DATA STRUCTURE 1







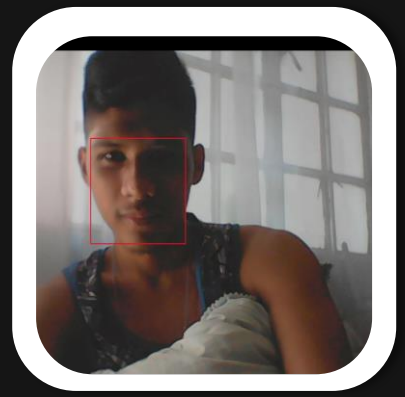


MEMBERS

Sollestre

Dalaya

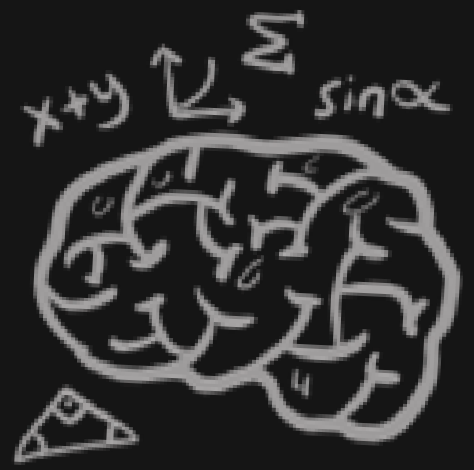
Turaray

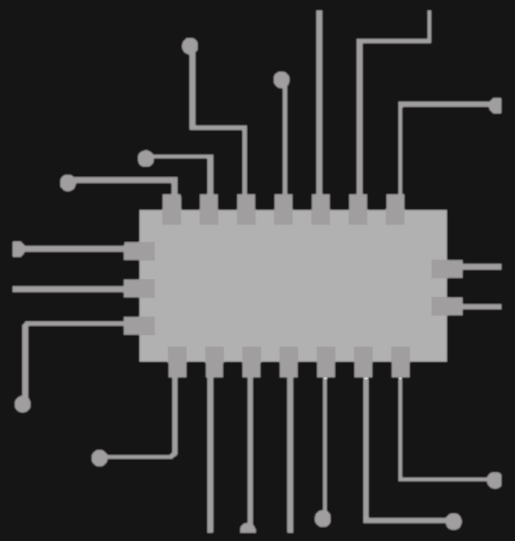


Esturas

Nabartey

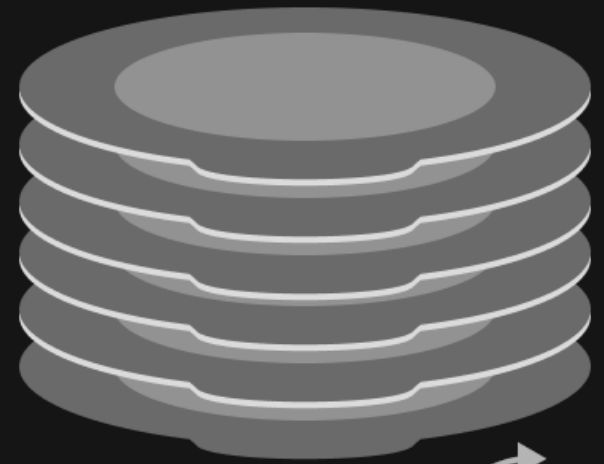
TOPICS





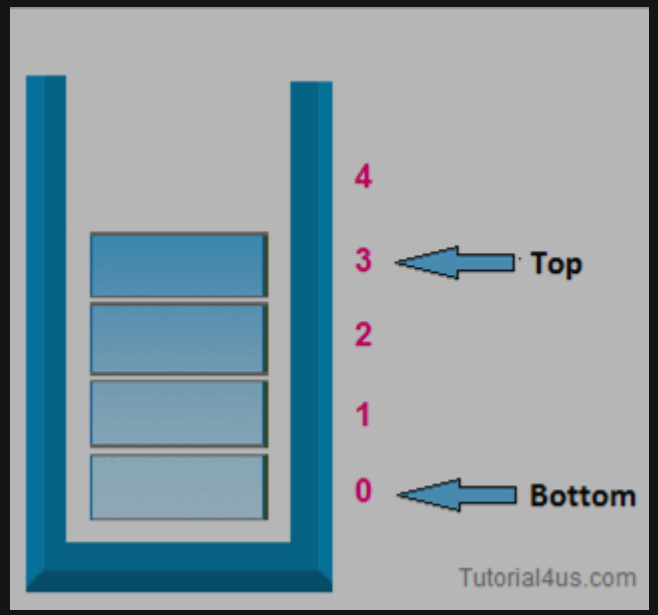
- STACK
- QUEUE
- TYPES OF QUEUE
- CIRCULAR QUEUE
- PRIORITY QUEUE
- DEQUE

STACK

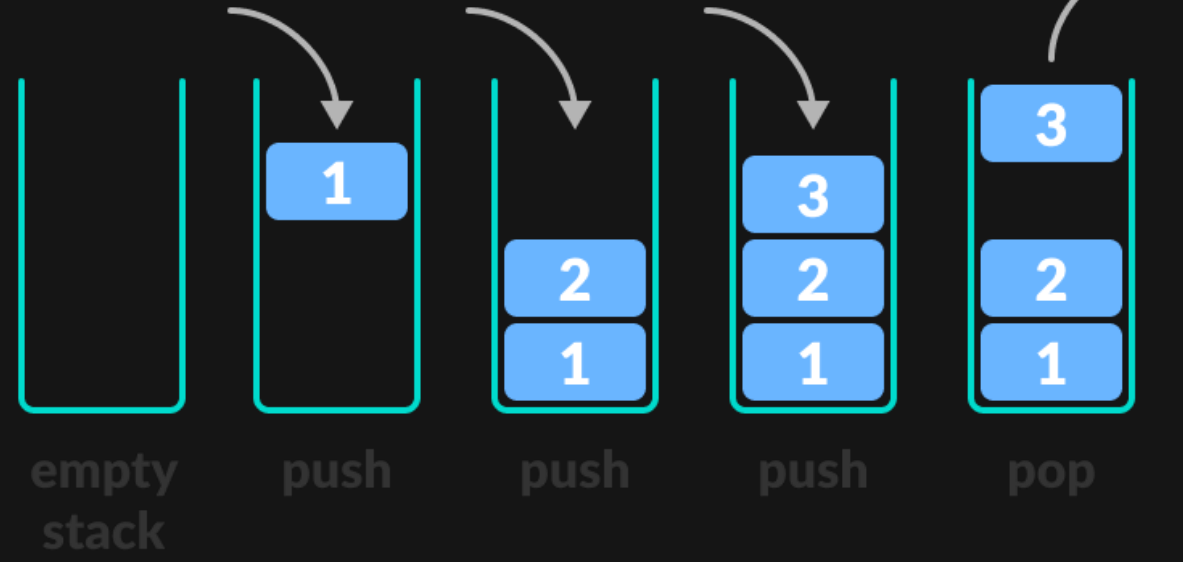


- It is a linear data structure that follows a particular order in which the operations are performed.

