

@Design and Analysis Algorithm

Pertemuan 02

Drs. Achmad Ridok M.Kom

Imam Cholissodin, S.Si., M.Kom

M. Ali Fauzi, S.Kom., M. Kom.

Ratih Kartika Dewi, ST, M.Kom



Contents (1 of 2)

1

Important Sum Manipulation



Important Sum Manipulations (1)

$$\sum_{i=1}^u c a_i = c \sum_{i=1}^u a_i$$

$$\sum_{i=1}^u (a_i \pm b_i) = \sum_{i=1}^u a_i \pm \sum_{i=1}^u b_i$$

$$\sum_{i=l}^u 1 = u - l + 1; l \leq u : \text{lower \& upper integer limits}$$

$$\sum_{i=0}^n i = \sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n(n+1)}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2)$$



Important Sum Manipulations (2)

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$$

$$\sum_{i=1}^n i^k = 1^k + 2^k + \dots + n^k \approx \frac{1}{k+1} n^{k+1}$$

$$\sum_{i=0}^n a^i = 1 + a + a^2 + \dots + a^n = \frac{a^{n+1} - 1}{a - 1} (a \neq 1)$$

$$\sum_{i=1}^n \gamma_i = 1 + \gamma_2 + \dots + \gamma_n \approx \ln n + \gamma; \gamma \approx 0.5772\dots$$

$$\sum_{i=1}^n \lg i \approx n \lg n$$



Contents (2 of 2)

1

Analisis Algoritma

2

Analisis Efisiensi Algoritma

3

Analisis Efisiensi Algoritma Non-Rekursif

4

Order Of Growth



Analisis Algoritma

- **Analisis Algoritma** bertujuan memeriksa efisiensi algoritma dari dua segi : waktu eksekusi dan penggunaan memori
- **Efisiensi waktu** seberapa cepat algoritma dieksekusi
- **Efisiensi memori** berapa banyak memori yang dibutuhkan untuk menjalankan algoritma



Analisis Efisiensi Algoritma

- Untuk melakukan analisis efisiensi waktu algoritma harus diestimasi dulu **waktu eksekusi algoritma**
- Bagaimana melakukannya ?



Analisis Efisiensi Algoritma

Algorithm *sequential search* ($A[0..n-1]$, K)

```
// searches for a given value in a given array by sequential search  
// input: an array A[0..n-1] and a search key K  
// output: returns the index of the first element of A that matches  
K or -1 if there are no matching elements
```

i ← 0	1 x
while i < n and A[i] ≠ K do	2 x
i ← i + 1	1 x
if i < n return i	2 x
else return -1	1 x



Analisis Efisiensi Algoritma

Baris kode mana yang sangat berpengaruh pada running time?

Bagian loop (baris 2 dan 3). Mengapa?

Karena dieksekusi berulang – ulang

Makin banyak eksekusinya, makin lama
running time program



Analisis Efisiensi Algoritma

Sequential Search

```
i ← 0          1 x  
while i < n and A[i] ≠ K do 2 x  
    i ← i + 1      1 x  
if i < n return i      2 x  
else return -1      1 x
```

Estimasi waktu eksekusi algoritma sequential search!



Analisis Efisiensi Algoritma

$$\text{time} = n\text{Loop} \times t\text{Loop}$$

- time = estimasi waktu eksekusi algoritma untuk input tertentu
- nLoop = berapa kali loop dieksekusi
- tLoop = waktu yang diperlukan untuk mengeksekusi loop 1 kali. Biasanya ditentukan 1 satuan waktu tanpa dispesifikasikan berapa nilainya



Analisis Efisiensi Algoritma

Asumsikan array A terdiri atas n elemen.

