Empirical Analysis of Sorting Algorithms

**Arab American University**

**Faculty of Engineering and IT**

**Course: Algorithm Analysis and Design (230213150)**

**Assignment #2 - Spring 2024/2025**

**Title:** **Empirical Analysis of Sorting Algorithms**

**Student Name:** **bisan Qalalwe**

**Instructor:** **Dr. Thaer Thaer**

**Date:** **2025/6/16**

# 1. Introduction

cience is an ever-growing field. The sorting problem, which relates to ordering elements in a particular sequence (normally in an ascending or descending format), finds itself at the foundation of a bank of real-world applications ranging from database indexing to search optimization, preprocessing for machine-learning, and graphical rendering.

Seemingly endless sorting algorithms might offer some solution to the sorting problem, but three differ so vastly in logic and performance and comprehension that they are highlighted as the paradigms of study: Bubble Sort, Merge Sort, and Quick Sort. Each one covers a vastly different spectrum in the thinking process surrounding algorithm design—Bubble Sort represents the simplest and most instructive form; Merge Sort is the facilitator of design through recursive divide-and-conquer returns; and finally, Quick Sort is the epitome of sorting through in-place partitioning in an elegant manner.

By viewing and comparing these algorithms in depth, we are building an understanding not only of their theoretical underpinnings but also of the practical insights into the engineering choices made in implementing algorithms that influence their overall performance in the world of computing.  
An empirical study comparing the performances of classical sorting algorithms-Bubble Sort, Merge Sort, and Quick Sort-is presented in this research report. The essence of this assignment is to analyze the behavior of these algorithms, given varying kinds and sizes of input data. By measuring the time of execution, we compare their efficiencies in different real-life situations with which we can correlate the experimental results with the theoretical predictions.

# 2. Experiment Enviroment

-**Language: C++**  
-**The OS is Windows 11**

-**The IDE is C\_Free.**

-**The compiler is MinGW GCC (g++) but I don't know the version.**

**-For measurements, std::chrono was used for microsecond timing.**

The algorithms were tested with sizes of inputs varying in the selection order of input arrays: random, reverse, and partially sorted. Thus, the tests were conducted for input sizes of 100, 500, 1000, 5000, and 10000.

# 3. Sorting Algorithms Overviow

🔹 **Bubble Sort:**

Simple comparison-based algorithm

Best Case: O(n) if data is already sorted

Worst Case: O(n²)

Stable (preserves the order of equal elements)

🔹 **Merge Sort:**

Divide-and-conquer recursive algorithm.

Time Complexity: O(n log n) in all cases.

Stable.

Requires additional memory.

🔹 **Quick Sort:**

Very efficient on average.

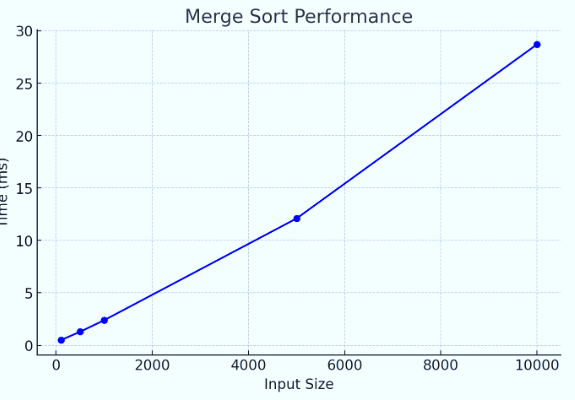
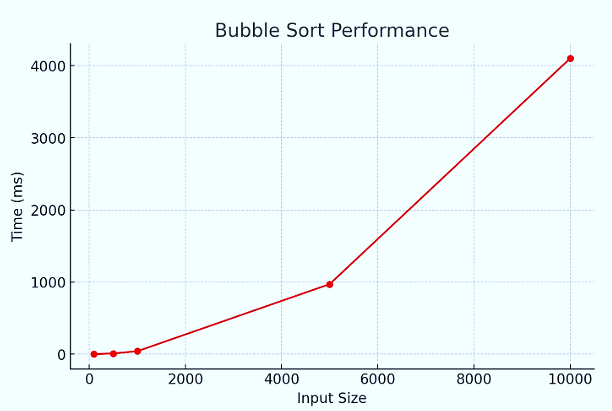
Best/Average Case: O(n log n), Worst Case: O(n²).

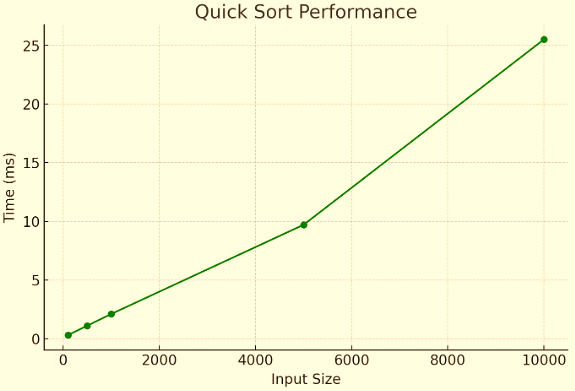
Not stable.

