ANALISIS KOMPARATIF EFISIENSI WAKTU DAN MEMORI: LINEAR, BINARY, DAN *HASH SEARCH* BERBASIS C++

Ikhsan Maulanaa[1)], Firda Rona Syahira[2)], Meila Fitri Amin[3)], Nevi Dwi Apriyanti[4)], Imam Prayogo Pujiono[5)]

[1,2,3,4]Prodi Bisnis Digital, Ekonomi dan Bisnis Islam, UIN K.H. Abdurrahman Wahid Pekalongan
[5]Prodi Informatika, Ekonomi dan Bisnis Islam, UIN K.H Abdurrahman Wahid Pekalongan

Correspondence author: FR Syahira, firda.rona.syahira25006@mhs.uingusdur.ac.id, Pekalongan, Indonesia

Abstract

This study aims to analyze the comparative memory efficiency and computation time of three commonly used data search algorithms in C++ programming: Linear Search, Binary Search, and Hash Search. This study analyzes the performance of each algorithm based on two main aspects: execution speed and memory usage, to determine the most appropriate algorithm for various data conditions. The research method used was a quantitative comparative experiment, where each algorithm was implemented in C++ and tested on three data size scenarios (100, 1000, and 10,000 elements). The results obtained indicate that Binary Search has the best balance between speed and memory efficiency. At the same time, Linear Search is more suitable for small data sizes due to its simplicity of implementation. Conversely, Hash Search has high search speed but requires more memory. Therefore, the selection of the appropriate search algorithm should be tailored to the characteristics of the data and system requirements. This study is expected to serve as a reference for software developers in selecting the optimal search algorithm based on computational resource efficiency.

Keyword *: algorithm comparison, data search, memory efficiency, computation time*

Abstrak

Penelitian ini bertujuan untuk menganalisis perbandingan efisiensi memori dan waktu komputasi pada tiga algoritma pencarian data yang umum digunakan dalam pemrograman C++, yaitu *Linear Search*, *Binary Search*, dan *Hash Search*. Penelitian ini menganalisis kemampuan kerja pada masing-masing algoritma berdasarkan dua aspek utama, yakni kecepatan eksekusi dan penggunaan memori, agar dapat menentukan algoritma yang paling sesuai untuk berbagai kondisi data. Metode penelitian yang digunakan adalah eksperimen komparatif kuantitatif, di mana setiap algoritma diimplementasikan menggunakan bahasa C++ dan diuji pada tiga skenario ukuran data (100, 1000, dan 10000 elemen). Hasil yang diperoleh menunjukkan bahwa *Binary Search* memiliki kinerja paling seimbang antara kecepatan dan efisiensi memori, sedangkan *Linear Search* lebih cocok untuk data yang ukuranya kecil karena kesederhanaan implementasinya.

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

Sebaliknya, *Hash Search* memiliki kecepatan pencarian yang tinggi, tetapi membutuhkan memori lebih besar. Dengan demikian, pemilihan algoritma pencarian yang tepat sebaiknya disesuaikan dengan karakteristik data dan kebutuhan sistem. Penelitian ini diharapkan dapat menjadi acuan bagi pengembang perangkat lunak dalam memilih algoritma pencarian yang optimal berdasarkan efisiensi sumber daya komputasi.

Kata Kunci : perbandingan algoritma, pencarian data, efisiensi memori, waktu komputasi

A. PENDAHULUAN

Pencarian data merupakan salah satu operasi fundamental dalam bidang ilmu komputer yang memiliki peran penting dalam proses pengolahan informasi (Budiansyah et al., 2025). Hampir seluruh sistem yang mengelola volume data besar seperti *database management system*, *search engine*, maupun aplikasi analitik melibatkan proses pencarian untuk menemukan elemen tertentu dari sekumpulan data (Akhsa et al., 2024). Efisiensi proses pencarian ini berpengaruh langsung terhadap kinerja sistem secara keseluruhan, terutama pada aplikasi *real-time* yang membutuhkan respons cepat serta penggunaan sumber daya yang optimal (Suryadi et al., 2024).

Dalam praktik pemrograman modern, khususnya dengan menggunakan bahasa C++, pemilihan algoritma pencarian yang tepat menjadi aspek penting yang menentukan performa program (Purbasari et al., 2024). C++ dikenal sebagai bahasa yang efisien karena memberikan kontrol tinggi terhadap penggunaan memori dan waktu eksekusi (R. A. Putra, 2024). Melalui dukungan berbagai struktur data seperti *array*, *vector*, dan *hash table*, pengembang memiliki fleksibilitas tinggi untuk menentukan metode pencarian yang paling sesuai dengan karakteristik data (Erkamim et al., 2024).

