Desain dan Analisis Algoritma

Kompleksitas Asimptotik

Kompleksitas Asimptotik

- Tinjau $T(n) = 2n^2 + 6n + 1$

Perbandingan pertumbuhan $T(n)$ dengan n^2

n	$T(n) = 2n^2 + 6n + 1$	n^2
10	261	100
100	20061	10000
1000	2.006.001	1.000.000
10.000	200.060.001	100.000.000

Kompleksitas Asimptotik

- Untuk n yang besar, pertumbuhan $T(n)$ sebanding dengan n^2. Pada kasus ini, $T(n)$ tumbuh seperti n^2 tumbuh.
- $T(n)$ tumbuh seperti n^2 tumbuh saat n bertambah. Kita katakan bahwa $T(n)$ berorde n^2 dan kita tuliskan

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

Notasi Asimptotik

- Big-oh (O)
- Big-omega (Ω)
- Big-theta (Θ)

Big Oh (O)

DEFINISI. $T(n) = O(f(n))$ (dibaca "$T(n)$ adalah $O(f(n)$" yang artinya $T(n)$ *berorde paling besar* $f(n)$) bila terdapat konstanta C dan n_0 sedemikian sehingga

$$T(n) \leq C(f(n))$$

untuk $n \geq n_0$.

$f(n)$ adalah batas atas (*upper bound*) dari $T(n)$ untuk n yang besar.

Atau

Kumpulan semua fungsi dengan tingkat pertumbuhan lebih kecil atau sama dengan f(n)

