LAPORAN PRAKTIKUM 2
ANALISIS ALGORITMA

Disusun oleh :
Alfian Fadhil Labib
140810180055

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

<u>procedure</u> HitungRerata (input x_1, x_2, …, x_n: integer, output r: real)
{ Menghitung nilai rata-rata dari sekumpulan elemen larik integer x_1, x_2, … x_n.
  Nilai rata-rata akan disimpan di dalam variable r.
    Input: x_1, x_2, … x_n
    Output: r (nilai rata-rata)
}

Deklarasi

  i : <u>integer</u>
  jumlah : <u>real</u>

Algoritma

  Jumlah ← 0
  i ← 1
  <u>while</u> i ≤ n do
    jumlah ← jumlah + a_i
    i ← i + 1
  <u>endwhile</u>
  {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 Hitung Rerata 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
```

Jawab:

Kompleksitas waktu:

$T(n) = 2(n – 2) + (n – 2) + 2$

$\quad = 3\,n - 4$

Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan $x_1, x_2, \dots\ x_n$ 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
```

Jawab:

Kompleksitas waktu:

- Best Case : ini terjadi bila $x_1 = z$

$T_{min}(n) = 1$

- Worst Case : bila $x_n = z$ atau z tidak ditemukan.

$T_{max}(n) = n$

- Average Case : Jika *z* ditemukan pada posisi ke-*j*, maka operasi perbandingan ($a_k = z$) 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 $x_1, x_2, \ldots\ x_n$ 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 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
```

Jawab:

Kompleksitas waktu:

- Best case: Jika ditemukan pada indeks di tengah yaitu $T_{min}(n) = 1$
- Average case: Jika ditemukan pada indeks di awal atau di akhir
- Worst case: Jika tidak ditemukan sama sekali yaitu $T_{max}(n) = {}^2\log n$

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.

```
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
```

Jawab:

Kompleksitas waktu:

- Best case : Jika array sudah terurut .
- Average case : Jika sebagian elemen array sudah terurut.
- Worst case: Jika array harus diurutkan sebanyak n kali, Dalam kasus terburuk, bisa ada inversi n * (n-1) / 2. Kasus terburuk terjadi ketika array dalam urutan terbalik. Jadi

kompleksitas waktu kasus terburuk 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 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
              endif
            endfor
            {pertukarkan ximaks dengan xi}
            temp ← xi
            xi ← ximaks
            ximaks ← temp
        endfor
```

Jawab:

Kompleksitas waktu:

(i) Jumlah operasi perbandingan elemen

Untuk setiap loop ke-i,
i = 1 → jumlah perbandingan = n-1
i = 2 → jumlah perbandingan = n-2
i = k → jumlah perbandingan = n-k
i = n-1 → jumlah perbandingan = 1
sehingga T(n) = (n-1) + (n-2) + … + 1 = n(n-1)/2 dimana kompleksitas waktu ini berlaku menjadi yang terbaik, rata-rata maupun yang terburuk karena algoritma ini tidak melihat apakah arraynya sudah urut atau tidak terlebih dahulu.

(ii) Jumlah operasi pertukaran

Untuk setiap loop ke-1 sampai n-1 terjadi satu kali pertukaran elemen sehingga T(n) = n-1.