JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3085-3092

Pengembangan Visualisasi Interaktif untuk Analisis Komparatif Binary Search dan Jump Search

Yohanes Gerardus Haga Zai[1], Callysa Elistia[2], Yehezkiel Haganta Tarigan[3*], Adidtya Perdana[4]

[1-4]Fakultas Matematika dan Ilmu Pengetahuan Alam, Program Studi Ilmu Komputer, Universitas Negeri Medan, Medan, Indonesia
Email: [1]yohanesgerardushagazai@gmail.com , [2]callysaelistia@gmail.com, [3*]yehezkielhaganta@gmail.com, [4]adidtya@unimed.ac.id
(* : coressponding author)

Abstrak−Penelitian ini berfokus pada pengembangan perangkat lunak visualisasi interaktif menggunakan bahasa pemrograman Python dan pustaka Pygame untuk menganalisis kinerja algoritma Binary Search dan Jump Search secara empiris. Meskipun teori asimtotik konvensional telah menetapkan kompleksitas waktu masing-masing algoritma, representasi kognitif murni melalui teks sering kali tidak memadai untuk memahami realitas eksekusi mekanis pada perangkat keras. Sistem visualisasi ini dirancang untuk mengeksekusi kedua algoritma secara konkuren (simultan) pada representasi spasial yang sama guna menyingkirkan bias kecepatan perangkat keras keras dalam proses pengujian. Pengujian dilakukan melalui simulasi variasi ukuran himpunan data berskala masif dengan kalkulasi batas langkah maksimum eksak pada kondisi terburuk. Hasil simulasi secara mutlak memvalidasi bahwa strategi pembagian ruang secara eksponensial pada Binary Search dengan kompleksitas $O(\log n)$ memiliki tingkat skalabilitas yang jauh lebih superior dibandingkan Jump Search dengan kompleksitas $O(\sqrt{n})$. Meskipun Jump Search telah dieksekusi menggunakan parameter ukuran lompatan matematis yang paling optimal yaitu $m = \sqrt{n}$, algoritma tersebut terbukti tetap tidak mampu mengungguli efisiensi waktu eksekusi Binary Search seiring bertambahnya jumlah elemen data.

Kata Kunci: Binary Search; Jump Search; Kompleksitas Waktu; Visualisasi Algoritma; Pygame.

Abstract−*This research focuses on the development of interactive visualization software using the Python programming language and the Pygame library to empirically analyze the performance of Binary Search and Jump Search algorithms. Although conventional asymptotic theory has established the time complexity of each algorithm, pure cognitive representation through text is often inadequate to understand the reality of mechanical execution on hardware. The visualization system is designed to execute both algorithms concurrently on the same spatial representation , effectively eliminating hardware speed bias during the testing process. Testing was conducted through simulations of varying large-scale dataset sizes , calculating the exact maximum step limits under worst-case scenarios. The simulation results absolutely validate that the exponential space division strategy in Binary Search , with a complexity of* $O(log\ n)$ *, possesses vastly superior scalability compared to Jump Search , which operates at* $O(\sqrt{n})$*. Even when Jump Search is configured using the most optimal mathematical step size parameter of* $m = \sqrt{n}$ *, the algorithm consistently fails to outperform the execution time efficiency of Binary Search in worst-case conditions as the number of data elements increases .*

Keywords*: Binary Search; Jump Search; Time Complexity; Algorithm Visualization; Pygame.*

1. PENDAHULUAN

Dalam lanskap komputasi modern dan rekayasa data berskala besar, efisiensi operasi pencarian informasi menjadi fondasi absolut yang mendasari berbagai sistem berkinerja tinggi, mulai dari arsitektur basis data terdistribusi tingkat perusahaan hingga mesin pencari web berskala global (Markuci & Prianto, 2022; Shabbir et al., 2023). Algoritma pencarian secara fundamental bertugas untuk mengidentifikasi keberadaan atau melokalisasi posisi spesifik dari sebuah elemen target di dalam sekumpulan ruang data. Secara teoretis, struktur data dan ilmu algoritma telah memberikan berbagai pendekatan paradigmatis untuk menyelesaikan masalah komputasional ini, di mana masing-masing pendekatan membawa karakteristik kompleksitas waktu dan efisiensi ruang yang berbeda secara signfikan. Pada ruang data yang telah terurut sebelumnya, dua strategi pencarian yang paling sering menjadi subjek studi komparatif dan perdebatan arsitektural adalah algoritma Binary Search dan Jump Search (Ariza et al., 2025). Keduanya menawarkan paradigma pemecahan masalah yang secara fundamental berlawanan: Binary Search beroperasi dengan metodologi membelah ruang data menjadi dua bagian secara berulang melalui pendekatan rekursif

