JURNAL TEKNIK INFORMATIKA DAN MULTIMEDIA Vol 6, No.1, Mei 2026, pp. 367 - 379
p-ISSN : 2808-909X (print)
e-ISSN : 2798-2505 (online)
http://journal.politeknik-pratama.ac.id/index.php/JTIM · page 367
DOI : https://doi.org/10.51903/informatika.v6i1.1661

ANALISIS PERFORMA ALGORITMA QUICK SORT DAN MERGE SORT PADA PENGURUTAN DATA BESAR (BIG DATA) MENGGUNAKAN NOTASI BIG-O

Calvin Syahputra[1*], Syti Salwaa Nafiisah[2], Shaqila Rahmayani Gultom[3], Ridho Affandi[4], Adidtya Perdana[5]
[1,2,3,4,5] Program Studi Ilmu Komputer, Fakultas Matematika Dan Ilmu Pengetahuan Alam, Universitas Negeri Medan
Jl. Willem Iskandar Pasar V Medan Estate, Kec. Percut Sei Tuan, Kab. Deli Serdang, Sumatera Utara.
e-mail: calvinsahputra4@gmail.com, sytisalwaanafiisah@gmail.com, saqilarahmayani269@gmail.com, ridhoaffandi997@gmail.com, adidtya@unimed.ac.id

ARTICLE INFO

Article history:

Received 10 December 2026
Received in revised form 24 January 2026
Accepted 8 Mei 2026
Available online 15 Mei 2026

ABSTRACT

The rapid development of information technology has led to an exponential increase in data volume, requiring efficient and high-performance sorting algorithms. Sorting is one of the fundamental operations in large-scale data processing. This study aims to analyze the performance of the Quick Sort and Merge Sort algorithms in sorting large datasets based on time complexity using Big-O notation. The research method employed is experimental, by implementing both algorithms on datasets of various sizes, then measuring execution time and analyzing their time complexity under best-case, average-case, and worst-case conditions. The results show that Quick Sort performs faster on average with a time complexity of O(n log n), but its performance can degrade to $O(n^2)$ in the worst case. Meanwhile, Merge Sort demonstrates more stable performance with a time complexity of O(n log n) in all cases, although it requires additional memory usage. Based on these findings, the selection of sorting algorithms for large-scale data

Received December 10, 2025; Accepted Mei 8, 2026; Available online Mei 15, 2026

p-ISSN : 2808-909X e-ISSN : 2798-2505

should consider data characteristics and memory requirements to achieve optimal performance.

Keywords: Quick Sort, Merge Sort, Big Data, Time Complexity, Big-O Notation.

Abstrak

Perkembangan teknologi informasi menyebabkan volume data yang diolah semakin besar sehingga dibutuhkan algoritma pengurutan yang efisien dan memiliki performa tinggi. Proses pengurutan data (sorting) merupakan salah satu operasi fundamental dalam pengolahan data berskala besar (big data). Penelitian ini bertujuan untuk menganalisis performa algoritma Quick Sort dan Merge Sort dalam mengurutkan data berukuran besar berdasarkan kompleksitas waktu menggunakan notasi Big-O. Metode penelitian yang digunakan adalah eksperimen dengan mengimplementasikan kedua algoritma pada berbagai ukuran data acak, kemudian mengukur waktu eksekusi serta menganalisis kompleksitas waktunya pada kondisi terbaik, rata-rata, dan terburuk. Hasil analisis menunjukkan bahwa Quick Sort memiliki performa yang lebih cepat pada kondisi rata-rata dengan kompleksitas waktu O(n log n), namun dapat mengalami penurunan performa hingga $O(n^2)$ pada kondisi terburuk. Sementara itu, Merge Sort menunjukkan performa yang lebih stabil dengan kompleksitas waktu O(n log n) pada semua kondisi, meskipun membutuhkan penggunaan memori tambahan. Berdasarkan hasil tersebut, dapat disimpulkan bahwa pemilihan algoritma pengurutan untuk data berukuran besar perlu mempertimbangkan karakteristik data dan kebutuhan memori agar diperoleh performa yang optimal.
Kata Kunci: Quick Sort, Merge Sort, Big Data, Kompleksitas Waktu, Big-O Notation.

1. PENDAHULUAN

Perkembangan teknologi digital dalam beberapa tahun terakhir telah menyebabkan peningkatan jumlah data yang dihasilkan oleh berbagai sistem informasi [1]. Data tersebut berasal dari berbagai sumber seperti media sosial, transaksi elektronik, sensor, serta aplikasi berbasis internet. Kondisi ini dikenal sebagai Big Data, yaitu fenomena meningkatnya volume, kecepatan, dan keberagaman data yang dihasilkan secara terus-menerus. Dalam konteks ini, kemampuan untuk mengelola serta memproses data secara efisien menjadi aspek penting dalam sistem komputasi modern. Salah satu proses dasar yang banyak digunakan dalam pengolahan data adalah sorting atau pengurutan data, yaitu proses menyusun data ke dalam urutan tertentu sehingga memudahkan proses pencarian, analisis, maupun pengolahan data selanjutnya [2]

Dalam ilmu komputer, proses pengurutan data merupakan salah satu operasi fundamental yang sering diterapkan pada berbagai sistem komputasi seperti sistem basis data, data mining, serta pengolahan data berskala besar. Data yang telah diurutkan akan lebih mudah diproses dalam berbagai operasi komputasi seperti pencarian data dan analisis informasi. Oleh karena itu, efisiensi algoritma sorting memiliki peranan penting dalam menentukan kinerja suatu sistem, terutama ketika sistem tersebut harus memproses data

dalam jumlah besar. Beberapa penelitian menunjukkan bahwa pemilihan algoritma pengurutan yang tepat dapat membantu meningkatkan efisiensi proses komputasi dan mengurangi waktu eksekusi secara signifikan [3], [4]

