**LAPORAN PRAKTIKUM 2**

**ANALISIS ALGORITMA**

****

**Disusun oleh :**

Alvin

140810180013

**PROGRAM STUDI S1 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, .

Contoh algoritma pengurutan elemen-elemen larik, adalah jumlah elemen larik. Sedangkan dalam algoritma perkalian matriks n adalah ukuran matriks .

1. Kompleksitas waktu, adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari input .
2. Kompleksitas ruang, , adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari input .

**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 x1, x2, …, xn: integer, output r: real)

{ Menghitung nilai rata-rata dari sekumpulan elemen larik integer x1, x2, … xn.

Nilai rata-rata akan disimpan di dalam variable r.

**Input:** x1, x2, … xn

**Output**: r (nilai rata-rata)

}

**Deklarasi**

i : integer

jumlah : real

**Algoritma**

Jumlah  0

i  1

while i ≤ n do

jumlah  jumlah + ai

i  i + 1

endwhile

{i > n}

r  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 “”)
* 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.

1. Operasi pengisian nilai (*assignment*)

jumlah  0, 1 kali

k  1, 1 kali

jumlah jumlah + ak n kali

k  k+1, n kali

r  jumlah/n, 1 kali

Jumlah seluruh operasi pengisian nilai (*assignment*) adalah

1. Operasi penjumlahan

Jumlah + ak, n kali

k+1, n kali

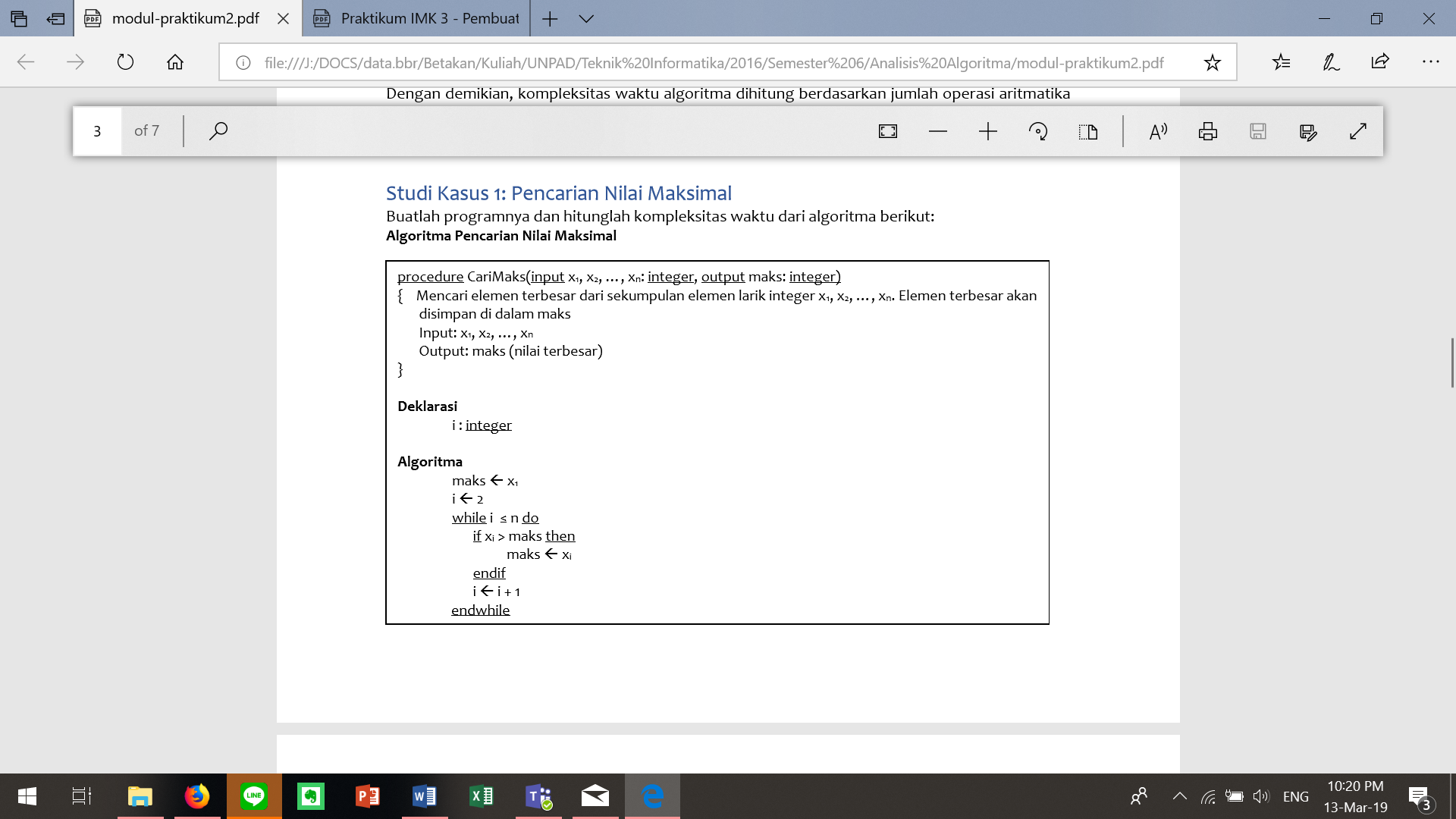
Jumlah seluruh operasi penjumlahan adalah

