Comparative Analysis of Sorting Algorithms: Performance Evaluation on Synthetic and Real-World Financial Data

Gong Da
Department of Mathematics and Information Technology
the Education University of Hong Kong
Hong Kong S.A.R.
s1153651@s.eduhk.hk

***Abstract*—Sorting algorithms are fundamental to computer science, with varying time complexities affecting their practical performance across different data characteristics. This paper presents a comprehensive empirical analysis of three classical sorting algorithms: Bubble Sort, Quick Sort, and Bucket Sort. The author evaluates these algorithms on both synthetic random datasets and real-world NVIDIA (NVDA) stock market data spanning over two decades. The experimental results demonstrate that while Bubble Sort exhibits predictable $O(n^2)$ behavior making it impractical for large datasets, the choice between Quick Sort $O(n \log n)$ and Bucket Sort $O(n + k)$ critically depends on the data value range. The author provides regression analysis and empirical formulas for runtime prediction, showing that Bucket Sort excels when the value range k is bounded and comparable to dataset size n, while Quick Sort maintains superior performance on wide-range data typical of financial applications. These findings are consistent with previous empirical studies [7] and extend the analysis to real-world financial data with unique characteristics. The results provide practical guidance for algorithm selection in real-world sorting tasks, with particular emphasis on the relationship between value range and dataset size.**

***Index Terms*—sorting algorithms, complexity analysis, empirical evaluation, financial data, performance benchmarking**

I. Introduction

Sorting is one of the most fundamental operations in computer science, with applications spanning database systems, data analytics, search algorithms, and scientific computing [1], [2]. The theoretical time complexity of sorting algorithms has been extensively studied, yet practical performance characteristics often depend heavily on data distribution, value ranges, and implementation details.

This paper investigates three representative sorting algorithms from different complexity classes: Bubble Sort ($O(n^2)$), Quick Sort ($O(n \log n)$ average case), and Bucket Sort ($O(n+k)$ where k is the value range). While asymptotic complexity provides theoretical guidance, the empirical analysis reveals critical insights into when each algorithm performs optimally in practice.

A. Motivation

The motivation for this study stems from two key observations:

1) Despite well-established theoretical complexity bounds, practitioners often lack empirical guidance on algorithm selection for specific data characteristics.
2) Financial and time-series data exhibit unique properties (wide value ranges, temporal patterns) that may challenge conventional wisdom about sorting algorithm performance.

Previous empirical studies [7] have compared sorting algorithms but primarily focused on synthetic data or specific application domains. Our work extends these studies by analyzing performance on financial time-series data with inherently wide value ranges.

B. Contributions

The main contributions are:

- Empirical performance analysis on 200 datasets ranging from 100 to 10,000 elements
- Regression-based runtime prediction formulas fitted to experimental data
- Comparative evaluation using real-world NVIDIA stock market data (6,743 trading days)
- Practical guidelines for algorithm selection based on data range characteristics
- Extension of previous empirical studies [7] to real-world financial data

II. Background and Related Work

A. Sorting Algorithm Complexity

Sorting algorithms can be categorized by their time complexity [1]:

Quadratic Time: Bubble Sort represents the class of simple comparison-based algorithms with $O(n^2)$ worst and average-case complexity. While easy to implement, these algorithms become impractical for large datasets.

Linearithmic Time: Quick Sort, invented by Hoare [3], achieves $O(n \log n)$ average-case performance through divide-and-conquer. It remains one of the fastest general-purpose sorting algorithms in practice.

Linear Time: Bucket Sort [2] achieves $O(n + k)$ time when the value range k is known and bounded. However, performance degrades when $k \gg n$.

B. Empirical Studies

Previous empirical evaluations [4], [5] have focused primarily on synthetic data or specific application domains. More recent studies [7] have compared various sorting algorithms and found that optimized Quick Sort is the best overall performer for most datasets. Our work extends these studies by analyzing performance on financial time-series data with inherently wide value ranges.

