PENGEMBANGAN APLIKASI BENCHMARKING WAKTU KOMPUTASI ALGORITMA SORTING MENGGUNAKAN OCTAVE

Muhammad Naufal Musyaafa, Aldo Bonifasius Simbolon, Khildan Rifail Azis, Debi Yandra Niska
Ilmu Komputer, Universitas Negeri Medan
Jl. Willem Iskandar Ps. V Medan, Indonesia
naufalmusyafaa@mhs.unimed.ac.id

ABSTRAK

Perkembangan teknologi informasi menuntut kecepatan pemrosesan data yang optimal, terutama dalam pengurutan data menggunakan algoritma sorting. Permasalahan yang dihadapi adalah kurangnya pemahaman mengenai efisiensi waktu komputasi dari berbagai algoritma sorting dalam kondisi berbeda. Penelitian ini bertujuan untuk mengembangkan sebuah aplikasi benchmarking yang mampu mengukur waktu komputasi dari algoritma Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, dan Quick Sort menggunakan bahasa pemrograman Octave. Metode yang digunakan adalah pendekatan eksperimental, dengan pengujian terhadap dataset berukuran kecil hingga besar untuk memperoleh waktu eksekusi setiap algoritma. Hasil penelitian menunjukkan bahwa algoritma Merge Sort dan Quick Sort memiliki performa waktu komputasi yang lebih efisien dibandingkan algoritma lainnya, khususnya pada dataset besar. Aplikasi yang dikembangkan juga memberikan tampilan hasil benchmarking secara visual, sehingga memudahkan pengguna dalam memahami performa masing-masing algoritma. Penelitian ini diharapkan dapat menjadi referensi dalam pemilihan algoritma sorting yang sesuai dengan kebutuhan komputasi.

Kata kunci : *Algoritma Sorting, Benchmarking, Octave, Waktu Komputasi, Aplikasi Komputasi*

1. PENDAHULUAN

Perkembangan teknologi informasi menuntut kecepatan pemrosesan data yang optimal, terutama dalam pengurutan data menggunakan algoritma sorting. Algoritma seperti *Bubble Sort*, *Selection Sort*, dan *Insertion Sort* memiliki karakteristik dan performa yang berbeda, sehingga pemilihan algoritma yang tepat sangat penting untuk meningkatkan efisiensi komputasi. Penelitian terdahulu menganalisis enam strategi algoritma sorting menggunakan bahasa pemrograman Java dan Python, menunjukkan bahwa performa algoritma dapat bervariasi tergantung pada bahasa pemrograman yang digunakan dan ukuran data yang diolah [1].

Namun, penelitian mengenai benchmarking algoritma sorting menggunakan GNU Octave masih terbatas. Berdasarkan hal tersebut, dirumuskan permasalahan: bagaimana merancang aplikasi benchmarking waktu komputasi untuk algoritma sorting menggunakan GNU Octave, algoritma mana yang menunjukkan performa paling efisien berdasarkan hasil benchmarking, dan bagaimana penerapan aplikasi ini dalam konteks pembelajaran atau penelitian algoritma. Tujuan penelitian ini adalah merancang dan membangun aplikasi benchmarking waktu komputasi algoritma sorting menggunakan GNU Octave, mengukur serta membandingkan performa beberapa algoritma berdasarkan waktu eksekusi, serta menyediakan alat bantu analisis bagi akademisi dan peneliti dalam mengevaluasi efisiensi algoritma sorting.

2. TINJAUAN PUSTAKA

2.1. Algoritma Sorting

Algoritma adalah langkah-langkah logis yang digunakan untuk menyelesaikan suatu masalah, dan keberadaannya sangat penting bagi setiap developer [1].

Jika algoritma yang digunakan tidak dirancang atau dipilih dengan tepat, hal ini bisa berdampak buruk, seperti waktu eksekusi program yang lebih lama, penggunaan memori yang berlebihan, hingga potensi kerusakan pada sistem maupun perangkat. Kesalahan dalam strategi pemilihan algoritma bisa membuat program berjalan tidak efisien, sistem menjadi tidak optimal, dan pada akhirnya membuat pengguna merasa frustrasi karena prosesnya yang lambat. Algoritma sorting merupakan komponen fundamental dalam ilmu komputer dengan aplikasi luas dalam pengolahan data dan optimasi komputasi [2].

