Computing Science

Bubble Sort and Quicksort Time Performance Comparison considering two features of the dataset: small/large and random/partially sorted

Wiktoria Szitenhelm

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: December 5, 2024

CS4040 Report

Department of Computing Science
University of Aberdeen
King's College
Aberdeen AB24 3UE

December 2024

Bubble Sort and Quicksort Time Performance Comparison considering two features of the dataset: small/large and random/partially sorted

Wiktoria Szitenhelm

Abstract: This study examines the time performance of two widely used sorting algorithms, Bubble Sort and Quicksort across four dataset categories defined by two features: size (small/large) and sorting degree (random/partially sorted). By implementing these algorithms in Python and analyzing their running times statistically, this research determines which algorithm performs better in specific scenarios. The results show that Quicksort outperformed Bubble Sort in large random and partially sorted datasets. Since dataset characteristics often influence algorithm performance, the findings are relevant to real-world applications where time efficiency is a key concern.

1 Introduction

Sorting algorithms are used to rearrange a given array or list of elements in an order [12]. These algorithms are foundational to computer science as a necessity for sorting arises in various contexts, from organising library catalogs to optimising search engine results [4]. As computing technology has advanced, the need for more efficient algorithms grew. This paper is about the time performance of the two popular sorting algorithms, Bubble Sort and Quicksort, and aims to determine which one performs better considering two features of the dataset: small/large and random/partially sorted.

In this report, the running times of these algorithms will be investigated under four categories: small random data, small partially sorted data, large random data, and large partially sorted data to assess their efficiency in different scenarios. The methodology involves implementing Bubble Sort and Quicksort in Python, measuring their time performance using consistent metrics, and analyzing the results statistically to identify a better algorithm for a given scenario. This approach ensures a systematic comparison of the algorithms, providing insights into their suitability for specific data scenarios.

This analysis is valuable and interesting because each scenario reflects real-world applications of sorting. For example, partially sorted data needs minimal adjustments, while small datasets may favor simpler algorithms due to lower overhead. Understanding the time performance of algorithms like Bubble Sort and Quicksort is essential as it directly impacts the application. In many computer science fields, such as database management systems, machine learning, computer graphics, operating systems, cryptography, and networking, sorting efficiency is crucial. This makes the study both an absorbing and significant task that contributes to identifying the best algorithm based on the dataset.

2 Background and Related Work

The development of sorting algorithms started in the 1950s, with early methods such as Bubble Sort [11] being simple but inefficient and having time complexities of $\mathcal{O}(n^2)$. With the invention of Quicksort by Tony Hoare, the step forward came in the 1960s. The concept of "divide and conquer" was introduced and it improved sorting efficiency [11]. The evolution of sorting algorithms is full of different sorting types but this paper focuses on these two - Bubble Sort as an early example and Quicksort as a breakthrough.

Bubble Sort works by repeatedly swapping adjacent elements if they are out of order, continuing until no swaps are needed [5]. Although popular, Bubble Sort is derided for its poor performance on random data [1], with an average and worst time complexity of $\mathcal{O}(n^2)$, making it impractical for large datasets. However, it runs in $\mathcal{O}(n)$ time on sorted data and works well on 'nearly sorted' data [1].

Quicksort, based on "divide and conquer", picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array [6]. It has an average time complexity of $\mathcal{O}(n \log n)$ and a worst-case complexity $\mathcal{O}(n^2)$ when the pivot is the largest or smallest element.

Many studies have compared the performance of Bubble Sort and Quick Sort algorithms, with Mert Metin Erdemli evaluating their efficiency in C language on arrays of varying sizes (1000, 2000, and 3000 elements). He comes to the conclusion that Bubble Sort significantly performs slower than Quick Sort for larger array sizes but Bubble Sort can be used for smaller datasets [8]. However, the study does not account for different data types, such as sorted or nearly sorted datasets, which could have influenced the results.

Carlos Rojas for his study uses JavaScript and reaches the same conclusions as Erdemli. However, he expands his research and mentions that Quicksort can exhibit less optimal behavior on nearly sorted data [3]. His research suggests that Quicksort's performance can be impacted when the data is already partially ordered, which makes the comparison more complex.

