JESICA: Journal of Enhance Studies in Informatics and Computer Applications
Journal homepage: jesica.itsk-soepraoen.ac.id
ISSN: e-ISSN: 3046-6938 p-ISSN: 3046-6997
Vol. 2, No. 2, July 2025, Pp. 52-61

Performance Analysis of Sorting Algorithm in Student Data Processing

Maulana Aditya[1], Hadani Zenitha[2], Syahwa Wahyu[3], Bintang Novala[4], Deva Muhammad[5], Ahsanun Naseh Khudori*[6]
1, 2, 3, 4, 5, 6 Institut Teknologi Sains dan Kesehatan RS.DR. Soepraoen Kesdam V/BRW, Indonesia

***Corresponding author**
E-mail address:
ahsanunnaseh@itsk-soepraoen.ac.id

Keywords:
Sorting algorithm, efficiency, speed, student data, merge sort, quick sort

Abstract
The arrangement of large amounts of student data usually presents a challenge for administrators in wrapping up student administrative tasks. Several sorting algorithms are used to solve this problem, but with the evolution of technology, various sorting algorithms have emerged. This study focuses on the performance of four widely used sorting algorithms—Bubble Sort, Insertion Sort, Merge Sort, and Quick Sort—in sorting data based on student ID numbers, names, and GPA. The study involves testing the algorithms across several scenarios, including pre-sorted data, random data, and data with duplicate elements, by measuring execution time and the number of computational operations as indicators of efficiency. The results show that Bubble Sort and Insertion Sort perform well on small datasets but are less effective on large datasets. Conversely, Merge Sort and Quick Sort are highly effective on complex datasets. While Merge Sort is more stable for large datasets with specific structures, Quick Sort operates faster on random data. These results provide practical guidance for selecting the best sorting algorithm based on data size and complexity. These results can support data management efficiency in academic environments. With the results obtained, this research can serve as an important reference for developing more efficient academic information systems and data management in educational institutions.

1. Introduction
Higher education is one of the last education providers that has some complexity to the management of administrative information of the academic community. One of the frequently used academic community administration information is student data. The scope of student data used is name, Student Identification Number, and Grade Point Average (GPA). This data is widely used in scheduling, achievement ranking, and academic analysis of each student, so efficient data management can facilitate the student administration process [1]. The efficiency of student data management can be done by implementing various kinds of algorithm methods developed by technological advances, one of which is the Sorting algorithm method or data sorting [2]. Sorting algorithm is worth considering as one of the solutions to data management problems [3].

Sorting is the structuring of random data with certain conditions. The data types commonly used in this algorithm are numeric and plain text [4]. There are various sorting algorithms that can be used to organize data, each algorithm has different characteristics, efficiency levels, and speeds, depending on the type and size of the data being processed [5]. In its development, many types of Sorting algorithms were created to handle the data sorting process, and each has advantages and disadvantages that can affect the user's decision in certain situations. Bubble Sort, Insertion Sort, Merge Sort, and Quick Sort are the most widely used types of Sorting algorithms, due to their ease of use and ability to handle various scenarios [5]. However, each algorithm has different performance characteristics. For example, the Merge Sort algorithm may require more memory than the Insertion Sort algorithm, although the execution speed is superior to Merge Sort [6].

As an effort to complement previous research conducted by Bima Sena et al by testing the Quick Sort, Bubble Sort, and Merge Sort algorithms to see the execution time and the number of steps taken in the process of sorting various amounts of data [7]. Testing sorted, randomized, and duplicate data is included in this test scenario. In addition, Insertion Sort is also included in this research so that there are four algorithms to be compared. Another research was also conducted by J. Hammad using Worst Case, Average Case and Best Case methods to compare Sorting algorithms [9]. In his research, comparisons were made to the Gnome Sort, Selection Sort, and Bubble Sort algorithms. The final result obtained is that the execution of Gnome Sort is faster for sorted data, while for unsorted data Selection Sort is faster in execution [9]. This study will look at many indicators that determine efficiency in various situations, the indicators in question are execution time, the number of steps taken, the amount of memory used by each algorithm, and trials using Worst Case, Average Case, and Best Case.

Evaluating the performance of the four most popular sorting algorithms in student data management is the objective of this study. It is hoped that this research can provide a thorough understanding of the advantages and disadvantages of each algorithm in various conditions, as well as practical guidance in choosing the most suitable algorithm. The selection of the right Sorting algorithm greatly affects the speed and efficiency of data processing [7]. The quality and performance of the program is directly affected by the development of algorithm efficiency, because the selection of the right algorithm is necessary to solve the problem properly [7]. The selection of the right algorithm based on the size and complexity of the data is very important for real-world applications, especially in the field of education to ensure that data processing can be done efficiently and in a timely manner [10]. With the results presented, this journal not only adds to the scientific literature on organizing algorithms, but also offers practical benefits for educational institutions to manage student data more efficiently and effectively.

