**Jurnal Ilmiah Ilmu Komputer & Elektronika (MURAKOM)**
**Volume. 2 Nomor. 2 April 2026**
e-ISSN : 3123-1985; p-ISSN : 3110-4754, Hal. 01-15
DOI: https://doi.org/10.64365/murakom.v2i2.278
*Available online at*: *https://jurnal.muaraedukasi.id/index.php/murakom*

# Analisis Komparatif Efisiensi *Merge Sort* dan *Quick Sort* Menggunakan Pendekatan *Divide* and *Conquer* pada Berbagai Kondisi Data Produk Sistem *E-Commerce*

**Naufal Aqiilah Asra [1*], Mohd. Rafiif Albani[2], Gus Rosauli Pandiagan[3], Angelica Barus[4], Adidtya Perdana [5]**
[1,2,3,4,5] Universitas Negeri Medan, Indonesia
Email : naufalaqiilahasra2025@gmail.com[1], rafiifalbani21@gmail.com[2], rosapandiangan71@gmail.com[3], angelicabrs.4243250029@mhs.unimed.ac.id[4], adidtya@unimed.ac.id[5],
Alamat: Jalan Willem Iskandar, Pasar V, Medan Estate, Kecamatan Percut Sei Tuan, Kabupaten Deli Serdang, Sumatera Utara
*Korespondensi penulis*: *naufalaqiilahasra2025@gmail.com*

***Abstract***. *Abstract. The efficiency of product data sorting on e-commerce platforms is a key factor in user satisfaction. Although Merge Sort and Quick Sort theoretically have the same complexity, their actual performance depends heavily on the initial state of the data being processed. This study compares the efficiency of both algorithms using Python simulations on random, sorted, and reversed data scenarios with up to 100,000 elements. The test results demonstrate that Quick Sort outperforms Merge Sort on random data with a 19.2% time efficiency advantage due to more optimal memory usage. On the other hand, Merge Sort demonstrates high stability as it is unaffected by data distribution patterns and is even 42.5% faster than Quick Sort on reverse-sorted data. In conclusion, Quick Sort is most recommended for processing random data, while Merge Sort is the best solution if the system requires stable performance on large-scale data or data with specific patterns.*

***Keywords:*** *Sorting Algorithms, Divide and Conquer, E-Commerce, Merge Sort, Quick Sort*

**Abstrak**. Efisiensi pengurutan data produk pada platform belanja daring menjadi faktor penting bagi kepuasan pengguna. Walaupun Merge Sort dan Quick Sort secara teori punya kompleksitas yang sama namun performa realitanya sangat bergantung pada kondisi awal data yang diolah. Penelitian ini membandingkan efisiensi kedua algoritma menggunakan simulasi Python pada skenario data acak, terurut, dan terbalik hingga 100.000 elemen. Hasil pengujian membuktikan Quick Sort lebih unggul pada data acak dengan efisiensi waktu 19,2% karena penggunaan memori yang lebih optimal. Di sisi lain Merge Sort menunjukkan kestabilan tinggi karena tidak terpengaruh oleh pola distribusi data bahkan 42,5% lebih cepat daripada Quick Sort pada kondisi data terurut terbalik. Kesimpulannya Quick Sort paling disarankan untuk pengolahan data acak sedangkan Merge Sort menjadi solusi terbaik jika sistem memerlukan kestabilan performa pada data berskala besar atau memiliki pola tertentu.

**Kata kunci**: Algoritma Pengurutan, Divide and Conquer, E-Commerce, Merge Sort, Quick Sort

## 1. LATAR BELAKANG

Dalam perkembangan sistem *e-commerce* saat ini, kecepatan dalam menampilkan hasil pencarian menjadi salah satu faktor penting yang menentukan kenyamanan pengguna. Ketika pengguna melakukan penyaringan produk berdasarkan kriteria tertentu seperti harga atau rating, sistem harus mampu mengolah dan mengurutkan data dalam jumlah besar secara cepat. Proses ini tidak hanya melibatkan sedikit data, tetapi bisa mencapai ribuan hingga jutaan item dalam satu waktu. Jika proses tersebut berjalan lambat, pengguna cenderung meninggalkan aplikasi dan beralih ke platform lain. Hal ini menunjukkan bahwa performa sistem, khususnya

*Received: Februari 15, 2026; Revised: Maret 20, 2026; Accepted: April 01, 2026;*
*Online Available: April 02, 2026; Published: April 14, 2026;*

dalam proses pengurutan data, memiliki peran yang sangat penting dalam menjaga kualitas layanan.

Untuk mendukung kebutuhan tersebut, berbagai algoritma pengurutan dikembangkan dalam ilmu komputer, masing-masing dengan karakteristik yang berbeda. Di antara algoritma yang sering digunakan, Merge Sort dan Quick Sort menjadi dua metode yang cukup populer karena efisiensinya dalam menangani data berukuran besar. Keduanya menggunakan pendekatan *divide and conquer*, yaitu memecah data menjadi bagian-bagian yang lebih kecil, kemudian menyelesaikannya sebelum digabungkan kembali. Secara teoritis, kedua algoritma ini memiliki kompleksitas waktu rata-rata yang sama, yaitu $O(n \log n)$, sehingga sering dianggap memiliki performa yang sebanding dalam banyak kasus (Golonka & Krużel, 2025).

Namun dalam implementasinya, performa kedua algoritma tersebut tidak selalu menunjukkan hasil yang sama. Perbedaan ini umumnya dipengaruhi oleh kondisi awal data yang diproses. Dalam sistem nyata seperti e-commerce, data tidak selalu berada dalam kondisi ideal, melainkan dapat berbentuk acak, sudah terurut, atau bahkan terurut terbalik. Penelitian terbaru menunjukkan bahwa distribusi data memiliki pengaruh yang cukup signifikan terhadap waktu eksekusi dan efisiensi algoritma sorting, meskipun secara teoritis memiliki kompleksitas yang sama (Ala'anzy et al., 2026). Sayangnya, masih banyak penelitian yang hanya menguji algoritma pada kondisi tertentu saja, sehingga belum sepenuhnya menggambarkan performa algoritma dalam situasi yang lebih beragam dan realistis.