III. Methodology

A. Experimental Setup

All experiments were conducted on an Apple M-series processor with Python 3.13. The author implemented three sorting algorithms following standard pseudocode [2]:

Bubble Sort: Repeatedly compares adjacent elements and swaps if out of order, with $O(n^2)$ comparisons.

Quick Sort: Partitions array around pivot, recursively sorting sub-arrays, with $O(n \log n)$ average complexity.

Bucket Sort: Distributes elements into buckets based on value range, sorting each bucket individually, with $O(n + k)$ time where k is the number of buckets.

B. Dataset Generation

1) Synthetic Random Data: The author generated 200 datasets with sizes ranging from 100 to 10,000 elements (step size 50), containing random integers uniformly distributed in range [0, 1000]. This configuration yields $k = 1001$ buckets for Bucket Sort.

2) NVIDIA Stock Market Data: The author extracted historical NVIDIA (NVDA) stock data from Yahoo Finance covering maximum available history (approximately 1999-2025, 6,743 trading days). Datasets were constructed from:

- Closing prices scaled by $\times 100$ (range: 10 to 14,000)
- Trading volumes scaled by $\div 10^6$ (range: 0 to 60,000)

This yields a combined value range $k \approx 60,000$, representing wide-range real-world data.

C. Performance Metrics

For each algorithm and dataset, the author measured wall-clock execution time using Python's `perf_counter`. Regression analysis was performed to fit theoretical complexity models:

$$T_{\text{bubble}}(n) = a \cdot n^2 + b \tag{1}$$

$$T_{\text{quick}}(n) = a \cdot n \log(n) + b \tag{2}$$

$$T_{\text{bucket}}(n) = a \cdot n + b \tag{3}$$

where a and b are empirically determined constants.

TABLE I
Performance Statistics on Synthetic Random Data

Algorithm	Mean (s)	Median (s)	Max (s)	Complexity
Bubble Sort	0.1037	0.0846	0.3073	$O(n^2)$
Quick Sort	0.0010	0.0010	0.0023	$O(n \log n)$
Bucket Sort	0.0003	0.0002	0.0019	$O(n + k)$

IV. Experimental Results

A. Individual Algorithm Performance

Figure 1 presents the runtime performance of each algorithm on synthetic random data.

1) Bubble Sort: The experiments confirm the expected $O(n^2)$ behavior. The fitted regression formula:

$$T_{\text{bubble}}(n) = 1.91 \times 10^{-8} \cdot n^2 + 1.90 \times 10^{-3} \tag{4}$$

Performance statistics (Table I) show mean runtime of 0.104s with maximum 0.307s at $n = 10,000$. The quadratic growth makes Bubble Sort impractical beyond small datasets.

2) Quick Sort: Quick Sort demonstrates efficient $O(n \log n)$ scaling:

$$T_{\text{quick}}(n) = 6.73 \times 10^{-8} \cdot n \log(n) - 2.70 \times 10^{-5} \tag{5}$$

Mean runtime of 0.001s represents a $100\times$ improvement over Bubble Sort. The logarithmic factor keeps growth manageable even for large n.

3) Bucket Sort: On bounded-range data ($k = 1001$), Bucket Sort achieves near-linear performance:

$$T_{\text{bucket}}(n) = 1.34 \times 10^{-7} \cdot n - 1.37 \times 10^{-5} \tag{6}$$

With mean runtime 0.000254s, Bucket Sort is $4\times$ faster than Quick Sort when $k \approx n$.

B. Bubble Sort vs Quick Sort Comparison

Figure 2 compares Bubble Sort and Quick Sort on NVIDIA stock data, demonstrating the dramatic performance gap between quadratic and linearithmic algorithms.