2. Research Method

The research method is a sequence of steps or procedures that researchers use to achieve research objectives by collecting and analyzing data [11]. In this study, the authors used the literature review research method, which is the process of collecting data by digging up information relevant to the main topic [12]. This approach aims to integrate the findings of reliable data or information and summarize the findings [12]. After obtaining the data, the author will analyze the findings with 3 steps, namely sorting data to filter relevant information, then presenting the data to present the findings, and drawing conclusions to describe the implications of the research results [12]. The figure below shows the stages of the research stages in more detail.

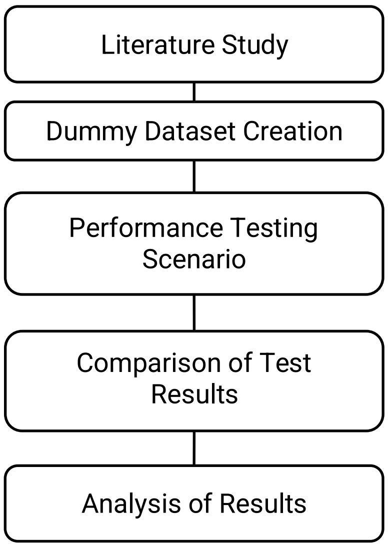

Figure 1. Research stages

It can be seen in Figure 1, the research stage begins with a literature study of previous research, followed by the creation of a dummy dataset as a variable that is tested into several different sizes. After completing the dummy dataset, the next research stage is to create a performance testing scenario for each algorithm and continue by comparing the test results. The last stage is to analyze the test results that have been carried out on each algorithm is the most recent stage in conducting this research.

2.1 Literature Study

The performance of the four algorithms we studied has been tested by several previous researchers, and various conclusions have been drawn according to the test scenario of each study. The first journal is research conducted by Bima Sena et al in their journal entitled "Comparison of Sorting Algorithm Performance in Sorting Data Using Python". In the journal, comparisons were made on the amount of data to get comparative results between execution time and the number of steps by each algorithm method, namely, Quick Sort, Bubble Sort, and Merge Sort [7]. The result obtained from the comparison is that the Quick Sort algorithm method is superior to other methods in several scenarios that have been applied.

The second journal that discusses the performance of sorting algorithms is Akmal Isyqi et al's journal, titled "Comparison of Bubble Sort, Insertion Sort and Selection Sort Algorithm Performance Using Numeric Data in Python Programming". According to the research, it was concluded that *Bubble Sort* is not superior to 2 other algorithms on large datasets. *Selection* and *Insertion Sort* have a more stable time increase as the dataset size increases [13].

Research in the Journal of Computer Engineering, Electronics and Information Technology (2023) demonstrated that for small datasets (10-200 elements), Insertion Sort was consistently faster-completing sorting in ±64 s compared to ±82 s for Bubble Sort-even when run in a C++ implementation [8]. This finding supports the theoretical understanding that Insertion Sort, which has less swap overhead and adaptive properties to nearly sorted data, excels under such conditions, while Bubble Sort is generally slower due to the large number of swap operations despite the same asymptotic complexity, i.e. $O(n^2)$ [8].

2.2 Dummy Dataset Creation

In this study, the data taken to be tested is dummy data which is flexible, so researchers may make changes to the data according to needs. There are 3 types of data needed. The first is the numeric data type often used in the Student Identification Number (NIM) data element. The second is the plain text data type (String) used in the student's name data element. The third is the *double* data type which is often used in the Grade Point Average (GPA) data element. The data collected varies in size, ranging from 100, 500 to 1000 and is done with care to ensure that the data represents various dataset sizes. In accordance with other studies that have been conducted, dummy data may be used for the purpose of validating the algorithms and simulations conducted in [14]. The accuracy of the data was maintained while carrying out this process, so that the results of the research on testing the Sorting algorithm were as expected.

As mentioned earlier, there are several different data types that will be used. Table 1 below describes in more detail the data elements or attributes that will be used in the test scenario. This aims to maximize the performance of each algorithm under various conditions.

