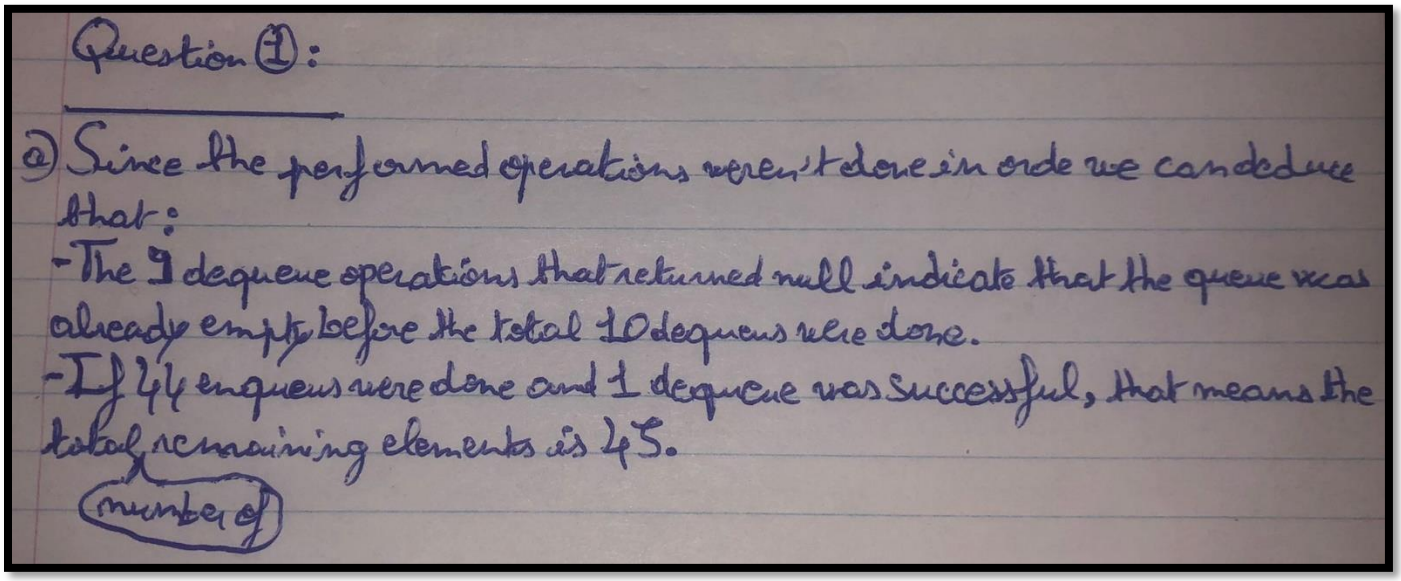
CSI2110-D – Written Assignment #2

Alae Boufarrachene (300188539)

Question 1 :



Question (1):

a) Since the performed operations weren't done in order we can deduce that:
- The 9 dequeue operations that returned null indicate that the queue was already empty before the total 10 dequeues were done.
- If 44 enqueues were done and 1 dequeue was successful, that means the total remaining elements is 45.

number of

b) In order : 5,9,3,3,2,2,7,1

```
stack.push(5) //Returns : Nothing
stack.top() //Returns : 5
stack.push(7) //Returns : Nothing
stack.push(2) //Returns : Nothing
stack.push(3) //Returns : Nothing
stack.push(9) //Returns : Nothing
stack.pop() //Returns : 9
stack.top() //Returns : 3
stack.pop() //Returns : 3
stack.push(1) //Returns : Nothing
stack.push(2) //Returns : Nothing
stack.top() //Returns : 2
stack.pop() //Returns : 2
stack.push(7) //Returns : Nothing
stack.pop() //Returns : 7
stack.pop() //Returns : 1
```

c) In order : 1,3,3,13,70,70,9,8,9

```
deque.addFirst(3) //Returns : Nothing
deque.size() //Returns : 1
deque.first() //Returns : 3
deque.last() //Returns : 3
deque.addLast(2) //Returns : Nothing
deque.addFirst(13) //Returns : Nothing
deque.addLast(70) //Returns : Nothing
deque.first() //Returns : 13
deque.addFirst(11) //Returns : Nothing
deque.last() //Returns : 70
deque.addFirst(1) //Returns : Nothing
deque.removeLast() //Returns : 70
deque.addFirst(9) //Returns : Nothing
deque.addLast(6) //Returns : Nothing
deque.addLast(8) //Returns : Nothing
deque.first() //Returns : 9
deque.last() //Returns : 8
deque.addLast(4) //Returns : Nothing
deque.size() //Returns : 9
```

Alae Boufarrachene (300188539)

Question 2 :

a) *(Java implementation available on the Question2 folder)*

b)

My algorithm is correct because it outputs the expected results, takes into consideration all possible edge cases (empty array/list of intervals, a value of T that's strictly inferior to zero), and the three possible base cases in the type of implementation I decided to adopt.

My implementation is O(n) because the least efficient operations performed in it are two loops that travel through a data structure linearly :
-The first one is a while loop used to navigate the array of busy intervals in a helper method. [O(n)]
-The second one is used to navigate the array of merged busy intervals in the freeTimesToMeet() method. [O(n)]