

Perbandingan Algoritma *String Matching* Brute Force dan Rabin-Karp pada Sistem Pencarian Judul Skripsi (Studi Kasus Universitas Bina Darma)

Joe Mario ^{1*}, Ahmad Haidar Mirza ²

^{1,2} Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Bina Darma, Kota Palembang, Provinsi Sumatera Selatan, Indonesia.

ABSTRACT

Fast and accurate information retrieval for thesis titles is crucial within the academic environment of Universitas Bina Darma to prevent research duplication. The massive increase in data volume demands the implementation of mathematically efficient string matching algorithms. This study aims to compare the performance of the Brute Force and Rabin-Karp algorithms in executing exact matching on a dataset of thesis titles. A quantitative experimental approach was utilized by executing and isolating pure logic computation within a Node.js environment. Measurements focused on execution time and the number of character comparison operations across variations in keyword length, data volume, and worst-case scenarios. Experimental results demonstrate that the Brute Force algorithm consistently outperforms Rabin-Karp in terms of execution speed. In the worst-case scenario, Brute Force required only 20.0970 ms to confirm that a pattern was not found, whereas Rabin-Karp required 42.4836 ms. Based on the comprehensive empirical data, Brute Force is concluded to be the most optimal algorithm for the text characteristics of the thesis title dataset at Universitas Bina Darma, while Rabin-Karp remains advantageous in reducing the number of character comparison operations.

ABSTRAK

Pencarian informasi judul skripsi yang cepat dan akurat sangat krusial di lingkungan akademik Universitas Bina Darma guna mencegah duplikasi penelitian. Peningkatan volume data secara masif menuntut penerapan algoritma string matching yang efisien secara matematis. Penelitian ini bertujuan membandingkan kinerja algoritma Brute Force dan Rabin-Karp dalam melakukan pencarian eksak (exact matching) pada himpunan data judul skripsi. Pendekatan eksperimental kuantitatif digunakan dengan mengeksekusi dan mengisolasi komputasi logika murni pada lingkungan Node.js. Pengukuran difokuskan pada kecepatan waktu eksekusi dan jumlah operasi perbandingan karakter, dengan variasi panjang kata kunci, volume data, dan skenario terburuk. Hasil eksperimen menunjukkan algoritma Brute Force secara konsisten lebih unggul dalam kecepatan waktu dibandingkan Rabin-Karp. Pada skenario terburuk, Brute Force hanya membutuhkan 20,0970 ms untuk memastikan pola tidak ditemukan, sedangkan Rabin-Karp membutuhkan 42,4836 ms. Berdasarkan keseluruhan data empiris, Brute Force disimpulkan sebagai algoritma paling optimal untuk karakteristik dataset judul skripsi di Universitas Bina Darma, sementara Rabin-Karp tetap unggul dalam mereduksi jumlah operasi perbandingan karakter.

ARTICLE HISTORY

Received: 9 September 2026.
Revised: 11 September 2026.
Accepted: 26 September 2026.
Available Online: 10 October 2026.
Published: 1 December 2026.

KEYWORDS

String Matching; Brute Force;
Rabin-Karp; Exact Matching;
Execution Time.

KATA KUNCI

String Matching; Brute Force;
Rabin-Karp; Pencarian Eksak;
Waktu Eksekusi.

1. Pendahuluan

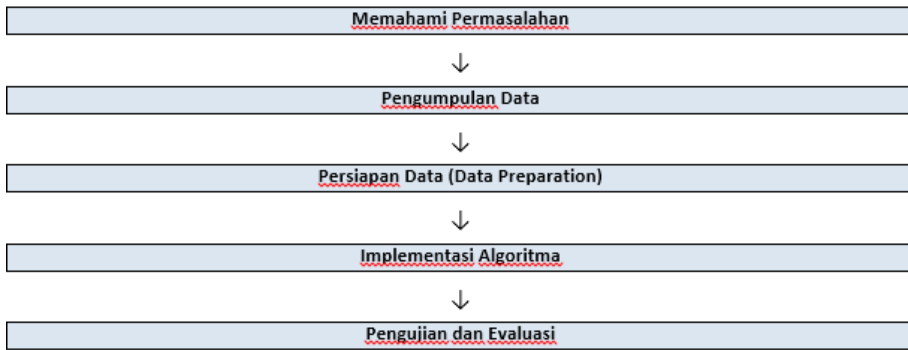
Pendahuluan Universitas Bina Darma Palembang merupakan salah satu perguruan tinggi swasta terkemuka di Indonesia, yang pada tahun 2025 masuk dalam 50 besar pemeringkatan Times Higher Education (THE) dari 4.300 perguruan tinggi di Indonesia dan menempati peringkat ke-46 secara nasional [tambahkan referensi]. Di balik pencapaian tersebut, pengarsipan dokumen skripsi pada bidang tata usaha masih banyak mengandalkan penyimpanan fisik, sehingga penelusuran judul skripsi oleh mahasiswa maupun staf menjadi semakin lambat seiring bertambahnya volume dokumen dari tahun ke tahun (Pratama *et al.*, 2021). Algoritma Brute Force dan Rabin-Karp merupakan dua pendekatan string matching yang umum digunakan untuk mengatasi permasalahan tersebut. Brute Force bekerja dengan membandingkan karakter demi karakter antara pola dan teks sumber secara berurutan (Danuri, 2016), sedangkan Rabin-Karp memanfaatkan fungsi hash untuk menyaring kecocokan sebelum melakukan verifikasi karakter, sehingga berpotensi lebih efisien pada pola yang panjang meskipun memiliki tahapan pemrosesan yang lebih kompleks (Yusuf *et al.*, 2019).

