AI Generated Sorting Algorithms Optimization Analysis:

BubbleSort, SelectionSort, and InsertionSort

Luis Gabriel Caceres Duran

Data structures & algorithms

Game Development, NBCC

October 3, 2024

Abstract

This analysis compares the performance of original and optimized versions of three AI-generated sorting algorithms—BubbleSortAI, InsertionSortAI, and SelectionSortAI—across various input sizes for both unsorted and already sorted arrays. The optimizations aimed to enhance readability and efficiency through specific algorithmic improvements.

Keywords: sorting algorithms, time complexity, ai generated, optimization

AI Generated Sorting Algorithms Optimization Analysis

Optimizations Implemented

BubbleSortOptimized: Replaced the traditional for-loop with a while-loop to allow early termination if the array is already sorted, enhancing code readability with a slight performance trade-off. InsertionSortOptimized: Implemented binary search to find the insertion point, reducing the number of comparisons in larger datasets. SelectionSortOptimized: Adopted a bidirectional selection technique to simultaneously pick the smallest and largest elements, effectively halving the number of iterations.

Performance Summary

The following summarizes the results in Figure 9. BubbleSortOptimized: Up to 48.6% slower than the AI version on sorted arrays; offers improved code readability despite minor performance decreases. InsertionSortOptimized: Achieved up to 41% faster performance on unsorted arrays but was up to 1,800% slower on sorted arrays due to binary search overhead.SelectionSortOptimized: Demonstrated consistent improvements, being up to 26% faster on unsorted arrays and 45% faster on sorted arrays, making it effective for larger datasets. The percentage improvement was calculated using the formula in Figure 10.

Conclusion

The optimized sorting algorithms exhibit varying degrees of performance changes: BubbleSortOptimized: Shows minor performance decreases but offers cleaner code, suitable for small to medium datasets where readability is a priority. InsertionSortOptimized: Provides significant improvements for unsorted data sets but is not recommended for sorted arrays due to the overhead introduced by binary search. SelectionSortOptimized: Delivers consistent performance gains in both unsorted and sorted cases, making it a versatile choice for larger datasets. Overall, while optimizations can enhance algorithm efficiency, their benefits are context-dependent. SelectionSortOptimized emerges as the most balanced option, offering steady improvements without significant drawbacks.

Figure 1

Unsorted Arrays Performance Comparison: BubbleSortAIOptimized, InsertionAiOptimized, SelectionSortAIOptimized

