SAINSTECH
Jurnal Penelitian dan Pengkajian Sains dan Teknologi
Volume 35 No.2
Maret 2025
SINTA 5

Analisis Perbandingan Algoritma Sorting Dalam Javascript Untuk Meningkatkan Efisiensi Pengurutan Produk Pada Aplikasi E-Commerce

Desta Adyangga Saputra[1], Mutiara Sofia Ramadhani[2], M. Azka Failandri[3], Ahmad Turmudi[4], Imam Prayogo Pujiono[5]

[12345]Program Studi Informatika, Fakultas Ekonomi dan Bisnis Islam, UIN K.H. Abdurrahman Wahid Pekalongan
Jl. Kusuma Bangsa No.9 Pekalongan, Jawa Tengah , Indonesia 51141
e-mail: [1]desta.adyangga.saputra24082@mhs.uingusdur.ac.id,
[2]mutiara.sofia.ramadhani24083@mhs.uingusdur.ac.id, [3]m.azka.failandri24052@mhs.uingusdur.ac.id,
[4]ahmad.turmudi24069@mhs.uingusdur.ac.id, [5]imam.prayogopujiono@uingusdur.ac.id

Abstrak

Penelitian ini bertujuan untuk membandingkan efisiensi enam algoritma sorting dalam JavaScript, yaitu Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Heap Sort, pada proses pengurutan produk dalam aplikasi e-commerce. Efisiensi dinilai berdasarkan dua metrik utama: rata-rata waktu eksekusi (dalam milidetik) dan konsumsi memori (dalam kilobyte). Evaluasi dilakukan melalui eksperimen komputasional terhadap dataset acak berukuran 100, 1.000, dan 10.000 item, yang dibuat menggunakan library Faker.Js Pengukuran dilakukan menggunakan fungsi performance.now() untuk waktu eksekusi dan process.memoryUsage().heapUsed untuk memori, dengan tiga kali pengulangan untuk memperoleh nilai rata-rata. Hasil menunjukkan bahwa Quick Sort mencatat waktu eksekusi rata-rata 4,553 ms dan penggunaan memori sebesar 15.891 KB pada dataset 10.000 item, menjadikannya algoritma paling efisien. Merge Sort juga menunjukkan performa stabil dengan waktu eksekusi 9,894 ms dan memori 15.810 KB. Sebaliknya, Bubble Sort dan Selection Sort menghasilkan waktu eksekusi yang jauh lebih tinggi, masing-masing 185.874 ms dan 125.708 ms, serta konsumsi memori terbesar. Temuan ini memberikan acuan praktis bagi pengembang dalam memilih algoritma sorting yang optimal untuk meningkatkan kinerja sistem dan responsivitas aplikasi e-commerce berbasis JavaScript.

Kata kunci: Algoritma Sorting, Aplikasi E-commerce, Efisiensi, Faker.Js, JavaScript

Abstract

This study aims to compare the efficiency of six sorting algorithms in JavaScript Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort for sorting product data in e-commerce applications. Efficiency was evaluated based on two key performance metrics: average execution time (in milliseconds) and memory usage (in kilobytes). A computational experiment was conducted using randomly generated datasets of 100, 1,000, and 10,000 items created with the Faker.Js library. Measurements were taken using performance.now() for execution time and process.memoryUsage(). .heapUsed for memory consumption, with each test repeated three times to obtain the average. The results show that Quick Sort achieved the best performance, with an average execution time of 4.553 ms and a memory usage of 15.891 KB on the 10,000-item dataset. Merge Sort also demonstrated stable performance with 9.894 ms execution time and 15.810 KB of memory usage. In contrast, Bubble Sort and Selection Sort recorded significantly higher execution times of 185.874 ms and 125.708 ms, respectively, along with the highest memory consumption. These findings provide practical guidance for developers in selecting efficient sorting algorithms to enhance system performance and responsiveness in JavaScript-based e-commerce applications.

Keywords: *Sorting Algorithm, E-commerce Application, Efficiency, Faker.Js, JavaScript*

DOI : https://10.37277/stch.v35i2.2337/

1. Pendahuluan

Perkembangan teknologi informasi yang begitu pesat telah membawa perubahan besar dalam berbagai aspek kehidupan manusia (Supriyatna, 2020). Percepatan ini turut mendorong lahirnya berbagai inovasi di beragam disiplin ilmu, termasuk dalam bidang teknologi informasi itu sendiri, yang terus beradaptasi untuk menjawab tantangan zaman (Awaliyah et al., 2024). Salah satu dampak nyata dari transformasi ini terlihat dalam dunia perdagangan, di mana digitalisasi telah menciptakan paradigma baru melalui e-commerce sebagai solusi utama yang menawarkan kemudahan akses, fleksibilitas waktu, dan sistem pembayaran terintegrasi, sehingga menjadi pilihan utama masyarakat modern (Ilyas et al., 2023). Dengan berbagai fitur seperti akses yang mudah, fleksibilitas dalam waktu transaksi, serta integrasi metode pembayaran digital, e-commerce semakin menjadi pilihan utama dalam kehidupan masyarakat modern (Hanani et al., 2024). Seiring dengan meningkatnya jumlah produk dan data pengguna yang dikelola, muncul tantangan dalam menyusun dan menyajikan informasi tersebut secara optimal terutama dalam hal pengurutan produk berdasarkan harga, nama, maupun popularitas (Fitri et al., 2024). Pengurutan yang tepat tidak hanya mempercepat proses pencarian produk oleh pengguna, tetapi juga meningkatkan efisiensi sistem dan kepuasan pelanggan (Iskandar et al., 2023)

