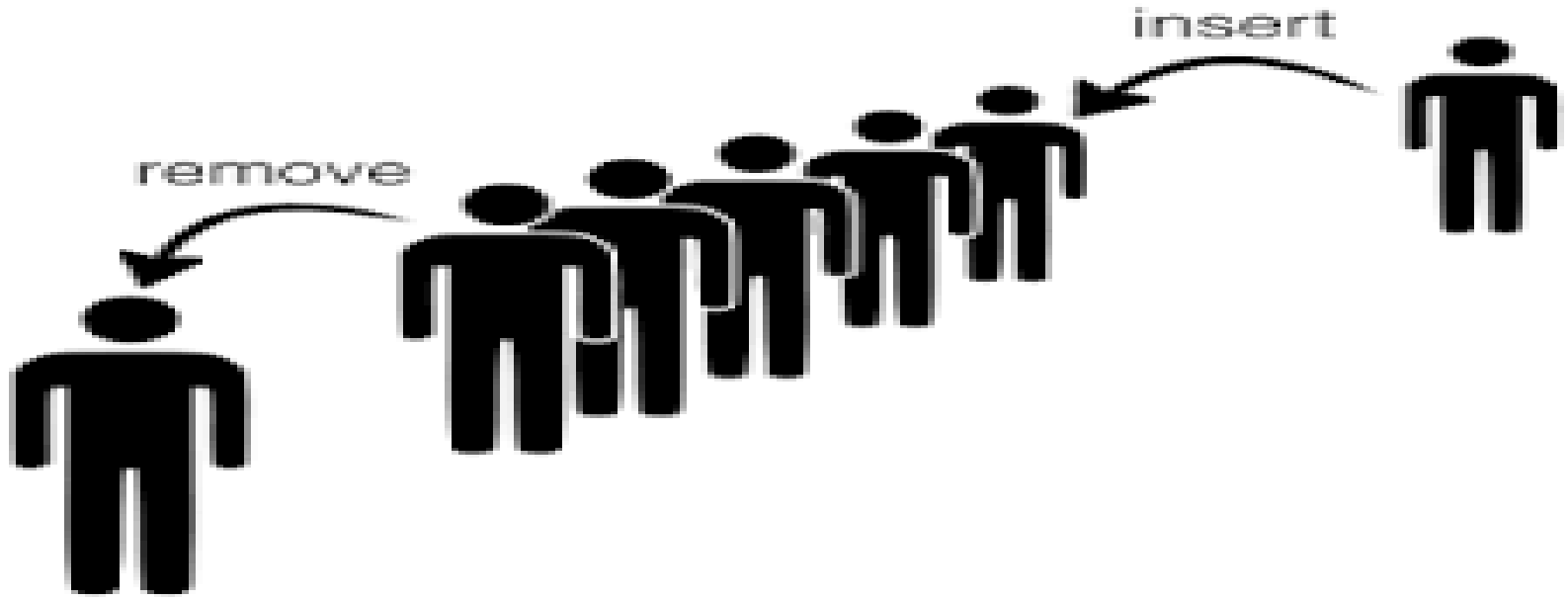
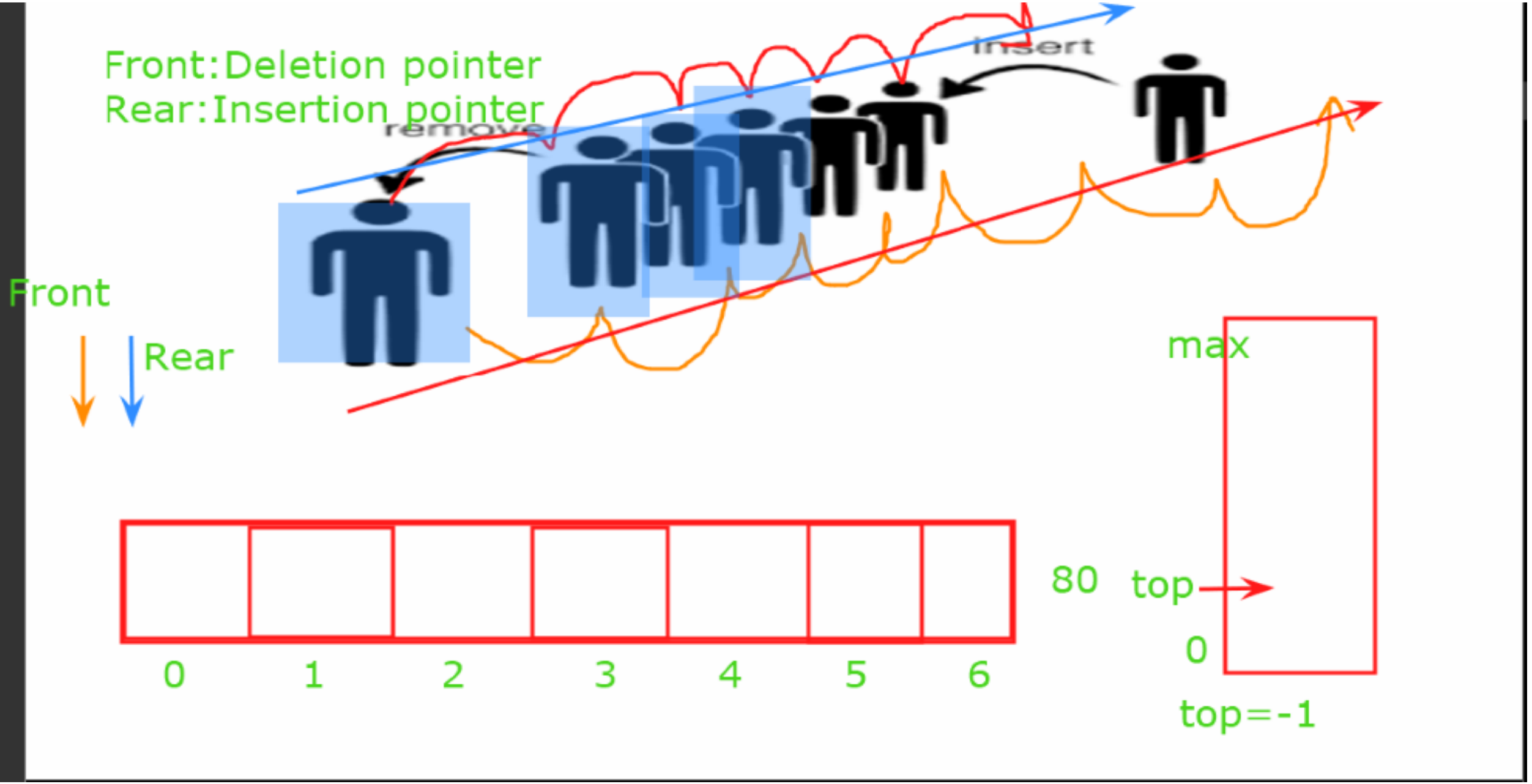
Algorithms & Data Structure
Day 9 : Queue

