Performance Analysis of Sorting Algorithms

In this assignment, you will explore and compare the performance of three sorting algorithms: Bubble Sort, Insertion Sort, and Selection Sort. You will implement these algorithms and analyze their efficiency and number of swaps under various scenarios. Finally, you will draw conclusions based on your findings.

Part 1: Implementation

Create file `hw6.py` and implement the following functions:

- `bubble_sort(L)`
  - Implement the Bubble Sort algorithm in Python. Ensure it is adaptive (runs in linear time on sorted input without negatively impacting general performance).
- `insertion_sort(L)`
  - Implement the Insertion Sort algorithm in Python. Ensure it is adaptive as well.
- `selection_sort(L)`
  - Implement the Selection Sort algorithm in Python.
- Feel free to write any additional functions that may be necessary to populate the results required in part 2.

Part 2: Performance Analysis

You will analyze the performance of these sorting algorithms under different input conditions:

Case A: Random List

In this scenario, the list contains elements randomly arranged in no particular order. It represents a common real-world situation where data is unorganized and needs to be sorted efficiently. You can use module `random` to generate the lists.

Case B: Sorted List

This scenario features a list with elements sorted in ascending order from start to end.

Case C: Reverse Order List

In this case, the list consists of elements sorted in descending order, meaning the largest elements are at the beginning.

Case D: Move 5% of the elements in a sorted list from end to front

This scenario involves taking a sorted list and relocating 5% of its elements from the end to the front. For example, suppose you have a sorted list of integers from 1 to 100:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ... 96, 97, 98, 99, 100
```

Now, the list anfter moving 5% of the elements from the end to the front:

```
96, 97, 98, 99, 100, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ... 95
```

Case E: Move 5% of the elements in a sorted list from front to end

Similar to Case D, here, 5% of the elements in a sorted list are moved, but this time from the front to the end. For example, suppose you have a sorted list of integers from 1 to 100:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ... 96, 97, 98, 99, 100
```

Now, the list anfter moving 5% of the elements from the front to the end:

```
6, 7, 8, 9, 10, ... 96, 97, 98, 99, 100, 1, 2, 3, 4, 5
```

For each case:

- Create a list of integers of different sizes (e.g., 2000, 3000, 4000, 5000, and 6000).
- Apply all three sorting algorithms to sort the lists.
- Record and compare the number of swaps made by each algorithm.
- Measure and compare the execution time of each algorithm.
- Please note that you should not import any modules other than `timeit` and `random`.
- Display the data in a tabular format, as shown below:

Table example: Performance Analysis of Sorting Algorithms

```
   Scenario    |List Size|  Bubble Sort       | Insertion Sort    |   Selection
-------------------------------------------------------------------------------------
               |         |    Swaps, time     |   Swaps, time     |   Swaps, time
-------------------------------------------------------------------------------------
   Random      |   2000  | (999346, 0.27772)  |(999346, 0.14726)  |(1994, 0.06436)
   ....
   Random      |   6000  | (2224648, 0.60708) |(2224648, 0.36831) |(2998, 0.15501)
-------------------------------------------------------------------------------------
   Sorted      |   2000  |      (0, 0.0)      |   (0, 0.001)      | (0, 0.07265)
   ....
   Sorted      |   6000  |      (0, 0.0)      |   (0, 0.001)      | (0, 0.16399)
-------------------------------------------------------------------------------------
   Reverse     |   2000  | (1999000, 0.33649) |(1999000, 0.29888) |(1000, 0.06772)
   ....
   Reverse     |   6000  | (4498500, 0.76627) |(4498500, 0.67564) |(1500, 0.15437)
-------------------------------------------------------------------------------------
Move front to end|  2000 | (360000, 0.19772)  | (360000, 0.05679)|(1800, 0.07376)
....
Move front to end|  6000 |  (810000, 0.4402)  | (810000, 0.12483)|(2700, 0.15929)
-------------------------------------------------------------------------------------
Move end to front|  2000 | (360000, 0.05093)  | (360000, 0.05691)|(1800, 0.07171)
....
Move end to front|  6000 | (810000, 0.11288)  | (810000, 0.12666)|(2700, 0.1587)
```

Number of swaps and execution time are recorded for each algorithm (values are for illustration purposes only).

Part 3: Conclusion

Based on your performance analysis, write a conclusion that addresses the following points:

1. Provide a ranking of the sorting algorithms based on their performance in each scenario. Which sorting algorithm performed best in each case (Random, Sorted, Reverse Order, Move End to Front, Move Front to End), and why do you think it performed better?

2. Explain how the number of swaps made by each algorithm affects both efficiency and time complexity across different scenarios.

3. Discuss the impact of the initial order of data on how well each algorithm performs. Explain why some algorithms perform differently on sorted, reversed, or random lists.

Submission:

1. File `hw6.py`

- The file should include the three required sorting functions (Bubble Sort, Insertion Sort, Selection Sort).
- Any additional functions created to populate the results should also be included.
- At the end of `hw6.py` add comments that answer the questions from Part 3.
- Following the conclusion comments, include the results table in the form of comments.

2. File `test_hw6.py`

- Create test cases to verify that the sorting functions in `hw6.py` work correctly.

Please note that the homework is 100% manually graded