Berdasarkan hal tersebut, penelitian ini dilakukan untuk menganalisis dan membandingkan performa *Merge Sort* dan *Quick Sort* pada beberapa kondisi data yang berbeda, yaitu data acak, data terurut, dan data terurut terbalik. Analisis dilakukan dengan melihat waktu eksekusi serta jumlah perbandingan elemen yang terjadi selama proses pengurutan. Dengan adanya penelitian ini, diharapkan dapat diperoleh pemahaman yang lebih jelas mengenai karakteristik kedua algoritma serta rekomendasi yang lebih tepat dalam memilih metode pengurutan yang sesuai untuk diterapkan pada sistem e-commerce.

## 2. KAJIAN TEORITIS

Perkembangan pesat ekosistem perdagangan elektronik (*e-commerce*) menuntut optimalisasi pemrosesan data berskala masif. Ketika konsumen memfilter produk berdasarkan harga atau popularitas, sistem harus menyusun ulang jutaan entitas data secara presisi dengan latensi minimum. Oleh karena itu, efisiensi algoritma pengurutan menjadi determinan krusial dalam rekayasa perangkat lunak untuk mengelola basis data secara adaptif (Ivanova, 2024). Pemilihan algoritma yang tepat akan memastikan waktu eksekusi yang optimal demi menjaga responsivitas aplikasi di tengah tingginya interaksi pengguna (Fitro, 2023).

Dalam upaya mencapai efisiensi komputasional tersebut, paradigma *Divide and Conquer* hadir sebagai landasan strategis yang mampu mendekonstruksi kompleksitas pemrosesan data. Pendekatan ini beroperasi secara rekursif dengan mempartisi himpunan data menjadi sub-himpunan yang lebih elementer, memprosesnya secara independen, lalu mensintesisnya menjadi solusi yang terurut secara utuh. Strategi ini terbukti secara empiris mampu memangkas waktu eksekusi secara signifikan dibandingkan algoritma perbandingan konvensional. Dua algoritma paling prominen dari paradigma ini yang sangat relevan untuk diaplikasikan pada e-commerce adalah Merge Sort dan Quick Sort (Gautam & Naman, 2021; Mishal dkk., 2021).

Algoritma *Merge Sort* bekerja dengan membelah struktur data secara simetris hingga mencapai unit tunggal, yang kemudian digabungkan kembali secara terurut. Keunggulan fundamental dari algoritma ini terletak pada tingkat stabilitas dan konsistensi kompleksitas waktunya yang selalu berada pada batas matematis pada berbagai permutasi data. Hal ini menjadikannya sangat andal untuk memproses dataset bervolume raksasa tanpa risiko penurunan performa secara tiba-tiba (Al-Raimi dkk., 2024). Meskipun demikian, kapabilitas ini menuntut kompensasi memori tambahan (*not in-place*) sebesar yang dapat membebani sistem pada lingkungan dengan sumber daya terbatas (Gautam & Naman, 2021).

Sebagai alternatif yang jauh lebih efisien secara ruang, Quick Sort menawarkan mekanisme pengurutan *in-place* dengan memilih elemen poros untuk mempartisi himpunan data secara dinamis. Algoritma ini sangat ramah terhadap memori *cache* perangkat keras dan secara rata-rata mendemonstrasikan kecepatan eksekusi yang superior (Aftab dkk., 2021). Namun, *Quick Sort* memiliki kerentanan performa saat dihadapkan pada kumpulan data yang sudah terurut maupun terurut terbalik. Pada kondisi patologis tersebut, distribusinya menjadi tidak seimbang sehingga kinerjanya dapat mengalami degradasi ekstrem hingga mencapai kompleksitas kuadratik. (Shorman dkk., 2024).

Berbagai literatur terdahulu menyimpulkan bahwa efisiensi operasional sebuah algoritma tidak murni bergantung pada logika internalnya saja, melainkan sangat terpengaruh oleh karakteristik dan distribusi awal data yang diproses (Gupta & Chaudhary, 2025). Mengingat platform *e-commerce* secara kontinu menerima input yang dinamis akibat interaksi pengguna, tidak ada algoritma tunggal yang mutlak sempurna untuk semua situasi. Oleh karena itu, evaluasi komparatif antara algoritma Merge Sort dan Quick Sort pada kondisi data acak, terurut, dan terurut terbalik menjadi esensial untuk dilakukan. Analisis mendalam ini akan menghasilkan landasan yang kuat dalam menentukan rekomendasi arsitektur pengurutan produk yang paling tangguh di dunia nyata (Mishal dkk., 2021).

## 3. METODE PENELITIAN

Secara metodologis, studi ini mengadopsi pendekatan eksperimen komputasional. Kami merancang sebuah perangkat lunak simulasi berbasis Python untuk menguji, mengukur, dan membandingkan performa algoritma secara empiris di bawah tekanan pemrosesan data berskala besar.

### 3.1. Rekayasa Skenario Dataset E-Commerce

Karena data transaksi di dunia nyata memiliki pola yang tidak menentu, kami tidak menggunakan dataset statis. Sebaliknya, kami memprogram sebuah fungsi generator (pembuat data) yang secara dinamis memproduksi struktur *list of dictionaries*. Setiap entitas di dalam *list* mewakili satu produk dengan atribut spesifik: ID produk, nama, harga (rentang acak yang luas), dan nilai *rating*.

