Question 1:

Consider the following two implementations of a function that if given a list, `lst`, create and return a new list containing the elements of `lst` in reverse order.

```
def reverse1(lst):
    rev_lst = []
    i = 0
    while(i < len(lst)):
        rev_lst.insert(0, lst[i])
        i += 1
    return rev_lst
```

```
def reverse2(lst):
    rev_lst = []
    i = len(lst) - 1
    while (i >= 0):
        rev_lst.append(lst[i])
        i -= 1
    return rev_lst
```

If `lst` is a list of n integers,

1. What is the worst case running time of `reverse1(lst)`? Explain of your answer.
2. What is the worst case running time of `reverse2(lst)`? Explain of your answer.

q2

1) First, an empty array means that its capacity is ≥1, therefore each append call will cost O(1) time. However after n calls, the append function will cost O(n) times because the capacity of the array will need expansion ("doubling") We will not be doubling because we already have capacity [2n]

After appending n times, the array is going to have n items to pop. Therefore, we will call pop() n times and each time it will cost as O(1).

$$2(n * O(1)) = 2n = O(n)$$

2) After n appends for an array, Capacity := n. Suppose we call pop $n/2$ times. By the definition of the strategy given, the array will resize its capacity to $n/2$. This follows another sequence of n appending and $n/2$ popping until the capacity shrink to 1.

Each resize = O(n) and it happens $\frac{n}{2}$ times

$$\frac{n}{2} * O(n) = \frac{1}{2}n^2 = \Omega(n^2)$$

3.b My code uses $O(n)$ runtime.

The first for loop iterates over lst once n times $= O(n)$.

The second for loop iterates over count once which has n elements $\therefore = O(n)$

note that n here has a different value than n in the 1st loop.

$$O(n) + O(n)$$

Overall Complexity

$= O(n)$ (linear time)

4.a Worst case, value is not in lst.

Each remove call will cost $O(n)$ to iterate over all n items of the lst and will be called n times.

$$O(n) \cdot n = O(n^2)$$

4.c My code has 2 for loops (not nested). Both of which contain operations that take constant time. However they ran $O(n)$ times

$= O(n)$