- **Best case** : k ditemukan di elemen pertama array A. time = 1×1 satuan waktu
- **Average case** : k ditemukan di elemen tengah array A. time = $n/2 \times 1$ satuan waktu
- **Worst case** : k ditemukan di elemen paling akhir array A. time = $n \times 1$ satuan waktu



Analisis Efisiensi Algoritma

Langkah-langkah umum untuk menganalisis efisiensi waktu algoritma

1. Tentukan parameter yang mengindikasikan ukuran input
2. Identifikasi basic operation algoritma
3. Tentukan apakah untuk ukuran input yang sama banyaknya eksekusi basic operation bisa berbeda
4. Tentukan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi
5. Selesaikan rumus sigma untuk menghitung banyaknya eksekusi basic operation



Analisis Efisiensi Algoritma

Step 1 = Tentukan parameter yang mengindikasikan ukuran input

- ❖ Sesuatu pada input yang jika nilainya bertambah akan menyebabkan banyaknya eksekusi loop bertambah
- ❖ Contoh, algoritma untuk menghitung X^n menggunakan cara $X^n = X * X * X * \dots * X$ sebanyak n kali. Parameter ukuran inputnya adalah nilai n, karena jika n makin besar, maka banyaknya eksekusi loop bertambah
- ❖ Bagaimana dengan nilai X?



Analisis Efisiensi Algoritma

Sequential Search

```
i ← 0                                1 x  
while i < n and A[i] ≠ K do    2 x  
    i ← i + 1                          1 x  
if i < n return i                      2 x  
else return -1                         1 x
```

Estimasi waktu eksekusi algoritma sequential search!



Analisis Efisiensi Algoritma

Step 1 = Tentukan parameter yang mengindikasikan ukuran input

- ❖ Sesuatu pada input yang jika nilainya bertambah akan menyebabkan banyaknya eksekusi loop bertambah
- ❖ Contoh, algoritma untuk menghitung X^n menggunakan cara $X^n = X * X * X * \dots * X$ sebanyak n kali. Parameter ukuran inputnya adalah nilai n, karena jika n makin besar, maka banyaknya eksekusi loop bertambah
- ❖ Bagaimana dengan nilai X?
 - ❖ Untuk algoritma sequential search, parameter ukuran inputnya adalah banyaknya elemen array (n)
 - ❖ Mengapa nilai elemen array tidak?



Analisis Efisiensi Algoritma

Step 2 = Identifikasi basic operation algoritma

- ❖ Operasi paling penting dalam algoritma tersebut
- ❖ Dapat diwakili oleh sebuah operasi pada loop paling dalam.
- ❖ Operasi yang dipilih adalah operasi yang selalu dilakukan ketika loop dieksekusi



Analisis Efisiensi Algoritma

Sequential Search

```
i ← 0                                1 x  
while i < n and A[i] ≠ K do    2 x  
    i ← i + 1                          1 x  
if i < n return i                      2 x  
else return -1                         1 x
```

Estimasi waktu eksekusi algoritma sequential search!



Analisis Efisiensi Algoritma

Step 2 = Identifikasi basic operation algoritma

- ❖ Waktu yang diperlukan untuk mengeksekusi loop 1 kali
- ❖ Dapat diwakili oleh sebuah **operasi pada loop paling dalam.**
- ❖ Operasi yang dipilih adalah operasi yang selalu dilakukan ketika loop dieksekusi
- ❖ Untuk algoritma sequential search, basic operationnya dapat digunakan **$A[i] \neq K$**
- ❖ **$A[i] \neq K$** dieksekusi 1 kali setiap loop dieksekusi



Analisis Efisiensi Algoritma

Step 2 = Identifikasi basic operation algoritma

ALGORITHM *MaxElement(A[0..n – 1])*

```
//Determines the value of the largest element in a given array  
//Input: An array A[0..n – 1] of real numbers  
//Output: The value of the largest element in A  
maxval  $\leftarrow$  A[0]  
for i  $\leftarrow$  1 to n – 1 do  
    if A[i] > maxval  
        maxval  $\leftarrow$  A[i]  
return maxval
```



Analisis Efisiensi Algoritma

Step 2 = Identifikasi basic operation algoritma

ALGORITHM *MaxElement(A[0..n – 1])*

```
//Determines the value of the largest element in a given array  
//Input: An array A[0..n – 1] of real numbers  
//Output: The value of the largest element in A  
maxval  $\leftarrow$  A[0]  
for i  $\leftarrow$  1 to n – 1 do  
    if A[i] > maxval  
        maxval  $\leftarrow$  A[i]  
return maxval
```

