LAPORAN PRAKTIKUM

ANALISIS ALGORITMA

Disusun Oleh:

Kefilino Khalifa Filardi

140810180028

PROGRAM STUDI S-1 TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

JATINANGOR

2020

MODUL PRAKTIKUM 3

KOMPLEKSITAS WAKTU ASIMPTOTIK DARI ALGORITMA

MATA KULIAH
ANALISIS ALGORITMA
D10G.4205 & D10K.0400601

PENGAJAR : (1) MIRA SURYANI, S.Pd., M.Kom
(2) INO SURYANA, Drs., M.Kom
(3) R. SUDRAJAT, Drs., M.Si
FAKULTAS : MIPA
SEMESTER : IV dan VI

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN

MARET 2019

Pendahuluan

Minggu lalu kita sudah mempelajari menghitung kompleksitas waktu T(n) untuk semua operasi yang ada pada suatu algoritma. Idealnya, kita memang harus menghitung semua operasi tersebut. Namun, untuk alasan praktis, kita cukup menghitung operasi abstrak yang **mendasari suatu algoritma**, dan memisahkan analisisnya dari implementasi. Contoh pada algoritma searching, operasi abstrak yang mendasarinya adalah operasi perbandingan elemen x dengan elemen-elemen dalam larik. Dengan menghitung berapa perbandingan untuk tiap-tiap elemen nilai n sehingga kita dapat memperoleh **efisiensi relative** dari algoritma tersebut. Setelah mengetahui T(n) kita dapat menentukan **kompleksitas waktu asimptotik** yang dinyatakan dalam notasi Big-O, Big-Ω, Big-ϴ, dan little-ω.

Setelah mengenal macam-macam kompleksitas waktu algoritma (best case, worst case, dan average case), dalam analisis algoritma kita selalu mengutamakan perhitungan **worst case** dengan alasan sebagai berikut:

- Worst-case running time merupakan *upper bound* (batas atas) dari running time untuk input apapun. Hal ini memberikan jaminan bahwa algoritma yang kita jalankan tidak akan lebih lama lagi dari ***worst-case***
- Untuk beberapa algoritma, ***worst-case*** cukup sering terjadi. Dalam beberapa aplikasi pencarian, pencarian info yang tidak ada mungkin sering dilakukan.
- Pada kasus ***average-case*** umumnya lebih sering seperti ***worst-case. Contoh:*** misalkan kita secara random memilih angka dan mengimplementasikan insertion sort, ***average-case* = *worst-case*** yaitu fungsi kuadratik dari .

Perhitungan worst case (*upper bound*) dalam kompleksitas waktu asimptotik dapat menggunakan **Big-O Notation. Perhatikan pembentukan Big-O Notation berikut!**

Misalkan kita memiliki kompleksitas waktu T(n) dari sebuah algoritma sebagai berikut:

$$()= 2 +6+1$$

- Untuk yang besar, pertumbuhan () sebanding dengan
- Suku 6+1 tidak berarti jika dibandingkan dengan 2 , dan boleh diabaikan sehingga T(n) = 2 + suku-suku lainnya.
- Koefisien 2 pada 2 boleh diabaikan, sehingga T(n) = O( ) → **Kompleksitas Waktu Asimptotik**

DEFINISI BIG-O NOTATION

Definisi 1. ()= (()) ***artinya*** () berorde paling besar () bila terdapat konstanta C dan sedemikian sehingga

$$\boxed{T(n) \leq C.f(n)}$$

Untuk ≥

Jika dibuat semakin besar, waktu yang dibutuhkan tidak akan melebihi konstanta dikalikan dengan (), → () adalah *upper bound*.

Dalam proses **pembuktian Big-O**, perlu dicari nilai dan nilai C sedemikan sehingga terpenuhi kondisi () ≤.().