Penelitian Tiwari mengidentifikasi lima algoritma utama—Quick Sort, Merge Sort, Bubble Sort, Selection Sort, dan Insertion Sort—yang menunjukkan karakteristik efisiensi berbeda berdasarkan skenario best-case, worst-case, dan average-case. Studi ini menemukan bahwa Quick Sort unggul untuk data acak berukuran besar dengan kompleksitas waktu rata-rata $O(n\log n)$, sementara Insertion Sort lebih efisien pada data kecil atau hampir terurut dengan kompleksitas waktu terbaik $O(n)$. Analisis kompleksitas ruang oleh Levi mengkategorikan algoritma menjadi in-place (seperti Quick Sort) yang membutuhkan ruang konstan $O(1)$,

dan out-of-place (seperti Merge Sort) yang memerlukan alokasi memori tambahan $O(n)$ [3]. Penelitian ini juga menekankan pentingnya stabilitas algoritma, di mana Merge Sort dan Insertion Sort mempertahankan urutan relatif elemen setara, berbeda dengan Quick Sort yang tidak stabil. Studi komparatif lain memperkuat temuan ini dengan menguji performa algoritma pada berbagai ukuran data [4].

Hasilnya menunjukkan bahwa Quick Sort optimal untuk data besar, sedangkan Insertion Sort cocok untuk data kecil. Penelitian ini juga menyoroti trade-off antara kompleksitas waktu dan ruang, serta potensi paralelisasi untuk meningkatkan kecepatan komputasi. Ketiga studi tersebut memberikan dasar teoretis yang relevan untuk pengembangan sistem benchmarking waktu eksekusi algoritma sorting.

2.2. Kompleksitas Waktu dalam Algoritma Sorting

Kompleksitas waktu dalam algoritma sorting merupakan salah satu aspek penting yang digunakan untuk mengevaluasi efisiensi algoritma dalam mengolah data [5].

Kompleksitas ini dinyatakan dalam notasi Big-O, yang menggambarkan hubungan antara ukuran data dan jumlah operasi yang diperlukan. Algoritma seperti *Merge Sort* dan *Quick Sort* memiliki kompleksitas rata-rata O (n log n) O (n log n), sementara algoritma sederhana seperti *Bubble Sort* memiliki kompleksitas O(n2). Pemilihan algoritma yang tepat bergantung pada karakteristik data, seperti ukuran dan tingkat keterurutan, serta kebutuhan akan efisiensi waktu dan ruang. Pola distribusi data, dan tingkat keterurutan awal mempengaruhi secara signifikan terhadap realisasi kompleksitas waktu aktual. Algoritma comparison-based seperti merge sort dan quick sort menunjukkan kompleksitas optimal O(n log n) pada kasus rata-rata, sementara algoritma sederhana seperti bubble sort dan selection sort konsisten pada $O(n^2)$ untuk berbagai skenario.

2.3. Metode Benchmarking dalam Komputasi

Benchmarking dalam komputasi adalah proses sistematis untuk mengukur dan membandingkan kinerja perangkat keras, perangkat lunak, atau algoritma dengan standar tertentu atau praktik terbaik [6].

Metode ini bertujuan untuk mengevaluasi efisiensi dan efektivitas sistem yang diuji, serta memberikan dasar bagi perbaikan dan pengembangan lebih lanjut. Dalam konteks komputasi, benchmarking sering digunakan untuk mengukur waktu eksekusi, penggunaan sumber daya, dan kinerja keseluruhan dari algoritma atau perangkat keras. Proses benchmarking biasanya melibatkan beberapa langkah utama. Pertama, objek yang akan diukur harus ditentukan, seperti algoritma atau perangkat keras tertentu. Selanjutnya, data dikumpulkan melalui pengujian langsung atau simulasi, yang kemudian dianalisis untuk mengidentifikasi kekuatan, kelemahan, dan peluang peningkatan. Hasil analisis ini digunakan untuk membandingkan kinerja dengan standar atau kompetitor terbaik di bidangnya. Metode ini dapat dilakukan berdasarkan konsep siklus seperti *Plan-Do-Check-Act (PDCA)*, yang memastikan proses berlangsung secara iteratif dan berkelanjutan. Dalam dunia komputasi modern, metode benchmarking juga telah berkembang mencakup berbagai level evaluasi. Misalnya, pada tingkat perangkat keras, pengujian dilakukan terhadap prosesor, RAM, GPU, dan perangkat penyimpanan untuk menilai efisiensi penggunaan sumber daya. Pada tingkat algoritma, pengujian dapat melibatkan analisis waktu eksekusi dan kompleksitas algoritmik dalam berbagai kondisi input. Selain itu, benchmarking juga dapat mencakup pengukuran kinerja aplikasi dalam skenario dunia nyata untuk memastikan relevansi hasil dengan kebutuhan pengguna [7].

