# Kompleksitas Algoritma dalam Menyelesaikan Sistem Persamaan Linier (C++)

Muh. Habibi Haidir / 13516085
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*muhhabibih@gmail.com, 13516085@std.stei.itb.ac.id*

***Abstract*—Persamaan linier adalah sebuah persamaan aljabar, yang tiap sukunya mengandung konstanta, atau perkalian konstanta dengan variabel tunggal. Dalam menyelesaikan persamaan linier tersebut dapat digunakan berbagai metode. Salah satu metode yang dapat digunakan ialah Gauss Jordan. Makalah ini membahas mengenai kompleksitas dari algoritma dalam menyelesaikan sistem persamaan linear menggunakan metode Gauss dan Gauss Jordan. Kemudian, makalah ini berfokus pada kompleksitas algoritma yang ditentukan dengan notasi Big-O.**

***Kata Kunci*—Big-O, Gauss-Jordan, Kompleksitas.**

## I. PENDAHULUAN

Dalam kehidupan sehari – hari, kita tidak terlepas dari permasalahan sistem persamaan linier. Sistem persamaan linier tersebut dapat berupa 2 variabel ataupun lebih. Contoh permasalahan sistem persamaan linier dalam kehidupan sehari – hari ialah menghitung uang belanja, memaksimalkan keuntungan dari suatu lahan ataupun suatu bisnis. Untuk variable yang masih sedikit (atau kurang dari 10) masih dapat diselesaikan dengan manual. Tetapi untuk variable yang sudah lumayan banyak (lebih dari 100) untuk diselesaikan dengan manual membutuhkan waktu yang lumayan lama. Untuk itu beberapa programmer membuat program untuk menyelesaikan system persamaan linier tersebut. Tetapi, membuat program saja tidak cukup. Ciri – ciri dari program yang baik ialah :

1. Mudah digunakan
2. Penampilan yang baik
3. Kompleksitas yang rendah
4. Reability
5. Mampu beradaptasi
6. Interopability
7. Mobility

Salah satu dari ciri – ciri program yang baik ialah kompleksitas yang rendah. Kita harus menganalisis kompleksitas dari algoritma program tersebut. Semakin baik program tersebut, maka semakin kecil kompleksitas algoritma program tersebut. Dalam makalah ini, akan dibahas mengenai kompleksitas dari salah satu metode penyelesaian system persamaan liner, yaitu Gauss – Jordan.

## II. METODE PENYELESAIAN SPL

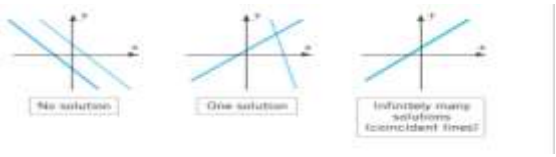
Dalam menyelesaikan sistem persamaan linier tersebut terdapat beberapa metode, yaitu :

1. Gauss
2. Gauss – Jordan
3. Elimination Method
4. Jacobi
5. Gauss Seidel

Dalam penyelesaian sistem linear tersebut terdapat 3 kemungkinan solusi, yaitu :

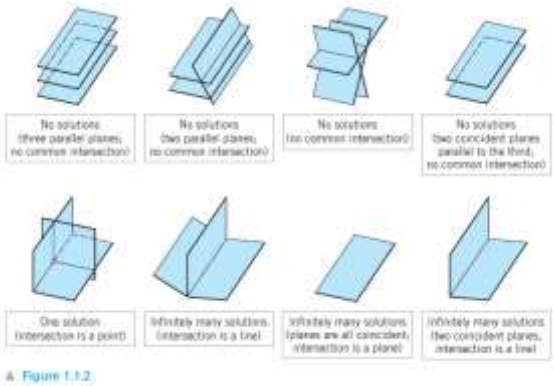
1. SPL memiliki solusi unik
2. SPL memiliki solusi tak terbatas
3. SPL tidak memiliki solusi

Berikut adalah contoh grafik dari sistem persamaan linier dua variabel :



Gambar 2.1 SPL 2 variabel

Berikut adalah contoh grafik dari sistem persamaan linier 3 variabel, yaitu :