Last In First Out (LIFO)



PRINCIPLE OF STACK



BASIC OPERATIONS

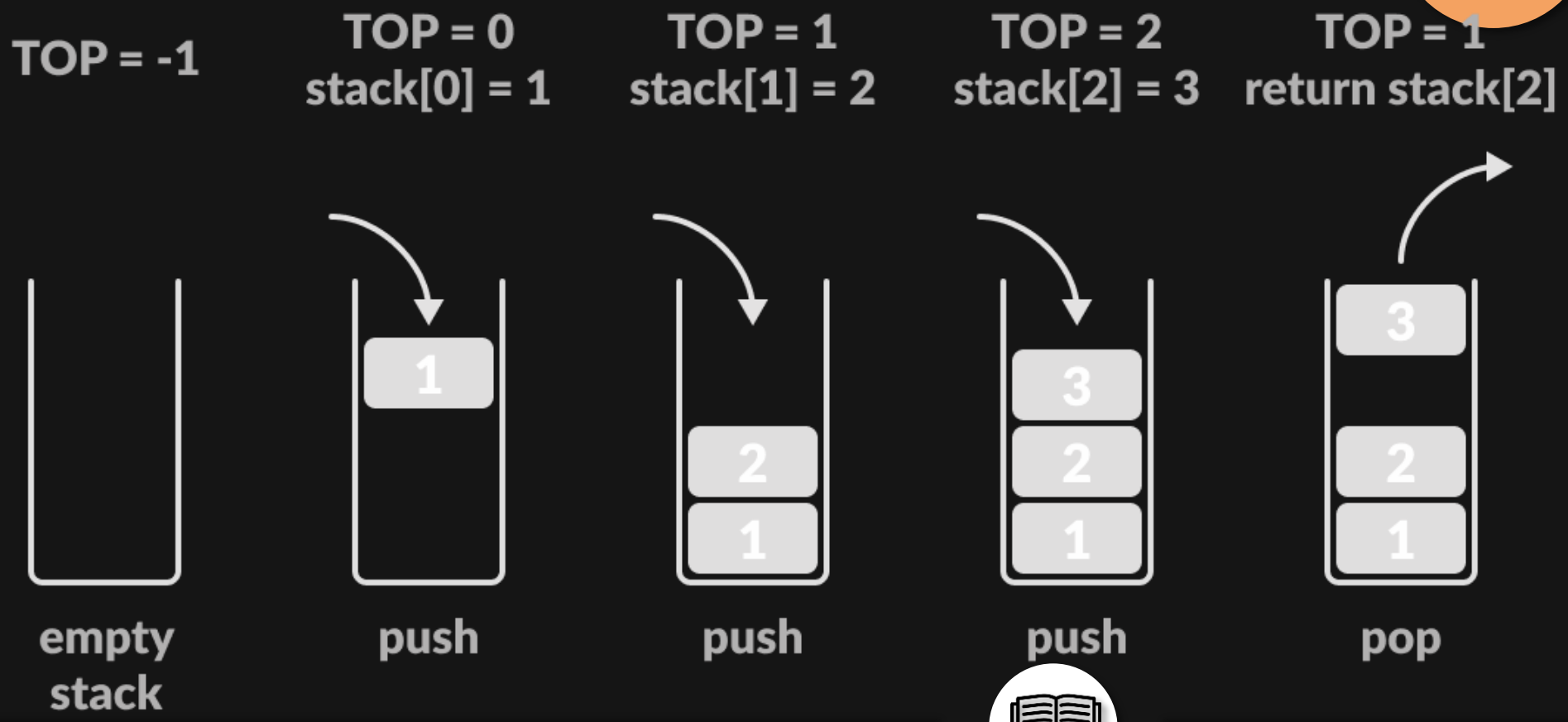
PUSH	POP	EMPTY	FULL	PEEK
to insert an element into the stack	to remove an element from the stack	Check if the stack is empty	Check if the stack is full	Get the value of the top element without removing it





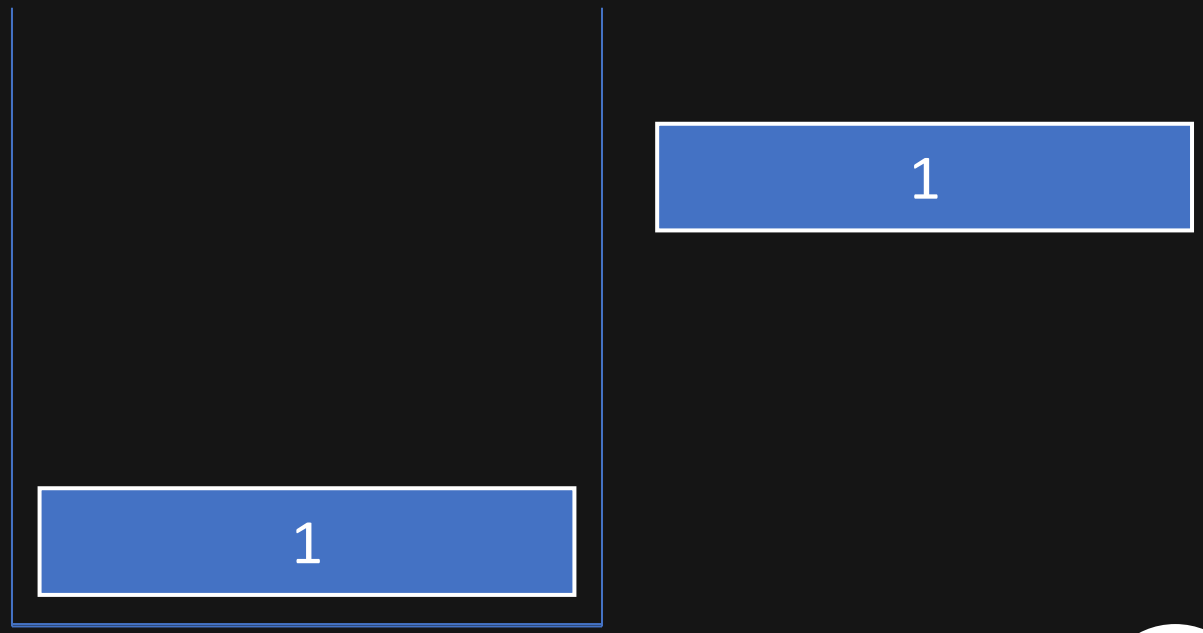



HOW STACK WORKS



HOW STACK WORKS

POP



HOW STACK WORKS

TYPES OF STACK

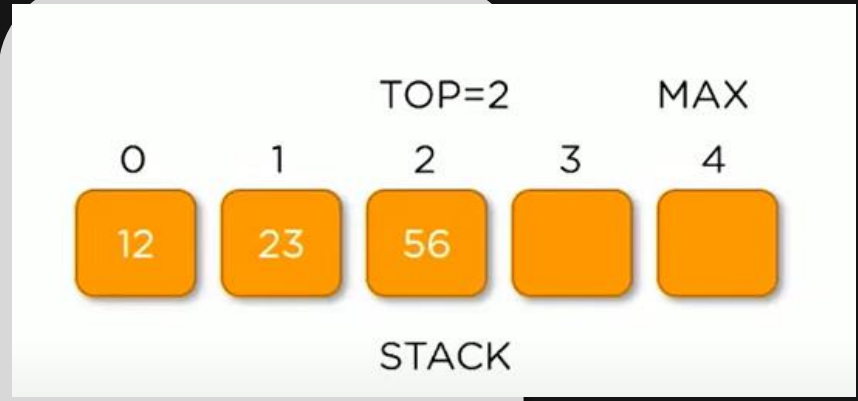
Register Stack: a memory element present in the memory unit and can handle a small amount of data only.