1. 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:



Jawab:

/\*

Nama : Alvin

NPM : 140810180013

Kelas : A

Program : Pencarian nilai maksimum

\*/

#include<iostream>

using namespace std;

main(){

int x[5]={21,22,99,24,25};

int n= sizeof(x)/sizeof(x[0]);

//deklarasi

int maks = x[0];

int i= 2;

//algoritma

while (i<= n){

if(x[i] > maks){

maks = x[i];

}

i=i+1;

}

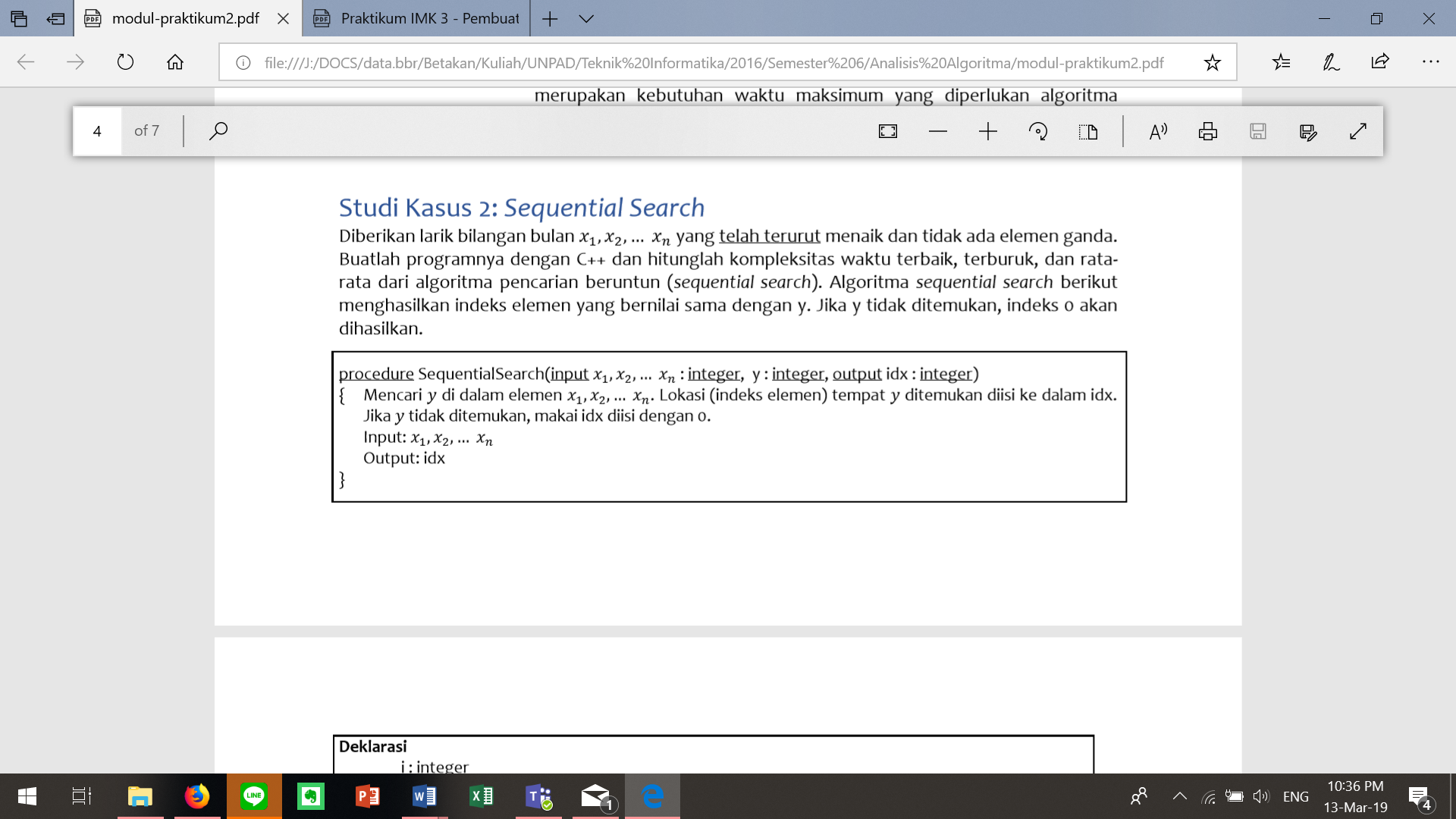
cout<<"Nilai maks dari array adalah : "<<maks;

