

**LAPORAN PRAKTIKUM**  
**ANALISIS ALGORITMA**



**DISUSUN OLEH**

Gede Bagus Darmagita - 140810180068

**PROGRAM STUDI 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  $n \times n$ .
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  $x_1, x_2, \dots, x_n$ : integer, output  $r$ : real)
{ Menghitung nilai rata-rata dari sekumpulan elemen larik integer  $x_1, x_2, \dots, x_n$ .
  Nilai rata-rata akan disimpan di dalam variable  $r$ .
  Input:  $x_1, x_2, \dots, x_n$ 
  Output:  $r$  (nilai rata-rata)
}

Deklarasi
   $i$ : integer
  jumlah: real

Algoritma
  Jumlah  $\leftarrow 0$ 
   $i \leftarrow 1$ 
  while  $i \leq n$  do
    jumlah  $\leftarrow$  jumlah +  $a_i$ 
     $i \leftarrow i + 1$ 
  endwhile
  { $i > n$ }
   $r \leftarrow$  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 " $\leftarrow$ ")
- 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 $\leftarrow$ 0,	1 kali
k $\leftarrow$ 1,	1 kali
jumlah $\leftarrow$ jumlah + $a_k$	n kali
k $\leftarrow$ k+1,	n kali
r $\leftarrow$ 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  $x_1, x_2, \dots, x_n$ : integer, output maks: integer)  
{ Mencari elemen terbesar dari sekumpulan elemen larik integer  $x_1, x_2, \dots, x_n$ . Elemen terbesar akan  
  disimpan di dalam maks  
  Input:  $x_1, x_2, \dots, x_n$   
  Output: maks (nilai terbesar)  
}
```

#### Deklarasi

$i$  : integer

#### Algoritma

```
maks  $\leftarrow$   $x_1$   
 $i \leftarrow 2$   
while  $i \leq n$  do  
  if  $x_i >$  maks then  
    maks  $\leftarrow x_i$   
  endif  
   $i \leftarrow i + 1$   
endwhile
```

Answer:

1. Operasi Assignment =  $1 + 1 + (n-1) + (n-1) = 2n$

2. Operasi Perbandingan =  $n-1$

3. Operasi Penjumlahan =  $n-1$

Maka  $T_{\max}(n) = 4n-2$

**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:

- Terdapat sebuah larik dengan panjang elemen 130 dimulai dari  $y_1, y_2, \dots, y_n$
- 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  $\frac{1}{2}$  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 bulat  $x_1, x_2, \dots, x_n$  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  $x_1, x_2, \dots, x_n$  : integer,  $y$  : integer, output idx : integer)
{
  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, maka idx diisi dengan 0.
  Input:  $x_1, x_2, \dots, x_n$ 
  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
```

Answer:

1. Operasi Assignment = 4

2. Operasi Perbandingan = 2

**Tmin(n) = 4 + 2 = 6**

1. Operasi Assignment = 1 + 1 + n + 1 = 3 + n

2. Operasi Perbandingan = n + 1

3. Operasi Penjumlahan = n

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

$(Tmin(n) + Tmax(n)) / 2 = (6 + 4 + 3n) / 2 = (10 + 3n) / 2$

**Tavg(n) = (10 + 3n) / 2**

### Studi Kasus 3: Binary Search

Diberikan larik bilangan bulat  $x_1, x_2, \dots, x_n$  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 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  $x_1, x_2, \dots, x_n$  : integer,  $x$  : integer, output :  $idx$  : integer)
{ 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 maka  $idx$  diisi dengan 0.
  Input:  $x_1, x_2, \dots, x_n$ 
  Output:  $idx$ 
}
Deklarasi
   $i, j, mid$  : integer
  found : Boolean
Algoritma
   $i \leftarrow 1$ 
   $j \leftarrow n$ 
  found  $\leftarrow$  false
  while (not found) and ( $i \leq j$ ) do
     $mid \leftarrow (i + j) \text{ div } 2$ 
    if  $x_{mid} = y$  then
      found  $\leftarrow$  true
    else

```

```

      if  $x_{mid} < y$  then {mencari di bagian kanan}
         $i \leftarrow mid + 1$ 
      else {mencari di bagian kiri}
         $j \leftarrow mid - 1$ 
      endif
    endif
  endwhile
  {found or  $i > j$ }

  If found then
     $idx \leftarrow mid$ 
  else
     $idx \leftarrow 0$ 
  endif

```

Answer:

1. Operasi Assignment = 6

2. Operasi Perbandingan = 2

**$T_{min}(n) = 6 + 2 = 8$**

Panjang array akan berubah pada setiap iterasi:

- Iterasi 1 =  $n$
- Iterasi 2 =  $n/2$
- Iterasi 3 =  $n/2^2$
- Iterasi  $x = n/2^{k-1} \sim n/2^k$  (-1 diabaikan karena kecil dibanding  $n/2^k$ )

Panjang array menjadi 1.

Maka,

$$n/2^k = 1$$

$$n = 2^k$$

$$\log_2(n) = \log_2(2^k) = k \log_2(2)$$

$$k = \log_2(n)$$

$$T_{\max}(n) = (\log_2(n))$$

$$(T_{\min}(n) + T_{\max}(n)) / 2 = (1 + \log_2(n)) / 2$$

$$T_{\text{avg}}(n) = (1 + \log_2(n)) / 2$$

## 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  $x_1, x_2, \dots, x_n$  : integer)
{ Mengurutkan elemen-elemen  $x_1, x_2, \dots, x_n$  dengan metode insertion sort.
  Input:  $x_1, x_2, \dots, x_n$ 
  Output:  $x_1, x_2, \dots, x_n$  (sudah terurut menaik)
}
Deklarasi
  i, j, insert : integer
Algoritma
  for i  $\leftarrow$  2 to n do
    insert  $\leftarrow$   $x_i$ 
    j  $\leftarrow$  i
    while (j < i) and ( $x[j]$  > insert) do
       $x[j]$   $\leftarrow$   $x[j-1]$ 
      j  $\leftarrow$  j-1
    endwhile
     $x[j]$  = insert
  endfor
```



Answer:

1. Operasi Assignment:  $2(n-1) + (n-1) = 3n-3$
2. Operasi Perbandingan:  $2*((n-1) + (n-1)) = 2*(2n-2) = 4n-4$
3. Operasi Pertukaran:  $(n-1) * n = n^2-n$

$$T_{\min}(n) = 3n-3 + 4n-4 + 1 = 7n-6$$

$$T_{\max}(n) = 3n-3 + 4n-4 + n^2-n = n^2+6n-6$$

$$(T_{\min}(n) + T_{\max}(n)) / 2 = (7n-6 + n^2+6n-6) / 2$$

$$T_{\text{avg}}(n) = (n^2 + 13n - 12) / 2$$

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

Answer:

$$1. \text{ Operasi Perbandingan} = \frac{1}{2}(n^2-n)$$

$$2. \text{ Operasi Pertukaran} = n-1$$

$$T_{\min}(n) = (4n-4) + \frac{1}{2}(n^2-n) + 1 \sim n^2$$

$$T_{\max}(n) = \frac{1}{2}(n^2-n) + (n-1) \sim n^2$$

$$(T_{\min}(n) + T_{\max}(n)) / 2 = (n^2 + n^2) / 2$$

$$T_{\text{avg}}(n) = n^2$$