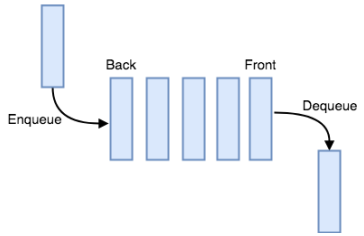
Al Hussein
Technical University
جامعة الحسين التقنية

Data Structures and Algorithms

Queue



Prepared by:

Eng. Malek Al-Louzi

School of Computing and Informatics – Al Hussein Technical University

Spring 2021/2022

Outlines

- Introduction to Queue
- Queue ADT and its operations
- Queue Applications
- Implementation of Queue
- Array implementation of Queue
- Pseudocode for Array implementation of Queue

What is Queue?

Introduction to Queue 1/2

- **Queue** data structure is exactly what we mean when we say **Queue** in real world.
- **Queue** is a data structure in which whatever goes in first, comes out first.
- First In First Out (FIFO).
  - Stack is LIFO.

Introduction to Queue 2/2

- Insertion must happen from one end called rear or tail.
- Removal must happen from the other end, which called front or head.

Queue Definition

- Formal definition of Queue ADT:

A list or collection with the restriction that insertion can be performed at one end (rear) and deletion can be performed at other end (front).

Queue Operations

Operations available with Queue:

- The main two operations are:
  - Insertion an element, called **EnQueue.**
  - Removing an element, called **DeQueue.**
- Returns the element at the top of Queue, Front.
  - Should not remove any element from the Queue.
- Check if Queue is empty or not, **IsEmty.**
- **EnQueue** and **DeQueue** one element at a time.
- **All operations can be performed in constant time -> O(1).**

(1) EnQueue(x)

(2) Dequeue()

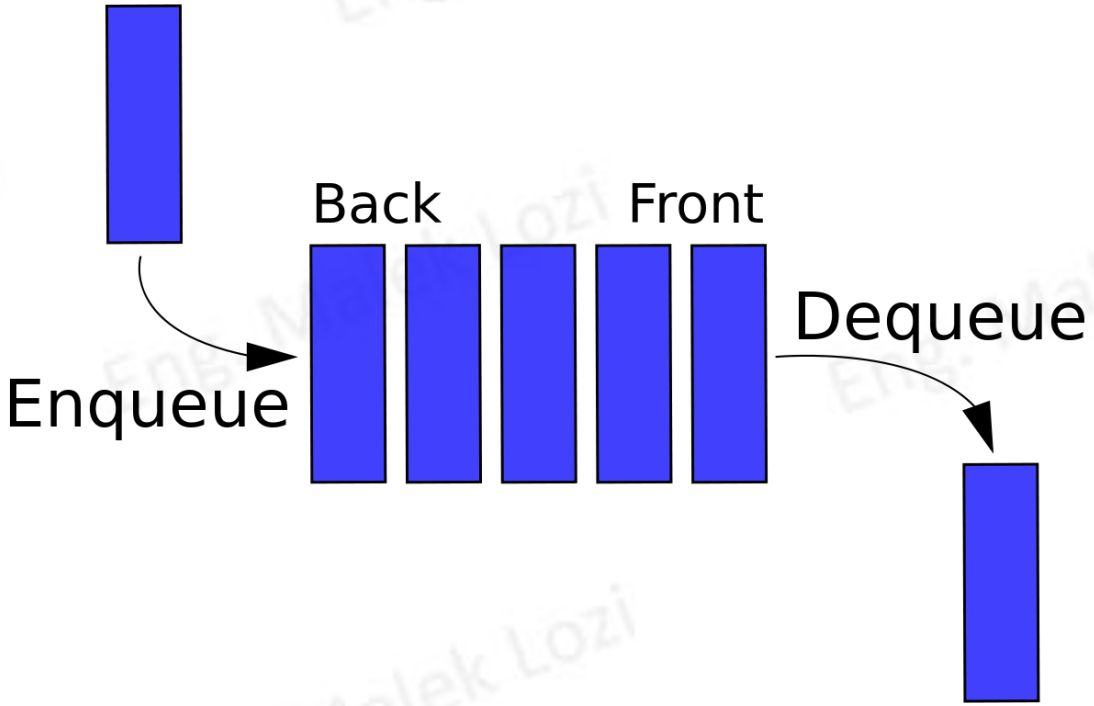
(3) front()

(4) IsEmpty()

Logical representation of Queue

- Logical representation of a Queue:
  - Container opened from Two sides.
  - An element can be inserted from one side.
  - An element can be removed from the other side.

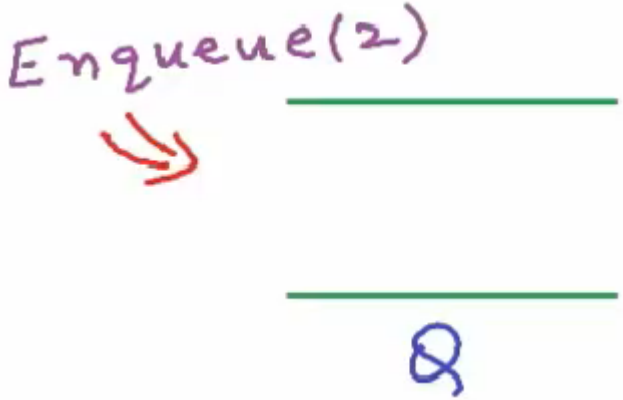
Insertion & Removing an element (EnQueue) & (DeQueue)



(EnQueue & DeQueue) (1/3)

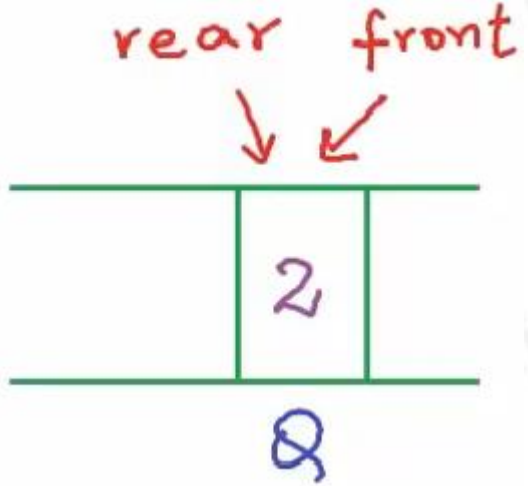
- We have an empty Queue called Q:

- Inserting number 2.

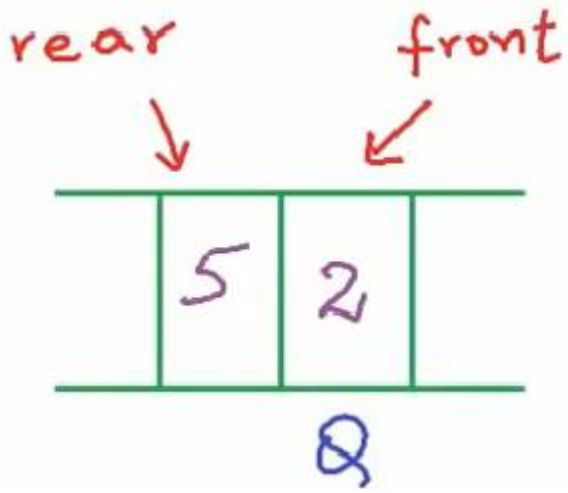


(EnQueue & DeQueue) (2/3)

- After the EnQueue, the Queue will look like:
  - It contains one integer.

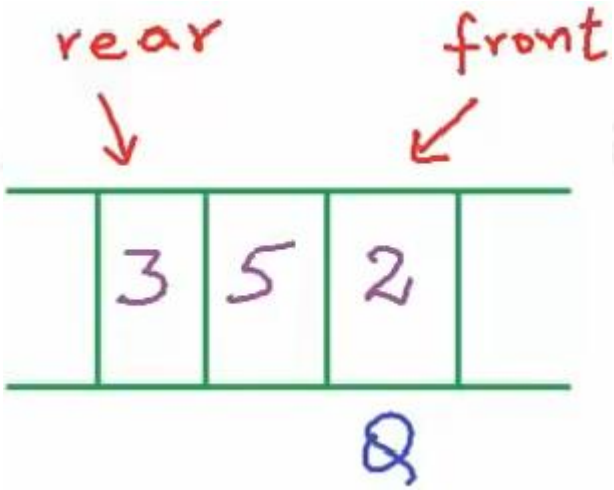


- Inserting number 5.
  - It will be at rear of the Queue.