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3085-3092

atau iteratif yang bertumpu pada strategi divide-and-conquer, sedangkan Jump Search mengambil pendekatan melompati sekumpulan elemen dalam blok-blok berukuran tetap secara linier, sebelum akhirnya melakukan pencarian sekuensial yang sangat presisi di dalam blok yang relevan (Ariza et al., 2025; Setiawan et al., 2024).

Pertarungan antara strategi pembagian ruang secara eksponensial versus pergerakan lompatan blok spasial ini menghadirkan sebuah narasi analitis yang sangat kuat dalam literatur Desain dan Analisis Algoritma. Di atas kertas, berdasarkan analisis asimtotik konvensional yang direpresentasikan melalui Notasi Big-O, algoritma Binary Search memiliki kompleksitas waktu ruang pada orde $O(\log n)$, yang secara matematis selalu dianggap jauh lebih superior dan mutlak lebih efisien dibandingkan dengan kompleksitas waktu algoritma Jump Search yang berada pada orde $O(\sqrt{n})$ (Yasmin et al., 2025). Teori kompleksitas komputasional tradisional akan dengan cepat menyimpulkan bahwa untuk ukuran kumpulan data yang meraksasa dan mendekati tak terhingga, sebuah algoritma dengan laju pertumbuhan logaritmik akan selalu mengalahkan algoritma dengan laju pertumbuhan akar kuadrat, seolah-olah menjadikan Binary Search sebagai solusi definitif tanpa celah untuk semua kebutuhan sistem.

Namun, realitas implementasi komputasi fisik di dunia nyata sering kali menghadirkan anomali mengejutkan yang menentang dan membongkar abstraksi matematis murni tersebut. Sebuah fenomena krusial yang secara aktif diteliti oleh para pengelola jurnal ilmiah, analis sistem, dan insinyur arsitektur komputer adalah pencarian bukti empiris dari fenomena perbenturan antara pengujian teori asimtotik versus realitas mekanika perangkat keras. Dalam eksekusi nyata di atas silikon prosesor modern, performa absolut dari sebuah algoritma tidak lagi hanya ditentukan oleh perhitungan jumlah agregat operasi perbandingan atau iterasi matematis semata, melainkan didikte secara ketat oleh faktor-faktor arsitektural internal yang sangat kompleks seperti topologi Hierarki Memori, tingginya Latensi Akses Memori, utilisasi CPU Cache pada level L1, L2, dan L3, serta efisiensi unit Prediksi Cabang di dalam jalur pipa instruksi prosesor (Rorong et al., 2025).

Selain fenomena anomali arsitektural yang tersembunyi tersebut, eksplorasi analitis yang mendalam terhadap algoritma Jump Search memberikan fleksibilitas eksperimental tingkat lanjut yang tidak dimiliki oleh arsitektur kaku dari Binary Search. Peneliti dan insinyur perangkat lunak memiliki kemampuan untuk memanipulasi parameter variabel arsitektural yang kritis, yakni ukuran langkah lompatan atau besaran blok data itu sendiri. Meskipun secara tradisional ilmu kalkulus menetapkan bahwa ukuran lompatan blok yang paling optimal secara matematis adalah akar kuadrat dari total data, manipulasi empiris secara sengaja terhadap nilai variabel ini berdasarkan pertimbangan ukuran baris tembolok perangkat keras fisik membuka dimensi metodologi penelitian baru yang dikenal sebagai algoritma sadar mesin (Ariza et al., 2025). Kemampuan untuk melakukan rekayasa parameter ini menjadi bukti empiris tentang bagaimana perangkat lunak dapat secara aktif disinkronkan dengan infrastruktur perangkat keras yang mendasarinya.