Table 1. Dummy data attributes

Attribute Name	Data Type	Description
NIM	*Integer*	Unique Attribute
Name	*String*	Student Name
Ipk	*Double*	Student Grade

2.3 Performance Testing Scenario

The performance testing scenario of the Sorting algorithm aims to evaluate its effectiveness and its efficiency in terms of execution time, memory usage, and the number of steps taken [9]. There are 3 factors that will be tested. First, the dataset that will be used in the test scenario consists of numeric and alphabetic variables, with the number of elements being 100, 500, and 1000, respectively. The use of this combination of data types aims to show the difference in algorithm performance depending on data characteristics [5]. In addition, the size of the dataset also allows observation of the scalability of the algorithm as the data load increases. The second is the use of Worst Case, Average Case, and Best Case scenarios to get the desired consistency of results and is done gradually as the number of data elements increases. Worst Case is the condition where the data is arranged so as to maximize the number of operations performed, Average Case is a general condition with random data that represents the typical performance of the algorithm, *Best* Case is the optimal condition where the data is almost sorted to minimize the number of algorithm operations [9]. In this context, it is important to understand the time complexity of the algorithm represented by the notation $\text{Big O}\ (O)$. The algorithm to be tested has the following time complexities:

1. Bubble Sort: The time complexity of Bubble Sort in the worst and average cases is explained by the number of comparisons being almost half the square of the number of elements.

$$T(n) = (N-1) + (N-2) + \cdots + 2 + 1$$
$$= \sum_{i=1}^{N-1} \quad N - i = \tfrac{N(N-1)}{2} \qquad (1)$$
$$T(n) = \frac{N(N-1)}{2} = O(n^2)$$

Thus, the time complexity of Bubble Sort is $O(n2)$.

2. Insertion Sort: Similar to Bubble Sort, Insertion Sort also exhibits quadratic time complexity in the worst and average cases. The total operations of comparing and shifting elements are proportional to the square of the number of elements.

$$T(n) = 1 + 2 + \cdots + n - 1 = \sum_{i=1}^{n-1} \quad i = \frac{n(n-1)}{2} = O(n^2) \ (2)$$

Hence, the time complexity of Insertion Sort is $O(n^2)$.
3. Merge Sort: it is a divide and conquer algorithm that recursively divides the list into two parts, sorts each part, and then merges them. The recurrence relationship is as follows:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad (3)$$

This shows that the time required to sort n elements is twice the time required to sort $\frac{n}{2}$ elements (based on the two sub-lists) plus the time n used for the merge process. Therefore, the time complexity for Merge Sort (both Best, Average, and Worst Case) is $O\ (n \log n)$.

Lastly is the implementation of algorithm testing, which will be done using the *Java* programming language. The Java programming language was chosen because of its capability to accurately measure algorithm performance [15]. The test assessment will be carried out to see the comparison of execution time as a measurement of the complexity of the algorithm's computational speed, then the use of additional memory as a measurement of the algorithm's space efficiency evaluation, and finally the number of steps/operations as a measurement of the algorithm's operational complexity [5].

2.4 Comparison of Test Results

Test results have been conducted by implementing several functions into the Java programming language. The function that will be implemented is to display the execution time required by each algorithm, then the number of steps taken by each algorithm, and finally the amount of space required by each algorithm. The Java function also stores the computational steps required by each algorithm.

The complexity of an algorithm measures the number of calculations required to achieve a certain result. Algorithms that are able to produce the output as expected by the researcher in a short time are categorized as having low complexity. Conversely, algorithms that take longer to produce results are usually considered to have high complexity. Generally, algorithm complexity is expressed asymptotically using *Big-O* notation [15].
If the time complexity to run an algorithm is expressed by $T(n)$, and satisfies $T(n) \leq C\big(f(n)\big)$. For $n\ \geq n_0$ then the complexity can be expressed by $T_{(n)} = O(f(n)$

Space complexity, usually denoted by S(n), is a measure of memory usage by data structures in an algorithm as a function of the input size n. By using a measure of the time or space complexity of the algorithm, we can measure how the space required by the algorithm increases as the input size n increases.

2.5 Analysis of Results

The analysis is done after getting the results from testing each algorithm by considering the time and space complexity. The tested algorithms are Bubble Sort, Insertion Sort, Quick Sort, and Merge Sort algorithms. Tests were conducted on datasets of size 100, 500, and 1000 to evaluate how the algorithm performs as the data size changes. Each algorithm has different time and space complexity consistency.

