

PERTEMUAN 7

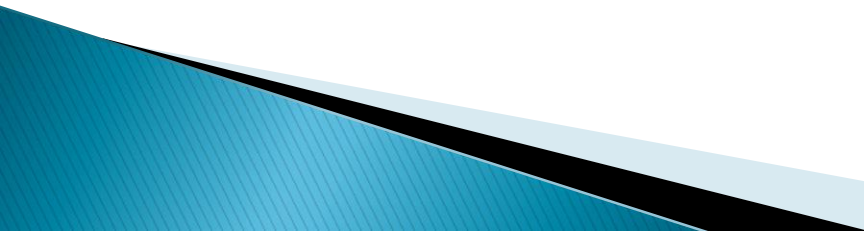
Kompleksitas Algoritma

Hermanto, S.Kom. M.Kom

Apa yang pertama kali Anda menjadi pertimbangan Anda saat membeli komputer, selain PERFORMA?

- ❖ Keandalan dalam menyelesaikan masalah (*robustness*)
- ❖ Fungsionalitas (*functionality*)
- ❖ Tampilan grafis (*user interface*)
- ❖ Daya tahan (*reliability*)
- ❖ Keamanan (*security*)
- ❖ Kesederhanaan (*simplicity*)
- ❖ Kemudahan dalam penggunaan (*user friendly*)
- ❖ Kemudahan dalam pemeliharaan (*maintainability*)

Algoritma dan Performa

- ❖ Hal-hal yang menjadi pertimbangan utama Anda tidak muncul dengan gratis
 - ❖ Performa sistem menjadi alat tukar seperti **uang**.
 - ❖ Algoritma program memegang peran kunci
 - ❖ Algoritma adalah teknologi, engineered, sebagaimana perangkat keras komputer
- 