- ❖ Untuk algoritma di atas, basic operationnya dapat digunakan **maxval<-A[i]**



Analisis Efisiensi Algoritma

Step 2 = Identifikasi basic operation algoritma

```
func bubblesort( var a as array )
    for i from 1 to N
        for j from 0 to N - 1
            if a[j] > a[j + 1]
                swap( a[j], a[j + 1] )
end func
```



Analisis Efisiensi Algoritma

Step 2 = Identifikasi basic operation algoritma

```
func bubblesort( var a as array )
    for i from 1 to N
        for j from 0 to N - 1
            if a[j] > a[j + 1]
                swap( a[j], a[j + 1] )
end func
```

- ❖ Untuk algoritma di atas, basic operationnya dapat digunakan **a[j] > a[j + 1]** dan/atau **swap()**



Analisis Efisiensi Algoritma

Just another Algorithm

```
N ← 0  
  
for i ← 1 to n do  
  
    S ← S + i * i * 2  
  
return S
```



Analisis Efisiensi Algoritma

Just another Algorithm

```
N ← 0  
  
for i ← 1 to n do  
  
    S ← S + i * i * 2  
  
return S
```

- ❖ Untuk algoritma di atas, basic operationnya dapat digunakan **perkalian** dan/atau **penambahan**



Analisis Efisiensi Algoritma

Step 3 = Tentukan apakah untuk ukuran input yang sama banyaknya eksekusi basic operation bisa berbeda

- ❖ Pada sequential search, parameter untuk ukuran input adalah n atau banyaknya elemen array
- ❖ Untuk n tertentu, apakah banyaknya eksekusi basic operation bisa berbeda?
- ❖ Jika elemen pertama array input A bernilai K, maka banyaknya eksekusi basic operation untuk n tertentu $C(n) = 1$
- ❖ Jika K ditemukan di elemen terakhir, maka $C(n) = n$
- ❖ Perlu diadakan analisa **best case, worst case dan average case**



Analisis Efisiensi Algoritma

Step 3 = Tentukan apakah untuk ukuran input yang sama banyaknya eksekusi basic operation bisa berbeda

```
i ← 0                                1 x
while i < n and A[i] ≠ K do    2 x
    i ← i + 1                          1 x
    if i < n return i                  2 x
else return -1                         1 x
```



Analisis Efisiensi Algoritma

Step 3 = Tentukan apakah untuk ukuran input yang sama banyaknya eksekusi basic operation bisa berbeda

ALGORITHM *MaxElement(A[0..n – 1])*

```
//Determines the value of the largest element in a given array  
//Input: An array A[0..n – 1] of real numbers  
//Output: The value of the largest element in A  
maxval  $\leftarrow A[0]$   
for i  $\leftarrow 1$  to n – 1 do  
    if A[i] > maxval  
        maxval  $\leftarrow A[i]$   
return maxval
```



Analisis Efisiensi Algoritma

Step 3 = Tentukan apakah untuk ukuran input yang sama banyaknya eksekusi basic operation bisa berbeda

```
func bubblesort( var a as array )
    for i from 1 to N
        for j from 0 to N - 1
            if a[j] > a[j + 1]
                swap( a[j], a[j + 1] )
end func
```



Analisis Efisiensi Algoritma

Just another Algorithm

```
N ← 0  
  
for i ← 1 to n do  
  
    S ← S + i * i * 2  
  
return S
```



Analisis Efisiensi Algoritma

Step 4 = Tentukan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

- ❖ $C(n)$ = banyaknya eksekusi basic operation untuk input ukuran n
- ❖ Untuk Best case :

$$C(n) = \sum_{i=1}^1 1$$

- ❖ Best case terjadi jika elemen pertama A bernilai K



Analisis Efisiensi Algoritma

Step 4 = Tentukan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

❖ Untuk Worst case :

$$C(n) = \sum_{i=1}^n 1$$

❖ Worst case terjadi jika elemen A yang bernilai K merupakan elemen terakhir atau tidak ada elemen A yang bernilai K



