Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

15th June 2025. Vol. 4. No. 3; e-ISSN: 2808-4519

Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C++ Programming Language

Moch. Nazril Ilham[1*], Andika Faza Setiawan[2], Isnaeni Kholifatun[3], M. Hamdan Aldiansyah[4], Imam Prayogo Pujiono[5]

[1,2,3,4,5] *Informatics Study Program, UIN K.H. Abdurrahman Wahid Pekalongan, Indonesia*
moch.nazril.ilham24071@mhs.uingusdur.ac.id[1*], andika.faza.setiawan24079@mhs.uingusdur.ac.id

Abstract

This study aims to analyze and compare the performance of five different data sorting algorithms, namely Shell Sort, Heap Sort, Counting Sort, Merge Sort, and Quick Sort, which are implemented using the C++ programming language. The main problem behind this research is the need for algorithms that can sequence data efficiently, both in terms of computing time and memory usage, especially when handling large datasets. The research method was carried out by testing each algorithm on three categories of datasets, namely small (100 data), medium (1,000 data), and large (10,000 data), which contained random numbers with a value range of 1–100. The test is carried out by recording the execution time and memory used during sequencing. The results show that Quick Sort is the algorithm with the fastest execution time on small and medium datasets, while Shell Sort is superior for large datasets. Meanwhile, Merge Sort tends to have the slowest runtime and highest memory consumption across all data categories. Implementing the right algorithm at the scale of the dataset has proven to be important to improve the system's efficiency in data processing. Therefore, the selection of appropriate sequencing algorithms can be a strategic solution in the development of optimal data-based systems.

Keywords: *Sorting Algorithm; Computation Time; Memory usage, C++*

1. Introduction

In this era of rapid technological development, there has been the impact of major and significant impact on data management, from manual systems to digital systems. In the field of digital data processing, the process of data sequencing (Sorting) is an important step that can help in the preparation of data so that it is easier to search, access, and analyze efficiently. [1]. Sorting is not just compiling data, it is also the foundation that underpins various search algorithms (Searching), data mining, to statistical data analysis. The data sequencing process plays a major role in improving the efficiency of data access in database systems, structured search systems, and complex big data management. [2].

In addition to the characteristics of the algorithm, the programming language also influences performance, Data Sorting. [3]. Previous researcher [4], [5] Shows that the same algorithm can show different efficiencies when implemented in different programming languages. For example, Java and C++ are known to excel in execution speed due to the nature of their compiled languages, while Python, which is interpreted in nature, tends to be slower, but is popular in conducting data research due to its simple syntax and extensive data analysis library.

Several previous studies have conducted tests to compare the efficiency of algorithms for sorting under a variety of specific conditions. Researchers [6] Show that Merge Sort and Quick Sort are superior to large-scale datasets in Python. Researchers [7] Show that Shell Sort is most optimal for medium-scale datasets in Java. Researchers [8] Show that Shell Sort is used for small-medium datasets, while Quick Sort Superior is used for large datasets. Researchers [9] show that Python performs better at running Merge Sort than other languages such as JavaScript, PHP, and C languages. [10] The Show Selection Sort is effectively used for small datasets in PHP.

Researchers [11] Show that Bubble Sort is not suitable for use on large datasets in Java. Researchers [12] They are also doing a comparison of the algorithms Bubble Sort, Insert Sort, and Quick Sort using the Python programming language, and the results show that Quick Sort has an advantage in execution time. Researchers [13] show that the Hybrid Sorting, i.e., combining algorithms Selection Hybrid Sort and algorithms Bucket Sort It can save time and also provide the efficiency of the sequencing algorithm and is proven to be able to speed up the sequencing process on large-scale datasets.