Gambar 2.2 SPL 3 variabel

Metode yang akan dibahas untuk menyelesaikan sistem persamaan linier ialah Gauss-Jordan. Sebelum melangkah lebih lanjut kita perlu mengetahui operasi yang dapat dilakukan pada matriks yang berisi sejumlah persamaan linier. Pada sistem Operasi Baris Elementer (OBE) jelas kita dapat :

1. Mengalikan sebuah baris dengan billangan konstan yang tidak nol
2. Menukarkan dua buah baris pada matriks
3. Mejumlahkan satu baris dengan baris lainnya

Dengan menggunakan sifat tersebut kita berusaha untuk membuat sistem persamaan kita yang dituliskan dalam bentuk matriks kedalam bentuk :

1) Bentuk Baris Echelon
Pada sistem eliminasi Gauss kita hanya berpatokan membuat 0 semua bagian dari bawah dari diagonal utama dan membuat tiap elemen daridiagonal utama tepat 1/0.

$$\begin{bmatrix} 1 & 4 & -3 & 7 \\ 0 & 1 & 6 & 2 \\ 0 & 0 & 1 & 5 \end{bmatrix}$$

2) Bentuk Baris Echelon Tereduksi
Pada sistem eliminasi Gauss – Jordan kita focus untuk membuat menjadi 0 semua nilai dari elemen matriks yang tidak termasuk dari diagonal utama dan membiarkan nilai dari diagonal utama tetap 1.

$$\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

## III. KOMPLEKSITAS ALGORITMA

a) Algoritma

Algoritma adalah urutan logis langkah – langkah penyelesaian masalah secara sistematis. Algoritma banyak digunakan dalam ilmu matematika dan ilmu komputer untuk pemrosesan data, perhitungan, pengurutan data, maupun penalaran otomatis. Algoritma juga terdiri atas intruksi – intruksi yang telah terdefinisi, mulai dari konsisi awal, sampai pada kondisi akhir.

Ketika membuat sebuah algoritma program untuk menyelesaikan suatu masalah, yang harus dipertanyakan ialah : Ketika diberikan input maksimum, apakah program tersebut masih dapat menyelesaikan masalah tersebut dengan waktu yang singkat atau cepat?.

Sebuah masalah dapat mempunyai banyak algoritma penyelesaian. Contoh : masalah pengurutan (sorting), terdapat puluhan algoritma pengurutan yang sudah dibuat seperti *Bubble Sort, Quick Sort, Heap Sort, Selection Sort*, dll. Tadi telah dibahas beberapa ciri algoritma yang baik, dan salah satunya ialah algoritma yang memiliki kompleksitas yang rendah atau effisien.

Efisiensi dari algoritma diukur dalam satuan waktu (satuan s atau ms) dan ruang memori (satuan byte atau KB) yang dibutuhkan untuk menjalankan algoritma tersebut. Waktu dan ruang memori yang dibutuhkan oleh sebuah algoritma sangat bergantung pada jumlah data yang diproses. Banyak data yang diproses berbanding lurus dengan waktu dan ruang yang dibutuhkan oleh suatu algoritma.

Untuk menghitung kompleksitas suatu algoritma tidak bias hanya berpatok pada waktu mengeksekusi program untuk suatu komputer tertentu karena tiap arsitektur komputer yang berbeda menghasilkan waktu yang berbeda – beda pula dalam melakukan operasi – operasi dasar seperti penjumlahan, perkalian, perbandingan, dan operasi – operasi lainnya. Selain itu, algoritma yang diterjemahkan menjadi executable code sangat dipengaruhi oleh compiler yang digunakan. Compiler yang berbeda maka akan menghasilkan kode tingkat mesin yang berbeda pula.

Semakin sedikit waktu dan ruang yang dibutuhkan untuk menjalankan algoritma, maka algoritma tersebut semakin efisien pula. Kini, banyak programmer yang berlomba – lomba untuk mengembangkan algoritma yang semakin baik dan efisien, sehingga terdapat begitu banyak alternative dan variasi algoritma untuk suatu tujuan yang sama. Salh satu contohnya ialah pengurutan suatu data. Sudah terdapat puluhan algoritma yang bertujuan sama yaitu mengurutkan suatu data, alternative yang tersedia yaitu *Counting Sort, Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Heap Sort,* dan sebagainya.

b) Kompleksitas Waktu dan Ruang

Kompleksitas Algorita terbagi menjadi 2, yaitu kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu dinotasikan T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan sebuah algoritma sebagai fungsi dari ukuran masukan n. Kompleksitas ruang, S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran masukan n.

