

# BAB I

## ALGORITMA

### 1.1. Pendahuluan

Algoritma memegang peranan penting dalam bidang pemrograman. Sebegitu pentingnya suatu algoritma, sehingga perlu dipahami konsep dasar algoritma. Apalagi untuk seorang programmer, tentu diperlukan suatu algoritma sehingga dapat membuat program yang lebih efektif dan efisien. Bagi kebanyakan orang, algoritma sangat membantu dalam memahami konsep logika pemrograman.

Algoritma adalah kumpulan instruksi yang dibuat secara jelas untuk menunjukkan langkah-langkah penyelesaian suatu masalah. Pada umumnya algoritma kurang lebih sama dengan suatu prosedur yang sering dilakukan setiap hari, misalnya prosedur untuk mengganti ban bocor/pecah, prosedur pemakaian telepon umum, prosedur membuat kue dan lain-lain.

Dalam bidang komputer, misalnya EDP (*Elektronik Data Processing*) atau MIS (*Management Information System*), algoritma sering dimanfaatkan untuk menyelesaikan suatu masalah atau untuk proses pengambilan keputusan. Seorang sistem analisis (*analisisist system*) tentunya menggunakan algoritma untuk merancang suatu sistem. Bagi seorang programmer, algoritma digunakan untuk membuat modul-modul program.

Guna memahami suatu algoritma, harus dimiliki pengetahuan dasar matematika karena pada dasarnya algoritma lahir dari konsep logika matematika. Disini yang perlu dilatih adalah kemampuan logikanya agar benar-benar bisa menyusun langkah-langkah penyelesaian masalah dengan baik.

Dalam buku ajar ini, disajikan konsep dasar dan analisis algoritma. Pada bagian konsep dasar dibahas komponen utama, desain, dan contoh pembuatan. Selanjutnya, untuk mendapatkan algoritma yang efisien serta mendapatkan rumusan matematika sebagai ukuran kerumitan (kompleksitas) maka dibahas analisis algoritma dengan menggunakan notasi  $O$  (big  $O$ ).

## 1.2. Konsep Dasar Algoritma

Algoritma adalah kumpulan instruksi/perintah yang dibuat secara jelas dan sistematis berdasarkan urutan yang logis (logika) untuk penyelesaian suatu masalah.

French, C.S. (1984) menyatakan sejumlah konsep yang mempunyai relevansi dengan masalah rancangan program yaitu kemampuan komputer, kesulitan dan ketepatan. Penerapan dari konsep tersebut biasanya digunakan dalam rancangan algoritma. Dalam merancang sebuah algoritma, Fletcher (1991) memberikan beberapa cara atau metode yaitu kumpulan perintah, ekspresi, tabel instruksi, program komputer, kode semu dan flow chart, sedangkan Knuth (1973) menyarankan algoritma fundamental. Untuk keperluan matematika dan program komputer metode yang sering digunakan yaitu :

1. Diagram Alir (Flow Chart)
2. Kode Semu (Pseudo Code)
3. Algoritma Fundamental

Knuth (1973) menyatakan 5 komponen utama dalam algoritma yaitu finiteness, definiteness, input, output dan effectiveness. Sehingga dalam merancang sebuah algoritma ada 3 (tiga) komponen yang harus ada yaitu:

1. Komponen masukan (input)

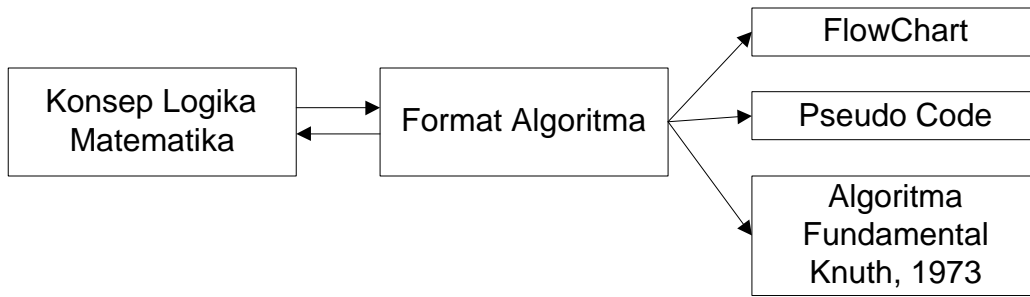
Komponen ini biasanya terdiri dari pemilihan variable, jenis variable, tipe variable, konstanta dan parameter (dalam fungsi).

