Kompleksitas Algoritma

Desain dan Analisis Algoritma

Made Windu Antara Kesiman

Jurusan PTIK
UNDIKSHA Singaraja
Oktober 2008

Menganalisis sebuah algoritma

- sumber daya yang dibutuhkan untuk mengimplementasikan algoritma
- kebutuhan memori, lebar jalur komunikasi (*bandwidth*), waktu komputasi / eksekusi
- *A good algorithm is like a sharp knife - it does exactly what it is supposed to do with a minimum amount of applied effort [Cormen90].*
- memotong daging menggunakan tang ?

Menganalisis sebuah algoritma

- ukuran alat penyimpan data bukan lagi sebuah batasan yang utama (giga byte data, bahkan mencapai satuan terra byte, dengan ukuran fisik media yang cukup kecil)

- kebutuhan akan waktu komputasi / eksekusi kini menjadi tolak ukur yang lebih penting dalam penyusunan algoritma yang efektif dan efisien.

Superkomputer vs PC ?

- superkomputer : *insertion sort* vs *PC merge sort*.
- mengurutkan sebuah array 1 juta elemen (bilangan).
- superkomputer : 100 juta perintah/detik, *PC* 1 juta perintah/detik.
- *insertion sort* untuk superkomputer : kebutuhan $2n^2$ instruksi untuk mengurutkan n bilangan, sementara *merge sort* untuk *PC* dengan kebutuhan 50n log n instruksi.
- untuk mengurutkan 1 juta bilangan, superkomputer membutuhkan waktu :

$$\frac{2.(10^6)^2\, instruksi}{10^8\, instruksi/\det} = 20.000\det \approx 5,56\, jam$$

sedangkan PC membutuhkan waktu :

$$\frac{50.10^6 \log 10^6\, instruksi}{10^6\, instruksi/\det} \approx 1.000\det \approx 16,67 menit$$

PC bisa bekerja 20 kali lebih cepat

Minimalisasi waktu komputasi

- Metode-metode atau algoritma-algoritma yang fundamental :
  - algoritma rekursif
  - *divide and conquer*

Model Perhitungan Kebutuhan Waktu

- asumsi : mesin eksekusi yang digunakan adalah model komputasi dengan satu prosesor (*generic one-processor*).
- deretan instruksi di eksekusi satu per satu, tanpa adanya operasi konkuren.
- sistem komputasi lain : menggunakan model komputer paralel

Running time

- menghitung banyaknya operasi/instruksi yang dieksekusi
- Jika kita mengetahui besaran waktu untuk melaksanakan sebuah operasi tertentu, maka kita dapat menghitung berapa waktu sesungguhnya untuk melaksanakan algoritma tersebut.
- *running time* : fungsi dari ukuran input
- *running time* bisa dianggap setara dengan jumlah operasi primitif yang dieksekusi
- asumsi : waktu eksekusi untuk baris-baris algoritma yang berbeda dianggap sama.

```
procedure Average(input T[1..N] : array of integer, output avg : real)
{ Menghitung nilai rata-rata dari N bilangan integer yang tersimpan dalam array T.
  Nilai rata-rata akan disimpan di dalam variabel output avg.
  Asumsi : nilai N (jumlah elemen array) diketahui
}
Kamus
   sum : real
   i   : integer
```

Algoritma

```
   sum ← 0
   for i=1 to N
     sum ← sum + T[i]
   endfor
   avg ← sum/N   {nilai rata-rata}
```

Operasi :

Operasi pengisian nilai (`sum ← 0, sum ← sum + T[i], avg ← sum/N`), $t_1 = 1 + N + 1 = 2 + N$ operasi

- Operasi penjumlahan (`sum + T[i]`), $t_2 = N$ operasi
- Operasi pembagian (`sum/N`), $t_3 = 1$ operasi
- total kebutuhan waktu algoritma Average adalah :

  $t = t_1 + t_2 + t_3 = (2 + N)a + Nb + 1c$, dimana *a*, *b* dan *c* masing-masing adalah satu unit waktu yang dibutuhkan untuk melakukan operasi pengisian nilai, penjumlahan dan pembagian.

