

Tugas Praktikum Analisis Algoritma

Laporan ini ditujukan untuk memenuhi tugas praktikum analisis algoritma



Anugerah Prima Bagaskara

140810180006

Fakultas Matematika dan Ilmu

Pengetahuan Alam Universitas Padjadjaran

2019

MODUL PRAKTIKUM 4
REKURENSI DAN PARADIGMA ALGORITMA DIVIDE & CONQUER

MATA KULIAH
ANALISIS ALGORITMA
D10G.4205 & D10K.0400601



PENGAJAR : (1) MIRA SURYANI, S.Pd., M.Kom
(2) INO SURYANA, Drs., M.Kom
(3) R. SUDRAJAT, Drs., M.Si
FAKULTAS : MIPA
SEMESTER : IV dan VI

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
MARET 2019

Pendahuluan

PARADIGMA DIVIDE & CONQUER

Divide & Conquer merupakan teknik algoritmik dengan cara memecah input menjadi beberapa bagian, memecahkan masalah di setiap bagian secara **rekursif**, dan kemudian menggabungkan solusi untuk subproblem ini menjadi solusi keseluruhan. Menganalisis *running time* dari algoritma *divide & conquer* umumnya melibatkan penyelesaian rekurensi yang membatasi *running time* secara rekursif pada instance yang lebih kecil

PENGENALAN REKURENSI

- Rekurensi adalah persamaan atau ketidaksetaraan yang menggambarkan fungsi terkait nilainya pada input yang lebih kecil. Ini adalah fungsi yang diekspresikan secara rekursif
- Ketika suatu algoritma berisi panggilan rekursif untuk dirinya sendiri, *running time*-nya sering dapat dijelaskan dengan perulangan
- Sebagai contoh, *running time worst case* () dari algoritma merge-sort dapat dideskripsikan dengan perulangan:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

with solution $T(n) = \Theta(n \lg n)$.

BEDAH ALGORITMA MERGE-SORT

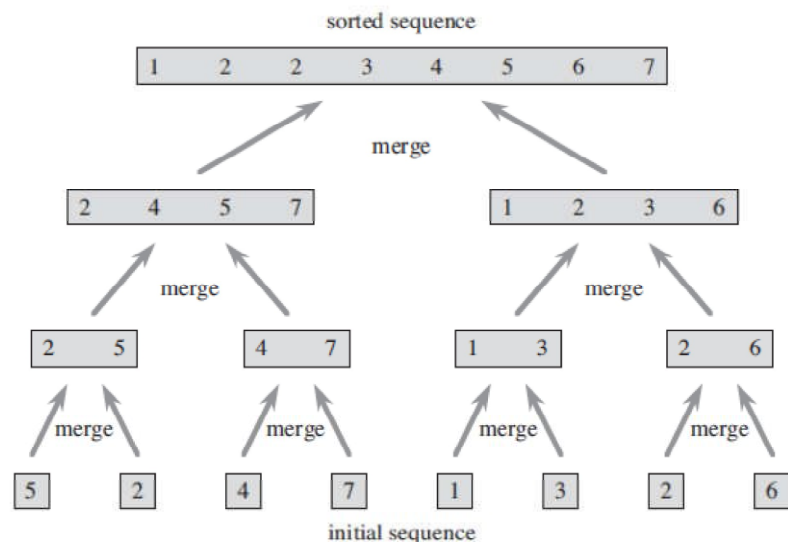
- Merupakan algoritma sorting dengan paradigma divide & conquer
- *Running time worst case*-nya mempunyai laju pertumbuhan yang lebih rendah dibandingkan insertion sort
- Karena kita berhadapan dengan banyak subproblem, kita notasikan setiap subproblem sebagai sorting sebuah subarray $A[p..r]$
- Inisialisasi, $p=1$ dan $r=n$, tetapi nilai ini berubah selama kita melakukan perulangan subproblem

Untuk mengurutkan $A[p..r]$:

- **Divide** dengan membagi input menjadi 2 subarray $A[p..q]$ dan $A[q+1 .. r]$
- **Conquer** dengan secara rekursif mengurutkan subarray $A[p..q]$ dan $A[q+1 .. r]$
- **Combine** dengan menggabungkan 2 subarray terurut $A[p..q]$ dan $A[q+1 .. r]$ untuk menghasilkan 1 subarray terurut $A[p..r]$
- Untuk menyelesaikan langkah ini, kita membuat prosedur $MERGE(A, p, q, r)$
- Rekursi berhenti apabila subarray hanya memiliki 1 elemen (secara trivial terurut)

