Keiffer Tan

Bubble Sort Analysis

```python
"""Simplest Bubble Sort implementation without optimizations."""
def bubble_sort(arr):
    n = len(arr)

    # Nested loops to traverse the array n*n times
    for _ in range(n - 1):
        for i in range(n - 1):
            if arr[i] > arr[i + 1]:
                # If the previous index is larger, then swap spots
                arr[i], arr[i + 1] = arr[i + 1], arr[i]
    return arr

"""Simplest Bubble Sort implementation without optimizations for tuple input stability testing."""
def bubble_sort_tuple(arr):
    n = len(arr)

    # Nested loops to traverse the array n*n times
    for _ in range(n - 1):
        for i in range(n - 1):
            if arr[i][0] > arr[i + 1][0]:
                # If the previous index is larger, then swap spots
                arr[i], arr[i + 1] = arr[i + 1], arr[i]
    return arr
```

The nested loops for the based case of bubble sort will make the worst-case time complexity of O(n^2) in all cases even if the given array is already sorted order as it is not optimized. (which I will discuss later).

The space complexity of the bubble sort algorithm is O(1) or constant because it is considered an in-place sorting algorithm. We are not allocating new memory for sorting but just swapping the values of the adjacent indexes.

For Stability, bubble sort is considered a stable algorithm and will preserve the relative order of same value pairs. I had to create a tuple version of the bubble sort in my algorithm so that it will only compare the first value and not the second value in the tuple below. The order of the 2nd tuple values are preserved when placing the array into the algorithm.

```python
arr = [(2, 'b'), (2, 'a'), (1, 'c'), (1, 'a')]
expected = [(1, 'c'), (1, 'a'), (2, 'b'), (2, 'a')]
```

```python
sorted_arr = bubble_sort_tuple(arr.copy())
# print('HERE ->>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>',sorted_arr)
self.assertEqual(sorted_arr, expected)
self.assertEqual(bubble_sort_tuple_optimized(arr.copy()), expected)
```

In my test file, I have thoroughly tested both my base bubble sort and optimized bubble sort algorithms to ensure that it is sorting all cases and edge cases while preserving relative order of same value pairs.

Keiffer Tan

```python
""" Implementation of Bubble Sort Optimized with early stopping"""
def bubble_sort_optimized(array):
    n = len(array)

    # iterate through array until no swaps are made
    while True:
        swapped = False

        for i in range(n - 1):
            if array[i] > array[i+1]:
                # If the previous index is larger, then swap spots
                array[i+1], array[i] = array[i], array[i+1]
                swapped = True # flag that a swap has been made

        # Break if there are no swaps made to reduce amount of iterations
        if swapped == False:
            break

    return array
```

In my optimized array, I implement a swapped flag that indicates whether a swap has been made during the iteration of the array when comparing adjacent values. When going through an iteration of the array and a swap between adjacent values are not made, the algorithm will prematurely break from the loop knowing that a swap has not been made. This results in best case scenario of O(n) if the algorithm is already sorted or nearly sorted. The space complexity with the non-optimized bubble sort is the same as it is both still in-place sorting.