Analisis Algoritm

Fundamentals of the Anlysis of Algorithm Efficiency

Hendri Karisma
Program Studi Teknik Informatika
Universitas Komputer Indonesia
2013

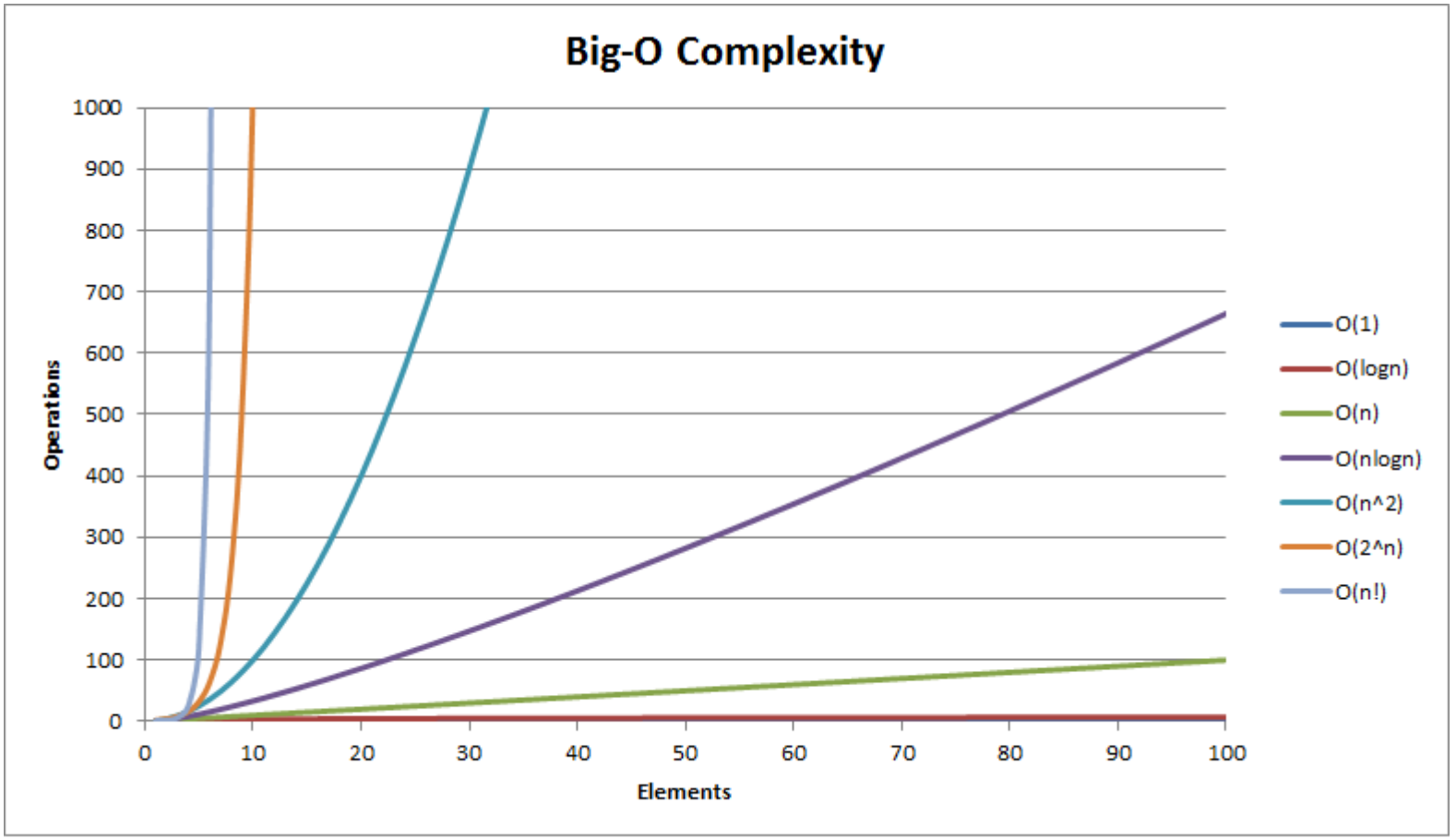
Review

- *An algorithm is a sequence of unambiguous **instructions** for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.*
- *Suatu algoritma harus benar (efektif) : Tujuannya dapat dicapai.*
- Suatu algoritma harus dapat berjalan dengan cepat dan lebih ringan (efisien) / mangkus.