Salah satu elemen kunci dalam proses tersebut adalah algoritma sorting yang digunakan oleh sistem (Basir, 2020). Setiap jenis algoritma memiliki pendekatan tersendiri (Syafnidawaty, 2020) terhadap pengolahan data misalnya *Quick Sort* yang dikenal cepat pada dataset besar, dan *Insertion Sort* yang lebih cocok untuk dataset kecil (Pujiono et al., 2024). Penelitian (Rosyadi & Utami, 2023) sebelumnya bahkan menyatakan bahwa pemilihan algoritma pengurutan yang sesuai dapat menurunkan waktu respons aplikasi secara signifikan. Hingga mencapai 40% lebih cepat dibanding algoritma (Poetra, 2022). Dalam studi lain, algoritma *Bubble Sort, Insertion Sort,* dan *Quick Sort* diuji berdasarkan kecepatan dan efisiensi memori menggunakan bahasa *JavaScript*, menunjukkan variasi performa tergantung ukuran data (Wijaya et al., 2024).

Lebih lanjut, (Pujiono et al., 2024) dalam studinya membandingkan tujuh algoritma pengurutan berbasis *Javascript* dan menemukan bahwa efisiensi memori dan waktu sangat tergantung pada ukuran dataset, serta menunjukkan bahwa *Quick Sort* dan *Merge Sort* cenderung stabil dalam skala besar (Rizquina & Ratnasari, 2023). Sedangkan penelitian oleh (Basir, 2020) menunjukkan bahwa algoritma *Heap Sort* memberikan hasil paling konsisten dalam aspek kompleksitas waktu dibandingkan Merge dan *Insertion Sort*, dengan performa stabil meskipun jumlah data meningkat secara signifikan (Salsabilah et al., 2023). Hasil ini semakin menegaskan bahwa kompleksitas algoritma baik dalam aspek ruang maupun waktu harus menjadi pertimbangan utama dalam pengembangan aplikasi pengurutan data.

Penelitian ini bertujuan untuk mengevaluasi performa beberapa algoritma sorting berbasis *JavaScript* dengan fokus pada efisiensi waktu eksekusi dan konsumsi memori pada sisi klien. Pengujian dilakukan menggunakan pendekatan eksperimen dengan dataset acak berukuran mulai dari 100 hingga 10.000 item yang merepresentasikan data produk *e-commerce* (Yanti & Emi Sita Eriana, 2024) Studi ini juga menyesuaikan implementasi algoritma dengan kebutuhan pemrosesan sisi pengguna untuk memastikan kecepatan akses data secara langsung tanpa harus membebani server utama (Poetra, 2022).

Secara praktis, hasil dari penelitian ini diharapkan dapat memberikan kontribusi nyata bagi pengembang aplikasi web dalam memilih strategi algoritma yang tepat untuk pengurutan produk (Siti Aisyah et al., 2022). Temuan ini memperkuat pernyataan (Hanani et al., 2024) yang menyatakan bahwa pemahaman terhadap struktur dasar algoritma pengurutan merupakan dasar penting dalam membangun aplikasi pengelolaan data yang efisien dan akurat

DOI : https://10.37277/stch.v35i2.2337/

(Anggreani et al., 2020). Di sisi lain, optimalisasi algoritma pengurutan juga memberi dampak langsung terhadap strategi penjualan dan kecepatan sistem dalam e-commerce yang terus berkembang, sebagaimana dijelaskan oleh (Harahap & Effendy, 2023) dalam kajiannya mengenai integrasi strategi pemasaran digital dan efisiensi teknis aplikasi (Milal et al., 2023). Dengan pendekatan yang terarah, penelitian ini tidak hanya memperluas pemahaman mengenai algoritma sorting, tetapi juga mendorong penerapannya dalam skenario dunia nyata secara lebih efektif dan terukur.

2. Metode Penelitian

Penelitian ini menggunakan pendekatan kuantitatif dengan metode eksperimen komputasional untuk menganalisis dan membandingkan efisiensi waktu eksekusi dari beberapa algoritma pengurutan (*sorting*) yang di implementasikan dalam bahasa pemrograman *JavaScript*. Fokus utama dari eksperimen ini adalah mengukur dan menganalisis waktu eksekusi algoritma dalam konteks pengurutan data produk pada aplikasi e-commerce.

Objek dalam penelitian ini terdiri dari enam algoritma pengurutan yang umum digunakan, yaitu *Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort*, dan *Heap Sort*. Semua algoritma tersebut di implementasikan secara manual menggunakan *JavaScript* dan dieksekusi di lingkungan Node.js agar proses pengujian berjalan secara optimal tanpa adanya gangguan rendering dari antarmuka pengguna (UI).