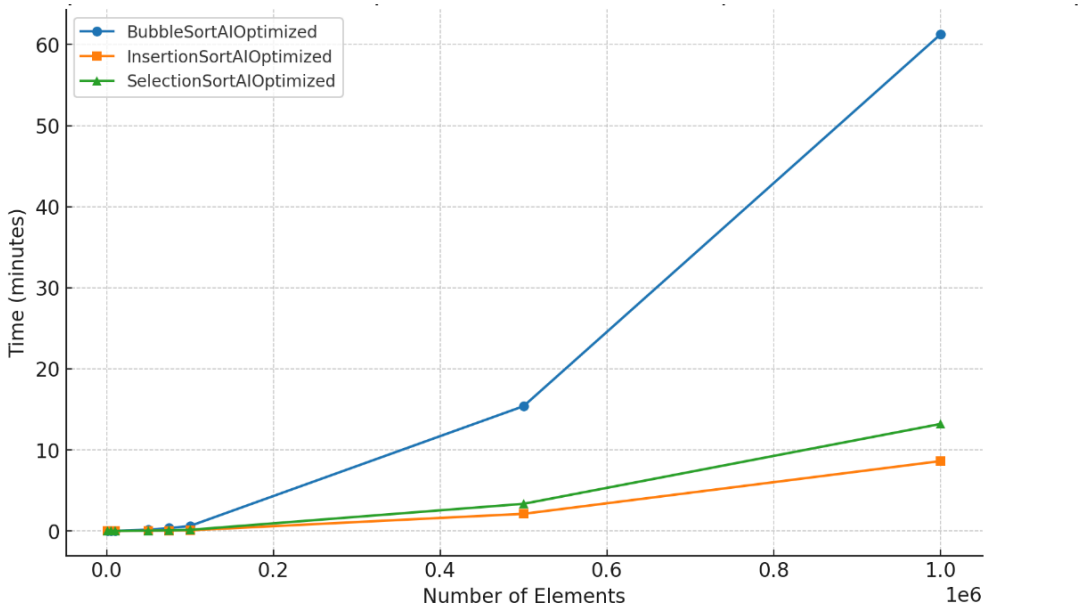

Figure 2

Sorted Arrays Performance Comparison: BubbleSortAIOptimized, InsertionAiOptimized, SelectionSortAIOptimized

