Jurnal Ilmiah Ilmu Komputer & Elektronika (MURAKOM)
Volume. 2 Nomor. 2 April 2026
e-ISSN : 3123-1985; p-ISSN : 3110-4754, Hal. 16-24
DOI: https://doi.org/10.64365/murakom.v2i2.281
Available online at: *https://jurnal.muaraedukasi.id/index.php/murakom*

Analisis Perbandingan Kompleksitas Waktu dan Ruang pada Algoritma *Quick Sort, Merge Sort,* dan *Heap Sort*

M. Fahmi Arafat[1*], Nuriana Sipahutar[2], Adidtya Perdana, S.T., M.Kom [3], Riski Immanuel Situmorang[4], Raja Ansel Hartama Sihombing[5]
[1-5] Universitas Negeri Medan, Indonesia
Email : *arafatfahmiarafat@gmail.com[1]
Alamat: Jl. William Iskandar Ps. V, Kenangan Baru, Kec. Percut Sei Tuan, Kabupaten Deli Serdang, Sumatera Utara 20221
Korespondensi penulis: arafatfahmiarafat@gmail.com

Abstract*. This study aims to analyze and compare the performance of Quick Sort, Merge Sort, and Heap Sort algorithms based on execution time and memory usage. The research employs a quantitative approach by conducting experiments on the three algorithms using datasets with varying sizes and conditions, namely random, sorted, and reverse-ordered data. The parameters measured include execution time obtained through experimental testing using the Python programming language, as well as memory usage analyzed theoretically. The results indicate that execution time increases as the size of the dataset grows. Quick Sort demonstrates superior performance on sorted and reverse-ordered data while remaining competitive on random data. Merge Sort shows consistent performance and tends to be more optimal for large random datasets. Meanwhile, Heap Sort has relatively higher execution time but is more efficient in terms of memory usage. Therefore, the selection of an appropriate sorting algorithm should consider the characteristics of the data and system requirements, particularly in terms of time efficiency and memory utilization.*

Keywords: *Sorting Algorithms, Quick Sort, Merge Sort, Heap Sort, Execution Time, Complexity.*

Abstrak. Penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja algoritma pengurutan Quick Sort, Merge Sort, dan Heap Sort berdasarkan waktu eksekusi dan penggunaan memori. Metode yang digunakan adalah pendekatan kuantitatif dengan melakukan pengujian terhadap ketiga algoritma menggunakan dataset dengan variasi ukuran dan kondisi data, yaitu acak, terurut, dan terbalik. Parameter yang diukur meliputi waktu eksekusi yang diperoleh melalui proses eksperimen menggunakan bahasa pemrograman Python, serta analisis penggunaan memori yang dilakukan secara teoritis. Hasil penelitian menunjukkan bahwa waktu eksekusi meningkat seiring dengan bertambahnya jumlah data. Quick Sort memiliki performa yang unggul pada kondisi data terurut dan terbalik, serta tetap kompetitif pada data acak. Merge Sort menunjukkan kestabilan performa dan cenderung lebih optimal pada data acak berukuran besar. Sementara itu, Heap Sort memiliki waktu eksekusi yang relatif lebih tinggi, namun lebih efisien dalam penggunaan memori. Dengan demikian, pemilihan algoritma pengurutan yang optimal perlu mempertimbangkan karakteristik data serta kebutuhan sistem, baik dari segi efisiensi waktu maupun penggunaan memori.

Kata kunci: Algoritma Pengurutan, Pengurutan Cepat, Pengurutan Penggabungan, Pengurutan Tumpukan, Waktu Eksekusi, Kompleksitas.

1. LATAR BELAKANG

Dengan kemajuan teknologi informasi, pengolahan data yang cepat dan efisien menjadi lebih penting, terutama dalam proses pengurutan (sorting), yang sangat penting untuk berbagai aplikasi seperti pencarian dan pengolahan basis data. Memilih algoritma pengurutan yang tepat sangat penting karena mempengaruhi kinerja sistem, terutama ketika menangani data yang sangat besar (Saputra, et al., 2025).

