Teknik: Jurnal Ilmu Teknik dan Informatika
Volume 6 Nomor 1 Mei 2026
e-ISSN: 2798-2513; p-ISSN: 2808-8751, Hal *60-79*
DOI: https://doi.org/10.51903/teknik.v6i1.1145
ble online at: https://journal.stiestekom.ac.id/index.php/TEKNIK

Perbandingan Algoritma Divide and Conquer dan Searching pada Pengolahan Data Nilai Mahasiswa Berbasis Web

Nazwa Aidilia Octa Mevia[1*], Yohana Kartika Marbun[2], Melika Debiyana Putri[3], Yunanda Rizki Sitompul[4]
[1-4]Universitas Negeri Medan, Indonesia

Korespondensi penulis: nazwaaidila9@gmail.com

Abstract*. The rapid digital transformation in educational institutions demands an efficient student grade data processing system capable of handling workloads responsively. This study aims to analyze and compare the efficiency of sorting algorithms (Merge Sort and Quick Sort) and searching algorithms (Linear Search and Binary Search) on a web-based platform. The research method employed is laboratory experimental, testing algorithm performance across various data volume stratifications, ranging from 50 to 1000 entities, using the V8 JavaScript engine. Research findings indicate that Quick Sort possesses superior speed compared to Merge Sort due to its efficient in-place sorting architecture, which minimizes memory overhead and Garbage Collection activity. Furthermore, a performance anomaly was discovered where the Just-In-Time (JIT) Compiler mechanism optimizes execution on large data volumes through a warm-up phase. In the searching aspect, Binary Search demonstrates superior O(log n) logarithmic stability compared to Linear Search, which risks causing interface freezing on massive data. The implication of this study is the critical importance of implementing data pre-sorting protocols to exploit logarithmic search speeds to ensure academic information system scalability. The integration of appropriate algorithms proves to be a crucial foundation in maintaining web application responsiveness amidst the ever-increasing escalation of educational data.*
Keywords*: Binary Search, Quick Sort, Searching Algorithm, Sorting Algorithm, Student Data.*

Abstrak. Pesatnya transformasi digital dalam institusi pendidikan menuntut efisiensi sistem pengolahan data nilai mahasiswa yang mampu menangani beban kerja secara responsif. Penelitian ini bertujuan untuk menganalisis dan membandingkan efisiensi algoritma *sorting* (*Merge Sort* dan *Quick Sort*) serta algoritma *searching* (*Linear Search* dan *Binary Search*) pada platform berbasis web. Metode penelitian yang digunakan adalah eksperimental laboratorium dengan menguji performa algoritma pada berbagai stratifikasi volume data, mulai dari 50 hingga 1000 entitas, menggunakan mesin *V8 JavaScript*. Temuan penelitian menunjukkan bahwa *Quick Sort* memiliki superioritas kecepatan dibandingkan *Merge Sort* karena efisiensi arsitektur *in-place* yang meminimalisasi beban memori dan aktivitas *Garbage Collection*. Selain itu, ditemukan anomali performa di mana mekanisme *Just-In-Time* (JIT) *Compiler* mengoptimalkan eksekusi pada volume data besar melalui fase *warm-up*. Dalam aspek pencarian, *Binary Search* mendemonstrasikan stabilitas logaritmik $O(\log n)$ yang jauh lebih efisien dibandingkan *Linear Search* yang berisiko menyebabkan kelumpuhan antarmuka (*freeze*) pada data masif. Implikasi dari penelitian ini adalah pentingnya penerapan protokol pra-pengurutan data untuk mengeksploitasi kecepatan pencarian logaritmik guna menjamin skalabilitas sistem informasi akademik. Integrasi algoritma yang tepat terbukti menjadi fondasi krusial dalam menjaga responsivitas aplikasi web di tengah ekskalasi data pendidikan yang terus meningkat.
Kata kunci: Algoritma Searching, Algoritma Sorting, Binary Search, Data Mahasiswa, Quick Sort.

1. LATAR BELAKANG

Secara fundamental, pergeseran paradigma dari manajemen data konvensional ke ekosistem berbasis web yang dinamis dalam institusi pendidikan bukan lagi sebuah pilihan, melainkan keniscayaan teknis guna menjamin aksesibilitas informasi (Muis Mappalotteng, Fathahillah, & Anas Punggawa, 2024). Namun, kegagalan sistemik sering

Diterima: 3 April 2026; Direvisi: 12 April 2026; Disetujui: 20 April 2026; Tersedia Online: 5 Mei 2026; Diterbitkan: 8 Mei 2026.

kali terjadi saat infrastruktur informasi akademik berhadapan dengan ledakan volume data mahasiswa yang bersifat eksponensial, yang memicu degradasi skalabilitas komputasi secara drastis (Purba & Rachmadi, 2022). Inti dari permasalahan ini terletak pada inefisiensi operasi pengurutan (sorting) dan pencarian (searching) dua pilar algoritmik utama yang menyokong fungsionalitas esensial seperti penentuan peringkat akademik serta analisis performa siswa (Liu, 2024; Prof.Mrs. Tejaswini.A. Puranik, 2025). Ketidakefektifan dalam pemilihan algoritma berimplikasi langsung pada pemborosan sumber daya memori dan pembengkakan waktu respons server (latency) yang tidak dapat ditoleransi dalam arsitektur aplikasi web modern (Huda et al., 2024). Hal ini menjadi sangat krusial mengingat sistem prediksi nilai atau pemrosesan akademik tingkat lanjut secara mutlak bergantung pada kecepatan ekstraksi data historis di level basis data(Aaron Afan et al., 2022; Alboaneen et al., 2022).

Paradigma Divide and Conquer, yang direpresentasikan oleh Merge Sort dan Quick Sort, selama ini diagungkan sebagai solusi mutakhir dengan jaminan kompleksitas waktu rata-rata $O(n\ log\ n)$ (Esau Taiwo, Christianah, Oluwatobi, Aderonke, & Kehinde, 2020; Mishal, Al-Khatib, & Hiasat, 2021) Akan tetapi, pengabaian terhadap aspek stabilitas struktural data dan anomali perilaku algoritma pada skenario kondisi terburuk (worst-case) menunjukkan kedangkalan analisis dalam banyak implementasi praktis (Christnatalis et al., 2024). Secara empiris, Quick Sort menyimpan kerentanan inheren; performanya dapat anjlok secara fatal hingga menyentuh kompleksitas $O(n^2)$ apabila mekanisme partisi pivot-nya dihadapkan pada distribusi data yang ekstrem (Roșca & Cărbureanu, 2025; Shorman, Rateb, Alshorman, & Shqair, n.d.). Di sisi lain, Merge Sort menggaransi stabilitas pengurutan, namun menuntut alokasi memori tambahan $O(n)$ yang membebani kapasitas memori server web secara berlebihan saat melakukan fase penggabungan (Pujiono, Rachmawanto, & Winarsih, 2025). Lebih jauh lagi, keterpautan fungsional antara algoritma sangatlah absolut; efisiensi pencarian logaritmik $O(log\ n)$ yang superior dari Binary Search mustahil direalisasikan tanpa adanya integritas pengurutan data yang solid sebagai prasyarat utamanya (Dubey & Mathur, 2017; Purnama, 2025).

Tinjauan kritis terhadap literatur yang ada mengungkap sebuah celah penelitian (gap analysis) yang sangat substansial: mayoritas studi komparatif algoritma hingga saat ini masih terjebak pada pengujian teoretis yang steril, yang hanya mengandalkan simulasi

pada dataset acak buatan (randomly generated data) dengan distribusi seragam. Penelitian terdahulu juga masih terbatas pada pengujian performa algoritma secara teoritis tanpa mempertimbangkan karakteristik data akademik nyata yang cenderung memiliki redundansi nilai tinggi (duplicate-heavy) serta kondisi partially sorted berdasarkan NIM atau abjad nama mahasiswa. Selain itu, sebagian besar penelitian hanya berfokus pada perbandingan waktu eksekusi algoritma tanpa mengevaluasi pengaruh lingkungan eksekusi JavaScript modern, seperti mekanisme Just-In-Time (JIT) Compiler pada V8 Engine.\n\nBerdasarkan permasalahan tersebut, penelitian ini berkontribusi dalam menganalisis performa algoritma Merge Sort, Quick Sort, Linear Search, dan Binary Search pada sistem pengolahan data nilai mahasiswa berbasis web. Penelitian ini juga mengevaluasi hubungan antara proses sorting dan searching dalam mendukung efisiensi serta skalabilitas sistem informasi akademik modern. (Purnomo & Putra, 2023; Suleiman Al-Kharabsheh, Mahmoud AlTurani, Mahmoud Ibrahim AlTurani, & Imhammed Zanoon, 2013). Pendekatan reduksionis semacam ini gagal memotret kompleksitas arsitektural dari data akademik di dunia nyata. Faktanya, rekam jejak nilai mahasiswa secara inheren memiliki tingkat redundansi yang ekstrem (duplicate-heavy) di mana banyak entitas memiliki nilai yang identik serta sering kali ditarik dari pangkalan data dalam kondisi yang sudah terurut sebagian (partially sorted) berdasarkan alfabet atau NIM(Kodua Wiredu, Aabaah, Wiredu Acheampong, & Aabaah, n.d.; Sabah et al., 2023). Ketidakmampuan literatur terdahulu dalam mengakomodasi karakteristik unik pangkalan data ini menyebabkan klaim efisiensi memori dan waktu eksekusi menjadi bias dan tidak relevan ketika dihadapkan pada konstrain sumber daya server web yang sesungguhnya. Oleh karena itu, urgensi dan kebaruan penelitian ini terletak pada pendobrakan kebuntuan teoretis tersebut melalui analisis empiris yang presisi terhadap algoritma Divide and Conquer serta algoritma pencarian. Melalui pengujian terukur pada berbagai variasi skala data yang secara akurat mereplikasi ekosistem nilai akademik nyata, penelitian ini bertujuan merumuskan metrik teknis yang definitif guna mengoptimalkan kecepatan, stabilitas, dan efisiensi memori dalam pengembangan sistem informasi akademik masa depan.

2. KAJIAN TEORITIS

Kompleksitas Komputasi dan Paradigma Algoritma

Secara konseptual, algoritma dalam ilmu komputer tidak sekadar dimaknai sebagai serangkaian instruksi logis untuk memecahkan masalah, melainkan sebagai instrumen krusial dalam optimasi pengelolaan sumber daya sistem. Kinerja sebuah algoritma dievaluasi secara matematis melalui pendekatan kompleksitas asimtotik, yang terbagi menjadi dua dimensi utama: kompleksitas waktu (time complexity) yang merepresentasikan durasi eksekusi instruksi, dan kompleksitas ruang (space complexity) yang mengukur konsumsi memori alokatif selama proses pemrosesan berlangsung (Liu, 2024; Prof.Mrs. Tejaswini.A. Puranik, 2025). Dalam lingkungan sistem informasi berbasis web khususnya sistem akademik cloud computing yang memiliki konstrain memori (resource-constrained) algoritma dituntut untuk memberikan output optimal dengan utilisasi sumber daya yang paling presisi. Evaluasi metrik kinerja ini menjadi fondasi teoretis mutlak, di mana efisiensi algoritma diuji tidak hanya pada skenario optimal (best-case), melainkan juga pada kondisi rata-rata (average-case) dan kondisi ekstrem (worst-case) ketika memproses kumpulan data mentah (raw datasets) (Purnomo & Putra, 2023; Suleiman Al-Kharabsheh et al., 2013).

Analisis Kritis Algoritma Divide and Conquer

Paradigma Divide and Conquer memecah kompleksitas komputasi dengan mendekomposisi masalah masif menjadi sub-masalah independen secara rekursif, sebagaimana diimplementasikan pada Merge Sort dan Quick Sort untuk menangani data skala besar (Christnatalis et al., 2026). Merge Sort diakui karena menjamin stabilitas pengurutan dengan waktu eksekusi yang konsisten pada kompleksitas $O(n\ log\ n)$. Akan tetapi, algoritma ini memiliki limitasi spasial karena menuntut alokasi memori tambahan $O(n)$ pada fase penggabungan, yang berisiko memicu anomali pemborosan memori server (memory overhead) (AL-Azzam & Qatawneh, 2018; Pujiono et al., 2025).

Sebagai antitesis, Quick Sort mengeksekusi pengurutan di tempat (in-place sorting) sehingga meniadakan kebutuhan memori tambahan, serta menawarkan supremasi kecepatan yang sering kali mengungguli Merge Sort pada kasus praktis(Mishal et al., 2021; Saputra, Meidiansyah, Sanputra, & Wardana, 2025). Meski demikian, algoritma ini menyimpan kerentanan teoretis yang fatal; apabila pemilihan pivot tidak adaptif terutama pada dataset terurut sebagian atau tersebar miring kompleksitas waktunya akan terdegradasi drastis menuju skenario terburuk $O(n^2)$. Hal ini membuktikan bahwa

stabilitas performa Quick Sort sangat fluktuatif dan murni bergantung pada metodologi partisinya (Shorman et al., n.d.).

Dependensi Algoritma Pencarian (Searching) pada Keterurutan Data

Operasi pencarian dalam pangkalan data mempresentasikan tantangan tersendiri yang secara logis bergantung pada hasil keluaran operasi pengurutan. Linear Search (Sequential Search) merepresentasikan pendekatan brute-force yang menginspeksi setiap elemen satu per satu hingga target ditemukan. Fleksibilitas utamanya terletak pada kemampuannya mengeksekusi himpunan data acak tanpa prasyarat keterurutan, namun kompleksitas waktu $O(n)$ menjadikannya opsi yang sangat inefisien dan obsolet untuk diterapkan pada sistem pencarian data akademik berskala institusi (Homepage, Tri Ridha Ridwan, & Alam, 2025).

Sebaliknya, Binary Search mendobrak batasan linieritas tersebut dengan menerapkan prinsip pemotongan rentang pencarian berbasis dikotomi secara rekursif, menghasilkan efisiensi logaritmik $O(log\ n)$ yang sangat impresif pada pangkalan data skala masif (Kurhade & Takawale, 2024). Meskipun demikian, postulat utama dari algoritma ini menyatakan bahwa efisiensi Binary Search hanya dapat terwujud apabila himpunan data telah melalui proses pengurutan yang sempurna sebelumnya. Oleh karena itu, terdapat korelasi dan dependensi algoritmik yang absolut; performa superior dari Binary Search merupakan produk turunan dari integritas dan kecepatan eksekusi algoritma *sorting* yang mendahuluinya (Purnama, 2025; Ukuran dan Kondisi Keterurutan Data, Abdul Rahman, Indra Junaedi, & Santika, 2025).