Mengingat kompleksitas interaksi tingkat rendah antara teori algoritma asimtotik dan arsitektur perangkat keras mikroskopis ini, representasi kognitif murni melalui teks sering kali dinilai tidak memadai bagi sivitas akademika maupun praktisi rekayasa perangkat lunak untuk memahami apa yang sebenarnya terjadi di dalam silikon. Oleh karena itu, pengembangan sebuah aplikasi visualisasi algoritma pencarian menjadi sebuah urgensi metodologis sekaligus inovasi edukasional yang mendasar. Pembangunan perangkat visualisasi komprehensif menggunakan ekosistem grafis Pygame di dalam bahasa pemrograman Python menawarkan representasi rendering waktu nyata yang sanggup menjembatani jurang pemahaman antara teori komputasi yang tak terlihat dengan realitas eksekusi mekanis yang empiris (B et al., 2025; Bartaula, 2025; Sahu, 2025). Sebuah aplikasi visualisasi algoritma tingkat lanjut tidak hanya sekadar merepresentasikan pergerakan indeks array yang sedang diakses, melainkan dirancang secara khusus untuk mampu merepresentasikan status simulasi memori, latensi siklus waktu nyata, dan metrik biaya komputasi komparatif, yang pada gilirannya memberikan wawasan analitis multidimensi yang sangat esensial bagi penelitian jurnal bertaraf internasional. Analisis mendalam ini akan secara komprehensif membongkar, mengevaluasi, dan menyintesis perbandingan antara Binary Search dan Jump Search dari tataran teori matematis abstrak hingga observasi arsitektural silikon yang sangat empiris, sembari meletakkan landasan kuat bagi sistem visualisasi Pygame sebagai instrumen eksplorasi komputasional masa depan.

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3085-3092

2. METODE

2.1 Pemodelan Antarmuka Interaktif untuk Pengujian Konkuren

Perangkat lunak visualisasi dalam penelitian ini dikembangkan menggunakan bahasa pemrograman Python yang didukung oleh pustaka antarmuka grafis Pygame. Penggunaan Pygame dipilih untuk menangani *rendering* visual dan interaksi pengguna secara *real-time* tanpa membebani memori komputasi secara berlebihan. Sistem ini dirancang sebagai lingkungan eksperimen interaktif yang memfasilitasi pengguna untuk melakukan evaluasi kinerja algoritma pencarian melalui beberapa modul fungsional utama:

2.1.1 Simulasi Berdampingan

Fitur utama dari simulasi ini adalah kemampuan untuk mengeksekusi algoritma *Binary Search* dan *Jump Search* secara simultan (konkuren) pada representasi visual spasial (grid) yang sama. Pendekatan "balap algoritma" ini diimplementasikan menggunakan fungsi *Generator* pada Python, sehingga proses iterasi pencarian dapat divisualisasikan langkah demi langkah (*step-by-step*). Eksekusi konkuren ini memastikan bahwa perbandingan kecepatan pencarian murni didasarkan pada kompleksitas logis algoritma, bukan dipengaruhi oleh *delay* perangkat keras.

2.1.2 Manipulasi Himpunan Data Dinamis

Untuk menguji skalabilitas algoritma, lingkungan simulasi menyediakan fitur pembangkitan himpunan data secara otomatis dalam skala kecil maupun besar . Selain itu, sistem mendukung manipulasi data dinamis di mana pengguna dapat menyisipkan nilai secara bebas ke dalam himpunan data yang sedang diuji. Guna menjaga validitas eksperimen, sistem dilengkapi dengan mekanisme *auto-sorting* yang memastikan himpunan data selalu dalam keadaan terurut (*sorted array*), yang merupakan prasyarat mutlak bagi kedua algoritma pencarian tersebut

2.1.3 Custom Step Size

Khusus untuk analisis *Jump Search*, sistem memfasilitasi pengujian parameter eksternal dengan menyediakan modul input batas lompatan (*step size*) secara dinamis. Pengguna dapat mengevaluasi efisiensi *Jump Search* dengan memasukkan nilai lompatan secara acak, atau menggunakan tombol kalkulasi optimal yang secara otomatis menghitung nilai akar kuadrat dari total himpunan data ($m = \sqrt{n}$). Fitur ini memungkinkan observasi mendalam mengenai degradasi kinerja *Jump Search* ketika parameter *step* berada dalam kondisi sub-optimal.

