LAPORAN PRAKTIKUM 2
ANALISIS ALGORITMA

KOMPLEKSITAS WAKTU DARI ALGORITMA

FAUZAN AKMAL HARIZ
140810180005
KULIAH: KELAS A
PRAKTIKUM: KELAS B

PROGRAM STUDI S-1 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, *n*.
   Contoh algoritma pengurutan elemen-elemen larik, *n* adalah jumlah elemen larik. Sedangkan dalam algoritma perkalian matriks n adalah ukuran matriks *n x n*.
2. Kompleksitas waktu, *n* (*n*), adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari input *n*.
3. Kompleksitas ruang, *n* (*n*), adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari input *n*.

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 + a_i
                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 sengan cra menghitung masing-masing jumlah operasi. Jika operasi tersebut berada di sebuah loop, maka jumlah operasinya bergantung berapa kali loop tersebut diulangi.

(i) 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

$t1 = 1 + 1 + n + n + 1 = 3 + 2n$

(ii) Operasi penjumlahan

Jumlah + ak,	n kali
k+1,	n kali

Jumlah seluruh operasi penjumlahan adalah

$t2 = n + n = 2\ n$

(iii) 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:

$T(n) = t1 + t2 + t3 = 3 + 2n + 2n + 1 = 4n + 4$

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 ← x1
        i ← 2
        while i ≤ n do
                if xi > maks then
                        maks ← xi
                endif
                i ← i + 1
        endwhile
```

Analisis:

(i) Operasi pengisian nilai (assignment)

maks ← x1	1 kali
i ← 2	1 kali
maks ← xi	n kali
i ← i + 1	n kali

Jumlah seluruh operasi assignment adalah $t1 = 1 + 1 + n + n = 2 + 2n$

(ii) Operasi penjumlahan

i ← i + 1	n kali

Jumlah seluruh operasi penjumlahan adalah $t2 = n$

(iii) Operasi perbandingan

if xi > maks then	n kali

Jumlah seluruh operasi penjumlahan adalah $t3 = n$

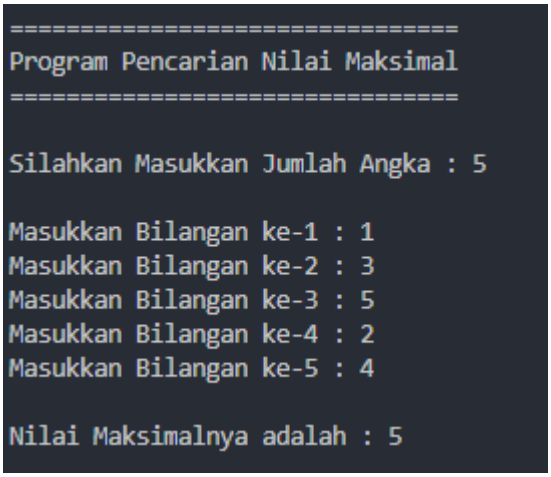
Jadi, kompleksitas waktu algoritma adalah:

$T(n) = t1 + t2 + t3 = 2 + 2n + n + n = 2 + 4n$

Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   10 Maret 2020
Praktikum       :   Analisis Algoritma
Program         :   studiKasus_1
Deskripsi       :   Studi Kasus 1 - Pencarian Nilai Maksimal
*/

#include <iostream>

using namespace std;

int main(){

    int x[100];
    int n;
    int i;
    int maks;

    cout << "\n================================\n";
    cout << "Program Pencarian Nilai Maksimal\n";
    cout << "================================\n";

    cout << "\nSilahkan Masukkan Jumlah Angka : "; cin >> n; cout << endl;
    for(i=0; i<n; i++){
        cout << "Masukkan Bilangan ke-" << i+1 << " : "; cin >> x[i];
    }

    maks = x[0];
    i = 2;

    for(int i=0; i<n; i++){
        if(x[i] > maks)
            maks = x[i];
    }

    while (i<n){
        if(x[i] > maks){
            maks = x[i];
        }
        i=i+1;
    }
    cout << "\nNilai Maksimalnya adalah : " << maks << endl;
}
```