Untuk memastikan pengujian mencakup segala kemungkinan (*best, average, dan worst-case*), generator ini kami instruksikan untuk menyusun data ke dalam tiga skenario mutlak:

a. **Kondisi Acak (*Random Dataset*):** Harga produk disebar tanpa pola untuk menyimulasikan sistem *e-commerce* pada kondisi normal.
b. **Kondisi Terurut (*Sorted Dataset*):** Data sudah tersusun dari harga termurah hingga termahal sebelum diolah algortima. Langkah ini kami ambil untuk menguji apakah algoritma membuang-buang waktu pada data yang sudah rapi.
c. **Kondisi Terurut Terbalik (*Reverse Dataset*):** Simulasi kondisi terekstrem di mana pengguna meminta pengurutan dari murah ke mahal, namun sistem justru menerima data yang terurut dari mahal ke murah.

### 3.2. Implementasi dan Injeksi Logika Algoritma

Kedua algoritma yang diteliti ditulis murni menggunakan pendekatan *Divide and Conquer*, namun dengan modifikasi logika yang kami sesuaikan untuk kebutuhan eksperimen:

a. **Merge Sort:** Kami merancang fungsi rekursif yang secara kaku membelah *array* produk tepat di titik tengah (mid). Pada fungsi ini, kami menyisipkan variabel *counter* global untuk menghitung dan merekam setiap kali elemen kiri dan kanan saling dibandingkan.
b. **Quick Sort (Modifikasi):** Menyadari bahwa *Quick Sort* konvensional rentan mengalami kerusakan performa (menjadi lambat secara kuadratik / $O(n^2)$) saat berhadapan dengan skenario *Sorted* dan *Reverse*, kami menginjeksikan teknik *Median-of-Three* ke dalam kode. Dengan teknik ini, program kami dipaksa untuk mengukur nilai awal, tengah, dan akhir terlebih dahulu sebelum menentukan titik batas (*pivot*), sehingga pembelahan data selalu terjaga keseimbangannya.

### 3.3. Instrumen Pengukuran Metrik (Benchmarking)

Untuk menghindari bias akibat fluktuasi spesifikasi *hardware* (perangkat keras) komputer yang digunakan, kami menetapkan standar pengukuran yang ketat pada fungsi eksekutor utama:

a. **Presisi Waktu Mikro:** Kami tidak menggunakan fungsi waktu standar, melainkan mengeksploitasi pustaka time.perf_counter() milik Python yang mampu melacak waktu komputasi CPU murni hingga tingkat pecahan mikrodetik.

b. **Redundansi Eksekusi:** Setiap ukuran data uji (mulai dari rasio 100 baris hingga menembus batas ratusan ribu baris produk) tidak hanya dieksekusi satu kali, melainkan diulang otomatis sebanyak lima kali (*5 iterations*). Nilai yang kami ambil sebagai hasil akhir adalah nilai rata-ratanya.

### 3.4. Teknik Analisis dan Proyeksi Visual

Ribuan baris data mentah yang mencakup waktu komputasi dan jumlah komparasi dari hasil eksperimen di atas dianalisis dengan memproyeksikannya ke dalam format visual. Kami menggunakan pustaka *Matplotlib* untuk merender grafik garis pertumbuhan waktu (*time complexity curve*), grafik komparasi fundamental, serta grafik komparasi pada beban komputasi maksimal. Visualisasi ini menjadi instrumen utama kami dalam menarik kesimpulan konkrit mengenai algoritma mana yang paling efisien untuk arsitektur *e-commerce*.

## 4. HASIL DAN PEMBAHASAN

### 4.1 Hasil Verifikasi Kebenaran

Sebelum pengujian performa dilaksanakan, kebenaran (correctness) kedua algoritma diverifikasi terlebih dahulu dengan membandingkan output pengurutan terhadap referensi dari fungsi sorted() bawaan Python. Pengujian dilakukan pada sembilan skenario yang mencakup tiga variasi ukuran data (n = 10, 100, dan 1.000) serta tiga kondisi awal data (random, sorted, dan reverse sorted). Hasil verifikasi selengkapnya disajikan pada Tabel 4.1.

**Tabel 4.1 Hasil Verifikasi Correctness Merge Sort dan Quick Sort**

| N | Kondisi | Merge Sort | Quick Sort | Keterangan |
|---|---|---|---|---|
| 10 | Random | ☑ BENAR | ☑ BENAR | Hasil identik |
| 10 | Sorted | ☑ BENAR | ☑ BENAR | Hasil identik |

| | | | | |
|---|---|---|---|---|
| 10 | Reverse | ☑ BENAR | ☑ BENAR | Hasil identik |
| 100 | Random | ☑ BENAR | ☑ BENAR | Hasil identik |
| 100 | Sorted | ☑ BENAR | ☑ BENAR | Hasil identik |
| 100 | Reverse | ☑ BENAR | ☑ BENAR | Hasil identik |
| 1.000 | Random | ☑ BENAR | ☑ BENAR | Hasil identik |
| 1.000 | Sorted | ☑ BENAR | ☑ BENAR | Hasil identik |
| 1.000 | Reverse | ☑ BENAR | ☑ BENAR | Hasil identik |

Berdasarkan Tabel 4.1, seluruh sembilan skenario uji menghasilkan status BENAR untuk kedua algoritma, dengan output yang identik antara Merge Sort, Quick Sort, maupun fungsi sorted() Python. Hal ini membuktikan bahwa implementasi kedua algoritma telah bebas dari kesalahan logika dan menghasilkan urutan yang benar secara konsisten untuk semua variasi kondisi data yang diuji.