Analisis Efisiensi Algoritma

Step 4 = Tentukan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

❖ Untuk Average case :

Asumsikan

- ❖ Data K memang ada di A
- ❖ Probabilitas K terletak di elemen tertentu A terdistribusi merata.
- ❖ Probabilitas K terletak di elemen ke $i = 1/n$



Analisis Efisiensi Algoritma

Step 4 = Tentukan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

Posisi K ditemukan	Banyaknya eksekusi basic operation	Probabilitas terjadi	Kontribusi pada C(n)
1	1	$1/n$	$1 * 1/n$
2	2	$1/n$	$2 * 1/n$
...
...
n	n	n	$N * 1/n$

Bentuk umum : $i * 1 / n$



Analisis Efisiensi Algoritma

Step 4 = Tentukan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

Sehingga untuk Average case :

$$C(n) = \sum_{i=1}^n i * \frac{1}{n}$$



Analisis Efisiensi Algoritma

Step 5 = Selesaikan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

Best case untuk sequential search

$$C(n) = \sum_{i=1}^1 1 \rightarrow C(n) = 1$$

- ❖ Best case pada sequential search $C(n) = 1$
- ❖ Untuk input berukuran n , basic operation dilakukan 1 kali



Analisis Efisiensi Algoritma

Step 5 = Selesaikan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

Worst case untuk sequential search

$$C(n) = \sum_{i=1}^n 1 \quad \rightarrow \quad C(n) = n$$

- ❖ Worst case pada sequential search $C(n) = n$
- ❖ Untuk input berukuran n , basic operation dilakukan n kali



Analisis Efisiensi Algoritma

Step 5 = Selesaikan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

Average case pada sequential search

$$C(n) = \sum_{i=1}^n i * \frac{1}{n}$$

$$C(n) = \frac{1}{n} \sum_{i=1}^n i$$

$$C(n) = \frac{1}{n} * \frac{1}{2} n(1+n) = \frac{1}{2} (1+n)$$



Analisis Efisiensi Algoritma

Step 5 = Selesaikan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi

Average case pada sequential search

$$C(n) = \frac{(n+1)}{2}$$

- ❖ Untuk $n = 10$, $C(n) = 5,5$
- ❖ Apakah itu berarti K berada pada elemen 5 atau 6
- ❖ Apa artinya?



Analisis Efisiensi Algoritma

Estimasi waktu running
algoritma sequential search

$$T(n) = C_{op} * C(n)$$

$T(n)$ = Waktu yang diperlukan untuk mengeksekusi algoritma dengan input berukuran n

C_{op} = Waktu untuk mengeksekusi basic operation 1 kali.
Biasanya ditentukan 1 satuan waktu

Hitung $T(n)$ untuk sequential search pada best case, worst case dan average case!



Latihan 1

- ❖ Buat algoritma untuk menghitung X^n secara iteratif menggunakan cara $X^n = X * X * X * \dots * X$ sebanyak n kali.

- ❖ Estimasi running time algoritma yang anda buat!



Latihan 2

Algorithm mystery(A[0..n-1])

```
X ← A[0]  
for i ← 1 to n – 1 do  
    if A[i] > X  
        X ← A[i]  
return X
```

1. Apa yang dilakukan algoritma mystery?
2. Estimasikan waktu eksekusi algoritma mystery
3. Estimasi waktu eksekusi algoritma mystery untuk input
 $A = [1, 2, 5, 9, 4, 4, 7, 10, 1, 6]$



Latihan 3

Algorithm mystery2(A[0..n-1])

```
X ← A[0]  
for i ← 0 to n-1 do  
    for j ← 0 to n-1 do  
        X ← A[i]+A[j]  
return X
```

1. Apa Basic Operationnya?
2. Estimasikan waktu eksekusi algoritma mystery2



Latihan 3

Algorithm mystery2(A[0..n-1])

```
X ← A[0]  
for i ← 0 to n-1 do  
    for j ← 0 to n-1-i do  
        X ← A[i]+A[j]  
return X
```