Although there have been many studies conducted before, there has still been no study that has conducted a study of five algorithmic sequences cooperatively with three different data set sizes using the C++ programming language. Most previous studies have only compared 2-3 sequencing algorithms, and have been limited to a single dataset scale. On that basis, in this study, the researcher conducted a comparative analysis of five different sorting algorithms, namely the Shell Sort, Heap Sort, Counting Sort, Merge Sort, and Quick Sort algorithms implemented in the C++ programming language with the data size category that the researcher had designed, namely 100 data, 1,000 data, and 10,000 data, using number-type data randomized with values Starting from 1-100.
Researchers [14] In his research, he emphasizes that when we talk about algorithm efficiency, often our focus is only on execution time, and no longer on memory. In the past, computer memory limitations were indeed a crucial issue, but today, with the rapid development of technology, memory is no longer the main obstacle. In fact, in this fast-paced data era, execution time is a major consideration, especially when algorithms are applied to process large-scale data. In other words, it is the algorithm that is capable of giving the fastest results that will be preferred, although it requires a little more memory space.
This makes this research's focus on time efficiency highly relevant and contextual to today's real-world needs. To determine which algorithm is more efficient, the five sequencing algorithms will pass tests by calculating how long it takes and how much memory is used to manage data from various data sources. This test was carried out on data that varied from the smallest size, namely (100) data, medium (1,000) data, and large (10,000) data. So that it can be identified later which algorithm in the C++ language is the most effective in memory usage and processing lead times on small, medium, and large data.

2. Research Methods

The researcher used a research method that started from creating a dataset filled with random numbers from the range of values 1-100 for testing. This dataset is classified into three groups based on the amount of data: 100 data for small-scale datasets, 1,000 data for medium-scale datasets, and 10,000 data for large-scale datasets, following the program syntax for creating datasets.

```
std::vector<int> createDataset(int n) {
   std::vector<int> data(s);
   std::mt19937 gen(std::random_device{}());
   std::uniform_int_distribution<> dis(1, 100);
   for (auto& x : data) x = dis(gen);
   return data;
}
```

Once the dataset has been successfully created, the next step is to apply various sorting algorithms to the dataset. These algorithms work by sequencing data in ascending order, so that their efficiency can be compared in handling different amounts of data. To measure the performance of the algorithm, this study recorded the execution time and memory usage before, during, and after the sequencing process was carried out. The execution time is calculated by logging the time difference before and after the algorithm is executed using the std::chrono::high_resolution_clock::now() function of the <chrono> library, which provides an accuracy of up to nanoseconds and provides a clear picture of the speed of the sequencing process.

Memory usage is measured approximately based on the memory allocation of the data container using the sizeof() function and monitoring memory usage during runtime with the help of an external library or memory profiler, so that information about memory usage can be seen. The measurement results of these two methods became a reference for comparing the time efficiency and resource use of the five sequencing algorithms tested in each dataset category. Once all the tests are completed, the results are presented in the form of a comparison table. This table shows how fast and efficient each algorithm is in handling small, medium, and large datasets. Figure 1 shows an overview of the scheme carried out in this study.

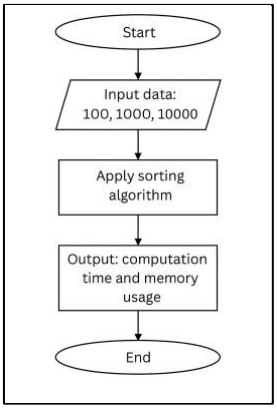

Figure 1: Sequencing Algorithm Process Network Diagram

3. Results and Discussion

Data sequencing is a fundamental process in programming that functions to arrange elements in a data structure based on certain rules. This process is used in various areas of computing, including data search, database processing, and algorithm optimization. Researchers [14] Explain that sorting algorithms can be categorized based on the methods used in compiling data elements, such as in algorithms Bubble Sort, Insert Sortand Selection Sort, which use simpler or less complicated methods. This algorithm is easy to implement, but it has a time complexity of $O(n^2)$, which is bad, making it less than optimal for large datasets.

Quick Sort applies a more efficient method by utilizing the divide and conquer technique or a specific data structure. The complexity of the O(n log n) time that this algorithm has makes it more suitable for large-scale data processing. Counting Sort uses an enumeration technique to determine the position of elements without conducting a direct comparison. This approach allows for a faster sequencing process under certain conditions, especially if the dataset has a limited range of values.

The selection of the sequencing algorithm depends on the size of the dataset, the data structure used, and the need for time and memory efficiency. This study compares the efficiency of several sorting algorithms based on execution time and memory usage with different dataset sizes. This research was implemented using the C++ programming language and VSCode (Visual Studio Code), which acts as a medium to operate or run the program code to be tested. The algorithm was tested using device specifications with Windows 11 64-bit, 8 GB of RAM, AMD Ryzen 5 CPU, and 512 GB SSD.

