

Praktikum Day #2

Kompleksitas Algoritma

Lab Assistant :
Fikri & Jo

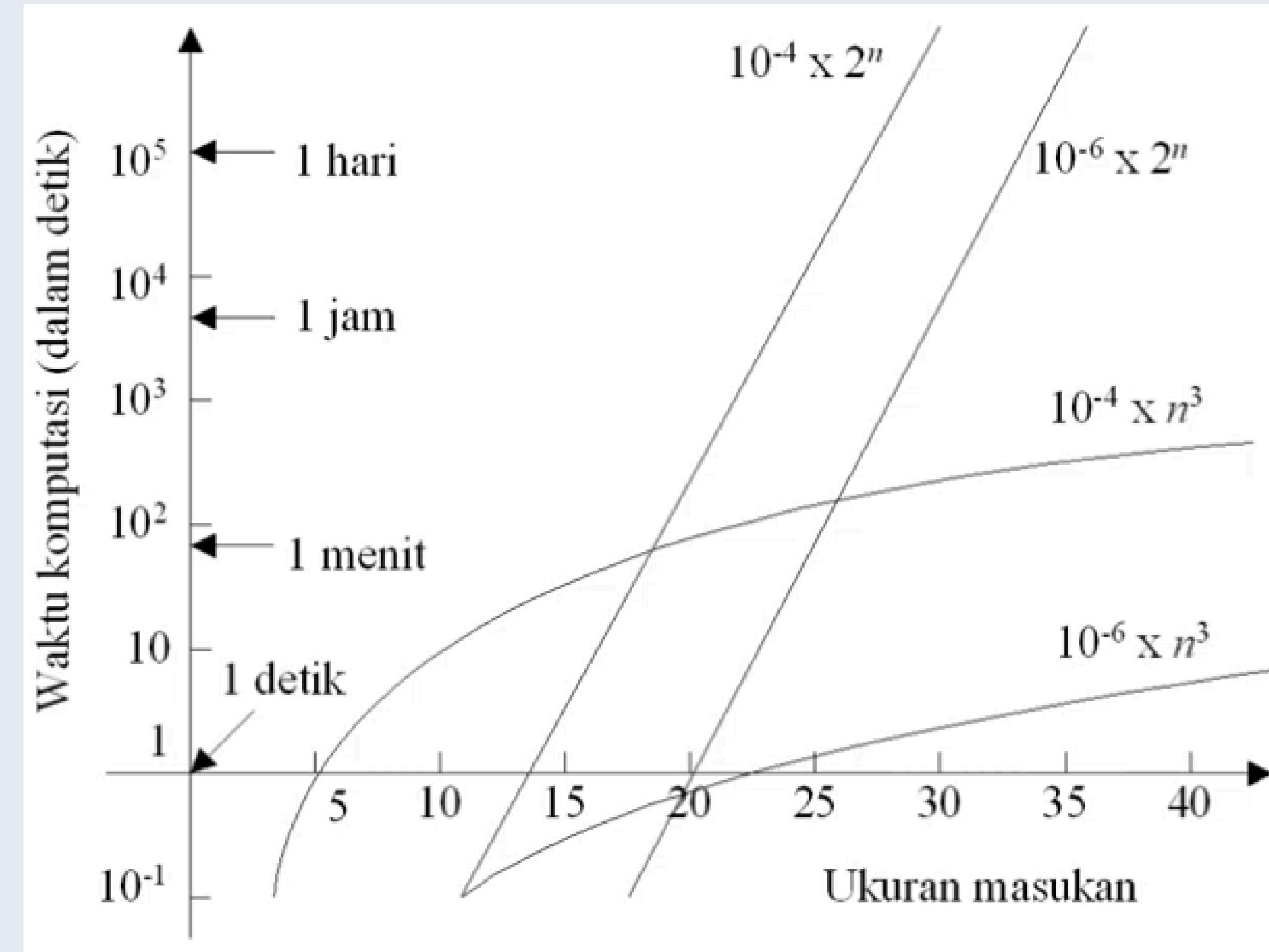


Overview

- **Apa itu kompleksitas algoritma?**
- **Jenis-jenis kompleksitas**
- **Notasi kompleksitas waktu
asimtotik (Big-O)**



KENAPA SIH ? Memahami **Kompleksitas Algoritma ?**



Apakah algoritma kita sudah **efisien**?