Gambar 1. Library *Faker.Js*

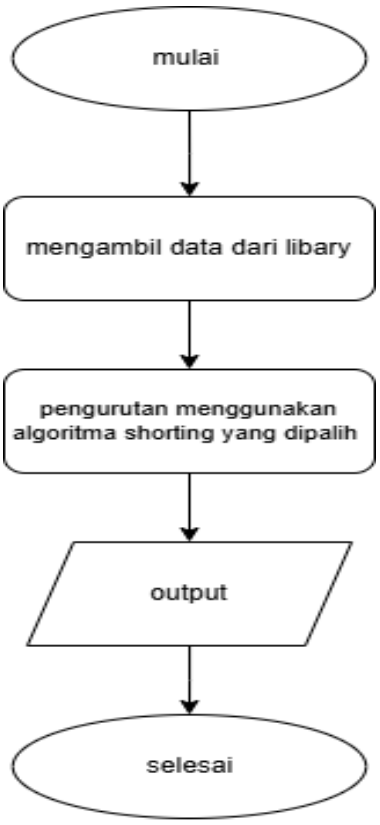
Untuk mendukung proses pengujian, data produk e-commerce dalam penelitian ini dibuat secara acak menggunakan *library Faker.Js*, sebuah library open-source yang banyak digunakan untuk menghasilkan dummy data (data tiruan). Data produk terdiri dari tiga atribut: id, name, dan price. Atribut name dihasilkan dari fungsi *faker.commerce.productName()* yang menghasilkan nama produk acak, sedangkan price berasal dari *faker.commerce.price()* yang menghasilkan harga dalam format desimal. Pembuatan data ini dilakukan melalui fungsi *generateData(size)*, yang memungkinkan program membuat dataset dalam skala yang bervariasi dan menghasilkan data yang banyak sekaligus dengan mudah.

Untuk memperoleh hasil pengujian yang menyeluruh, data dummy dihasilkan dalam tiga variasi skala, yaitu sebanyak 100, 1000, dan 10.000 item. Variasi ini dirancang untuk menguji ketahanan dan efisiensi masing-masing algoritma terhadap peningkatan jumlah data. Data awal bersifat acak untuk mencerminkan kondisi riil aplikasi e-commerce di mana urutan produk tidak selalu konsisten.

Proses pengumpulan data dilakukan dengan mencatat waktu eksekusi dari masing-masing algoritma menggunakan *fungsi performance.now()* atau *console.time()* dalam *JavaScript*. Setiap algoritma diuji dalam dua skenario: pengurutan berdasarkan harga produk (secara ascending) dan pengurutan berdasarkan nama produk (secara alfabetis A-Z). Masing-masing pengujian diulang tiga kali untuk memastikan akurasi data, dan nilai rata-rata waktu eksekusi dari ketiga pengujian digunakan sebagai data akhir.

DOI : https://10.37277/stch.v35i2.2337/

Analisis dalam penelitian ini sepenuhnya difokuskan pada perbandingan efisiensi waktu eksekusi antar algoritma. Hasil yang diperoleh disajikan dalam bentuk tabel dan grafik untuk memberikan gambaran yang jelas mengenai performa relatif dari tiap algoritma dalam setiap skala dan skenario pengurutan. Visualisasi ini bertujuan untuk mempermudah pembaca dalam menarik kesimpulan tentang algoritma mana yang paling efisien dalam pengurutan data produk e-commerce berbasis *JavaScript*.



Gambar 2. Diagram Alur Pemrosesan Algoritma Sorting

Beradasarkan Gambar 2 menyajikan diagran alur yang diawali dari proses inisialisasi lingkungan kerja, termasuk pengaturan library pendukung seperti *Faker.Js* untuk membuat sebuah data set. Langkah selanjutnya adalah proses data generation dengan jumlah masing-masing 100, 1.000, dan 10.000 entri, terdiri atas atribut id, name, dan price. Setelah data dibuat, dilakukan proses seleksi algoritma secara manual dari enam algoritma pengurutan seperti *Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort*, dan Heap Sort. Data yang telah disiapkan kemudian diproses melalui fungsi algoritma sesuai metode yang dipilih. Seluruh proses dijalankan dalam lingkungan *Node.js* tanpa antarmuka pengguna CLI guna memastikan kestabilan eksekusi. Setelah proses pengurutan berlangsung, sistem melakukan pengukuran performa dengan dua parameter utama: waktu eksekusi menggunakan *performance.now()* dan konsumsi memori melalui *process.memoryUsage().heapUsed*.
Pengujian dilakukan sebanyak tiga kali untuk tiap ukuran data, menghasilkan total sembilan percobaan per algoritma. Seluruh hasil pengukuran direkam dan disajikan dalam bentuk tabel serta visualisasi grafik, yang digunakan sebagai dasar evaluasi komparatif antar algoritma berdasarkan efisiensi waktu dan alokasi memori. Rangkaian alur ini dirancang untuk menghasilkan data eksperimental yang dapat dipertanggungjawabkan dan replikatif.