Pada waktu silam, kompleksitas memori sangat dipertimbankan dalam menilai suatu algoritma. Hal ini disebabkan karena mahalnya teknologi untuk penyimpanan data dalam jumlah besar pada masa tersebut. Namun pada zaman sekarang, biaya untuk memiliki memori relative murah, sehingga ukuran memori tidak lagi menjadi persoalan yang terlalu diangga penting, sehingga kompleksitas ruang tidak akan dibahas lebih lanjut pada makalah ini.

Kompleksitas waktu yang kecil selamanya akan menjadi kejaran bagi para programmer. Oleh karena itu, kompleksitas waktu ialah salah satu yang wajib dipelajari bagi programmer pemula. Kompleksitas waktu menghitung banyaknya operasi yang dilakukan oleh algoritma. Operasi – operasi yang umum terdapat dalam algoritma ialah penjumlahan, perbandingan, pengisian, pembacaan nilai, pemanggilan prosedur, dan sebagainya.

Beberapa algoritma mempunyai operasi dasar masing

– masing yang jika diperhitungkan bias menentukan kompleksitas algoritma tanpa perlu menghitung operasi sisanya dalam algoritma tersebut. Milsanya, algoritma pencarian didasari oleh operasi perbandingan. Algoritma pengurutan didasari oleh operasi perbandingan dan operasi pertukara elemen. Untuk algoritma perkalian, operasi dasarnya adalah operasi penjumlahan dan perkalian.

```
Void FindMax (int a[], int &max) {
/* Kamus */
    i : integer;
/* Algoritma */
  i = 1;
  max = a_1;
  while (i <= n) {
    if (a_i > max) {
      max = a_i;
    }
    i++;
  }
}
```

Algoritma pada gambar memiliki beberapa operasi perbandingan dan penjumlahan.
Operasi perbandingan :

- $a_i$ > max n kali

Sehingga total waktu operasi perbangindan dari algoritma diatas ialah $t_1$= n.

Operasi penjumlahan :

- $i$++ n kali

Total waktu dari operasi penjumlahan dari alogirtma diatas ialah $t_2$= n.

Operasi pengisian :

- max = $a_i$ n kali

Dengan demikian, kompleksitas waktu dari algoritma diatas berdasarkan kedua jenis operasi tersebut ialah :

$$T(n) = t_1 + t_2 = n + n + n = 3n.$$

n diatas ialah banyaknya data, berdasarkan kasusnya, kompleksitas dapat dibedakan menjadi tiga macam, yaitu :

1. $T_{max}(n)$ yang merupakan kompleksitas waktu untuk kasus yang terburuk pada suatu masalah.
2. $T_{min}(n)$ yang merupakan kompleksitas waktu untuk kasus terbaik pada suatu masalah(best cas).
3. $T_{avg}(n)$ yaitu kebutuhan waktu rata – rata yang diperlukan algoritma sebagai fungsi dari n.

Contoh pada algoritma diatas :

1. $T_{max}(n)$ = 3n.
2. $T_{min}(n)$ = 2n + 1.
3. $T_{avg}(n)$ = (5n – 1)/2.

c) Kompleksitas Waktu Asimptotik

Untuk mengetahui pertumbuhan $T_{min}(n)$ dan $T_{max}(n)$ bersamaan dengan meningkatnya ukuran masukan, digunakan tiga notasi kompleksitas waktu asimptotik, yaitu Big-O, Big-Omega, dan Big-Theta.

$T(n) = O(f(n))$ artinya $T(n)$ berorde paling besar $f(n)$, bila terdapat konstanta C dan $n_0$ maka berlaku $T(n) \leq Cf(n)$, untuk $n \geq n_0$.

$T(n) = \Omega(g(n))$ artinya $T(n)$ berorde paling kecil $g(n)$, bila terdapat konstanta C dan $n_0$ maka berlaku $T(n) \geq Cg(n)$, untuk $n \geq n_0$.

$T(n) = \Theta(h(n))$ artinya $T(n)$ berorde sama dengan $h(n)$ jika $T(n) = \Omega(h(n))$ dan $T(n) = O(h(n))$.