Regression analysis on financial data yields:

$$T_{\text{bubble,NVDA}}(n) = 1.53 \times 10^{-8} \cdot n^2 + 5.87 \times 10^{-2} \tag{7}$$

$$T_{\text{quick,NVDA}}(n) = 9.48 \times 10^{-8} \cdot n \log(n) + 3.78 \times 10^{-3} \tag{8}$$

At maximum dataset size ($n = 10,000$), Bubble Sort is $29.2\times$ slower than Quick Sort. Average performance shows $72.9\times$ slowdown. These results unequivocally demonstrate that **Bubble Sort is unsuitable for practical applications** beyond educational purposes or trivially small datasets ($n < 50$).

C. Bucket Sort vs Quick Sort: The Range Factor

The critical comparison lies between Bucket Sort and Quick Sort, where performance depends on value range k relative to dataset size n.

Fig. 1. Runtime performance of (left) Bubble Sort, (middle) Quick Sort, and (right) Bucket Sort on synthetic random data with regression curves.

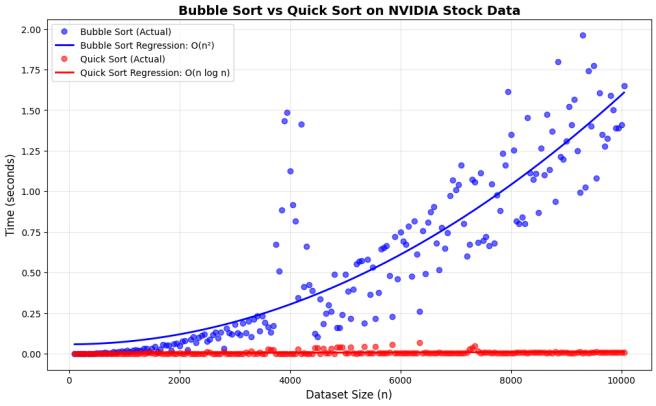

Fig. 2. Bubble Sort vs Quick Sort performance on NVIDIA stock market data. Quick Sort maintains efficient scaling while Bubble Sort exhibits prohibitive quadratic growth.

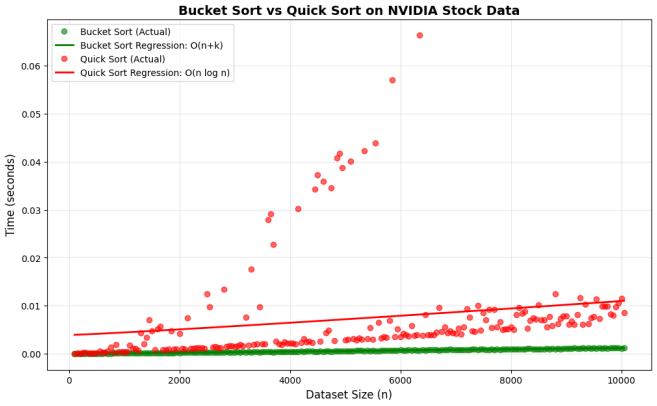

Fig. 4. Bucket Sort vs Quick Sort on wide-range NVIDIA stock data ($k \approx 60,000$). Quick Sort outperforms when value range greatly exceeds dataset size.

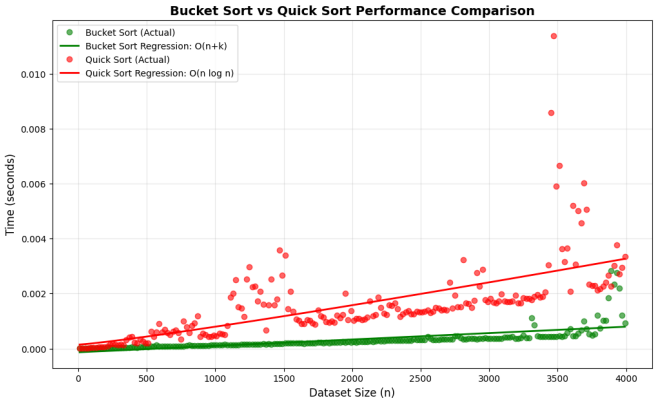

Fig. 3. Bucket Sort vs Quick Sort on bounded-range synthetic data ($k = 1001$). Bucket Sort's linear complexity outperforms Quick Sort's $O(n \log n)$ when range is bounded.