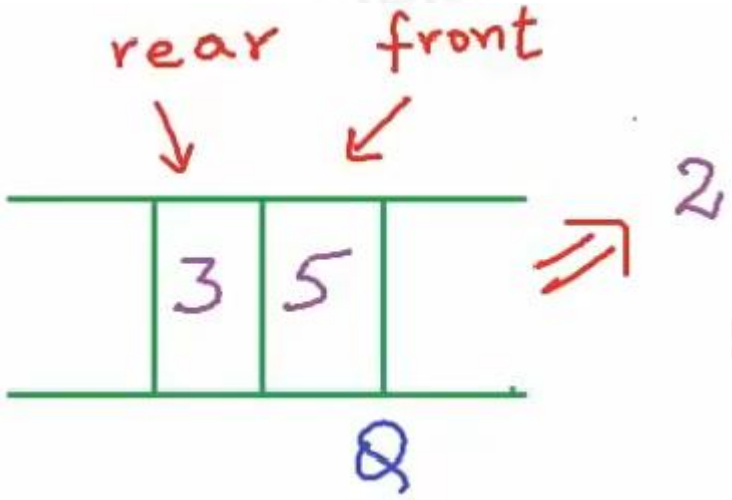


(EnQueue & DeQueue) (3/3)

- Inserting number 3:

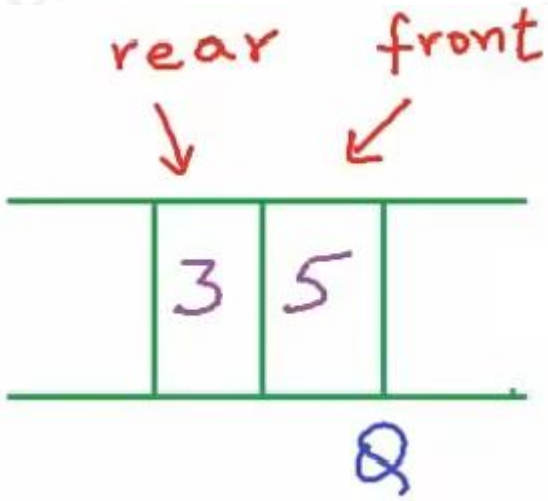


- Removing an element.
  - We will get number 2.



Calling Front () & IsEmpty() in Queue

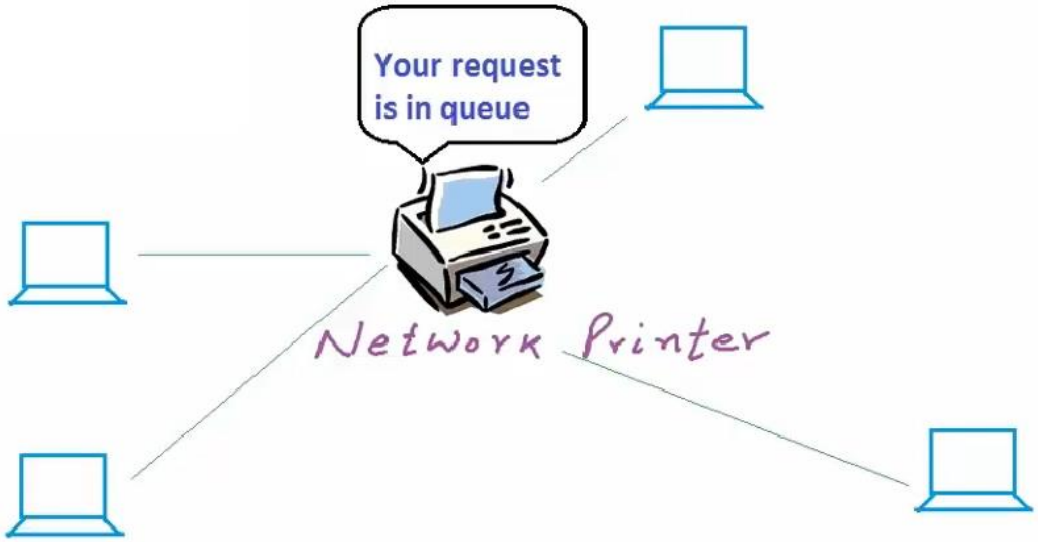
- Calling Front at this stage:
  - Will return number 5.



- Calling IsEmpty at this stage:
  - Will return False.

Queue Applications

- Real applications of Queue.
  - Shared resource that can handle only one request at a time.
    - Queuing the requests.
    - The request that comes first, get served first.
  - Process scheduling.
    - Processor on computer is a shared resource.



Implementation of Queue

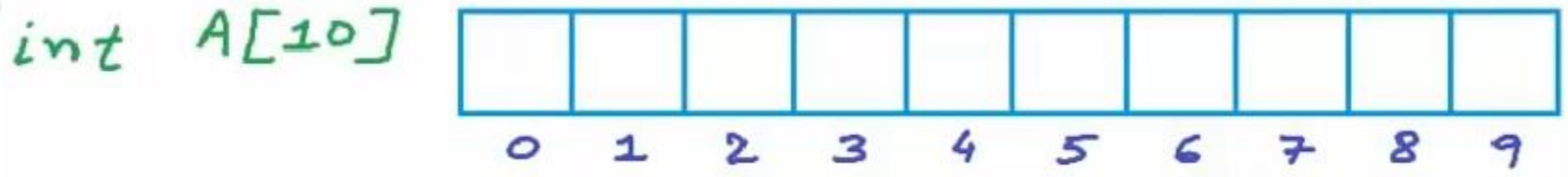
- Referring to the formal definition, A Queue is a special kind of **List**, in which an element can be inserted at One end and an element can be removed from the other end.

We can implement Queues using:

1) Arrays
2) Linked Lists

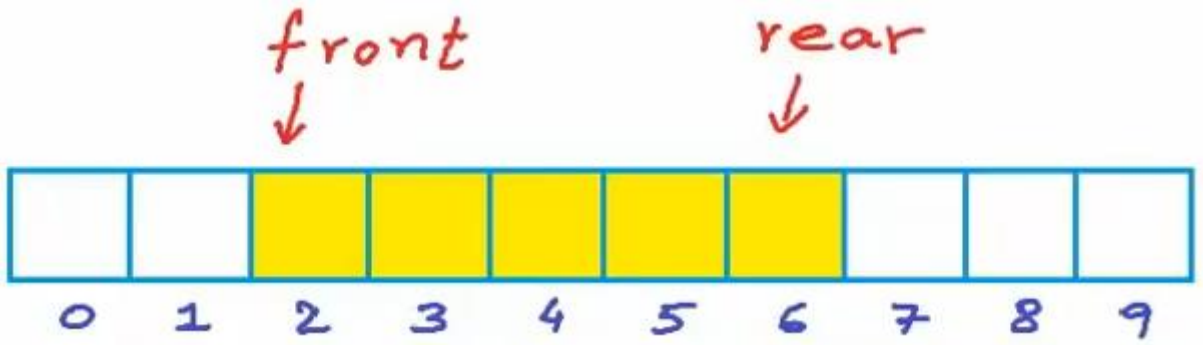
Array implementation of Queue (1/9)

- Let us create a Queue of Integers.
- First, we create an array of integers.
  - Array of Ten integers.



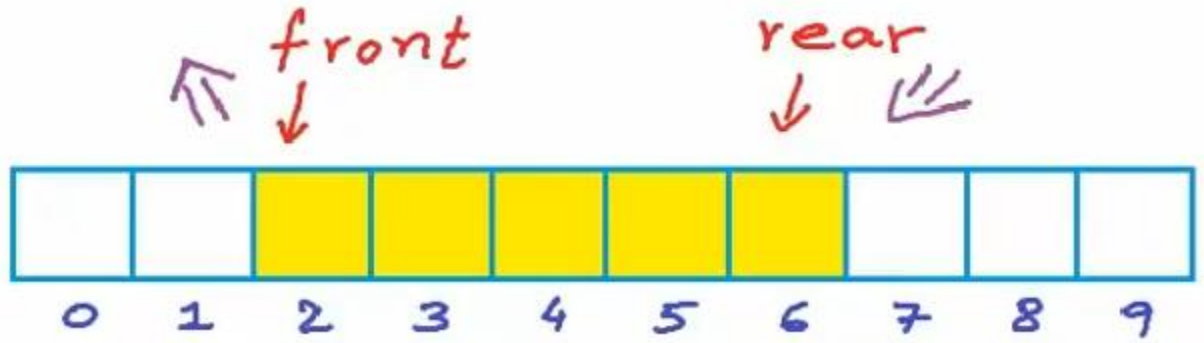
Array implementation of Queue (2/9)

- At any point, some part after array starting at index marked as front to index marked as rear will be our Queue.
  - Rest of the Array position are free space that can be used to expand the Queue.