Tetapi dalam makalah ini kita akan berfokus pada notasi Big-O.

d) Big-O Theorem

Berdasarkan kompleksitas waktu, terdapat pengelompokan algoritma dalam notasi Big-O, yaitu :

| Kelompok Algoritma | Nama |
|---|---|
| $O(1)$ | Konstan |
| $O(\log n)$ | Logaritmik |
| $O(n)$ | Lanjar |
| $O(n \log n)$ | $n \log n$ |
| $O(n^2)$ | kuadratik |
| $O(n^3)$ | Kubik |
| $O(2^n)$ | Eksponensial |
| $O(n!)$ | Factorial |

Tabel 3.1 Kelompok algoritma dalam notasi Big-O

Urutan spectrum kompleksitas waktu algoritma adalah :

1. Algoritma polynomial : $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3)$.
2. Algoritma eksponensial : $O(2^n) < O(n!)$.
3. Algoritma polynomial < Algoritma eksponensial

Setiap kelompok algoritma mempunyai penjelasan masing – masing, yaitu :

- O(1) : Kompleksitas O(1) berarti waktu pelaksanaan algoritma tetap tidak bergantung pada seberapa besar input yang diberikan oleh user.
- $O(\log n)$ : Kompleksitas O($\log n$) berarti grafik laju pertumbuhan algoritma tersebut hampir sama dengan grafik $\log n$. Contoh algoritma yang memiliki kompleksitas algoritma $O(\log n)$ ialah binary search.
- $O(n)$ : Algoritma yang memiliki kompleksitas waktu lanjar umumnya terdapat pada kasus yang setiap elemennya harus diproses dengan proses yang sama, misalnya mencari maksimum dari suatu array.
- $O(n \log n)$ : Algoritma yang memilki kompleksitas waktu $O(n \log n)$ biasanya terdapat pada algoritma yang memecahkan solusinya didalam beberapa bagian. Contoh algoritma yang memiliki kompleksitas waktu $O(n \log n)$ ialah QuickSort.
- $O(n^2)$ : Algoritma yang memiliki komplesitas waktu kuadratik umumnya terdapat pada algoritma yang tiap elemennya harus diproses kembali sesuai dengan indeksnya.
- $O(n^3)$ : Algoritma kuadratik sama seperti algoritma kubik.

Di dalam algoritma ini biasanya terdapat for yang berkelipatan 3.

- $O(2^n)$ : Algoritma yang tergolong kelompok ini biasanya mencari solusi dengan secara "brute force". Algoritma ini biasanya harus memeriksa setiap kasus untuk himpunan bagian dari suatu himpunan.
- $O(n!)$ : Algoritma ini biasanya untuk setiap masukan n, dibutuhkan perbandingan atau proses lain terhadap n-1 masukan lainnya.

| $\log n$ | $n$ | $n \log n$ | $n^2$ | $n^3$ | $2^n$ | $n!$ |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 2 | 1 |
| 1 | 2 | 2 | 4 | 8 | 4 | 2 |
| 2 | 4 | 8 | 16 | 64 | 16 | 24 |
| 3 | 9 | 24 | 64 | 512 | 256 | 362880 |

Tabel 3.2 Besar setiap kompleksitas untuk beberapa input n

## IV. ALGORITMA MENCARI SOLUSI SISTEM PERSAMAAN LINEAR

Pada pembelajaran Aljabar Geometri kemarin, diberi tugas untuk membuat sebuah algoritma untuk menyelesaikan beberapa permasalahan system persamaan linear dalam bahasa pemrograman Java, tetapi yang bakal dibahas pada makalah ini ialah algoritma dalam bahasa pemrograman C. Pada algoritma ini, akan berfokus pada metode Gauss Jordan untuk menyelesaikan system persamaan linear. Akan dibahas kompleksitas waktu dari beberapa prosedur atau fungsi yang digunakan dalam algoritma tersebut.