The third study compares Bubble Sort and Quicksort with other algorithms on six different sizes of datasets and it concludes that Bubble Sort did not perform well on big datasets because of its time complexity being quadratic [2]. On the other hand, Quick Sort stands out from the rest of the algorithms which makes it better suited than the simpler ones on medium and large datasets.

Peter P. Puschner in his research measures average execution and worst-case execution times for different algorithms and sizes of datasets. Based on the results of his experiment in the 'Average execution times' table, one can see that Bubble Sort performs better than Quicksort only on the dataset of 5 elements [14].

While these studies provide valuable insights, they all neglect scenarios where simpler algorithms like Bubble Sort might perform better. This gap presents an opportunity for further exploration, especially with small or partially sorted datasets, where the characteristics of the data might influence the performance of sorting algorithms differently.

3 Research Question

Finding the right sorting algorithm for the application is an important task and this paper will investigate in detail under which circumstances Bubble Sort is better than Quicksort. The previous work came to some useful conclusions that Quicksort is better for large datasets and that Bubble Sort works well on "nearly sorted" data but it's general and didn't explore various datasets to expand the research.

The Research Question is: "How do the dataset size (small/large) and sorting degree (random/partially sorted) impact the relative time performance of Bubble Sort and Quicksort."

The performance of the two sorting algorithms will be evaluated under four sorting scenarios resulting in four hypotheses related to the research question: Small Random Dataset Hypothesis, Small Partially Sorted Dataset Hypothesis, Large Random Dataset Hypothesis, and Large Partially Sorted Dataset Hypothesis.

The approach chosen to address these hypotheses is to use two sorting algorithms implemented in Python, a language different than the one used by the researchers mentioned in the related work section. Quicksort will be implemented using a recursive approach as its default implementation that naturally divides the problem into smaller subproblems, aligning well with the divide-and-conquer strategy. Contradictory, Bubble Sort's default implementation is iterative for simplicity and efficiency, as the recursive version can introduce unnecessary function call overhead and that is why iterative application is used for this algorithm.

Python's ease of use and support for performance testing make it a reasonable choice for conducting this experiment, allowing efficient implementation and accurate time comparisons. The performance of the algorithms will be assessed on four types of dataset structures: small and large random dataset (randomly ordered datasets with 10 and 10000 elements), and small and large partially sorted dataset (different degrees of partially sorted data of 10 and 10000 elements). To ensure reliable results, 50 datasets will be generated for each scenario, allowing for a thorough comparison.

The performance of two sorting algorithms in each scenario will be measured using Python function `time.perf_counter()` [9], which is known for its high precision in measuring small time intervals. Precision is essential when comparing sorting algorithms' time performance, which is why this tool was chosen to accurately record and automate performance metrics, minimizing human errors. This approach will allow not only a comparison of end results but also an in-depth observation of edge cases or significant performance shifts in specific scenarios.

There are two objectives set to achieve the aim of this project. The primary one is to analyse and compare the performance of Bubble Sort and Quicksort across different datasets and secondary to identify edge cases where either algorithm shows significant performance deviations from the expected behavior, e.g. performs too slow or too fast.

4 Experimental Design

Four concrete, testable hypotheses will first be evaluated by formulating them as null hypotheses and using statistical tools to calculate the likelihood of the data under the assumption that the null

hypothesis is true. If any null hypothesis is rejected, further analysis of the gathered data will be conducted to derive insights.

The hypotheses are:

Small Random Dataset Hypothesis: "Bubble Sort will demonstrate a better time performance than Quicksort when sorting small datasets (10 elements) with no prior order, due to its simplicity and minimal overhead on small arrays."

Small Partially Sorted Dataset Hypothesis: "Bubble Sort will show superior time efficiency compared to Quicksort on small (10 elements) partially sorted data, due to its ability to sort with minimal swaps."

Large Random Dataset Hypothesis: "Quicksort will consistently show better time performance than Bubble Sort when sorting large (10000 elements) random data with no prior order, as it efficiently handles large, randomized datasets."