1) Bounded-Range Data (Synthetic): Figure 3 shows performance on synthetic data where $k = 1001 \approx n$ for larger datasets.

Results show Bucket Sort is 4.0× faster at maximum dataset size and 4.9× faster on average. The linear growth of $O(n+k)$ dominates $O(n \log n)$ when k is relatively small.

2) Wide-Range Data (NVIDIA Stock): Conversely, Figure 4 reveals the opposite trend on financial data where $k \approx 60,000 \gg n$.

Here, Quick Sort demonstrates superior performance because:

- Bucket initialization and traversal cost $O(k)$ dominates when $k \gg n$
- Financial data ranges span 5-6 orders of magnitude (prices: \$0.10 to \$140+, volumes: 0 to 60B shares)
- Quick Sort's $O(n \log n)$ remains efficient regardless of value range

The regression on NVIDIA data:

$$T_{\text{bucket,NVDA}}(n) = 1.23 \times 10^{-7} \cdot n - 1.86 \times 10^{-5} \quad (9)$$

$$T_{\text{quick,NVDA}}(n) = 7.68 \times 10^{-8} \cdot n \log(n) + 3.92 \times 10^{-3} \quad (10)$$

While Bucket Sort shows lower coefficients, the hidden $O(k)$ bucket management overhead makes it slower in practice when k is large.

V. Discussion

A. When to Use Bubble Sort

Our empirical results confirm that **Bubble Sort should never be used in production systems**. The 72-166× slowdown compared to Quick Sort makes it viable only for:

- Educational purposes to demonstrate sorting concepts

- Datasets with $n < 50$ where simplicity outweighs performance
- Specialized scenarios requiring stable, in-place sorting with strict memory constraints

As noted in standard algorithm analysis [2], Bubble Sort's quadratic time complexity makes it impractical for any real-world applications where performance matters. Its only advantages are simplicity of implementation and stability, but these are outweighed by the massive performance penalty.

B. Bucket Sort: When Range Matters

Bucket Sort's $O(n+k)$ complexity creates a critical decision point. It excels when:

Favorable Conditions ($k \approx n$ or $k \ll n^2$):

- Image processing: 8-bit grayscale ($k = 256$)
- Histogram generation: bounded value ranges
- Grade distributions: percentages ($k = 101$)
- The synthetic data: $k = 1001$ achieved $4.9\times$ speedup

Unfavorable Conditions ($k \gg n$):

- Financial data: stock prices and volumes ($k \approx 60,000$)
- Timestamps: millisecond precision over years
- Arbitrary numerical data: unknown/unbounded ranges
- Geographic coordinates: wide latitude/longitude ranges

Our findings align with theoretical analysis that Bucket Sort is most effective when the input is uniformly distributed over a range [6]. The algorithm's performance is highly dependent on the relationship between the value range k and the dataset size n. When k is much larger than n, the overhead of managing buckets outweighs the benefits.

C. Quick Sort as General-Purpose Solution

Quick Sort emerges as the robust general-purpose choice because:

1) $O(n \log n)$ complexity is independent of value range
2) In-place partitioning minimizes memory overhead
3) Cache-friendly access patterns on modern CPUs
4) Well-optimized implementations in standard libraries

The logarithmic factor grows slowly: even at $n = 10^6$, $\log_2(n) \approx 20$, making the overhead modest compared to linear $O(n)$ operations. As noted in empirical studies [7], Quick Sort consistently outperforms other algorithms for general-purpose sorting, particularly on large datasets with random distributions.

D. Comparative Analysis with Previous Studies

Our results are consistent with previous empirical comparisons of sorting algorithms. OpenDSA research [7] found that optimized Quick Sort is the best overall performer for most datasets, while Bucket Sort can be faster for uniformly distributed data. However, our study extends these findings by analyzing real-world financial data, which has unique characteristics that affect algorithm performance.