2. Komponen keluaran (output)

Komponen ini merupakan tujuan dari perancangan algoritma dan program. Permasalahan yang diselesaikan dalam algoritma dan program harus ditampilkan dalam komponen keluaran. Karakteristik keluaran yang baik adalah benar (menjawab) permasalahan dan tampilan yang ramah (Friendly).

3. Komponen proses (processing)

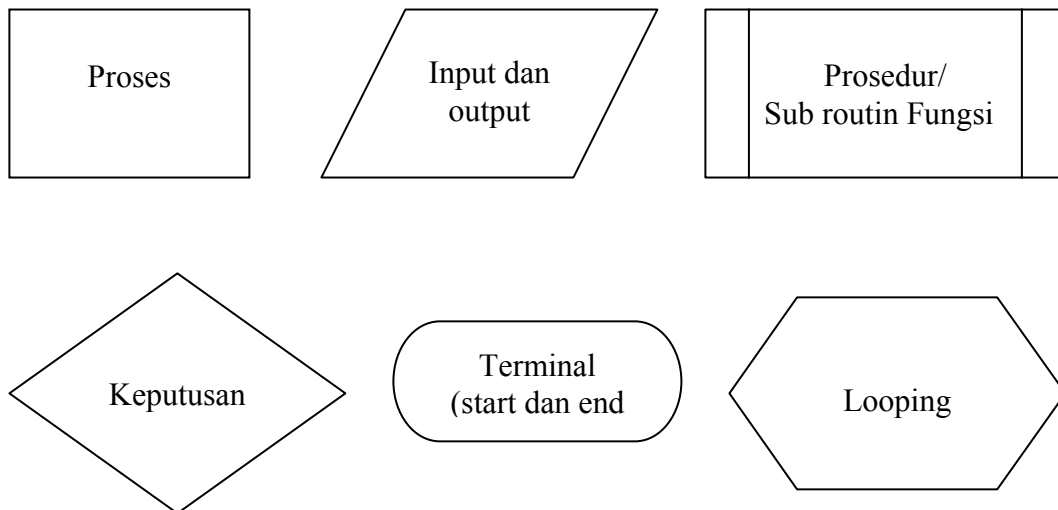
Komponen ini merupakan bagian utama dan terpenting dalam merancang sebuah algoritma. Dalam bagian ini terdapat logika masalah, logika algoritma (sintaksis dan semantik), rumusan, metode (rekursi, perbandingan, penggabungan, pengurangan dan lain-lain).