Seiring dengan perkembangan penelitian di bidang algoritma, berbagai metode sorting telah dikembangkan untuk meningkatkan efisiensi proses pengurutan data. Algoritma sederhana seperti Bubble Sort atau Selection Sort memiliki kompleksitas waktu yang cukup tinggi yaitu $O(n^2)$ sehingga kurang efisien ketika digunakan untuk memproses dataset yang berukuran besar [5]. Sebaliknya, algoritma seperti Quick Sort dan Merge Sort memiliki kompleksitas waktu yang lebih rendah yaitu O(n log n) sehingga lebih banyak digunakan dalam berbagai aplikasi komputasi modern yang memerlukan pengolahan data dalam skala besar [6].

Quick Sort dan Merge Sort merupakan algoritma yang dirancang menggunakan pendekatan divide and conquer, yaitu metode yang memecah suatu permasalahan besar menjadi beberapa sub-permasalahan yang lebih kecil, kemudian menyelesaikan setiap bagian secara terpisah sebelum digabungkan kembali menjadi solusi akhir [7]. Pendekatan ini terbukti mampu meningkatkan efisiensi proses pengolahan data karena memungkinkan proses komputasi dilakukan secara lebih terstruktur dan sistematis dibandingkan metode pengurutan sederhana [8].

Quick Sort dikenal sebagai salah satu algoritma pengurutan yang memiliki performa tinggi pada berbagai kondisi dataset. Algoritma ini bekerja dengan memilih suatu elemen sebagai pivot yang digunakan untuk membagi data menjadi dua bagian sebelum proses pengurutan dilakukan secara rekursif[9]. Dalam kondisi rata-rata, Quick Sort memiliki kompleksitas waktu O(n log n). Namun demikian, algoritma ini dapat mengalami penurunan performa pada kondisi tertentu sehingga kompleksitas waktu terburuknya dapat mencapai $O(n^2)$ apabila pemilihan pivot tidak dilakukan secara optimal [10].

Sementara itu, Merge Sort merupakan algoritma pengurutan yang memiliki kompleksitas waktu O(n log n) pada hampir semua kondisi, baik pada kasus terbaik, rata-rata, maupun terburuk. Keunggulan utama dari algoritma ini adalah kestabilannya dalam mempertahankan urutan elemen yang memiliki nilai yang sama selama proses pengurutan berlangsung [11]. Namun demikian, Merge Sort memerlukan ruang memori tambahan untuk menyimpan hasil penggabungan data sementara, sehingga penggunaannya perlu mempertimbangkan kapasitas memori yang tersedia dalam sistem [12].

Menurut penelitian sebelumnya [13] mengenai optimasi algoritma sorting juga dilakukan dengan memanfaatkan teknologi komputasi modern seperti komputasi paralel dan sistem terdistribusi. Pendekatan ini memungkinkan proses pengurutan data dilakukan secara simultan menggunakan beberapa prosesor sehingga dapat meningkatkan efisiensi pengolahan dataset yang sangat besar. Penelitian [14] menunjukkan bahwa penggunaan metode paralel dalam proses sorting mampu meningkatkan performa sistem secara signifikan terutama dalam lingkungan komputasi berskala besar.

Selain itu, Penelitian [15] juga mulai mengeksplorasi penggunaan teknologi kecerdasan buatan dalam pengembangan algoritma sorting yang lebih adaptif terhadap karakteristik dataset. Integrasi metode pembelajaran mesin memungkinkan sistem untuk menyesuaikan strategi pengurutan data berdasarkan pola data yang diproses sehingga dapat meningkatkan efisiensi komputasi dalam berbagai kondisi dataset [16]

Berdasarkan berbagai penelitian sebelumnya, dapat disimpulkan bahwa pemilihan algoritma sorting yang tepat sangat dipengaruhi oleh ukuran dataset, karakteristik data, serta kebutuhan sistem yang digunakan. Meskipun berbagai penelitian telah membahas performa berbagai algoritma sorting, analisis yang secara khusus membandingkan

p-ISSN : 2808-909X e-ISSN : 2798-2505

performa Quick Sort dan Merge Sort pada pengolahan data berskala besar dengan pendekatan analisis kompleksitas algoritma menggunakan notasi Big-O masih perlu dilakukan. Oleh karena itu, penelitian ini bertujuan untuk menganalisis performa algoritma Quick Sort dan Merge Sort dalam proses pengurutan data besar guna mengetahui karakteristik efisiensi masing-masing algoritma serta menentukan algoritma yang lebih efektif digunakan dalam pengolahan data berskala besar.

