

LAPORAN PRAKTIKUM 3

Analisis Algoritma (ANALGO)



Irfan Satrio Nugroho

140810180003

Kelas A

**Program Studi S-1 Teknik Informatika
Departemen Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Padjadjaran**

Pendahuluan

Minggu lalu kita sudah mempelajari menghitung kompleksitas waktu $T(n)$ untuk semua operasi yang ada pada suatu algoritma. Idealnya, kita memang harus menghitung semua operasi tersebut. Namun, untuk alasan praktis, kita cukup menghitung operasi abstrak yang **mendasari suatu algoritma**, dan memisahkan analisisnya dari implementasi. Contoh pada algoritma searching, operasi abstrak yang mendasarinya adalah operasi perbandingan elemen x dengan elemen-elemen dalam larik. Dengan menghitung berapa perbandingan untuk tiap-tiap elemen nilai sehingga kita dapat memperoleh **efisiensi relative** dari algoritma tersebut. Setelah mengetahui $T(n)$ kita dapat menentukan **kompleksitas waktu asimptotik** yang dinyatakan dalam notasi Big-O, Big-Ω, Big-Θ, dan little-ω.

Setelah mengenal macam-macam kompleksitas waktu algoritma (best case, worst case, dan average case), dalam analisis algoritma kita selalu mengutamakan perhitungan **worst case** dengan alasan sebagai berikut:

- Worst-case running time merupakan *upper bound* (batas atas) dari running time untuk input apapun. Hal ini memberikan jaminan bahwa algoritma yang kita jalankan tidak akan lebih lama lagi dari *worst-case*
- Untuk beberapa algoritma, *worst-case* cukup sering terjadi. Dalam beberapa aplikasi pencarian, pencarian info yang tidak ada mungkin sering dilakukan.
- Pada kasus *average-case* umumnya lebih sering seperti *worst-case*. Contoh: misalkan kita secara random memilih n angka dan mengimplementasikan insertion sort, *average-case* = *worst-case* yaitu fungsi kuadrat dari n .

Perhitungan worst case (*upper bound*) dalam kompleksitas waktu asimptotik dapat menggunakan **Big-O Notation**. Perhatikan pembentukan Big-O Notation berikut!

Misalkan kita memiliki kompleksitas waktu $T(n)$ dari sebuah algoritma sebagai berikut:

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

- Untuk n yang besar, pertumbuhan $T(n)$ sebanding dengan n^2
- Suku $6n + 1$ tidak berarti jika dibandingkan dengan $2n^2$, dan boleh diabaikan sehingga $T(n) = 2n^2 + \text{suku-suku lainnya}$.
- Koefisien 2 pada $2n^2$ boleh diabaikan, sehingga $T(n) = O(n^2)$ ➔ **Kompleksitas Waktu Asimptotik**

DEFINISI BIG-O NOTATION

Definisi 1. $T(n) = O(f(n))$ artinya $T(n)$ berorde paling besar $f(n)$ bila terdapat konstanta C dan n_0 sedemikian sehingga

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

Untuk $n \geq n_0$

Jika n dibuat semakin besar, waktu yang dibutuhkan tidak akan melebihi konstanta C dikalikan dengan $f(n)$, ➔ $f(n)$ adalah *upper bound*.

Dalam proses **pembuktian Big-O**, perlu dicari nilai n_0 dan nilai C sedemikian sehingga terpenuhi kondisi $T(n) \leq C \cdot f(n)$.

