SMASH: Journal of Social Management Sains and Health
E-ISSN: 3089-252X P-ISSN: 3089-2538
Vol. 3 No. 1 April 2026

Implementasi dan Visualisasi Interaktif Algoritma Sorting Serta Analisis Performa Berbasis Python

Syuhada Simbolon[1] Micael Zecsen Saragih[2] Dwi Nina P Anakampun[3] Khairida Octavia Ramadhani[4]
Program Studi Ilmu Komputer, Universitas Negeri Medan, Kabupaten Deli Serdang, Provinsi Sumatra Utara, Indonesia[1,2,3,4]
Email: syuhadasimbolon@gmail.com[1] micaelzecsen.4243250056@mhs.unimed.ac.id[2] dwininaputrianakampun12@gmail.com[3] khairidaoctavia@gmail.com[4]

Abstrak

Algoritma sorting merupakan salah satu materi dasar dalam ilmu komputer yang digunakan untuk mengurutkan data. Namun, dalam proses pembelajarannya, mahasiswa sering mengalami kesulitan dalam memahami cara kerja algoritma secara langsung karena penyampaian yang cenderung bersifat teoritis. Oleh karena itu, penelitian ini bertujuan untuk mengimplementasikan serta memvisualisasikan algoritma sorting secara interaktif agar lebih mudah dipahami. Pada penelitian ini digunakan beberapa algoritma sorting, yaitu Bubble Sort, Selection Sort, Insertion Sort, dan Quick Sort yang diimplementasikan menggunakan bahasa pemrograman Python. Sistem yang dibuat dilengkapi dengan fitur visualisasi berbentuk grafik batang, pengaturan jumlah data, kontrol kecepatan animasi, mode step-by-step, serta analisis performa berupa jumlah perbandingan dan pertukaran data. Hasil pengujian menunjukkan bahwa penggunaan visualisasi interaktif sangat membantu dalam memperjelas tahapan algoritma yang sebelumnya sulit dipahami. Berdasarkan analisis performa pada dataset kecil ($n=5$), *Insertion Sort* mencatat waktu eksekusi paling optimal dibandingkan algoritma lainnya. Meski demikian, hasil penelitian juga menegaskan bahwa *Quick Sort* tetap menunjukkan potensi efisiensi yang lebih baik untuk penanganan skala data yang lebih kompleks sesuai dengan landasan teoritisnya. Secara keseluruhan, sistem ini tidak hanya efektif sebagai media pembelajaran, tetapi juga mampu menjadi alat bantu analisis perbandingan algoritma bagi pengguna.
Kata Kunci: Algoritma Sorting, Visualisasi Interaktif, Bubble Sort, Selection Sort, Insertion Sort;, Quick Sort, Python

Abstract

Sorting algorithms are among the fundamental topics in computer science used to organize data. However, in the learning process, students often find it difficult to understand how these algorithms work because the material is typically presented in a purely theoretical manner. Therefore, this study aims to implement and visualize sorting algorithms interactively to make them easier to comprehend. Several sorting algorithms—Bubble Sort, Selection Sort, Insertion Sort, and Quick Sort—were implemented using the Python programming language. The developed system features bar chart visualizations, adjustable data sizes, animation speed control, a step-by-step mode, and performance analysis including the number of comparisons and swaps. The results show that the use of interactive visualization significantly helps in clarifying algorithmic stages that were previously difficult to understand. Based on the performance analysis of a small dataset ($n=5$), Insertion Sort recorded the most optimal execution time compared to the other algorithms. Nevertheless, the findings also confirm that Quick Sort maintains the potential for better efficiency when handling more complex data scales, in line with its theoretical foundation. Overall, this system serves not only as an effective learning tool but also as a medium for analyzing and comparing the performance of sorting algorithms.
Keywords: *Sorting Algorithms, Interactive Visualization, Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Python*