1. Apa Basic Operationnya?
2. Estimasikan waktu eksekusi algoritma mystery2



Latihan 4

- **Algorithm** *MatrixMultiplication(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])*
//multiplies two square matrices of order n by the
// definition-based algorithm
//input: two n-by-n matrices A and B
//output: matrix C = AB
for $i \leftarrow 0$ **to** $n-1$ **do**
 for $j \leftarrow 0$ **to** $n-1$ **do**
 $C[i,j] \leftarrow 0,0$
 for $k \leftarrow 0$ **to** $n-1$ **do**
 $C[i,j] \leftarrow C[i,j] + A[i,k]^* B[k,j]$
return C



Latihan 4: Analysis (1)

- Input's size = matrix order n
- In the innermost loop: multiplication & addition → basic operation candidates
 - MUL & ADD executed exactly once on each repetition on innermost loop → we don't have to choose between these two operations
- Sum of total number of multiplication

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3$$



Latihan 4: Analysis (2)

- Estimate the running time of the algorithm on a particular machine

$$T(n) \approx c_m M(n) = c_m n^3$$

- More accurate estimation (include addition)

$$T(n) \approx c_m M(n) + c_a A(n) = c_m n^3 + c_a n^3 = (c_m + c_a) n^3$$

- c_m : the time of one multiplication
- c_a : the time of one addition



Example 4

- **Algorithm** *Binary(n)*

//input: a positive decimal integer n

//output: the number of binary digits in n 's binary representation

count $\leftarrow 1$

while $n > 1$ **do**

count \leftarrow *count* + 1

$n \leftarrow \lfloor n/2 \rfloor$

return *count*



Exp4: Analysis (1)

- The most frequent executed operation is the comparison $n > 1$
- The number of times the comparison will be executed is larger than the number of repetition of the loop's body by exactly 1



Exp4: Analysis (2)

- The value of n is about halved on each repetition of the loop \rightarrow about $\log_2 n$
- The exact formula: $\lfloor \log_2 n \rfloor + 1$
- Another approach \rightarrow analysis techniques based on recurrence relation



Soal 1

Algorithm uniqueElement(A[0..n-1])

//memeriksa apakah setiap elemen A unik

//input : array A[0..n-1]

//output : mengembalikan true jika setiap elemen A unik dan false jika terdapat beberapa elemen yang nilainya sama

```
for i ← 0 to n - 2 do
    for j ← i + 1 to n - 1 do
        If A[i] = A[j] return false
Return true
```

Estimasi running time algoritma uniqueElement !

(Anany levitin halaman 63)



Exercises (1)

1. Compute the following sums:

a. $1 + 3 + 5 + 7 + \dots + 999$

b. $2 + 4 + 6 + 8 + \dots + 1024$

c.

$$\sum_{i=3}^{n+1} 1; \quad \sum_{i=3}^{n+1} i; \quad \sum_{i=0}^{n-1} i(i+1); \quad \sum_{j=1}^n 3^{j+1}; \quad \sum_{i=1}^n \sum_{j=1}^n ij$$

2. Compute order of growth of the following sums

$$\sum_{i=0}^{n-1} (i^2 + 1)^2; \quad \sum_{i=2}^{n-1} \lg i^2; \quad \sum_{i=1}^n (i+1)2^{i-1};$$



Exercises (2)

3. Algorithm *Mystery*(n)

//input: a nonnegative integer n

S \leftarrow 0

for $i \leftarrow 1$ **to** n **do**

S \leftarrow **S** + $i * i$

return S

- a. What does this algorithm compute?
- b. What is its basic operation?
- c. How many times is the basic op executed?
- d. What is the efficiency class of this algorithm?
- e. Can you make any improvement?