2 Macam kompleksitas algoritma

Kompleksitas Ruang S(n)

diukur dari **memori yang digunakan** oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran input n

Kompleksitas Waktu T(n)

diukur dari **jumlah tahapan** komputasi yang dilakukan di dalam algoritma sebagai fungsi dari ukuran input n .

CONTOH 1

Tinjau algoritma **menghitung rata-rata** elemen di dalam sebuah array

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

11	9	17	89	1	90	19	5	3	23	43	99
0	1	2	3	4	5	6	7	8	9	10	11

Index

Operasi yang mendasar pada algoritma tersebut adalah **operasi penjumlahan** elemen-elemen array, yaitu ($\text{sum} \leftarrow \text{sum} + a[i]$) yang dilakukan sebanyak n kali.

Jadi, kompleksitas waktu nya adalah **$T(n) = n$**



3 MACAM Kompleksitas Waktu

Worst Case

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

Average Case

$$T_{avg} = (n)$$

Best Case

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

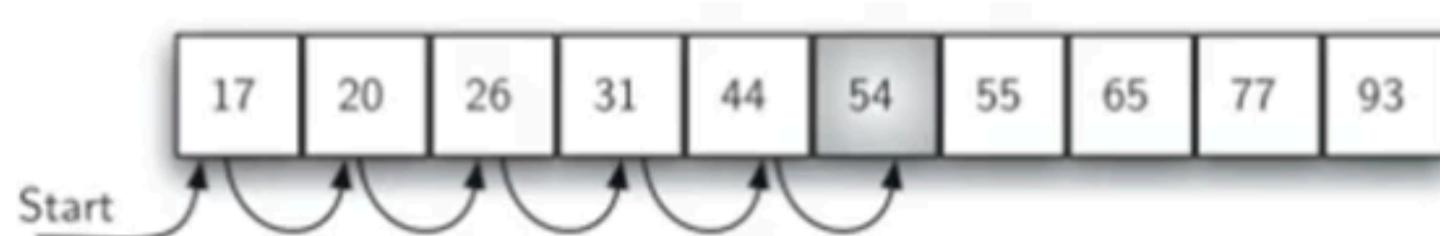
CONTOH 2

Tinjau algoritma **linear search** berikut

```
procedure PencarianBeruntun(input  $a_1, a_2, \dots, a_n : \text{integer}$ ,  $x : \text{integer}$ , output  $\text{idx} : \text{integer}$ )
{ Mencari elemen  $x$  di dalam larik  $A$  yang berisi  $n$  elemen. Jika  $x$  ditemukan, maka indeks elemen larik disimpan
di dalam  $\text{idx}$ ,  $\text{idx}$  bernilai  $-1$  jika  $x$  tidak ditemukan }

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

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



CONTOH 2 - JWB N

Best Case

terjadi bila $a_1 = x$

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

Worst Case

terjadi bila $a_n = x$

atau x tidak ditemukan

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

Average Case

jika x ditemukan pada posisi ke $-j$, maka operasi perbandingan $a_k = x$ akan dieksekusi sebanyak j kali

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

CONTOH 3

Tinjau algoritma **selection sort** berikut

```
procedure SelectionSort(input/output  $a_1, a_2, \dots, a_n : \text{integer}$ )
{ Mengurutkan elemen-elemen larik A yang berisi n elemen integer sehingga terurut menaik }
```

Deklarasi