In the previous discussion, it was found that the Quick Sort and Merge Sort algorithms are superior in execution time speed compared to the Bubble Sort and Insertion Sort algorithms. This is because the time complexity of the Quick Sort and Merge Sort algorithms is $O(n \log n)$, where the execution time will increase as the dataset size grows, but the growth that occurs tends to be slower than algorithms with complexity $O(n^2)$ and $O(n!)$ [16].
In contrast to the superior time complexity comparison results in the Quick Sort and Merge Sort algorithms, the resulting space complexity is that the Bubble Sort and Insertion Sort algorithms are superior to the others. This is because the complexity of the two algorithms is $O(1)$. The space complexity of $O(1)$ is a constant space complexity, where the algorithm only needs space to store temporary variables when sorting [15].

3. Results and Discussion

The calculation results of the Bubble Sort, Insertion Sort, Quick Sort, and Merge Sort algorithms show some significant differences based on execution time, space requirements, the number of steps taken using different devices.

3.1 Testing Device

This research was implemented on 3 computer devices with different specifications. In table 2 there are specifications of each device used. The use of different devices is done in the hope that the test results are obtained maximally and in conditions that suit the user. Each computer uses the Windows 11 operating system for computer programming.

Table 2. Testing device specifications

Device name	Processor	RAM - speed
ASUS ROG	Intel Core i7-8705H	16GB - 2.667 MT/s
ASUS TUF F15	Intel Core i5 Gen 12	16GB - 3,200 MT/s
HP Victus A16	AMD Ryzen 5-8645HS	16GB - 5,600 MT/s

3.2 Bubble Sort

This algorithm is inspired by bubbles that appear on the surface due to the small mass of bubbles with air content. This underlies the working principle of this type of algorithm, where the smallest element of data will be located at the top or bottom depending on the type of ascending or descending sort. This algorithm works by performing repeated traversal (*looping* process) on the elements of the unordered data structure. During the traversal process, the values of two elements in the data structure are compared. If the order of the elements is not as desired, then a swap is performed between the two elements. This algorithm is often referred to as comparison *sort* because it only uses the comparison of element values to organize the order of the elements.

Table 3 Testing results of bubble sort algorithm

Dataset size 100			
Type of testing	**Worse case**	**Average case**	**Best case**
	ASUS ROG		
Time Taken	0.008726 s	0.003670 s	0.004368 s
Memory Used	0.09 KB	0.11 KB	0.09 KB
Steps	10100	5150	5050
ASUS TUF F15			
Time Taken	0.010762 s	0.005706 s	0.005392 s
Memory Used	0.09 KB	0.11 KB	0.09 KB
Steps	10100	5150	5050
HP Victus A16			
Time Taken	0.00434 s	0.001924 s	0.001288 s
Memory Used	0.09 KB	0.11 KB	0.09 KB
Steps	10100	5150	5050
Dataset Size 500			
ASUS ROG			
Time Taken	0.274606 s	0.124706 s	0.123513 s
Memory Used	0.10 KB	0.23 KB	0.21 KB
Steps	251000	125750	125250
ASUS TUF F15			
Time Taken	0.291978 s	0.130679 s	0.120674 s
Memory Used	0.21 KB	0.23 KB	0.21 KB
Steps	250500	125750	125250
HP Victus A16			
Time Taken	0.108963 s	0.049836 s	0.048295 s
Memory Used	0.21 KB	0.23 KB	0.21 KB
Steps	250500	125750	125250
Dataset Size 1000			
ASUS ROG			
Time Taken	2.193209 s	0.972343 s	1.041249 s
Memory Used	0.24 KB	0.26 KB	0.24 KB
Steps	999000	500499	499500
ASUS TUF F15			
Time Taken	1.411694 s	0.732354 s	0.67841 s
Memory Used	0.24 KB	0.26 KB	0.10 KB
Step	999000	500499	499500
HP Victus A16			
Time Taken	0.597781 s	0.271898 s	0.268738 s

Memory Used	0.24 KB	0.26 KB	0.10 KB
Steps	999000	500499	499500

The result obtained from the display in Table 3 is a very high spike in execution time when the dataset size grows to 500. On the growth of the required space is quite constant, on the change of data size from 500 to 1000, the size of the space does not grow quickly, only a difference of 2KB. This is slightly different when the data size changes from 100 to 500 which increases by 2 times. The number of steps taken on each dataset size has a considerable difference, on a change in size from 500 - 1000 it only increases about 5 times. Unlike the case with the change in data size from 100 - 500 which increases up to 10 times more.