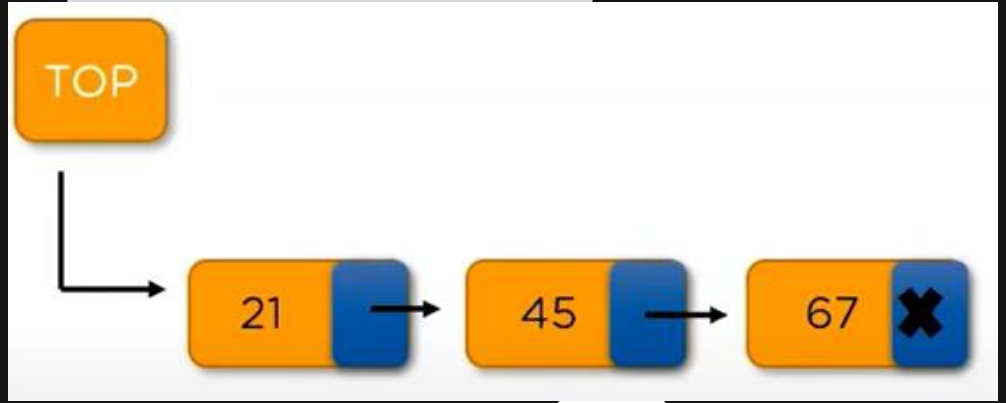
Memory Stack: This type of stack can handle a large amount of memory data.

IMPLEMENTATION OF STACK

There are two ways to implement a stack

- Using array
- Using linked list





QUEUE

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data, and the other is used to remove data Queue follows **First-In-First-Out methodology**

BASIC OPERATION WORKING OF QUEUE

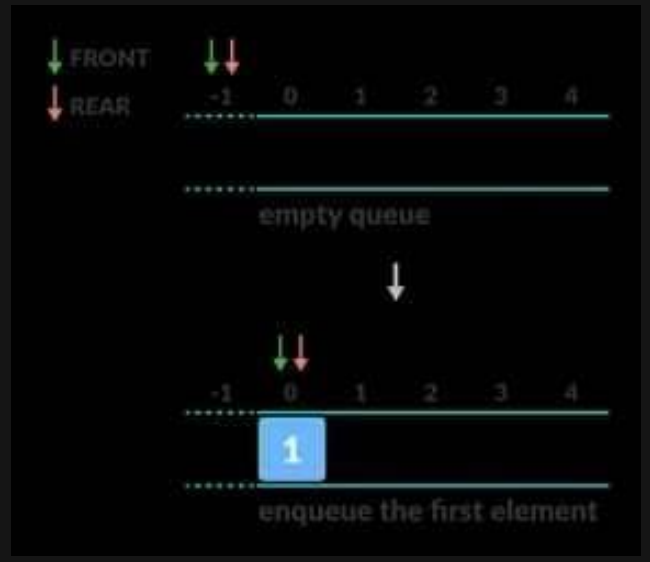
Enqueue: Add an element to the end of the queue
Dequeue: Remove an element from the front of the queue
Peek: Get the value of the front of the queue without removing it
IsEmpty: Check if the queue is empty

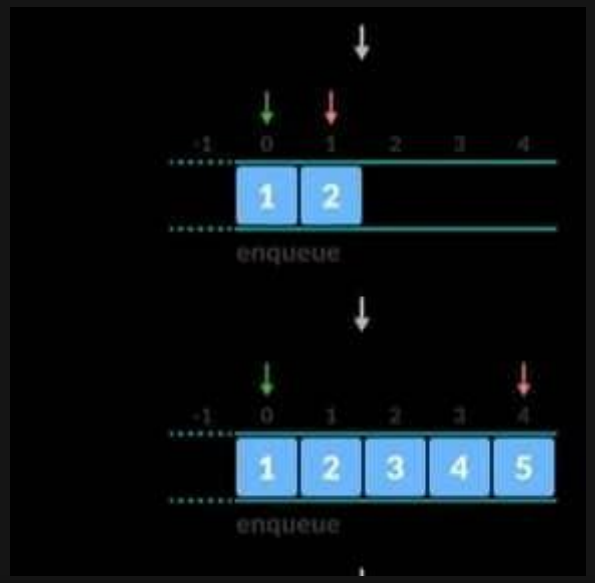
Queue operations work as follows:

- two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last element of the queue
- initially, set value of FRONT and REAR to -1

ENQUEUE

- check if the queue is full
- for the first element, set the value of FRONT to 0
- increase the REAR index by 1
- add the new element in the position pointed to by REAR



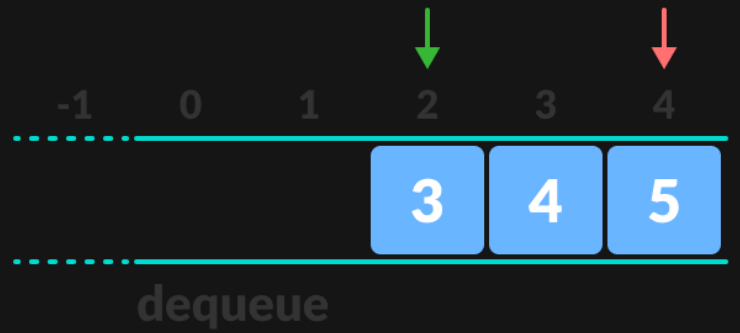


DEQUEUE

- check if the queue is empty
- return the value pointed by FRONT
- increase the FRONT index by 1
- for the last element, reset the values of FRONT and REAR to -1

LIMITATION

As you can see in the image below, after a bit of enqueuing and dequeuing, the size of the queue has been reduced.



And we can only add indexes 0 and 1 only when the queue is reset (when all the elements have been dequeued).

After REAR reaches the last index, if we can store extra elements in the empty spaces (0 and 1), we can make use of the empty spaces. This is implemented by a modified queue called the circular queue.

APPLICATION

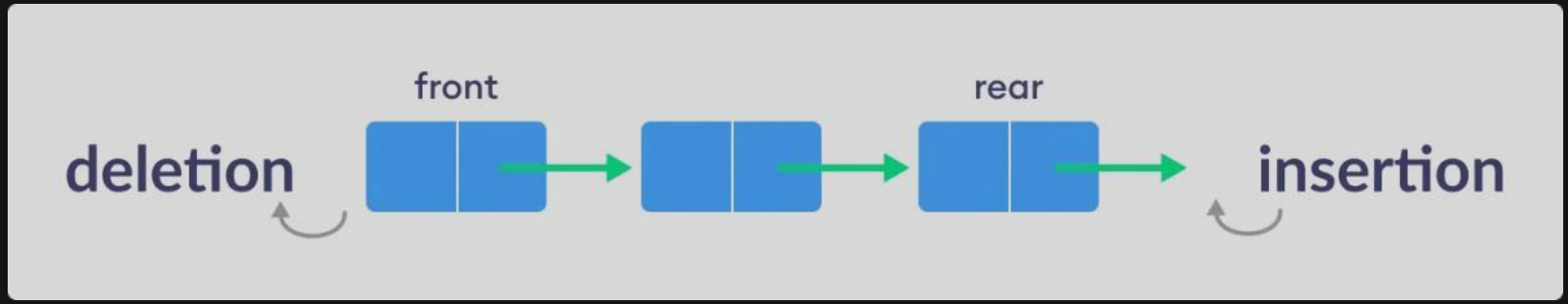
- **CPU scheduling, Disk Scheduling**
- **When data is transferred asynchronously between two processes. The queue is used for synchronization. For example: IO Buffers, pipes, file IO, etc**
- **Handling of interrupts in real-time systems.**
- **Call Center phone systems use Queues to hold people calling them in order.**

TYPES OF QUEUE

There are four different types of queues: SIMPLE , CIRCULAR AND DEQUE

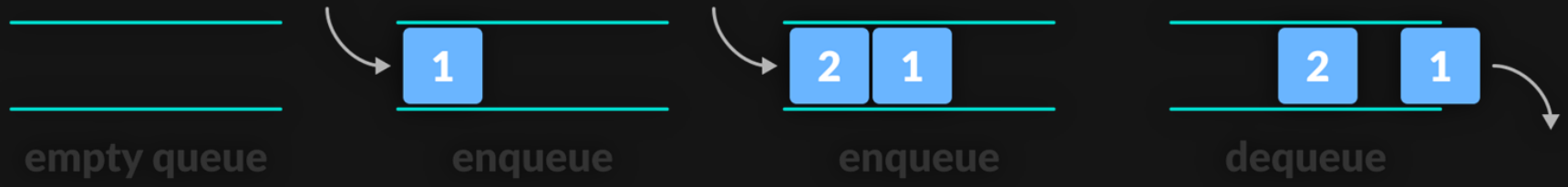
SIMPLE QUEUE

In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.