```
 $i, j, imin, temp : \text{integer}$ 
```

Algoritma

```
for  $i \leftarrow 1$  to  $n - 1$  do { pass sebanyak  $n - 1$  kali }
```

```
 $imin \leftarrow i$ 
```

```
for  $j \leftarrow i + 1$  to  $n$  do
```

```
    if  $a_j < a_{imin}$  then
```

```
         $imin \leftarrow j$ 
```

```
    endif
```

```
endfor
```

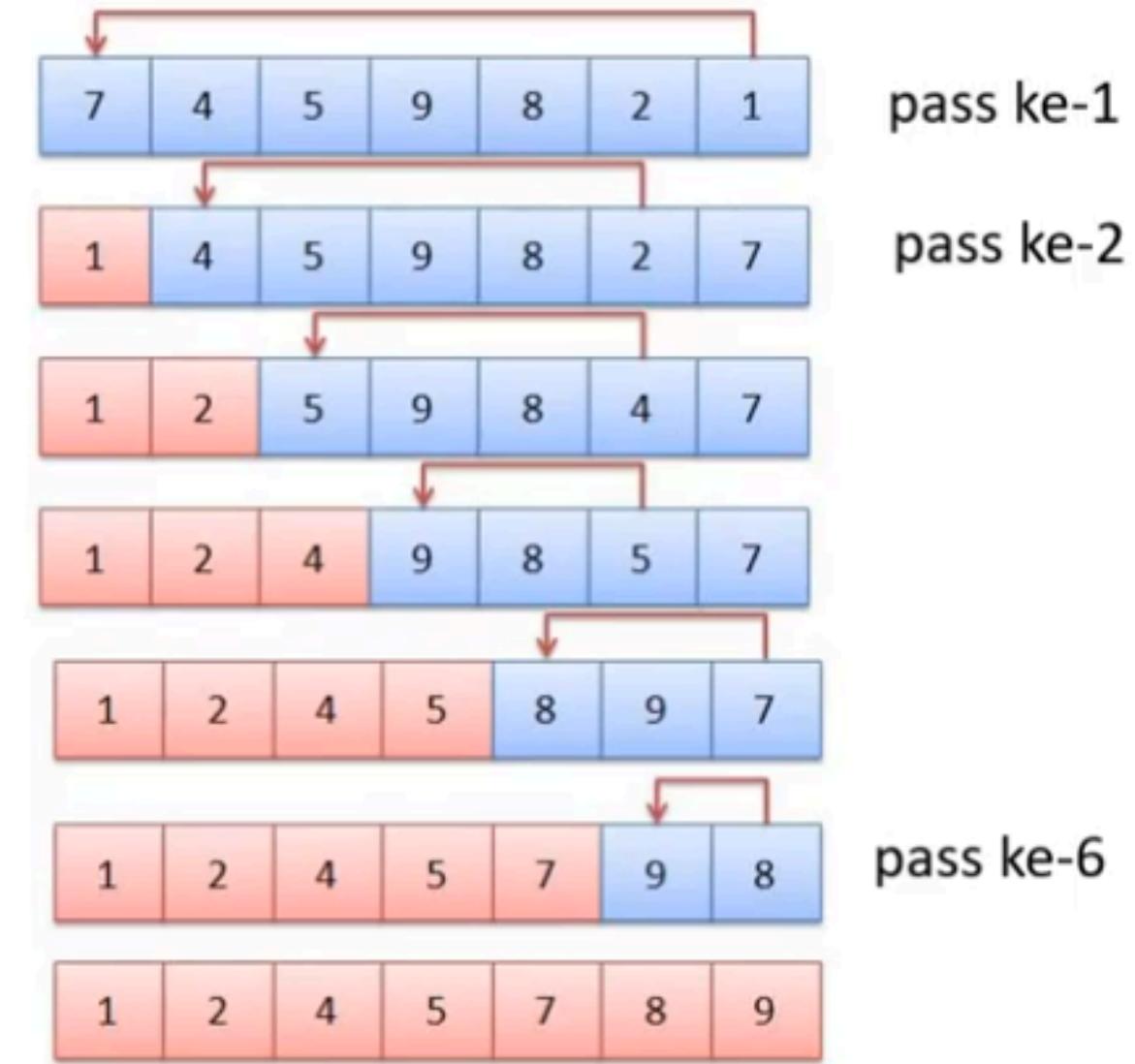
```
{ pertukarkan  $a_{imin}$  dengan  $a_i$ }
```