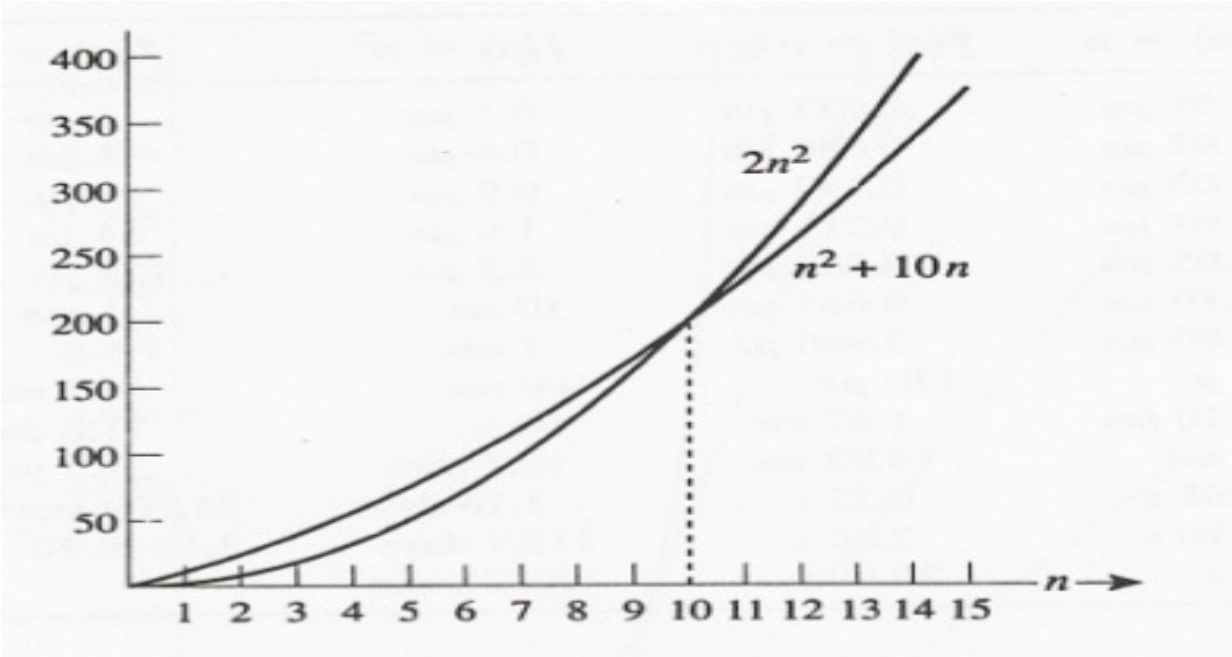
Contoh

Buktikan bahwa : $n^2 + 10n \in O(n^2)$

$T(n) \leq C.f(n)$

$n^2 + 10n \leq n^2 + n^2$

$n^2 + 10n \leq 2n^2$, untuk c=2 dan n_0=10 , $n \geq 10$



Cara Lain untuk Membuktikan

$n^2 + 10n \in O(n^2)$

$n^2 + 10n \leq n^2 + 10n^2$

$n^2 + 10n \leq 11n^2$, for $c=11$ and $n_0=0$, $n \geq 0$

Contoh 2

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

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

$2n^2 + 6n + 1 \leq 2n^2 + 6n^2 + n^2 = 9n^2$, untuk C=9 dan $n \geq 1$

Implikasi Notasi Big-Oh

- Misalkan kita mengetahui sebuah algoritma adalah O(f(n)).
- Ini artinya, untuk n yang besar, algoritma akan berhenti setelah melakukan operasi dasar paling banyak sebesar konstanta dikali f(n)
- Kita tahu bahwa sebuah operasi dasar membutuhkan waktu yang konstan dalam sebuah mesin.
- Jadi, algoritma membutuhkan konstanta kali f(n) unit waktu.

Pengelompokan Algoritma Berdasarkan Notasi *O*-Besar

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
$O(n!)$	faktorial

Urutan spektrum kompleksitas waktu algoritma adalah :

$$\underbrace{O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < \ldots <}\ \underbrace{O(2^n) < O(n!)}$$

algoritma polinomial algoritma eksponensial

Teorema

TEOREMA. Bila $T(n) = a_m n^m + a_{m-1} n^{m-1} + ... + a_1 n + a_0$ 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

(*a*) $T_1(n) + T_2(n) = O(f(n)) + O(g(n)) = O(\max(f(n), g(n))$

(*b*) $T_1(n)T_2(n) = O(f(n))O(g(n)) = O(f(n)g(n))$

(c) $O(cf(n)) = O(f(n))$, c adalah konstanta

(d) (*d*) $f(n) = O(f(n))$

Aturan Untuk Menentukan Kompleksitas Waktu Asimptotik

1. Jika kompleksitas waktu $T(n)$ dari algoritma diketahui,

   Contoh:

   (i) pada algoritma cari_maksimum

   $T(n) = n - 1 = O(n)$

   (ii) pada algoritma pencarian_beruntun

   $T_{min}(n) = 1 = O(1)$

   $T_{max}(n) = n = O(n)$

   $T_{avg}(n) = (n + 1)/2 = O(n)$

2. Menghitung *O*-Besar untuk setiap instruksi di dalam algoritma dengan panduan di bawah ini, kemudian menerapkan teorema *O*-Besar.

   a) Pengisian nilai (*assignment*), perbandingan, operasi aritmetik, *read*, *write* membutuhkan waktu $O(1)$.

   b) Pengaksesan elemen larik 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)
```

Lanjutan

Kompleksitas waktu asimptotik =

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

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

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

Lanjutan

(c) **if** C **then** S1 **else** S2; membutuhkan waktu

$$T_C + \max(T_{S1}, T_{S2})$$

contoh :

```
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)
```

Lanjutan

Kompleksitas waktu asimptotik:

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

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

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

$= O(1)$

Lanjutan

(d) Kalang **for**. Kompleksitas waktu kalang **for** adalah jumlah pengulangan dikali dengan kompleksitas waktu badan (*body*) kalang.

contoh :

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

Lanjutan

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

= $O(n$ .1)

= $O(n)$

Contoh for bersarang :

Kompleksitas waktu asimptotik:

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

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

Lanjutan

(e) while C do S; dan repeat S until C; Untuk kedua buah kalang, kompleksitas waktunya adalah jumlah pengulangan dikali dengan kompleksitas waktu badan C dan S.

Contoh: kalang 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;
```

Lanjutan

Kompleksitas waktu asimptotiknya adalah

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

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

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

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

$= O(n)$

Lanjutan

(f)Prosedur dan fungsi. Waktu yang dibutuhkan untuk memindahkan kendali ke rutin yang dipanggil adalah $O(1)$.

Tugas

1. Buktikan bahwa :

   a)½ n(n-1)∈ $O(n^2)$

   b)$2n^2$+ 9n ∈ $O(n^2)$

   c)$6.2^n + 2n^2 = O(2^n)$

2. Tentukan big O dari penjumlahan 2 buah matriks dari tugas sebelumnya.