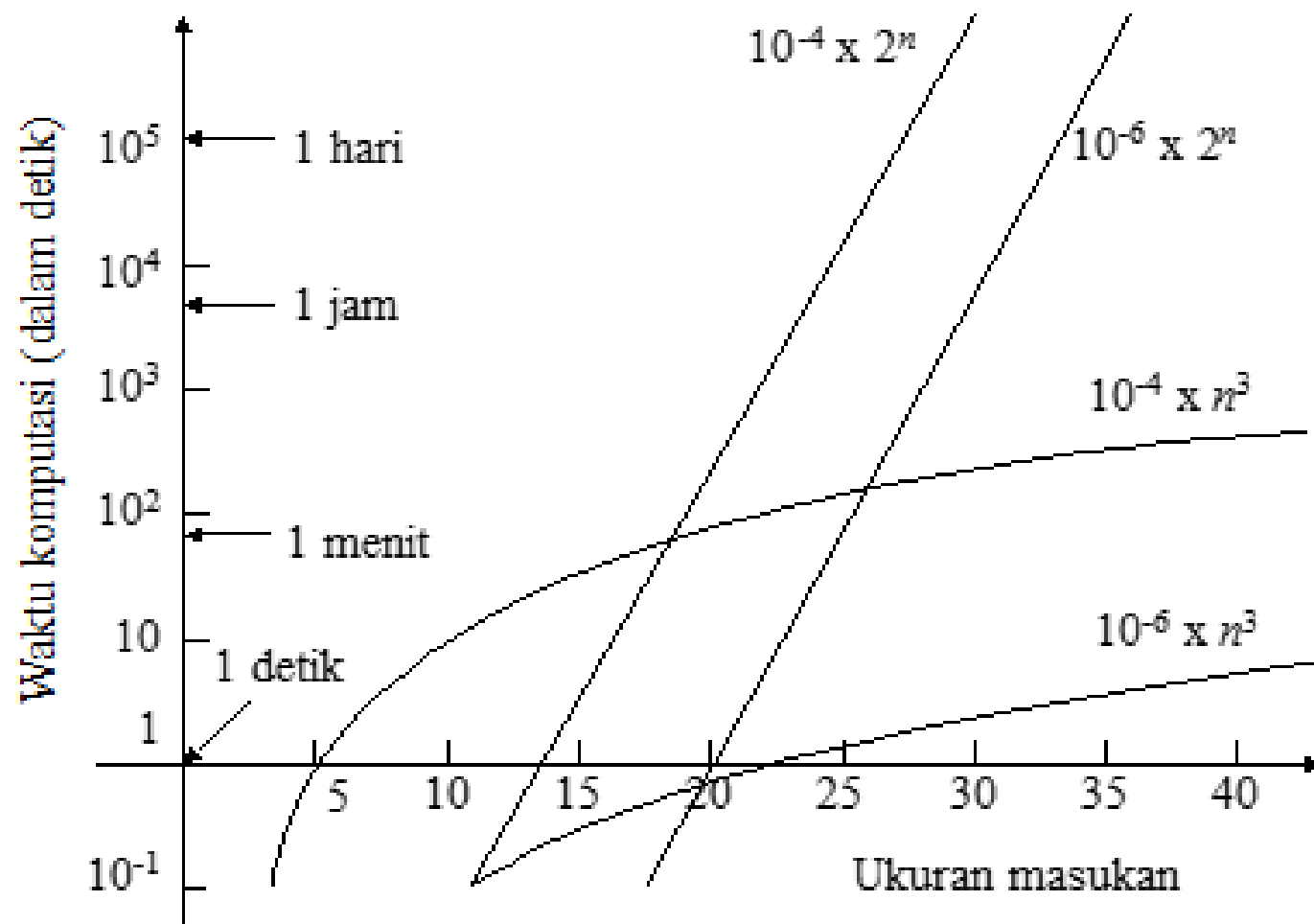
Pentingnya Analisa Algoritma

- ❖ Algoritma membantu kita memahami skalabilitas program kita
- ❖ Performa terkadang menjadi pembeda antara yang mungkin dilakukan dan **yang tidak mungkin dilakukan**
- ❖ Analisa algoritma memberi gambaran informasi tentang 'perilaku program' kita
- ❖ Mempelajari bagaimana menerapkan algoritma yang baik untuk kasus tertentu membedakan profesi system analyst dan programmer

Pentingnya Analisa 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 terbaik.
- Mengapa Kita Memerlukan Algoritma yang Mangkus?

Pentingnya Analisa Algoritma



Hal-hal yang perlu diperhatikan pada analisa algoritma

- ❖ Apa yang membuat sebuah algoritma dikatakan LEBIH BAIK dari algoritma yang lain?
 - **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
- ❖ **Kompleksitas waktu** menjadi variabel yang **sangat penting**

Hal-hal yang perlu diperhatikan pada analisa algoritma

Ukuran masukan (n): jumlah data yang diproses oleh sebuah algoritma.

Contoh:

- Algoritma pengurutan 1000 elemen larik, maka $n=1000$
- Algoritma perkalian 2 buah matriks berukuran 50×50 , maka $n = 50$

Dalam praktek perhitungan kompleksitas, ukuran masukan dinyatakan sebagai variabel n saja

Kompleksitas Waktu

- Jumlah tahapan komputasi dihitung dari berapa kali suatu operasi di dalam sebuah algoritma sebagai fungsi ukuran masukan
- Di dalam sebuah algoritma, terdapat bermacam jenis operasi, misalnya:
 - Operasi baca tulis
 - Operasi aritmatika
 - Operasi pengisian nilai
 - Operasi pengaksesan elemen larik
 - Operasi pemanggilan fungsi/prosedur
- Dalam praktek, kita hanya menghitung jumlah operasi khas (tipikal) yang mendasari suatu algoritma

Contoh :

- Algoritma untuk menghitung perpangkatan dua bilangan.
- $f(x,y) = x^y$

```
int pangkat(x, y):  
    hasil = 1  
    for i in range(0, y):  
        hasil = x * hasil  
    return hasil
```

Contoh :

- Pada dasarnya yang kita lakukan pada kode di atas adalah mengkalikan **x** dengan dirinya sendiri sebanyak **y** kali, dan menyimpan hasil kali tersebut di dalam variabel **hasil**.
- Baris **hasil = x * hasil** melakukan perkalian **x** dengan dirinya sendiri, dan perulangan dilakukan untuk memastikan baris ini dijalankan sebanyak **y** kali

Contoh :

Semakin besar nilai Y, jml eksekusi semakin besar, jadi bayangkan jika jml eksekusi yg diperlukan Y^2

| Y | Jml Langkah (Y) | Jml Langkah (Y^2) |
|-------|-----------------|-----------------------|
| 1 | 1 | 1 |
| 10 | 10 | 100 |
| 100 | 100 | 10000 |
| 1000 | 1000 | 1000000 |
| 10000 | 10000 | 100000000 |



Contoh operasi khas di dalam algoritma

❑ Algoritma pencarian di dalam larik

Operasi khas: perbandingan elemen larik

❑ Algoritma pengurutan

Operasi khas: perbandingan elemen dan pertukaran elemen

❑ Algoritma penjumlahan 2 buah matriks

Operasi khas: penjumlahan

❑ Algoritma perkalian 2 buah matriks

Operasi khas: perkalian dan penjumlahan



Tinjau algoritma menghitung rerata sebuah larik.

```
sum ← 0
for i ← 1 to n do
    sum ← sum + a[i]
endfor
rata_rata ← sum/n
```

- Operasi yang mendasar pada algoritma tersebut adalah operasi penjumlahan elemen-elemen a_i (yaitu $\text{sum} \leftarrow \text{sum} + a[i]$) yang dilakukan sebanyak n kali.
- Kompleksitas waktu: $T(n) = n$.

