ANALISIS ALGORITMA

Tennov Simanjuntak

Topik

- Kinerja algoritma
- Tujuan analisis algoritma
- Waktu eksekusi algoritma
- Pertumbuhan fungsi
- Notasi asimtotik

Kinerja algoritma

- Pada kuliah sebelumnya: “Peranan Algoritma dalam Komputasi”, disebutkan bahwa kinerja algoritma dicerminkan oleh efisiensi algoritma.
- Ada 2 ukuran efisiensi algoritma, yakni:
  - **Waktu eksekusi (*running time*)**
  - **Besar memori yang digunakan (*memory space*)**
- Waktu eksekusi yang cepat menyebabkan algoritma berakhir dan menemukan solusi dengan cepat.
- Semakin kecil memori yang digunakan maka semakin baik algoritma.
- Waktu eksekusi yang cepat dan memori yang kecil menyebabkan penggunaan sumber daya komputasi (*computing resources*) semakin kecil.
- Oleh sebab itu, jika waktu eksekusi dan kebutuhan memori penyimpanan suatu algoritma semakin kecil, maka algoritma semakin baik.

Seberapa penting kinerja algoritma?

Illustrasi:

- Link ini: http://www.mersenne.org/primes/ menyebutkan bahwa bilangan prima terbesar saat ini adalah: $\mathbf{2^{82,589,933} - 1}$. Menurut anda, apakah algoritma *sieve of erathostenes* yang diterapkan di komputer anda bisa mencari bilangan ini?

Tujuan melakukan analisis algoritma

- Mengetahui waktu eksekusi dan kebutuhan memori sehingga dapat diketahui apakah sumber daya komputasi cukup untuk menjalankan algoritma.
- Mengetahui apakah algoritma dapat/tidak dapat memecahkan masalah komputasi dalam periode tertentu. Periode tertentu ini biasanya mengacu kepada periode dimana keluaran algoritma diperlukan atau dapat juga periode dimana keluaran algoritma kadaluwarsa.
- Dapat memprediksi kinerja algoritma jika diterapkan dalam lingkungan yang baru (misalnya, mesin komputer yang baru).

Waktu eksekusi algoritma: faktor

- Ada 2 faktor yang mempengaruhi waktu eksekusi algoritma, yakni
  - **Jumlah langkah** yang terdapat pada prosedur algoritma.
  - **Jumlah masukan** atau input.
- Analisis waktu eksekusi algoritma dilakukan dengan asumsi jumlah input yang besar (tak hingga), bukan dengan jumlah input yang kecil. Illustrasi: waktu eksekusi **insertion sort** (**$2n^2$**) lebih kecil dari **merge sort** (**50n log n**) hanya ketika input < 191.
- Oleh karena analisis dilakukan pada kondisi input tak hingga, maka **jumlah langkah algoritma yang konstan tidak terlalu berpengaruh kepada waktu eksekusi**.
- Dengan kata lain, **waktu eksekusi merupakan fungsi dari input atau fungsi input**.

Waktu eksekusi dan notasi asimptotik

- **Waktu eksekusi merupakan fungsi input**.
- **Oleh sebab itu, waktu eksekusi akan bertumbuh saat input bertumbuh.**
- Dengan kata lain waktu eksekusi merupakan fungsi yang monoton bertumbuh sejalan dengan pertumbuhan jumlah input.

Fungsi monoton

- DEFINISI KEMONOTONAN FUNGSI (*Monotonicity*)
- Sebuah fungsi *f(n)* disebut monoton bertumbuh jika untuk $m \leq n$

  $f(m) \leq f(n)$

- Sebaliknya, monoton berkurang jika untuk $m \leq n$

  $f(m) \geq f(n)$

- Sebuah fungsi disebut bertumbuh dengan ketat (*strictly increasing*), jika untuk $m < n$ $f(m) < f(n)$