3. Hasil dan Pembahasan

Penelitian ini dilakukan menggunakan bahasa pemrograman NodeJs dengan Visual Studio Code (VSCode) sebagai Integrated Development Environment (IDE) dan ekstensi Code Runner untuk menjalankan kode program. Compiler yang digunakan adalah *NodeJs* untuk memastikan kompatibilitas eksekusi dan dokumentasi hasil pengujian dilakukan menggunakan Snipping Tool untuk mengambil screenshot. Spesifikasi perangkat yang digunakan adalah:

1. Sistem Operasi: Windows 11 Pro
2. RAM: 16 GB
3. CPU: Intel Core i3-12100F (@3.30 GHZ)
4. Penyimpanan: SSD 512 GB

Penelitian ini mengevaluasi kinerja enam algoritma pengurutan, yakni *Bubble Sort,*

DOI : https://10.37277/stch.v35i2.2337/

Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort, Semua algoritma tersebut diuji dengan menggunakan dataset yang telah distandarisasi dan dikelompokkan ke dalam tiga kategori ukuran: ArrayKecil terdiri dari 100 elemen, ArraySedang mencakup 1.000 elemen, dan ArrayBesar memuat 10.000 elemen. Data yang digunakan dihasilkan secara acak dari library *Faker.Js* untuk menjaga objektivitas pengujian lalu salah satu algoritma dipilih untuk melakukan pengujian. Dalam protokol pengujian, setiap algoritma dijalankan sebanyak tiga kali pada masing-masing kategori untuk memastikan konsistensi hasil. Secara rinci, percobaan untuk dataset 100 elemen dilakukan pada uji ke-1 hingga ke-3, untuk dataset 1.000 elemen pada uji ke-4 hingga ke-6, dan untuk dataset 10.000 elemen pada uji ke-7 hingga ke-9. Selama pengujian, hanya dua aplikasi yang aktif. Visual Studio Code untuk eksekusi kode dan Snipping Tools untuk dokumentasi visual melalui screenshot untuk meminimalkan gangguan pada CPU.

Berikut adalah penjelasan dari pengujian tiap algoritma.

3.1 Bubble Sort

// Fungsi *Bubble Sort* melakukan perbandingan antar-elemen secara berulang dan menukar pasangan elemen yang keliru urutannya, hingga array terutama.

```
function bubbleSort(arr, key) {
  for (let i = 0; i < arr.length - 1; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j][key] > arr[j + 1][key]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
}
```

Gambar 3. Kode function dari algortima *Bubble Sort*

Pada kode tersebut, algoritma *Bubble Sort* di implementasikan dalam bahasa *Javascript*. Data set yang dipakai telah dimasukkan langsung ke dalam program dan dioperasikan menurut skema pengujian yang sudah ditentukan. Pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam memanfaatkan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan melibatkan 10.000 data (Dataset Besar). Hasil pengukuran waktu komputasi dan konsumsi memori selama proses pengurutan dengan *Bubble Sort* dapat diamati secara lengkap pada Gambar 4.

```
# Uji Performa Algoritma: BUBBLE Sort
------------------------------------------------
Percobaan 1 - Data 100 →Time: 0.579 ms | Memory: 1.223 KB
Percobaan 2 - Data 100 →Time: 0.657 ms | Memory: 1.311 KB
Percobaan 3 - Data 100 →Time: 0.546 ms | Memory: 1.294 KB
Percobaan 4 - Data 1000 →Time: 2.143 ms | Memory: 5.735 KB
Percobaan 5 - Data 1000 →Time: 1.550 ms | Memory: 5.842 KB
Percobaan 6 - Data 1000 →Time: 3.842 ms | Memory: 5.798 KB
Percobaan 7 - Data 10000 →Time: 185.545 ms | Memory: 15.967 KB
Percobaan 8 - Data 10000 →Time: 190.697 ms | Memory: 15.893 KB
Percobaan 9 - Data 10000 →Time: 181.379 ms | Memory: 15.921 KB
```

Gambar 4. Pengujian data (100, 1000, 10000) dalam 3 kali percobaan per data

Berdasarkan data pada Gambar 4, waktu eksekusi rata-rata untuk algoritma *Bubble Sort* pada data sebanyak 100 adalah 0.594 ms dan rata rata penggunaan memori nya adalah 1.276 kb. Nilai ini meningkat menjadi 2.512 ms dan penggunaan memori juga meningkat sebesar 5.792 kb saat data diperbesar menjadi 1000, dan meningkat pesat menjadi rata-rata 185.874 ms untuk waktu eksekusinya dan rata rata penggunaan memori 15.927 kb saat data diperbesar menjadi 10000 data.

3.2 Selection Sort

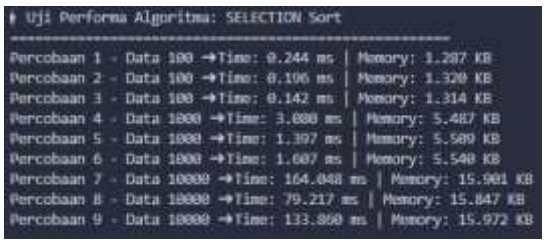
// Fungsi *Selection Sort* mencari elemen terkecil dan menempatkannya di posisi awal, dilanjutkan ke posisi berikutnya hingga array terurut.

```
function selectionSort(arr, key) {
  for (let i = 0; i < arr.length - 1; i++) {
    let minIdx = i;
    for (let j = i + 1; j < arr.length; j++) {
      if (arr[j][key] < arr[minIdx][key]) minIdx = j;
    }
    [arr[i], arr[minIdx]] = [arr[minIdx], arr[i]];
  }
  return arr;
}
```

DOI : https://10.37277/stch.v35i2.2337/

Gambar 5. Kode function dari algortima *Selection Sort*

Pada kode tersebut, algoritma *Bubble Sort* di implementasikan dalam bahasa *Javascript*. Data set yang dipakai telah dimasukkan langsung ke dalam program dan dioperasikan menurut skema pengujian yang sudah ditentukan. Pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam memanfaatkan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan melibatkan 10.000 data (Dataset Besar). Hasil pengukuran waktu komputasi dan konsumsi memori selama proses pengurutan dengan *Selection Sort* dapat diamati secara lengkap pada Gambar 6.

