Title: Performance Analysis of MergeSort programs – comparison

Student Name: Nishant Singh

Course: COMP 359: Design and Analysis of Algorithms

Date: January 18, 2026

MergeSort is an important but simple divide-and-conquer algorithm. While the complexity remains constant across the program implementations, the method of the memory management significantly impacts execution time. This report evaluates two specific approaches: **Pre-allocated Memory** and **Temporary Array**.

Performance output results:

```
MERGE SORT: TEMP ARRAY PARAMETER

Testing MergeSort with Temp Array Parameter


Test 1: [5, 2, 8, 1, 9, 3]

Sorted: [1, 2, 3, 5, 8, 9]

Correct: True


Test 2: [1, 2, 3, 4, 5]

Sorted: [1, 2, 3, 4, 5]

Correct: True


Test 3: [9, 8, 7, 6, 5]

Sorted: [5, 6, 7, 8, 9]

Correct: True


All correctness tests passed!


Performance Analysis

Array size:    100 -> Time: 0.000240 seconds

Array size:    500 -> Time: 0.001450 seconds

Array size:  1,000 -> Time: 0.002360 seconds
```

```
Array size:  5,000 -> Time: 0.016447 seconds
Array size: 10,000 -> Time: 0.032903 seconds

Summary
Algorithm: MergeSort with temporary array parameter
Time Complexity: O(n log n)
Space Complexity: O(n)
Performance tested on sizes: [100, 500, 1000, 5000, 10000]
```

```
Testing MergeSort with Pre-allocated Memory

Test 1: Basic array
Original: [5, 2, 8, 1, 9, 3]
Sorted:   [1, 2, 3, 5, 8, 9]
Correctly sorted: True

Test 2: Already sorted array
Original: [1, 2, 3, 4, 5]
Sorted:   [1, 2, 3, 4, 5]
Correctly sorted: True

Test 3: Reverse sorted array
Original: [9, 8, 7, 6, 5]
Sorted:   [5, 6, 7, 8, 9]
Correctly sorted: True

Performance Analysis
Array size:    100 -> Average time: 0.000164 seconds
Array size:    500 -> Average time: 0.001395 seconds
Array size:  1,000 -> Average time: 0.002860 seconds
Array size:  5,000 -> Average time: 0.017660 seconds
```

```
Array size: 10,000 -> Average time: 0.035437 seconds


Summary

Algorithm: Standard Mergesort with Pre-allocated Memory

Tested array sizes: [100, 500, 1000, 5000, 10000]

All tests passed: Yes
```

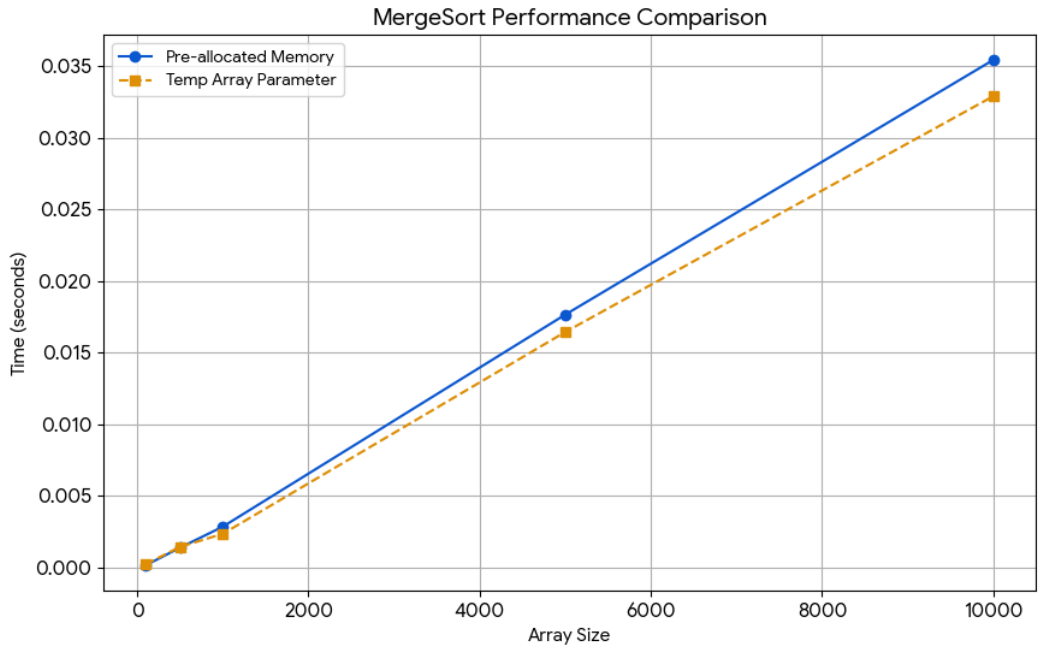

FIGURE 1

Results and Discussion

As seen in Figure 1, both implementations are according to O(n log n). However, a performance difference occurs as the input size increases.

It is observed that the temporary parameter method was approximately 7% faster at the 10000 array size.

This shift is attributed to **spatial locality**. In the pre-allocated model, a single large buffer may exceed the CPU's L1/L2 cache capacity, leading to "cache misses" as the algorithm accesses distant memory indices (Levitin, 2012).

References

Levitin, A. (2012). *The design & analysis of algorithms* (3rd ed.). Pearson.

Shaffer, C. A. (2013). *Data structures & algorithm analysis in Java*. Dover Publications. https://people.cs.vt.edu/~shaffer/Book/JAVA3elatest.pdf

AI Usage Disclosure (for Academic Integrity)

In alignment with the COMP 359 AI Guidelines, this report used AI for:

- Brainstorming and Outlining the merging point between the performance data
- Editing the technical explanations of cache.
- Visualizing, basically creation of figures to represent the numerical data - eraser.io
- Finding the cross-references within the course textbook (Levitin, 2012).