PENDAHULUAN

Perkembangan teknologi informasi yang semakin pesat menuntut pengolahan data yang cepat dan efisien. Salah satu proses dasar dalam pengolahan data adalah sorting atau pengurutan data. Sorting digunakan dalam berbagai bidang, seperti pencarian data, pengolahan database, hingga analisis data. Lebih jauh lagi, prinsip pengurutan juga banyak diaplikasikan dalam penyelesaian masalah pengaturan informasi kompleks di dunia nyata, seperti pengaturan tata letak visual atau kolase gambar[1]. Oleh karena itu, pemahaman terhadap algoritma sorting menjadi hal yang penting dalam ilmu komputer. Terdapat berbagai jenis algoritma sorting yang sering digunakan, seperti Bubble Sort, Selection Sort, Insertion Sort, dan Quick Sort. Masing-masing algoritma memiliki cara kerja dan tingkat efisiensi yang berbeda-beda.Memahami algoritma tradisional sangat penting karena algoritma ini memproses data secara seragam, dan teknik pengurutan di tempat (*in-place sorting*) sangat krusial untuk meminimalkan penggunaan memori tambahan[2]. Namun, dalam proses pembelajaran, mahasiswa sering mengalami kesulitan dalam memahami bagaimana algoritma tersebut bekerja secara detail, terutama ketika hanya dijelaskan melalui teori atau kode program saja. Oleh karena itu, penggunaan representasi visual sangat membantu pemahaman mahasiswa dengan mendemonstrasikan secara jelas bagaimana algoritma tersebut berfungsi secara langkah demi langkah[3].

Untuk mengatasi permasalahan tersebut, diperlukan suatu pendekatan yang dapat membantu memvisualisasikan proses algoritma secara lebih jelas. Visualisasi interaktif menjadi salah satu solusi yang dapat digunakan karena mampu menampilkan proses algoritma secara langsung dalam bentuk animasi. Dengan adanya visualisasi, pengguna dapat melihat bagaimana data dibandingkan, dipindahkan, dan diurutkan secara bertahap. Selain itu, tidak hanya pemahaman konsep yang penting, tetapi juga analisis terhadap performa algoritma. Setiap algoritma memiliki kompleksitas waktu yang berbeda, sehingga perlu dilakukan perbandingan untuk mengetahui algoritma mana yang lebih efisien dalam kondisi tertentu. Efisiensi sebuah algoritma sangat penting dan dinilai dari aspek kebutuhan waktu komputasi yang sesingkat mungkin untuk menyelesaikan masalah[4]. Analisis performa ini akan selalu relevan mengingat algoritma pengurutan konvensional seringkali menghadapi keterbatasan dalam hal efisiensi dan adaptabilitas ketika dihadapkan pada data modern yang berskala besar dan dinamis[5].Oleh karena itu, penelitian ini tidak hanya berfokus pada visualisasi, tetapi juga pada analisis performa berdasarkan jumlah perbandingan dan pertukaran data. Berdasarkan latar belakang tersebut, penelitian ini bertujuan untuk mengimplementasikan dan memvisualisasikan algoritma sorting secara interaktif serta menganalisis performanya menggunakan bahasa pemrograman Python. Diharapkan sistem yang dikembangkan dapat membantu proses pembelajaran sekaligus menjadi media analisis dalam membandingkan efisiensi algoritma sorting.

Landasan Teori

1. Algoritma. Algoritma adalah kumpulan langkah sistematis untuk memperoleh hasil yang diinginkan. Sebelum sebuah algoritma dijalankan, biasanya ada suatu kondisi awal (initial state) yang harus dipenuhi. Kemudian, langkah-langkah ini diproses hingga mencapai suatu kondisi akhir (final state)
2. Algoritma Sorting. Sorting didefinisikan sebagai pengurutan sejumlah data berdasarkan nilai kunci tertentu. Pengurutan dapat dilakukan dari nilai terkecil ke nilai terbesar (ascending) atau sebaliknya (descending).
   a. Bubble Sort. *Bubble Sort* merupakan cara pengurutan yang sederhana. Konsep dari ide dasarnya adalah seperti “gelembung air” untuk elemen struktur data yang semestinya

berada pada posisi awal. Cara kerjanya adalah dengan berulang-ulang melakukan traversal (proses looping) terhadap elemen-elemen struktur data yang belum diurutkan. Di dalam traversal tersebut, nilai dari dua elemen struktur data dibandingkan. Jika ternyata urutannya tidak sesuai dengan "pesanan", maka dilakukan pertukaran (swap). Algoritma sorting ini disebut juga dengan comparison sort dikarenakan hanya mengandalkan perbandingan nilai elemen untuk mengoperasikan elemennya.