Studi terdahulu telah menerapkan berbagai algoritma string matching pada domain pencarian dokumen akademik, di antaranya Islamiyati dan Fikri (2022) yang menggunakan Knuth-Morris-Pratt untuk mendeteksi kemiripan judul skripsi, serta Zaen (2021) yang menerapkan Rabin-Karp untuk pencocokan judul skripsi. Rajagukguk dan Girsang (2015) menerapkan Brute Force pada sistem informasi pemilihan judul skripsi berbasis web, sementara Alan (2021) dan Andriansyah *et al.* (2021) masing-masing menerapkan Brute Force pada pencarian katalog buku perpustakaan dan pencarian berita berbasis web, yang keduanya melaporkan bahwa algoritma tersebut mampu memberikan hasil pencarian dalam waktu singkat pada dataset berskala menengah. Di sisi lain, penerapan Rabin-Karp banyak dijumpai pada domain deteksi kemiripan teks, seperti pada penelitian Ilham dan Mirza (2020) untuk pengarsipan dokumen sekolah, Pohan *et al.* (2022) untuk pencarian sinonim kata, serta Zidan dan Setiawan (2025) untuk pendeteksian plagiarisme pada dokumen makalah mahasiswa.

Meskipun kedua algoritma telah banyak diterapkan secara terpisah, perbandingan performa empiris antara Brute Force dan Rabin-Karp secara khusus pada karakteristik dataset judul skripsi Universitas Bina Darma—yang umumnya berupa frasa pendek dengan variasi kosakata akademik yang tinggi—belum banyak dilakukan. Sebagian besar penelitian terdahulu juga cenderung mengevaluasi satu algoritma secara mandiri tanpa perbandingan langsung pada himpunan data dan lingkungan eksekusi yang identik, sehingga kesimpulan mengenai algoritma mana yang lebih unggul untuk domain pencarian judul skripsi belum dapat ditarik secara meyakinkan dari literatur yang ada. Oleh karena itu, penelitian ini bertujuan mengukur dan membandingkan waktu eksekusi (running time) serta jumlah operasi perbandingan karakter dari kedua algoritma tersebut pada berbagai skala volume data dan skenario pengujian, sebagai dasar rekomendasi algoritma yang paling optimal untuk diterapkan pada sistem pencarian judul skripsi di Universitas Bina Darma.

2. Metodologi

Metodologi Penelitian ini menggunakan metode eksperimental kuantitatif untuk mengukur, mencatat, dan membandingkan waktu eksekusi serta jumlah operasi perbandingan karakter dari algoritma Brute Force dan Rabin-Karp secara objektif. Seluruh tahapan penelitian dilaksanakan di lingkungan kampus Universitas Bina Darma Palembang selama semester ganjil Tahun Akademik 2025/2026, menggunakan dataset sekunder berupa 21.666 judul skripsi unik yang diekstraksi dari repositori kampus. Tahapan penelitian disusun mengikuti alur logis yang terdiri atas lima tahap utama sebagaimana disajikan pada Gambar 1.



