Empirical Analysis of Sorting Algorithms

# 1. Introduction

The aim of this experiment is to compare the performance of three popular sorting algorithms: Bubble Sort, Merge Sort, and Quick Sort. To determine which algorithm performs best in different scenarios and to comprehend the practical implications of their theoretical complexities, we evaluate their execution time on datasets of varying sizes and orders.

# 2. Methodology and Environment

High-resolution clock functions in C++ were used to measure the time required by each sorting algorithm.

Three different data types—random, sorted, and reverse sorted—as well as dataset sizes of 10, 1000, and 10,000 were used to test each algorithm.

• Programming Language : C++

• Time Measurement Library : <chrono>

• Execution Method :

auto start = high\_resolution\_clock::now();

// sorting...

auto end = high\_resolution\_clock::now();

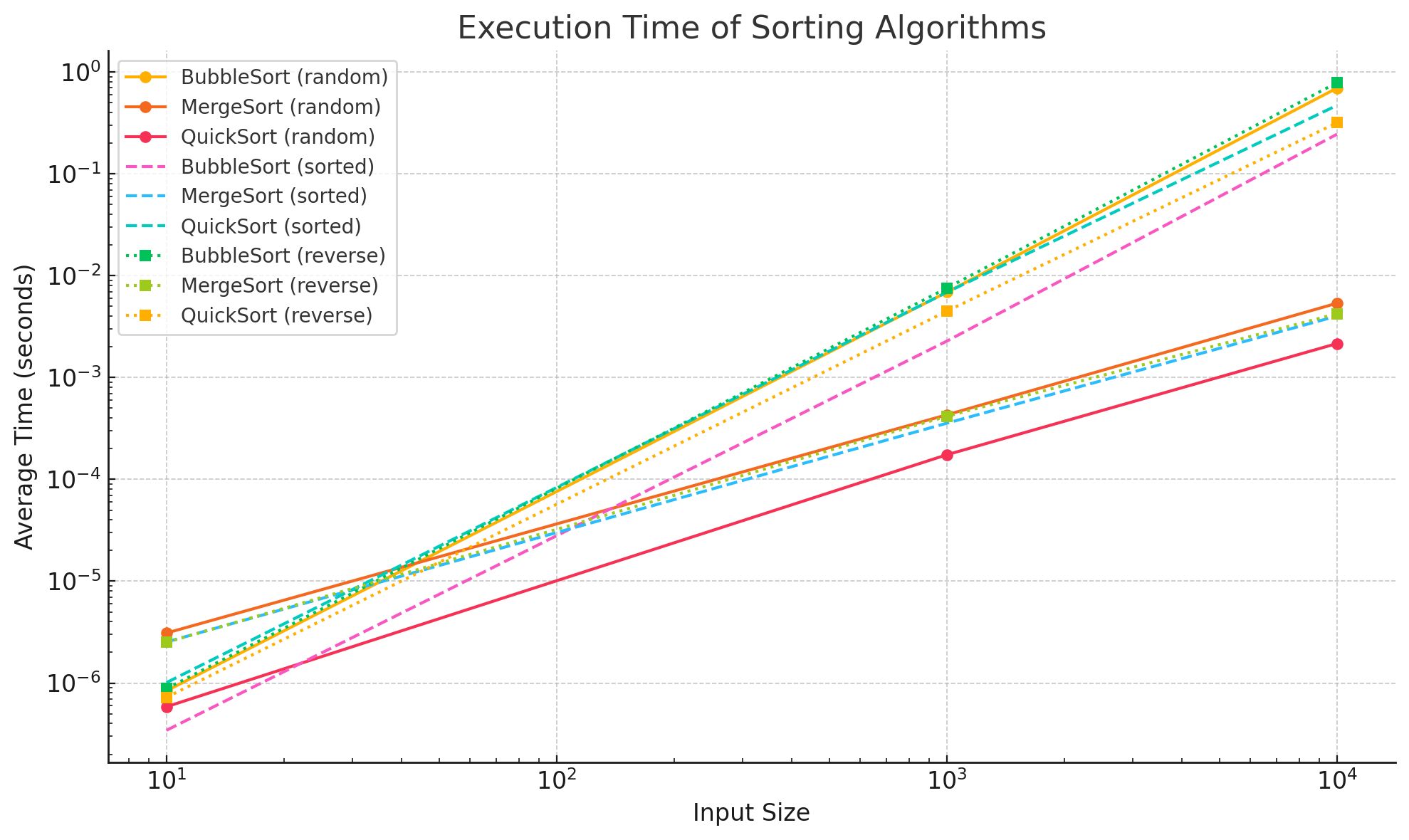
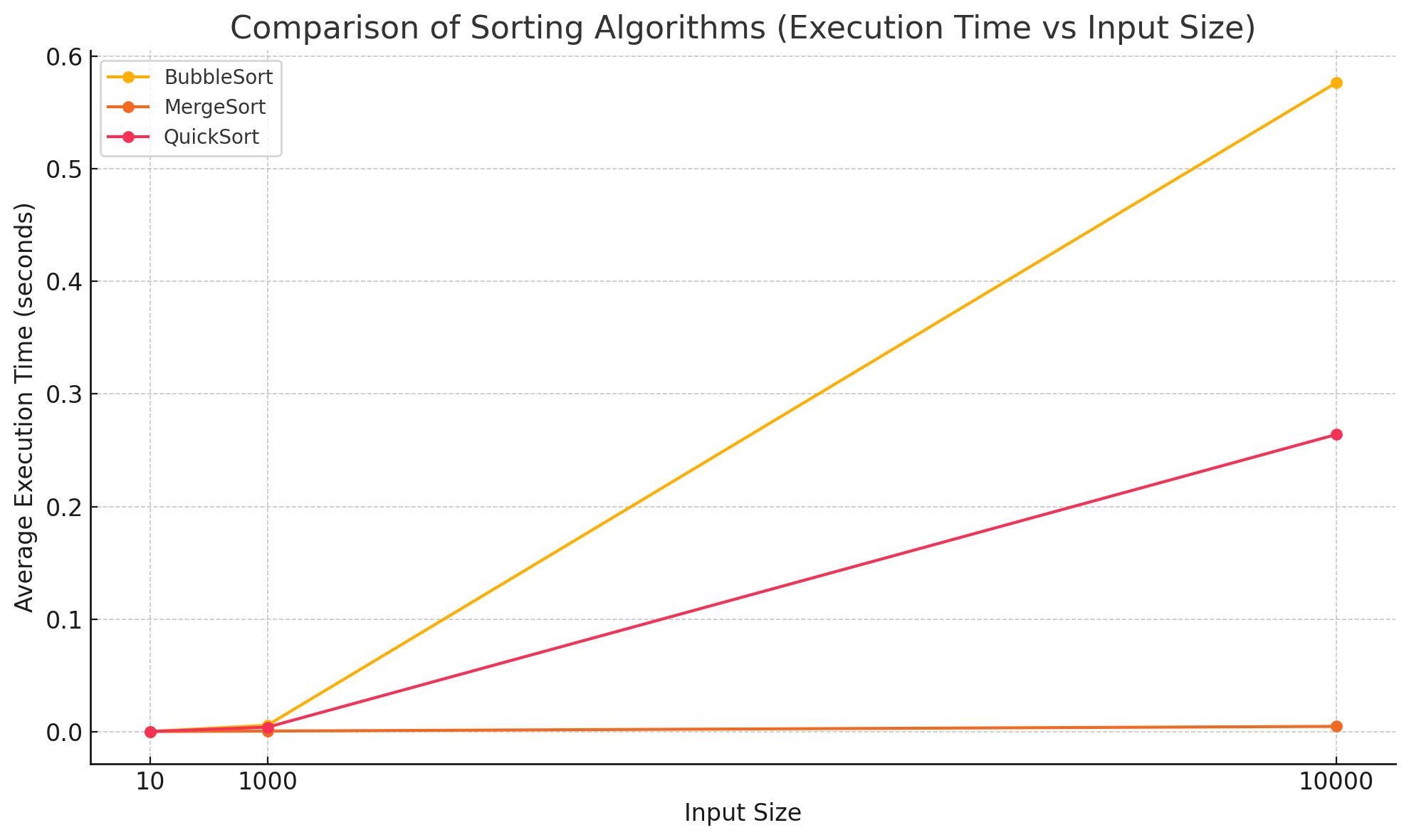
duration<double> duration = end - start;