b. Selection Sort. *Selection sort* adalah metode pengurutan data yang membandingkan item saat ini dengan item berikutnya dengan item terbaru atau item terakhir. Jika elemen lain ditemukan lebih kecil dari elemen saat ini, ia dihapus dari posisinya dan segera ditukar atau diganti. Metode selection sort adalah memilih nilai minimum, dan kemudian bertukar dengan item pertama, Kemudian bandingkan elemen saat ini dengan item berikutnya, dan terus bandingkan sampai tidak ada pertukaran data.Tujuan utama dari metode pemilahan data ini adalah untuk mengorganisir informasi sehingga mempermudah proses pencarian data pada saat dibutuhkan[6].
c. Insertion Sort. *Insertion Sort* bekerja dengan cara membandingkan dan menyisipkan elemen data ke posisi yang tepat secara berurutan, sehingga cocok digunakan dalam aplikasi pengolahan data sederhana. Algoritma bekerja dengan menyisipkan elemen pada posisi yang sesuai, menyerupai proses menyusun kartu secara manual.
d. Quick Sort. *Quick Sort* merupakan algoritma pengurutan yang menggunakan strategi *divide and conquer*.Keunggulan utama dari algoritma ini adalah efisiensi dalam penggunaan memori serta kemampuannya untuk melakukan pengurutan dengan sangat cepat dibandingkan dengan algoritma pengurutan lainnya[7]. Algoritma ini bekerja dengan memilih satu elemen sebagai pivot untuk mempartisi barisan data menjadi dua bagian. Dalam implementasi penelitian ini, *pivot* dipilih dari elemen terakhir dari *sub-array* yang sedang diproses.

Setelah *pivot* ditentukan, variabel penunjuk akan bergerak untuk membandingkan setiap elemen dengan *pivot*. Elemen-elemen yang memiliki nilai lebih kecil dari *pivot* akan dipindahkan ke sebelah kiri, sementara elemen yang lebih besar akan berada di sebelah kanan *pivot*. Proses pemisahan atau partisi ini dilakukan secara rekursif pada bagian kiri dan kanan hingga seluruh data mencapai posisi finalnya dan terurut secara sempurn

METODE PENELITIAN

Jenis Penelitian

Jenis penelitian yang digunakan dalam penelitian ini adalah penelitian eksperimen. Penelitian ini dilakukan dengan cara mengimplementasikan beberapa algoritma sorting ke dalam sebuah program, kemudian melakukan pengujian untuk membandingkan performa masing-masing algoritma berdasarkan parameter tertentu seperti waktu eksekusi dan proses yang terjadi selama sorting berlangsung.

Objek Penelitian

Objek penelitian dalam penelitian ini adalah algoritma sorting, yaitu Bubble Sort, Selection Sort, Insertion Sort, dan Quick Sort, serta proses visualisasi dan analisis performa dari masing-masing algoritma tersebut dalam sebuah program berbasis Python. Secara detail metodologi penelitian ini dirancang seperti diagram blok pada Gambar berikut:

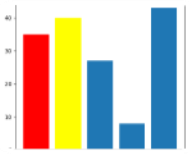
a. Identifikasi Masalah. Tahap ini dilakukan untuk mengetahui permasalahan yang ada, yaitu kesulitan dalam memahami proses algoritma sorting tanpa adanya visualisasi.
b. Studi Literatur. Pada tahap ini dilakukan pengumpulan referensi dari buku dan jurnal terkait algoritma sorting dan visualisasi untuk mendukung penelitian.
c. Perancangan Sistem. Tahap ini meliputi perancangan fitur-fitur program seperti visualisasi sorting, pemilihan algoritma, pengaturan jumlah data, dan kecepatan animasi.
d. Implementasi. Pada tahap ini dilakukan pembuatan program menggunakan bahasa Python dengan mengimplementasikan algoritma Bubble Sort, Selection Sort, Insertion Sort, dan Quick Sort.
e. Pengujian. Pengujian dilakukan dengan menjalankan program menggunakan data acak untuk melihat proses sorting serta mengukur performa algoritma.
f. Analisis Hasil. Hasil pengujian kemudian dianalisis untuk membandingkan performa masing-masing algoritma dan menentukan algoritma yang paling efisien.

