Exercise#3

1) i) Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

The number of comparisons is the same for the best, average and worst cases, which is,

$$\frac{[n(n-1)]}{2}$$

The number of comparisons can be calculated as follows:

1. In the first pass, n-1 comparisons are made.
2. In the second pass, n-2 comparisons are made.
3. This pattern continues until the last pass, where only one comparison is made.

So, if we sum up all of them till n-1 positive integers, we can get the formula

ii) The average-case analysis for sorting algorithms considers all possible input permutations and calculates the average number of swaps. In the case of Bubble Sort, the formula goes like,

$$\frac{[n(n-1)]}{4}$$

Previously we got the number of comparisons in a pass is approximately [n(n-1)]/2. Now, for each comparison, there may or may not be a swap. Supposing, on average half of the comparisons result in swaps, then the average number of swaps in a pass is (n-1)/4. Now, if we add them all up and multiply it with the number oof passes we get [n(n-1)]/4. A point to be noted, this swap process execution is based on the random order for average case.

2) The implementation of bubble sort goes below:

```
def bubble_sorting(arr):
  n = len(arr)
  comparisons = 0
  swaps = 0

  for i in range(n):
    for j in range(0, n-i-1):
      comparisons += 1

      if arr[j] > arr[j+1]:
```

```
            swaps += 1
            arr[j], arr[j+1] = arr[j+1], arr[j]

    return arr, comparisons, swaps
```

3) Testing the code with different sizes of inputs (increasing size) is shown below:

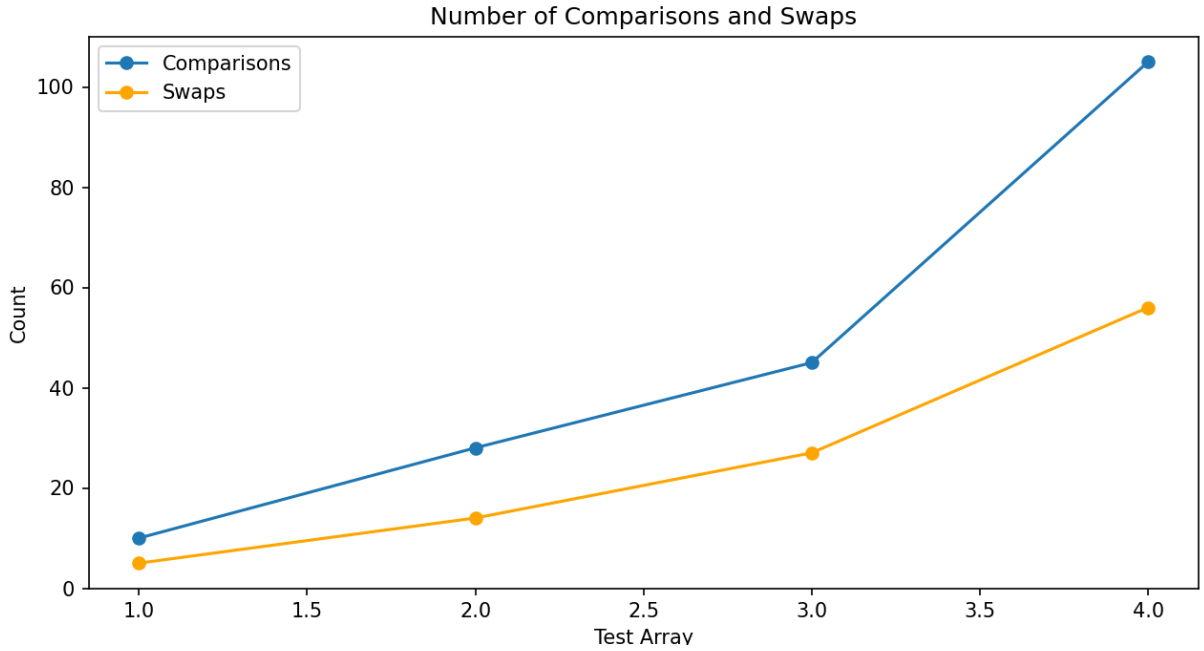
```
def bubble_sort_with_counts(arr):
    n = len(arr)
    comparisons = 0
    swaps = 0

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

    return arr, comparisons, swaps

def test_bubble_sort(array):
    sorted_array, comparisons, swaps = bubble_sort_with_counts(array.copy())
    print("Original Array:", array)
    print("Sorted Array:", sorted_array)
    print("Comparisons:", comparisons)
    print("Swaps:", swaps)
    print()

# Test arrays of increasing size
array1 = [5, 2, 9, 1, 5]
```

```
array2 = [8, 3, 6, 2, 7, 1, 4, 9]

array3 = [12, 6, 2, 9, 5, 7, 10, 1, 8, 3]

array4 = [15, 8, 4, 11, 2, 10, 7, 14, 1, 6, 12, 3, 9, 5, 13]


test_bubble_sort(array1)

test_bubble_sort(array2)

test_bubble_sort(array3)

test_bubble_sort(array4)
```

4) The code is given below where is plots the result of #comparison and #swaps by the input size of the array and the plot is given below right after the source code.

```
import matplotlib.pyplot as plt


def bubble_sort_with_counts(arr):
    n = len(arr)
    comparisons = 0
    swaps = 0

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

    return arr, comparisons, swaps


def test_bubble_sort(array):
    sorted_array, comparisons, swaps = bubble_sort_with_counts(array.copy())
```

```
    return comparisons, swaps

array1 = [5, 2, 9, 1, 5]
array2 = [8, 3, 6, 2, 7, 1, 4, 9]
array3 = [12, 6, 2, 9, 5, 7, 10, 1, 8, 3]
array4 = [15, 8, 4, 11, 2, 10, 7, 14, 1, 6, 12, 3, 9, 5, 13]

arrays = [array1, array2, array3, array4]
comparisons_list = []
swaps_list = []

for array in arrays:
    comparisons, swaps = test_bubble_sort(array)
    comparisons_list.append(comparisons)
    swaps_list.append(swaps)

plt.figure(figsize=(10, 5))

plt.plot(range(1, len(arrays) + 1), comparisons_list, marker='o', label='Comparisons')
plt.plot(range(1, len(arrays) + 1), swaps_list, marker='o', color='orange', label='Swaps')

plt.title('Number of Comparisons and Swaps')
plt.xlabel('Test Array')
plt.ylabel('Count')
plt.legend()

plt.show()
```



Yes, it follows and matches the complexity analysis as discusses.