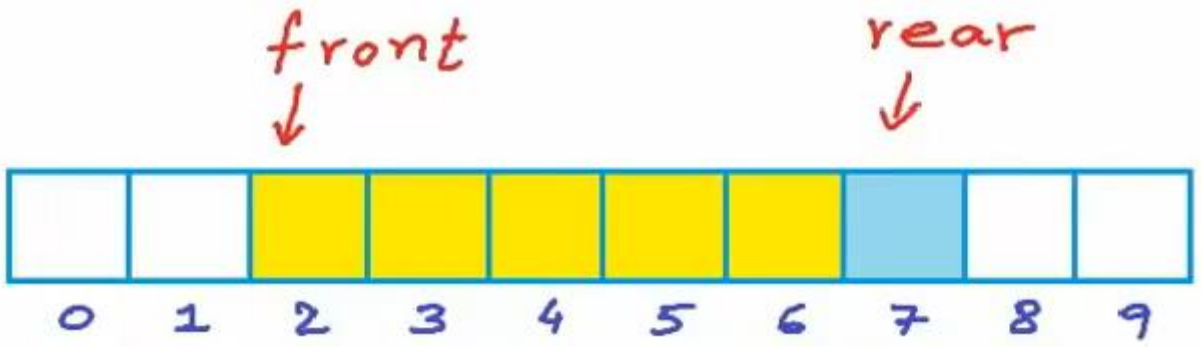
Array implementation of Queue (3/9)

- An element must always be added from rear side and must always be removed from front side.



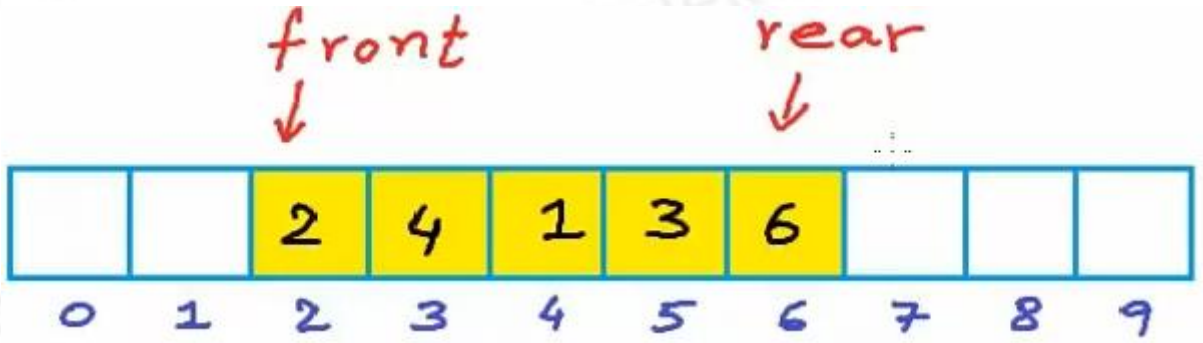
Array implementation of Queue (4/9)

- To insert an element (EnQueue), we can increment rear.
  - In this position we can add the new element.

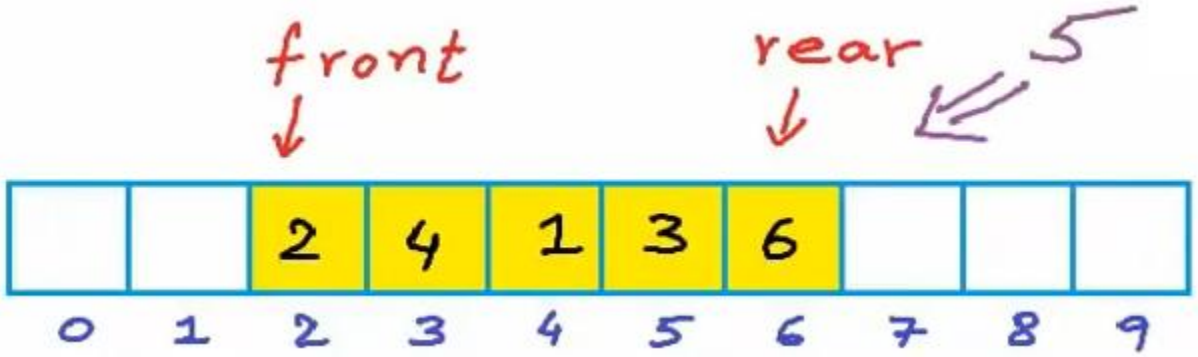


Array implementation of Queue (5/9)

- Let us fill some values in the Queue.

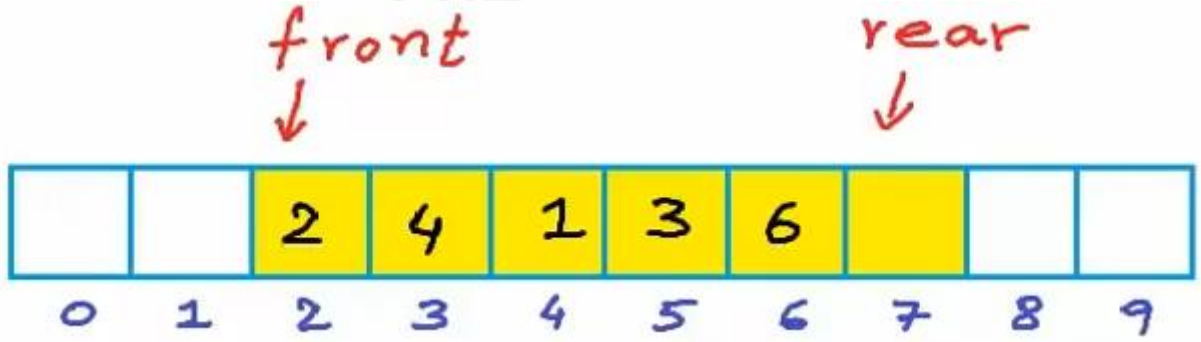


- We want to insert number 5.

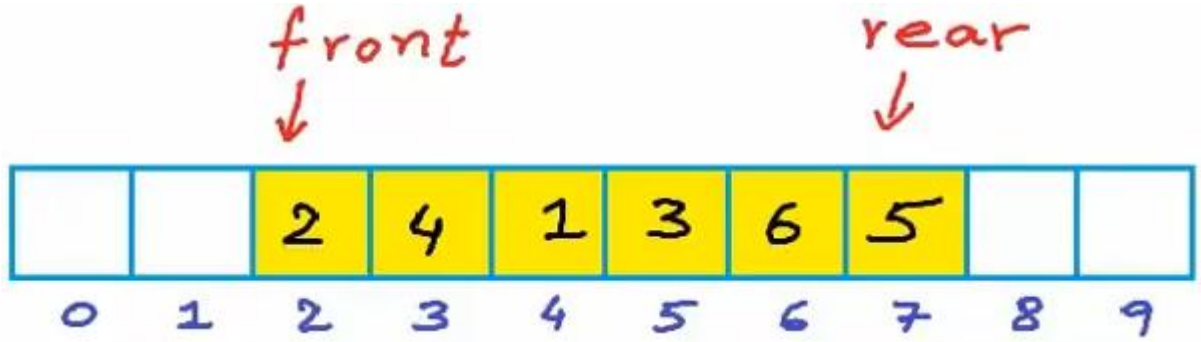


Array implementation of Queue (6/9)

- We will increment rear.
  - There should be an available empty cell in the right.