Contoh soal 1:

Tunjukan bahwa, ()=2 +6+1= ()

Penyelesaian:

Kita mengamati bahwa ≥1, maka ≤ dan 1≤ sehingga

$$2 \quad +6+1 \leq 2 \qquad +6+ \qquad = 9, \qquad \geq 1$$

Maka kita bisa mengambil C=9 dan =1 untuk memperlihatkan:

$$()=2 +6+1= ()$$

BIG-O NOTATION DARI POLINOMIAL BERDERAJAT M

Big-O Notation juga dapat ditentukan dari Polinomial n berderajat m, dengan TEOREMA 1 sebagai berikut:

Polinomial berderajat dapat digunakan untuk memperkirakan kompleksitas waktu asimptotik dengan mengabaikan suku berorde rendah

Contoh: ()= + 6 + +8= (), dinyatakan pada

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

Artinya kita mengambil suku paling tinggi derajatnya ("Mendominasi") yang diartikan laju pertumbuhannya lebih cepat dibandingkan yang lainnya ketika diberikan sembarang besaran input. Besaran dominan lainnya adalah:

- Eksponensial mendominasi sembarang perpangkatan (yaitu, > ,>1)
- Perpangkatan mendominasi ln (yaitu >ln)
- Semua logaritma tumbuh pada laju yang sama (yaitu log()= log()
- log tumbuh lebih cepat daripada tetapi lebih lambat dari

Teorema lain dari Big-O Notation yang harus dihafalkan untuk membantu kita menentukan nilai Big-O dari suatu algoritma adalah:

TEOREMA 2
Misalkan $T_1(n) = O(f(n))$ dan $T_2(n) = O(g(n))$, maka

(a)(i) $T_1(n) + T_2(n) = O(\max(f(n), g(n)))$

(ii) $T_1(n) + T_2(n) = O(f(n) + g(n))$

(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) $f(n) = O(f(n))$

Berikut adalah contoh soal yang mengaplikasikan Teorema 2 dari Big-O notation:

Contoh Soal 2

Misalkan, ()= () ()= (), dan ()= ( ), dengan m sebagai peubah, maka

(a) ()+ ()= (max((,)= ( ) Teorema 2(a)(i)

(b) ()+ ()= (+) Teorema 2(a)(ii)

(c) ().()= (.)= () Teorema 2(b)

Contoh Soal 3

(d) (5)= () Teorema 2(c)

(e) = () Teorema 2(d)

Aturan Menentukan Kompleksitas Waktu Asimptotik

• Cara I

Jika kompleksitas waktu T(n) dari algoritma sudah dihitung, maka kompleksitas waktu asimptotiknya dapat langsung ditentukan dengan mengambil suku yang mendominasi fungsi T dan menghilangkan koefisiennya (sesuai TEOREMA 1) **Contoh:**

Pada algoritma cariMax, ()= −1= ()

• Cara 2

Kita bisa langsung menggunakan notasi Big-O, dengan cara:

Pengisian nilai (assignment), perbandingan, operasi aritmatika (+,-,/,*, div, mod), read, write, pengaksesan elemen larik, memilih field tertentu dari sebuah record, dan pemanggilan function/void membutuhkan waktu O(1)

Contoh Soal 4:

Tinjau potongan algoritma berikut: read(x) O(1)

x ← x + 1 O(1) + O(1)=O(1)

write(x) O(1)

Kompleksitas waktu asimptotik algoritmanya (1)+(1)+(1)= (1)

Penjelasan:

$O(1) + O(1) + O(1) = O(max(1,1)) + O(1)$ Teorema 2(a)(i)

= (1)+(1)

= ((1,1)) Teorema 2(a)(ii)

= (1)

DEFINISI BIG-Ω DAN BIG-Θ NOTATION

Notasi Big-O hanya menyediakan batas atas *(upper bound)* untuk perhitungan kompleksitas waktu asimptotik, tetapi tidak menyediakan batas bawah *(lower bound)*. Untuk itu, lower bound dapat ditentukan dengan Big-Ω Notation dan Big-θ Notation.