Screenshot Program:



PEMBAGIAN KOMPLEKSITAS WAKTU

Hal lain yang harus diperhatikan dalam menghitung kompleksitas waktu suatu algoritma adalah parameter yang mencirikan ukuran input. Contoh pada algoritma pencarian, waktu yang dibutuhkan untuk melakukan pencarian tidak hanya bergantung pada ukuran larik (n) saja, tetapi juga bergantung pada nilai elemen (x) yang dicari.
Misalkan:

- Terdapat sebuah larik dengan panjang elemen 130 dimulai dari *y1*, *y2*, … *yn*.
- Asumsikan elemen-elemen larik sudah terurut. Jika $y1 = x$, maka waktu pencariannya lebih cepat 130 kali dari pada $y_{130} = x$ atau x tidak ada di dalam larik.
- Demikian pula, jika $y_{65} = x$, maka waktu pencariannya ½ kali lebih cepat daripada $y_{130} = x$

  Oleh karena itu, kompleksitas waktu dibedakan menjadi 3 macam:

  (1) $Tmin(n)$: kompleksitas waktu untuk kasus terbaik (best case) merupakan kebutuhan waktu minimum yang diperlukan algoritma sebagai fungsi dari n.

  (2) $Tavg(n)$: kompleksitas waktu untuk kasus rata-rata (average case) merupakan kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari n. Biasanya pada kasus ini dibuat asumsi bahwa semua barisan input bersifat sama. Contoh pada kasus searching diandaikan data yang dicari mempunyai peluang yang sama untuk tertarik dari larik.

  (3) $Tmax(n)$: kompleksitas waktu untuk kasus terburuk (worst case) merupakan kebutuhan waktu maksimum yang diperlukan algoritma sebagai fungsi dari.

Studi Kasus 2: Sequential Search

Diberikan larik bilangan bulat *x1,x2, ... xn* yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan ratarata 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 ← 1
        found ← false
        while (i ≤ n) and (not found) do
                if xi = y then
                        found ← 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
```

Analisis:

- Kasus terbaik (best case):
  Ini terjadi bila $x_1 = y$.
  $T_{min}(n) = 1$
- Kasus rata-rata (average case):
  Ini terjadi jika x ditemukan pada posisi ke-j, maka operasi perbandingan ($a_k = y$) akan dieksekusi sebanyak j kali.
  $T_{avg}(n) = \frac{(1+2+3+\cdots+n)}{n} = \frac{\frac{1}{2}n(1+n)}{n} = \frac{(n+1)}{2}$
- Kasus terburuk (worst case):
  Ini terjadi jika $x_n = y$ atau y tidak ditemukan.
  $T_{max}(n) = n$

Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   10 Maret 2020
Mata Kuliah     :   Analisis Algoritma
Program         :   studiKasus_2
Deskripsi       :   Studi Kasus 2 - Sequential Search
*/

#include <iostream>

using namespace std;

int main(){

    int n;
    int A[100];
    int cari;
    int index;
    bool found = false;

    cout << "\n=========================\n";
    cout << "Program Sequential Search\n";
    cout << "=========================\n";

    cout << "\nSilahkan Masukkan Banyak Data :  "; cin >> n; cout <<endl;

    for(int i=0; i<n; i++){
        cout << "Masukkan Data ke-" << i+1 << " : "; cin >> A[i];
```

```
    }

    cout << "\nMasukkan Data yang Dicari : "; cin >> cari; cout << endl;

    for(int i=0; i<n; i++){
        if(A[i] == cari){
            found = true;
            index = i;
            i = n;
        }
    }

    if(found == true){
        cout << "Data Ditemukan Pada Data ke-" << index+1 << endl;
    }
    else{
        cout << "Maaf, Data yang Anda Cari Tidak Ditemukan!\n";
    }

    return 0;
}
```