Karakteristik Komputasi pada Arsitektur Sistem Informasi Akademik

Implementasi sistem informasi akademik dalam ekosistem berbasis web menuntut pemahaman mendalam mengenai arsitektur pemrosesan data. Transisi menuju aplikasi berbasis web yang responsif sangat krusial untuk mendukung fitur prediksi performa akademik mahasiswa secara real-time (Alboaneen et al., 2022; Muis Mappalotteng et al., 2024) . Tantangan utama dalam arsitektur ini adalah bagaimana sistem mampu mengelola lonjakan beban kerja (workload) saat melakukan kalkulasi nilai dan penentuan kelulusan tanpa menyebabkan *bottleneck* pada antarmuka pengguna (Aaron Afan et al., 2022; Purba & Rachmadi, 2022). Oleh karena itu, integrasi algoritma dalam sistem akademik bukan sekadar masalah teknis fungsional, melainkan strategi optimasi untuk memastikan stabilitas sistem di bawah beban data masif.

Teknologi Pengembangan Sistem Web Berbasis React.js

Pengembangan sistem komparasi algoritma pada penelitian ini diimplementasikan menggunakan arsitektur antarmuka modern berbasis React.js. React.js adalah pustaka (library) JavaScript bersifat open-source yang secara spesifik dirancang untuk membangun antarmuka pengguna interaktif pada aplikasi Single-Page Application (SPA). Pemilihan teknologi ini didasarkan pada mekanisme Virtual Document Object Model (Virtual DOM) yang dimilikinya. Dalam proses eksekusi dan visualisasi data algoritmik berskala masif, memanipulasi DOM asli secara langsung akan menciptakan bottleneck (kemacetan komputasi) yang memicu degradasi performa pada peramban (browser). React.js mengatasi limitasi ini dengan membuat salinan virtual dari DOM; ketika state data berubah selama proses pengurutan berlangsung, sistem akan mengeksekusi kalkulasi diferensial (diffing algorithm) di latar belakang, lalu secara presisi hanya memperbarui elemen DOM asli yang benar-benar mengalami mutasi (Singh, Singh, Singh, & Hazela, 2024). Selain itu, arsitektur berbasis komponen (component-based) pada framework modern ini terbukti mampu mengoptimalkan performa interaktivitas halaman web secara keseluruhan meskipun memproses manipulasi data komputasional secara intensif.

Instrumen Pengukuran Waktu Presisi Tinggi (High Resolution Time API)

Guna menguji kebenaran kompleksitas algoritma secara objektif dan presisi di dalam lingkungan eksekusi JavaScript, mekanisme pencatatan waktu yang bebas dari latensi internal sangatlah mutlak. Oleh karena itu, sistem komparasi ini mengadopsi High Resolution Time API, secara spesifik mengandalkan fungsi performance.now(). Dalam benchmarking efisiensi algoritma di ekosistem web, fungsi standar seperti Date.now() tidak lagi relevan karena presisinya terikat pada jam sistem operasi dan hanya mengukur dalam skala bilangan bulat milidetik. Sebaliknya, performance.now() memberikan resolusi pengukuran waktu dengan tingkat presisi fraksional (pecahan sub-milidetik) dan bersifat monotonik. Karakteristik monotonik ini memberikan garansi bahwa waktu yang dicatat terus berjalan maju secara independen, tanpa dapat didistorsi oleh sinkronisasi jam eksternal atau fluktuasi aktivitas latar belakang sistem operasi. Pengukuran menggunakan instrumen ini merupakan standar mutakhir dalam mengevaluasi kecepatan waktu eksekusi skrip komputasional, sehingga validitas perbandingan performa waktu antara

algoritma Divide and Conquer dan algoritma pencarian dapat dipertanggungjawabkan secara saintifik (Saputra et al., 2025).

Tinjauan Studi Terdahulu dan Relevansi Karakteristik Data Akademik

Transisi menuju arsitektur web yang tangguh menjadi krusial akibat bottleneck pemrosesan data pada basis data pendidikan konvensional(Alboaneen et al., 2022; Muis Mappalotteng et al., 2024; Purba & Rachmadi, 2022). Dalam ekosistem ini, pengujian algoritma mutlak menuntut penggunaan parameter dunia nyata, mengingat metrik kinerja terbukti berfluktuasi secara ekstrem pada dataset yang terurut sebagian(Ayazuddin, 2025; Sabah et al., 2023) maupun pada data nilai akademik yang memiliki redundansi masif atau duplicate-heavy (Kodua Wiredu et al., n.d.). Mengantisipasi disrupsi masa depan yang mengarah pada komputasi GPU (Bakare, Okewu, Abiola, Jaji, & Muhammed, 2024; Capannini, Silvestri, & Baraglia, 2012), integrasi kecerdasan buatan (Mezied & Abu-Naser, 2025), serta pemanfaatan deteksi anomali visual ((Singh et al., 2024), penetapan baseline metric bagi algoritma klasik dan Divide and Conquer menjadi sangat mendesak. Berpijak pada literatur tersebut, komparasi empiris terhadap Merge Sort, Quick Sort, Linear Search, dan Binary Search menggunakan data riil menjadi landasan krusial untuk mencegah kegagalan sistemik dan menjamin skalabilitas pada platform akademik berbasis web.

Kebaruan Penelitian

Penelitian ini terletak pada pendekatan komparatif yang tidak hanya mengevaluasi kompleksitas teoretis algoritma, tetapi juga menguji perilaku algoritma pada lingkungan eksekusi web modern berbasis JavaScript V8 Engine. Berbeda dengan penelitian terdahulu yang umumnya menggunakan simulasi teoritis sederhana, penelitian ini memanfaatkan skenario data akademik yang merepresentasikan karakteristik sistem informasi pendidikan. Selain itu, penelitian ini mengintegrasikan analisis performa algoritma sorting dan searching secara berkesinambungan untuk mengevaluasi dampaknya terhadap responsivitas sistem akademik berbasis web. Penelitian juga menyoroti fenomena optimasi Just-In-Time (JIT) Compiler yang memengaruhi hasil benchmark algoritma pada browser modern. akademik berbasis web.

3. METODE PENELITIAN

Penelitian ini merupakan penelitian komparatif dengan pendekatan kuantitatif. Metode komparatif digunakan secara spesifik untuk membandingkan performa arsitektur

algoritma *sorting* (Merge Sort dan Quick Sort) serta algoritma searching (Linear Search dan Binary Search) dalam pengolahan data nilai mahasiswa. Pendekatan kuantitatif dipilih karena penelitian ini berfokus pada pengukuran performa algoritma secara objektif menggunakan indikator numerik, seperti waktu eksekusi dan kompleksitas algoritma. Melalui pendekatan ini, hasil penelitian diharapkan dapat memberikan gambaran yang lebih jelas dan terukur mengenai tingkat efisiensi masing-masing algoritma dalam berbagai kondisi data.

Perangkat dan Teknologi Eksperimen

Memastikan validitas eksperimen dan mencegah bias performa dari variabel pengganggu, pengujian dieksekusi pada lingkungan pengembangan (development environment) yang terisolasi. Arsitektur front-end dibangun menggunakan pustaka React.js; pemanfaatan teknologi berbasis komponen dan Virtual DOM ini terbukti krusial untuk mencegah antarmuka mengalami kemacetan saat merender visualisasi dari ribuan entitas data secara dinamis (Singh et al., 2024). Proses ekstraksi laporan hasil benchmarking kemudian diotomatisasi secara penuh melalui implementasi pustaka jsPDF dan jspdf-autotable. Adapun rincian spesifikasi perangkat keras dan perangkat lunak yang digunakan sebagai standar instrumen pengujian ini diuraikan secara lengkap pada Tabel 1.