2. TINJAUAN PUSTAKA

2.1 Konsep Dasar Algoritma Sorting

Sorting merupakan salah satu proses dasar dalam ilmu komputer yang digunakan untuk menyusun sekumpulan data ke dalam urutan tertentu, baik secara menaik (ascending) maupun menurun (descending) [17]. Proses pengurutan data ini bertujuan untuk mempermudah berbagai operasi komputasi seperti pencarian, pengelompokan, serta analisis data. Dalam berbagai sistem informasi, data yang telah diurutkan akan lebih mudah diproses sehingga efisiensi algoritma sorting menjadi faktor penting dalam meningkatkan kinerja sistem komputasi [18]. Algoritma sorting memiliki berbagai pendekatan yang berbeda dalam melakukan proses pengurutan data. Setiap algoritma memiliki karakteristik yang berbeda dari segi waktu eksekusi, penggunaan memori, serta cara kerja algoritma tersebut dalam memproses dataset[19]. Oleh karena itu, pemilihan algoritma sorting yang tepat perlu mempertimbangkan beberapa faktor seperti ukuran data, struktur data, serta kebutuhan sistem dalam mengolah informasi. Dalam perkembangan penelitian saat ini, berbagai studi terus dilakukan untuk mengevaluasi performa algoritma sorting agar mampu mengolah data dalam jumlah besar secara lebih efisien [20].

2.2 Algoritma Quick Sort

Quick Sort merupakan salah satu algoritma pengurutan yang banyak digunakan dalam bidang komputasi karena efisiensinya dalam menangani berbagai ukuran dataset [21]. Quick Sort bekerja dengan memilih sebuah elemen sebagai pivot yang digunakan untuk membagi data menjadi dua kelompok, yaitu elemen yang lebih kecil dan elemen yang lebih besar dari pivot. Proses ini kemudian dilakukan secara rekursif hingga seluruh data tersusun secara terurut [22].Dari sisi kompleksitas, Quick Sort memiliki waktu eksekusi rata-rata sebesar O(n log n) sehingga sering digunakan dalam berbagai aplikasi pengolahan data. Namun, pada kondisi tertentu kompleksitasnya dapat meningkat hingga $O(n^2)$ apabila pemilihan pivot tidak optimal. Meskipun demikian, Quick Sort tetap menjadi salah satu algoritma sorting yang efisien dan banyak digunakan dalam berbagai sistem komputasi modern [23].

2.3 Algoritma Merge Sort

Merge Sort merupakan algoritma pengurutan yang juga menggunakan pendekatan divide and conquer**.** Algoritma ini bekerja dengan cara membagi kumpulan data menjadi beberapa bagian yang lebih kecil hingga setiap bagian hanya terdiri dari satu elemen[24]. Setelah itu, bagian-bagian tersebut akan digabungkan kembali secara bertahap sambil menyusun elemen-elemen tersebut ke dalam urutan yang benar. Pendekatan ini membuat proses pengurutan menjadi lebih sistematis karena data diurutkan selama proses penggabungan berlangsung [25]. Dari sisi analisis algoritma, Merge Sort memiliki kompleksitas waktu O(n log n) baik pada kondisi terbaik, rata-rata, maupun terburuk. Hal ini menjadikan Merge Sort dikenal sebagai algoritma yang stabil dalam proses pengurutan data. Namun, algoritma ini membutuhkan ruang

p-ISSN : 2808-909X e-ISSN : 2798-2505

memori tambahan selama proses penggabungan data sehingga penggunaan Merge Sort biasanya mempertimbangkan kapasitas memori yang tersedia dalam sistem [26].

2.4 Kompleksitas Algoritma dan Notasi Big-O

Kompleksitas algoritma digunakan untuk mengukur tingkat efisiensi suatu algoritma dalam memproses data. Analisis ini dilakukan dengan melihat jumlah operasi yang dibutuhkan oleh algoritma ketika ukuran data semakin besar. Dengan memahami kompleksitas algoritma, peneliti dapat menilai kinerja suatu algoritma serta membandingkannya dengan algoritma lainnya [27]. Salah satu cara yang umum digunakan untuk menganalisis kompleksitas algoritma adalah notasi Big-O**.** Notasi ini menggambarkan hubungan antara pertumbuhan waktu eksekusi algoritma dengan ukuran input data. Dalam penelitian mengenai algoritma sorting, notasi Big-O sering digunakan untuk membandingkan efisiensi algoritma seperti Quick Sort dan Merge Sort dalam mengolah data berukuran besar [28].

2.5 Analisis Performa Algoritma Sorting

Performa algoritma sorting dapat dianalisis berdasarkan beberapa aspek, seperti waktu eksekusi, jumlah operasi yang dilakukan, serta penggunaan sumber daya komputasi selama proses pengurutan data. Analisis ini penting untuk mengetahui seberapa efisien suatu algoritma ketika digunakan untuk memproses dataset dengan ukuran yang berbeda [29]. Dengan melakukan evaluasi performa algoritma, peneliti dapat menentukan metode pengurutan yang lebih efektif dalam berbagai kondisi pengolahan data. Dalam konteks pengolahan data berskala besar, pemilihan algoritma sorting yang tepat menjadi faktor penting dalam meningkatkan efisiensi sistem. Beberapa penelitian menunjukkan bahwa algoritma dengan kompleksitas yang lebih rendah cenderung memberikan performa yang lebih baik ketika digunakan pada dataset berukuran besar. Oleh karena itu, analisis performa algoritma seperti Quick Sort dan Merge Sort menjadi penting untuk mengetahui karakteristik efisiensi masing-masing algoritma dalam proses pengurutan data [30].