Gambar 1. Alur Penelitian

Tahap awal dimulai dengan memahami permasalahan melalui identifikasi inefisiensi pencarian judul skripsi secara manual serta perumusan kebutuhan pengukuran performa algoritma secara terkontrol. Pengumpulan data dilakukan dengan mengambil data sekunder berupa daftar judul skripsi dari repositori Universitas Bina Darma dalam format CSV. Selanjutnya, tahap persiapan data meliputi pembersihan data (data cleansing), penyeragaman huruf kapital, dan konversi ke format JSON, yang kemudian dikelompokkan ke dalam empat skala volume pengujian: 5.000, 10.000, 15.000, dan 21.666 data guna mengamati tren perubahan waktu eksekusi seiring bertambahnya beban data. Pada tahap implementasi, logika Brute Force dan Rabin-Karp diterjemahkan ke dalam bahasa JavaScript yang dieksekusi pada lingkungan Node.js tanpa antarmuka grafis agar komputasi logika murni terisolasi dari proses lain. Terakhir, pengujian dan evaluasi dilakukan melalui tiga skenario—variasi volume data, skenario terburuk (worst-case), dan pengukuran reduksi operasi perbandingan karakter—dengan durasi komputasi direkam menggunakan fungsi presisi tinggi `process.hrtime.bigint()` dalam satuan milidetik (ms). Setiap skenario dijalankan secara berulang pada masing-masing skala data untuk memastikan stabilitas hasil sebelum dirata-ratakan dan disajikan dalam bentuk tabel serta grafik komparatif. Algoritma Brute Force mencocokkan pola P terhadap teks T dengan membandingkan karakter secara berurutan dari kiri ke kanan pada setiap pergeseran posisi; apabila terjadi ketidakcocokan, pola digeser satu karakter ke kanan dan proses perbandingan diulang (Munir & Lidya, 2016). Prinsip kerja tersebut dirangkum pada Pseudocode 1.

```

ALGORITMA
BruteForceSearch(T, P)
n ← panjang(T); m ←
panjang(P)   untuk i ←
0 sampai (n - m):
cocok ← benar
untuk j ← 0 sampai (m
- 1):       jika
T[i+j] ≠ P[j] maka
            cocok ← salah; keluar dari perulangan
jika cocok maka
            kembalikan "Ditemukan pada posisi i"
kembalikan "Tidak ditemukan"
  
```

Algoritma Brute Force

Algoritma Rabin-Karp menghitung nilai hash dari pola dan dari setiap jendela substring teks berukuran sama menggunakan basis alfabet d dan modulus bilangan prima q [tambahkan data: nilai d dan q]. Verifikasi karakter penuh hanya dilakukan apabila nilai hash pola dan substring bersesuaian, sehingga sebagian besar posisi yang tidak relevan dapat dilewati tanpa perbandingan karakter langsung (Purba & Sitorus, 2018). Prinsip kerja tersebut disajikan pada Pseudocode 2.

```

ALGORITMA RabinKarpSearch(T, P,
d, q)  n ← panjang(T); m ←
panjang(P)  hashP ← Hash(P);
hashT ← Hash(T[0..m-1])  untuk
i ← 0 sampai (n - m):      jika
hashP = hashT maka
jika T[i..i+m-1] = P maka
    kembalikan "Ditemukan pada posisi i"
jika i < (n - m) maka
    hashT ← RollingHash(hashT, T[i], T[i+m], d, q)
    kembalikan "Tidak ditemukan"

```

Algoritma Rabin-Karp

Secara kompleksitas komputasi, Brute Force memiliki kompleksitas waktu $O(n \cdot m)$ pada kasus terburuk karena setiap dari $n - m + 1$ posisi pergeseran berpotensi memerlukan hingga m kali perbandingan karakter. Rabin-Karp secara teoretis memiliki kompleksitas rata-rata $O(n + m)$ berkat penyaringan berbasis nilai hash, namun tetap berpotensi mendekati $O(n \cdot m)$ pada kasus terburuk apabila terjadi tabrakan nilai hash (hash collision) secara berulang (Purba & Sitorus, 2018). Ringkasan perbandingan kompleksitas teoretis kedua algoritma disajikan pada Tabel 1.

Tabel 1. Perbandingan Kompleksitas Teoretis Algoritma

Algoritma	Kompleksitas Rata-Rata	Kompleksitas Kasus Terburuk
Brute Force	$O(n \cdot m)O(n \cdot m)$	$O(n \cdot m)O(n \cdot m)$
Rabin-Karp	$O(n + m)O(n + m)$	$O(n \cdot m)O(n \cdot m)$

Lingkungan pengujian menggunakan runtime Node.js pada satu unit komputer lokal [tambahkan data: tipe CPU/RAM] tanpa antarmuka grafis, dengan seluruh proses I/O berkas dataset diselesaikan secara sinkron sebelum pengukuran waktu dimulai, sehingga durasi yang tercatat murni merepresentasikan waktu komputasi algoritma dan bukan waktu pembacaan berkas. Pengukuran waktu menggunakan fungsi presisi tinggi `process.hrtime.bigint()` yang memiliki resolusi nanodetik sebelum dikonversi ke satuan milidetik (ms) untuk pelaporan. Dataset judul skripsi yang digunakan memiliki karakteristik teks berupa frasa akademik dengan panjang bervariasi, umumnya berkisar antara 5 hingga 20 kata per judul, serta mencakup kombinasi huruf kapital, tanda baca, dan istilah teknis lintas program studi. Karakteristik ini menjadikan dataset representatif untuk menguji performa algoritma pencarian eksak pada domain teks akademik berskala menengah hingga besar, yang berbeda dari karakteristik teks naratif panjang seperti dokumen jurnal atau artikel berita.