Di antara berbagai metode yang tersedia, tiga algoritma pencarian yang umum digunakan dan sering dibandingkan dari segi efisiensi adalah *Linear Search*, *Binary Search*, dan *Hash Search* (Situmorang, 2017). Ketiganya memiliki perbedaan mendasar dalam prinsip kerja, kompleksitas waktu, serta kebutuhan memori (Firmansyah et al., 2025). *Linear Search* dikenal dengan implementasinya yang sederhana karena melakukan pencarian dengan membandingkan setiap elemen secara berurutan hingga data ditemukan (Leana et al., 2025). Meskipun efektif untuk data berukuran kecil dan tidak terurut, algoritma ini memiliki keterbatasan dalam skala besar karena kompleksitas waktunya bersifat linear O(n) (Izhari, 2025).

Sebaliknya, *Binary Search* menggunakan pendekatan *divide and conquer* dengan kompleksitas waktu rata-rata O(log n), sehingga lebih efisien untuk data besar yang sudah terurut (Ariza et al., 2025). Namun, kebutuhan proses pengurutan awal menjadi kelemahan tersendiri ketika data sering berubah atau bersifat dinamis (Mevia et al., 2026). Di sisi lain, *Hash Search* mampu memberikan waktu pencarian yang hampir konstan O(1) pada kondisi ideal dengan memanfaatkan *hash table* sebagai indeks data (Bender et al., 2022). Walaupun sangat cepat, metode ini memiliki kelemahan dalam efisiensi memori karena memerlukan ruang tambahan untuk mengatasi collision antar nilai hash (Yusuf et al., 2021).

Evaluasi terhadap ketiga algoritma tersebut penting dilakukan mengingat performa aktual algoritma sering kali dipengaruhi oleh karakteristik data dan konfigurasi system (R. F. Putra et al., 2023). Secara teoritis, algoritma dengan

kompleksitas waktu lebih baik belum tentu menunjukkan performa terbaik pada implementasi nyata, terutama dalam kondisi sistem dengan keterbatasan sumber daya *(resource constraints)* (Puranik, 2025). Oleh karena itu, penelitian ini berfokus tidak hanya pada pengukuran kecepatan eksekusi, tetapi juga pada efisiensi penggunaan memori, dimana dua hal tersebut menjadi faktor utama yang saling memengaruhi dalam optimalisasi kinerja algoritma.

Sejumlah penelitian sebelumnya telah membahas aspek efisiensi algoritma pencarian dari perspektif waktu komputasi, namun kajian yang menggabungkan analisis waktu dan memori secara bersamaan masih terbatas, khususnya dengan implementasi langsung di C++ (Sitorus et al., 2024). Misalnya, penelitian oleh (Budiansyah et al., 2025) menitikberatkan pada optimalisasi pencarian pada kecerdasan buatan, sementara studi oleh (Purnama, 2025) menunjukkan bahwa *Hash Search* memiliki kinerja waktu terbaik pada dataset besar, namun belum memperhitungkan aspek penggunaan memori. Dengan demikian, penelitian ini menghadirkan nilai kebaruan *(novelty)* melalui perbandingan langsung antara efisiensi waktu dan memori dari ketiga algoritma pencarian tersebut dalam lingkungan terkontrol berbasis C++.

Selain itu, penelitian ini memiliki kontribusi praktis yang signifikan bagi pengembang perangkat lunak *(software developer)*, khususnya dalam membantu pemilihan metode pencarian yang paling tepat berdasarkan ukuran data, frekuensi pencarian, serta batasan sumber daya sistem. Hasil penelitian diharapkan dapat menjadi acuan empiris yang bermanfaat dalam pengembangan aplikasi berbasis data besar, seperti sistem *machine learning*, data mining, maupun *text retrieval system*.

Penelitian diharapkan dapat menyajikan analisis kuantitatif dan kualitatif terhadap *Linear Search*, *Binary Search*, dan *Hash Search* dalam hal efisiensi waktu serta penggunaan memori pada implementasi nyata di bahasa pemrograman C++.

B. METODE PENELITIAN

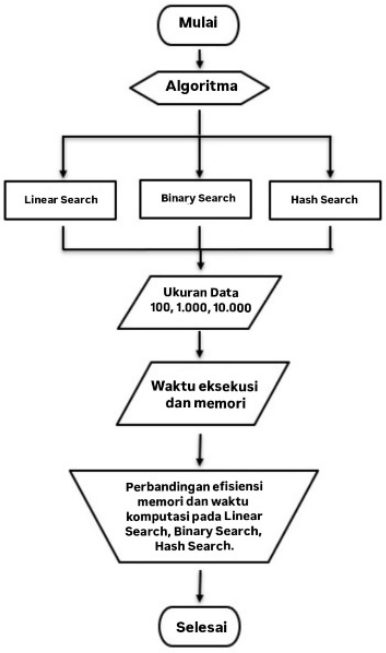
Penelitian ini menggunakan bahasa pemrograman C++ sebagai alat utama untuk menjalankan berbagai kode yang diperlukan. Bahasa pemrograman C++ dipilih karena bahasa ini dikenal karena kecepatan, efisiensi, serta kemampuannya dalam memberikan kontrol yang baik terhadap penggunaan memori, sehingga sangat sesuai untuk penelitian yang fokus pada kinerja algoritma. Seluruh pengujian dilaksanakan menggunakan sebuah laptop dengan spesifikasi : Prosesor AMD Ryzen 5 7520U, SSD 512 GB M.2, Memori RAM: 16 GB, Sistem operasi Windows 11.

Studi ini menerapkan pendekatan kuantitatif dengan metode eksperimen komparatif untuk mengevaluasi efisiensi penggunaan memori dan waktu hitung antara algoritma pencarian data (*Linear Search*, *Binary Search*, dan *Hash Search*). Metode eksperimen dipilih karena melibatkan pengujian secara langsung terhadap performa algoritma yang menghasilkan data yang dapat diukur. Penelitian ini bertujuan untuk menilai performa algoritma secara sistematis, berfokus pada dua aspek: penggunaan memori dan durasi komputasi. Untuk memperjelas proses, tahapan dalam penelitian ini terdiri dari pengumpulan data, implementasi algoritma, pengujian performa, dan analisis perbandingan.

Tahap eksperimen dimulai dengan pengumpulan data yang dibagi menjadi tiga jenis elemen : 100 elemen, 1000 elemen, dan 10000 elemen untuk menguji skalabilitas algoritma. Seluruh proses tersebut divisualisasikan menggunakan diagram alur untuk memperjelas urutan langkah dalam penelitian.

Diagram alur tersebut dapat dilihat pada gambar 1 berikut :

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767



Gambar 1. Diagram Alur Penelitian

Dengan cara ini, penelitian dapat mengamati apakah kinerja algoritma tetap stabil atau mengalami perubahan signifikan saat ukuran data bertambah. Setelah data siap, ketiga algoritma pencarian tersebut kemudian diimplementasikan memakai C++. Seluruh program ditulis dan dijalankan menggunakan *Code::Blocks*, sebuah lingkungan pengembangan terintegrasi yang sangat mendukung proses kompilasi dan *debugging*. Setelah implementasi selesai, langkah selanjutnya adalah melakukan evaluasi kinerja. Pada tahap ini, setiap algoritma diuji berulang kali dengan tiga ukuran data yang sudah disiapkan. Pengulangan ini dilakukan untuk memastikan bahwa hasil yang diperoleh konsisten dan tidak terpengaruh oleh kebetulan.

Dari setiap uji coba, dua jenis informasi dicatat, yaitu waktu pemrosesan dalam milidetik (*ms*) dan penggunaan memori dalam kilobyte (*KB*). Data-data ini kemudian akan menjadi dasar utama untuk perbandingan antar algoritma. Dengan menganalisis hasil dari ketiga algoritma tersebut, peneliti dapat menentukan algoritma mana yang paling cepat, mana yang paling hemat memori, serta bagaimana perubahan jumlah elemen mempengaruhi kinerja masing-masing program.

C. HASIL DAN PEMBAHASAN

Linear Search

Algoritma *Linear Search* direalisasikan menggunakan bahasa pemrograman C++. Algoritma *Linear Search* adalah salah satu metode paling sederhana dalam algoritma pencarian yang berfungsi dengan memeriksa setiap elemen dalam struktur data secara berurutan, mulai dari indeks awal hingga indeks akhir, sampai elemen yang dicari ditemukan atau semua elemen telah diperiksa. Implementasi algoritma ini dalam bahasa C++ di program yang dimaksud dilakukan secara langsung, di mana semua data diinput, diproses, dan dijalankan oleh program yang telah dirancang sebelumnya. Dengan model tersebut, proses pengujian menjadi lebih teratur karena semua langkah mulai dari memasukkan data, menjalankan algoritma, hingga mengukur kinerja dapat dilakukan dalam satu arus komputasi tanpa campur tangan manual.