The Bubble Sort algorithm can be summarized as, if N is the length of the data structure elements, with the elements being $\mathrm{T1}, \mathrm{T2}, \mathrm{T3}\ \dots, \mathrm{TN}-1, \mathrm{TN}$, then:

1. Perform traversal to compare two adjacent elements. This traversal is done from the back.
2. If the element is $\mathrm{TN}-1\ >\ \mathrm{TN}$, then perform a swap. Otherwise, continue with the next traversal process until it meets the sorted part of the data structure.
3. Repeat the above steps for the remaining data structures.

The Bubble Sort algorithm consists of two nested loops. The first loop lasts for *N-1* times. The index for the first loop is *Pass*. Then the second level of the loop lasts from *N* to *Pass* + 1.

If expanded, the complexity of Bubble Sort algorithm will reach $O(n)$, Therefore, Bubble Sort is considered as the least efficient algorithm among sorting algorithms with complexity $O(n^2)$. At this point, the Bubble Sort algorithm is not recommended for repeated sorting of data or sorting of datasets with a size of more than 200 data elements.

3.3 Insertion Sort

With almost sorted lists, it is simple and efficient enough that some of these points are important factors to recognize what Insertion Sort is. Putting the data in the right place is the method of this algorithm. This algorithm uses a sorting method by inserting data elements in the right position. Insertion Sort algorithm sorting is similar to the concept of a card game, where each card is inserted sequentially from left to right according to the value of the card. The algorithm method has a provision that when a card is inserted into a certain position, the other cards will shift forward or backward according to their value. The algorithm works by taking elements from the list one by one and placing them in the appropriate position. Memory saving is done by applying in-place sorting, which compares the current element with the previous, sorted element and swaps them until they reach the correct position. This process is done repeatedly until all data elements are exhausted.

Table 4. Insertion sort algorithm testing results

Dataset size 100			
Type of testing	**Worse case**	**Average case**	**Best case**
		ASUS ROG	
Time Taken	0.009408 s	0.000084 s	0.000029 s
Memory Used	0.04 KB	0.06 KB	0.05 KB
Steps	10200	300	100
		ASUS TUF F15	
Time Taken	0.013446 s	0.00013 s	0.000073 s
Memory Used	0.04 KB	0.06 KB	0.04 KB
Steps	10200	300	100
		HP Victus A16	
Time Taken	0.003183 s	0.000051 s	0.000012 s
Memory Used	0.04 KB	0.06 KB	0.05 KB
Steps	10200	300	100
		Dataset Size 500	
		ASUS ROG	
Time Taken	0.223776 s	0.001938 s	0.000809 s
Memory Used	0.10 KB	0.12 KB	0.10 KB
Steps	251000	1500	500
		ASUS TUF F15	
Time Taken	0.231136 s	0.002405 s	0.000801 s
Memory Used	0.10 KB	0.12 KB	0.10 KB
Steps	251000	1500	500
		HP Victus A16	
Time Taken	0.093855 s	0.001190 s	0.00044 s

Memory Used	0.10 KB	0.12 KB	0.10 KB
Steps	251000	1500	500
Dataset Size 1000			
ASUS ROG			
Time Taken	1.591603	0.007077	0.002917
Memory Used	0.10	0.12	0.10
Steps	999999	2997	999
ASUS TUF F15			
Time Taken	1.051896 s	0.00453 s	0.001989 s
Memory Used	0.10 KB	0.12 KB	0.10 KB
Step	999999	2997	999
HP Victus A16			
Time Taken	0.453110 s	0.001925 s	0.000944 s
Memory Used	0.10 KB	0.12 KB	0.10 KB
Steps	999999	2997	999

The results obtained from the table 4 display are the time changes in the Worst Case, Average Case, and Best Case test methods. The distance between test methods is almost as large as the time growth required when the data size change occurs. However, similar to the previous algorithm, the growth of the required space develops not as fast as the change in data size from 100 to 500. At small dataset sizes, the number of steps taken is no more than 100,000, but as the data size increases, the steps taken spike to almost 1 million in the worst case test method. The summary of the Insertion Sort Algorithm is as follows:

1. Store the value T, into a temporary variable with i=1.
2. Compare its value with the previous element.
3. If the previous element (T_{i-1}) is greater than T_i, overlap the value of T_i with the value of T_{i-1}. *Decrement* I (reduce the value by 1).
4. Keep doing the third point, until $\mathrm{Ti} - 1 \leq \mathrm{Ti}$
5. If $\mathrm{Ti} - 1 \leq \mathrm{Ti}$ is met, overlap the value in T_i with the temporary variable stored earlier.
6. Repeat the steps from point 1 above with I in - *Increment* (plus 1).