Contoh soal 1:

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

Penyelesaian:

Kita mengamati bahwa $n \geq 1$, maka $n \leq n^2$ dan $1 \leq n^2$ sehingga

$$2n^2 + 6n + 1 \leq 2n^2 + 6n^2 + n^2 = 9n^2, \text{ untuk } n \geq 1$$

Maka kita bisa mengambil $C=9$ dan $n_0=1$ untuk memperlihatkan:

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

BIG-O NOTATION DARI POLINOMIAL BERDERAJAT M

Big-O Notation juga dapat ditentukan dari Polinomial n berderajat m , dengan TEOREMA 1 sebagai berikut:

Polinomial n berderajat N dapat digunakan untuk memperkirakan kompleksitas waktu asimptotik dengan mengabaikan suku berorde rendah

Contoh: $T(n) = 3n^3 + 6n^2 + n + 8 = O(n^3)$, dinyatakan pada

TEOREMA 1

Bila $T(n) = a_N n^N + a_{N-1} n^{N-1} + a_1 n + a_0$ adalah polinom berderajat m maka $T(n) = O(n^N)$

Artinya kita mengambil suku paling tinggi derajatnya ("Mendominasi") yang diartikan laju pertumbuhannya lebih cepat dibandingkan yang lainnya ketika diberikan sembarang besaran input. Besaran dominan lainnya adalah:

- Eksponensial mendominasi sembarang perpangkatan (yaitu, $y^n > n^p, y > 1$)
- Perpangkatan mendominasi $\ln n$ (yaitu $n^p > \ln n$)
- Semua logaritma tumbuh pada laju yang sama (yaitu $^a \log(n) = ^b \log(n)$)
- $n \log n$ tumbuh lebih cepat daripada n tetapi lebih lambat dari n^2

Teorema lain dari Big-O Notation yang harus dihafalkan untuk membantu kita menentukan nilai Big-O dari suatu algoritma adalah:

TEOREMA 2

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

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

(ii) $T_1(n) \cdot T_2(n) = O(f(n) \cdot g(n))$

(b) $T_1(n) \cdot T_2(n) = O(f(n))O(g(n)) = O(f(n) \cdot g(n))$

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

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

Berikut adalah contoh soal yang mengaplikasikan Teorema 2 dari Big-O notation:

Contoh Soal 2

Misalkan, $T_1(n) = O(n)$ dan $T_2(n) = O(n^2)$, dan $T_3(n) = O(NN)$, dengan m sebagai peubah, maka

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

Teorema 2(a)(i)

(b) $T_2(n) + T_3(n) = O(n^2 + NN)$

Teorema 2(a)(ii)

(c) $T_1(n) \cdot T_2(n) = O(n \cdot n^2) = O(n^3)$

Teorema 2(B)

Contoh Soal 3

- (d) $O(5n^2) = O(n^2)$
(e) $n^2 = O(n^2)$

Teorema 2(C)
Teorema 2(D)

Aturan Menentukan Kompleksitas Waktu Asimptotik

• Cara 1

Jika kompleksitas waktu $T(n)$ dari algoritma sudah dihitung, maka kompleksitas waktu asimptotiknya dapat langsung ditentukan dengan mengambil suku yang mendominasi fungsi T dan menghilangkan koefisiennya (sesuai TEOREMA 1)

Contoh:

Pada algoritma cariMax, $T(n) = n - 1 = O(n)$

• Cara 2

Kita bisa langsung menggunakan notasi Big-O, dengan cara:

Pengisian nilai (assignment), perbandingan, operasi aritmatika (+, -, /, *, div, mod), read, write, pengaksesan elemen larik, memilih field tertentu dari sebuah record, dan pemanggilan function/void membutuhkan waktu $O(1)$

Contoh Soal 4:

Tinjau potongan algoritma berikut:

read(x) $O(1)$
 $x \leftarrow x + 1$ $O(1) + O(1) = O(1)$
write(x) $O(1)$

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

Penjelasan:

$$\begin{aligned} O(1) + O(1) + O(1) &= O(\text{MAX}(1,1)) + O(1) && \text{Teorema 2(A)(i)} \\ &= O(1) + O(1) \\ &= O(\text{MAX}(1,1)) && \text{Teorema 2(A)(ii)} \\ &= O(1) \end{aligned}$$

DEFINISI BIG-Ω DAN BIG-Θ NOTATION

Notasi Big-O hanya menyediakan batas atas (*upper bound*) untuk perhitungan kompleksitas waktu asimptotik, tetapi tidak menyediakan batas bawah (*lower bound*). Untuk itu, lower bound dapat ditentukan dengan Big-Ω Notation dan Big-θ Notation.

