Project 1 - Empirical Analysis of Sorting Algorithms

Toghrul Tahirov

Bahçeşehir Cyprus University - Faculty of Architecture and Engineering Computer Engineering Department

CMP6001 – Advanced Algorithm Design Techniques

Dr. Professor

Jan 2, 2025

Abstract

This paper is meant to serve as an empirical report of the performance and time complexity analysis of five sorting algorithms, namely, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, and Quick Sort. The said metrics are investigated by measuring the system time, as converted from CPU ticks to nano seconds, taken during execution of the mentioned algorithms for arrays of size 10^2, 10^3, and 10^4 populated with randomly generated integers. Since this is an empirical analysis, we make a point of comparing the experimental findings to the expected theoretical shapes of the curve calculated using the theoretical asymptotic ($\mathscr{O}(n)$) notation. A brief discussion of factors that might have influenced the reported execution times such as the overhead from the programming language, which is Python in this case, array generation, memory usage, and implementation details is also included in this report. Ultimately, through the aforementioned analysis, we have confirmed the influence of mathematical knowledge in design of algorithms. The ($\mathscr{O}(n)$) notation is proven to provide a good perspective into an algorithm's efficiency in terms of execution time. Empricial data largely overlaps the theorertically calculated time complexity values and conforms to the same shape when the data acquired across a large range of different array sizes is plotted.

Project 1 - Empirical Analysis of Sorting Algorithms

Introduction

Sorting algorithms have historically been a fundamental topic in computer science, lying at the heart of all data processing tasks. Any system that processes large quantities of any data is prone to unintended time wasted during insertion, deletion, or update operations. Thus, importance of sorting such vast quantities of data based on a practical heuristic becomes a focal point in low level system design. While the theoretical efficiencies of these algorithms ($\mathcal{O}(n^2)$ for Selection, Bubble, and Insertion Sorts, and $\mathcal{O}(n \log n)$ for Merge Sort and Quick Sort) are often well-documented, it remains essential to verify these behaviors through empirical means.

It is beyond any doubt that an asymptotic time complexity calculation with the assumption of an infinitely large array will not serve as practical algorithm analysis tool. The reason for this statement is the fact, implementation of the algorithm, the specific behaviors of the programming language and the compilers involved, the overheads of operations such as function call, loops, and recursion, the cost of array manipulation in Python all have an impact on the actual execution time of an algorithm. This work implements each sorting method in Python, measures their execution times on arrays of increasing size, and compares the empirical measurements with theoretical expectations.

Materials and Methods

Sorting Algorithms

In this study, five sorting algorithms are investigated: *Selection Sort, Bubble Sort, Insertion Sort, Merge Sort,* and *Quick Sort*. Selection, Bubble, and Insertion Sorts are straightforward in logic and include comparisons/swaps for every element with every other element of the array, leading to worst-case and average-case time complexities being $\mathcal{O}(n^2)$. Merge Sort and Quick Sort, however, rely on the divide-and-conquer dynamic programming paradigms, and achieve an average and best-case time complexity of $\mathcal{O}(n \log n)$. One important point to note is that Quick Sort can theoretically degrade to $\mathcal{O}(n^2)$ in the worst case if the division pivots are poorly chosen.

Experimental Setup

As has been mentioned above, all algorithms were implemented in Python. The testing was done, on a per algorithm basis, with arrays of sizes 10^2, 10^3, and 10^4 that were populated with random integers in the range [1, 1000]. Each algorithm was tested multiple times (three runs per array size), and the mean execution time was recorded using `time.perf_counter_ns()` which records the system time taken in between operations by measuring the CPU ticks and converting into nanoseconds. Results are saved in a CSV file and plotted on both linear scale and on log-log scale.

Results and Discussion

Figure 1 (linear scale) and Figure 2 (log-log scale) show the empirical and theoretical curves for each sorting method studied in this project. For $n = 10^2$, there's practically minimal difference between the mentioned algorithms.

If we judge by Figure 1, even for an array of size $n = 10^3$, the differences are almost negligible. However, Figure 2 reveals that, on a log scale, as we move an order of magnitude higher in the array size, the time difference between the $\mathcal{O}(n^2)$ and $\mathcal{O}(n \log n)$ methods increases two fold. This is far beyond a considerable margin of error in time keeping. However, as n increases (particularly approaching 10^4), Merge Sort and Quick Sort generally outperform Selection Sort, Bubble Sort, and Insertion Sort, confirming their superior scaling behavior. Especially in a real life scenario where the elements of the array are much more likely to be complex data structures instead of atomic data types, this difference would be further amplified.

