JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3107-3113

Studi Komparatif Algoritma Pengurutan Hibrida dan Klasik: Evaluasi Timsort dan Quick Sort Berdasarkan Skenario Karakteristik Data

M. Zikry Sahendra Chan[1], Qatrunnada Athirah H[2], Chelsea Febiola Sebayang[3]

[1,2,3] Fakultas Matematika dan Ilmu Pengetahuan Alam, Ilmu Komputer, Universitas Negeri Medan, Medan, Indonesia
Email: [1]zikrysahendra@gmail.com, [2*]qatrunn.2006@gmail.com, [3]chelsea.sby04@gmail.com
(* : coressponding author)

Abstrak− Algoritma pengurutan (*sorting algorithm*) merupakan salah satu komponen fundamental dalam pengolahan data karena berperan penting dalam meningkatkan efisiensi proses pencarian, analisis, dan pengelolaan data. Namun, performa algoritma pengurutan sangat dipengaruhi oleh karakteristik distribusi data yang digunakan. Penelitian ini bertujuan untuk melakukan analisis komparatif terhadap kinerja algoritma *Quick Sort* sebagai algoritma klasik dan *Timsort* sebagai algoritma *hybrid sorting* berdasarkan variasi karakteristik dataset. Metode penelitian yang digunakan adalah pendekatan kuantitatif eksperimental melalui pengujian waktu eksekusi pada empat skenario data, yaitu data acak, data terurut, data terurut terbalik, dan data dengan banyak elemen duplikat, dengan ukuran dataset 1000, 2500, dan 5000 elemen. Hasil pengujian menunjukkan bahwa *Quick Sort* mengalami penurunan performa yang signifikan pada kondisi data terurut dan terurut terbalik, sedangkan *Timsort* menunjukkan performa yang lebih stabil dan adaptif pada seluruh skenario pengujian. Selain itu, *Timsort* juga lebih efisien dalam menangani dataset dengan banyak elemen duplikat. Oleh karena itu, algoritma *Timsort* direkomendasikan sebagai solusi yang lebih optimal dibandingkan *Quick Sort* klasik dalam pengolahan data dengan karakteristik distribusi yang bervariasi.

Kata Kunci: *sorting algorithm*, *Quick Sort*, *Timsort*, analisis komparatif, karakteristik dataset

Abstract− *Sorting algorithms are a fundamental component of data processing because they play a crucial role in improving the efficiency of data search, analysis, and management. However, the performance of sorting algorithms is significantly influenced by the distribution characteristics of the data used. This study aims to conduct a comparative analysis of the performance of the Quick Sort algorithm, a classical algorithm, and Timsort, a hybrid sorting algorithm, based on varying dataset characteristics. The research method used is a quantitative experimental approach through execution time testing on four data scenarios: random data, sorted data, reverse sorted data, and data with many duplicate elements, with dataset sizes of 1000, 2500, and 5000 elements. The test results show that Quick Sort experiences a significant performance decrease in both sorted and reverse sorted data conditions, while Timsort demonstrates more stable and adaptive performance across all test scenarios. Furthermore, Timsort is also more efficient in handling datasets with many duplicate elements. Therefore, the Timsort algorithm is recommended as a more optimal solution than the classical Quick Sort for processing data with varying distribution characteristics. Keywords: sorting algorithm, Quick Sort, Timsort, comparative analysis, dataset characteristics*

Keywords*: sorting algorithm, Quick Sort, Timsort, comparative analysis, dataset characteristics*

1. PENDAHULUAN

Perkembangan teknologi informasi yang pesat menyebabkan peningkatan volume data yang harus diproses secara cepat dan efisien. Dalam proses pengolahan data digital, algoritma pengurutan (sorting algorithm) merupakan salah satu operasi fundamental yang berperan penting dalam meningkatkan efisiensi pencarian data, pengolahan basis data, analisis statistik, serta berbagai aplikasi komputasi modern lainnya. Proses sorting tidak hanya berfungsi untuk menyusun data secara sistematis, tetapi juga menjadi dasar bagi berbagai algoritma lanjutan seperti searching, data mining, dan sistem analitik berskala besar (Nazril Ilham et al., 2025).