Definisi Big-Ω Notation:

$T(n) = \Omega(g(n))$ yang artinya $T(n)$ berorde paling kecil $g(n)$ bila terdapat konstanta C dan n_0 sedemikian sehingga

$$T(n) \geq C \cdot (g(n))$$

untuk $n \geq n_0$

Definisi Big-θ Notation:

$T(n) = \theta(h(n))$ yang artinya $T(n)$ berorde sama dengan $h(n)$ jika $T(n) = O(h(n))$ dan $T(n) = \Omega(g(n))$

Contoh Soal 5:

Tentukan Big-Ω dan Big-Θ Notation untuk $T(n) = 2n^2 + 6n + 1$

Penyelesaian:

Karena $2n^2 + 6n + 1 \geq 2n^2$ untuk $n \geq 1$, dengan mengambil $C=2$, kita memperoleh

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

Karena $2n^2 + 6n + 1 = O(n^2)$ dan $2n^2 + 6n + 1 = G(n^2)$, maka $2n^2 + 6n + 1 = \Theta(n^2)$

Penentuan Big-Ω dan Big-Θ dari Polinomial Berderajat m

Sebuah fakta yang berguna dalam menentukan orde kompleksitas adalah dari suku tertinggi di dalam polinomial berdasarkan teorema berikut:

TEOREMA 3

Bila $T(n) = a_N n^N + a_{N-1} n^{N-1} + a_1 n + a_0$ adalah polinom berderajat m maka $T(n) = \Theta(n^N)$

Contoh soal 6:

$$\text{Bila } T(n) = 6n^4 + 12n^3 + 24n + 2,$$

maka $T(n)$ adalah berorde n^4 , yaitu $O(n^4)$, $\Omega(n^4)$, dan $\Theta(n^4)$.

Latihan Analisa

Minggu ini kegiatan praktikum difokuskan pada latihan menganalisa, sebagian besar tidak perlu menggunakan komputer dan mengkode program, gunakan pensil dan kertas untuk menjawab persoalan berikut!

1. Untuk $T(n) = 2 + 4 + 8 + 16 + \dots + 2^n$, tentukan nilai C , $f(n)$, n_0 , dan notasi Big-O sedemikian sehingga $T(n) = O(f(n))$ jika $T(n) \leq C$ untuk semua $n \geq n_0$

$$\begin{aligned} \textcircled{1} T(n) &= 2 + 4 + 8 + 16 + \dots + 2^n \\ &= \frac{2(2^{n+1} - 1)}{2 - 1} = 2(2^{n+1} - 1) = 2^{n+1} - 2 \\ T(n) &= 2^{n+1} - 2 = O(2^n) \quad | \quad n_0 = 1 \\ T(n) &\leq C f(n) \quad | \quad 2 - \frac{2}{2} \leq C \\ 2^{n+1} - 2 &\leq C \cdot 2^n \quad | \quad 1 \leq C \\ 2 - \frac{2}{2^n} &\leq C \end{aligned}$$

2. Buktikan bahwa untuk konstanta-konstanta positif p , q , dan r :
 $T(n) = pn^2 + qn + r$ adalah $O(n^2)$, $\Omega(n^2)$, dan $\Theta(n^2)$

$$\begin{aligned} \textcircled{2} T(n) &= pn^2 + qn + r \\ O(n^2) &\rightarrow \text{Big O} \\ T(n) &\leq C_f(n) \\ pn^2 + qn + r &\leq C \cdot n^2 \\ p + \frac{q}{n} + \frac{r}{n^2} &\leq C \end{aligned} \quad \left| \begin{array}{l} n_0 = 1 \\ p + q + r \leq C \end{array} \right.$$

$$\begin{aligned} \bullet \Omega(n^2) &\rightarrow \text{Big } \Omega \\ T(n) &\geq C_g(n) \\ pn^2 + qn + r &\geq C \cdot n \\ np + q + \frac{r}{n} &\geq C \end{aligned} \quad \left| \begin{array}{l} n_0 = 1 \\ p + q + r \geq C \end{array} \right.$$

• Big $\Theta(n^2) \rightarrow$ dikarenakan hasil kedua
 big O & Ω berderajat sama, maka
 $\Theta(n^2)$ dinyatakan benar.