Definisi Big-Ω Notation:

()=Ω(g(n)) yang artinya ()berorde paling kecil () bila terdapat konstanta C dan sedemikian sehingga

$$T(n) \geq C.(g(n))$$

untuk ≥

Definisi Big-θ Notation:

$T(n) = \theta(h(n))$ yang artinya $T(n)$ berorde sama dengan $h(n)$ jika $T(n) = O(h(n))$ dan $T(n) = \Omega\,(g(n))$

Contoh Soal 5:

Tentukan Big-Ω dan Big- Θ Notation untuk ()=2 +6+1

Penyelesaian:

Karena 2 +6+1≥2 untuk ≥1, dengan mengambil C=2, kita memperoleh

2 +6+1= ()

Karena 2 +6+1= () dan 2 +6+1= (), maka 2 +6+1= ( )

Penentuan Big-Ω dan Big-Θ dari Polinomial Berderajat m

Sebuah fakta yang berguna dalam menentukan orde kompleksitas adalah dari suku tertinggi di dalam polinomial berdasarkan teorema berikut:

TEOREMA 3 Bila $T(n) = a_m n^m + a_{m-1} n^{m-1} + a_1 n + a_0$ adalah polinom berderajat m maka $T(n) = n^m$

Contoh soal 6:

Bila ()= 6 +12+24+2,

maka T(n) adalah berorde , yaitu (),Ω(), Θ().

Latihan Analisa

Minggu ini kegiatan praktikum difokuskan pada latihan menganalisa, sebagian besar tidak perlu menggunakan komputer dan mengkoding program, gunakan pensil dan kertas untuk menjawab persoalan berikut!

1. Untuk ()=2+4+6+8+16+⋯+ , tentukan nilai C, f(n), , dan notasi Big-O sedemikian sehingga ()= (()) jika ()≤ ≥
   Jawaban :
   $T(n) = a(r^n - 1)/(r - 1) = 2(2^n - 1)/(2 - 1) = 2^{n+1} - 2 = 2^n$
   Notasi Big-O adalah $O(2^n)$

   Mencari nilai C, f(n), n_o :
   f(n) adalah ordo paling tinggi makan $f(n) = n^2$
   $T(n) \leq C \cdot 2^n$ dengan $n_0 = 1$
   $2^1 \leq C \cdot 2^1$
   $C \geq 1$

2. Buktikan bahwa untuk konstanta-konstanta positif p, q, dan r:
   ()= + + adalah (),Ω(), Θ()
   Jawaban :
   - Pembuktian Big-O
     $T(n) \leq C \cdot f(n)$
     $pn^2 + qn + r \leq C \cdot n^2$
     misal $n_o = p = q = r = 1$, maka
     $1(1)^2 + 1(1) + 1 \leq C \cdot (1)^2$
     $1 + 1 + 1 \leq C$
     $C \geq 3$

   - Pembuktian Big-Ω
     $T(n) \geq C \cdot f(n)$
     $pn^2 + qn + r \geq C \cdot n^2$
     misal $n_o = p = q = r = 1$, maka
     $1(1)^2 + 1(1) + 1 \geq C \cdot (1)^2$
     $1 + 1 + 1 \geq C$
     $C \leq 3$

   - Pembuktian Big-Θ
     Karena $O(n^2)$ dan $\Omega(n^2)$ terbukti dan sama maka $\Theta(n^2)$ pun benar

3. Tentukan waktu kompleksitas asimptotik (Big-O, Big-Ω, dan Big-Θ) dari kode program berikut:
   for k ← 1 to n do   for i ← 1 to n do
   for j ← to n do
   ← or and
   endfor
   endfor
   endfor
   Jawaban :
   W_{ij} ← W_{ij} or W_{ik} and W_{kj} = 1
   for j ← to n do = n
   for i ← 1 to n do = n
   for k ← 1 to n do = n