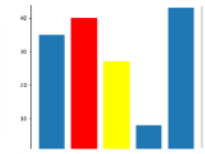
HASIL PENELITIAN DAN PEMBAHASAN

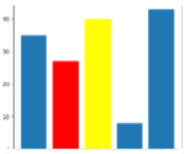
Hasil Implementasi dan Visualisasi

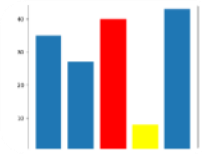
Penelitian ini berhasil mengimplementasikan empat algoritma *sorting* yaitu *Bubble Sort*, *Selection Sort*, *Insertion Sort*, dan *Quick Sort* ke dalam sebuah program interaktif berbasis Python. Program ini dilengkapi dengan fitur visualisasi grafis berupa diagram batang yang merepresentasikan data acak. Proses pengurutan data dapat diamati secara langsung, di mana elemen data yang sedang dibandingkan atau ditukar posisinya akan diberi warna berbeda (merah dan kuning) untuk memudahkan pemahaman. Selain itu, program mencatat dan menampilkan jumlah perbandingan (*comparisons*) dan pertukaran (*swaps*) yang terjadi selama proses *sorting* berlangsung. Hasil dari implementasi dan visualisasi untuk masing-masing algoritma disajikan pada Gambar 1 hingga Gambar 5.

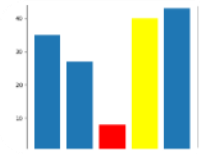
1. Visualisasi *Bubble Sort.* Sebelum menjalankan visualisasi, pengguna terlebih dahulu memilih algoritma dan mengatur parameter melalui terminal. Pengguna memilih *Bubble Sort* dengan jumlah data 5, kecepatan animasi 0.3 detik, dan mode langsung (tanpa step-by-step). *Bubble Sort* bekerja dengan membandingkan elemen yang berdekatan dan menukarnya jika urutannya salah. Proses ini diulang hingga seluruh data terurut. Visualisasi dari algoritma ini dapat dilihat pada Gambar 1.

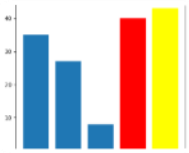

Gambar 1.1

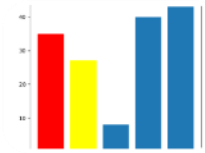

Gambar 1.2

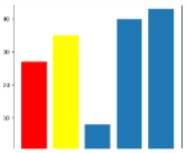

Gambar 1.3

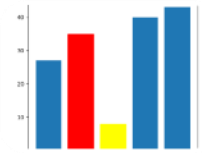

Gambar 1.4

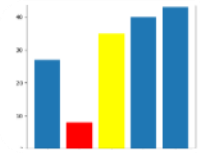

Gambar 1.5

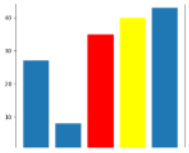

Gambar 1.6

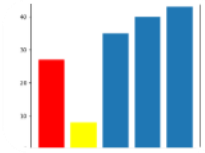

Gambar 1.7

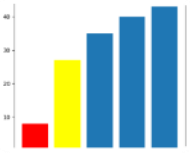

Gambar 1.8

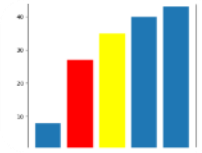

Gambar 1.9

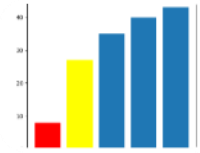

Gambar 1.10


Gambar 1.11


Gambar 1.12

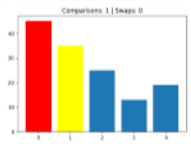

Gambar 1.13

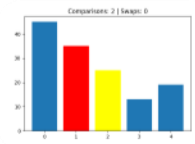

Gambar 1.14

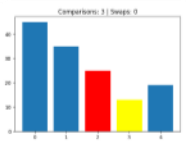

Gambar 1.15

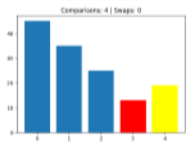
Gambar 1 menunjukkan serangkaian tangkapan layar dari proses *Bubble Sort*. Pada Gambar 1.1 dan 1.2, terlihat dua batang berwarna merah dan kuning saling berdekatan, mengindikasikan proses perbandingan antar elemen. Jika elemen di sebelah kiri (merah) lebih besar dari elemen di sebelah kanan (kuning), maka posisi keduanya akan ditukar, seperti yang terlihat pada Gambar 1.3. Proses ini terus berulang hingga di akhir visualisasi, seluruh batang tersusun dari nilai terkecil hingga terbesar, menandakan bahwa data telah terurut secara *ascending*.