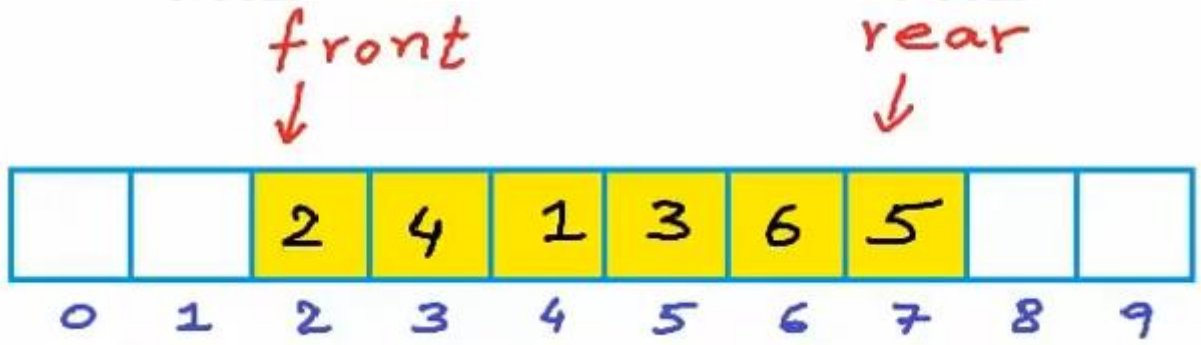


- Now we can insert number 5.



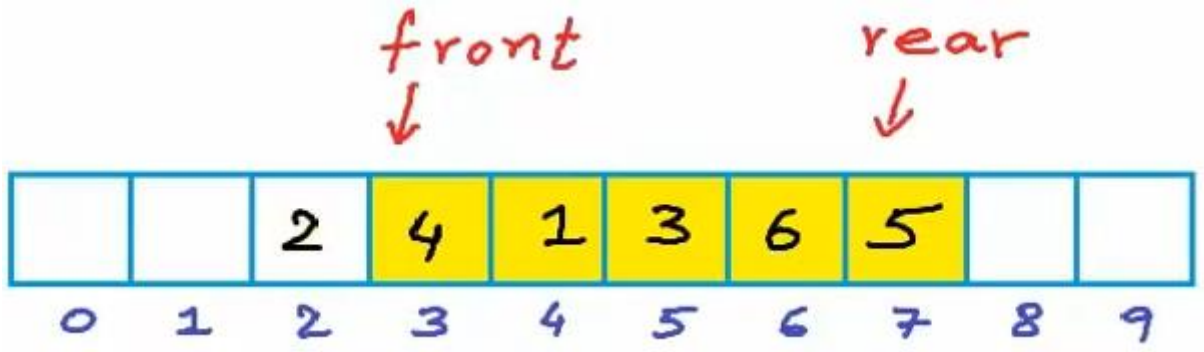
Array implementation of Queue (7/9)

- After this insertion, the rear is at index 7.
- Now DeQueue means we must remove an element from front of the Queue.
- Here, DeQueue operation should remove number 2 from the Queue.



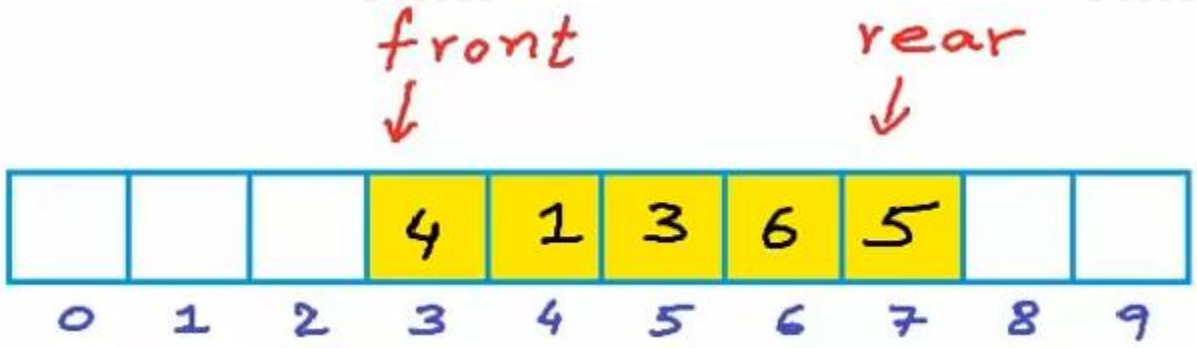
Array implementation of Queue (8/9)

- To DeQueue, we can simply increment front.
- Because at any point only the cells starting from **front** to **rear** are part of the Queue.
- By incrementing **front**, we have discarded index 2 from the Queue.



Array implementation of Queue (9/9)

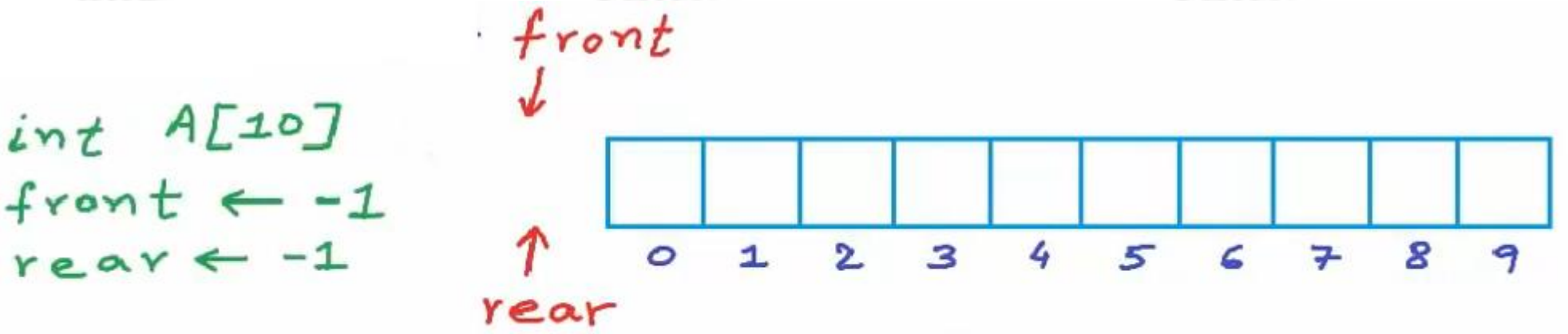
- We don’t care what value in a cell that is not part of the Queue.
  - When we include a cell in the Queue, we will overwrite the value in that cell.
- So, just incrementing **front** is good enough for DeQueue operation.



Let us write the  of Queue using Array Implementation

PseudoCode for Queue using Array (1/13)

- Let us write the PseudoCode now:
- We will declare two variable named front and rear, initially we will set them both to -1.
- So, for an empty Queue both front and rear will be -1.



PseudoCode for Queue using Array (2/13)

- To check whether a Queue is empty or not, we can simply check the value of front and rear.
- If they are both -1, then the Queue is empty.

```
IsEmpty()
{  if front==-1 && rear==-1
        return true
   else
        return false
}
```

PseudoCode for Queue using Array (3/13)

- Now, EnQueue function.
- EnQueue will take integer x as an argument.

```
Enqueue(x)
{ if rear == size(A)-1
     Print "Queue is Full"
     return
  else if IsEmpty()
  {
     front ← rear ← 0
  }
  else
  {
     rear ← rear + 1
  }
  A[rear] ← x
}
```

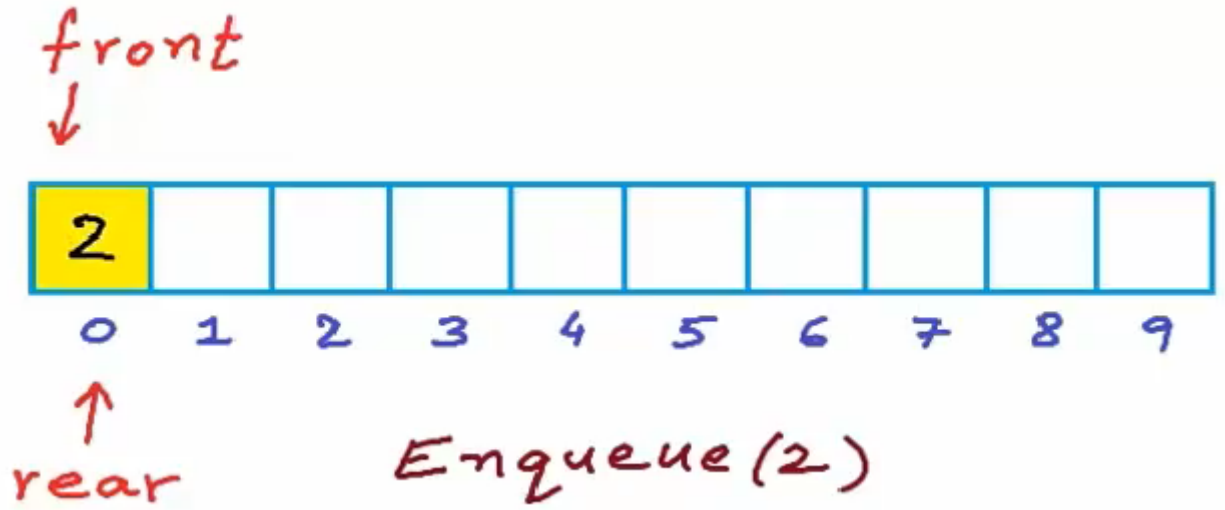
PseudoCode for Queue using Array (4/13)

