Matematika Diskrit
ALGORITMA DAN KOMPLEKSITAS

Dosen: Haryadi
Program Studi Ilmu Komputer
Universitas Muhammadiyah Palangkaraya

May 2, 2026

1 Algoritma

Dalam masalah komputasi, ada banyak persoalan yang harus dicari penyelesaiannya. Misalnya, diketahui suatu barisan integer, masalah yang mungkin terjadi misalnya adalah bagaimana mencari nilai terbesarnya, mengurutkan dengan aturan tertentu atau mugkin mencari nilai tertentu pada barisan tersebut. Hal demikian memerlukan prosedur yang meliputi serangkaian langkah yang akan memandu pada jawaban persoalan tersebut.

Definisi 1. Algoritma *adalah barisan berhingga dari instruksi-instruksi untuk menyelesaikan suatu persoalan.*

Contoh 1. Gambarkan suatu algoritma untuk mencari nilai terbesar dalam suatu barisan berhingga bilangan integer.

PENYELESAIAN Persoalan tersebut dapat diselesaikan dengan langkah berikut.

1. Tetapkan suku pertama barisan tersebut sebagai maksimum sementara.

2. Bandingkan integer berikutnya dalam barisan tersebut dengan maksimum sementara. Jika suku tersebut lebih besar dari maksimum sementara, tetapkan suku tersebut sebagai maksimum sementara.

3. Ulangi langkah sebelumnya jika masih ada integer pada barisan tersebut.

4. Hentikan jika tidak ada lagi integer pada barisan tersebut. Maksimum sementara pada langkah ini merupakan nilai terbesar dalam barisan tersebut.

Suatu algoritma dapat juga dituliskan dengan bahasa komputer, namun hal ini akan sangat tergantung bahasa yang digunakan. Agar lebih mudah difahami orang banyak dapat digunakan *pseudocode*, yang menyediakan langkah antara deskripsi algoritma dalam bahasa manusia dan implementasi algoritma dalam bahasa pemrograman.

Deksripsi pseudocode untuk mencari nilai terbesar pada Contoh 1 dapat dinyatakan dalam barisan berhingga berikut.

ALGORITMA PENCARIAN NILAI MAKSIMUM SUATU DAFTAR BERHINGGA

prosedur $max(a_1, a_2, \cdots, a_n$: integers)
$max := a_1$
for $i := 2$ **to** n
    if $max < a_i$ **then** $max := a_i$
return max $\{max$ adalah nilai terbesar$\}$.

Algoritma memiliki sifat-sifat sebagai berikut:

- *Input.* Setiap algoritma memiliki nilai input tertentu.
- *Output.* Dari setiap nilai input, algoritma menghasilkan nilai output. Output ini merupakan penyelesaian masalah.
- *Ketertentuan.* Langkah-langkah dalam suatu algoritma harus ditentukan secara teliti.
- *Correctness.* Suatu algoritma harus menghasilkan output yang benar untuk setiap nilai input.
- *Keberhinggaan.* Suatu algoritma harus menghasilan output yang diinginkan setelah sejumlah berhingga langkah.
- *Keefektifan.* Setiap langkah dalam algoritma harus mungkin untuk dijalankan dalam jangka waktu berhingga.
- *Perumuman.* Prosedur seharusnya dapat diterapkan pada semua persoalan yang dikehendaki, bukan hanya pada nilai input tertentu saja.

Pencarian

Persoalan mencari elemen dalam daftar terurut terjadi dalam banyak hal. Persoalan pencarian secara umum dapat dideskripsikan sebagai berikut: Mencari elemen x di dalam suatu daftar elemen-elemen berbeda $a_1, a_2, \cdots, a_n$, atau menetapkan bahwa elemen x tersebut tidak ada di dalam daftar. Penyelesaian untuk persoalan ini adalah lokasi suku di dalam daftar tersebut yang sama dengan x (yaitu i adalah penyelesaian jika $x = a_i$), dan menetapkan 0 jika x tidak terdapat di dalam daftar tersebut.

Algoritma pencarian linear dimulai dengan membandingkan x dan a_1. Jika $x = a_1$, maka penyelesaiannya adalah lokasi a_1, yaitu 1. Jika $x \neq a_1$, bandingkan x dengan a_2. Jika $x = a_2$, penyelesainnya adalah lokasi a_2 yaitu 2. Jika $x \neq a_2$, bandingkan x dengan a_3. Lanjutkan proses pembandingan x dengan suku selanjutnya sampai kesamaan dicapai. Jika semua daftar telah dibandingkan nemun tidak ada yang sama dengan x, penyelesainnya adalah 0. Pseudocode untuk algoritma ini adalah sebagai berikut.