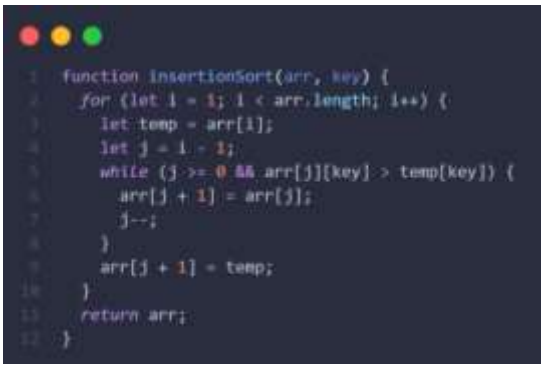


Gambar 6. Pengujian data (100, 1000, 10000) dalam 3 kali percobaan per data

Berdasarkan data pada Gambar 6, waktu eksekusi rata-rata untuk algoritma *Selection Sort* pada data sebanyak 100 adalah 0.194 ms dan rata rata penggunaan memori nya adalah 1.307 kb. Nilai ini meningkat menjadi 2.028 ms dan penggunaan memori juga meningkat sebesar 5.512 kb saat data diperbesar menjadi 1000, dan meningkat pesat menjadi rata-rata 125.708 ms untuk waktu eksekusinya dan rata rata penggunaan memori 15.907 kb saat data diperbesar menjadi 10000 data. Pada setiap percobaan data tersebut menunjukan angka yang sedikit tidak konsisten di setiap percobaanya.

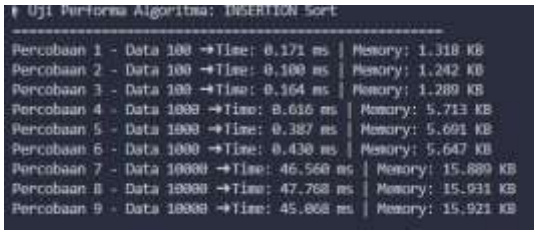
3.3 Insertion Sort

// Fungsi *Insertion Sort* menyisipkan elemen satu per satu pada posisi yang tepat dengan cara menggeser elemen yang lebih besar ke kanan



Gambar 7. Kode function dari algortima *Insertion Sort*

Pada kode tersebut, algoritma *Insertion sort* di implementasikan dalam bahasa *Javascript*. Data set yang dipakai telah dimasukkan langsung ke dalam program dan dioperasikan menurut skema pengujian yang sudah ditentukan. Pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam memanfaatkan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan melibatkan 10.000 data (Dataset Besar). Hasil pengukuran waktu komputasi dan konsumsi memori selama proses pengurutan dengan *Insertion Sort* dapat diamati secara lengkap pada Gambar 8.



Gambar 8. Pengujian data (100, 1000, 10000) dalam 3 kali percobaan per data

Berdasarkan data pada Gambar 8, waktu eksekusi rata-rata untuk algoritma *Insertion Sort* pada data sebanyak 100 adalah 0.145 ms dan rata rata penggunaan memori nya adalah 1.283 kb. Nilai ini meningkat menjadi 0.478 ms dan penggunaan memori juga meningkat sebesar 5.684 kb saat data diperbesar menjadi 1000, dan meningkat pesat menjadi rata-rata 46.465 ms untuk waktu eksekusinya dan rata rata penggunaan memori 15.914 kb saat data diperbesar menjadi 10000 data. Jika dilihat dari data tersebut perubahan yang terjadi dari 100 ke 1000 data tidak terlalu

DOI : https://10.37277/stch.v35i2.2337/

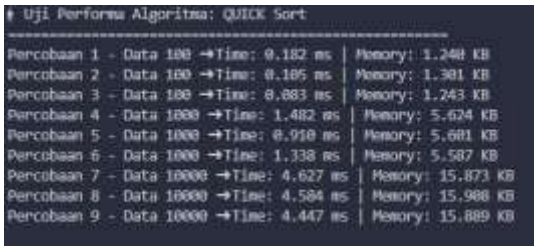
signifikan tetapi berubah pesat saat data nya diperbesar menjadi 10000 data.

3.4 Quick Sort

// Fungsi *Quick Sort* menempatkan pivot di posisi yang benar di dalam array dengan cara memindah elemen yang lebih kecil pivot ke kiri, dan yang lebih besar ke kanan.

Gambar 9. Kode function dari algortima *Quick Sort*

Pada kode tersebut, algoritma *Quick Sort* di implementasikan dalam bahasa *Javascript.* Data set yang dipakai telah dimasukkan langsung ke dalam program dan dioperasikan menurut skema pengujian yang sudah ditentukan. Pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam memanfaatkan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan melibatkan 10.000 data (Dataset Besar). Hasil pengukuran waktu komputasi dan konsumsi memori selama proses pengurutan dengan *Quick Sort* dapat diamati secara lengkap pada Gambar 10.