CIRCULAR

A circular queue is the extended version of a regular queue where the last element is connected to the first element. Thus, forming a circle-like structure.



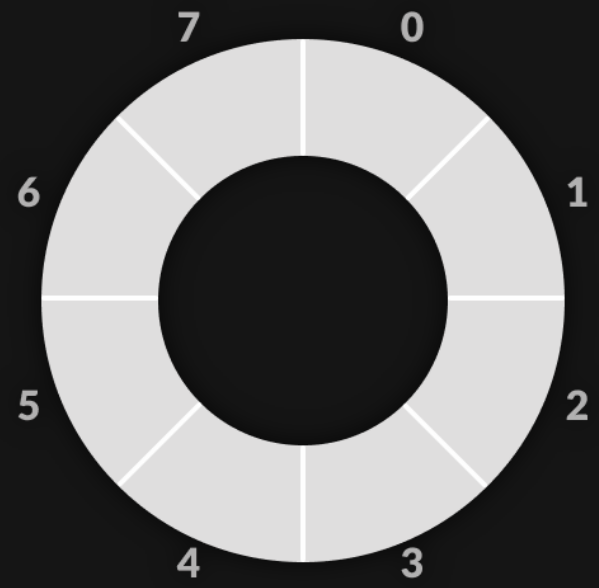


Figure 2. circular queue

The circular queue solves the major limitation of the normal queue. In a normal queue, after a bit of insertion and deletion, there will be non-usable empty space.