3. Tentukan waktu kompleksitas asimtotik (Big-O, Big- Ω , dan Big- Θ) dari kode program berikut:

```
for k ← 1 to n do
  for i ← 1 to n do
    for j ← 1 to n do
      wij ← wij or wik and wkj
    endfor
  endfor
endfor
```

③ For k ← 1 to n do
 For i ← 1 to n do
 For j ← 1 to n do
 w_{ij} ← w_{ij} or w_{ik} and w_{kj}
 endfor
endfor
endfor

$T(n) = n^3$

$$\begin{array}{ll} \bullet \text{Big O} & \bullet \text{Big } \Omega \\ n^3 \leq Cn^2 & n^3 \geq Cn \\ 1 \leq C & 1 \geq C \end{array}$$

• Big $\Theta \rightarrow$ karena $O(n^3) \neq \Omega(n^3)$
 Sederajat sama maka $\Theta(n^3)$ benar.

4. Tulislah algoritma untuk menjumlahkan dua buah matriks yang masing-masing berukuran $n \times n$. Berapa kompleksitas waktunya $T(n)$? dan berapa kompleksitas waktu asimptotiknya yang dinyatakan dalam Big-O, Big-Ω, dan Big-Θ?

④ algoritma Penjumlahan matriks $n \times n$

For $i \leftarrow 1$ to n do

For $j \leftarrow 1$ to n do

$m_{ij} \leftarrow a_{ij} + b_{ij} \rightarrow n \cdot n$

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

• Big O

$$n^2 \leq cn^2$$

$$1 \leq c$$

• Big Ω

$$n^2 \geq cn^2$$

$$1 \geq c$$

• Big Θ → karena $O(n^2)$ & $\Omega(n^2)$ sederhana
maka $\Theta(n^2)$

5. Tulislah algoritma untuk menyalin (copy) isi sebuah larik ke larik lain. Ukuran elemen larik adalah n elemen. Berapa kompleksitas waktunya $T(n)$? dan berapa kompleksitas waktu asimptotiknya yang dinyatakan dalam Big-O, Big-Ω, dan Big-Θ?

⑤ Copy larik

For $i \leftarrow 1$ to n do

$a_i \leftarrow b_i \rightarrow n \rightarrow T(n) = n$

endfor

• Big O

$$n \leq cn$$

$$1 \leq c$$

• Big Ω

$$n \geq cn$$

$$1 \geq c$$

• Big Θ

$$O(n) = \Omega(n)$$

sederajat maka

$$\Theta(n)$$

6. Diberikan algoritma Bubble Sort sebagai berikut:

```

procedure BubbleSort(input/output  $a_1, a_2, \dots, a_n$ ; integer)
{ Mengurut tabel integer TabInt[1..n] dengan metode pengurutan bubble-
sort
  Masukan:  $a_1, a_2, \dots, a_n$ 
  Keluaran:  $a_1, a_2, \dots, a_n$  (terurut menaik)
}
Deklarasi
  k : integer { indeks untuk traversal tabel }
  pass : integer { tahapan pengurutan }
  temp : integer { peubah bantu untuk pertukaran elemen tabel }
Algoritma
  for pass ← 1 to n - 1 do
    for k ← n downto pass + 1 do
      if  $a_k < a_{k-1}$  then
        { pertukarkan  $a_k$  dengan  $a_{k-1}$  }
        temp ←  $a_k$ 
         $a_k$  ←  $a_{k-1}$ 
         $a_{k-1}$  ← temp
      endif
    endfor
  endfor