Dalam tahap pengujian performa program, jumlah elemen data yang digunakan ditentukan dalam tiga skala yang berbeda, yaitu 100, 1.000, dan 10.000 elemen. Pemilihan variasi jumlah elemen tersebut ditujukan untuk melihat bagaimana kinerja algoritma *Linear Search* beradaptasi saat ukuran data meningkat secara drastis. Secara fundamental, *Linear Search* memiliki kompleksitas waktu O(n), yang menunjukkan bahwa waktu yang diperlukan untuk menyelesaikan proses pencarian meningkat secara linier seiring dengan bertambahnya jumlah elemen yang harus dicek. Oleh karena itu, pengujian dengan ketiga ukuran data tersebut dapat memberikan gambaran yang cukup jelas mengenai sejauh mana algoritma terpengaruh oleh peningkatan ukuran *input*.

Proses pengujian pada setiap set data dilakukan melalui prosedur yang sama, yaitu dengan menjalankan algoritma untuk mencari elemen tertentu pada posisi acak dalam daftar data. Tujuannya adalah untuk memastikan bahwa hasil pengukuran waktu komputasi yang diperoleh adalah objektif dan mencerminkan kondisi nyata. Selain itu, data yang digunakan merupakan data yang dihasilkan oleh program itu sendiri sehingga tidak ada ketergantungan pada faktor luar yang dapat mempengaruhi hasil pengujian. Dengan cara ini, semua proses pencarian berlangsung dalam lingkungan komputasi yang terkontrol dan konsisten.

Hasil pengukuran yang didapat dari pengujian ini mencakup dua aspek utama, yakni efisiensi memori dan waktu komputasi. Efisiensi memori berkaitan dengan jumlah memori yang dimanfaatkan oleh program untuk menyimpan data dan menjalankan algoritma pencarian. Karena *Linear Search* tidak memerlukan struktur data tambahan atau proses penyimpanan data sementara yang rumit. Di sisi lain, waktu komputasi menjadi aspek yang paling mencolok dalam ujian ini. Seperti yang telah diketahui, semakin banyak elemen yang harus diperiksa, semakin lama waktu yang dibutuhkan oleh algoritma *Linear Search* untuk menemukan elemen tertentu, terutama jika elemen yang dicari terletak di bagian akhir daftar atau bahkan tidak ditemukan sama sekali. Oleh karena itu, perbandingan waktu eksekusi pada ukuran data 100, 1000, dan 10000 elemen sangat penting untuk menunjukkan sejauh mana *Linear Search* dapat diandalkan ketika diterapkan dalam konteks data yang semakin besar.

Seluruh data pengukuran diperoleh dengan merekam waktu mulai dan waktu akhir pelaksanaan algoritma menggunakan fungsi bawaan dalam C++, seperti *chrono*, yang memberikan akurasi tinggi dalam pengukuran waktu. Sementara itu, pengukuran penggunaan memori dilakukan dengan memanfaatkan fungsi pemantauan memori yang sesuai dengan lingkungan pengembangan program. Data hasil pengukuran mengenai efisiensi memori dan waktu komputasi kemudian disajikan secara menyeluruh pada Tabel 1. sehingga memudahkan pembaca dalam melakukan analisis dan perbandingan antara berbagai ukuran data.

Tabel 1. Hasil uji *Linear Search*

Ukuran Data	Hasil Pengujian	
	Waktu *(/ms)*	Memori (*KB)*
100	0,3382	0,390635
1.000	0,2212	3,90625
10.000	0,4013	39,0625
Rata-rata	0,320234	14,253125

Berdasarkan Tabel 1. diketahui bahwa rata-rata kecepatan waktu eksekusi program dengan tiga elemen adalah 0,320234 *ms*. Dan rata-rata penggunaan memori dengan tiga jenis elemen adalah 14,453125 *KB*. Dengan penyajian data tersebut, pembaca bisa memahami bagaimana kinerja algoritma *Linear Search* dalam beragam skenario ukuran *input* dan dapat mengevaluasi apakah algoritma ini cocok digunakan dalam aplikasi tertentu, terutama saat berhadapan dengan kumpulan data besar yang membutuhkan efisiensi tinggi.