The algorithms that have been determined by the researcher in this study are Shell Sort, Heap Sort, Counting Sort, Merge Sort, and Quick Sort. Where each algorithm will be evaluated through a similar data set, namely 100 data, 1,000 data, and 10,000 data, using data types in the form of random numbers ranging from 1-100. A dataset of 100 data is used for the first to third experiments of each sequencing algorithm, while a dataset of 1,000 data is used for the fourth to sixth experiments, and a dataset of 10,000 data is used for the seventh to ninth experiments of each sequencing algorithm.

In this study, the researcher conducted three tests so that the results obtained were valid and reliable. During the testing process, researchers only open and run three applications to maximize CPU performance and ensure nothing interferes with CPU and memory performance during testing: VSCode for where the test takes place, the Snipping Tool is used to take pictures of test results, and Microsoft Word is used to manage data or record test results for comparison.

Here is the dataset link that the researchers used in this test: Experimental Dataset Link. In this test, the researcher assesses each algorithm based on the length of computing time and the amount of memory usage during the computational time. Documentation for each test conducted can be accessed at the following links: Shell Sort Experiment Link, Counting Sort Experiment Link, Heap Sort Experiment Link, Merge Sort Experiment Link, and Quick Sort Experiment Link. The explanation related to the testing of each algorithm obtained is as follows.

3.1. Shell Sort Algorithm

The Shell Sort algorithm is a development of the Insertion Sort algorithm that works by first sorting the spaced elements using a certain interval, then gradually shrinking the interval to 1 [15]. Shell Sort is more efficient than Insertion Sort for large datasets, as it can reduce the number of element shifts by performing initial sorting on scattered elements.

```
void shellSort(vector<int>& a) {
   for (int g = a.size()/2; g > 0; g /= 2)
      for (int i = g; i < (int)a.size(); i++) {
         int t = a[i], j = i;
         for (; j >= g &&&a[j-g] > t; j -= g) a[j] = a[j-g];
         a[j] = t;
      }
}
```

The code view above is an example of an implementation of the Shell Sort algorithm within the C++ programming language. The dataset obtained will be input into the program and executed based on a predetermined test scheme: the first to third tests use 100 data or small categories, the fourth to sixth tests use 1,000 data or medium categories, and the seventh to ninth tests use 10,000 data or large categories. And the results of this test the researcher got the results of the Shell Sort algorithm test, which can be seen in Table 2 below.

Table 1: Shell Sort Experiment

Experiment To	Data Size	Processing Time (nanoseconds)	Memory Usage (bytes)
1	100	7.900	8.192
2	100	10.900	8.192
3	100	10.800	8.192
4	1.000	166.900	8.192
5	1.000	114.300	8.192
6	1.000	114.400	8.192
7	10.000	1.602.900	8.192
8	10.000	1.548.900	8.192
9	10.000	1.739.700	8.192

If you look at the table above, the average computation time is 9,866 nanoseconds for testing with 100 data (small), for testing with 1,000 data (medium) it is 131,866 nanoseconds, and for testing with 10,000 data (large) it is 1,630,500 nanoseconds. However, based on the size of memory used during computing time, the average memory used for all data sizes is 8,192 bytes.

3.2. Counting Sort Algorithm

Algorithm Counting Sort algorithm is one of the many data sorting methods that operate by counting the number of occurrences of each element in the list, then determining the appropriate position for each element in the final result. Called Counting Sort because the process is based on recording the number of occurrences of elements, rather than comparisons between elements, like other sorting algorithms**.** According to [16]Counting Sort has linear time complexity $O(n + k)$**,** which makes it faster than algorithms like Bubble Sort and Insert Sort when processing datasets with small value ranges**.**

```
void countingSort(vector<int>& a) {
   if (a.empty()) return;
   int m = *max_element(a.begin(), a.end());
   vector<int> c(m + 1, 0);
   for (int x : a) c[x]++;
   int i = 0;
   for (int j = 0; j <= m; j++)
      while (c[j]--) a[i++] = j;
}
```

The code view above is an example of an implementation of the Counting Sort algorithm in the C++ programming language. The dataset obtained will be input into the program and executed based on a predetermined test scheme: the first to third tests use 100 data or small categories, the fourth to sixth tests use 1,000 data or medium categories, and the seventh to ninth tests use 10,000 data or large categories. And the results of this test the researcher obtained the results of the Counting Sort algorithm test, which can be seen in Table 2 below.