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

Algoritma pengurutan berbasis perbandingan yang paling populer adalah Quick Sort, Merge Sort, dan Heap Sort. Quick Sort unggul dalam kinerja rata-rata, Merge Sort stabil dengan kompleksitas waktu yang konsisten, dan Heap Sort lebih efisien dalam penggunaan memori. Namun, penelitian menunjukkan bahwa tidak ada algoritma yang ideal untuk semua kondisi data (Pujiono, et al., 2025).

Oleh karena itu, tujuan dari penelitian ini yaitu untuk menentukan algoritma yang paling efektif dan efisien berdasarkan karakteristik data yang digunakan. Penelitian ini akan menganalisis perbandingan kompleksitas waktu dan ruang dari ketiga algoritma tersebut.

2. KAJIAN TEORITIS

Algoritma sorting merupakan salah satu proses penting dalam pengolahan data yang bertujuan untuk mengurutkan data berdasarkan kriteria tertentu, seperti urutan menaik atau menurun. Proses ini sangat penting karena dapat meningkatkan efisiensi algoritma lain, seperti pencarian dan analisis data. Oleh karena itu, pemilihan algoritma yang tepat sangat merupakan hal ynag penting (Tumanggor, et al., 2022).

Dua komponen utama yang harus diperhatikan saat menganalisis performa algoritma adalah kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu menunjukkan berapa lama algoritma membutuhkan untuk menyelesaikan suatu masalah berdasarkan ukuran input, sedangkan kompleksitas ruang menunjukkan berapa banyak memori yang digunakan algoritma selama proses eksekusi. Notasi Big O biasanya digunakan untuk menganalisis kedua komponen ini untuk mengetahui tingkat efisiensi algoritma terhadap pertumbuhan data (Ali, et al., 2025).

Salah satu algoritma pengaturan yang dikenal sebagai Quick Sort menggunakan pendekatan divide and conquer, yang berarti memilih elemen sebagai pivot dan kemudian membagi data menjadi dua bagian. Dikenal bahwa algoritma ini memiliki kompleksitas waktu rata-rata O(n log n), tetapi dalam kondisi terburuk dapat mencapai O(n2). Karena kecepatan eksekusinya yang tinggi dan kemampuan untuk bekerja secara in-place tanpa membutuhkan memori yang besar, Quick Sort masih sangat populer (Ali, et al., 2025).

Selain Quick Sort, Merge Sort juga merupakan algoritma yang menggunakan pendekatan split-and-conquer, yang berarti data dibagi menjadi bagian yang lebih kecil dan kemudian digabungkan kembali dalam keadaan terurut. Pada semua kondisi, algoritma ini memiliki keunggulan dalam hal kestabilan dan konsistensi kompleksitas waktu sebesar O(n log n). Namun, dibandingkan dengan algoritma lainnya, Merge Sort kurang efisien dalam penggunaan memori karena membutuhkan ruang tambahan sebesar O(n) (Tumanggor, et al., 2022).

Dalam proses pengurutan, heap sort adalah algoritma pengurutan yang menggunakan struktur data heap. Pada kondisi terbaik, rata-rata, dan terburuk, algoritma ini memiliki kompleksitas waktu sebesar O(n log n). Selain itu, Heap Sort berada di tempat, yang membuatnya lebih efisien dalam penggunaan memori. Karena kestabilan dan kompleksitasnya, Heap Sort sering digunakan sebagai pembanding dalam analisis performa algoritma sortasi dalam beberapa penelitian (Pujiono, et al., 2025).

Berdasarkan berbagai penelitian yang telah dilakukan, dapat disimpulkan bahwa setiap algoritma penyortiran memiliki keunggulan dan kelemahan sendiri. Heap Sort unggul dalam efisiensi penggunaan memori, Quick Sort unggul dalam kecepatan rata-rata, dan Merge Sort unggul dalam kestabilan. Oleh karena itu, untuk mendapatkan hasil terbaik, algoritma yang tepat harus disesuaikan dengan karakteristik data dan kebutuhan sistem.

