Analisis Perbandingan Kompleksitas Waktu Algoritma Gauss, Gauss-Jordan, dan Kaidah Cramer dalam penyelesaian Sistem Persamaan Linier

Diero Arga Purnama - 13522056[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522056@itb.ac.id

***Abstract*—Dalam menyelesaikan Sistem Persamaan Linier (SPL), terdapat banyak metode yang dapat digunakan. Metode-metode tersebut meliputi: metode substitusi, metode eliminasi, metode eliminasi Gauss, metode eliminasi Gauss-Jordan, metode matrix balikan, kaidah Cramer, dll. Dari metode-metode yang ada, tentunya terdapat metode yang lebih baik dibandingkan dengan metode lainnya. Salah satu cara untuk membandingkan metode-metode tersebut adalah dengan membandingkan kompleksitas waktu masing-masing metode. Makalah ini akan metode yang akan dibandingkan adalah metode eliminasi Gauss, metode eliminasi Gauss-Jordan, dan kaidah Cramer. Pada makalah ini, SPL yang akan digunakan akan dibatasi pada SPL yang memiliki jumlah variabel dan jumlah persamaan yang sama, serta memiliki solusi yang unik dikarenakan kaidah Cramer hanya dapat digunakan pada SPL dengan batasan-batasan tersebut.**

***Keywords*—Kaidah Cramer, Kompleksitas Algoritma, Metode Gauss, SPL.**

I. PENDAHULUAN

Sistem Persamaan Linier atau SPL adalah kumpulan persamaan linier dengan variabel-variabel yang sama. Persamaan linier adalah persamaan aljabar yang mengandung variabel dengan pangkat sama dengan satu.

Dalam penyelesaian sebuah SPL, terdapat bermacam-macam metode yang dapat digunakan. Dalam makalah ini, metode yang akan dibahas meliputi metode eliminasi Gauss, metode eliminasi Gauss-Jordan, dan kaidah Cramer. Untuk mengetahui metode mana diantara tiga metode diatas yang paling pantas untuk digunakan dalam sebuah program, dapat dilakukan pengujian terhadadap efisiensi algoritma tiap-tiap metode tersebut.

Efisiensi algoritma dapat diukur menggunakan waktu dan ruang memori yang digunakan untuk menjalankan algoritma tersebut. Algoritma yang efisien merupakan algoritma yang meminimumkan waktu dan ruang memori yang dibutuhkan untuk menjalankannya.

Besaran yang dipakai untuk mengukur waktu dan ruang memori yang dibutuhkan oleh suatu algoritma disebut dengan kompleksitas algoritma. Terdapat dua macam kompleksitas algoritma, yaitu kompleksitas waktu atau *time complexity* dan kompleksitas ruang atau *space complexity*. Kompleksitas waktu merupakan jumlah tahapan komputasi yang harus dijalankan oleh sebuah algoritma, dan kompleksitas ruang adalah ruang memori yang digunakan oleh sebuah algoritma.

Dalam makalah ini, perbandingan antar metode-metode penyelesaian SPL akan dilakukan menggunakan perbandingan kompeksitas waktu masing-masing algoritma dikarenakan pada saat ini, memori komputer mrupakan persoalan yang kurang kritis jika dibandingkan dengan waktu.

II. LANDASAN TEORI

A. Sistem Persamaan Linier

Persamaan linier adalah persamaan aljabar yang mengandung variabel dengan pangkat sama dengan satu. Persamaan linier dengan variabel $x_1, x_2, \ldots, x_n$ dapat dituliskan dalam bentuk

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b,$$

dengan b dan koefisien $a_1, a_2, \ldots, a_n$ merupakan bilangan real atau bilangan kompleks dan n merupakan bilangan bulat positif.