Table 2: Counting Sort Experiment

Experiment To	Data Size	Processing Time (nanoseconds)	Memory Usage (bytes)
1	100	15.400	12.288
2	100	7.500	8.192
3	100	5.700	8.192
4	1.000	21.000	8.192
5	1.000	20.900	8.192
6	1.000	38.300	12.288
7	10.000	174.400	8.192
8	10.000	179.900	8.192
9	10.000	179.400	8.192

If you look at the table above, the average computation time is 9,533 nanoseconds for testing with 100 data (small), for testing with 1,000 data (medium) it is 26,733 nanoseconds, and for testing with 10,000 data (large) it is 177,900 nanoseconds. However, based on the size of the memory used during computing time, the average memory used for testing 100 data is 9,560 bytes, for testing with 1,000 data is 9,560 bytes, and for testing with 10,000 data is 8,192 bytes.

3.3. Heap Sort Algorithm

The Heap Sort algorithm is a data sorting method that operates by building a Heap structure from a set of elements, and then repeatedly removing the largest or smallest elements to achieve the exact sequence. [17]. The main advantage of Heap Sort is that it doesn't require additional space like Merge Sort, as the sorting process is done directly within an existing array.

```
def heap_sort(arr):
   for i in range(len(arr) // 2 - 1, -1, -1): Heapify(arr, len(arr), i)
   for i in range(len(arr) - 1, 0, -1): arr[i], arr[0] = arr[0], arr[i]; Heapify(arr, i, 0)
   return arr void mergeSort(vector<int>& a, int l, int r) {
   if (l >= r) return;
   int m = (l + r)/2;
   mergeSort(a, l, m);
   mergeSort(a, m+1, r);
   vector<int> t; int i = l, j = m + 1;
   while (i <= m && j <= r) t.push_back(a[i] < a[j] ? a[i++] : a[j++]);
   while (i <= m) t.push_back(a[i++]);
   while (j <= r) t.push_back(a[j++]);
   for (int k = 0; k < t.size(); k++) a[l + k] = t[k];
}
```

The code view above is an example of an implementation of the Heap Sort algorithm in the C++ programming language. The dataset used is input into the program and executed according to a predetermined test scheme: the first to third tests use 100 data (small), the fourth to sixth tests use 1,000 data (medium), and the seventh to ninth tests use 10,000 data (large). The results of this test are obtained the results of the Heap Sort algorithm test, which can be seen in Table 3 below.

Table 3: Heap Sort Experiment

Experiment To	Data Size	Processing Time (nanoseconds)	Memory Usage (bytes)
1	100	12.600	8.192
2	100	19.000	8.192
3	100	13.000	8.192
4	1.000	184.000	8.192
5	1.000	184.500	8.192
6	1.000	179.600	8.192
7	10.000	2.434.600	8.192
8	10.000	2.454.600	8.192
9	10.000	2.493.700	8.192

If you look at the table above, the average computation time, which is 14,866 nanoseconds for testing with 100 (small) data, for testing with 1,000 data (medium) is 182,700 nanoseconds, and for testing with 10,000 data (large) is 2,460,966 nanoseconds. However, based on the size of memory used during computing time, the average memory used for all data sizes is 8,192 bytes.

3.4. Merge Sort Algorithm

The Heap Sort algorithm is a data sorting method that operates by building a Heap structure from a set of elements, and then repeatedly removing the largest or smallest elements to achieve the exact sequence. [18]. The main advantage of Heap Sort is that it doesn't require additional space like Merge Sort, as the sorting process is done directly within an existing array.

```
void mergeSort(vector<int>& a, int l, int r) {
   if (l >= r) return;
   int m = (l + r)/2;
   mergeSort(a, l, m);
   mergeSort(a, m+1, r);
   vector<int> t;
   int i = l, j = m + 1;
   while (i <= m && j <= r)
      t.push_back(a[i] < a[j] ? a[i++] : a[j++]);
   while (i <= m) t.push_back(a[i++]);
   while (j <= r) t.push_back(a[j++]);
   for (int k = 0; k < t.size(); k++) a[l + k] = t[k];
}
```

The code view above is an example of an implementation of the Merge Sort algorithm within the C++ programming language. The dataset obtained will be input into the program and executed based on a predetermined test scheme: the first to third tests use 100 data or small categories, the fourth to sixth tests use 1,000 data or medium categories, and the seventh to ninth tests use 10,000 data or large categories. The results of this test are obtained from the results of the Merge Sort algorithm test, which can be seen in Table 4 below.

