Fakultas Informatika
School of Computing
Telkom University

CII2C2
Analisis Kompleksitas Algoritma

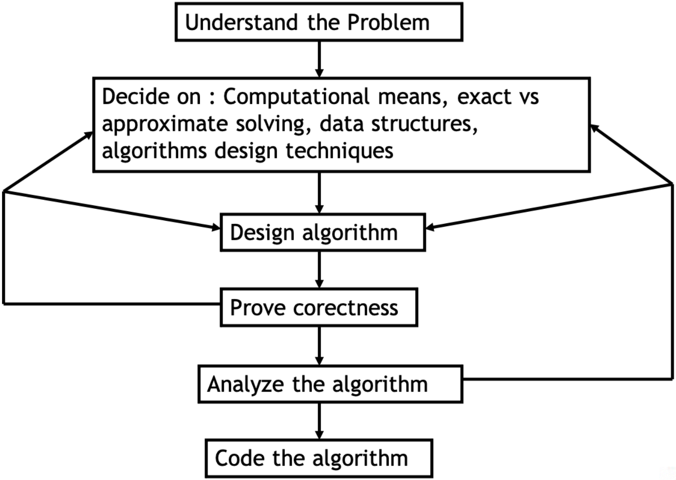
Pertemuan 1-2
TIME COMPLEXITY

Outline

1. Analisis Algoritma
2. Input Size dan Operasi Dasar
3. Efisiensi Algoritma
4. Latihan 1
5. Contoh perhitungan komplesitas waktu

Analisis Algoritma

Dasar dari *Algorithmic Problem Solving*



Algoritma tidak cukup hanya benar saja (*correct*), tetapi juga harus efisien.

Tujuan Analisis Algoritma

- Pengukuran efisiensi algoritma dari segi waktu/ruang memori
- Perbandingan dua atau lebih algoritma untuk masalah yang sama

Input Size dan Operasi Dasar

Pengukur kompleksitas waktu: *input size* (1)

- Hampir semua algoritma membutuhkan waktu yang lebih lama untuk *input* yang lebih besar, contoh:
  - Pengurutan array
  - Perkalian matriks
- Untuk menghitung keefisienan suatu algoritma, kita perlu menyelidiki parameter n yang mengindikasikan *input size* dari algoritma tersebut
  - Pengurutan, pencarian, penentuan bilangan terbesar
  - Perhitungan polinomial $p(x) = a_n x^n + \ldots + a_0$
  - Perkalian matriks $A \times B$

Pengukur kompleksitas waktu: *input size* (1)

- Hampir semua algoritma membutuhkan waktu yang lebih lama untuk *input* yang lebih besar, contoh:
  - Pengurutan array
  - Perkalian matriks
- Untuk menghitung keefisienan suatu algoritma, kita perlu menyelidiki parameter n yang mengindikasikan *input size* dari algoritma tersebut
  - Pengurutan, pencarian, penentuan bilangan terbesar $\rightarrow$ukuran dari list
  - Perhitungan polinomial $p(x) = a_n x^n + \ldots + a_0$
  - Perkalian matriks $A \times B$

Pengukur kompleksitas waktu: *input size* (1)

- Hampir semua algoritma membutuhkan waktu yang lebih lama untuk *input* yang lebih besar, contoh:
  - Pengurutan array
  - Perkalian matriks
- Untuk menghitung keefisienan suatu algoritma, kita perlu menyelidiki parameter n yang mengindikasikan *input size* dari algoritma tersebut
  - Pengurutan, pencarian, penentuan bilangan terbesar $\rightarrow$ukuran dari list
  - Perhitungan polinomial $p(x) = a_n x^n + \ldots + a_0$ $\rightarrow$derajat polinomial/banyaknya koefisien
  - Perkalian matriks $A \times B$

Pengukur kompleksitas waktu: *input size* (1)

- Hampir semua algoritma membutuhkan waktu yang lebih lama untuk *input* yang lebih besar, contoh:
  - Pengurutan array
  - Perkalian matriks
- Untuk menghitung keefisienan suatu algoritma, kita perlu menyelidiki parameter n yang mengindikasikan *input size* dari algoritma tersebut
  - Pengurutan, pencarian, penentuan bilangan terbesar
    $\rightarrow$ukuran dari list
  - Perhitungan polinomial $p(x) = a_n x^n + \ldots + a_0$
    $\rightarrow$derajat polinomial/banyaknya koefisien
  - Perkalian matriks $A \times B$
    $\rightarrow$ukuran matriks

