Custom Sorting Algorithm Performance Analysis

Luis Gabriel Caceres Duran

Data structures & algorithms

Game Development, NBCC

October 3, 2024

Abstract

This report analyzes the performance of the CustomSort algorithm, which sorts data in chunks of 500 elements using C#'s built-in .Sort() method. It compares CustomSort with three AI-generated sorting algorithms: BubbleSortAI, InsertionSortAI, and SelectionSortAI. The evaluation focuses on efficiency in handling both unsorted and pre-sorted arrays, with each algorithm tested three times under identical conditions to record average sorting times.

Keywords: sorting algorithms, time complexity, c# built-in sort, optimization

Custom Sorting Algorithm Performance Analysis

Superior Performance of .Sort() Method

The built-in .Sort() method in C# outperforms traditional sorting algorithms like Bubble Sort, Insertion Sort, and Selection Sort. As seen in Figure 1, sorting 1,000,000 elements took CustomSort 142 milliseconds, while BubbleSortAI took over 55 minutes. Similarly, in Figure 2, sorting 10,000 pre-sorted elements took CustomSort 301 microseconds, whereas BubbleSortAI took 24.5 milliseconds. This superior performance is primarily because .Sort() uses Timsort, a highly optimized algorithm with a time complexity of $O(n \log n)$, which is far more efficient than the $O(n^2)$ time complexity of algorithms like Bubble Sort and Selection Sort (Microsoft, n.d.).

Memory Efficiency

Sorting an array with .Sort() is generally much faster than sorting more complex data structures like linked lists or trees, where memory references slow down the process. Therefore, the choice of data structure plays a key role in the relative performance of .Sort(). While it is optimized for arrays, other data structures may incur more overhead during the sorting process because it requires dereferencing memory pointers, which adds to the processing time.

Conclusion

The CustomSort algorithm, which leverages the built-in .Sort() method, performs dramatically better than traditional AI-generated sorting algorithms like BubbleSortAI and SelectionSortAI. For large datasets, the performance difference is orders of magnitude, with CustomSort consistently completing tasks in milliseconds compared to the minutes required by the other algorithms. However, when using objects or complex data structures, there is a computational cost to consider, as the overhead of handling objects can introduce delays. Despite this, CustomSort is an excellent choice for most practical scenarios, especially when sorting large datasets or when the sorting can be divided into chunks, as in the case of the 500-element restriction. Its ability to leverage built-in optimizations makes it far more efficient than conventional sorting algorithms.

AI Generated Sorting Algorithms Optimization Analysis

Microsoft. (n.d.). Array.Sort Method. Microsoft Documentation.

https://learn.microsoft.com/en-us/dotnet/api/system.array.sort

Figure 1

Sorted Arrays Custom Sort Performance

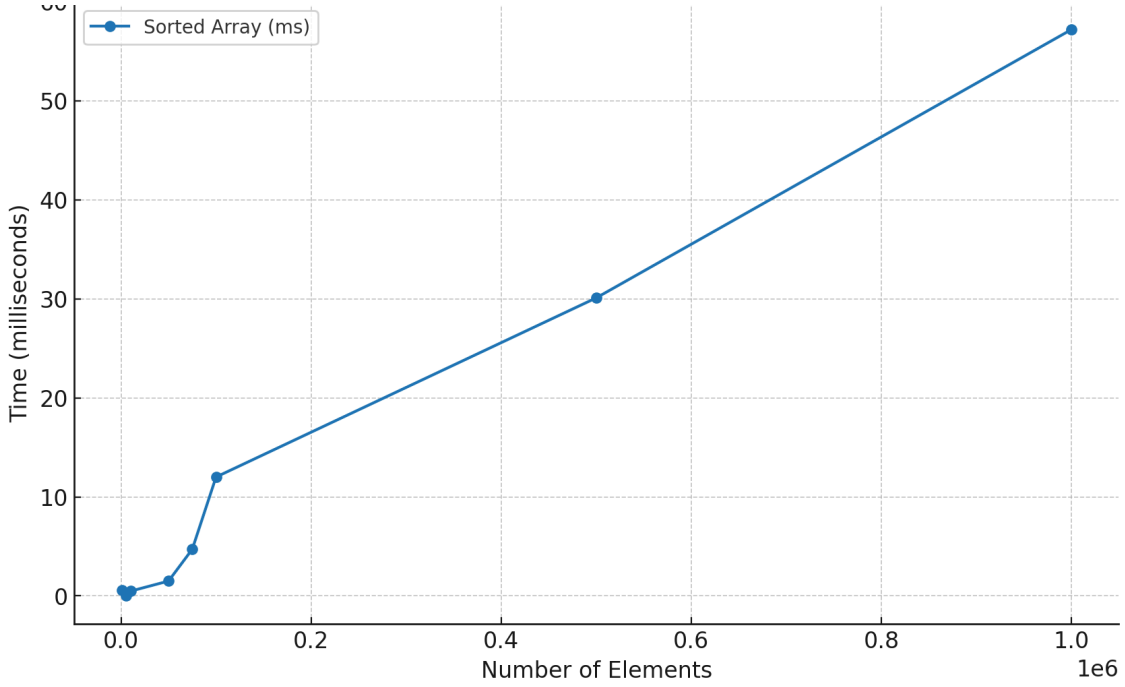

Figure 2

Unsorted Arrays Custom Sort Performance