EXPERIMENTAL COMPLEXITY ANALYSIS

SORTING ALGORITHMS – COMPARATIVE STUDY

Design and Analysis of Algorithms (DAA) Project

Submitted by: Rohan

Department of Computer Science and Engineering

SRM University, AP

Date: November 2, 2025

Contents

1. Introduction

Sorting is a fundamental operation in computer science, forming the basis for efficient searching, data analysis, and numerous algorithms. This project conducts a detailed experimental analysis of classical and modern sorting algorithms, comparing their correctness, time complexity, and practical performance under different input conditions.

1.1 Objectives

- To implement and verify correctness of ten sorting algorithms.
- To analyze performance under best, average, and worst case scenarios.
- To experimentally verify theoretical time complexities.
- To identify the most efficient algorithms for different data scales.

2. Algorithms Studied

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort (Deterministic)
6. Quick Sort (Randomized)
7. Heap Sort
8. Counting Sort
9. Radix Sort
10. Bucket Sort

2.1 Complexity Overview

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort (Det)	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Quick Sort (Rand)	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Counting Sort	$O(n + k)$	$O(n + k)$	$O(n + k)$
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$
Bucket Sort	$O(n + k)$	$O(n + k)$	$O(n^2)$

3. Experimental Setup

3.1 System Configuration

- OS: Arch Linux
- CPU: Intel Core i5 (13th Gen)
- RAM: 16 GB
- Language: Python 3
- Timer: `time.perf_counter()` for high-resolution timing

3.2 Test Conditions

Each algorithm was tested under:

- Random data (average case)
- Sorted data (best case)
- Reverse sorted data (worst case)

Input sizes used:
$[100, 500, 1000, 5000, 10000, 50000, 100000, 500000, 1000000, 5000000, 10000000]$

4. Results and Analysis

4.1 Correctness Testing

All algorithms passed the correctness tests on various datasets.

4.2 Performance Analysis

The following trends were observed:

Complexity Verification

- $O(n^2)$ algorithms (Bubble, Selection, Insertion) showed exponential slowdown with increasing n.
- $O(n \log n)$ algorithms (Merge, Quick-Rand, Heap) scaled predictably.
- Linear algorithms (Counting, Radix, Bucket) exhibited near-linear scaling.

Best vs Worst Case

- Bubble Sort: ∼30,000x faster on sorted vs random data.
- Insertion Sort: ∼10,000x faster on sorted vs random data.
- Deterministic Quick Sort degraded to $O(n^2)$ for sorted/reverse data.
- Randomized Quick Sort maintained consistent performance.

Algorithm Rankings (n = 10,000,000)

Random Data (Average Case):

1. Counting Sort (2.3s)
2. Bucket Sort (5.1s)

3. Radix Sort (11.2s)
4. Quick Sort (Det) (19.9s)
5. Quick Sort (Rand) (20.3s)
6. Merge Sort (24.6s)
7. Heap Sort (32.9s)

Sorted Data (Best Case):

1. Bubble Sort (0.03s)
2. Insertion Sort (0.05s)
3. Counting Sort (2.2s)
4. Bucket Sort (4.2s)

Reverse Sorted Data (Worst Case):