- We can replace the first condition with a function named IsFull to determine the Queue is full or not.

```
Enqueue(x)
{ if  IsFull()
        return
  else if IsEmpty()
  {
    front ← rear ← 0
  }
  else
  {
    rear ← rear + 1
  }
  A[rear] ← x
}
```

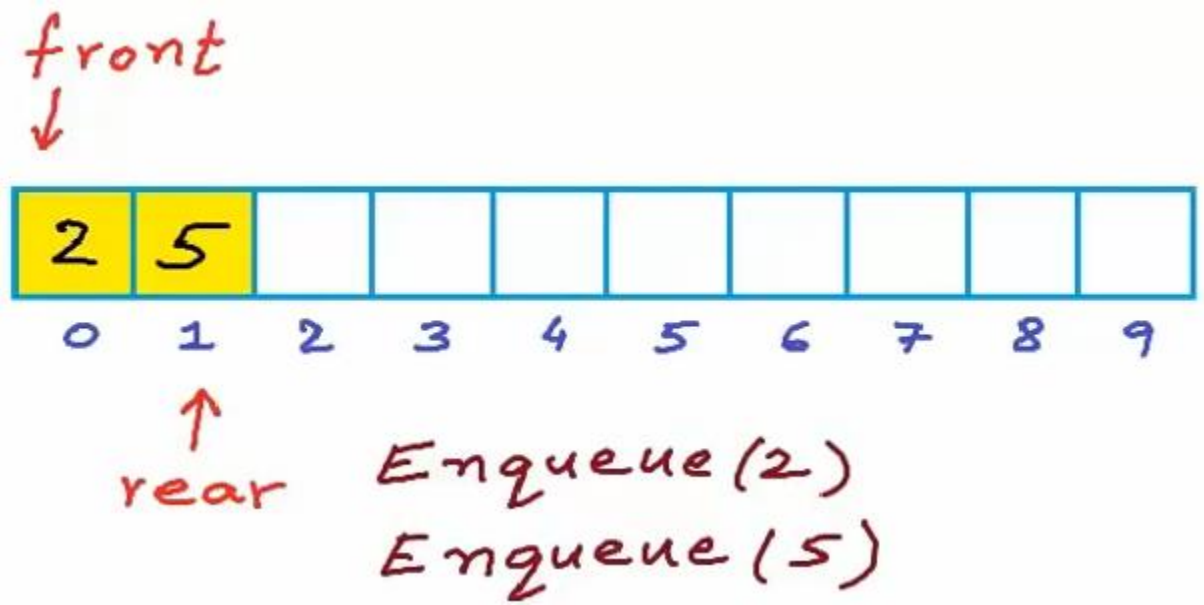
PseudoCode for Queue using Array (5/13)

- Let us EnQueue some items.
- This is the Queue after One EnQueue operation.



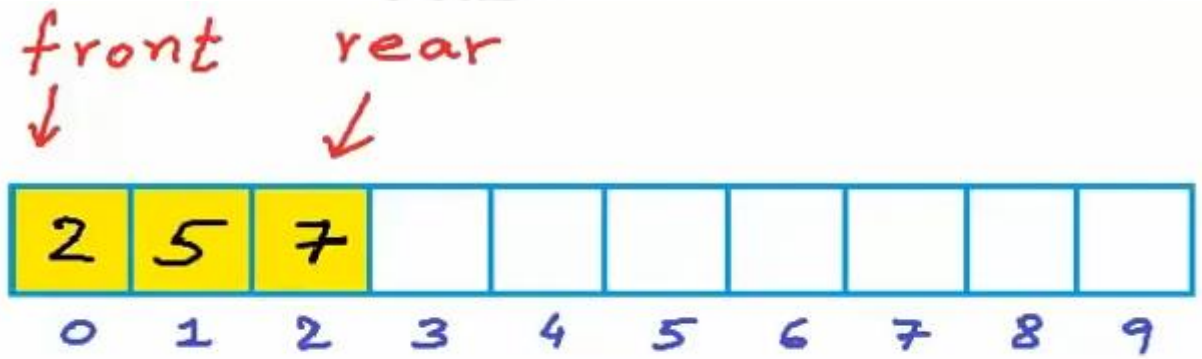
PseudoCode for Queue using Array (6/13)

- Let us make another call to EnQueue.



PseudoCode for Queue using Array (7/13)

- Let us EnQueue one more item.



Enqueue(2)
Enqueue(5)
Enqueue(7)

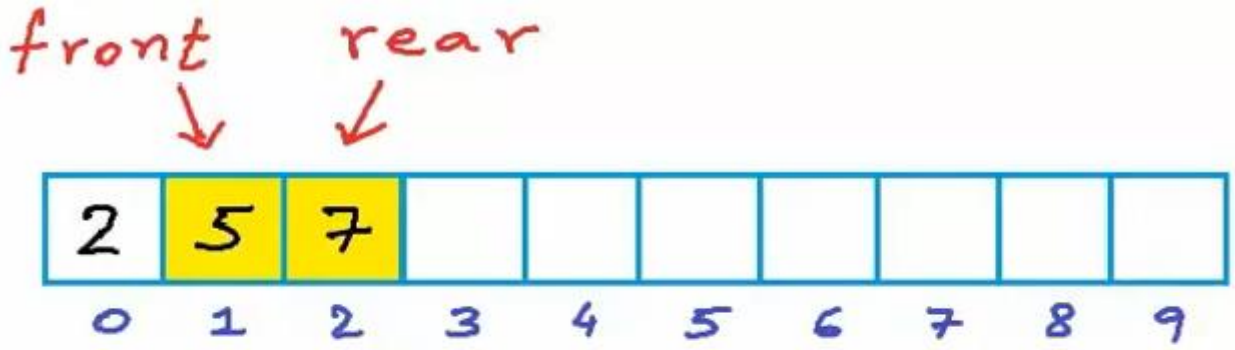
PseudoCode for Queue using Array (8/13)

- Let us now write DeQueue operation.

```
Dequeue()
{
  if IsEmpty()
      return
  else if front == rear
      front← rear← -1
  else
      front← front +1
}
```