Komponen Sistem	Detail Spesifikasi
Unit Pemrosesan (CPU)	AMD Ryzen 5 7520U terintegrasi Radeon Graphics
Alokasi Memori (RAM)	8 GB dengan arsitektur DDR5
Sistem Operasi	Windows 11 (Arsitektur 64-bit)
Lingkungan Eksekusi	Google Chrome termutakhir (*V8 JavaScript Engine*)
Arsitektur Front-End	Pustaka React.js (*Virtual DOM*)
Pustaka Ekstraksi Data	jsPDF dan jspdf-autotable

Tabel 1. Spesifikasi Lingkungan Pengujian Eksperimental

Objek dan Skala Data Penelitian

Objek fundamental dalam penelitian ini adalah entitas himpunan data simulasi (dummy data) yang merepresentasikan nilai akademik mahasiswa, memuat atribut Nomor Induk Mahasiswa (NIM), Nama Mahasiswa, dan Nilai dengan rentang probabilitas 0 hingga 100. Guna memitigasi bias susunan pra-pengurutan (pre-sorted bias) yang dapat menguntungkan metrik kecepatan secara artifisial, sistem dikonstruksi dengan modul *Data Generator* untuk membangkitkan himpunan data terdistribusi seragam secara acak (Sabah et al., 2023). Adapun stratifikasi pengujian komputasional ini dibagi ke dalam tiga

variasi dimensi volume yang dirancang secara spesifik untuk menguji batasan algoritma, sebagaimana diuraikan pada Tabel 2.

Skala Data	Jumlah Elemen	Tujuan Observasi dan Evaluasi
Mikro	50	Observasi presisi Langkah manipulasi data dan tahapan animasi algoritmik secara visual.
Menengah	500	Mengidentifikasi tren batas ambang (*threshold*) waktu komputasi seiring ekskalasi beban.
Makro	1000	Mengevaluasi stabilitas manajemen memori algoritma pada cenario kondisi terburuk (*worst-case*).

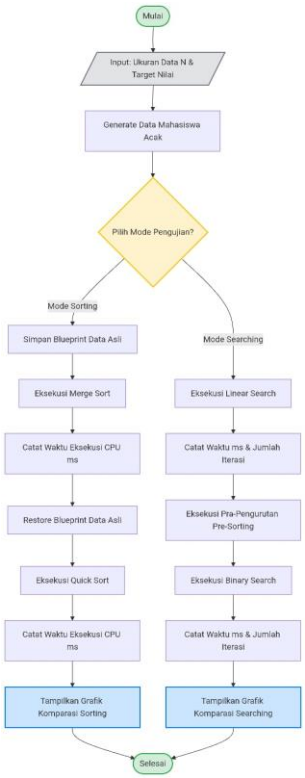
Tabel 2. Stratifikasi Dimensi Volume Data Pengujian

Desain Arsitektur dan Implementasi Algoritma

Keempat arsitektur algoritma klasik diimplementasikan ke dalam fungsi JavaScript murni (vanilla) yang diisolasi, sehingga kalkulasi komputasionalnya tidak terdistorsi oleh manipulasi *Document Object Model* (DOM) pada peramban (Saputra et al., 2025). *Merge Sort* direkayasa menggunakan metode rekursif yang secara persisten memecah himpunan larik tepat di titik ekuilibrium ($indeks\ m = \frac{l+r}{2}$), mengkomparasikan nilai, dan menyatukannya ke dalam alokasi larik sementara untuk menjamin batas waktu konstan $O(n\ log\ n)$ (Paira et al., 2014). Sebagai antitesis, Quick Sort diimplementasikan menggunakan teknik partisi in-place dengan metode Lomuto Partition. Penentuan pivot dilakukan pada elemen terakhir (indeks high), sedangkan proses pertukaran data dilakukan menggunakan dua penunjuk indeks untuk meminimalkan penggunaan memori tambahan (Lubis et al., 2022). Pada subsistem pencarian, Linear Search dikonstruksi melalui perulangan logis yang menyisir indeks 0 hingga $n - 1$ dengan sisipan variabel perekam jumlah komparasi aktual (Homepage et al., 2025). Sementara itu, *Binary Search* diimplementasikan melalui pendekatan iteratif yang secara konstan mereduksi himpunan pencarian menjadi separuh ukuran berdasarkan kalkulasi indeks pembatas *low*, *high*, dan mid (Kurhade & Takawale, 2024). Untuk menjaga konsistensi hasil pengujian, setiap algoritma dijalankan sebanyak 10 kali pada masing-masing skala data, kemudian dihitung nilai rata-rata waktu eksekusinya. Sebelum proses benchmarking dilakukan, sistem terlebih dahulu menjalankan fase warm-up guna mengurangi bias optimasi Just-In-Time (JIT) Compiler pada JavaScript V8 Engine. Pengukuran waktu eksekusi dilakukan

menggunakan fungsi performance.now() untuk memperoleh hasil pengukuran dengan resolusi tinggi.

Skenario Pengujian (Apple-to-Apple Comparison)



Gambar 1. Diagram Alir Pemrosesan Algoritma Pengurutan dan Pencarian

Sistem eksperimental ini dikonfigurasi dengan dua protokol rekayasa perangkat lunak tingkat tinggi guna menjamin komparasi efisiensi yang absolut. Protokol pertama adalah Sistem *Blueprint Data* (Isolasi Modifikasi Referensi); mengingat karakteristik ekosistem JavaScript yang memodifikasi data berbasis rujukan memori (pass-by-reference), sistem menyimpan salinan cetak biru murni di latar belakang (Bakare et al., 2024). Sistem diinstruksikan secara otomatis untuk memulihkan (restore) susunan larik utama ke bentuk cetak biru acak semula sebelum algoritma pembanding dieksekusi, sehingga seluruh algoritma dipastikan beroperasi dari garis awal yang 100% identik. Protokol kedua bertumpu pada Sistem Mode Instan (Bypass DOM Bottleneck). Untuk mengatasi latensi ratusan milidetik akibat transisi visual penukaran elemen pada data berskala makro, sistem secara dinamis menonaktifkan rendering animasi (skip animation). Algoritma diproses secara murni di latar belakang (headless execution) guna

mengekstraksi metrik waktu unit sentral (CPU) yang absolut dan bebas distorsi (Mohammadagha, 2025).

Metode Pengumpulan dan Kriteria Analisis Data

Prosedur akuisisi data waktu *benchmarking* direkam secara sinkron menggunakan High Resolution Time API bawaan peramban melalui metode performance.now() (Pujiono et al., 2025). Metrik waktu eksekusi dihitung dengan menetapkan penanda waktu awal (T_{awal}) sesaat sebelum algoritma dijalankan dan penanda waktu akhir (T_{akhir}) Selisih antara kedua penanda tersebut dihitung menggunakan persamaan $\Delta T = T_{akhir} - T_{awal}$, digunakan sebagai indikator waktu eksekusi murni algoritma dalam satuan milidetik (ms). Pengukuran dilakukan menggunakan fungsi performance.now() karena memiliki tingkat presisi tinggi dan mampu meminimalkan distorsi waktu dari proses sistem operasi maupun browser.Untuk meningkatkan validitas hasil benchmarking, setiap pengujian dilakukan sebanyak 10 kali pada dataset yang sama, kemudian dihitung nilai rata-rata waktu eksekusinya. Analisis data dilakukan dengan metode komparatif deskriptif yang mengevaluasi selisih kecepatan waktu beban CPU, mengkomparasi akumulasi jumlah langkah operasional pada pencarian nilai target, serta mencocokkan tren peningkatan asimtotik empiris di lapangan terhadap fungsi batas kurva teoretis notasi *Big-O* (Purnomo & Putra, 2023). Selain pengukuran waktu eksekusi, validasi hasil pengujian juga dilakukan dengan memverifikasi ketepatan hasil sorting dan searching. Validasi sorting dilakukan dengan memastikan data tersusun secara ascending atau descending sesuai instruksi sistem, sedangkan validasi searching dilakukan dengan mencocokkan indeks hasil pencarian terhadap posisi aktual data target. Parameter evaluasi untuk menilai supremasi masing-masing arsitektur algoritma didasarkan pada empat pilar yang dirangkum pada Tabel 3.