3. METODOLOGI PENELITIAN

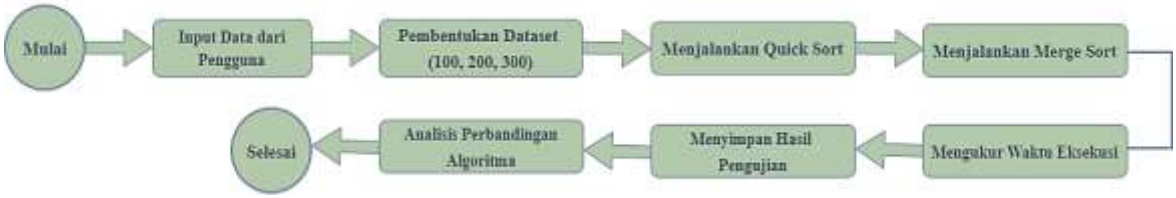
3.1 Jenis Penelitian

Penelitian ini merupakan penelitian kuantitatif eksperimental komparatif. Pendekatan kuantitatif digunakan karena penelitian berfokus pada pengukuran numerik terhadap waktu eksekusi algoritma pengurutan. Metode eksperimen dilakukan dengan menguji secara langsung dua algoritma pengurutan, yaitu Quick Sort dan Merge Sort, menggunakan dataset yang berasal dari input pengguna. Penelitian ini juga bersifat komparatif, karena bertujuan untuk membandingkan performa kedua algoritma berdasarkan waktu eksekusi yang dihasilkan pada ukuran dataset yang berbeda.

3.2 Desain Penelitian

Desain penelitian menggunakan metode eksperimen terkontrol, di mana kedua algoritma diuji dalam kondisi lingkungan komputasi yang sama menggunakan bahasa pemrograman Python.

p-ISSN : 2808-909X e-ISSN : 2798-2505



Gambar 1 Flowchart Tahapan Penelitian

Ukuran dataset yang digunakan dalam penelitian ini ditunjukkan pada Tabel 1

Tabel 1 Variasi Ukuran Dataset

No	Ukuran Dataset (n)
1	100
2	200
3	300

Dataset tersebut diperoleh dari data yang dimasukkan pengguna dengan mengambil sebagian elemen data menggunakan teknik data slicing pada Python.

3.3 Sumber dan Jenis Data

Data yang digunakan dalam penelitian ini berasal dari input pengguna yang dimasukkan melalui terminal atau konsol program. Pengguna diminta memasukkan minimal 300 angka yang dipisahkan menggunakan spasi.

Setelah data dimasukkan, program akan mengonversi data tersebut menjadi array bilangan bulat menggunakan fungsi map() dan split() pada Python.

Selanjutnya, data tersebut dibagi menjadi tiga kelompok dataset menggunakan teknik slicing array, yaitu :

- Dataset 100 elemen → data[:100]
- Dataset 200 elemen → data[:200]
- Dataset 300 elemen → data[:300]

Pembagian dataset ini bertujuan untuk mengamati perubahan performa algoritma ketika ukuran data meningkat.

3.4 Lingkungan dan Tools Pengujian

Penelitian ini menggunakan beberapa perangkat lunak dan library Python sebagai berikut.

Tabel 2 Lingkungan Pengujian

Komponen	Keterangan
Bahasa Pemrograman	Python
Library Pengukuran Waktu	time
Library Visualisasi	matplotlib
Platform Eksekusi	Google Colab / Python Environment
Metode Pengukuran	time.perf_counter()

Fungsi time.perf_counter() digunakan karena memiliki tingkat presisi yang tinggi dalam mengukur waktu eksekusi program.

3.5 Implementasi Algoritma

Penelitian ini mengimplementasikan dua algoritma pengurutan, yaitu Quick Sort dan Merge Sort.

p-ISSN : 2808-909X e-ISSN : 2798-2505

1. Quick Sort

Quick Sort merupakan algoritma pengurutan yang menggunakan metode divide and conquer dengan memilih sebuah elemen sebagai pivot. Data kemudian dibagi menjadi tiga bagian, yaitu elemen yang lebih kecil dari pivot, sama dengan pivot, dan lebih besar dari pivot.

Pada penelitian ini, pivot dipilih dari elemen tengah array.

2. Merge Sort

Merge Sort juga menggunakan pendekatan divide and conquer. Algoritma ini bekerja dengan membagi array menjadi dua bagian secara rekursif hingga tersisa satu elemen, kemudian menggabungkannya kembali dalam keadaan terurut menggunakan proses merge.

3.6 Prosedur Pengujian

Prosedur pengujian dalam penelitian ini dilakukan melalui langkah-langkah berikut :

1. Mengimplementasikan algoritma Quick Sort dan Merge Sort dalam bahasa Python.
2. Meminta pengguna memasukkan minimal 300 angka yang dipisahkan dengan spasi.
3. Mengonversi input pengguna menjadi array bilangan bulat.
4. Membagi dataset menjadi tiga ukuran data yaitu 100, 200, dan 300 elemen.
5. Menjalankan algoritma Quick Sort pada setiap dataset.
6. Menjalankan algoritma Merge Sort pada dataset yang sama.
7. Mengukur waktu eksekusi menggunakan fungsi time.perf_counter().
8. Menyimpan hasil pengukuran waktu ke dalam tabel hasil pengujian.
9. Menampilkan hasil pengujian pada layar.
10. Membuat grafik perbandingan performa menggunakan library matplotlib.

3.7 Teknik Analisis Data

Analisis data dilakukan dengan membandingkan waktu eksekusi algoritma Quick Sort dan Merge Sort pada setiap ukuran dataset.

Hasil pengujian disajikan dalam dua bentuk, yaitu :

1. Tabel Hasil Pengujian

Tabel ini menunjukkan waktu eksekusi masing-masing algoritma pada setiap ukuran dataset.

