LAPORAN PRAKTIKUM 3
DESAIN DAN ANALISIS ALGORITMA

DISUSUN OLEH:

SURIADI VAJRAKARUNA 140810180038

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

I. Tujuan

1. Mahasiswa mengerti cara menghitung *worst case* dan dapat mengimplementasikannya.
2. Mahasiswa mengerti dan dapat menghitung *Big-O Notation.*
3. Mahasiswa mengerti dan dapat menghitung aturan kompleksitas waktu asimptotik.
4. Mahasiswa mengerti dan dapat menghitung *Big-Ω dan Big-Θ*.

II. Landasan Teori

1. Setelah mengetahui T(n) kita dapat menentukan kompleksitas waktu asimptotik yang dinyatakan dalam notasi Big-O, Big-Ω, Big-Θ, dan little-ω.
2. Dalam analisis algoritma kita selalu mengutamakan perhitungan worst case dengan alasan sebagai berikut:
   a. 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.*
   b. Untuk beberapa algoritma, *worst-case* cukup sering terjadi. Dalam beberapa aplikasi pencarian, pencarian info yang tidak ada mungkin sering dilakukan.
   c. 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 n.
3. Big-O Notation adalah cara untuk mengkonversi keseluruhan langkah-langkah suatu algoritma kedalam bentuk Aljabar, yaitu denganmenghiraukan konstanta yang lebih kecil dan koefisien yang tidakberdampak besar terhadap keseluruhan kompleksitas permasalahan yang diselesaikan oleh algoritma tersebut.
   a. *Worst-case* dihitung dengan *Big-O Notation.*
   b. $T(n) = O(f(n))$ artinya $T(n)$ berorde paling besar $f(n)$ bila terdapat konstanta C dan n_0 sehingga $T(n) \leq C.f(n)$, untuk $n \geq n_0$.
   c. Dalam pembuktian *Big-O Notation,* perlu dicari nilai n_0 dan C sehingga terpenuhi kondisi $T(n) \leq C.f(n)$.
4. Big-O Notation polinomial berderajat n digunakan untuk memperkirakan kompleksitas dengan mengabaikan suku berorde rendah.
   a. Teorema 1
      $T(n) = a_m n^m + a_{m-1} n^{m-1} + a_1 n + a_0$ adalah polinom berderajat m maka $T(n) = O(n^m)$
   b. Teorema 2
      Misalkan $T_1(n)=O(f(n))$ *dan* $T_2(n) = O(g(n))$, maka
      - $T_1(n)+T_2(n)=O(max(f(n), g(n))$ atau $T_1(n)+T_2(n) = O(f(n)+g(n))$

- $T_1(n).T_2(n)=O(f(n)).O(g(n))=O(f(n).g(n))$
- $O(c.f(n))=O(f(n))$, *c* adalah konstanta
- $f(n) =O(f(n))$

5. Aturan Menentukan Kompleksitas Waktu Asimptptik
   a. 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)
   b. 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)
6. 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.
7. $T(n)=\Omega(f(n))$, artinya $T(n)$ berorde paling kecil $f(n)$ bila terdapat konstanta C dan n sehingga $T(n) \geq C.(f(n))$, dengan syarat nilai c dan n positif.
8. $T(n)=\Theta(h(n))$, artinya $T(n)$ berorde sama dengan $h(n)$ Jika $T(n)=O(h(n))$ dan $T(n)=\Omega(g(n))$.

$$C_1f(n) \leq T(n) \leq C_2f(n), \text{ dengan syarat nilai } c \text{ dan } n \text{ positif}$$

III. Worksheet 2

1. Untuk $T(n) = 2 + 4 + 6 + 8 + 16 + \cdots + n^2$, tentukan nilai C, f(n), n_o, dan notasi Big-O sedemikian sehingga $T(n) = O(f(n))$ jika $T(n) \leq C\ untuk\ semua\ n \geq n_0$

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

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
```

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?

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?

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!
(b) Berapa kali maksimum pertukaran elemen-elemen tabel dilakukan?
(c) Hitung kompleksitas waktu asimptotik (Big-O, Big-Ω, dan Big-Θ) dari algoritma Bubble Sort tersebut!

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?

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?

① $T(n) = 2+4+6+8+\dots+n^2$

$$= \sum_{i=1}^{n^2} 2i = n^2(n^2+1)$$

$$= n^4 + n^2$$

Jika $n \geqslant 1$ maka $n^2 \leq n^4$ sehingga

$n^4 + n^2 \leq n^4 + n^4 = 2n^4, \quad n \geqslant 1$

$T(n) = 2+4+6+8+\dots+n^2 = O(n^4)$ dengan $C \geqslant 2$, $n_0 = 1$, $f(n) = 2n^4$

② $T(n) = pn^2 + qn + r$

$O \rightarrow pn^2 + qn + r \leq (p+q+r)n^2$, $n \geqslant 1$ dengan $C \geqslant (p+q+r)$ dan $n_0 = 1$,

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

$\Omega \rightarrow pn^2 + qn + r \geqslant pn^2$, $n \geqslant 0$ dengan $C \geqslant p$ dan $n_0 = 1$,

$T(n) = \Omega(n^2)$

$\theta \rightarrow O(n^2)$ dan $\Omega(n^2)$ maka $\theta(n^2)$

$T(n) = \theta(n^2)$

③ $T(n) = \sum_{k=1}^{n} \sum_{j=1}^{n} \sum_{i=1}^{n} 1 = n^3$

$T(n) = O(n^3) = \Omega(n^3) = \theta(n^3)$

④ for i ← n
  for j ← n
    c[i][j] = a[i][j] + b[i][j]

$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} 1 = n^2$

$T(n) = O(n^2) = \Omega(n^2) = \theta(n^2)$

⑤ for i ← n
  b[i] = a[i]

$T(n) = \sum_{i=1}^{n} 1 = n$

$T(n) = O(n) = \Omega(n) = \theta(n)$

⑥ a) $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = n(n-1)$

b) Pada worst case, operasi swap = operasi perbandingan = $n(n-1)$

c) $T(n) = n^2 - n$

$O \rightarrow n \geq 0,\ c \geq 1$ maka $n^2 - n \leq cn^2$

$\Omega \rightarrow n \geq 0,\ 0 < c \leq 1$ maka $n^2 - n \geq cn^2$

$\theta \rightarrow O(n^2)$ dan $\Omega(n^2)$ maka $T(n) = \theta(n^2)$

⑦ Tidak dapat ditentukan karena c tidak diketahui. Secara asimptotik, algoritma A dengan $T(n) = O(\log n)$ paling cepat.

⑧ Waktu asimptotik P_2

$T(n) = n + n$

$T(n) = \theta(n)$ dengan $C \geq 2,\ n_0 = 0$

Kompleksitas P

$P(x) = (\dots(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-2})x + \dots a_2)x + a_1)x + a_0$

$\left.\begin{array}{l} n \text{ pertambahan } \& \ n \text{ perkalian} \\ \\ n+n \text{ operasi} \end{array}\right\}$ kompleksitas algoritma sama

Daftar Pustaka

Ismail, Asep Maulana. 2018. *Penjelasan Sederhana tentang Time Complexity dan Big-O Notation*. Bandung: Universitas Widyatama.

Suryani, Mira, Ino Suryana, R. Sudrajat. 2019. *ANALISIS ALGORITMA: MODUL PRAKTIKUM 2.* Jatinangor: Universitas Padjadjaran.