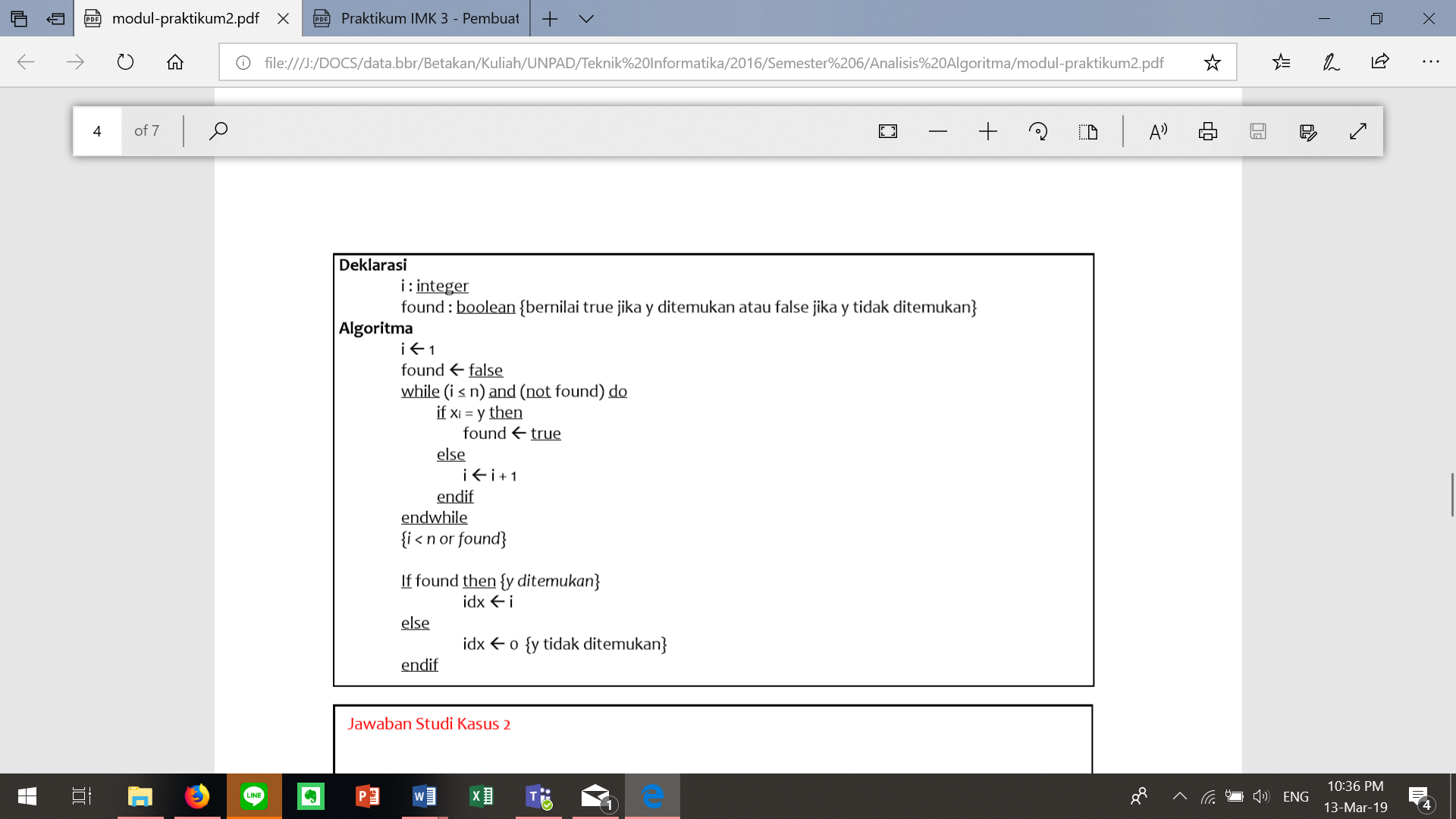
}

**Kompleksitas waktu:**

T(n) = 2(n – 2) + (n – 2) + 2

= 3 n - 4





Jawab:

/\*

Nama : Alvin

NPM : 140810180013

Kelas : A

Prigram : Sequential Search

\*/

#include<iostream>

using namespace std;

main(){

int x[5] = {1,2,99,10,1};//daftar list yang ada

int y = 99;//yang dicari

int n = sizeof(x)/sizeof(x[0]);

//deklrasi

int i = 1;

int idx;//output

bool found = false;

//algoritma

while(i<=n && !found){

if(x[i] == y){

found = true;

}else

i = i+1;

}

if(found == true){

idx = i;

}else

idx = 0;//tidak ditemukan

cout<<"Hasil cari index elemen : "<<idx;

