**LAPORAN PRAKTIKUM**

**ANALISIS ALGORITMA**

****

Muhammad Risqullah Sudanta Gorau

140810180066

**PROGRAM STUDI S-1 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 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 ß x1 i ß 2

while i ≤ n do

if xi > maks then

maks ß xi

endif

i ß i + 1 endwhile

Jawaban :

**Source Code**

/\*

Nama : Muhammad Risqullah Sudanta Gorau

NPM : 140810180066

Kelas : B

\*/

#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;

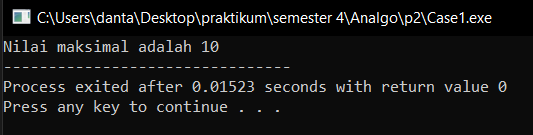
}

int main(){

int x[N] = {1,2,3,4,5,6,7,8,9,10};

cout << "Nilai maksimal adalah "<<CariMaks(x);

}



1. Operasi pengisian nilai (*assignment*)

maks ß x1 1 kali

#### i ß 2, 1 kali

maks ß x1 n-1 kali (worst case) 0 kali (best case)

i ß i + 1, n kali

#### Jumlah seluruh operasi pengisian nilai (*assignment*) adalah

𝑡1 = 1 + 1 + 𝑛 − 1 + 𝑛 = 2𝑛 + 1 (𝑤𝑜𝑟𝑠𝑡 𝑐𝑎𝑠𝑒)

𝑡1 = 1 + 1 + 0 + 𝑛 = 𝑛 + 2 (𝑏𝑒𝑠𝑡 𝑐𝑎𝑠𝑒)

#### Operasi penjumlahan

i + 1k, n kali

#### Jumlah seluruh operasi penjumlahan adalah

𝑡2 = 𝑛

𝑇𝑚𝑖𝑛(𝑛) = 𝑡1 + 𝑡2 = 𝑛 + 2 + 𝑛 = 2𝑛 + 2

𝑇𝑚𝑎𝑥(𝑛) = 𝑡1 + 𝑡2 = 2𝑛 + 1 + 𝑛 = 3𝑛 + 1

Studi Kasus 2: *Sequential Search*

#### Diberikan larik bilangan bulan 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 𝑦 ditemukan diisi ke dalam idx. Jika 𝑦 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 ß 1

found ß false

while (i ≤ n) and (not found) do if xi = y then

found ß true

else

i ß i + 1 endif

endwhile

{*i < n or found*}

If found then {*y ditemukan*}

idx ß i

else endif

Jawaban :

1. Best Case: ini terjadi bila a1 = x. Tmin(n) = 1
2. Worst Case : bila an = x atau x tidak ditemukan. Tmax(n) = n
3. *Average Case*: Jika *x* ditemukan pada posisi ke-*j*, maka operasi perbandingan (ak = x) akan dieksekusi sebanyak *j* kali.

*T*avg(*n*) = (1+2+3+..+n)/n = (1/2n(1+n))/n = (n+1)/2

**Source Code**

/\*

Nama : Muhammad Risqullah Sudanta Gorau

NPM : 140810180066

Kelas : B

\*/

#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;

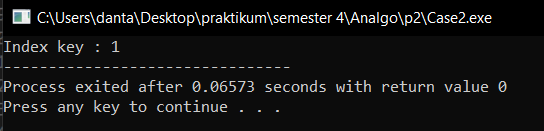
}

int main(){

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

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

}



Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan 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 ß 1 j ß n

found ß false

while (not found) and ( i ≤ j) do mid ß (i + j) div 2

if xmid = y then found ß true

else

if xmid < y then {*mencari di bagian kanan*} i ß mid + 1

else {*mencari di bagian kiri*} j ß mid – 1

endif endif

endwhile

{*found or i > j* }

If found then

Idx ß mid

else Endif

Idx ß 0

Jawaban:

1. Kasus terbaik

Tmin(n) = 1

1. Kasus terburuk Tmax (n) = 2log n

**Source Code**

/\*

Nama : Muhammad Risqullah Sudanta Gorau

NPM : 140810180066

Kelas : B

\*/

#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;

}

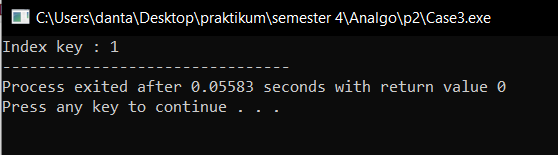
}

int main(){

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

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

}



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

OutputL 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

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 (n2).

**Source Code**

/\*

Nama : Muhammad Risqullah Sudanta Gorau

NPM : 140810180066

Kelas : B

\*/

#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(){

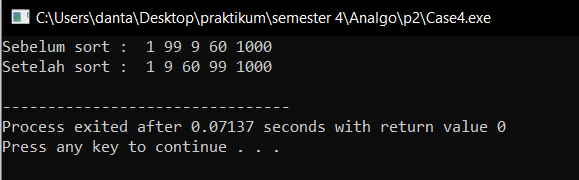
int x[N] = {1,99,9,60,1000};

cout << "Sebelum sort : "; printArray(x);

InsertionSort(x);

cout << "Setelah sort : "; printArray(x);

}



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 selection sort.

Input: 𝑥1, 𝑥2, … 𝑥𝑛

OutputL 𝑥1, 𝑥2, … 𝑥𝑛 (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 xj > ximaks then imaks  j

endif endfor

{pertukarkan ximaks dengan xi} temp  xi

xi  ximaks ximaks  temp

endfor

Jawaban Studi Kasus 5

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

*i* = 1 –> jumlah perbandingan = *n* – 1

*i* = 2 –> jumlah perbandingan = *n* – 2

*i* = 3 –> jumlah perbandingan = *n* – 3

*i* = *k* –> jumlah perbandingan = *n* – *k*

*i* = *n* – 1 –> jumlah perbandingan = 1

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah *T*(*n*) = (*n* – 1) + (*n* – 2) + … + 1

Ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma Urut tidak bergantung pada batasan apakah data masukannya sudah terurut atau acak.

1. 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 : Muhammad Risqullah Sudanta Gorau

NPM : 140810180066

Kelas : B

\*/

#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);

}

