Name: Veeransh Shah

Reg No.: 221070063

One Stack using Two Queues

**Aim:**

To create A stack using 2 queues.

**Theory:**

Here Queues follow ‘FIFO’ which means First in First Out, i.e. the first element Enqueued into the Queue is the first element to get Dequeued from the Queue, when given the command. Whereas Stack follows ‘LIFO’ which means Last In First Out, i.e. the first element to get pushed into the stack is the last element to get popped from it.

We can use 2 Queues following ‘FIFO’ to make one Stack following ‘LIFO’.

Push: To insert an element into the stack

Pop: To remove an element from the stack

Enqueue: To insert an element into the Queue

Dequeue: To remove an element from the Queue

**Algorithm:**

1. Start

1. Create a class named Stack.

1. Initialize the number of elements in a queue and values of rear & front.

1. Giving conditions to the code, if the stack is full (isfull()) or if the stack is empty(isempty()). These conditions are given with the help of rear and front.

1. Create functions named push & pop using front and rear pointers.
2. Give the command to push the element into the queue.
3. If queue is full print “Stack overflow”.

1. Give the command to pop the element from the queue.

1. If queue is empty print “Stack Underflow”

1. Now, in the main function define 2 queues, q1 and q2.

1. Enqueue elements into the q1 as many as you want till the Queue is full.

1. Now dequeue all the elements from the q1 till rear = 0 and enqueue them into q2.

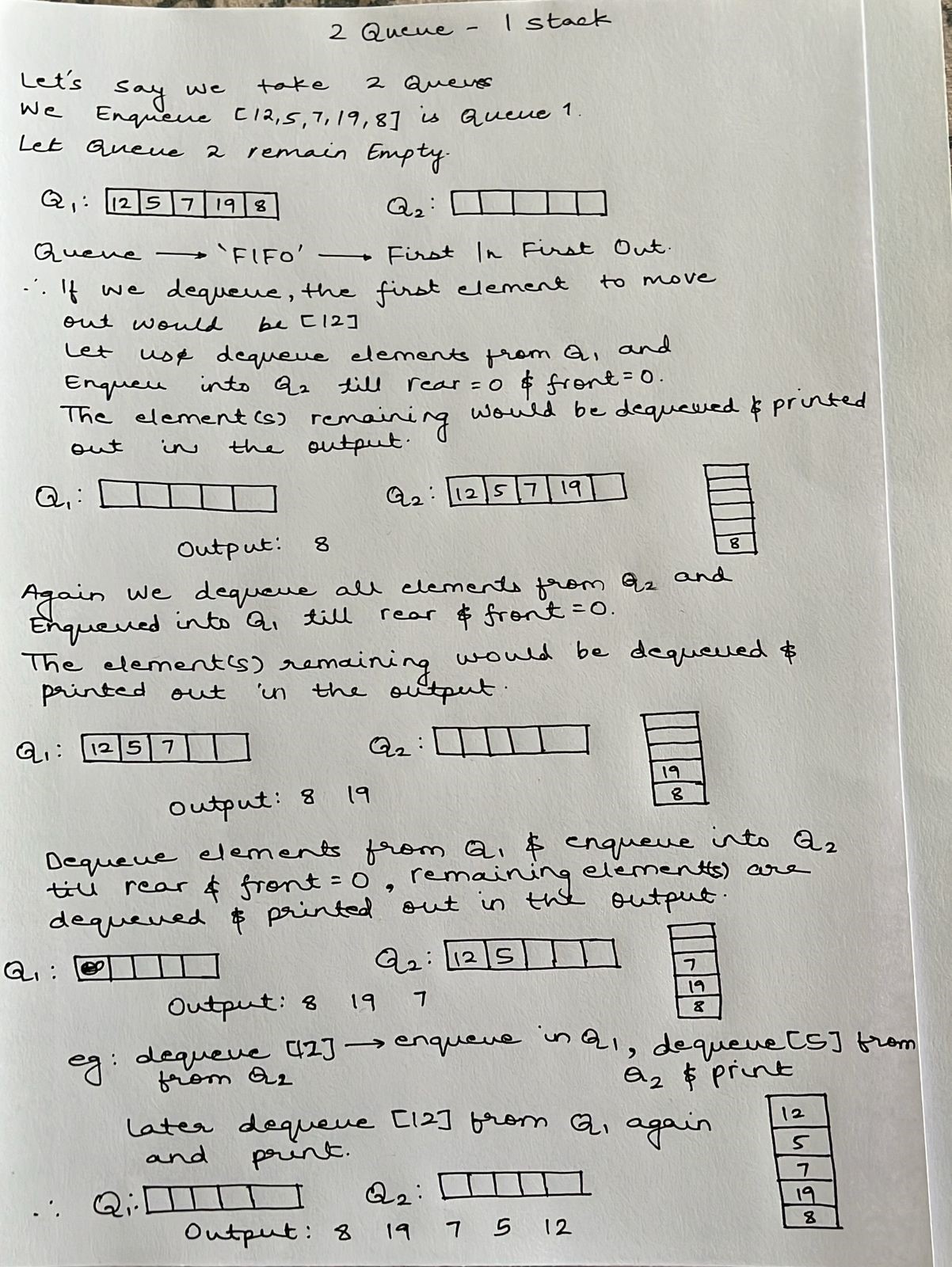
1. The remaining element (The last element) is to be dequeued and printed in the output.
2. Now repeat step 12 for q2 .

1. This process continues till both the queues gets underflowed i.e till the queue gets empty.

1. End.

After following this we find that both the queues get underflowed (empty) and in the output we find all the elements in a stack.

**EXAMPLE:**



**Conclusion:**

This theoretical approach demonstrates how you can implement a stack using two queues efficiently. It also highlights the trade-offs between push and pop operations and helps you appreciate the underlying principles of data structure design and algorithmic efficiency.

ScreenShot:

