My algorithm makes an extra pass after the list is sorted. As Dr. Lusth said in class, you should know when it is sorted when nothing is added to the stack. However sometimes on the final pass, something is pushed onto the stack then popped off in the correct order. So then it makes an extra useless pass, resulting in a duplicate sorted list, then exits. In this particular case the logic fails. What am I missing?

It is not sorted if:

1. When the input is empty and the stack is not
2. and, when moving from the stack to the output you have an item out of order

You can test that way.

Seems simpler to ask, did I do any pushes?

Does this check to see if it's sorted have to be in constant time? Or can it be linear?

In class, Lusth said we should not be using an "isSorted" function which checks a queue and sees if it's sorted using comparisons (linear). Rather, keep a flag/bool that keeps track of whether is the list is sorted per the advice above (which is constant).
I have an isSorted function that uses a count of sorted items. If the item that is enqueued onto the output is in the correct order... I add to the count. If the the count is equal to number of items in the input queue - 1. It's sorted. The count is updated while things are being enqueued vs scanning the whole queue to see if it's sorted. You think that counts?

It's constant, so that works. Didn't think about doing it that way.

That's what I thought. Just checking if the cached output value is greater than the new output value vs checking if there was a push.