## 4.2 Hasil Pengujian Performa

Pengujian performa dilaksanakan menggunakan empat variasi ukuran dataset (n = 100, 1.000, 10.000, dan 100.000 elemen) pada tiga kondisi awal data (random, sorted, dan reverse sorted), dengan setiap skenario dijalankan sebanyak 5 kali iterasi dan hasilnya dirata-ratakan. Pengukuran mencakup dua metrik utama, yaitu waktu eksekusi rata-rata (dalam detik) dan jumlah operasi perbandingan rata-rata.

### a. Kondisi Data Acak (Random)

Pada kondisi data acak yang merupakan kondisi paling representatif dari situasi nyata di platform e-commerce, hasil pengujian menunjukkan bahwa *Quick Sort* secara konsisten lebih cepat dibandingkan Merge Sort pada semua ukuran dataset. Tabel 4.2 dan Tabel 4.3 menyajikan perbandingan waktu eksekusi dan jumlah perbandingan secara rinci.

**Tabel 4.2 Perbandingan Waktu Eksekusi pada Data Acak (Random)**

| n | Merge Sort (detik) | Quick Sort (detik) | Lebih Cepat |
|---|---|---|---|
| 100 | 0.000213 | 0.000111 | **Quick Sort (+47.9%)** |

| | | | |
|---|---|---|---|
| 1.000 | 0.003573 | 0.002676 | **Quick Sort (+25.1%)** |
| 10.000 | 0.046131 | 0.032213 | **Quick Sort (+30.2%)** |
| 100.000 | 0.578740 | 0.468036 | **Quick Sort (+19.2%)** |

**Tabel 4.3 Perbandingan Jumlah Perbandingan pada Data Acak (Random)**

| n | Merge Sort (perbandingan) | Quick Sort (perbandingan) | Lebih Sedikit |
|---|---|---|---|
| 100 | 531 | 531 | **Quick Sort** |
| 1.000 | 8.719 | 9.492 | **Merge Sort** |
| 10.000 | 120.325 | 135.429 | **Merge Sort** |
| 100.000 | 1.536.704 | 1.749.248 | **Merge Sort** |

Pada data acak dengan n = 100.000, *Quick Sort* membutuhkan waktu eksekusi sebesar 0,468036 detik, sedangkan Merge Sort membutuhkan 0,578740 detik, sehingga *Quick Sort* unggul sekitar 19,2%. Dari sisi jumlah perbandingan, Merge Sort mencatat lebih sedikit perbandingan dibandingkan Quick Sort; namun demikian, waktu eksekusi Quick Sort tetap lebih cepat karena efisiensi akses memori yang lebih baik (in-place sorting) dan konstanta implementasi yang lebih kecil dibandingkan Merge Sort yang memerlukan alokasi dan penyalinan array tambahan.

**b. Kondisi Data Terurut (Sorted)**

Pada kondisi data yang sudah terurut sebelumnya, Quick Sort dengan strategi median-of-three menunjukkan performa yang kompetitif. Tabel 4.4 dan Tabel 4.5 menyajikan perbandingan hasil pengujian pada kondisi ini.

**Tabel 4.4 Perbandingan Waktu Eksekusi pada Data Terurut (Sorted)**

| n | Merge Sort (detik) | Quick Sort (detik) | Lebih Cepat |
|---|---|---|---|
| 100 | 0.000142 | 0.000267 | Merge Sort (+47.2%) |
| 1.000 | 0.002948 | 0.001547 | Quick Sort (+47.6%) |
| 10.000 | 0.024882 | 0.020883 | Quick Sort (+16.1%) |
| 100.000 | 0.332078 | 0.309451 | Quick Sort (+6.8%) |

**Tabel 4.5 Perbandingan Jumlah Perbandingan pada Data Terurut (Sorted)**

| n | Merge Sort (perbandingan) | Quick Sort (perbandingan) | Lebih Sedikit |
|---|---|---|---|
| 100 | 316 | 480 | Merge Sort |
| 1.000 | 4.932 | 7.987 | Merge Sort |
| 10.000 | 64.608 | 113.631 | Merge Sort |
| 100.000 | 815.024 | 1.468.959 | Merge Sort |

Pada kondisi data terurut, Quick Sort justru lebih cepat pada dataset besar (n = 100.000), dengan waktu 0,309451 detik dibandingkan Merge Sort 0,332078 detik Quick Sort unggul sekitar 6,8%. Hal yang paling mencolok adalah jumlah perbandingan Merge Sort yang jauh lebih sedikit dibandingkan Quick Sort. Fenomena ini disebabkan karena pada data yang sudah terurut, pembelahan simetris Merge Sort menghasilkan sub-array yang hampir seimbang, sedangkan Quick Sort dengan median-of-three masih memerlukan lebih banyak rotasi dan perbandingan tambahan dalam proses pemilihan pivot dan partisi.

### c. Kondisi Data Terurut Terbalik (Reverse Sorted)

Kondisi data terurut terbalik merupakan skenario yang paling kritis bagi Quick Sort tanpa optimasi, karena pemilihan pivot yang buruk berpotensi menyebabkan degradasi ke $O(n^2)$. Namun, berkat strategi median-of-three, pengujian menunjukkan hasil yang menarik. Tabel 4.6 dan Tabel 4.7 menyajikan perbandingan lengkapnya.