The Insertion Sort algorithm consists of two nested loops. Where there are N-1 Passes (where N is the number of data structure elements), with each Pass occurring *i* comparison operations, *i is* worth 2 for the first Pass, worth 2 for the second Pass and so on until the N-1th Pass.

Insertion Sort shows better performance than Bubble Sort, especially on small and mostly sorted datasets. However, Insertion Sort is less efficient when used on large dataset sizes.

3.4 Merge Sort

The Merge Sort algorithm was first generalized by John von Neumann in 1945. Merge Sort is a data sorting method that applies the divide and conquer principle. It works by dividing the data to be sorted into small groups, where each group consists of a maximum of two elements that are compared and then merged back together. This process proceeds recursively until it reaches the smallest element, i.e. when the sorted part contains only one element, which means it is sorted. This algorithm is based on two main ideas:

1. A small list requires fewer steps to sort than a large list.
2. To form a sorted list from two unordered lists requires fewer steps than forming a sorted list from two unordered lists. For example, when each list is sorted, only one traversal is required.

Table 5. Testing results of merge sort algorithm

Dataset size 100			
Type of testing	**Worse case**	**Average case**	**Best case**
	ASUS ROG		
Time Taken	0.001020 s	0.000896 s	0.001216 s
Memory Used	1.60 KB	1.62 KB	1.60 KB
Steps	680	680	680
ASUS TUF F15			
Time Taken	0.001096 s	0.001315 s	0.001193 s
Memory Used	1.60 KB	1.62 KB	1.60 KB
Steps	680	680	680
HP Victus A16			
Time Taken	0.000320 s	0.000390 s	0.000357 s
Memory Used	1.60 KB	1.62 KB	1.60 KB

Steps	680	680	680
Dataset Size500			
ASUS ROG			
Time Taken	0.006357 s	0.006781 s	0.007598 s
Memory Used	7.84 KB	7.87 KB	8.41 KB
Steps	4498	4498	3761
ASUS TUF F15			
Time Taken	0.008846 s	0.009276 s	0.006937 s
Memory Used	7.84 KB	7.87 KB	7.84 KB
Steps	4498	4498	4498
HP Victus A16			
Time Taken	0.002896 s	0.002808 s	0.003191 s
Memory Used	7.84 KB	7.87 KB	7.84 KB
Steps	4498	4498	4498
Dataset Size 1000			
ASUS ROG			
Time Taken	0.025275 s	0.040523 s	0.023921 s
Memory Used	15.66 KB	15.69 KB	15.66 KB
Steps	9976	9976	9976
ASUS TUF F15			
Time Taken	0.0242 s	0.024838 s	0.025238 s
Memory Used	15.66 KB	15.69 KB	15.66 KB
Step	9976	9976	9976
HP Victus A16			
Time Taken	0.007110 s	0.008855 s	0.008841 s
Memory Used	15.66 KB	15.69 KB	15.66 KB
Steps	9976	9976	9976

The result obtained from the display of table 5 is the growth of execution time which is not as fast as the previous 2 algorithms. The execution time required by this algorithm is quite consistent when looking at various data changes. However, the growth of storage space required increases rapidly, along with the growth of dataset size. The number of steps taken by this algorithm changes significantly, according to the size of the data. The Merge Sort algorithm can be written simply, as follows:

1. Divide the unordered list into two equal lengths or one of which is one element longer.
2. Divide each of the 2 sub-lists recursively until we get a list of size 1.
3. Merge the 2 sub-lists back into one sorted list.

For sorting with n elements, Merge Sort has a time complexity of $O(n \log n)$, which applies to both the average case and the worst case. If the execution time of Merge Sort for data with nnn elements is expressed as T(n), then the recursive relation can be written as $T(n) = 2T\left(\frac{n}{2}\right) + n$ according to the algorithm definition. This algorithm is recursive, so it is inefficient if run on computer devices that have limited memory space.

3.5 Quick Sort

The algorithm known as Quick Sort was designed in 1960 by Hoare while he was working at Elliot Brothers which was a small computer manufacturing company, Elliot Brothers. Quick Sort, also known as partition exchange Sort, belongs to the category of divide and conquer algorithms. It shows good "average behavior" performance compared to other methods, as the sorting process is done by dividing the data into partitions and sorting each partition. This method divides the data table to be sorted into two subsections, which are then searched from the left and right sides.

