Kompleksitas Algoritma (Bagian 1)

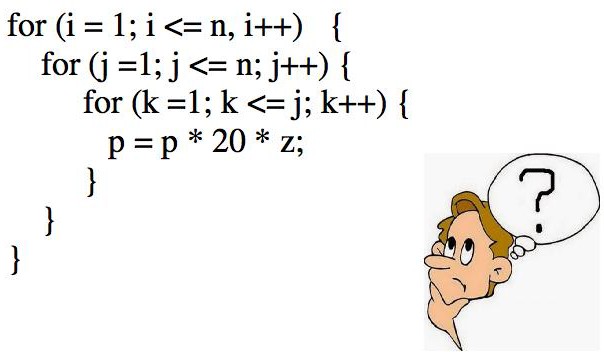
Bahan Kuliah IF2120 Matematika Diskrit

Oleh: Rinaldi Munir

### Program Studi Teknik Informatika

**STEI - ITB**

1



Rinaldi M/IF2120 Matdis 2

# Pendahuluan

* Sebuah algoritma tidak saja harus benar (sesuai spesifikasi persoalan), tetapi juga harus sangkil (*efisien*).
* Algoritma yang bagus adalah algoritma yang sangkil (*efficient*).
* Kesangkilan algoritma diukur dari waktu (*time*) yang diperlukan untuk menjalankan algoritma dan ruang (*space*) memori yang dibutuhkan oleh algoritma tersebut.
* Algoritma yang sangkil ialah algoritma yang **meminimumkan** kebutuhan waktu dan ruang memori.

3

#### Kebutuhan waktu dan ruang memori suatu algoritma bergantung pada ukuran masukan (*n*), yang menyatakan ukuran data yang diproses oleh algoritma.

* + Kesangkilan algoritma dapat digunakan untuk menilai algoritma yang bagus dari sejumlah algoritma penyelesaian persoalan.
  + Sebab, sebuah persoalan dapat memiliki banyak algoritma penyelesaian. Contoh: persoalan pengurutan (*sort*), ada puluhan algoritma pengurutan (*selection sort*, *insertion sort*, *bubble sort*, dll).

Rinaldi M/IF2120 Matdis 4

#### Mengapa kita memerlukan algoritma yang sangkil? Lihat grafik di bawah ini.

Waktu komputasi (dalam detik)

10-4 x 2*n*

105

1 hari

10-6 x 2*n*

104

1 jam

103

10-4 x *n*3

102

1 menit

10

1

10 x *n*

-6

3

1 detik

5

10

15

20

25

30

35

40

10-1

Ukuran masukan

Rinaldi M/IF2120 Matdis 5

# Model Perhitungan Kebutuhan Waktu

* Menghitung kebutuhan waktu algoritma dengan mengukur waktu eksekusi riil nya (dalam satuan detik) ketika program (yang merepresentasikan sebuah algoritma) dijalankan oleh komputer bukanlah cara yang tepat.
* Alasan:
  1. Setiap komputer dengan arsitektur berbeda memiliki bahasa mesin yang berbeda  waktu setiap operasi antara satu komputer dengan komputer lain tidak sama.
  2. *Compiler* bahasa pemrograman yang berbeda menghasilkan kode Bahasa mesin yang berbeda  waktu setiap operasi antara *compiler* dengan *compiler* lain tidak sama.

Rinaldi M/IF2120 Matdis 6

#### Model abstrak pengukuran waktu/ruang memori algoritma harus independen dari pertimbangan mesin (*computer*) dan *compiler* apapun.

* Besaran yang dipakai untuk menerangkan model abstrak pengukuran waktu/ruang ini adalah **kompleksitas algoritma**.
* Ada dua macam kompleksitas algoritma, yaitu: **kompleksitas waktu**

(*time complexity*) dan **kompleksitas ruang** (*space complexity*).