Tabel 3 Format Hasil Pengujian

Ukuran Dataset	Quick Sort (detik)	Merge Sort (detik)
100	x.xxx	x.xxx
200	x.xxx	x.xxx
300	x.xxx	x.xxx

2. Grafik Perbandingan Algoritma

Selain tabel, hasil pengujian juga divisualisasikan dalam bentuk grafik garis (line chart) menggunakan library matplotlib.

Struktur grafik adalah sebagai berikut:

- **Sumbu X** : Ukuran dataset (100, 200, 300)
- **Sumbu Y** : Waktu eksekusi algoritma (detik)

Grafik ini digunakan untuk melihat perbandingan performa dan pola pertumbuhan waktu eksekusi kedua algoritma.

p-ISSN : 2808-909X e-ISSN : 2798-2505

3.8 Kriteria Evaluasi

Penelitian ini menggunakan beberapa indikator evaluasi untuk menilai performa algoritma, yaitu :

1. Waktu eksekusi algoritma (running time).
2. Perbandingan kecepatan Quick Sort dan Merge Sort.
3. Pola peningkatan waktu eksekusi terhadap ukuran dataset.
4. Algoritma yang memiliki waktu eksekusi paling cepat pada setiap ukuran dataset.

Melalui metode ini, dapat diketahui algoritma pengurutan mana yang lebih efisien dalam mengolah dataset dengan ukuran yang berbeda.

4. HASIL DAN PEMBAHASAN

4.1 Proses Input dan Output Program

Program yang dibuat dalam penelitian ini menerima input data berupa sekumpulan angka dari pengguna. Data yang dimasukkan digunakan sebagai dataset untuk diuji menggunakan algoritma Quick Sort dan Merge Sort. Program mengharuskan pengguna memasukkan minimal 300 angka yang dipisahkan dengan spasi.

Contoh proses input data pada program ditunjukkan sebagai berikut:

Masukkan minimal 300 angka (pisahkan dengan spasi):

Input data :

87 23 45 12 78 34 90 56 11 67 43 21 89 54 32 76 10 98 65 44 73 29 60 41 85 19 92
36 58 14 63 27 71 52 39 84 16 95 48 30 70 22 57 13 88 33 64 49 91 26 62 37 80 17
53 24 75 42 96 18 68 28 74 35 82 20 59 46 94 15 66 31 81 25 72 40 97 38 69 47 83
34 77 50 61 21 93 44 55 12 79 32 86 19 67 23 91 41 73 28 60 14 84 37 95 18 52 26
70 48 62 35 80 17 74 29 90 45 58 16 71 33 85 20 63 27 94 39 76 24 88 42 53 30 97
21 65 36 81 19 72 28 59 46 92 15 68 31 83 25 64 40 75 22 87 34 91 47 60 13 78 29
86 41 93 18 69 26 82 35 74 20 95 38 57 24 80 33 71 16 84 45 62 27 96 39 73 21 88
31 70 14 52 36 79 23 91 42 65 28 83 17 94 34 68 26 77 49 60 12 72 30 85 41 97 19
63 24 89 35 75 28 90 37 54 16 82 43 66 21

Apabila jumlah data yang dimasukkan kurang dari 300 angka, maka program akan memberikan peringatan kepada pengguna bahwa data belum memenuhi jumlah minimum yang ditentukan.

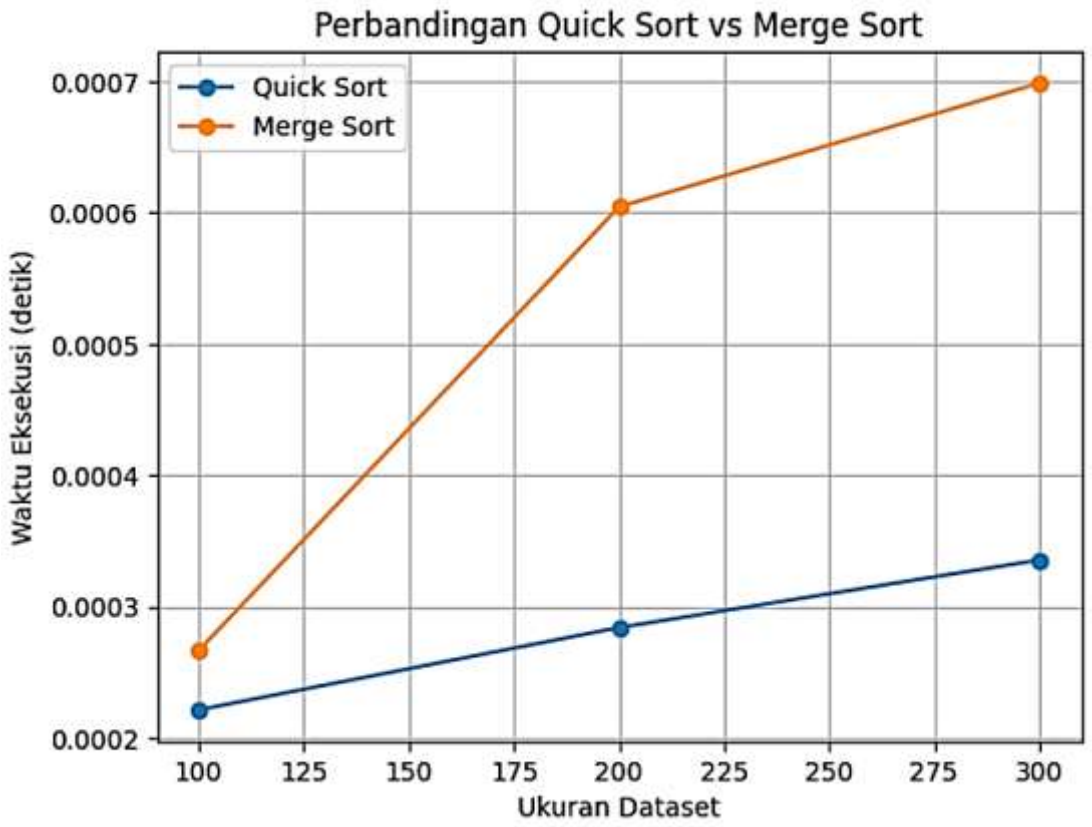
Setelah data dimasukkan dan diproses oleh program, sistem akan menampilkan hasil pengujian waktu eksekusi kedua algoritma.

```
HASIL PENGUJIAN
Size    QuickSort   MergeSort
100 0.000221    0.000267
200 0.000284    0.000604
300 0.000336    0.000699

=== ANALISIS ===
Dataset 100 data → Algoritma lebih cepat: Quick Sort
Dataset 200 data → Algoritma lebih cepat: Quick Sort
Dataset 300 data → Algoritma lebih cepat: Quick Sort
```