Kelemahan

- kebutuhan waktu yang nyata dari suatu algoritma tidak akan dapat diketahui apabila kebutuhan waktu dari masing-masing operasi dasar yang dieksekusi tidak diketahui.

- arsitektur komputer dengan *processor* dan *compiler* bahasa pemrogaman yang berbeda => ukuran unit waktu eksekusi yang berbeda pula.

Kompleksitas waktu, *T(n)*

- Analisis kompleksitas waktu, *T(n)*: hanya dilakukan pada operasi-operasi yang signifikan dalam algoritma.
- Identifikasi operasi-operasi yang paling signifikan di suatu algoritma.
- Misalnya, dalam algoritma Average, operasi yang paling penting adalah operasi penjumlahan `sum` $\leftarrow$ `sum + T[i]`. Jadi, kompleksitas waktu algoritma Average adalah $T(n) = n$.

Tiga jenis kompleksitas waktu

- $T_{max}(n)$: kompleksitas waktu untuk kasus terburuk (*worst case*), kebutuhan waktu maksimum yang mungkin diperlukan untuk kasus paling sulit

- $T_{min}(n)$: kompleksitas waktu untuk kasus terbaik (*best case*), kebutuhan waktu minimum yang mungkin diperlukan untuk kasus paling mudah

  $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata (*average case*), kebutuhan waktu rata-rata yang diperlukan untuk kasus yang rata-rata terjadi.

```
procedure SeqSearch(input T[1..N] : array of integer, x : integer, output index : integer)
{ Mencari integer x dalam array T. Bila ditemukan, index akan berisi posisi x dalam T, bila tidak ditemukan, index diisi 0.
Asumsi : nilai N (jumlah elemen array) diketahui
}
Kamus
   found : boolean
   i     : integer


Algoritma
   found ← false
   i ← 1
   while (not found) and (i≤N) do
     if (T[i]=x) then
        found ← true
     else
        i ← i + 1
     endif
   endwhile
   {found or i>N}

   if (found) then {x ditemukan}
       index ← i
   else              {x tidak ditemukan}
       index ← 0
```

SeqSearch

- Kasus terbaik : *x* berada pada elemen *T[1]*, sehingga $T_{min}(n)=1$.
- Kasus terburuk : *x* berada pada posisi terakhir dari *T*, atau *x* tidak ditemukan dalam *T*, sehingga $T_{max}(n)=n$.
- Kasus rata-rata, misalkan *x* berada pada posisi *j* dari *T*, *T[j]=x*, maka waktu yang dibutuhkan adalah *j*. Kompleksitas waktu rata-ratanya adalah :

$$T_{avg}(n)=\frac{1}{n}\sum_{j=1}^{n}T(j)=\frac{1+2+...+n}{n}=\frac{\frac{1}{2}n(n+1)}{n}=\frac{n+1}{2}$$

Kompleksitas waktu terburuk (*worst case*)

- Sangat penting : memberikan informasi tentang kebutuhan maksimum akan sumber daya yang mungkin diperlukan pada saat implementasi algoritma.

- *Worst case* secara alami sering terjadi dalam sebuah sistem, misalnya proses pencarian data yang tidak terdapat dalam sebuah database.

