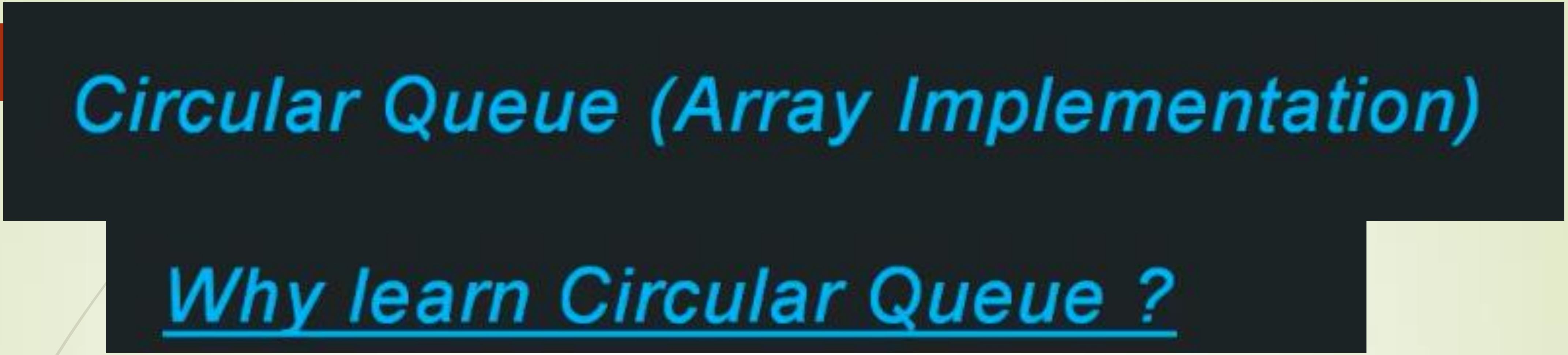
Data Structure and Algorithms

Session-12

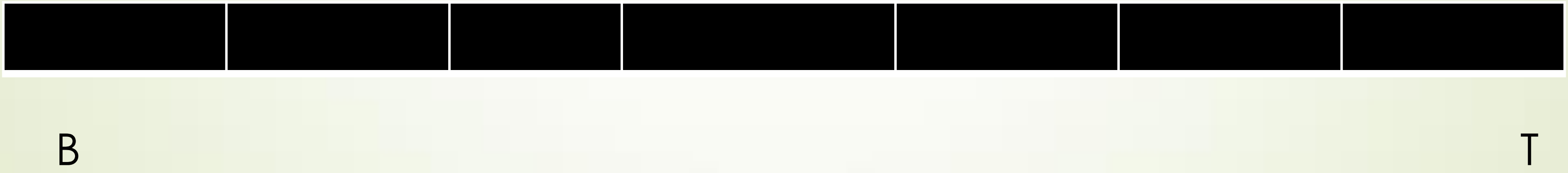
Dr. Subhra Rani Patra
SCOPE, VIT Chennai

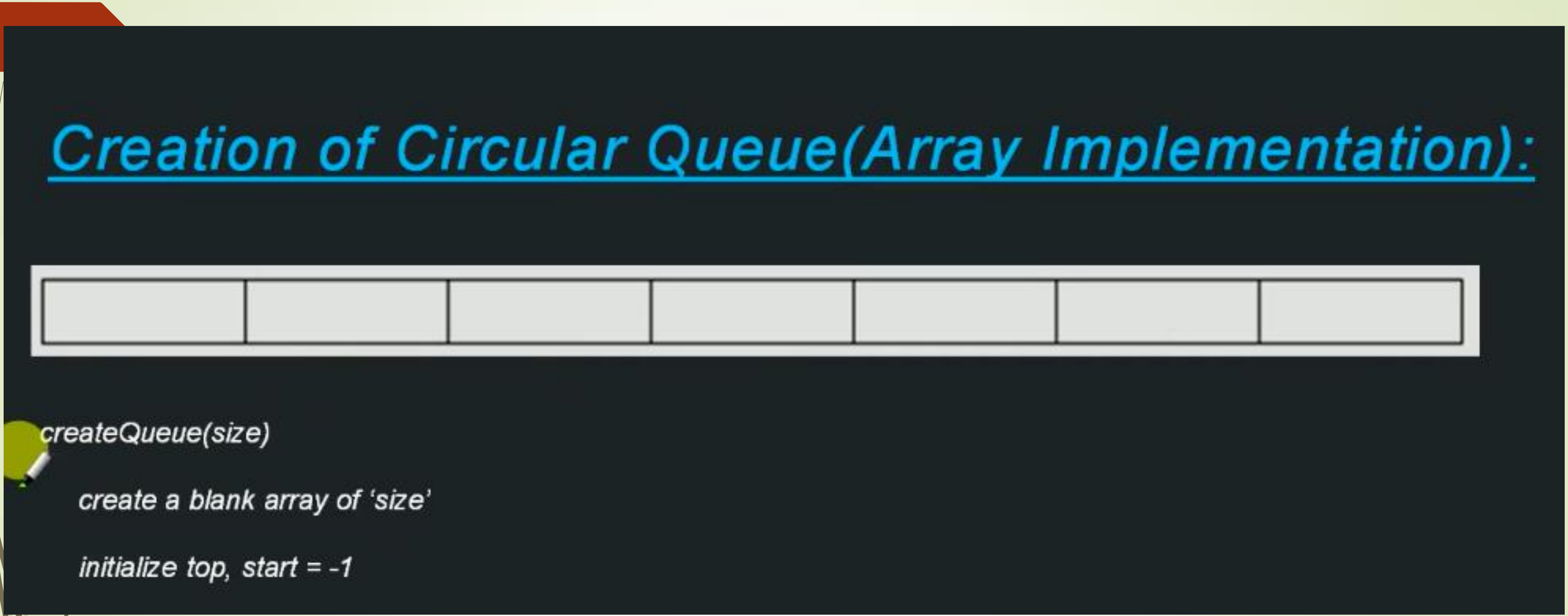


Circular Queue (Array Implementation)

Why learn Circular Queue ?

✓deQueue operation causes blank cells Linear Queue(Array Implementation). We need to improve that.

