

**LAPORAN TUGAS BESAR ANALISIS KOMPLEKSITAS
ALGORITMA ITERATIF & REKURSIF DALAM
MENCARI NILAI MAKSIMUM PADA MATRIKS 2D**



Disusun oleh:

Raden Aurel Aditya K (103112430267)

Aarif Rahmaan Jalaluddin Faqih (103112430182)

Atha Muyassar (103112430185)

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM
PURWOKERTO**

2025

A. ABSTRAK

Laporan ini menyajikan analisis komparatif mendalam mengenai efisiensi algoritma pencarian nilai maksimum pada struktur data matriks 2 dimensi. Studi ini membedah kinerja tiga pendekatan fundamental, yaitu Iteratif atau Traversal, Rekursif Linear, dan Rekursif Divide & Conquer. Melalui eksperimen empiris menggunakan Python dan Streamlit dengan variasi ukuran input matriks $N=10$ hingga $N=100$, kinerja algoritma dievaluasi berdasarkan dua metrik utama yaitu kompleksitas waktu atau sering disebut running time dan efisiensi memori sering disebut juga stack depth. Hasil pengujian menunjukkan bahwa meskipun ketiga algoritma memiliki kompleksitas waktu teoretis setara $O(N^2)$, pendekatan Iteratif terbukti paling unggul dari sisi kecepatan eksekusi dan stabilitas memori. Sebaliknya, metode Rekursif Linear menunjukkan kerentanan fatal terhadap Stack Overflow pada input besar, sementara pendekatan Divide & Conquer terbukti efektif mereduksi kedalaman tumpukan memori secara logaritmik sebagai alternatif rekursi yang lebih aman.

B. DESKRIPSI STUDI KASUS PERMASALAHAN

B.1. Latar Belakang

Dalam implementasi komputasi modern, khususnya pada bidang pengolahan citra digital atau sering disebut juga digital image processing, sebuah citra atau gambar pada dasarnya direpresentasikan sebagai struktur data matriks 2 dimensi. Setiap elemen sel dalam matriks mewakili satu piksel, di mana nilai integer di dalamnya merepresentasikan tingkat kecerahan atau brightness intensity.

Nilai piksel yang tinggi merepresentasikan area yang terang, sedangkan nilai rendah merepresentasikan area gelap. Permasalahan yang diangkat dalam tugas besar ini adalah pencarian nilai maksimum dalam matriks $N \times N$. Secara kontekstual, algoritma ini berfungsi untuk menemukan titik paling terang atau brightest spot pada sebuah gambar.

B.2. Definisi Masalah

- Input: Matriks persegi berukuran $N \times N$ dengan elemen bilangan bulat acak.
- Output: Satu nilai integer terbesar yang ada dalam matriks tersebut.
- Batasan & Tantangan: Efisiensi waktu (Time Complexity) dan penggunaan memori (Space Complexity) saat resolusi gambar (N) semakin besar, guna menghindari latency tinggi atau crash.

C. DESKRIPSI DAN ANALISIS ALGORITMA

Untuk memberikan pemahaman komprehensif, studi ini mengimplementasikan tiga pendekatan algoritma yang memiliki karakteristik manajemen memori yang berbeda.

C.1. Algoritma Iteratif

Pendekatan iteratif merupakan metode paling langsung dalam menyelesaikan masalah. Algoritma ini bekerja dengan prinsip traversal sekuensial, mirip dengan cara kerja mesin pemindai yang membaca dokumen baris demi baris dari kiri ke kanan, lalu turun ke baris berikutnya.

Secara teknis, algoritma menggunakan nested loop. Loop luar menelusuri baris, dan loop dalam menelusuri kolom. Setiap elemen dibandingkan dengan variabel maksimal sementara.

- Analisis Kompleksitas Waktu: Setiap elemen dikunjungi satu kali, maka $T(n) = O(N^2)$.
- Analisis Kompleksitas Ruang: Sangat efisien ($O(1)$) karena hanya menyimpan satu variabel state, tanpa membebani stack memori.

