

Laporan Praktikum II

Analisis Algoritma



Disusun Oleh :

Irfan Satrio Nugroho
140810180003

Worksheet02

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  $x_i > \text{maks}$  then
    maks  $\leftarrow x_i$ 
  endif
   $i \leftarrow i + 1$ 
endwhile
```

Jawaban Studi Kasus 1

$$\begin{aligned} T(n) &= 2(n-2) + (n-2) + 2 \\ &= 3n - 4 \end{aligned}$$

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 () saja, tetapi juga bergantung pada nilai elemen () 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 $=$, maka waktu pencariannya lebih cepat 130 kali dari pada $=$ atau 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 .
- (2) $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata (**average case**) merupakan kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari . 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 .

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 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 }
i  $\leftarrow$  1
found  $\leftarrow$  false
while (i  $\leq$  n) and (not found) do
  if  $x_i = y$  then
    found  $\leftarrow$  true
  else
    i  $\leftarrow$  i + 1
  endif
endwhile
{ i < n or found }

If found then {  $y$  ditemukan }
  idx  $\leftarrow$  i
else
  idx  $\leftarrow$  0 {  $y$  tidak ditemukan }
endif
```

Jawaban Studi Kasus 2

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.

$$T_{\text{avg}}(n) = \frac{(1 + 2 + 3 + \dots + n)}{n} = \frac{\frac{1}{2}n(1 + n)}{n} = \frac{(n + 1)}{2}$$

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 :  $\text{idx}$  : integer)
{
  Mencari  $y$  di dalam elemen  $x_1, x_2, \dots, x_n$ . Lokasi (indeks elemen) tempat  $y$  ditemukan diisi ke dalam
   $\text{idx}$ . Jika  $y$  tidak ditemukan maka  $\text{idx}$  diisi dengan 0.

  Input:  $x_1, x_2, \dots, x_n$ 
  Output:  $\text{idx}$ 
}

```

Deklarasi $i, j,$
 mid : integer
 found :

Boolean

Algoritma

$i \leftarrow 1$

$j \leftarrow n$

$\text{found} \leftarrow \text{false}$

while (not found) and ($i \leq j$) do

$\text{mid} \leftarrow (i + j) \text{ div } 2$

if $x_{\text{mid}} = y$ then

found

$\leftarrow \text{true}$

else

if $x_{\text{mid}} < y$ then {mencari di bagian kanan}

$i \leftarrow \text{mid} + 1$

else

 {mencari di bagian kiri}

$j \leftarrow \text{mid} - 1$

endif

endif

endwhile

{*found or* $i > j$ }

If found then

$\text{Idx} \leftarrow \text{mid}$

else

$\text{Idx} \leftarrow 0$

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.

```
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:  $x_1, x_2, \dots, x_n$ 
    OutputL  $x_1, x_2, \dots, x_n$  (sudah terurut menaik)
}
```

Deklarasi

```
    i, j, insert : integer Algoritma
    for i  $\square$  2 to n do
        insert  $\square$   $x_i$ 
        j  $\square$  i
        while (j < i) and ( $x[j-i] > \text{insert}$ ) do
             $x[j] \square x[j-1]$ 
            j  $\square$  j-1
        endwhile
         $x[j] = \text{insert}$     endfor
```

Jawaban Studi Kasus 4

Loop sementara dijalankan hanya jika $i > j$ dan $\text{arr}[i] < \text{arr}[j]$. 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 $O(n^2)$.

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  $x_1, x_2, \dots, x_n$  : 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  $\square$  n downto 2 do {pass sebanyak n-1 kali}
        imaks  $\square$  1
        for j  $\square$  2 to i do    if  $x_j$ 
>  $x_{\text{imaks}}$  then
            imaks  $\square$  j
        endif endfor
```

```
{pertukarkan  $x_{\text{maks}}$  dengan  $x_i$ }  
temp  $\leftarrow x_i$   
 $x_i \leftarrow x_{\text{maks}}$   
 $x_{\text{maks}} \leftarrow \text{temp}$   
endfor
```

Jawaban Studi Kasus 5

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

$i = 1 \rightarrow \text{jumlah perbandingan} = n - 1$

$i = 2 \rightarrow \text{jumlah perbandingan} = n - 2$

$i = 3 \rightarrow \text{jumlah perbandingan} = n - 3$

:

$i = k \rightarrow \text{jumlah perbandingan} = n - k$

:

$i = n - 1 \rightarrow \text{jumlah perbandingan} = 1$

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

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.