TUGAS 2 PRAKTIKUM

KOMPLEKSITAS WAKTU DARI ALGORITMA



Muhammad Luthfiansyah

140810170023

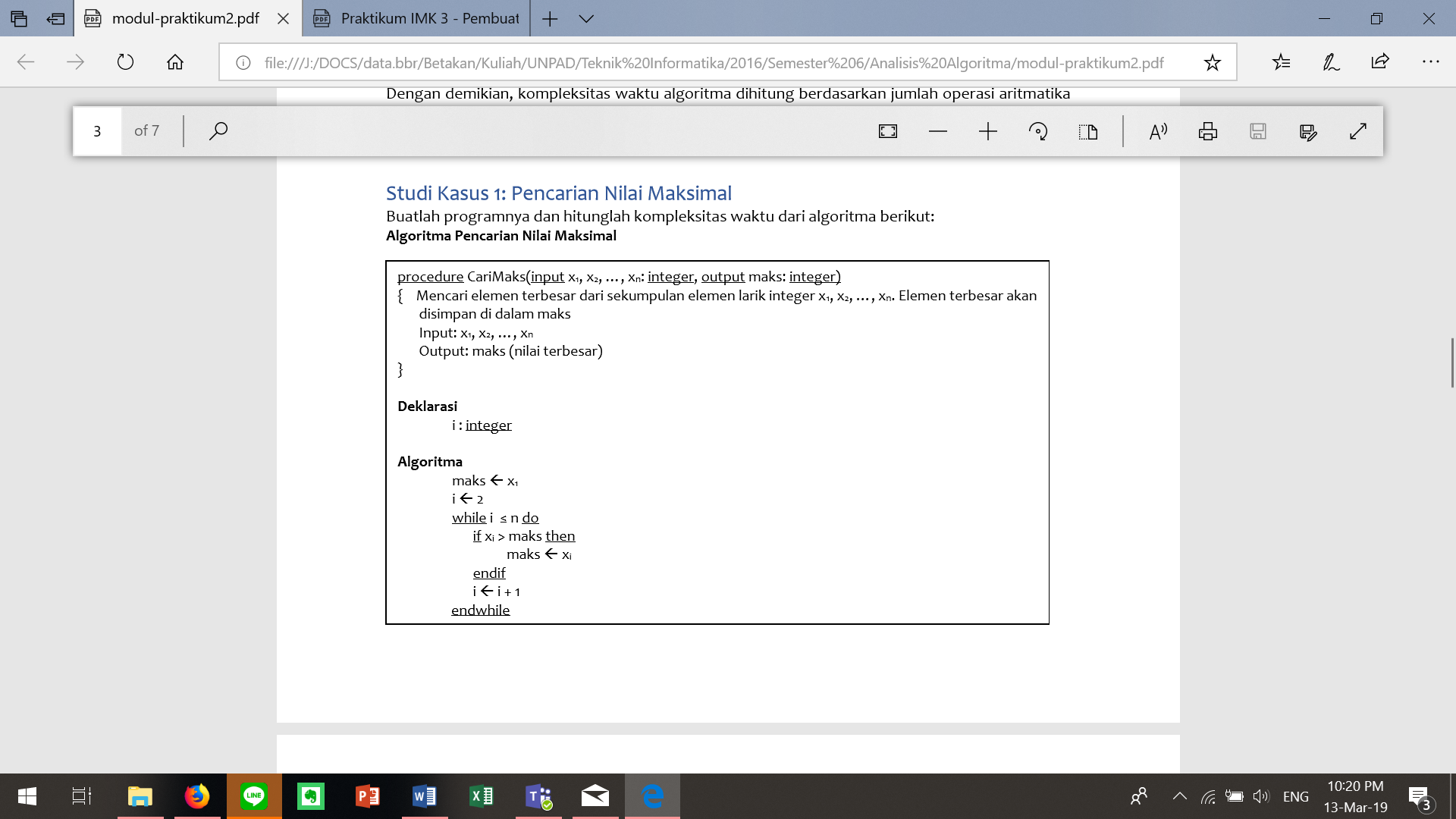
PROGRAM STUDI S-1 TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2019