Sorting merupakan permasalahan klasik dalam ilmu komputer yang tetap relevan hingga saat ini karena efisiensi algoritma pengurutan sangat mempengaruhi performa keseluruhan sistem komputasi modern(Pujiono et al., 2025). Berbagai algoritma pengurutan telah dikembangkan dengan pendekatan yang berbeda, seperti divide and conquer, comparison-based sorting, maupun algoritma hybrid sorting modern yang dirancang untuk meningkatkan stabilitas dan efisiensi pada berbagai karakteristik dataset. Setiap algoritma memiliki karakteristik kompleksitas waktu dan

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3107-3113

penggunaan memori yang berbeda, sehingga pemilihan algoritma yang tepat sangat bergantung pada ukuran dataset serta distribusi data yang digunakan dalam proses komputasi(Prof.Mrs. Tejaswini.A. Puranik, 2025; Sundaramoorthy & Karunanidhi, 2025).

Salah satu algoritma pengurutan yang banyak digunakan adalah Quick Sort, yang memiliki kompleksitas waktu rata-rata sebesar $O(n\log n)$, namun dapat mengalami penurunan performa menjadi $O(n^2)$pada kondisi tertentu seperti data yang telah terurut atau memiliki distribusi nilai yang tidak seimbang. Hal ini menunjukkan bahwa karakteristik data input merupakan faktor penting yang mempengaruhi performa algoritma pengurutan secara nyata. Oleh karena itu, analisis performa algoritma tidak cukup hanya dilakukan secara teoritis melalui kompleksitas asimtotik, tetapi perlu didukung dengan pengujian empiris pada berbagai kondisi dataset (Analysis of Time Complexity in Sorting Algorithms, 2023).

Seiring berkembangnya kebutuhan pengolahan data modern, muncul algoritma *hybrid sorting* seperti *Timsort* yang menggabungkan keunggulan *Merge Sort* dan *Insertion Sort* untuk meningkatkan efisiensi pengurutan pada dataset nyata yang umumnya memiliki pola tertentu seperti data yang sebagian telah terurut (*partially sorted data*) (Rabiu et al., 2022). Pendekatan *hybrid sorting* dirancang untuk meningkatkan stabilitas performa algoritma serta mengurangi kemungkinan terjadinya degradasi kinerja pada kondisi dataset tertentu dibandingkan algoritma klasik (Ali, 2022).

Beberapa penelitian sebelumnya telah melakukan analisis perbandingan performa algoritma pengurutan berdasarkan ukuran dataset dan kompleksitas waktu eksekusi. Misalnya, penelitian Ilham dkk. membandingkan lima algoritma pengurutan dan menunjukkan bahwa performa algoritma sangat dipengaruhi oleh skala dataset serta bahasa pemrograman yang digunakan dalam implementasi(Nazril Ilham et al., 2025). Selain itu, penelitian Sundaramoorthy dan Karunanidhi menunjukkan bahwa karakteristik input data seperti tipe data dan distribusi nilai sangat mempengaruhi efektivitas algoritma pengurutan dalam praktik nyata (Sundaramoorthy & Karunanidhi, 2025). Penelitian lain juga menekankan bahwa pemilihan algoritma pengurutan tidak hanya ditentukan oleh kompleksitas teoritis, tetapi juga oleh kondisi distribusi data serta lingkungan komputasi yang digunakan (Prof.Mrs. Tejaswini.A. Puranik, 2025).

Meskipun berbagai penelitian telah membahas analisis performa algoritma pengurutan, sebagian besar penelitian tersebut masih berfokus pada perbandingan antar algoritma klasik atau dilakukan pada variasi ukuran dataset tanpa mempertimbangkan secara spesifik pengaruh karakteristik distribusi data terhadap performa algoritma hibrida modern seperti Timsort. Selain itu, studi yang secara langsung membandingkan performa Quick Sort klasik dengan Timsort dalam berbagai skenario distribusi data seperti data acak, data terurut, data terbalik, dan data dengan banyak elemen duplikat masih relatif terbatas, khususnya dalam konteks pengujian empiris menggunakan lingkungan implementasi yang seragam.