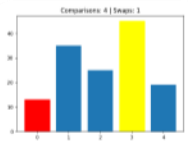
2. Visualisasi *Selection Sort.* Pengguna memilih algoritma *Selection Sort* dengan jumlah data yang sama untuk memudahkan perbandingan. Algoritma *Selection Sort* bekerja dengan mencari nilai terkecil dalam data, lalu menukarnya dengan elemen pertama. Proses ini diulang untuk sisa data yang belum terurut. Visualisasinya disajikan pada Gambar 2.

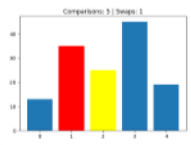

Gambar 2.1

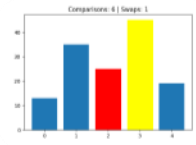

Gambar 2.2

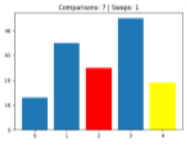

Gambar 2.3

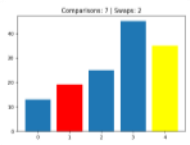

Gambar 2.4

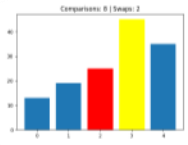

Gambar 2.5

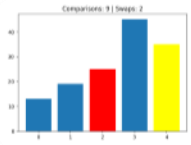

Gambar 2.6

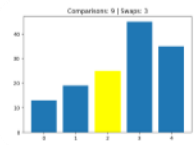

Gambar 2.7

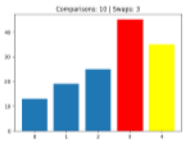

Gambar 2.8

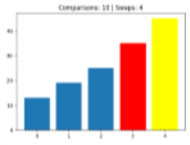

Gambar 2.9

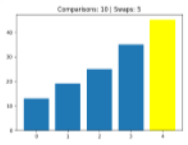

Gambar 2.10


Gambar 2.11


Gambar 2.12

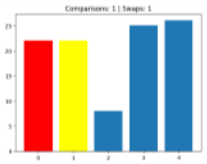

Gambar 2.13

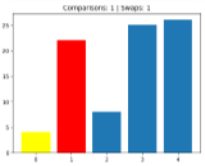

Gambar 2.14

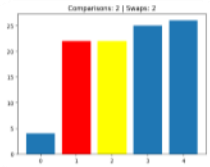

Gambar 2.15

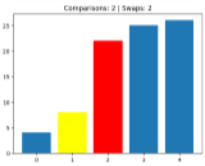
Pada Gambar 2.1 dan 2.2, terlihat proses iterasi untuk mencari nilai minimum. Satu batang berwarna merah menandai posisi yang dianggap sebagai nilai minimum sementara, sementara batang kuning adalah elemen yang sedang dibandingkan. Ketika nilai minimum yang sebenarnya ditemukan, kedua elemen tersebut akan ditukar posisinya. Gambar 2.3 menunjukkan pertukaran yang terjadi, di mana nilai minimum dipindahkan ke posisi paling awal dari sub-array yang belum terurut. Proses ini berlanjut hingga seluruh data terurut.

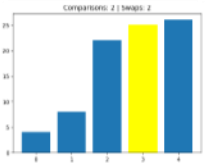
3. Visualisasi *Insertion Sort.* Pengguna memilih algoritma *Insertion Sort* dengan jumlah data yang sama untuk memudahkan perbandingan. *Insertion Sort* bekerja dengan membagi data menjadi dua bagian: terurut dan belum terurut. Elemen dari bagian belum terurut diambil dan disisipkan ke posisi yang tepat di bagian terurut. Proses ini divisualisasikan pada Gambar 3.