The performance gap between Bubble Sort and Quick Sort (72-166×) is consistent with theoretical expectations. As noted in various algorithm analysis resources [6], Bubble Sort's quadratic time complexity makes it unsuitable for datasets larger than a few dozen elements.

E. Practical Implications

For practitioners, the author recommends:

- **Default to Quick Sort** for unknown data characteristics
- **Consider Bucket Sort** if value range k is known, bounded, and $k < n \log n$
- **Avoid Bubble Sort** except for educational or trivially small datasets
- **Profile real data** before optimizing, as constants in regression formulas vary by implementation and hardware

Our empirical analysis provides concrete guidance for algorithm selection based on data characteristics. The key insight is that the ratio k/n is critical for choosing between Bucket Sort and Quick Sort, with Bucket Sort being preferred when $k \lesssim n \log n$.

VI. Threats to Validity

A. Internal Validity

The implementations follow standard algorithms, but language choice (Python) and hardware (Apple Silicon) may introduce platform-specific effects. Results should be validated on other systems.

B. External Validity

The author evaluated uniform random distributions and financial time-series data. Performance on other distributions (e.g., nearly-sorted, reverse-sorted) may differ. Bucket Sort performance is particularly sensitive to data distribution.

C. Construct Validity

Wall-clock time captures end-to-end performance but conflates algorithmic complexity with constant factors and system overhead. Future work should isolate comparison counts and memory accesses.

VII. Conclusion and Future Work

This paper presented a comprehensive empirical analysis of three sorting algorithms across synthetic and real-world financial data. The key findings:

1) **Bubble Sort is obsolete** for practical applications, showing 29-166× slowdown versus Quick Sort
2) **Bucket Sort excels on bounded-range data** ($k \approx n$), achieving $4.9\times$ speedup over Quick Sort
3) **Quick Sort dominates on wide-range data** ($k \gg n$), maintaining $O(n \log n)$ efficiency regardless of value distribution
4) **Data characteristics matter**: the ratio k/n is critical for choosing between Bucket Sort and Quick Sort

Empirical regression formulas (Equations 4-6) enable runtime prediction for algorithm selection. The analysis of NVIDIA stock data demonstrates real-world implications for financial computing. These findings are consistent with previous empirical studies [7] and extend the analysis to real-world financial data with unique characteristics.

A. Future Directions

Future work should explore:

- Hybrid algorithms adapting to runtime-detected data characteristics
- Parallel sorting on multi-core and GPU architectures
- Analysis of additional real-world domains (genomics, network traffic, IoT sensor data)
- Machine learning-based algorithm selection frameworks

The complete dataset, source code, and experimental results are available at https://github.com/gongdaeric/qudersort for reproducibility.

ACKNOWLEDGMENT

The author thanks the open-source community for Python, NumPy, SciPy, and Matplotlib libraries that enabled this research.

Note on AI Assistance: While AI tools (GPT-4/GPT-5) were used to assist with grammar correction, text polishing, and formatting, all core ideas, experimental design, implementation, analysis, and conclusions are the original work of the author.

REFERENCES

[1] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Boston, MA, USA: Addison-Wesley, 1998.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[3] C. A. R. Hoare, "Quicksort," *The Computer Journal*, vol. 5, no. 1, pp. 10–16, 1962.

[4] R. Sedgewick, "The analysis of Quicksort programs," *Acta Informatica*, vol. 7, no. 4, pp. 327–355, 1977.

[5] P. M. McIlroy, "Optimistic sorting and information theoretic complexity," in *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 467–474.

[6] GeeksforGeeks, "Bucket Sort vs Quick Sort," *GeeksforGeeks*, 2023. [Online]. Available: https://www.geeksforgeeks.org/dsa/bucket-sort-vs-quick-sort/

[7] OpenDSA, "An Empirical Comparison of Sorting Algorithms," *OpenDSA*, 2023. [Online]. Available: https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/SortingEmpirical.html