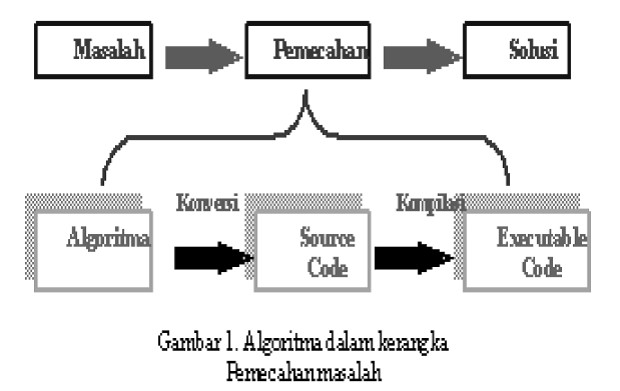
**Kompleksitas Algoritma**

* 1. **Tujuan analisa algoritma**
     1. Menggunakan tools dan teknik-teknik yang lazim digunakan untuk analisis dan desain algoritma,
     2. Menganalisis disain algoritma, dan menentukan kebenaran suatu algoritma terhadap kasus-kasus tertentu,
     3. Membandingkan beberapa algoritma dan menentukan algoritma yang terbaik untuk memecahkan kasus-kasus tertentu.
  2. **Pengertian Algoritma**

Sebagai basis pemerograman komputer, algoritma mendeskripsikan urutan langkah-langkah yang diperlukan untuk pemecahan masalah (penyelesaian persoalan), yang memiliki ciri-ciri sebagai berikut :

* + 1. selalu memiliki terminasi/langkah akhir
    2. setiap langkah dinyatakan secara jelas dan tegas
    3. setiap langkah sederhana, sehingga kinerjanya
    4. sehubungan dengan waktu yang effisien/bisa diterima akal
    5. memberikan hasil (output), mungkin dengan satu atau tanpa input.



* 1. **Tahapan Perancangan Algoritma**
* ***Mendefinisikan masalah,***

Masalah yang ingin dipecahkan harus jelas lingkupnya.

* ***Membuat model***

Yang dimaksud model ini adalah model (bentuk) matematis yang dapat digunakan untuk memecahkan masalah, misalnya apakah harus dilakukan pengurutan terhadap data, apakah menggunakan perhitungan kombinatorik dan sebagainya.

* ***Merancang algoritma***

(flowchart/pseudocode), Apa maksudnya, bagaimana rincian, prosesnya, apa keluarannya.

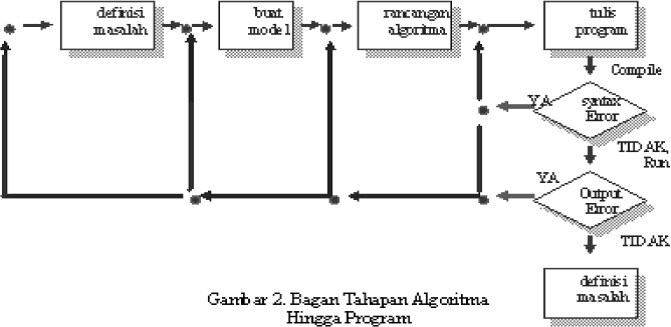
* ***Menulis program***

Ubah algoritma menjadi program (source code) dalam bahasa pemrograman tertentu.

* ***Mengubah source code menjadi executable code*** melalui proses compiling.
* ***Memeriksa hasil compiling***, jika salah maka kembali ke tahap empat.
* ***Menjalankan program (run)*** untuk diuji kebenarannya dengan menggunakan berbagai data
* ***Memperbaiki kesalahan*** (debugging dan testing)

Apabila hasilnya salah, kesalahan mungkin terjadi saat konversi rancangan algoritma manjadi program, atau salah rancang algoritma, atau salah menentukan model, atau salah mendefinisikan masalah. Ulangi langkah yang sesuai.

* ***Mendokumentasikan*** Program jika sudah benar



* 1. **Kompleksitas Algoritma**

Sebuah algoritma tidak saja harus benar, tetapi juga harus mangkus (*efisien*). Algoritma yang bagus adalah algoritma yang mangkus. Kemangkusan algoritma diukur dari berapa jumlah waktu dan ruang (*space*) memori yang dibutuhkan untuk menjalankannya.

Algoritma yang mangkus ialah algoritma yang meminimumkan kebutuhan waktu dan ruang. Kebutuhan waktu dan ruang suatu algoritma bergantung pada ukuran masukan (*n*), yang menyatakan jumlah data yang diproses. Kemangkusan algoritma dapat digunakan untuk menilai algoritma yang bagus.

Mengapa kita memerlukan algoritma yang mangkus? Lihat grafik di bawah ini.