1. Pindahkan (Untuk mengurutkan matriks agar mendapatkan variable yang tepat untuk dijadikan parameter).

```
void Pindahkan() {
//Untuk mengurutkan matriks agar mendapatkan variabel yang
tepat untuk dijadikan parameter
    double t;

    for(int i = 0; i < N; i++) {
        if (fabs(Aug.mat[i][i]) <= 0.0000000001) {
            for(int j = i-1; j >= 0; j--) {
                if (fabs(Aug.mat[j][i]) >= 0.000000001) {
                    if (fabs(Aug.mat[j][j]) <= 0.00000001) {
                        for(int k = 0; k <= N; k++) {
                            t = Aug.mat[i][k], Aug.mat[i][k]
= Aug.mat[j][k], Aug.mat[j][k] = t;
                        }
                    }
                }
            }
        }
    }
}
```

Pada algoritma diatas terdapat beberapa proses utama, yaitu : perbandingan dan pertukaran.
Operasi perbandingan :

- `if (fabs(Aug.mat[i][i]) <= 0.0000000001` n kali
- `if (fabs(Aug.mat[j][i]) >= 0.000000001)` $n^2$ kali
- `if (fabs(Aug.mat[j][i]) >= 0.000000001)` $n^2$ kali

Total waktu dari operasi perbandingan pada algoritma diatas ialah $t_1 = 2n^2 + n$.

Operasi pertukaran :

- `t = Aug.mat[i][k], Aug.mat[i][k] = Aug.mat[j][k], Aug.mat[j][k] = t;` $n^3$ kali

Total waktu dari operasi pertukaran pada algoritma diatas ialah $t_2 = n^3$.

Dengan demikian, kompleksitas waktu dari Algoritma prosedur diatas (pindahkan) ialah $T(n) = n^3 + 2n^2 + n$. Big-O dari prosedur pindahkan ialah $O(n^3)$.

2. SederhanakanJordan (Untuk menyederhanakan matriks agar menjadi matriks Gauss-Jordan).

```
void SederhanakanJordan() {
// Untuk menyederhanakan Matriks agar menjadi Matriks Gauss-
Jordan
    double t;

    for(int i = 1; i < N; i++) {
        if (fabs(Aug.mat[i][i]) <= 1.00000001 &&
fabs(Aug.mat[i][i]) >= 0.000000001) {
            for(int k = i-1; k >= 0; k--) {
                if (fabs(Aug.mat[k][i]) > 0.0000000001) {
                    t = Aug.mat[k][i];
                    for(int j = i; j <= N; j++) {
                        Aug.mat[k][j] -= Aug.mat[i][j] * t;
                    }
                }
            }
        }
    }
}
```

Pada algoritma diatas terdapat beberapa proses utama, yaitu : perbandingan, dan pengisian.
Operasi perbandingan :

- `if (fabs(Aug.mat[i][i]) <= 1.00000001 && fabs(Aug.mat[i][i]) >= 0.000000001)` n kali
- `if (fabs(Aug.mat[k][i]) > 0.0000000001` $n^2$kali

Total waktu dari operasi perbandingan pada algoritma diatas ialah $t_1 = n^2 + n$.

Opearsi pengisian :

- `Aug.mat[k][j] -= Aug.mat[i][j] * t;` $n^3$ kali

Total waktu dari operasi pengisian pada algoritma diatas ialah $t_2 = n^3$.

Dengan demikian, kompleksitas waktu dari Algoritma prosedur diatas (SederhanakanJordan) ialah $T(n) = n^3 + n^2 + n$. Big-O dari prosedur pindahkan ialah $O(n^3)$.

3. GaussJordanElimination

```
void GausJordanElimination() {
// input: N, Augmented Matrix Aug, output: Column vector X,
the answer
    int i, j, k, l; double t;

    for (i = 0; i < N; i++) { // the forward elimination
phase
        l = i;
        for (j = i + 1; j < N; j++) {// which row has
largest column value
            if (fabs(Aug.mat[j][i]) > fabs(Aug.mat[l][i])) {
                l = j; // remember this row l
            }
        }
```

```
        // swap this pivot row, reason: minimize floating
point error
        for (k = i; k <= N; k++) { // t is a temporary
double variable
            t = Aug.mat[i][k], Aug.mat[i][k] =
Aug.mat[l][k], Aug.mat[l][k] = t;
        }

        if (fabs(Aug.mat[i][i]) < 0.00000001) {
            continue;
        } else {
            for (j = i + 1; j < N; j++) {// the actual
forward elimination phase
                for (k = N; k >= i; k--) {
                    Aug.mat[j][k] -= Aug.mat[i][k] *
Aug.mat[j][i] / Aug.mat[i][i];
                }
            }
            t = 1/Aug.mat[i][i];
            for(j = i; j <= N; j++) {
                Aug.mat[i][j] *= t;
            }
        }
        Pindahkan();
    }

    SederhanakanJordan();
}
```

Pada algoritma diatas terdapat beberapa proses utama, yaitu : Pindahkan, pertukaran, perbandingan, pengisian, SederhanakanJordan.