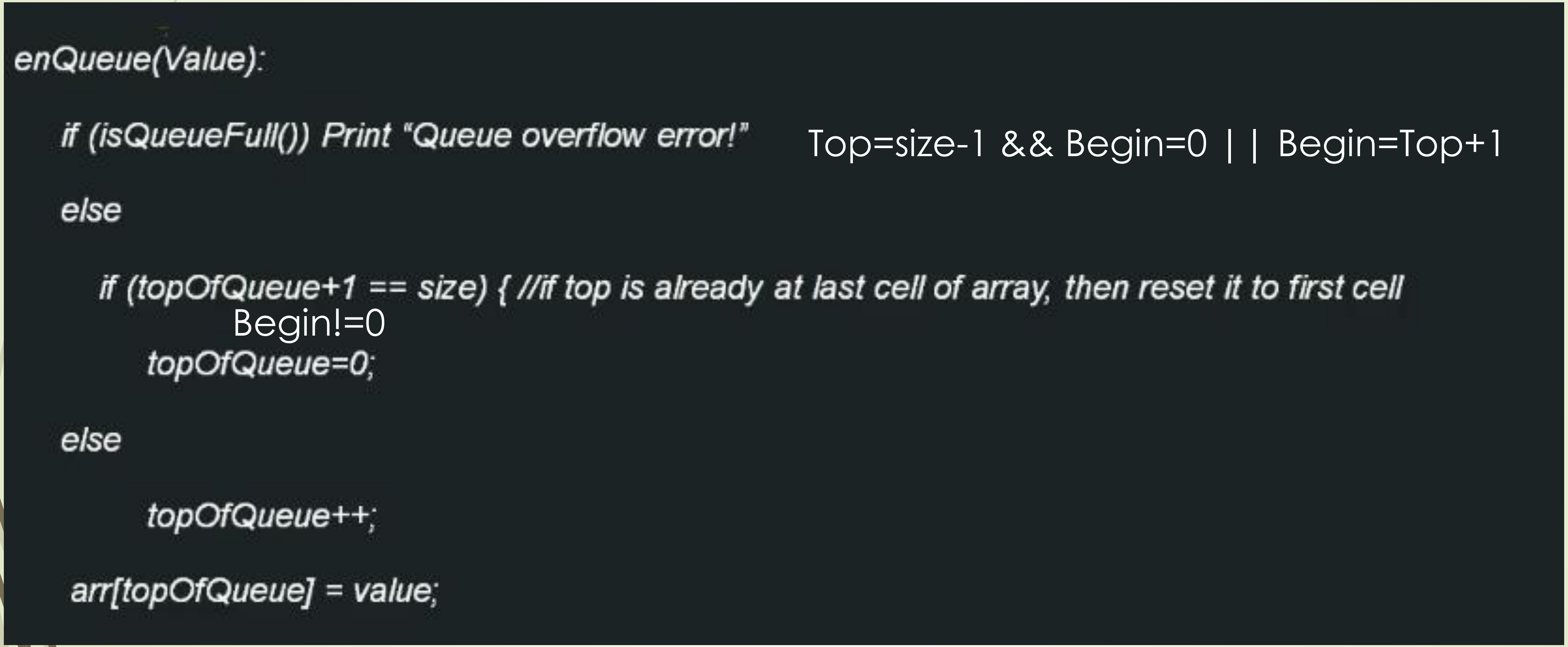




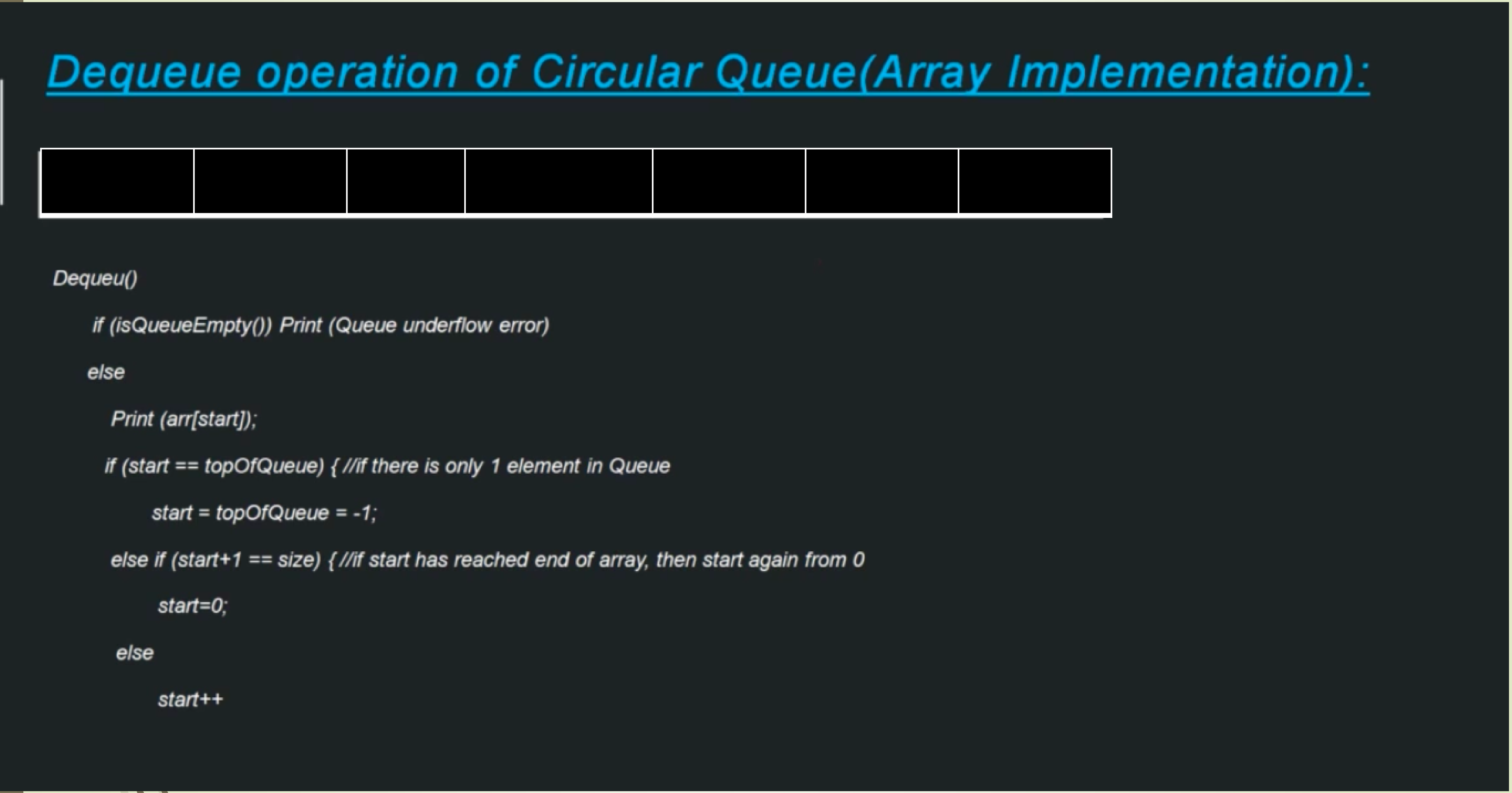
Creation of Circular Queue(Array Implementation):


```
createQueue(size)
    create a blank array of 'size'
    initialize top, start = -1
```

How Circular Queue works :



```
enQueue(Value):

    if (isQueueFull()) Print "Queue overflow error!"      Top=size-1 && Begin=0 || Begin=Top+1

    else

        if (topOfQueue+1 == size) { //if top is already at last cell of array, then reset it to first cell
                    Begin!=0
            topOfQueue=0;

    else

            topOfQueue++;

     arr[topOfQueue] = value;
```



Dequeue operation of Circular Queue(Array Implementation):

```
Dequeu()
    if (isQueueEmpty()) Print (Queue underflow error)
   else
      Print (arr[start]);
     if (start == topOfQueue) { //if there is only 1 element in Queue
           start = topOfQueue = -1;
      else if (start+1 == size) { //if start has reached end of array, then start again from 0
            start=0;
      else
            start++
```

Peek operation of Circular Queue(Array Implementation):

10	20	30	40			

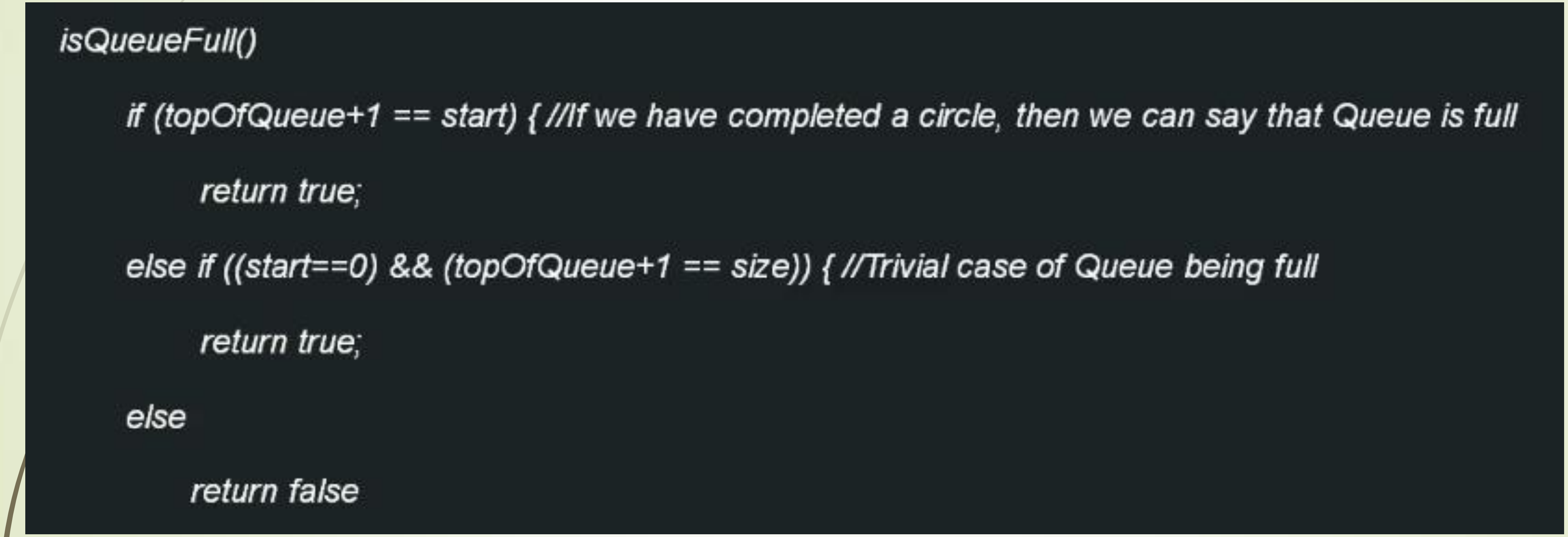
```
peek()
 if (!isQueueEmpty())
        print(arr[start])
 else
     print ("The queue is empty!!")
```