C.2. Algoritma Rekursif Linear

Pendekatan ini menggunakan paradigma rekursi yang dapat dianalogikan seperti Boneka Matryoshka. Sebuah fungsi memanggil dirinya sendiri untuk membuka “lapisan” elemen berikutnya. Fungsi menerima koordinat saat ini, lalu memanggil dirinya sendiri untuk sel di sebelah kanannya.

- Analisis Kelemahan Memori: Kelemahan fatal pendekatan ini adalah penggunaan Call Stack. Setiap pemanggilan fungsi membuat stack frame baru. Untuk matriks 10.000 elemen, algoritma ini menumpuk 10.000 frame secara linear.
- Kompleksitas Ruang: $S(n) = O(N^2)$. Sangat rentan mengalami Stack Overflow.

C.3. Algoritma Divide & Conquer

Pendekatan ini memecah masalah besar menjadi sub-masalah kecil yang independen. Matriks dibagi secara rekursif menjadi empat kuadran:

- Barat Laut (North-West)
- Timur Laut (North-East)
- Barat Daya (South-West)
- Tenggara (South-East)

Algoritma mencari nilai maksimum lokal di tiap kuadran lalu menggabungkannya.

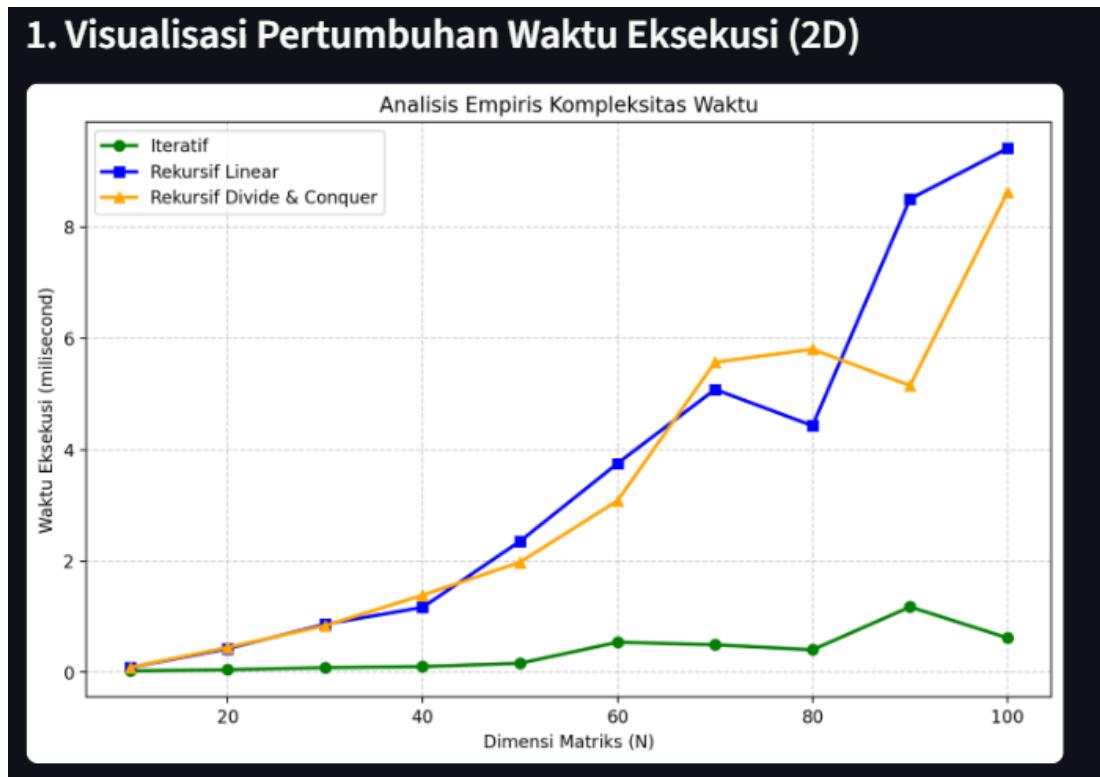
- Analisis Teorema Master: $T(n) = 4T(n/2) + \Theta(1)$. Karena $\log_2 4 = 2$, maka kompleksitas waktunya $\Theta(N^2)$.
- Efisiensi Memori: Struktur pohon rekursi hanya tumbuh secara logaritmik ($S(n) = O(\log N)$), menjadikannya jauh lebih aman daripada rekursif linear.