3. METODE PENELITIAN

Penelitian ini menerapkan pendekatan kuantitatif dengan metode eksperimen yang bertujuan untuk mengevaluasi serta membandingkan kinerja algoritma Quick Sort, Merge Sort, dan Heap Sort ditinjau dari kompleksitas waktu dan ruang. Pendekatan kuantitatif dipilih karena penelitian ini berorientasi pada pengolahan data numerik yang bersifat objektif, khususnya waktu eksekusi dan penggunaan memori.

Objek penelitian meliputi tiga algoritma pengurutan berbasis perbandingan (comparison-based sorting), yaitu Quick Sort, Merge Sort, dan Heap Sort. Ketiga algoritma tersebut diimplementasikan menggunakan bahasa pemrograman Python, kemudian diuji menggunakan dataset dengan variasi ukuran dan kondisi. Dataset yang digunakan mencakup data acak, data yang telah terurut, serta data terbalik, dengan jumlah elemen yang bervariasi, yaitu 1.000, 10.000, hingga 100.000 data.

Pengumpulan data dilakukan melalui serangkaian eksperimen dengan menjalankan masing-masing algoritma pada dataset yang sama guna menjaga konsistensi dan objektivitas hasil. Parameter yang diamati meliputi waktu eksekusi (running time) serta penggunaan memori (space complexity). Untuk memperoleh hasil yang lebih akurat, setiap pengujian dilakukan secara berulang dan nilai rata-rata digunakan sebagai representasi hasil pengukuran.

Analisis data dilakukan dengan pendekatan deskriptif kuantitatif melalui proses perbandingan hasil pengujian antar algoritma. Data yang diperoleh disajikan dalam bentuk tabel maupun grafik untuk mempermudah pemahaman dan interpretasi. Selanjutnya, hasil tersebut dianalisis dengan mengacu pada teori kompleksitas algoritma guna menilai kesesuaian

antara hasil empiris dan konsep teoritis. Berdasarkan analisis tersebut, ditentukan algoritma yang paling efisien dan optimal sesuai dengan karakteristik data yang digunakan.

4. HASIL DAN PEMBAHASAN

Bagian ini menguraikan hasil penelitian yang diperoleh melalui proses pengujian terhadap algoritma Quick Sort, Merge Sort, dan Heap Sort, serta pembahasan yang berkaitan dengan kinerja masing-masing algoritma. Hasil penelitian yang disajikan mencakup pengukuran waktu eksekusi dan penggunaan memori pada berbagai variasi ukuran dan kondisi data. Selanjutnya, hasil tersebut dianalisis untuk mengidentifikasi perbedaan performa antar algoritma serta mengevaluasi kesesuaiannya dengan teori kompleksitas yang telah dikemukakan.

4.1 Hasil Pengujian

Pengujian dilakukan terhadap tiga algoritma pengurutan, yaitu Quick Sort, Merge Sort, dan Heap Sort, dengan menggunakan dataset yang memiliki variasi ukuran dan kondisi data. Ukuran data yang digunakan dalam penelitian ini meliputi 1.000, 5.000, dan 10.000 elemen. Adapun kondisi data yang diuji terdiri atas data acak, data terurut, dan data terbalik, dengan tujuan untuk mengetahui pengaruh karakteristik data terhadap kinerja masing-masing algoritma.

Parameter yang diamati dalam penelitian ini adalah waktu eksekusi (running time) yang diukur dalam satuan detik. Setiap algoritma diuji menggunakan dataset yang sama untuk menjaga konsistensi dan objektivitas hasil. Proses pengujian dilakukan dengan menjalankan masing-masing algoritma pada setiap variasi ukuran dan kondisi data, kemudian mencatat waktu yang dibutuhkan untuk menyelesaikan proses pengurutan.

Hasil pengujian waktu eksekusi dari ketiga algoritma tersebut disajikan pada Tabel 1.