Large Partially Sorted Dataset Hypothesis: "Quick Sort will show superior time efficiency compared to Bubble Sort on large (10000 elements) partially sorted data, due to its lower computational overhead and divide-and-conquer approach."

For each hypothesis in this experiment, 50 datasets will be generated to evaluate the time performance of Bubble Sort and Quicksort algorithms. The datasets are divided into two categories: random datasets and partially sorted datasets. The functions to generate datasets along with the generated sets can be found here [13].

1. **Random Datasets:** To generate random datasets, Python's feature `random.randint(1, 1000000)` will be used to generate each number in the dataset. Such a big range of numbers was selected to avoid collisions, ensuring that each dataset is unique.

2. **Partially Sorted Datasets:** Creating datasets that are partially sorted is more complex. Firstly, a set in order will be generated with the first number of the set generated using `random.randint(1, 1000000)` to ensure variability across datasets. Then a percentage of elements that are out of order in the set will be chosen randomly from a range(0,20) to allow datasets with varying levels of disorder. The range is designed to also include fully sorted datasets (with 0% disorder) allowing for a broader study, reflecting real-world scenarios where data can be slightly out of order but also perfectly ordered.

The main function takes an array of 50 elements (datasets) as input and sorts each dataset using both Bubble Sort and Quicksort algorithms within a for loop. It measures the running times of each sorting operation with the chosen Python tool `time.perf_counter()`, and stores the results in an output directory for analysis. The codebase for testing the sorting algorithms is available here [13].

To maximise the accuracy and minimise the external intervention, all other applications and background processes will be closed, leaving only the program responsible for calculating the sorting times of algorithms running. This approach provides a more accurate measure.

With the computed results (two sets of 50 running times for Bubble Sort and QuickSort for each scenario), a two-tailed t-test will be conducted on the running times of the two algorithms to

check if the p-value is less than 0.05 which is the standard threshold for significance in computer science. The purpose of this statistical test is to determine whether there is a significant difference between the execution times of the two algorithms. The null hypothesis will state that there is no significant difference in time performance between the two algorithms. If the p-value is below the stated threshold, the null hypothesis will be rejected. Based on the results, such as the average execution time, the better-performing algorithm will be determined. The algorithm with the lower average execution time will be considered the better one in terms of time performance. Results can be found here [13].

The independent variables of the experiment are two features: small/large and random/partially sorted which correspond to dataset characteristics. Dependent variables are results of the experiment which are tested time taken lengths of the sorting.

5 Results

The results obtained highlight the differences between the two sorting algorithms and they clearly inform which one has a better time performance than the other. This section will be separated into four subsections, each corresponding to one of the four hypotheses. Table 1 presents the results for both algorithms across four categories, which will be analyzed in detail.

Each subsection begins by examining the two-tailed p-value to determine whether there is sufficient evidence to reject the null hypothesis. This statistical analysis ensures that any observed patterns are not due to random chance or noise [15], which supports the alternative hypothesis. This structured approach enables a comprehensive evaluation of algorithm performance across all four tested scenarios.

Dataset Type	Algorithm	Average Time (s)	Standard Deviation (s)
Small Random Data	Bubble Sort	5.76e-06	9.53e-07
	Quick Sort	7.40e-06	8.11e-07
Small Partially Sorted Data	Bubble Sort	2.65e-06	1.65e-06
	Quick Sort	9.32e-06	1.15e-06
Large Random Data	Bubble Sort	620.13	87.55
	Quick Sort	0.19	0.02
Large Partially Sorted Data	Bubble Sort	3.22	1.00
	Quick Sort	0.32	0.90

Table 1: Comparison of Bubble Sort and Quick Sort Time Performance Across Different Datasets with Average Time and Standard Deviation.

