LAPORAN PRAKTIKUM 2
KOMPLEKSITAS WAKTU 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

DISUSUN OLEH : AHMAD FAAIZ A (140810180023)

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
MARET 2019

Pendahuluan

Dalam memecahkan suatu masalah dengan komputer seringkali kita dihadapkan pada pilihan berikut:

1. Menggunakan algoritma yang waktu eksekusinya cepat dengan komputer standar
2. Menggunakan algoritma yang waktu eksekusinya tidak terlalu cepat dengan komputer yang cepat

Dikarenakan keterbatasan sumber daya, pola pemecahan masalah beralih ke pertimbangan menggunakan algoritma. Oleh karena itu diperlukan algoritma yang efektif dan efisien atau lebih tepatnya Algoritma yang mangkus.

Algoritma yang mangkus diukur dari berapa **jumlah waktu dan ruang (space) memori** yang dibutuhkan untuk menjalankannya. Algoritma yang mangkus ialah algoritma yang meminimumkan kebutuhan waktu dan ruang. Penentuan kemangkusan algoritma adakah dengan melakukan pengukuran kompleksitas algoritma.

Kompleksitas algoritma terdiri dari kompleksitas waktu dan ruang. Terminologi yang diperlukan dalam membahas kompleksitas waktu dan ruang adalah:

1. Ukuran input data untuk suatu algoritma, n.
   Contoh algoritma pengurutan elemen-elemen larik, n adalah jumlah elemen larik. Sedangkan dalam algoritma perkalian matriks n adalah ukuran matriks $n \times n$.
2. Kompleksitas waktu, $T(n)$, adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari input n.
3. Kompleksitas ruang, $S(n)$, adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari input n.

KOMPLEKSITAS WAKTU

Kompleksitas waktu sebuah algoritma dapat dihitung dengan langkah-langkah sebagai berikut:

1. Menetapkan ukuran input
2. Menghitung banyaknya operasi yang dilakukan oleh algoritma.
   Dalam sebuah algoritma terdapat banyak jenis operasi seperti operasi penjumlahan, pengurangan, perbandingan, pembagian, pembacaan, pemanggilan prosedur, dsb.

CONTOH

Algoritma Menghitung Nilai Rata-rata

```
procedure HitungRerata (input x1, x2, …, xn: integer, output r: real)
{   Menghitung nilai rata-rata dari sekumpulan elemen larik integer x1, x2, … xn.
    Nilai rata-rata akan disimpan di dalam variable r.
        Input: x1, x2, … xn
        Output: r (nilai rata-rata)
}
Deklarasi
        i : integer
        jumlah : real

Algoritma
        Jumlah ← 0
        i ← 1
        while i ≤ n do
            jumlah ← jumlah + ai
            i ← i + 1
        endwhile
        {i > n}
        r ← jumlah/n      {nilai rata-rata}
```

Menghitung Kompleksitas Waktu dari Algoritma Menghitung Nilai Rata-rata

Jenis-jenis operasi yang terdapat di dalam Algoritma HitungRerata adalah:

- Operasi pengisian nilai/*assignment* (dengan operator "←")
- Operasi penjumlahan (dengan operator "+")
- Operasi pembagian (dengan operator "/")

Cara menghitung kompleksitas waktu dari algoritma tersebut adalah sengan cra menghitung masing-masing jumlah operasi. Jika operasi tersebut berada di sebuah loop, maka jumlah operasinya bergantung berapa kali loop tersebut diulangi.