**Nomor 1**



Jawab:

* Source code

#include <iostream>

using namespace std;

int main(){

    int arr[5] = {2,4,8,16,32};

    int panjangArr = sizeof(arr)/sizeof(arr[0]);

    int maks = arr[0];

    int i = 2;

    while(i<panjangArr){

        if(arr[i] > maks){

            maks = arr[i];

        }

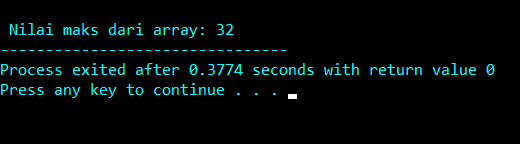
        i++;

    }

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

}

* Hasil screenshoot



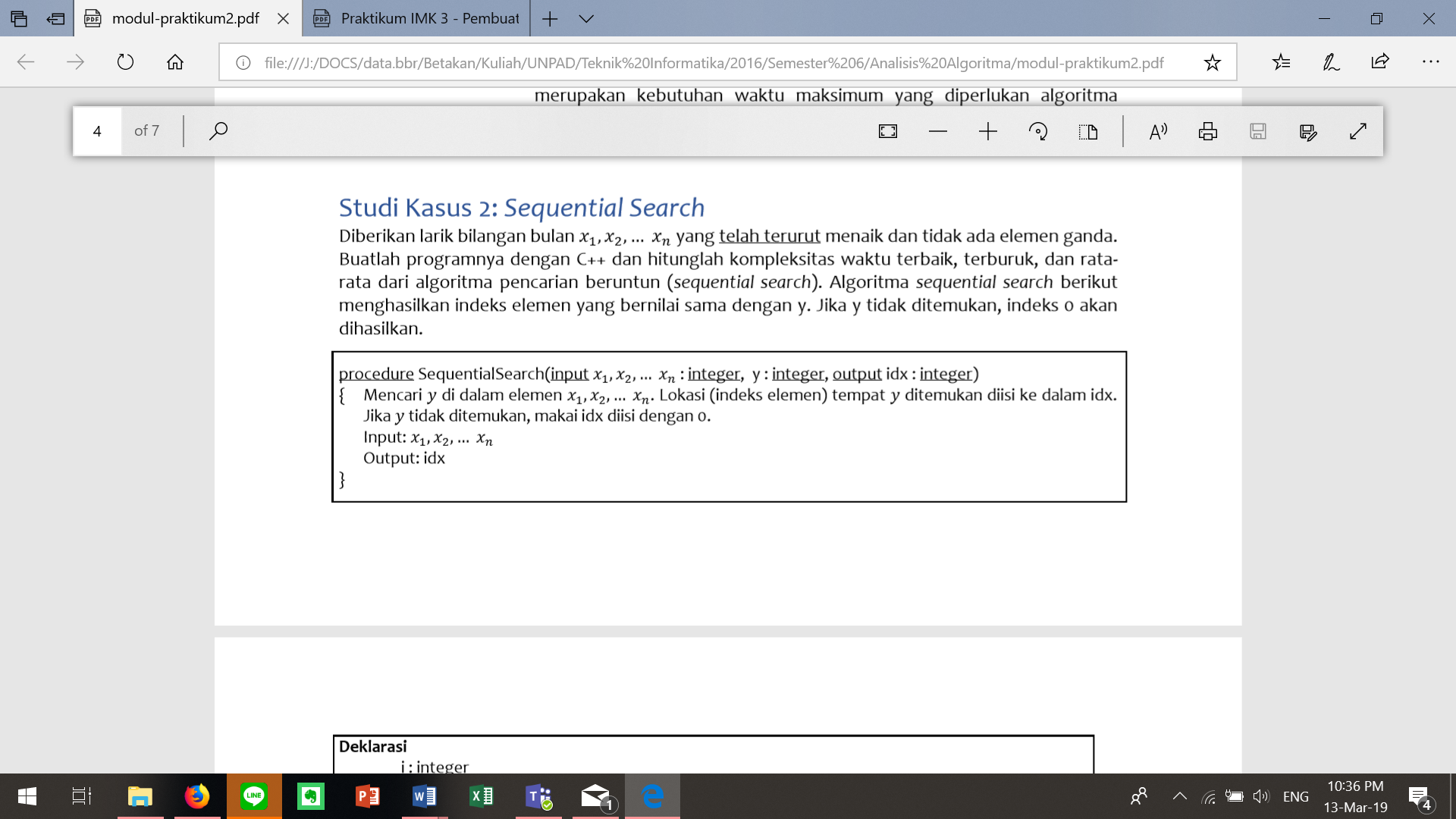
* Kompleksitas waktu:

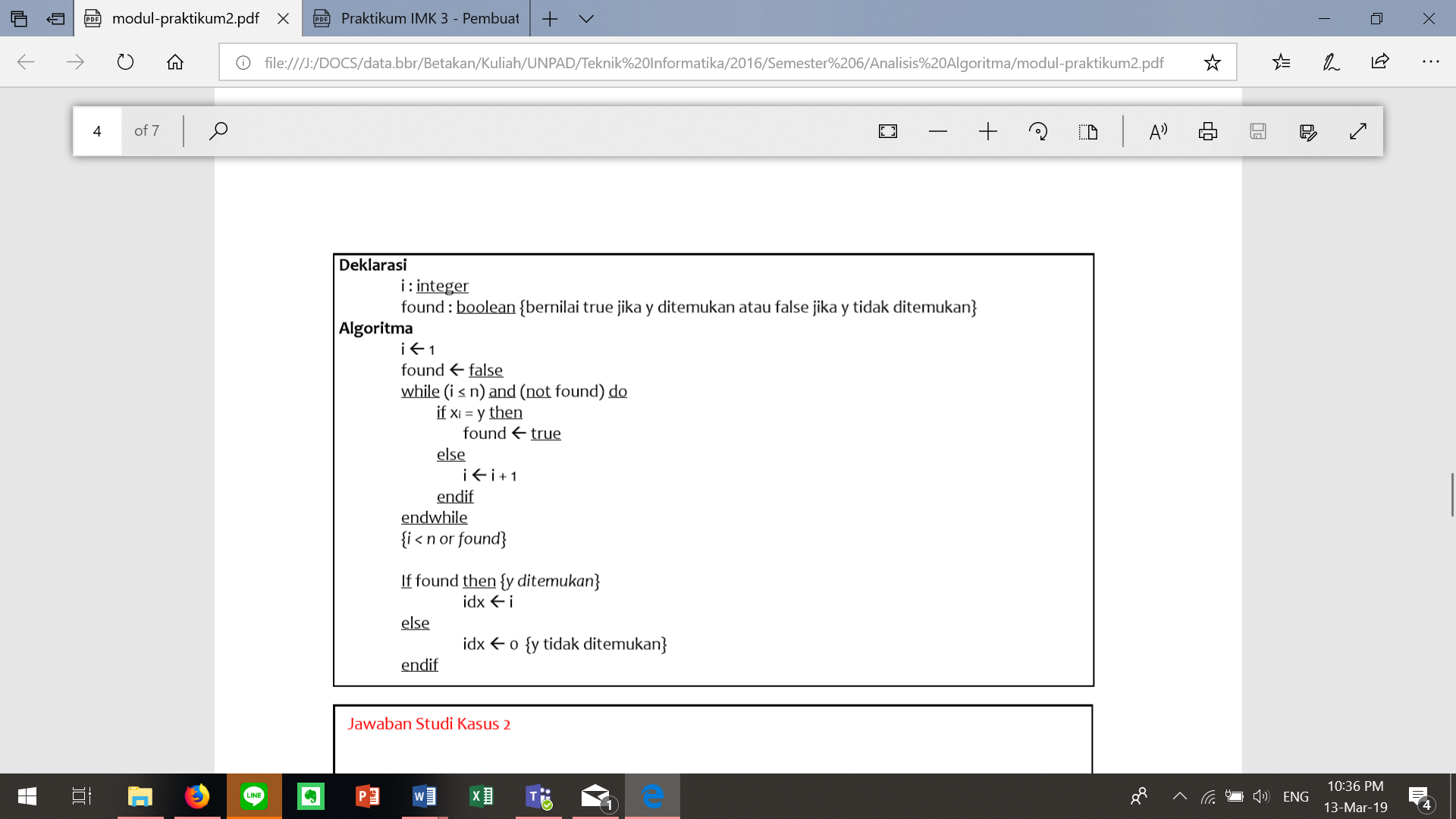
Best case: Jika nilai maks berada pada arr[0] atau index paling awal