IsEmpty operation of Circular Queue(Array Implementation

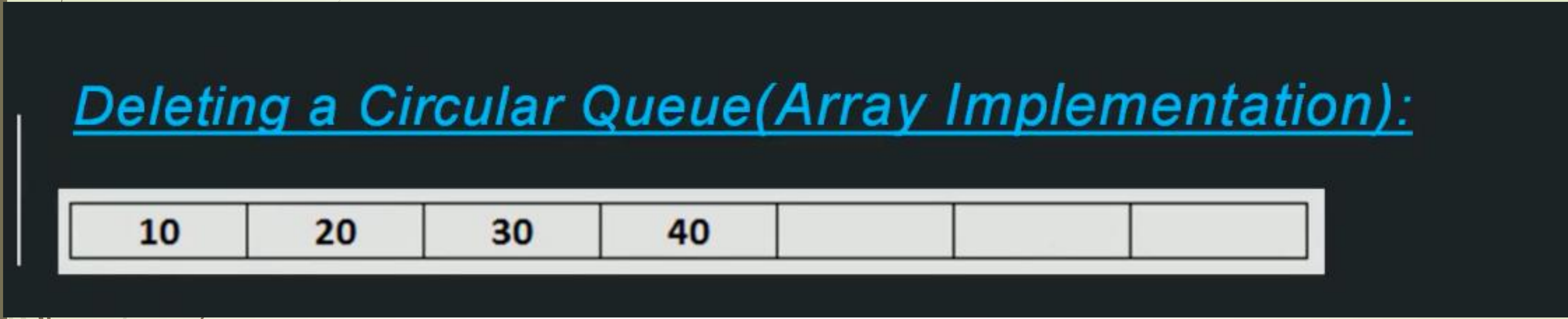
```
IsQueueEmpty():
    if (topOfQueue == -1)
          return true;
   else
         return false;
```

IsFull operation of Circular Queue(Array Implementation):

10	20	30	40	50	60	70

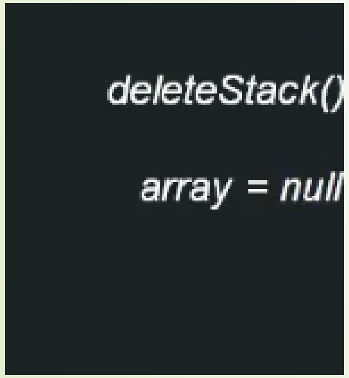


```
isQueueFull()

    if (topOfQueue+1 == start) { //If we have completed a circle, then we can say that Queue is full

        return true;

    else if ((start==0) && (topOfQueue+1 == size)) { //Trivial case of Queue being full

        return true;

    else

        return false
```



Deleting a Circular Queue(Array Implementation):

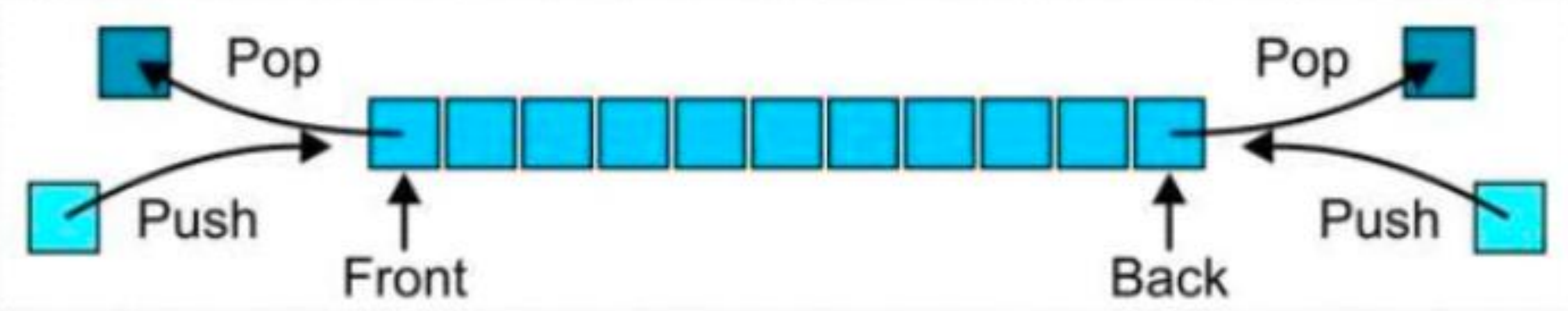
10	20	30	40			

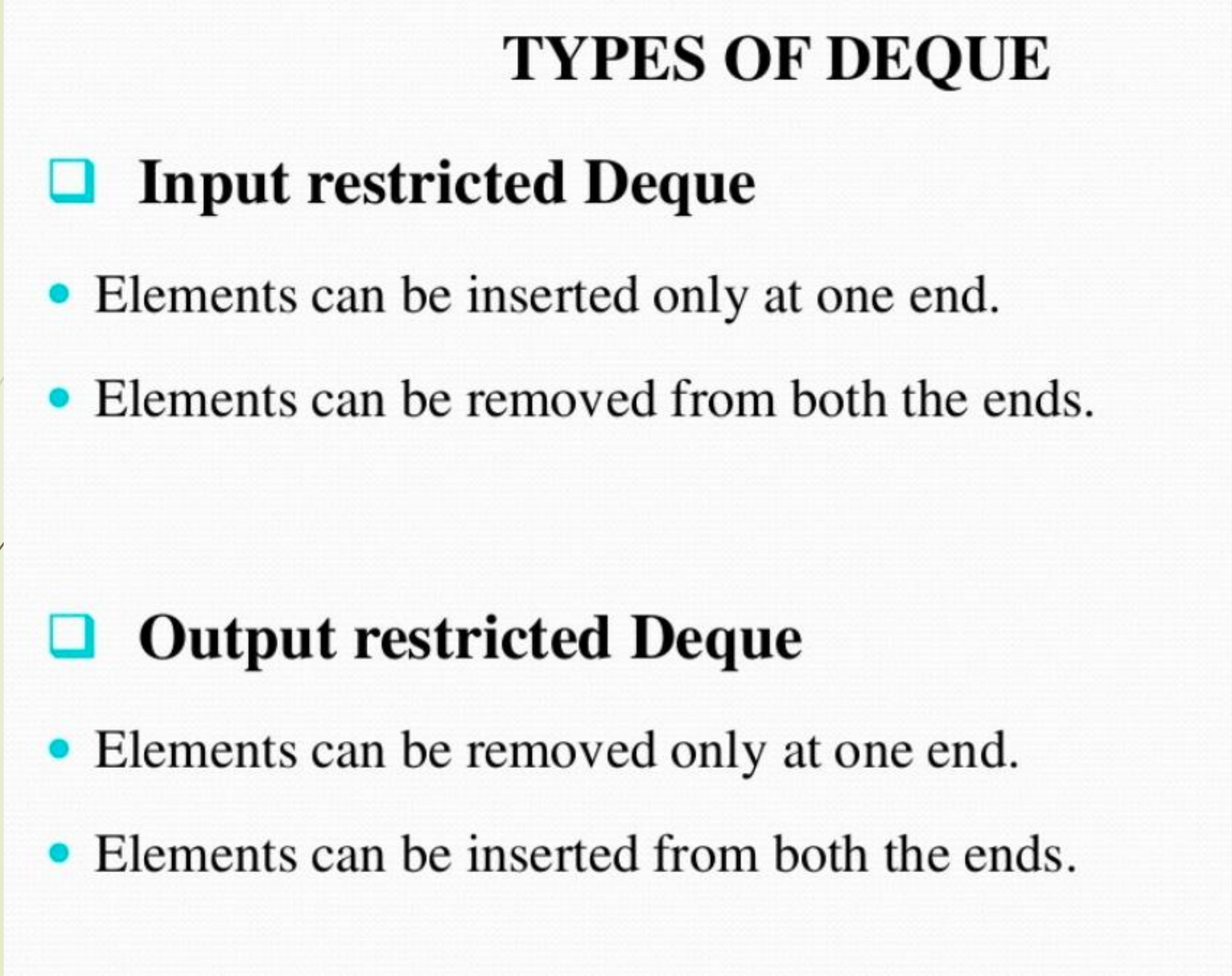


```
deleteStack()
  array = null
```

Double-Ended Queue

- A Deque or deck is a double-ended queue.
- Allows elements to be added or removed on either the ends.





TYPES OF DEQUE

□ Input restricted Deque

- Elements can be inserted only at one end.
- Elements can be removed from both the ends.