PSEUDOCODE MERGE-SORT

```
> MERGE-SORT(A, p, r)
  //sorts the elements in the subarray A[p..r]
  1  if p < r
  2    then q ← ⌊(p + r)/2⌋
  3      MERGE-SORT(A, p, q)
  4      MERGE-SORT(A, q + 1, r)
  5      MERGE(A, p, q, r)
```



Gambar 1. Ilustrasi algoritma merge-sort

PROSEDUR MERGE

- Prosedur merge berikut mengasumsikan bahwa subarray $A[p..q]$ dan $A[q+1 .. r]$ berada pada kondisi terurut. Prosedur merge menggabungkan kedua subarray untuk membentuk 1 subarray terurut yang menggantikan array saat ini $A[p..r]$ (input).
- Ini membutuhkan waktu $\Theta(n)$, dimana $n = r-p+1$ adalah jumlah yang digabungkan
- Untuk menyederhanakan code, digunakanlah elemen sentinel (dengan nilai ∞) untuk menghindari keharusan memeriksa apakah subarray kosong di setiap langkah dasar.

PSEUDOCODE PROSEDUR MERGE

MERGE(A, p, q, r)

1. $n_1 \leftarrow q - p + 1$; $n_2 \leftarrow r - q$
2. //create arrays $L[1 .. n_1 + 1]$ and $R[1 .. n_2 + 1]$
3. for $i \leftarrow 1$ to n_1 do $L[i] \leftarrow A[p + i - 1]$
4. for $j \leftarrow 1$ to n_2 do $R[j] \leftarrow A[q + j]$
5. $L[n_1 + 1] \leftarrow \infty$; $R[n_2 + 1] \leftarrow \infty$
6. $i \leftarrow 1$; $j \leftarrow 1$
7. for $k \leftarrow p$ to r
8. do if $L[i] \leq R[j]$
9. then $A[k] \leftarrow L[i]$
10. $i \leftarrow i + 1$
11. else $A[k] \leftarrow R[j]$
12. $j \leftarrow j + 1$

RUNNING TIME MERGE

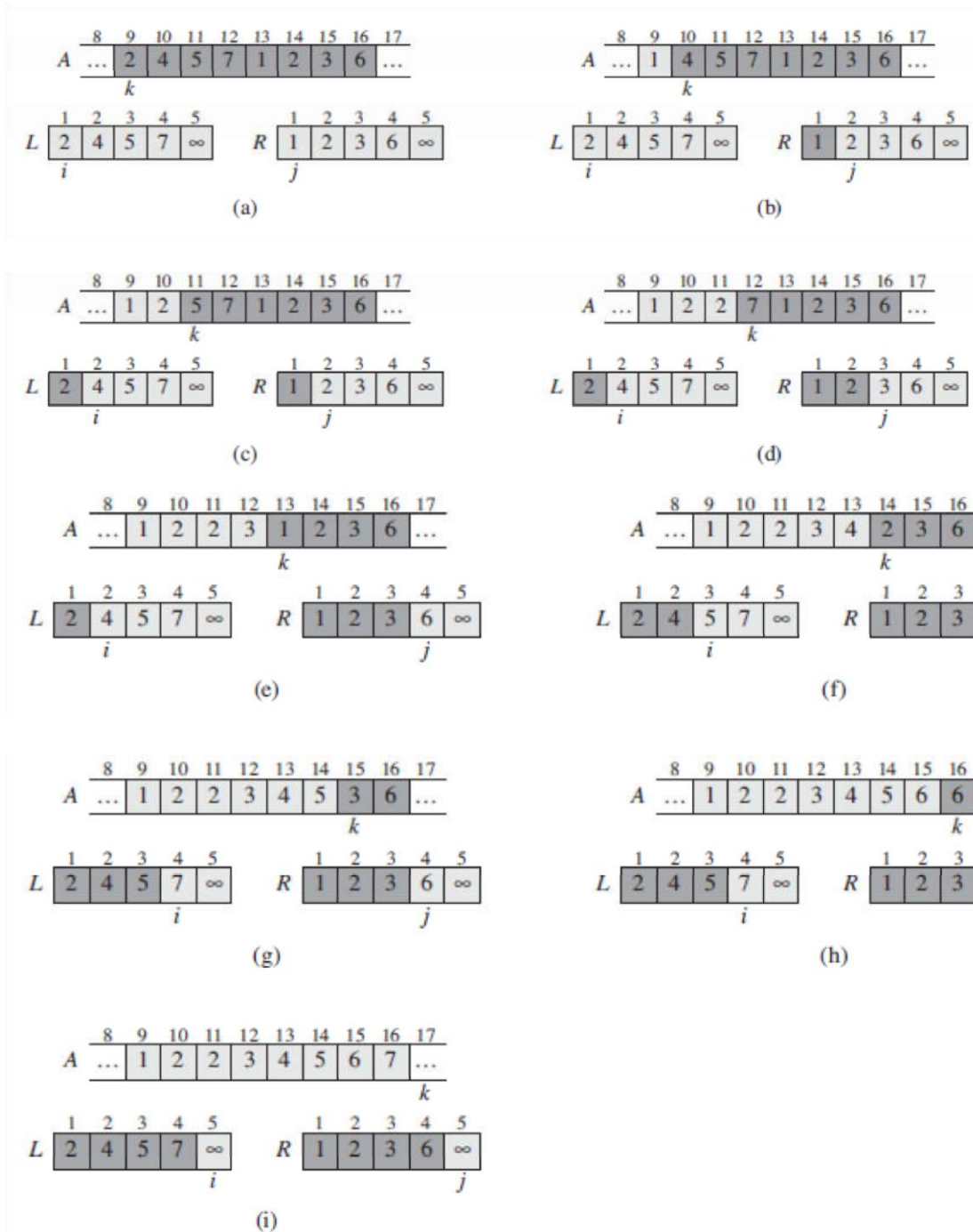
Untuk melihat running time prosedur MERGE berjalan di $\Theta(n)$, dimana $n-p+1$, perhatikan perulangan for pada baris ke 3 dan 4,

