Assignment # 4

Sorting Algorithms Analysis

Abubakar Ahmad

SAP ID: 54603

Class Instructor

Mr. Usman Sharif

Riphah International University Islamabad
Pakistan

Empirical Analysis of Sorting Algorithms

1. Introduction

Sorting algorithms are crucial in computer science for organizing data in a specified order, which enables efficient searching, data retrieval, and processing. The performance of sorting algorithms is generally evaluated through theoretical time complexities. However, empirical analysis, involving the actual measurement of execution time on real hardware, provides valuable insights into practical performance.

This assignment aims to analyze and compare the empirical execution times of **Bubble Sort**, **Selection Sort**, **Insertion Sort**, and **Merge Sort** for different input sizes. The purpose is to verify whether empirical results align with theoretical time complexities and to observe algorithm behavior in practice.

2. Methodology

2.1 Implementation

The following sorting algorithms were implemented in **C++**:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort

Four different array sizes were used as inputs:

- **Arr1:** 5 elements
- **Arr2:** 10 elements
- **Arr3:** 50 elements
- **Arr4:** 100 elements

Each array contained already sorted integer sequences from 1 to N.

2.2 Measuring Execution Time

- Execution time was measured using **std::chrono::high_resolution_clock** from the C++ <chrono> library for high-precision timing.
- For each array size and sorting algorithm:
  - The sorting function was executed **five times**
  - The execution time for each run was recorded in **milliseconds (ms)**
  - The **average execution time** over five runs was computed to ensure result reliability.
- Fresh copies of the original arrays were used for each run to maintain consistency.

3. Results & Graph

3.1 Recorded Average Execution Times

Input Size (N)	Bubble Sort (ms)	Selection Sort (ms)	Insertion Sort (ms)	Merge Sort (ms)
5	0.0001	0.0002	0.0001	0.0003
10	0.0002	0.0004	0.0002	0.0005
50	0.0045	0.0041	0.0004	0.0036
100	0.0130	0.0147	0.0007	0.0075