  dan berkurang dengan ketat (*strictly decreasing*), jika untuk

  $m < n$ $f(m) > f(n)$

Pertumbuhan fungsi

- **Pertumbuhan fungsi waktu eksekusi algoritma dimodelkan dengan notasi asimtotik atau notasi tak hingga.** Hal ini disebabkan analisis algoritma dilakukan/diasumsikan untuk jumlah input yang sangat besar (tak hingga).
- Algoritma yang secara asimtotik lebih efisien (lebih cepat) menjadi pilihan untuk semua kasus jumlah input, kecuali untuk input yang kecil.
- Notasi asimptotik yang dibahas pada topik ini adalah untuk fungsi yang domainnya adalah bilangan natural N = {0,1,2...}. Dengan kata lain, fungsi waktu eksekusi yang jumlah inputnya merupakan bilangan bulat >= 0.
- Ada 3 notasi asimtotik yang digunakan untuk memodelkan fungsi pertumbuhan waktu eksekusi algoritma yakni: **notasi-O (big Oh)**, **notasi-Ω (big Omega)**, dan **notasi-Θ (big teta)**.

Notasi – **O** (batas atas asimtotik)

- Notasi – **O** merupakan batas atas (*asymptotic upper bound*) dari fungsi waktu eksekusi algoritma. Dapat juga diartikan sebagai fungsi dari waktu eksekusi yang paling maksimal (terlama) suatu algoritma.

$$f(n) = O(g(n)) \text{ jika ada konstanta positif } c \text{ dan } n_0 \text{ sedemikian hingga}$$

$$0 \le f(n) \le cg(n) \text{ untuk semua } n \ge n_0$$

- Contoh:

$$\frac{1}{2}n^2 - 3\mathrm{n} = On^2$$

- Bukti: pilih $c \ge \frac{1}{2} \text{ dan } n \ge 1$

Notasi – **Ω (batas bawah asimtotik)**

- Notasi – **Ω** merupakan batas bawah (*asymptotic lower bound*) dari fungsi waktu eksekusi algoritma. Dapat juga diartikan sebagai fungsi dari waktu eksekusi yang paling minimal (cepat) suatu algoritma.

$$f(n)=\Omega(g(n)) \text{ } jika\ ada\ konstanta\ positif\ c\ dan\ n_0\ sedemikian\ hingga$$
$$0 \le cg(n) \le f(n)\ untuk\ semua\ n \ge n_0$$

- Contoh:

$$\frac{1}{2}n^2 - 3\mathrm{n} = \Omega n^2$$

- Bukti: pilih $c \le \frac{1}{14}\ dan\ n \ge 7$

Notasi – **Θ (batas ketat asimtotik)**

- Notasi – **Θ** merupakan batas ketat (*asymptotic tight bound*) dari fungsi waktu eksekusi algoritma.

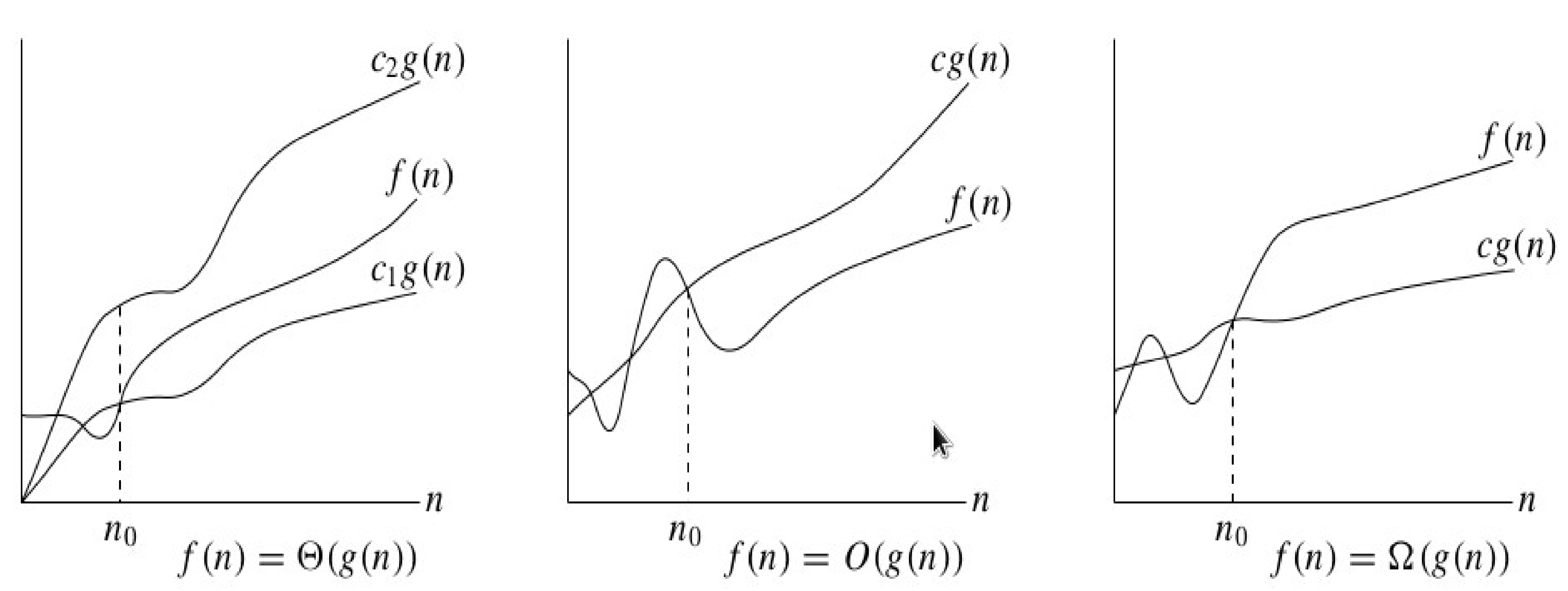
$$f(n) = \Theta(g(n)) \text{ jika ada konstanta positif } c_1, c_2, \text{dan } n_0$$
$$\text{sedemikian hingga}$$
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ untuk semua } n \geq n_0$$