Table 6. Quick sort algorithm testing results

Dataset size 100			
Type of testing	**Worse case**	**Average case**	**Best case**
	ASUS ROG		
Time Taken	0.000586 s	0.000868 s	0.000685 s
Memory Used	1.94 KB	2.03 KB	1.91 KB
Steps	549	551	549
ASUS TUF F15			
Time Taken	0.002781 s	0.001204 s	0.000959 s
Memory Used	1.94 KB	2.03 KB	1.91 KB
Steps	549	551	549

HP Victus A16			
Time Taken	0.000242 s	0.000344 s	0.000247 s
Memory Used	1.94 KB	2.03 KB	1.91 KB
Steps	549	551	549
Dataset Size 500			
ASUS ROG			
Time Taken	0.002387 s	0.002108 s	0.002125 s
Memory Used	8.44 KB	8.53 KB	8.41 KB
Steps	3761	3763	3761
ASUS TUF F15			
Time Taken	0.002816 s	0.003595 s	0.006937 s
Memory Used	8.44 KB	8.53 KB	8.41 KB
Steps	3761	3763	3761
HP Victus A16			
Time Taken	0.001038 s	0.001091 s	0.001189 s
Memory Used	8.44 KB	8.53 KB	8.41 KB
Steps	3761	3763	3761
Dataset Size 1000			
ASUS ROG			
Time Taken	0.007098 s	0.007553 s	0.006556 s
Memory Used	16.59 KB	16.66 KB	16.57 KB
Steps	8498	8500	8498
ASUS TUF F15			
Time Taken	0.004946 s	0.005822 s	0.00594 s
Memory Used	16.59 KB	16.66 KB	16.57 KB
Step	8498	8500	8498
HP Victus A16			
Time Taken	0.002279 s	0.002214 s	0.002585 s
Memory Used	16.59 KB	16.66 KB	16.57 KB
Steps	8498	8500	8498

Referring to Table 6, this algorithm has a fairly rapid growth in the space required as the dataset size grows, even greater than the previous algorithm. But for the execution time obtained, this algorithm is faster than other algorithms under any conditions. Even in each test method, the execution time of this algorithm does not experience a high spike. The number of steps taken by this algorithm experienced a slight shrinkage in the change of large data size. Even in all test methods, there is no significant difference in the change. This algorithm consists of 4 main steps:

1. If the data structure consists of 1 or 0 elements to be sorted, return the data structure as it is.
2. Take an element that will be used as the pivot point, usually the leftmost element.
3. Split the data structure into two parts, one with elements larger than the pivot point, and the other with elements smaller than the pivot point.
4. Repeat the algorithm recursively on both halves of the data structure.

The efficiency of this algorithm is highly dependent on the selection of elements as pivots. In the worst case, the complexity can reach $O(n^2)$, which usually happens when the data is already sorted and the first element is chosen as the pivot. To avoid such problems, it is recommended to select pivots randomly, especially if the data to be sorted is not random. With random pivot selection, quick sort can achieve complexity of $O\ (n \log n\ )$.

Based on three tests conducted on three different laptop configurations, it can be concluded that each sorting algorithm (such as Bubble Sort, Insertion Sort, Merge Sort, and Quick Sort) shows very different performance characteristics, especially in terms of execution time the most superior algorithm is the Insertion Sort algorithm, for memory usage the most superior is the Bubble Sort algorithm, and the most superior number of operational steps is the Quick Sort algorithm. In the calculation of the number of datasets, Insertion Sort algorithm is most superior in small datasets and Quick Sort is most superior in large datasets.

4. Conclusion and Suggestion

Bubble Sort and Insertion Sort have complexity algorithms $O(n^2)$, which states that the execution time will increase significantly as the dataset size increases. Of the two algorithms, Insertion Sort is more efficient than Bubble Sort, because the number of operations is more minimal especially on small datasets and almost sorted data. Next are Quick Sort and Merge Sort algorithms that have time complexity $O\ (\ n \log n\ )$, with guaranteed algorithm performance consistency. Among the two algorithms, Quick Sort is superior because it does not require additional storage space. In contrast, the Merge Sort algorithm requires additional storage space to store a copy of the data during the computation process, which can be an obstacle for devices that have limited memory.

The test results show that hardware also has a role in algorithm execution time. In this test, the HP Victus computer device with an AMD Ryzen 5 Processor was able to provide a shorter execution time compared to other devices. Even so, the use of storage space and the number of steps taken by the algorithm remained the same as other devices. In line with the statement regarding device usage, the analysis obtained on the performance of all algorithms shows that Merge Sort is the most efficient among all algorithms. The Merge Sort algorithm on the various scenarios tested has a stable work consistency on time complexity $O\ (n \log n\ )$.