```

- Hitung berapa jumlah operasi perbandingan elemen-elemen tabel!
- Berapa kali maksimum pertukaran elemen-elemen tabel dilakukan?
- Hitung kompleksitas waktu asimtotik (Big-O, Big-Ω, dan Big-Θ) dari algoritma Bubble Sort tersebut!

a. Jumlah op. Perbandingan
 $1+2+3+4+\dots+(n-1) = \frac{n(n-1)}{2}$ kali

b. Berapa kali mak. pertukaran elmn tabel terjadi
 $\frac{n(n-1)}{2}$ kali

c. Kompleksitas

* best case (semua data sudah berurut)
 $\approx \frac{(n-1)n}{2}$ kali, $T_{\min}(n) = \frac{n(n-1)}{2} \approx \frac{n^2-n}{2}$

* worst case (data terurut terbalik)
 Perbandingan $\rightarrow \frac{n(n-1)}{2}$
 assignment $\rightarrow \frac{3n(n-1)}{2}$ $T_{\max} = \frac{4n(n-1)}{2} = 2n^2 - 2n$

• Big O
 $2n^2 - 2n \leq C(n^2)$
 $2 - \frac{2}{n} \leq C$
 $n_0 = 1 \rightarrow 2 - 2 \leq C$
 $0 \leq C$

• Big Ω
 $\frac{n^2-n}{2} \geq Cn^2$
 $\frac{1}{2} - \frac{1}{2n} \geq C$
 $n_0 = 1 \rightarrow \frac{1}{2} - \frac{1}{2} \geq C$
 $0 \geq C$

7. Untuk menyelesaikan problem X dengan ukuran N tersedia 3 macam algoritma:

- (a) Algoritma A mempunyai kompleksitas waktu $O(\log N)$
- (b) Algoritma B mempunyai kompleksitas waktu $O(N \log N)$
- (c) Algoritma C mempunyai kompleksitas waktu $O(N^2)$

Untuk problem X dengan ukuran $N=8$, algoritma manakah yang paling cepat? Secara asimptotik, algoritma manakah yang paling cepat?

- ⑦
- a. Algoritma A $\rightarrow O(\log N)$
 - b. Algoritma B $\rightarrow O(N \log N)$
 - c. Algoritma C $\rightarrow O(N^2)$

misal $N=8$

- a. A $\rightarrow O(\log 8) = O(3 \log 2)$
- b. B $\rightarrow O(8 \log 8) = O(24 \log 2)$
- c. C $\rightarrow O(8^2) = O(64)$

dari 3 algoritma diatas yang paling efektif adalah (a) karena nilai nya paling kecil

8. Algoritma mengevaluasi polinom yang lebih baik dapat dibuat dengan metode Horner berikut:

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \dots + x(a_{n-1} + a_n x)))) \dots)$$

function p2(input x : real) \rightarrow real
(Mengembalikan nilai $p(x)$ dengan metode Horner)

Deklarasi

k : integer
 b_1, b_2, \dots, b_n : real

Algoritma

$b_n \leftarrow a_n$
for k \leftarrow n - 1 downto 0 do
 $b_k \leftarrow a_k + b_{k+1} * x$
endfor
return b_0

Hitunglah berapa operasi perkalian dan penjumlahan yang dilakukan oleh algoritma diatas, Jumlahkan kedua hitungan tersebut, lalu tentukan kompleksitas waktu asimptotik (Big-O)nya. Manakah yang terbaik, algoritma p atau p^2

⑧ Operasi Assignment

$b_n \leftarrow a_n$ \rightarrow 1 kali

$b_k \leftarrow a_k + b_{k+1} * x$ \rightarrow n kali

$$T(n) = n + 1$$

$O(n) =$ untuk p^2

Algoritma P

Penjumlahan : n kali

Pertalian : n kali

$$T(n) = 2n$$

At. P^2 lebih baik