Aspek	Kriteria
Waktu Eksekusi	Kecepatan waktu pemrosesan algoritma yang terekam menggunakan API performance.now() (diukur secara absolut dalam rentang satuan milidetik / ms).
Kompleksitas Teoretis	Tingkat efisiensi logaritmik dan polinomialberdasarkan deret hukum teoretis *Big-O* dari setiap arsitektur algoritma yang diuji.
Efisiensi Iterasi	Jumlah akumulasi siklus perulangan (komparasi operasional) yang mutlak dilakukan oleh algoritma pencarian untuk menemukan atau memverifikasi letak nilai referensi.

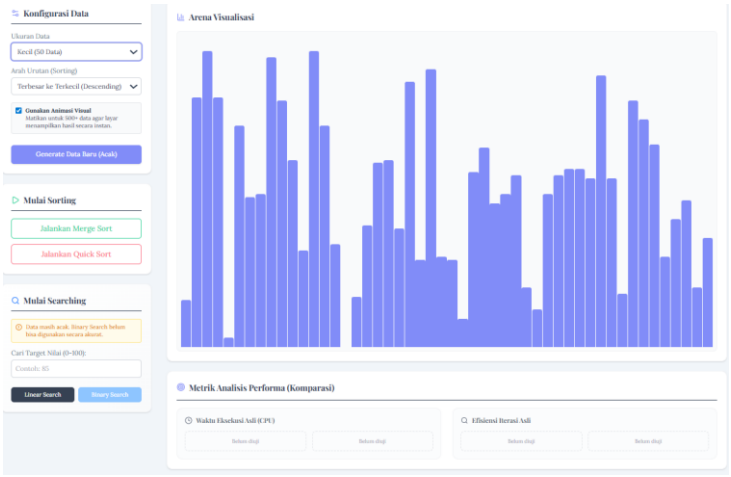
Akurasi Validasi	Derajat keberhasilan dan ketepatan algoritma dalam menuntaskan instruksi pengurutan (pembentukan hierarki baris *Ascending* / *Descending*) dan pencarian data (Ditemukan / Tidak).

Tabel 3. Kriteria Evaluasi Performa Algoritma

4. HASIL DAN PEMBAHASAN

Data Pengujian dan Antarmuka Sistem

Eksperimen pada penelitian ini menggunakan himpunan dataset simulasi yang dibangkitkan secara dinamis melalui modul Data Generator. Dataset ini merepresentasikan entitas akademik mahasiswa dengan atribut komprehensif berupa Nama, NIM, dan Nilai. Seluruh tahapan simulasi dieksekusi melalui antarmuka berbasis React.js yang responsif. Rancangan antarmuka untuk konfigurasi awal parameter dataset ini ditampilkan pada Gambar 2.

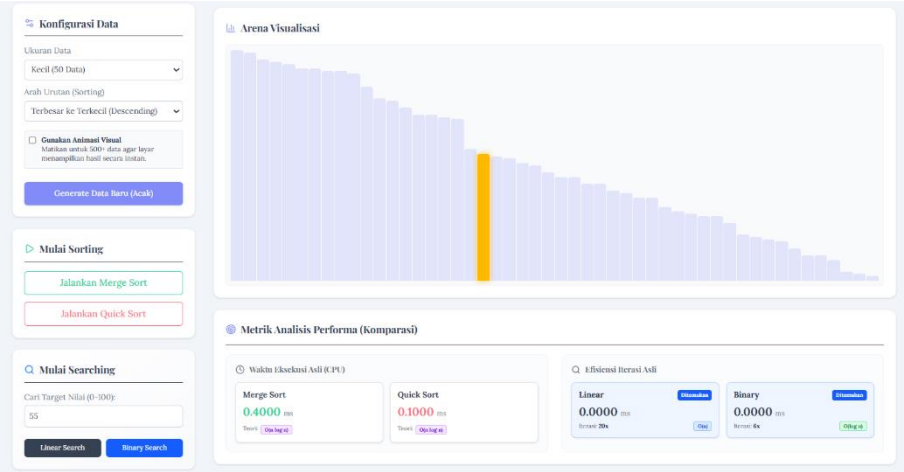


Gambar 2. Antarmuka Konfigurasi Dataset Mahasiswa (Setup Awal)

Implementasi front-end menggunakan pustaka React.js terbukti esensial dalam menyajikan data visual secara responsif. Pemanfaatan Virtual DOM memungkinkan sistem memanipulasi ratusan hingga ribuan entitas data visual secara simultan tanpa memicu degradasi stabilitas pada peramban web.

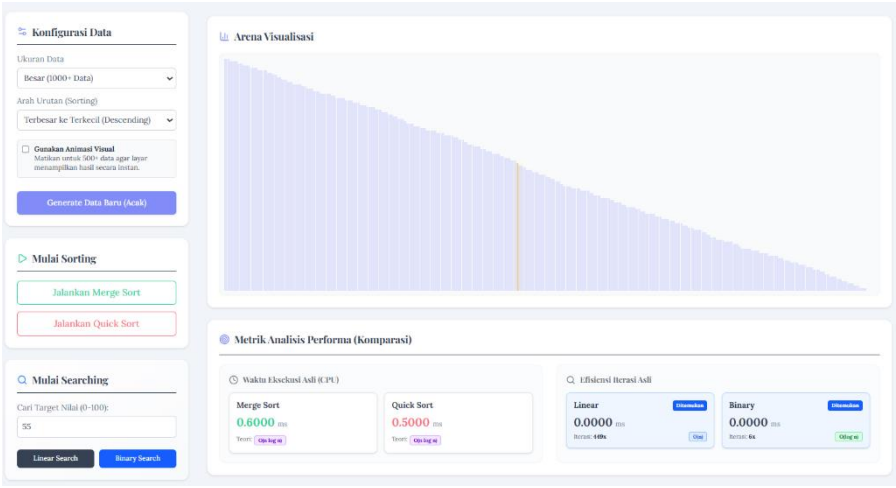
Hasil Pengujian Algoritma Sorting

Proses pengurutan data (sorting) dilakukan untuk membandingkan efisiensi dua algoritma fundamental yang berbasis pada paradigma Divide and Conquer, yaitu Merge



Gambar 3. Hasil Komparasi Algoritma Menggunakan 50 Data

Sort dan Quick Sort. Parameter utama yang diukur dalam pengujian ini adalah waktu eksekusi murni pemrosesan unit sentral (CPU) yang diukur dalam satuan milidetik (ms). Untuk menguji ketahanan dan kelemahan asimtotik dari kedua algoritma, skenario pengujian dirancang dengan mengondisikan susunan awal data pada skala menengah (500 elemen) dan skala makro (1000 elemen) dalam keadaan Telah Terurut Naik (Ascending). Pengujian data yang sudah terurut ini merupakan standar pengujian stress-test dalam rekayasa algoritma untuk menemukan batas terburuk kinerja komputasi.