5.1 Small Random Datasets

The first null hypothesis stating that Bubble Sort and Quicksort will demonstrate the same time performance when sorting small random datasets is rejected as the two-tailed p-value for this analysis

is smaller than 0.001. Bubble Sort performed better but the difference is not very impactful as it performed approximately 22.14% faster than Quick Sort for this dataset. Quicksort is characterised by a smaller Standard Deviation in comparison to Bubble Sort which means that data points in a dataset are less spread out or dispersed from the mean (average). This is likely due to Quicksort's divide-and-conquer approach, which typically results in more consistent performance across datasets. The T-statistic value is equal to -10.89 and it indicates a substantial difference between the sample means of the two sets. Looking at all of the timings, Bubble Sort performed better for 48 out of 50 datasets.

Looking at the outliers enables the achievement of the secondary aim of this project and the identification of edge cases where either algorithm shows significant performance deviations from the expected behavior. In Figure 1, for both Box Plots, there are some outliers that may indicate extreme variations in data that are rare or unusual. For Bubble Sort, the outliers represent unusually low execution times. These could occur if the dataset was already partially or entirely sorted, reducing the number of comparisons needed [1]. Quicksort's outlier shows unusually high execution time. This might happen if the algorithm selected poor pivot elements, such as the smallest or largest element, leading to imbalanced partitions [6].

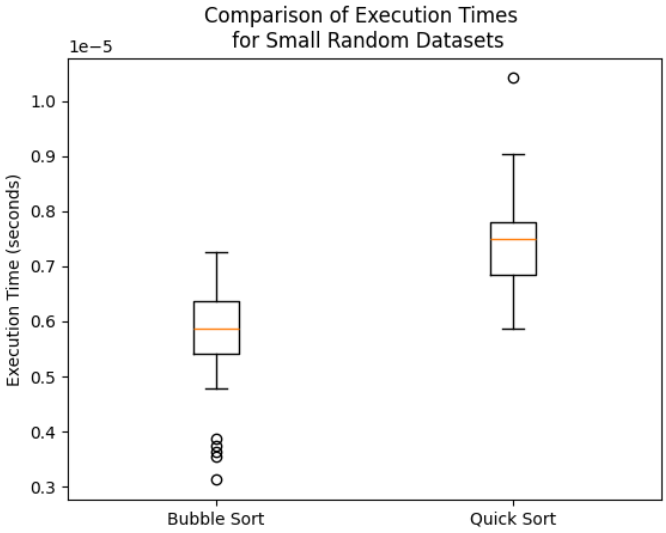

Figure 1: Bubble Sort and Quick Sort Box Plot for Small Random Datasets

5.2 Small Partially Sorted Datasets

The second null hypothesis "Bubble Sort and Quicksort will demonstrate the same time performance on small partially sorted datasets" is rejected as the test yields a two-tailed p-value smaller than 0.001. Bubble Sort performed significantly better in comparison to Quicksort having approximately 71% faster average time. Bubble Sort has a bigger Standard Deviation compared to Quicksort, meaning that data points are more spread due to its higher variability in execution times. Looking at all of the

results, Bubble Sort performed better for all datasets.

The T-statistic value of -19.64 reflects a strong difference between the two sample means. The only outlier in Figure 2 corresponds to Quicksort and represents a high execution time, which may be due to the same issue explained in the previous subsection (e.g., poor pivot selection leading to imbalanced partitions).

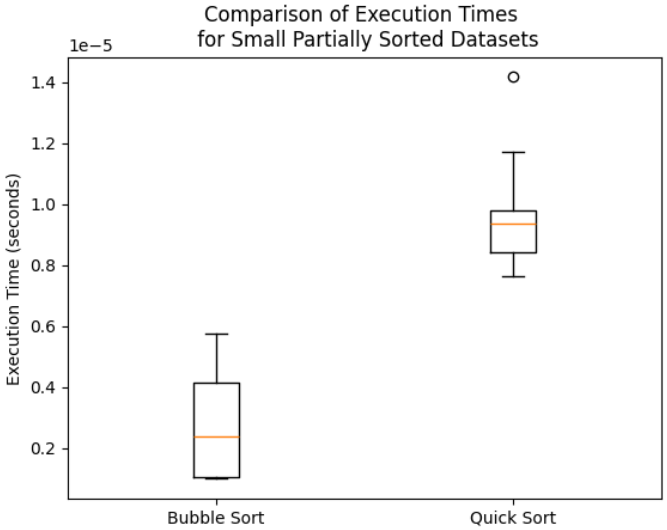

Figure 2: Bubble Sort and Quick Sort Box Plot for Small Partially Sorted Datasets

5.3 Large Random Datasets

With the two-tailed p-value smaller than 0.001, the third null hypothesis, which states that Bubble Sort and Quicksort will demonstrate the same time performance on large random datasets is rejected. Quicksort performed 32,367% faster than Bubble Sort for the given data. Quicksort has a much smaller Standard Deviation compared to Bubble Sort meaning the results are much less spread. This aligns with the previously explained behavior of Quicksort.