2.1.4 Metrik Hasil dan Kompleksitas Eksak

Sebagai luaran eksperimen, antarmuka sistem dilengkapi dengan panel status analitis. Ketika target data ditemukan, sistem tidak hanya menampilkan indikator visual (legenda warna), tetapi juga menyajikan rekapitulasi performa. Panel ini menampilkan algoritma mana yang menyelesaikan pencarian lebih cepat beserta kalkulasi batas langkah eksak (matematis) berdasarkan ukuran himpunan data (n) pada saat itu.

2.2 Kompleksitas Binary Search dan Jump Search

Dalam lingkungan simulasi ini, kinerja kedua algoritma pencarian dievaluasi dan dibandingkan berdasarkan jumlah maksimum langkah operasional (iterasi) yang diperlukan untuk menemukan target pada kondisi terburuk (*worst-case scenario*). Karena kedua algoritma beroperasi pada himpunan data yang terurut (*sorted array*), perhitungan kompleksitas waktu direpresentasikan melalui pendekatan matematis langkah demi langkah yang tervisualisasi pada sistem.

2.2.1 Analisis Langkah Binary Search

Binary Search menerapkan paradigma *divide and conquer* dengan membagi ruang pencarian menjadi dua bagian pada setiap iterasinya. Algoritma ini membandingkan nilai target dengan elemen tengah (*midpoint*) dari himpunan data. Jika nilai target lebih kecil, pencarian difokuskan pada paruh pertama; sebaliknya, jika lebih besar, pencarian dilanjutkan pada paruh kedua (Nurvita & Putri, 2025).

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3085-3092

Reduksi ruang pencarian secara eksponensial ini menghasilkan kompleksitas waktu logaritmik, yakni $O(\log n)$ Pada perangkat lunak visualisasi yang dikembangkan, indikator evaluasi memproyeksikan batas eksak jumlah iterasi maksimum (*Max Steps*) yang dihitung menggunakan formulasi:

$$\text{Max Steps}_{binary} = \lceil \log_2(n) \rceil + 1 \quad (1)$$

di mana n merepresentasikan jumlah total elemen data. Tambahan konstanta 1 mengindikasikan langkah komparasi akhir ketika ruang pencarian hanya menyisakan satu elemen.

2.2.2 Analisis Langkah Jump Search

Jump Search merupakan algoritma hibrida yang menggabungkan pencarian berbasis blok dengan pencarian sekuensial (*Sequential Search*). Algoritma ini melompat melewati himpunan data dengan interval langkah (blok) sebesar m hingga menemukan blok di mana elemen target berada, untuk kemudian melakukan pencarian linier di dalam blok tersebut.

Tingkat efisiensi *Jump Search* sangat bergantung pada penentuan ukuran lompatan m. Secara teoritis, kinerja paling optimal (*best-case step*) dicapai ketika ukuran lompatan adalah akar kuadrat dari jumlah elemen data ($m = \sqrt{n}$). Dengan menggunakan lompatan optimal ini, kompleksitas waktu *Jump Search* berada pada $O(\sqrt{n})$. Dalam status panel sistem visualisasi, kalkulasi maksimum iterasi (*Max Steps*) yang mungkin terjadi dihitung melalui pemodelan:

$$\text{Max Steps}_{jump} = \left\lfloor \frac{n}{m} \right\rfloor + m - 1 \quad (2)$$

Pada formulasi di atas, [n/m] merepresentasikan jumlah lompatan maksimum antar blok, sedangkan $m - 1$ adalah jumlah perbandingan maksimum saat pencarian linier di dalam blok terakhir.

2.3 Optimasi Ukuran Lompatan (Step Size) pada Jump Search

Kinerja algoritma *Jump Search* sangat dipengaruhi oleh penentuan interval lompatan atau *step size* (disimbolkan dengan m). Berbeda dengan algoritma pencarian tradisional yang statis, *Jump Search* memberikan fleksibilitas untuk memanipulasi ukuran blok data yang akan dilewati sebelum melakukan pencarian linier. Secara konseptual, eksekusi algoritma ini terbagi menjadi dua fase utama: fase pelompatan antar batas blok dan fase pemindaian linier (*Sequential Search*) di dalam blok yang teridentifikasi.