Kiran Waghmare


insert
remove


In
Data
Data
Data
Data
Data
Data
Out
Last In Last Out
First In First Out

Queue


Front:Deletion pointer
Rear:Insertion pointer
remove
insert
Front
Rear
0
1
2
3
4
5
6
max
80
top
0
top=-1

Who can see what you share here? Recording On

Queue:

-linear, ordered collection, homogeneous in nature

-non-primitive data type

-using 2 pointer:

-Rear pointer: Insertion operation: enqueue

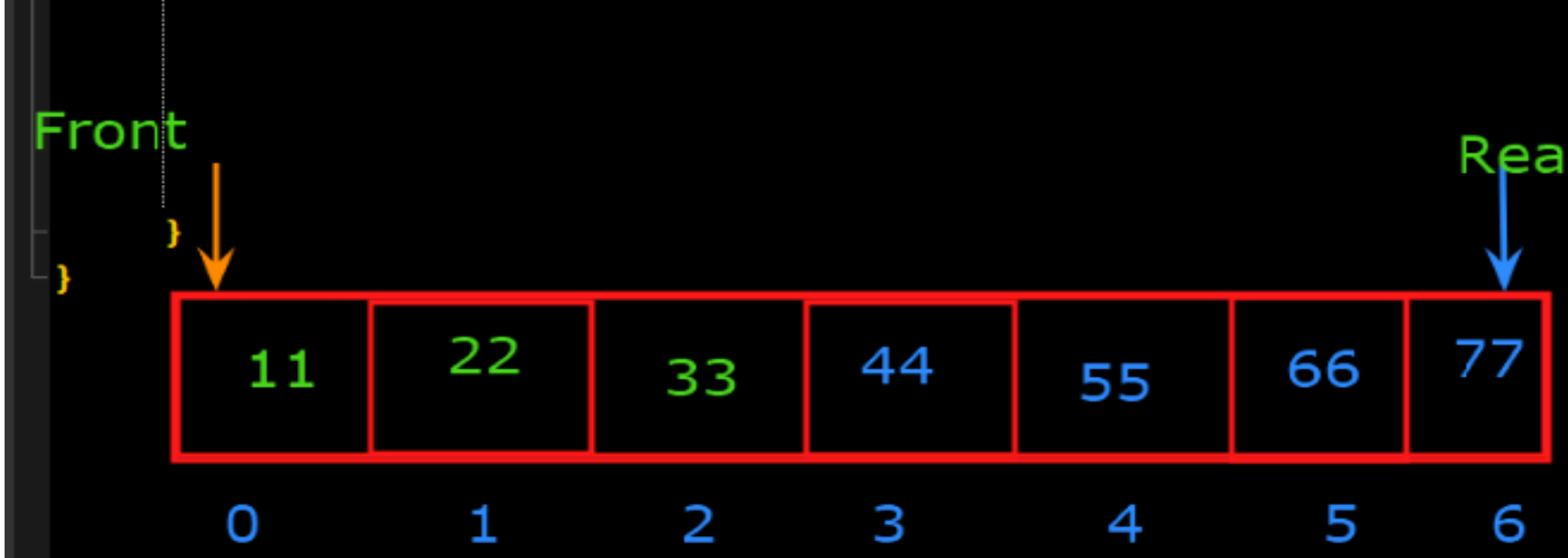
-Front pointer: Deletion operation: dequeue

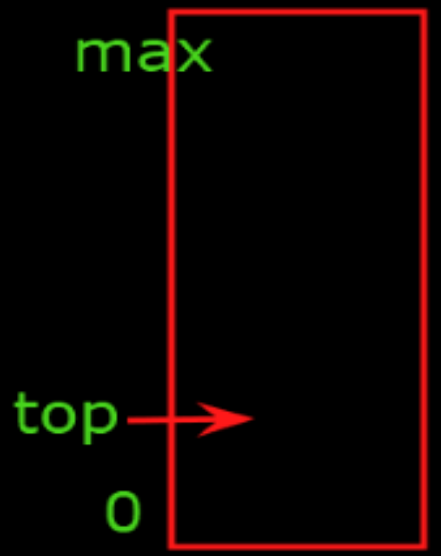
Front:Deletion pointer

Rear:Insertion pointer

```
    }

    boolean isFull()
    {
        if(front==0 && rear == max-1)
            return true;
    }

    public static void main(String args[])
    {


    }
}
```