Binary Search

Binary Search adalah salah satu algoritma pencarian yang paling sering digunakan karena memiliki kompleksitas waktu yang jauh lebih efektif dibandingkan dengan metode pencarian *linear*. *Binary Search* hanya dapat digunakan pada data yang sudah terurut, baik itu dalam urutan menaik maupun menurun, sehingga proses pencariannya dapat dilakukan dengan cara yang teratur dan sistematis. Salah satu keunggulan dari algoritma ini adalah kemampuannya untuk mengurangi area pencarian dengan terus membagi data menjadi dua bagian sampai elemen yang dicari ditemukan.

Secara konseptual, cara kerja dari algoritma *Binary Search* diawali dengan

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

menentukan nilai tengah dari rentang indeks data yang sedang diperiksa. Jika nilai di indeks tengah tersebut sesuai dengan yang dicari, proses pencarian dapat dihentikan. Namun, jika nilai tengah lebih besar daripada target dalam data yang terurut menaik, pencarian akan dilanjutkan pada bagian kiri dari data. Sebaliknya, jika nilai tengah lebih kecil, pencarian akan beralih ke bagian kanan. Proses pembagian area pencarian ini akan terus berlangsung hingga elemen ditemukan atau sampai tidak ada lagi ruang pencarian yang tersisa. Mekanisme ini memungkinkan *Binary Search* memiliki kompleksitas waktu rata-rata O(log n), yang menjadikannya jauh lebih efisien dibandingkan *Linear Search* yang memiliki kompleksitas O(n). Dalam penelitian atau penerapan praktis, pengukuran kinerja dari suatu algoritma sangat krusial untuk menentukan seberapa efisien waktu komputasi dan penggunaan memori yang dibutuhkan. Dalam studi ini, *Binary Search* diuji dengan tiga kategori jumlah elemen data, yaitu 100, 1000, dan 10000 elemen. Ketiga ukuran data ini dipilih untuk mewakili skala kecil, menengah, dan besar, sehingga analisis perbandingan kinerjanya dapat dilakukan dengan lebih komprehensif. Data yang digunakan sudah diurutkan sebelumnya agar sesuai dengan syarat penerapan algoritma *Binary Search*.

Pengujian dilakukan melalui beberapa algoritma pemrograman yang mengikuti logika dasar dari *Binary Search* yang serupa. Setiap pengujian mengukur dua parameter utama, yaitu waktu eksekusi dan penggunaan memori. Hasil lengkap dari pengukuran bisa dilihat pada Tabel 2.

Tabel 2. Hasil uji *Binary Search*

Ukuran Data	Hasil Pengujian	
	Waktu *(/ms)*	Memori (*KB)*
100	0,0002	0,390625
1.000	0,0003	3,90625
10.000	0,0004	39,0625
Rata-rata	0,0003	14,453125

Berdasarkan tabel 2 tersebut, didapatkan fakta bahwa rata-rata kecepatan waktu eksekusi dari program dengan tiga variasi jumlah elemen adalah 0,0003 *ms*. Angka ini menunjukkan bahwa algoritma *Binary Search* memiliki kinerja yang sangat cepat meskipun pada data yang berukuran besar. Efisiensi waktu ini tidak lepas dari struktur logaritmik dalam algoritma yang mampu mengurangi jumlah langkah pencarian dengan signifikan. Selain itu, rata-rata penggunaan memori untuk tiga jenis elemen adalah 14,453125 *KB*. Nilai penggunaan memori yang relatif kecil ini menunjukkan bahwa *Binary Search* tidak membutuhkan alokasi memori tambahan yang signifikan, selain dari struktur data utama yang sudah ada. Dengan menggunakan teknik pembagian rentang indeks tanpa membuat salinan data baru, *Binary Search* dapat menjaga efisiensi memori, sehingga sangat cocok untuk digunakan dalam aplikasi yang memiliki keterbatasan ruang penyimpanan. Melalui analisis ini, dapat dirangkum bahwa *Binary Search* adalah salah satu metode pencarian yang paling efektif terkait waktu pemrosesan serta penggunaan memori. Keunggulan ini tetap terjaga meskipun jumlah data meningkat. Inilah alasan mengapa *Binary Search* sering digunakan di berbagai sektor, termasuk sistem *database*, pencarian di struktur pohon, optimisasi algoritma, serta dalam berbagai aplikasi yang memerlukan pencarian cepat pada data yang telah diurutkan. Secara keseluruhan, hasil studi ini dapat memberikan gambaran yang lebih menyeluruh tentang kinerja algoritma *Binary Search* dalam berbagai situasi ukuran data yang berbeda.