Sistem Persamaan Linier (SPL) merupakan kumpulan dari satu atau lebih persamaan linier dengan variabel yang sama. [5]

Dalam makalah ini, SPL yang dibahas akan dibatasi pada SPL yang memiliki solusi yang unik dan memiliki jumlah persamaan yang sama dengan jumlah variabelnya. Pembatasan ini diterapkan karena kaidah Cramer hanya dapat digunakan untuk menyelesaikan SPL dengan ketentuan-ketentuan tersebut.

B. Kompleksitas Algoritma

Kompleksitas algoritma adalah besaran yang dipakai untuk menerangkan model abstrak pengukuran waktu atau ruang memori yang dibutuhkan oleh suatu algoritma yang independen dari pertimbangan komputer dan *compiler* apapun. Kompleksitas algoritma dibagi menjadi dua macam, yaitu kompleksitas waktu atau *time complexity* dan kompleksitas ruang atau *space complexity*. [1]

C. Kompleksitas Waktu

Kompleksitas waktu, sering dilambangkan dengan fungsi $T(n)$, adalah jumlah tahapan komputasi yang harus dijalankan

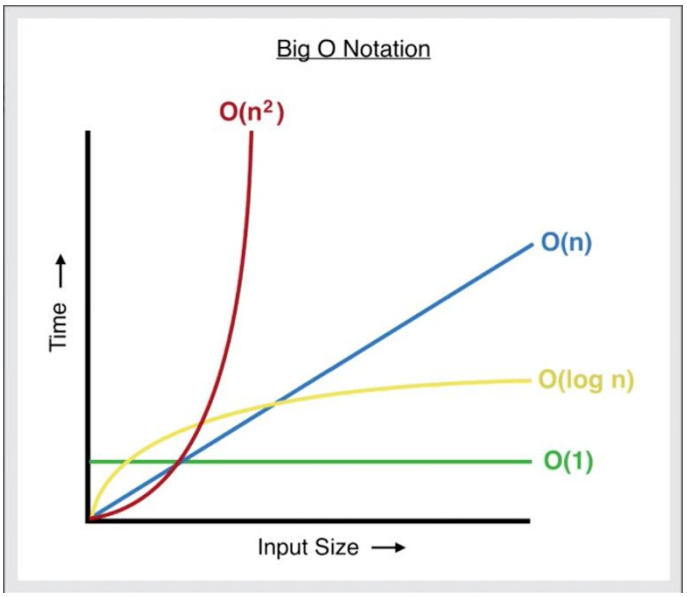
oleh sebuah algoritma dengan ukuran masukan sebesar *n*. Variabel *n* melambangkan jumlah data yang dimasukkan ke dalam suatu algoritma untuk diproses. [1] Kompleksitas waktu dapat digunakan untuk membandingkan algoritma untuk menentukan algoritma manakah yang lebih efisien. Dalam membandingkan algoritma dengan kompleksitas waktu, algoritma yang memiliki nilai kompleksitas waktu yang lebih kecil ialah algoritma yang lebih efisien.

Seringkali, kompleksitas waktu suatu algoritma dinyatakan dengan notasi "O-Besar" atau $O(n)$ yang merupakan notasi kompleksitas waktu asimptotik. Notasi "O-Besar" digunakan saat kita tidak membutuhkan ukuran kompleksitas waktu yang presisi. Notasi ini menunjukkan seberapa cepat kompleksitas waktu bertambah ketika ukuran masukan (n) membesar. Jika kompleksitas waktu suatu algoritma dinyatakan dengan $T(n)$, dan terdapat konstanta C yang memenuhi

$$T(n) \leq Cf(n)$$

untuk $n \geq n_0$, maka

$$T(n) = O(f(n)).$$



Gambar 1. Grafik kompleksitas waktu dalam notasi O-Besar

Sumber: https://medium.com/dataseries/how-to-calculate-time-complexity-with-big-o-notation-9afe33aa4c46

D. Metode Eliminasi Gauss

Metode eliminasi Gauss adalah metode yang dapat digunakan untuk menemukan solusi dari sebuah SPL dengan memanfaatkan matriks *augmented* dan Operasi Baris Elementer (OBE).