}

**Kompleksitas waktu:**

* Kasus terbaik: ini terjadi bila x1 = y

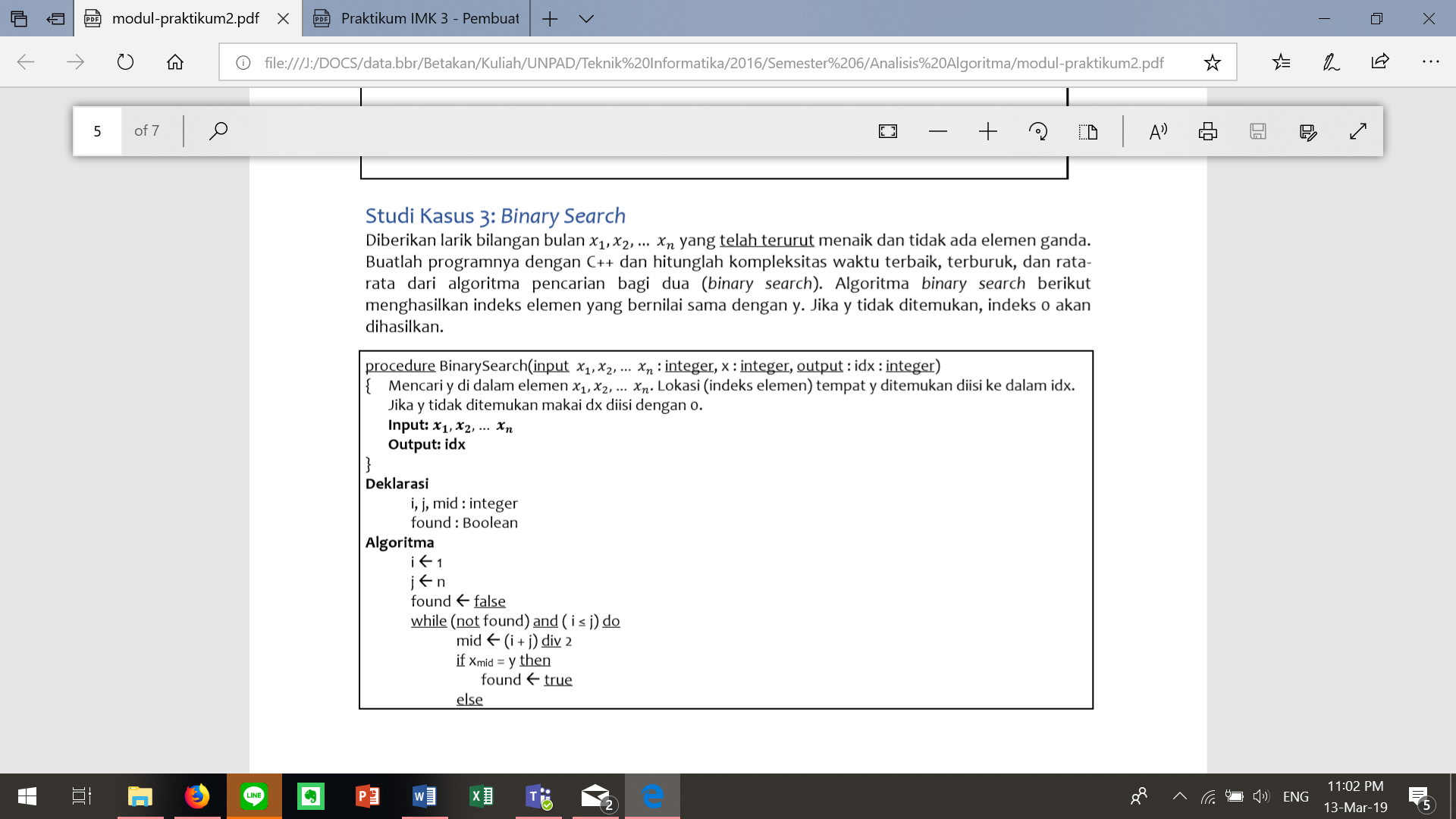