Operasi Pindahkan :

- Pindahkan — n kali

Total waktu dari operasi Pindahkan pada algoritma diatas ialah $t_1 = n.n^3 = n^4$.

Operasi pertukaran :

- `t = Aug.mat[i][k], Aug.mat[i][k] = Aug.mat[l][k], Aug.mat[l][k] = t;` — $n^2$ kali

Total waktu dari operasi pertukaran pada algoritma diatas ialah $t_2 = n^2$.

Operasi perbandingan :

- `if (fabs(Aug.mat[j][i]) > fabs(Aug.mat[l][i]))` — $n^2$ kali
- `if (fabs(Aug.mat[i][i]) < 0.00000001)` — n kali

Total waktu dari operasi pertukaran pada algoritma diatas ialah $t_3 = n^2 + n$.

Operasi pengisian :

- `l = I` — n kali
- `l = j` — $n^2$ kali
- `Aug.mat[j][k] -= Aug.mat[i][k] * Aug.mat[j][i] / Aug.mat[i][i]` — $n^2$ kali
- `t = 1/Aug.mat[i][i]` — n kali
- `Aug.mat[i][j] *= t` — n kali

Total waktu dari operasi pengisian pada algoritma diatas ialah $t_4 = 2n^2 + 3n$.

Operasi SederhanakanJordan :

- `SederhanakanJordan();` — 1 kali

Total waktu dari operasi SederhanakanJordan pada algoritma diatas ialah $t_5 = n^3$.

Dengan demikian, kompleksitas waktu dari algoritma prosedur diatas (GaussJordanElimination) ialah ialah $T(n) = n^4 + n^3 + 4n^2 + 4n$. Big-O dari prosedur pindahkan ialah $O(n^4)$.

4. NoSolution

```
bool NoSolution() {
// Untuk mengecek apakah matriks tersebut mempunyai solusi
atau tidak
    bool NoSolution = false;
    for(int i = 0; i < N; i++) {
        if (fabs(Aug.mat[i][i]) <= 0.0000000001) {
            NoSolution = true;
            for(int j = i+1; j < N; j++) {
                if (fabs(Aug.mat[i][j]) >= 0.0000000001) {
                    NoSolution = false;
                }
            }
            if (NoSolution) {
                if (fabs(Aug.mat[i][N]) <= 0.0000000001) {
                    NoSolution = false;
                }
            }
        }
    }

    return NoSolution;
}
```

Pada algoritma diatas terdapat beberapa proses utama, yaitu : perbandingan, pengisian, SederhanakanJordan.

Operasi perbandingan :

- `if (fabs(Aug.mat[i][i]) <= 0.0000000001)` — $n$ kali
- `if (fabs(Aug.mat[i][j]) >= 0.0000000001)` — $n^2$ kali

Total waktu dari operasi pertukaran pada algoritma diatas ialah $t_1 = n^2 + n$.

Operasi pengisian :

- `NoSolution = true` — n kali
- `NoSolution = false` — $n^2$ kali
- `NoSolution = false` — n kali

Total waktu dari operasi pengisian pada algoritma diatas ialah $t_2 = n^2 + 2n$.

Dengan demikian, kompleksitas waktu dari algoritma fungsi diatas (GaussJordanElimination) ialah ialah $T(n) = 2n^2 + 3n$. Big-O dari prosedur pindahkan ialah $O(n^2)$.

5. Solution

```
void Solution() {
// Untuk mengeluarkan Solusi dari matriks tersebut dengan
metode Gauss-Jordan
    GausJordanElimination();

    for(int i = 0; i < NS; i++) {
        if (fabs(Aug.mat[i][i]) <= 0.0000000001) {
            Parameter[i] = true;
        }
    }

    if (NoSolution()) {
        cout << "TIDAK ADA SOLUSI\n";
    } else if (!NoSolution()) {
        for(int i = 0; i < NS; i++) {
            if (Parameter[i]) {
                cout << "x" << i << " = " << Variabel[i] <<
"\n";
            } else {
                cout << "x" << i << " = " << Aug.mat[i][N];
                for(int j = i + 1; j < NS; j++) {
```

```
                    if (Parameter[j] && fabs(Aug.mat[i][j])
>= 0.0000000001) {
                        if (Aug.mat[i][j] < 0) {
                            cout << " + " <<
fabs(Aug.mat[i][j]) << Variabel[j];
                        } else {
                            cout << " - " << Aug.mat[i][j]
<< Variabel[j];
                        }
                    }
                }
                printf("\n");
            }
        }
    }
}
```

Pada algoritma diatas terdapat beberapa proses utama, yaitu : GaussJordanElimination, NoSolution, perbandingan, pengisian.