Metode benchmarking tidak hanya membantu dalam memahami performa sistem saat ini tetapi juga memberikan panduan bagi inovasi teknologi. Dengan menggunakan pendekatan yang terstruktur dan standar yang jelas, benchmarking memungkinkan reproduksi hasil serta perbandingan yang adil antar sistem atau algoritma.

2.4. Penggunaan Octave untuk Analisis Kinerja Algoritma

Octave merupakan alat komputasi numerik bersifat *open-source* yang banyak digunakan untuk analisis kinerja algoritma, terutama dalam konteks pengukuran waktu eksekusi, optimasi kode, dan evaluasi efisiensi komputasi [8].

Kemampuannya dalam menangani operasi matriks, integrasi dengan bahasa pemrograman lain, serta dukungan paket khusus membuatnya efektif untuk benchmarking algoritma dalam berbagai skenario. Penelitian terdahulu mengembangkan paket AutonomousSystems4D di Octave untuk analisis kualitatif sistem non-linear 4D. Paket ini memanfaatkan fungsi bawaan Octave seperti lsode untuk menyelesaikan persamaan diferensial dan tic/toc untuk mengukur waktu komputasi solusi numerik. Hasilnya menunjukkan kemampuan Octave dalam mengevaluasi kinerja algoritma melalui metrik seperti waktu eksekusi dan kompleksitas solusi. Studi lain memperkenalkan queueing package di Octave untuk analisis jaringan antrian dan rantai Markov. Paket ini menyediakan fungsi untuk menghitung throughput, response time, dan utilisasi sumber daya, yang menjadi dasar dalam menilai efisiensi algoritma pada sistem terdistribusi atau real-time [9].

Dalam konteks komputasi kinerja tinggi, Octave juga digunakan untuk membandingkan implementasi algoritma *Fast Fourier Transform (FFT)* dan peningkatan citra [10].

Penelitian pada konferensi HPCMP-UGC menunjukkan bahwa Octave mampu memberikan hasil benchmarking yang konsisten dengan MATLAB,

baik dalam hal produktivitas pengembangan maupun akurasi performa, meskipun dengan sumber daya komputasi yang lebih terbatas. Fleksibilitas Octave dalam mengintegrasikan paket eksternal dan fungsi kustom memungkinkan peneliti untuk mengadaptasi metodologi benchmarking sesuai kebutuhan spesifik algoritma yang diuji.

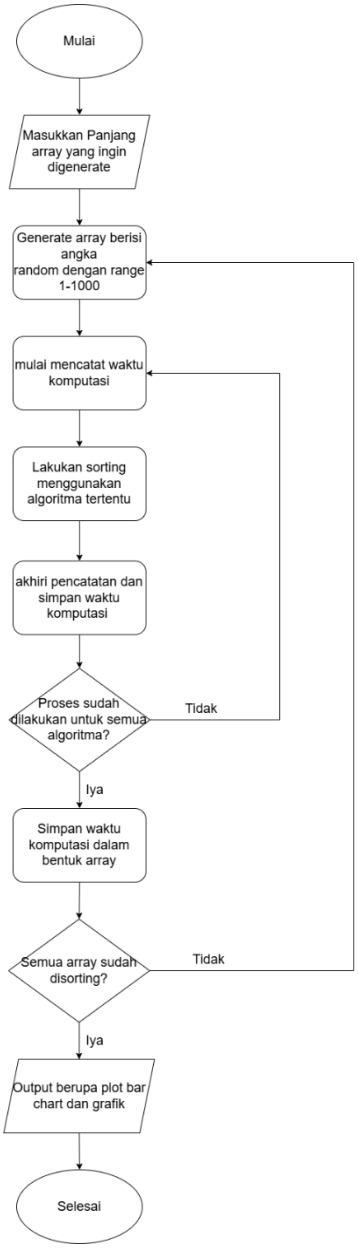
3. METODE PENELITIAN

Penelitian ini merupakan studi eksperimental yang bertujuan untuk mengukur dan membandingkan waktu komputasi dari lima algoritma sorting (Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, dan Shell Sort) menggunakan GNU Octave sebagai platform benchmarking.