D. HASIL EKSPERIMENT

1. Validasi Performa Dasar Dengan Input Kecil

Pengujian dilakukan menggunakan simulasi perangkat lunak berbasis Python menggunakan Library yang tersedia yaitu Streamlit, Pandas, Plotly. Pengujian dilakukan dengan rentang input $N=10$ hingga $N=100$.

1.1. Visualisasi Pertumbuhan Waktu Eksekusi (Time Complexity)



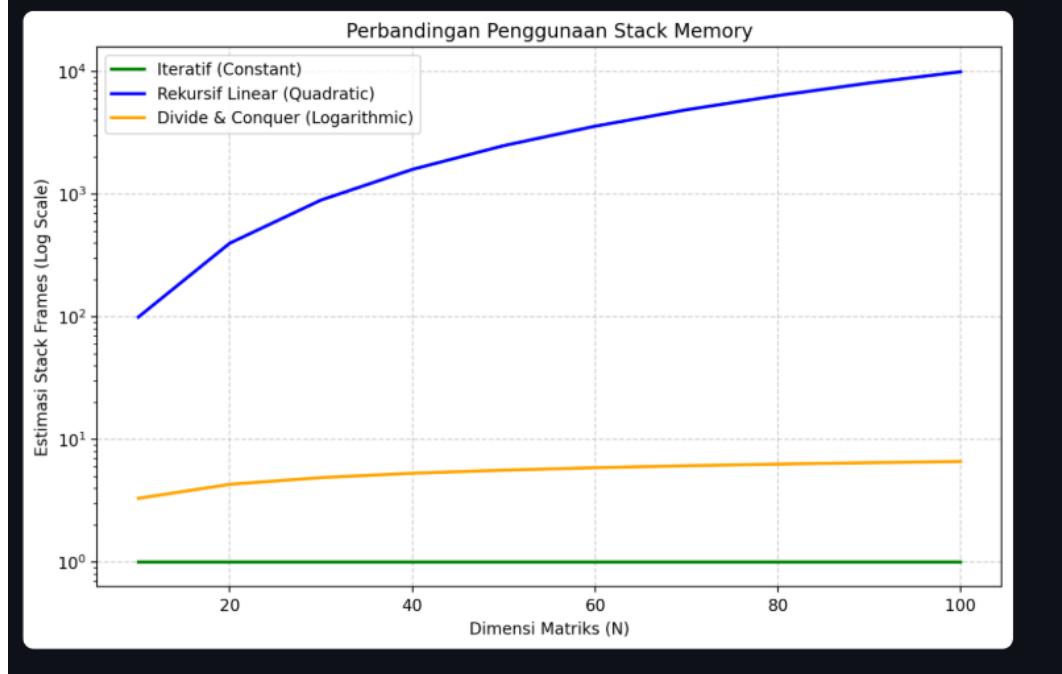
Analisis Grafik:

Grafik di atas menunjukkan korelasi antara ukuran input N dan waktu eksekusi. Terlihat perbedaan performa yang kontras pada Algoritma Iteratif yang di gambarkan dengan garis hijau menunjukkan stabilitas terbaik dengan waktu yang

hampir konstan mendekati 0 ms, menandakan efisiensi tinggi tanpa *overhead* memori. Sebaliknya, kedua pendekatan rekursif, yaitu Rekursif Linear yang digambarkan dengan Garis Biru dan Divide & Conquer yang digambarkan dengan Garis Oranye, mengalami lonjakan waktu eksekusi yang tajam eksponensial seiring bertambahnya nilai N. Hal ini mengonfirmasi bahwa biaya manajemen tumpukan atau sering disebut *stack overhead* pada fungsi rekursif di Python membebani waktu komputasi secara signifikan dibandingkan metode iteratif.