Kondisi terburuk (*worst-case*) pada algoritma ini terjadi apabila elemen target berada tepat di indeks terakhir dari blok pencarian terakhir, atau elemen tersebut sama sekali tidak terdapat dalam himpunan data. Pada skenario tersebut, algoritma harus melakukan lompatan maksimum sebanyak $\frac{n}{m}$ kali, diikuti dengan pencarian linier sebanyak $m - 1$ kali. Total perbandingan maksimum yang dilakukan dapat direpresentasikan melalui fungsi $f(m) = \frac{n}{m} + m - 1$. Untuk mencapai efisiensi komputasi tertinggi, fungsi total perbandingan tersebut harus diminimalkan. Berdasarkan prinsip kalkulus diferensial, nilai minimum tercapai ketika turunan pertama dari fungsi tersebut terhadap m bernilai nol, yaitu:

$$f'(m) = -\frac{n}{m^2} + 1 = 0 \quad (3)$$

$$\frac{n}{m^2} = 1 \Rightarrow m^2 = n \Rightarrow m = \sqrt{n} \quad (4)$$

Melalui pembuktian matematis tersebut, disimpulkan bahwa *step size* paling optimal untuk meminimalkan kompleksitas waktu adalah akar kuadrat dari total elemen himpunan data ($m = \sqrt{n}$).

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3085-3092

Perangkat lunak visualisasi yang dikembangkan mengakomodasi pembuktian empiris dari teori optimasi ini melalui parameter *custom step* yang interaktif. Pengguna dapat mengevaluasi degradasi kinerja *Jump Search* saat algoritma dijalankan dengan parameter sub-optimal. Sebagai contoh, jika pengguna menginput nilai lompatan yang terlalu kecil (misalnya $m = 1$), visualisasi akan memperlihatkan perilaku *Jump Search* yang mereduksi menjadi *Sequential Search* murni dengan kompleksitas waktu yang membengkak menjadi $O(n)$. Sebaliknya, jika nilai lompatan diatur terlalu besar (mendekati nilai n), jumlah pelompatan memang berkurang secara signifikan, namun fase pencarian linier di dalam blok menjadi terlalu panjang, yang juga memicu inefisiensi komputasi. Untuk memvalidasi kondisi ideal secara praktis, sistem menyediakan fitur kalkulasi otomatis (tombol *Opt*) yang langsung mengimplementasikan nilai $m = \sqrt{n}$. Fitur ini memungkinkan pengguna untuk secara presisi membandingkan efisiensi waktu eksekusi *Jump Search* yang telah dioptimasi melawan algoritma *Binary Search* secara langsung pada himpunan data yang sama.

2.4 Skenario Pengujian dan Parameter Evaluasi Komparatif

Untuk mengukur efisiensi *Binary Search* dan *Jump Search* secara komprehensif, serangkaian skenario pengujian komparatif dieksekusi melalui perangkat lunak visualisasi. Skenario pertama berfokus pada variasi ukuran himpunan data guna mengobservasi skalabilitas performa antara kompleksitas $O(\log n)$ dan $O(\sqrt{n})$ seiring bertambahnya jumlah elemen (n). Skenario kedua meninjau distribusi posisi elemen target—meliputi posisi awal, tengah, akhir, hingga kondisi data tidak ditemukan (*not found*)—untuk memvalidasi algoritma pada kondisi *best-case, average-case*, dan *worst-case*. Selain itu, uji sensitivitas ukuran lompatan (*step size*) juga diterapkan secara eksklusif pada *Jump Search* untuk membuktikan degradasi efisiensi waktu komputasi apabila menggunakan parameter arbitrer dibandingkan parameter optimalnya ($m = \sqrt{n}$). Seluruh skenario pengujian ini dievaluasi memadukan parameter kualitatif, yakni pelacakan iterasi logis secara *real-time* melalui animasi pencarian konkuren, serta parameter kuantitatif berupa kalkulasi batas langkah maksimum eksak (teoritis) pada panel status sistem, sehingga perbandingan kinerja kedua algoritma dapat diukur secara objektif dan akurat.