With the T-statistic value of 50.08, the results suggest a significant difference between the sample means of the two groups, highlighting a strong disparity in their values. Looking at Figure 3, multiple outliers below the Bubble Sort Box Plot, represent cases where the algorithm had faster-than-expected execution times. This could indicate scenarios where the dataset was partially sorted, reducing the number of swaps required. However, these outliers are still significantly higher in execution time compared to QuickSort.

5.4 Large Partially Sorted Datasets

The fourth null hypothesis ”Bubble Sort and Quicksort will demonstrate the same time performance on big nearly sorted datasets” is rejected with a two-tailed p-value smaller than 0.001. Quicksort performed much much better in comparison to Bubble Sort being 90.1% faster for the given data. Bubble Sort has a bigger Standard Deviation in comparison to Quicksort, indicating more variability

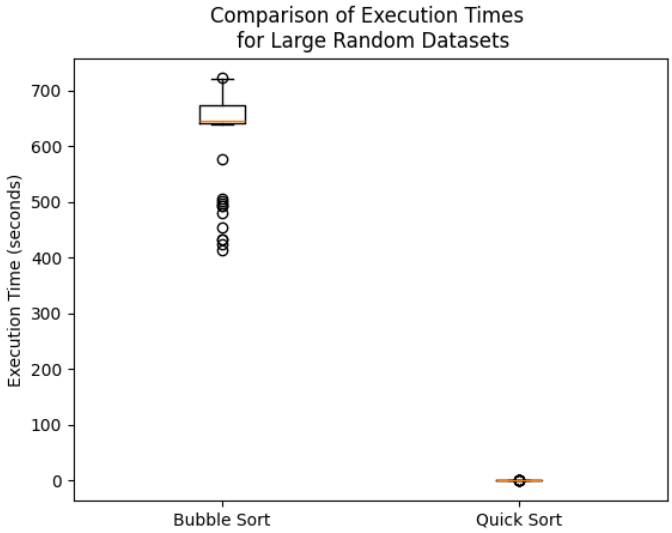

Figure 3: Bubble Sort and Quick Sort Box Plot for Large Random Datasets

in the results, consistent with the previously discussed reasons.

The T-statistic equal to 10.91 shows a considerable difference between the sample means of the two sets. There are significant outliers for both algorithms that show some edge cases present in Figure 4. Bubble Sort's outliner likely represents a scenario where the dataset was very close to being fully sorted whereas Quicksort's less-than-ideal pivot choices lead to temporarily imbalanced partitions.

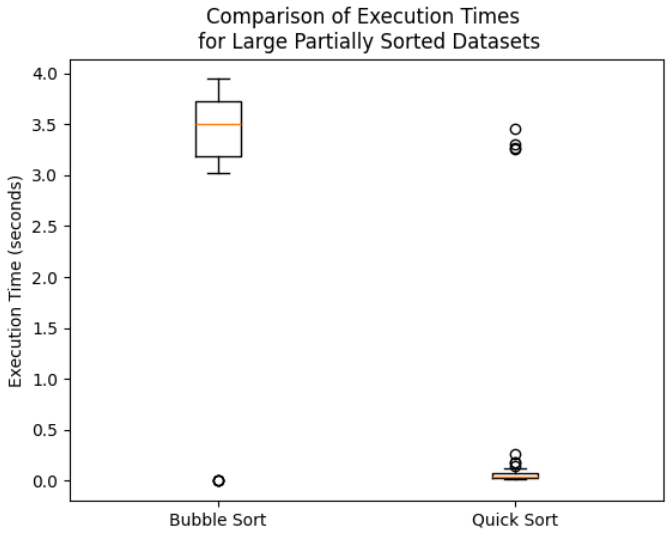

Figure 4: Bubble Sort and Quick Sort Box Plot for Large Partially Sorted Datasets

