TUGAS MODUL PRAKTIKUM 1

Disusun oleh :

Anne Audistya Fernanda

140810180059

Kelas A

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2020

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 nxn.
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 dengan cara 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

T(n) = 2(n – 2) + (n – 2) + 2

= 3 n - 4

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:

- Terdapat sebuah larik dengan panjang elemen 130 dimulai dari $y_1, y_2, \ldots y_n$
- 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.
- 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) $T_{min}(n)$: 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) $T_{max}(n)$: 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 $x_1, x_2, \ldots x_n$ yang <u>telah terurut</u> 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.

<u>procedure</u> SequentialSearch(<u>input</u> $x_1, x_2, \ldots x_n$: <u>integer</u>, y : <u>integer</u>, <u>output</u> idx : <u>integer</u>)

{ Mencari y di dalam elemen $x_1, x_2, \ldots x_n$. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
Jika y tidak ditemukan, makai idx diisi dengan 0.
Input: $x_1, x_2, \ldots x_n$
Output: idx
}

Deklarasi

i : <u>integer</u>
found : <u>boolean</u> {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. Kasus terbaik: ini terjadi bila a1 = x.
   $T_{min}(n) = 1$
2. Kasus terburuk: bila $a_n = x$ atau x tidak ditemukan.
   $T_{max}(n) = n$
3. *Kasus rata-rata*: Jika *x* ditemukan pada posisi ke-*j*, maka operasi perbandingan ($a_k = x$) akan dieksekusi sebanyak *j* kali.
   $T_{avg}(n) = (1+2+3+..+n)/n = (1/2n(1+n))/n = (n+1)/2$

Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan $x_1, x_2, \dots x_n$ yang <u>telah terurut</u> 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.

<u>procedure</u> BinarySearch(<u>input</u> $x_1, x_2, \dots x_n$: <u>integer</u>, x : <u>integer</u>, <u>output</u> : idx : <u>integer</u>)
{ Mencari y di dalam elemen $x_1, x_2, \dots x_n$. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
Jika y tidak ditemukan makai dx diisi dengan 0.
Input: $x_1, x_2, \dots x_n$
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 x_mid = y then
        found ← true
    else
        if x_mid < y then     {mencari di bagian kanan}
            i ← mid + 1
        else                  {mencari di bagian kiri}

            j ← mid – 1
        endif
    endif
endwhile
{found or i > j }

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

Jawaban Studi Kasus 3

1. Kasus terbaik

   $T_{min}(n) = 1$
2. Kasus terburuk

   $T_{max}(n) = {}^2\log n$

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.

<u>procedure</u> InsertionSort(<u>input/output</u> $x_1, x_2, \ldots x_n$: <u>integer</u>)

{ Mengurutkan elemen-elemen $x_1, x_2, \ldots x_n$ dengan metode insertion sort.
Input: $x_1, x_2, \ldots x_n$
OutputL $x_1, x_2, \ldots x_n$ (sudah terurut menaik)
}

Deklarasi

i, j, insert : <u>integer</u>

Algoritma

```
for i ← 2 to n do
    insert ← x_i
    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
Loop sementara dijalankan hanya jika i> j dan arr [i] <arr [j]. Jumlah total iterasi loop sementara (Untuk semua nilai i) sama dengan jumlah inversi.
Kompleksitas waktu keseluruhan dari jenis penyisipan adalah O (n + f (n)) di mana f (n) adalah jumlah inversi.Jika jumlah inversi adalah O(n), maka kompleksitas waktu dari jenis penyisipan adalah O(n).

Dalam kasus terburuk, bisa ada inversi n * (n-1) / 2. Kasus terburuk terjadi ketika array diurutkan dalam urutan terbalik. Jadi kompleksitas waktu kasus terburuk dari jenis penyisipan adalah O (n2).

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 $x_1, x_2, \dots x_n$: integer)
{ Mengurutkan elemen-elemen $x_1, x_2, \dots x_n$ dengan metode selection sort.
Input: $x_1, x_2, \dots x_n$
OutputL $x_1, x_2, \dots x_n$ (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 $x_j > x_{imaks}$ then
imaks ← j
endif
endfor
{pertukarkan x_{imaks} dengan x_i}
temp ← x_i
x_i ← x_{imaks}
x_{imaks} ← temp
endfor

Jawaban Studi Kasus 5

a. Jumlah operasi perbandingan element. Untuk setiap *pass* ke-i,

$i = 1$ –> jumlah perbandingan $= n - 1$

$i = 2$ –> jumlah perbandingan $= n - 2$

$i = 3$ –> jumlah perbandingan $= n - 3$

:

$i = k$ –> jumlah perbandingan $= n - k$

:

$i = n - 1$ –> jumlah perbandingan $= 1$

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah $T(n) = (n - 1) + (n - 2) + \ldots + 1$

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

b. Jumlah 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.

Teknik Pengumpulan

- Lakukan push ke github/gitlab untuk semua program dan laporan hasil analisa yang berisi jawaban dari pertanyaan-pertanyaan yang diajukan. Silahkan sepakati dengan asisten 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.