Average case: Jika nilai maks berada pada arr[(n-1)/2] atau index di tengah

Worst case: Jika nilai maks berada pada arr[n-1] atau index paling akhir

**Nomor 2**





Jawab:

* Source code

#include <iostream>

using namespace std;

int main(){

  int arr[5] = {1,3,5,7,9};

  int panjangArr = sizeof(arr)/sizeof(arr[0]);

  int i = 1;

  int y = 3;

  int index;

  bool found = false;

  while(i <= panjangArr && !found){

    if(arr[i] == y){

        found = true;

    }

    else i++;

  }

  if(found == true){

    index = i;

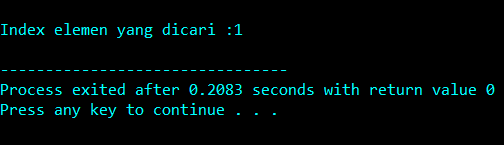
  }

  else index = 0; //jika tidak ditemukan index = 0

  cout<<"\nIndex elemen yang dicari :"<<index<<endl;

}

* Hasil screenshoot



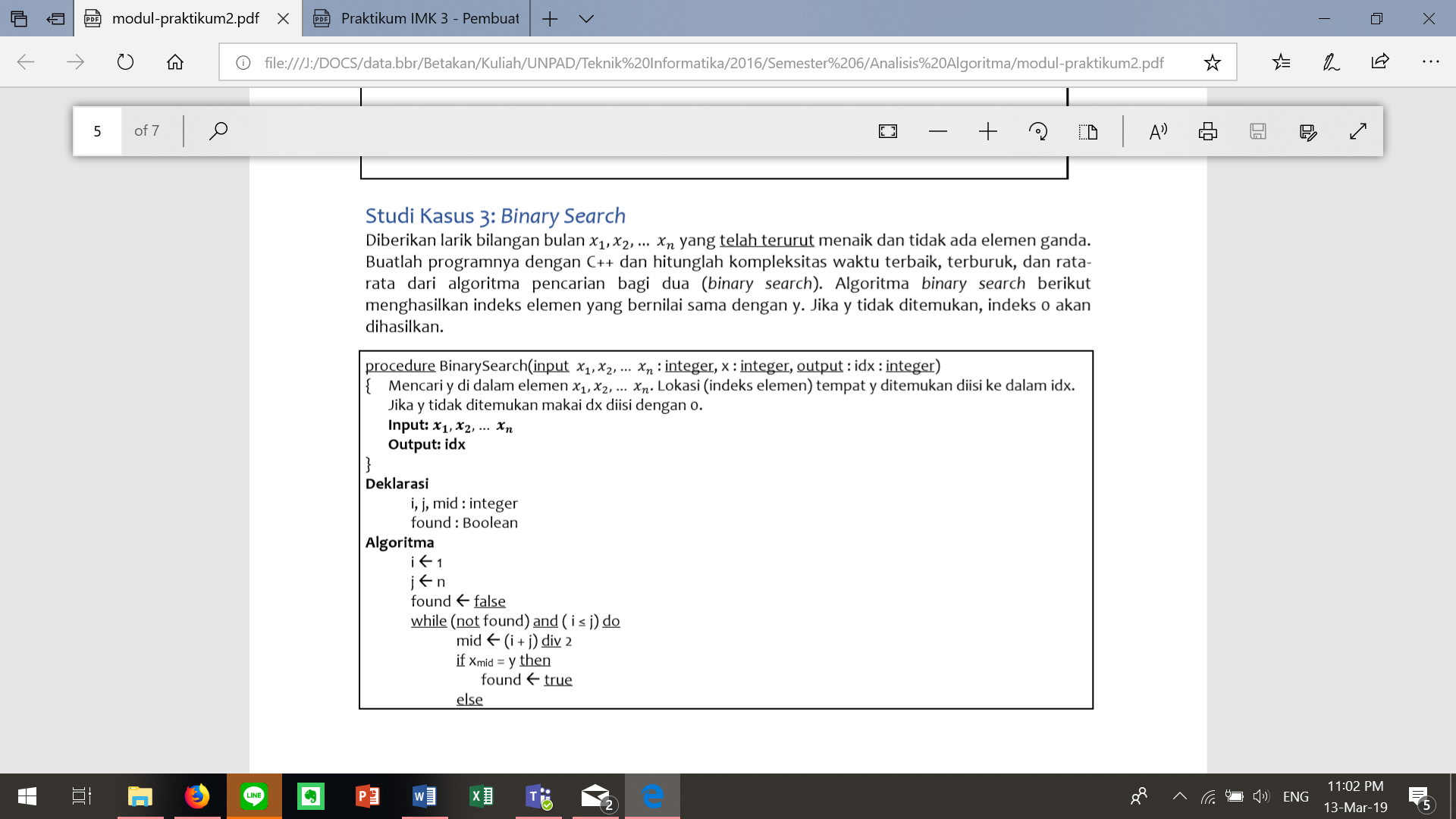
* Kompleksitas waktu:

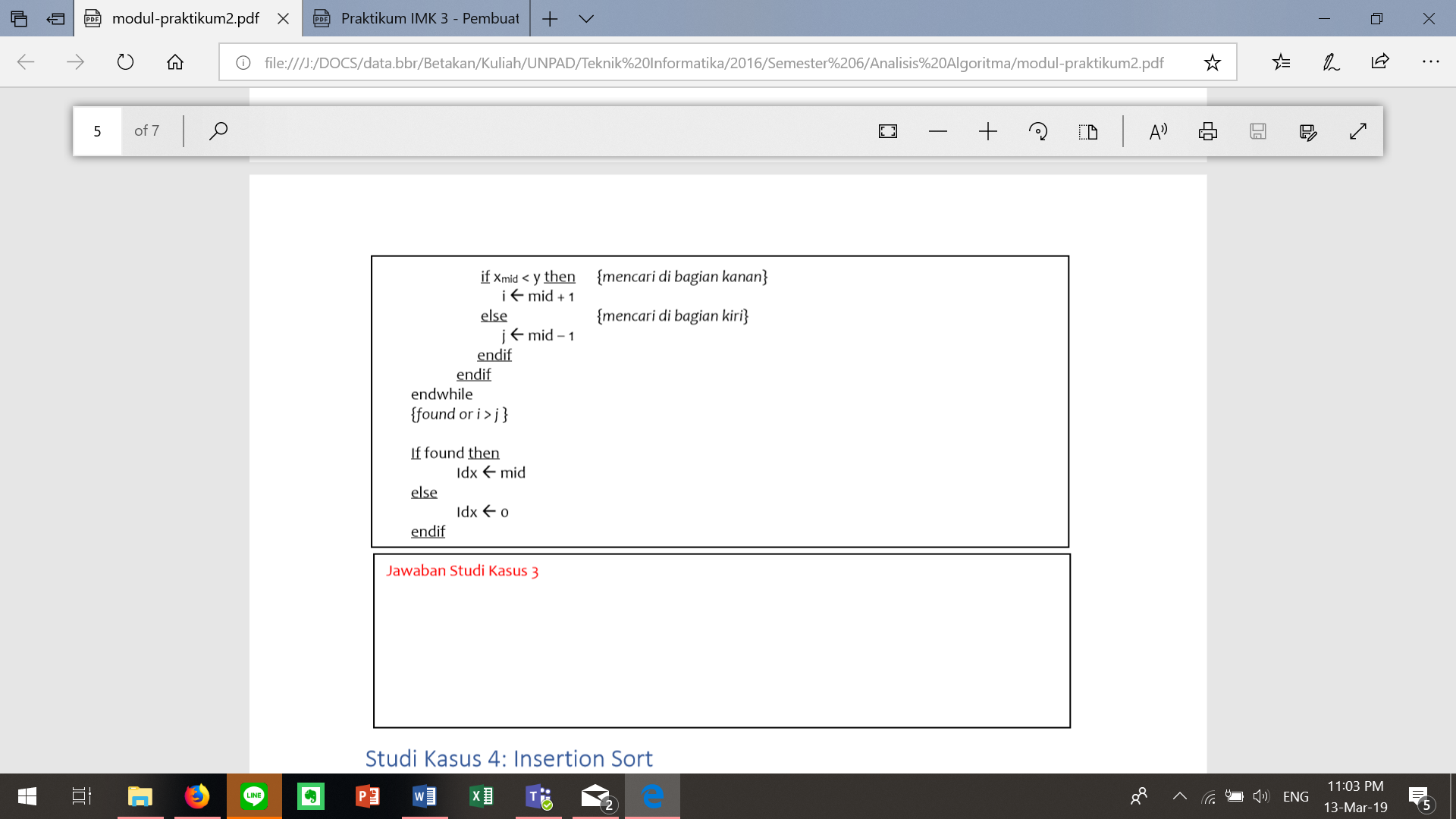
Best case: Jika ditemukan pada arr[0] atau indeks paling awal