Front:Deletion pointer
Rear:Insertion pointer





```
        System.out.println(rear+" => Rear pointer");
        System.out.println(front+" => Front pointer");

    }


    public static void main(String args
    {

        Queue q = new Queue();

        q.enqueue(10);
        q.enqueue(20);
        q.enqueue(30);
        q.display();

        System.out.println(" ");
        q.dequeue();
        q.display();


    }
}
```

Who can see what you share here? Recording On

C:\Windows\System32\cmd.exe

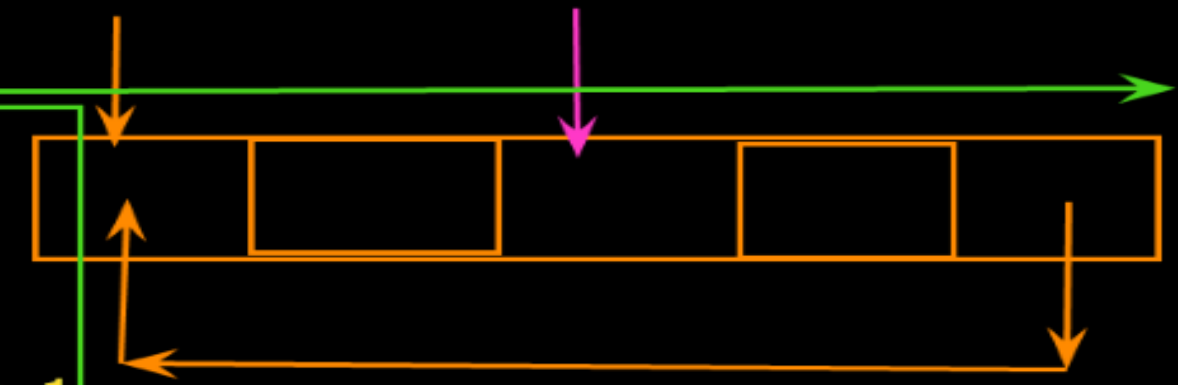
```
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Test>javac Queue.java

C:\Test>java Queue
Insertion done !!!
Insertion done !!!
Insertion done !!!
10
20
30
2 => Rear pointer
0 => Front pointer

Deleted element = 10
20
30
2 => Rear pointer
1 => Front pointer

C:\Test>
```

```
        System.out.println(rear+" => Rear pointer");
        System.out.println(front+" => Front pointer");

    }


    public static void main(String args
    {

        Queue q = new Queue();

        q.enqueue(10);
        q.enqueue(20);
        q.enqueue(30);
        q.enqueue(40);
        q.enqueue(50);
        q.enqueue(60);
        q.display();

        System.out.println(" ");
        q.dequeue();
        q.display();


    }
}
```

```
C:\Windows\System32\cmd.exe
30
2 => Rear pointer
1 => Front pointer

C:\Test>javac Queue.java

C:\Test>java Queue
Insertion done !!!
Insertion done !!!
Insertion done !!!
Insertion done !!!
Insertion done !!!
Queue is Full !!!
10 20 30 40 50 4 => Rear pointer
0 => Front pointer

Deleted element = 10
20 30 40 50 4 => Rear pointer
1 => Front pointer

C:\Test>
```

Circulr Queue:

Who can see what you share here? Recording On



circular Queue is Full !!

front=rear+1

front==0&&rear==max-1

rear + 1 == Max=> Overflow!

