

Praktikum **Algoritma dan Pemrograman II**

# Kompleksitas Waktu dan Searching Algorithm

5002221053 - Putri Ghaida Tsuroyya

5002221072 - Mohammad Febryan Khamim

Kelompok 5

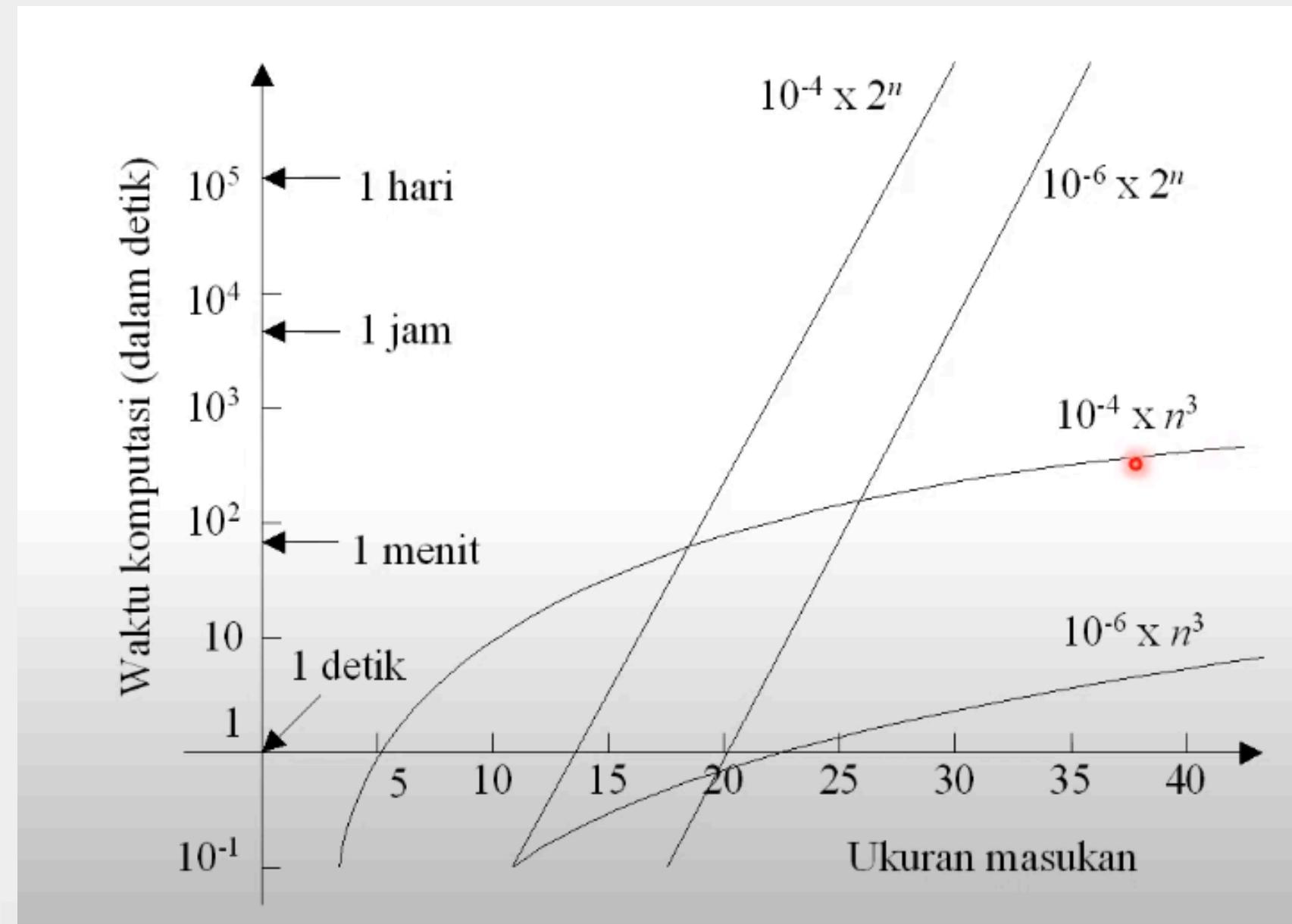


# Outline Pembahasan

- Kompleksitas Waktu
- Searching Algorithm

# Kompleksitas Waktu

# Mengapa algoritma harus efisien?



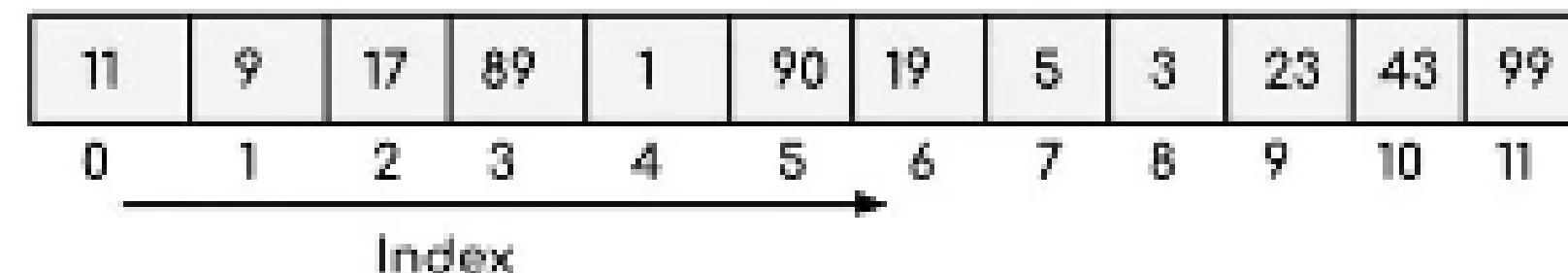
**Sumber :** Rinaldi Munir on [YouTube](#)

# Kompleksitas Waktu

**Kompleksitas waktu** dalam program diukur dari jumlah tahapan komputasi yang dilakukan dalam algoritma dengan sebuah masukan  $n$

**Contoh 1.** Tinjau algoritma menghitung rerata elemen di dalam sebuah larik (array).

```
sum ← 0  
for i ← 1 to n do  
    sum ← sum + a[i]  
endfor  
rata_rata ← sum/n
```



# Macam Kompleksitas Waktu

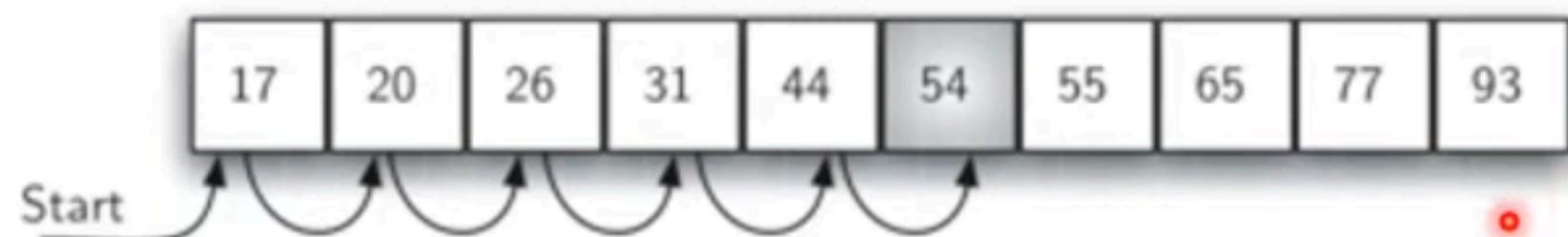