Gambar 10. Pengujian data (100, 1000, 10000) dalam 3 kali percobaan per data

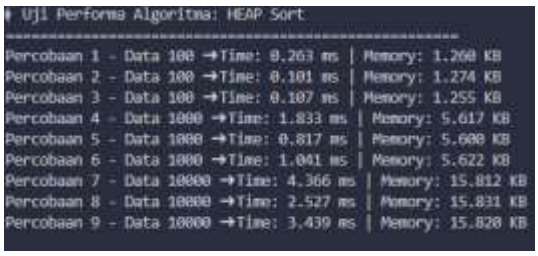
Berdasarkan data pada Gambar 10, waktu eksekusi rata-rata untuk algoritma *Quick Sort* pada data sebanyak 100 adalah 0.123 ms dan rata rata penggunaan memori nya adalah 1.261 kb. Nilai ini meningkat menjadi 1.243 ms dan penggunaan memori juga meningkat sebesar 5.604 kb saat data diperbesar menjadi 1000, dan menjadi 4.553 ms untuk waktu eksekusinya dan rata rata penggunaan memori 15.891 kb saat data diperbesar menjadi 10000 data. Jika dilihat dari data tersebut data perbandingan waktu eksekusi 1000 dan 10000 data tidak terlalu signifikan.

3.5 Merge Sort

// Fungsi merge menggabungkan dua sub-array terurut menjadi satu secara berurutan

Gambar 11. Kode function dari algortima *Merge Sort*

Pada kode tersebut, algoritma *Merge sort* di implementasikan dalam bahasa *Javascript.* Data set yang dipakai telah dimasukkan langsung ke dalam program dan dioperasikan menurut skema pengujian yang sudah ditentukan. Pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam memanfaatkan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan melibatkan 10.000 data (Dataset Besar). Hasil pengukuran waktu komputasi dan konsumsi memori selama proses pengurutan dengan *Merge Sort* dapat diamati secara lengkap pada Gambar 12.



Gambar 12. Pengujian data (100, 1000, 10000) dalam 3 kali percobaan per data

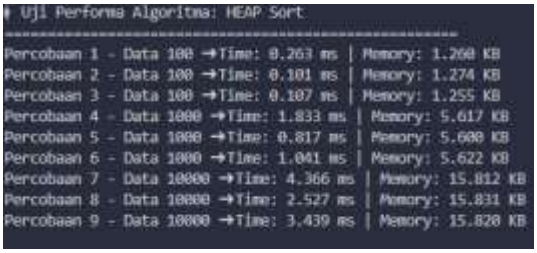
Berdasarkan data pada Gambar 12, waktu eksekusi rata-rata untuk algoritma *Merge Sort* pada data sebanyak 100 adalah 0.118 ms dan rata rata penggunaan memori nya adalah

DOI : https://10.37277/stch.v35i2.2337/

1.246 kb. Nilai ini meningkat menjadi 1.854 ms dan penggunaan memori juga meningkat sebesar 5.417 kb saat data diperbesar menjadi 1000, dan menjadi 9.894 ms untuk waktu eksekusinya dan rata rata penggunaan memori 15.810 kb saat data diperbesar menjadi 10000 data. Jika dilihat dari data tersebut terjadi perubahanya cukup signifikan setiap datanya.

3.6 Heap Sort

// Fungsi *Heap Sort* membangun max-heap di sekitar indeks i void heapify



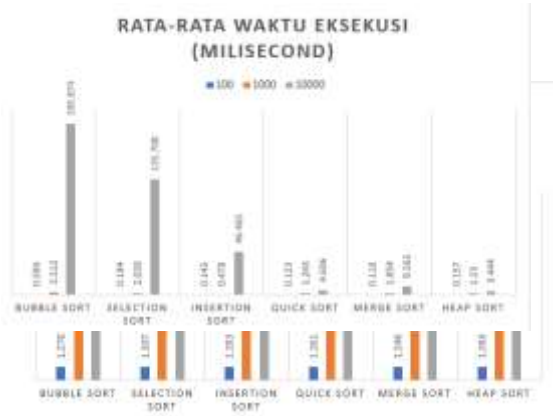
Gambar 13. Kode function dari algortima *Heap Sort*

Pada kode tersebut, algoritma *Heap Sort* di implementasikan dalam bahasa *Javascript*. Data set yang dipakai telah dimasukkan langsung ke dalam program dan dioperasikan menurut skema pengujian yang sudah ditentukan. Pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam memanfaatkan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan melibatkan 10.000 data (Dataset Besar). Hasil pengukuran waktu komputasi dan konsumsi memori selama proses pengurutan dengan *Heap Sort* dapat diamati secara lengkap pada Gambar 14.

Gambar 14. Pengujian data (100, 1000, 10000) dalam 3 kali percobaan per data

Berdasarkan data pada Gambar 14. waktu eksekusi rata-rata untuk algoritma *Heap Sort* pada data sebanyak 100 adalah 0.157 ms dan rata rata penggunaan memori nya adalah 1.263 kb. Nilai ini meningkat menjadi 1.230 ms dan penggunaan memori juga meningkat sebesar 5.613 kb saat data diperbesar menjadi 1000, dan menjadi 3.444 ms untuk waktu eksekusinya dan rata rata penggunaan memori 15.821 kb saat data diperbesar menjadi 10000 data. Jika dilihat dari data tersebut waktu eksekusinya tidak stabil setiap percobaanya.

Berdasarkan hasil pengujian algoritma yang sudah di uji, berikut ini adalah rangkuman rata-rata nilai yang di dapat untuk setiap algoritma dalam bentuk gambar grafik dibawah ini.



