**Explanation of the lab problem with examples**

**Problem 1 Explanation**

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

Implement the MyStack class:

void push(int x) Pushes element x to the top of the stack.

int pop() Removes the element on the top of the stack and returns it.

int top() Returns the element on the top of the stack.

boolean empty() Returns true if the stack is empty, false otherwise.

Example 1:

Operations: empty(),push(3), push(5), push(6),push(8),top(),pop()

Sequence:

Push 1: Stack becomes [3]

Push 2: Stack becomes [3, 5]

Push 3: Stack becomes [3,5,6]

Push 4: Stack becomes [3,5,6,8]

Expected Output: [true,8,8,6]

Example 2:

Operations: push(9), pop(), pop(), empty()

Push 3: Stack becomes [3,5,6,9]

Pop: Removes 9, stack becomes [3,5,6]

Pop: Removes 6, stack become [3,5]

Empty: returns false

Expected Output: [9, 6, false]

Example 3:

Operations: push(7), pop(), push(9), top(), empty()

Sequence:

Push 7: Stack becomes [3,5,7]

Pop: Removes 7, stack becomes [3,5]

Push 9: Stack becomes [3,5, 9]

Top: Returns 9 which is the top of the stack

Empty: Returns false (stack has elements)

Expected Output: [7,9,false]

**Problem 1 Possible Solutions**

In order to simulate the behavior of a stack (LIFO - Last In, First Out), I am using two queues (FIFO - First In, First Out). A stack allows the following operations:

Push (x): Push function adds an element x to the top of the stack.

Pop: Pop function removes and returns the top element of the stack.

Top: Top function returns the top element of the stack without removing it.

Empty: Empty function returns true if the stack is empty, and false otherwise.

|  |  |
| --- | --- |
| Operation | Time Complexity |
| push(x) | O(1) |
| pop() | O(n) |
| top() | O(n) |
| empty() | O(1) |

**Problem 2 Explanation**

This problem is an iterative algorithm to reverse a singly linked list. The function takes the head of a singly linked list as input and reverses the list in-place, returning the new head of the reversed list

Example 1

Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Example 2

Input: head = [1,2]

Output: [2,1]

Example 3

Input: head = []

Output: []

**Problem 2 Possible Solution**

We can use Iterative Approach, stack-based solution, two-pointer swapping solution. As for my solution, I used Iterative Approach solution.

|  |  |
| --- | --- |
| Operation | Time Complexity |
| reverseList () | O(n) |

**Problem 3 Explanation:** Given two strings, s and t, and it is required to determine if t is an anagram of s. An anagram is formed by rearranging the letters of one string to create another, where both strings have the same letters and frequency of characters.

Example 1:

Input: s = "anagram", t = "nagaram"

Output: true

Example 2:

Input: s = "dig", t = "dog"

Output: false

Example 3 :

Input: s = "listen", t = "silent"

Output: true

**Possible Solutions:** The below is the possible solution to solve this problem

If the length of both strings is not equal to, returns false; Then sorts both strings and compares them. If both sorted strings are identical, then t is an anagram of s.

|  |  |
| --- | --- |
| Operation | Time Complexity |
| isAnagram() | O(n log n) |