**BEST CASE**

**AVERAGE CASE**

**BEST CASE**

# CONTOH

**procedure** PencarianBeruntun(**input**  $a_1, a_2, \dots, a_n : \text{integer}$ ,  $x : \text{integer}$ , **output**  $idx : \text{integer}$ )  
 { Mencari elemen  $x$  di dalam larik  $A$  yang berisi  $n$  elemen. Jika  $x$  ditemukan, maka indeks elemen larik disimpan di dalam  $idx$ ,  $idx$  bernilai  $-1$  jika  $x$  tidak ditemukan }

**Deklarasi** $k : \text{integer}$  $\text{ketemu} : \text{boolean}$  { bernilai true jika  $x$  ditemukan atau false jika  $x$  tidak ditemukan }**Algoritma:** $k \leftarrow 1$  $\text{ketemu} \leftarrow \text{false}$ **while** ( $k \leq n$ ) and (not  $\text{ketemu}$ ) **do**  **if**  $a_k = x$  **then**     $\text{ketemu} \leftarrow \text{true}$   **else**     $k \leftarrow k + 1$   **endif****endwhile****if**  $\text{ketemu}$  **then** {  $x$  ditemukan }   $idx \leftarrow k$ **else**   $idx \leftarrow -1$  {  $x$  tidak ditemukan }**endif**

# JAWABAN

Jumlah operasi perbandingan elemen tabel:

1. *Kasus terbaik*: ini terjadi bila  $a_1 = x$ .

$$T_{\min}(n) = 1$$

2. *Kasus terburuk*: bila  $a_n = x$  atau  $x$  tidak ditemukan.

$$T_{\max}(n) = n$$

3. *Kasus rata-rata*: Jika  $x$  ditemukan pada posisi ke- $j$ , maka operasi perbandingan ( $a_k = x$ ) akan dicksekusi sebanyak  $j$  kali.

$$T_{\text{avg}}(n) = \frac{(1 + 2 + 3 + \dots + n)}{n} = \frac{\frac{1}{2}n(1+n)}{n} = \frac{(n+1)}{2}$$

# Big O Notation

Kelompok 5

# Big O Notation

**Big O Notation** adalah sebuah notasi matematika yang digunakan untuk menentukan batasan atas waktu yang dibutuhkan oleh sebuah algoritma atau struktur data.

Mengapa harus menggunakan **big O notation**?

- Fokus pada pertumbuhan kompleksitas algoritma
- Mengabaikan faktor konstan dan variabel lingkungan
- Tidak menunjukkan kecepatan absolut

# Big O Notation

## Teorema 1

Jika  $T(n)$  adalah suatu polinom dengan derajat kurang dari atau sama dengan  $m$ , maka  $T(n) = O(n^m)$ .

$$T(n) = 5 = 5n^0 = O(n^0) = O(1)$$

$$T(n) = 2n + 3 = O(n)$$

$$T(n) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

$$T(n) = 3n^3 + 2n^2 + 10 = O(n^3)$$

# Big O Notation

## Ingat!

Dalam **Big-O Notation**, tidak ada koefisien atau suku-suku lainnya, hanya berisi fungsi-fungsi standar :  $n$ ,  $1$ ,  $n^2$ ,  $\log n$ , dan sebagainya.

**TEOREMA 2.** Misalkan  $T_1(n) = O(f(n))$  dan  $T_2(n) = O(g(n))$ , maka

- (a)  $T_1(n) + T_2(n) = O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$
- (b)  $T_1(n)T_2(n) = O(f(n))O(g(n)) = O(f(n)g(n))$
- (c)  $O(cf(n)) = O(f(n))$ ,  $c$  adalah konstanta
- (d)  $f(n) = O(f(n))$

# Big O Notation

Kelompok Algoritma	Nama
$O(1)$	konstan
$O(\log n)$	logaritmik
$O(n)$	linier
$O(n \log n)$	linier logaritmik
$O(n^2)$	kuadratik
$O(n^3)$	kubik
$O(2^n)$	eksponensial
$O(n!)$	faktorial

Urutan spektrum kompleksitas waktu algoritma adalah :

$$\underbrace{1 < \log n < n < n \log n < n^2 < n^3 < \dots < 2^n < n!}_{\text{algoritma polinomial (bagus)}}$$



algoritma eksponensial  
(buruk)

# Big-O Notation dalam Program

Kelompok 5

```
public class latihan_1 {
    public static void main(String[] args) {
        Scanner Khamim = new Scanner(System.in);
        System.out.print("Masukkan peringkat klub top 3 yang ingin anda cari: ");
        int i = Khamim.nextInt();
        String[] klasemen = {"Real Madrid", "Barcelona", "Atletico Madrid"};

        System.out.println("Tim yang menduduki peringkat ke-" + i + " adalah : " + klasemen[i-1]);
    }
}
```

```
public class latihan_1 {  
    public static void main(String[] args) {  
        Scanner Khamim = new Scanner(System.in);  
        System.out.print("Masukkan peringkat klub top 3 yang ingin anda cari: ");  
        int i = Khamim.nextInt();  
        String[] klasemen = {"Real Madrid", "Barcelona", "Atletico Madrid"};  
  
        System.out.println("Tim yang menduduki peringkat ke-" + i + " adalah : " + klasemen[i-1]);  
    }  
}
```

O(1)

```
public static void main(String[] args) {
    int[] arr = {1, 2, 3, 4};
    int total = 0;
    for (int i = 0; i<arr.length; i++) {
        total += arr[i];
    }
    System.out.println("Total: " + total);
}
```

```
public static void main(String[] args) {  
    int[] arr = {1, 2, 3, 4};  
    int total = 0;  
    for (int i = 0; i<arr.length; i++) {  
        total += arr[i];  
    }  
    System.out.println("Total: " + total);  
}
```