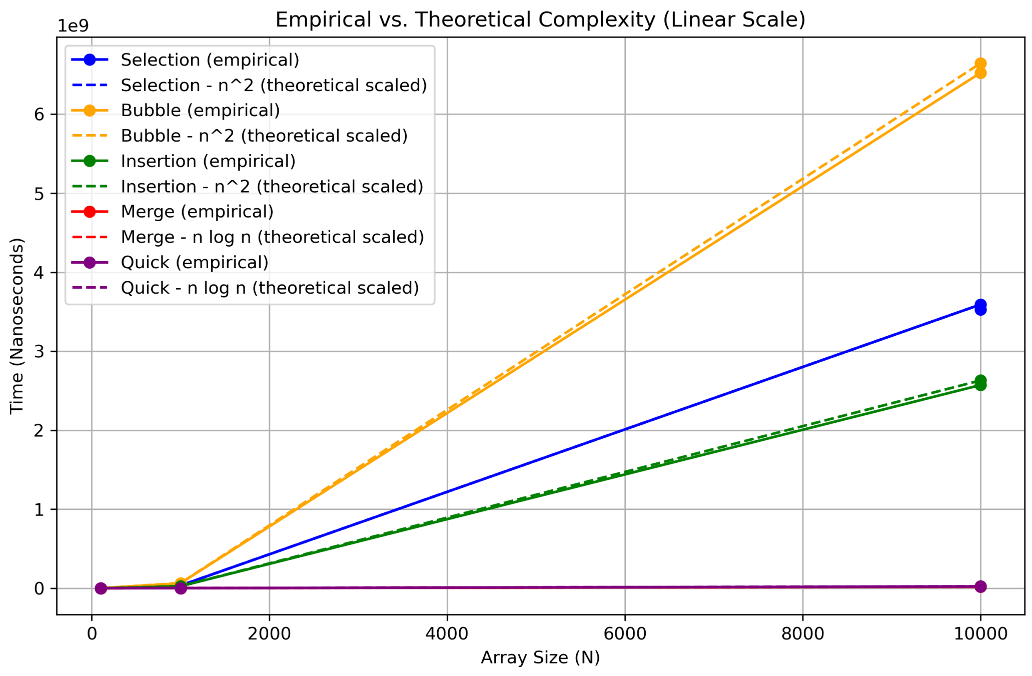

Figure 1

Array Size vs Time taken in Nanoseconds (linear scale)

Discussion of Findings Using Mathematical Analysis

The empirical analysis aligns closely with the theoretical complexities: the $\mathcal{O}(n^2)$ algorithms experience a more pronounced increase in execution time, whereas the $\mathcal{O}(n \log n)$ algorithms exhibit better scalability, showing slower growth in their running times. Quick Sort can degrade to worst-case $\mathcal{O}(n^2)$ in pathological cases. Additionally, implementation details such as recursive function overhead for Merge Sort or pivot selection strategy for Quick Sort can cause minor deviations from pure theoretical expectations, but the log-log plots confirm these algorithms' asymptotic relationships.

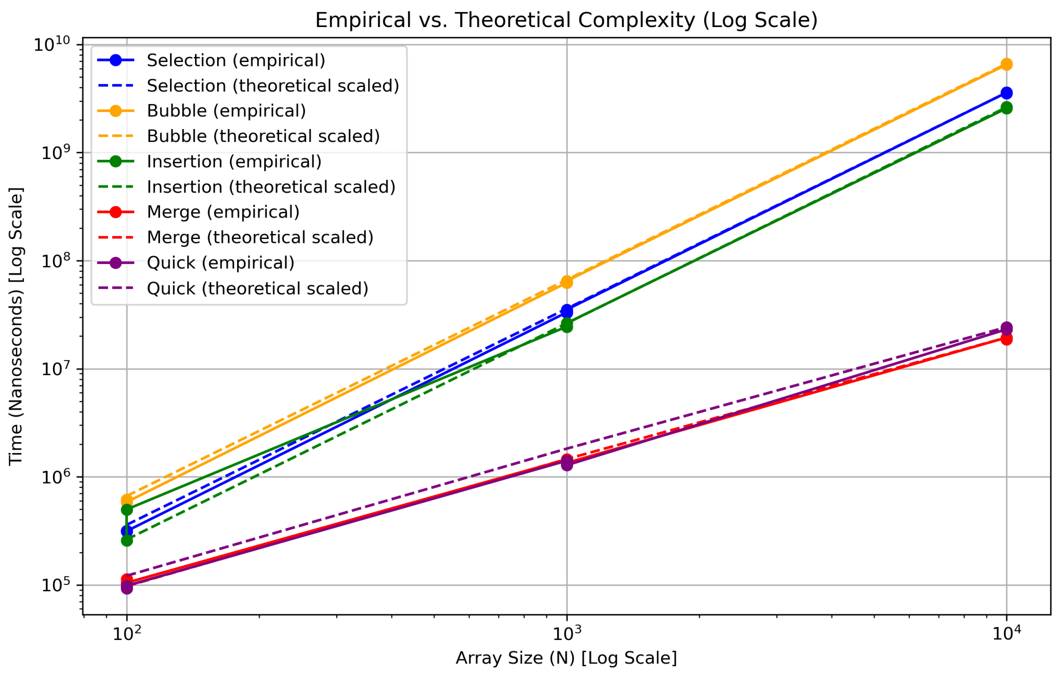

Figure 2

Array Size vs Time taken in Nanoseconds (log-log scale)

Conclusion

This study has showcased the need to support theoretical time-complexity claims made based on a purely mathematical analysis of an algorithm with empirical testing. While Merge Sort and Quick Sort tend to excel for large datasets, simpler algorithms like Insertion Sort can be more practical for smaller arrays or where low constant factors and fewer function/recursive calls are of essence. Overall, integration of mathematical analysis with empirical measurement allows development of a well-rounded perspective on algorithm performance in real-world scenarios.