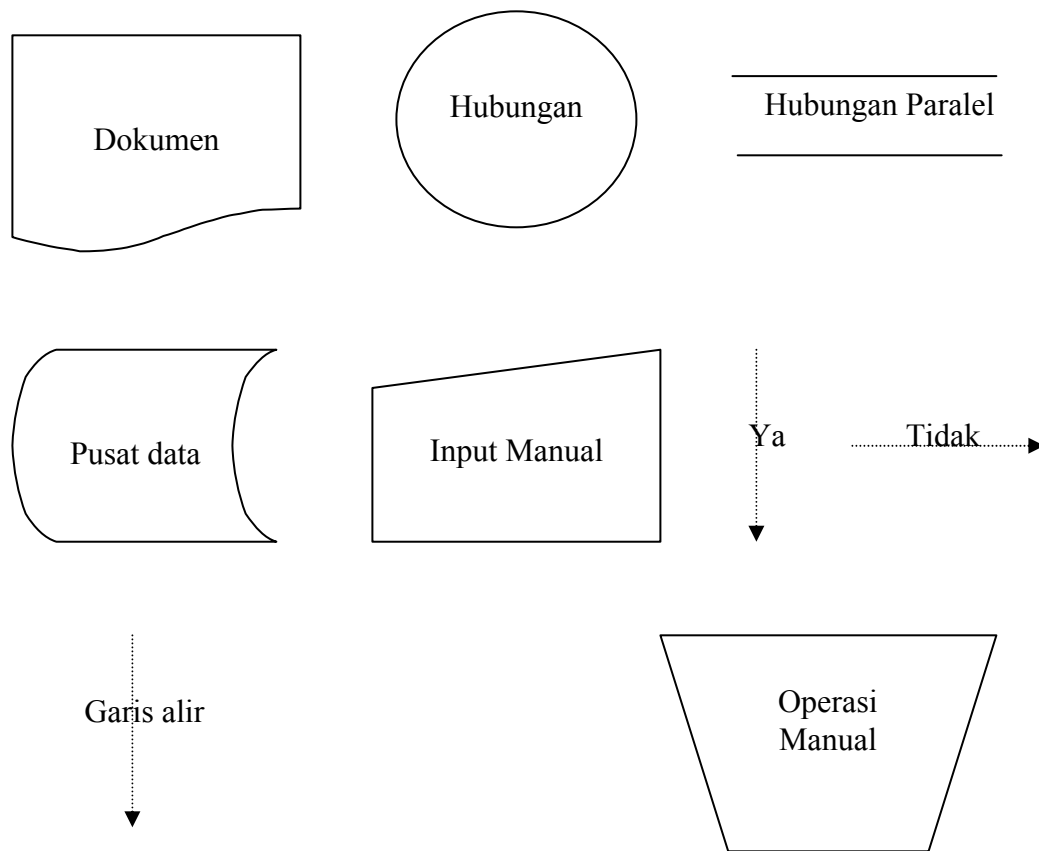


Gambar 1.1. Struktur Hubungan dan Jenis Algoritma

### 1.3. Diagram Alir

Algoritma ini menggunakan sejumlah simbol untuk menyatakan kegiatan-kegiatan secara keseluruhan. Simbol dan artinya dalam diagram alir sebagai berikut:





Gambar 1.2. Simbol dan arti Diagram Alir

#### 1.4. Kode Semu

Dalam merancang sebuah algoritma menggunakan kode semu, komponen-komponen input, output dan proses harus terdefinisi secara jelas. Disamping itu beberapa ketentuan dan aturan pendefinisian memang secara baku tidak ditemukan dalam beberapa buku literatur, namun aturan-aturan yang di ajukan dibawah ini akan membantu mempermudah perancangan algoritma dan evaluasi serta analisis algoritma.

Aturan-aturan tersebut :

1. Kode semu harus dimulai dengan judul. Aturan ini secara mudah dapat dimengerti fungsi dan manfaatnya. Judul harus dapat menjelaskan spesifikasi masalah yang dirancang algoritamanya. Penulisannya dapat dengan huruf kapital semuanya atau tidak.
2. Kode semu harus ditulis dengan nomor yang menunjukkan urutan-urutan langkah-langkah dalam algoritma.
3. Pendeklarasian variabel, konstanta, parameter, rumus dan pernyataan harus sederhana

### Contoh. 1.1

Bandingkan kedua algoritma ini. Masalah : Mencari akar-akar persamaan non linear dengan metode bagi dua.

A. Kode semu yang dirancang tidak menggunakan aturan.

Penyelesaian:

1. Formulasikan sebuah persamaan non linier
2. Cari nilai bawah  $x_b$  yang menyebabkan nilai fungsi  $f(x_b)$  positif atau negatif, kemudian cari nilai atas  $x_a$  yang menyebabkan nilai fungsi  $f(x_a)$  berlawanan (positif negatif) dengan nilai bawah.
3. Bandingkan nilai  $f(x_b)$  dengan  $f(x_a)$
4. Jika  $f(x_b).f(x_a) > 0$  maka ulangi langkah 2
5. Jika  $f(x_b).f(x_a) < 0$  maka bagi dua interval  $x_b$  dengan  $x_a$ . Ulangi langkah 3
6. Jika  $f(x_b).f(x_a) = 0$  maka iterasi berhenti, akar-akar persamaan  $x$  diperoleh

B. Kode semu yang dirancang menggunakan aturan

Penyelesaian:

Algoritma Bagi Dua

1. Formulasikan masalah  $f(x)$
2. Cari taksiran bawah ( $x_b$ ) dan taksiran atas ( $x_a$ )
3. Bandingkan dan evaluasi, jika  $f(x_b).f(x_a) > 0$  maka ulangi langkah 2
4. Jika  $f(x_b).f(x_a) < 0$  maka bagi dua interval dengan  $(x_b+x_a) / 2$ , kembali bandingkan dan evaluasi.
5. Jika  $f(x_b).f(x_a) = 0$  maka iterasi berhenti, akar-akar persamaan  $x$  diperoleh

