Kompleksitas Algoritma

Sesi 14

1

Pendahuluan

- Sebuah algoritma tidak saja harus benar, tetapi juga harus mangkus (*efisien*).
- Algoritma yang mangkus ialah algoritma yang meminimumkan kebutuhan waktu dan ruang.
- Kemangkusan algoritma diukur dari berapa jumlah waktu dan ruang (*space*) memori yang dibutuhkan untuk menjalankannya. ➔ **Analisis Algoritma**
- 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 terbaik.

2

Analisis Algoritma

- Tujuan
  - Mengukur efisiensi sebuah algoritma.
  - Membanding-bandingkan dua/lebih algoritma untuk masalah yang sama.
- Efisiensi diukur dari diukur dari: waktu (*time*) dan memori (*space*).
- Dua beasaran yang digunakan: kompleksitas algoritma
  1. Kompleksitas waktu – T(n)
  2. Kompleksitas ruang – S(n)
- Independen dari spesifikasi komputer dan *compiler*

3

Ukuran input (input's size)

- Hampir seluruh algoritma memiliki waktu eksekusi lebih lama jika mendapat input yang berukuran besar
- Ukuran masukan (n): jumlah data yang diproses oleh sebuah algoritma.
- Contoh:
  - Algoritma pengurutan 1000 elemen larik, maka $n = 1000$.
  - Algoritma *TSP* pada sebuah graf lengkap dengan 100 simpul, maka $n = 100$.
  - Algoritma perkalian 2 buah matriks berukuran 50 x 50, maka $n = 50$.
- Dalam praktek perhitungan kompleksitas, ukuran masukan dinyatakan sebagai variabel n saja.

4

Kompleksitas Waktu

- Kompleksitas waktu, T(n),
  - Diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n.
  - Dihitung dari jumlah operasi dasar yang dilakukan di dalam algoritma sebagai fungsi ukuran masukan (n). Asumsi: setiap operasi dasar membutuhkan waktu konstan.
- **Operasi dasar (Basic operation)**
  - The most important operation of the algorithm
  - The most time-consuming operation in the algorithm's innermost loop
  - Contoh:
    - Penjumlahan, pengurangan, perkalian
    - Perbandingan
    - Pengkasesan memori (mis: akses elemen array)
    - Pembacaan, Penulisan, dll
- Dalam praktek, **hanya dihitung jumlah sebuah operasi dasar yang khas (tipikal) dari suatu algoritma**.

5

Contoh operasi khas di dalam algoritma

- Algoritma pencarian di dalam larik
  - Operasi khas: perbandingan elemen larik
- Algoritma pengurutan
  - Operasi khas: perbandingan elemen, penukaran elemen
- Algoritma penjumlahan 2 buah matriks
  - Operasi khas: penjumlahan
- Algoritma perkalian 2 buah matriks
  - Operasi khas: perkalian dan penjumlahan

6

Contoh Perhitungan Komplesitas Waktu (1)

- Menghitung rata-rata nilai dalam sebuah array/larik 1 dimensi

a_1	a_2	a_3	...	a_n

Larik bilangan bulat

```
procedure HitungRata_Rata(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 }
```

7

Contoh Perhitungan Kompleksitas Waktu (2)

- Operasi pengisian nilai (jumlah←0, k←1, jumlah←jumlah+a_k, k←k+1, dan r ← jumlah/n)
  - Jumlah seluruh operasi pengisian nilai :
    $t_1 = 1 + 1 + n + n + 1 = 3 + 2n$
- Operasi penjumlahan (jumlah+a_k, dan k+1)
  - Jumlah seluruh operasi penjumlahan : $t_2 = n + n = 2n$
- Operasi pembagian (jumlah/n)
  - Jumlah seluruh operasi pembagian : $t_3 = 1$
- Total kebutuhan waktu algoritma HitungRerata:

$t = t_1 + t_2 + t_3$

$= (3 + 2n)a + 2nb + c$ detik

8

Kompleksitas Waktu

- Dibedakan atas tiga macam :

1. $T_{max}(n)$: kompleksitas waktu untuk kasus terburuk *(worst case)*, → kebutuhan waktu maksimum.
2. $T_{min}(n)$: kompleksitas waktu untuk kasus terbaik *(best case)*, → kebutuhan waktu minimum.
3. $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata *(average case)* → kebutuhan waktu secara rata-rata

