

Praktikum Analisis Algoritma

Worksheet 2



Disusun Oleh:

Ahmad Egy Aranda
140810180043

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN**

2020

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:

$$T(n) = 2(n - 2) + (n - 2) + 2$$

$$= 3n - 4$$

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

$i \leftarrow 1$

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. Best Case: ini terjadi bila $a_1 = x$.

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

2. Worst Case: bila $a_n = x$ atau x tidak ditemukan.

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

3. Average: 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 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 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 makai dx 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) div 2
    if  $x_{\text{mid}} = y$  then
      found  $\leftarrow$  true
    else
```

```
      if  $x_{\text{mid}} < y$  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
```

Jawaban Studi Kasus 3 :

1. Best Case
 $T_{\min}(n) = 1$
2. Worst Case
 $T_{\max}(n) = {}^2\log n$
3. Average
Jika terdapat pada index pada awal atau akhir elemen.

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 ( $x[j-i]$  > insert) do
       $x[j] \leftarrow x[j-1]$ 
      j  $\leftarrow$  j-1
    endwhile
     $x[j] =$  insert
  endfor
```

Jawaban Studi Kasus 4 :

- Best Case : jika array sudah terurut dengan benar, jadi tidak ada looping.
- Average : jika array sudah terurut setengahnya / sebagian dari seluruh elemen.

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 Worst Case, bisa ada inversi $n * (n-1) / 2$. Worst Case terjadi saat array terurut secara terbalik. Jadi kompleksitas waktu saat worst case 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:  $x_1, x_2, \dots, x_n$ 
  Output:  $x_1, x_2, \dots, x_n$  (sudah terurut menaik)
}
Deklarasi
  i, j, imaks, temp : integer
Algoritma
  for i  $\leftarrow$  n downto 2 do {pass sebanyak n-1 kali}
    imaks  $\leftarrow$  1
    for j  $\leftarrow$  2 to i do
      if  $x_j > x_{\text{imaks}}$  then
        imaks  $\leftarrow$  j
      endif
    endfor
    {pertukarkan  $x_{\text{imaks}}$  dengan  $x_i$ }
    temp  $\leftarrow$   $x_i$ 
     $x_i \leftarrow x_{\text{imaks}}$ 
     $x_{\text{imaks}} \leftarrow$  temp
  endfor
```

Jawaban Studi Kasus 5 :

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

$i = 1 \rightarrow$ jumlah perbandingan = $n - 1$

$i = 2 \rightarrow$ jumlah perbandingan = $n - 2$

$i = 3 \rightarrow$ jumlah perbandingan = $n - 3$

:

$i = k \rightarrow$ jumlah perbandingan = $n - k$

:

$i = n - 1 \rightarrow$ jumlah perbandingan = 1

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah

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

Ini adalah kompleksitas waktu untuk worst case dan best case, 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.