LAPORAN PRAKTIKUM 3
Analysis Algorithm

Kompleksitas Waktu Asimptotik dari Algoritma

Disusun oleh :

Mohammad Dhikri
140810180075

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
2020

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:

$$T(n) = 2n^2 + 6n + 1$$

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

DEFINISI BIG-O NOTATION

Definisi 1. $T(n) = O(f(n))$ ***artinya*** $T(n)$ berorde paling besar $f(n)$ bila terdapat konstanta C dan n_o sedemikian sehingga

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

Untuk $n \geq n_o$

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

Dalam proses **pembuktian Big-O,** perlu dicari nilai n_o dan nilai C sedemikian sehingga terpenuhi kondisi $T(n) \leq C.f(n)$.

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: $T(n) = \mathbf{3}\ \boldsymbol{n^3} + 6n^2 + n + 8 = \boldsymbol{O(n^3)}$**,** 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 $T(n) = O(n^m)$

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

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:

$T(n) = \Omega(g(n))$ yang artinya $T(n)$berorde paling kecil $g(n)$ bila terdapat konstanta C dan $n_o$sedemikian sehingga

$$\boxed{T(n) \geq C.(g(n))}$$

untuk $n \geq n_o$

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 $T(n) = \Omega(g(n))$

Penentuan Big-Ω dan Big- (-) 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 $T(n) = n^m$

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 T(n) = 2 + 4 + 8 + 16 + …. + 2 ^n, tentukan nilai C, f(n), n0, dan notasi Big-O sedemikian sehingga T(n) = O (f(n)) jika T(n) =< C untuk semua n>=n0

T(n) = 2 + 4 + 8 + 16 + … + 2^n
T(n) = 2(2^n – 1)/(2-1)
T(n) = 2^(n+1) + 2

Untuk Big-O, maka f(n) = 2^n
Bukti bahwa T(n) = 2^(n+1) + 2 = O(2^n) adalah T(n) <= C. f(n)
= 2^(n+1) + 2 <= C.2^n, maka 2 < 2^n
= 2^(n+1) + 2 <= 2^(n+1) + 2^n = 2^(2n+1) untuk n >= 1
Maka bisa kita ambil C = 4 dan n0 = 1 untuk memperlihatkan
T(n) = 2^(2n+1) = O (2^n)

2. Buktikan bahwa untuk konstanta-konstanta positif p, q, dan r:
   $T(n) = pn^2 + qn + r$ adalah $O(n^2), \Omega(n^2), dan\ \Theta(n^2)$

Big - O

- Untuk n yang besar, pertumbuhan T(n) sebanding dengan n^2.
- Suku qn + r tidak berarti jika disbanding pn^2 dan boleh diabaikan sehingga T(n) = pn^2 + suku – suku lainnya
- Koefisien p pada pn^2 boleh diabaikan, sehingga T(n) = O(n^2) -> kompleksitas waktu asimtotik

Big - Ω

$T(n) >= C\ g(n)$ untuk $n>=n0$

Karena pn^2+qn+r >= pn^2 untuk n>=1 dengan c=p diperoleh pn^2+qn+r = Ω(n^2)

Big – Θ

Karena O(n^2) = Ω(n^2) maka Θ(n^2).

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
            w_ij ← w_ij or w_ik and w_kj
        endfor
    endfor
endfor
```

Untuk k=1

Untuk i = 1, jumlah perhitungan = n kali

Untuk k=2

Untuk i = 1, jumlah perhitungan = n kali

Untuk i = 2, jumlah perhitungan = n-1 kali

Untuk k=n

Untuk i = 1, jumlah perhitungan = n kali

Untuk i = 2, jumlah perhitungan = n-1 kali

…..

Untuk i = n, jumlah perhitungan = 1 kali

Jumlah perhitungan = n^2 + (n-1)^2 + (n-2)^2+ … +1^2

T(n) = n^2 + (n-1)^2 + (n-2)^2+ … +1^2

= n(n+1)(2n+1)/6

= $2n^3 + 3n^2 + 1$

$T(n) = O(n^3) = \Omega(n^3) = \Theta(n^3)$.

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?

Algoritma :

```
for i = 0 to n do
  for j = 0 to n do
    for k = 0 to n do
        C_ij = C_ij + C_ik * C_kj
    endfor
  endfor
endfor
```

Jumlah persamaan = jumlah penjumlahan = jumlah perkalian

Untuk i=1

Untuk j = 1, jumlah perhitungan = n kali

Untuk i=2

Untuk j = 1, jumlah perhitungan = n kali

Untuk j = 2, jumlah perhitungan = n-1 kali

Untuk i=n

Untuk j = 1, jumlah perhitungan = n kali

Untuk j = 2, jumlah perhitungan = n-1 kali

…..

Untuk j = n, jumlah perhitungan = 1 kali

Jumlah perhitungan = n^2 + (n-1)^2 + (n-2)^2+ … +1^2

T(n) = n^2 + (n-1)^2 + (n-2)^2+ … +1^2

= n(n+1)(2n+1)/6

= (2n^3 + 3n^2 + 1) * 3

= $6n^3 + 9n^2 + 3$

$T(n) = O(n^3) = \Omega(n^3) = \Theta(n^3)$.

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?

Algoritma :

```
j = 0
for i = 0 to n do  //loop sebanyak n kali
   Bj = Ai
   j = j + 1
endfor
```

Jumlah perbandingan : n kali + n kali = 2n kali

Jumlah penjumlahan : n kali

T(n) = 3n = O(n) -> Kompleksitas waktu asimtotik

Big-Ω = Big-O = Big-Θ

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!

Jumlah operasi perbandingan :
Perbandingan = (n – 1) + (n – 2) + (n – 3) + …+ 1 = n/2(n – 1)

b. Berapa kali maksimum pertukaran elemen-elemen tabel dilakukan?

Maksimal pertukaran elemen
Pertukaran = pertukaran terjadi ada

- Baris 5 -> n/2(n – 1)
- Baris 6-> n/2(n – 1)
- Baris 7 -> n/2(n – 1)

T_{max}(n) = 4(n)(n – 1)/2 = 2n^2 – 2n

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

Big – O :
2n^2 – 2n <= C.n^2
2 – 2/n <= C, n0 ! = 0 dan n0 ! = ½
C >= 0
Big-Ω
2n^2 – 2n <= C.n^2
n^2/2 – n/2 >= C. n^2
½ - 1/2n >= C no ! = 0 dan n0 != ½
C<= 0
Big-Θ
Karena 2n^2 – 2n = O(n^2) dan 2n^2 – 2n = Ω(n^2)
Maka 2n^2 – 2n + 1 = Θ(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(   )

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

   Jawab :

   Secara asimptotik algoritma yang tercepat adalah O(log N). pembuktiannya adalah inputan ke N ke ketiga algoritma. Kompleksitas O(log N) adalah log 8 = 0.903. Kompleksitas O (N log N) = 8 log 8 = 7.222222. Kompleksitas O(n^2) = 8^2 = 64. Jadi yang tercepat adalah O(log N).

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?

P1

Operasi penjumlahan = n kali -> loop for

Operasi perkalian = 1 + 2 + 3 + 4 + … + n = n/2(n+1)

Operasi penjumlahan + operasi perkalian = n + n/2(n+1) = O(n^2)

P2

Operasi BK = n kali -> loop for

Operasi total O(n)

Yang terbaik adalah algoritma adalah algoritma p2 karena kompleksitas waktunya lebih efektif yaitu O(n)