ALGORITMA PENCARIAN LINEAR

procedure $linearsearch(x : \text{integer}, a_1, a_2, \cdots, a_n : \text{ integer berbeda})$
$i := 1$
while $(i \leq n \text{ and } x \neq a_i)$
    $i := i + 1$
if $i \leq n$ **then** $location := i$
else $location := 0$
return $location$ $\{location$ adalah subskrip dari suku yang sama dengan x, atau 0 jika x tidak didapat$\}$

Algoritma pencarian biner dimulai dengan membandingkan unsur yang lokasinya di tengah daftar. Daftar tersebut dibelah menjadi dua subdaftar yang lebih kecil berukuran sama, atau salah satu subdaftar lebih sedikit elemennya dari subdaftar lainnya. Pencarian dibatasi pada sub daftar yang sesuai berdasarkan hasil pembandingan suku yang berlikasi di tengah.

Contoh 2. Akan dicari nilai 19 pada daftar berikut:

$$1 \quad 2 \quad 3 \quad 5 \quad 6 \quad 7 \quad 8 \quad 10 \quad 12 \quad 13 \quad 15 \quad 16 \quad 18 \quad 19 \quad 20 \quad 22.$$

Daftar di atas terlebih dahulu dijadikan dua kelompok masing-masing delapan suku, yaitu

$$1 \quad 2 \quad 3 \quad 5 \quad 6 \quad 7 \quad 8 \quad 10 \qquad \text{dan} \qquad 12 \quad 13 \quad 15 \quad 16 \quad 18 \quad 19 \quad 20 \quad 22.$$

Kemudian bandingkan 19 dan suku terbesar di dalam daftar pertama. Karena $10 < 19$, pencarian 19 dapat dibatasi pada daftar yanag berisi suku ke-9 sampai dengan ke-16. Selanjutnya kelompokan daftar ini menjadi dua masing-masing memuat empat suku, yaitu

$$12 \quad 13 \quad 15 \quad 16 \qquad \text{dan} \qquad 18 \quad 19 \quad 20 \quad 22.$$

Karena $16 < 19$, pencarian dibatasi pada dafar kedua yang berisi suku ke-13 sampai dengan ke-16. Selanjutnya daftar ini dikelompokan menjadi dua, yaitu

$$18 \quad 19 \qquad \text{dan} \qquad 20 \quad 22.$$

Karena 19 tidak lebih besar dibanding suku terbesar pada daftar pertama, (yang juga 19), pencarian dibatasi pada daftar 18 19, yang memuat suku ke-13 dan ke-14 dari daftar asli. Selanjutnya daftar terakhir dibagi dua, yaitu

$$18 \quad \text{dan} \quad 19.$$

Karena $18 < 19$, pencarian dibatasi pada daftar kedua yang memat suku ke-14 dari daftar asli, yaitu 19. Sekarang pencarian telah menyempit menjadi satu suku. Selanjutnya dilakukan pembandingan dan 19 berlokasi pada suku ke-14 dari daftar asli.

Untuk mencari integer x di dalam daftar $a_1, a_2, \cdots, a_n$, dimana $a_1 < a_2 < \cdots < a_n$, dimulai dengan membandingkan x dengan suku tengah a_m, dimana $m = \lfloor (n+1)/2 \rfloor$. Jika $x > a_m$, pencarian x dibatasi pada subdaftar kedua yaitu $a_{m+1}, a_{m+2}, \cdots, a_n$. Jika x tidak lebih besar dari a_m, pencarian x dibatasi pada subdaftar pertama yaitu $a_1, a_2, \cdots, a_m$.

Algoritma Pencarian Biner

procedur *binary search*(x : *interger*, $a_1, a_2, \cdots, a_n$: interger naik monoton)
$i := 1$ {i adalah titik ujung kiri dari integer yang akan dicari}
$j := n$ {j adalah titik ujung kanan dari interval yang akan dicari}
while $i < j$
    $m := \lfloor (i+j)/2 \rfloor$
    if $x \geq a_m$ **then** $i := m + 1$
    else $j := m$
if $x = a_j$ **then** *location* $:= i$
else *location* $:= 0$
return *location* {*location* adalah subskrip i dari suku a_i yang sama dengan x, atau 0 jika x tidak didapat}

Sorting

Misalkan suatu daftar elemen akan diurutkan. Pengurutan (*sorting*) adalah meletakan elemen-elemen sehingga menjadi daftar yang elemen-elemen dalam urutan naik. Misalnya $7, 2, 8, 3, 5$ menjadi $2, 3, 5, 7, 8$, daftar c, f, e, d, a menjadi a, c, d, e, f.