6 Discussion

As presented in the above Results section, this study provides insights into the time performance of the two sorting algorithms. Relating the findings of this experiment to past studies, the observation regarding Bubble Sort's time performance on small datasets aligns with some prior research. For example, some studies suggest that Bubble Sort performs better with small or nearly sorted data due to its simplicity [3], [8], [14].

For all the hypotheses the null hypothesis was rejected and it means that the results were not impacted by the noise. Across the board of testing the performance of Bubble Sort and Quicksort under two features, small/large and sorted/random, one can learn many things. The summary of all four results (small random data, small partially sorted data, large random data, large partially sorted data) is that Bubble Sort works better on small datasets no matter if they are sorted or not, whereas Quicksort outperforms Bubble Sort on large datasets and this is the main learning from this experiment.

The big insight of data is that for the partially sorted data Bubble Sort performed 71% faster than Quicksort, whereas for random data it was 22,14%. This shows that Bubble Sort is optimal for small datasets and especially for small partially sorted datasets. While partially sorted datasets can benefit Bubble Sort, it still remains significantly slower for large datasets compared to QuickSort.

The insight gained from the comparison results for large datasets with small datasets is that the difference in time performance for large datasets is much bigger than for small datasets and it informs that Quicksort outperforms Bubble Sort on large datasets more than Bubble Sort outperforms Quicksort on small datasets. Quicksort is faster by 90,1% for partially sorted data and 32367% for random data in comparison to Bubble Sort being faster by 71% and 22,14%.

Some edge cases were observed by looking at outliers and the biggest abnormal results are visible in Figure 4. Understanding these scenarios can highlight QuickSort's sensitivity to pivot selection which can lead to much longer executions and Bubble Sort's excellent performance on fully sorted data which results in minimal time as the number of comparisons needed is reduced.

There are two main limitations of this work. The first one is the size of data as it was tested on the datasets of a size 10000 but really big datasets that apply to real-world problems are millions of elements and testing on datasets of such size would provide a more complete analysis. Another limitation is conducting the whole experiment on CPU and there are some chances that the results would be different on GPU as processing power can significantly affect algorithm performance.

7 Conclusion & Future Work

The time performance of Bubble Sort and Quicksort under four distinct conditions based on the two dataset features: size (small/large) and sorting degree (random/sorted) was tested in this study to expand the research done so far on this topic and to learn more about sorting algorithms that are widely used in computer science. Experiments revealed that Bubble Sort outperforms Quicksort for small datasets, while Quicksort demonstrates a significantly better performance for large datasets. Notably, the sorting order (random vs. sorted) didn't change which algorithm should be the preferred

algorithm. Based on these findings, the small/large feature is a crucial factor in determining the more efficient algorithm for a given context.

Understanding this can help developers select the most suitable algorithm depending on the specific characteristics of their data, improving applications where time efficiency is a key consideration. Sorting is a fundamental operation across a wide range of computer science fields like databases, machine learning, and big data processing, in which efficiency is critical. By deepening understanding of sorting algorithms and their time performance, this study contributes to optimizing computational tasks in real-world applications.

During the project comparing the time performance of Bubble Sort and Quicksort across various dataset conditions worked well. However, the limited data sizes tested may not fully cover the algorithms' behavior on larger, real-world datasets, which would provide a better analysis of their time efficiency. If this study were to start all over again, it would test a wider variety of sorting algorithms beyond Bubble Sort and Quicksort, including Merge Sort, Insertion Sort, and perhaps newer, hybrid algorithms like TimSort, which is optimized for real-world data [10]. This would provide a broader analysis of algorithmic time performance.