□ Output restricted Deque

- Elements can be removed only at one end.
- Elements can be inserted from both the ends.

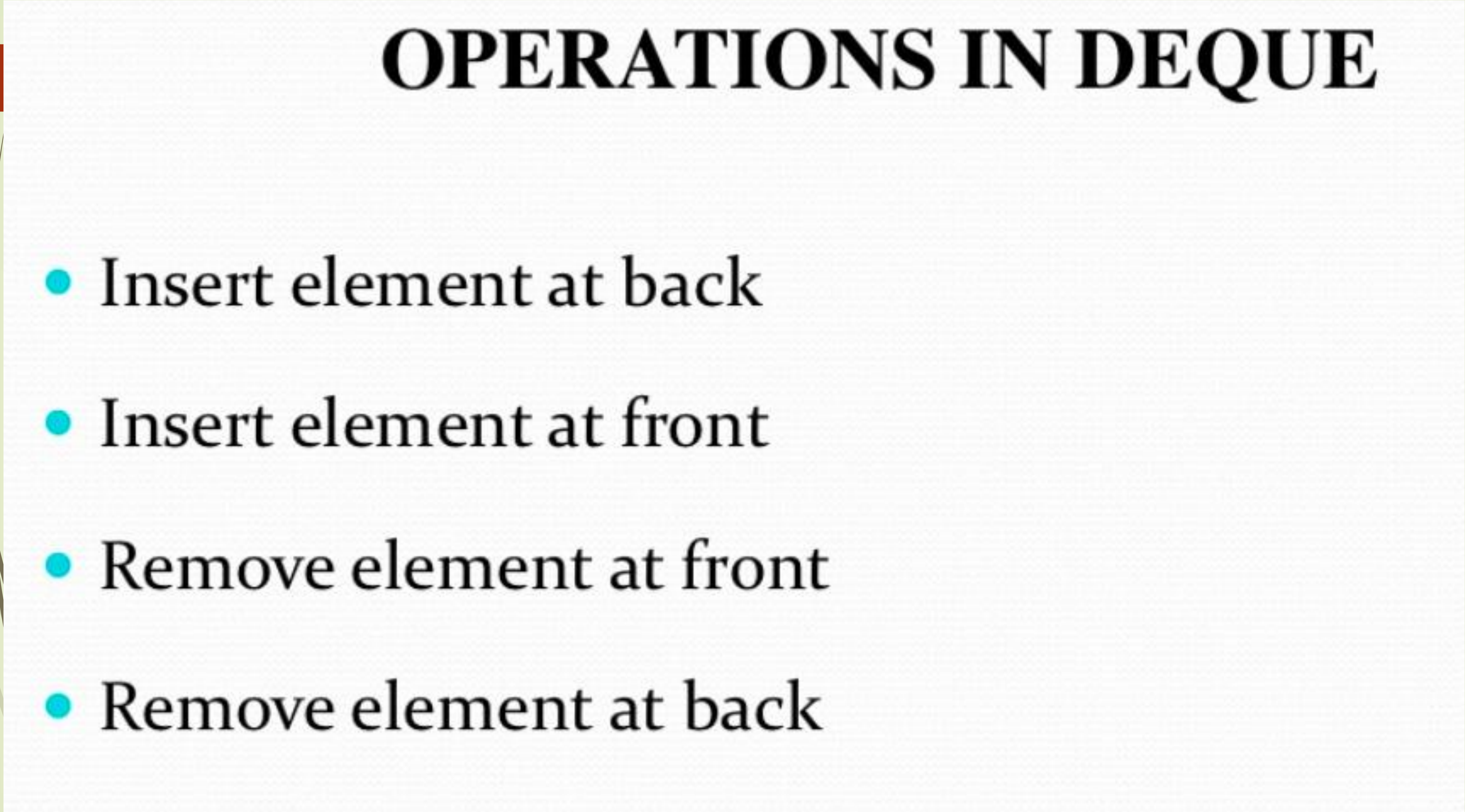
Deque as Stack and Queue

As STACK

- When insertion and deletion is made at the same side.

As Queue

- When items are inserted at one end and removed at the other end.