Tmin(n) = 1

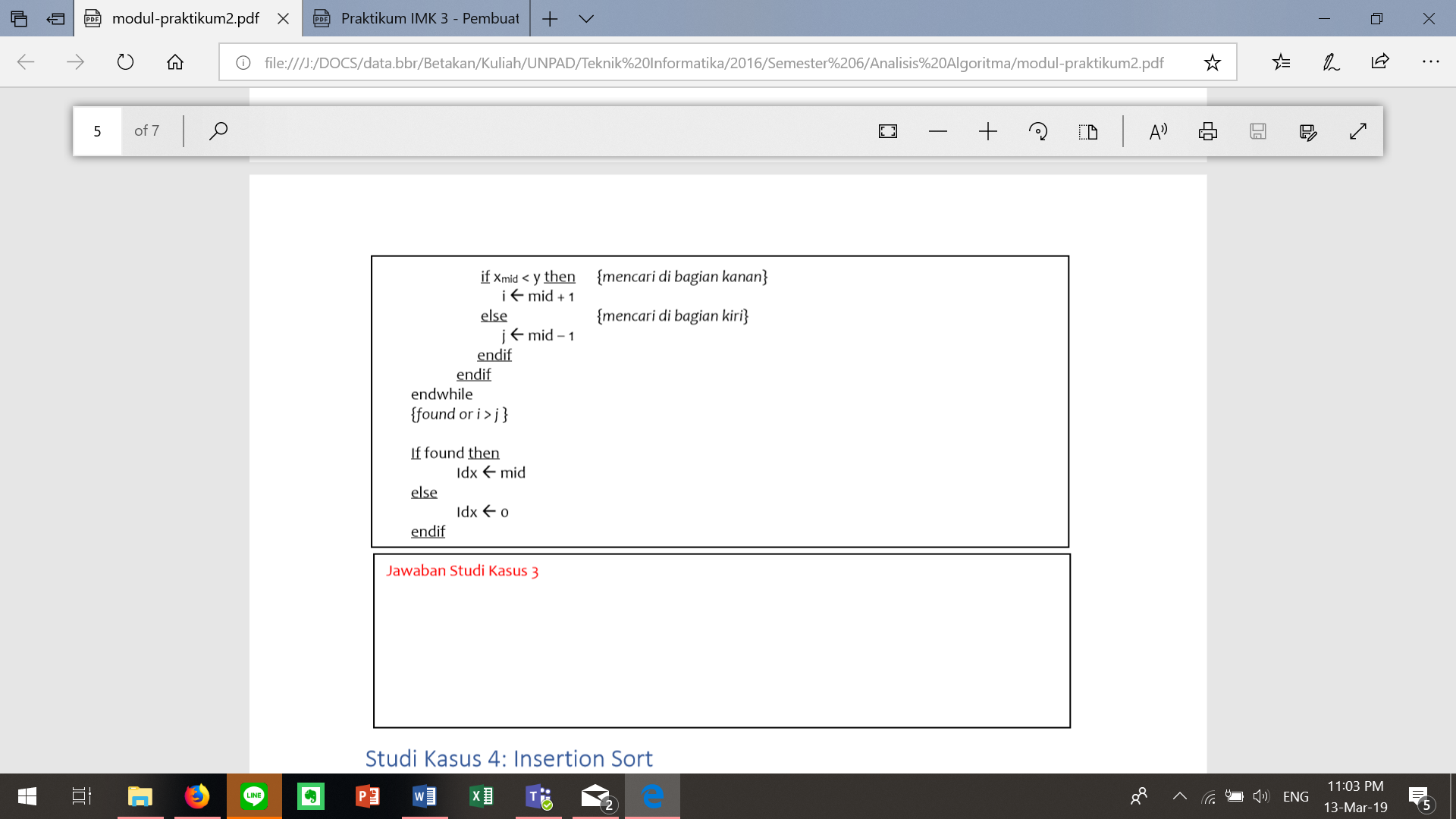
* Kasus terburuk: bila xn = y atau y tidak ditemukan.