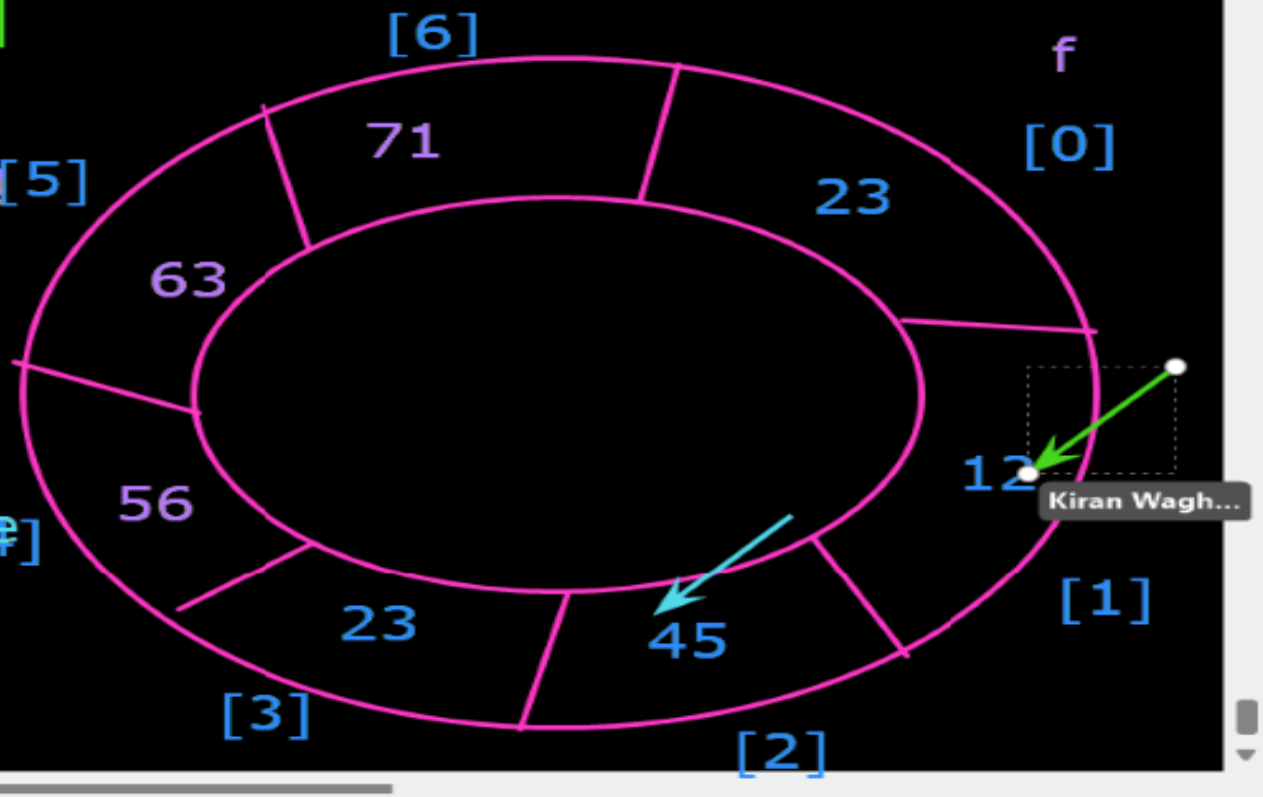
rear=(rear+1)%max

=(6+1)%7

=7%7

=0 start of my queue

front=(front+1)%max



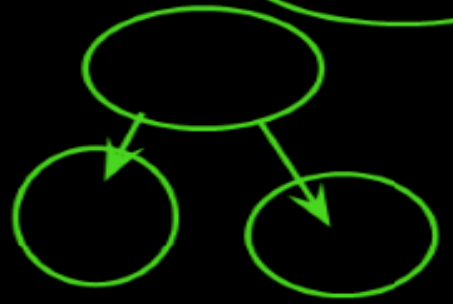
-Enqueue
-Dequeue
-Display
-isEmpty()
-isFull()

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo

Who can see what you share here? Recording On

1 2 3 4 5 6 7 8 =>smallest element (priority)

Heap



Input----------->Output

100	100
2	2 100
76	2 67 100
54	2 54 67 100
205	2 54 67 100 205

~~205~~ 100 ~~67~~ 54 2

Min-max tree
deaps
binomial

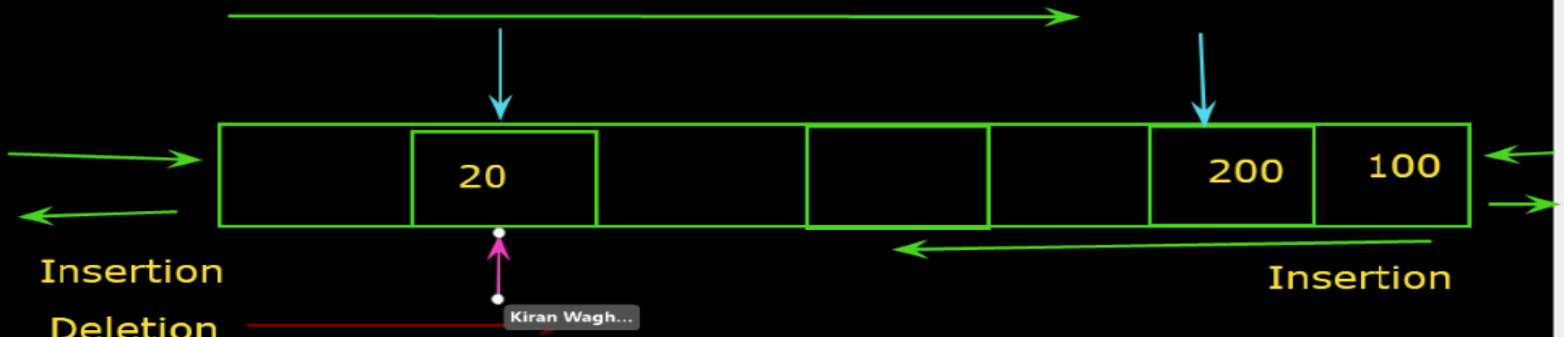
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo

Who can see what you share here? Recording On

Double Ended Queue (Deque):

1.Input Restricted Deque:
-Input is restricted at a single end but it allows deletion at both the ends.

2.Output Restricted Deque:
-Outpu is restricted at a single end but it allows insertion at both the ends.