Gambar 15. Rata-rata Waktu Eksekusi pada 6 algoritma sorting

Gambar 16. Rata-rata Penggunaan Memori pada 6 algoritma sorting

DOI : https://10.37277/stch.v35i2.2337/

Berdasarkan gambar 15 dan gambar 16 pada diagram tersebut hasil perbandingan waktu eksekusi dari enam algoritma sorting (*Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort,* dan *Heap Sort*) dalam tiga ukuran dataset yang berbeda: 100, 1.000, dan 10.000 item.

Hasil evaluasi menunjukkan bahwa:

1. *Bubble Sort* dan *Selection Sort* memiliki waktu eksekusi tertinggi di semua skala data, terutama pada dataset besar (10.000 item), di mana waktu eksekusi keduanya melonjak drastis. Hal ini mencerminkan kelemahan utama algoritma dengan kompleksitas $O(n^2)$ yang tidak efisien untuk data besar dan penggunaan memori relatif kecil namun meningkat secara linear dengan jumlah data. Ini disebabkan karena algoritma ini bekerja secara in-place tanpa banyak alokasi memori tambahan.
2. *Insertion Sort* sedikit lebih cepat dari *Bubble* dan *Selection*, terutama pada dataset kecil dan sedang, namun tetap tidak optimal pada dataset besar dan memiliki penggunaan sama seperti *Bubble Sort* dan *Selection Sort*.
3. *Quick Sort* dan *Merge Sort* tampil sebagai algoritma dengan kinerja terbaik. *Quick Sort* secara konsisten menunjukkan waktu eksekusi paling rendah di semua ukuran dataset, disusul oleh *Merge Sort*. Keduanya memperlihatkan kestabilan dan efisiensi yang konsisten, sejalan dengan kompleksitas waktu O(n log n). *Merge Sort* memiliki penggunaan memori yang sedikit lebih tinggi dibanding algoritma lainnya karena memerlukan alokasi array tambahan selama proses penggabungan (*merge*). Ini sesuai dengan sifatnya sebagai algoritma divide-and-conquer yang tidak bekerja sepenuhnya in-place. Sedangkan *Quick Sort* memiliki konsumsi memori yang efisien dan stabil, bahkan pada dataset besar. Meskipun menggunakan rekursi, implementasi yang tepat memungkinkan penggunaan memori tetap terkendali.
4. *Heap Sort* menunjukkan performa yang baik, berada di antara *Merge Sort* dan *Insertion Sort*, meskipun tidak secepat *Quick Sort*. *Heap Sort* menunjukkan efisiensi memori yang baik dan relatif konsisten di semua ukuran data.

4. Kesimpulan

Penelitian ini telah mengevaluasi performa enam algoritma pengurutan *Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort, dan Heap Sort* berdasarkan waktu eksekusi dan konsumsi memori terhadap tiga skala data 100, 1.000, dan 10.000 item, dan menghasilkan temuan bahwa *Quick Sort* memberikan performa paling optimal dengan waktu rata-rata tercepat sebesar 0,123 ms (100 data), 1,243 ms (1.000 data), dan 4,553 ms (10.000 data), serta penggunaan memori yang efisien (15,30 KB pada 10.000 data), menjadikannya sebagai algoritma paling layak diterapkan dalam sistem e-commerce berskala besar dan dinamis. *Merge Sort* menunjukkan stabilitas performa dengan konsumsi memori yang konsisten dan waktu eksekusi yang relatif cepat (9,894 ms pada 10.000 data), menjadikannya pilihan ideal saat kestabilan proses pengurutan lebih diutamakan, terutama untuk data yang telah sebagian terurut. *Heap Sort* menempati posisi menengah dengan efisiensi cukup baik, sementara *Insertion Sort* efektif hanya pada dataset kecil namun tidak efisien pada data besar. Sebaliknya, *Bubble Sort* dan *Selection Sort* mencatatkan waktu eksekusi paling lambat dan konsumsi memori tertinggi, sehingga tidak direkomendasikan dalam implementasi praktis yang membutuhkan efisiensi tinggi. Dengan demikian, pemilihan algoritma sorting yang tepat, seperti *Quick Sort*, terbukti memberikan dampak signifikan terhadap kecepatan pencarian dan efisiensi pemrosesan data dalam sistem digital yang kompleks (Agustin, 2023).

Daftar Pustaka

Anggreani, D., Wibawa, A. P., Purnawansyah, P., & Herman, H. (2020). Perbandingan Efisiensi Algoritma Sorting dalam Penggunaan Bandwidth. *ILKOM Jurnal Ilmiah*, *12*(2), 96–103. https://doi.org/10.33096/ilkom.v12i2.538.96-103

Awaliyah, D. A., Prasetiyo, B., Muzayanah, R., & Lestari, A. D. (2024). Optimizing Customer Segmentation in Online Retail Transactions through the Implementation of the K-Means Clustering Algorithm. *Scientific Journal of Informatics*, *11*(2), 539–548. https://doi.org/10.15294/sji.v11i2.6137

Basir, R. R. (2020). Analisis Kompleksitas Ruang dan Waktu Terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort dan Merge dengan Pemrograman Java. *STRING (Satuan Tulisan Riset Dan Inovasi Teknologi)*, *5*(2), 109. https://doi.org/10.30998/string.v5i2.6250

Fitri, K., Affandi, A., Situngkir, W. J., & ... (2024). Sosialisasi Dan Pendampingan Penggunaan E-Commerce Dalam Meningkatkan Pemasaran Produk Lokal Didesa Bukit Jaya Kabupaten Pelalawan. *ARSY: Jurnal ….* http://www.journal.al-matani.com/index.php/arsy/article/view/1006

Hanani, F., Sukma, A., Sesoleh, C., Zahra, D., Ekaputri, A., Isabella, S., Fika, A., Nisriinaa, R., Ramadhani, N. O., Putri, Y. A., & Khoerunnisa, A. (2024). *Pengaruh Kemudahan Transaksi e-Payment terhadap Keputusan Pembelian Produk UMKM oleh Mahasiswa Universitas Negeri Semarang*. *1*(2), 322–333.