**Tabel 4.6 Perbandingan Waktu Eksekusi pada Data Terbalik *(Reverse Sorted)***

| n | Merge Sort (detik) | Quick Sort (detik) | Lebih Cepat |
|---|---|---|---|
| 100 | 0.000088 | 0.000134 | **Quick Sort (+25.4%)** |
| 1.000 | 0.002369 | 0.002367 | **Merge Sort (+0.1%)** |
| 10.000 | 0.027119 | 0.038069 | **Merge Sort (+28.7%)** |
| 100.000 | 0.329044 | 0.572599 | **Merge Sort (+42.5%)** |

**Tabel 4.7 Perbandingan Jumlah Perbandingan pada Data Terbalik *(Reverse Sorted)***

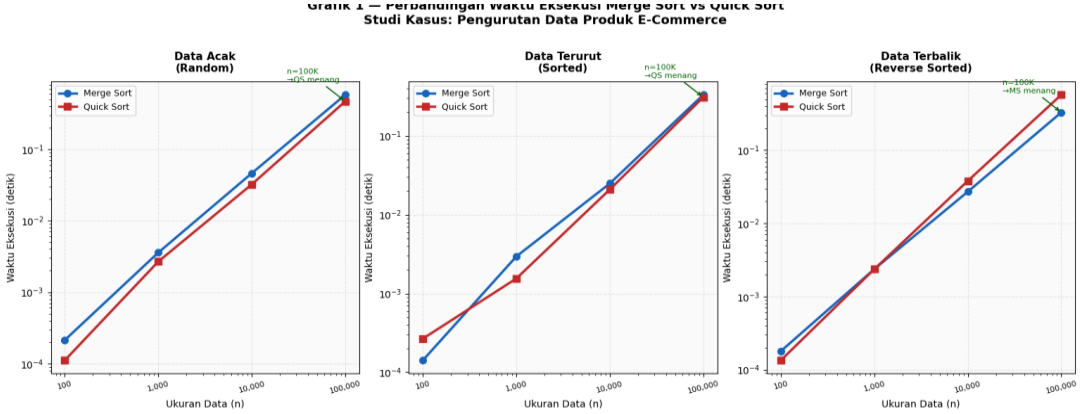
| n | Merge Sort (perbandingan) | Quick Sort (perbandingan) | Lebih Sedikit |
|---|---|---|---|
| 100 | 356 | 700 | **Merge Sort** |
| 1.000 | 5.044 | 14.205 | **Merge Sort** |
| 10.000 | 69.010 | 218.587 | **Merge Sort** |
| 100.000 | 854.064 | 2.941.502 | **Merge Sort** |

Pada kondisi data terurut terbalik, Merge Sort secara signifikan lebih unggul, terutama pada dataset besar. Pada n = 100.000, Merge Sort menyelesaikan pengurutan dalam 0,329044 detik sementara Quick Sort membutuhkan 0,572599 detik — Merge Sort sekitar 42,5% lebih cepat. Perbedaan jumlah perbandingan pun sangat dramatis, dengan Merge Sort jauh lebih

sedikit melakukan perbandingan dibandingkan Quick Sort. Meskipun median-of-three berhasil mencegah degenerasi total ke $O(n^2)$, kondisi data terbalik tetap menghasilkan partisi yang kurang seimbang, sehingga Quick Sort membutuhkan lebih banyak rekursi dan perbandingan dibandingkan Merge Sort yang tidak terpengaruh oleh pola distribusi data.

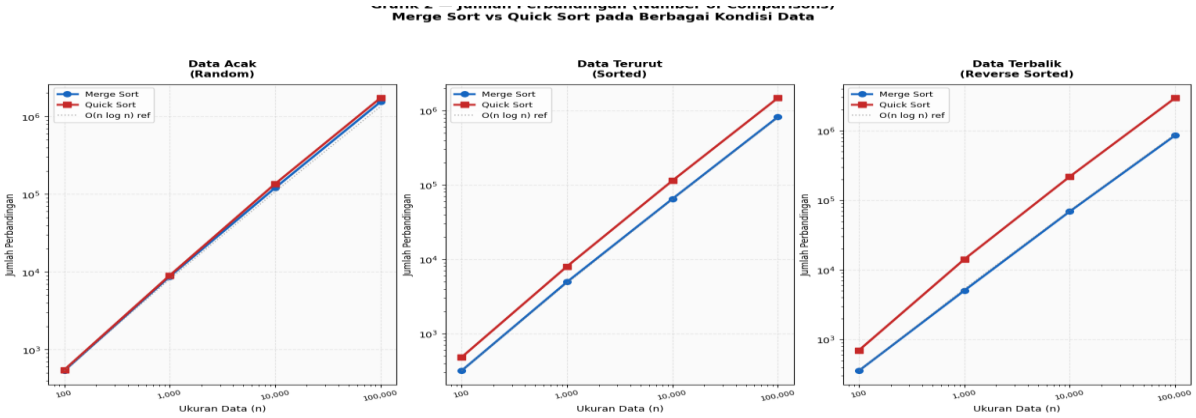
### 4.3 Visualisasi Hasil Pengujian

Setelah dijalankan program secara otomatis menghasilkan empat grafik visualisasi yang disimpan sebagai file PNG. Keempat grafik tersebut menyajikan hasil eksperimen dalam format visual yang mudah diinterpretasikan.