9

Contoh : Algoritma *sequential search (1)*

```
procedure PencarianBeruntun(input a1, a2, ..., an : integer, x :
   integer,output idx : integer)
Deklarasi
  k : integer
  ketemu : boolean   { bernilai true jika x ditemukan atau false
  jika x tidak ditemukan }
Algoritma:
  k←1
  ketemu ← false
  while (k ≤ n) and (not ketemu) do
    if ak = x then  ketemu←true
    else  k ← k + 1
    endif
  endwhile
  { k > n or ketemu }
  if ketemu then idx←k  { x ditemukan }
  else idx← 0 { x tidak ditemukan }
  endif
```

10

Contoh : Algoritma *sequential search (2)*

Jumlah operasi perbandingan elemen tabel:

1.*Kasus terbaik*: ini terjadi bila $a_1 = x$.

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

2.*Kasus terburuk*: bila $a_n = x$ atau x tidak ditemukan.

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

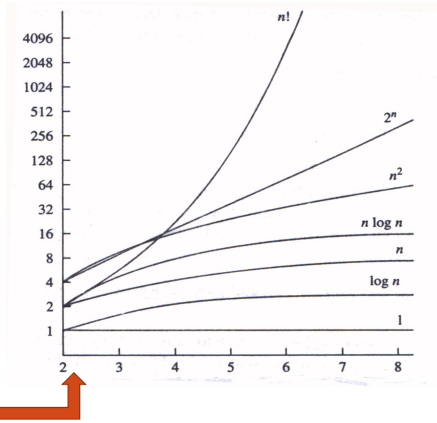
3.*Kasus rata-rata*: Jika x ditemukan pada posisi ke-k, maka operasi perbandingan ($a_k = x$) akan dieksekusi sebanyak k kali.

$$T_{\text{avg}}(n) = \frac{(1+2+3+\ldots+n)}{n} = \frac{\frac{1}{2}n(1+n)}{n} = \frac{(n+1)}{2}$$

11

Kompleksitas waktu asimptotik

- Dalam praktek, nilai T(n) yang eksak tidak terlalu penting.
  - Misal: T(n) = n(n – 1)/2
- Yang lebih penting: berapa *laju* peningkatan T(n) ketika n membesar
  - Contoh: T(n) = n dan T(n) = 2n → laju pertumbuhan T(n) sama saja → satu kategori algoritma efisien
- Tetapi kita membedakan antara T(n) = n, T(n) = exp(n), T(n) = log(n)



12

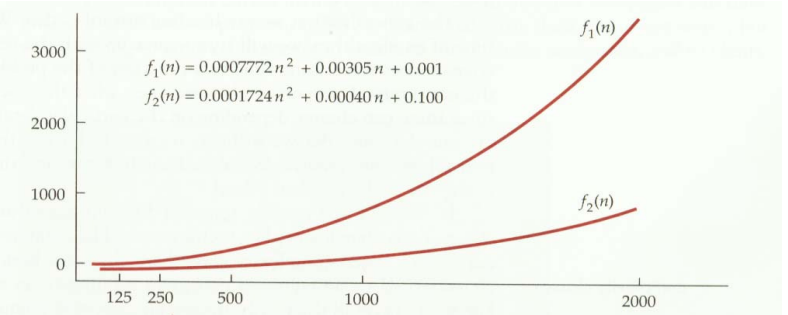
Intro to asymptotic

- $f(n)=an^2 + bn + c$
- Coefficients (a,b,c) associated with particular
  - computers,
  - languages, and
  - compilers

Array Size n	Home Computer	Desktop Computer
125	12.5	2.8
250	49.3	11.0
500	195.8	43.4
1000	780.3	172.9
2000	3114.9	690.5

13

Fitting Curves to the Data



$f_1(n)=0.0007772\,n^2 + 0.00305n + 0.001$ → home computer data

$f_2(n)=0.0001724\,n^2 + 0.00040n + 0.100$ → desktop computer data

14

Kompleksitas Waktu 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	2061	1000
1000	2.006.001	1.000.000
10.000	2.000.060.001	1.000.000.000