Hash Search

Hash Search merupakan metode atau algoritma pencarian yang dirancang untuk memproses volume data yang besar dengan cara yang cepat dan efisien. Secara umum, algoritma ini beroperasi dengan memanfaatkan struktur data yang dikenal sebagai *hash table*, yaitu tabel yang

menyimpan data berdasarkan indeks yang diperoleh dari perhitungan fungsi *hash*. Fungsi *hash* sendiri bertugas mengonversi nilai atau kunci data menjadi indeks tertentu, memungkinkan pencarian dilakukan secara langsung tanpa harus memeriksa data satu per satu. Dengan mekanisme ini, *Hash Search* dapat memberikan kecepatan pencarian yang jauh lebih tinggi, terutama ketika ukuran *database* sangat besar atau jumlah entri mencapai ribuan hingga jutaan.

Namun, walaupun *Hash Search* dikenal sangat efisien untuk data berskala besar, algoritma ini tidak selalu menjadi pilihan terbaik untuk *database* dengan ukuran kecil. Pada *database* yang hanya memiliki sedikit elemen, penggunaan *Hash Search* justru dapat memberikan *overhead*, yaitu tambahan beban pada proses pembuatan *hash table* itu sendiri. *Overhead* ini mencakup alokasi memori untuk struktur tabel, perhitungan fungsi *hash*, serta penanganan potensi terjadinya benturan data di indeks tertentu. Ketika data yang tersedia sangat sedikit, waktu dan sumber daya yang diperlukan untuk membuat *hash table* sering kali tidak sebanding dengan keuntungan yang diperoleh. Dengan kata lain, penyiapan struktur *hash* justru bisa memperlama waktu eksekusi karena data yang sedikit seharusnya bisa diproses dengan lebih cepat melalui metode pencarian sederhana seperti *Linear Search* atau *Binary Search.* Dalam prinsip operasionalnya, *Hash Search* memungkinkan sistem untuk langsung mengetahui lokasi data berdasarkan hasil fungsi *hash*. Inilah alasannya *Hash Search* sangat efisien untuk data yang besar, karena kompleksitas waktunya dapat mendekati O(1) atau waktu konstan dalam banyak situasi.

Tabel 3. Hasil uji *Hash Search*

Ukuran Data	Hasil Pengujian	
	Waktu *(/ms)*	Memori (*KB)*
100	4852	0,390625
1.000	5623	3,90625
10.000	12492	39,0625
Rata-rata	7655,67	14,453125

Berdasarkan data yang terdapat dalam Tabel 3. didapatkan informasi tentang rata-rata kinerja program saat menangani tiga elemen data. Dari tabel tersebut, diketahui bahwa rata-rata waktu eksekusi program untuk tiga elemen adalah 7655,67 *ms*. Angka ini menunjukkan bahwa ketika jumlah data sangat sedikit, waktu yang dibutuhkan tidak secepat yang diharapkan dari algoritma *hash*. Seperti penjelasan sebelumnya, lambatnya waktu eksekusi bukanlah akibat dari proses pencarian itu sendiri, melainkan karena tahap persiapan struktur *hash table* yang tetap menghabiskan waktu meskipun data yang ada sedikit. Di samping itu, dari tabel yang sama, tercatat bahwa rata-rata penggunaan memori selama pengujian tiga elemen adalah 14,453125 *KB*. Angka tersebut menunjukkan bahwa meskipun ukuran datanya minimal, penggunaan *Hash Search* tetap memerlukan memori untuk menyimpan struktur tabel. Hal ini memperkuat kesimpulan bahwa algoritma *Hash Search* lebih tepat digunakan ketika ukuran data cukup besar, sehingga penggunaan memori dan waktu yang diperlukan untuk persiapan struktur *hash* menjadi relatif kecil dibandingkan dengan manfaat percepatan dalam proses pencariannya.