3. Hasil

Hasil Data uji yang digunakan dalam penelitian ini adalah 21.666 judul skripsi mahasiswa Universitas Bina Dharma dari berbagai program studi, yang telah melalui proses pembersihan dan disimpan dalam format JSON. Kedua algoritma—Brute Force dan Rabin-Karp—diujikan pada himpunan data yang identik untuk membandingkan kinerjanya secara objektif, guna menentukan algoritma yang paling optimal untuk diterapkan pada sistem pencarian judul skripsi. Pengujian dijalankan dengan memasukkan kata kunci pencarian tema penelitian skripsi, seperti frasa "SISTEM INFORMASI" sebagai salah satu parameter uji. Untuk mempermudah visualisasi pengujian, sistem dilengkapi antarmuka web sebagaimana ditunjukkan pada Gambar 2, yang menyediakan kolom masukan kata kunci dan pemicu komparasi, di mana komputasi pencarian inti tetap diisolasi pada runtime Node.js.



Gambar 2. Antarmuka Menu Utama Sistem Pencarian Judul Skripsi

Setelah pengujian dieksekusi, sistem menampilkan waktu eksekusi total dan jumlah operasi perbandingan karakter untuk masing-masing algoritma pada setiap skala volume data, sebagaimana diilustrasikan pada Gambar 3. Rekaman hasil ini menegaskan bahwa seluruh fungsi pencocokan string, penghitungan komparasi, dan perekaman durasi komputasi telah berjalan dengan baik pada setiap skenario pengujian.

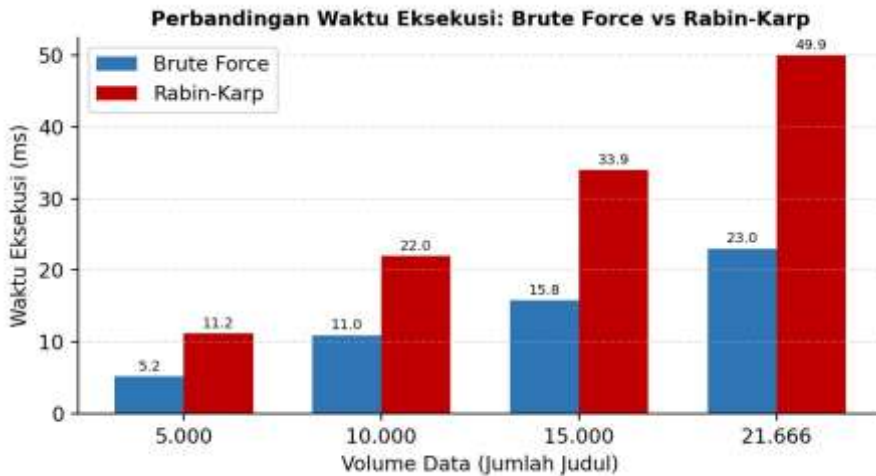
Volume Data / Judul	Nama Algoritma	Waktu / Durasi Total	Total Perbandingan Karakter	Status Pencarian
5.000	Brute Force	5,2113 ms	576.440 kali operasi	Ditemukan
	Rabin-Karp	11,1762 ms	7.499 kali operasi	Ditemukan
10.000	Brute Force	10,9623 ms	1.136.631 kali operasi	Ditemukan
	Rabin-Karp	21,9645 ms	13.529 kali operasi	Ditemukan
15.000	Brute Force	15,8147 ms	1.805.234 kali operasi	Ditemukan
	Rabin-Karp	33,9364 ms	22.059 kali operasi	Ditemukan
21.666	Brute Force	23,0269 ms	2.536.332 kali operasi	Ditemukan
	Rabin-Karp	49,9421 ms	36.002 kali operasi	Ditemukan

Gambar 3. Antarmuka Hasil Pengujian pada Variasi Volume Data

Hasil pengujian pada variasi volume data disajikan pada Tabel 2 dan divisualisasikan secara komparatif pada Gambar 4. Algoritma Brute Force secara konsisten mencatatkan waktu eksekusi yang lebih rendah dibandingkan Rabin-Karp pada seluruh skala data, dengan selisih waktu yang semakin melebar seiring bertambahnya volume data yang diproses.

Tabel 2. Perbandingan Waktu Eksekusi pada Variasi Volume Data

Volume Data	Algoritma	Waktu Eksekusi	Total Perbandingan Karakter	Status
5.000	Brute Force	5,2113 ms	576.440 kali operasi	Ditemukan
5.000	Rabin-Karp	11,1762 ms	7.499 kali operasi	Ditemukan
10.000	Brute Force	10,9623 ms	1.136.631 kali operasi	Ditemukan
10.000	Rabin-Karp	21,9645 ms	13.529 kali operasi	Ditemukan
15.000	Brute Force	15,8147 ms	1.805.234 kali operasi	Ditemukan
15.000	Rabin-Karp	33,9364 ms	22.059 kali operasi	Ditemukan
21.666	Brute Force	23,0269 ms	2.536.332 kali operasi	Ditemukan
21.666	Rabin-Karp	49,9421 ms	36.002 kali operasi	Ditemukan