- Untuk n yang besar, pertumbuhan $T(n)$ sebanding dengan n^2. $T(n)$ tumbuh seperti n^2 tumbuh.
- $T(n)$ tumbuh seperti n^2 tumbuh saat n bertambah. Ditulis:
  $T(n) = O(n^2)$
- Notasi "O" disebut notasi "O-Besar" (*Big-O*) yang merupakan notasi **kompleksitas waktu asimptotik**

15

Introducing the language of O-notation

- Dominant terms
- Ignoring the constant of proportionality

$f(n) = an^2 + bn + c$
where $a = 0.0001724$, $b = 0.0004$ and $c = 0.1$

n	f(n)	an^2	n^2-term as % of total
125	2.8	2.7	94.7
250	11.0	10.8	98.2
500	43.4	43.1	99.3
1000	172.9	172.4	99.7
2000	690.5	689.6	99.9

16

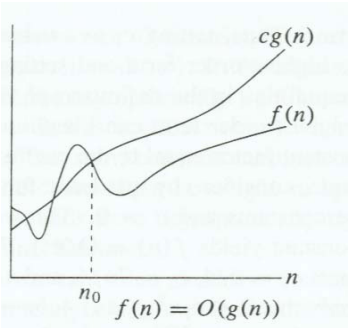
Asymptotic Notations

- Big-oh (O)
- Big-omega (Ω)
- Big-theta (Θ)
- Little-oh (o)
- Little-omega (ω)

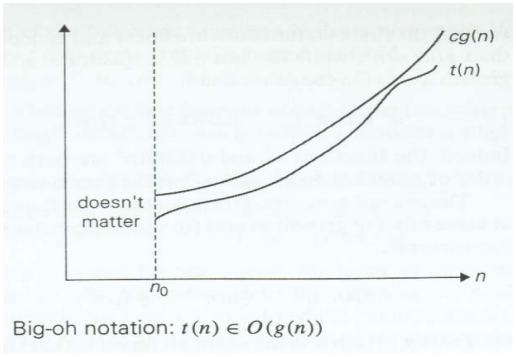
17

O-notation

- $O(g(n)) = \{f(n)$: there exist positive constants c and nonnegative integer n_o such that
  $0 \le f(n) \le cg(n)$
  for all $n \ge n_o\}$
- $f(n) = O(g(n))$ indicates that **$f(n)$ is a member of the set $O(g(n))$**



18



19

Contoh

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

Penyelesaian:

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

karena

$2n^2 + 6n + 1 \le 2n^2 + 6n^2 + n^2 = 9n^2$ untuk semua $n \ge 1$
($C = 9$ dan $n_0 = 1$).

atau karena

$2n^2 + 6n + 1 \le n^2 + n^2 + n^2 = 3n^2$ untuk semua $n \ge 6$
($C = 3$ dan $n_0 = 6$).

20

Contoh

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

Penyelesaian:

$3n + 2 = O(n)$

karena

$3n + 2 \le 3n + 2n = 5n$ untuk semua $n \ge 1$

($C = 5$ dan $n_0 = 1$).

21

Contoh

Tunjukkan bahwa $T(n) = 5 = O(1)$.

Penyelesaian:

- $5 = O(1)$ karena $5 \le 6.1$ untuk $n \ge 1$.
  ($C = 6$ dan $n_0 = 1$)
- Kita juga dapat memperlihatkan bahwa
  $5 = O(1)$ karena $5 \le 10 \cdot 1$ untuk $n \ge 1$

22

Contoh

Tunjukkan bahwa kompleksitas waktu algoritma pengurutan seleksi (*selection sort*) adalah $T(n) = n(n-1)/2 = O(n^2)$.

Penyelesaian:

- $n(n-1)/2 = O(n^2)$ karena
  $n(n-1)/2 \le n^2/2 + n^2/2 = n^2$
  untuk semua $n \ge 1$ ($C = 1$ dan $n_0 = 1$).

Tunjukkan $T(n) = 6*2^n + 2n^2 = O(2^n)$

Penyelesaian:

- $6*2^n + 2n^2 = O(2^n)$ karena
  $6*2^n + 2n^2 \le 6*2^n + 2*2^n = 8*2^n$
  untuk semua $n \ge 1$ ($C = 8$ dan $n_0 = 1$).