## 1.5. Algoritma Fundamental

Knuth (1973) menyajikan format algoritma yang dapat digunakan secara bebas untuk berbagai bahasa pemrograman, artinya dapat dengan mudah diimplementasikan menggunakan Pascal, C, Fortran, PL atau BASIC. Secara umum notasi dan aturan yang digunakan sebagai berikut :

1. Nama/judul algoritma harus ditulis dengan huruf kapital

Contoh : Algoritma BAGI DUA

2. Berikan komentar dan penjelasan pendahuluan. Penjelasan secara singkat tentang algoritma.

Contoh : Algoritma BAGI DUA

Mencari akar persamaan dengan taksiran pertama  $x_b$  dan  $x_a$

3. Langkah-langkah. Algoritma tersusun menurut nomor langkah-langkah diawali dengan '[.....]' untuk memberikan keterangan tentang langkah tersebut.

Contoh : 1. [formulasikan  $f(x)$ ]

4. Komentar (comments). Komentar untuk penjelasan bagi pembaca ditulis dengan tanda (.....)

5. Pernyataan dan struktur Kontrol

Pernyataan adalah perintah yang terdapat didalam algoritma, sedangkan struktur kontrol untuk mengendalikan pernyataan yang digunakan. Pernyataan dan struktur kontrol terdiri dari :

- a. Perintah pemberian nilai menggunakan  $\leftrightarrow$ ,  $\leftarrow$

Contoh :  $A \leftarrow B$  (artinya  $A = B$ )

$X \leftarrow 0$  (artinya  $x$  bernilai 0)

$X \leftrightarrow Y$  (artinya  $x$  dan  $y$  saling tukar)

- b. Pernyataan IF

Perintah yang digunakan:

- IF kondisi

Then.....

- IF kondisi

Then.....

.....

else.....

c. Pernyataan Case

Perintah ini untuk menyeleksi pilihan tertentu. Bentuknya :

Select Case (ekspresi)

Case nilai 1 :

Case nilai 2 :

.

.

.

Case nilai n :

Default :

d. Pernyataan Repeat

Perintah pengulangan digunakan dengan bentuk :

- Repeat for indeks = barisan nilai
- Repeat while ekspresi logika
- Repeat for indeks = barisan nilai while ekspresi logika

e. Pernyataan Goto dan Exitloop

Perintah untuk melompat ke langkah yang telah ditentukan dan keluar dari pengulangan.

Bentuknya :

Goto step.....

Exitloop

f. Pernyataan Exit

Perintah untuk menghentikan algoritma.

6. Nama-nama variabel harus ditulis dengan huruf besar

7. Input dan output

Data dapat dimasukkan melalui variabel dengan pernyataan READ dengan bentuk :

Read : NAMA VARIABEL

Untuk mencetak pesan-pesan/tulisan (diapit dengan tanda kutip) dan juga variabel digunakan pernyataan :

Write : tulisan dan atau nama variabel

## 8. Prosedur

Bentuk prosedur digunakan untuk modul algoritma yang berdiri sendiri untuk menyelesaikan masalah tertentu. Pemakaian prosedur untuk masalah sederhana, sedangkan algoritma untuk masalah umum. Bentuk yang digunakan :

Procedure nama prosedur

## 9. Fungsi

Sama dengan prosedur menggunakan bentuk :

Function nama fungsi

### Contoh 1.2.

Masalah : Mencari elemen terbesar dari data dengan  $n$  bilangan.