Tmax(n) = n

* Kasus rata-rata: Jika *y* ditemukan pada posisi ke-*j*, maka operasi perbandingan (ak = y) akan dieksekusi sebanyak *j*kali.

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





Jawab:

/\*

Nama : Alvin

NPM : 140810180013

Kelas : A

Program : Binary Search

\*/

#include<iostream>

using namespace std;

main(){

int x[5]={1,3,7,9,11};//input

int idx; //output

int y = 7;//angka yang dicari

int n = sizeof(x)/sizeof(x[0]);

//deklarasi

int i, j, mid;

bool found;

//algoritma

i = 1;

j = n;

found = false;

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

mid = (i + j)/2;

if (x[mid] == y){

found = true;

}

else if(x[mid] < y){//menacri bagian kanan

i = mid+1;

}

else{

j = mid - 1; //mencari bagian kiri

}

}

if(found == true){

idx=mid;

}else

idx= 0;

cout<<"Hasil cari indeks elemen : "<<idx;

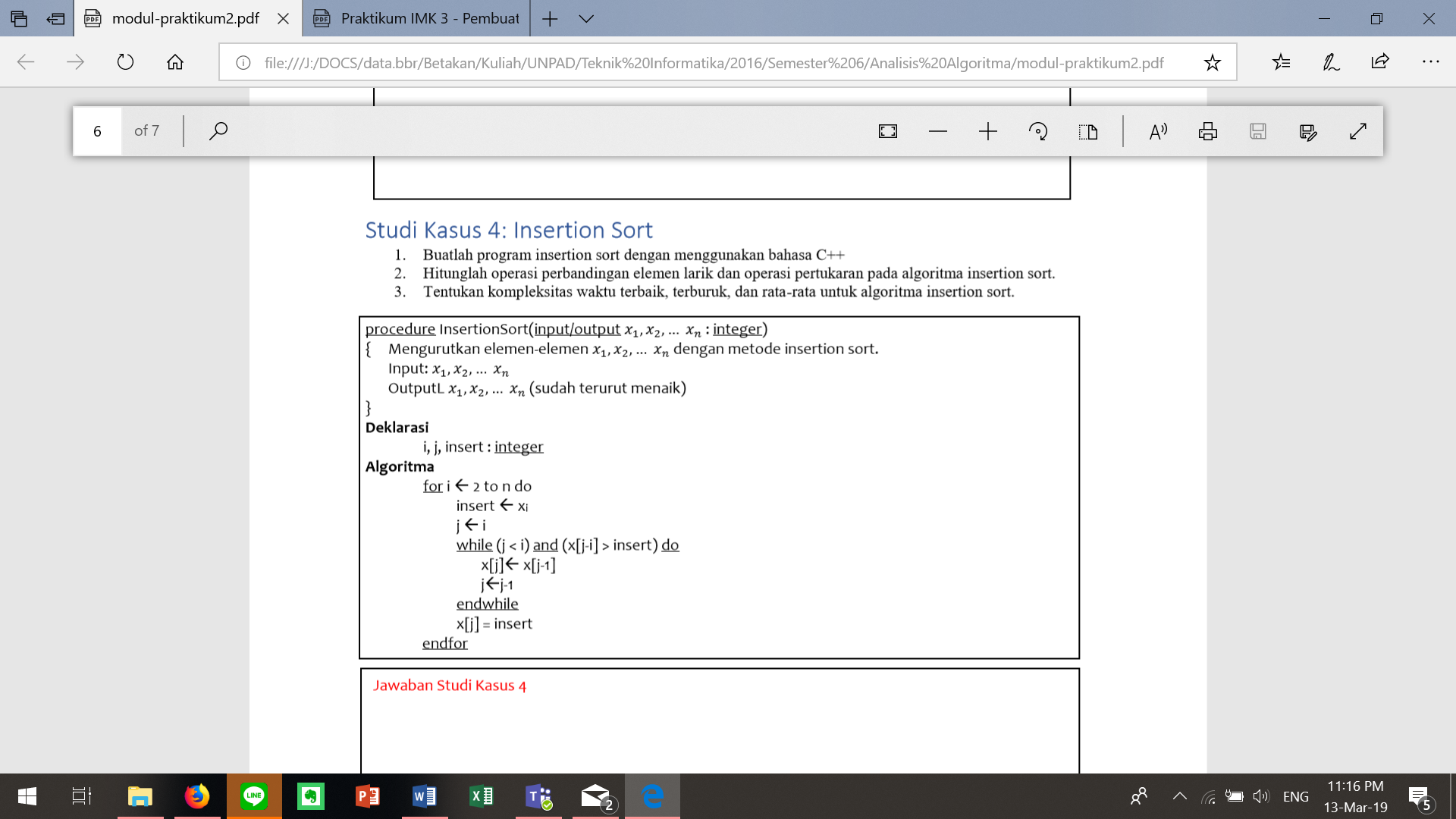
}

**Kompleksitas waktu:**

Best case: Jika ditemukan pada arr[mid] atau indeks di tengah yaitu Tmin(n) = 1

Average case: Jika ditemukan pada indeks di awal atau di akhir

Worst case: Jika tidak ditemukan sama sekali yaitu Tmax (n) = 2log n



Jawab:

/\*

Nama : Alvin

NPM : 140810180013

Kelas : A

Program : Insertion Sort

\*/

#include<iostream>

using namespace std;

main(){

int x[5]={1,7,11,31,2};

int n = sizeof(x)/sizeof(x[0]);

//deklarasi

int i , j, insert;

//Algoritma

for(i=1; i<n; i++){

insert= x[i];

j = i - 1;

while(j >= 0 && x[j] > insert){

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

j = j - 1;

}

x[j+1] = insert;

}

for(j = 0; j < n ; j++ ){

cout<<x[j]<<",";