Pengukur kompleksitas waktu: *input size* (2)

Tentukan *input size* dari permasalahan berikut:

1. Algoritma pengecekan pengejaan

2. Pengecekan bilangan prima

Pengukur kompleksitas waktu: *input size* (2)

Tentukan *input size* dari permasalahan berikut:

1. Algoritma pengecekan pengejaan
   →banyaknya huruf
2. Pengecekan bilangan prima

Pengukur kompleksitas waktu: *input size* (2)

Tentukan *input size* dari permasalahan berikut:

1. Algoritma pengecekan pengejaan
   → banyaknya huruf
2. Pengecekan bilangan prima
   → bilangan yang akan dicek

Pengukur kompleksitas waktu: operasi dasar

Yang dianggap sebagai operasi dasar (*basic operation*):

- Operasi terpenting dari suatu algoritma
- Operasi yang memakan waktu paling banyak di iterasi terdalam dari suatu algoritma

Perhitungan kompleksitas waktu

Perhitungan berapa kali algoritma melakukan operasi dasar pada *input size* n

Running time

- Approksimasi *running time* ($T(n)$) dari suatu algoritma dapat dihitung dengan:

  $$T(n) \approx c_{op}C(n)$$

  dimana
  c_{op} menyatakan waktu eksekusi dari operasi dasar algoritma tersebut pada suatu komputer
  $C(n)$ menyatakan berapa kali operasi dasar dari algoritma tersebut dieksekusi pada keseluruhan algoritma tersebut

Running time

- Approksimasi *running time* ($T(n)$) dari suatu algoritma dapat dihitung dengan:
$$T(n) \approx c_{op}C(n)$$
dimana
c_{op} menyatakan waktu eksekusi dari operasi dasar algoritma tersebut pada suatu komputer
$C(n)$ menyatakan berapa kali operasi dasar dari algoritma tersebut dieksekusi pada keseluruhan algoritma tersebut
- Perlu diperhatikan bahwa perhitungan di atas hanya menghitung nilai aproksimasi, karena:
  - $C(n)$ hanya menghitung operasi dasar dan tidak melihat operasi lain selain operasi dasar yang mungkin saja ada di dalam algoritma tersebut
  - c_{op} pun biasanya tersedia dalam bentuk estimasi

Waktu eksekusi operasi dasar

Perhatikan bahwa dari perhitungan aproksimasi *running time* yang kita miliki:

$$T(n) \approx c_{op}C(n)$$

- Nilai $C(n)$ bergantung pada berapa banyak suatu operasi dasar dieksekusi
- Nilai c_{op} bergantung pada seberapa cepat satu operasi dasar tersebut dapat dieksekusi oleh komputer

Waktu eksekusi operasi dasar

Perhatikan bahwa dari perhitungan aproksimasi *running time* yang kita miliki:

$$T(n) \approx c_{op}C(n)$$

- Nilai $C(n)$ bergantung pada berapa banyak suatu operasi dasar dieksekusi
- Nilai c_{op} bergantung pada seberapa cepat satu operasi dasar tersebut dapat dieksekusi oleh komputer
- Misalkan, dari empat operasi aritmetika: penambahan, pengurangan, perkalian, dan pembagian; pembagian dikenal sebagai operasi yang memiliki waktu eksekusi paling banyak, disusu oleh perkalian dan kemudian penambahan dan pengurangan (dua operasi terakhir biasanya dianggap setara)[1]

Efisiensi Algoritma

Contoh perbandingan efisiensi algoritma

- Terdapat dua algoritma:
  - Insertion sort: mengeksekusi n^2 operasi dasar
  - Merge sort: mengeksekusi $n \log n$ operasi dasar

Contoh perbandingan efisiensi algoritma

- Terdapat dua algoritma:
  - Insertion sort: mengeksekusi n^2 operasi dasar
  - Merge sort: mengeksekusi $n \log n$ operasi dasar
- Terdapat pula dua hardware:
  - Kecepatan komputer A: satu milyar instruksi per detik
  - Kecepatan komputer B: sepuluh juta instruksi per detik