- Contoh:

$$\frac{1}{2}n^2 - 3\mathrm{n} = \Theta n^2$$

- Bukti: gabungkan pembuktikan *upper bound* dan *lower bound*.

Contoh grafik dari notasi asimtotik



Contoh fungsi waktu eksekusi algoritma

c	konstant
$\log n$	logaritmik
$\log^2 n$	logaritmik kuadrat
n	linier
$n \log n$	
n^2	kuadratik
n^3	kubik
2^n	eksponensial
$n!$	faktorial

Algoritma yang efisien

- Algoritma disebut efisien jika memiliki **fungsi waktu eksekusi polinomial**.
- Hal ini disebabkan komputer yang ada saat ini memiliki kecepatan eksekusi yang mengikuti fungsi polinomial.
- Sama dengan waktu eksekusi, algoritma disebut memiliki kebutuhan memori yang efisien jika fungsi kebutuhan memori polinomial.

Perhitungan waktu eksekusi

- **ATURAN 1 – Perulangan (*Loop*)**

Waktu eksekusi blok perulangan adalah hasil perkalian dari banyaknya pernyataan dan jumlah iterasi.

- **ATURAN 2 – Perulangan bersarang (*Nested loops*)**

Total waktu eksekusi dalam blok perulangan bersarang adalah hasil perkalian dari banyaknya pernyataan dan ukuran dari semua blok perulangan.

- **ATURAN 3 – Pernyataan yang berurut**

Total waktu pernyataan berurut adalah penjumlah dari pernyataan.

- **ATURAN 4 – Kondisional (IF/ELSE)**

Waktu eksekusi blok kondisional adalah jumlah kondisi ditambah waktu eksekusi terbesar dari solusi.

Contoh perhitungan waktu eksekusi

- ***MAXIMUM SUBSEQUENCE SUM PROBLEM***

Jika diberikan bilangan bulat (mungkin negatif) $\boldsymbol{a_1, a_2, \ldots a_n}$ carilah jumlah maksikum dari

$$\sum_{k=i}^{j} a_k$$

- Contoh:

Untuk input: {-2, 11, -4, 13, -5, -2} jawaban = 20 ($\boldsymbol{a_2}$ s/d $\boldsymbol{a_4}$).