   T(n) = 1 . n . n . n = n^3
   - Big-O
     $T(n) \leq C . f(n)$
     $n^3 \leq C . n^3$
     misal n_o = 1, maka
     $(1)^3 \leq C . (1)^3$
     $1 \leq C$
     $C \geq 1$

   - Big-Ω
     $T(n) \geq C . f(n)$
     $n^3 \geq C . n^3$
     misal n_o = 1, maka
     $(1)^3 \geq C . (1)^3$
     $1 \geq C$
     $C \leq 1$

   - Big-Θ
     Karena $O(n^3) = \Omega(n^3)$ maka $\Theta(n^3)$

4. Tulislah algoritma untuk menjumlahkan dua buah matriks yang masing-masing berukuran n x n. Berapa kompleksitas waktunya T(n)? dan berapa kompleksitas waktu asimptotiknya yang dinyatakan dalam Big-O, Big-Ω, dan Big-Θ?
   Jawaban :
```
for i := 1 to N do
    for j := 1 to M do
        C[i, j] := A[i, j] + B[i, j]
    endfor
endfor
```

   T(n) = 1 . n . n = n^2
   - Big-O
     $T(n) \leq C . f(n)$
     $n^2 \leq C . n^2$
     misal n_o = 1, maka
     $(1)^2 \leq C . (1)^2$
     $1 \leq C$
     $C \geq 1$

- Big-Ω
  $T(n) \geq C \cdot f(n)$
  $n^2 \geq C \cdot n^2$
  misal $n_0 = 1$, maka
  $(1)^2 \geq C \cdot (1)^2$
  $1 \geq C$
  $C \leq 1$

- Big-Θ
  Karena $O(n^2) = \Omega(n^2)$ maka $\Theta(n^2)$

5. Tulislah algoritma untuk menyalin (copy) isi sebuah larik ke larik lain. Ukuran elemen larik adalah n elemen. Berapa kompleksitas waktunya T(n)? dan berapa kompleksitas waktu asimptotiknya yang dinyatakan dalam Big-O, Big-Ω, dan Big-Θ?
   Jawaban :
```
for i := 1 to N do
    B[i] := A[i]
endfor
```

   $T(n) = 1 \cdot n = n$
   - Big-O
     $T(n) \leq C \cdot f(n)$
     $n \leq C \cdot n$
     misal $n_0 = 1$, maka
     $1 \leq C \cdot 1$
     $1 \leq C$
     $C \geq 1$

   - Big-Ω
     $T(n) \geq C \cdot f(n)$
     $n \geq C \cdot n$
     misal $n_0 = 1$, maka
     $1 \geq C \cdot 1$
     $1 \geq C$
     $C \leq 1$

   - Big-Θ
     Karena $O(n^2) = \Omega(n^2)$ maka $\Theta(n^2)$

6. Diberikan algoritma Bubble Sort sebagai berikut:

```
procedure BubbleSort(input/output a1, a2, ..., an : integer)
{ Mengurut tabel integer TabInt[1..n] dengan metode pengurutan bubble-
sort
  Masukan: a1, a2, ..., an
  Keluaran: a1, a2, ..., an (terurut menaik)
}
Deklarasi
   k : integer      { indeks untuk traversal tabel }
   pass : integer { tahapan pengurutan }
   temp : integer { peubah bantu untuk pertukaran elemen tabel }
Algoritma
   for pass ← 1 to n - 1 do
     for k ← n downto pass + 1 do
       if ak < ak-1 then
           { pertukarkan ak dengan ak-1 }
           temp ← ak
           ak ← ak-1
           ak-1←temp
       endif
     endfor
   endfor
```

(a) Hitung berapa jumlah operasi perbandingan elemen-elemen tabel!

Jawaban :

$T(n) = (n - 1) + (n - 2) + (n - 3) + \ldots + 1$

$= n \cdot (n - 1)/2 = (n^2 - n)/2$

(b) Berapa kali maksimum pertukaran elemen-elemen tabel dilakukan?