Buatlah algoritma dari masalah ini menggunakan

- a. Kode Semu
- b. Diagram Alir
- c. Algoritma Fundamental

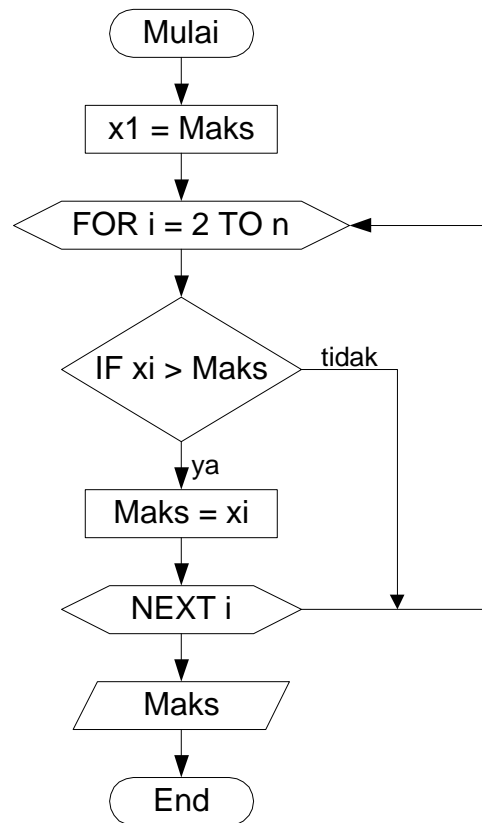
Penyelesaian :

### a. Kode semu (Pseudo Code)

Algoritma Maksimum

1. Mula-mula masukkan bilangan dalam register  $x_i$  ke dalam register yang dinamakan maks.
2. Untuk  $i = 2, 3, \dots, n$ , lakukan : Bandingkan bilangan dalam register  $x_i$  dengan bilangan dalam register maks. Jika bilangan dalam register  $x_i$  lebih besar daripada bilangan dalam register maks, pindahkan bilangan dalam register  $x_i$  ke register maks; jika tidak jangan lakukan apa-apa.
3. Terakhir, bilangan dalam register maks adalah elemen terbesar di antara  $n$  bilangan.

b. FlowChart (Diagram Alir)



c. Algoritma Fundamental

Algoritma MAKSIMUM

Mencari elemen terbesar di dalam data dengan n bilangan.

1. [Inisialisasi]

Maks  $\leftarrow$  x1

2. [Mulai Loop]

I  $\leftarrow$  2

3. [Naikkan Pencacah]

I  $\leftarrow$  I + 1

4. [Bandingkan]

IF Maks < x<sub>i</sub>

THEN Maks  $\leftarrow$  x<sub>i</sub> ELSE GOTO 3

5. [Ulangi Loop]

GOTO 3

6. [Selesai]

Exit

## 1.6. Analisis Algoritma dan Kompleksitas Algoritma

Seorang programmer atau sistem analisis paling tidak harus memiliki dasar untuk menganalisis algoritma. Analisis algoritma sangat membantu di dalam meningkatkan efisiensi program. Kecanggihan suatu program bukan dilihat dari tampilan program, tetapi berdasarkan efisiensi algoritma yang terdapat didalam program tersebut.

Pembuatan program komputer tidak terlepas dari algoritma, apalagi program yang dibuat sangat kompleks. Program dapat dibuat dengan mengabaikan algoritma, tetapi jangan heran bila seandainya ada orang lain yang membuat program seperti program anda tersebut memiliki akses yang lebih cepat dan memakai memori yang sangat sedikit.

Analisis algoritma adalah bahasan utama dalam ilmu komputer. Dalam menguji suatu algoritma, dibutuhkan beberapa kriteria untuk mengukur efisiensi algoritma.

Terdapat dua tipe analisis algoritma, yaitu :

### 1. Memeriksa kebenaran algoritma

Dapat dilakukan dengan cara perurutan, memeriksa bentuk logika, implementasi algoritma, pengujian dengan data dan menggunakan cara matematika untuk membuktikan kebenaran.

### 2. Penyederhanaan Algoritma

Membagi algoritma menjadi bentuk yang sederhana.

## 1.7. Notasi O (Big O)

Misalkan 4 program yang mensorting n bilangan dengan fungsi yang menyatakan sejumlah langkah yang dijumlahkan masing-masing program untuk sorting n bilangan :

$$f_1(n) = n, \quad f_2(n) = n^2, \quad f_3(n) = 2^n, \quad f_4(n) = n!$$

Bila  $n = 4$  maka  $f_1(n) = 4$ ,  $f_2(n) = f_3(n) = 16$  dan  $f_4(n) = 24$  sedangkan untuk  $n = 100$ , program ketiga akan memerlukan  $2^{1000}$  langkah.

