Lab 2B

A) specify a best case and a worst case for Bubble sort

The worst case would be if the input array is in inverse order and we have to swap n times the best case is if the input array is in sorted order

B) what are the best case and worst-case running time for Bubble sort?

The running time would be the same because to see elements to sort we are going through nested for loop. so, the best and worst time is O(n^2).

C) modify Bubble sort so that it has a best-case running time of O(n). call your modified version BubbleSort1. use the sort environment to verify that your modified version runs faster.

I tried to enhance 2 things

1. is the inner loop that was iterating from j=0 to j<n-1 should do as much work because after a given single loop i the nth element in the array is in its sorted position. so, i modified it to be j<n-i-1

2. I introduced a count method that will keep count that will increment its value if the condition in the inner loop is executed. This will be reset back to 0 on the outer loop. so, what this count variable do is, it helps us figure out if in the ith pass of the outer loop there was swapping in the inner loop and if there was no swapping then break from the loop because the array is already sorted.

so, as mentioned above the best case for bubble sort is when the given input is sorted, let's see how my modified version enhance the running time to be O(n).

So, the outer loop will run i=0 count is initialized to be 0
inner loop will scan for all elements from j=0 to j<len-i-1 comparing the consecutive elements arr[j] and arr[j+1] but the condition will always be false because the input is sorted and element arr[j]>arr[j+1], now count is not incremented.

So, the loop will break. The inner loop is going to run up to n since i=0 and our function would look something like this f(n)=cn+c which is O(n).