Jawaban :

Maksimum pertukaran elemen-elemen tabel adalah $(n^2 - n)/2$

(c) Hitung kompleksitas waktu asimptotik (Big-O, Big-Ω, dan Big-Θ) dari algoritma Bubble Sort tersebut!

Jawaban :

- Big-O

Perbandingan = $(n^2 - n)/2$

Pertukaran Nilai = $3 \cdot (n^2 - n)/2$

$T_{max}(n) = (n^2 - n)/2 + 3 \cdot (n^2 - n)/2$

$= 4 \cdot (n^2 - n)/2$

$= 2 \cdot (n^2 - n)$

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

$2 \cdot (n^2 - n) \leq C \cdot n^2$

Misal $n_0 = 1$, maka

$2 \cdot (1^2 - 1) \leq C \cdot 1^2$

$2 \cdot 0 \leq C$

$C \geq 0$

- Big-Ω

  Perbandingan saja = $(n^2 - n)/2$

  $T_{min}(n) = (n^2 - n)/2$

  $T(n) \geq C \cdot f(n)$

  $(n^2 - n)/2 \geq C \cdot n^2$

  Misal $n_0 = 1$, maka

  $(1^2 - 1)/2 \geq C \cdot 1^2$

  $0/2 \geq C$

  $C \leq 0$

- Big-Θ

  Karena $O(n^2) = \Omega(n^2)$ maka $\Theta(n^2)$

7. Untuk menyelesaikan problem X dengan ukuran N tersedia 3 macam algoritma:
   (a) Algoritma A mempunyai kompleksitas waktu O(log N)
   (b) Algoritma B mempunyai kompleksitas waktu O(N log N)
   (c) Algoritma C mempunyai kompleksitas waktu $O(N^2)$

   Untuk problem X dengan ukuran N=8, algoritma manakah yang paling cepat? Secara asimptotik, algoritma manakah yang paling cepat?

   Jawaban :

   A -> O(log N) = O(log 8) = 0.90308998699

   B -> O(N log N) = O(8 log 8) = 7.22471989594

   C -> $O(N^2) = O(8^2) = 64$

   Algoritma A lebih cepat dibanding algoritma B dan C.

8. Algoritma mengevaluasi polinom yang lebih baik dapat dibuat dengan metode Horner berikut:

   $$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \ldots + x(a_{n-1} + a_n x)))\ldots))$$

```
function p2(input x : real) → real
{ Mengembalikan nilai p(x) dengan metode Horner}

Deklarasi
  k : integer
  b1, b2, ..., bn : real

Algoritma
  bn ← an
  for k ← n - 1 downto 0 do
     bk ← ak + bk+1 * x
  endfor
  return b0
```

   Hitunglah berapa operasi perkalian dan penjumlahan yang dilakukan oleh algoritma diatas, Jumlahkan kedua hitungan tersebut, lalu tentukan kompleksitas waktu asimptotik (Big-O)nya. Manakah yang terbaik, algoritma p atau p2?

Jawaban :

Algoritma P_2

bn <- an = 1

bk <- $a_k + b_{k+1} . x$ = n

$T(n) = n + 1$

$O(n)$ untuk P_2

Algoritma P

$a_k + b_{k+1} . x$ = n

$b_{k+1} . x$ = n

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

Maka algoritma P_2 lebih baik karena waktunya lebih kecil dari algoritma P

Teknik Pengumpulan

- **Semua jawaban ditulis di kertas dan dikumpulkan ke asisten praktikum pada akhir praktikum**

Penutup

- Ingat, berdasarkan Peraturan Rektor No 46 Tahun 2016 tentang Penyelenggaraan Pendidikan, mahasiswa wajib mengikuti praktikum 100%
- Apabila tidak hadir pada salah satu kegiatan praktikum segeralah minta tugas pengganti ke asisten praktikum
- Kurangnya kehadiran Anda di praktikum, memungkinkan nilai praktikum Anda tidak akan dimasukkan ke nilai mata kuliah.