LAPORAN PRAKTIKUM 2
ANALISIS ALGORITMA

NAMA	:	Nadhifal Abdurrahman Rendusara
NPM	:	140810180048
KELAS	:	B

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN

1. Landasan Teori

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 x 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.

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:

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

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.

2. Tugas

1. Worksheet 2

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

A. Operasi Assignment
    maks <- x1 1 kali
    i <- 2 1 kali
    maks <- xi n kali
    i <- i + 1 n kali

    t1 = 1 + 1 + n + n = 2+2n (ii)

B. Operasi Perbandingan
    if xi > maks then n kali

    t2 = n

C. Operasi Penjumlahan
    i + 1 n kali

    t3 = n

    Jadi, T(n) = t1 + t2 + t3 = 2 + 2n + n + n = 2 + 4n

Studi Kasus 2: Sequential Search

Diberikan larik bilangan bulat x1, x2, … xn 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 ditemukan diisi ke dalam idx. Jika 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

A. Terbaik (Tmin(n))
   Operasi Assignment = 4
   Operasi Perbandingan = 2

   Tmin(n) = 4 + 2 = 6 = Ω(n)

B. Terburuk (Tmax(n))
   Operasi Assignment = 3 + n
   Operasi Perbandingan = 1 + n
   Operasi Penambahan = n

   Tmax(n) = 3 + n + 1 + n + n = 4 + 3n = O(n)

C. Rata-rata (Tavg(n))
   Tavg(n) = (T(min) + T(max))/2 = (6 + 4 + 3n)/2 = (10 + 3n)/2 = **Θ**(n)

Studi Kasus 3: Binary Search

Diberikan larik bilangan bulat x1, x2, … xn 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   {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

A. Terbaik (Tmin(n))
   Operasi Assignment = 6
   Operasi Perbandingan = 2

   Tmin(n) = 6 + 2 = 8 = Ω(n)

B. Terburuk (Tmax(n))
   Karena ini adalah binary search, maka:
   Iterasi 1 = n
   Iterasi 2 = n/2
   Iterasi 3 = $n/2^2$
   Iterasi x = $n/2^x$
   Hingga array menjadi 1

   Maka:
   n/2^k = 1
   n = 2^k
   log2(n) = log2(2^k)
   log2(n) = k log2(2)
   k = log2(n)

   sehingga:
   Tmax(n) = O(log2(n))

C. Rata-rata (Tavg(n))
   Tavg(n) = (T(min) + T(max))/2 = (8 + log2(n))/2 = **Θ**(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
Output: 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:

1. Operasi Perbandingan = 2*((n-1) + (n-1)) = 2*(2n-2) = 4n - 4
2. Operasi Pertukaran = (n-1) * n = (n^2)-n

A. Terbaik (Tmin(n))
   Tmin(n) = 4n - 4 + 1 = 4n - 3 = Ω(n)

B. Terburuk (Tmax(n))
   Tmax(n) = 4n - 4 + (n^2) - n = (n^2) + 3n - 4 = O(n^2)

C. Rata-rata (Tavg(n))
   Tavg(n) = (Tmin(n) + Tmax(n)) / 2 = (n + n^2) / 2 = **Θ**(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 insertion sort.
Input: x1, x2, ... xn
Output: x1, x2, ... xn(sudah terurut menaik)
}
```

Deklarasi

```
i, j, imaks, temp : integer
```

Algoritma

```
for i <- n down to 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
```

Jawaban Studi Kasus 5

Jumlah Operasi:

1. Operasi Perbandingan =

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

2. Operasi Pertukaran = (n-1)

A. Terbaik (Tmin(n))
   Tmin(n) = ½ (n^2 - n) + 1 = Ω(n^2)

B. Terburuk (Tmax(n))
   Tmax(n) = ½ (n^2 - n) + (n-1) = O(n^2)

C. Rata-rata (Tavg(n))
   Tavg(n) = (Tmin(n) + Tmax(n)) / 2 = (n^2 + n^2) / 2 = **Θ**(n^2)

2. Program C++

Studi Kasus 1

```cpp
/*
Nama    = Nadhifal A. Rendusara
NPM     = 140810180048
Kelas   = B
*/
#include <iostream>
using namespace std;

int main()
{
    int i=0;
    int n=10;
    int x[n]{1,20,12,41,10,32,41,22,92,70};
    int maks = x[0];

    do{
        if(x[i]>maks){
            maks=x[i];
        }
        i++;
    }while(i<n);

    cout<<"Maks : "<<maks;

    return 0;
}
```

Studi Kasus 2

```cpp
/*
Nama    = Nadhifal A. Rendusara
NPM     = 140810180048
Kelas   = B
*/
#include <iostream>
using namespace std;

int main()
{
    int i = 0;
    bool found = false;
    int n=10;
    int x[n]{10,9,8,7,6,5,4,3,2,1};
    int y;
    cout<<"enter number : ";
    cin>>y;

    do{
        if(x[i]==y){
            found=true;
            cout<<"founded in array number "<<i;
        }
        else
            i++;
    }while((i<n) && (not found));

    return 0;
}
```

Studi Kasus 3

```cpp
/*
Nama    = Nadhifal A. Rendusara
NPM     = 140810180048
Kelas   = B
*/
#include <iostream>
using namespace std;

int main(){
    int i,j,y,mid;
    bool found;

    int n=10;
    int x[n]{1,19,27,36,45,54,63,72,81,90};
    i=0;
    j=n;
    found=false;

    cout<<"enter number : ";
    cin>>y;

    do{
        mid=(i+j)/2;

        if(x[mid]==y){
            found=true;
            cout<<"founded in array number "<<mid;
        }
        else{
            if(x[mid]<y){
                i=mid+1;
            }
            else{
```

```
                j=mid-1;
            }
        }
    }while((i<=j) && (not found));

    return 0;
}
```

Studi Kasus 4

```
/*
Nama    = Nadhifal A. Rendusara
NPM     = 140810180048
Kelas   = B
*/
#include <iostream>
using namespace std;

int main(){
    int i,j,ins,n=10,x[n]{1,20,12,41,10,32,41,22,92,70};

    for(i=1;i<n;i++){
        ins=x[i];
        j=i;

        while((j>0)&&(x[j-1]>ins)){
            x[j]=x[j-1];
            j=j-1;
        };
        x[j]=ins;
    }

    for(i=0;i<n;i++){
        cout<<x[i]<<endl;
    }

    return 0;
}
```

Studi Kasus 5

```
/*
Nama    = Nadhifal A. Rendusara
NPM     = 140810180048
Kelas   = B
*/
#include <iostream>
using namespace std;

int main(){
    int i,j,temp,n=10,x[n]{1,20,12,41,10,32,41,22,92,70};

    for(i=0;i<n;i++){
        for(j=i;j<n;j++){
            if(x[j]<x[i]){
                temp=x[i];
                x[i]=x[j];
                x[j]=temp;
            }
        }
    }

    for(i=0;i<n;i++){
        cout<<x[i]<<endl;
```

```
    }

    return 0;
}
```