(i) Operasi pengisian nilai (*assignment*)

jumlah ← 0,	1 kali
k ← 1,	1 kali
jumlah ← jumlah + a_k	n kali
k ← k+1,	n kali
r ← jumlah/n,	1 kali

Jumlah seluruh operasi pengisian nilai (*assignment*) adalah

$t_1 = 1 + 1 + n + n + 1 = 3 + 2n$

(ii) Operasi penjumlahan

Jumlah + a_k,	n kali
k+1,	n kali

Jumlah seluruh operasi penjumlahan adalah

$t_2 = n + n = 2n$

(iii) Operasi pembagian

Jumlah seluruh operasi pembagian adalah

Jumlah/n	1 kali

Dengan demikian, kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi aritmatika dan operasi pengisian nilai adalah:

$$T(n) = t_1 + t_2 + t_3 = 3 + 2n + 2n + 1 = 4n + 4$$

Studi Kasus 1: Pencarian Nilai Maksimal

Buatlah programnya dan hitunglah kompleksitas waktu dari algoritma berikut:

Algoritma Pencarian Nilai Maksimal

```
procedure CariMaks(input x1, x2, …, xn: integer, output maks: integer)
{   Mencari elemen terbesar dari sekumpulan elemen larik integer x1, x2, …, xn. Elemen terbesar akan
    disimpan di dalam maks
    Input: x1, x2, …, xn
    Output: maks (nilai terbesar)
}

Deklarasi
        i : integer

Algoritma
        maks ← x1
        i ← 2
        while i ≤ n do
            if xi > maks then
                maks ← xi
            endif
            i ← i + 1
        endwhile
```

Jawaban studi kasus 1

Jenis-jenis operasi yang terdapat di dalam Algoritma Pencarian Nilai Maksimal adalah:

- Operasi pengisian nilai/*assignment* (dengan operator "$\leftarrow$")
- Operasi penjumlahan (dengan operator "+")
- Operasi perbandingan (dengan operator ">")

1. Operasi pengisian nilai (*assignment*)

maks $\leftarrow x_1$,	1 kali
i $\leftarrow$ 2,	1 kali
maks $\leftarrow x_i$	n - 1 kali
i $\leftarrow$ i + 1	n - 1 kali

Jumlah seluruh operasi pengisian nilai (*assignment*) adalah
$t_1 = 1 + 1 + (n\text{-}1) + (n\text{-}1) = n$

2. Operasi penjumlahan

i $\leftarrow$ i + 1,	n - 1 kali

Jumlah seluruh operasi penjumlahan adalah
$t_2 = n - 1$

3. Operasi perbandingan

x_i > maks,	n - 1 kali

Jumlah seluruh operasi penjumlahan adalah
$t_2 = n - 1$

Dengan demikian, kompleksitas waktu algoritma adalah:

$$T(n) = t_1 + t_2 + t_3 = n + (n\text{-}1) + (n\text{-}1) = 3n - 2$$

PEMBAGIAN KOMPLEKSITAS WAKTU

Hal lain yang harus diperhatikan dalam menghitung kompleksitas waktu suatu algoritma adalah parameter yang mencirikan ukuran input. Contoh pada algoritma pencarian, waktu yang dibutuhkan untuk melakukan pencarian tidak hanya bergantung pada ukuran larik (n) saja, tetapi juga bergantung pada nilai elemen (x) yang dicari.

Misalkan:

1. Terdapat sebuah larik dengan panjang elemen 130 dimulai dari $y_1, y_2, \ldots y_n$
2. Asumsikan elemen-elemen larik sudah terurut. Jika $y_1 = x$, maka waktu pencariannya lebih cepat 130 kali dari pada $y_{130} = x$ atau x tidak ada di dalam larik.
3. Demikian pula, jika $y_{65} = x$, maka waktu pencariannya ½ kali lebih cepat daripada $y_{130} = x$

Oleh karena itu, kompleksitas waktu dibedakan menjadi 3 macam:

(1) $T_{NIN}(n)$: kompleksitas waktu untuk kasus terbaik (***best case***) merupakan kebutuhan waktu minimum yang diperlukan algoritma sebagai fungsi dari n.

(2) $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata (***average case***) merupakan kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari n. Biasanya pada kasus ini dibuat asumsi bahwa semua barisan input bersifat sama. Contoh pada kasus *searching* diandaikan data yang dicari mempunyai peluang yang sama untuk tertarik dari larik.