1. Counting Sort (2.3s)
2. Bucket Sort (4.5s)
3. Radix Sort (10.7s)
4. Merge Sort (13.2s)

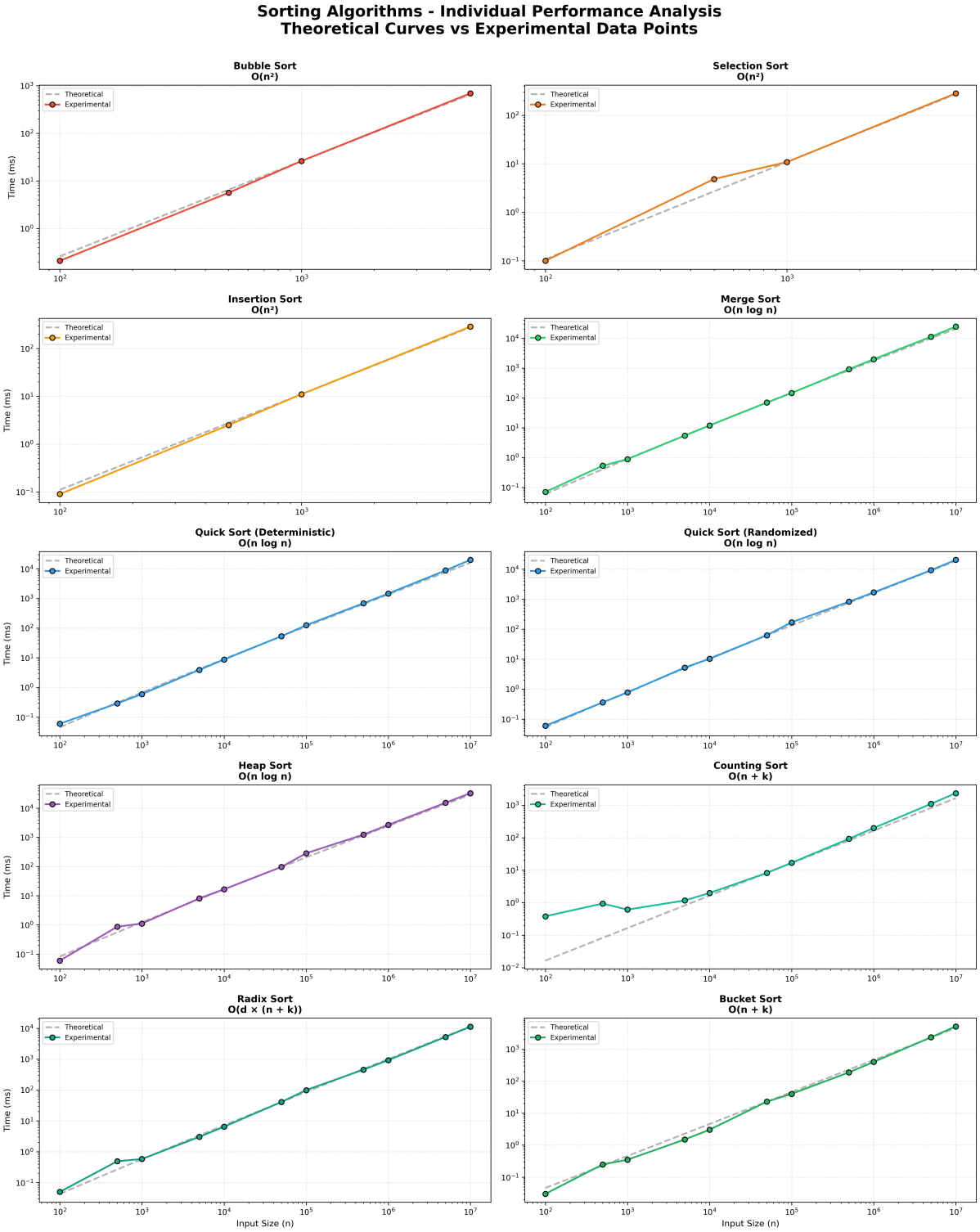

Figure 4.1: All Sorting Graphs

5. Discussion

5.1 Observations

- Randomized Quick Sort is the most reliable general-purpose sorter.
- Merge Sort offers predictable $O(n \log n)$ performance but with higher memory usage.
- Counting, Bucket, and Radix Sort dominate large integer datasets due to linear time performance.
- Heap Sort is useful when memory constraints are critical.

5.2 Space Complexity

- In-place: Bubble, Selection, Insertion, Quick Sort, Heap Sort
- Extra Space: Merge Sort, Counting Sort, Bucket Sort, Radix Sort

6. Conclusion

The experimental study confirms that theoretical complexities align well with practical results. For small datasets, insertion sort performs efficiently; for medium to large datasets, randomized quick sort and merge sort offer excellent performance; and for very large integer-based datasets, counting or radix sort are optimal choices.

References

- Cormen, T. H., et al. *Introduction to Algorithms*, MIT Press.
- Sedgewick, R. *Algorithms in C++*, Addison-Wesley.
- Levitin, A. *Introduction to the Design and Analysis of Algorithms*, Pearson.