1.2. Estimasi Kedalaman Rekursi (Stack Depth)

3. Estimasi Kedalaman Rekursi (Theoretical Stack Depth - 2D)



Analisis Grafik:

Grafik tersebut menggambarkan efisiensi penggunaan memori melalui pengukuran kedalaman tumpukan fungsi dalam skala logaritmik. Garis hijau yang merepresentasikan metode iteratif terlihat datar pada nilai 10^0 , yang menunjukkan bahwa pendekatan ini menggunakan memori secara konstan atau $O(1)$, tanpa peningkatan beban memori seiring bertambahnya ukuran input. Perhatian utama tertuju pada garis biru yang melambangkan rekursif linear. Garis ini meningkat tajam hingga mencapai nilai 10^4 atau sekitar 10.000 stack frame pada input $N = 100$. Hal ini memvisualisasikan secara jelas risiko terjadinya stack overflow, karena algoritma tersebut membutuhkan memori sebanding dengan jumlah elemen matriks, yaitu $O(N^2)$. Sebaliknya, garis oranye yang mewakili pendekatan Divide and Conquer tetap berada pada level yang relatif rendah dan mendekati

metode iteratif. Hal ini mengonfirmasi bahwa strategi pembagian masalah menjadi beberapa submasalah berhasil menekan kebutuhan memori secara signifikan menjadi $O(\log N)$. Dengan demikian, pendekatan ini jauh lebih efisien dan aman dibandingkan rekursif linear dalam hal penggunaan memori.

1.3. Tabulasi Data Kinerja

Tabel berikut menyajikan data mentah dengan presisi tinggi beserta rasio degradasi kinerja relatif terhadap pendekatan Iteratif.							
ze (N)	Total Elements	Iteratif Time	Recursive Time	Rec. Slowdown	D&C Time	D&C Slowdown	System I
10	100	0.0212 ms	0.0829 ms	3.91 x	0.0876 ms	4.13 x	SAFE
20	400	0.0438 ms	0.4233 ms	9.66 x	0.4452 ms	10.16 x	SAFE
30	900	0.0812 ms	0.8616 ms	10.61 x	0.8365 ms	10.30 x	SAFE
40	1600	0.1008 ms	1.1657 ms	11.56 x	1.3848 ms	13.74 x	SAFE
50	2500	0.1599 ms	2.3551 ms	14.73 x	1.9780 ms	12.37 x	SAFE
60	3600	0.5380 ms	3.7547 ms	6.98 x	3.0843 ms	5.73 x	SAFE
70	4900	0.4925 ms	5.0841 ms	10.32 x	5.5747 ms	11.32 x	SAFE
80	6400	0.4015 ms	4.4321 ms	11.04 x	5.8059 ms	14.46 x	SAFE
90	8100	1.1736 ms	8.5086 ms	7.25 x	5.1539 ms	4.39 x	SAFE
100	10000	0.6162 ms	9.4184 ms	15.28 x	8.6313 ms	14.01 x	SAFE