(3) $T_{NAS}(n)$: kompleksitas waktu untuk kasus terburuk (***worst case***) merupakan kebutuhan waktu maksimum yang diperlukan algoritma sebagai fungsi dari n.

Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan x_1, x_2, ... x_n yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian beruntun (*sequential search*). Algoritma *sequential search* berikut menghasilkan indeks elemen yang bernilai sama dengan y. Jika y tidak ditemukan, indeks 0 akan dihasilkan.

```
procedure SequentialSearch(input x1, x2, ... xn : integer, y : integer, output idx : integer)
{   Mencari y di dalam elemen x1, x2, ...  xn. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
    Jika y tidak ditemukan, makai idx diisi dengan 0.
    Input: x1, x2, ... xn
    Output: idx
}
```

```
Deklarasi
        i : integer
        found : boolean {bernilai true jika y ditemukan atau false jika y tidak ditemukan}
Algoritma
        i ← 1
        found ← false
        while (i ≤ n) and (not found) do
            if xi = y then
                found ← true
            else
                i ← i + 1
            endif
        endwhile
        {i < n or found}

        If found then {y ditemukan}
                idx ← i
        else
                idx ← 0 {y tidak ditemukan}
        endif
```

Jawaban Studi Kasus 2

1. **Best Case**: terjadi bila (x1 = y),
   Operasi perbandingan (xi = y)hanya dilakukan 1 kali, maka
   Tmin (n) = 1

2. **Worst Case**: bila (xn = y) atau y tidak ditemukan
   Seluruh elemen larik dibandingkan, maka jumlah perbandingannya:
   Tmax (n) = n

3. **Average Case**:
   asumsikan peluang y terdapat disembarang lokasi adalah sama, peluang elemen ke–j= y adalah 1/n atau P(xj= y) = 1/n. Jika xj= y maka Tj yang dibutuhkan adalah Tj = j, maka jumlah perbandingan elemen lariknya:

$$T_{avg}(n) = \sum_{j=1}^{n} T_j P(x[j] = y) = \sum_{j=1}^{n} T_j \frac{1}{n} = \frac{1}{n}\sum_{j=1}^{n} T_j = \frac{1}{n}\sum_{j=1}^{n} j = \frac{1}{n}\left(\frac{n(n+1)}{2}\right) = \frac{n+1}{2}$$

Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan $x_1, x_2, \ldots x_n$ yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian bagi dua (*binary search*). Algoritma *binary search* berikut menghasilkan indeks elemen yang bernilai sama dengan y. Jika y tidak ditemukan, indeks 0 akan dihasilkan.

```
procedure BinarySearch(input x1, x2, ... xn : integer, x : integer, output : idx : integer)
{   Mencari y di dalam elemen x1, x2, ... xn. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
    Jika y tidak ditemukan makai dx diisi dengan 0.
    Input: x1, x2, ... xn
    Output: idx
}
Deklarasi
        i, j, mid : integer found : Boolean
Algoritma
        i ← 1
        j ← n
        found ← false
        while (not found) and ( i ≤ j)
                do mid ← (i + j) div 2
                if xmid = y then
                    found ← true
                else
                   if xmid < y then
                        i ← mid + 1
                   else
                        j ← mid – 1 endif
                endif
        endwhile
        {found or i > j }

        If found then
                Idx ← mid
        else
                Idx ← 0
        endif
```

Jawaban Studi Kasus 3

1. **Best Case** (T min(n))
   Terjadi bila x ditemukan pada elemen pertengahan (amid) , dan operasi perbandingan elemen (amid = x) yang dilakukan hanya satu kali, sehingga :
   T min(n) = 1

2. **Worst Case** (T max(n))
   Terjadi ketika elemen x ditemukan ketika ukuran larik = 1. Pada kasus terburuk ini ukuran larik setiap kali memasuki while-do adalah n, n/2, n/4, n/8, . . ., 1 (sebanyak 2 log n kali). Sehingga :
   $$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow k = log_2\, n$$
   T max(n) = 2log n = log n