Berdasarkan kesenjangan penelitian tersebut, penelitian ini bertujuan untuk melakukan studi komparatif terhadap performa algoritma Quick Sort sebagai representasi algoritma klasik dan Timsort sebagai representasi algoritma pengurutan hibrida modern melalui pendekatan eksperimental. Evaluasi dilakukan berdasarkan variasi karakteristik dataset yang meliputi data acak, data terurut, data terurut terbalik, dan data dengan banyak elemen duplikat. Hasil penelitian ini diharapkan dapat memberikan kontribusi empiris dalam memahami pengaruh karakteristik distribusi data terhadap performa algoritma pengurutan serta menjadi referensi dalam menentukan algoritma yang paling optimal untuk berbagai kondisi pengolahan data modern.

2. METODE

2.1 Desain Penelitian

Penelitian ini menggunakan pendekatan kuantitatif eksperimental. Peneliti melakukan perbandingan kinerja (*benchmarking*) antara algoritma pengurutan klasik (Quick Sort) dan algoritma pengurutan hibrida adaptif (Timsort). Evaluasi difokuskan pada pengukuran waktu komputasi (*running time*) dalam merespons berbagai variasi karakteristik kondisi awal data pada skala yang telah ditentukan (Iskandar et al., 2023).

2.2 Lingkungan Komputasi (Test Environment)

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3107-3113

Untuk memastikan validitas dan reproduktibilitas penelitian, seluruh eksperimen dijalankan pada lingkungan komputasi tunggal yang terisolasi untuk meminimalisasi intervensi dari proses latar belakang (background processes). Spesifikasi lingkungan pengujian yang digunakan adalah sebagai berikut:

a. Perangkat Keras: Prosesor Intel Core i5-12450H @ 2.40GHz, RAM 24 GB DDR4, dan media penyimpanan SSD NVMe 512 GB.
b. Perangkat Lunak: Sistem Operasi Windows 11 64-bit, Interpreter Python versi 3.13.2, dan pustaka visualisasi data Matplotlib.

2.3 Skenario dan Karakteristik Data Uji

Dataset yang digunakan dalam penelitian ini merupakan data sintetis (*dummy data*) bertipe *integer* yang dibangkitkan secara otomatis melalui pustaka random pada Python. Pengujian dilakukan pada tiga variasi ukuran array (N), yaitu N = 1000, N = 2500, dan N = 5000 elemen. Untuk menguji tingkat adaptabilitas algoritma, masing-masing ukuran diuji menggunakan empat skenario karakteristik data:

a. Data Acak (Random Case): Elemen array diacak secara merata. Skenario ini bertindak sebagai baseline untuk mengukur kinerja rata-rata (average-case performance).
b. Data Terurut (Sorted / Best-Case untuk Hibrida): Elemen array telah terurut secara menaik (ascending). Skenario ini dirancang untuk menguji deteksi "runs" pada Timsort serta memicu degradasi kinerja (worst-case) pada Quick Sort.
c. Data Acak (Random Case): Elemen array diacak secara merata. Skenario ini bertindak sebagai baseline untuk mengukur kinerja rata-rata (average-case performance).
d. Data Terurut (Sorted / Best-Case untuk Hibrida): Elemen array telah terurut secara menaik (ascending). Skenario ini dirancang untuk menguji deteksi "runs" pada Timsort serta memicu degradasi kinerja (worst-case) pada Quick Sort.

2.4 Implementasi Algoritma dan Batasan Sistem

a. Quick Sort Klasik: Diimplementasikan secara kustom menggunakan skema partisi Lomuto (Lomuto partition scheme) dengan elemen terakhir (rightmost element) sebagai pivot. Pemilihan skema ini sengaja dilakukan untuk mereplikasi kerentanan algoritma klasik terhadap kompleksitas waktu O(N2).
b. Timsort: Menggunakan implementasi bawaan bahasa pemrograman Python yang dipanggil melalui metode list.sort().
c. Modifikasi Limitasi Rekursi: Mengingat arsitektur Python memiliki batas pemanggilan rekursif bawaan (default recursion limit) sekitar 1000 pemanggilan, peneliti melakukan intervensi dengan fungsi sys.setrecursionlimit(20000). Hal ini krusial untuk mencegah terjadinya RecursionError saat Quick Sort mengeksekusi skenario worst-case pada array berukuran di atas 1000 elemen, sehingga metrik waktu tetap dapat direkam secara utuh.