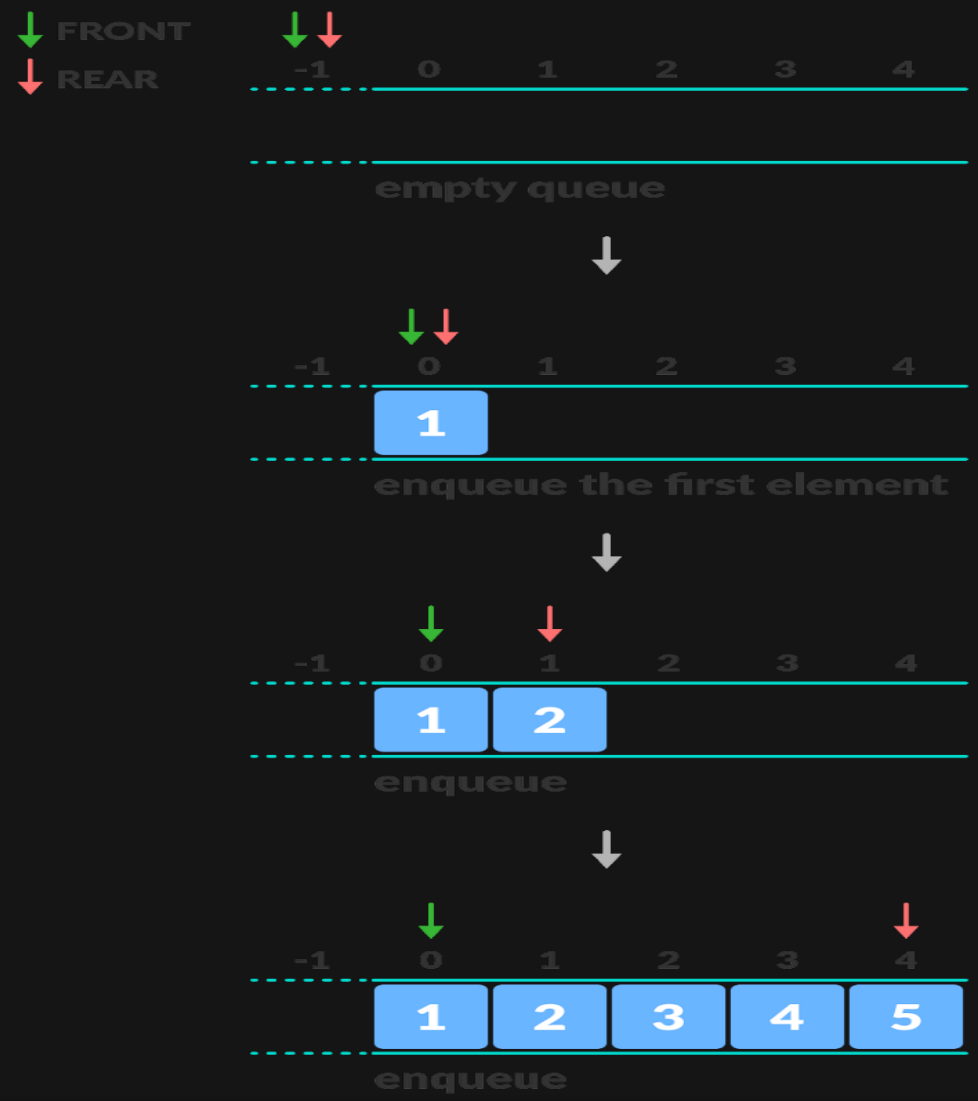


Figure 1.
regular queue

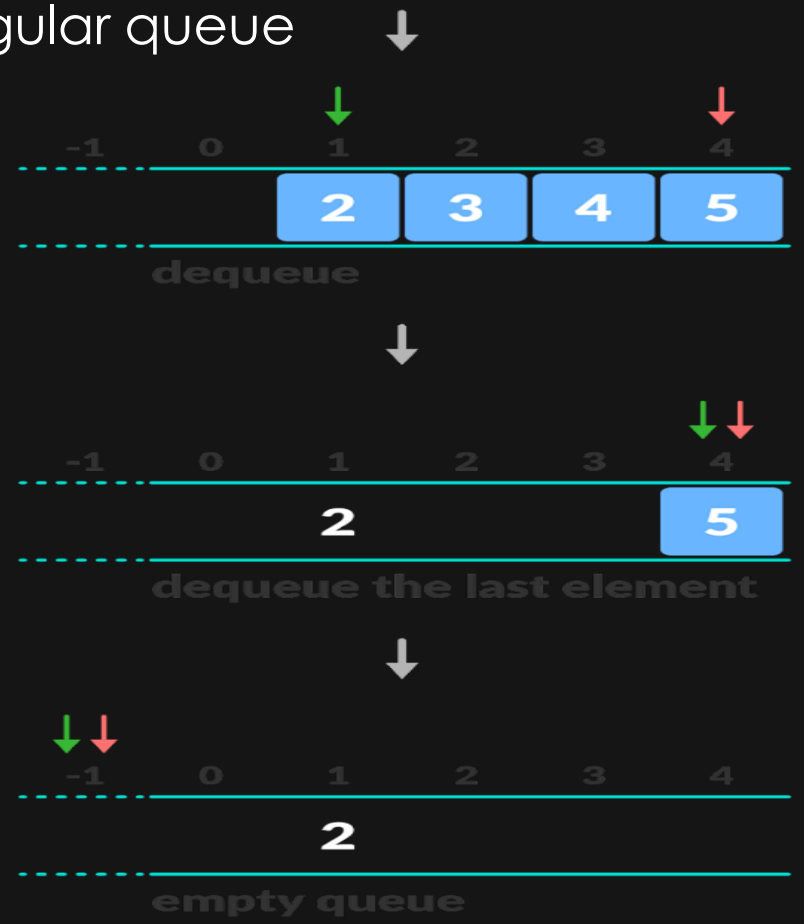
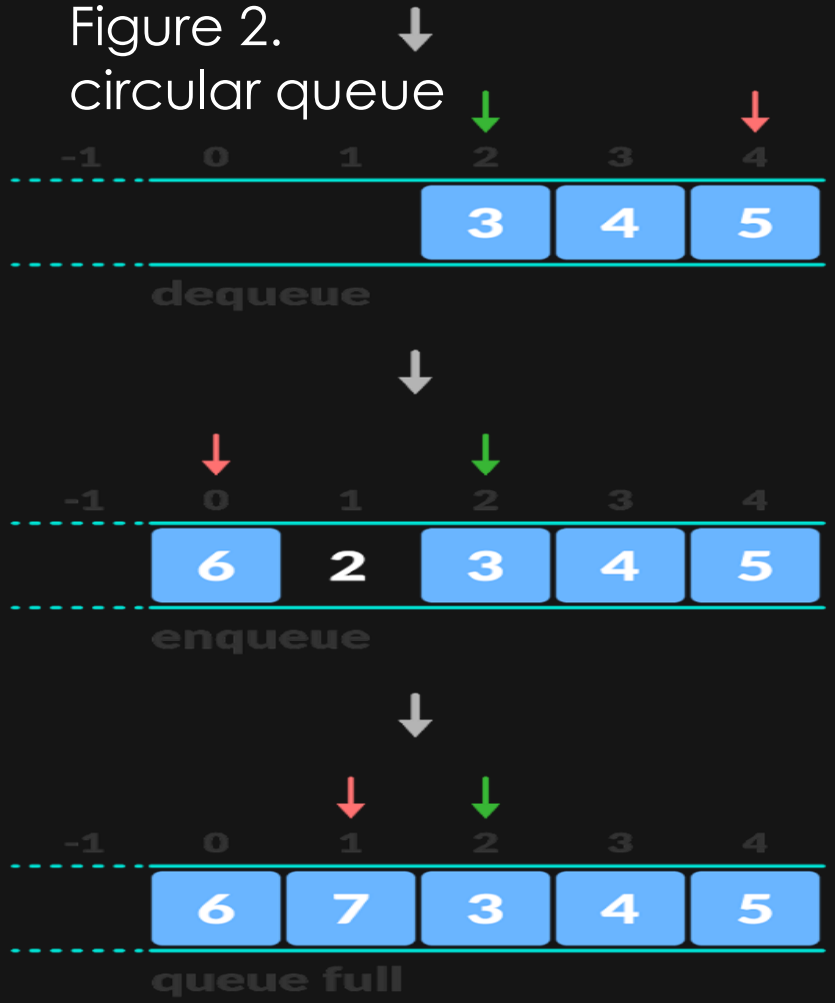


Figure 2.
circular queue



HOW CIRCULAR WORKS

A circular Queue works by the process of circular increment i.e., when we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue.

Here, the circular increment is performed by modulo division with the queue size. That is, if REAR + 1 == 5 (overflow!), REAR = (REAR + 1)%5 = 0 (start of queue)

CASE 1:

-1	0	1	2	3	4
REAR					

CASE 2:

-1	0	1	2	3	4
	REAR				

CASE 3:

-1	0	1	2	3	4
					REAR

OPERATION

The circular queue work as follows:
- two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last elements of the queue
- initially, set value of FRONT and REAR to -1

INT FRONT = -1

INT REAR = -1

TRACKS THE FIRST ELEMENT — INT FRONT = -1

TRACKS THE LAST T ELEMENT — INT REAR = -1

-1	0	1	2	3	4

IF THE QUEUE IS FULL, WE CAN'T ENQUEUE AN ELEMENT









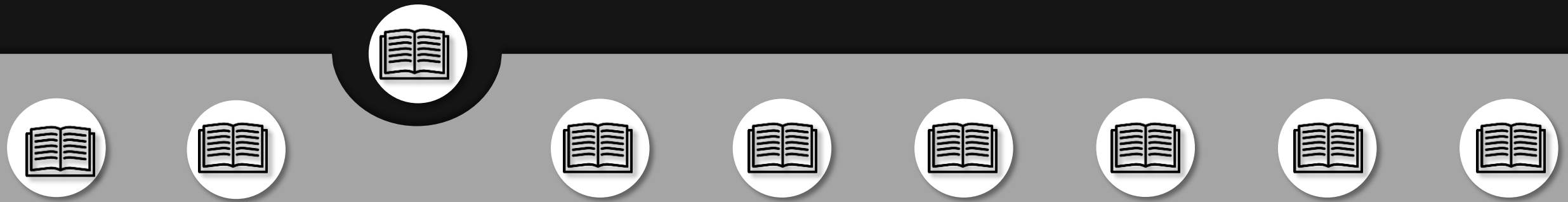
IF THE QUEUE IS EMPTY, WE CAN'T DEQUEUE AN ELEMENT

However, the check for full queue has a new additional case:

- Case 1: FRONT = 0 && REAR == SIZE - 1
- Case 2: FRONT = REAR + 1

The second case happens when REAR starts from 0 due to circular increment and when its value is just 1 less than FRONT, the queue is full.

0	1	2	3	4



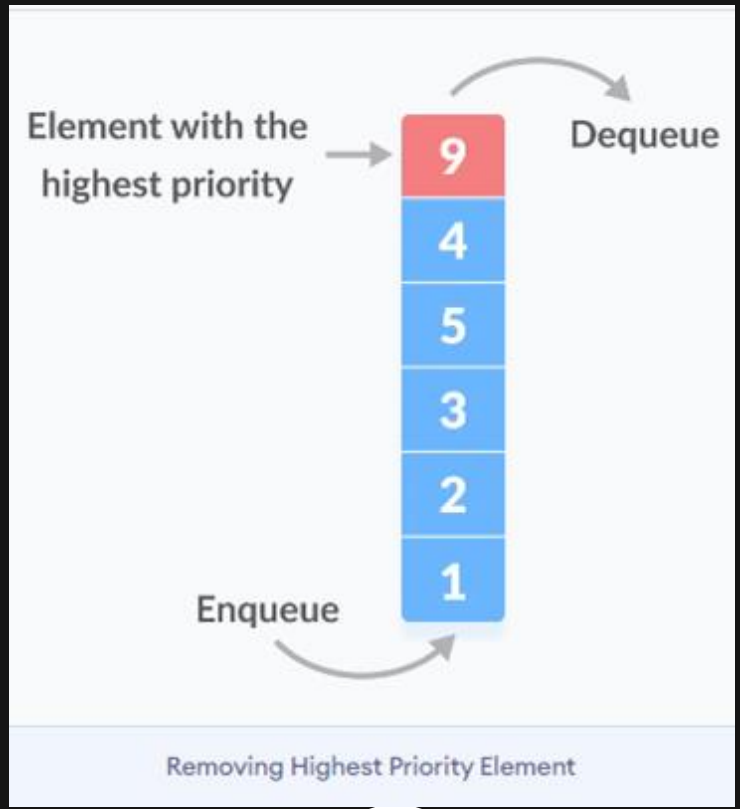
PRIORITY QUEUE

A priority queue is a special type of queue in which each element is associated with a priority value. elements are served on the basis of their priority. The higher priority elements are served first.