Tabel 4. Hasil uji 3 algoritma

Jenis Program	*Rata-rata waktu (ms)*	*Rata-rata memori (kb)*
Linear Search	0,320234	14,453125
Binary Search	0,0003	14,453125
Hash Search	7655,67	14,453125

Berdasarkan hasil simulasi, terlihat bahwa *Hash Search* merupakan algoritma yang paling tidak efisien dari segi waktu komputasi. Akan tetapi perlu kita catat bahwa waktu pada *Hash Search* itu termasuk proses pembentukan struktur *hash*, yang mana hasil ini tidak sepenuhnya mencerminkan

performa murni pencarian, tapi juga operasi persiapan data. *Binary Search* menunjukkan keefisienan terbaik dalam waktu komputasi, meskipun harus mengurutkan datanya terlebih dahulu. Sedangkan *Linear Search* menjadi alternatif paling sederhana namun tidak efisien untuk data berukuran besar. Selain itu, terllihat juga ketiga algoritma tersebut memiliki rata-rata memori yang sama walau dengan waktu komputasi yang berbeda. Penelitian lanjutan disarankan untuk menguji variasi algoritma pencarian lainnya seperti *Interpolation Search* atau *Exponential Search*.

D. PENUTUP

Berdasarkan hasil pengujian terhadap algoritma pencarian *Linear Search, Binary Search*, dan *Hash Search* pada tiga jenis elemen data yang berbeda yaitu 100, 1000, dan 10000 elemen pada bahasa C++. Penelitian ini menguji antara waktu komputasi dan penggunaan memori pada setiap algoritma pemrogramannya. Penelitian ini dapat di buktikan, bahwa *Binary Search* adalah algoritma yang lebih efektif dan efisien dari segi waktu komputasi dengan kecepatan rata-rata 0,0003 ms. Sementara itu, *Linear Search* membuktikan kecepatannya, namun sedikit lebih lambat dari *Binary Search* dengan kecepatan waktu komputasi rata-rata 0,320234 ms, tetapi *Linear Search* ini lebih cocok untuk pencarian data dengan jumlah data yang lebih sedikit dan tidak urut. Namun sebaliknya, hasil pengujian pada *Hash Search* meskipun pada teorinya merupakan algoritma pencarian data tercepat dari kedua algoritma tersebut, pada pengujiannya *Hash Search* justru menunjukkan waktu komputasi yang lebih lambat dari kedua algoritma tersebut dengan waktu komputasi rata-rata 7655,67 ms. Dalam penggunaan memorinya, *Linear Search*, *Binary Search*, dan *Hash Search* keduanya menggunakan memori dengan jumlah yang sama besarannya, yaitu rata-rata sebesar 14,453125 kb.

Dengan demikian, pemilihan algoritma pemrograman dapat dilakukan sesuai kebutuhan, untuk penggunaannya *Binary Search* lebih unggul dalam mencari data dengan waktu komputasi yang lebih cepat walaupun data yang digunakkan harus diurutkan terlebih dahulu. Sedangkan untuk *Hash Search* sendiri, berdasarkan hasil implementasi yang dilakukan dalam penelitian, algoritma tersebut cenderung menunjukkan waktu yang lebih lama dalam proses komputasinya dibandingkan *Linear Search* dan *Binary Search*, walaupun dengan penggunaan memori yang sama. Berbanding terbalik dengan teori yang ada, penelitian ini justru menemukan ketidakefisienan waktu komputasi dan kesamaan dalam pengunaan memori pada *Hash Search,* yang mana jikalau menurut teori yang ada, *Hash Search* itu lebih cepat dan lebih banyak memakan memori kalau dibandingkan dengan *Linear Search* maupun *Binary Search.* Hal ini diduga disebabkan oleh *overhead* pembentukan struktur hash pada scenario data yang relatif kecil.

E. DAFTAR PUSTAKA

Akhsa, A. T. P. D., Kelvin, K., Eldo, H., & Efitra, E. (2024). *Buku Ajar Big Data*. Jambi : PT. Sonpedia Publishing Indonesia.

Ariza, S. F., Majid, A., Himawan, I., Ardhiartha P.U., S., & Pujiono, I. P. (2025). Studi Perbandingan Algoritma Pencarian Binary, Jump, Interpolation, dan Fibonacci: Efisiensi Memori dan Waktu Eksekusi. *JATI : Jurnal Mahasiswa Teknik Informatika*, *9*(4), 7227–7234. https://doi.org/10.36040/jati.v9i4.14360

Bender, M. A., Farach-Colton, M., Kuszmaul, J., Kuszmaul, W., & Liu, M. (2022). On the optimal time/space tradeoff for hash tables. *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*,

1284–1297. https://doi.org/10.1145/3519935.35199

Budiansyah, Komala, I. R., & Nurhayati, L. (2025). Analisis Komparatif Strategi Algoritma Pencarian Dalam Penyelesaian Masalah Kecerdasan Buatan. *Infoman's : Jurnal Ilmu-Ilmu Informatika Dan Manajemen*, *19*(2), 1–5. https://ejournal.unsap.ac.id/index.php/infomans/article/view/2432

Erkamim, E., Abdurrohim, I., Yuliyanti, S., Karim, R., Rahman, A., Admira, T. M. A., & Ridwan, A. (2024). *Buku Ajar Algoritma dan Struktur Data*. Jambi : PT. Sonpedia Publishing Indonesia.