```
 $temp \leftarrow a_i$ 
```

```
 $a_i \leftarrow a_{imin}$ 
```

```
 $a_{imin} \leftarrow temp$ 
```

```
endfor
```



CONTOH 3 - JWB N

(i). Jumlah operasi **perbandingan** elemen array ($a_j < a_{imin}$)

Untuk setiap pass ke - i :

$i = 1 \rightarrow$ jumlah perbandingan = $n - 1$

$i = 2 \rightarrow$ jumlah perbandingan = $n - 2$

$i = 3 \rightarrow$ jumlah perbandingan = $n - 3$

:

$i = n-1 \rightarrow$ jumlah perbandingan = 1

```
for i←1 to n-1 do {pass sebanyak n-1 kali}
    imin←i
    for j←i+1 to n do
        if aj < aimin then
            imin←j
        endif
    endfor
    {pertukarkan aimin dengan ai}
    temp←ai
    ai←aimin
    aimin←temp
endfor
```

Jumlah seluruh operasi **perbandingan** elemen-elemen array adalah :

$$T(n) = (n - 1) + (n - 2) + \dots + 2 + 1 = \frac{n(n - 1)}{2}$$

ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma SelectionSort tidak bergantung pada apakah data masukannya sudah terurut atau acak

CONTOH 3 -JWBN

(ii). Jumlah operasi **pertukaran**

untuk setiap i dari 1 sampai $n-1$, terjadi satu kali pertukaran elemen, sehingga jumlah operasi pertukaran seluruhnya adalah :

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

Ini adalah jumlah pertukaran untuk semua kasus.

Jadi, algoritma SelectionSort membutuhkan $n(n-1)/2$ buah operasi **perbandingan** elemen dan $n-1$ buah operasi **pertukaran**