Tabel 1. Waktu Eksekusi Algoritma (detik)

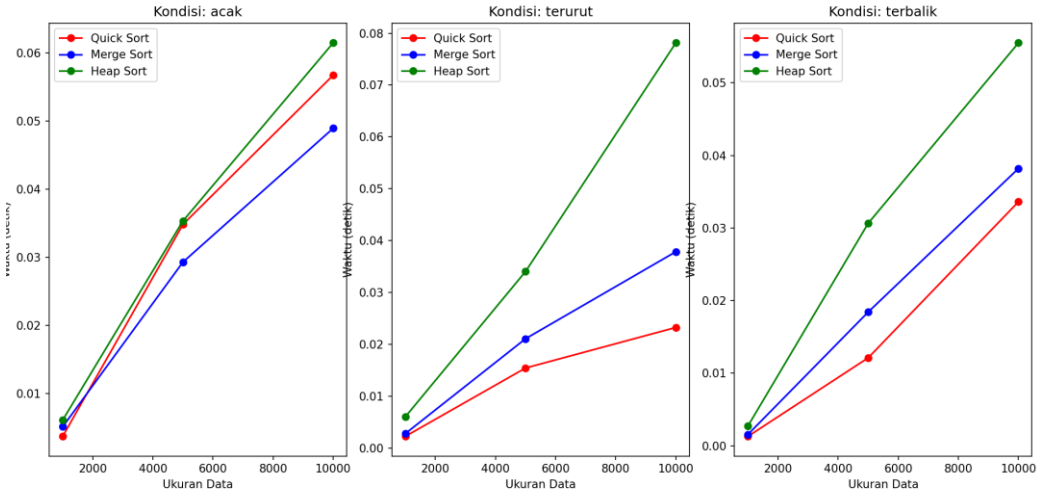
Jumlah Data	Kondisi Data	Quick Sort	Merge Sort	Heap Sort
1.000	Acak	0.003685	0.005092	0.006084
5.000	Acak	0.034804	0.029272	0.035315
10.000	Acak	0.056718	0.048918	0.061467
1.000	Terurut	0.002267	0.002798	0.006043
5.000	Terurut	0.015447	0.021097	0.034063
10.000	Terurut	0.023262	0.037864	0.078149
1.000	Terbalik	0.001302	0.001570	0.002738

Jumlah Data	Kondisi Data	Quick Sort	Merge Sort	Heap Sort
5.000	Terbalik	0.012120	0.018428	0.030651
10.000	Terbalik	0.033618	0.038198	0.055505

Sumber: Hasil penelitian (2026)

Selain dalam bentuk tabel, hasil pengujian juga disajikan dalam bentuk grafik untuk memperjelas perbandingan waktu eksekusi masing-masing algoritma pada berbagai kondisi data.

Gambar 1. Grafik Perbandingan Waktu Eksekusi Algoritma pada Berbagai Kondisi Data



Berdasarkan Tabel 1 dan Gambar 1, dapat diketahui bahwa waktu eksekusi ketiga algoritma meningkat seiring dengan bertambahnya jumlah data yang diolah. Pada kondisi data acak, Merge Sort menunjukkan waktu eksekusi yang sedikit lebih cepat dibandingkan Quick Sort pada ukuran data yang lebih besar, sedangkan Heap Sort tetap memiliki waktu eksekusi paling tinggi. Pada kondisi data terurut dan terbalik, Quick Sort menunjukkan performa yang lebih cepat dibandingkan algoritma lainnya. Sementara itu, Merge Sort cenderung memiliki performa yang stabil pada berbagai kondisi data, dan Heap Sort secara konsisten menunjukkan waktu eksekusi yang lebih besar dibandingkan kedua algoritma lainnya.

4.2 Analisis Waktu Eksekusi

Berdasarkan hasil pengujian yang ditampilkan pada Tabel 1 dan Gambar 1, dapat diamati bahwa waktu eksekusi seluruh algoritma mengalami peningkatan seiring dengan bertambahnya ukuran data. Hal ini menunjukkan bahwa besarnya input berpengaruh langsung terhadap kinerja algoritma pengurutan, sebagaimana dijelaskan dalam konsep kompleksitas waktu.