$$\Theta(1 + 2) = \Theta()$$

dan ada sejumlah n iterasi pada baris ke 8-12 yang membutuhkan waktu konstan.

CONTOH SOAL MERGE-SORT

MERGE(A, 9, 12, 16), dimana subarray A[9 .. 16] mengandung sekuen (2,4,5,7,1,2,3,6)



Algoritma merge-sort sangat mengikuti paradigma divide & conquer:

- **Divide** problem besar ke dalam beberapa subproblem
- **Conquer** subproblem dengan menyelesaikannya secara **rekursif**. Namun, apabila subproblem berukuran kecil, diselesaikan saja secara langsung.
- **Combine** solusi untuk subproblem ke dalam solusi untuk original problem

Gunakan sebuah persamaan rekurensi (umumnya sebuah perulangan) untuk mendeskripsikan running time dari algoritma berparadigma divide & conquer.

$T(n)$ = running time dari sebuah algoritma berukuran n

- Jika ukuran problem cukup kecil (misalkan $\leq c$, untuk nilai c konstan), kita mempunyai *best case*. Solusi brute-force membutuhkan waktu konstan $\Theta(1)$
- Sebaiknya, kita membagi input ke dalam sejumlah subproblem, setiap $(1/b)$ dari ukuran original problem (Pada merge sort $b = 2$)
- Misalkan waktu yang dibutuhkan untuk membagi ke dalam n -ukuran problem adalah $\Theta(n)$
- Ada sebanyak n/b subproblem yang harus diselesaikan, setiap subproblem (n/b) \rightarrow setiap subproblem membutuhkan waktu $T(n/b)$ sehingga kita menghabiskan $\Theta(n \cdot T(n/b))$
- Waktu untuk **combine** solusi kita misalkan $\Theta(n)$
- Maka persamaan **rekurensinya untuk divide & conquer** adalah:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ \Theta(n) + b \cdot T(n/b) & \text{otherwise} \end{cases}$$

Setelah mendapatkan rekurensi dari sebuah algoritma divide & conquer, selanjutnya rekurensi harus diselesaikan untuk dapat menentukan kompleksitas waktu asimptotiknya. Penyelesaian rekurensi dapat menggunakan 3 cara yaitu, **metode substitusi**, **metode recursion-tree** dan **metode master**. Ketiga metode ini dapat dilihat pada slide yang diberikan.