Gambar 2 Tampilan Output Program Pengujian

p-ISSN : 2808-909X e-ISSN : 2798-2505



Gambar 3 Grafik Perbandingan Waktu Eksekusi Quick dan Merge Sort

Output tersebut menunjukkan waktu eksekusi dari masing-masing algoritma pada ukuran dataset yang berbeda serta memberikan informasi algoritma mana yang memiliki performa lebih cepat pada setiap pengujian.

4.2 Hasil Pengujian Algoritma

Pengujian dilakukan untuk membandingkan performa algoritma Quick Sort dan Merge Sort berdasarkan waktu eksekusi pada beberapa ukuran dataset. Dataset yang digunakan dalam penelitian ini berjumlah 100, 200, dan 300 elemen, yang diperoleh dari input data pengguna.

Hasil pengujian waktu eksekusi kedua algoritma ditunjukkan pada Tabel 4.

Tabel 4 Perbandingan Waktu Eksekusi Quick Sort dan Merge Sort

Ukuran Dataset	***Quick Sort (detik)***	***Merge Sort (detik)***
100	0.000221	0.000267
200	0.000284	0.000604
300	0.000336	0.000699

Berdasarkan hasil pengujian pada Tabel 4, terlihat bahwa waktu eksekusi kedua algoritma meningkat seiring dengan bertambahnya ukuran dataset. Namun demikian, algoritma Quick Sort menunjukkan waktu eksekusi yang lebih cepat dibandingkan Merge Sort pada seluruh ukuran data yang diuji.

4.3 Analisis Perbandingan Performa

Perbandingan performa antara Quick Sort dan Merge Sort dapat dianalisis dari nilai waktu eksekusi pada setiap ukuran dataset.

p-ISSN : 2808-909X e-ISSN : 2798-2505

Pada dataset berukuran 100 elemen, Quick Sort menghasilkan waktu eksekusi sebesar 0.000221 detik, sedangkan Merge Sort memerlukan waktu 0.000267 detik. Hal ini menunjukkan bahwa Quick Sort sedikit lebih cepat dibandingkan Merge Sort pada ukuran data kecil.

Pada dataset berukuran 200 elemen, perbedaan performa kedua algoritma semakin terlihat. Quick Sort memiliki waktu eksekusi sebesar 0.000284 detik, sementara Merge Sort membutuhkan waktu 0.000604 detik. Hal ini menunjukkan bahwa Quick Sort hampir dua kali lebih cepat dibandingkan Merge Sort pada ukuran data tersebut.

Pada dataset berukuran 300 elemen, Quick Sort tetap menunjukkan performa yang lebih baik dengan waktu eksekusi 0.000336 detik, sedangkan Merge Sort membutuhkan waktu 0.000699 detik.

Secara keseluruhan, hasil pengujian menunjukkan bahwa Quick Sort memiliki performa yang lebih cepat dibandingkan Merge Sort pada seluruh ukuran dataset yang diuji dalam penelitian ini.

4.4 Visualisasi Grafik Perbandingan Algoritma

Selain disajikan dalam bentuk tabel, hasil pengujian juga divisualisasikan dalam bentuk grafik garis untuk memperlihatkan perbandingan performa kedua algoritma.

Pada grafik yang dihasilkan oleh program:

- **Sumbu X** menunjukkan ukuran dataset (100, 200, 300).
- **Sumbu Y** menunjukkan waktu eksekusi algoritma dalam satuan detik.

Grafik menunjukkan bahwa kedua algoritma mengalami peningkatan waktu eksekusi seiring bertambahnya jumlah data, namun garis Quick Sort berada di bawah garis Merge Sort yang menandakan bahwa Quick Sort memiliki waktu eksekusi yang lebih rendah.

4.5 Pembahasan

Berdasarkan hasil pengujian empiris, algoritma Quick Sort menunjukkan performa yang lebih baik dibandingkan Merge Sort pada dataset yang digunakan dalam penelitian ini.

Hal ini dapat terjadi karena Quick Sort memiliki overhead memori yang lebih kecil dibandingkan Merge Sort. Merge Sort memerlukan proses tambahan untuk menggabungkan dua bagian array yang telah diurutkan, sehingga waktu eksekusinya relatif lebih besar.

Selain itu, pada implementasi yang digunakan dalam penelitian ini, Quick Sort menggunakan pivot dari elemen tengah, yang dapat membantu menghasilkan pembagian data yang relatif seimbang sehingga proses pengurutan menjadi lebih efisien.