Max subsequence problem: **Algoritma 1**

```
int max_subsequence_sum(int *seq, unsigned int n){
    int this_sum, max_sum, best_i, best_j, best_k;
    int i, j, k;

    max_sum = 0; best_i = best_j = -1;
    for(i = 0; i < n; i++){
        for(j = i; j < n; j++){
            this_sum = 0;
            for( k = i; k <= j; k++){
                this_sum +=seq[k];
            }
            if(this_sum > max_sum){
                max_sum = this_sum;
                best_i = i;
                best_j = j;
            }
        }
    }
    return max_sum;
}
```

Waktu eksekusi (*running time*) = $O(n^3)$

Efisiensi algoritma: jumlah memori

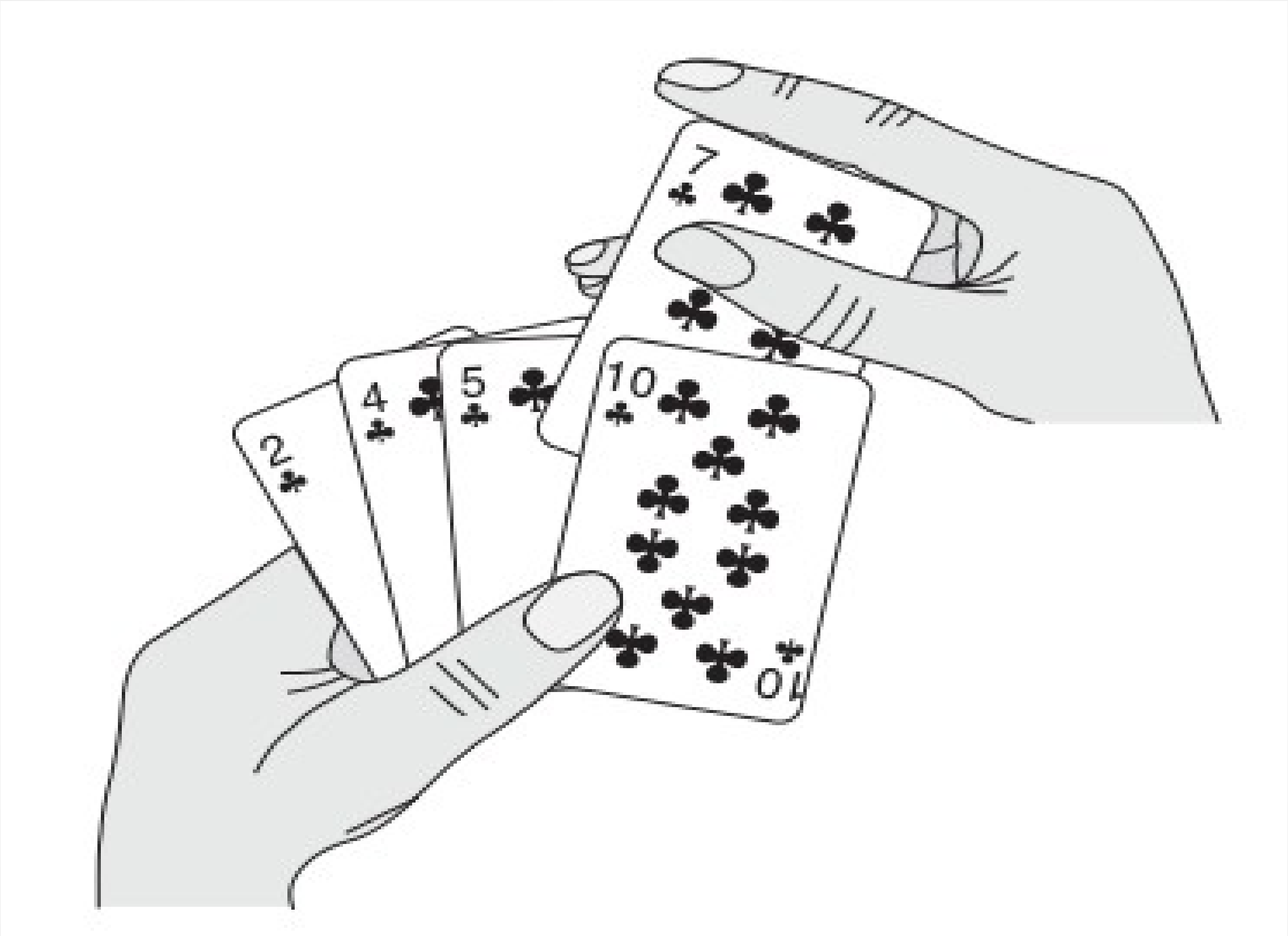
- Kebutuhan memori dalam hal ini RAM, tergantung pada:
  - Jumlah memori yang dibutuhkan untuk menyimpan kode program.
  - Jumlah memori yang dibutuhkan untuk menyimpan data yang digunakan oleh kode (jumlah memori untuk input dan output)
- Segmentasi memori: *code (text) segment*, *data segment*, *bss segment*, *heap segment*, dan *stack segment*.