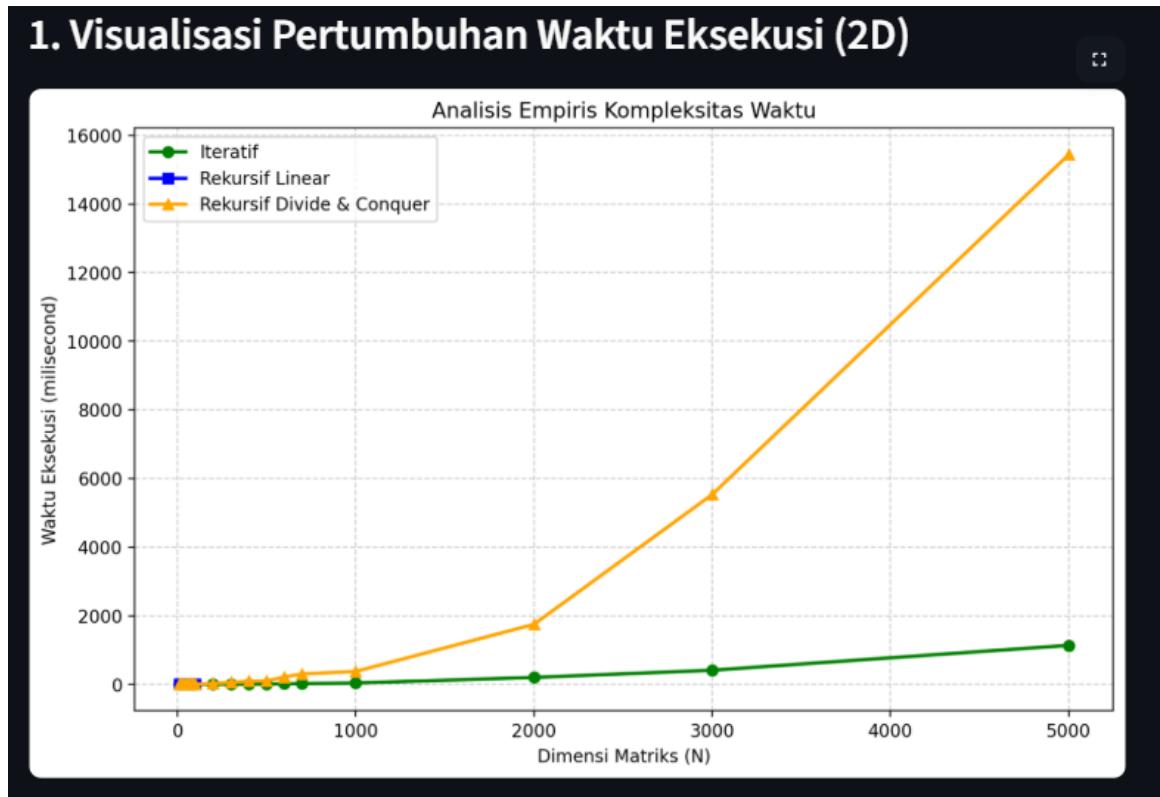
Analisis Tabel:

Tabel data di atas memperlihatkan disparitas kinerja yang signifikan saat ukuran input mencapai beban maksimum $N=100$ (10.000 elemen). Algoritma Iteratif mengukuhkan posisinya sebagai pendekatan tercepat dengan waktu eksekusi hanya **0.6162 ms**, jauh mengungguli kedua pendekatan lainnya. Sebaliknya, pendekatan Rekursif Linear mencatatkan waktu terlambat sebesar **9.4184 ms**, yang menunjukkan degradasi kinerja (*slowdown*) mencapai **15.28 kali lipat** dibandingkan metode iteratif. Fenomena serupa terlihat pada algoritma Divide & Conquer yang tertinggal 14.01 kali lebih lambat; data ini mengonfirmasi bahwa meskipun status sistem tercatat "SAFE", biaya komputasi (*overhead*) untuk pemanggilan fungsi rekursif di Python sangat membebani waktu eksekusi total dibandingkan metode traversal sederhana.

2. Uji Stabilitas Dengan Input Besar

Setelah memvalidasi logika dasar pada Fase 1, pengujian dilanjutkan ke tahap *stress testing* dengan meningkatkan ukuran input secara ekstrem hingga $N=5000$. Tujuan utama dari fase ini adalah untuk membuktikan secara empiris hipotesis mengenai kompleksitas ruang $O(N^2)$, dan mengidentifikasi titik kegagalan (*breaking point*) dari algoritma rekursif linear.