Hasil penelitian ini juga menunjukkan bahwa kedua algoritma mengalami peningkatan waktu eksekusi seiring dengan bertambahnya ukuran dataset, yang sejalan dengan teori kompleksitas algoritma pengurutan yang menyatakan bahwa Quick Sort dan Merge Sort memiliki kompleksitas rata-rata O(n log n).

Namun demikian, pada dataset yang digunakan dalam penelitian ini, Quick Sort memberikan performa yang lebih cepat dibandingkan Merge Sort.

5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pengujian yang telah dilakukan terhadap algoritma Quick Sort dan Merge Sort menggunakan dataset berukuran 100, 200, dan 300 elemen, maka dapat diambil beberapa kesimpulan sebagai berikut:

1. Hasil pengujian menunjukkan bahwa algoritma Quick Sort memiliki waktu eksekusi yang lebih cepat dibandingkan dengan Merge Sort pada seluruh ukuran dataset yang digunakan dalam penelitian ini.
2. Pada dataset berukuran 100 data, Quick Sort memiliki waktu eksekusi sebesar 0.000221 detik, sedangkan Merge Sort membutuhkan waktu 0.000267 detik.
3. Pada dataset berukuran 200 data, Quick Sort menghasilkan waktu eksekusi sebesar 0.000284 detik, sementara Merge Sort memerlukan waktu 0.000604 detik.
4. Pada dataset berukuran 300 data, Quick Sort tetap menunjukkan performa yang lebih baik dengan waktu eksekusi 0.000336 detik, sedangkan Merge Sort memerlukan waktu 0.000699 detik.
5. Berdasarkan hasil tersebut, dapat disimpulkan bahwa Quick Sort lebih efisien dalam proses pengurutan data pada dataset yang digunakan dalam penelitian ini, meskipun secara teori kedua algoritma memiliki kompleksitas rata-rata O(n log n).
6. Hasil penelitian juga menunjukkan bahwa waktu eksekusi kedua algoritma meningkat seiring dengan bertambahnya ukuran dataset, yang sesuai dengan karakteristik kompleksitas algoritma pengurutan.

5.2 Saran

Berdasarkan hasil penelitian yang telah dilakukan, terdapat beberapa saran yang dapat diberikan untuk pengembangan penelitian selanjutnya, yaitu:

1. Penelitian selanjutnya dapat menggunakan ukuran dataset yang lebih besar agar perbedaan performa algoritma dapat diamati dengan lebih jelas.
2. Pengujian algoritma dapat dilakukan dengan berbagai jenis dataset, seperti data acak, data terurut, dan data terbalik (reverse sorted) untuk melihat performa algoritma pada kondisi yang berbeda.
3. Penelitian selanjutnya juga dapat menambahkan algoritma pengurutan lainnya, seperti Heap Sort, Bubble Sort, atau Insertion Sort, sehingga perbandingan algoritma menjadi lebih luas.
4. Pengujian dapat dilakukan dengan pengulangan eksperimen beberapa kali untuk memperoleh nilai rata-rata waktu eksekusi yang lebih stabil.
5. Penelitian berikutnya juga dapat menganalisis penggunaan memori (memory usage) selain waktu eksekusi untuk memperoleh gambaran efisiensi algoritma yang lebih komprehensif.

DAFTAR PUSTAKA

[1] E. Panggabean, Yuda Perwira, Dedi Candro Parulian Sinaga, Annisa Tri Utami, and Vincha Swe Meiya Pricilla Sembiring, “Performance Analysis of Parallel Merge Sort Using MPI (Message Passing Interface) on Big Data Dataset,” *Jurnal Sistem Komputer dan Informatika (JSON)*, vol. 7, no. 2, pp. 594–560, Dec. 2025, doi: 10.30865/json.v7i2.9307.

[2] O. Esau Taiwo, A. O. Christianah, A. N. Oluwatobi, K. A. Aderonke, and A. J. Kehinde, “COMPARATI*VE STUDY of TWO DIVIDE and CONQUER SORTING ALGORITHMS: QUICKSORT and MERGESORT,” in *Procedia Computer Science*, Elsevier B.V., 2020, pp. 2532–2540. doi: 10.1016/j.procs.2020.04.274.

p-ISSN : 2808-909X e-ISSN : 2798-2505

[3] "Optimization of Sorting Algorithms for Big Data and Cloud Computing Environments," *Journal of Academic Trends and Innovative Research*, vol. 1, no. 1, Dec. 2025, doi: 10.64643/jatirv1i1-140030-001.

[4] R. Ayazuddin, "A Comprehensive Study of Sorting Algorithm Performance Using Real-World Dataset Metrics," Sep. 30, 2025. doi: 10.20944/preprints202509.2550.v1.

[5] S. Khalafallah and S. S. Abu-Naser, "AI-Driven Sorting Algorithms for Big Data: Techniques and Real-World Applications," 2025. [Online]. Available: www.ijeais.org/ijaer

[6] M. Nazril Ilham, A. Faza Setiawan, I. Kholifatun, and M. H. Aldiansyah, "Journal of Artificial Intelligence and Engineering Applications Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C++ Programming Language," 2025. [Online]. Available: https://ioinformatic.org/

[7] "Optimization of Sorting Algorithms for Big Data and Cloud Computing Environments," *Journal of Academic Trends and Innovative Research*, vol. 1, no. 1, Dec. 2025, doi: 10.64643/jatirv1i1-140030-001.