Screenshot Program:

```
========================
Program Sequential Search
========================

Silahkan Masukkan Banyak Data :  5

Masukkan Data ke-1 : 1
Masukkan Data ke-2 : 3
Masukkan Data ke-3 : 5
Masukkan Data ke-4 : 2
Masukkan Data ke-5 : 4

Masukkan Data yang Dicari : 5

Data Ditemukan Pada Data ke-3
```

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 ← 1
        j ← n
        found ← false
        while (not found) and ( i ≤ j) do
                mid ← (i + j) div 2
                if xmid = y then
                        found ← true
                else
                        if xmid < 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
```

Analisis:

- Kasus terbaik (best case):
  Ini terjadi jika ditemukan pada arr[mid] atau indeks di tengah.
  $T_{min}(n) = 1$
- Kasus rata-rata (average case):
  Ini terjadi jika ditemukan pada indeks di awal atau di akhir elemen.
- Kasus terburuk (worst case):
  Ini terjadi jika tidak ditemukan sama sekali.
  $T_{max}(n) = {}^{2}\log n$

Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   10 Maret 2020
Mata Kuliah     :   Analisis Algoritma
Program         :   studiKasus_3
Deskripsi       :   Studi Kasus 3 - Binary Search
*/

#include <iostream>

using namespace std;

int main(){

    int n;
    int arr[100];
    int cari;
    int awal;
    int akhir;
    int tengah;

    cout << "\n=====================\n";
    cout << "Program Binary Search\n";
    cout << "=====================\n";

    cout << "\nSilahkan Masukkan Banyak Data : "; cin >> n; cout << endl;
    for (int i=0; i<n; i++){
        cout << "Masukkan Data ke-" << i+1 << " : "; cin >> arr[i];
    }

    cout << "\nMasukkan Data yang Dicari : "; cin >> cari; cout << endl;
```

```
    awal = 0;
    akhir = n-1;
    while (awal <= akhir){
        tengah = (awal+akhir)/2;
        if(arr[tengah] < cari){
            awal = tengah + 1;
        }
        else if(arr[tengah] == cari){
            cout << "Data Ditemukan Pada Data ke-" << tengah+1 << endl;
            break;
        }
        else{
            akhir = tengah - 1;
        }
        tengah = (awal + akhir)/2;
    }
    if(awal > akhir){
        cout << "Maaf, Data yang Anda Cari Tidak Ditemukan!\n";
    }

    return 0;
}
```

Screenshot Program:

```
====================
Program Binary Search
====================

Silahkan Masukkan Banyak Data : 5

Masukkan Data ke-1 : 1
Masukkan Data ke-2 : 3
Masukkan Data ke-3 : 5
Masukkan Data ke-4 : 2
Masukkan Data ke-5 : 4

Masukkan Data yang Dicari : 5

Data Ditemukan Pada Data ke-3
```