Fundamentals of Algorithmic Problem Solving (Review)

- Understanding the Problem
- Ascertaining the Capabilities of the Computational Device
- Choosing between Exact and Approximate Problem Solving
- Algorithm Design Techniques (strategy)
- Designing an Algorithm and Data Structures
- Methods of Specifying an Algorithm (exp: pseudocode)
- Proving an Algorithm's Correctness
- Analyzing an Algorithm
  - (efficiency, effectivity, simplycity, generality)
- Coding an Algorithm

Kompleksitas Algoritma



Apa itu anlisis algoritma

- Artinya melakukan investigasi terhadap suatu algoritma dari sis efisiensi dengan melihat dua sumber yaitu, running time dan memory space.
- Efisiensi didapatkan dalam bentuk quantitative

Materi Kompleksitas

- Dasar Analisis Kompleksitas
- Notasi asimptotik
- Non-recursively algorithm
- Recursively algorithm

Kompleksitas

- Time Complexity

  Mengindikasikan seberapa cepat algoritma.

  Diekspresikan sebagai jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma.

- Space Complexity

  Seberapa besar memory yang dibutuhkan algoritma terutama untuk input dan output.

  Diekspresikan sebagai jumlah memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai ukuran masukan *n*

Terminologi Kompleksitas Ruang dan Waktu

- Ukuran masukan data untuk suatu algoritma, *n.*

  Sebagai contaoh adalah elemen-elemen larik, n adalah jumlah larik atau produk matriks yang berupa n x n. Semakin besar n, maka semakin kompleks.
- Kompleksitas waktu, dengan symbol T(n), adalah jumlah operasi yang dilakukan untuk menjalankan sebuah algoritma sebagai fungsi dari ukuran masukan n
- Kompleksitas ruang, dengan symbol S(n), adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari ukuran masukan n

Kompleksitas Ruang

- Materi diluar bahasan, yaitu struktur data.
- Kebutuhan akan ruang memori pada teknologi masa kini tidak lagi menjadi persoalan kritis, karena komputer sekarang memiliki ukuran memori yang besar dibandingkan komputer mainframe 25 tahun lalu.
- Namun bukan berarti kita melupakan kompleksitas ruang.

Komleksitas Waktu (dahulu kala)

- Misalkan c_{op} adalah waktu eksekusi algoritma operasi dasar, dan C(n) adalah jumlah waktu operasi tersebut butuh dieksekusi untuk algoritma tersebut, maka kita dapat menghitung

$$T(n) \approx c_{op} C(n).$$

- Namun itu hanya bisa mendekati, artinya, kita tidak bisa mendapatkan perhitungan aslinya, kecuali data yang diolah sangat kecil.
- Kita hanya bisa mengatakan suatu algoritma berjalan lebih cepat atau 10 kali lebih cepat atau jawabannya t = 10.
- Apa yang terjadi jika kita menggandakan inputnya jika $C(n) = \frac{1}{2}n(n-1),$?

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

$$\frac{T(2n)}{T(n)} \approx \frac{c_{op}C(2n)}{c_{op}C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4.$$

Namun formula tersebut tidak berhasil seiring penambahan data yang besar

Komplekasitas Function

TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Bagaimana mendapatkan fungsi kompleksitas algoritma?

- Tetapkan ukuran masukan.
- Mengukur kompleksitas waktu dengan menentukan banyaknya operasi yang dilakukan oleh algoritma.
- Cukup menghitung jumlah operasi abstrak (operasi dasar) yang mendasari suatu algoritma dan memisahkan analisanya dari implementasi.
- Operasi dasar contohnya : pencarian operasi abstraknya jumlah n-kali perbandingan elemen, untuk operasi pengurutan: jumlah pertukaran elemen.

Contoh

```
1   Procedure SeqSearchTanpaSentinel (Input nama_array:tipe_array)
2   {I.S. : elemen array [1..maks_array] sudah terdefinisi}
3   {F.S. : menampilkan hasil pencarian (ditemukan/tidak)}
4   Kamus:
5       i : integer
6       data_cari   :   tipedata
7   Algoritma:
8       input(data_cari)
9       i ← 1
10      while(nama_array [i] ≠ data_cari) and (i < maks_array) do
11          i ← i + 1
12      endwhile
13      if (nama_array[i] = data_cari)
14      then
15          output(data_cari,' ditemukan pada indeks ke-',i)
16      else
17          output(data_cari,' tidak ditemukan')
18      endif
19  EndProcedure
```

Seq Search

```
Procedure SeqSearchBoolean (Input nama_array:tipe_array)
{I.S. : elemen array [1..maks_array] sudah terdefinisi}
{F.S. : menampilkan data yg dicari ditemukan atau tidak ditemukan}
Kamus:
     i : integer
     ketemu : boolean
     data_cari : tipedata
Algoritma:
     input(data_cari) (1)
     I = 1 (1)
     ketemu ≔ false (1)
     while (not ketemu) and (i ≤ maks_array) do (n)
          if(nama_var_array(i) = data_cari) (1)
          then
               ketemu ≔ true    (1)
          else                   OR
               i ≔ i + 1        (1)
          endif
     endwhile
     if (ketemu)  (1)
     then
          output(data_cari,' ditemukan pada indeks ke-',i)  (1)
     Else                                                  (OR)
           output(data_cari,' tidak ditemukan')             (1)
     endif
EndProcedure
```