3. **Average Case** T avg(n)
   Kompleksitas waktu rata-rata T avg(n) algoritma Binary Search adalah
   T avg(n) = 2log n = log n

Studi Kasus 4: Insertion Sort

1. Buatlah program insertion sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma insertion sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

```
procedure InsertionSort(input/output x1, x2, ... xn : integer)
{   Mengurutkan elemen-elemen x1, x2, ... xn dengan metode insertion  sort.
    Input: x1, x2, ... xn
    OutputL x1, x2, ... xn (sudah terurut menaik)
}
Deklarasi
         i, j, insert : integer
Algoritma
         for i ← 2 to n do
              insert ← xi
              j ← i
              while (j < i) and (x[j-i] > insert) do
                  x[j]← x[j-1]
                  j←j-1
              endwhile
              x[j] = insert
         endfor
```

Jawaban Studi Kasus 4

Jumlah operasi perbandingan dan operasi pertukaran

1. Operasi perbandingan elemen larik
   Jumlah seluruh operasi perbandingan elemen-elemen larik adalah

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

   Ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma Urut tidak bergantung pada batasan apakah data masukannya sudah terurut atau acak.

2. Operasi pertukaran
   Jumlah seluruh operasi pertukaran sama dengan jumlah operasi perbandingan elemen larik, karena berada dalam satu while-loop yang sama. Sehingga:

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

Kompleksitas Algoritma

1. **Best Case** (T min(n))
   Terjadi ketika elemen - elemen pada larik sudah terurut menaik (dari terkecil ke terbesar), maka hanya akan terjadi operasi perbandingan sejumlah n elemen, sehingga :
   Tmin(n) = n

2. **Worst Case** (T max(n))
   Terjadi ketika elemen – elemen pada larik terurut menurun (dari terbesar ke terkecil), maka dibutuhkan
   T max(n) = n^2

3. **Average Case** T avg(n)
   Kompleksitas waktu rata-rata T avg(n) algoritma Insertion Sort adalah:
   T avg(n) = n^2

Studi Kasus 5: Selection Sort

1. Buatlah program selection sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma selection sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

```
procedure SelectionSort(input/output x1, x2, … xn : integer)
{ Mengurutkan elemen-elemen x1, x2, … xn dengan metode selection sort.
   Input: x1, x2, … xn
   OutputL x1, x2, … xn (sudah terurut menaik)
}
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 xj > ximaks then
                 imaks ← j
               endf
             endfor
             {pertukarkan ximaks dengan xi}
             temp ← xi
             xi ← ximaks
             ximaks ← temp
        endfor
```

Jawaban Studi Kasus 5

Jumlah operasi perbandingan dan operasi pertukaran

1. Operasi perbandingan elemen larik

   Jumlah seluruh operasi perbandingan elemen-elemen larik adalah

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

   Ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma Urut tidak bergantung pada batasan apakah data masukannya sudah terurut atau acak.

2. Operasi pertukaran

   Untuk setiap i dari 1 sampai n – 1, terjadi satu kali pertukaran elemen, sehingga jumlah operasi pertukaran seluruhnya adalah T(n) = n – 1.

   Jadi, algoritma pengurutan maksimum membutuhkan n(n – 1)/2 buah operasi perbandingan elemen dan n – 1 buah operasi pertukaran.

Kompleksitas Algoritma

1. **Best Case** (T min(n))
   Terjadi ketika elemen - elemen pada larik sudah terurut menaik (dari terkecil ke terbesar), maka akan terjadi operasi perbandingan sejumlah n^2 elemen, sehingga :
   $Tmin(n) = n^2$

2. **Worst Case** (T max(n))
   Terjadi ketika elemen – elemen pada larik terurut menurun (dari terbesar ke terkecil), maka dibutuhkan
   $T\ max(n) = n^2$

3. **Average Case** T avg(n)
   Kompleksitas waktu rata-rata T avg(n) algoritma Insertion Sort adalah:
   $T\ avg(n) = n^2$