Penelitian ini mengkaji dua variabel utama, yaitu variabel independen dan dependen. Variabel independen terdiri dari jenis algoritma sorting yang digunakan (Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, dan Shell Sort) serta variasi panjang array yang diujikan. Variabel dependen adalah waktu eksekusi, yaitu waktu komputasi yang diperlukan setiap algoritma untuk mengurutkan data, diukur dalam satuan detik menggunakan fungsi tic dan toc pada Octave.

Metode pada penelitian ini dilakukan dalam beberapa tahapan. Pertama, program meminta lima input panjang array dari pengguna. Berdasarkan input tersebut, program akan melakukan *generate* lima array acak dengan nilai antara 1 hingga 1000. Masing-masing array ini digunakan untuk menguji kelima algoritma sorting secara terpisah. Untuk setiap array, program melakukan proses sorting menggunakan kelima algoritma yang telah diimplementasikan dalam bentuk fungsi, Pada setiap eksekusi algoritma, waktu komputasi diukur dengan cara memanggil tic sebelum proses sorting dan toc setelah proses selesai. Hasil pengukuran ini kemudian disimpan dalam sebuah matriks, di mana baris matriks mewakili masing-masing panjang array dan kolom mewakili algoritma sorting.

Setelah pengumpulan data, hasil pengukuran dianalisis untuk mengevaluasi performa masing-masing algoritma serta pengaruh variasi panjang array terhadap waktu komputasi. Hasil analisis divisualisasikan melalui dua jenis plot: pertama, plot yang menampilkan bar chart per array untuk memperlihatkan perbandingan waktu eksekusi antar algoritma, dan kedua, plot yang menampilkan grafik garis untuk masing-masing algoritma yang menggambarkan variasi waktu eksekusi terhadap kelima panjang array yang diuji. Dengan demikian, penelitian ini tidak hanya mengukur efisiensi setiap algoritma sorting tetapi juga mengkaji hubungan antara ukuran data dan performa algoritma. Berikut merupakan flowchart dari program ini.



Gambar 1. Alur metode penelitian

3.1. Parameter Pengujian

Dalam pengetan program ini, parameter utama yang digunakan adalah ukuran dataset berupa array, yang diwakili oleh lima variasi, yaitu 1000, 5000, 10000, 15000, dan 20000 data. Setiap dataset terdiri dari angka-angka acak yang dihasilkan dalam rentang 1 hingga 1000. Variasi ukuran dataset ini dipilih untuk mengamati bagaimana peningkatan jumlah elemen data mempengaruhi waktu eksekusi masing-masing algoritma sorting yang diuji (Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, dan Shell Sort). Dengan menggunakan parameter ini, penelitian diharapkan dapat mengungkap perbedaan performa dan skalabilitas algoritma sorting, serta memberikan wawasan mengenai efektivitas algoritma dalam mengelola data dengan volume yang berbeda.