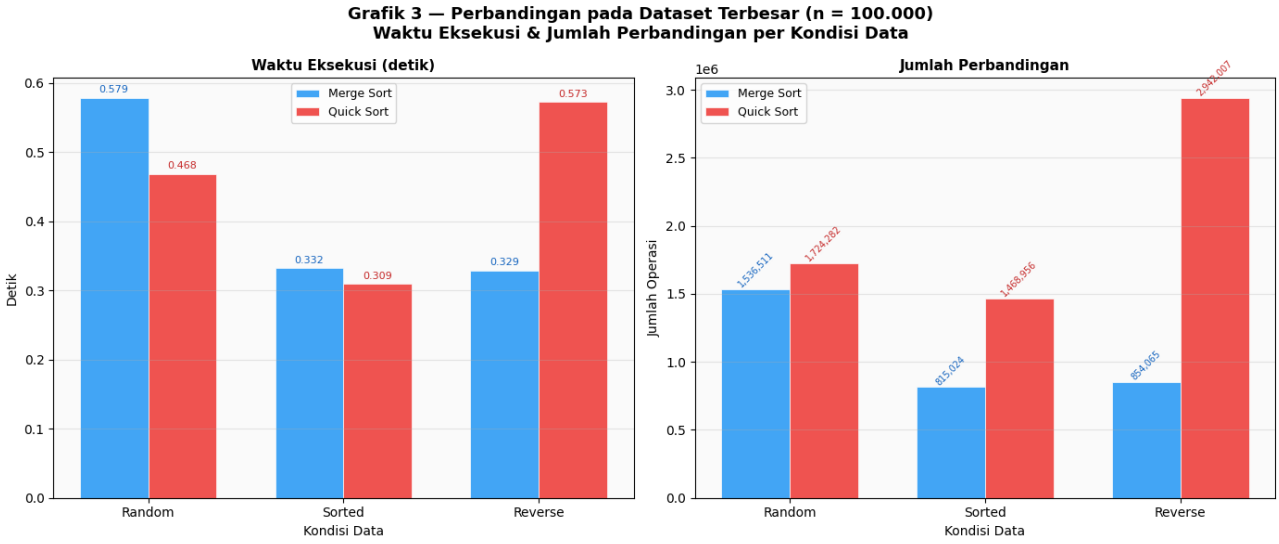

**Gambar 1. Grafik Perbandingan Waktu Eksekusi**

Grafik garis dengan sumbu logaritmik (log-log plot) yang menampilkan tren pertumbuhan waktu eksekusi kedua algoritma pada tiga kondisi data secara terpisah dalam tiga subplot berdampingan. Dari grafik ini terlihat bahwa kedua garis tumbuh mengikuti pola O(n log n), namun Quick Sort konsisten berada di bawah garis Merge Sort pada data acak, sementara posisi keduanya berbalik pada data terbalik.



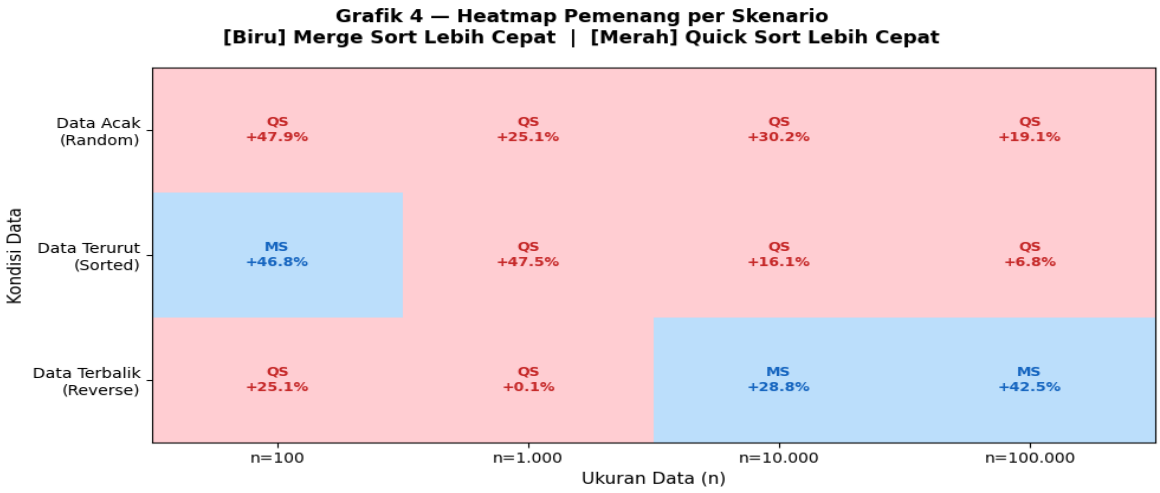

**Gambar 2. Grafik Jumlah Perbandingan**

Grafik garis serupa yang membandingkan jumlah operasi perbandingan disertai kurva referensi teoritis O(n log n) berskala, sehingga pembaca dapat langsung membandingkan hasil eksperimen dengan prediksi matematis. Garis kedua algoritma mengikuti kurva referensi dengan sangat rapat pada data acak, sedangkan pada data terbalik garis Quick Sort mulai menjauh ke atas, mengindikasikan jumlah perbandingan yang jauh lebih banyak.




**Gambar 3. Grafik Bar Chart Dataset Terbesar (n = 100.000)**

Dua bar chart berdampingan yang membandingkan waktu eksekusi dan jumlah perbandingan kedua algoritma pada dataset terbesar untuk ketiga kondisi data, Grafik ini memperlihatkan dengan jelas bahwa perbedaan paling dramatis terjadi pada kondisi data terbalik, di mana batang Quick Sort hampir dua kali lebih tinggi dari Merge Sort untuk waktu eksekusi, dan lebih dari tiga kali lipat untuk jumlah perbandingan.




**Gambar 4. Grafik Heatmap Pemenang per Skenario**

Visualisasi heatmap berwarna yang menampilkan algoritma pemenang untuk setiap kombinasi skenario (kondisi data × ukuran data), disertai persentase keunggulan relatif. Sel berwarna biru menandakan Merge Sort lebih cepat, sedangkan sel merah menandakan Quick Sort lebih cepat, disertai angka persentase keunggulan. Dari heatmap ini terlihat pola yang jelas: Quick Sort dominan pada data acak di semua ukuran data, sementara Merge Sort dominan pada data terurut (n besar) dan hampir seluruh kondisi data terbalik.

