

LAPORAN PRAKTIKUM 2
KOMPLEKSITAS WAKTU DARI ALGORITMA

MATA KULIAH
ANALISIS ALGORITMA



Disusun Oleh :

Putri Nabila

140810180007

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
MARET 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

Source Code:

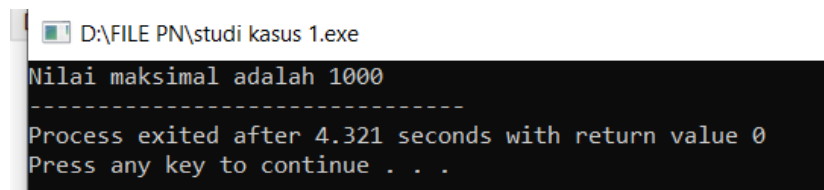
```
/* Nama : Putri Nabila
Kelas : A
NPM : 140810180007
Tanggal 08 Maret 2020
Deskripsi : studi kasus 1 "algoritma Pencarian Nilai Maksimal"
*/
```

```
#include <iostream>
using namespace std;
```

```
#define N 10
int CariMaks(int x[]){
    int maks = x[0];
    for(int i = 1; i < N; i++){
        if(x[i] > maks)
            maks = x[i];
    }
    return maks;
}
```

```
int main(){
    int x[N] = {5,6,1000,9,7,15,40,6,10,300};
    cout << "Nilai maksimal adalah "<<CariMaks(x);
}
```

Hasil Program:



```
D:\FILE PN\studi kasus 1.exe
Nilai maksimal adalah 1000
-----
Process exited after 4.321 seconds with return value 0
Press any key to continue . . .
```

(i) Operasi pengisian nilai (*assignment*)

$\text{maks} \leftarrow x_1$	1 kali
$i \leftarrow 2,$	1 kali
$\text{maks} \leftarrow x_i$	$n-1$ kali (worst case)
	0 kali (best case)
$i \leftarrow i + 1,$	n kali

Jumlah seluruh operasi pengisian nilai (*assignment*) adalah

$$t_1 = 1 + 1 + n - 1 + n = 2n + 1 \text{ (worst case)}$$

$$t_1 = 1 + 1 + 0 + n = n + 2 \text{ (best case)}$$

(ii) Operasi penjumlahan

$i + 1_k,$	n kali
------------	----------

Jumlah seluruh operasi penjumlahan adalah

$$t_2 = n$$

$$T_{\min}(n) = t_1 + t_2 = n + 2 + n = 2n + 2$$

$$T_{\max}(n) = t_1 + t_2 = 2n + 1 + n = 3n + 1$$

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  $x_1, x_2, \dots, x_n$  : 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 ← 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

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 Case: Jika x ditemukan pada posisi ke- j , maka operasi perbandingan ($a_k = x$) akan dieksekusi sebanyak j kali.
 $T_{\text{avg}}(n) = (1+2+3+..+n)/n = (1/2n(1+n))/n = (n+1)/2$

Source Code:

```

/* Nama : Putri Nabila
Kelas : A
NPM : 140810180007
Tanggal 08 Maret 2020
Deskripsi : studi kasus 2 "suqeantial Search"
*/

#include <iostream>
using namespace std;

#define N 4
int SequentialSearch(int *x, int y){
    int idx;
    int i = 0;
    bool found = false;
    while( i < sizeof(x) && !found){
        if(x[i] == y)
            found = true;
        else
            i++;
    }

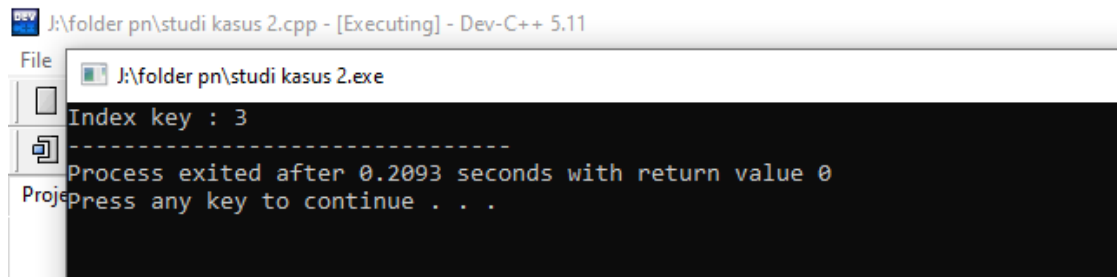
    if(found)
        idx = i;
    else
        idx = 0;
    return idx;
}

int main(){
    int x[N] = {1,3,99,2};
    cout << "Index key : " << SequentialSearch(x,2);
}

```

}

Hasil program :



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 maka idx diisi dengan 0.
Input: x_1, x_2, \dots, x_n
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  $x_{\text{mid}} = y$  then
        found ← true
    else
        if  $x_{\text{mid}} < y$  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
```

Jawaban Studi Kasus 3

1. Kasus terbaik

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

2. Kasus terburuk

$$T_{\max}(n) = {}^2\log n$$

Source code nya :