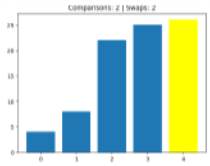

Gambar 3.1


Gambar 3.2


Gambar 3.3


Gambar 3.4

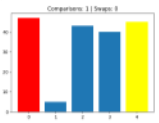

Gambar 3.5

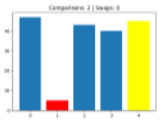

Gambar 3.6

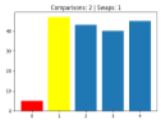
Pada Gambar 3.1, batang kuning (yang akan disisipkan) dibandingkan dengan elemen-elemen di sebelah kirinya (bagian yang sudah terurut). Warna merah pada batang di sebelah

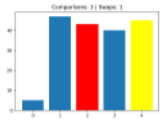
kiri menandakan elemen yang lebih besar sedang digeser ke kanan untuk memberikan ruang bagi elemen baru. Gambar 3.2 menunjukkan posisi setelah penyisipan, di mana batang kuning telah ditempatkan pada urutan yang benar di antara elemen-elemen di sebelah kirinya.

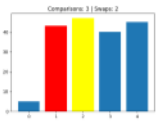
4. Visualisasi *Quick Sort.* Pengguna memilih algoritma *Quick Sort* dengan jumlah data yang sama untuk memudahkan perbandingan. *Quick Sort* menggunakan strategi divide and conquer dengan memilih sebuah elemen sebagai pivot. Elemen yang lebih kecil dari *pivot* ditempatkan di sebelah kirinya, dan yang lebih besar di sebelah kanannya. Proses ini dilakukan secara rekursif pada sub-array kiri dan kanan. Visualisasinya ditampilkan pada Gambar 4.

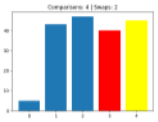

Gambar 4.1

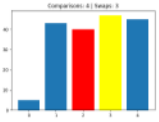

Gambar 4.2

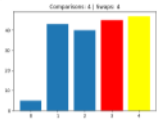

Gambar 4.3

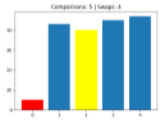

Gambar 4.4

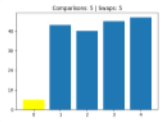

Gambar 4.5

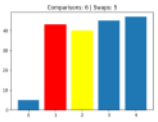

Gambar 4.6

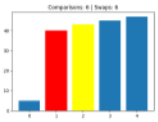

Gambar 4.7


Gambar 4.8


Gambar 4.9


v.10


Gambar 4.11


Gambar 4.12

Gambar 4.1 mengilustrasikan proses partisi. Batang kuning kemungkinan besar menandai elemen *pivot* (misalnya, elemen terakhir), sementara batang merah adalah elemen yang sedang dibandingkan dengan *pivot*. Batang-batang akan dipindahkan (ditukar) sehingga semua elemen yang lebih kecil dari *pivot* berada di sebelah kirinya. Gambar 4.2 menunjukkan keadaan setelah partisi, di mana *pivot* (sekarang berwarna kuning) telah berada pada posisi finalnya, dengan elemen-elemen yang lebih kecil di kiri dan lebih besar di kanan.

Tabel 1. Rata-rata Waktu Eksekusi (n=5, 5 kali pengujian)

Algoritma	Run 1	Run 2	Run 3	Run 4	Run 5	Rata-rata
Bubble Sort	0.00004	0.00003	0.00004	0.00003	0.00004	0.000036
Selection Sort	0.00003	0.00002	0.00003	0.00002	0.00003	0.000026
Insertion Sort	0.00002	0.00002	0.00002	0.00002	0.00003	0.000022
Quick Sort	0.00004	0.00003	0.00003	0.00004	0.00003	0.000034

Berdasarkan Tabel 1, terlihat bahwa waktu eksekusi antar run mengalami fluktuasi meskipun jumlah data (n=5) dan kecepatan animasi (0.3 detik) dibuat sama. Hal ini disebabkan oleh dua faktor utama. Pertama, data yang dihasilkan secara acak pada setiap run memiliki struktur yang berbeda, sehingga jumlah perbandingan dan pertukaran yang dilakukan algoritma tidak selalu identik. Kedua, untuk waktu sekecil orde 10^{-5} detik, kondisi latar belakang sistem seperti proses lain yang berjalan di CPU dapat mempengaruhi hasil pengukuran. Meskipun demikian, pola efisiensi algoritma masih terlihat jelas, di mana *Insertion Sort* dan *Selection Sort* cenderung lebih cepat untuk dataset berukuran sangat kecil (n=5), sementara *Quick Sort* menunjukkan performa yang bervariasi tergantung pada pemilihan pivot.