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
```

Analisis:

- Kasus terbaik (best case):
  Ini terjadi jika array sudah terurut sehingga loop while tidak dijalankan
  $T_{min}(n) = 1$
- Kasus rata-rata (average case):
  Ini terjadi jika sebagian elemen array sudah terurut
- Kasus terburuk (worst case):
  Ini terjadi 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*).

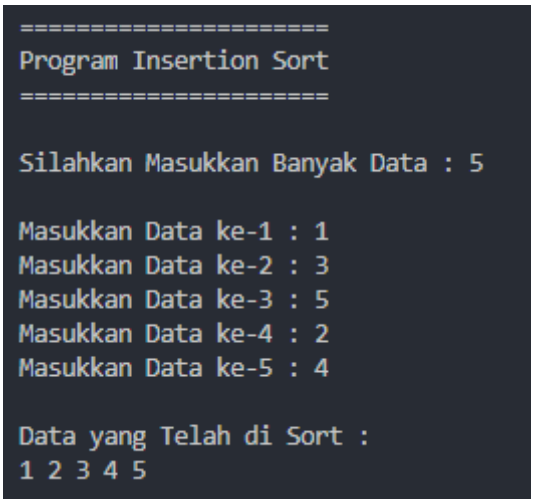
J	Perbandingan	Perpindahan	Total Operasi
2	1	1	2
3	2	2	4
4	3	3	6
n	(n-1)	(n-1)	**2(n-1)**

Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   10 Maret 2020
Mata Kuliah     :   Analisis Algoritma
Program         :   studiKasus_4
Deskripsi       :   Studi Kasus 4 - Insertion Sort
*/

#include <iostream>

using namespace std;

int data1[100];
int data2[100];
int n;

void insertion_sort(){

    int temp;

    for(int i=1; i<=n; i++){
        temp = data1[i];
        int j = i-1;
        while(data1[j]>temp && j>=0){
            data1[j+1] = data1[j];
            j--;
        }
        data1[j+1] = temp;
    }
}

int main(){

    cout << "\n======================\n";
    cout << "Program Insertion Sort\n";
    cout << "======================\n";

    cout << "\nSilahkan Masukkan Banyak Data : "; cin >> n; cout <<endl;

    for(int i=1; i<=n; i++){
        cout << "Masukkan Data ke-" << i << " : "; cin >> data1[i];
        data2[i]=data1[i];
    }
```

```
    insertion_sort();
    cout << "\nData yang Telah di Sort : " << endl;
    for(int i=1; i<=n; i++){
        cout << data1[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Screenshot Program:



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
        Output: 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
```

Analisis:

(i) 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 = $\frac{n(n-1)}{2}$ 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.

(ii) Jumlah operasi pertukaran

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

Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   10 Maret 2020
Mata Kuliah     :   Analisis Algoritma
Program         :   studiKasus_5
Deskripsi       :   Studi Kasus 5 - Selection Sort
*/

#include <iostream>

using namespace std;

int data1[100];
int data2[100];
int n;

void tukar(int a, int b){
    int t;
    t = data1[b];
    data1[b] = data1[a];
    data1[a] = t;
}

void selection_sort(){
    int pos;
    int i;
    int j;
    for(i=1; i<=n-1; i++){
        pos = i;
        for(j = i+1;j<=n;j++){
            if(data1[j] < data1[pos]){
                pos = j;
            }
        }
        if(pos != i){
            tukar(pos, i);
        }
    }
}

int main(){

    cout << "\n======================\n";
    cout << "Program Selection Sort\n";
```

```
    cout << "======================\n";

    cout << "\nSilahkan Masukkan Banyak Data : "; cin >> n; cout <<endl;

    for(int i=1; i<=n; i++){
        cout << "Masukkan Data ke-" << i << " : "; cin >> data1[i];
        data2[i]=data1[i];
    }

    selection_sort();
    cout << "\nData yang Telah di Sort : " << endl;
    for(int i=1; i<=n; i++){
        cout << data1[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Screenshot Program:

```
======================
Program Selection Sort
======================

Silahkan Masukkan Banyak Data : 5

Masukkan Data ke-1 : 1
Masukkan Data ke-2 : 3
Masukkan Data ke-3 : 5
Masukkan Data ke-4 : 2
Masukkan Data ke-5 : 4

Data yang Telah di Sort :
1 2 3 4 5
```

TEKNIK PENGUMPULAN

- Lakukan push ke github/gitlab untuk semua program dan laporan hasil analisa yang berisi jawaban dari pertanyaan-pertanyaan yang diajukan. Silahkan sepakati dengan asisten praktikum.

PENUTUP

- Ingat, berdasarkan Peraturan Rektor No 46 Tahun 2016 tentang Penyelenggaraan Pendidikan, mahasiswa wajib mengikuti praktikum 100%.
- Apabila tidak hadir pada salah satu kegiatan praktikum segeralah minta tugas pengganti ke asisten praktikum.
- Kurangnya kehadiran Anda di praktikum, memungkinkan nilai praktikum Anda tidak akan dimasukkan ke nilai mata kuliah.

REFERENSI

PPT Praktikum Analisis Algortima (Pertemuan 2)
Modul Praktikum 2 Analisis Algoritma