Rinaldi M/IF2120 Matdis 7

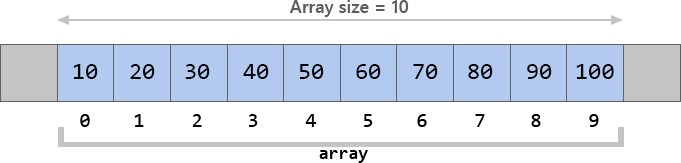
* Kompleksitas waktu, *T*(*n*), diukur dari jumlah tahapan komputasi yang dilakukan di dalam algoritma sebagai fungsi dari ukuran masukan *n*.
* Kompleksitas ruang, *S*(*n*), diukur dari memori yang digunakan oleh struktur data

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*.
* Di dalam kuliah ini kita hanya membatasi bahasan kompleksitas waktu saja, karena dua alasan:

1. Materi struktur data diluar lingkup mata kuliah matematika diskrit
2. Saat ini memori komputer bukan persoalan yang kritis dibandingkan waktu

Rinaldi M/IF2120 Matdis 8

* + Ukuran masukan (*n*) menyatakan banyaknya data yang diproses oleh sebuah algoritma.

Contoh:

1. algoritma pengurutan 10 elemen larik (*array*), maka *n* = 10.
2. algoritma pencarian pada 500 elemen larik, maka *n* = 500
3. algoritma *TSP* pada sebuah graf lengkap dengan 100 simpul, maka *n* = 100.
4. algoritma perkalian 2 buah matriks berukuran 50 x 50, maka *n* = 50.
5. algoritma menghitung polinom dengan derajat  100, maka *n* = 100
   * Dalam perhitungan kompleksitas waktu, ukuran masukan dinyatakan sebagai variabel *n* saja (bukan instans suatu nilai).

Rinaldi M/IF2120 Matdis 9

# Kompleksitas Waktu

* + - Pekerjaan utama di dalam kompleksitas waktu adalah menghitung (*counting*) jumlah

tahapan komputasi di dalam algoritma .

* + - Jumlah tahapan komputasi dihitung dari berapa kali suatu operasi dilakukan sebagai fungsi ukuran masukan (*n*).
    - Di dalam sebuah algoritma terdapat banyak jenis operasi:
      * Operasi baca/tulis (input a, print a)
      * Operasi aritmetika (+, -, \*, /) ( a + b, M \* N)
      * Operasi pengisian nilai (*assignment*) ( a  10)
      * Operasi perbandingan ( a < b, k >= 10)
      * Operasi pengaksesan elemen larik, pemanggilan prosedur/fungsi, dll
    - Untuk menyederhanakan perhitungan, kita tidak menghitung semua jenis operasi, tetapi kita hanya menghitung jumlah operasi khas (tipikal) yang *mendasari* suatu algoritma.

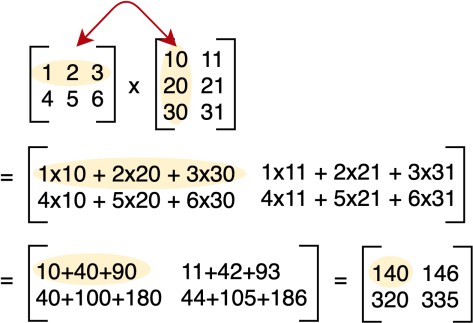
Rinaldi M/IF2120 Matdis 10

### Graphical user interface, website Description automatically generatedContoh operasi khas di dalam algoritma

* + - * Algoritma pencarian (*searching*)

Operasi khas: operasi perbandingan elemen larik

* + - * Algoritma pengurutan (*sorting*)

Operasi khas: operasi perbandingan elemen dan operasi pertukaran elemen

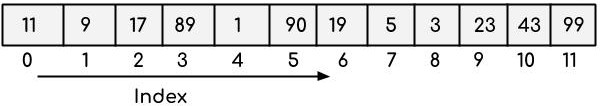
* + - * Algoritma perkalian dua buah matriks *AB* = *C*

Operasi khas: operasi perkalian dan penjumlahan

* + - * Algoritma menghitung nilai sebuah polinom p(x) = a0 + a1x + a2x2 + … + anxn Operasi khas: operasi perkalian dan penjumlahan