Pembahasan

Dari hasil pengujian yang dilakukan, terlihat bahwa setiap algoritma sorting memiliki karakteristik yang berbeda-beda, baik dari cara kerja maupun waktu eksekusinya. Visualisasi yang ditampilkan pada program membantu memperjelas proses tersebut, sehingga memudahkan untuk memahami mengapa suatu algoritma bisa lebih cepat atau lambat dibandingkan algoritma lainnya. Bubble Sort misalnya, dari hasil pengujian dengan 5 data, waktu eksekusinya berkisar antara 0.00003 sampai 0.00004 detik. Secara visual, algoritma ini bekerja dengan membandingkan elemen yang berdekatan dan menukarnya jika tidak sesuai. Proses ini berulang terus sampai semua data terurut. Selection Sort menunjukkan waktu yang sedikit lebih cepat dibanding Bubble Sort, yaitu antara 0.00002 sampai 0.00003 detik. Cara kerjanya berbeda, algoritma ini mencari nilai terkecil dari data yang belum terurut, lalu menukarnya dengan elemen pertama. Proses pencarian ini membuat jumlah pertukaran yang dilakukan lebih sedikit dibanding Bubble Sort.Hal ini karena metode yang digunakan berfokus pada pencarian elemen terkecil dari array dan menukarnya langsung dengan elemen di posisi pertama, sehingga secara signifikan menghemat beban komputasi penukaran[8]. Dari visualisasi terlihat bahwa pertukaran hanya terjadi sekali setiap iterasi, berbeda dengan Bubble Sort yang bisa berkali-kali melakukan pertukaran dalam satu iterasi. Ini yang membuat Selection Sort relatif lebih efisien.

Insertion Sort justru mencatat waktu tercepat di antara keempat algoritma untuk pengujian kali ini, yaitu 0.00002 detik secara konsisten. Hal ini sejalan dengan karakteristik algoritma *Insertion Sort* yang dikenal karena kesederhanaannya dan efisiensinya yang sangat baik dalam menangani dataset berukuran kecil hingga menengah[9]. Algoritma ini bekerja dengan cara menyisipkan elemen ke posisi yang tepat pada bagian data yang sudah terurut. Visualisasinya menarik karena mirip dengan cara orang mengurutkan kartu di tangan. Quick Sort justru menunjukkan waktu yang bervariasi, ada yang 0.00003 detik dan ada yang 0.00004 detik. Dari teori yang dipelajari, Quick Sort seharusnya lebih unggul untuk data dalam jumlah besar karena kompleksitas rata-ratanya O(n log n). Visualisasi Quick Sort juga terlihat lebih kompleks karena melibatkan proses partisi dan rekursif, berbeda dengan algoritma lain yang bekerja secara berurutan.Berdasarkan metrik pengujian pada dunia nyata, algoritma seperti *Bubble Sort* membutuhkan upaya komputasi dan jumlah pertukaran (*swaps*) yang sangat tinggi sehingga kinerjanya lambat, sedangkan *Quick Sort* mampu mencapai efisiensi yang tinggi meskipun performa terburuknya sangat bergantung pada pemilihan *pivot* yang ideal[10]. Perbedaan waktu yang muncul antara satu percobaan dengan percobaan berikutnya juga wajar terjadi. Penyebab utamanya karena setiap kali program dijalankan, data yang dihasilkan berbeda-beda karena bersifat acak. Selain itu, kondisi komputer seperti proses yang berjalan di latar belakang juga bisa mempengaruhi hasil pengukuran, apalagi untuk waktu sekecil 0.00001 detik. Itu sebabnya dalam penelitian sebaiknya dilakukan pengujian berulang dan diambil rata-ratanya agar hasilnya lebih akurat.