3.2 Graph: Execution Time vs. Input Size

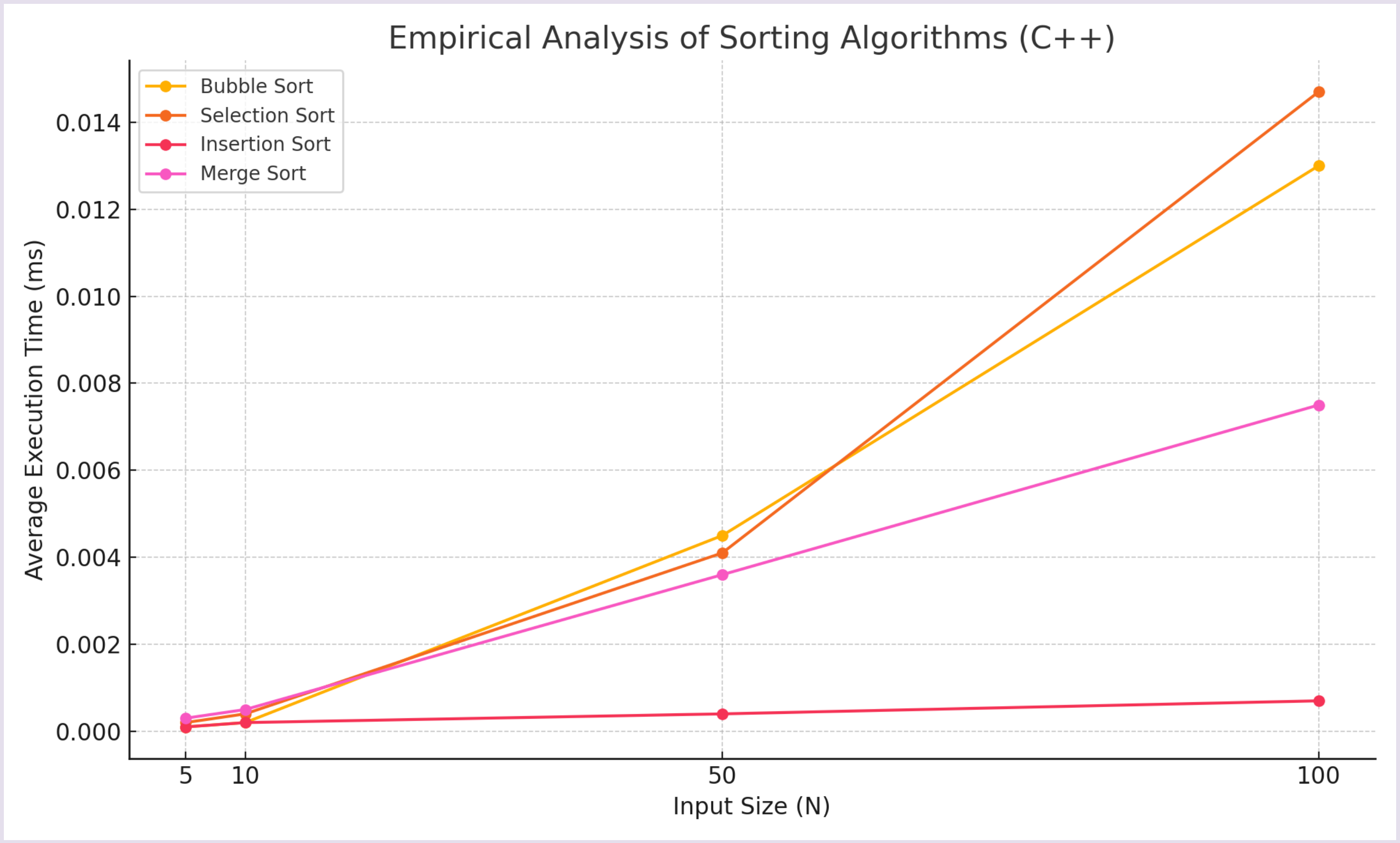

Figure 1: Comparison of average execution time for sorting algorithms over increasing input sizes (in ms)

4. Analysis

4.1 Growth Trend vs. Theoretical Expectations

The empirical results generally match theoretical time complexities:

- **Bubble Sort & Selection Sort**
  Both exhibit O(N2) O(N^2) O(N2) time complexities, with execution times increasing sharply as input size grows. This is visible in the 50 and 100 element input cases.

- **Insertion Sort**
  Although Insertion Sort has a worst-case time complexity of O(N^2), it performed exceptionally well on the already sorted input, reflecting its **best-case behavior of O(N)O(N)O(N)**. Its execution times remained nearly constant across input sizes, growing much slower than the other quadratic algorithms.
- **Merge Sort**
  Merge Sort demonstrated superior performance on larger input sizes due to its O(NlogN) time complexity. The growth in execution time was significantly less steep compared to Bubble and Selection Sort, confirming its efficiency for larger datasets.

4.2 Anomalies or Deviations

- **Insertion Sort** performed much better than expected for larger inputs because the arrays were already sorted, representing its best-case scenario.
- No unexpected anomalies were observed in the empirical data; trends closely followed theoretical expectations.

5. Conclusion

The empirical analysis confirmed the theoretical time complexities of sorting algorithms:

- **Merge Sort** consistently outperformed the other algorithms on larger input sizes.
- **Insertion Sort** demonstrated excellent performance on sorted data due to its linear best-case behavior.
- **Bubble Sort and Selection Sort** showed poor scalability, with execution times increasing rapidly as input size grew.

This analysis highlights the importance of selecting appropriate sorting algorithms based on input characteristics and dataset size for optimal performance in practical applications.

6. GitHub Repository Link

Repository: https://github.com/abuba-akar0/AnalysisOfAlgorithm.git

This public repository contains:

- Complete C++ source code for all sorting algorithms
- Program output screenshots
- Graph image file
- README with execution instructions
- PDF report (if required)