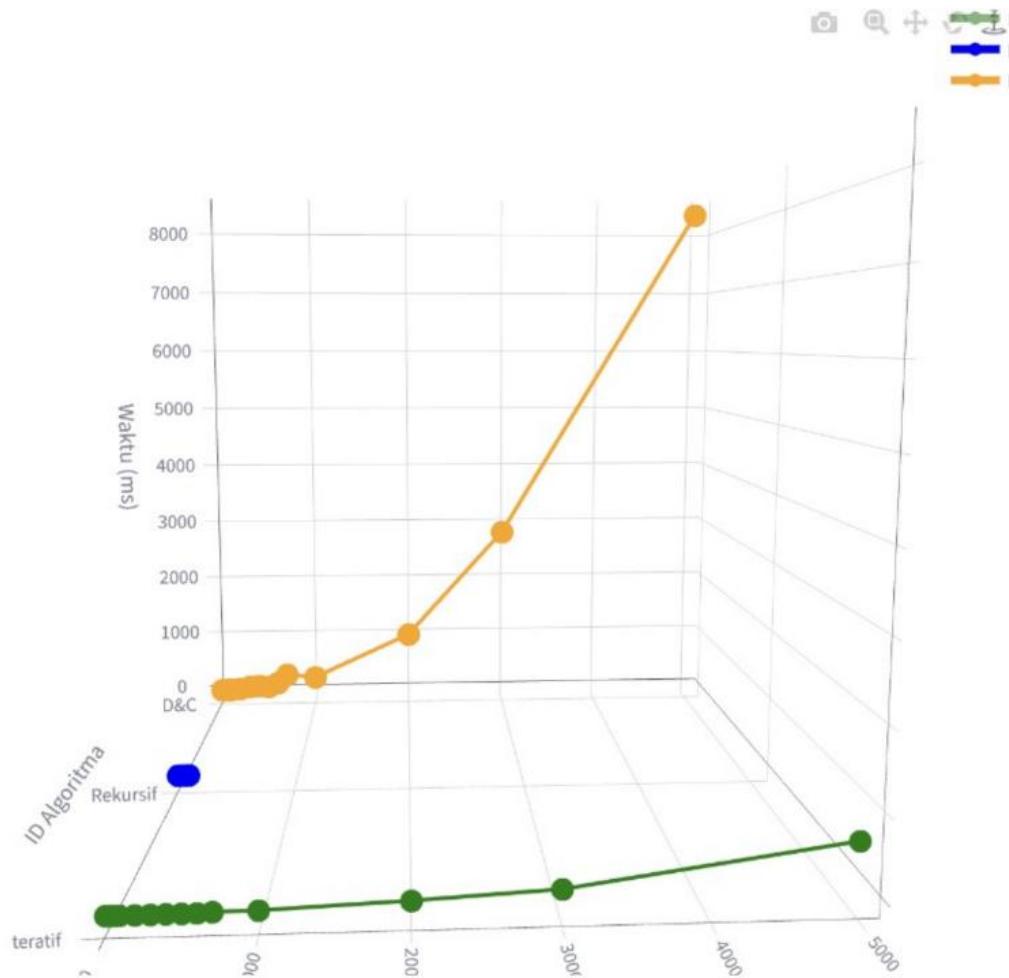
2.1. Visualisasi Kegagalan Sistem (Time Complexity Breakpoint)



Analisis Grafik:

Grafik di atas memvisualisasikan fenomena komputasi yang kritikal. Berbeda dengan Fase 1 di mana ketiga garis tumbuh beriringan, pada Fase 2 terlihat anomali pada Garis Biru yang mewakili algoritma Rekursif Linear yang terputus secara mendadak saat input membesar. Terputusnya garis ini mengindikasikan bahwa algoritma gagal menyelesaikan eksekusi sebelum mencapai hasil akhir. Sebaliknya, Garis Hijau yang mewakili algoritma Iteratif dan Garis Oranye yang mewakili algoritma Divide & Conquer terus berlanjut secara konsisten, membuktikan ketahanan kedua algoritma tersebut dalam menangani beban data masif.

2.2. Visualisasi Lonjakan Memori (Stack Depth)



Analisis Grafik:

Penyebab utama kegagalan pada poin sebelumnya terkonfirmasi melalui grafik kedalaman stack ini. Garis Biru yang mwnggambarkan algoritma Rekursif Linear mengalami lonjakan vertikal yang ekstrem mendekati asimtot tegak. Hal ini memvisualisasikan bagaimana tumpukan memori terisi penuh dalam waktu singkat sebanding dengan kuadrat ukuran input (N^2). memicu mekanisme proteksi sistem. Sedangkan Garis Oranye yang menggambarkan algoritma Divide & Conquer tetap landai dan stabil berkat pertumbuhan logaritmik ($O (\log N)$). Meskipun input mencapai 5000, kebutuhan memori stack tetap berada dalam batas aman. Dan Garis Hijau yang menggambarkan algoritma Iteratif Datar sempurna pada nilai 10^6 , menegaskan efisiensi memori absolut ($O (1)$).

