

## Pengenalan Algoritma

### A. Algoritma Konstan

Suatu algoritma memiliki waktu eksekusi konstan jika tidak ada ketergantungan antara waktu eksekusi algoritma dengan ukuran masukan. Dalam algoritma ini, jumlah operasi yang dilakukan tetap konstan terlepas dari ukuran masukan yang diberikan.

Misalnya, ketika mengakses elemen dalam array menggunakan operator kurung siku (`[]`), operasi ini memerlukan waktu konstan. Tidak peduli seberapa besar ukuran array tersebut, waktu yang diperlukan untuk mengakses satu elemen tetap sama. Sebagai contoh, mengakses elemen ke-5 dari array dengan 10 elemen dan mengakses elemen ke-5 dari array dengan 1000 elemen, keduanya membutuhkan jumlah operasi yang sama.

Berikut adalah contoh studi kasus yang menggunakan algoritma dengan waktu eksekusi konstan dengan menggunakan perulangan `for` sesuai jumlah elemen dalam array:

A screenshot of a code editor window titled 'main.java' with a dark background and light-colored text. The code defines a class 'ConstantTimeAlgorithm' with a 'main' method and a 'processArray' method. The 'main' method creates an array {5, 8, 2, 10, 3} and calls 'processArray'. The 'processArray' method uses a 'for' loop to iterate over each element in the array, printing a message for each. Comments in Indonesian explain the steps. At the bottom right, there is a 'CodeImage' logo.

```
public class ConstantTimeAlgorithm {  
  
    public static void main(String[] args) {  
        int[] array = { 5, 8, 2, 10, 3 };  
  
        processArray(array); // Memanggil method processArray dengan argumen array  
    }  
  
    public static void processArray(int[] array) {  
        for (int i = 0; i < array.length; i++) { // Melakukan perulangan sebanyak elemen dalam array  
            System.out.println("Processing element at index " + i + ": " + array[i]); // Mencetak elemen array pada indeks yang sedang  
            diproses  
        }  
    }  
}  
  
/*  
Hasil yang diharapkan:  
Processing element at index 0: 5  
Processing element at index 1: 8  
Processing element at index 2: 2  
Processing element at index 3: 10  
Processing element at index 4: 3  
*/
```

### B. Algoritma Linear

Suatu algoritma berjalan secara linear jika waktu eksekusinya berbanding lurus dengan ukuran masukan. Artinya, semakin besar ukuran masukan, semakin banyak operasi yang dilakukan oleh algoritma.

Sebagai contoh, saat menjumlahkan elemen-elemen dalam sebuah array, algoritma tersebut harus mengakses setiap elemen dalam array untuk melakukan penjumlahan. Jumlah operasi yang dilakukan adalah sebanyak  $n$  (ukuran array) dikurangi 1, karena ada  $n-1$  penjumlahan antara elemen-elemen. Total jumlah operasi adalah  $2n - 1$ , yang berbanding lurus dengan  $n$  saat ukuran masukan bertambah.

Berikut adalah contoh implementasi algoritma dengan waktu eksekusi linear untuk menjumlahkan elemen-elemen dalam sebuah array:

```
main.java

public class LinearTimeAlgorithm {

    public static void main(String[] args) {
        int[] array = { 2, 4, 6, 8, 10 };

        int sum = sumArray(array);
        System.out.println("Sum of the array elements: " + sum);
    }

    public static int sumArray(int[] array) {
        int sum = 0; // Inisialisasi variabel sum sebagai 0
        for (int i = 0; i < array.length; i++) {
            sum += array[i]; // Menambahkan nilai elemen pada array ke variabel
sum
        }
        return sum; // Mengembalikan nilai sum
    }
}

/*
    Hasil yang diharapkan:
    Sum of the array elements: 30
*/
```

### C. Algoritma Kuadratik

Suatu algoritma berjalan secara kuadratik jika waktu eksekusinya berbanding lurus dengan kuadrat dari ukuran masukan ( $n^2$ ). Artinya, semakin besar ukuran masukan, semakin banyak operasi yang dilakukan oleh algoritma dan jumlah operasinya tumbuh dengan cepat.

Misalnya, untuk memeriksa apakah ada elemen yang muncul lebih dari sekali dalam sebuah daftar, algoritma sederhana adalah dengan membandingkan setiap elemen dengan elemen-elemen lainnya. Jika ada  $n$  elemen dalam daftar, setiap elemen akan dibandingkan dengan  $n-1$  elemen lainnya. Jumlah total perbandingan adalah  $n^2 - n$ , yang berbanding lurus dengan  $n^2$  saat ukuran masukan bertambah.

Berikut adalah contoh implementasi algoritma dengan waktu eksekusi kuadratik untuk memeriksa apakah ada elemen duplikat dalam sebuah array:

```
public class QuadraticTimeAlgorithm {

    public static void main(String[] args) {
        int[] array1 = { 1, 2, 3, 4, 5 };
        int[] array2 = { 1, 2, 3, 4, 4 };

        boolean hasDuplicates1 = hasDuplicates(array1);
        System.out.println("Array 1 has duplicates: " + hasDuplicates1);

        boolean hasDuplicates2 = hasDuplicates(array2);
        System.out.println("Array 2 has duplicates: " + hasDuplicates2);
    }

    public static boolean hasDuplicates(int[] array) {
        for (int i = 0; i < array.length; i++) {
            for (int j = i + 1; j < array.length; j++) {
                if (array[i] == array[j]) {
                    return true; // Mengembalikan true jika ada elemen duplikat
                }
            }
        }
        return false; // Mengembalikan false jika tidak ada elemen duplikat
    }
}

/*
    Hasil yang diharapkan:
    Array 1 has duplicates: false
    Array 2 has duplicates: true
*/
```

Sumber

Allen B. Downey, Think Data Structures, Algorithms and Information Retrieval in Java