Gambar 4. Grafik Perbandingan Waktu Eksekusi Brute Force vs Rabin-Karp

Pada volume data terkecil (5.000 judul), Brute Force menyelesaikan pencarian dalam 5,2113 ms, sedangkan Rabin-Karp membutuhkan waktu 11,1762 ms atau lebih dari dua kali lipat lebih lama. Pola keunggulan ini konsisten terjaga hingga volume data menengah (10.000 dan 15.000 judul), di mana Brute Force masing-masing mencatatkan waktu 10,9623 ms dan 15,8147 ms, sedangkan Rabin-Karp mencatatkan 21,9645 ms dan 33,9364 ms. Pada volume data maksimal (21.666 judul), Brute Force menuntaskan pencarian dalam 23,0269 ms, sekitar 2,17 kali lebih cepat dibandingkan Rabin-Karp yang membutuhkan 49,9421 ms. Konsistensi keunggulan tersebut diukur lebih lanjut melalui penghitungan rasio percepatan (*speedup ratio*) Rabin-Karp terhadap Brute Force pada tiap skala data, sebagaimana disajikan pada Tabel 3. Nilai rasio yang stabil pada rentang 2,0–2,2 kali menegaskan bahwa keunggulan kecepatan Brute Force bersifat sistematis di seluruh rentang skala data.

Tabel 3. Rasio Percepatan Brute Force terhadap Rabin-Karp

Volume Data	Brute Force (ms)	Rabin-Karp (ms)	Rasio Percepatan
5.000	5,2113	11,1762	2,14×
10.000	10,9623	21,9645	2,00×
15.000	15,8147	33,9364	2,15×
21.666	23,0269	49,9421	2,17×

Berdasarkan data pada Tabel 3, rasio percepatan (*speedup ratio*) Brute Force terhadap Rabin-Karp berada pada rentang yang sangat konsisten, yakni 2,00× hingga 2,17× di seluruh variasi volume data. Stabilitas rasio ini menunjukkan bahwa keunggulan kecepatan Brute Force bersifat linear dan deterministik terhadap penambahan ukuran data, serta membuktikan bahwa faktor *overhead* komputasi nilai *hash* pada Rabin-Karp membebani waktu eksekusi secara proporsional sejak skala data terkecil hingga volume maksimal. Konsistensi tersebut sekaligus mengonfirmasi bahwa kesederhanaan logika pemindaian Brute Force jauh lebih efisien untuk dieksekusi oleh mesin JavaScript modern pada teks pendek hingga menengah, di mana optimasi tingkat rendah (*low-level string indexing*) bekerja lebih dominan daripada reduksi matematis berbasis nilai *hash*. Guna memastikan bahwa dominasi Brute Force ini bukan dipicu oleh letak kemunculan pola yang menguntungkan (*early match*) pada skenario pencarian normal, diperlukan pengujian ketahanan performa pada kondisi ekstrem. Pengujian ini dirancang melalui skenario terburuk (*worst-case scenario*) menggunakan kata kunci acak yang dipastikan tidak terdapat sama sekali pada dataset, sehingga seluruh 21.666 data judul skripsi harus dipindai secara menyeluruh dari karakter pertama hingga terakhir tanpa adanya penghentian awal (*early break*). Antarmuka hasil pengujian skenario ini ditunjukkan pada Gambar 5 dengan status pencarian “Tidak Ditemukan”, dan ringkasan numeriknya disajikan pada Tabel 4.



Algoritma	Waktu Eksekusi	Total Perbandingan Karakter	Status
Brute Force	20,0970 ms	2.505.939 kali operasi	Tidak Ditemukan
Rabin-Karp	42,4836 ms	24.967 kali operasi	Tidak Ditemukan

Penyaringan awal berbasis nilai hash pada Rabin-Karp berhasil mereduksi operasi perbandingan karakter sebesar 5,14% pada pengujian kata kunci tunggal sebelum verifikasi karakter penuh dilakukan (Putra & Sularno, 2019). Namun, reduksi operasi ini tidak serta-merta mempercepat waktu eksekusi secara keseluruhan. Evaluasi konsistensi reduksi operasi pada volume data yang lebih besar disajikan pada Tabel 6. yang menunjukkan efisiensi stabil di atas 98% pada seluruh skala data.

Algoritma	Total Huruf Dibandingkan	Reduksi Operasi
Brute Force	49.684 kali operasi	0% (Pencarian Linier)
Rabin-Karp	47.130 kali operasi	5,14%

Volume Data	Operasi Brute Force	Operasi Rabin-Karp	Reduksi Operasi
5.000	576.440	7.499	98,7%
10.000	1.136.631	13.529	98,8%
15.000	1.805.234	22.059	98,8%
21.666	2.536.332	36.002	98,6%

Hasil pada Tabel 6 membuktikan efektivitas mekanisme penyaringan (*filtering*) nilai *rolling hash* pada algoritma Rabin-Karp, yang secara konsisten mampu memangkas lebih dari 98,6% operasi perbandingan karakter langsung dibandingkan pendekatan Brute Force pada setiap skala data. Kendati demikian, efisiensi reduksi karakter yang sangat tinggi ini menegaskan adanya anomali performa praktis; penurunan drastis jumlah perbandingan karakter tidak serta-merta menghasilkan waktu eksekusi yang lebih cepat akibat tingginya biaya komputasi aritmetika modular pada setiap pergeseran jendela teks.