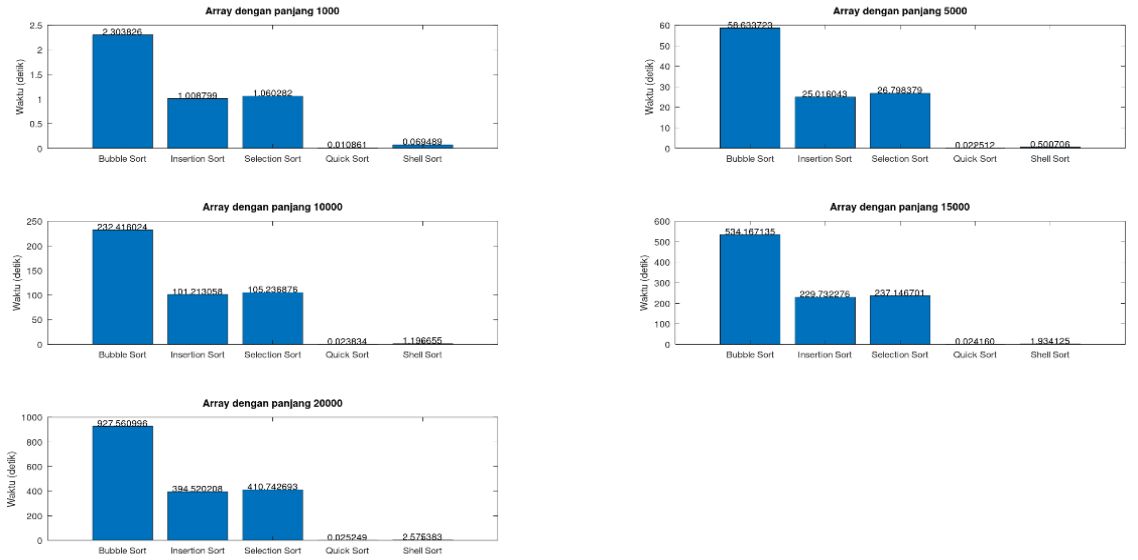
3.2. Perangkat

Aplikasi ini dirancang menggunakan GNU Octave versi 9.4.0. Pengujian dilakukan pada PC dengan spesifikasi sebagai berikut:

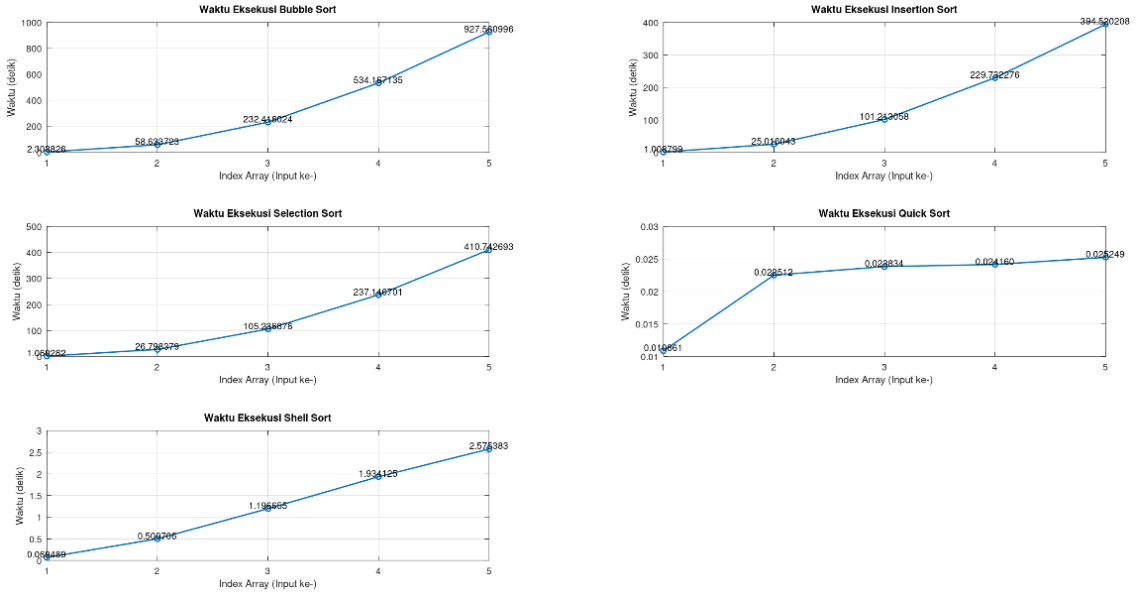
a. CPU AMD Ryzen 5 5500
b. RAM 12 GB DDR4 2666MHz
c. Sistem Operasi Windows 11 Pro versi 24H2

4. HASIL DAN PEMBAHASAN

Dari pengujian menggunakan parameter di atas, program menghasilkan duah buah plot berupa plot bar chart per array dan plot perbandingan antar algoritma sebagai berikut:



Gambar 2. Plot bar chart per array



Gambar 3. Plot grafik per algoritma

Berikut merupakan tabel waktu komputasi untuk tiap-tiap array pada setiap algoritma sorting:

Tabel 1. Waktu komputasi pada tiap algoritma dan array

Algoritma	Waktu Komputasi				
	1000	5000	10000	15000	20000
Bubble Sort	2,303826	58,633723	232,416024	534,167135	927,560996
Insertion Sort	1,008799	25,016043	101,213058	229,732276	394,520208
Selection Sort	1,060282	26,798379	105,236876	237,146701	410,742693
Quick Sort	0,010861	0,022512	0,023834	0,024160	0,025249
Shell Sort	0,069489	0,500706	1,196655	1,934125	2,575383