FIFO

1.Input Restricted Deque

2.Output Restricted Deque

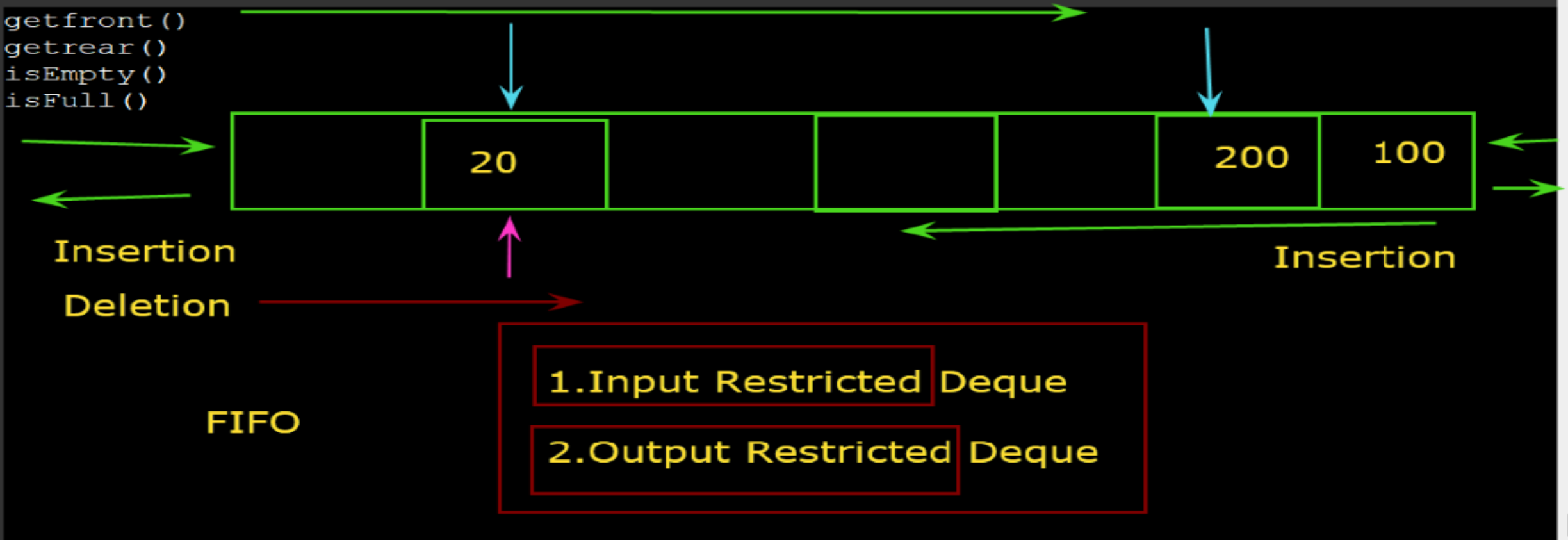
Who can see what you share here? Recording On

Operations on Deque:

insertfront():adds an element at the front.
insertlast():adds an element at the rear.

deletefront():deletes an element from the front.
deletelast():deletes an element from the rear.

getfront()
getrear()
isEmpty()
isFull()



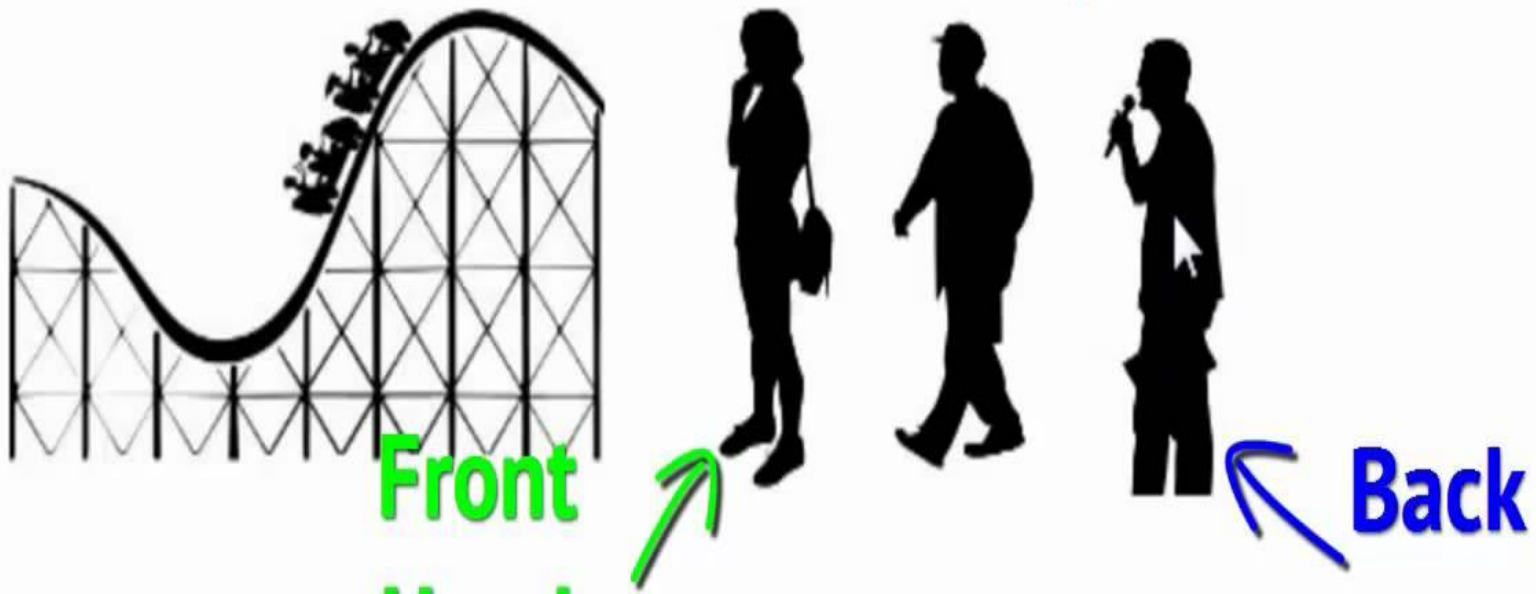
QUEUE

FIFO (First In First Out)