Contoh perhitungan jumlah memori

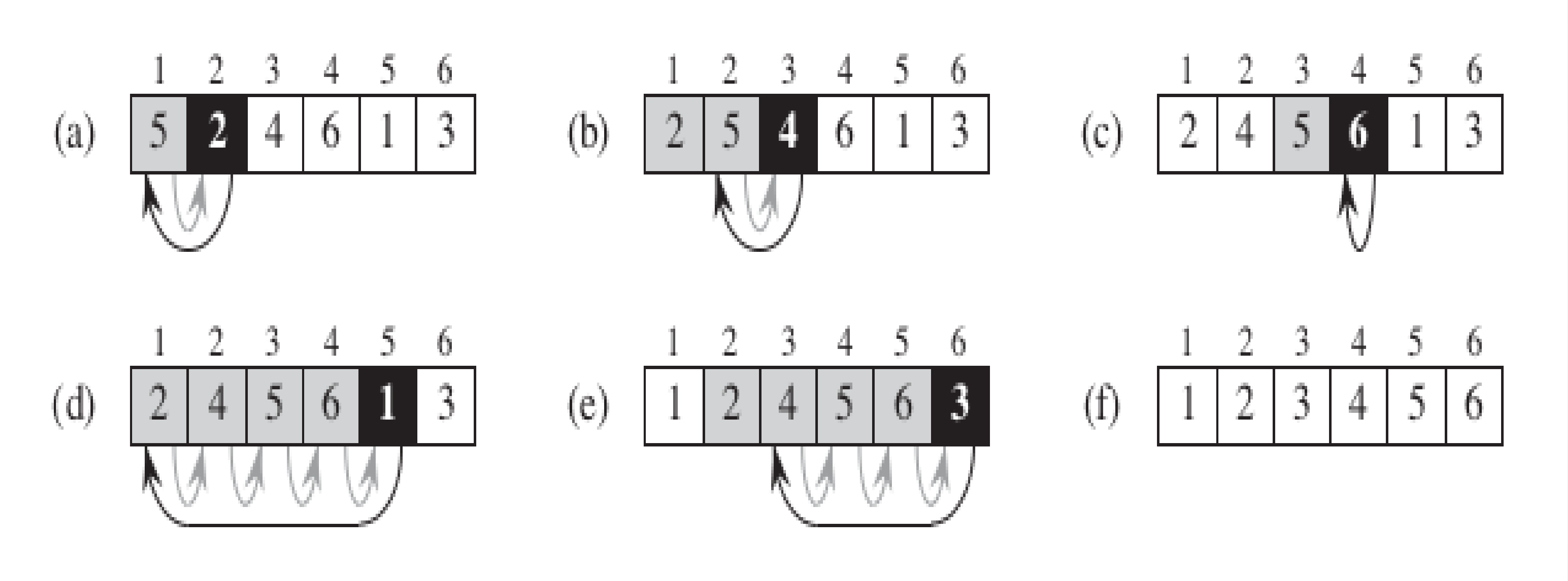
MAXIMUM SUBSEQUENCE PROBLEM: ALGORITMA 1

- Memori untuk kode: dapat anda lihat besar memori untuk executable file yang dihasilkan ketikan program dikompilasi.
- Memori untuk input/output: Hitung jumlah kebutuhan memori untuk variabel.

Contoh lain: insertion sort



Insertion sort



insertion sort algorithm

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

- **Calculate algorithm's bounds.**

Referensi

- T.H.Cormen, C.E. Leiserson, R.L. Rivest & C. Stein, *Introduction to Algoritms, 2nd eds*., MIT Press 2001.
- M.A.Weiss, *Data Structures and Algorithm Analysis in C, 2nd eds.*, Addison-Wesly, 1997
- R. Sedgewick, *Algorithms in C, Parts 1 -4, 3rd edition*, Addison-Wesley 1998.