$O(n)$

```
public static void main(String[] args) {  
  
    for (int i = 0; i < 10; i++) {  
        System.out.println(i);  
    }  
  
    for (int j = 0; j < 10; j++) {  
        System.out.println(j);  
    }  
}
```

```
public static void main(String[] args) {  
  
    for (int i = 0; i < 10; i++) {  
        System.out.println(i);  
    }  
  
    for (int j = 0; j < 10; j++) {  
        System.out.println(j);  
    }  
}
```

**O(n)**

```
public static void main(String[] args) {  
  
    for (int i = 0; i < 10; i++) {  
        for (int j = 0; j < 10; j++) {  
            System.out.println(i);  
            System.out.println(j);  
        }  
    }  
}
```

```
public static void main(String[] args) {  
  
    for (int i = 0; i < 10; i++) {  
        for (int j = 0; j < 10; j++) {  
            System.out.println(i);  
            System.out.println(j);  
        }  
    }  
}
```

$O(n^2)$

```
for(int i = 2; i<= n; i++) {  
    for(j=5; j <= n; j = j*2) {  
        k = k+j;  
    }  
}
```

```
for(int i = 2; i<= n; i++) {  
    for(j=5; j <= n; j = j*2) {  
        k = k+j;  
    }  
}
```

$O(n \log n)$

# Rangkuman

Kompleksitas	Operasi
$O(1)$	Menjalankan sebuah perintah
$O(1)$	Mendapatkan sebuah item dari array, objek atau variabel
$O(\log n)$	Pengulangan yang berkurang setengahnya setiap iterasi
$O(n^2)$	Pengulangan dalam pengulangan
$O(n^3)$	Pengulangan dalam pengulangan dalam pengulangan

# Searching Algorithm

Kelompok 5

# Searching Algorithm

**Searching Algorithm** adalah Proses pencarian data yang ada pada suatu deret data dengan cara menelusuri data-data tersebut.

## Jenis - Jenis Searching Algorithm

- Single match
- Multiple match

## Metode Searching Algorithm

- Linear Search
- Binary Search

# Linear Search

**Linear Search** adalah suatu teknik pencarian data yang akan menelusuri tiap elemen satu per-satu dari awal sampai akhir.

## Langkah - Langkah

- Mulai dari elemen pertama.
- Bandingkan setiap elemen dengan elemen yang dicari.
- Jika cocok, kembalikan indeks elemen tersebut.
- Jika tidak ditemukan, kembalikan -1 atau "not found".

# Linear Search

## Contoh

- Misalnya terdapat array satu dimensi sebagai berikut:

0	1	2	3	4	5	6	7	indeks
8	10	6	-2	11	7	1	100	value

- Kemudian program akan meminta data yang akan dicari, misalnya **6**.
- Iterasi :
  - 6 = 8 (tidak!)
  - 6 = 10 (tidak!)
  - 6 = 6 (Ya!) => output : 2 (index)

## Contoh

# Linear Search

### Contoh Kasus

Diketahui array: [5, 8, 2, 10, 3, 7], cari angka 10.

### Ilustrasi Langkah

- Bandingkan 10 dengan 5 (tidak cocok).
- Bandingkan 10 dengan 8 (tidak cocok).
- Bandingkan 10 dengan 2 (tidak cocok).
- Bandingkan 10 dengan 10 (cocok, ditemukan di indeks 3).

```
1 public class LinearSearch {  
2     public static int search(int[] arr, int x) {  
3         for (int i = 0; i < arr.length; i++) {  
4             if (arr[i] == x) {  
5                 return i; // Mengembalikan indeks jika ditemukan  
6             }  
7         }  
8         return -1; // Jika tidak ditemukan  
9     }  
10  
11    public static void main(String[] args) {  
12        int[] arr = {5, 8, 2, 10, 3, 7};  
13        int x = 10;  
14        int result = search(arr, x);  
15        System.out.println(result != -1 ? "Elemen ditemukan di indeks " + result :  
16                                "Elemen tidak ditemukan");  
17    }  
18}
```

# Binary Search

**Bibary Search** adalah pencarian data dimulai dari pertengahan data yang telah terurut.

## Langkah - Langkah

- Data diambil dari posisi 1 sampai posisi akhir N.
- Cari posisi tengah dengan rumus  $(\text{posisi awal} + \text{posisi akhir}) / 2$
- Bandingkan data yang dicari dengan data tengah:
- Jika sama, maka elemen ditemukan.
- Jika lebih besar, pencarian dilanjutkan di bagian kanan ( $\text{posisi awal} = \text{posisi tengah} + 1$ ).
- Jika lebih kecil, pencarian dilanjutkan di bagian kiri ( $\text{posisi akhir} = \text{posisi tengah} - 1$ ).