2.5 Parameter Evaluasi

Metrik utama yang dievaluasi dalam penelitian ini adalah waktu eksekusi aktual (*actual execution time*). Pengukuran dilakukan menggunakan fungsi time.time() pada Python yang mencatat waktu sesaat sebelum algoritma dipanggil hingga proses pengurutan selesai. Hasil kalkulasi selisih waktu tersebut kemudian dikonversi ke dalam satuan milidetik (ms) untuk analisis komparatif lebih lanjut.

3. ANALISA DAN PEMBAHASAN

3.1 Hasil

Penyajian data mentah hasil pengukuran waktu eksekusi (running time) dari kedua algoritma, yaitu Quick Sort klasik dan Timsort (bawaan Python). Data dikumpulkan berdasarkan empat skenario karakteristik data (Acak, Terurut, Terurut Terbalik, dan Duplikat) pada tiga variasi ukuran array (N=1000, 2500, 5000).
Hasil rekapitulasi waktu eksekusi dalam satuan milidetik (ms) dirangkum dalam Tabel 1 berikut:

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3107-3113

Tabel 1. Rekapitulasi Waktu Eksekusi (ms) Quick Sort vs Timsort

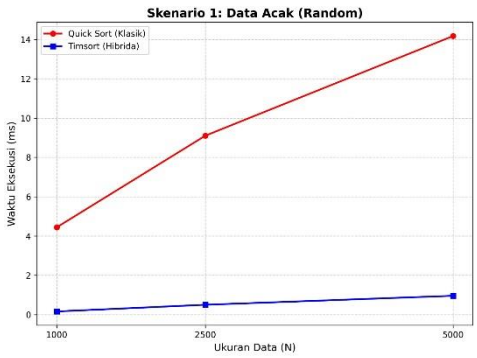
Ukuran Data (N)	Skenario Data	Waktu Quick Sort (ms)	Waktu Timsort (ms)	Faktor Pelambatan QS*
1000	Acak (*Random*)	4.4434	0.1628	27x
	Terurut (*Sorted*)	72.6888	0.0107	6793x
	Terbalik (*Reverse*)	69.0575	0.0122	5660x
	Duplikat (*Duplicate*)	52.2621	0.0556	940x
2500	Acak (*Random*)	9.1054	0.5026	18x
	Terurut (*Sorted*)	461.0436	0.0207	22272x
	Terbalik (*Reverse*)	329.0470	0.0141	23336x
	Duplikat (*Duplicate*)	180.8879	0.1757	1029x
5000	Acak (*Random*)	14.1737	0.9594	15x
	Terurut (*Sorted*)	1380.0998	0.0377	36607x
	Terbalik (*Reverse*)	2319.9353	0.0257	90270x
	Duplikat (*Duplicate*)	652.9014	0.1829	3570x

Faktor Pelambatan QS dihitung dari (Waktu Quick Sort / Waktu Timsort), menunjukkan berapa kali lipat Quick Sort lebih lambat dibanding Timsort.
Pembahasan dan Analisis Kinerja

3.2 Pembahasan dan Analisis Kinerja

Berdasarkan data eksperimen yang telah disajikan pada Tabel 4.1, peneliti melakukan analisis mendalam terhadap perilaku masing-masing algoritma yang divisualisasikan secara terpisah berdasarkan skenario uji.