Table 4: Merge Sort experiment

Experiment To	Data Size	Processing Time (nanoseconds)	Memory Usage (bytes)
1	100	50.200	8.192
2	100	55.900	8.192
3	100	57.300	8.192
4	1.000	469.900	12.288
5	1.000	315.300	12.288
6	1.000	325.800	16.384
7	10.000	3.277.700	53.248
8	10.000	3.278.000	53.248
9	10.000	3.307.800	53.248

If you look at the table above, the average computation time is 54,466 nanoseconds for testing with 100 data (small), for testing with 1,000 data (medium) it is 370,333 nanoseconds, and for testing with 10,000 data (large) it is 3,287,833 nanoseconds. However, based on the size of the memory used during compute time, the average memory used for testing 100 data is 8,192 bytes, for testing with 1,000 data is 13,653 bytes, and for testing with 10,000 data is 53,248 bytes.

3.5. Quick Sort Algorithm

Algorithm Quick Sort is one of the algorithms with a data sequencing method that uses the principle of divide and conquer In the sequencing method, where one existing element will be selected as a pivot and then the list is divided into two parts namely: on the left the element is considered smaller than the pivot and on the right the element is considered to be larger than the pivot [19].

```
void quickSort(vector<int>& arr, int low, int high) {
        if (low < high) {
        int p = partition(arr, low, high); pivot at the end
        quickSort(arr, low, p-1);
        quickSort(arr, p + 1, high);
    }
}
```

The code view above is an example of an implementation of the Quick Sort algorithm within the C++ programming language. The dataset obtained will be input into the program and executed based on a predetermined test scheme: the first to third tests use 100 data or small categories, the fourth to sixth tests use 1,000 data or medium categories, and the seventh to ninth tests use 10,000 data or large categories. And the results of this test the researcher obtained the results of the Quick Sort algorithm test, which can be seen in Table 5 below.

Table 5: Quick Sort Experiment

Experiment To	Data Size	Processing Time (nanoseconds)	Memory Usage (bytes)
1	100	11.600	8.192
2	100	6.700	8.192
3	100	7.600	8.192
4	1.000	88.200	8.192
5	1.000	89.000	8.192
6	1.000	89.400	8.192
7	10.000	1.701.600	8.192
8	10.000	1.679.800	8.192
9	10.000	2.083.200	8.192

If you look at the table above, the average computation time is 8,633 nanoseconds for testing with 100 data (small), for testing with 1,000 data (medium) it is 88,866 nanoseconds, and for testing with 10,000 data (large) it is 1,821,533 nanoseconds. However, based on the size of memory used during computing time, the average memory used for all data sizes is 8,192 bytes.

Based on the analysis that has been carried out, all test results are obtained with the average value of each algorithm, and we summarize the results in Table 6 below.

Table 6: Average Results of Five Sequencing Algorithms

Algorithm Name	Data Size	Processing Time (nanoseconds)	Memory Usage (bytes)
Shell Sort	100	9.866	8.192
Counting Sort	100	12.866	8.192
Heap Sort	100	14.866	8.192
Merge Sort	100	54.466	8.192
Quick Sort	100	8.633	8.192
Shell Sort	1.000	131.866	8.192
Counting Sort	1.000	182.700	8.192
Heap Sort	1.000	182.700	8.192
Merge Sort	1.000	370.333	13.653
Quick Sort	1.000	88.866	8.192
Shell Sort	10.000	1.630.500	8.192
Counting Sort	10.000	2.460.966	8.192
Heap Sort	10.000	2.460.966	8.192
Merge Sort	10.000	3.287.833	53.248
Quick Sort	10.000	1.821.533	8.192

Based on the results of testing five sorting algorithms with three categories of data quantity, namely 100, 1,000, and 10,000, some information can be obtained as follows:

1. Almost all algorithms in the dataset of 100 data show the same memory usage, which is 8,192 bytes. The fastest processing time was achieved by Quick Sort with an average of 8,633 nanoseconds, followed by Shell Sort and Counting Sort. Meanwhile, Merge Sort is an algorithm with the slowest execution time, which is 54,466 nanoseconds.