Parameter evaluasi komparatif utama yang dinilai dalam seluruh skenario pengujian tersebut adalah jumlah iterasi logis yang direpresentasikan melalui animasi pencarian konkuren. Perangkat lunak secara otomatis melacak dan membandingkan pergerakan kedua algoritma secara real-time, kemudian menentukan algoritma mana yang mencapai kondisi terminasi lebih awal.

3. ANALISA DAN PEMBAHASAN

3.1 Implementasi Lingkungan Visualisasi Interaktif

Perangkat lunak visualisasi dalam penelitian ini berhasil direalisasikan menggunakan bahasa pemrograman Python dengan pustaka antarmuka grafis *Pygame*. Antarmuka pengguna (*User Interface*) dirancang secara modular dan dibagi menjadi tiga area observasi utama guna memfasilitasi jalannya eksperimen secara interaktif dan *real-time*.

3.1.1 Implementasi Lingkungan Visualisasi Interaktif

Layar utama aplikasi terbagi menjadi area kendali dan area visualisasi. Sisi kiri layar dialokasikan sebagai panel kendali (*control panel*) yang memuat parameter input dan pemicu eksekusi. Panel ini mencakup tombol pembangkitan himpunan data dinamis ("Data Kecil" dan "Data Besar"), modul penambahan data mandiri ("Tambah Data"), serta parameter input pencarian dan modifikasi *step size* khusus untuk *Jump Search*. Sisi kanan layar merepresentasikan ruang observasi spasial di mana himpunan data dirender ke dalam bentuk matriks (*grid*) berwujud kotak-kotak berurutan yang dilengkapi dengan fungsionalitas *scroll* vertikal otomatis untuk menangani himpunan data berskala besar.

3.1.2 Implementasi Lingkungan Visualisasi Interaktif

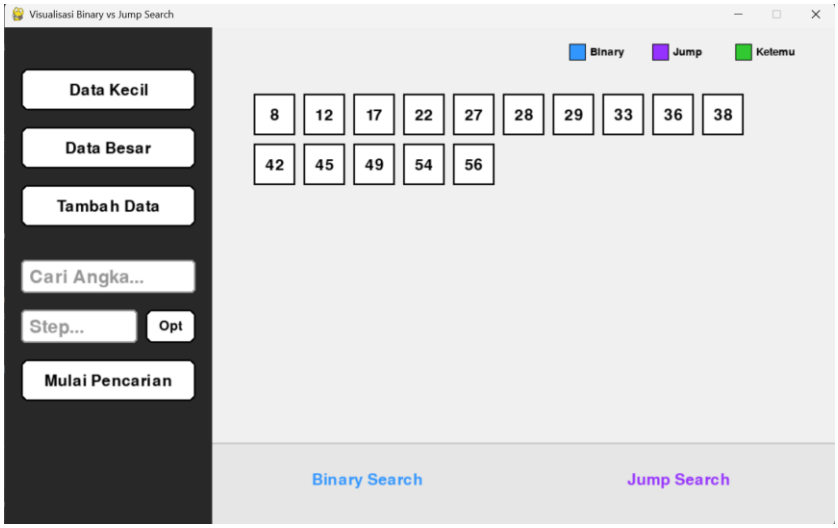
Untuk membedakan pergerakan algoritma saat dieksekusi secara konkuren, sistem menerapkan metode pengkodean warna (*color-coding*) yang direpresentasikan melalui legenda di pojok kanan atas. Algoritma *Binary Search* direpresentasikan dengan sorotan warna biru, sementara

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3085-3092

Jump Search ditandai dengan sorotan warna ungu. Ketika sebuah kotak elemen menjadi target evaluasi dari kedua algoritma pada iterasi yang sama, sistem secara dinamis merender kedua warna tersebut secara tumpang tindih (*overlapping*). Elemen target yang berhasil diidentifikasi akan ditandai dengan perubahan warna menjadi hijau beserta ikon centang sebagai indikator terminasi keberhasilan pencarian.