The use of an appropriate sorting algorithm is highly dependent on the dataset characteristics and performance priorities. To sort a small dataset, the Insertion Sort algorithm is better. To get better execution time consistency performance, Quick Sort algorithm could be used. But, in the other hand, this algorithm is less efficient on nearly sorted data because it relies heavily on pivot selection. To have stability and guaranteed performance in all case scenarios, Merge Sort algorithm is best. But, in the other hand, this algorithm requires additional storage space. Lastly, Bubble Sort has less efficient performance on large datasets and tends to have slow execution time.

References

[1] W. Krisna, H. J. Muhammad, and N. Ambadar, “Rancang Bangun Sistem Informasi Akademik Menggunakan Framework Codeigniter Pada universitas Muhammadiyah Purworejo,” *Jurnal Sistem Cerdas*, vol. 5, no. 2, pp. 107–116, 2022, doi: 10.37396/jsc.v5i2.187.

[2] H. Turnip, “Penggunaan Big Data untuk Optimalisasi Pengambilan Keputusan di Sekolah The Use of Big Data to Optimize Decision Making in Schools,” vol. 7, no. 8, pp. 3138–3145, 2024, doi: 10.56338/jks.v7i8.5971.

[3] M. R. Zikri, “Performance Analysis of Sorting Algorithms in Big Data Environments: Efficiency, Scalability, and Practical Applications”, Idea, vol. 1, no. 3, pp. 132–139, Oct. 2023.

[4] 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.

[5] M. N. Ilham, A. F. Setiawan, I. Kholifatun, M. H. Aldiansyah, and I. P. Pujiono, “Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C++ Programming Language”, j. of artif. intell. and eng. appl., vol. 4, no. 3, pp. 1950–1956, Jun. 2025.

[6] F. A. Hia, “Analisis Perbandingan Algoritma Pengurutan (Sorting) Berdasarkan Kompleksitas Waktu,” *Makalah IF2120 Matematika Diskrit*, 2022.

[7] M. B. Sena *et al.*, “Perbandingan kinerja algoritma sorting dalam pengurutan data menggunakan bahasa python,” vol. 1, no. 2, pp. 310–314, 2024.

[8] D. Vierdansyah, G. Al Ghafiqi, M. D. Putra, and D. Pradeka, "Comparison analysis of bubble sort and insertion sort algorithm on the selection of a shop according to the criteria," J. Comput. Eng. Electron. Inf. Technol., vol. 2, no. 1, pp. 39–52, 2023, doi: 10.17509/coelite.v2i1.57091.

[9] Z. . Mamatnabiyev and M. . Zhaparov, “COMPARATIVE ANALYSIS OF SORTING ALGORITHMS USED INCOMPETITIVE PROGRAMMING”, JETC, vol. 55, no. 2, pp. 64–70, Oct. 2024, doi: 10.47344/sdubnts.v55i2.546.

[10] D. Anggreani, A. P. Wibawa, P. Purnawansyah, and H. Herman, “Perbandingan Efisiensi Algoritma Sorting dalam Penggunaan Bandwidth,” *ILKOM Jurnal Ilmiah*, vol. 12, no. 2, pp. 96–103, 2020, doi: 10.33096/ilkom.v12i2.538.96-103.

[11] S. Ibnu, “Metodologi Penelitian,” *Widina Bhakti Persada Bandung*, pp. 12–26, 2022.

[12] M. Yusuf, M. Sodik, S. Darussalam, K. Nganjuk, and U. Blitar, “Penggunaan Teknologi Internet of Things (Iot) Dalam Pengelolaan Fasilitas Dan Infrastruktur Lembaga Pendidikan Islam,” *PROPHETIK Jurnal Kajian Keislaman*, vol. 1, no. 2, pp. 1–18, 2023.

[13] A. Isyqi *et al.*, “Perbandingan kinerja algoritma bubble sort, insertion sort, dan selection sort menggunakan data bilangan numerik pada program python,” vol. 1, no. 2, pp. 266–271, 2024.

[14] S. Mondal, A. Basu, and N. Mukherjee, “Building a trust-based doctor recommendation system on top of multilayer graph database,” *J Biomed Inform*, vol. 110, Oct. 2020, doi: 10.1016/j.jbi.2020.103549.

[15] W. F. Wisudawan, “Kompleksitas Algoritma Sorting yang Populer Dipakai,” pp. 1–8, 2020.

[16] I. Mishal, R. AL-Khatib, and R. Hiasat, "Comparative study of two divide and conquer sorting algorithms: Modified quick sort and merge sort," Int. J. Comput. Appl., vol. 183, no. 31, pp. 28–33, Oct. 2021, doi: 10.5120/ijca2021921702.