Waktu komputasi (dalam detik)

10-4 x 2*n*

105

1 hari

10-6 x 2*n*

104

1 jam

103

10-4 x *n*3

102

1 menit

10

1

-6 3

1 detik

10 x *n*

5 10 15 20 25 30 35 40

10-1

Ukuran masukan

Gambar 1 Grafik Kebutuhan waktu terhadap jumlah masukan

* 1. **Model Perhitungan Kebutuhan Waktu/Ruang**

Kita dapat mengukur waktu yang diperlukan oleh sebuah algoritma dengan menghitung banyaknya operasi/instruksi yang dieksekusi. Jika kita mengetahui besaran waktu (dalam satuan detik) untuk melaksanakan sebuah operasi tertentu, maka kita dapat menghitung berapa waktu sesungguhnya untuk melaksanakan algoritma tersebut.

**Contoh 1.** Menghitung rerata

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *a*1 | *a*2 | *a*3 | … | *an* |

Larik bilangan bulat

procedure HitungRerata(input a1, a2, ..., an : integer, output r : real)

*{ Menghitung nilai rata-rata dari sekumpulan elemen larik integer a1, a2,*

*..., an.*

*Nilai rata-rata akan disimpan di dalam peubah r. Masukan: a1, a2, ..., an*

*Keluaran: r (nilai rata-rata)*

*}*

**Deklarasi**

k : integer jumlah : real

**Algoritma**

jumlah0 k1

while k  n do jumlahjumlah + ak kk+1

endwhile

*{ k > n }*

r  jumlah/n

{ nilai rata-rata }

1. Operasi pengisian nilai (jumlah0, k1, jumlahjumlah+ak, kk+1, dan r  jumlah/n). Jumlah seluruh operasi pengisian nilai adalah

*t*1 = 1 + 1 + *n* + *n* + 1 = 3 + 2*n*

1. Operasi penjumlahan (jumlah+ak, dan k+1). Jumlah seluruh operasi penjumlahan adalah

*t*2 = *n* + *n* = 2*n*

1. Operasi pembagian (jumlah/n). Jumlah seluruh operasi pembagian adalah

*t*3 = 1

Total kebutuhan waktu algoritma HitungRerata:

*t* = *t*1 + *t*2 + *t*3 = (3 + 2*n*)a + 2*nb* + *c* detik

Model perhitungan kebutuhan waktu seperti di atas kurang berguna, karena:

* 1. Dalam praktek kita tidak mempunyai informasi berapa waktu sesungguhnya untuk melaksanakan suatu operasi tertentu
  2. Komputer dengan arsitektur yang berbeda akan berbeda pula lama waktu untuk setiap jenis operasinya.

Model abstrak pengukuran waktu/ruang harus independen dari pertimbangan mesin dan *compiler* apapun. Besaran yang dipakai untuk menerangkan model abstrak pengukuran waktu/ruang ini adalah **kompleksitas algoritma**. Ada dua macam kompleksitas algoritma, yaitu: **kompleksitas waktu** dan **kompleksitas ruang**.

Kompleksitas waktu, *T*(*n*), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan 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*.

Dengan menggunakan besaran kompleksitas waktu/ruang algoritma, kita dapat menentukan *laju* peningkatan waktu (ruang) yang diperlukan algoritma dengan meningkatnya ukuran masukan *n*.

* 1. Kompleksitas Waktu

Dalam praktek, kompleksitas waktu dihitung berdasarkan jumlah operasi abstrak yang

*mendasari* suatu algoritma, dan memisahkan analisisnya dari implementasi.

**Contoh 2.** Tinjau algoritma menghitung rerata pada Contoh 1. Operasi yang mendasar pada algoritma tersebut adalah operasi penjumlahan elemen-elemen *ak* (yaitu jumlahjumlah+ak), Kompleksitas waktu HitungRerata adalah *T*(*n*) = *n.*

**Contoh 3.** Algoritma untuk mencari elemen terbesar di dalam sebuah larik (*array*) yang berukuran *n* elemen.

procedure CariElemenTerbesar(input a1, a2, ..., an : integer, output maks : integer)

*{ Mencari elemen terbesar dari sekumpulan elemen larik integer a1, a2, ..., an. Elemen terbesar akan disimpan di dalam maks.*

*Masukan: a1, a2, ..., an Keluaran: maks (nilai terbesar)}*

**Deklarasi**

k : integer

**Algoritma** maksa1 k2

while k  n do

if ak > maks then maksak

endif kk+1 endwhile

*{ k > n }*

Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi perbandingan elemen larik (A[k] > maks).

Kompleksitas waktu CariElemenTerbesar : *T*(*n*) = *n* – 1. Kompleksitas waktu dibedakan atas tiga macam :

1. *Tmax(n)* : kompleksitas waktu untuk kasus terburuk *(worst case),*

 kebutuhan waktu maksimum.

1. *Tmin(n)* : kompleksitas waktu untuk kasus terbaik *(best case),*

 kebutuhan waktu minimum.

1. *Tavg(n)*: kompleksitas waktu untuk kasus rata-rata *(average case)*

 kebutuhan waktu secara rata-rata