Langkah pertama dalam metode eliminasi Gauss adalah membuat matriks *augmented* dari SPL yang ingin diselesaikan. Matriks *augmented* merupakan gabungan dari matriks koefisien dan matriks konstanta sebuah SPL. Jika terdapat sebuah SPL

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m,$$

matriks *augmented* dari SPL tersebut dapat dinyatakan dengan

$$[a \mid b] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array}\right].$$

Setelah itu, gunakan OBE untuk mengubah matriks *augmented* hingga terbentuk sebuah matriks eselon baris. Terdapat tiga OBE yang dapat digunakan dalam sebuah matriks augmented, yaitu mengalikan sebuah baris dengan konstanta bukan nol, menukar dua baris, dan menambahkan sebuah baris dengan kelipatan baris lainnya.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_n \end{bmatrix} \sim\text{OBE}\sim \begin{bmatrix} 1 & * & * & \cdots & * & * \\ 0 & 1 & * & \cdots & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \vdots & 1 & * \end{bmatrix}$$

Gambar 2. Metode eliminasi Gauss

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-03-Sistem-Persamaan-Linier-2023.pdf

Setelah didapatkan matriks eselon baris, gunakan teknik *backward substitution* untuk mendapatkan solusi SPL.

E. Metode Eliminasi Gauss-Jordan

Metode Eliminasi Gauss-Jordan adalah metode yang dapat digunakan untuk menemukan solusi dari sebuah SPL yang merupakan pengembangan dari metode eliminasi Gauss.

Langkah pertama dalam metode ini sama seperti dengan metode eliminasi Gauss, yaitu membuat matriks *augmented* dari SPL yang ingin diselesaikan.

Setelah itu, gunakan OBE untuk mengubah matriks *augmented* hingga terbentuk matriks eselon baris tereduksi.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{bmatrix} \sim\text{OBE}\sim \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & * \\ 0 & 1 & 0 & \cdots & 0 & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \vdots & 1 & * \end{bmatrix}$$

Gambar 3. Metode eliminasi Gauss-Jordan

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-05-Sistem-Persamaan-Linier-2023.pdf

Setelah terbentuk matriks eselon baris tereduksi, nilai variabel akan didapatkan.

F. Kaidah Cramer

Kaidah Cramer adalah metode yang dapat digunakan untuk menemukan solusi dari sebuah SPL dengan menggunakan determinan matriks. Karena kaidah Cramer menggunakan determinan matriks untuk menemukan solusi dari SPL, metode ini hanya dapat digunakan untuk SPL yang memiliki *n* variabel dan *n* persamaan.

Tahap pertama adalah untuk membuat matriks koefisien dan

matriks konstanta dari sebuah SPL. Jika terdapat sebuah SPL

$$ax_1 + bx_2 + cx_3 = j$$
$$dx_1 + ex_2 + fx_3 = k$$
$$gx_1 + hx_2 + ix_3 = l,$$

maka matriks koefisien dan konstanta dari SPL tersebut adalah sebagai berikut

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} j \\ k \\ l \end{bmatrix}.$$

Setelah itu, hitung determinan dari matriks koefisien. Untuk mendapatkan solusi ke-n dari SPL, dapat digunakan rumus berikut

$$x_n = \frac{|A_n|}{|A|},$$

Dengan $|A_n|$ merupakan determinan dari matriks yang didapatkan dengan mengganti kolom ke-n dari matriks koefisien dengan matriks konstanta dan $|A|$ merupakan determinan dari matriks koefisien.