KESIMPULAN

Penelitian ini berhasil mengimplementasikan empat algoritma sorting yaitu Bubble Sort, Selection Sort, Insertion Sort, dan Quick Sort ke dalam program visualisasi interaktif berbasis Python. Program ini dilengkapi fitur diagram batang, pengaturan jumlah data, kontrol kecepatan, mode step-by-step, serta analisis jumlah perbandingan dan pertukaran. Dari hasil pengujian dengan 5 data acak, Insertion Sort mencatat waktu tercepat rata-rata 0.00002 detik, diikuti Selection Sort 0.000026 detik, sedangkan Bubble Sort dan Quick Sort berada di kisaran 0.00003-0.00004 detik. Perbedaan waktu ini terjadi karena setiap algoritma memiliki cara kerja yang berbeda. Visualisasi menunjukkan Bubble Sort banyak melakukan pertukaran,

Selection Sort lebih sedikit pertukaran, Insertion Sort bekerja seperti menyusun kartu, dan Quick Sort menggunakan metode partisi yang kompleks. Program visualisasi ini cukup membantu sebagai media pembelajaran karena pengguna bisa melihat langsung proses sorting langkah demi langkah. Mahasiswa atau siapa pun yang baru belajar algoritma sorting bisa lebih mudah memahami konsep yang sebelumnya hanya abstrak melalui teori.

DAFTAR PUSTAKA

(1) Y. Song, F. Tang, W. Dong, and C. Xu, "Non-dominated sorting based multi-page photo collage," *Comput. Vis. Media (Beijing).*, vol. 8, no. 2, pp. 199–212, Jun. 2022, doi: 10.1007/s41095-021-0221-0.
(2) M. Subramaniam, T. Tripathi, and O. Chandraumakantham, "Cluster Sort: A Novel Hybrid Approach to Efficient In-Place Sorting Using Data Clustering," *IEEE Access*, vol. 13, pp. 74359–74374, 2025, doi: 10.1109/ACCESS.2025.3564380.
(3) Z. Sitorus, D. Prayogi, M. A. Rizko, A. G. Suteja, and M. R. Harahap, "Implementation of the Insertion Sort Algorithm to Sort Positive Integers in Ascending Order Using Flowgorithm," *Journal of Information Technology, computer science and Electrical Engineering (JITCSE)*, vol. 1, no. 3, pp. 323–328, 2024, doi: 10.30596/jitcse.
(4) F. R. Wibowo and M. Faisal, "Comparative Analysis of Sorting Algorithms: TimSort Python and Classical Sorting Methods," *Jurnal Informatika dan Sains)*, vol. 07, no. 01, 2024.
(5) N. S. Aljerjawi and S. S. Abu-Naser, "Improving Sorting Algorithms Using Artificial Intelligence: A Cross Tactic," 2025. [Online]. Available: www.ijeais.org/ijaisr
(6) F. A. D.E. and D. O. Kurniawati, "Penggunaan Algoritma Sorting Bubble Sort Untuk Penentuan Nilai Prestasi Siswa," *SISTEMASI*, vol. 8, no. 2, p. 296, May 2019, doi: 10.32520/stmsi.v8i2.493.
(7) D. Rosdiana, "Implementasi Algoritma Quick Sort dalam Sistem Pemesanan Makanan di Rumah Makan," vol. 1, no. 2, [Online]. Available: https://jurnal.komputasi.org/index.php/jst/article/view/28
(8) I. P. Sari, F. Ramadhani, and O. K. Sulaiman, "Implementation of the Selection Sort Algorithm to Sort Data in PHP Programming Language," *Journal of Computer Science, Information Technology and Telecommunication Engineering*, vol. 4, no. 1, Mar. 2023, doi: 10.30596/jcositte.v4i1.14362.
(9) A. Ghofur *et al.*, "Penerapan Algoritma Insertion Sort Pada Aplikasi Pengolahan Data Mahasiswa Menggunakan Java Di Fakultas Sains Dan Teknologi Prodi Teknologi Informasi," *Eastasouth Journal of Positive Community Services*, vol. 4, no. 01, pp. 12–19, 2025, doi: 10.58812/ejpcs.v4i01.
(10) R. Ayazuddin, "A Comprehensive Study of Sorting Algorithm Performance Using Real-World Dataset Metrics," Sep. 30, 2025. doi: 10.20944/preprints202509.2550.v1.