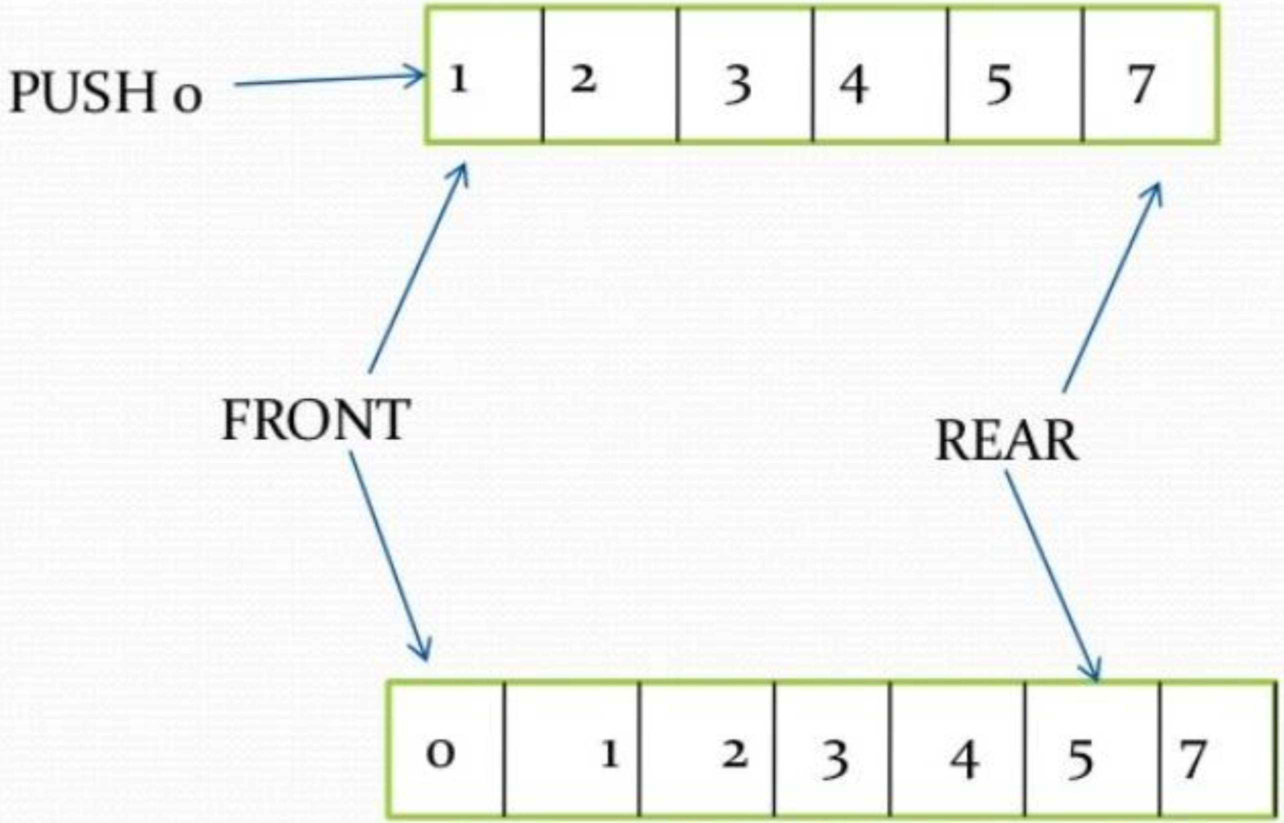


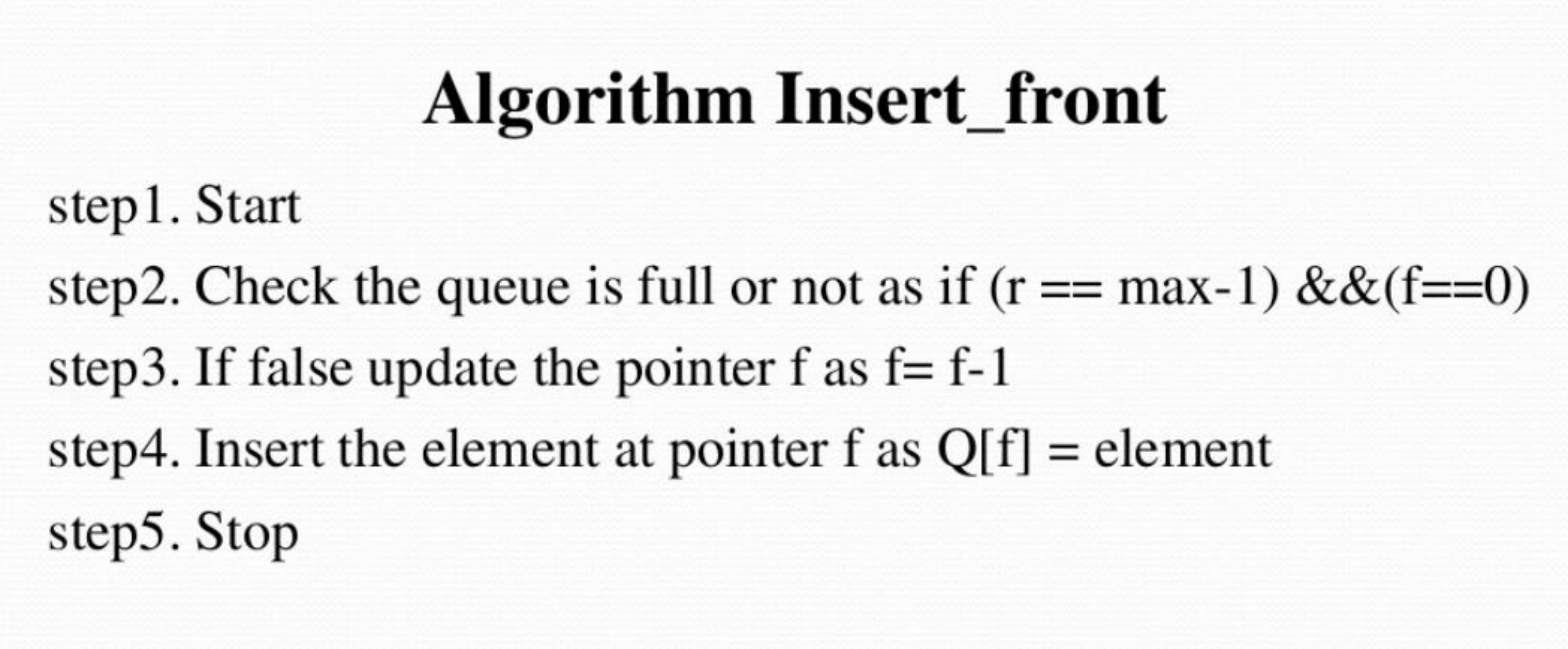
OPERATIONS IN DEQUE

- Insert element at back
- Insert element at front
- Remove element at front
- Remove element at back

Insert_front

- insert_front() is a operation used to push an element into the front of the *Deque*.



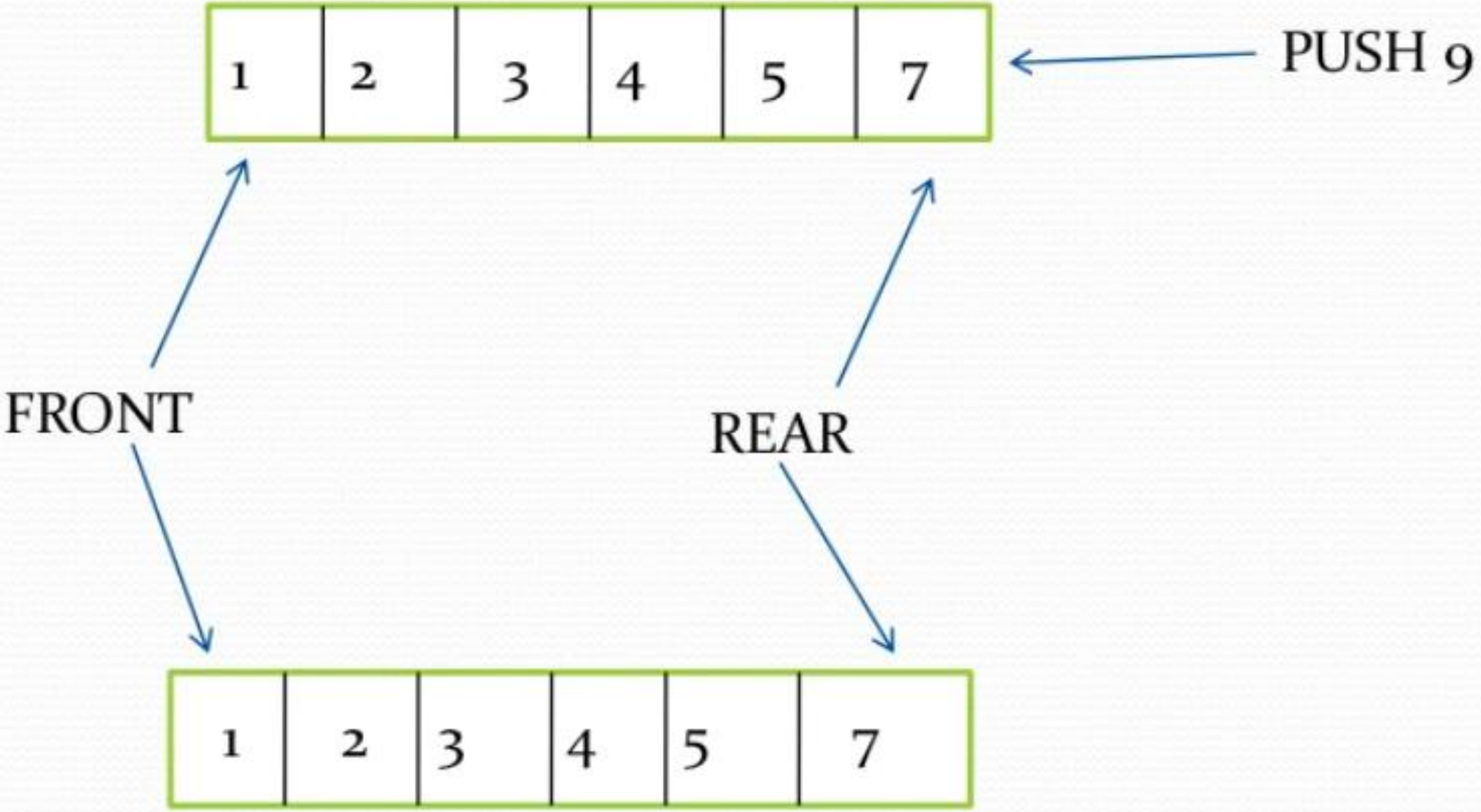


Algorithm Insert_front

```
step1. Start
step2. Check the queue is full or not as if (r == max-1) &&(f==0)
step3. If false update the pointer f as f= f-1
step4. Insert the element at pointer f as Q[f] = element
step5. Stop
```

Insert_back

- insert_back() is a operation used to push an element at the back of a *Deque*.



Alogrithm insert_back

Step1: Start

Step2: Check the queue is full or not as if (r == max-1) &&(f==0) if yes queue is full

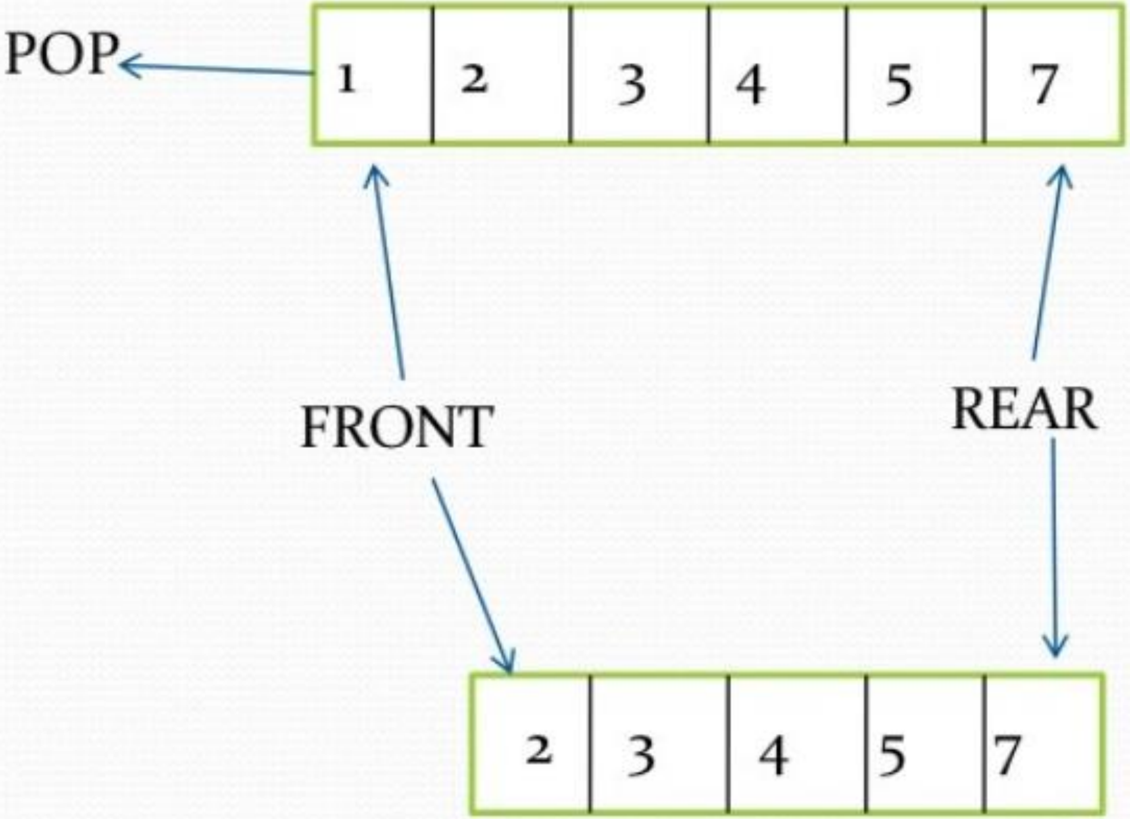
Step3: If false update the pointer r as r= r+1