```
for i←1 to n – 1 do { pass sebanyak n – 1 kali }
    imin←i
    for j←i + 1 to n do
        if  $a_j < a_{imin}$  then
            imin←j
        endif
    endfor
    { pertukarkan  $a_{imin}$  dengan  $a_i$  }
    temp← $a_i$ 
     $a_i \leftarrow a_{imin}$ 
     $a_{imin} \leftarrow temp$ 
endfor
```



kompleksitas waktu asimptotik

“Big-O”

Notasi Big-O

Notasi "O" disebut Big-O merupakan notasi **kompleksitas waktu asimptotik**

Definisi 1.

$T(n) = O(f(n))$ (dibaca " $T(n)$ adalah $O(f(n))$ "), yang artinya $T(n)$ berorde paling besar $f(n)$ bila terdapat konstanta C dan n_0 sedemikian hingga :

$$T(n) \leq C f(n); \text{ untuk } n \geq n_0$$

- $f(n)$ adalah batas lebih atas (upper bound) dari $T(n)$ untuk n yang besar
- Catatan : Ada tak berhingga nilai C dan n_0 yang memenuhi $T(n) \leq C f(n)$, kita cukup menunjukkan satu pasang (C, n_0) yang memenuhi definisi sehingga $T(n) = O(f(n))$

CONTOH 4

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

Penyelesaian :

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

$2n^2 + 6n + 1 \leq 2n^2 + 6n^2 + n^2 = 9n^2$ untuk semua $n \geq 1$

atau karena

($C=9, f(n)=n^2, n_0=1$).

$2n^2 + 6n + 1 \leq n^2 + n^2 + n^2 = 3n^2$ untuk semua $n \geq 7$

($C=3, f(n)=n^2, n_0=7$).

CONTOH 5

Tunjukkan bahwa: $3n + 2 = O(n)$

Penyelesaian:

$3n + 2 = O(n)$ karena

$3n + 2 \leqslant 3n + 2n = 5n$ untuk semua $n \geqslant 1$

$(C = 5, f(n) = n, \text{ dan } n_0 = 1).$

CONTOH 6

Tunjukkan bahwa: $5 = O(1)$

Penyelesaian:

$5 = O(1)$ karena $5 \leqslant 6 \times 1$ untuk $n \geqslant 1$ ($C = 6, f(n) = 1$, dan $n_0 = 1$)

Kita juga dapat memperlihatkan bahwa

$5 = O(1)$ karena $5 \leqslant 10 \times 1$ untuk $n \geqslant 1$ ($C = 10, f(n) = 1$, dan $n_0 = 1$)

CONTOH 7

Tunjukkan bahwa kompleksitas waktu algoritma *selection sort* adalah :

$$T(n) = \frac{n(n-1)}{2} = O(n^2)$$

Penyelesaian :

$$\frac{n(n-1)}{2} = O(n^2)$$

karena

$$\frac{n(n-1)}{2} \leq \frac{1}{2}n^2 + \frac{1}{2}n^2 = n^2 \text{ untuk } n \geq 1$$

($C = 1$, $f(n) = n^2$, dan $n_0 = 1$).

CONTOH 8

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

Penyelesaian:

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

$6n^2 + 2n^2 \leqslant 6 \times 2^n + 2 \times 2^n = 8 \times 2^n$ untuk $n \geqslant 1$

($C = 8$, $f(n) = 2^n$, dan $n_0 = 1$)

CONTOH 9

- Tunjukkan $1 + 2 + \dots + n = O(n^2)$

Penyelesaian:

Cara 1: $1 + 2 + \dots + n \leq n + n + \dots + n = n^2$ untuk $n \geq 1$

Cara 2: $1 + 2 + \dots + n = \frac{1}{2}n(n+1) = \frac{1}{2}n^2 + \frac{1}{2}n^2 = n^2$ untuk $n \geq 1$

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

Penyelesaian:

$n! = 1 \times 2 \times \dots \times n \leq n \times n \times \dots \times n = n^n$ untuk $n \geq 1$

CONTOH 10

- Tunjukkan $\log n! = O(n \log n)$

Penyelesaian :

Dari contoh 9 sudah diperoleh bahwa $n! \leq n^n$ untuk $n \geq 1$ maka
 $\log n! \leq \log n^n = n \log n$ untuk $n \geq 1$ maka
sehingga $\log n! = O(n \log n)$

- Tunjukkan $8n^2 = O(n^3)$

Penyelesaian :

$8n^2 = O(n^3)$ karena $8n^2 \leq O(n^3)$

6. Tunjukkan $\log n! = O(n \log n)$

Jawaban:

Dari soal 5 sudah diperoleh bahwa $n! \leq n^n$ untuk $n \geq 1$ maka
 $\log n! \leq \log n^n = n \log n$ untuk $n \geq 1$ maka
sehingga $\log n! = O(n \log n)$

7. Tunjukkan $8n^2 = O(n^3)$

Jawaban:

$8n^2 = O(n^3)$ karena $8n^2 \leq n^3$ untuk $n \geq 8$

Teorema 1: Bila $T(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$ adalah polinom derajat $\leq m$ maka $T(n) = O(n^m)$.

- Jadi, untuk menentukan notasi *Big-Oh*, cukup melihat suku (*term*) yang mempunyai pangkat terbesar di dalam $T(n)$.