Kompleksitas Waktu Asimptotik

- Upaya perhitungan *running time* algoritma sampai tingkat presisi yang tinggi tidak selalu dibutuhkan.
- Untuk ukuran input yang relatif besar, berarti kita akan bekerja dengan kompleksitas asimptotik dari algoritma tersebut.
- Kompleksitas waktu asimptotik : kompleksitas waktu saat ukuran input menuju limit, meningkat tanpa batas.

```
procedure InsertionSort(input/output T[1..N] : array of integer)
{ Mengurutkan elemen-elemen pada T, dari elemen yang paling kecil
menuju yang paling besar. Elemen-elemen yang telah terurut tetap
tersimpan dalam T.
Asumsi : nilai N (jumlah elemen array) diketahui
}
Kamus
   key : integer
   i,j : integer
```

Algoritma

```
   for j=2 to N
     key ← T[j]
     {sisipkan T[j] ke elemen yang telah terurut T[1..j-1]}
     while (i>0) and (T[i]>key) do
         T[i+1] ← T[i]
         i ← i - 1
     endwhile
     T[i+1] ← key
   endfor
```

InsertionSort

- Kasus terbaik : $T(n) = an - b$, berupa fungsi linear,
- Kasus terburuk : $T(n) = an^2 + bn - c$, berupa fungsi kuadratik terhadap ukuran input (n).

- Untuk kompleksitas berupa fungsi kuadratik, *rate of growth / order of growth* untuk ukuran input yang besar hanya akan dipengaruhi secara signifikan oleh komponen kuadratiknya, yaitu n^2.

- Kita katakan bahwa $T(n)$ berorde n^2.
- Ketika kita hanya menggunakan batas atas (*upper bound*) asimptotik dari kompleksitas algoritma, maka kita menggunakan notasi "*O*", dikenal dengan notasi *Big-O*.

Definisi :

- Untuk sebuah fungsi g(n) :

O(g(n)) = {f(n) : terdapat konstanta c dan n_0 sehingga

$$0 \leq f(n) \leq cg(n)$$

untuk setiap $n \geq n_0$ }

- Contoh 1 :

Tunjukkan bahwa $T(n) = 3n + 2 = O(n)$.

Penyelesaian:

$3n + 2 = O(n)$ karena

$3n + 2 \leq 3n + 2n = 5n$

untuk semua $n \geq 1$ ($C = 5$ dan $n_0 = 1$).

- Contoh 2 :

Tunjukkan bahwa $T(n) = 2n^2 + 6n + 1 = O(n^2)$.

Penyelesaian:

$2n^2 + 6n + 1 = O(n^2)$

karena

$2n^2 + 6n + 1 \leq 2n^2 + 6n^2 + n^2 = 9n^2$ untuk semua $n \geq 1$.

- atau karena

$2n^2 + 6n + 1 \leq n^2 + n^2 + n^2 = 3n^2$ untuk semua $n \geq 6$

($C = 3$ dan $n_0 = 6$).

TEOREMA

Bila $T(n) = a_m\, n^m + a_{m-1}\, n^{m-1} + \ldots + a_1 n + a_0$ adalah polinom derajat m maka $T(n) = O(n^m)$.

TEOREMA

Misalkan $T_1(n) = O(f(n))$ dan $T_2(n) = O(g(n))$, maka

- $T_1(n) + T_2(n) = O(f(n)) + O(g(n)) = O(\max(f(n), g(n))$
- $T_1(n)T_2(n) = O(f(n))O(g(n)) = O(f(n)g(n))$
- $O(cf(n)) = O(f(n))$, c adalah konstanta
- $f(n) = O(f(n))$

- Contoh 3 :

Misalkan $T_1(n) = O(n)$ dan $T_2(n) = O(n^2)$, maka

(a) $T_1(n) + T_2(n) = O(\max(n, n^2)) = O(n^2)$

(b) $T_1(n)T_2(n) = O(n.n^2) = O(n^3)$

- Contoh 4 :

$O(5n^2) = O(n^2)$

$n^2 = O(n^2)$

- Contoh 5 :

$T(n) = (n + 2)\log(n^2 + 1) + 5n^2 = O(n^2)$

Penjelasannya adalah sebagai berikut:

$T(n) = (n + 2)\log(n^2 + 1) + 5n^2 = f(n)g(n) + h(n)$,

Kita rinci satu per satu:

$f(n) = (n + 2) = O(n)$

$g(n) = \log(n^2 + 1) = O(\log n)$, karena

$\log(n^2 + 1) \leq \log(2n^2)$

$\leq \log 2 + \log n^2$

$\leq \log 2 + 2 \log n \leq 3 \log n$ untuk $n > 2$

$h(n) = 5n^2 = O(n^2)$, maka $T(n) = (n + 2)\log(n^2 + 1) + 5n^2$

$= O(n)O(\log n) + O(n^2)$

$= O(n \log n) + O(n^2)$

$= O(\max(n \log n, n^2)) = O(n^2)$

Aturan Untuk Menentukan Kompleksitas Waktu Asimptotik

- Pengisian nilai (*assignment*), perbandingan, operasi aritmetik, *read*, *write* membutuhkan waktu $O(1)$
- Pengaksesan elemen array atau memilih *field* tertentu dari sebuah *record* membutuhkan waktu $O(1)$

read(x);	$O(1)$
x:=x + a[k];	$O(1) + O(1) + O(1) = O(1)$
writeln(x);	$O(1)$

Kompleksitas waktu asimptotik = $O(1) + O(1) + O(1) = O(1)$

Penjelasan: $O(1) + O(1) + O(1) = O(\max(1,1)) + O(1)$

$= O(1) + O(1)$

$= O(\max(1,1)) = O(1)$

- `if C then A1 else A2;` membutuhkan waktu $T_C + \max(T_{A1}, T_{A2})$

```
read(x);                          O(1)
if x mod 2 = 0 then               O(1)
 begin
   x:=x+1;                         O(1)
   writeln(x);                     O(1)
 end
else
   writeln(x);                     O(1)
```

- Kompleksitas waktu asimptotik:

$$= O(1) + O(1) + \max(O(1)+O(1), O(1))$$
$$= O(1) + \max(O(1), O(1))$$
$$= O(1) + O(1)$$
$$= O(1)$$

- *Loop* `for`. Kompleksitas waktu *loop* `for` adalah jumlah pengulangan dikali dengan kompleksitas waktu badan (*body*) *loop*

```
for i:=1 to n do
  jumlah:=jumlah + a[I];          O(1)
```

Kompleksitas waktu asimptotik = n . $O(1) = O(n .1) = O(n)$

for bersarang

```
for i:=1 to n do
   for j:=1 to n do
            a[i,j]:=0;    O(1)
```

Kompleksitas waktu asimptotik : $nO(n) = O(n.n) = O(n^2)$

- for bersarang dengan dua buah instruksi

```
for i:=1 to n do
   for j:=1 to i do
begin
            a:=a+1;   O(1)
            b:=b-2              O(1)
 end;
```

waktu untuk a:=a+1 : $O(1)$

waktu untuk b:=b-2 : $O(1)$

total waktu untuk badan loop = $O(1) + O(1) = O(1)$

loop terluar dieksekusi sebanyak n kali

loop terdalam dieksekusi sebanyak i kali, $i = 1, 2, \ldots, n$

jumlah pengulangan seluruhnya

$= 1 + 2 + \ldots + n = n(n + 1)/2$

kompleksitas waktu asimptotik

$= n(n + 1)/2 \,.O(1) = O(n(n + 1)/2) = O(n^2)$

- `while C do A;` dan `repeat A until C;` Untuk kedua buah *loop*, kompleksitas waktunya adalah jumlah pengulangan dikali dengan kompleksitas waktu badan C dan A.