4. Pembahasan

Pembahasan Hasil pengujian menunjukkan adanya fenomena *trade-off* yang konsisten antara efisiensi jumlah operasi perbandingan karakter dan kecepatan waktu eksekusi riil. Pada volume data maksimal (21.666 judul), Brute Force mengeksekusi 2.536.332 kali operasi perbandingan karakter, jauh melampaui Rabin-Karp yang hanya memerlukan 36.002 kali operasi; namun secara waktu eksekusi, Brute Force tetap 2,17 kali lebih cepat. Pola serupa teramati pada skenario terburuk (*worst-case scenario*), di mana Rabin-Karp mencatatkan reduksi operasi hingga 99,0% (24.967 berbanding 2.505.939 operasi), tetapi durasi pencariannya tetap lebih dari dua kali lipat lebih lambat dibandingkan Brute Force. Fenomena ini mengonfirmasi bahwa beban komputasi tambahan (*overhead*) pada Rabin-Karp tidak bersumber dari proses perbandingan karakter itu sendiri, melainkan dari kalkulasi nilai *rolling hash* berbasis operasi perkalian dan aritmetika modular yang dieksekusi berulang pada setiap pergeseran jendela teks (Purba & Sitorus, 2018). Pada lingkungan *runtime* Node.js, beban aritmetika tersebut secara empiris memerlukan siklus pemrosesan CPU yang lebih tinggi dibandingkan perbandingan karakter langsung antarnilai string pada Brute Force. Terlebih lagi, mesin JavaScript V8 menerapkan optimasi kompilasi Just-In-Time (JIT) yang sangat agresif terhadap perulangan sederhana (*simple loops*) dan manipulasi string primitif. Akibatnya, logika pencarian Brute Force dapat langsung diterjemahkan menjadi instruksi mesin tingkat rendah (*native instructions*) yang sangat efisien, mengungguli komputasi hashing dinamis pada Rabin-Karp. Temuan empiris ini sejalan dengan hasil Danuri (2016) yang melaporkan bahwa algoritma Brute Force mampu memproses pencarian teks berbasis konten dengan waktu rata-rata yang singkat pada skala dokumen pendek hingga menengah. Hasil ini juga memperkuat observasi Yusuf dkk. (2019) bahwa Rabin-Karp melalui tahapan komputasi awal yang lebih panjang sehingga membutuhkan waktu eksekusi lebih lama, kendati unggul secara teoretis dalam meminimalkan komparasi karakter. Dengan demikian, evaluasi performa algoritma *string matching* pada sistem berbasis perangkat lunak modern tidak dapat hanya bertumpu pada kompleksitas asimtotik teoretis, melainkan wajib memperhitungkan karakteristik lingkungan eksekusi (*execution runtime*), panjang teks, dan pola pencarian.

Secara praktis, penelitian ini membuktikan bahwa efisiensi teoretis dari sisi kompleksitas komparasi tidak selalu berbanding lurus dengan kecepatan eksekusi riil. Pada korpus judul skripsi yang memiliki karakteristik teks relatif pendek dan pola pencarian ringkas, kesederhanaan struktur Brute Force terbukti jauh lebih optimal. Sebaliknya, keunggulan Rabin-Karp dalam memangkas operasi perbandingan berpotensi memberikan manfaat nyata pada korpus berskala dokumen panjang (*full-text documents*) atau pada skenario pencarian banyak pola secara simultan (*multiple pattern matching*), seperti pada sistem deteksi plagiarisme karya ilmiah (Zidan & Setiawan, 2025). Dari sisi implikasi teknis dan rekayasa perangkat lunak, hasil penelitian ini merekomendasikan algoritma Brute Force sebagai pendekatan utama dalam pembangunan sistem pencarian judul tugas akhir atau skripsi berskala puluhan ribu data. Brute Force menawarkan implementasi kode yang ringkas, performa responsif yang stabil di bawah 25 ms untuk seluruh dataset, bebas dari kebutuhan konfigurasi parameter *hashing* (seperti penentuan bilangan basis dan modulus prima), serta terhindar dari degradasi performa akibat tabrakan nilai *hash* (*hash collision*). Kendati demikian, penelitian ini memiliki beberapa keterbatasan metodologis. Pengujian performa dilakukan pada satu konfigurasi perangkat keras lokal dengan *runtime* Node.js (mesin V8), sehingga durasi waktu eksekusi absolut dapat bervariasi apabila diuji pada lingkungan komputasi terdistribusi, perangkat keras server yang berbeda, atau bahasa pemrograman tingkat rendah seperti C/C++ dan Rust. Selain itu, parameter kata kunci yang diuji masih terbatas pada variasi frasa representatif tertentu; pengujian dengan rentang variasi panjang pola yang lebih ekstrem berpotensi memperluas spektrum evaluasi titik impas (*break-even point*) performa kedua algoritma. Sebagai sintesis menyeluruh terhadap seluruh skenario pengujian, Tabel 7 merangkum perbandingan lima metrik evaluasi utama antara algoritma Brute Force dan Rabin-Karp. Rangkuman ini menegaskan bahwa keunggulan Brute Force terpusat secara konsisten pada metrik kecepatan waktu eksekusi riil, sedangkan Rabin-Karp unggul mutlak dalam efisiensi perbandingan karakter.