Rinaldi M/IF2120 Matdis 11

**Contoh 1.** Tinjau algoritma menghitung rerata elemen di dalam sebuah larik (*array*).

*sum* 0

**for** *i*  1 **to** *n* **do**

*sum*  *sum* + *a*[*i*]

endfor

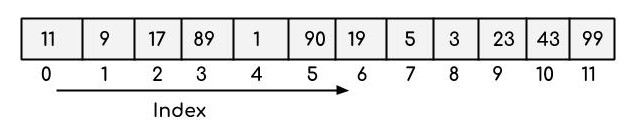
*rata\_rata*  *sum*/*n*

* + - Operasi yang mendasar pada algoritma tersebut adalah operasi penjumlahan elemen-elemen larik (yaitu *sum**sum*+*a*[*i*] ) yang dilakukan sebanyak *n* kali.
    - Kompleksitas waktu: *T*(*n*) = *n.*

Rinaldi M/IF2120 Matdis 12

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

**procedure** *CariElemenTerbesar*(**input** *a*1, *a*2, ..., *an* : **integer**, **output** *maks* : **integer**)



*{ Mencari elemen terbesar dari sekumpulan elemen larik integer a1, a2, ..., an. Elemen terbesar akan disimpan di dalam maks. }*

**Deklarasi**

*k* : **integer Algoritma**

*maks*  *a*1

*k*2

**while** *k*  *n* **do**

**if** *ak* > *maks* **then**

*maks**ak*

**endif**

*k*  *k* + 1

**endwhile**

Kompleksitas waktu algoritma dihitung dari jumlah operasi perbandingan elemen larik (*ak* > *maks*).

Kompleksitas waktu *CariElemenTerbesar* : *T*(*n*) = *n* – 1.

Kompleksitas waktu dibedakan atas tiga macam :

1. *Tmax(n)* : kompleksitas waktu untuk kasus terburuk *(worst case),*

 kebutuhan waktu maksimum.

1. *Tmin(n)* : kompleksitas waktu untuk kasus terbaik *(best case),*

 kebutuhan waktu minimum.

1. *Tavg(n)*: kompleksitas waktu untuk kasus rata-rata *(average case)*

 kebutuhan waktu secara rata-rata

**Contoh 3.** Algoritma *sequential search* (linear search)

**procedure** PencarianBeruntun(**input** *a*1, *a*2, ..., *an* : **integer**, *x* : **integer**, **output** *idx* : *integer*)



*{ Mencari elemen x di dalam larik A yang berisi n elemen. Jika x ditemukan, maka indeks elemen larik disimpan*

*di dalam idx, idx bernilai –1 jika x tidak ditemukan*

**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**

**if** *ketemu* **then** *{ x ditemukan } idx**k*

**else**

*idx* –1 *{ 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 *an* = *x* atau *x* tidak ditemukan.

*T*max(*n*) = *n*

##### *Kasus rata-rata*: Jika *x* ditemukan pada posisi ke-*j*, maka operasi perbandingan (ak = x)akan dieksekusi sebanyak *j* kali.

1 *n*(1  *n*)

*T*avg(*n*) =

(1  2  3  ...  *n*)

###### *n*

#####  2 

###### *n*

(*n*  1) 2

Rinaldi M/IF2120 Matdis 16

#### Cara lain: asumsikan bahwa *P*(*aj* = *x*) = 1/*n*. Jika *aj* = *x* maka *Tj* yang dibutuhkan adalah *Tj = j.* Jumlah perbandingan elemen larik rata-rata:

*T*avg(*n*) = 𝑛

𝑗 =1

𝑇𝑗 𝑃(𝑎[𝑗] = 𝑥) = 𝑛

1 1

𝑇𝑗 𝑛 = 𝑛

𝑗 =1

𝑛

𝑗 =1

𝑇𝑗

#### = 1 

*n*

*j* = 1

(*n*(*n*

####  1)) 

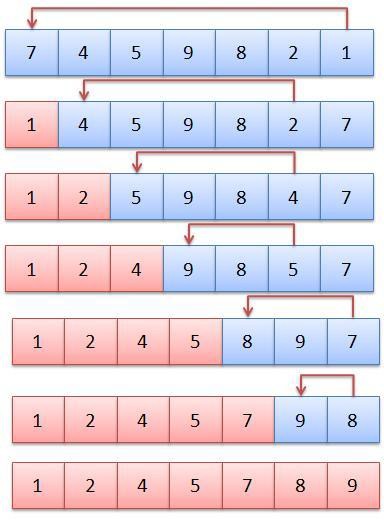
*n*  1

*n j* 1 *n* 2 2

Rinaldi M/IF2120 Matdis 17

**Contoh 4**: Algoritma pengurutan seleksi (*selection sort*)

**procedure** *SelectionSort*(**input/output** *a*1, *a*2, ..., *an* : **integer**)



*{ Mengurutkan elemen-elemen larik A yang berisi n elemen integer sehingga terurut menaik }*

**Deklarasi**

*i*, *j*, *imin*, *temp* : **integer Algoritma**

**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*

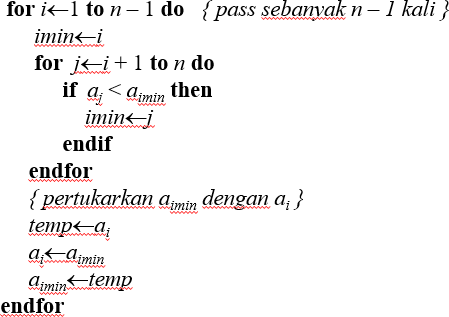
**endfor**

pass ke-1 pass ke-2

pass ke-6

1. Jumlah operasi perbandingan elemen-elemen larik (*aj* < *aimin* )

Untuk setiap pass ke-*i*,

*i* = 1  jumlah perbandingan = *n* – 1 *i* = 2  jumlah perbandingan = *n* – 2 *i* = 3  jumlah perbandingan = *n* – 3

⋮

*i* = n – 1  jumlah perbandingan = 1

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah

𝑛(𝑛−1)

*T*(*n*) = (*n* – 1) + (*n* – 2) + … + 2 + 1 =

2

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

### Jumlah operasi pertukaran

Untuk setiap *i* dari 1 sampai *n* – 1, terjadi satu kali pertukaran elemen, sehingga jumlah operasi pertukaran seluruhnya adalah

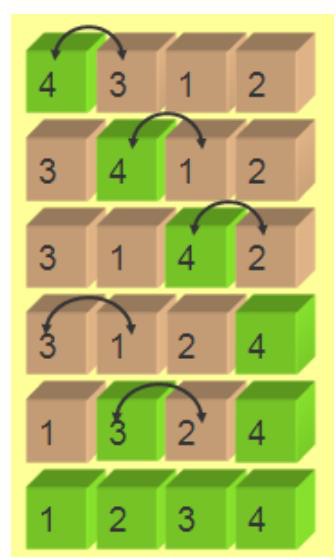
*T*(*n*) = *n* – 1.

Ini adalah jumlah pertukaran untuk semua kasus.

Jadi, algoritma pengurutan seleksi membutuhkan *n*(*n* – 1 )/2 buah operasi perbandingan elemen dan *n* – 1 buah operasi pertukaran.

**Contoh 5**: Diberikan algoritma pengurutan *bubble-sort* seperti berikut ini. Hitung kompleksitas waktu algoritma didasarkan pada jumlah operasi perbandingan elemen-elemen larik dan jumlah operasi pertukaran.