2. In a dataset of 1,000 datasets, the memory usage of most algorithms remained stable at 8,192 bytes, except for Merge Sort, which requires a larger memory of 13,653 bytes. Quick Sort remains the fastest algorithm with a processing time of 88,866 nanoseconds, followed by Shell Sort. Merge Sort is again an algorithm with the slowest execution time, which is 370,333 nanoseconds.

3. For datasets of 10,000 data points, almost all algorithms use 8,192 bytes of memory, except for Merge Sort, which increases significantly to 53,248 bytes. Shell Sort became the algorithm with the fastest processing time, which was 1,630,500 nanoseconds, followed by Quick Sort. Merge Sort still shows the slowest performance with a time of 3,287,833 nanoseconds.

4. Conclusion

Based on the research that has been conducted, each sequencing algorithm shows significant differences in computing time efficiency and memory usage. On a small dataset (100 data points), Quick Sort was the algorithm with the fastest execution time, followed by Shell Sort and Counting Sort, while Merge Sort showed the slowest performance. For medium datasets (1,000 data points), Quick Sort remains superior in execution speed, while Shell Sort and Counting Sort also maintain stable performance. Merge Sort consistently uses the most memory at this dataset size. When the amount of data increases to 10,000, Shell Sort becomes the best choice because it has the fastest computation time, followed by Quick Sort. Merge Sort remains the algorithm with the highest time and memory usage on all dataset size variations. Therefore, when choosing a sorting algorithm, Shell Sort and Quick Sort are more recommended for different dataset sizes, while Merge Sort is more suitable when data sequence stability is a top priority. These findings provide important insights into determining the most efficient sequencing algorithm based on dataset size, as well as time and memory optimization needs.

Acknowledgement

Finally, the researcher was grateful and thanked UIN K.H. Abdurrahman Wahid Pekalongan for the support and facilities provided during this research process. Thank you also to the supervisor, Mr. Imam Prayogo Pujiono, M. Kom., for his guidance, direction, and very meaningful input in every stage of research until the preparation of this journal.

References

[1] M. F. Nanang Mahrozi, "Analisis Perbandingan Kecepatan Algoritma Selection Sort Dan Bubble Sort," *Jurnal Ilmiah Sain dan Teknologi*, vol. 1, pp. 89–98, 2023.

[2] S. Aisyah, *Memahami Konsep Dasar Algoritma Pemrograman Dengan Menggunakan Metode Pseuducode*. 2022. doi: 10.31219/osf.io/n269k.

[3] Halimatussya'diyah Purba and Yahfizham, "Konsep Dasar Pemahaman Algoritma Pemrograman," *Jurnal Arjuna : Publikasi Ilmu Pendidikan, Bahasa dan Matematika*, vol. 1, no. 6, pp. 290–301, 2023, doi: 10.61132/arjuna.v1i6.356.

[4] J. Iskandar, H. Suhendar, and B. D. Pamungkas, "Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python," *G-Tech: Jurnal Teknologi Terapan*, vol. 8, no. 1, pp. 104–113, 2023, doi: 10.33379/gtech.v8i1.3556.

[5] N. Maulida Surbakti, A. Talia, C. Br Perangin-Angin, D. Olivia Nainggolan, N. Devi Friskauly, and S. Ruth Br Tumorang, "Penggunaan Bahasa Pemrograman Python dalam Pembelajaran Kalkulus Fungsi Dua Variabel," *Algoritma : Jurnal Matematika, Ilmu pengetahuan Alam, Kebumian dan Angkasa*, vol. 2, no. 3, pp. 98–107, 2024.

[6] Y. Heryanto and T. Wira Harjanti, "Analisis Perbandingan Ruang dan Waktu pada Algoritma Sorting Menggunakan Bahasa Pemrograman Python," *KESATRIA: Jurnal Penerapan Sistem Informasi (Komputer & Manajemen)*, vol. 4, no. 2, pp. 342–347, 2023.