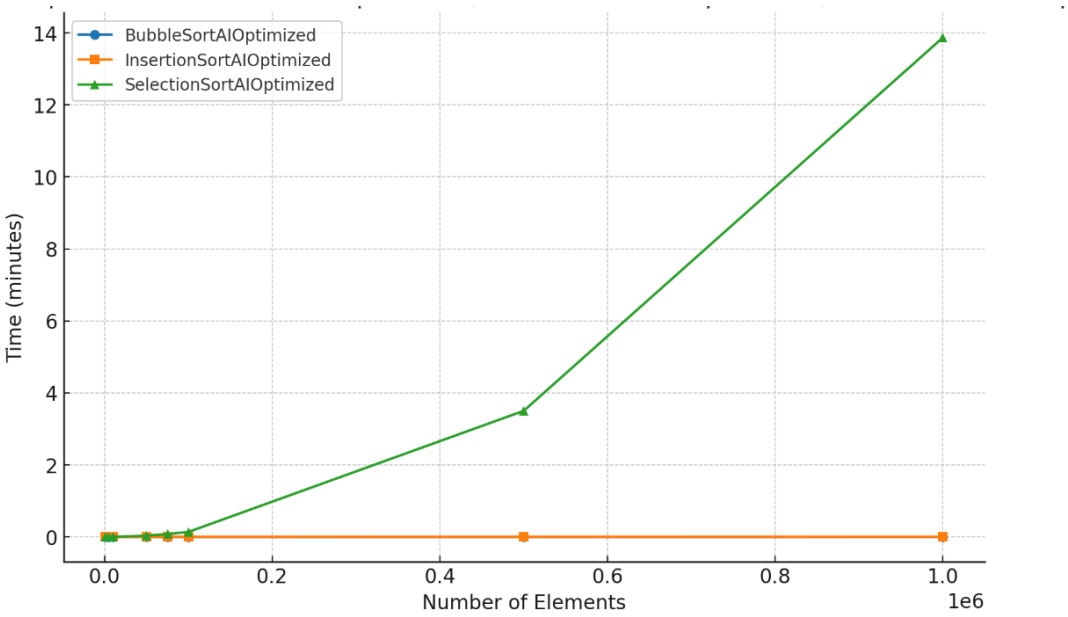

Figure 3

Unsorted Arrays Performance Comparison: BubbleSortAI, BubbleSortAIOptimized

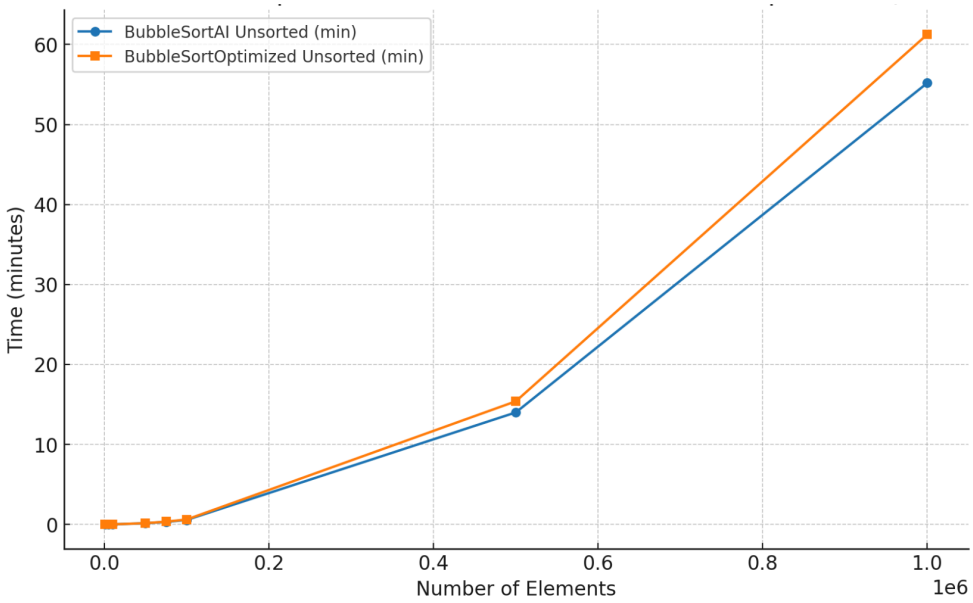

Figure 4

Sorted Arrays Performance Comparison: BubbleSortAI, BubbleSortAIOptimized

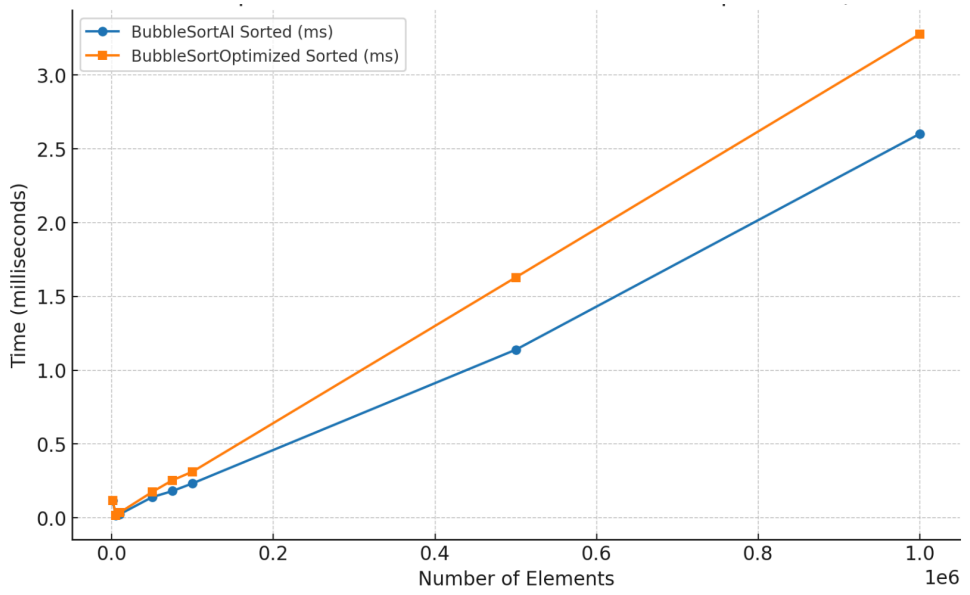

Figure 5

Unsorted Arrays Performance Comparison: InsertionSortAI, InsertionSortAIOptimized