Exercises (3)

4. Algorithm *Secret*(A[0..n-1])

//input: an array A[0..n-1] of n real number

mi \leftarrow A[0]; *ma* \leftarrow A[0]

for *i* \leftarrow 1 **to** n-1 **do**

if A[i] < *mi* **then** *mi* \leftarrow A[i]

if A[i] > *ma* **then** *ma* \leftarrow A[i]

return *ma* - *mi*

- a. What does this algorithm compute?
- b. What is its basic operation?
- c. How many times is the basic op executed?
- d. What is the efficiency class of this algorithm?
- e. Can you make any improvement?



Analisis Algoritma Non-Rekursif

Langkah-langkah umum untuk menganalisis efisiensi waktu algoritma

1. Tentukan parameter yang mengindikasikan ukuran input
2. Identifikasi basic operation algoritma
3. Tentukan apakah untuk ukuran input yang sama banyaknya eksekusi basic operation bisa berbeda
4. Tentukan rumus sigma yang menunjukkan berapa kali basic operation dieksekusi
5. Selesaikan rumus sigma untuk menghitung banyaknya eksekusi basic operation



Latihan

Algorithm mystery(A[0..n-1])

```
X ← A[0]  
for i ← 1 to n – 1 do  
    if A[i] > X  
        X ← A[i]  
return X
```

1. Apa yang dilakukan algoritma mystery?
2. Estimasikan waktu eksekusi algoritma mystery
3. Estimasi waktu eksekusi algoritma mystery untuk input
 $A = [1, 2, 5, 9, 4, 4, 7, 10, 1, 6]$



Untuk apa kita mencari $T(n)$?

Apakah untuk mengestimasi running time algoritma?

- Tujuan utama mencari $T(n)$ bukan mencari waktu eksak yang dibutuhkan untuk mengeksekusi sebuah algoritma
- Tetapi untuk mengetahui tingkat pertumbuhan waktu eksekusi algoritma jika ukuran input bertambah (*order of growth*)



Latihan

- Algoritma mystery $T(n) = n - 1$. Estimasi waktu eksekusi algoritma jika array inputnya memiliki anggota
 - 10 elemen
 - 20 elemen
 - 30 elemen
- Buat grafik yang menunjukkan hubungan antara banyaknya elemen array yang dieksekusi dengan waktu eksekusi



Orders of Growth

- Order of Growth adalah Tingkat pertumbuhan waktu eksekusi algoritma jika ukuran input bertambah



Latihan

- Urutkan waktu eksekusi algoritma 1 – 4 berdasar order of growthnya dari kecil ke besar

$$T_1(n) = n^2$$

$$T_2(n) = n^3$$

$$T_3(n) = n$$

$$T_4(n) = \log_2 n$$

$$T_1(10) = 100$$

$$T_2(10) = 1,000$$

$$T_3(10) = 10$$

$$T_4(10) = 3.3$$

$$T_1(100) = 10,000$$

$$T_2(100) = 1,000,000$$

$$T_3(100) = 100$$

$$T_4(100) = 6.6$$



Membandingkan Orders of Growth

Algoritma A dan B merupakan algoritma untuk menyelesaikan permasalahan yang sama.

Untuk input berukuran n , waktu eksekusi algoritma A adalah $T_A(n)$ sedangkan waktu eksekusi algoritma B adalah $T_B(n)$.

Orders of growth mana yang paling besar?

$$\lim_{n \rightarrow \infty} \frac{T_A(n)}{T_B(n)}$$



Membandingkan Orders of Growth

$$\lim_{n \rightarrow \infty} \frac{T_A(n)}{T_B(n)}$$

- 0 maka OoG $T_A(n) < OoG T_B(n)$
- C maka OoG $T_A(n) = OoG T_B(n)$
- ∞ maka OoG $T_A(n) > OoG T_B(n)$



Example (1)

- Compare OoG of $\frac{1}{2}n(n-1)$ and n^2 .

$$\lim_{n \rightarrow \infty} \frac{\frac{1}{2}n(n-1)}{n^2} = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{n^2 - n}{n^2} = \frac{1}{2} \lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right) = \frac{1}{2}$$