The future experiments that would shed light on the research question would explore sorting algorithms with respect to large, heterogeneous datasets. This could include experimenting with even larger datasets, perhaps reaching millions of elements, to better understand how the algorithms perform under extreme conditions. Additionally, testing with datasets of intermediate sizes, ranging from 100 to 10,000 elements, could help identify the point where each algorithm begins to show clear advantages. Future studies could also explore how different hardware constraints, like multi-core processors, impact the performance of these algorithms.

References

[1] O. Astrachan, "Bubble Sort: An Archaeological Algorithmic Analysis", *ACM SIGCSE Bulletin*, vol. 35, no. 1, pp. 1–5, 2003. https://dl.acm.org/doi/abs/10.1145/792548.611918?casa$_t$*oken* = *SVFLVDwsuYAAAAAA* : *WkQlkf*9*WzL* − *dIUvWTLZPWzVhMaJxcyA* − 8*T*76*U*3*x*9*Sq*8*ikshq*8*oNVF*9*XnJBs*76*b*7*lziM*2*v*3*AIC*3*q*1, Accessed: November 7, 2024.

[2] K. A. Bakare, A. A. Okewu, Z. A. Abiola, A. Jaji, and A. Muhammed, "A comparative study of sorting algorithms: Efficiency and performance in Nigerian data systems", *FUDMA Journal of Sciences*, vol. 8, no. 5, pp. 1–5, 2024. Available: https://fjs.fudutsinma.edu.ng/index.php/fjs/article/view/2730.

[3] Carlos Rojas, "Understanding Algorithms: Bubble Sort vs Quick Sort in JavaScript", 2021, https://blog.carlosrojas.dev/ understanding-algorithms-bubble-sort-vs-quick-sort-in-javascript-f8dcdb80de85, Accessed: November 7, 2024.

[4] Ihor Gudzyk, "The Importance and Evolution of Sorting Algorithms", 2024, https://codefinity.com/blog/The-Importance-and-Evolution-of-Sorting-Algorithms, Accessed: November 7, 2024.

[5] GeeksforGeeks, "Bubble Sort Algorithm - GeeksforGeeks", 2024, https://www.geeksforgeeks.org/bubble-sort-algorithm/, Accessed: November 7, 2024.

[6] GeeksforGeeks, "Quick Sort Algorithm - GeeksforGeeks", 2024, https://www.geeksforgeeks.org/quick-sort-algorithm/?ref=header_outind, Accessed: November 7, 2024.

[7] GeeksforGeeks, "When Does the Worst Case of QuickSort Occur?", 2024, https://www.geeksforgeeks.org/when-does-the-worst-case-of-quicksort-occur/, Accessed: December 5, 2024.

[8] Mert Metin, "Introduction", 2023, https://mertmetin-1.medium.com/introduction-a12c801927a2, Accessed: November 7, 2024.

[9] Python Software Foundation, "time — Time access and conversions", n.d., https://docs.python.org/3/library/time.html, Accessed: November 7, 2024.

[10] Skerritt, "Understanding Timsort: The Hybrid Sorting Algorithm", 2021, https://skerritt.blog/timsort/, Accessed: December 2, 2024.

[11] Sifat, "The Evolution of Sorting Algorithms Over the Years: Bubble Sort to AI-Driven Sort", 2023, https://medium.com/@sifat777/ the-evolution-of-sorting-algorithms-over-the-years-bubble-sort-to-ai-driven-sort-5b009ac771e2, Accessed: November 7, 2024.

[12] GeeksforGeeks, ”Sorting Algorithms - GeeksforGeeks”, 2024, https://www.geeksforgeeks.org/sorting-algorithms/, Accessed: November 7, 2024.

[13] W. Szitenhelm, ”Bubble Sort and QuickSort Comparison”, *GitHub*, 2024. https://github.com/wszitenhelm/Bubble-Sort-and-Quicksort-Comparison, Accessed: December 2, 2024.

[14] P. P. Puschner, ”Real-time performance of sorting algorithms”, *Real-Time Systems*, vol. 16, pp. 63–79, 1999. Available: https://link.springer.com/article/10.1023/a:1008055919262#citeas.

[15] Investopedia, ”P-Value”, 2024, https://www.investopedia.com/terms/p/p-value.asp, Accessed: December 5, 2024.