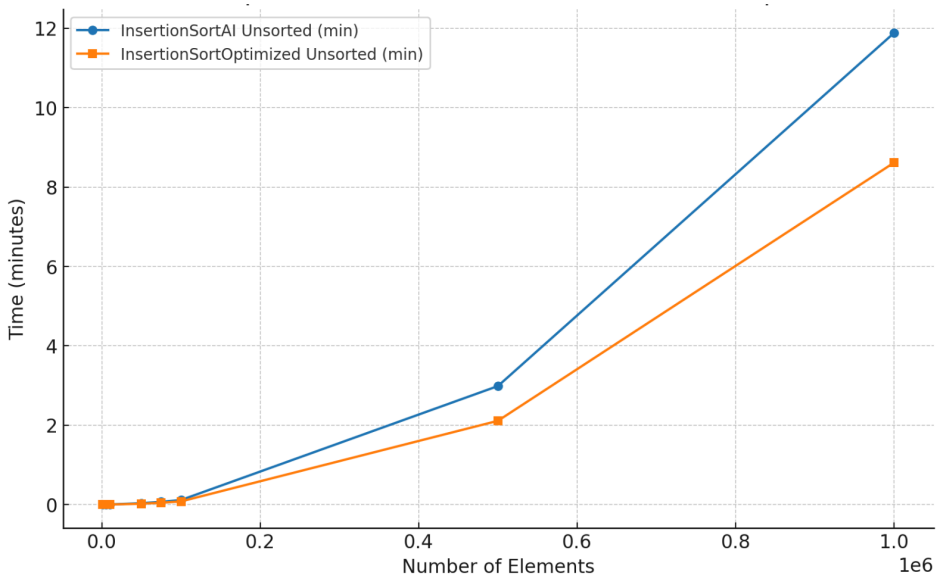

Figure 6

Sorted Arrays Performance Comparison: InsertionSortAI, InsertionSortAIOptimized

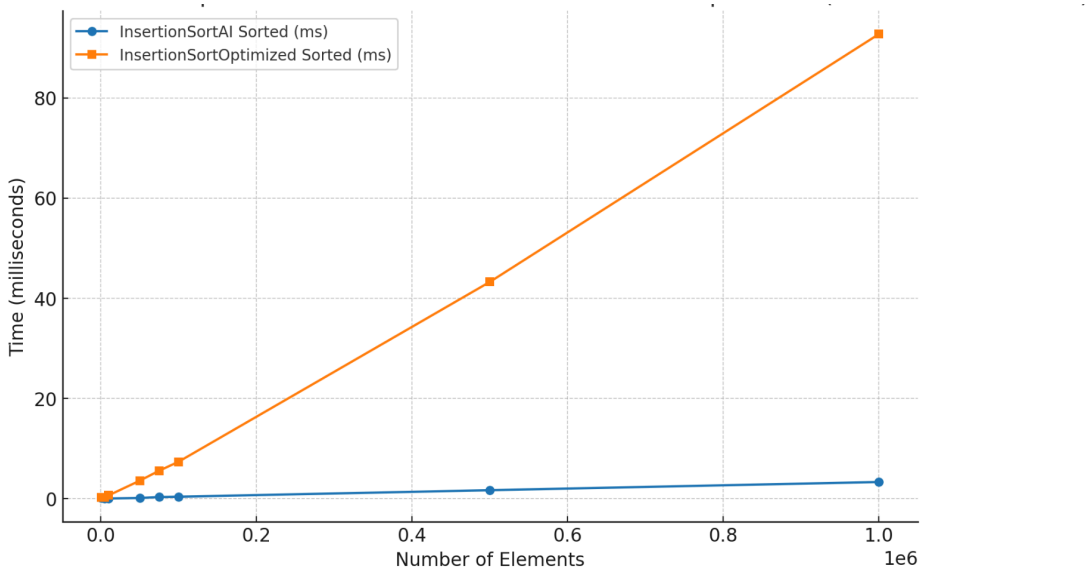

Figure 7

Unsorted Arrays Performance Comparison: SelectionSortAI, SelectionSortAIOptimized