III. Analisis Kompleksitas Waktu

A. Metode Eliminasi Gauss

1. Mengubah matriks *augmented* menjadi matriks eselon baris

```python
def metodeEliminasiGauss(spl):
    rows, cols = len(spl), rows + 1

    for row in range(rows):
        elemenDiagonal = spl[row][row]
        if elemenDiagonal == 0:
            for nextRow in range(row + 1, rows):
                if spl[nextRow][row] != 0:
                    temp = spl[nextRow]
                    spl[nextRow] = spl[row]
                    spl[row] = temp
                    elemenDiagonal = spl[row][row]

        for col in range(cols):
            spl[row][col] /= elemenDiagonal

        for nextRow in range(row + 1, rows):
            faktor = spl[nextRow][row] / spl[row][row]
            for col in range(cols):
                spl[nextRow][col] -= spl[row][col] * faktor
    return spl
```

Gambar 4. Fungsi untuk mengubah matriks *augmented* menjadi matriks eselon baris

Sumber: Dokumen penulis

Tahap pertama dalam metode eliminasi Gauss adalah mengubah sebuah SPL menjadi bentuk matriks *augmented*. Fungsi ini menerima masukan matriks *augmented berukuran* $n \times (n + 1)$ dan mengubah matriks tersebut menjadi matriks eselon baris.

Proses ini dilakukan melalui sebuah *loop* yang mengandung tiga *loop* lain didalamnya. *Loop* luar berfungsi untuk mengulangi operasi didalamnya sebanyak n kali. Loop dalam pertama digunakan untuk menukar barisan dalam matriks, jumlah maksimum operasi yang dapat dilakukan adalah sebanyak $n - 1$ operasi. *Loop* dalam kedua digunakan untuk membagi semua barisan dengan elemen utama-nya, proses ini akan dilakukan sebanyak $n + 1$ kali. *Loop* dalam terakhir digunakan untuk mengurangi barisan bawah dengan barisan atas, jumlah operasi yang akan dilakukan adalah sebanyak $n^2 - 1$ kali. Sehingga didapatkan kompleksitas waktu total sebesar

$$T(n) = T_{luar} \times T_{dalam}$$
$$T(n) = n \times (n^2 + 2n - 1)$$
$$T(n) = n^3 + 2n^2 - n$$

atau dalam notasi O-besar sebesar $O(n^3)$.

2. Mengubah matriks eselon baris menjadi solusi SPL

```python
def EselonKeSolusi(matriks):
    rows, cols = len(matriks), rows + 1

    solusi = [0 for i in range(cols - 1)]
    nilaiX = {}

    for i in range(cols - 2, -1, -1):
        for row in range(rows - 1, i - 1, -1):
            for col in range(i + 1, cols):
                tanda = 1 if col == cols - 1 else -1
                elemen = matriks[row][col] if col == cols - 1 else matriks[row][col] * nilaiX[col]
                solusi[i] += tanda * elemen
            nilaiX[i] = solusi[i]
    return solusi
```

Gambar 5. Fungsi untuk mengubah matriks eselon baris menjadi *array* yang berisi solusi SPL

Sumber: Dokumen penulis

Fungsi ini menerima matriks eselon baris berukuran $n \times (n + 1)$ dan mengembalikan array berisi solusi SPL berukuran n. Proses ini dilakukan melalui tiga *nested loop berukuran* $O(n)$ sehingga didapatkan kompleksitas waktu dalam notasi O-besar sebesar $O(n^3)$.

3. Kompleksitas waktu total

Setelah mendapatkan kompleksitas waktu kedua fungsi dalam bentuk notasi O-besar, kompleksitas waktu total dari algoritma metode eliminasi Gauss dapat dihitung dengan menjumlahkan kompleksitas waktu kedua fungsi sebagai berikut

$$T_g(n) = O(n^3) + O(n^3)$$

sehingga diperoleh

$$T_g(n) = O(n^3).$$

B. Metode Eliminasi Gauss-Jordan

1. Mengubah matriks *augmented* menjadi matriks eselon baris tereduksi

```
def metodeEliminasiGaussJordan(spl):
    spl = metodeEliminasiGauss(spl)
    rows, cols = len(spl), rows + 1

    for row in range(rows - 1, 0, -1):
        for col in range(row - 1, -1, -1):
            faktor = spl[col][row] / spl[row][row]
            for i in range(col, cols):
                spl[col][i] -= faktor * spl[row][i]
    return spl
```

Gambar 6. Fungsi untuk mengubah matriks *augmented* menjadi matriks eselon baris tereduksi

Sumber: Dokumen penulis

Fungsi ini diawali dengan memanggil fungsi untuk mengubah matriks *augmented* menjadi matriks eselon baris yang telah dihitung memiliki kompleksitas waktu dalam notasi O-Besar sebesar $O(n^3)$. Setelah didapatkan matriks eselon baris, setiap baris akan dikurangi dengan seluruh barisan yang dibawahnya.

