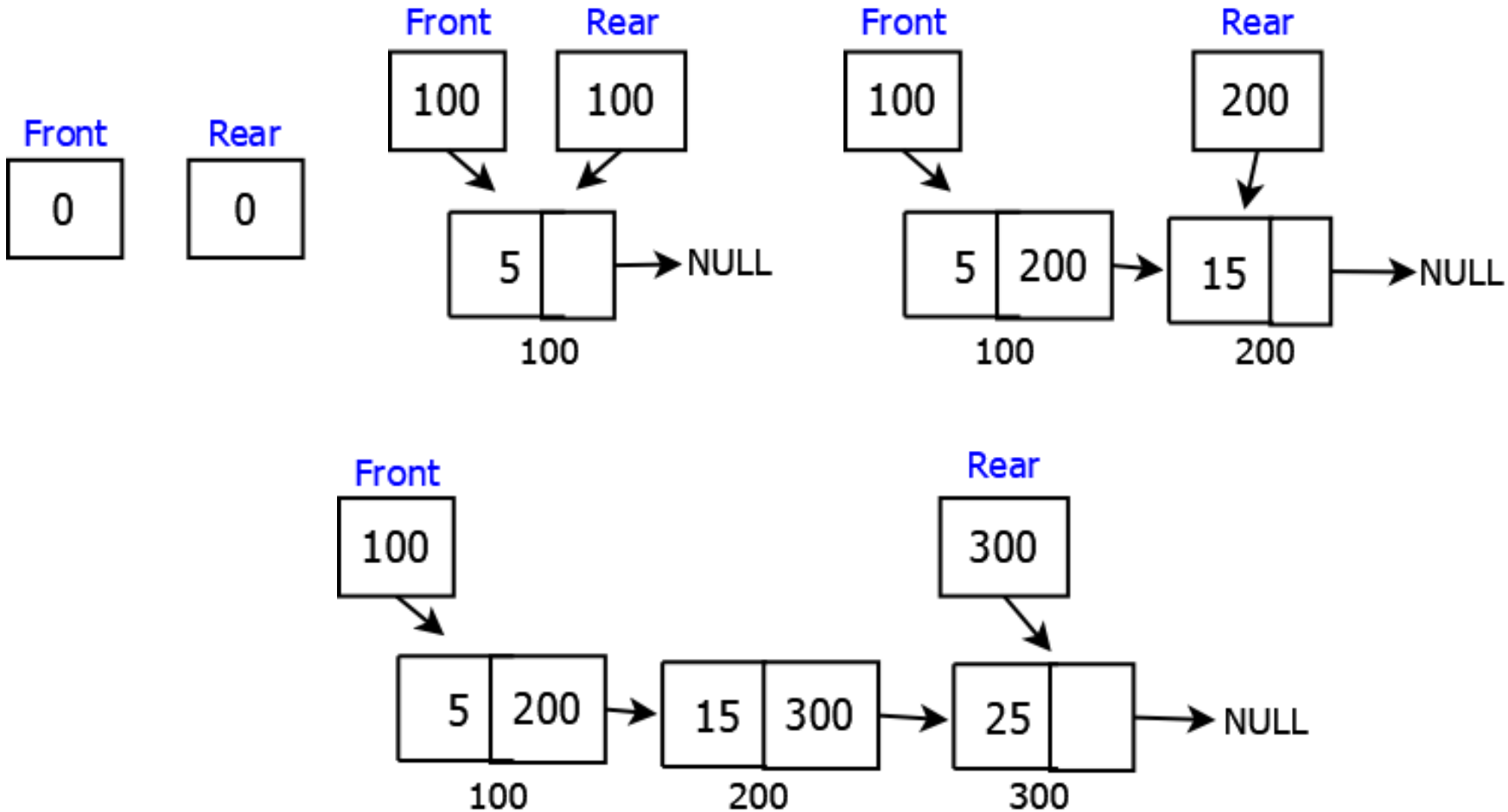
# Working Example: Enqueue Operation



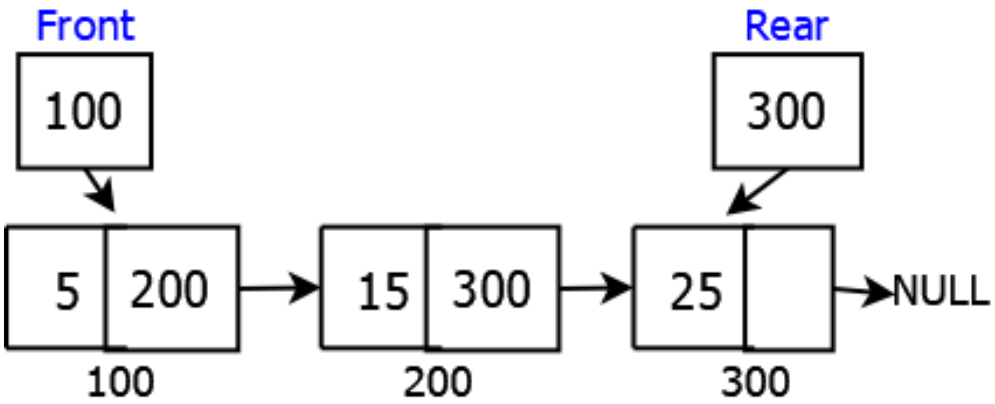
# Working Example: Enqueue Operation



# Working Example: Enqueue Operation



# Working Example: Dequeue Operation



# Working Example: Dequeue Operation