Bubble sort merupakan metode pengurutan paling sederhana, namun tidak efisien. Dalam metode ini, daftar diurutkan dengan secara berturut-turut membandingkan elemen berikutnya, menukarnya jika kedua elemen ini urutanya belum benar. Proses ini dimulai dari elemen pertama dan diulang hingga diperoleh daftar yang terurut.

Contoh 3. Gunakan bubble sort untuk mengurutkan daftar $3, 2, 4, 1, 5$.
Penyelesaian

Di dalam tahapan berikut, notasi $\leftrightarrow$ berarti ditukar dan — berarti urutan sudah benar.

Langkah-1:

$\overleftrightarrow{3 \quad 2} \quad 4 \quad 1 \quad 5$

$2 \quad \overline{3 \quad 4} \quad 1 \quad 5$

$2 \quad 3 \quad \overleftrightarrow{4 \quad 1} \quad 5$

$2 \quad 3 \quad 1 \quad \overline{4 \quad 5}$

Langkah-2:

$\overline{2 \quad 3} \quad 1 \quad 4 \quad 5$

$2 \quad \overleftrightarrow{3 \quad 1} \quad 4 \quad 5$

$2 \quad 1 \quad 3 \quad \overline{4 \quad 5}$

Langkah-3:

$\overleftrightarrow{2 \quad 1} \quad 3 \quad 4 \quad 5$

$1 \quad \overline{2 \quad 3} \quad 4 \quad 5$

Langkah- 4:

$\overline{1 \quad 2} \quad 3 \quad 4 \quad 5$

ALGORITMA BUBBLE SORT

procedure *bubble sort*$(a_1, a_2, \cdots, a_n$: bilangan real dengan $n \geq 2)$
for $i := 1$ **to** $n - 1$
    for $j := 1$ **to** $n - i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
$\{a_1, a_2, \cdots, a_n$ dalam urutan naik$\}$

algoritma **insertion sort** merupakan algoritma yang sederhana, namun biasanya bukan metode yang paling efisien. Untuk mengurutkan n elemen, metode ini dimulai dengan elemen kedua. Kemudian elemen kedua dibandingkan dengan elemen pertama dan sisipkan elemen kedua sebelum elemen pertama jika elemen kedua tidak lebih besar dari elemen pertama. Pada tahap ini kedua elemen dalam urutan yang benar. Selanjutnya elemen ketiga dibandingkan dengan elemen pertama, dan jika elemen ketiga lebih besar dari elemen pertama maka dibandingkan dengan elemen kedua; elemen ketiga disisipkan ke posisi yang benar diantara ketiga elemen tersebut. Langkah ini dilanjutkan hingga dicapai daftar yang urutan yang benar.

Contoh 4. Urutkan daftar 3 2 4 1 5 dalam urutan naik.
PENYELESAIAN Dimulai dengan membandingkan 2 dan 3. Karena $3 > 2$, maka 2 pada posisi pertama sehingga menghasilkan daftar $2, 3, 4, 1, 5$. Pada tahap ini 2 dan 3 pada urutan yang benar. Selanjutnya elemen ketiga (4) disisipkan kedalam bagian daftar yang sudah terurut dengan membandingkan $4 > 2$ dan $4 > 3$. Karena $4 > 3$, 4 tetap pada posisi ketiga. Pada tahap ini diperoleh daftar $2, 3, 4, 1, 5$ dan urutan tiga elemen pertama sudah benar. Selanjutnya dicari posisi yang benar untuk elemen keempat (yaitu 1) diantara elemen-elemen yang sudah urut, yaitu $2, 3, 4$. Karena $1 < 2$ maka diperoleh daftar $1, 2, 3, 4, 5$. Akhirnya 5 disisipkan pada posisi yang benar dengan secara berurutan membandingkan dengan $1, 2, 3$ dan 4. Karena $5 > 4$ maka 5 tetap pada posisi semula.

ALGORITMA INSERTION SORT

```
prosedur insertion sort(a_1, a_2, ..., a_n : bilangan real dengan n >= 2)
for j := 2 to n
        i := 1
        while a_j > a_i
              i := i + 1
        m := a_j
        for k := 0 to j - i - 1
              a_{j-k} := a_{j-k-1}
        a_i := m
{a_1, a_2, ..., a_n dalam urutan naik}
```

2 Pertumbuhan fungsi dan kompleksitas algoritma

Ada lebih satu cara untuk menyelesaian suatu persoalan. Pilihan algoritma untuk memperoleh output jawaban suatu persoalan tergantung pada "efisiensi" dan "kompleksitas" algoritma.