Proses ini dilakukan menggunakan tiga *nested* loop masing-masing berukuran $O(n)$, sehingga diperoleh kompleksitas waktu *loop* sebesar $O(n^3)$.

Didapatkan kompleksitas waktu fungsi sebagai berikut

$$T(n) = O(n^3) + O(n^3) = O(n^3)$$

2. Mengubah matriks eselon baris tereduksi menjadi solusi SPL

```
def EselonTereduksiKeSolusi(matriks):
    rows, cols = len(matriks), rows + 1

    solusi = [0 for i in range(cols - 1)]

    for i in range(cols - 1):
        solusi[i] = matriks[i][cols -  1]
    return solusi
```

Gambar 7. Fungsi untuk mengubah matriks eselon baris tereduksi menjadi *array* yang berisi solusi SPL

Sumber: Dokumen penulis

Karena matriks *augmented* sudah dalam bentuk matriks eselon barist tereduksi, tidak perlu dilakukan *backwards substitution* untuk mendapatkan solusi SPL. Fungsi ini hanya mengambil elemen pada kolom terakhir dan meletakkannya dalam *array* solusi.

Fungsi ini mengandung dua *loop*. *Loop* pertama berfungsi untuk menginisialisasi array dengan nilai 0, jumlah operasi yang akan dilakukan sebesar *n* operasi dengan *n* ukuran baris matriks. *Loop* kedua berfungsi untuk mengisi *array* solusi dengan solusi SPL, loop ini diulang sebanyak *n* kali.

Sehingga didapatkan kompleksitas waktu total dari fungsi sebesar $O(n)$.

3. Kompleksitas waktu total

Dengan menjumlahkan kompleksitas waktu kedua fungsi dalam bentuk notasi O-Besar, dapat dihitung kompleksitas waktu total dari algoritma metode eliminasi Gauss-Jordan sebagai berikut

$$T_{gj}(n) = O(n^3) + O(n)$$
$$T_{gj}(n) = O(n^3).$$

C. Kaidah Cramer

1. Mendapatkan kofaktor elemen sebuah matriks

```
def kofaktorMatriks(matriks, rowKof, colKof):
    rows, cols = len(matriks), rows

    kofaktor = [[0 for i in range(cols - 1)] for j in range(rows - 1)]
    cRow, cCol = 0, 0
    for row in range(rows):
        if row != rowKof:
            for col in range(cols):
                if col != colKof:
                    kofaktor[cRow][cCol] = matriks[row][col]
                    cCol += 1
            cRow += 1
            cCol = 0
    return kofaktor
```

Gambar 8. Fungsi untuk mendapatkan matriks kofaktor dari elemen sebuah matriks

Sumber: Dokumen penulis

Fungsi ini menerima masukan matriks berukuran $n \times n$, serta baris dan kolom elemen yang ingin dicari matriks kofaktor-nya. Fungsi ini terdiri dari dua bagian, inisialisasi matriks, dan pengisian matriks.