Hospital Emergency Queue

PRIORITY QUEUE

priority queue in programming also works in same approach. However, if elements with the same priority occur, they are served according to their order in the queue.

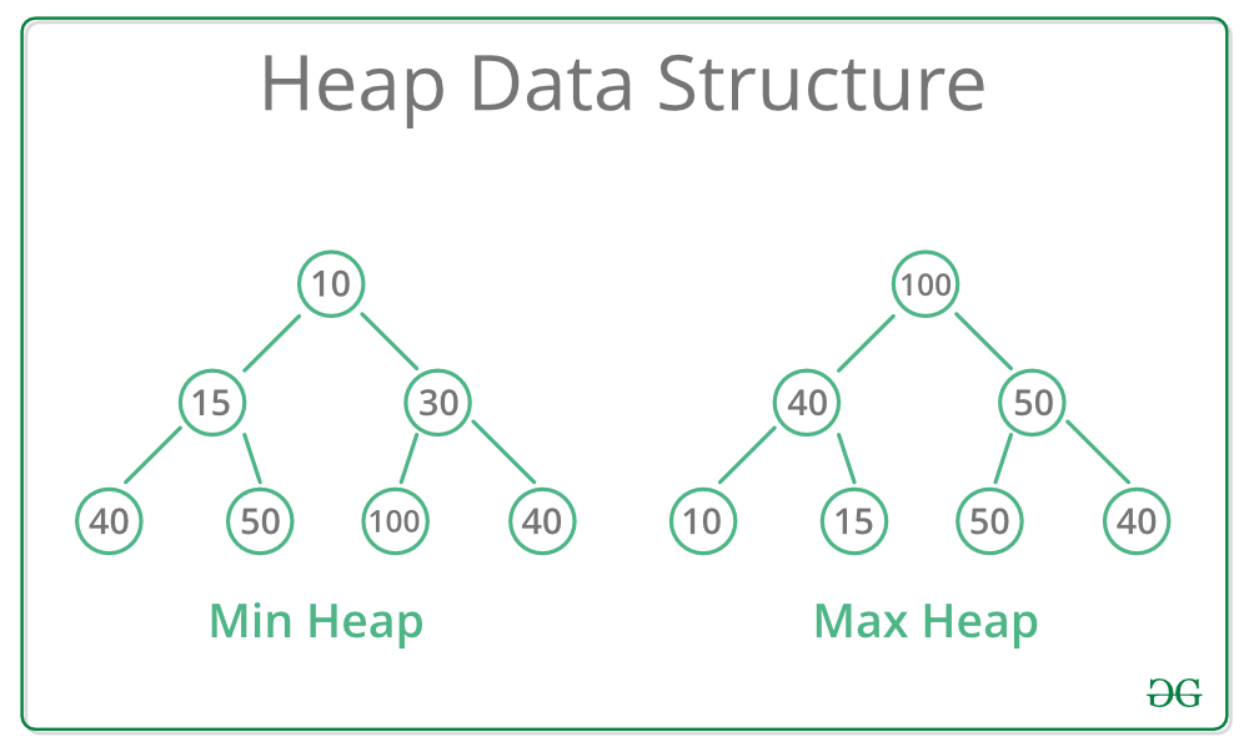


The element with the highest value is considered the highest priority element. However, in other cases, we can assume the element with the lowest value as the highest priority element.

We can also set priorities according to our needs.

IMPLEMENTATION

Priority queue can be implemented using an array, a linked list, a heap data structure, or a binary search tree. Among these data structures, heap data structure provides an efficient implementation of priority queues.



A Heap is a special Tree-based data structure in which the tree is a complete binary tree.

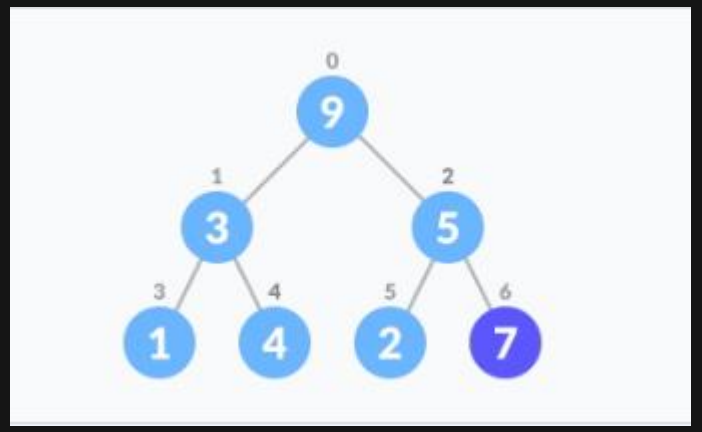
OPERATIONS

Basic operations of a priority queue are inserting, removing, and peeking elements.

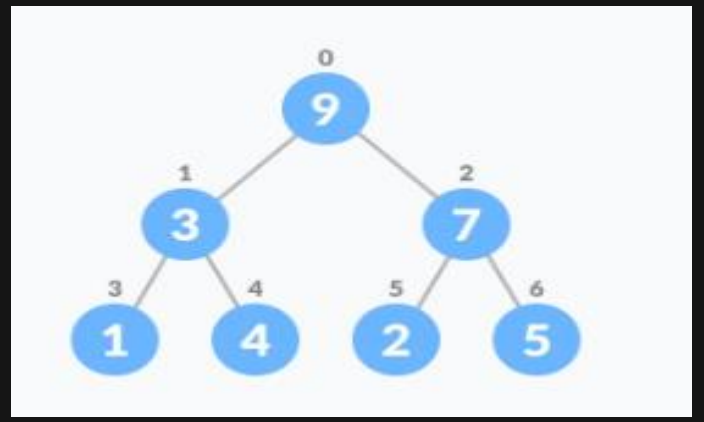
1. Inserting an Element into the Priority Queue

Inserting an element into a priority queue (max-heap) is done by the following steps.

Insert the new element at the end of the tree



Heapify the tree.

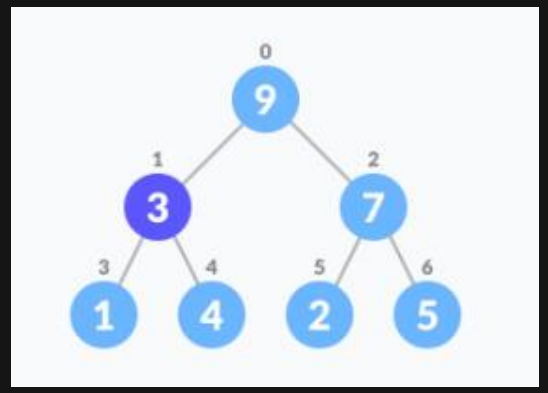


OPERATIONS

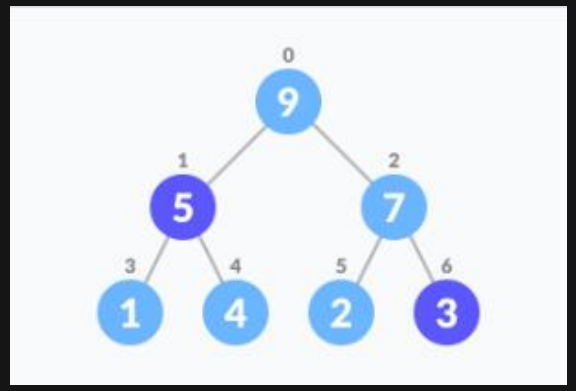
2. Deleting an Element from the Priority Queue

Deleting an element from a priority queue (max-heap) is done as follows:

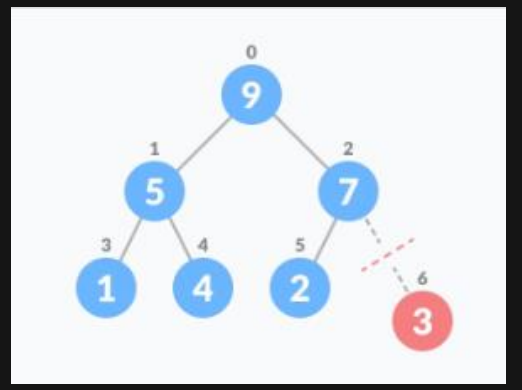
Select the element to be deleted.