- **Contoh 8:**

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

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

$$T(n) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

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

- Teorema 1 tersebut digeneralisasi untuk suku-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 9: $T(n) = 2^n + 2n^2 = O(2^n)$.

$$T(n) = 2n \log(n) + 3n = O(n \log n)$$

$$T(n) = \log n^3 = 3 \log(n) = O(\log n)$$

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

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
$$T(n) = 2n^2 = O(n^2) \rightarrow \text{benar.}$$
$$T(n) = 2n^2 = O(n^3) \rightarrow \text{juga benar, karena } 2n^2 \leq 2n^3 \text{ untuk } n \geq 1$$
$$T(n) = 2n^2 = O(n^4) \rightarrow \text{juga benar, karena } 2n^2 \leq 2n^4 \text{ untuk } n \geq 1$$
- Namun, untuk alasan praktikal 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 $O(n^4)$

Perhatikan ...(3)

- Menuliskan
 - $O(2n)$ tidak standard, seharusnya $O(n)$
 - $O(n - 1)$ tidak standard, seharusnya $O(n)$
 - $O(\frac{n^2}{2})$ tidak standard, seharusnya $O(n^2)$
 - $O((n - 1)!)$ tidak standard, seharusnya $O(n!)$
- Ingat, di dalam notasi Big-Oh tidak ada koefisien atau suku-suku lainnya, hanya berisi fungsi-fungsi standard seperti $1, n^2, n^3, \dots, \log n, n \log n, 2^n, n!,$ dan sebagainya

TEOREMA 2. 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) $f(n) = O(f(n))$

Contoh 9. Misalkan $T_1(n) = O(n)$ dan $T_2(n) = O(n^2)$, maka

- (a) $T_1(n) + T_2(n) = O(\max(n, n^2)) = O(n^2)$
- (b) $T_1(n)T_2(n) = O(nn^2) = O(n^3)$

Contoh 10. $O(5n^2) = O(n^2)$

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

Contoh 11: Tentukan notasi O -besar untuk $T(n) = (n + 1)\log(n^2 + 1) + 3n^2$.

Jawaban:

Cara 1: • $n + 1 = O(n)$

$$\bullet \log(n^2 + 1) \leq \log(2n^2) = \log(2) + \log(n^2)$$

$$= \log(2) + 2 \log(n)$$

$$\leq \log(n) + 2 \log(n) = 3 \log(n) \text{ untuk } n \geq 2$$

$$= O(\log n)$$

$$\bullet (n + 1) \log(n^2 + 1) = O(n) O(\log n) = O(n \log n)$$

$$\bullet 3n^2 = O(n^2)$$

$$\bullet (n + 1) \log(n^2 + 1) + 3n^2 = O(n \log n) + O(n^2) = O(\max(n \log n, n^2)) = O(n^2)$$

Cara 2: suku yang dominan di dalam $(n + 1)\log(n^2 + 1) + 3n^2$ untuk n yang besar adalah $3n^2$, sehingga $(n + 1) \log(n^2 + 1) + 3n^2 = O(n^2)$

Pengelompokan Algoritma Berdasarkan Notasi O -Besar

Kelompok Algoritma	Nama
$O(1)$	konstan
$O(\log n)$	logaritmik
$O(n)$	linier
$O(n \log n)$	linier logaritmik
$O(n^2)$	kuadratik
$O(n^3)$	kubik
$O(2^n)$	eksponensial
$O(n!)$	faktorial

Urutan spektrum kompleksitas waktu algoritma adalah :

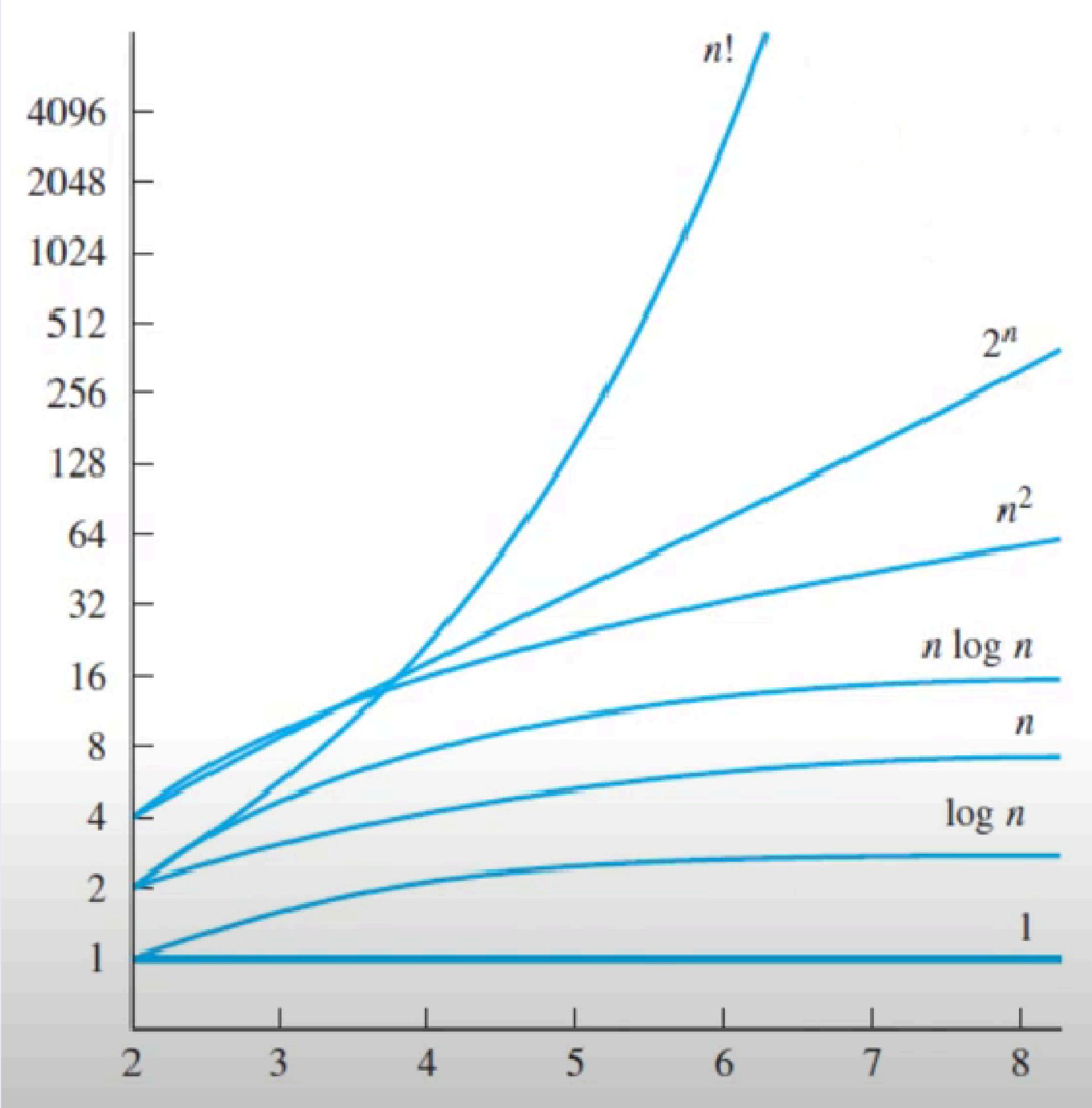
$$\underbrace{1 < \log n < n < n \log n < n^2 < n^3 < \dots < 2^n < n!}_{\downarrow \quad \downarrow}$$

algoritma polinomial
(bagus)

algoritma eksponensial
(buruk)

Nilai masing-masing fungsi untuk setiap bermacam-macam nilai n

$\log n$	n	$n \log n$	n^2	n^3	2^n	$n!$
0	1	0	1	1	2	1
1	2	2	4	8	4	2
2	4	8	16	64	16	24
3	8	24	64	512	256	362880
4	16	64	256	4096	65536	20922789888000
5	32	160	1024	32768	4294967296	(terlalu besar untuk ditulis)



Kegunaan Notasi *Big-Oh*

- Notasi *Big-Oh* berguna untuk membandingkan beberapa algoritma untuk persoalan yang sama
→ menentukan yang terbaik.
- Contoh: persoalan pengurutan memiliki banyak algoritma penyelesaian,
Selection sort, bubble 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*.

LATIHAN

Tentukan kompleksitas waktu dari algoritma dibawah ini dihitung dari banyaknya operasi penjumlahan $a \leftarrow a + 1$