**procedure** *BubbleSort*(**input/output** *a*1, *a*2, ..., *an* : **integer**)



*{ Mengurut larik A yang berisi n elemen integer sehingga terrut menaik }*

**Deklarasi**

*i, j*, *temp* : **integer Algoritma**

**for** *i*  *n* – 1 **downto** 1 do

**for** *j*  1 **to** *i* do

**if** *aj +* 1 < *aj* **then**

*{ pertukarkan aj dengan aj + 1 } temp*  *aj*

*aj*  *aj* + 1

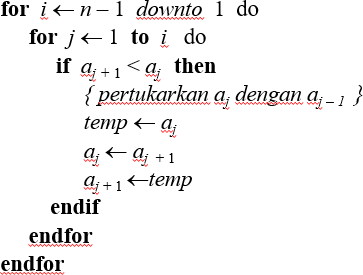
*aj* + 1 *temp*

**endif**

**endfor endfor**

Sumber gambar: Prof. Amr Goneid, Department of Computer Science, AUC

1. Jumlah operasi perbandingan elemen-elemen larik (*aj* +1 < *aj* )

Untuk setiap *pass* ke-*i*,

*i* = *n* – 1  jumlah perbandingan = *n* – 1 *i* = *n* – 2  jumlah perbandingan = *n* – 2 *i* = *n* – 3  jumlah perbandingan = *n* – 3

⋮

*i* = 1  jumlah perbandingan = 1

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah

𝑛(𝑛−1)

*T*(*n*) = (*n* – 1) + (*n* – 2) + … + 2 + 1 =

2

Ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma *BubbleSort* tidak bergantung pada apakah data masukannya sudah terurut atau acak. Jumlah operasi perbandingan sama dengan *selection sort.*

1. **Jumlah operasi pertukaran (***temp**ai* ; *ai**aimin* ; *aimin**temp* **)**

Jumlah operasi pertukaran di dalam *bubble sort* hanya dapat dihitung pada kasus terbaik dan kasus terburuk. Kasus terbaik adalah tidak ada pertukaran (yaitu jika **if** *aj +* 1 < *aj* false), yaitu semua elemen larik pada awalnya sudah terurut menaik, sehingga

*Tmin* (*n*) = 0.

Pada kasus terburuk, (yaitu jika **if** *aj +* 1 < *aj* bernilai true), pertukaran elemen selalu dilakukan. Jadi, jumlah operasi pertukaran elemen pada kasus terburuk sama dengan jumlah operasi perbandingan elemen-elemen larik, yaitu

*Tmax*(*n*) = (*n* – 1) + (*n* – 2) + … + 2 + 1 =

𝑛(𝑛−1) 2

Jadi, algoritma pengurutan *bubble sort* membutuhkan *n*(*n* – 1 )/2 buah operasi pertukaran, lebih banyak daripada algoritma *selection sort*. Ini berarti secara keseluruhan *bubble sort* lebih buruk daripada *selection sort*.

## Latihan 1

Hitung kompleksitas waktu algoritma berikut berdasarkan jumlah operasi perkalian.

**procedure** *Kali*(**input** *x* : **integer**, *n* : **integer**, **output** *jumlah* : **integer**)

*{Mengalikan x dengan i = 1, 2, …, j, yang dalam hal ini j = n, n/2, n/4, …,1. Hasil perkalian disimpan di dalam peubah jumlah. }*

**Deklarasi**

*i*, *j*, *k* : **integer**

**Algoritma**

*j*  *n*

**while** *j*  1 **do**

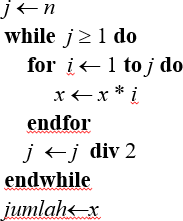
**for** *i*  1 **to** *j* **do**

*x*  *x* \* *i*

**endfor**

*j*  *j* **div** 2 **endwhile** *jumlah**x*

Rinaldi M/IF2120 Matdis 24

Jawaban

Untuk

*j* = *n*, jumlah operasi perkalian = *n*

*j* = *n*/2, jumlah operasi perkalian = *n*/2 *j* = *n*/4, jumlah operasi perkalian = *n*/4

…

*j* = 1, jumlah operasi perkalian = 1

Jumlah operasi perkalian seluruhnya adalah

= *n* + *n*/2 + *n*/4 + … + 2 + 1  deret geometri

*n*(1  2 2 log *n*1 )

2(*n* − 1)

= 1 

1 

2

2n – 1

Rinaldi M/IF2120 Matdis 25

## Latihan 2

Di bawah ini adalah algoritma untuk menguji apakah dua buah matriks, *A* dan *B*,

yang masing-masing berukuran *n*  *n*, sama.