3.1.3 Implementasi Lingkungan Visualisasi Interaktif

Sebagai pelengkap observasi, bagian bawah layar dialokasikan sebagai panel status analitis. Panel ini bertugas menangkap luaran eksperimen secara kuantitatif. Segera setelah salah satu atau kedua algoritma mencapai terminasi, panel ini menampilkan rekapitulasi algoritma mana yang memenangkan komputasi waktu lebih cepat. Lebih lanjut, panel ini menyajikan proyeksi matematis berupa kalkulasi batas langkah eksak (teoritis) dari *Binary Search* dan *Jump Search* berdasarkan total elemen data yang sedang diuji, sehingga pengguna dapat memvalidasi animasi visual dengan teori kompleksitas secara presisi.



Gambar 1. Antarmuka utama perangkat lunak visualisasi algoritma pencarian.

3.2 Analisis Pengaruh Ukuran Himpunan Data (Skalabilitas)

Skenario pengujian skalabilitas dilakukan untuk mengobservasi bagaimana kinerja *Binary Search* dan *Jump Search* beradaptasi terhadap peningkatan jumlah elemen dalam himpunan data. Sesuai dengan teori kompleksitas asimtotik, algoritma yang efisien harus mampu meminimalkan lonjakan jumlah iterasi komputasi ketika ukuran himpunan data diperbesar secara signifikan. Dalam pengujian ini, algoritma *Jump Search* dikonfigurasi menggunakan ukuran lompatan optimal yang dikalkulasi secara otomatis oleh sistem, guna memastikan perbandingan performa dilakukan pada kondisi efisiensi terbaik dari kedua algoritma.

Tabel 1. Perbandingan Kalkulasi Iterasi Maksimum Berdasarkan Ukuran Data

Ukuran Data	Iterasi *Jump Search*	Iterasi *Binary Search*	Selisih Langkah
16	7	5	2
22	7	14	7

Berdasarkan pengujian pada lingkungan simulasi, perbedaan skalabilitas antara kompleksitas terbukti secara empiris. Untuk memperjelas perbandingan pertumbuhan langkah komputasi pada kondisi terburuk (worst-case scenario), Tabel 1 menyajikan rekapitulasi kalkulasi batas langkah maksimum eksak dari kedua algoritma berdasarkan simulasi.

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3085-3092

Data pada Tabel 1 memvalidasi bahwa Binary Search memiliki tingkat skalabilitas yang jauh lebih superior. Validasi empiris dari Tabel 1 divisualisasikan secara real-time melalui antarmuka perangkat lunak pada Gambar 2.

Gambar 2. Perbandingan hasil kalkulasi iterasi eksak kondisi worst-case pada himpunan data kecil dan data besar dengan parameter lompatan optimal.

Melalui Gambar 2(a) dan 2(b), terlihat jelas dominasi performa *Binary Search*. Ketika ukuran elemen data dinaikkan dari 15 menjadi 61 elemen, jumlah iterasi maksimum *Binary Search* hanya bertambah 2 langkah (dari 5 menjadi 7). Sebaliknya, *Jump Search* yang dioptimasi secara matematis sekalipun mengalami lonjakan iterasi yang signifikan, bertambah sebanyak 7 langkah (dari 7 menjadi 14). Hal ini secara empiris mengonfirmasi bahwa untuk himpunan data terurut dalam skala masif, reduksi ruang pencarian secara eksponensial (dibagi dua) membuat *Binary Search* jauh lebih unggul dibandingkan *Jump Search*.

4. KESIMPULAN

Pengembangan perangkat lunak visualisasi interaktif ini berhasil diimplementasikan sebagai instrumen evaluasi empiris yang efektif untuk mengukur efisiensi algoritma pencarian. Melalui pendekatan eksekusi konkuren, sistem ini memfasilitasi pengamatan visual secara *real-time* terhadap cara kerja logis algoritma, sekaligus menyingkirkan bias kecepatan perangkat keras keras dalam proses pengujian.