[7] M. Luthfi Zulfa, B. Nurina Sari, and U. Singaperbangsa Karawang Abstract, "Analisis Perbandingan Algoritma Bubble Sort, Shell Sort, dan Quick Sort dalam Mengurutkan Baris Angka Acak Menggunakan Bahasa Java," *Jurnal Ilmiah Wahana Pendidikan*, vol. 2022, no. 13, pp. 237–246, 2022.

[8] I. P. Pujiono, R. B. Trianto, and F. M. Hana, "Perbandingan Efisiensi Memori Dan Waktu Komputasi Pada 7 Algoritma Sorting Menggunakan Bahasa Pemrograman Java," *Jurnal Sistem Informasi dan Sistem Komputer*, vol. 9, no. 2, pp. 218–230, 2024, doi: 10.51717/simkom.v9i2.481.

[9] S. M. B. Syed Muqeet Aqib, Haque Nawaz, "Analysis of Merge Sort and Bubble Sort in Python, PHP, JavaScript, and C language," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 10, no. 2, pp. 680–686, 2021, doi: 10.30534/ijatcse/2021/311022021.

[10] Y. A. Sandria, M. R. A. Nurhayoto, L. Ramadhani, R. S. Harefa, and A. Syahputra, "Penerapan Algoritma Selection Sort untuk Melakukan Pengurutan Data dalam Bahasa Pemrograman PHP," *Hello World Jurnal Ilmu Komputer*, vol. 1, no. 4, pp. 190–194, 2022, doi: 10.56211/helloworld.v1i4.187.

[11] N. Sari, W. A. Gunawan, P. K. Sari, I. Zikri, and A. Syahputra, "Analisis Algoritma Bubble Sort Secara Ascending Dan Descending Serta Implementasinya Dengan Menggunakan Bahasa Pemrograman Java," *ADI Bisnis Digital Interdisiplin Jurnal*, vol. 3, no. 1, pp. 16–23, 2022, doi: 10.34306/abdi.v3i1.625.

[12] M. B. Sena *et al.*, "Perbandingan Kinerja Algoritma Sorting Dalam Pengurutan Data Menggunakan Bahasa Python," *Prosiding Seminar Nasional Sains dan Teknologi Seri 02 Fakultas Sains dan Teknologi*, vol. 1, no. 2, pp. 310–314, 2024.

[13] R. P. Aryanto, A. Nilogiri, and A. E. Wardoyo, "Optimasi Pengurutan Data Bilangan dengan Menggabungkan Algoritma Selection Sort Hybrid dan Bucket Sort," *Edumatic: Jurnal Pendidikan Informatika*, vol. 7, no. 1, pp. 39–48, 2023, doi: 10.29408/edumatic.v7i1.12358.

[14] F. A. Hia, "Analisis Perbandingan Algoritma Pengurutan (Sorting) Berdasarkan Kompleksitas Waktu," 2022.

[15] M. Faridz, T. S. Hidayah, P. Studi, T. Informatika, and P. N. Pontianak, "Perbandingan Algoritma Selection Sort , Shell Sort , Dan Merge Sort Pada Data Sampling Numerik Menggunakan Matplotlib," vol. 1, no. 2, pp. 253–265, 2024.

[16] C. Song and H. Li, "Improvement of Counting Sorting Algorithm," *Journal of Computer and Communications*, vol. 11, no. 10, pp. 12–22, 2023, doi: 10.4236/jcc.2023.1110002.

[17] R. R. Basir, "Analisis Kompleksitas Ruang dan Waktu Terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort dan Merge dengan Pemrograman Java," *STRING (Satuan Tulisan Riset dan Inovasi Teknologi)*, vol. 5, no. 2, p. 109, 2020, doi: 10.30998/string.v5i2.6250.

[18] S. M. B. Syed Muqeet Aqib, Haque Nawaz, "Analysis of Merge Sort and Bubble Sort in Python, PHP, JavaScript, and C language," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 10, no. 2, pp. 680–686, 2021, doi: 10.30534/ijatcse/2021/311022021.

[19] D. S. Rita Wahyuni Arifin, "Algoritma Metode Pengurutan Bubble Sort dan Quick Dalam Bahasa Pemrograman C++," *Information System for Educators and Professionals*, vol. 4, no. 2, pp. 178–187, 2020.