Pada kondisi data acak, Merge Sort menunjukkan kinerja yang sedikit lebih baik dibandingkan Quick Sort pada ukuran data yang lebih besar, terutama pada 5.000 dan 10.000 elemen. Hal ini menunjukkan bahwa Merge Sort mampu menjaga konsistensi performanya karena memiliki kompleksitas waktu O(n log n) pada semua kondisi. Sementara itu, Quick Sort tetap memiliki kinerja yang baik, meskipun sedikit berada di bawah Merge Sort pada kondisi tersebut.

Pada data yang telah terurut dan data terbalik, Quick Sort menunjukkan performa yang lebih unggul dibandingkan algoritma lainnya. Hal ini mengindikasikan bahwa implementasi Quick Sort dalam penelitian ini mampu meminimalkan kemungkinan terjadinya kondisi terburuk, sehingga tetap menghasilkan waktu eksekusi yang efisien. Di sisi lain, Merge Sort tetap menunjukkan kestabilan performa, meskipun waktu eksekusinya sedikit lebih tinggi dibandingkan Quick Sort.

Heap Sort menunjukkan waktu eksekusi yang paling besar pada seluruh variasi data yang diuji. Hal ini disebabkan oleh proses pembentukan dan penyesuaian heap (heapify) yang dilakukan secara berulang, sehingga meningkatkan beban komputasi selama proses pengurutan. Meskipun secara teoritis memiliki kompleksitas waktu yang sama dengan Merge Sort, yaitu O(n log n), faktor implementasi menyebabkan waktu eksekusinya relatif lebih tinggi.

Dengan demikian, dapat disimpulkan bahwa Quick Sort dan Merge Sort memiliki performa yang lebih baik dibandingkan Heap Sort dalam hal waktu eksekusi. Quick Sort lebih unggul pada kondisi data terurut dan terbalik, sedangkan Merge Sort menunjukkan performa yang lebih konsisten dan sedikit lebih optimal pada data acak berukuran besar. Oleh karena itu, pemilihan algoritma yang tepat perlu mempertimbangkan karakteristik data yang akan diolah.

4.3 Analisis Penggunaan Memori

Selain waktu eksekusi, penggunaan memori (space complexity) juga menjadi aspek penting dalam mengevaluasi kinerja algoritma pengurutan. Pada penelitian ini, analisis penggunaan memori dilakukan secara teoritis berdasarkan karakteristik masing-masing algoritma yang digunakan.

Merge Sort merupakan algoritma yang membutuhkan memori tambahan yang cukup besar karena menggunakan array sementara dalam proses penggabungan data. Kompleksitas ruang dari Merge Sort adalah O(n), sehingga kebutuhan memorinya akan meningkat seiring dengan bertambahnya jumlah data yang diproses. Hal ini menjadi salah satu kelemahan utama dari Merge Sort, meskipun algoritma ini memiliki kestabilan dalam hal waktu eksekusi.

Quick Sort memiliki penggunaan memori yang lebih efisien dibandingkan Merge Sort karena tidak memerlukan array tambahan yang besar. Namun, algoritma ini menggunakan rekursi yang membutuhkan ruang pada stack memori, dengan kompleksitas ruang rata-rata sebesar O(log n) dan dapat mencapai O(n) pada kondisi terburuk. Dengan demikian, penggunaan memori Quick Sort masih tergolong efisien, terutama pada kondisi rata-rata.

Sementara itu, Heap Sort merupakan algoritma yang paling efisien dalam penggunaan memori karena bersifat in-place, yaitu tidak memerlukan memori tambahan yang signifikan selain beberapa variabel bantu. Kompleksitas ruang Heap Sort adalah O(1), sehingga algoritma ini sangat cocok digunakan pada kondisi dengan keterbatasan memori.