```
/* Nama : Putri Nabila
```

```
Kelas : A
```

```
NPM : 140810180007
```

```
Tanggal 08 Maret 2020
```

```
Deskripsi : studi kasus 2 "Binary Search"
```

```
*
```

```
#include <iostream>
```

```
using namespace std;
```

```
#define N 5
```

```
int BinarySearch(int *x, int y){
```

```
    int i = 0, j = N, mid;
```

```
    bool found = false;
```

```
    while (!found && i <= j){
```

```
        mid = (i+j)/2;
```

```
        if( x[mid] == y )
```

```
            found = true;
```

```
        else if( x[mid] < y)
```

```
            i = mid + 1;
```

```
        else
```

```
            j = mid - 1;
```

```
    }
```

```
}
```

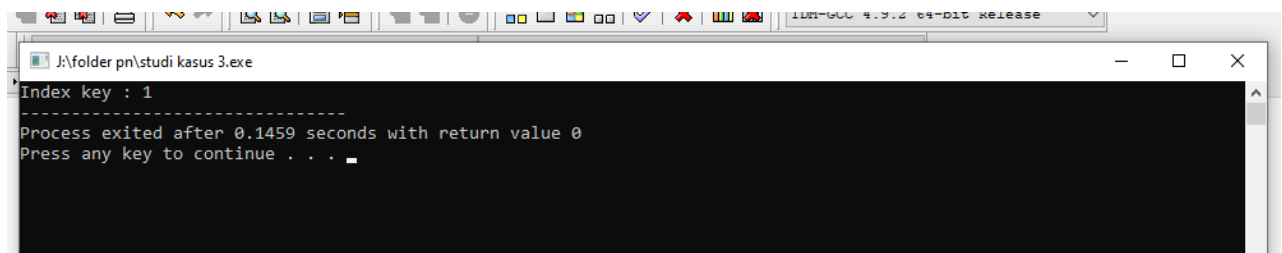
```
int main(){
```

```
    int x[N] = {1,3,99,2,4};
```

```
    cout << "Index key : " << BinarySearch(x,2);
```

```
}
```

Hasil programnya :



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
```

Jawaban Studi Kasus 4

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 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 $O(n^2)$.

Endfor

Source code :

```
/* Nama : Putri Nabila
Kelas : A
NPM : 140810180007
Tanggal 08 Maret 2020
Deskripsi : studi kasus 4 "Insertion Search"
*/
```

```
#include <iostream>
```

```
using namespace std;
```

```
#define N 5
```

```
void InsertionSort(int *x){
  int insert,j;
  for(int i = 1; i < N; i++){
    insert = x[i];
    j = i-1;
    while(j >= 0 && x[j] > insert){
      x[j+1] = x[j];
      j--;
    }
  }
}
```

```

    }
    x[j+1] = insert;
}
}

void printArray(int *x){
    for(int i = 0; i < N; i++)
    {
        cout << " " << x[i];
    }
    cout << endl;
}

int main(){
    int x[N] = {1,99,9,60,1000};
    cout << "Sebelum sort : "; printArray(x);
    InsertionSort(x);
    cout << "Setelah sort : "; printArray(x);
}

```

Hasil program :

```

J:\folder pn\studi kasus 4.exe
Sebelum sort : 1 99 9 60 1000
Setelah sort : 1 9 60 99 1000

-----
Process exited after 0.1426 seconds with return value 0
Press any key to continue . . .

```

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  $x_j > x_{imaks}$  then
            imaks ← j
        endif
    endfor

```



```

    {pertukarkan  $x_{\text{imaks}}$  dengan  $x_i$ }
    temp  $\leftarrow x_i$ 
     $x_i \leftarrow x_{\text{imaks}}$ 
     $x_{\text{imaks}} \leftarrow \text{temp}$ 
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.

Source code:

```
/* Nama : Putri Nabila
```

```
Kelas : A
```

```
NPM : 140810180007
```

```
Tanggal 08 Maret 2020
```

```
Deskripsi : studi kasus 5 "Selection Search"
```

```
*/
```

```
#include <iostream>
```

```
using namespace std;
```

```
#define N 5
```

```
void SelectionSort(int *x){
    int imaks,temp;
    for(int i = N-1; i >= 1; i--){
        imaks = 0;
        for(int j = 1; j <= i; j++){
            if(x[j] > x[imaks])
                imaks = j;
        }
        temp = x[i];
        x[i] = x[imaks];
    }
}
```

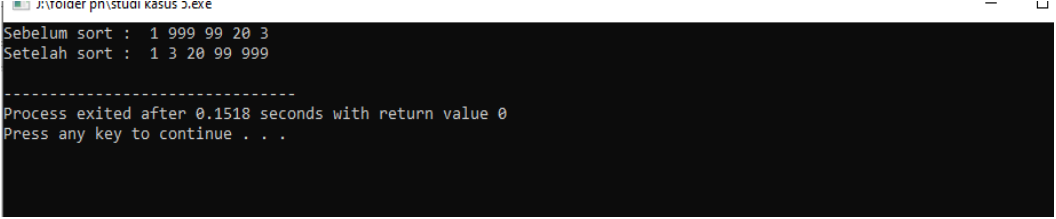
```

        x[imaks] = temp;
    }
}

void printArray(int *x){
    for(int i = 0; i < N; i++)
    {
        cout << " " << x[i];
    }
    cout << endl;
}

int main(){
    int x[N] = {1,999,99,20,3};
    cout << "Sebelum sort : "; printArray(x);
    SelectionSort(x);
    cout << "Setelah sort : "; printArray(x);
}

```



```

J:\Volder pn\studi kasus 3.exe
Sebelum sort : 1 999 99 20 3
Setelah sort : 1 3 20 99 999

-----
Process exited after 0.1518 seconds with return value 0
Press any key to continue . . .

```