**4.4 Pembahasan dan Analisis**

Perbedaan utama antara kedua algoritma terletak pada kompleksitas ruang dan perilaku pada kondisi data spesifik. Merge Sort memiliki kompleksitas ruang O(n) karena memerlukan array sementara selama fase penggabungan, sementara Quick Sort hanya memerlukan O(log n) untuk stack rekursi karena bersifat in-place. Namun, Quick Sort memiliki risiko worst case $O(n^2)$ jika pemilihan pivot tidak optimal, sedangkan Merge Sort selalu menjamin O(n log n) pada semua kondisi.

Berdasarkan hasil pengujian dan visualisasi yang telah disajikan, terlihat bahwa performa algoritma Merge Sort dan Quick Sort dipengaruhi secara signifikan oleh kondisi awal data serta karakteristik implementasinya. Merge Sort menunjukkan performa yang relatif stabil pada seluruh skenario pengujian, baik pada data acak, terurut, maupun terbalik, yang mencerminkan sifat algoritma ini yang tidak bergantung pada distribusi data. Sebaliknya, Quick Sort memperlihatkan performa yang lebih fluktuatif, di mana algoritma ini unggul pada data acak namun mengalami penurunan performa pada kondisi tertentu, khususnya pada data terbalik. Hal ini menunjukkan bahwa efektivitas Quick Sort sangat dipengaruhi oleh kualitas partisi yang dihasilkan dari pemilihan pivot.

Namun, dari sisi jumlah perbandingan, Merge Sort justru melakukan jumlah perbandingan yang lebih sedikit dibandingkan Quick Sort pada beberapa ukuran dataset. Meskipun demikian, waktu eksekusi Merge Sort tetap lebih lambat karena Merge Sort memerlukan proses penggabungan (merge) yang membutuhkan alokasi array sementara, sehingga menambah beban memori dan waktu akses data. Hal ini menunjukkan bahwa waktu eksekusi tidak hanya dipengaruhi oleh jumlah perbandingan, tetapi juga oleh faktor penggunaan memori dan efisiensi akses data di dalam memori komputer

Pada kondisi data acak, Quick Sort menunjukkan performa waktu eksekusi yang lebih cepat dibandingkan Merge Sort pada seluruh ukuran dataset. Hal ini terjadi karena Quick Sort bekerja secara in-place sehingga tidak memerlukan alokasi memori tambahan yang besar seperti Merge Sort. Selain itu, konstanta implementasi Quick Sort relatif lebih kecil sehingga

dalam praktiknya Quick Sort sering kali lebih cepat meskipun secara teoritis kedua algoritma memiliki kompleksitas rata-rata yang sama, yaitu O(n log n).

Pada kondisi data terurut, performa kedua algoritma relatif tidak berbeda jauh pada dataset kecil, namun pada dataset yang lebih besar Merge Sort mulai menunjukkan performa yang lebih stabil. Hal ini disebabkan karena kompleksitas Merge Sort selalu berada pada O(n log n) pada semua kondisi data, sedangkan Quick Sort sangat bergantung pada pemilihan pivot. Jika partisi yang dihasilkan tidak seimbang, maka jumlah rekursi yang dilakukan Quick Sort akan semakin banyak dan menyebabkan waktu eksekusi meningkat.

Perbedaan performa paling signifikan terjadi pada kondisi data terurut terbalik. Pada kondisi ini Merge Sort secara signifikan lebih cepat dibandingkan Quick Sort, terutama pada dataset berukuran besar. Hal ini terjadi karena Quick Sort menghasilkan partisi yang kurang seimbang pada data terbalik, sehingga jumlah perbandingan dan rekursi meningkat secara signifikan. Sementara itu, Merge Sort tidak terpengaruh oleh pola distribusi data karena proses pembagian data selalu dilakukan menjadi dua bagian yang sama besar, sehingga performanya tetap stabil.

Hasil penelitian ini menunjukkan bahwa tidak ada satu algoritma yang selalu lebih unggul pada semua kondisi data. Quick Sort lebih unggul pada kondisi data acak karena memiliki waktu eksekusi yang lebih cepat dan penggunaan memori yang lebih efisien. Sebaliknya, Merge Sort lebih unggul pada kondisi data terurut dan data terurut terbalik karena memiliki performa yang lebih stabil dan tidak dipengaruhi oleh distribusi data. Oleh karena itu, pemilihan algoritma sorting yang digunakan dalam sistem nyata harus disesuaikan dengan karakteristik data yang akan diolah serta kebutuhan sistem, apakah lebih memprioritaskan kecepatan eksekusi atau kestabilan performa.

### 4.5 Implikasi Penelitian

Implikasi dari penelitian ini dalam konteks sistem e-commerce adalah bahwa Quick Sort lebih cocok digunakan untuk proses pengurutan data produk secara umum karena sebagian besar data dalam sistem bersifat acak. Namun, untuk kondisi tertentu seperti proses pengurutan data yang sudah hampir terurut atau pengolahan data dalam jumlah sangat besar yang membutuhkan kestabilan performa, Merge Sort dapat menjadi pilihan yang lebih tepat

## 5. KESIMPULAN DAN SARAN

Penelitian ini membuktikan bahwa tidak ada satu pun algoritma pengurutan yang superior secara mutlak untuk seluruh skenario dalam arsitektur e-commerce, karena efisiensi komputasionalnya sangat dipengaruhi oleh distribusi awal data produk. Secara empiris, Quick

Sort terbukti menjadi instrumen yang paling gesit dan hemat memori saat menangani kumpulan data yang tersusun secara acak. Mengingat distribusi acak adalah realitas yang paling mendominasi saat pengguna melakukan pencarian produk sehari-hari, Quick Sort sangat direkomendasikan sebagai arsitektur utama pengurutan katalog. Akan tetapi, ketika interaksi pengguna memicu anomali distribusi seperti memfilter data yang sudah terurut sebagian maupun terurut terbalik secara ekstrem Merge Sort menunjukkan ketangguhan yang jauh lebih unggul. Konsistensi logikanya memberikan jaminan stabilitas performa pada kondisi terburuk, sehingga Merge Sort menjadi pilihan krusial bagi sistem yang memprioritaskan kepastian waktu respons dan tidak memiliki kendala pada kapasitas memori.