• operating system : Windows 10 Pro

# 3. Results

The average execution times (in seconds) for each algorithm are summarized in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Input Size | Type | Average Time (s) |
| BubbleSort | 10 | random | 8.38333e-07 |
| MergeSort | 10 | random | 3.09967e-06 |
| QuickSort | 10 | random | 5.85333e-07 |
| BubbleSort | 10 | sorted | 3.43667e-07 |
| MergeSort | 10 | sorted | 2.53133e-06 |
| QuickSort | 10 | sorted | 1.01067e-06 |
| BubbleSort | 10 | reverse | 8.82333e-07 |
| MergeSort | 10 | reverse | 2.51033e-06 |
| QuickSort | 10 | reverse | 7.24333e-07 |
| BubbleSort | 1000 | random | 0.00689557 |
| MergeSort | 1000 | random | 0.000430956 |
| QuickSort | 1000 | random | 0.00017465 |
| BubbleSort | 1000 | sorted | 0.00228054 |
| MergeSort | 1000 | sorted | 0.000356647 |
| QuickSort | 1000 | sorted | 0.00688135 |
| BubbleSort | 1000 | reverse | 0.00757002 |
| MergeSort | 1000 | reverse | 0.000416656 |
| QuickSort | 1000 | reverse | 0.00448687 |
| BubbleSort | 10000 | random | 0.696608 |
| MergeSort | 10000 | random | 0.00537519 |
| QuickSort | 10000 | random | 0.00215895 |
| BubbleSort | 10000 | sorted | 0.245528 |
| MergeSort | 10000 | sorted | 0.00402343 |
| QuickSort | 10000 | sorted | 0.471403 |
| BubbleSort | 10000 | reverse | 0.787775 |
| MergeSort | 10000 | reverse | 0.00424259 |
| QuickSort | 10000 | reverse | 0.318238 |

The following chart visualizes the performance of each algorithm on random data: 

# 4. Analysis and Discussion

* Bubble sort is a simple sorting algorithm, but it doesn't work well in practical situations, especially when large datasets are involved. When the array is already sorted, it has an O(n) best case, but an O(n²) average and worst case, making it inefficient.
* Merge Sort guarantees O(n log n) performance regardless of the input and is well known for its stability.  
  It is efficient and widely used in scenarios where stable sorting is needed.
* Quick Sort, despite having a worst case of O(n²), is often the fastest in practice due to its in-place sorting and good cache

performance. Its average and best case are O(n log n), making it highly efficient for large datasets.

Time Complexity and Performance Comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Best Case (Big O) | Average Case (Big O) | Worst Case (Big O) | Practical Performance |
| Bubble Sort | O(n) | O(n²) | O(n²) | Very slow |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) | Stable and fast |
| Quick Sort | O(n log n) | O(n log n) | O(n²) | Usually the fastest |

# 5.conclusion

In conclusion, Quick Sort usually provides the best performance in most real-world scenarios, making it the ideal choice for large, randomly ordered datasets. Merge Sort performs better and is appropriate in scenarios requiring a constant execution time. Bubble Sort's poor scalability makes it unsuitable for large datasets . Merge Sort is more stable in terms of performance and suitable for situations where consistent execution time is required. Bubble Sort should be avoided for large datasets due to its poor scalability.

Dr : thaer thaher

Student id : 202210677

Student name : kinda massad

" Wish me luck "