23

Contoh

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

Penyelesaian:

$1 + 2 + .. + n \le n + n + \ldots + n = n^2$ untuk $n \ge 1$

Tunjukkan $T(n) = n! = O(n^n)$

Penyelesaian:

$n! = 1 \cdot 2 \cdot \ldots \cdot n \le n \cdot n \cdot \ldots \cdot n = n^n$ untuk $n \ge 1$

24

Teorema 1 (1)

- Bila $T(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ adalah polinom derajat m maka $T(n) = O(n^m)$.
- Jadi, cukup melihat suku (*term*) yang mempunyai pangkat terbesar.
- Contoh:

  $T(n) = 5 = 5n^0 = O(n^0) = O(1)$

  $T(n) = n(n-1)/2 = n^2/2 - n/2 = O(n^2)$

  $T(n) = 3n^3 + 2n^2 + 10 = O(n^3)$

25

Teorema 1 (2)

- Teorema tersebut digeneralisasi untuk suku dominan lainnya:
  1. Eksponensial mendominasi sembarang perpangkatan (yaitu, $y^n > n^p, y > 1$)
  2. Perpangkatan mendominasi ln n (yaitu $n^p > \ln n$)
  3. Semua logaritma tumbuh pada laju yang sama (yaitu $a \log(n) = b \log(n)$)
  4. $n \log n$ tumbuh lebih cepat daripada n tetapi lebih lambat daripada n^2

  Contoh: $T(n) = 2^n + 2n^2 = O(2^n)$.
  $T(n) = 2n \log(n) + 3n = O(n \log(n))$
  $T(n) = \log(n^3) = 3 \log(n) = O(\log(n))$
  $T(n) = 2n \log(n) + 3n^2 = O(n^2)$

26

Perhatikan....(1)

- Tunjukkan bahwa $T(n) = 5n^2 = O(n^3)$, tetapi $T(n) = n^3 \neq O(n^2)$.

Penyelesaian:

- $5n^2 = O(n^3)$ karena $5n^2 \leq n^3$ untuk semua $n \geq 5$.
- Tetapi, $T(n) = n^3 \neq O(n^2)$ karena tidak ada konstanta C dan n^0 sedemikian sehingga

  $n^3 \leq Cn^2 \Leftrightarrow n \leq C$ untuk semua n_0 karena n dapat berupa sembarang bilangan yang besar.

27

Perhatikan ...(2)

- Definisi: $T(n) = O(f(n))$ jika terdapat C dan n_0 sedemikian sehingga $T(n) \leq C.f(n)$ untuk $n \geq n_0$

  → tidak menyiratkan seberapa atas fungsi f itu.
- Jadi, menyatakan bahwa

  $T(n) = 2n^2 = O(n^2)$ → benar

  $T(n) = 2n^2 = O(n^3)$ → juga benar

  $T(n) = 2n^2 = O(n^4)$ → juga benar
- Namun, untuk alasan praktis kita memilih fungsi yang sekecil mungkin agar $O(f(n))$ memiliki makna
- Jadi, kita menulis $2n^2 = O(n^2)$, bukan $O(n^3)$ atau $O(n^4)$

28

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.

29

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 :

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

30

Kegunaan Notasi *Big-Oh*

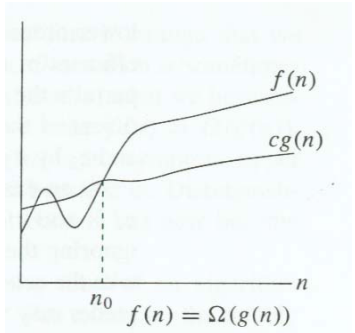
- Notasi *Big-Oh* berguna untuk membandingkan beberapa algoritma dari untuk masalah yang sama
  → menentukan yang terbaik.
- Contoh: masalah pengurutan memiliki banyak algoritma penyelesaian,
  Selection sort, *insertion sort* → $T(n) = O(n^2)$
  Quicksort → $T(n) = O(n \log n)$

Karena $n \log n < n^2$ untuk n yang besar, maka algoritma *quicksort* lebih cepat (lebih baik, lebih mangkus) daripada algoritma *selection sort* dan *insertion sort*.