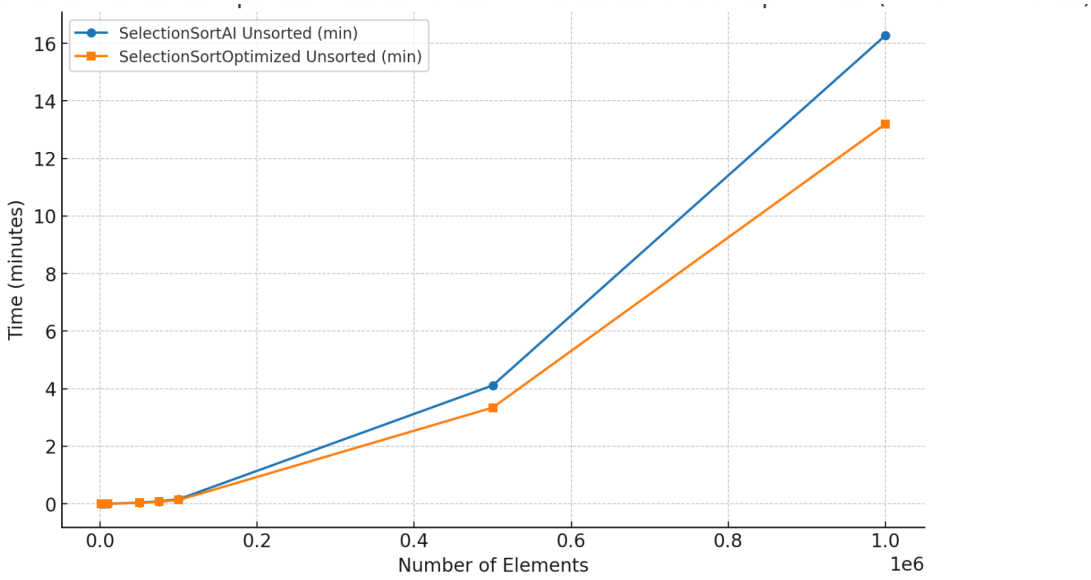

Figure 8

Sorted Arrays Performance Comparison: SelectionSortAI, SelectionSortAIOptimized

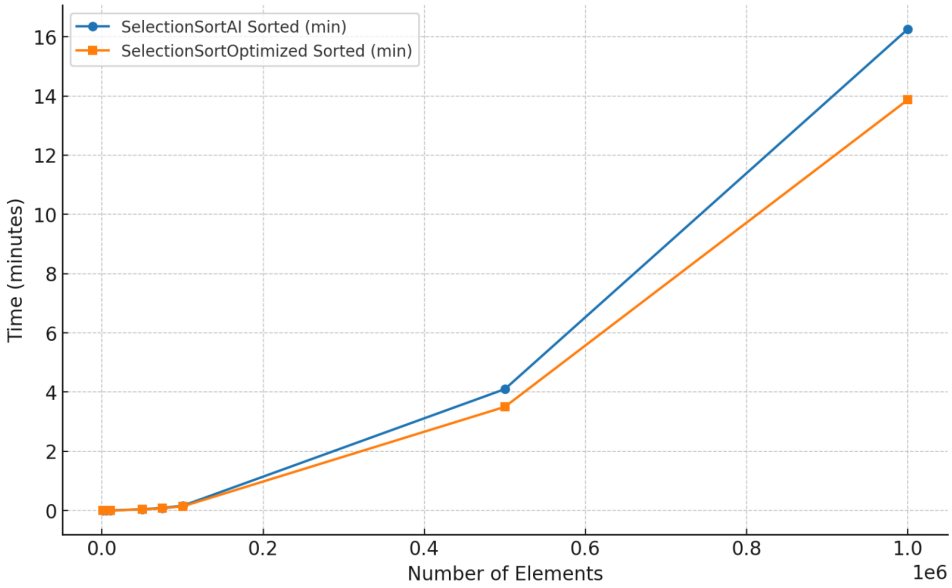

Figure 9

Performance summary table

Algorithm	Unsorted Improvement (%)	Sorted Improvement (%)
BubbleSortOptimized	–5% to –14%	–0.3% to –48.6%
InsertionSortOptimized	+23% to +41%	–147% to –1,800%
SelectionSortOptimized	+7% to +26%	+12% to +45%

Figure 10

Improvement Percentage formula

$$\text{Improvement Percentage} = \left(\frac{\text{Optimized Time} - \text{AI Time}}{\text{AI Time}} \right) \times 100$$