Dari segi analisis kinerja, hasil simulasi secara mutlak memvalidasi teori kompleksitas asimtotik bahwa algoritma Binary Search $O(\log n)$ memiliki tingkat skalabilitas yang jauh lebih superior dibandingkan Jump Search $O(\sqrt{n})$. Metode pembagian ruang pencarian secara eksponensial pada Binary Search sangat efektif menekan pembengkakan jumlah iterasi pada data berskala masif. Di sisi lain, meskipun Jump Search telah dikonfigurasi menggunakan ukuran lompatan matematis yang paling optimal ($m = \sqrt{n}$), algoritma ini tetap tidak mampu mengungguli efisiensi waktu eksekusi Binary Search pada kondisi terburuk (worst-case) seiring bertambahnya ukuran data.

Secara keseluruhan, penelitian ini berhasil menjembatani analisis kompleksitas algoritma yang awalnya murni bersifat teoretis-matematis menjadi sebuah pembuktian visual yang interaktif, terukur, dan objektif.

REFERENCES

Ariza, S. F., Majid, A., Yahya, M. H., Himawan, I., Ardhiartha, S., Prayogo, I., Informatika, P., Negeri, U., Abdurrahman, I. K. H., Pekalongan, W., Pahlawan, J., Kajen, R., & Pekalongan, K. (2025). STUDI PERBANDINGAN ALGORITMA PENCARIAN BINARY, JUMP, INTERPOLATION, DAN FIBONACCI : EFISIENSI MEMORI DAN WAKTU EKSEKUSI. In *Jurnal Mahasiswa Teknik Informatika)* (Vol. 9, Number 4).

B, D. V, H, S. N., Gatti, S. N., K, C. B., Professor at JNNCE, A., & at JNNCE, S. (2025). *Algorithm Visualizer With Pygame And Tkinter*. www.ijcrt.org

JRIIN : Jurnal Riset Informatika dan Inovasi
Volume 3, No. 12 Tahun 2026
ISSN 3025-0919 (media online)
Hal 3085-3092

Bartaula, B. (2025). *Interactive Maze Generation and Pathfinding Simulation: A Comprehensive Educational Platform for Algorithm Visualization*.

Markuci, D., & Prianto, C. (2022). ANALISIS PERBANDINGAN PENGGUNAAN ALGORITMA SEQUENTIAL SEARCH DAN BINARY SEARCH PADA APLIKASI SURAT PERJALANAN DINAS. *JATI (Jurnal Mahasiswa Teknik Informatika)*, *6*, 110–119. https://doi.org/10.36040/jati.v6i1.4569

Rorong, G. Y., Fachri Pane, S., Amran, M., Siregar, H., & Bangsa, M. (2025). Optimizing Search Efficiency in Ordered Data: A Hybrid Approach Using Jump Binary Search. *Journal of Dinda Data Science, Information Technology, and Data Analytics*, *5*(1), 36–44. http://journal.ittelkom-pwt.ac.id/index.php/dinda

Sahu, P. K. (2025). Algo Visualix: A Python-Based Algorithm Visualizer for Educational Enhancement. *International Journal for Research in Applied Science and Engineering Technology*, *13*, 2610–2614. https://doi.org/10.22214/ijraset.2025.70719

Setiawan, M. N., Roring, R. S., Atma, Y. D., & Tetiawadi, H. (2024). Studi Empiris Terhadap Asistensi Artificial Intelligence (AI) Dalam Rancang Bangun Aplikasi. *Digital Transformation Technology*, *4*, 364–373. https://doi.org/10.47709/digitech.v4i1.4115

Shabbir, A., Majeed, A., Iftikhar, M., Ali, R. H., Arshad, U., Shabbir, M. Z., Zeeshan Ijaz, A., Ali, N., & Aftab, A. (2023). A Review of Algorithms's Complexities on Different Valued Sorted and Unsorted Data. *2023 International Conference on IT and Industrial Technologies, ICIT 2023*. https://doi.org/10.1109/ICIT59216.2023.10335840

Yasmin, N. N., Sofia, L., Sabrina, A. R. P., Putri, A. A.-Z., & Pujiono, I. P. (2025). Interpolation Searching Algorithm Vs Algoritma Pencarian Tradisional: Analisis Efisiensi Memori dan Waktu Komputasi. *SIMKOM*, *10*, 212–223. https://doi.org/10.51717/simkom.v10i2.857