Berdasarkan analisis tersebut, dapat disimpulkan bahwa Heap Sort memiliki keunggulan dalam efisiensi penggunaan memori, diikuti oleh Quick Sort, dan kemudian Merge Sort sebagai algoritma dengan kebutuhan memori terbesar. Oleh karena itu, dalam pemilihan algoritma, aspek penggunaan memori perlu dipertimbangkan selain waktu eksekusi, terutama pada sistem dengan sumber daya terbatas.

5. KESIMPULAN DAN SARAN

Berdasarkan penelitian yang telah dilakukan, dapat disimpulkan bahwa algoritma Quick Sort, Merge Sort, dan Heap Sort menunjukkan perbedaan kinerja dalam proses pengurutan data. Hasil pengujian memperlihatkan bahwa waktu eksekusi ketiga algoritma cenderung meningkat seiring dengan bertambahnya jumlah data yang diproses. Quick Sort memiliki performa yang lebih baik pada kondisi data terurut dan terbalik, serta tetap menunjukkan kinerja yang baik pada data acak. Sementara itu, Merge Sort unggul dalam menjaga konsistensi performa pada berbagai kondisi data dan menunjukkan hasil yang lebih optimal pada data acak dengan ukuran besar. Di sisi lain, Heap Sort memiliki waktu eksekusi yang relatif lebih tinggi dibandingkan kedua algoritma lainnya. Dari aspek penggunaan memori, Heap Sort merupakan algoritma yang paling efisien karena tidak memerlukan memori tambahan yang besar, diikuti oleh Quick Sort, sedangkan Merge Sort membutuhkan alokasi memori tambahan yang lebih besar. Dengan demikian, pemilihan algoritma pengurutan yang tepat perlu disesuaikan dengan karakteristik data serta kebutuhan sistem, baik dari segi efisiensi waktu maupun penggunaan memori.

DAFTAR REFERENSI

Ali, M. I., Fardiarsyah, R. D., Shodik, L., Kinanti, F. Z. W. & Pujiono, I. P. (2025). Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C++. *LogicLink: Journal of Artificial Intelligence and Multimedia in Informatics*, 2(1) 1-17

Pujiono, I. P., Kamal, M. R., Prayogi, A., Sari, C. A. & Ikhsanuddin, R. M. (2025). Algoritma Counting Sort Vs Algoritma Pengurutan Modern: Analisis Efisiensi Memori Dan Waktu Komputasi. *JITET (Jurnal Informatika dan Teknik Elektro Terapan)*, 13(3), 135-142

Saputra, D. A., Ramadhani, M. S., Failandri, M. A., Turmudi, A. & Pujiono, I. P. (2025). Analisis Perbandingan Algoritma Sorting Dalam Javascript Untuk Meningkatkan Efisiensi Pengurutan Produk Pada Aplikasi E-Commerce. *SAINSTECH Jurnal Penelitian dan Pengkajian Sains dan Teknologi*, 35(2), 1-10

Tumanggor, H. Y., Lubis, R. M. F., Sianturi, M., P. & Purba, R. G. (2022). Metode Algoritma Bubble Sort, Algoritma Merge Sort Dan Algoitma Quick Sort Dalam Pengujian Perbandingan Proses Penelitian Kualitatif. *JUTISAL (Jurnal Teknik Informatika Komputer Universal)*, 2(2), 47-58

Pujiono, I. P., Rachmawanto, E. H. & Winarsih, N. A. S. (2025). Array Sorting Algorithm vs Algoritma Pengurutan Tradisional: Analisis Efisiensi Memori dan Waktu. *Jurnal Manajemen Informatika (JAMIKA)*, 15(1), 47-59

Aryanto, R. P., Nilogiri, A. & Wardoyo, A. E. (2023). Optimasi Pengurutan Data Bilangan dengan Menggabungkan Algoritma Selection Sort Hybrid dan Bucket Sort. *Edumatic: Jurnal Pendidikan Informatika*, 7(1), 39-48

Iskandar, J., Suhendar, H. & Pamungkas, B. D. (2024). Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python. *G-Tech : Jurnal Teknologi Terapan*, 8(1), 104-113

