Kompleksitas Algoritma

- Sebuah algoritma tidak saja harus benar, tetapi juga harus mangkus (*efisien*).

- Algoritma yang bagus adalah algoritma yang mangkus.

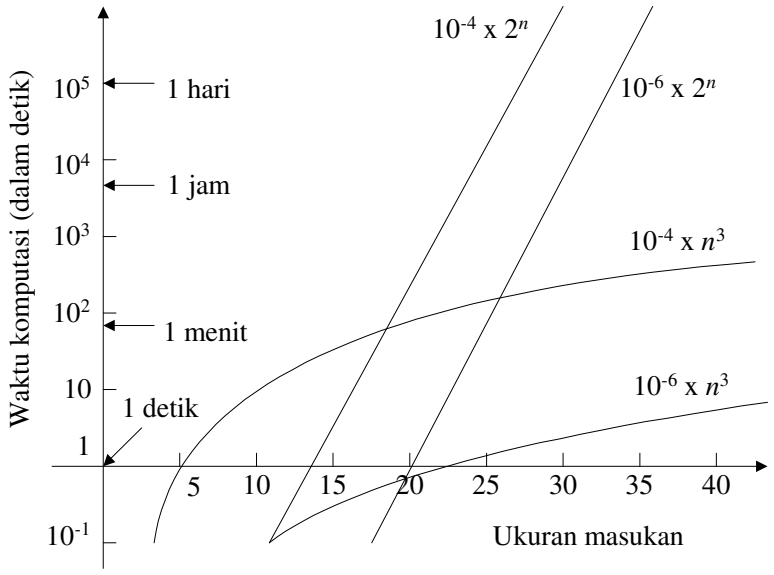
- Kemangkusan algoritma diukur dari berapa jumlah waktu dan ruang (*space*) memori yang dibutuhkan untuk menjalankannya.

- 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?



Model Perhitungan Kebutuhan Waktu/Ruang

- Kita dapat mengukur waktu yang diperlukan oleh sebuah algoritma dengan menghitung banyaknya operasi/instruksi yang dieksekusi.

- Jika kita mengetahui besaran waktu (dalam satuan detik) untuk melaksanakan sebuah operasi tertentu, maka kita dapat menghitung berapa waktu sesungguhnya untuk melaksanakan algoritma tersebut.

Contoh 1. Menghitung rerata

Larik bilangan bulat

```
procedure HitungRerata(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 }
```

(i) Operasi pengisian nilai (`jumlah`←`0`, `k`←`1`, `jumlah`←`jumlah+`a_k, `k`←`k+1`, dan `r` ← `jumlah/n`)

Jumlah seluruh operasi pengisian nilai adalah

$$t_1 = 1 + 1 + n + n + 1 = 3 + 2n$$

(ii) Operasi penjumlahan (`jumlah+`a_k, dan `k+1`)

Jumlah seluruh operasi penjumlahan adalah

$$t_2 = n + n = 2n$$

(iii) Operasi pembagian (`jumlah/n`)

Jumlah seluruh operasi pembagian adalah

$$t_3 = 1$$

Total kebutuhan waktu algoritma `HitungRerata`:

$$t = t_1 + t_2 + t_3 = (3 + 2n)\text{a} + 2nb + c \text{ detik}$$

- Model perhitungan kebutuhan waktu seperti di atas kurang dapat diterima:
  1. Dalam praktek kita tidak mempunyai informasi berapa waktu sesungguhnya untuk melaksanakan suatu operasi tertentu

  2. Komputer dengan arsitektur yang berbeda akan berbeda pula lama waktu untuk setiap jenis operasinya.

- Selain bergantung pada komputer, kebutuhan waktu sebuah program juga ditentukan oleh *compiler* bahasa yang digunakan.

- Model abstrak pengukuran waktu/ruang harus independen dari pertimbangan mesin 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** dan **kompleksitas ruang**.

- 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

- Dalam praktek, kompleksitas waktu dihitung berdasarkan jumlah operasi abstrak yang *mendasari* suatu algoritma, dan memisahkan analisisnya dari implementasi.

Contoh 2. Tinjau algoritma menghitung rerata pada Contoh 1. Operasi yang mendasar pada algoritma tersebut adalah operasi penjumlahan elemen-elemen a_k (yaitu `jumlah`$\leftarrow$`jumlah+`a_k),

Kompleksitas waktu `HitungRerata` adalah $T(n) = n$.