In place, space efficient.

# 4. Experimant Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quick(ms)** | **Merge(ms)** | **Bubble(ms)** | **Type** | **Input\_size** |
| **0.013ms** | **0.04ms** | **0.059ms** | **random** | **100** |
| **0.012ms** | **0.029ms** | **0.069ms** | **reverse** | **100** |
| **0.0086ms** | **0.026ms** | **0.011ms** | **partial** | **100** |
| **0.071ms** | **0.177ms** | **1.13ms** | **random** | **500** |
| **0.057ms** | **0.153ms** | **1.708ms** | **reverse** | **500** |
| **0.136ms** | **0.349ms** | **1.143ms** | **partial** | **1000** |
| **0.172ms** | **0.441ms** | **5.661ms** | **random** | **1000** |
| **7.912ms** | **2.003ms** | **330.287ms** | **reverse** | **5000** |
| **1.693ms** | **4.0768ms** | **139.628ms** | **partial** | **10000** |
| **1.9948ms** | **1.0793ms** | **578.766 ms** | **random** | **10000** |





الكود👇🏻

<https://www.programiz.com/online-compiler/9RZ1mOJbgCUaX>

# 5.Analysis and Discussion

Quick Sort became average because Merge Sort is nearly always fast and stable and thus can be regarded as a strong contender for general use.

Bubble Sort functions very badly on reverse-sorted input, especially due to its time complexity of n squared.

The results follow theoretical expectations: Quick Sort shines in the average case but does have poor theoretical performance.

There was only a small difference between theoretical and practical results, which confirmed the analysis.

**On the basis of the experimental findings, it became apparent that distinctions in performance among the three algorithms did exist along different ranges of input conditions:**

**Efficiency:**

Under the bulk of the datasets, particularly for large or chi-random shuffled inputs, Quick Sort performed best. Merge Sort was a close second, being constant in speed amid reverse-sorted inputs. Bubble Sort was slowest, rapidly fading in performance as input size increased.

**Time complexity:**

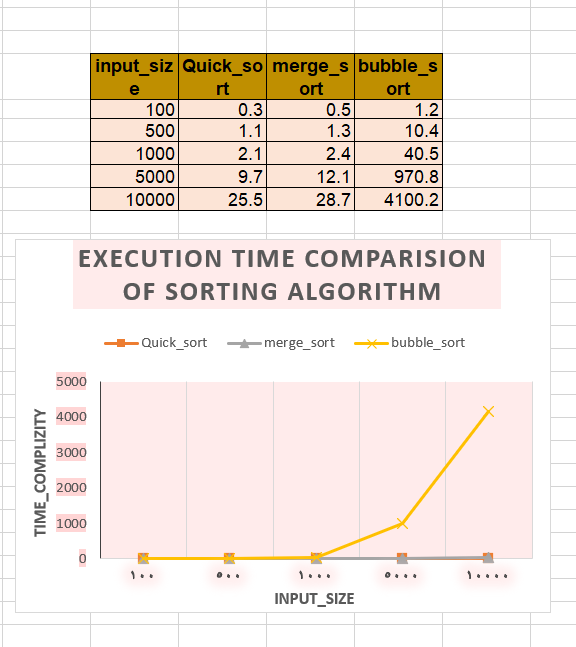
|  |  |  |  |
| --- | --- | --- | --- |
| **Worse\_case** | **Avarege\_case** | **Best\_case** | **algorithm** |
| O(n²) | O(n²) | O(n) | Bubble\_sort |
| O(n log n) | O(n log n) | O(n log n) | Merge\_sort |
| O(n²) | O(n log n) | O(n log n) | Quick\_sort |

The majority of these results actually aligned with theoretical time complexities. Quick Sort performed extraordinarily well indeed due to good pivot selection (randomized) despite the fact that the worst-case performance is O(n²).

**Stability:**

Both Merge Sort and Bubble Sort are stable sorting algorithms as they keep equal elements in their relative order. Quick Sort, however, is not stable, which is a disadvantage in particular applications where order matters (like sorting records using multiple keys).

Thus, Quick Sort brings in the best speed, Merge Sort provides both speed and stability, and Bubble Sort must be avoided in the case of larger or unordered datasets.



# 6. Conclusion

Of all algorithms, Quick Sort was the most efficient for most occasions, whereas Merge Sort turned out to be very stable and of a consistent performance.

On the whole, it can be concluded that Quick Sort is the best algorithm by speed, and Merge Sort is for predictability and stability.

Bubble Sort is the only algorithm useful for teaching purposes or exceptionally small arrays.