**function** *samaMatriks*(*A*, *B* : *matriks*; *n* : integer)  *boolean*

*{ true jika A dan B sama; sebaliknya false jika A * *B }*

**Deklarasi**

*i*, *j* : **integer Algoritma:**

**for** *i*  1 **to** *n* **do**

**for** j  1 **to** *n* **do**

**if** *Ai,j*  *Bi,j* **then**

**return false**

**endif endfor**

**endfor return true**

1. Apa kasus terbaik dan terburuk untuk algoritma di atas?
2. Tentukan kompleksitas waktu terbaik dan terburuknya.

Jawaban:

#### Kasus terbaik terjadi jika ketidaksamaan matriks ditemukan pada elemen pertama (*A*1,1  *B*1,1)

Kasus terburuk terjadi jika ketidaksamaan matriks ditemukan pada elemen ujung kanan bawah (*A*n,n  *B*n,n) atau pada kasus matriks A dan B sama, sehingga seluruh elemen matriks dibandingkan.

1. Tmin(*n*) = 1

Tmax(*n*) = *n*2

## Latihan Mandiri

1. Diberikan matriks persegi berukuran n x n. Hitung kompleksitas waktu untuk memeriksa apakah matriks tersebut merupakan matriks simetri terhadap diagonal utama.
2. Berapa kompleksitas waktu untuk menjumlahkan matriks A dan B yang keduanya berukuran n x n?
3. Ulangi soal 2 untuk perkalian matriks A dan B.
4. Tulislah algoritma pengurutan *insertion sort* pada larik yang berukuran n elemen, hitung masing-masing kompleksitas waktu algoritma diukur dari jumlah operasi perbandingan dan jumlah operasi pertukaran elemen-elemen larik.
5. Berapa kali operasi penjumlahan pada potongan algoritma ini dilakukan?

**for** *i*  1 **to** *n* **do for** *j*  1 **to** *n* **do**

**for** *k*  1 to *j* do

*x*  *x* + 1

endfor endfor

**endfor**

1. Algoritma di bawah ini menghitung nilai polinom *p*(*x*) = *a*0 + *a*1*x* + *a*2*x*2 + … + *anxn*

**function** *p*(**input** *x*:**real**)**real**

*{ Mengembalikan nilai p(x)}*

**Deklarasi**

*j*, *k* : **integer**

*jumlah*, *suku* : **real**

**Algoritma**

*jumlah*  *a*0

**for** *j*  1 **to** *n* **do**

*{ hitung ajxj } suku*  *aj*

**for** *k*  1 **to** *j* **do**

*suku*  *suku* \* *x*

**endfor**

*jumlah*  *jumlah* + *suku endfor*

**return** *jumlah*

Hitunglah berapa operasi perkalian dan berapa operasi penjumlahan yang dilakukan oleh algoritma tsb

Rinaldi M/IF2120 Matdis 30

Algoritma menghitung polinom yang lebih baik dapat dibuat dengan metode Horner berikut: *p*(*x*) = *a*0 + *x*(*a*1 + *x*(*a*2 + *x*(*a*3 + … + *x*(*an*-1 + *anx*)))…))

**function** *p2*(**input** *x*:**real**)**real**

*{ Mengembalikan nilai p(x) dengan metode Horner}*

**Deklarasi**

*k* : **integer**

*b*1, *b*2, ..., *bn* : **real Algoritma**

*bn**an*

**for** *k**n* – 1 **downto** 0 **do**

*bk**ak* + *bk*+1 \* *x*

**endfor**

**return** *b*0

Hitunglah berapa operasi perkalian dan berapa operasi penjumlahan yang dilakukan oleh algoritma di atas? Manakah yang terbaik, algoritma *p* atau *p*2?

Rinaldi M/IF2120 Matdis 31

BERSAMBUNG