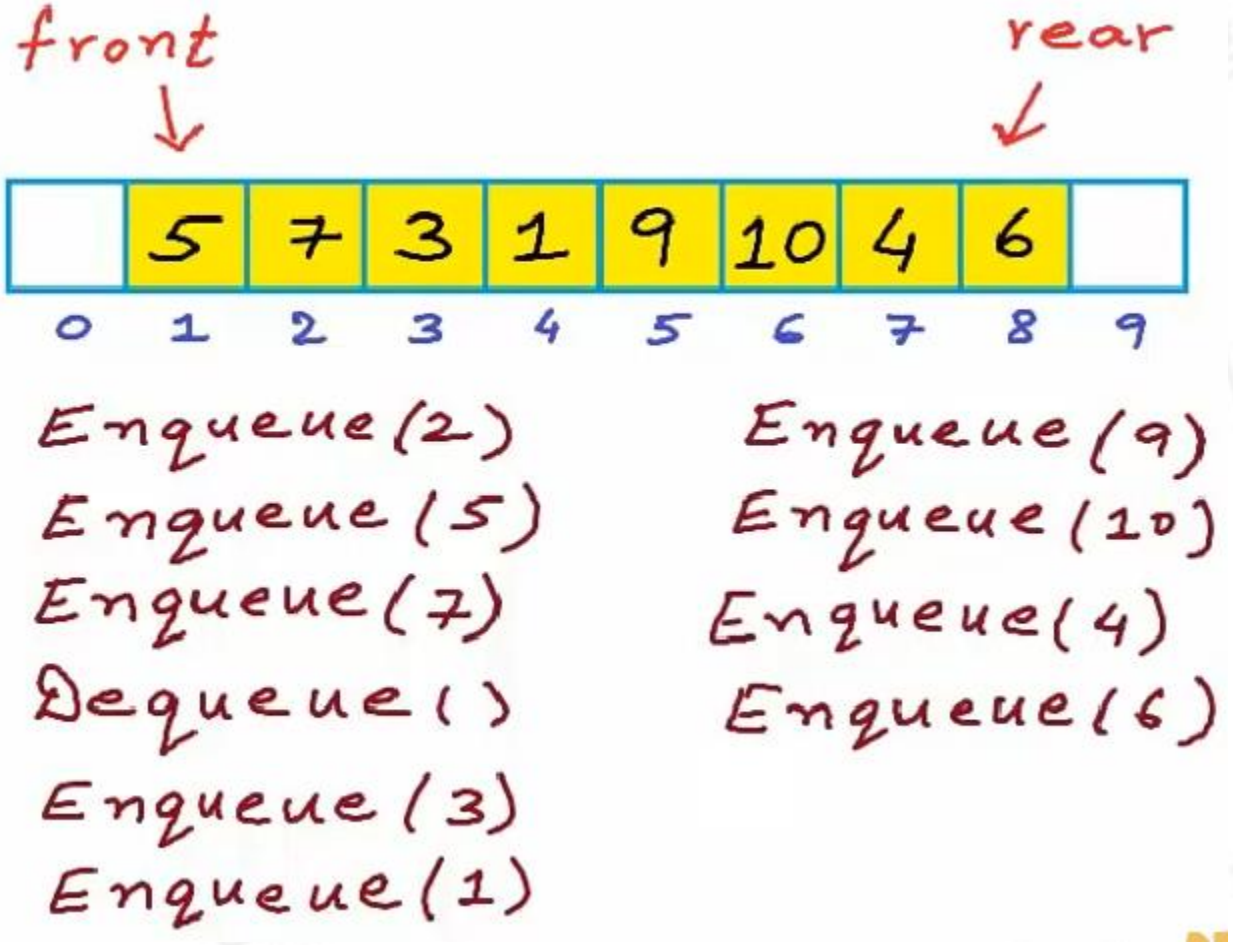
PseudoCode for Queue using Array (9/13)

- Now, we will continue the previous example and try to call DeQueue operation.



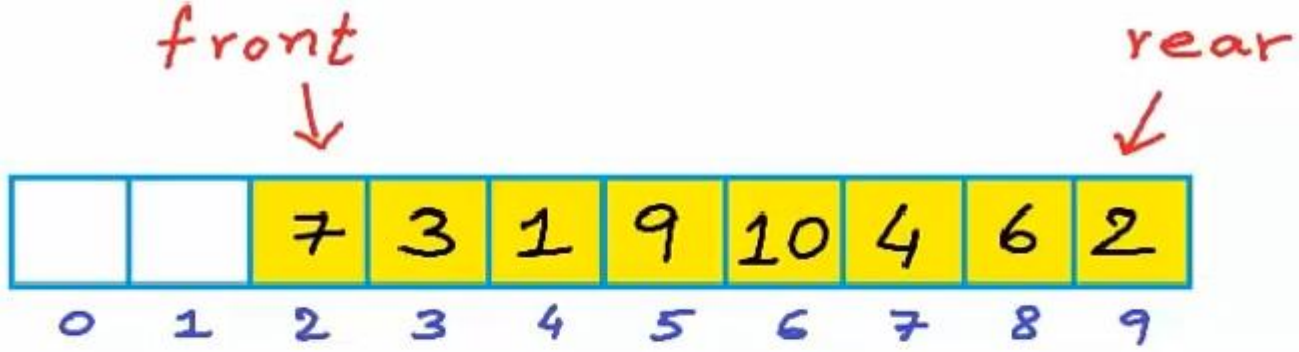
PseudoCode for Queue using Array (10/13)

- Let's perform more EnQueue operations.



PseudoCode for Queue using Array (11/13)

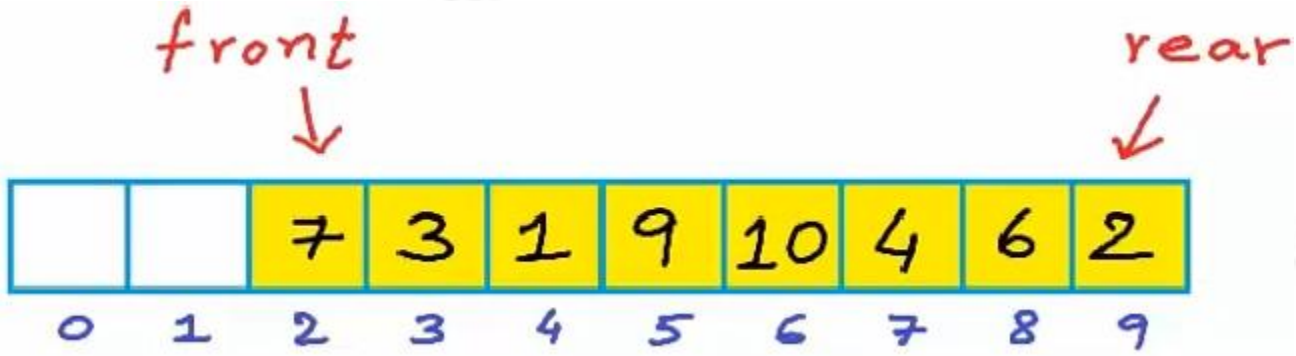
- Let's perform DeQueue and EnQueue operations.



Enqueue(2)
Enqueue(5)
Enqueue(7)
Dequeue()
Enqueue(3)
Enqueue(1)
Enqueue(9)
Enqueue(10)
Enqueue(4)
Enqueue(6)
Dequeue()
Enqueue(2)

PseudoCode for Queue using Array (12/13)

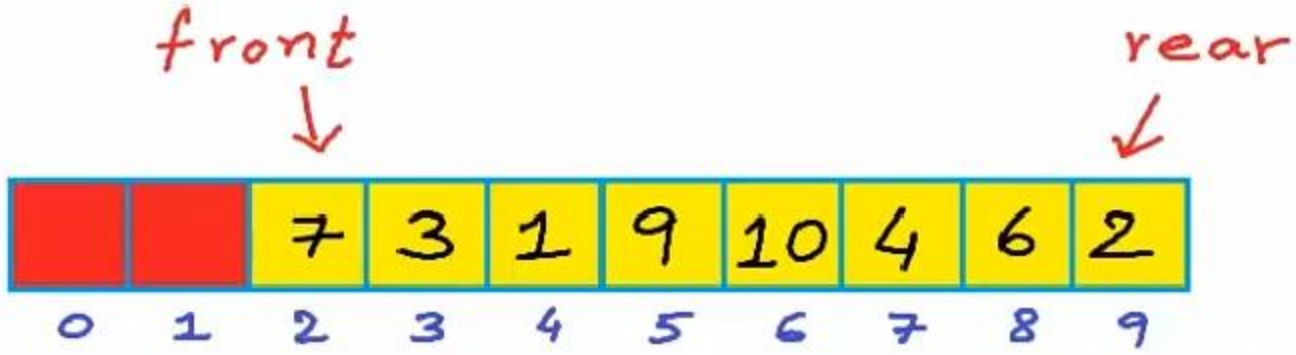
- At this stage we cannot EnQueue an item anymore, because we cannot increment rear.



- EnQueue operation will fail now.
- There are two unused cells.