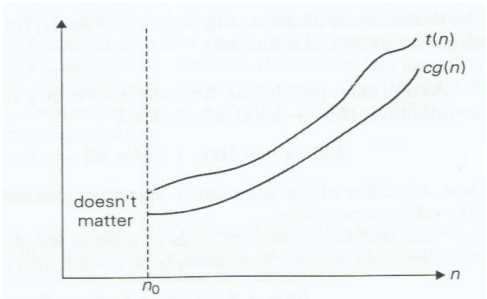
31

Ω-notation

- $\Omega(g(n)) = \{f(n)$: there exist positive constants c and nonnegative integer n_o such that
  $f(n) \geq cg(n) \geq 0$
  for all $n \geq n_o\}$
- $f(n) = \Omega(g(n))$ indicates that **$f(n)$ is a member of the set $\Omega(g(n))$**



32

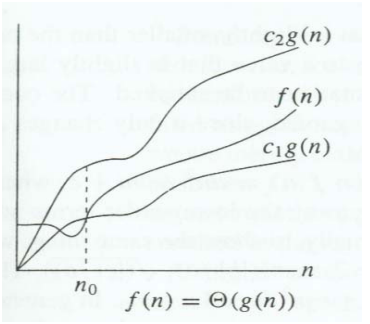


Big-omega notation: $t(n) \in \Omega(g(n))$

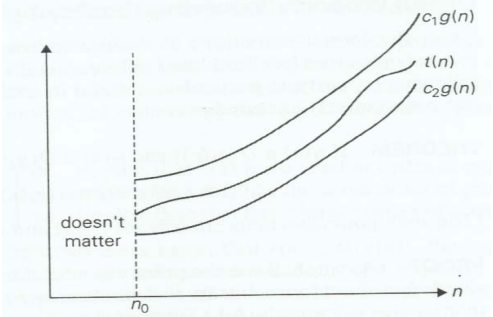
33

Θ-notation

- $\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and nonnegative integer n_o such that $c_2 g(n) \geq f(n) \geq c_1 g(n) \geq 0$ for all $n \geq n_o\}$
- $f(n) = \Theta(g(n))$ indicates that **$f(n)$ is a member of the set $\Theta(g(n))$**



34



Big-theta notation: $t(n) \in \Theta(g(n))$

35

Contoh

Tentukan notasi Ω dan Θ untuk $T(n) = 2n^2 + 6n + 1$.

Penyelesaian:

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

$$2n^2 + 6n + 1 = \Omega(n^2)$$

Karena $2n^2 + 5n + 1 = O(n^2)$ dan $2n^2 + 6n + 1 = \Omega(n^2)$,
maka $2n^2 + 6n + 1 = \Theta(n^2)$.

36

Contoh

Tunjukkan bahwa kompleksitas waktu algoritma pengurutan seleksi (*selection sort*) adalah $T(n) = n(n-1)/2 = \Theta(n^2)$.

Penyelesaian:

- $n(n-1)/2 = O(n^2)$ karena
  $n(n-1)/2 \leq n^2/2 + n^2/2 = n^2$
  untuk semua $n \geq 1$ ($C = 1$ dan $n_0 = 1$).
- $n(n-1)/2 = \Omega(n^2)$ karena
  $n(n-1)/2 \geq n^2/4 - n^2/8 = n^2/8$ untuk $n \geq 2$
- Karena $n(n-1)/2 = O(n^2)$ dan $n(n-1)/2 = \Omega(n^2)$ maka $n(n-1)/2 = \Theta(n^2)$.

37

Contoh

Tentukan notasi notasi O, Ω dan Θ untuk $T(n) = 5n^3 + 6n^2 \log n$.