Queue

Size = 3

Enqueue



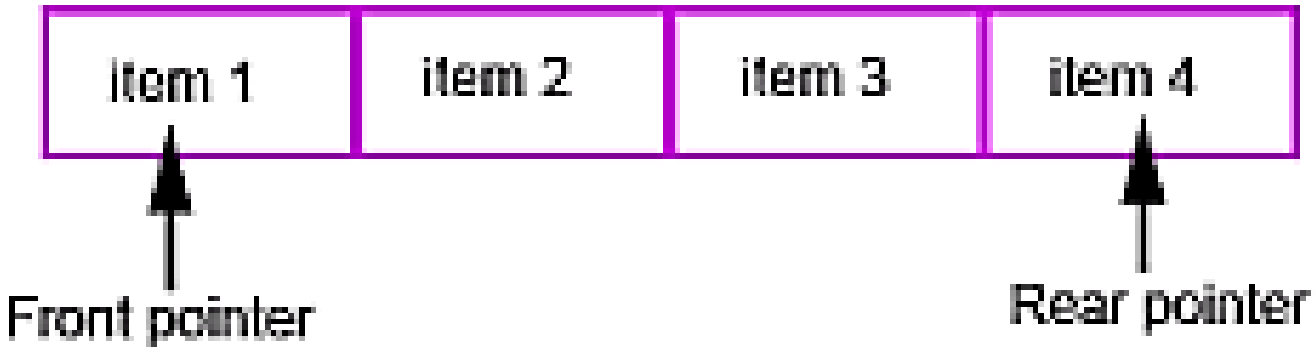
Front

Head

Back

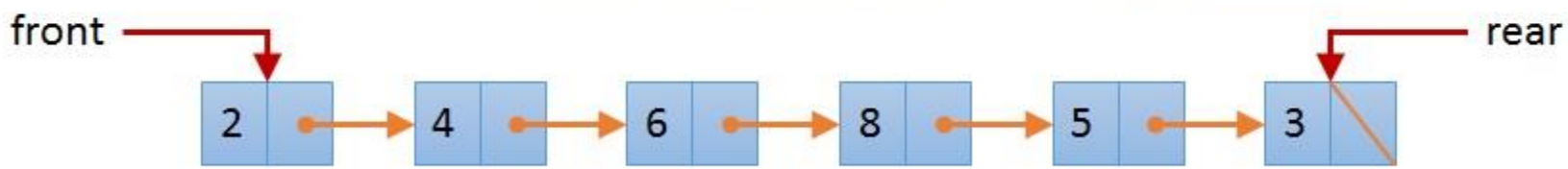
Tail

A Queue with its pointers



Front of Queue

Rear (end) of Queue



Front pointer

*Pointing to **first** element of Queue*

Rear pointer

*Pointing to **Last** element of Queue*

Representation of Queue

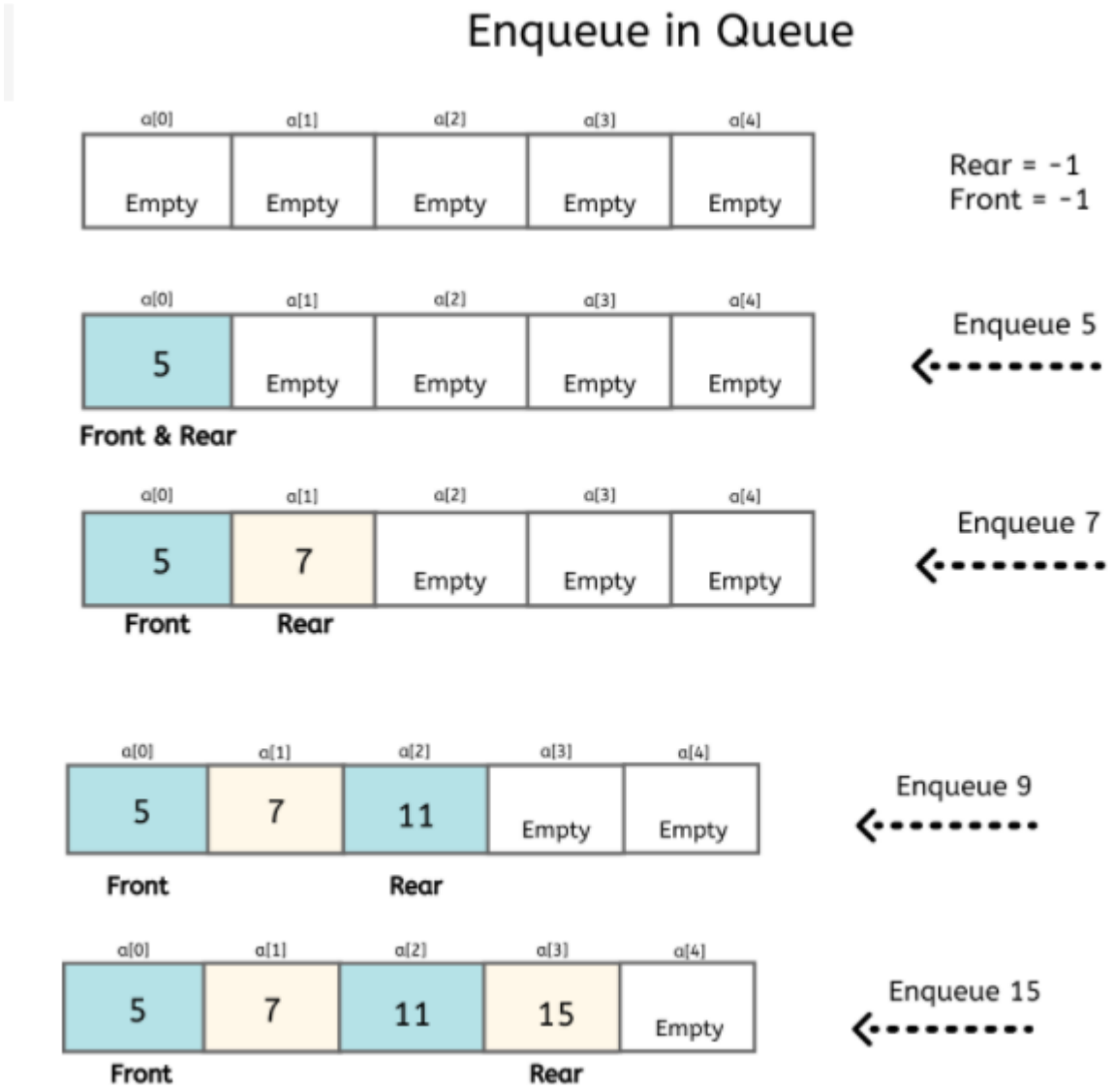
Queue as a data structure can be represented in two ways.