Average case: Jika ditemukan pada arr[(n-1)/2] atau indeks di tengah

Worst case: Jika ditemukan pada arr[n-1] atau indeks paling akhir atau tidak ditemukan sama sekali

**Nomor 3**





Jawab:

* Source code

#include <iostream>

using namespace std;

int main(){

    int arr[5] = {1,2,3,4,5};

    int i = 2;

    int j = sizeof(arr)/sizeof(arr[0]);

    int y = 4;

    int index, mid;

    bool found = false;

    while(!found && i <= j){

        mid  = (i + j)/2;

        if(arr[mid] == y){

            found = true;

        }

        else if(arr[mid]< y){

                i = mid + 1;

        }

        else {

            j = mid - 1;

        }

    }

    if(found == true){

        index = mid;

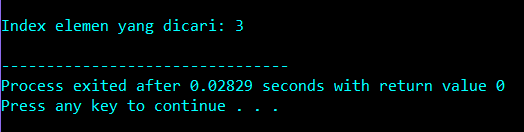
    }

    else index = 0;

    cout<<"\nIndex elemen yang dicari: "<<index<<endl;

}

* Hasil screenshoot



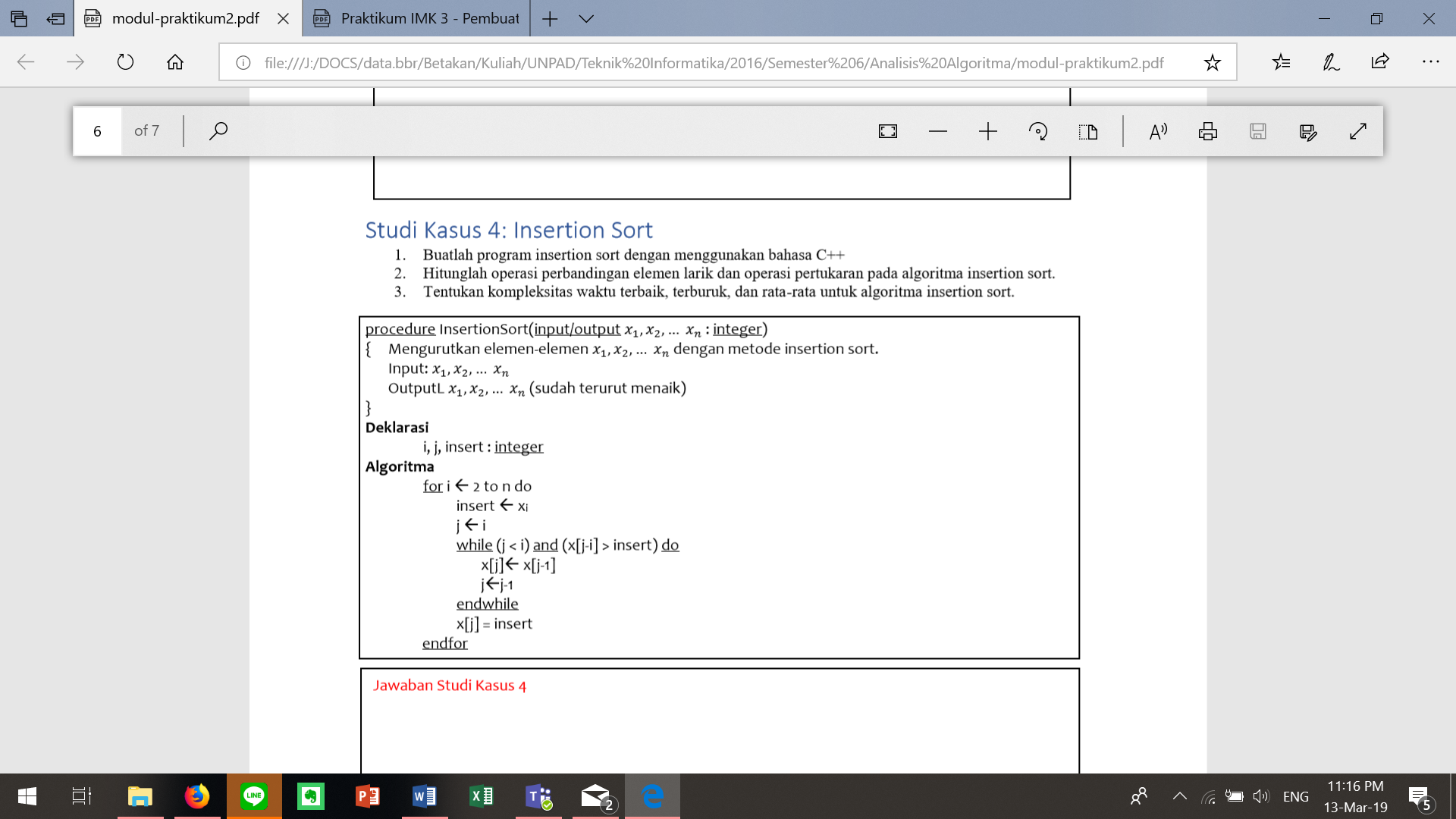
* Kompleksitas waktu:

Best case: Jika ditemukan pada arr[mid] atau indeks di tengah

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

Worst case: Jika tidak ditemukan sama sekali

**Nomor 4**



Jawab:

* Source code

#include <iostream>

using namespace std;

int main(){

    int arr[5] = {4,5,1,2,3};

    int panjangArr = sizeof(arr)/sizeof(\*arr);

    int i, j, insert;

    for (i = 1; i < panjangArr; i++){

        insert = arr[i];

        j = i - 1;

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

            arr[j+1] = arr[j];

            j = j - 1;

        }

        arr[j+1] = insert;

    }

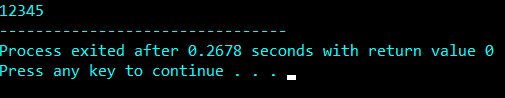
    for (j = 0; j < panjangArr; j++){

        cout<<arr[j];

    }

}

* Hasil screenshot



* Kompleksitas waktu:

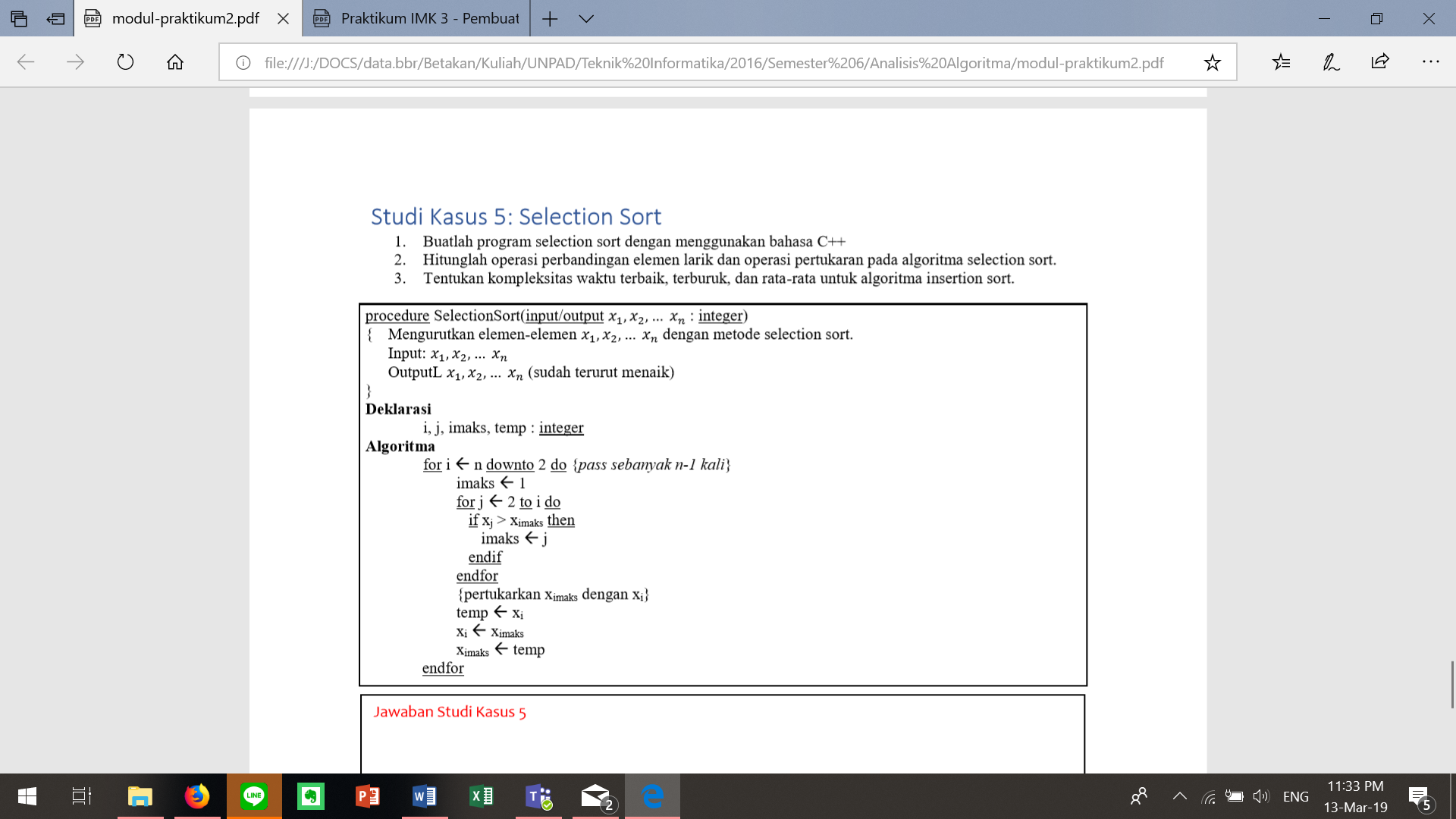
Best case: Jika array sudah terurut sehingga loop while tidak dijalankan

Average case: Jika sebagian elemen array sudah terurut

Worst case: Jika array harus diurutkan sebanyak n kali

|  |  |  |  |
| --- | --- | --- | --- |
| 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)** |

**Nomor 5**



Jawab:

* Source code

#include <iostream>

using namespace std;

int main(){

    int arr[5] = {5,2,4,3,1};

    int panjangArr = sizeof(arr)/sizeof(\*arr);

    for(int i = 0; i < panjangArr - 1; i++){

        int minIdx = i;

        for(int j = i + 1; j < panjangArr; j++){

            if(arr[j] < arr[minIdx]) minIdx = j;

        }

        int temp = arr[minIdx];

        arr[minIdx] = arr[i];

        arr[i] = temp;

    }

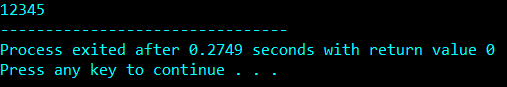
    for(int i = 0; i < panjangArr; i++){

        cout<<arr[i];

    }

}

* Hasil screenshoot



* 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.