Ionian University

Comparative Analysis of Sorting Algorithms: Part 1

Department of Informatics

Laboratory of Algorithms and Data Structures

April 2, 2025

Names	Student ID
Nikolaos Roufas	inf2024146
Konstantinos Tzortis	inf2024168
Nikolaos Levadiotis	inf2024168

Contents

1 Introduction

In this study, we present a comparative analysis of various sorting algorithms. The studied algorithms are:

- Bubble Sort
- Quick Sort
- Merge Sort
- Selection Sort
- Insertion Sort
- Shell Sort
- Heap Sort

The analysis includes executing the algorithms on integer arrays of different sizes and recording execution times. The goal is to compare the efficiency of the algorithms and derive conclusions about their behavior in different scenarios.

2 Methodology

2.1 Algorithm Implementation

The sorting algorithms were implemented in Python and executed in different scenarios with the following characteristics:

- Array sizes: 100, 200, 300, ..., 1000 integers
- Number of executions: 5 repetitions for each algorithm and each array size
- Element values: Random integers in the range [0, 1000000]

2.2 Data Collection

For each algorithm and array size, the average execution time from the 5 repetitions was recorded. The results were stored in a text file, and graphs were generated for visualization.

3 Results and Analysis

3.1 Comparative Charts

Below are the charts comparing the execution times of different sorting algorithms:

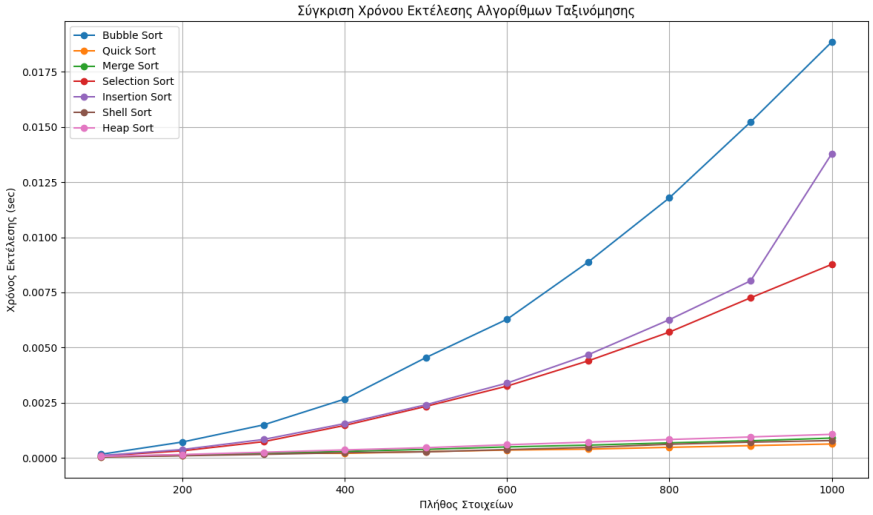

Figure 1: Comparison of all sorting algorithms

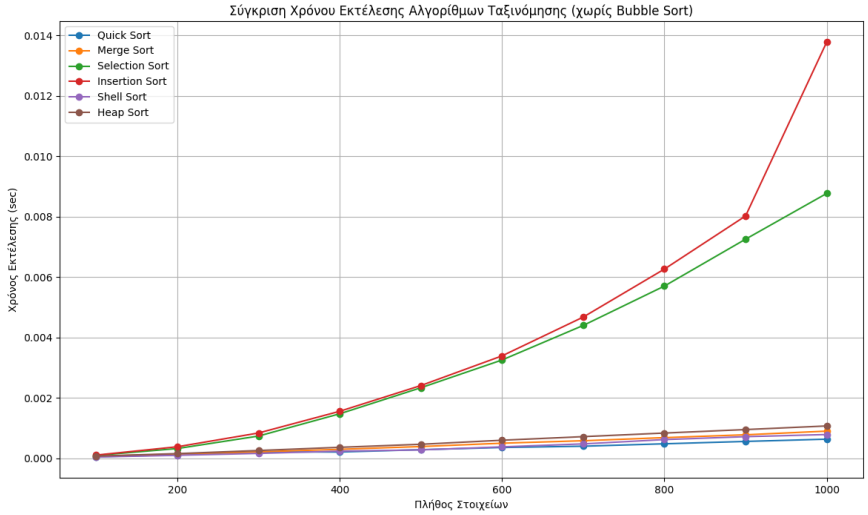

Figure 2: Comparison of sorting algorithms (excluding Bubble Sort)

4 Comparative Analysis and Conclusions

4.1 Algorithm Comparison

Based on performance, sorting algorithms can be categorized as follows:

- **$O(n^2)$ Algorithms**: Bubble Sort, Selection Sort, Insertion Sort
  - Suitable only for small datasets
  - Bubble Sort is the least efficient
  - Insertion Sort performs well for nearly sorted arrays
- **Intermediate Performance Algorithm**: Shell Sort
  - Significant improvement over $O(n^2)$ algorithms
  - Performance depends on the chosen gap sequence
- **O(n log n) Algorithms**: Quick Sort, Merge Sort, Heap Sort
  - The most efficient for large datasets
  - Quick Sort is the fastest in our tests
  - Merge Sort is stable and predictable
  - Heap Sort does not require additional memory

4.2 Conclusions

From the results, we derive the following conclusions:

1. The theoretical complexity of the algorithms reflects in their measured execution times, with O(n log n) algorithms outperforming $O(n^2)$ algorithms.

2. Quick Sort is the most efficient in our tests, but its performance can degrade in special cases (e.g., already sorted arrays).

3. Merge Sort, though slightly slower than Quick Sort, offers stable performance and is suitable for applications requiring predictable behavior.

4. Simple algorithms (Bubble, Selection, Insertion) are unsuitable for large datasets but may be useful in specific cases (e.g., Insertion Sort for nearly sorted arrays).

5. Shell Sort is a good alternative when the highest efficiency of O(n log n) algorithms is not necessary but improvement over $O(n^2)$ algorithms is desired.