Operasi GaussJordanElimination :

- GaussJordanElimination 1 kali

Total waktu dari operasi Pindahkan pada algoritma diatas ialah $t_1 = n^4$.

Operasi NoSolution :

- NoSolution 1 kali

Total waktu dari operasi pertukaran pada algoritma diatas ialah $t_2 = n^2$.

Operasi perbandingan :

- `if (fabs(Aug.mat[i][i]) <= 0.0000000001` n kali
- `Aug.mat[i][j] < 0` $n^2$ kali

Total waktu dari operasi pertukaran pada algoritma diatas ialah $t_3 = n^2 + n$.

Operasi pengisian :

- `Parameter[i] = true` n kali

Total waktu dari operasi pengisian pada algoritma diatas ialah $t_3 = n$.

Dengan demikian, kompleksitas waktu dari algoritma prosedur diatas (GaussJordanElimination) ialah ialah $T(n) = n^4 + 2n^2 + 2n$. Big-O dari prosedur pindahkan ialah $O(n^4)$.

Sehingga untuk keseluruhan programnya diperoleh kebutuhan waktunya :

$O(f(n)) = n^3 + n^3 + n^4 + n^2 + n^4 = 2n^4 + 2n^3 + n^2$.

Kompleksitas Big-O dari keseluruhan program ialah $O(n^4)$.

## V. KESIMPULAN

Sistem persamaan linear ialah masalah yang dapat ditemukan dimana saja. Sistem persamaan linear dapat berupa 2 variabel ataupun lebih. Untuk jumlah variable yang cukup banyak dibutuhkan algoritma untuk mengatasi masalah tersebut.

Salah satu ciri algoritma yang baik ialah algoritma yang efisien. Artinya algoritma tersebut memiliki kompleksitas ruang dan waktu yang cukup kecil. Terdapat pula beberapa notasi yang digunakan untuk membandingkan suatu kompleksitas algoritma, yaitu : Big-O, Big-Omega, dan Big-Theta. Tetapi pada makalah ini kita berfokus pada notasi Big-O.

Algoritma diatas yang telah dibuat oleh beberapa siswa Aljabar Geometri dalam bahasa C++ memiliki kompleksitas Big-O = $O(n^4)$ yang masih sangat besar. Untuk n yang kurang dari 100 masih dapat diatasi dalam beberapa detik saja. Tetapi untuk n yang cukup besar, maka dibutuhkan beberapa jam atau mungkin dalam beberapa hari. Jadi, algoritma diatas masih dapat dikatakan belum efisien untuk penggunaannya.

## VI. PENUTUP

Puji Syukur pertama kali saya ucapkan kepada Tuhan Yang Maha Esa karena berkat-Nya yang melimpah sehingga penulis dapat menyelesaikan makalah ini dengan baik. Ucapan terima kasih juga penulis sampaikan kepada kedua orang tua, keluarga, serta teman – teman yang terus memberikan dukungan baik secara moral maupun doa. Penulis juga turut mengucapkan terima kasih kepada Bapak Rinaldi Munir, selaku dosen dari mata kuliah Matematika Diskrit. Akhir kata, penulis memohon maaf atas salah kata atau ketidaksempurnaannya makalah ini. Penulis berharap makalah ini dapat bermanfaat bagi orang lain dan terus dikembangkan sehingga menjadi semakin baik lagi.

## REFERENCES

[1] R. Munir, *Matematika Diskrit*, 3rd ed. Bandung: Penerbit INFORMATIKA Bandung, 2010, ch. 10.

[2] Anton, Howard, 2013. Elementary Linear Aljebra 11th Edition. Portland : Wiley Global Education.

## 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, 3 Desember 2017

Muh. Habibi Haidir / 13516085