- The limit = c $\rightarrow \frac{1}{2}n(n-1) \in \Theta(n^2)$
- Compare OoG of $\log_2 n$ and \sqrt{n}

$$\lim_{n \rightarrow \infty} \frac{\log_2 n}{\sqrt{n}} = \lim_{n \rightarrow \infty} \frac{(\log_2 n)'}{(\sqrt{n})'} = \lim_{n \rightarrow \infty} \frac{(\log_2 e)^{\frac{1}{n}}}{\frac{1}{2\sqrt{n}}} = 2\log_2 e \lim_{n \rightarrow \infty} \frac{\sqrt{n}}{n} = 0$$

- The limit = 0 $\rightarrow \log_2 n$ has smaller order of \sqrt{n}



Example (2)

- Compare OoG of $n!$ and 2^n .

$$\lim_{n \rightarrow \infty} \frac{n!}{2^n} = \lim_{n \rightarrow \infty} \frac{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n}{2^n} = \lim_{n \rightarrow \infty} \sqrt{2\pi n} \frac{n^n}{2^n e^n} = \lim_{n \rightarrow \infty} \sqrt{2\pi n} \left(\frac{n}{2e}\right)^n = \infty$$

- The limit $= \infty \rightarrow n! \in \Omega(2^n)$



Tugas 1

Terdapat dua algoritma yang menyelesaikan permasalahan yang sama. Untuk input berukuran n , Algoritma 1 menyelesaikan dalam

$$T_1(n) = 30n^2 + 2n + 5.$$

$$T_2(n) = n^3 + n$$

- Mana yang lebih besar, OoG T_1 atau T_2 ? Mengapa?
- Untuk n kecil, mana yang anda pilih? Mengapa?
- Untuk n besar, mana yang anda pilih? Mengapa?