Penyelesaian:

Karena $0 \leq 6n^2 \log n \leq 6n^3$, maka $5n^3 + 6n^2 \log n \leq 11n^3$ untuk $n \geq 1$. Dengan mengambil $C = 11$, maka

$$5n^3 + 6n^2 \log n = O(n^3)$$

Karena $5n^3 + 6n^2 \log n \geq 5n^3$ untuk $n \geq 1$, maka maka dengan mengambil $C = 5$ kita memperoleh

$$5n^3 + 6n^2 \log n = \Omega(n^3)$$

Karena $5n^3 + 6n^2 \log n = O(n^3)$ dan $5n^3 + 6n^2 \log n = \Omega(n^3)$, maka $5n^3 + 6n^2 \log n = \Theta(n^3)$

38

Implikasi Notasi Big-Ω

- Misalkan kita mengetahui sebuah algoritma adalah $\Omega(g(n))$.
- Ini artinya, untuk n yang besar, terdapat paling sedikit satu masukan dimana algoritma melakukan operasi dasar paling sedikit sebesar konstanta dikali g(n)

39

ω-notation

- $\omega(g(n)) = \{f(n)$: for any positive constants $c > 0$, there exist a constant $n_o > 0$ such that
  $f(n) > cg(n) \geq 0$
  for all $n \geq n_o\}$
- Example: $n^2/2 = \omega(n)$, but $n^2/2 \neq \omega(n^2)$

40

Property of asymptotic

- If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then
  $f_1(n) + f_2(n) \in O(max(g_1(n), g_2(n)))$

- If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then
  $f_1(n) * f_2(n) \in O(g_1(n) * g_2(n))$

(also work for Θ and Ω)

Transitivity:

$$
\begin{array}{llll}
f(n) = \Theta(g(n)) & \text{and } g(n) = \Theta(h(n)) & \text{imply} & f(n) = \Theta(h(n)), \\
f(n) = O(g(n)) & \text{and } g(n) = O(h(n)) & \text{imply} & f(n) = O(h(n)), \\
f(n) = \Omega(g(n)) & \text{and } g(n) = \Omega(h(n)) & \text{imply} & f(n) = \Omega(h(n)), \\
f(n) = o(g(n)) & \text{and } g(n) = o(h(n)) & \text{imply} & f(n) = o(h(n)), \\
f(n) = \omega(g(n)) & \text{and } g(n) = \omega(h(n)) & \text{imply} & f(n) = \omega(h(n)).
\end{array}
$$

Reflexivity:

$$
\begin{aligned}
f(n) &= \Theta(f(n)), \\
f(n) &= O(f(n)), \\
f(n) &= \Omega(f(n)).
\end{aligned}
$$

Symmetry:

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)).$$

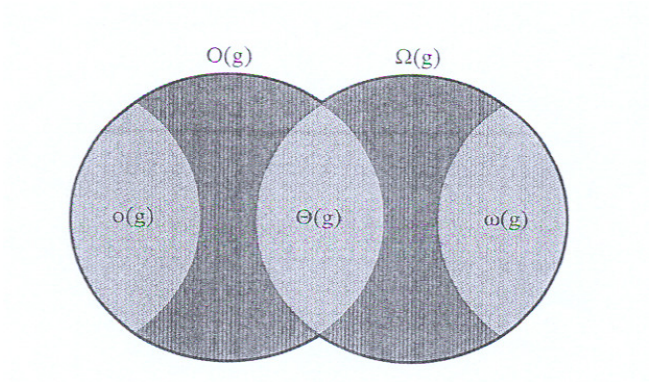
Transpose symmetry:

$$
\begin{aligned}
f(n) &= O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)), \\
f(n) &= o(g(n)) \text{ if and only if } g(n) = \omega(f(n)).
\end{aligned}
$$

Asymptotic Analogy

$$
\begin{aligned}
f(n) = O(g(n)) &\approx a \le b, \\
f(n) = \Omega(g(n)) &\approx a \ge b, \\
f(n) = \Theta(g(n)) &\approx a = b, \\
f(n) = o(g(n)) &\approx a < b, \\
f(n) = \omega(g(n)) &\approx a > b.
\end{aligned}
$$

Complexity Classes



45

The Seven Most Important Functions

1. The Constant Function

   $f(n) = c$,

   for some fixed constant c.

2. The Linear Function

   $f(n) = n$.

3. The Quadratic Function

   $f(n) = n^2$.

4. Cubic function

   $f(n) = n^3$.