Dalam analisis sebuah algoritma biasanya yang dijadikan ukuran adalah operasi aljabar seperti penjumlahan, pengurangan, perkalian dan pembagian, proses pengulangan (looping/Iterasi), proses pengurutan (sorting) dan proses pencarian (searching).

**Definisi 1.1**

Misalkan  $S$  sebuah himpunan fungsi riil untuk setiap domain  $D$ , dengan  $D = \mathbb{N}, \mathbb{Z}$  atau  $\mathbb{R}$ .

Misalkan  $f, g \in S$ . Dikatakan  $f = O(g)$  jika ada bilangan positif  $c$  dan  $k$  sehingga

$$|f(x)| \leq c |g(x)| \text{ untuk semua } x \in D \text{ yang } x > k$$

Dalam analisis algoritma ada 3 bagian yang dapat dianalisis yaitu kecepatan waktu, kapasitas biaya dan kapasitas ruang. Ketiganya menggunakan notasi  $O$ .

**Teorema 1.1**

Misalkan  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  adalah fungsi polinom dan misalkan  $g(x) = x^n$  maka  $f = O(g)$ .

Bukti :

Misalkan  $c = |a_n| + |a_{n-1}| + \dots + |a_1| + |a_0|$ . Misalkan  $x$  sebuah bilangan riil yang lebih dari 1, maka

$$\begin{aligned} |F(x)| &= |a_n x^n + \dots + a_1 x + a_0| \\ &\leq |a_n| x^n + |a_{n-1}| x^{n-1} + \dots + |a_1| x + |a_0| \\ &\leq |a_n| x^n + |a_{n-1}| x^n + \dots + |a_1| x^n + |a_0| x^n \\ &= (|a_n| + |a_{n-1}| + \dots + |a_1| + |a_0|) x^n \\ &= c x^n = c |g(x)| \end{aligned}$$

Sehingga  $f = O(g)$

**Teorema 1.2.**

Misalkan  $S$  sebuah himpunan fungsi dengan dominan  $D$  yang sama dengan  $D = \mathbb{N}, \mathbb{Z}$  atau  $\mathbb{R}$  dan kodomain  $(0, \infty)$ . Misalkan  $f_1, f_2, g_1, g_2 \in S$  sehingga  $f_1 = O(g_1)$  dan  $f_2 = O(g_2)$ , maka dengan hubungan  $O$  diperoleh :

a).  $f_1 + f_2 = O(\max\{g_1, g_2\})$  dan

b).  $f_1 \cdot f_2 = O(g_1 g_2)$

Bukti:

- a) Karena  $f_1=O(g_1)$  dan  $f_2=O(g_2)$ , ada bilangan positif  $c_1$ ,  $k_1$ ,  $c_2$ , dan  $k_2$  sedemikian sehingga jika  $x \in D$  dan  $x > k_1$  maka  $|f_1(x)| < c_1 |g_1(x)|$  dan jika  $x > k_2$  maka  $|f_2(x)| < c_2 |g_2(x)|$ . Misalkan  $k=\max\{k_1, k_2\}$  dan  $c = c_1 + c_2$ , maka

$$\begin{aligned} |(f_1 + f_2)(x)| &= (f_1+f_2)(x) \\ &= f_1(x) + f_2(x) \\ &< c_1 |g_1(x)| + c_2 |g_2(x)| \\ &= c_1 g_1(x) + c_2 g_2(x) \\ &\leq c_1 \max\{g_1(x), g_2(x)\} + c_2 \max\{g_1(x), g_2(x)\} \\ &= (c_1+c_2) \max\{g_1(x), g_2(x)\} \\ &= c \max\{g_1(x), g_2(x)\} \\ &= c |(\max\{g_1, g_2\})(x)| \end{aligned}$$

maka  $f_1 + f_2 = O(\max\{g_1, g_2\}(x))$

b) 
$$\begin{aligned} |(f_1 f_2)(x)| &= |f_1(x)| |f_2(x)| \\ &\leq c_1 |g_1(x)| c_2 |g_2(x)| \\ &= c_1 c_2 |g_1(x) g_2(x)| \\ &= c_1 c_2 |(g_1 g_2)(x)| \\ &= c |(g_1 g_2)(x)| \end{aligned}$$

sehingga  $f_1 f_2 = O(g_1 g_2)$