[8] F. Laia, F. Tharorogo Wau, and J. Manurung, "Optimization of quicksort algorithm for real-time data processing in IoT systems with random pivot division and tail recursion," vol. 14, no. 1, pp. 96–103, 2025, [Online]. Available: www.ejournal.isha.or.id/index.php/Mandiri

[9] Prof.Mrs. Tejaswini.A. Puranik, "Performance Analysis of Sorting and Searching Algorithms," *International Research Journal on Advanced Engineering and Management (IRJAEM)*, vol. 3, no. 08, pp. 2741–2746, Aug. 2025, doi: 10.47392/irjaem.2025.0430.

[10] Y. K. Tamandani, "Performance analysis of parallel sorting algorithms Using Message Passing Interface (MPI)", doi: 10.22034/jcse.2025.182105.

[11] Y. Li, "Performance Analysis of Efficient Sorting Algorithms in Big Data Processing," in *Procedia Computer Science*, Elsevier B.V., 2025, pp. 44–50. doi: 10.1016/j.procs.2025.05.026.

[12] K. Carter, S. M. G. Ho, M. M. A. Larsen, M. Sundman, and M. H. Kirkeby, "Energy and Time Complexity for Sorting Algorithms in Java," May 2024, [Online]. Available: http://arxiv.org/abs/2311.07298

[13] A. S. Sabah *et al.*, "Exponential (10K)," 2023. [Online]. Available: www.ijeais.org/ijaer

[14] A. G. Aly, A. E. Jensen, and H. ElAarag, "Improving Merge Sort and Quick Sort Performance by Utilizing Alphadev's Sorting Networks as Base Cases," in *ACMSE 2025 - Proceedings of the 2025 ACM Southeast Conference*, Association for Computing Machinery, Inc, May 2025, pp. 239–244. doi: 10.1145/3696673.3723083.

[15] Y. Shi and Z. Li, "Leyenda: An Adaptive, Hybrid Sorting Algorithm for Large Scale Data with Limited Memory."

[16] A. Farhan, "Sorting Execution Time and Comparison Analysis-Quick Sort, Hybrid Quick Sort, Top-Down & Bottom-Up Merge Sort on Synthetic Data."

[17] T. Alghamdi and G. Alaghband, "HIGH PERFORMANCE PARALLEL SORT FOR SHARED AND DISTRIBUTED MEMORY MIMD."

[18] I. Mishal, R. Al-Khatib, and R. Hiasat, "Comparative Study of Two Divide and Conquer Sorting Algorithms: Modified Quick Sort and Merge Sort," 2021.

[19] Y. Alhabboub, F. Almutairi, M. Safhi, Y. Alqahtani, A. Almeedani, and Y. Alguwaifli, "Accelerating Large-Scale Sorting through Parallel Algorithms," *Journal of Computer and Communications*, vol. 12, no. 01, pp. 131–138, 2024, doi: 10.4236/jcc.2024.121009.

[20] 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, Sep. 2024, doi: 10.33003/fjs-2024-0805-2730.

[21] N. S. Abuba, E. Y. Baagyere, C. I. Nakpih, and J. K. Wiredu, "OptiFlexSort: A Hybrid Sorting Algorithm for Efficient Large-Scale Data Processing," *Journal of Advances in Mathematics and Computer Science*, vol. 40, no. 2, pp. 67–81, Feb. 2025, doi: 10.9734/jamcs/2025/v40i21970.

[22] Z. Mamatnabiyev and M. Zhaparov, "COMPARATIVE ANALYSIS OF SORTING ALGORITHMS USED IN COMPETITIVE PROGRAMMING."

[23] J. K. Wiredu, S. Akobre, I. Aabaah, and U. A. Wumpini, "A Unified Framework for Theoretical and Experimental Evaluation of Classical and Modern Sorting Algorithms in Real-Time Systems," *Journal of Information Systems and Informatics*, vol. 7, no. 4, pp. 3143–3185, Dec. 2025, doi: 10.63158/journalisi.v7i4.1287.

[24] A. G. Aly, A. E. Jensen, and H. Elaarag, "EasyChair Preprint Improving Merge Sort and Quick Sort Performance by Utilizing Alphadev's Sorting Networks as Base Cases Improving Merge Sort and Quick Sort Performance by Utilizing Alphadev's Sorting Networks as Base Cases," 2025, doi: 10.1145/YYYYYYY.YYYYYYY.

[25] A. Amini, "TwinArray Sort: An Ultrarapid Conditional Non-Comparison Based Sorting Algorithm," Sep. 2024, [Online]. Available: http://arxiv.org/abs/2409.20362

[26] S. Sundaramoorthy and G. Karunanidhi, "A systematic analysis on performance and computational complexity of sorting algorithms," Dec. 01, 2025, *Springer Science and Business Media B.V.* doi: 10.1007/s10791-025-09724-w.

[27] M. Hassan, G. Muhammad, J. A. Malik, M. Akhtar, M. A. Baloch, and M. A. Rajwana, "Sort Data Faster: Comparing Algorithms."

[28] S. Arunachalam Balasubramanian, "Adaptive Hybrid Sort: Dynamic Strategy Selection for Optimal Sorting Across Diverse Data Distributions."

[29] Achmad Fitro, "Comparison of Efficiency Data Sorting Algorithms Based on Execution Time," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp. 15–21, Mar. 2023, doi: 10.32628/cseit2390151.

[30] A. Fahreza Alif Pahlevi, "Quick sort and merge sort performance comparison in the flutter framework 1*," *Journal of Informatics and Science Media*, vol. 1, no. 1.