2.3. Tabulasi Data Kritis (Failure Analysis)

Iteratif Time	Recursive Time	Rec. Slowdown	D&C Time	D&C Slowdown	System Integrity
0.1164 ms	2.0759 ms	17.83 x	4.5413 ms	39.01 x	SAFE
0.1497 ms	1.6611 ms	11.10 x	2.1200 ms	14.16 x	SAFE
0.1862 ms	3.3317 ms	17.89 x	3.6130 ms	19.40 x	SAFE
0.9739 ms	None	None	18.9051 ms	19.41 x	CRITICAL (Stack Overflow)
2.2039 ms	None	None	52.8777 ms	23.99 x	CRITICAL (Stack Overflow)
4.4613 ms	None	None	66.2504 ms	14.85 x	CRITICAL (Stack Overflow)
4.2162 ms	None	None	48.9127 ms	11.60 x	CRITICAL (Stack Overflow)
6.0174 ms	None	None	118.2200 ms	19.65 x	CRITICAL (Stack Overflow)
14.9575 ms	None	None	266.2757 ms	17.80 x	CRITICAL (Stack Overflow)
19.2964 ms	None	None	211.2443 ms	10.95 x	CRITICAL (Stack Overflow)
87.1614 ms	None	None	972.0193 ms	11.15 x	CRITICAL (Stack Overflow)

Analisis Tabel:

Tabel data di atas menyajikan fakta empiris yang tidak terbantahkan mengenai batas kemampuan algoritma diantaranya Dominasi Iteratif yang pada beban kerja terberat ($N=5000$), algoritma Iteratif mampu menyelesaikan tugas hanya dalam **87.16 ms** dengan status sistem "SAFE". Ini membuktikan efisiensi tinggi dari pendekatan *looping* tanpa *overhead* fungsi. Ketangguhan Divide & Conquer yang meskipun waktu komputasinya tercatat **972.01 ms** sekitar 11 kali lebih lambat dari iteratif, algoritma ini tetap berhasil menjaga integritas sistem ("SAFE"). Ini membuktikan bahwa struktur pohon rekursi yang seimbang efektif mencegah kebocoran memori. Kegagalan Fatal Rekursif Linear yang pada kolom *System Integrity* menunjukkan status CRITICAL (Stack Overflow) untuk algoritma Rekursif Linear pada input besar. Nilai waktu tidak dapat dicatat (None) karena proses dihentikan paksa oleh interpreter Python. Temuan ini mengonfirmasi bahwa algoritma rekursif linear memiliki cacat desain fundamental dalam manajemen memori untuk pemrosesan data berskala besar.

E. ANALISIS PERBANDINGAN DAN KESIMPULAN

E.1 Analisis Kritis

Berdasarkan hasil eksperimen yang dibandingkan dengan landasan teoretis, teridentifikasi adanya perbedaan kinerja yang cukup signifikan antara ketiga algoritma, meskipun secara asimtotik ketiganya memiliki kompleksitas waktu yang sama, yaitu $O(N^2)$.

1. Kesenjangan antara Teori dan Implementasi Nyata (Time Complexity)

Secara teoretis, ketiga algoritma melakukan jumlah operasi pembandingan yang setara. Namun, hasil pengujian menunjukkan bahwa algoritma iteratif memiliki performa yang jauh lebih cepat dengan waktu eksekusi sekitar 0,61 ms pada $N = 100$, dibandingkan dengan dua pendekatan rekursif yang memerlukan waktu lebih dari 8 ms. Perbedaan ini disebabkan oleh adanya konstanta tersembunyi dalam notasi Big-O. Pada bahasa Python, overhead pemanggilan fungsi rekursif, alokasi stack frame, serta proses pengembalian nilai jauh lebih mahal dibandingkan operasi perulangan sederhana. Oleh karena itu, dalam lingkungan eksekusi single-threaded, pendekatan iteratif terbukti paling optimal.