Definisi 2. *Diberikan fungsi f dan g. Fungsi $f(x)$ adalah $O(g(x))$ jika terdapat konstanta C dan k sehingga*

$$|f(x)| \leq C|g(x)|$$

jika $x > k$.

$f(x)$ adalah $O(g(x))$ dibaca "$f(x)$ adalah big-O dari $g(x)$.

Contoh 5. Tunjukkan bahwa $f(x) = x^2 + 2x + 1$ adalah $O(x^2)$.

PENYELESAIAN Karena untuk $x > 1$, $|x| < |x^2|$ dan $1 < |x^2|$, maka

$$\begin{aligned} |f(x)| = |x^2 + 2x + 1| &\leq |x^2| + 2|x| + 1 \\ &\leq |x^2| + 2|x^2| + |x^2| = 4|x^2| \end{aligned}$$

Dengan $g(x) = x^2$, $C = 4$ dan $k = 1$ disimpulkan bahwa $f(x)$ adalah $O(x^2)$.

Notasi $f(x) = O(g(x))$ berarti $f(x)$ adalah big-O dari $g(x)$. Jadi pada Contoh 5, $f(x) = O(x^2)$.

Definisi 3. *Diberikan fungsi f dan g. Fungsi $f(x) \in \Omega(g(x))$ jika terdapat konstanta C dan k sehingga*

$$|f(x)| \geq C|g(x)|$$

dimana $x > k$.

Selanjutnya $f(x) \in \Omega(g(x))$ dibaca "$f(x)$ adalah big-Omega dari $g(x)$".

Contoh 6. Fungsi $f(x) = 8x^3 + 5x^2 + 7$ adalah $\Omega(g(x))$ dengan $g(x) = x^3$. Hal ini karena $f(x) = 8x^3 + 5x^2 + 7 \geq 8x^3$ untuk semua $x > 0$.

Definisi 4. *Diberikan fungsi f dan g. Dikatakan $f(x)$ adalah $\Theta(g(x))$ jika $f(x)$ adalah $O(g(x))$ dan $f(x)$ adalah $\Omega(g(x))$. Jika $f(x)$ adalah $\Theta(g(x))$ maka $f(x)$ dan $g(x)$ dikatakan berorde sama.*

$f(x)$ adalah $\Theta(g(x))$ dibaca "$f(x)$ adalah big-Theta dari $g(x)$. Berdasarkan definisi 4, $f(x)$ adalah $\Theta(g(x))$ jika dan hanya jika terdapat konstanta C_1 dan C_2 dan bilangan $k > 0$ sehingga

$$C_1|g(x)| \leq f(x) \leq C_2|g(x)|$$

dengan $x > k$.

Contoh 7. Tunjukkan bahwa fungsi $f(n) = 1 + 2 + 3 + \cdots + n$ adalah $\Theta(n^2)$.

PENYELESAIAN

$$f(n) = 1 + 2 + 3 + \cdots + n \leq n + n + n + \cdots + n = n^2$$

yakni $f(x)$ adalah $O(n^2)$. Selanjutnya dari jumlah $1+2+3+\cdots+n$ dijumlahkan suku-suku yang lebih besar dan $\lceil n/2 \rceil$ saja, diperoleh

$$\begin{aligned} f(n) = 1 + 2 + 3 + \cdots + n &\geq \lceil n/2 \rceil + (\lceil n/2 \rceil + 1) + \cdots + n \\ &\geq \lceil n/2 \rceil + \lceil n/2 \rceil + \cdots + \lceil n/2 \rceil \\ &= (n - \lceil n/2 \rceil + 1)\lceil n/2 \rceil \\ &\geq (n/2)(n/2) \\ &= n^2/4, \end{aligned}$$

yang berarti $f(n)$ adalah $\Omega(n^2)$. Dengan demikian $f(n)$ adalah $\Theta(n^2)$.

Analisis mengenai algoritma merupakan pekerjaan utama dalam ilmu komputer. Untuk bisa membandingkan beberapa algoritma, harus ada kriteria untuk mengukur efesiensi algoritma. Misalnya suatu algoritma memiliki ukuran input data n. Waktu dan ruang yang digunakan oleh algoritma merupakan dua ukuran utama efisiensi algoritma. Waktu tersebut diukur dengan menghitung jumlah "operasi kunci".

Contoh 8. Di dalam algoritma *sorting* dan *searching*, dilakukan penghitungan banyaknya pembandingan. Di dalam algoritma aritmetika, dihitung perkalian dan penjumlahan.