Studi Kasus

Studi Kasus 1: MERGE SORT

Setelah Anda mengetahui Algoritma Merge-Sort mengadopsi paradigma divide & conquer, lakukan Hal berikut:

1. Buat program Merge-Sort dengan bahasa C++
2. Kompleksitas waktu algoritma merge sort adalah $O(n \lg n)$. Cari tahu kecepatan komputer Anda dalam memproses program. Hitung berapa running time yang dibutuhkan apabila input untuk merge sort-nya adalah 20?

```
/*
 * Nama   : Anugerah Prima Bagaskara
 * NPM    : 140810180006
 * Kelas  : B
 * Program : Pengurut bilangan dalam array integer.
 * Tanggal : 24 Maret 2020
 */
```

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

```
void merge(int arr[], int p, int q, int r){
    int n1 = q - p + 1;
    int n2 = r - q;
    int L[n1+1];
    int R[n2+1];

    for (int i = 0; i < n1; i++)
```

```

    L[i] = arr[(p - 1) + i];

for (int j = 0; j < n2; j++)
    R[j] = arr[q + j];

L[n1] = 2147483647;
R[n2] = 2147483647;
int i = 0;
int j = 0;

for (int k = (p-1); k < r; k++){
    if (L[i] <= R[j]){
        arr[k] = L[i];
        i++;
    }
    else {
        arr[k] = R[j];
        j++;
    }
}

}

void mergeSort(int arr[], int p, int r){
    int q;
    if (p < r) {
        q = (p + r)/2;

        mergeSort(arr, p, q);
        mergeSort(arr, q + 1, r);

        merge(arr, p, q, r);
    }
}

void printArray(int arr[], int size) {
    for (int i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[10];

    for (int i = 0; i < 10; i++) {
        cout << "arr[" << i << "] = ";
        cin >> arr[i];
    }

    cout << endl << "array sebelum disort : ";
    printArray(arr, 10);

```

```

cout << endl << "array setelah disort : ";

auto start = chrono::steady_clock::now();
mergeSort(arr, 1, 10);
auto end = chrono::steady_clock::now();
auto diff = end - start;
printArray(arr, 10);

cout << endl << "waktu yang dibutuhkan komputer : "
    << chrono::duration <double, nano> (diff).count()
    << " nanosekon." << endl;
}

```

Output : 0 nanosekon

Studi Kasus 2: SELECTION SORT

Selection sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma selection sort, lakukan langkah-langkah berikut:

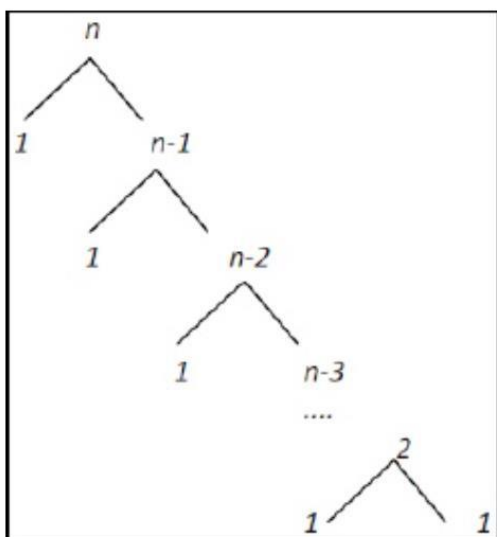
- Pelajari cara kerja algoritma selection sort
- Tentukan $T(n)$ dari rekurensi (pengulangan) selection sort berdasarkan penentuan rekurensi divide & conquer:

(1) if \leq — + () + () otherwise

() =

- Selesaikan persamaan rekurensi $T(n)$ dengan **metode recursion-tree** untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big- Ω , dan Big- Θ
- Lakukan implementasi coding program untuk algoritma selection sort dengan menggunakan bahasa C++

$$T(n) = \theta(1) T(n-1) + \theta(n)$$



$$\begin{aligned}
 T(n) &= cn + (cn-c) + (cn-2c) + \dots + 2c + cn \\
 &= c((n-1)(n-2)/2) + cn \\
 &= c((n^2-3n+2)/2) + cn
 \end{aligned}$$

$$= c((n^2)/2) - (3n/2) + 1 + cn$$

$$= O(n^2)$$

$$T(n) = cn + (cn - c) + (cn - 2c) + \dots + 2c + cn$$

$$= c((n-1)(n-2)/2) + cn$$

$$= c((n^2 - 3n + 2)/2) + cn$$

$$= c((n^2)/2) - (3n/2) + 1 + cn$$

$$= \Omega(n^2)$$

$$T(n) = cn^2$$

$$= \Theta(n^2)$$

Studi Kasus 3: INSERTION SORT

Insertion sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma insertion sort, lakukan langkah-langkah berikut:

- Pelajari cara kerja algoritma insertion sort
- Tentukan $T(n)$ dari rekurensi (pengulangan) insertion sort berdasarkan penentuan rekurensi divide & conquer:

$$(1) \text{ if } \leq \quad - \quad + \quad () + () \quad \text{otherwise}$$

$$() =$$

- Selesaikan persamaan rekurensi $T(n)$ dengan **metode substitusi** untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ
- Lakukan implementasi koding program untuk algoritma insertion sort dengan menggunakan bahasa C++

$$T(n) = \{ \theta(1) T(n-1) + \theta(n) \}$$

$$T(n) = cn + cn - c + cn - 2c + \dots + 2c + cn \leq 2cn^2 + cn^2$$

$$= c((n-1)(n-2)/2) + cn \leq 2cn^2 + cn^2$$

$$= c((n^2 - 3n + 2)/2) + cn \leq 2cn^2 + cn^2$$

$$= c((n^2)/2) - c(3n/2) + c + cn \leq 2cn^2 + cn^2$$

$$= O(n^2)$$

$$T(n) = cn \leq cn$$

$$= \Omega(n)$$

$$T(n) = (cn + cn^2)/n$$

$$= \Theta(n)$$

```
/*
* Nama   : Anugerah Prima Bagaskara
* NPM    : 140810180006
* Kelas  : B
* Tanggal : 24 Maret 2020
*/
```

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

```

void swap(int& x, int& y) {
    int temp = x;
    x = y;
    y = temp;
}

void insertionSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int j = i + 1;

        if (arr[i] > arr[j]) {
            swap(arr[i], arr[j]);

            for (; i > 0; i--) {
                if (arr[i-1] > arr[i])
                    swap(arr[i-1], arr[i]);
                else
                    break;
            }
        }
    }
}

void printArray(int arr[], int size) {
    for (int i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[10];

    for (int i = 0; i < 10; i++) {
        cout << "arr[" << i << "] = ";
        cin >> arr[i];
    }

    cout << endl << "array sebelum disort : ";
    printArray(arr, 10);

    cout << endl << "array setelah disort : ";
    insertionSort(arr, 10);
    printArray(arr, 10);
}

```

Studi Kasus 4: BUBBLE SORT

Bubble sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma bubble sort, lakukan langkah-langkah berikut:

- Pelajari cara kerja algoritma bubble sort

- Tentukan $T(n)$ dari rekurensi (pengulangan) insertion sort berdasarkan penentuan rekurensi divide & conquer:

$$(1) \text{ if } \leq \quad - \quad + \quad () + () \quad \text{otherwise}$$

$$() =$$

- Selesaikan persamaan rekurensi $T(n)$ dengan **metode master** untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ
- Lakukan implementasi koding program untuk algoritma bubble sort dengan menggunakan bahasa C++

$$T(n) = \theta(1) T(n-1) + \theta(n)$$

$$T(n) = cn + cn - c + cn - 2c + \dots + 2c + c \leq 2cn^2 + cn^2$$

$$= c((n-1)(n-2)/2) + c \leq 2cn^2 + cn^2$$

$$= c((n^2 - 3n + 2)/2) + c \leq 2cn^2 + cn^2$$

$$= c((n^2)/2) - c(3n/2) + 2c \leq 2cn^2 + cn^2$$

$$= O(n^2)$$

$$T(n) = cn + cn - c + cn - 2c + \dots + 2c + c \leq 2cn^2 + cn^2$$

$$= c((n-1)(n-2)/2) + c \leq 2cn^2 + cn^2$$

$$= c((n^2 - 3n + 2)/2) + c \leq 2cn^2 + cn^2$$

$$= c((n^2)/2) - c(3n/2) + 2c \leq 2cn^2 + cn^2$$

$$= \Omega(n^2)$$

$$T(n) = cn^2 + cn^2$$

$$= \Theta(n^2)$$

```
/*
 * Nama : Anugerah Prima Bagaskara
 * NPM : 140810180006
 * Kelas : B
 * Tanggal : 24 Maret 2020
 */
```

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

```
void swap(int& x, int& y) {
    int temp = x;
    x = y;
    y = temp;
}
```

```
void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
        }
    }
}
```

```

void printArray(int arr[], int size) {
    for (int i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[10];

    for (int i = 0; i < 10; i++) {
        cout << "arr[" << i << "] = ";
        cin >> arr[i];
    }

    cout << endl << "array sebelum disort : ";
    printArray(arr, 10);

    cout << endl << "array setelah disort : ";
    bubbleSort(arr, 10);
    printArray(arr, 10);
}

```

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.