5. KESIMPULAN DAN SARAN

Berdasarkan hasil pengujian, dapat disimpulkan bahwa algoritma dengan kompleksitas $O(n^2)$ seperti Bubble Sort, Insertion Sort, dan Selection Sort menunjukkan peningkatan waktu eksekusi yang sangat signifikan seiring dengan bertambahnya ukuran array, sehingga kurang efisien untuk pengolahan data dalam jumlah besar. Bubble Sort mencatat waktu eksekusi tertinggi, sedangkan Insertion Sort dan Selection Sort sedikit lebih baik namun tetap mengalami lonjakan waktu yang tajam pada dataset besar. Sebaliknya, Quick Sort menunjukkan performa yang luar biasa dengan waktu eksekusi yang sangat rendah dan peningkatan yang hampir konstan terhadap variasi ukuran array, menegaskan keunggulannya dalam menangani data besar. Shell Sort, meskipun tidak secepat Quick Sort, algoritma ini memberikan peningkatan waktu eksekusi yang lebih moderat dan merupakan alternatif yang lebih efisien dibandingkan algoritma $O(n^2)$. Secara keseluruhan, Quick Sort merupakan pilihan optimal untuk pengurutan dataset besar, sementara Shell Sort dapat dipertimbangkan sebagai opsi efisien untuk kondisi tertentu, sedangkan algoritma berbasis $O(n^2)$ sebaiknya dihindari untuk data dengan volume yang tinggi. Untuk pengembangan selanjutnya, disarankan agar aplikasi benchmarking diperluas dengan menambahkan lebih banyak jenis algoritma sorting, termasuk algoritma yang lebih kompleks seperti Heap Sort, Radix Sort, dan Tim Sort, guna memberikan perbandingan yang lebih komprehensif. Selain itu, pengujian dapat dilakukan dengan variasi jenis data, seperti data acak, data terurut sebagian, dan data terbalik, untuk menguji performa algoritma dalam skenario yang lebih realistis. Integrasi visualisasi proses sorting secara real-time juga dapat meningkatkan pemahaman pengguna terhadap cara kerja masing-masing algoritma. Penambahan fitur untuk mencatat hasil benchmarking secara otomatis ke dalam laporan atau grafik interaktif akan semakin memperkuat fungsi aplikasi sebagai alat bantu dalam kegiatan pembelajaran maupun penelitian di bidang algoritma dan struktur data.

DAFTAR PUSTAKA

[1] J. Iskandar, H. Suhendar, dan B. D. Pamungkas, “Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python,” *G-Tech: Jurnal Teknologi Terapan*, vol. 8, no. 1, pp. 104–113, Jan. 2024.

[2] N. Tiwari, “Sorting Smarter: Unveiling Algorithmic Efficiency and User-Friendly Applications,” *TechRxiv*, Dec. 7, 2023.

[3] J. Levi, “Analisis Algoritma Penyortiran Bogosort dan Bozosort dengan Kombinatorika dan Kompleksitas Algoritma,” Des. 2024.

[4] M. Qureshi, D. Surve, P. Waghela, dan N. Sakpal, “Comparative Analysis of Sorting and Searching Algorithms,” *SSRN*, no. 4815467, 2024.

[5] Y. W. Yunanri, A. Fauzan, A. Yani, dan M. A. Aziz, “Analisis Performance Central Processing Unit (CPU) Realtime Menggunakan Metode Benchmarking,” *Matrik: Jurnal Manajemen, Teknik Informatika, dan Rekayasa Komputer*, vol. 20, no. 2, pp. 237–248, Mei 2021.

[6] V. G. Shubham, G. R. S. Vishwas, and A. Kumar, "Analysis of sorting algorithms using time complexity," *Int. J. Eng. Res. Technol.*, vol. 5, no. 21, pp. 1–3, 2020.

[7] J. M. Lorenz *et al*., "Systematic benchmarking of quantum computers: status and recommendations," *arXiv preprint* arXiv:2503.04905, 2025.

[8] E. Escobar, R. Abramonte, A. Aliaga, and F. Gutierrez, "An Octave package to perform qualitative analysis of nonlinear systems immersed in R^4," pp. 136–145, 2020.

[9] M. Marzolla, "Queueing networks and Markov chains analysis with the Octave queueing package," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 49, no. 4, pp. 47–52, 2022.

[10] A. Pajankar and S. Chandu, *GNU Octave by Example: A Fast and Practical Approach to Learning GNU Octave*. 2020.