Step4: Insert the element at pointer r as Q[r] = element

Step5: Stop

Remove_front

- remove_front() is a operation used to pop an element on front of the *Deque*.

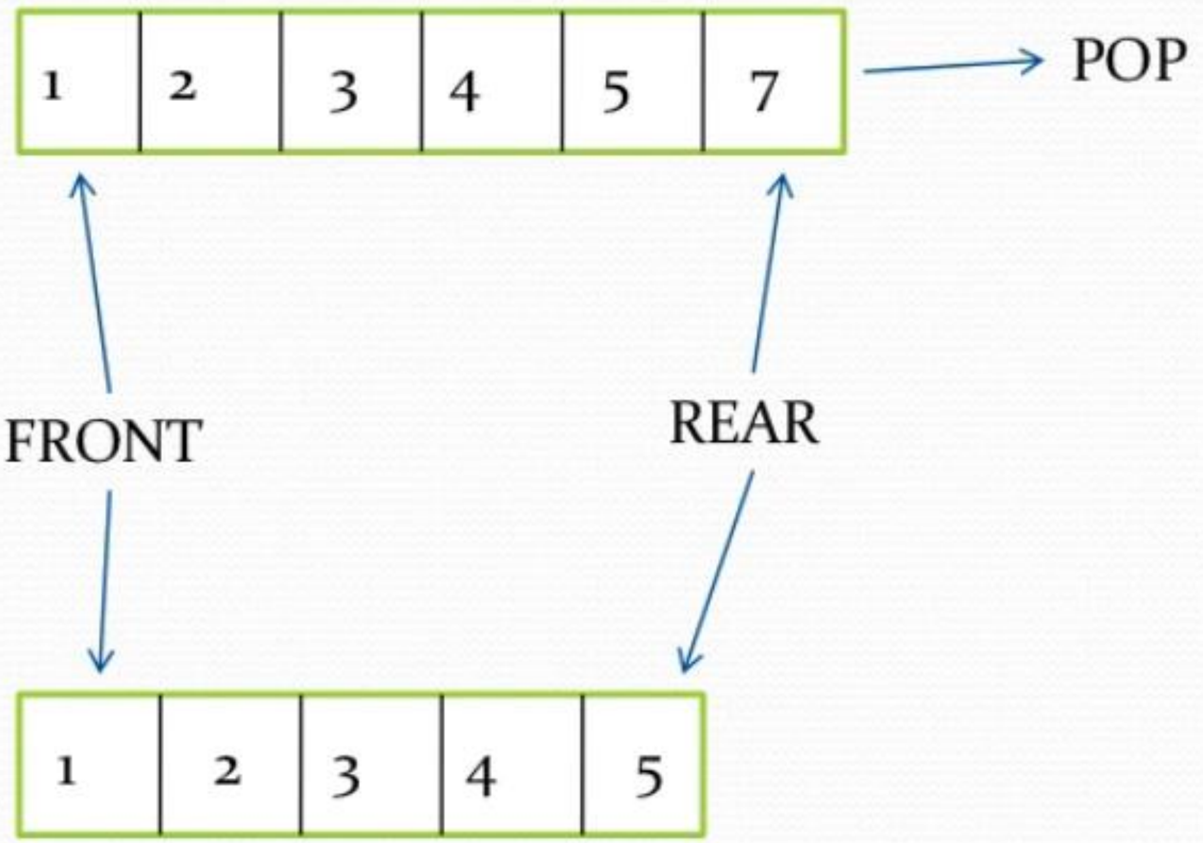


Alogrithm Remove_front

```
Step1: Start
Step2: Check the queue is empty or not as if (f == r) if yes queue is empty.
Step3: If false update pointer f as f = f+1 and delete element at position f as element = Q[f]
Step4: If ( f== r) reset pointer f and r as f = r = -1
Step5: Stop
```

Remove_back

- remove_back() is a operation used to pop an element on back of the *Deque*.



Alogrithm Remove_back

step1. Start

step2. Check the queue is empty or not as if (f == r) if yes queue is empty

step3. If false delete element at position r as element = Q[r]

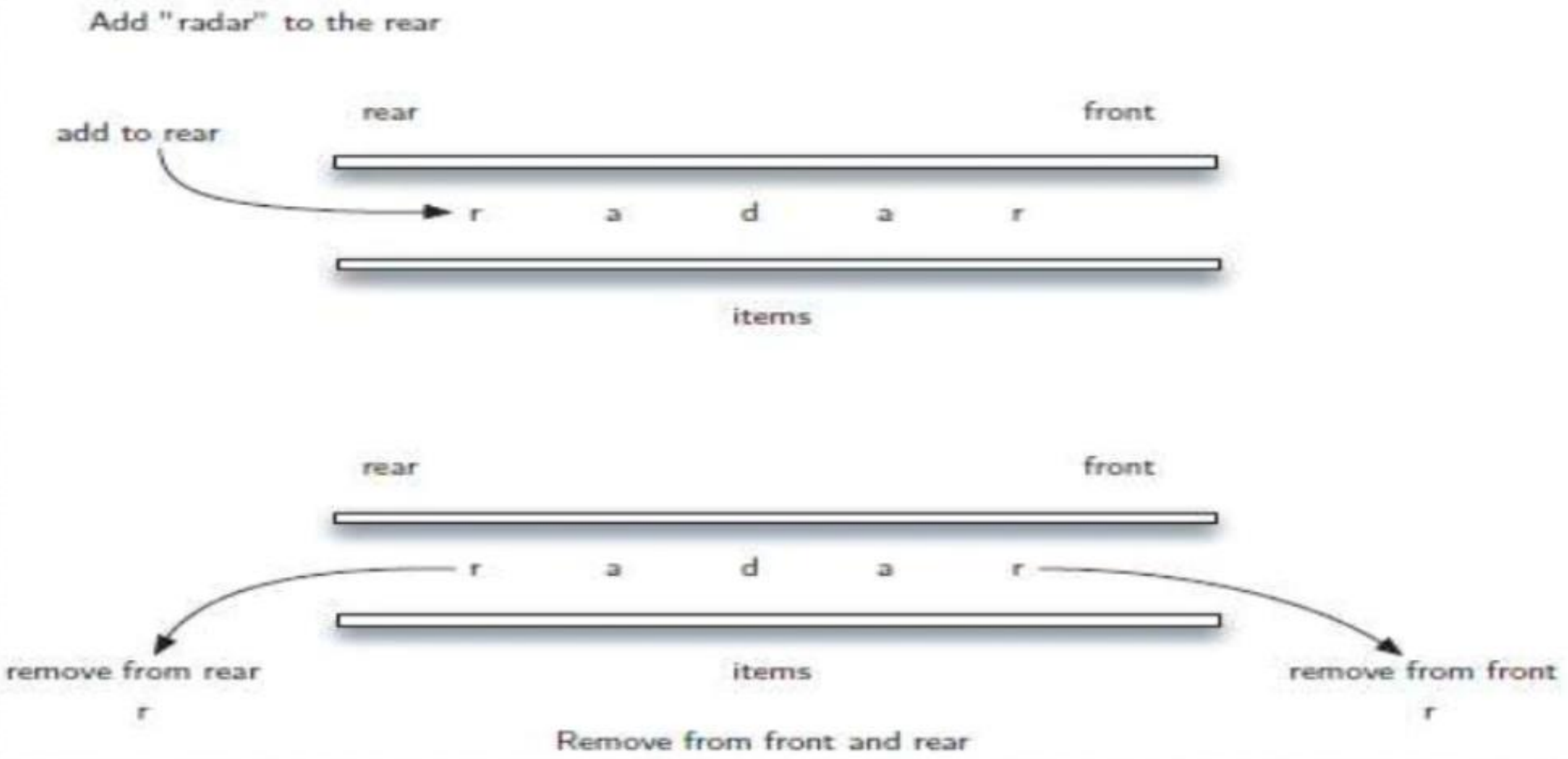
step4. Update pointer r as r = r-1

step5. If (f == r) reset pointer f and r as f = r= -1

step6. Stop

APPLICATIONS OF DEQUE

Palindrome-checker



Priority Queue

- In priority queue, each element is assigned a priority.
- Priority of an element determines the order in which the elements will be processed.
- Rules:
  1. An element with higher priority will processed before an element with a lower priority.
  2. Two elements with the same priority are processed on a First Come First Serve basis.

