LAPORAN PRAKTIKUM
ANALISIS ALGORITMA

Disusun oleh:

Fadlan Mulya Priatna
140810180041
Kelas A

Program Studi S-1 Teknik Informatika
Departemen Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Padjadjaran

A. Materi

Algoritma yang mangkus/efisien tergantung dari ***kompleksitas algoritma*** *Diukur dari berapa* ***jumlah waktu dan ruang (space)*** *Sequential Search atau Binary Search Algoritma mana yang lebih cepat?*

Kompleksitas bisa dihitung dengan langkah:

1. ***Kompleksitas waktu, T(n)***

   Jumlah ***operasi*** *yang dilakukan untuk melaksanakan algoritma*

2. ***Kompleksitas ruang, S(n)***

   Jumlah ***ruang memori*** *yang dibutuhkan algoritma*

Bingung? Lihat contoh langsung aja ya

Dapat dihitung dengan:

1. *Menetapkan ukuran input (n)*
2. *Menghitung banyaknya operasi*
   - *Penjumlahan*
   - *Pengurangan*
   - *Perbandingan*
   - *Pembagian*
   - *Pembacaan*
   - *Pemanggilan prosedur*
   - *dsb*

B. Contoh

1) jumlah <- 0
2) i <- 1
3) while i $\leq$ n do
4)   jumlah <- jumlah + a_i
5)   i <- i + 1
6) Endwhile
7) r <- jumlah/n

Kira-kira ada operasi apa saja dalam kodingan di atas?

Jenis operasi yang bisa dihitung:

- *Operasi assignment (<-)*
- *Operasi penjumlahan (+)*
- *Operasi pembagian (/)*

Jika operasi ada di dalam loop, maka jumlah operasi bergantung berapa kali loop tersebut diulangi (n)

Operator Assignment:

1) jumlah <- 0
2) i <- 1
3) while i ≤ n do
4) jumlah <- jumlah + a_i
5) i <- i + 1
6) Endwhile
7) r <- jumlah/n

Baris 1) 1 kali

Baris 2) 1 kali

Baris 4) n kali

Baris 5) n kali

Baris 7) 1 kali

t_1 = 1 + 1 + n + n + 1 = **3 + 2n**

Operator Pertambahan:

1) jumlah <- 0
2) i <- 1
3) while i ≤ n do
4) jumlah <- jumlah + a_i
5) i <- i + 1
6) Endwhile
7) r <- jumlah/n

Baris 4) n kali

Baris 5) n kali

t_2 = n + n = **2n**

Operator Pembagian:

1) jumlah <- 0
2) i <- 1
3) while i ≤ n do
4) jumlah <- jumlah + a_i

5) i <- $i + 1$

6) Endwhile

7) r <- jumlah/n

Baris 7) 1 kali

t_3 = **1**

Kompleksitas:

1) jumlah <- 0

2) i <- 1

3) while $i \leq n$ do

4) jumlah <- jumlah + a_i

5) i <- i + 1

6) Endwhile

7) r <- jumlah/n

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

$t_2 = n + n =$ **2n**

t_3 = **1**

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

T(n) = 4n + 4

Macam-macam kompleksitas waktu

1. *Best case* = $T_{min}(n)$

   Contoh: Sequential search yang (x_i = found) dimana i = 1

2. *Average case* = $T_{avg}(n)$

   Contoh: Searching dengan data yang dicari berpeluang sama untuk dicari = ***(n+1)/2***

3. *Worst case* = $T_{max}(n)$

   Contoh: Sequential search yang (x_i = found) dimana i = array.length atau x_i tidak ditemukan

C. Bagian Analisis di Modul Praktikum

1. 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: T(n) = 2(n – 2) + (n – 2) + 2   = 3 n - 4

2. 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 ratarata 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 x_1, x_2, …, x_n : integer, y : integer, output idx : integer){ Mencari di dalam elemen x_1, x_2, …, x_n. Lokasi (indeks elemen) tempat ditemukan diisi ke dalam idx. Jika tidak ditemukan, makai idx diisi dengan 0. Input: x_1, x_2, …, x_n 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:

Jumlah operasi perbandingan elemen tabel:

1. Kasus terbaik: ini terjadi bila $a_1 = x$

$$T_{min}(n) = 1$$

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.

$$T_{avg}(n) = (1+2+3+..+n)/n = (1/2n(1+n))/n = (n+1)/2$$

3. Studi Kasus 3: Binary 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 ratarata 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 {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:

1. Kasus terbaik : Tmin(n) = 1
2. Kasus terburuk : Tmax (n) = 2log n

4. Studi Kasus 4: Insertion Sort
    a. Buatlah program insertion sort dengan menggunakan bahasa C++
    b. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma insertion sort.
    c. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

```
procedure InsertionSort(input/output , , … : integer) {
Mengurutkan elemen-elemen , , … dengan metode insertion sort.
Input: , , …
OutputL , ,… (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
```

Jawaba:

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).

5. Studi Kasus 5: Selection Sort
    a. Buatlah program selection sort dengan menggunakan bahasa C++
    b. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma selection sort.
    c. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

```
procedure SelectionSort(input/output ,     ,… : integer) {
Mengurutkan elemen-elemen ,     , … dengan metode selection sort.
Input: ,     , …
OutputL    ,     ,…  (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
            endif
        endfor  {pertukarkan ximaks dengan xi}
        temp ← xi
```

xi ← ximaks ximaks ← temp endfor
Jawaban: 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) + … + 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.