# Contoh

# Binary Search

Misalnya data yang dicari 17



- 0      1      2      3      4      5      6      7      8
  - **3**      **9**      **11**      **12**      **15**      **17**      **23**      **31**      **35**
  - **A**      **B**      **C**
  - Karena  $17 < 23$  (data tengah), maka: akhir = tengah - 1

- 0      1      2      3      4      5      6      7      8
  - **3      9      11      12      15      17      23      31      35**
  - **A=B=C**
  - Karena  $17 = 17$  (data tengah), maka KETEMU!

## Contoh

# Binary Search

### Contoh Kasus

Diketahui array terurut: [2, 3, 5, 7, 8, 10], cari angka 7.

### Ilustrasi Langkah

1. Tengah = 5 → 7 lebih besar, cari di kanan.
2. Tengah = 8 → 7 lebih kecil, cari di kiri.
3. Tengah = 7 → ditemukan di indeks 3.

```
1 public class BinarySearch {  
2     public static int search(int[] arr, int x) {  
3         int left = 0, right = arr.length - 1;  
4         while (left <= right) {  
5             int mid = left + (right - left) / 2;  
6             if (arr[mid] == x) {  
7                 return mid;  
8             } else if (arr[mid] < x) {  
9                 left = mid + 1;  
10            } else {  
11                right = mid - 1;  
12            }  
13        }  
14        return -1;  
15    }  
16  
17    public static void main(String[] args) {  
18        int[] arr = {2, 3, 5, 7, 8, 10};  
19        int x = 7;  
20        int result = search(arr, x);  
21        System.out.println(result != -1 ? "Elemen ditemukan di indeks " + result :  
22            "Elemen tidak ditemukan");  
23    }  
24}
```

# Kompleksitas Searching Algorithm

Kelompok 5

# Linear Search

Jumlah operasi perbandingan elemen tabel:

1. *Kasus terbaik*: ini terjadi bila  $a_1 = x$ .

$$T_{\min}(n) = 1$$

2. *Kasus terburuk*: bila  $a_n = x$  atau  $x$  tidak ditemukan.

$$T_{\max}(n) = n$$

3. *Kasus rata-rata*: Jika  $x$  ditemukan pada posisi ke- $j$ , maka operasi perbandingan ( $a_k = x$ ) akan dieksekusi sebanyak  $j$  kali.

$$T_{\text{avg}}(n) = \frac{(1 + 2 + 3 + \dots + n)}{n} = \frac{\frac{1}{2}n(1+n)}{n} = \frac{(n+1)}{2}$$

# Binary Search

1. Kasus terbaik

$$T_{\min}(n) = 1$$

2. Kasus terburuk:

$$T_{\max}(n) = \log_2 n + 1$$

$n=1 \rightarrow$  pencarian  $T=1$

$n=2=2^1 \rightarrow$  pencarian  $T=2$

$n=4=2^2 \rightarrow$  pencarian  $T=3$

$n=8=2^3 \rightarrow$  pencarian  $T=4 = 3 + 1$

:

$n=2^k \rightarrow$  pencarian  $T = k + 1 = \log_2 n + 1$

# Perbandingan

Metode	Linear Search	Binary Search
<b>Kecepatan</b>	Lambat ( $O(n)$ )	Cepat ( $O(\log n)$ )
<b>Kondisi Data</b>	Tidak perlu terurut	Harus terurut
<b>Implementasi</b>	Sederhana	Kompleks
<b>Efisiensi</b>	Kurang efisien untuk data besar	Sangat efisien untuk data besar

# QnA

# Tugas

Kelompok 5

## Soal no. 1

Buatkan sebuah program untuk menampilkan nilai elemen suatu matriks pada baris ke- $i$  kolom ke- $j$  pada sebuah matriks berukuran  $m \times n$  (gunakan Array 2D)! Tentukan pula kompleksitas waktu dan nyatakan dalam **big-O notation!**

## Soal no. 2

Buatlah program Java untuk mencari angka 23 dalam array [5, 12, 18, 23, 29, 35, 42, 50, 60] menggunakan Linear Search dan Binary Search. Bandingkan kompleksitas algoritma masing-masing dalam notasi Big-O dalam tabel dan berikan kesimpulannya.

# Thank You