Tabel 7. Ringkasan Perbandingan Keseluruhan Kedua Algoritma

Metrik	Brute Force	Rabin-Karp
Rata-rata waktu eksekusi (4 skala data)	13,75 ms	29,25 ms
Waktu eksekusi skenario terburuk	20,0970 ms	42,4836 ms

Total operasi pada volume maksimal	2.536.332 kali	36.002 kali
Reduksi operasi rata-rata terhadap Brute Force	0% (<i>baseline</i>)	$\approx 98,7\%$
Kompleksitas kasus terburuk	$O(n \cdot m)O(n \cdot m)$	$O(n \cdot m)O(n \cdot m)$

Sintesis metrik pada Tabel 7 secara gamblang mempertegas batas pemisah antara efisiensi komputasi teoretis dan performa sistem secara riil. Meskipun Rabin-Karp membuktikan keunggulan algoritmiknya dengan mereduksi rata-rata 98,7% operasi perbandingan karakter dan menekan jumlah pemindaian hingga puluhan ribu kali operasi saja, keunggulan tersebut tereduksi oleh tingginya beban komputasi fungsi *rolling hash* di tingkat instruksi mesin. Sebaliknya, Brute Force membukukan rata-rata waktu eksekusi yang dua kali lebih cepat (13,75 ms berbanding 29,25 ms) serta ketahanan waktu yang superior pada skenario terburuk. Hal ini menegaskan bahwa untuk domain pencarian berbasis teks pendek pada arsitektur perangkat lunak modern, efisiensi eksekusi instruksi primitif jauh lebih menentukan responsivitas sistem dibandingkan sekadar meminimalkan kuantitas komparasi karakter.

5. Kesimpulan

Berdasarkan hasil pengujian komputasi dan analisis empiris yang telah dilakukan, dapat disimpulkan tiga hal utama. Pertama, algoritma Brute Force secara konsisten lebih unggul dalam kecepatan waktu eksekusi dibandingkan Rabin-Karp pada seluruh skala volume data yang diuji, dengan rasio percepatan (*speedup ratio*) stabil pada rentang $2,00\times$ hingga $2,17\times$ serta performa terbaik 2,17 kali lebih cepat pada volume data maksimal, yakni 23,0269 ms berbanding 49,9421 ms. Kedua, ketahanan Brute Force juga terbukti superior pada skenario terburuk (*worst-case scenario*), dengan waktu eksekusi 20,0970 ms berbanding 42,4836 ms milik Rabin-Karp ketika pola pencarian tidak ditemukan pada seluruh korpus data. Ketiga, meskipun Rabin-Karp secara konsisten unggul mutlak dalam mereduksi jumlah operasi perbandingan karakter hingga di atas 98,6% pada pengujian bertahap dan 99,0% pada skenario terburuk, keunggulan teoretis tersebut tidak berkontribusi positif terhadap kecepatan waktu eksekusi riil karena tergantikan oleh tingginya beban komputasi (*overhead*) dari kalkulasi fungsi *rolling hash* modular pada lingkungan *runtime* Node.js. Dengan demikian, algoritma Brute Force disimpulkan sebagai algoritma yang paling optimal, efisien, dan andal untuk diterapkan pada karakteristik dataset judul skripsi di Universitas Bina Darma, sedangkan Rabin-Karp kurang direkomendasikan pada kasus ini karena kompleksitas fungsi *hash* justru memperlambat waktu respons sistem secara keseluruhan. Saran bagi pengembangan sistem repositori digital kampus adalah mengimplementasikan algoritma Brute Force pada sisi *backend* untuk mempercepat penelusuran judul skripsi tanpa membebani latensi server maupun memerlukan penyesuaian parameter *hashing* yang rumit. Sementara itu, bagi penelitian akademik lanjutan, disarankan untuk menguji algoritma pencocokan pola tingkat lanjut lainnya, seperti Knuth-Morris-Pratt (KMP), Boyer-Moore, atau algoritma hibrida seperti Zhu-Takaoka dan Raita pada korpus dokumen teks berskala penuh (*full-text*). Pengujian lanjutan ini penting untuk mengidentifikasi titik temu di mana efisiensi reduksi operasi karakter mulai mendominasi efisiensi waktu pemrosesan, sekaligus menguji konsistensi performa kedua algoritma pada variasi arsitektur perangkat keras serta lingkungan eksekusi bahasa pemrograman tingkat rendah yang berbeda.

Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Ahmad Haidar Mirza selaku pembimbing atas bimbingan, arahan, dan masukan kritis yang diberikan selama penelitian ini berlangsung, serta kepada civitas akademika Program Studi Teknik Informatika, Fakultas Sains Teknologi, Universitas Bina Darma Palembang atas dukungan fasilitas dan data repositori yang digunakan dalam penelitian ini.

Referensi

- Al Riansyah, A., & Mirza, A. H. (2023). Pendeteksi mobil berdasarkan merek dan tipe menggunakan algoritma YOLO. *SMATIKA: STIKI Informatika Jurnal*, 13(2), 241–250.
- Andriansyah, S., Baidarus, B., & Gunawan, R. (2021). Implementasi algoritma Brute Force pada pencarian berita berbasis web. *Jurnal Computer Science and Information Technology (CoSciTech)*, 2(2), 127–134.
- Alan, S. H. (2021). Implementasi algoritma Brute Force dalam pencarian data katalog buku perpustakaan. *Jurnal Teknik Informatika dan Sistem Informasi (JURTISI)*, 1(1), 1–8.
- Danuri. (2016). Pencarian file teks berbasis content dengan pencocokan string menggunakan algoritma Brute Force. *Scientific Journal of Informatics*, 3(1), 77–86. <https://doi.org/10.15294/sji.v3i1.6111>
- Ilham, M., & Mirza, A. H. (2020). Penerapan algoritma Knuth Morris Pratt dalam fitur pencarian pengarsipan dokumen pada SMA Plus Negeri 17 Palembang. *Journal of Software Engineering Ampera*, 1(2), 85–94.
- Islamiyati, D. C., & Fikri, A. (2022). Implementasi algoritma Knuth-Morris-Pratt dalam mendeteksi tingkat kemiripan judul skripsi berbasis web. *Journal of Information System Research (JOSH)*, 3(2), 58–62.
- Munir, R., & Lidya, L. (2016). *Algoritma dan pemrograman dalam bahasa Pascal, C, dan C++*. Informatika.
- Pohan, M. O., Mesran, M., & Sianturi, R. D. (2022). Penerapan algoritma Rabin-Karp pada pencarian sinonim kata. *Journal of Informatics Management and Information Technology (JIMIT)*, 2(1), 32–37.
- Pratama, N. P. N., Mustaqiem, M., & Minarni, M. (2021). Pengembangan sistem informasi pengelolaan judul skripsi dan tugas akhir dengan fitur deteksi kemiripan menggunakan algoritma Winnowing. *TIN: Terapan Informatika Nusantara*, 2(5), 271–278.
- Purba, A. H., & Sitorus, L. (2018). Perancangan aplikasi untuk menghitung persentase kemiripan proposal dan isi skripsi dengan algoritma Rabin-Karp. *Jurnal Teknik Informatika Unika St. Thomas (JTIUST)*, 3(1), 1–7.
- Putra, N. D., & Sularno, S. (2019). Penerapan algoritma Rabin-Karp dengan pendekatan synonym recognition sebagai antisipasi plagiarisme pada penulisan skripsi. *Jurnal Teknologi dan Sistem Informasi Bisnis*, 1(2), 88–97.
- Rajagukguk, R., & Girsang, V. L. Y. (2015). Sistem informasi pemilihan judul skripsi dengan metode string matching Brute Force berbasis web di Fikom Umi. *Journal Methodika*, 1(1), 16–23. <https://doi.org/10.46880/mtk.v1i1.16>
- Siahaan, R. F. (2020). *Desain dan analisis algoritma*. CV. Penerbit Lakeisha.
- Universitas Bina Darma. (2021). *Panduan penulisan skripsi Program Studi Sistem Informasi*. Fakultas Ilmu Komputer, Universitas Bina Darma.
- Yusuf, B., Vivianie, S., Marsya, J. M., & Sofyan, Z. (2019). Analisis perbandingan algoritma Rabin-Karp dan Ratcliff/Obershelp untuk menghitung kesamaan teks dalam bahasa Indonesia. *Prosiding Seminar Nasional APTIKOM (SEMNASTIK)*, 381–388.

- Zaen, A. A. M. (2021). *String matching menggunakan algoritma Rabin Karp untuk pencocokan judul skripsi mahasiswa* [Skripsi/Naskah Publikasi, Universitas AMIKOM Yogyakarta]. Repositori Universitas AMIKOM Yogyakarta.
- Zidan, M. R., & Setiawan, K. (2025). Implementasi algoritma Rabin-Karp dalam pendeteksian plagiarisme pada dokumen makalah mahasiswa. *Jurnal Indonesia: Manajemen Informatika dan Komunikasi (JIMIK)*, 6(1), 112–122.