Gambar 4. Hasil Komparasi Algoritma Menggunakan 500 Data

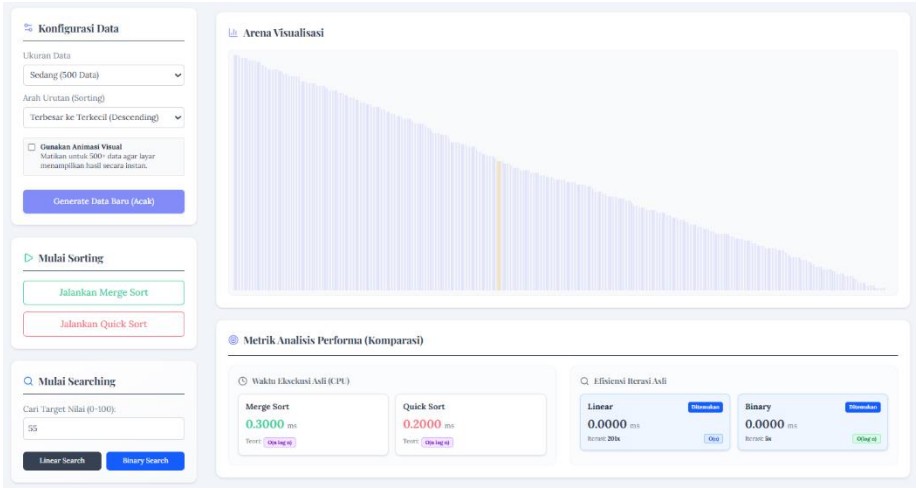
Gambar 5. Hasil Komparasi Algoritma Menggunakan 1000 Data

Jumlah Data	Merge Sort (ms)	Quick Sort (ms)
50	0,4000 ms	0,1000 ms
500	0,3000 ms	0,2000 ms
1000	0,6000 ms	0,3000 ms

Tabel 4 Perbandingan Waktu Eksekusi Algoritma Sorting

Berdasarkan Tabel 4, diperoleh serangkaian temuan empiris yang menunjukkan superioritas performa algoritma tertentu di dalam ekosistem eksekusi peramban web (JavaScript V8 Engine).

Berdasarkan metrik pengujian pada tabel tersebut, data empiris mendemonstrasikan



superioritas kecepatan arsitektur in-place milik Quick Sort. Sebaliknya, Merge Sort mencatatkan waktu eksekusi yang secara konsisten lebih lambat (0,4 ms, 0,3 ms, dan 0,6 ms). Keterlambatan ini bukan diakibatkan oleh rasio komparasi logis, melainkan terhambat oleh overhead alokasi memori. Pada setiap pemanggilan fungsi, Merge Sort diwajibkan mengalokasikan ruang larik (array) sementara. Dalam ekosistem bahasa pemrograman tingkat tinggi seperti JavaScript, proses permintaan dan pembuangan memori yang repetitif ini memicu hiperaktivitas sistem pembersih memori otomatis (Garbage Collector). Beban alokasi spasial inilah yang menciptakan bottleneck utama, sehingga Merge Sort tertinggal dari Quick Sort yang beroperasi tanpa konsumsi memori bayangan (Saputra et al., 2025).

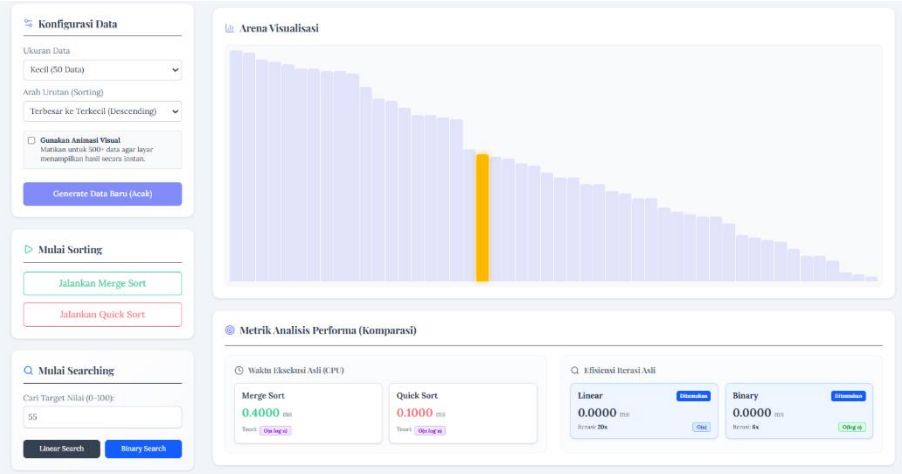
Lebih lanjut, rekaman waktu tersebut menyingkap sebuah anomali komputasional yang sangat khas: waktu eksekusi Merge Sort pada beban 500 data (0,3000 ms) justru terekam lebih cepat dibandingkan beban 50 data (0,4000 ms). Dalam penalaran matematis

murni, ekskalasi data semestinya memicu peningkatan durasi. Namun, anomali ini merupakan manifestasi dari intervensi mekanisme Just-In-Time (JIT) Compiler pada mesin V8 JavaScript. Saat eksekusi beban kecil berjalan (fase warm-up), kompilator masih menerjemahkan kode mentah. Ketika eksekusi beban 500 data dilanjutkan, mesin telah mengenali pola perulangan algoritmik dan secara otomatis mengaktifkan optimasi TurboFan tingkat tinggi di memori internal (Pratam, 2024). Fenomena empiris ini memvalidasi bahwa pengujian algoritma di platform web modern tidak semata-mata didikte oleh postulat teoretis Big-O, melainkan terikat erat pada kecerdasan arsitektur mesin kompilatornya.

Selain perbedaan waktu eksekusi, hasil pengujian menunjukkan bahwa Quick Sort memiliki efisiensi memori yang lebih baik dibandingkan Merge Sort karena menggunakan mekanisme in-place sorting. Hal ini menyebabkan konsumsi memori tambahan pada Quick Sort relatif kecil dibandingkan Merge Sort yang membutuhkan array sementara selama proses merging. Namun demikian, Merge Sort menunjukkan kestabilan performa yang lebih konsisten pada seluruh kondisi data karena kompleksitas waktunya tetap berada pada *O(n log n)*, sedangkan Quick Sort berpotensi mengalami penurunan performa hingga $O(n^2)$ pada kondisi pivot yang tidak optimal.

Hasil Pengujian Algoritma Searching

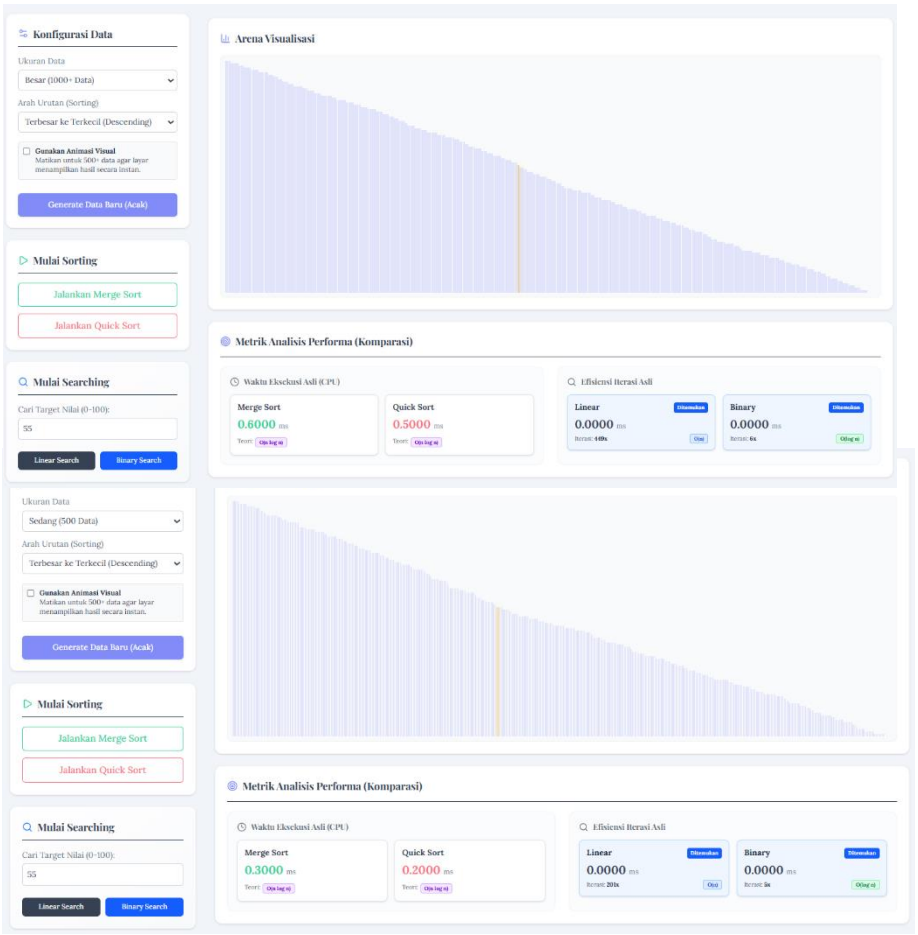
Pengujian pencarian data (searching) dilakukan untuk membandingkan tingkat efisiensi antara metode pencarian sekuensial (Linear Search) dan metode pencarian interval (Binary Search). Pada pengujian ini, digunakan tiga skala volume data (50, 500, dan 1000 elemen). Karena pencarian pada ruang memori internal sering kali berlangsung