5. Polynomials

   $f(n) = a_0 + a_1 n + a_2 n^2 + \ldots\ldots + a_d n^d$

   integer d is called the degree of the polynomial.

46

The Seven Most Important Functions

6. Exponential Function

   $f(n) = b^n$, where b is a positive constant, called the base and the argument n is the exponent.

7. Logarithm Function

   $f(n) = \log_b n$.

8. N-log-N Function

   $f(n) = n \log n$.

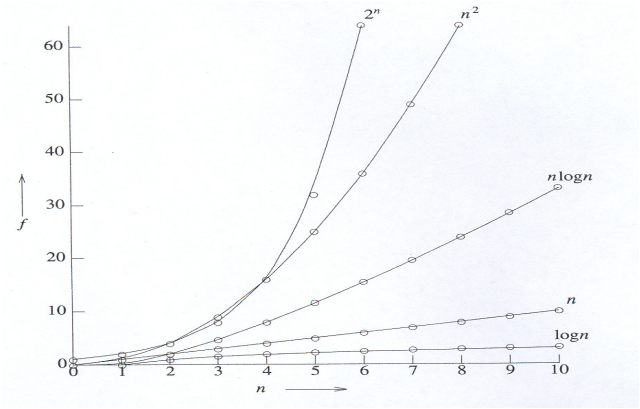
47

Comparing Growth Rates

constant	logarithm	linear	n-log-n	quadratic	cubic	exponent
1	log n	n	n log n	n^2	n^3	a^n (a > 1)

48

Growth of Function Values



49

Basic Efficiency Class

class	name	
1	constant	
log n	logarithmic	polynomial
n	linear	polynomial
$n \log n$	$n \log n$	polynomial
n^2	quadratic	polynomial
n^3	cubic	polynomial
2^n	exponential	exponential
$n!$	factorial	exponential

50

Kompleksitas Masalah vs Algoritma

- Sebuah masalah berorde O(f(n)) berarti terdapat beberapa algoritma O(f(n)) untuk menyelesaikan masalah tersebut.
- Sebuah masalah berorde Ω(g(n)) berarti setiap algoritma yang dapat menyelesaikan masalah tersebut adalah Ω(g(n)) .

51

Latihan

- Tentukan kompleksitas waktu dari algoritma dibawah ini jika melihat banyaknya operasi a←a+1

```
for i ← 1 to n do
  for j ← 1 to i do
     for k ← j to n do
          a ← a + 1
    endfor
  endfor
endfor
```

- Tentukan pula nilai *O*-besar, *Ω*-besar, dan *Θ*-besar dari algoritma diatas (harus penjelasan)

52

Jawaban (1)

Untuk $i = 1$,
  Untuk $j = 1$, jumlah perhitungan = n kali

Untuk $i = 2$,
  Untuk $j = 1$, jumlah perhitungan = n kali
  Untuk $j = 2$, jumlah perhitungan = $n - 1$ kali
…
Untuk $i = n$,
  Untuk $j = 1$, jumlah perhitungan = n kali
  Untuk $j = 2$, jumlah perhitungan = $n - 1$ kali
  …
  Untuk $j = n$, jumlah perhitungan = 1 kali.

Jadi jumlah perhitungan = $T(n) = n^2 + (n-1)^2 + (n-2)^2 + \ldots + 1$

53

Jawaban (2)

- $T(n) = O(n^3) = \Omega(n^3) = \Theta(n^3)$.
- Salah satu cara penjelasan:

  $T(n) = n^2 + (n-1)^2 + (n-2)^2 + \ldots + 1$
  $= n(n+1)(2n+1)/6$
  $= 2n^3 + 3n^2 + 1$.

- Diperoleh $T(n) \leq 3n^3$ untuk $n \geq 4$ dan $T(n) \geq 2n^3$ untuk $n \geq 1$.

54

Referensi

1. Rinaldi Munir. "Materi Kuliah Matematika Diskrit", Informatika-ITB. Bandung. 2003
2. Rinaldi Munir. "Matematika Diskrit". Informatika. Bandung. 2001
3. Levitin, Anany. "*Introduction to the design and analysis of algorithm*". Addison Wesley. 2003
4. Cormen., Leiserson., Rivest. "*Introduction to algorithms*"

55