Kompleksitas Algoritma

Teosofi Hidayah Agung
Hafidz Mulia

Departemen Matematika
Institut Teknologi Sepuluh Nopember

22 Maret 2025

Warm Up

Apakah kedua algoritma berikut menghasilkan output yang sama?

Algorithm Hmmm

```
1: procedure Fofo(a_1, a_2, ..., a_n)
2:     m ← 0
3:     for i ← 1 to n − 1 do
4:         for j ← i + 1 to n do
5:             if |a_i − a_j| > m then
6:                 m ← |a_i − a_j|
7:             end if
8:         end for
9:     end for
10: end procedure
```

Algorithm Ermmm

```
1: procedure Fifi(a_1, a_2, ..., a_n)
2:     m_1 ← a_1
3:     m_2 ← a_1
4:     for i ← 2 to n do
5:         if a_i < m_1 then
6:             m_1 ← a_i
7:         else if a_i > m_2 then
8:             m_2 ← a_i
9:         end if
10:    end for
11:    m ← m_2 − m_1
12: end procedure
```

Sebuah algoritma haruslah ~~anggaran pendidikan~~ **efisien**, artinya:

- Memiliki **waktu eksekusi** yang cepat.
- **Penggunaan memori** yang optimal.
- Ukuran input dapat **diskalakan** tanpa mempengaruhi kinerja algoritma.

Daftar isi

Complexity

- **Kompleksitas waktu** diukur dari jumlah tahapan komputasi yang dilakukan oleh algoritma, biasanya disimbolkan dengan fungsi $T(n)$.
- **Kompleksitas ruang** diukur dari jumlah memori yang digunakan oleh struktur data di algoritma, biasanya disimbolkan dengan fungsi $S(n)$.

Kompleksitas selalu ditentukan berdasarkan ukuran masukannya.

Akibat

Jika terdapat sebanyak k buah masukan/input, maka kompleksitasnya merupakan fungsi dengan variabel bebas sebanyak k. ($F(n_1, n_2, \ldots, n_k)$)

Complexity

Contoh

Method penjumlahan dan perkalian matriks bergantung pada ukuran matriks yang diinputkan.

Contoh

Menghitung FPB atau KPK dari dua bilangan bergantung pada kedua bilangan tersebut.

Contoh

Mencari nilai terbesar (maksimum) dari sebuah array bergantung pada panjang array.

Kompleksitas Waktu

- **Worst Case** $T_{max}(n) :=$ Kompleksitas waktu terburuk yang mungkin terjadi.
- **Best Case** $T_{min}(n) :=$ Kompleksitas waktu terbaik yang mungkin terjadi.
- **Average Case** $T_{avg}(n) :=$ Kompleksitas waktu rata-rata yang mungkin terjadi.

Kompleksitas Waktu

Biasanya hitungan kompleksitas waktu meliputi:

- Operasi baca/tulis `(input a, print a)`
- Operasi aritmatika `(b + c, m * n)`
- Operasi perbandingan `(a < b, x == y)`
- Operasi *assignment* `(a ← b, x ← 5)`
- dll.

- **Algoritma pencarian (searching)**
  Operasi khas: operasi perbandingan elemen larik.
- **Algoritma pengurutan (sorting)**
  Operasi khas: operasi perbandingan elemen dan operasi pertukaran elemen.
- **Algoritma perkalian dua buah matriks** $AB = C$
  Operasi khas: operasi perkalian dan penjumlahan.