3.2.1 Analisis Skenario Data Acak (Random Case)

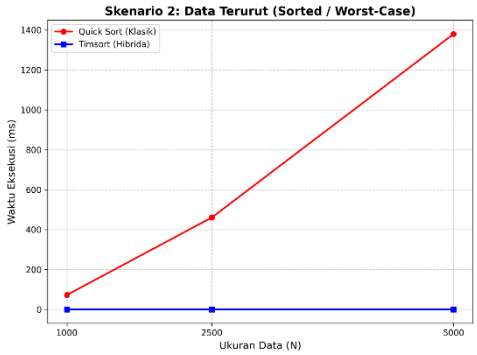


Gambar 1. Visualisasi kinerja Algoritma pada Skenario Data Acak

Berdasarkan Gambar 4.1, pada skenario data acak, kedua algoritma menunjukkan kurva pertumbuhan waktu yang stabil dan landai seiring bertambahnya ukuran data (N). Hal ini sesuai dengan landasan teoritis bahwa Quick Sort dan Timsort memiliki kompleksitas waktu rata-rata (average-case) sebesar O(N log N). Timsort terlihat beroperasi sedikit lebih cepat secara konstan (berkisar 0.95 ms pada N=5000) dibandingkan Quick Sort kustom (14.17 ms). Perbedaan ini murni disebabkan oleh overhead interpretasi; Timsort dieksekusi menggunakan bahasa C yang sangat teroptimasi di bawah kap mesin Python, sementara Quick Sort berjalan sebagai kode Python murni.

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3107-3113

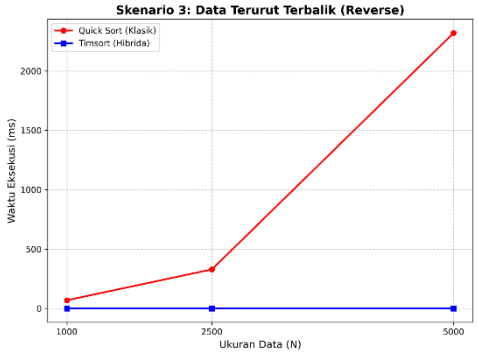
3.2.2 Analisis Skenario Data Terurut (Sorted / Worst-Case)



Gambar 2. Visualisasi Kinerja algoritma pada Skenario Data Terurut

Gambar 2 menyajikan temuan yang sangat kontras. Kurva Quick Sort melesat naik secara eksponensial, menjadi bukti visual terjadinya degradasi kinerja dari O(N log N) menjadi kompleksitas waktu kuadratik O(N2). Penggunaan elemen terakhir sebagai pivot pada skema Lomuto menyebabkan array terbagi secara tidak seimbang (bagian berukuran 1 dan N-1) secara berulang. Sebaliknya, garis Timsort tetap datar mendekati sumbu X. Timsort berhasil mengidentifikasi "runs" (potongan data terurut) secara langsung, menekan waktu komputasi menjadi hanya 0.0377 ms pada N=5000 berkat pencapaian kompleksitas linear O (N).

3.2.3 Analisis Skenario Data Terurut Terbalik (Reverse Sorted Case)

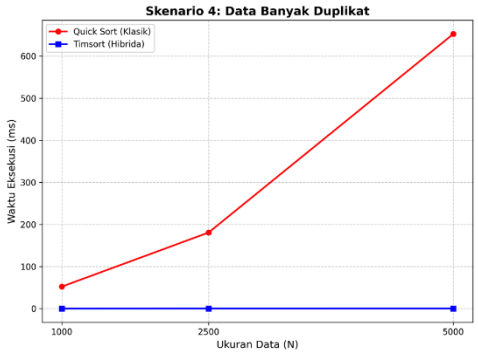


Gambar 3. Visualisasi Kinerja Algoritma pada Skenario Data Terurut Terbalik

Skenario data terurut terbalik (Gambar 3) memberikan beban partisi maksimal bagi algoritma Quick Sort. Kurva merah menunjukkan lonjakan waktu komputasi paling ekstrem dalam penelitian ini, mencapai 2319.93 ms pada N=5000 (lebih dari 90.000 kali lebih lambat dibanding Timsort). Tanpa adanya intervensi peningkatkan batas rekursi sistem (sys.setrecursionlimit), eksekusi Quick Sort pada tahap ini dipastikan akan memicu RecursionError. Di sisi lain, Timsort kembali membuktikan ketahanan sistem hibridanya. Ia mendeteksi pola descending dengan cepat, membalikannya di tingkat memori, dan menyelesaikannya dalam waktu 0.0257 ms.