Types of Priority Queue

1. Ascending Priority Queue

   In this type of priority queue, elements can be inserted into any order but only the smallest element can be removed.

2. Descending Priority Queue

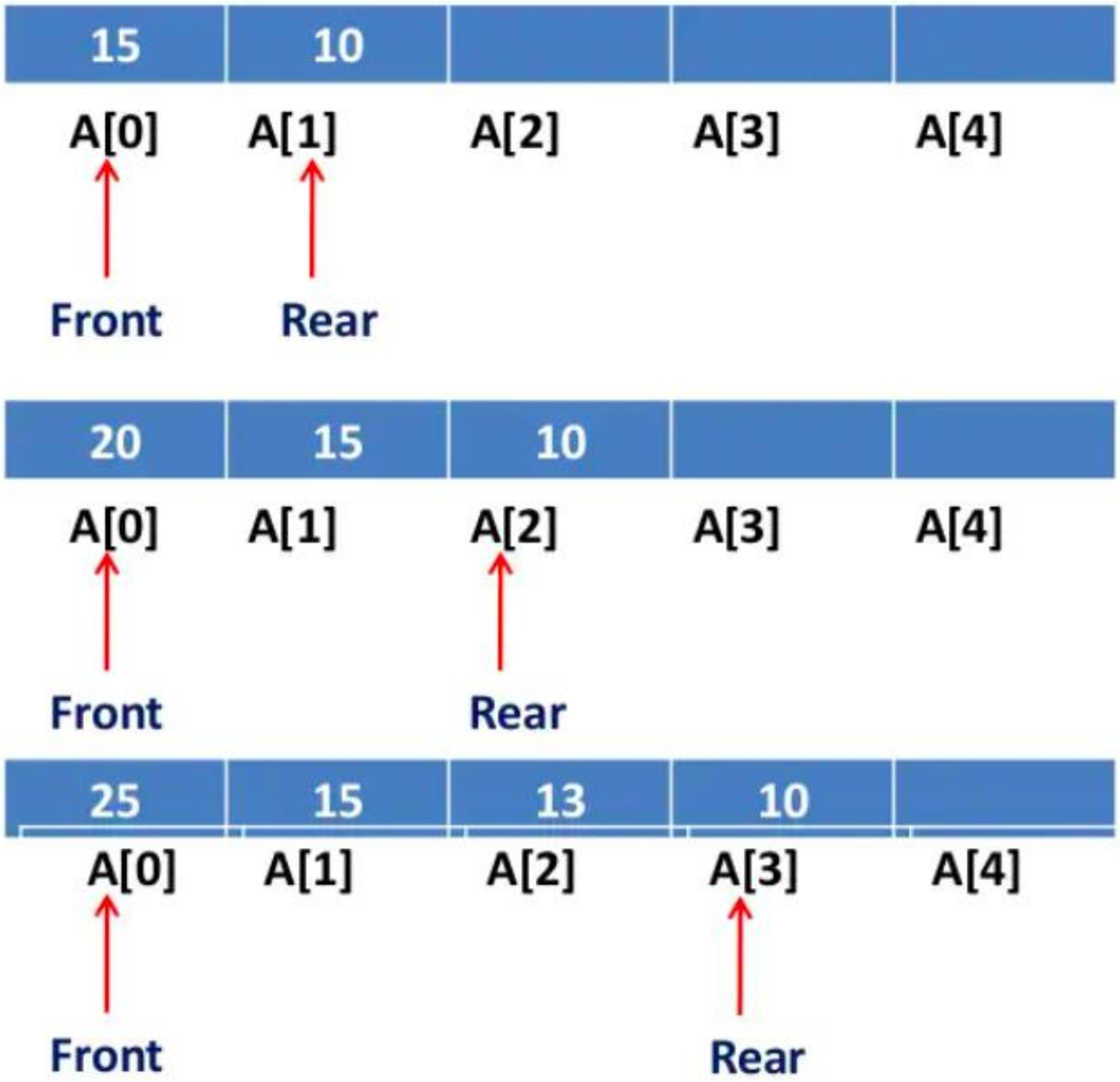
   In this type of priority queue, elements can be inserted into any order but only the largest element can be removed.

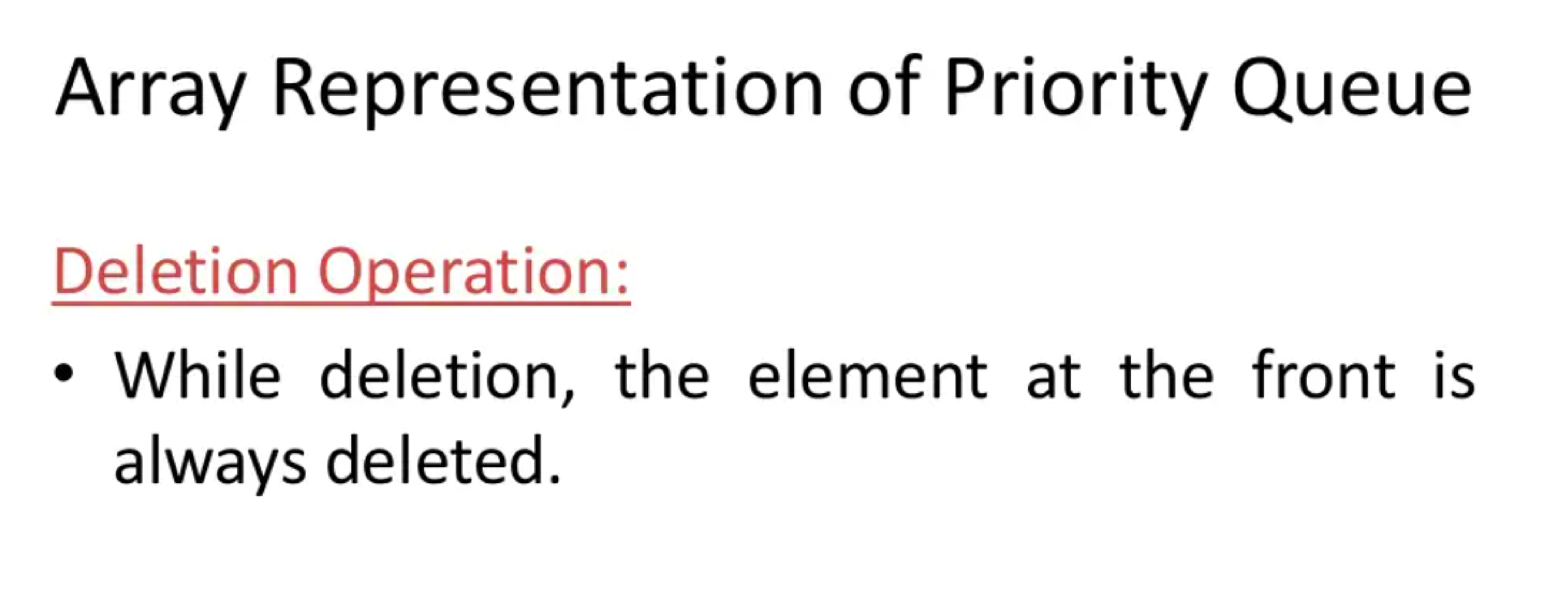
Array Representation of Priority Queue

Insertion Operation:

- While inserting elements in priority queue we will add it at the appropriate position depending on its priority
- It is inserted in such a way that the elements are always ordered either in Ascending or descending sequence