Sunandar, E. & Indrianto. (2020). Implementasi Algoritma Bubble Sort Terhadap 2 Buah Model Varian Pengurutan Data Menggunakan Bahasa Program Java. *PETIR: Jurnal Pengkajian dan Penerapan Teknik Informatika*, 13(2), 225-265

Wijaya, S., Fauziah & 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), 294-302

Rahmawati, Y., & Setiawan, H. (2026). Analisis Perbandingan Algoritma Sorting terhadap Data Numerik, Karakter, dan String di Python Berdasarkan Waktu, Memori Puncak, dan Perpindahan Data. *JATI (Jurnal Mahasiswa Teknik Informatika)*, *10*(2), 3134-3138

Mahrozi, N., & Faisal, M. (2023). Analisis perbandingan kecepatan algoritma selection sort dan bubble sort. *Scientica: Jurnal Ilmiah Sain dan Teknologi*, *1*(2), 89-98

Musyaafa, M. N., Simbolon, A. B., Azis, K. R., & Niska, D. Y. (2025). Pengembangan Aplikasi Benchmarking Waktu Komputasi Algoritma Sorting Menggunakan Octave. *JATI (Jurnal Mahasiswa Teknik Informatika)* , *9*(4), 6196–6200

Astuti, Y. A. (2023). Analisis Pengujian Data Algoritma Bubble Sort. *Remik: Riset dan E-Jurnal Manajemen Informatika Komputer*, *7*(3), 1413–1417

Purnomo, R., & Putra, T. D. (2023). Theoretical Analysis of Standard Selection Sort Algorithm. *Sinkron: Jurnal dan Penelitian Teknik Informatika*, *7*(2), 512–519. https://doi.org/10.33395/sinkron.v8i2.12153

Yuanggara, V., & Mahenra, R. (2026). Perbandingan Efisiensi Algoritma Sorting Hybrid untuk Data Skala Menengah. *DINAMIK*, *31*(1), 84-94

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), 117–124. http://dx.doi.org/10.30998/string.v5i2.6250

Rijaya, R., & Rivan, M. E. A. (2023). Perbandingan Penempatan Pivot Pada Quick Sort Berdasarkan Ukuran Pemusatan Data. *Jurnal Algoritme*, *4*(1), 13–20. https://doi.org/10.35957/algoritme.xxxx

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

Amanda, S., Anastasya, L. D., Sulistia, F., Purnamasari, N., & Pujiono, I. P. (2026). Analisis Perbandingan Efisiensi Algoritma Introsort dengan Algoritma Tradisional Bubble Sort dan Selection Sort. *Jurnal Elektro & Informatika Swadharma (JEIS)*, *6*(1), 155-165

Ilham, M. N., Setiawan, A. F., Kholifatun, I., Aldiansyah, M. H., & Pujiono, I. P. (2025). Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C++ Programming Language. *Journal of Artificial Intelligence and Engineering Applications*, *4*(3), 1950-1956

Musyaffa, M. Z., Raharjo, K., Faiz, M., Ammarullah, S., & Pujiono, I. P. (2024). Efisiensi Memori dan Waktu: Array Sorting Algorithm vs Algoritma Pengurutan Tradisional Menggunakan Python. *JURNAL ELEKTRO & INFORMATIKA SWADHARMA (JEIS)*, 5(2), 72-81

Yusuf, A., & Ramadhani, Y. (2024). Analisis Algoritma Bubble Sort Ascending/Descending dan Implementasinya Menggunakan Bahasa Pemrograman Python. *JISCO (Journal of Information System and Computing)*, *2*(2), 53–57

Aryanto, R. P., Nilogiri, A., & Wardoyo, A. E. (2023). Optimasi Pengurutan Data Bilangan dengan Menggabungkan Algoritma Selection Sort Hybrid dan Bucket Sort. *Edumatic: Jurnal Pendidikan Informatika*, *7*(1), 39–48. https://doi.org/10.29408/edumatic.v7i1.12358