  Loop tunggal sebanyak n-1 putaran

```
i:=2;                                   
                  O(1)
while i <= n do                         O(1)
 begin
   jumlah:=jumlah + a[i];               O(1)
   i:=i+1;                              O(1)
 end;
```

Kompleksitas waktu asimptotiknya adalah

$= O(1) + (n\text{-}1)\ \{ O(1) + O(1) + O(1) \}$

$= O(1) + (n\text{-}1)\ O(1)$

$= O(1) + O(n\text{-}1)$

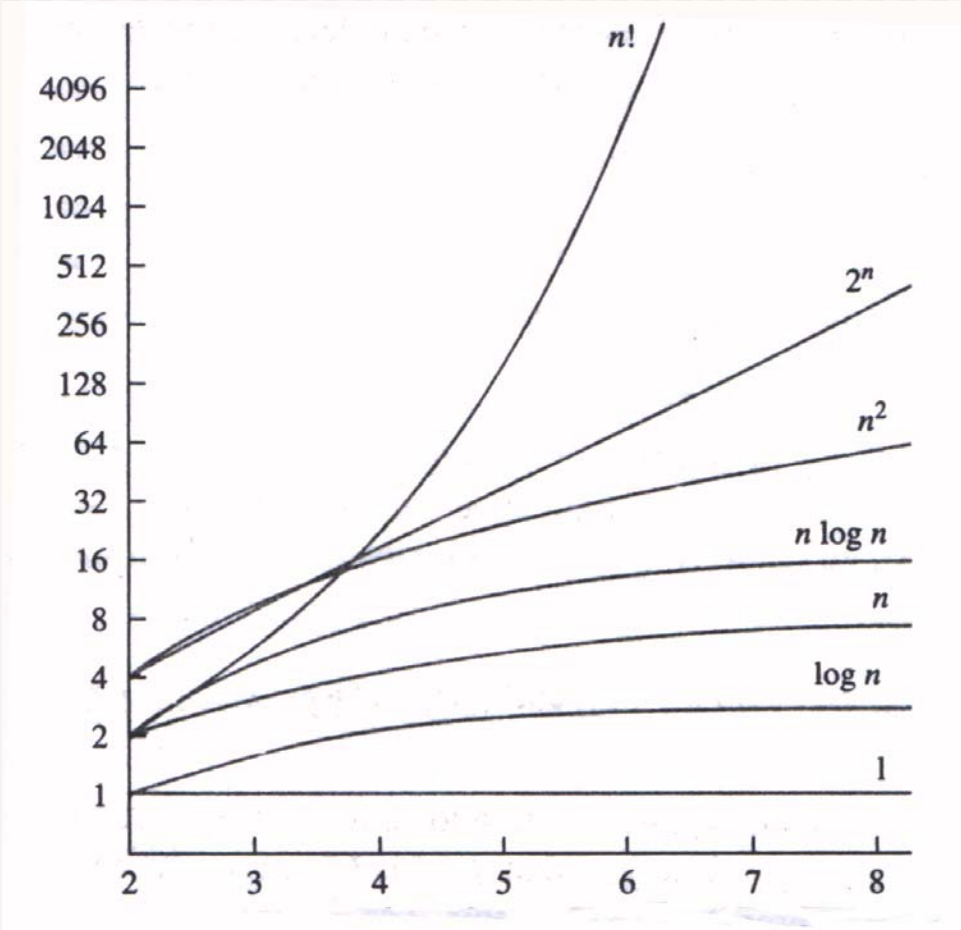
$= O(1) + O(n)$

$= O(n)$

- Prosedur dan fungsi : waktu yang dibutuhkan untuk memindahkan kendali ke rutin yang dipanggil adalah $O(1)$

Pengelompokan Algoritma Berdasarkan Notasi O-Besar

Kelompok Algoritma	Nama
$O(1)$	konstan
$O(\log n)$	logaritmik
$O(n)$	linear
$O(n \log n)$	$n \log n$
$O(n^2)$	kuadratik
$O(n^3)$	kubik
$O(2^n)$	eksponensial
$O(n!)$	faktorial



Spektrum kompleksitas waktu algoritma

$$\underbrace{O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < \ldots <}_{\text{algoritma polinomial}} \underbrace{O(2^n) < O(n!)}_{\text{algoritma eksponensial}}$$