Kedua tahap dilakukan melalui dua *nested loop* yang berukuran $O(n)$ sehingga didapatkan kompleksitas waktu masing-masing tahap sebesar $O(n^2)$. Kompleksitas waktu total dari fungsi didapatkan dari penjumlahan kompleksitas waktu kedua tahap sebagai berikut

$$T(n) = O(n^2) + O(n^2) = O(n^2).$$

2. Menghitung determinan matriks

```
def determinanMatriks(matriks):
    rows, cols = len(matriks), rows
    if rows != cols:
        print("Matriks tidak memiliki determinan")
        return
    elif rows == 1:
        return matriks[0][0]
    elif rows == 2:
        return (matriks[0][0] * matriks[1][1] - matriks[0][1] * matriks[1][0])

    sum = 0
    for i in range(0, rows):
        tanda = 1 if i % 2 == 0 else -1
        sum += tanda * matriks[0][i] * determinanMatriks(kofaktorMatriks(matriks, 0, i))
    return sum
```

Gambar 9. Fungsi untuk mendapatkan determinan dari sebuah matriks

Sumber: Dokumen penulis

Fungsi ini digunakan untuk menghitung determinan sebuah matriks dengan menggunakan metode ekspansi kofaktor. Fungsi menerima masukan matriks sebesar $n \times n$. Jika matriks besar n

kurang dari 3, fungsi akan memiliki kompleksitas waktu $O(1)$, tetapi untuk matriks dengan ukuran $n \geq 3$, penghitungan determinan akan dilakukan secara rekursif.

Rekursi akan dilakukan sebanyak 1 kali untuk n = 3, 3 kali untuk n = 4, dan 12 kali untuk n = 5, sehingga diperoleh jumlah rekursi yang dilakukan sebesar $\frac{(n-1)!}{2}$ rekursi. Karena rekursi dilakukan didalam *loop*, kompleksitas waktu dari fungsi dapat dihitung dengan mengalikan kompleksitas waktu loop dengan kompleksitas waktu rekursi sebagai berikut

$$T(n) = \frac{n \times (n-1)!}{2}$$

atau dalam notasi O-Besar sebesar

$$T(n) = O(n!).$$

3. Hitung solusi SPL menggunakan kaidah Cramer

```
def kaidahCramer(spl):
    rows, cols = len(spl), rows + 1

    matriksA = [[0 for i in range(cols - 1)] for j in range(rows)]
    matriksB = [[0] for j in range(rows)]
    for row in range(rows):
        for col in range(cols - 1):
            matriksA[row][col] = spl[row][col]
    for row in range(rows):
        matriksB[row][0] = spl[row][cols - 1]

    determinanAwal = determinanMatriks(matriksA)
    solusi = [0 for i in range(rows)]
    for n in range(rows):
        matriksN = [[0 for i in range(cols - 1)] for j in range(rows)]
        for row in range(rows):
            for col in range(cols - 1):
                matriksN[row][col] = matriksA[row][col] if n != col else matriksB[row][0]
        solusi[n] = determinanMatriks(matriksN) / determinanAwal
    return solusi
```

Gambar 10. Fungsi untuk mencari solusi dari SPL menggunakan kaidah Cramer

Sumber: Dokumen penulis

Karena fungsi ini terdiri dari banyak *loop* kecil yang terpisah, untuk memudahkan penghitungan, perhitungan kompleksitas waktu akan difokuskan pada *loop* yang terbesar, yaitu *loop* terakhir.

Didalam *loop* terakhir, terdapat pemanggilan fungsi determinanMatriks yang memiliki kompleksitas waktu sebesar O(n!) sehingga didapatkan kompleksitas waktu dari algoritma kaidah Cramer sebesar

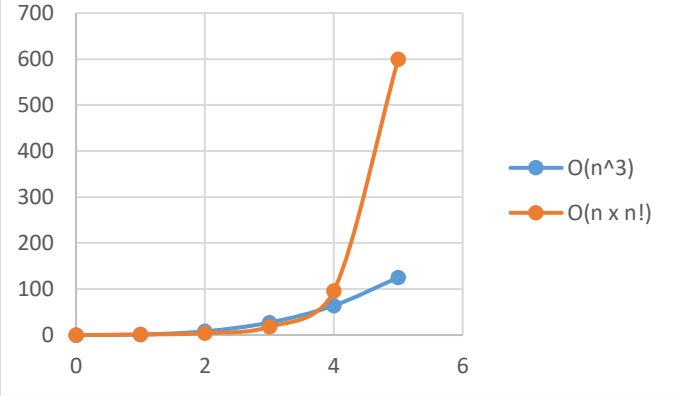
$$T_c(n) = O(n) \times O(n!)$$
$$T_c(n) = O(n \times n!).$$