Algorithm Cari Maksimum

```
1: procedure Max(a_1, a_2, ..., a_n)          ▷ Input: Array integer
2:     max ← a_1                              ▷ Inisialisasi dengan elemen pertama
3:     for i ← 2 to n do
4:         if max < a_i then
5:             max ← a_i
6:         end if
7:     end for
8:     return max                             ▷ Nilai maksimum dalam array
9: end procedure
```

Algorithm Cari Maksimum

```
1: procedure Max(a_1, a_2, ..., a_n)          ▷ Input: Array integer
2:     max ← a_1                              ▷ Inisialisasi dengan elemen pertama
3:     for i ← 2 to n do
4:         if max < a_i then
5:             max ← a_i
6:         end if
7:     end for
8:     return max                             ▷ Nilai maksimum dalam array
9: end procedure
```

Operasi dasarnya adalah perbandingan elemen ($max < a_1$) yang dilakukan sebanyak $n - 1$ kali. Kompleksitas waktu: $T(n) = n - 1$

Algorithm Perhitungan dengan Nested Loop dan Pembagian

1: **Input**: n
2: $j \leftarrow n$, $x \leftarrow 1$
3: **while** $j \geq 1$ **do**
4: **for** $i \leftarrow 1$ to j **do** $x \leftarrow x \times i$
5: **end for**
6: $j \leftarrow j \div 2$
7: **end while**
8: jumlah $\leftarrow x$

Algorithm Perhitungan dengan Nested Loop dan Pembagian

1: **Input**: n
2: $j \leftarrow n$, $x \leftarrow 1$
3: **while** $j \geq 1$ **do**
4: **for** $i \leftarrow 1$ to j **do** $x \leftarrow x \times i$
5: **end for**
6: $j \leftarrow j \div 2$
7: **end while**
8: jumlah $\leftarrow x$

Kompleksitas waktu: $T(n) = n\left(2 - \frac{1}{2^{n-1}}\right)$

Kompleksitas Ruang

- **Worst Case** $S_{max}(n) :=$ Kompleksitas ruang terburuk yang mungkin terjadi.
- **Best Case** $S_{min}(n) :=$ Kompleksitas ruang terbaik yang mungkin terjadi.
- **Average Case** $S_{avg}(n) :=$ Kompleksitas ruang rata-rata yang mungkin terjadi.

Fakta saat ini

Kompleksitas ruang tidak akan dibahas lebih jauh karena pada dasarnya solusi untuk permasalahan *storage* lebih mudah diatasi daripada *running time*.

Analisa kompleksitas ruang biasanya digunakan untuk fungsi tipe rekursif.

Contoh

Space complexity: $S(n) = n$

```
1 int sum(int n){
2   if (n <= 0)return 0;
3   return n + add (n-1);
4 }
```

Kode: Jumlahan n

Program akan menyimpan n nilai memori sebelum fungsi berakhir.

Contoh

Space complexity: $S(n) = 1$

```
1  int addSequence(int n){
2    int sum = 0;
3    for (int i = 0; i < n; i++){
4        sum += pairSum(i, i+1);
5    }
6    return sum;
7  }
8  int pairSum(int x, int y){
9    return x + y;
10 }
```

Kode: Faktorial

Program hanya menyimpan satu nilai memori selama eksekusi. (tidak ter-*stack*)

Contoh

Space complexity: $S(n) = 2^n$

```
1 int fib(int n){
2   if (n <= 1)return n;
3   return fib (n-1) + fib (n-2);
4 }
```

Kode: Fibonacci

Program memiliki 2 cabang setiap pemanggilan fungsi, sehingga memori yang digunakan akan berlipat dua setiap pemanggilan.

Daftar isi

Big-O

“Hope for the best, prepare for the worst.”

Masalah

Dalam pemograman dan algoritma, $T_{max}(n)$ seringkali menjadi fokus utama daripada karena meminimalisir kemungkinan terburuk lebih penting daripada memaksimalkan kemungkinan terbaik.
Namun tidak semua program memiliki kompleksitas waktu yang dapat dideterminasi. Sehingga diperlukan pendekatan lain untuk mendefinisikan kompleksitas waktu secara umum.

Big-O

Definisi (Big-O)

Misalkan $f(n)$ dan $g(n)$ adalah dua fungsi. Kita mengatakan bahwa $f(n)$ adalah $\mathcal{O}(g(n))$ jika dan hanya jika terdapat konstanta positif C dan n_0 sedemikian sehingga

$$C \cdot g(n) > f(n)$$

untuk semua $n > n_0$. Biasanya disimbolkan sebagai $f(n) = \mathcal{O}(g(n))$ atau $f(n) \in \mathcal{O}(g(n))$.