Operasi kunci ditentukan jika waktu untuk operasi lain adalah lebih sedikit atau paling banyak proposional dengan waktu operasi kunci. Ruang diukur dengan menghitung memory maksimum yang diperlukan oleh algoritma.

Kompleksitas algoritma adalah fungsi dengan domain integer positif $f(n)$ yang memberikan waktu berjalan atau atau ruang penyimpanan yang diperlukan algoritma dalam ukuran n data input. Seringkali, ruang penyimpanan yang dibutuhkan oleh algoritma hanyalah kelipatan dari ukuran data. Dengan demikian istilah "kompleksitas" harus mengacu pada waktu berjalan algoritma.

Fungsi kompleksitas $f(n)$, yang dianggap memberikan waktu berjalan algoritma, biasanya tergantung pada tidak hanya pada ukuran n data input tetapi juga pada data tertentu.

Kasus yang biasanya diselidiki dalam teori kompleksitas adalah sebagai berikut:

(1) Kasus terburuk, yaitu nilai maksimum $f(n)$ untuk setiap input yang mungkin.

(2) Kasus rata-rata, yaitu nilai yang diharapkan dari $f(n)$.

Analisis kasus rata-rata mengasumsikan distribusi probabilistik tertentu untuk data input. Rata-rata kasus juga mengikuti konsep dalam teori probabilitas. Misalkan $n_1, n_2, ..., n_k$ terjadi dengan probabilitas masing-masing $p_1, p_2, ..., p_k$. Kemudian nilai harapan atau nilai rata-rata E diberikan oleh

$$E = n_1p_1 + n_2p_2 + \quad + n_kp_k.$$

Misalkan suatu algoritma memiliki data input berukuran n. Kompleksitas $f(n)$ meningkat seiring dengan meningkatnya n. Laju peningkatan $f(n)$ dilakukan dengan membandingkan dengan beberapa fungsi standar, seperti

$$\log n, \quad n, \quad n \log n, \quad n^2, \quad n^3, \quad 2^n.$$

Untuk beberapa nilai n, tingkat pertumbuhan fungsi-fungsi ini diberikan pada tabel berikut.

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

Contoh 9. Gambarkan bagaimana kompleksitas algoritma untuk mencari nilai maksimum dari sejumlah berhingga integer.

PENYELESAIAN Banyaknya pembandingan akan digunakan sebagai ukuran kompleksitas waktu, karena pembandingan merupakan operasi kunci dalam algoritma ini. Misalkan banyaknya integer yang akan dicari nilai maksimumnya adalah n dalam urutan sebarang. Pembandingan dimulai dengan elemen kedua hingga ke n, sehingga ada $2(n-1)$ pembandingan. Satu pembandingan untuk keluar dari loop dilakukan pada $i = n+1$. Dengan demikian banyaknya pembandingan adalah $2(n-1)+1 = 2n-1$. Jadi algoritma pencarian maksimum ini memiliki kompleksitas waktu $O(n)$ atau dinamakan *kompleksitas linear*.

Terminologi umum yang digunakan untuk menggambarkan kompleksitas algoritma disajikan pada tabel berikut.

Kompleksitas	**Terminologi**
$\Theta(1)$	Kompleksitas konstan
$\Theta(\log n)$	Kompleksitas logaritma
$\Theta(n)$	Kompleksitas linear
$\Theta(n \log n)$	Kompleksitas lineararitmetik
$\Theta(n^b)$	Kompleksitas Polinomial
$\Theta(n^b),\ b > 1$	Kompleksitas eksponensial
$\Theta(n!)$	Kompleksitas faktorial

Tugas

1. Diantara fungsi-fungsi berikut mana yang $O(1)$, maka yang $O(x)$ dan mana yang $O(x^2)$.
   (a) $f(x) = 10$
   (b) $f(x) = x^2 + x + 1$
   (c) $f(x) = \lfloor x \rfloor$
2. Tunjukkan bahwa $f(x) = 3x^2 + x + 1$ adalah $\Theta(3x^2)$.
3. Prediksikan kempleksitas untuk algoritma berikut.

   $t := 0$
       for $i := 1$ **to** 3
           for$j := 1$ **to** 4
               $t := t + ij$
4. Prediksikan kompleksitas algoritma berikut.

   prosedur *perkalian matriks* (A,B:matriks)
   for $i := 1$ **to** m
       for $j := 1$ **to** n
       $c_{ij} = 0$

for $q := 1$ **to** k

$c_{ij} := c_{ij} + a_{iq}b_{qj}$

return C {C =(c_{ij}) adalah hasil kali A dan B}.