terlalu cepat, pengukuran jumlah iterasi (langkah komparasi) digunakan sebagai metrik validasi utama mendampingi waktu eksekusi CPU (dalam ms).

Gambar 6. Hasil Komparasi Algoritma Menggunakan 50 Data

Gambar 7. Hasil Komparasi Algoritma Menggunakan 500 Data



Gambar *8.* Hasil Komparasi Algoritma Menggunakan 1000 Data

Jumlah Data	Linear Search		Binary Search	
	Waktu (ms)	Jumlah Iterasi	Waktu (ms)	Jumlah Iterasi
50	0,0000	20 kali	0,0000	6 kali
500	0,0000	201 kali	0,0000	5 kali
1000	0,0000	449 kali	0,0000	6 kali

Tabel 5. Perbandingan Waktu Eksekusi dan Iterasi Algoritma Searching

Berdasarkan Tabel 5, diperoleh pembuktian empiris mengenai perbedaan fundamental cara kerja algoritma linier dan logaritmik, sekaligus memperlihatkan fenomena resolusi waktu pada mesin eksekusi web.

Sorotan utama pada tabel komparasi pencarian adalah rekaman waktu eksekusi sub-milidetik ($0{,}0000\ ms$) pada seluruh skala data. Hal ini tidak berarti waktu pemrosesan

adalah nol detik, melainkan eksekusi berlangsung sangat cepat ($< 0{,}1\,ms$) yang dibulatkan ke bawah oleh fungsi performance.now() pada V8 Engine. Mekanisme ini merupakan mitigasi keamanan bawaan peramban terhadap eksploitasi serangan siber berbasis waktu (timing attacks seperti Spectre/Meltdown). Akibat limitasi sekuritas ini, besaran iterasi diplot sebagai indikator validasi performa algoritmik yang paling objektif. Berdasarkan analisis iterasi, Linear Search secara empiris memvalidasi skenario rata-rata (Average-Case Scenario) dari kompleksitas $O(n)$ dengan mencatatkan 20, 201, dan 449 komparasi untuk himpunan 50, 500, dan 1000 entitas. Sesuai dengan postulat teoretis $\frac{l+n}{2}$, angka tersebut mengonfirmasi bahwa target ditemukan sedikit sebelum titik tengah himpunan. Eskalasi linier ini memendam risiko sistemik; apabila basis data diekspansi hingga 1.000.000 entitas, akumulasi 500.000 iterasi seketika akan memaksa antarmuka peramban mengalami kelumpuhan sementara.

Sebaliknya, arsitektur Binary Search mendemonstrasikan efisiensi reduktif yang ekstrem. Meskipun volume data dilipatgandakan hingga 20 kali lipat, beban langkah komputasi secara stabil terkunci pada kisaran 5 hingga 6 iterasi. Pencapaian konstan ini menjadi purwarupa empiris dari supremasi logaritmik $O(log\, n)$, di mana batas maksimum pencarian untuk 1000 data secara teoretis hanyalah≈ 10langkah ($2^{10} = 1024$). Fakta ini mendeklarasikan secara absolut bahwa metode pencarian logaritmik adalah fondasi mutlak untuk menjamin skalabilitas fungsional pada sistem informasi berskala masif.

Hasil pengujian menunjukkan bahwa Binary Search mampu mempertahankan jumlah iterasi yang sangat kecil meskipun volume data meningkat secara signifikan. Fenomena ini membuktikan bahwa algoritma berbasis logaritmik lebih cocok digunakan pada sistem informasi akademik berskala besar. Sebaliknya, Linear Search menunjukkan peningkatan jumlah iterasi secara linear seiring bertambahnya jumlah data. Jika diterapkan pada sistem akademik dengan jutaan data mahasiswa, pendekatan ini berpotensi menyebabkan penurunan responsivitas aplikasi.

Komparasi Sistem

Analisis empiris menegaskan bahwa evaluasi algoritma mutlak terikat pada platform eksekusinya. Ilusi kecepatan 0,0000 ms pada mesin JavaScript modern tidak boleh memvalidasi penggunaan Linear Search, mengingat adanya ancaman eskalasi iterasi sistemik yang tersembunyi (mencapai 449 komparasi untuk 1000 data). Sebagai konklusi,

arsitektur data paling tangguh untuk platform akademik web adalah pewajiban pra-pengurutan (pre-sorting) berbasis Quick Sort atau Merge Sort. Langkah ini mutlak diperlukan untuk mengeksploitasi efisiensi ekstrem dari Binary Search, yang secara konsisten mampu mereduksi pencarian data masif menjadi sekadar 5 hingga 6 unit langkah operasional.

5. KESIMPULAN DAN SARAN

Penelitian ini berhasil membandingkan performa algoritma Merge Sort, Quick Sort, Linear Search, dan Binary Search pada pengolahan data nilai mahasiswa berbasis web. Hasil pengujian menunjukkan bahwa Quick Sort memiliki performa waktu yang lebih cepat dibandingkan Merge Sort karena menggunakan mekanisme in-place sorting yang lebih efisien terhadap penggunaan memori.

Sementara itu, Binary Search terbukti jauh lebih efisien dibandingkan Linear Search dalam proses pencarian data karena mampu mempertahankan jumlah iterasi yang rendah meskipun volume data meningkat secara signifikan.

Penelitian ini juga menemukan bahwa lingkungan eksekusi JavaScript modern, khususnya mekanisme Just-In-Time Compiler pada V8 Engine, memberikan pengaruh terhadap hasil benchmarking algoritma. Oleh karena itu, pengujian performa algoritma pada platform web modern perlu mempertimbangkan faktor optimasi mesin eksekusi.

Secara keseluruhan, kombinasi Quick Sort dan Binary Search direkomendasikan sebagai strategi optimal dalam pengembangan sistem informasi akademik berbasis web yang membutuhkan kecepatan akses data dan skalabilitas tinggi.