D. Perbandingan Kompleksitas Waktu

No.	Algoritma	Kompleksitas Waktu
1	Metode Eliminasi Gauss	$O(n^3)$
2	Metode Eliminasi Gauss-Jordan	$O(n^3)$
3	Kaidah Cramer	$O(n \times n!)$

Tabel 1. Kompleksitas waktu masing-masing algoritma

Dapat dilihat dari tabel, algoritma metode eliminasi Gauss dan algoritma metode eliminasi Gauss-Jordan memiliki kompleksitas waktu yang sama, yaitu $O(n^3)$ sedangkan algoritma kaidah Cramer memiliki kompleksitas waktu yang jauh lebih besar yaitu $O(n \times n!)$.



Gambar 11. Grafik kompleksitas waktu

Sumber: Dokumen penulis

Berdasarkan gambar diatas, dapat dilihat bahwa kebutuhan waktu algoritma dengan kompleksitas waktu $O(n \times n!)$ akan bertumbuh jauh lebih cepat dibandingkan algoritma dengan kompleksitas waktu $O(n^3)$, sehingga didapatkan bahwa algoritma kaidah Cramer kurang efisien jika dibandingkan dengan algoritma lainnya untuk SPL yang berukuran besar tetapi lebih efisien untuk SPL yang berukuran sangat kecil.

IV. KESIMPULAN

Berdasarkan hasil analisis kompleksitas waktu ketiga algoritma, dalam mencari solusi sebuah Sistem Persamaan Linier (SPL) dengan jumlah variabel dan persamaan sebanyak *n*, didapatkan kompleksitas waktu dari algoritma metode eliminasi Gauss dan metode eliminasi Gauss-Jordan sebesar $O(n^3)$, dan kaidah Cramer sebesar $O(n \times n!)$.

Jika dalam sebuah program dibutuhkan algoritma untuk memecahkan sebuah SPL, Jika SPL memiliki $n \leq 3$, kaidah Cramer akan lebih cepat, sedangkan untuk SPL dengan $n > 3$, metode eliminasi Gauss dan metode eliminasi Gauss-Jordan akan memberikan hasil yang lebih baik.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan terimakasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. selaku dosen Matematika Diskrit kelas 01 untuk bimbingannya, dan bapak Dr. Ir. Rinaldi Munir, M.T. untuk website-nya yang memudahkan proses pencarian informasi mengenai topik yang dibahas dalam makalah ini. Penulis juga ingin mengucapkan terimakasih kepada teman-teman dan orang tua penulis yang telah memberi dukungan selama proses pembuatan makalah ini.

REFERENSI

[1] Munir, Rinaldi, 2023. "Kompleksitas Algoritma (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/25-Kompleksitas-Algoritma-Bagian2-2023.pdf (diakses pada 8 Desember 2023)

[2] Munir, Rinaldi, 2023. "Sistem Persamaan Linier (SPL)". https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-

2024/Algeo-03-Sistem-Persamaan-Linier-2023.pdf (diakses pada 8 Desember 2023)
[3] Munir, Rinaldi, 2023. “Sistem Persamaan Linier (SPL) Pokok bahasan: Metode Eliminasi Gauss-Jordan”. https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-05-Sistem-Persamaan-Linier-2023.pdf (diakses pada 8 Desember 2023)
[4] Munir, Rinaldi, 2023. “Determinan (Bagian 2)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-09-Determinan-bagian2-2023.pdf (diakses pada 8 Desember 2023)
[5] D. C. Lay, S. R. Lay, and J. Mcdonald, *Linear algebra and its applications*. Boston: Pearson, 2016.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2023

Diero Arga Purnama, 13522056