Kompleksitas Seq Search

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

  $= 2n + 5$

Kompleksitas Waktu

- Worse Case :
  - Kompleksitas waktu untuk kasus terburuk, yaitu kebutuhan waktu maksimum yang diperlukan sebuah algoritma sebagai fungsi dari n. $T_{max}(n)$
- Best Case :
  - Kompleksitas waktu untuk kasus terbaik, yaitu kebutuhan waktu minimum yang diperlukan sebuah algoritma sebagai fungsi dari n. $T_{min}(n)$
- Avarage Case:
  - Kompleksitas waktu untuk kasus rata-rata, yaitu kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari n. Untuk kebutuhan rata-rata ini, biasanya dibuat asumsi bahwa semua barisan masukan bersifat sama. Misalkan pada persoalan pencarian diandaikan bahwa data yang dicari mempunyai peluang yang sama untuk terletak dalam larik. $T_{avg}(n)$

Kembali ke Seq Search

```
Procedure SeqSearchBoolean (Input nama_array:tipe_array)
{I.S. : elemen array [1..maks_array] sudah terdefinisi}
{F.S. : menampilkan data yg dicari ditemukan atau tidak ditemukan}
Kamus:
     i : integer
     ketemu : boolean
     data_cari : tipedata
Algoritma:
     input(data_cari) (1)
     I = 1 (1)
     ketemu ≔ false (1)
     while (not ketemu) and (i ≤ maks_array) do (n)
          if(nama_var_array(i) = data_cari) (1)
          then
               ketemu ≔ true    (1)
          else                   OR
               i ≔ i + 1        (1)
          endif
     endwhile
     if (ketemu)  (1)
     then
          output(data_cari,' ditemukan pada indeks ke-',i)  (1)
     Else                                                  (OR)
           output(data_cari,' tidak ditemukan')             (1)
     endif
EndProcedure
```

Contoh pada Algoritma Seq Search

- Best Case
  - Ketika datanya hanya 1 atau data langsung ketemu di index nomor 1:
  - $T_{min}(n) = 1$
- Worse Case
  - Ketika data yang dicari berada di paling akhir data (indeks terakhir) :
  - $T_{max}(n) = 2n + 5 = n$

Contoh pada Algoritma Seq Search

- Avarage Case
  - Data sebanyak n dengan pencarian rata-rata:
  - $T_{avg}(n) = 1+1+1+(1+2+3+...(n+(n+1)))/n+1+1+1$

    $= (1/2n(1+1+(n-1)1))/n+3$

    $= (1/2n(1+n))/n$

    $= 1/2(1+n)$

- *Ingat!*

  Un = a + (n-1)b = bn + (a-b)

  Sn = 1/2 n(a+Un)

  - *a = suku awal*
  - *b = beda*
  - *n = banyak suku*
  - *Un = a + (n - 1) b adalah suku ke-n*

Binary Search

```
Procedure  BinarySearch (Input nama_array : tipe_array)
{I.S. : elemen array yang terurut secara ascending sudah terdefinisi}
{F.S. : menampilkan data yg dicari ditemukan atau tidak ditemukan}
Kamus:
     Ia, Ib, k  :  integer
     ketemu  :  boolean
     data_cari  :  tipedata
Algoritma:
     input(data_cari)
     Ia ← 1
     Ib ← maks_array
     ketemu ← false
     while (not ketemu) and (Ia ≤ Ib) do
          k  ← (Ia + Ib) div 2
          if (nama_var_array[k] = data_cari)
            then
               ketemu ←  true
            else
            if (nama_var_array[k] < data_cari)
             then
                 Ia  ←  k + 1
             else
                 Ib  ←  k - 1
            endif
          endif
     endwhile
```

Kompleksitas Binary Search

- Best Case
  - Sama seperti sebelumnya jika data langsung ditemukan (data berada ditengah).
  - $T_{min}(n) = 1$
- Worse Case
  - Data berada di akhir perulangan.
    - Looping-1 = n
    - Looping-2 = n/2
    - Looping-3 = (n/2)/2 = n/4
    - Looping-4 = (n/4)/2 = n/8
    - Looping-5 = (n/8)/2 = n/16 ===>>>> n/(2^(m-1))
    - m : langkah looping ke-m
    - Sehingga nilainya sama dengan $Log_2(n+1)$

Kompleksitas Binary Search

- Worse Case
  - Worse Case pada binary search sama dengan Avarage Case dikarenakan prosesnya yang melakukan pembagian jumlah langkah pada setiap step perulangan.

Sekian...