```
for  $i \leftarrow 1$  to  $n$  do
    for  $j \leftarrow 1$  to  $i$  do
        for  $k \leftarrow j$  to  $n$  do
             $a \leftarrow a + 1$ 
    endfor
endfor
endfor
```

Jawaban

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 = $T(n) = n^2 + (n - 1)^2 + (n - 2)^2 + \dots + 1$

Menentukan Notasi Big-O suatu Algoritma

Cara 1:

- Tentukan $T(n)$ dari algoritma.
- Notasi Big-O dapat langsung ditentukan dengan mengambil suku yang mendominasi fungsi T dan menghilangkan koefisiennya.

Contoh:

1. Algoritma mencari nilai maksimum: $T(n) = n - 1 = O(n)$

2. Algoritma sequential search:

$$T_{\min}(n) = 1 = O(1), \quad T_{\max}(n) = n = O(n), \quad T_{\text{avg}}(n) = (n + 1)/2 = O(n)$$

3. Algoritma *selection sort*: $T(n) = \frac{n(n - 1)}{2} = O(n^2)$

Cara 2:

- Setiap operasi yang terdapat di dalam algoritma (baca/tulis, *assignment*, operasi aritmetika, operasi perbandingan, dll) memiliki kompleksitas $O(1)$. Jumlahkan semuanya.
- Jika ada pengulangan, hitung jumlah pengulangan, lalu kalikan dengan total Big-O semua instruksi di dalam pengulangan
- Contoh 1:

read(x)	$O(1)$
if $x \bmod 2 = 0$ then	$O(1)$
$x \leftarrow x + 1$	$O(1)$
write(x)	$O(1)$
else	
write(x)	$O(1)$
endif	

Kompleksitas waktu asimptotik algoritma:
 $= O(1) + O(1) + \max(O(1)+O(1), O(1))$
 $= O(1) + \max(O(1), O(1))$
 $= O(1) + O(1)$
 $= O(1)$

CONTOH

jumlah \leftarrow 0	$O(1)$
$i \leftarrow 2$	$O(1)$
while $i \leq n$ do	$O(1)$
$jumlah \leftarrow jumlah + a[i]$	$O(1)$
$i \leftarrow i + 1$	$O(1)$
endwhile	
$rata \leftarrow jumlah/n$	$O(1)$

Kalang **while** dieksekusi sebanyak $n - 1$ kali, sehingga kompleksitas asimptotiknya

$$\begin{aligned} &= O(1) + O(1) + (n - 1) \{ O(1) + O(1) + O(1) \} + O(1) \\ &= O(1) + (n - 1) O(1) + O(1) \\ &= O(1) + O(1) + O(n - 1) \\ &= O(1) + O(n) \\ &= O(\max(1, n)) = O(n) \end{aligned}$$

Jadi, kompleksitas waktu algoritma adalah $O(n)$.

CONTOH