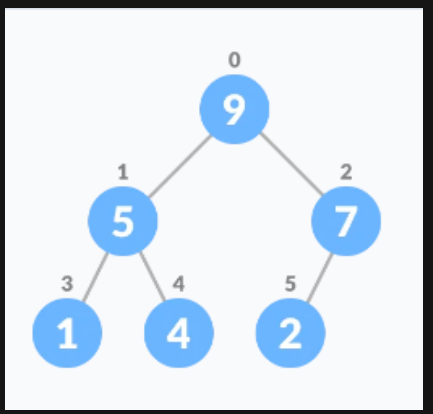
Swap it with the last element.



Remove the last element



Heapify the tree



OPERATIONS

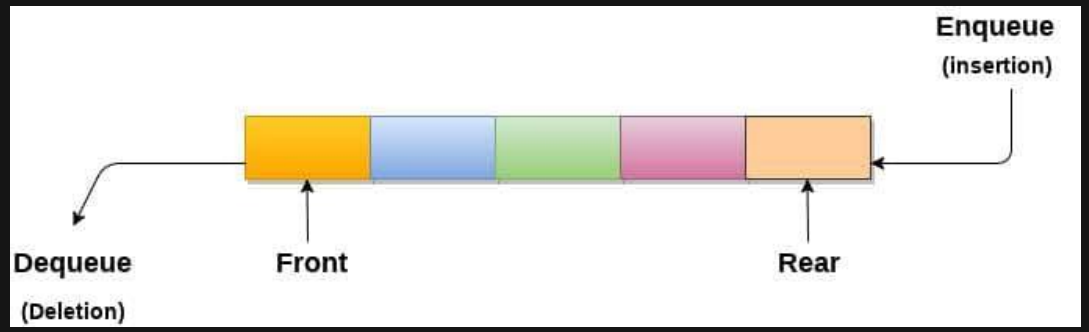
3. Peeking from the Priority Queue (Find max/min)

Peek operation returns the maximum element from Max Heap or minimum element from Min Heap without deleting the node.

For both Max heap and Min Heap

DEQUE

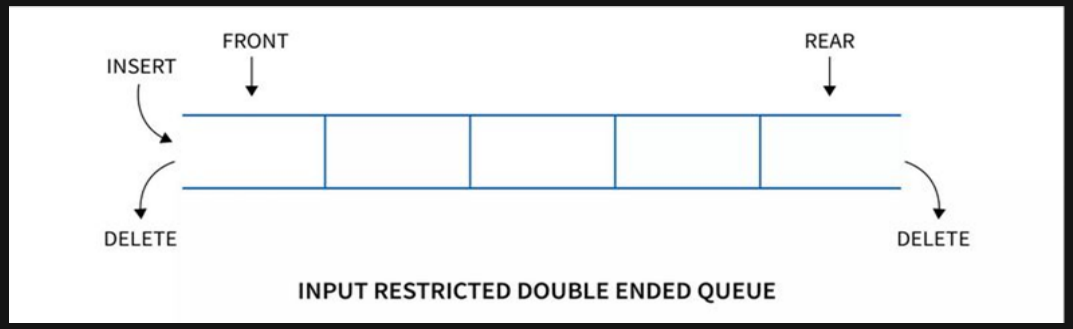
- Deque or Double Ended Queue is a type of queue in which insertion and removal of elements can either be performed from the front or the rear. Thus, it does not follow FIFO rule (First In First Out).



TYPES OF DEQUE

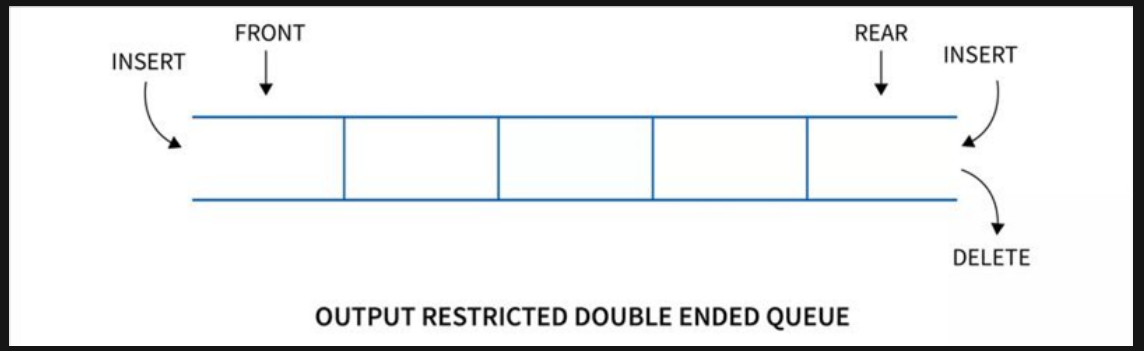
Input Restricted Deque

In an input restricted queue, the data can be inserted from only one end, while it can be deleted from both the ends.



Output Restricted Deque

In an Output restricted queue, the data can be deleted from only one end, while it can be inserted from both the ends.



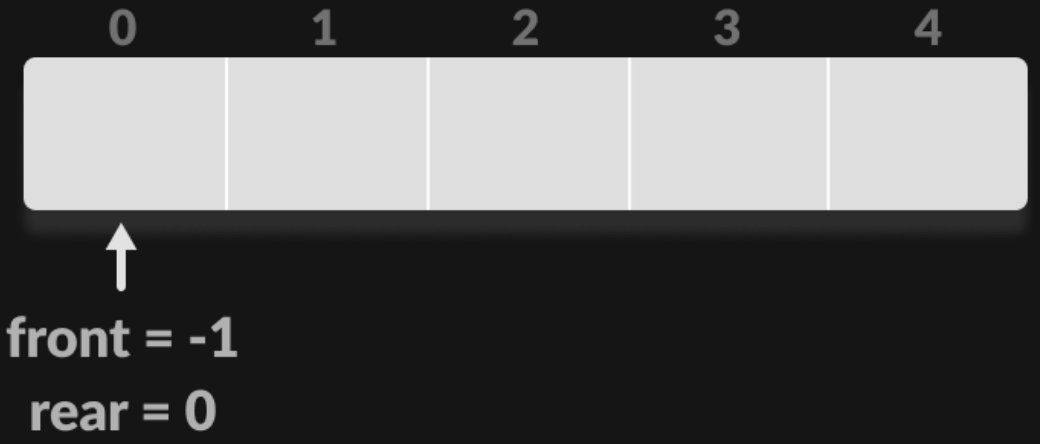
OPERATIONS

Below is the circular array implementation of deque. In a circular array, if the array is full, we start from the beginning.
But in a linear array implementation, if the array is full, no more elements can be inserted. In each of the operations below, if the array is full, "overflow message" is thrown.