Untuk alasan praktikal, nantinya kita akan memilih fungsi $g(n)$ sekecil mungkin agar $\mathcal{O}(g(n))$ memiliki makna.

Big-O

Contoh

- $f(n) = 2n^2 + 3n + 1$ dan $g(n) = n^2$
  Karena $2n^2 + 3n + 1 \leq 2n^2 + 3n^2 + n^2 = 6n^2$ untuk $n > 1$
  atau $2n^2 + 3n + 1 \leq n^2 + n^2 + n^2 = 3n^2$ untuk $n > 34$, maka $f(n) \in \mathcal{O}(n^2)$.
- $f(n) = \ln(x + 4)$ dan $g(n) = \log n$
  Karena $\ln(x + 4) \leq 6 \log n$ untuk $n > 2$, maka $f(n) \in \mathcal{O}(\log n)$.
- $T(n) = 6 \cdot 2^n + 2n^2$ dan $g(n) = 2^n$
  Karena $6 \cdot 2^n + 2n^2 \leq 6 \cdot 2^n + 2 \cdot 2^n = 8 \cdot 2^n$ untuk $n > 1$ ($C = 8$, $n_0 = 1$), maka $T(n) \in \mathcal{O}(2^n)$.

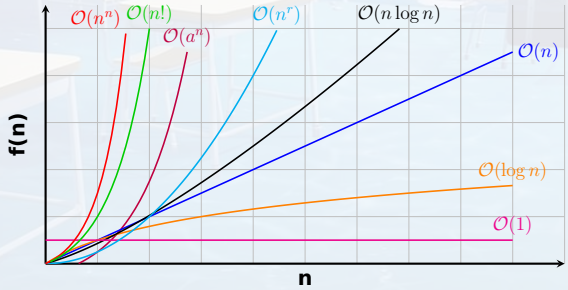
Big-O

Teorema 1

untuk n yang cukup besar, berlaku

$$1 < \log n < n^r < a^n < n! < n^n$$

untuk $r \geq 1$ dan $a > 1$.



Big-O

Teorema 2

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

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

Big-O

Bentuk Sederhana

Ingat, di dalam notasi Big-Oh tidak ada koefisien atau suku-suku lainnya, hanya berisi fungsi-fungsi standard seperti $1, n^2, n^3, \ldots, \log n, n \log n, 2^n, n!$, dan sebagainya. Contoh:

- $O(2n)$ tidak standard, seharusnya $O(n)$
- $O(\log(n^2))$ tidak standard, seharusnya $O(\log n)$
- $O\left(\frac{n^2}{2}\right)$ tidak standard, seharusnya $O(n^2)$
- $O((n-1)!)$ tidak standard, seharusnya $O(n!)$

Daftar isi

Latihan 1

Tunjukkan/buktikan pernyataan berikut:

- $1 + 2 + 3 + \cdots + n = \mathcal{O}(n^2)$
- $n! = \mathcal{O}(n^n)$
- $\log n! = \mathcal{O}(n \log n)$
- $2^n = \mathcal{O}(3^n)$
- $(n + 1) \log(n^3 + 2) + 4n^2 \in \mathcal{O}(n^2)$

Latihan 2

Jawablah pertanyaan dibawah ini yang mengacu pada algoritma dibawahnya

- Apa yang dilakukan algoritma berikut?
- Berapakah kompleksitas waktu dari algoritma berikut? Tuliskan jawaban dalam notasi Big-O.
- Coba ubahlah algoritma tersebut agar lebih efisien terhadap kompleksitas waktu.

```
1  public static int solution(int n) {
2      int solutions = 0;
3      for (int a = 0; a <= n; a++)
4          for (int b = 0; b <= n; b++)
5              for (int c = 0; c <= n; c++)
6                  if (a + b + c == n) solutions++;
7
8      return solutions;
9  }
```