Contoh 2. Algoritma untuk mencari elemen terbesar di dalam sebuah larik (*array*) yang berukuran n elemen.

```
procedure CariElemenTerbesar(input   $a_1, a_2, \dots, a_n$  : integer, output
maks : integer)
{ Mencari elemen terbesar dari sekumpulan elemen larik integer  $a_1, a_2, \dots, a_n$ .
  Elemen terbesar akan disimpan di dalam maks.
  Masukan:  $a_1, a_2, \dots, a_n$ 
  Keluaran: maks (nilai terbesar)
}
Deklarasi
  k : integer

Algoritma
  maks  $\leftarrow a_1$ 
  k  $\leftarrow 2$ 
  while k  $\leq$  n do
    if  $a_k >$  maks then
      maks  $\leftarrow a_k$ 
    endif
    i  $\leftarrow i + 1$ 
  endwhile
  { k > n }
```

Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi perbandingan elemen larik ($A[i] > \text{maks}$).

Kompleksitas waktu CariElemenTerbesar : $T(n) = n - 1$.

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. $T_{\text{avg}}(n)$: Kompleksitas waktu untuk kasus rata-rata (*average case*)

→ kebutuhan waktu secara rata-rata



Contoh 3. Algoritma *sequential search*.

```
procedure PencarianBeruntun(input  $a_1, a_2, \dots, a_n$  : integer,  $x$  : integer,  
                           output  $idx$  : integer)
```

Deklarasi

```
   $k$  : integer  
   $ketemu$  : boolean    { bernilai true jika  $x$  ditemukan atau false jika  $x$   
                        tidak ditemukan }
```

Algoritma:

```
   $k \leftarrow 1$   
   $ketemu \leftarrow \text{false}$   
  while ( $k \leq n$ ) and (not  $ketemu$ ) do  
    if  $a_k = x$  then  
       $ketemu \leftarrow \text{true}$   
    else  
       $k \leftarrow k + 1$   
    endif  
  endwhile  
  {  $k > n$  or  $ketemu$  }  
  
  if  $ketemu$  then    {  $x$  ditemukan }  
     $idx \leftarrow k$   
  else  
     $idx \leftarrow 0$     {  $x$  tidak ditemukan }  
  endif
```

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- j , maka operasi perbandingan ($a_k = x$) akan dieksekusi sebanyak j kali.

$$T_{AVG}(n) = \frac{(1 + 2 + 3 + \dots + n)}{n} = \frac{\frac{1}{2}n(1 + n)}{n} = \frac{(n + 1)}{2}$$

Kompleksitas Waktu Asimptotik

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

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

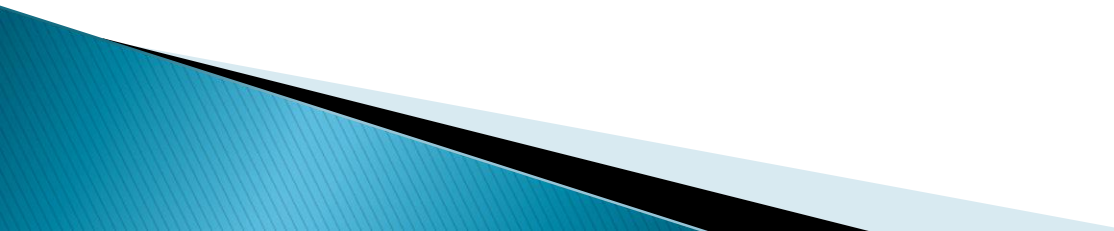
| n | $T(n) = 2n^2 + 6n + 1$ | n^2 |
|--------|------------------------|---------------|
| 10 | 261 | 100 |
| 100 | 2061 | 1000 |
| 1000 | 2.006.001 | 1.000.000 |
| 10.000 | 1.000.060.001 | 1.000.000.000 |

- 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 “ O ” disebut notasi “ O -Besar” (*Big-O*) yang merupakan notasi **kompleksitas waktu asimptotik**.

Kesimpulan

- ❑ Analisa algoritma diperlukan untuk perbandingan algoritma tanpa tergantung spesifikasi mesin
 - ❑ Analisa algoritma membantu kita memecahkan masalah sesuai kondisi dan data yang kita hadapi
 - ❑ Analisa algoritma bukan alat mutlak untuk memilih algoritma terbaik, tapi membantu memahami perilaku algoritma saat diterapkan di dunia nyata
- 



Terima
kasih