PseudoCode for Queue using Array (13/13)

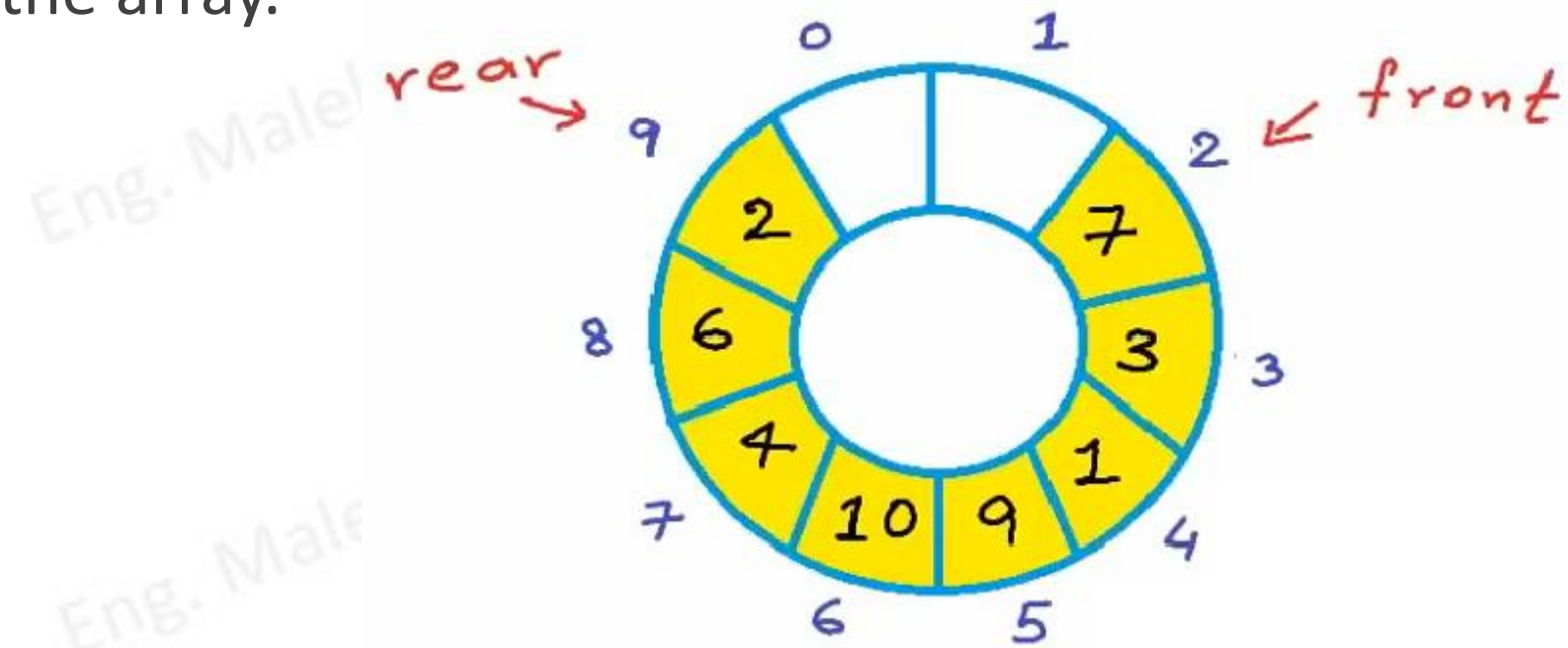
- In fact, this is a real problem.



- If we DeQueue more and more, all the cells left of front index will never be used again.
- Can we do something to use these cells?

PseudoCode for Queue using Circular Array (1/9)

- We can use the concept of Circular Array.
- Circular array is an idea that we use in a lot of scenarios.
- The idea is very simple, as we traverse an array, we can imagine that there is no end in the array.



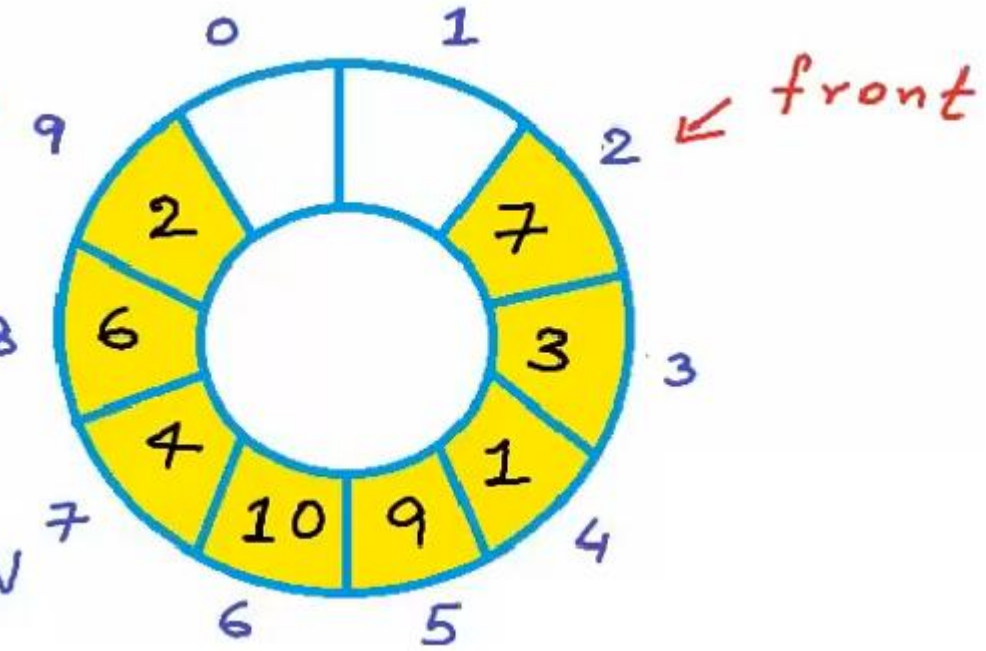
PseudoCode for Queue using Circular Array (2/9)

- In circular array, if we are pointing at any position i, the next position and the previous position will be the following:
  - N is the number of elements in Array.

Current position = i

Next position = (i+1) % N

Previous position = (i+N-1) % N



PseudoCode for Queue using Circular Array (3/9)

- Now, in an EnQueue operation we can increment rear as long as there is any unused cell in the array.
- We are going to modify the PseudoCodes.

```
Enqueue(x)
{ if (rear+1)%N == front
        return
  else if IsEmpty()
  {
     front ← rear ← 0
  }
  else
  {
     rear ← (rear+1)% N
  }
  A[rear] ← x
}
```

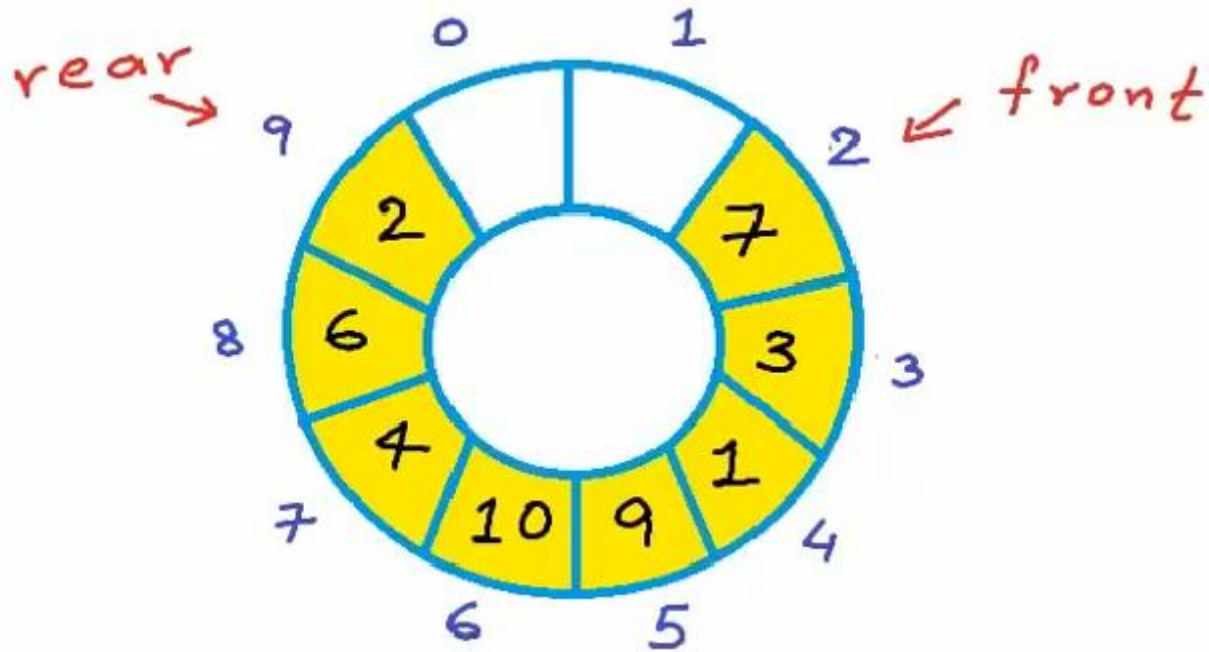
PseudoCode for Queue using Circular Array (4/9)

