Analysis of Two Sorting Algorithms

Reid Bandy
East Tennessee State University

***Abstract*—This report analyses Bubble and Merge sort algorithms across input lengths and between two compatible types. We see that Bubble sort performs worse and gets progressively worse at scale than Merge sort in all situations.**

I. Introduction

Sorting algorithms are fundamental in computer science for a variety of operations and are critical to many programs and systems. This experiment compares the performance of merge and bubble sort across different list sizes, data types, and how sorted the initial list is. By analyzing these factors, we can get a good understanding of how these algorithms hold up in different conditions.

II. Experimental Setup

We implemented Bubble Sort and Merge Sort algorithms in C# and tested them with two types of data:

- **Random Numbers**: Value type data consisting of integers.
- **Book Artificial Class**: Reference type data consisting of instances of a custom `Book` class composed of strings.

And two initial states:

- **Random**: A randomly ordered list
- **Semi-Random**: A list where each element has a 5% chance of being out of order, or more specifically each element has a 10% of not being sorted.

For each data type, we tested the algorithms with varying numbers of elements: 10, 100, 1,000, and 10,000.

III. Results

A. Random Numbers

The runtimes for sorting random numbers are presented in Table I.

TABLE I: Runtime for Sorting Random Numbers

Algorithm	Initial Sort	Iterations	Time (ms)
BubbleSort	Random	10	0
BubbleSort	SemiSorted	10	0
MergeSort	Random	10	0
MergeSort	SemiSorted	10	0
BubbleSort	Random	100	3
BubbleSort	SemiSorted	100	3
MergeSort	Random	100	1
MergeSort	SemiSorted	100	1
BubbleSort	Random	1,000	294
BubbleSort	SemiSorted	1,000	253
MergeSort	Random	1,000	10
MergeSort	SemiSorted	1,000	9
BubbleSort	Random	10,000	34,636
BubbleSort	SemiSorted	10,000	24,227
MergeSort	Random	10,000	124
MergeSort	SemiSorted	10,000	121

B. Book Artificial Class

The runtimes for sorting the `Book` class instances are presented in Table II.

TABLE II: Runtime for Sorting Book Class Instances

Algorithm	Initial Sort	Iterations	Time (ms)
BubbleSort	Random	10	3
BubbleSort	SemiSorted	10	3
MergeSort	Random	10	3
MergeSort	SemiSorted	10	3
BubbleSort	Random	100	77
BubbleSort	SemiSorted	100	79
MergeSort	Random	100	33
MergeSort	SemiSorted	100	36
BubbleSort	Random	1,000	4,048
BubbleSort	SemiSorted	1,000	4,168
MergeSort	Random	1,000	332
MergeSort	SemiSorted	1,000	397
BubbleSort	Random	10,000	458,791
BubbleSort	SemiSorted	10,000	471,868
MergeSort	Random	10,000	4,320
MergeSort	SemiSorted	10,000	4,971

C. Graphs

Figures 1 and 2 illustrate the runtimes of the algorithms for different numbers of elements and initial list states.

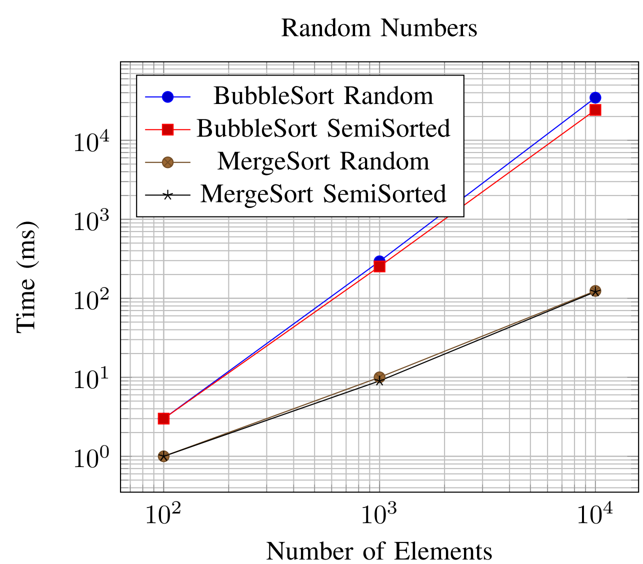

Fig. 1: Runtime Comparison for Random Numbers

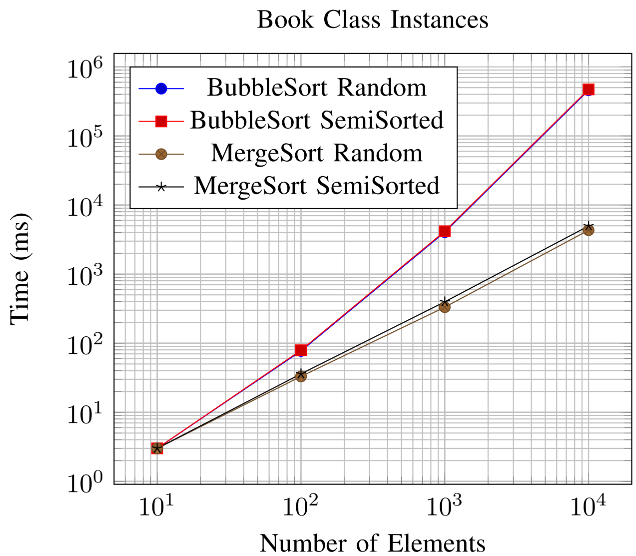

Fig. 2: Runtime Comparison for Book Class Instances

IV. Discussion

A. Effect of Number of Elements

As the number of elements increases, the runtime for both algorithms increases. However, Merge Sort consistently outperforms Bubble Sort. This is due to Merge Sort's $O(n \log n)$ complexity, while Bubble Sort operates at $O(n^2)$.

B. Effect of Initial Sort State

Semi-sorted lists reduce the runtime for both algorithms compared to random lists, but the effect is more noticeable for Bubble Sort. Bubble Sort's runtime drops significantly with semi-sorted data, especially for larger datasets, because it has to perform fewer swaps in nearly sorted data.

C. Effect of Data Type

Sorting the `Book` class instances took significantly longer than sorting random numbers. This is because comparing Book types involves at least, and in almost all cases, two pointer dereferences and at least, and in almost all cases, one string comparison.

D. Algorithm Performance

1) Best Performance: Merge Sort performed best in all conditions, especially with larger datasets.

2) Worst Performance: Bubble Sort performed worst, particularly with larger datasets of reference types. Even though it benefited more from semi-sorted lists, it was far from compensated with diminished scalability and overall worse performance.

V. Conclusion

This experiment demonstrated that Merge Sort is consistently more efficient than Bubble Sort, particularly for large datasets. The number of elements, data type, and initial list state all have significant impacts on sorting algorithm performance.