Exercise Three:

***Question 1:*** Derive the formulas for (i) number of comparisons, and (ii) average-case number of swaps for bubble sort [0.4 pts]

***Answer:***

(i)

Bubble Sort

```
def bubble_sort(arr):
    start_time = timeit.default_timer()

    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

Comparisions occur within the inner loop.

The number of comparisons is the same whether it is worst case, best case, or avg. case.

They happen $1 + 2 + \ldots + (n-2) + (n-1)$ times.

I.e., Big-O $= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2-n}{2} = O(n^2)$.

So bubble sort has a comparison complexity of $O(n^2)$.

(ii)

Bubble Sort

```
def bubble_sort(arr):
    start_time = timeit.default_timer()

    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

For number of swaps:

Best case: List is sorted. 0 swaps would happen.

Worst case: List is in reverse order, swaps would occur for every comparison.

i.e., $\cancel{2} \times \frac{n(n-1)}{\cancel{2}} = n(n-1)$ swaps

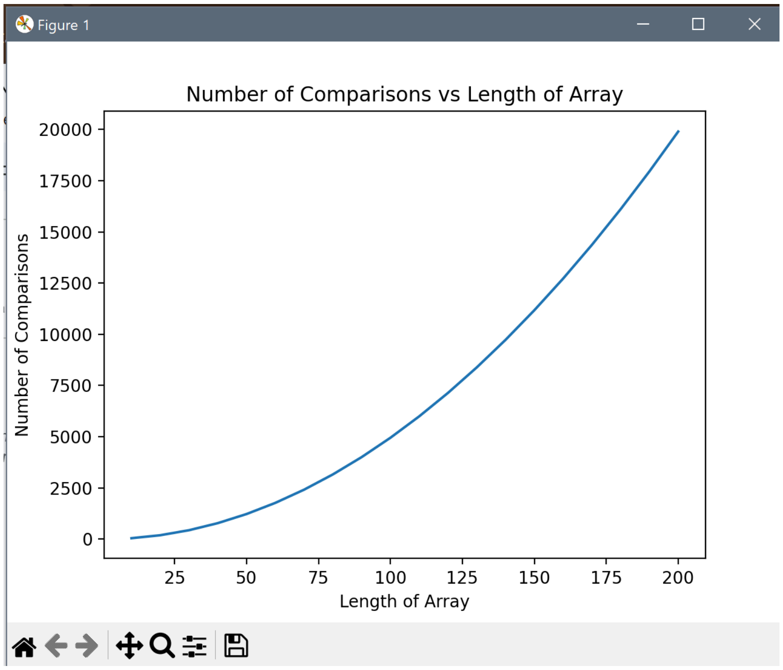
Average case:

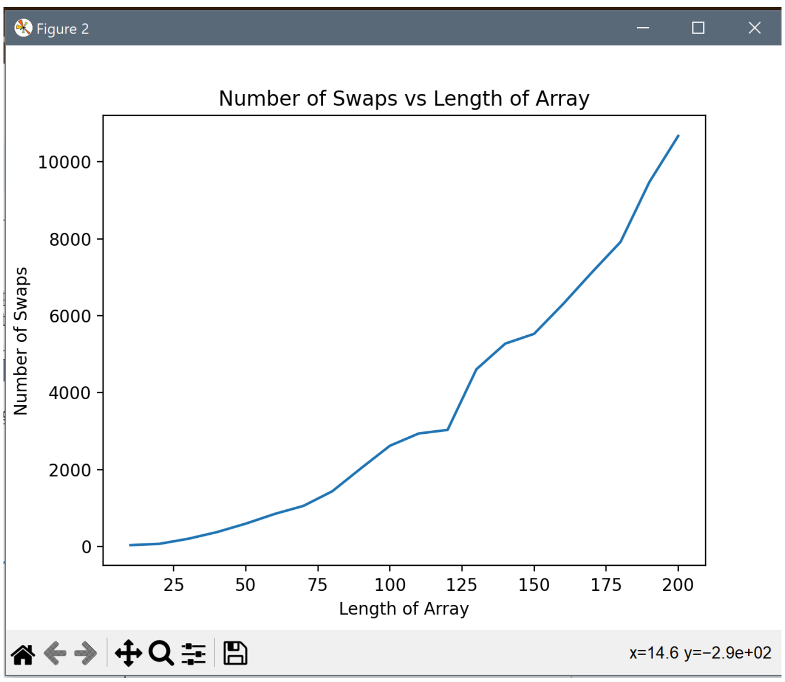
$$\frac{0 + n(n-1)}{2} = \frac{n(n-1)}{2}$$

$O(n^2)$ complexity

***Question 4:***

Separately plot the results of #comparisons and #swaps by input size, together with appropriate interpolating functions. Discuss your results: do they match your complexity analysis? [0.2 pts]




***Answer:*** In our analysis we predicted both the average number of swaps and the number of comparisons to have an overall complexity of n squared. Both of these graphs correlate with that assumption. As each graph resembles that of a degree two function.