Harahap, F., & Effendy, I. (2023). Implementasi Algoritma Selection Sort dalam Membangun Aplikasi Android Pemesanan Jasa Make-up. *Jurnal JUPITER*, *15*(1), 61–72.

Ilyas, I., Achmady, S., & Salat, J. (2023). Aplikasi E-Comerce Dalam Pemasaran Kopi Berbasis Android Di Kabupaten Pidie. *Jurnal Real Riset*, *5*(2), 359–364. https://doi.org/10.47647/jrr.v5i2.1178

Iskandar, J., Suhendar, H., & Pamungkas, B. D. (2023). Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python. *G-Tech: Jurnal Teknologi Terapan*, *8*(1), 104–113. https://doi.org/10.33379/gtech.v8i1.3556

Milal, I. S., M. Hasanudin, M. H., Nur Azhari, M. A., Nugraha, R. A., Agustina, N., & Damayanti, S. E. (2023). Klasifikasi Teks Review Pada E-Commerce Tokopedia Menggunakan Algoritma Svm. *Naratif : Jurnal Nasional Riset, Aplikasi Dan Teknik Informatika*, *5*(1), 34–45. https://doi.org/10.53580/naratif.v5i1.191

Poetra, D. R. (2022). Performa Algoritma Bubble Sort dan Quick Sort pada Framework Flutter dan Dart SDK(Studi Kasus Aplikasi E-Commerce). *JATISI (Jurnal Teknik Informatika Dan Sistem Informasi)*, *9*(2), 806–816. https://doi.org/10.35957/jatisi.v9i2.1886

Poetra, D. R. (2022). Performa Algoritma Bubble Sort dan Quick Sort pada Framework Flutter dan Dart SDK(Studi Kasus Aplikasi E-Commerce). *JATISI (Jurnal Teknik Informatika Dan Sistem Informasi)*, *9*(2), 806–816. https://doi.org/10.35957/jatisi.v9i2.1886

Pujiono, I. P., Trianto, R. B., & Hana, F. M. (2024). Perbandingan Efisiensi Memori dan Waktu Komputasi Pada 7 Algoritma Sorting Menggunakan Bahasa Pemrograman Java. *Simkom*, *9*(2), 218–230. https://doi.org/10.51717/simkom.v9i2.481

Rizquina, A. Z., & Ratnasari, C. I. (2023). Implementasi Web Scraping untuk Pengambilan Data Pada Website E-Commerce. *Jurnal Teknologi Dan Sistem Informasi Bisnis*, *5*(4), 377–383. https://doi.org/10.47233/jteksis.v5i4.913

Rosyadi, Q. D., & Utami, A. W. (2023). Implementasi Algoritma Frequent Pattern Growth untuk Menentukan Pola Pembelian Konsumen pada Toko Tanaman Berbasis Website. 04(03),

107–114.

Salsabilah, B. N., Kadek, I., & Nuryana, D. (2023). Komparasi Algoritma Naive Bayes dan K Nearest Neighbor dalam Kepuasan Pengguna Fitur Tiktok Shop. *Journal of Emerging Information System and Business Intelligence (JEISBI)*, *4*(3), 31–39. https://ejournal.unesa.ac.id/index.php/JEISBI/article/view/54211%0Ahttps://ejournal.unesa.ac.id

Siti Aisyah, Muhammad Iqbal Sayuti Harahap, Alfi Hidayat, Miftahul Jannah, & Dian Irmawani. (2022). Pengenalan E-Comerce Shopee Dalam Upaya Meningkatkan Penjualan Basreng Sua Snack Tembung Medan. *Sejahtera: Jurnal Inspirasi Mengabdi Untuk Negeri*, *2*(1), 13–22. https://doi.org/10.58192/sejahtera.v2i1.378

Supriyatna, S. (2020). *Perancangan Dan Implementasi Aplikasi Monitoring Berkas Pencairan Dana Berbasis Web Menggunakan Metode Rapid Application Development (RAD)*. *34*(3), 1–8.

Syafnidawaty. (2020). Pengertian Logika Fuzzy. *06 April 2020*, *II*(September), 3. https://raharja.ac.id/2020/04/06/logika-fuzzy/

Wijaya, S., Fauziah, F., & Harjanti, T. W. (2024). Perbandingan Algoritma Sorting dengan Menggunakan Bahasa Pemograman Javascript dalam Penggunaan Waktu Komputasi dan Penggunaan Memori. *STRING (Satuan Tulisan Riset Dan Inovasi Teknologi)*, *8*(3), 294. https://doi.org/10.30998/string.v8i3.17972

Yanti, F., & Emi Sita Eriana, Mk. (2024). *Algoritma Sorting Dan Searching*. *1*, 1–122.