Array Representation of Priority Queue



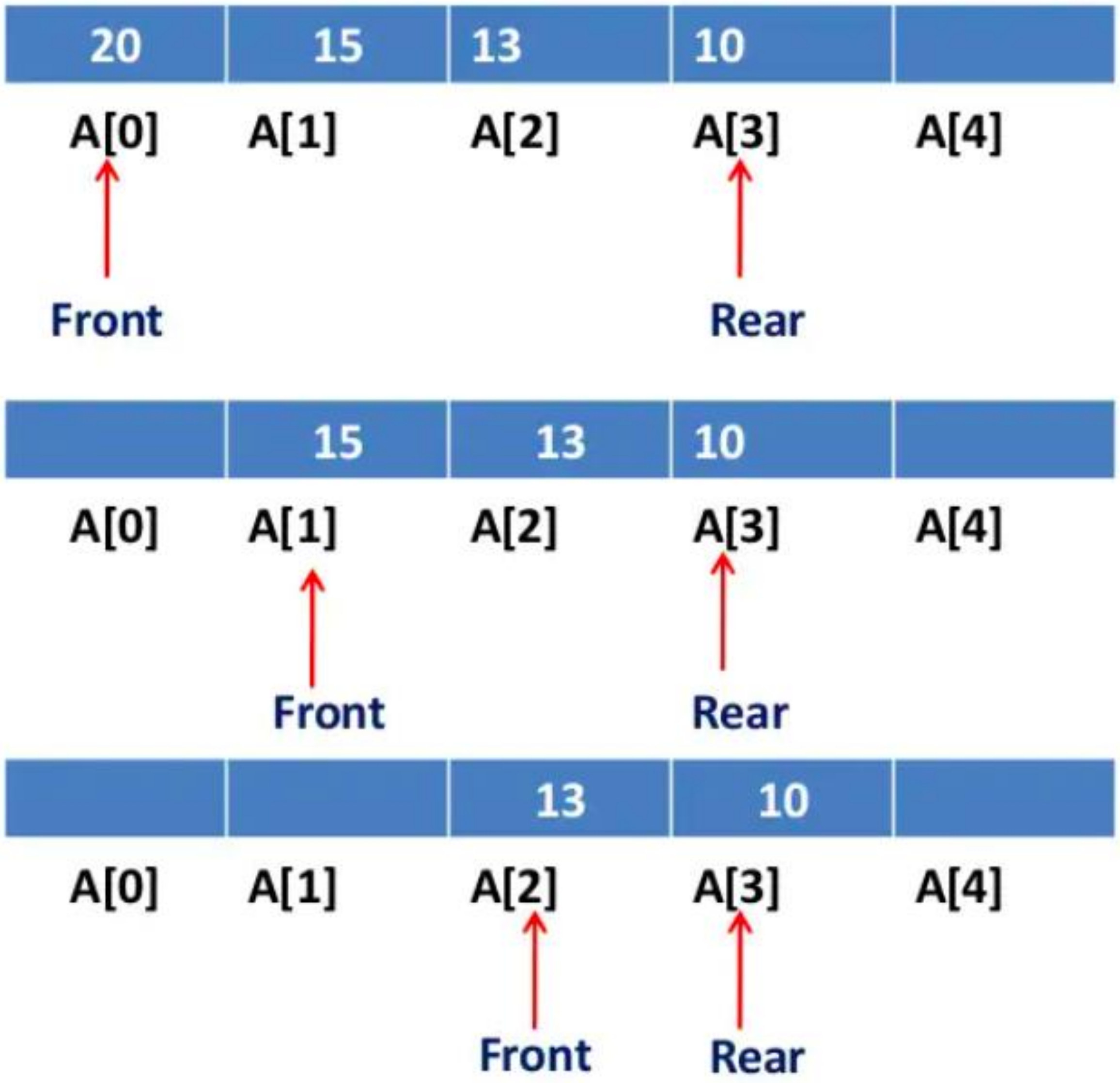


Array Representation of Priority Queue

Deletion Operation:

- While deletion, the element at the front is always deleted.

Array Representation of Priority Queue



20	15	13	10	
A[0]	A[1]	A[2]	A[3]	A[4]

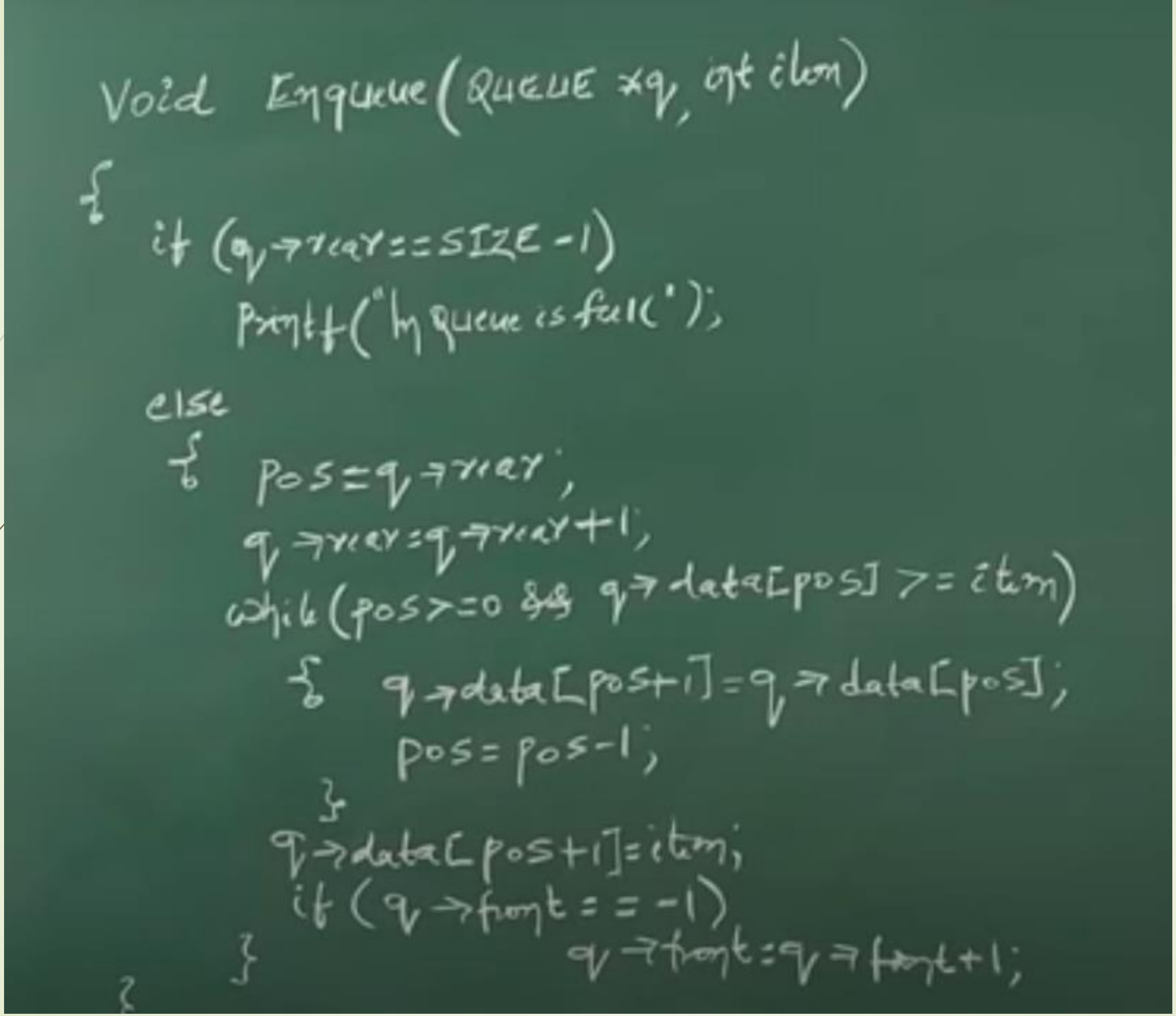
Front Rear

	15	13	10	
A[0]	A[1]	A[2]	A[3]	A[4]

Front Rear

		13	10	
A[0]	A[1]	A[2]	A[3]	A[4]

Front Rear



```
Void Enqueue(QUEUE *q, int item)
{
  if (q->rear==SIZE-1)
       Printf("\n queue is full");
  else
   {  Pos=q->rear;
      q->rear=q->rear+1;
      while(pos>=0 && q->data[pos] >= item)
        {  q->data[pos+1]=q->data[pos];
           pos=pos-1;
        }
      q->data[pos+1]=item;
      if (q->front==-1)
               q->front=q->front+1;
   }
}
```

Thank you