- Stack as an Array (Most popular)
- Stack as a struct (Popular)
- Stack as a Linked List.

1. Enqueue()

When we requirea to add an element to the Queue we perform Enqueue() operation.

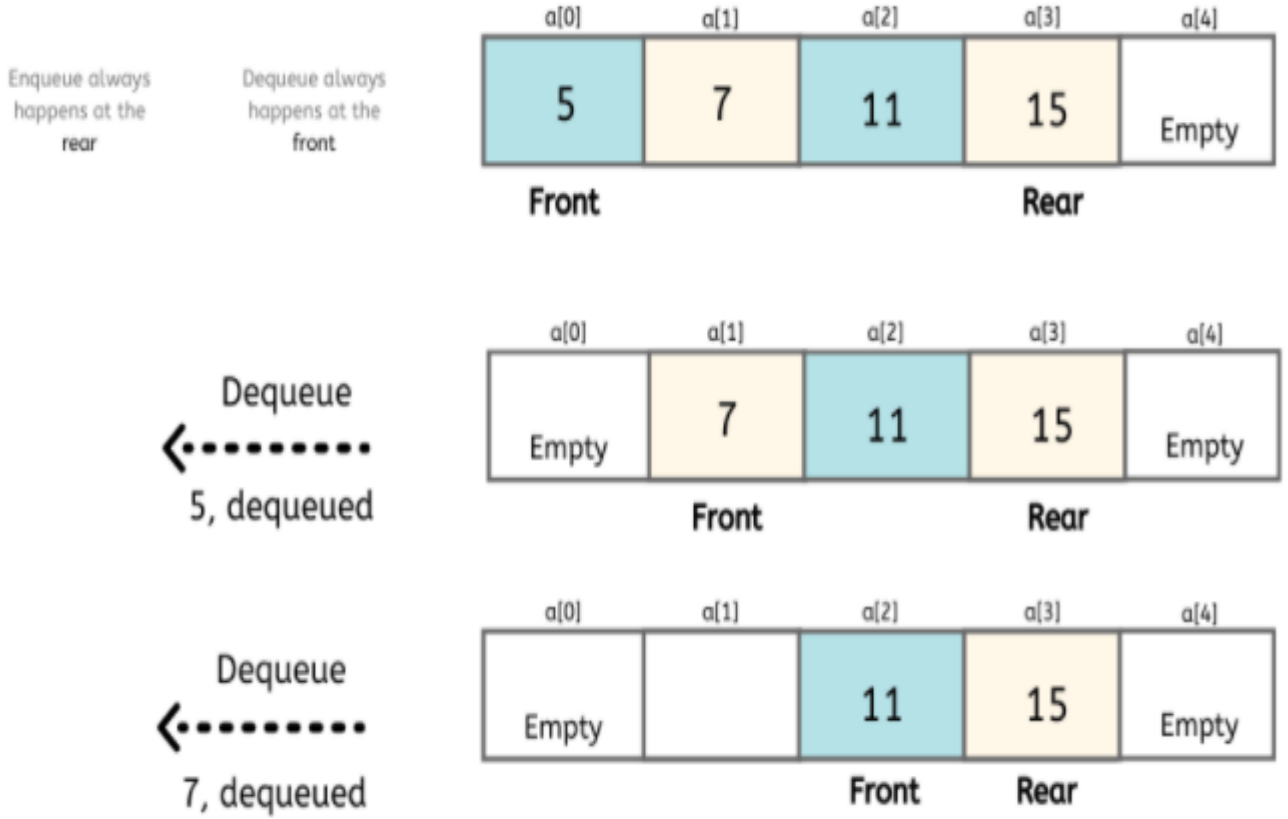
Push() operation is synonymous of insertion/addition in a data structure.



2. Dequeue()

When we require to delete/remove an element to the Queue we perform Dequeue() operation.

Dequeue() operation is synonymous of deletion/removal in a data structure.



Simple Queue

A simple queue are the general queue that we use on perform insertion and deletion on FIFO basis i.e. the new element is inserted at the rear of the queue and an element is deleted from the front of the queue.

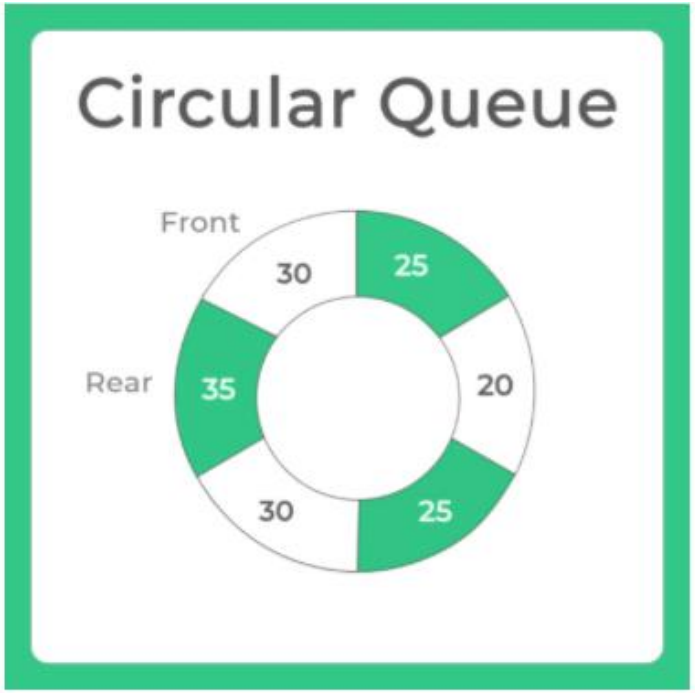
Applications of Simple Queue

The applications of simple queue are:-

- CPU scheduling
- Disk Scheduling
- Synchronization between two process.