Before performing the following operations, these steps are followed.

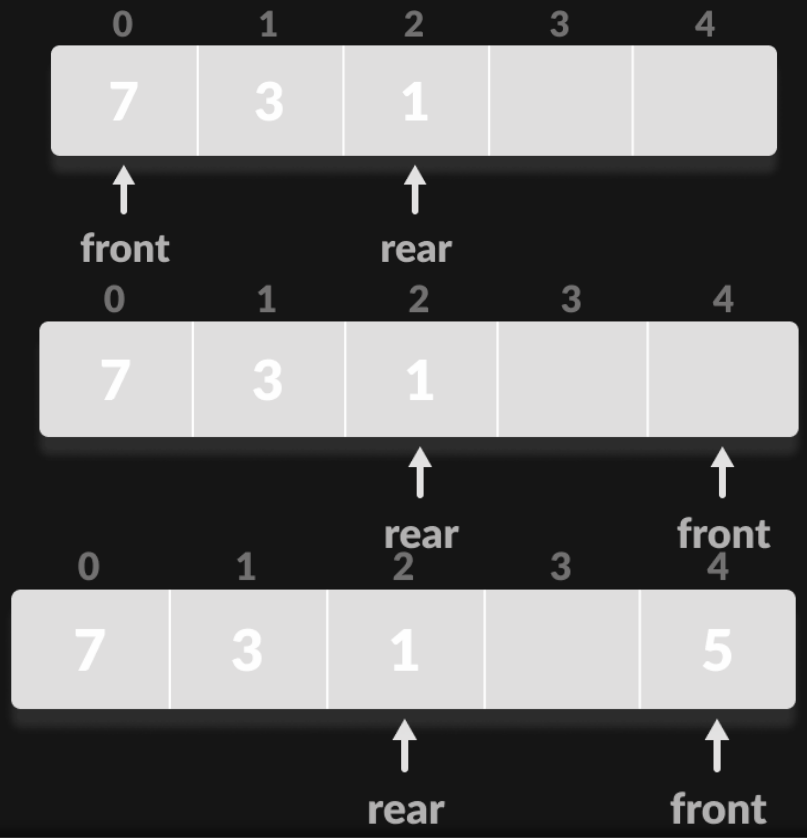
1.Take an array (deque) of size n.

2.Set two pointers at the first position and set front = -1 and rear = 0.



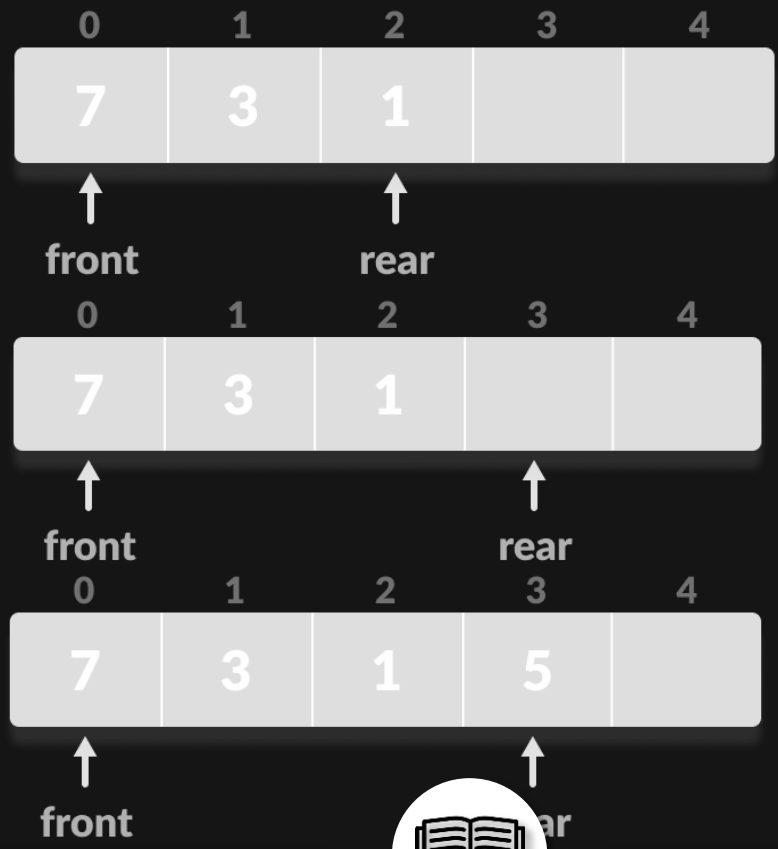
1. Insert at the Front

- This operation adds an element at the front.
- Check the position of front.



2. Insert at the Rear

- This operation adds an element to the rear.
- Check if the array is full.

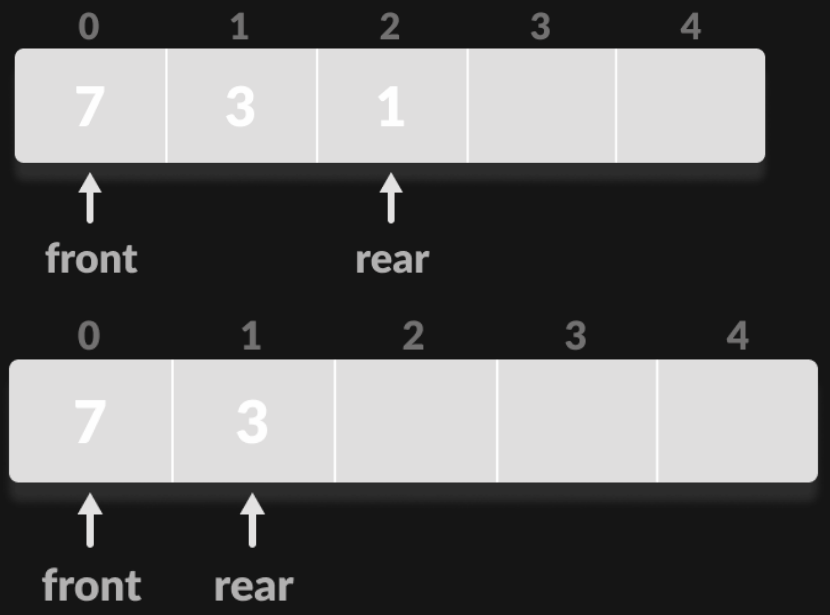


3. Delete from the Front

- The operation deletes an element from the front.
- Check if the deque is empty.

4. Delete the Rear

- This operation adds an element at the front.
- Check the position of front.



5. Check Empty

This operation checks if the deque is empty. If front = -1, the deque is empty.

6. Check Full

This operation checks if the deque is full. If front = 0 and rear = n - 1 OR front = rear + 1, the deque is full.

TIME COMPLEXITY

The time complexity of all the above operations is constant i.e. O(1).

Applications of Deque Data Structure

1. In undo operations on software.
2. To store history in browsers.
3. For implementing both stacks and queues.

THANK YOU!

RESOURCES:

https://www.programiz.com/dsa/stack

https://www.geeksforgeeks.org/stack-data-structure-introduction-program/

https://www.programiz.com/dsa/queue

https://www.programiz.com/dsa/types-of-queue

https://www.programiz.com/dsa/circular-queue

https://www.programiz.com/dsa/priority-queue

https://www.programiz.com/dsa/deque

https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm

Bollestre

Dalaya

Duraray

Esturas

Nabartey