Firmansyah, H., Julian, E., & Ruhiat, A. (2025). Analisis Perbandingan Algoritma Linear Search dan Binary Search dalam Efisiensi Pencarian Data. *Infoman's : Jurnal Ilmu-Ilmu Informatika Dan Manajemen*, *19*(2), 1–7. https://ejournal.unsap.ac.id/index.php/infomans/article/view/2398

Izhari, F. (2025). *Algoritma dan Pemrograman*. Payakumbuh : Serasi Media Teknologi.

Leana, S. A., Ginting, M. A. P., Izdihar, M. B., Pratama, T. S. A., Manik, D. A. A., & Gunawan, I. (2025). Perbandingan Efisiensi Linear dan Binary Search Dalam Pencarian Nama Siswa Pada Struktur Data Array. *JRSIKOM : Jurnal Riset Sistem Informasi Dan Aplikasi Komputer*, *1*(2), 45–50. https://doi.org/10.180997/jrsikom.v1i2.41

Mevia, N. A. O., Marbun, Y. K., Putri, M. D., & Sitompul, Y. R. (2026). Perbandingan Algoritma Divide and Conquer dan Searching pada Pengolahan Data Nilai Mahasiswa Berbasis Web. *TEKNIK : Jurnal Ilmu Teknik Dan Informatika*, *6*(1), 60–79. https://doi.org/10.51903/teknik.v1i1.1145

Puranik, T. A. (2025). Performance Analysis of Sorting and Searching Algorithms. *IRJAEM : International Research Journal on Advanced Engineering and Management*, *3*(8), 2741–2746. https://doi.org/10.47392/IRJAEM.2025.0430

Purbasari, W., Iqbal, T., Inayah, I., Munawir, M., Sutjiningtyas, S., Hikmawati, E., Natsir, F., Widhiyanti, A. A. S., Wall, M., & Haris, M. S. (2024). *Algoritma Pemrograman*. Sukabumi : CV Haura Utama.

Purnama, N. (2025). Comparatrive Performance Study of Search Algorithms on Large-Scale Data Sstructures. *JITK : Jurnal Ilmu Pengetahuan Dan Teknologi Komputer*, *11*(1), 99–109. https://doi.org/10.33480/jitk.v11i1.6592

Putra, R. A. (2024). *Buku Sakti Pemrograman C++ Untuk Pemula*. Yogyakarta : Anak Hebat Indonesia.

Putra, R. F., Zebua, R. S. Y., Budiman, B., Rahayu, P. W., Bangsa, M. T. A., Zulfadhilah, M., Choirina, P., Wahyudi, F., & Andiyan, A. (2023). *Data Mining: Algoritma dan Penerapannya*. Jambi : PT. Sonpedia Publishing Indonesia.

Sitorus, Z., Renyaan, A. S., Si, S., Kmurawak, R. M. B., & Lokollo, P. D. (2024). *Tinjauan mendalam tentang ilmu komputer: konsep dasar, algoritma, dan perkembangan terkini: buku referensi*. Medan : PT Media Penerbit Indonesia.

Situmorang, H. (2017). Analisa algoritma pada metoda pencarian linier, biner dan interpolasi. *Jurnal Mahajana Informasi*, *2*(2), 31–41. https://doi.org/10.51544/jurnalmi.v2i2.177

Suryadi, D., Octiva, C. S., Fajri, T. I., Nuryanto, U. W., & Hakim, M. L. (2024). Optimasi Kinerja Sistem IoT Menggunakan Teknik Edge Computing.

Jurnal Minfo Polgan, *13*(2), 1456–1461. https://doi.org/10.33395/jmp.v13i2.14102

Yusuf, A. D., Abdullahi, S., Boukar, M. M., & Yusuf, S. I. (2021). Collision Resolution Techniques in Hash Table: A Review. *IJACSA : International Journal of Advanced Computer Science and Applications*, *12*(9), 757–762. https://doi.org/10.14569/IJACSA.2021.0120984