Circular Queue

Circular queue is a type of queue in which all nodes are treated as circular such that the first node follows the last node. It is also called ring buffer. In this type of queue operations are performed on first in first out basis i.e the element that has inserted first will be one that will be deleted first.



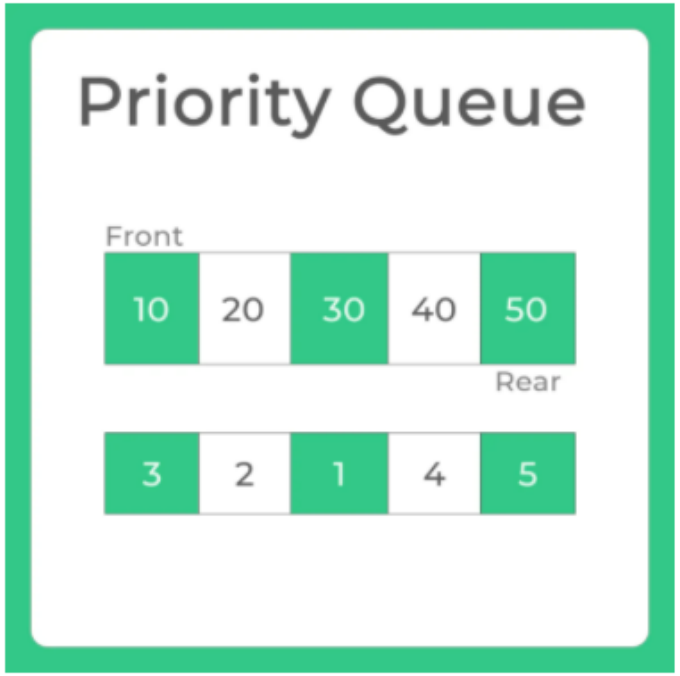
Applications of Circular Queue

The applications of circular queue are:-

- CPU scheduling
- Memory management
- Traffic Management

Priority Queue

Priority Queue is a special type of queue in which elements are treated according to their priority. Insertion of an element take place at the rear of the queue but the deletion or removal of an element take place according to the priority of the element. Element with the highest priority is removed first and element with the lowest priority is removed last.

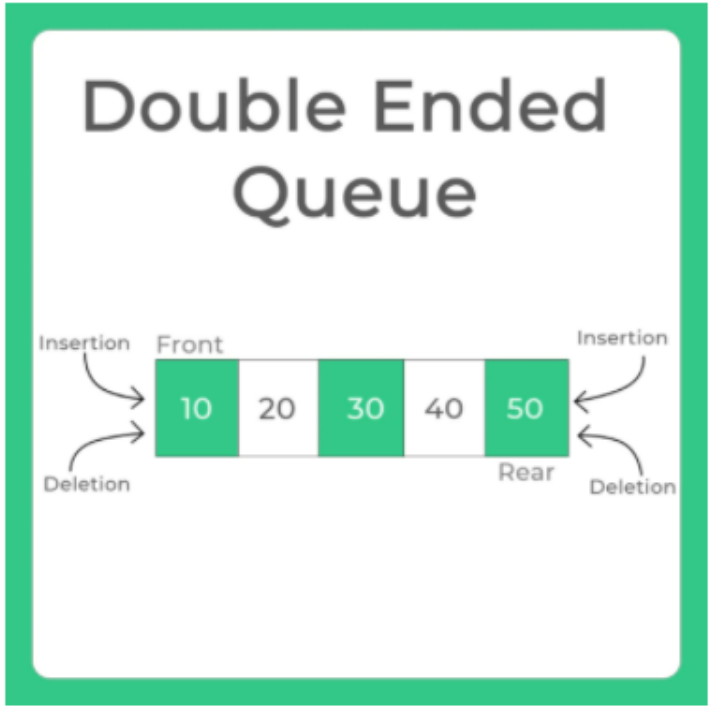


Applications of Priority Queue

The applications of priority queue are:-

- Dijkstra's shortest path algorithm
- Data compression in huffman codes.
- Load balancing and interrupt handling in operating system.
- Sorting heap.

Double Ended Queue



Double ended queue are also known as deque. In this type of queue insertion and deletion of an element can take place at both the ends. Further deque is divided into two types:-

- Input Restricted Deque :- In this, input is blocked at a single end but allows deletion at both the ends.
- Output Restricted Deque :- In this, output is blocked at a single end but allows insertion at both the ends.

Applications of Double Ended Queue

The applications of double ended queue are:-

- To execute undo and redo operation.
- For implementing stacks.
- Storing the history of web browsers.

Applications and uses for Queues

- Heavily used in almost all applications of the operating system, to schedule processes, moving them in or out of process scheduler.

- FCFS, SJF etc

- Asynchronously i.e. when data resource may be the same but not received at the same rate.

- Anything that has to do with process and schedule, in the system or code.

Thanks