DAFTAR REFERENSI

Aaron Afan, I., Halimah, A. D., Chukwudi John, H., Oluwajenyo, O. B., Praise, W. M., Aaron, I. A., & Henry, J. C. (2022). *Web-Based Grade Prediction System*. *2023*(1), 20–30. Retrieved from https://hal.science/hal-04270856v1

AL-Azzam, S., & Qatawneh, M. (2018). Parallel Processing of Sorting and Searching Algorithms Comparative Study. *Modern Applied Science*, *12*(4), 143. https://doi.org/10.5539/mas.v12n4p143

Alboaneen, D., Almelihi, M., Alsubaie, R., Alghamdi, R., Alshehri, L., & Alharthi, R. (2022). Development of a Web-Based Prediction System for Students' Academic Performance. *Data*, *7*(2). https://doi.org/10.3390/data7020021

Ayazuddin, R. (2025, September 30). *A Comprehensive Study of Sorting Algorithm Performance Using Real-World Dataset Metrics*. https://doi.org/10.20944/preprints202509.2550.v1

Bakare, K. A., Okewu, A. A., Abiola, Z. A., Jaji, A., & Muhammed, A. (2024). A COMPARATIVE STUDY OF SORTING ALGORITHMS: EFFICIENCY AND

PERFORMANCE IN NIGERIAN DATA SYSTEMS. *FUDMA JOURNAL OF SCIENCES*, *8*(5), 1–5. https://doi.org/10.33003/fjs-2024-0805-2730

Capannini, G., Silvestri, F., & Baraglia, R. (2012). Sorting on GPUs for large scale datasets: A thorough comparison. *Information Processing and Management*, *48*(5), 903–917. https://doi.org/10.1016/j.ipm.2010.11.010

Christnatalis, H. S., Yennimar, Y., Prabowo, A., Manday, D. R., Simarmata, A. M., Sofhia, M., & Fauzi, A. (2026). *The Timeless Power of Divide and Conquer in Algorithm Design*. https://doi.org/10.2991/978-94-6463-998-8_7

Dubey, S., & Mathur, K. (2017). Comparative Performance Analysis of Binary Search in Sequential and Parallel Processing. In *International Journal for Research in Applied Science & Engineering Technology (IJRASET)* (Vol. 887). Retrieved from www.ijraset.com

Esau Taiwo, O., Christianah, A. O., Oluwatobi, A. N., Aderonke, K. A., & Kehinde, A. J. (2020). COMPARATIVE STUDY OF TWO DIVIDE AND CONQUER SORTING ALGORITHMS: QUICKSORT AND MERGESORT. *Procedia Computer Science*, *171*, 2532–2540. https://doi.org/10.1016/J.PROCS.2020.04.274

Homepage, J., Tri Ridha Ridwan, W., & Alam, S. (2025). Implementasi Algoritma Sequential Search dalam Pencarian Data Cuti pada Aplikasi Cuti Pegawai. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, *5*(1), 198–206. https://doi.org/10.57152/malcom.v5i1.1735

Huda, N., Rasyid Munthe, I., Putra Juledi, A., Sains dan Teknologi, F., Informasi Unversitas Labuhan Batu, S., Email Penulis Korespondensi, I., … Sumber Daya, A. (2024). Optimisasi Manajemen Sumber Daya pada Sistem Operasi C untuk Lingkungan Cloud Computing. *Jurnal Ilmu Komputer Dan Sistem Informasi (JIKOMSI)*, *7*(1), 96–99. https://doi.org/10.55338/jikomsi.v7i1.2721

Kodua Wiredu, J., Aabaah, I., Wiredu Acheampong, R., & Aabaah, I. (n.d.). Duplicate-Heavy Data Sets. *International Journal of Advanced Research in Computer Science*, *2024*(6), 12–18. https://doi.org/10.26483/ijarcs.v15i6.7152ï

Kurhade, Mrs. A., & Takawale, Ms. N. N. (2024). Comparative Study of Searching Algorithm: Linear Search and Binary search. *IJARCCE*, *13*(5). https://doi.org/10.17148/ijarcce.2024.13506

Liu, P. (2024). An in-depth study of sorting algorithms. *Applied and Computational Engineering*, *92*(1), 187–195. https://doi.org/10.54254/2755-2721/92/20241750

Mezied, A. A., & Abu-Naser, S. S. (2025). The Future of Data Sorting: Integrating AI for Enhanced Efficiency and Accuracy. In *International Journal of Academic Engineering Research* (Vol. 9). Retrieved from www.ijeais.org/ijaer

Mishal, I., Al-Khatib, R., & Hiasat, R. (2021). Comparative Study of Two Divide and Conquer Sorting Algorithms: Modified Quick Sort and Merge Sort. In *International Journal of Computer Applications* (Vol. 183).

Muis Mappalotteng, A., Fathahillah, F., & Anas Punggawa, M. (2024). Web-Based Student Academic Grade Processing Information System. *ITM Web of Conferences*, *58*, 03006. https://doi.org/10.1051/itmconf/20245803006

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., Rachmawanto, E. H., & Winarsih, N. A. S. (2025). Array Sorting Algorithm vs Traditional Sorting Algorithm: Memory and Time Efficiency

Analysis. *Jurnal Manajemen Informatika (JAMIKA)*, *15*(1), 47–59. https://doi.org/10.34010/jamika.v15i1.13230

Purba, R., & Rachmadi, P. (2022). Web-Based ManagementSystem for Student ScorePassing Grades. *International Journal of Advances in Scientific Research and Engineering*, *08*(06), 59–64. https://doi.org/10.31695/ijasre.2022.8.6.6

Purnama, N. (2025). *COMPARATIVE PERFORMANCE STUDY OF SEARCH ALGORITHMS ON LARGE-SCALE DATA STRUCTURES*. *11*(1). https://doi.org/10.33480/jitk.v11i1.6592

Purnomo, R., & Putra, T. D. (2023). Theoretical Analysis of Standard Selection Sort Algorithm. *SinkrOn*, *8*(2), 666–673. https://doi.org/10.33395/sinkron.v8i2.12153

Roșca, C. M., & Cărbureanu, M. (2025). A Comparative Analysis of Sorting Algorithms for Large-Scale Data: Performance Metrics and Language Efficiency. *Lecture Notes in Networks and Systems*, *1073*, 99–113. https://doi.org/10.1007/978-981-97-5703-9_8

Sabah, A. S., Abu-Naser, S. S., Emad Helles, Y., Fikri Abdallatif, R., Abu Samra, F. Y., Helmi Abu Taha, A., … Hamouda, A. A. (2023). Exponential (10K). In *International Journal of Academic Engineering Research* (Vol. 7). Retrieved from www.ijeais.org/ijaer

Saputra, M. S. A., Meidiansyah, M. R., Sanputra, A. D., & Wardana, J. (2025). Comparative Analysis of Searching and Sorting Algorithms in Student Data Processing. *International Journal of Education, Information Technology, and Others*, *8*(3.B), 148–157. Retrieved from http://www.jurnal.peneliti.net/index.php/IJEIT/article/view/12718

Shorman, A., Rateb, R., Alshorman, A., & Shqair, S. A. (n.d.). International Journal of INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING Performance Analysis of Quicksort Algorithm: An Experimental Study of Its Variants. In *Original Research Paper International Journal of Intelligent Systems and Applications in Engineering IJISAE* (Vol. 2024). Retrieved from www.ijisae.org

Singh, S., Singh, S., Singh, V., & Hazela, B. (2024). Sorting Visualizer: A Visual Journey Through Sorting Algorithms. *Journal of In-Formatics Electrical and Electronics Engineering*, *05*(107), 1–9. https://doi.org/10.54060/a2zjourna

Suleiman Al-Kharabsheh, K., Mahmoud AlTurani, I., Mahmoud Ibrahim AlTurani, A., & Imhammed Zanoon, N. (2013). Review on Sorting Algorithms A Comparative Study. In *International Journal of Computer Science and Security (IJCSS)*.

Ukuran dan Kondisi Keterurutan Data, B., Abdul Rahman, G., Indra Junaedi, D., & Santika, D. (2025). Algoritma Linear Search dan Binary Search. *Jurnal Ilmu-Ilmu Informatika Dan Manajemen*, *19*(2), 45323. https://doi.org/10.33481/infomans.vxxixx