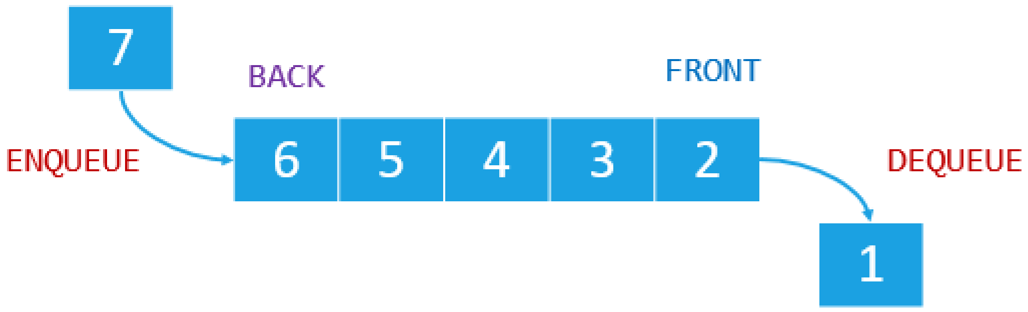
Queues

Created	@August 16, 2022 10:06 PM
Class	
Type	
Materials	
Reviewed	□

- Queues are linear data structures which support FIFO operations i.e. First In First Out
- So the element which is added first in the queue is removed first also.
- Queues will give you a sense of real life queues, for example to buy a ticket for metro we have to stand in a queue, and the person who came first queue gets the ticket first.

Applications

- Queues are one of the most important data structures in computer science.
- Queues are used for message queues.
- Queues are used for many OS algorithms
- Media playlist kind of implementations



In queues, insertion operation is called as `Enqueue` and deletion operation is called as `Dequeue`

So in a queue, we have to enqueue from the Back and dequeue from the front.

The last element of the queue is called Back and the first element is called as front

Types of Queues

- Simple Queue → This acts as a normal queue in which element gets added at the last and gets removed from front. Element which is added first is removed first.
- Deque → In this queue, we can add from back and front both and remove from back and front both.
- Circular Queue → In a circular queue, the next element of back is the fornt.
- Priority Queue → In a priority queue, elements are not arranged on the basis of when they are added. Like if the element get's added first then it might not be removed first. In a priority queue, every element has a custom priority and the highest priority element is removed first.

How to implement Queues ?

- Queues Can Be Implemented using Linked Lists
- Queues Can be implemented using arrays
- Queues can be implemented using stacks

Queue using LL

To implement queues using LL, we can use addAtTail and removeAtHead. Head of the LL will act as the front and tail will act as back.

```
class Node {
  constructor(d) {
    this.data = d; // data parameter represents the actual data stored in node
    this.next = null; // this will be a ref to the next node connected to the curr node
  }
}
```

```
class LinkedList {
  // singly
  constructor() {
    // when we initialise a new linked list head will be empty
    this.head = null;
        this.tail = null;
  }

  addAtHead(data) {
        /**
         * Time: O(1)
         * Space: O(1)
         */
    let newNode = new Node(data); // created a new node
        if(this.head == null) {
            this.tail = newNode;
        }
    newNode.next = this.head; // set the next of new node to head
    this.head = newNode; // update the head to the new node
  }

  removeAtHead() {
        /**
         * Time: O(1)
         * Space: O(1)
         */
    if(this.head == null) return;
    let temp = this.head.next; // stored access to new head
    this.head.next = null; // de linked the old head
    this.head = temp; // updated the head
        if(this.head == null) this.tail = null;
  }

  addAtTail(data) {
        /**
         * Time: O(1)
         * Space: O(1)
         */
    if(this.head == null) { // if ll is empty, addattail is equal to addathead
      this.addAtHead(data);
      return;
    }
    let newNode = new Node(data);
    this.tail.next = newNode;
        this.tail = newNode;
  }

  getHead() {
    if(this.head == null) return undefined;
    return this.head.data;
  }

}
```

```
class Queue {
  constructor() {
    this.ll = new LinkedList();
  }
  enqueue(x) {
    this.ll.addAtTail(x);
  }

  dequeue() {
    this.ll.removeAtHead();
  }

  getFront() {
    return this.ll.getHead();
  }
}

let qu = new Queue();
qu.enqueue(10);
qu.enqueue(20);
qu.enqueue(30);
qu.enqueue(40);
qu.dequeue();
qu.dequeue();
qu.dequeue();
qu.enqueue(50);
qu.enqueue(60);

console.log(qu.getFront())
```

Problem:

Write a function to reverse a queue.

50, 40, 30, 20, 10→ qu

St →