Contoh 3. Algoritma untuk mencari elemen terbesar di dalam sebuah larik (*array*) yang berukuran *n* elemen.

```
procedure CariElemenTerbesar(input a1, a2, ..., an : integer, output
maks : integer)
{ Mencari elemen terbesar dari sekumpulan elemen larik integer a1, a2,
..., an.
  Elemen terbesar akan disimpan di dalam maks.
  Masukan: a1, a2, ..., an
  Keluaran: maks (nilai terbesar)
}
Deklarasi
   k : integer

Algoritma
   maks←a1
   k←2
   while k ≤ n do
     if ak > maks then
        maks←ak
     endif
     i←i+1
   endwhile
   { k > n }
```

Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi perbandingan elemen larik (`A[i] > 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. $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata *(average case)*
   → kebutuhan waktu secara rata-rata

Contoh 4. Algoritma *sequential search*.

```
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    { x ditemukan }
     idx←k
  else
     idx← 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.

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

Cara lain: asumsikan bahwa $P(a_j = x) = 1/n$. Jika $a_j = x$ maka T_j yang dibutuhkan adalah $T_j = j$. Jumlah perbandingan elemen larik rata-rata:

$$T_{\text{avg}}(n) = \sum_{j=1}^{n} T_j P(A[j] = X) = \sum_{j=1}^{n} T_j \frac{1}{n} = \frac{1}{n}\sum_{j=1}^{n} T_j$$

$$= \frac{1}{n}\sum_{j=1}^{n} j = \frac{1}{n}\left(\frac{n(n+1)}{2}\right) = \frac{n+1}{2}$$

Contoh 5. Algoritma pencarian biner (*bynary search*).

```
procedure PencarianBiner(input a1, a2, ..., an : integer, x : integer,
                         output idx : integer)
Deklarasi
   i, j, mid : integer
   ketemu : boolean

Algoritma
   i←1
   j←n
   ketemu←false
   while (not ketemu) and ( i ≤ j) do
      mid ← (i+j) div 2
      if amid = x then
        ketemu ← true
      else
         if amid < x then    { cari di belahan kanan }
          i←mid + 1
         else                   { cari di belahan kiri }
          j←mid - 1;
         endif
      endif
   endwhile
   {ketemu or i > j }

   if ketemu then
      idx←mid
   else
      idx←0
   endif
```

1. *Kasus terbaik*

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

2. *Kasus terburuk*:

$T_{\max}(n) = {}^2\log n$

Contoh 6. Algoritma algoritma pengurutan pilih (*selection sort*).

```
procedure Urut(input/output a1, a2, ..., an : integer)
Deklarasi
   i, j, imaks, temp : integer

Algoritma
  for i←n downto 2 do   { pass sebanyak n - 1 kali }
    imaks←1
    for j←2 to i do
      if aj > aimaks then
         imaks←j
      endif
    endfor
    { pertukarkan aimaks dengan ai }
    temp←ai
    ai←aimaks
    aimaks←temp

  endfor
```

(i) Jumlah operasi perbandingan elemen

Untuk setiap *pass* ke-*i*,

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

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

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

$\vdots$

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

$\vdots$

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

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah

$$T(n) = (n-1) + (n-2) + \ldots + 1 = \sum_{i=1}^{n-1} n-k = \frac{n(n-1)}{2}$$

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

(ii) Jumlah operasi pertukaran

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

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

Jadi, algoritma pengurutan maksimum membutuhkan $n(n - 1)/2$ buah operasi perbandingan elemen dan $n - 1$ buah operasi pertukaran.

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. 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**.

DEFINISI. $T(n) = O(f(n))$ (dibaca "$T(n)$ adalah $O(f(n)$" yang artinya *T(n) berorde paling besar f(n)*) bila terdapat konstanta C dan n_0 sedemikian sehingga

$$T(n) \leq C(f(n))$$

untuk $n \geq n_0$.

$f(n)$ adalah batas atas (*upper bound*) dari $T(n)$ untuk n yang besar.

Contoh 7. Tunjukkan bahwa $T(n) = 3n + 2 = O(n)$.

Penyelesaian:

$3n + 2 = O(n)$

karena

$3n + 2 \leq 3n + 2n = 5n$ untuk semua $n \geq 1$ ($C = 5$ dan $n_0 = 1$).

Contoh 8. Tunjukkan bahwa $T(n) = 2n^2 + 6n + 1 = O(n^2)$.

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

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

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

TEOREMA. 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) $T_1(n)T_2(n) = O(n.n^2) = O(n^3)$

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

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