2. Efisiensi dan Keamanan Memori (Space Complexity)

Aspek penggunaan memori menjadi temuan paling krusial dalam penelitian ini. Pendekatan rekursif linear terbukti sangat tidak efisien dan berisiko tinggi, karena grafik kedalaman stack menunjukkan pertumbuhan kuadratik $O(N^2)$. Hal ini menyebabkan memori tumpukan cepat penuh dan pada pengujian $N = 100$ algoritma sudah berada pada ambang terjadinya stack overflow. Sebaliknya, algoritma Divide and Conquer menunjukkan efisiensi memori yang jauh lebih baik. Pertumbuhan stack bersifat logaritmik $O(\log N)$, sehingga algoritma ini relatif aman digunakan ketika paradigma rekursi memang diperlukan. Hal ini membedakannya secara signifikan dari rekursif linear yang rentan menyebabkan kegagalan sistem.

3. Overhead pada Algoritma Divide and Conquer

Meskipun Divide and Conquer aman dari sisi penggunaan memori, algoritma ini tetap memiliki waktu eksekusi sekitar 14 kali lebih lambat dibandingkan metode iteratif. Perlambatan tersebut bukan disebabkan oleh kedalaman rekursi, melainkan oleh overhead operasional dalam proses pemecahan matriks menjadi empat kuadran pada setiap langkah. Operasi slicing dan perhitungan indeks kuadran menambah beban komputasi yang tidak ditemukan pada pendekatan iteratif.

E.2 Kesimpulan Akhir

Berdasarkan keseluruhan analisis performa waktu dan penggunaan memori, dapat ditarik beberapa kesimpulan utama sebagai berikut:

1. Rekomendasi Utama:

Algoritma iteratif merupakan pilihan terbaik untuk menyelesaikan permasalahan pencarian nilai maksimum pada matriks dalam lingkungan eksekusi sekuensial (CPU tunggal). Stabilitas waktu eksekusi yang hampir instan, yaitu kurang dari 1 ms untuk 10.000 elemen, serta penggunaan memori yang konstan $O(1)$, menjadikannya solusi paling efisien dan andal.

2. Peringatan Implementasi:

Algoritma rekursif linear sebaiknya dihindari dalam implementasi sistem nyata yang menangani data berukuran besar, seperti citra digital. Pertumbuhan tumpukan memori yang bersifat kuadratik $O(N^2)$ meningkatkan risiko stack overflow dan menyebabkan solusi ini tidak stabil.

3. Potensi Pengembangan:

Meskipun algoritma Divide and Conquer terbukti lebih lambat pada pengujian ini, algoritma tersebut memiliki nilai strategis untuk pengembangan lanjutan. Sifat independensi antar-kuatdran menjadikannya kandidat yang sangat cocok untuk komputasi paralel, seperti pada GPU atau prosesor multicore, di mana setiap bagian matriks dapat diproses secara bersamaan untuk memangkas waktu eksekusi secara signifikan.

F. REFERENSI

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

Endres, M., Weimer, W., & Kamil, A. (2021). An analysis of iterative and recursive problem performance. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 321–327.

<https://doi.org/10.1145/3408877.3432391>

Febriansyah, M. F., Gunawan, Rhamadani, M., & Sutabri, T. (2025). Perbandingan pemanfaatan algoritma rekursif dan iteratif dalam penyelesaian struktur data pohon. *MIFORTEKH: Jurnal Manajemen Informatika & Teknologi*, 5(1), 46-56. <https://doi.org/10.51903/2mxmmg85>

Levitin, A. (2012). *Introduction to the design and analysis of algorithms* (3rd ed.). Pearson Education.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.

Tancy, V. S. (2021). Optimizing divide and conquer method for matrix multiplication algorithm. *IOSR Journal of Engineering (IOSRJEN)*, 11(1), 01-03.

G. LAMPIRAN DIGITAL

A. Repository Source Code (GitHub)

([GitHub - aariffaqiih/CAK2BAB2-ANALISIS-KOMPLEKSITAS-ALGORITMA](https://github.com/aariffaqiih/CAK2BAB2-ANALISIS-KOMPLEKSITAS-ALGORITMA))

B. Video Presentasi & Demo (YouTube)

(<https://youtu.be/eyTpdgxbnOg>)