Analisis Framework

Review

- Tujuan analisa : mengukur efesiensi algoritma
- Efisiensi diukur dari diukur dari: waktu (*time*) dan memori (*space*).
- Dua besaran yang digunakan: kompleksitas algoritma
  1. Kompleksitas waktu – T(n)
  2. Kompleksitas ruang – S(n)

Analisis Framework

- Mengukur ukuran atau jumlah input
- Mengukur waktu eksekusi
- Tingkat pertumbuhan
- Efiesiensi worst-case, best-case dan average-case

Measuring Input Space (n)
(Mengukur jumlah input)

- Ukuran masukan (*n*): jumlah data yang diproses oleh sebuah algoritma.
- Hampir semua algoritma membutuhkan waktu yang lebih lama untuk menyelesaikan jika inputannya lebih banyak
- Contoh
  - pengurutan 1000 elemen larik, maka $n = 1000$
  - pencarian elemen pada matriks 3 x 4, maka n=12
  - algoritma *TSP* pada sebuah graf lengkap dengan 100 simpul, maka $n = 100$.

- Dalam praktek perhitungan kompleksitas, ukuran masukan dinyatakan sebagai variabel *n*.

Measuring Running Time (Pengukuran Running Time)

- Dapat menggunakan satuan waktu standar (s/ms), tapi tergantung pada kecepatan komputer, kualitas program, compiler.
- Running time ditentukan oleh **operasi dasar,** yaitu operasi yang paling mendominasi sebagian besar running time algoritma. Operasi tersebut memakai waktu yang paling banyak dari keseluruhan running time.
- Operasi dasar dapat berupa:
  - penjumlahan/pengurangan
  - pengurangan
  - perkalian
  - pengaksesan memori
  - pembacaan
  - penulisan
- Menghitung jumlah waktu yang diperlukan untuk mengeksekusi operasi dasar.
- Framework yang telah ada untuk analisis efisiensi waktu suatu algoritma : menghitung jumlah waktu eksekusi operasi dasar dalam algoritma dengan inputan ukuran n.

Contoh operasi khas di dalam algoritma

- Algoritma pencarian di dalam larik
  Operasi khas: perbandingan elemen larik

- Algoritma pengurutan
  Operasi khas: perbandingan elemen, pertukaran elemen

- Algoritma penjumlahan 2 buah matriks
  Operasi khas: penjumlahan

- Algoritma perkalian 2 buah matriks
  Operasi khas: perkalian dan penjumlahan

Contoh : 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` : $T(n) = n - 1$.

Tingkat Pertumbuhan

- Untuk n dengan jumlah besar, yang diperhitungkan adalah fungsi tingkat pertumbuhan
- Abaikan pengali yang konstan
- Exp : ½ n → n, 50 n log n → n log n
- Fungsi penting :
- $\log_2 n$, n, $n \log_2 n$, n^2 n^3 2^n n!

Order of Growth

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10	3.3×10	10^2	10^3	10^3	3.6×10^6
10^2	6.6	10^2	6.6×10^2	10^4	10^6	1.3×10^{30}	9.3×10^{157}
10^3	10	10^3	1.0×10^4	10^6	10^9		
10^4	13	10^4	1.3×10^5	10^8	10^{12}		
10^5	17	10^5	1.7×10^6	10^{10}	10^{15}		
10^6	20	10^6	2.0×10^7	10^{12}	10^{18}		

Worst case, best case, average case

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 2. 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.

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

Contoh 3. Algoritma pengurutan seleksi (*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,

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

$i = n - 1 \quad \rightarrow$ jumlah perbandingan $= n - 2$

$i = n - 2 \quad \rightarrow$ jumlah perbandingan $= n - 3$

$\vdots$

$i = 2 \rightarrow$ 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.

Latihan

- **Contoh 4.** Hitung kompleksitas waktu algoritma berikut berdasarkan jumlah operasi kali.

```
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
  Masukan: x dan n (n adalah perpangakatan dua).
  Keluaran: 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
   { j > 1 }
   jumlah←x
```

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

  $= n + n/2 + n/4 + \ldots + 2 + 1$ → deret geometri

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

Next...

Kompleksitas Waktu Asimptotik