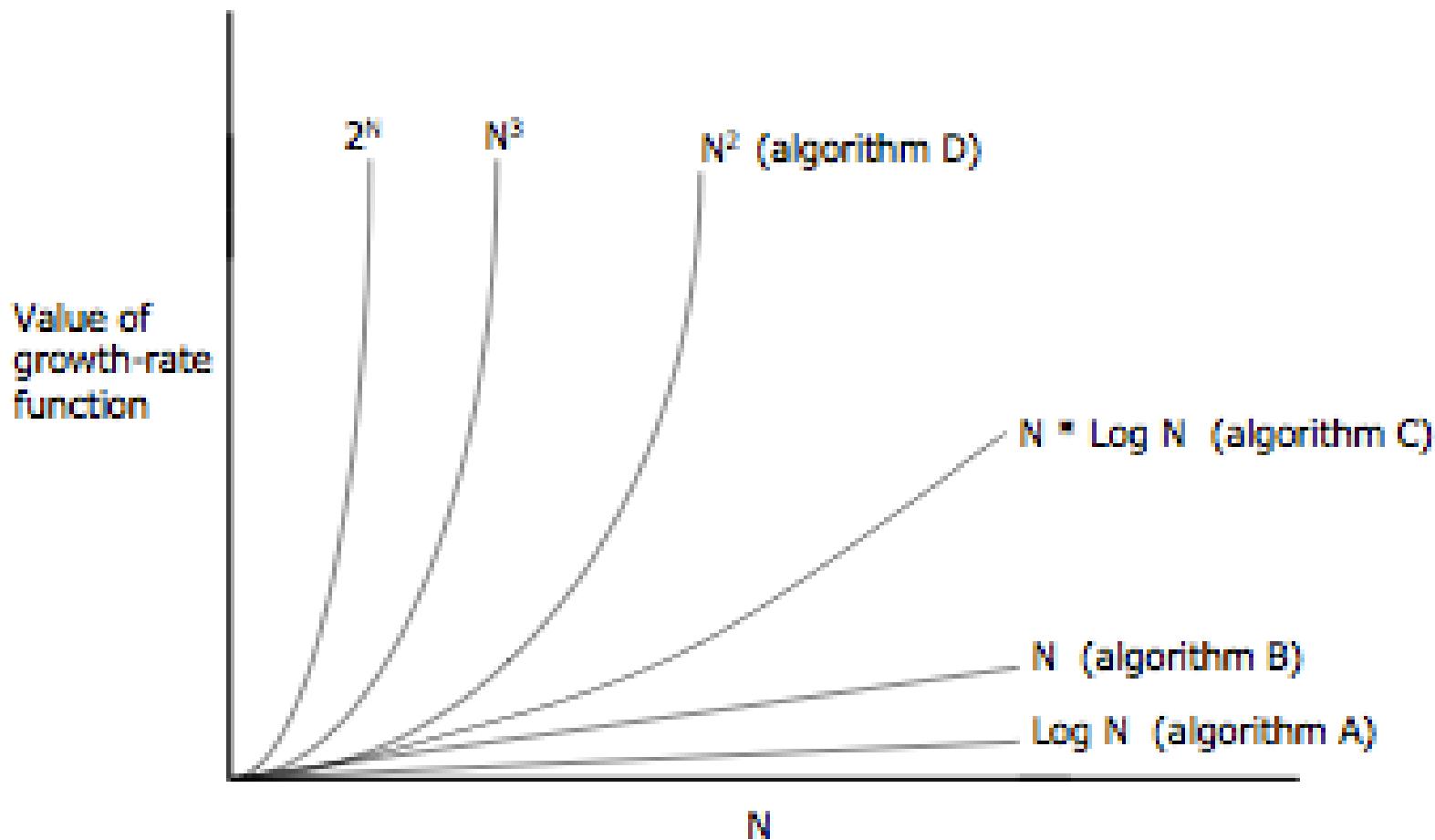
Kelas-Kelas Order of Growth

Makin ke bawah, OoGnya makin besar

C	constant
logN	logarithmic
N	linear
NlogN	
N^2	quadratic
N^3	cubic
2^N	exponential
$N!$	factorial



Grafik Kelas-Kelas Order of Growth





Sifat Order of Growth

- Misal $T(n) = T_1(n) + T_2(n) + \dots + T_i(n)$
Maka OoG $T(n) = \max \text{OoG}(T_1(n), T_2(n), \dots, T_i(n))$
- Misal $T(n) = cf(n)$
Maka OoG $T(n) = f(n)$



Example

- Alg to check whether an array has identical elements:
 1. Sort the array
 2. Scan the sorted array to check its consecutive elements for equality
- (1) = $\leq \frac{1}{2}n(n-1)$ comparison $\rightarrow O(n^2)$
- (2) = $\leq n-1$ comparison $\rightarrow O(n)$
- The efficiency of (1)+(2) = $O(\max\{n^2, n\}) = O(n^2)$



Tugas 2

Tentukan kelas orders of growth dari

- $T_1(n) = 2n^3 + 4n + 1$
- $T_2(n) = 0,5 n! + n^{10}$
- $T_3(n) = n^3 + n \log n$
- $T_4(n) = 2^n + 4n^3 + \log n + 10$



Kelas-Kelas OoG

- (1) Waktu pelaksanaan algoritma adalah tetap, tidak bergantung pada ukuran input.
- $(\log n)$ Kompleksitas waktu logaritmik berarti laju pertumbuhan waktunya berjalan lebih lambat daripada pertumbuhan n .
- (n) Bila n dijadikan dua kali semula, maka waktu pelaksanaan algoritma juga dua kali semula.



Kelas-Kelas OoG

- $(n \log n)$ Bila n dijadikan dua kali semula, maka $n \log n$ menjadi lebih dari dua kali semula (tetapi tidak terlalu banyak)
- (n^2) Bila n dinaikkan menjadi dua kali semula, maka waktu pelaksanaan algoritma meningkat menjadi empat kali semula.
- (n^3) Bila n dinaikkan menjadi dua kali semula, waktu pelaksanaan algoritma meningkat menjadi delapan kali semula.



Kelas-Kelas OoG

- (2^n) Bila n dijadikan dua kali semula, waktu pelaksanaan menjadi kuadrat kali semula!
- $(n!)$ Bila n dijadikan dua kali semula, maka waktu pelaksanaan algoritma menjadi faktorial dari $2n$.

Click to edit subtitle style

Thank You !