```
for  $i \leftarrow 1$  to  $n$  do
    for  $j \leftarrow 1$  to  $i$  do
         $a \leftarrow a + 1$        $O(1)$ 
         $b \leftarrow b - 2$        $O(1)$ 
    endfor
endfor
```

Kompleksitas untuk $a \leftarrow a + 1$	$= O(1)$
Kompleksitas untuk $b \leftarrow b - 2$	$= O(1)$
Kompleksitas total keduanya	$= O(1) + O(1) = O(1)$
Jumlah pengulangan seluruhnya	$= 1 + 2 + \dots + n = n(n + 1)/2$
Kompleksitas seluruhnya	$= n(n + 1)/2 O(1) = O(n(n + 1)/2)$ $= O(n^2/2 + n/2)$ $= O(n^2)$

Latihan Mandiri

1. Untuk soal (a) sampai (e) berikut, tentukan $C, f(n), n_0$, dan notasi O -besar sedemikian sehingga $T(n) = O(f(n))$ jika $T(n) \leq Cf(n)$ untuk semua $n \geq n_0$:
 - (a) $T(n) = 2 + 4 + 6 + \dots + 2n$
 - (b) $T(n) = (n + 1)(n + 3)/(n + 2)$
 - (c) $T(n) = \textcolor{brown}{n} \log(\textcolor{brown}{n}^2 + 1) + \textcolor{brown}{n}^2 \log n$
 - (d) $T(n) = (\textcolor{brown}{n} \log \textcolor{blue}{n} + 1)^2 + (\log \textcolor{brown}{n} + 1)(\textcolor{brown}{n}^2 + 1)$
 - (e) $T(n) = \textcolor{brown}{n}^{2^n} + \textcolor{brown}{n}^{n^2}$

Perhatikan potongan kode java berikut :

```
int a = 0, b = 0;
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        a = a + j;
    }
}
for (k = 0; k < N; k++) {
    b = b + k;
}
```

- (a) Hitung kompleksitas waktu algoritma berdasarkan banyaknya operasi penjumlahan

**GA PEKA
BGT SIHHH!**

JANGAN BERI KAMI

tugas lagi

ADA TUGAS
GAESSS !



OY LIBUR OY