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3107-3113

3.2.4 Analisis Skenario Data Banyak Duplikat (Many Duplicates Case)



Gambar 4. Visualisasi Kinerja Algoritma pada Skenario Data Mayoritas Duplikat

Gambar 4 mengonfirmasi kerentanan implementasi Quick Sort terhadap kolisi nilai. Keberadaan 90% elemen duplikat menyebabkan algoritma sering membandingkan nilai yang sama, sehingga pivot kembali gagal membagi data ke kiri dan kanan secara merata. Hal ini terlihat dari kurva Quick Sort yang mulai melengkung tajam ke atas, mencatatkan waktu 652.90 ms untuk N=5000. Timsort, berkat pondasi Merge Sort yang stabil dalam menggabungkan blok-blok memori, mampu menangani anomali duplikat ini tanpa menunjukkan lonjakan kurva yang berarti (0.1829 ms).

4. KESIMPULAN

Berdasarkan hasil pengujian empiris dan analisis komparatif yang telah dilakukan terhadap algoritma *Quick Sort* klasik dan *Timsort* bawaan Python pada berbagai skenario karakteristik data, dapat disimpulkan bahwa *Quick Sort* dengan skema partisi Lomuto menunjukkan kerentanan yang signifikan terhadap kondisi *worst-case*, khususnya pada dataset yang telah terurut (*sorted*) dan terurut terbalik (*reverse sorted*), sehingga mengalami degradasi kompleksitas waktu dari $O(N \log N)$menjadi $O(N^2)$. Pada pengujian dengan 5000 elemen data terurut terbalik, kinerja *Quick Sort* bahkan tercatat lebih dari 90.000 kali lebih lambat dibandingkan *Timsort*. Sebaliknya, *Timsort* sebagai algoritma *hybrid sorting* terbukti memiliki tingkat adaptabilitas yang tinggi karena mampu mendeteksi keberadaan sub-array yang telah terurut (*runs*) dan memanfaatkannya untuk menekan kompleksitas waktu komputasi hingga mendekati linear $O(N)$, yang terlihat dari waktu eksekusi yang jauh lebih cepat pada data terurut dibandingkan data acak. Selain itu, pendekatan *Merge Sort* yang menjadi fondasi *Timsort* membuat algoritma ini lebih stabil dalam menangani dataset dengan banyak elemen duplikat dibandingkan *Quick Sort* klasik yang cenderung menghasilkan partisi pivot tidak seimbang. Secara keseluruhan, hasil penelitian menunjukkan bahwa algoritma *hybrid sorting* modern seperti *Timsort* menawarkan stabilitas performa yang lebih konsisten serta ketahanan yang lebih baik terhadap berbagai karakteristik dataset dibandingkan algoritma klasik, sehingga lebih direkomendasikan untuk implementasi pengolahan data dalam aplikasi perangkat lunak modern.

REFERENCES

Ali, R. (2022). Hardware Solution to Sorting Algorithms: A Review. *Journal of Global Research in Computer Sciences Research and Reviews: Journal of Global Research in Computer Sciences RRJGRCS|*, *13*, 2022. https://doi.org/10.4172/2229

Analysis of Time Complexity in Sorting Algorithms. (n.d.). Retrieved https://www.programiz.com/dsa/counting-sort

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

Nazril Ilham, M., Faza Setiawan, A., Kholifatun, I., & Aldiansyah, M. H. (2025). *Journal of Artificial Intelligence and Engineering Applications Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C++ Programming Language* (Vol. 4, Number 3). https://ioinformatic.org/

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3107-3113

Prof.Mrs. Tejaswini.A. Puranik. (2025). Performance Analysis of Sorting and Searching Algorithms. *International Research Journal on Advanced Engineering and Management (IRJAEM)*, *3*(08), 2741–2746. https://doi.org/10.47392/irjaem.2025.0430

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. *Jurnal Informatika Dan Teknik Elektro Terapan*, *13*(3). https://doi.org/10.23960/jitet.v13i3.6657

Rabiu, A. M., Garba, E. J., Baha, B. Y., & Mukhtar, M. I. (2022). Comparative Analysis between Selection Sort and Merge Sort Algorithms. *Nigerian Journal of Basic and Applied Sciences*, *29*(1), 43–48. https://doi.org/10.4314/njbas.v29i1.5

Sundaramoorthy, S., & Karunanidhi, G. (2025). A systematic analysis on performance and computational complexity of sorting algorithms. In *Discover Computing* (Vol. 28, Number 1). Springer Science and Business Media B.V. https://doi.org/10.1007/s10791-025-09724-w