}

}

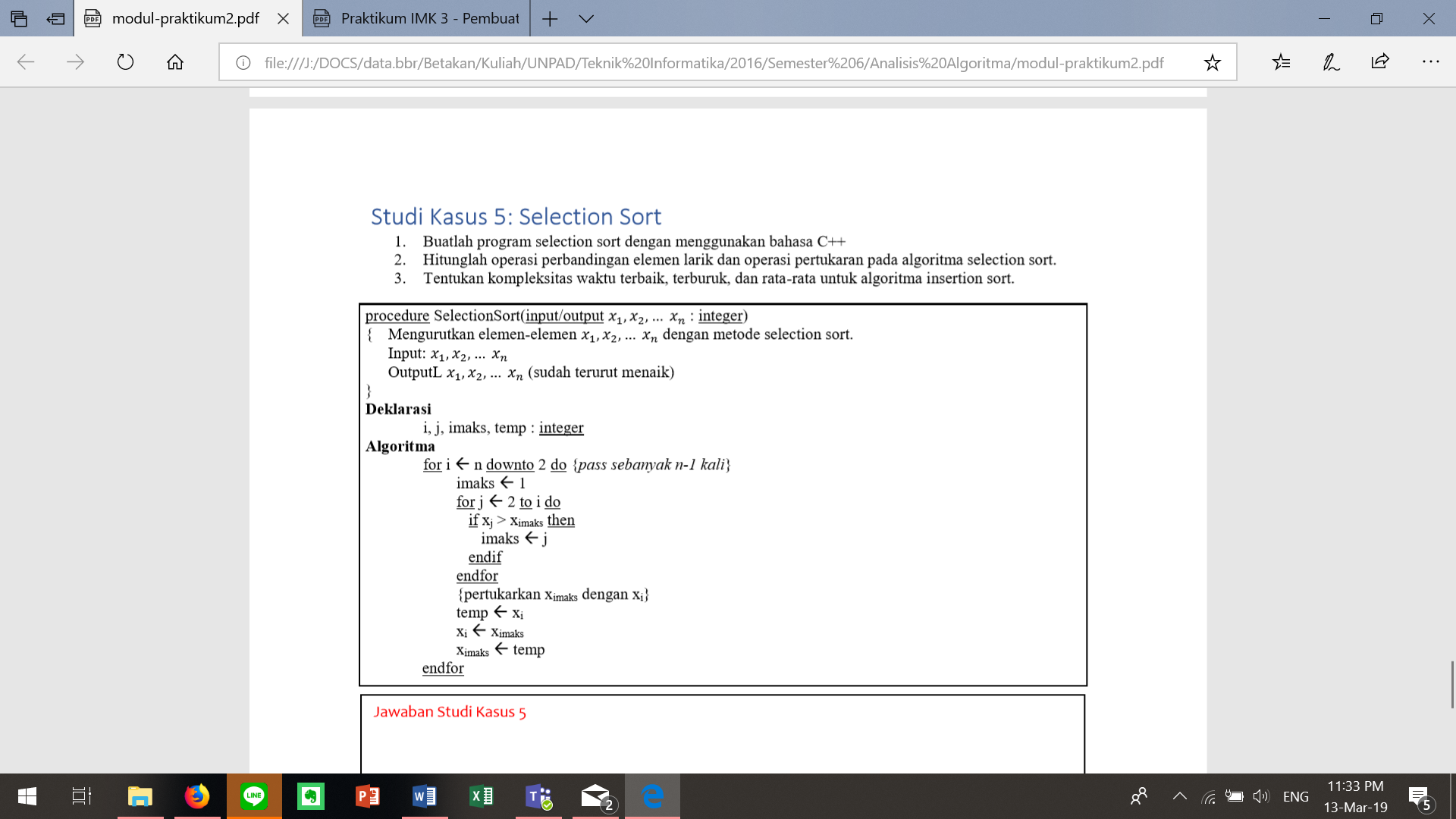
**Kompleksitas waktu:**

Best case: Jika array sudah terurut sehingga loop while tidak dijalankan

Average case: Jika sebagian elemen array sudah terurut

Worst case: Jika array harus diurutkan sebanyak n kali, 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).

|  |  |  |  |
| --- | --- | --- | --- |
| J | Perbandingan | Perpindahan | Total operasi |
| 2 | 1 | 1 | 2 |
| 3 | 2 | 2 | 4 |
| 4 | 3 | 3 | 6 |
| n | (n-1) | (n-1) | **2(n-1)** |



Jawab:

/\*

Nama : Alvin

NPM : 140810180013

Kelas : A

Program : Selection Sort

\*/

#include<iostream>

using namespace std;

main(){

int x[5] = { 1,4,2,5,3};

int n = sizeof(x)/sizeof(x[0]);

//deklarasi

int i, j, imaks, temp;

//algoritma

for ( i=2 ; i<n; i++){

imaks = 1;

for( j=2; j<i; j++){

if ( x[j] > x[imaks]){

imaks = j;

}

}

temp = x[i];

x[i] = x[imaks];

x[imaks] = temp;

}

for (int i=0; i<n; i++){

cout<<x[i]<<",";

}

}

**Kompleksitas waktu:**

1. Jumlah operasi perbandingan elemen

Untuk setiap loop ke-i,

i = 1 🡪 jumlah perbandingan = n-1

i = 2 🡪 jumlah perbandingan = n-2

i = k 🡪 jumlah perbandingan = n-k

i = n-1 🡪 jumlah perbandingan = 1

sehingga T(n) = (n-1) + (n-2) + … + 1 = n(n-1)/2 dimana kompleksitas waktu ini berlaku menjadi yang terbaik, rata-rata maupun yang terburuk karena algoritma ini tidak melihat apakah arraynya sudah urut atau tidak terlebih dahulu.

1. Jumlah operasi pertukaran

Untuk setiap loop ke-1 sampai n-1 terjadi satu kali pertukaran elemen sehingga T(n) = n-1.