- The DeQueue function will be:

```
Dequeue()
{
  if IsEmpty()
      return
  else if front == rear
      front ← rear ← -1
  else
      front ← (front +1) % N
}
```

PseudoCode for Queue using Circular Array (5/9)

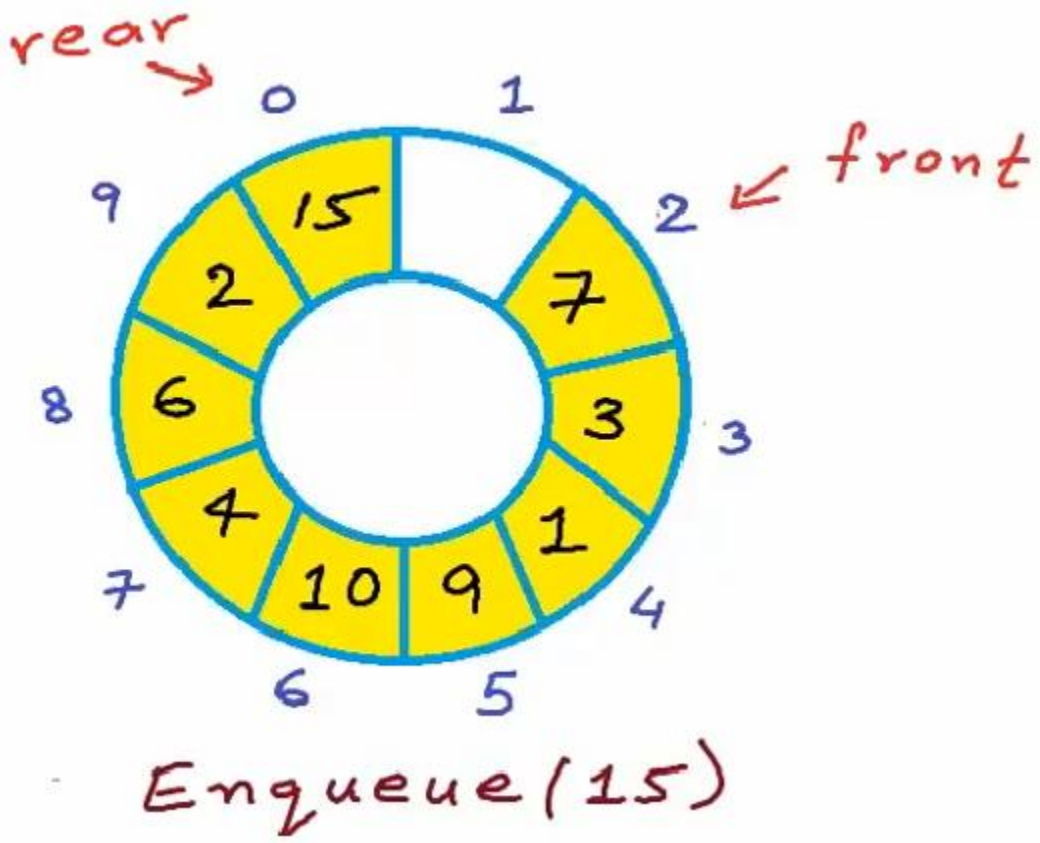
- Let us now try EnQueue function on our Queue.



PseudoCode for Queue using Circular Array (6/9)

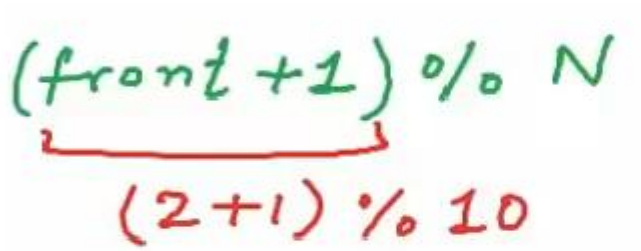
- Call EnQueue function.

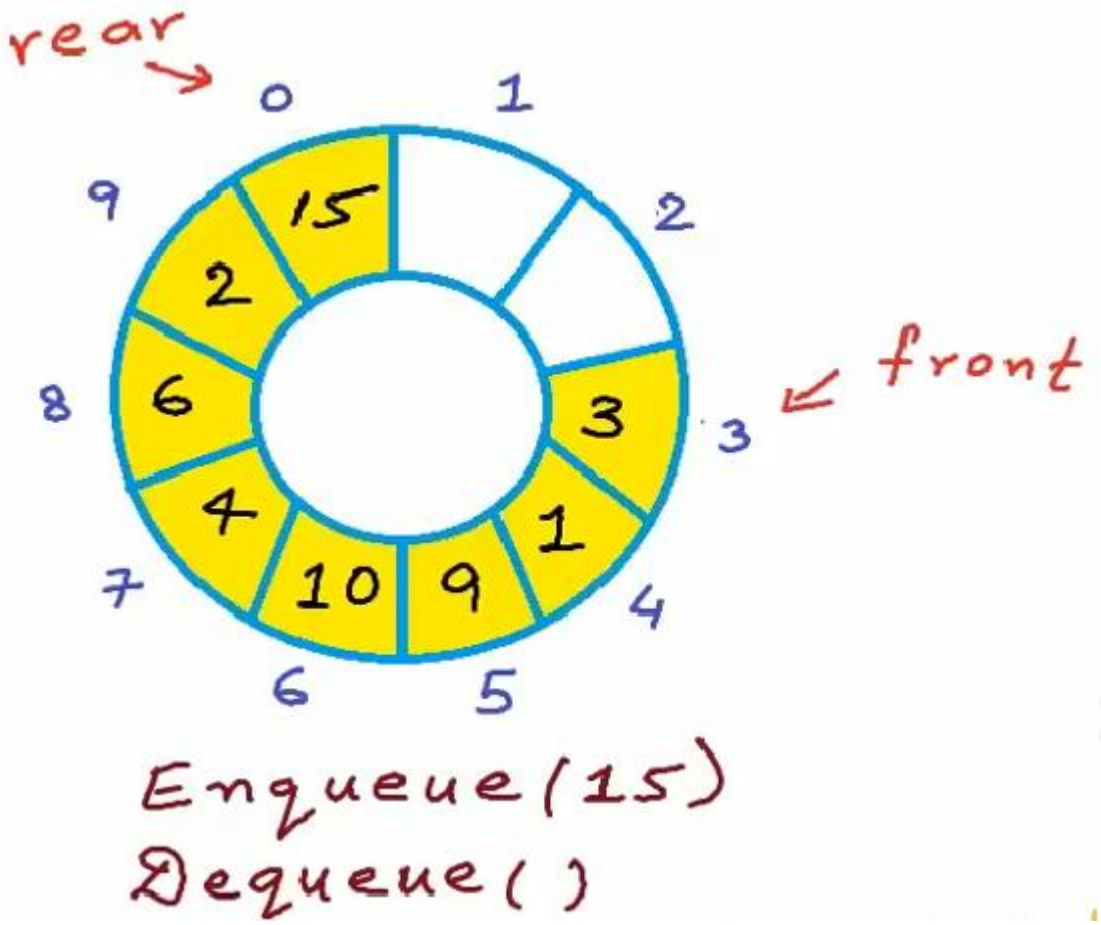
$$\underbrace{(rear+1)\,\%\,N}_{(9+1)\,\%\,10 = 0}$$



PseudoCode for Queue using Circular Array (7/9)

- Call DeQueue function.





PseudoCode for Queue using Circular Array (8/9)

- Now, front operation will be straight forward.
- We only need to return the element at front index.

```
front()
{
    return A[front]
}
```

PseudoCode for Queue using Circular Array (9/9)

- All the previous functions will take constant time, the time complexity will be O(1).
- We are performing simple arithmetic and assignments in the functions.
- What about the Queue Implementation using Linked list!
  - Think about it.

Any Questions???…