Meskipun temuan ini telah memberikan evaluasi yang komprehensif, studi ini menyadari adanya keterbatasan eksperimental karena simulasi difokuskan pada pengurutan berdasarkan atribut tunggal, yakni harga produk. Pada ekosistem digital yang sesungguhnya, platform e-commerce sering kali menuntut pemrosesan berdimensi ganda yang jauh lebih kompleks, seperti menyortir harga termurah yang sekaligus memiliki ulasan tertinggi. Sebagai upaya penyempurnaan, penelitian di masa mendatang sangat disarankan untuk memperluas pengujian menggunakan parameter dataset multi-kriteria guna merepresentasikan beban kerja server yang lebih realistis. Selain itu, eksplorasi lanjutan yang menguji efektivitas algoritma hibrida—sebuah metode yang secara dinamis meleburkan kecepatan partisi Quick Sort dengan stabilitas penggabungan Merge Sort akan menjadi langkah strategis untuk menciptakan sistem komputasi e-commerce yang sepenuhnya cerdas dan adaptif.

## DAFTAR REFERENSI

Aditya, M., Zenitha, H., Wahyu, S., Novala, B., Muhammad, D., & Khudori, A. N. (2025). Performance analysis of sorting algorithm in student data processing. *JESICA: Journal of Enhance Studies in Informatics and Computer Applications*, 2(2), 52-61.

Aditsania, A., & Fathan, F. B. U. (2025). *Membongkar Struktur Data: Solusi Pintar Untuk Permasalahan Komputasi*. PT Mafy Media Literasi Indonesia.

Aftab, A., Ishfaq, H. M., Shujaat, M., Ali, M. A., Ghaffar, A., & Shah, A. U. R. (2021). Review on Performance of Quick Sort Algorithm. *International Journal of Computer Science and Information Security (IJCSIS)*, 19(2), 114–120. https://doi.org/10.5281/zenodo.4602255

Akobre, S., Wiredu, J. K., Aabaah, I., & Wumpini, U. A. (2025). A unified framework for theoretical and experimental evaluation of classical and modern sorting algorithms in real-time systems. *ISI Journal of Information Systems and Informatics*, 7(4).

Ala'anzy, M. A., Tolendi, N., Baubek, B., & Algarni, A. (2026). Performance evaluation of GPU-based parallel sorting algorithms. *PLOS One*, 21(2), e0342167. https://doi.org/10.1371/journal.pone.0342167

Ala'Anzy, M. A., Mazhit, Z., Ala'Anzy, A. F., Algarni, A., Akhmedov, R., & Bauyrzhan, A. (2024, November). Comparative analysis of sorting algorithms: A review. In *2024 11th International Conference on Soft Computing & Machine Intelligence (ISCMI)* (pp. 88-100). IEEE.

Alfina, Ahmad, A., Yanti, Y., & Munawir. (2024). *Buku Ajar Algoritma dan Pemrograman*. CV. Naskah Aceh.

Al-Raimi, M. S., Al-Qurashi, A. A., Albinali, R. S., Aljiban, M. A., Aljuhani, A. S., Noor, M. S., Alzahrani, K. A., & AlGuwaifli, Y. (2024). Merge Sort Sequential and Parallel Programming: Comparative Analysis. *International Journal of Computer Science and Information Security (IJCSIS)*, 22(3), 1–7.

Aqib, S. M., Nawaz, H., & Butt, S. M. (2021). Analysis of merge sort and bubble sort in Python, PHP, JavaScript, and C language. *International Journal*, 10(2).

Axtmann, M., Witt, S., Ferizovic, D., & Sanders, P. (2022). Engineering in-place (shared-memory) sorting algorithms. *ACM Transactions on Parallel Computing*, 9(1), 1-62.

Bahit, M. (2024). *Algoritma Pemrograman Terstruktur*. Poliban Press.

Do, T., Graefe, G., & Naughton, J. (2023). Efficient sorting, duplicate removal, grouping, and aggregation. *ACM Transactions on Database Systems*, 47(4), 1-35.

Durghadevi, P., & Rawal, S. (2024). Comparison of three sorting techniques using design and analysis of algorithms. *International Journal of Combined Research & Development (IJCRD)*, 13(5).

Durrani, O. K., & Abdulhayan, S. (2022). Performance Measurement of Popular Sorting Algorithms Implemented using Java and Python. In *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)* (pp. 1-6). IEEE.

Durrani, O. K., Farooqi, A. S., Chinmai, A. G., & Prasad, K. S. (2022). Performances of sorting algorithms in popular programming languages. In *2022 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)* (pp. 1-7). IEEE.

Fitro, A. (2023). Comparison of Efficiency Data Sorting Algorithms Based on Execution Time. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 9(2), 15–21. https://doi.org/10.32628/CSEIT2390151

Gautam, A., & Naman, A. (2021). Divide and Conquer Sorting Techniques. *International Research Journal of Engineering and Technology (IRJET)*, 8(12), 507–510.

Golonka, J., & Krużel, F. (2025). Empirical Evaluation of Unoptimized Sorting Algorithms on 8-Bit AVR Arduino Microcontrollers. *Sensors*, 26(1), 214. https://doi.org/10.3390/s26010214

Gupta, L., & Chaudhary, A. (2025). Combining Quick Sort and Merge Sort for Improved Average-Case Performance. *International Journal of Advanced Research in Science, Communication and Technology International (IJARSCT)*, 5(5), 598–604. https://doi.org/10.48175/IJARSCT-30076