Contoh perbandingan efisiensi algoritma

- Terdapat dua algoritma:
  - Insertion sort: mengeksekusi n^2 operasi dasar
  - Merge sort: mengeksekusi $n \log n$ operasi dasar
- Terdapat pula dua hardware:
  - Kecepatan komputer A: satu milyar instruksi per detik
  - Kecepatan komputer B: sepuluh juta instruksi per detik
- Komputer A digunakan untuk eksekusi insertion sort dan komputer B digunakan untuk eksekusi merge sort

Contoh perbandingan efisiensi algoritma

- Terdapat dua algoritma:
  - Insertion sort: mengeksekusi n^2 operasi dasar
  - Merge sort: mengeksekusi $n \log n$ operasi dasar
- Terdapat pula dua hardware:
  - Kecepatan komputer A: satu milyar instruksi per detik
  - Kecepatan komputer B: sepuluh juta instruksi per detik
- Komputer A digunakan untuk eksekusi insertion sort dan komputer B digunakan untuk eksekusi merge sort
- Waktu eksekusi operasi dasar insertion sort setara dengan waktu eksekusi 2 instruksi di komputer A dan waktu eksekusi operasi dasar merge sort setara dengan 50 instruksi di komputer B
- Waktu yang dibutuhkan komputer A dan B untuk mengurutkan bilangan sebanyak satu juta ($n = 10^6$):

Contoh perbandingan efisiensi algoritma

- Terdapat dua algoritma:
  - Insertion sort: mengeksekusi n^2 operasi dasar
  - Merge sort: mengeksekusi $n \log n$ operasi dasar
- Terdapat pula dua hardware:
  - Kecepatan komputer A: satu milyar instruksi per detik
  - Kecepatan komputer B: sepuluh juta instruksi per detik
- Komputer A digunakan untuk eksekusi insertion sort dan komputer B digunakan untuk eksekusi merge sort
- Waktu eksekusi operasi dasar insertion sort setara dengan waktu eksekusi 2 instruksi di komputer A dan waktu eksekusi operasi dasar merge sort setara dengan 50 instruksi di komputer B
- Waktu yang dibutuhkan komputer A dan B untuk mengurutkan bilangan sebanyak satu juta ($n = 10^6$):

  $\rightarrow$ Komputer A: $\frac{2.n^2}{10^9} = \frac{(2.10^6)^2}{10^9} = 2000$ detik

  $\rightarrow$ Komputer B: $\frac{50.n \log n}{10^7} = \frac{50.10^6 \log(10^6)}{10^7} = 30$ detik

Insertion sort vs merge sort

Dari contoh tersebut, dapat kita lihat:

- Banyaknya eksekusi operasi dasar pada insertion sort (n^2) lebih banyak dibandingkan dengan banyaknya eksekusi operasi dasar pada merge sort ($n \log n$)
- Perhatikan bahwa waktu eksekusi satu operasi dasar insertion sort lebih cepat daripada waktu eksekusi satu operasi dasar merge sort
- Walaupun insertion sort dioperasikan pada komputer yang lebih cepat, *running time* insertion sort tetap lebih lambat dibandingkan dengan *running time* merge sort

Ilustrasi efisiensi algoritma (1)

- Misalkan terdapat dua algoritma:
  - Algoritma I: mengeksekusi n^3 operasi dasar
  - Algoritma II: mengeksekusi 2^n operasi dasar
- ~~Misalkan terdapat dua komputer:~~
  - ~~Komputer A: mempunyai kecepatan 1.000.000 instruksi per detik~~
  - ~~Komputer B: mempunyai kecepatan 10.000 instruksi per detik~~
- Diketahui bahwa:
  - Waktu eksekusi operasi dasar Algoritma I setara dengan waktu eksekusi 80 instruksi
  - Waktu eksekusi operasi dasar Algoritma II setara dengan waktu eksekusi 1 instruksi

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

		input size (n)					
		5	10	15	20	$\ldots$	40
I-A	$10^{-6} \times n^3$					$\ldots$	
I-B	$10^{-4} \times n^3$					$\ldots$	
II-A	$10^{-6} \times 2^n$					$\ldots$	
II-B	$10^{-4} \times 2^n$					$\ldots$	

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

		input size (n)					
		5	10	15	20	$\ldots$	40
I-A	$10^{-6} \times n^3$	0, 01				$\ldots$	
I-B	$10^{-4} \times n^3$					$\ldots$	
II-A	$10^{-6} \times 2^n$					$\ldots$	
II-B	$10^{-4} \times 2^n$					$\ldots$	

$10^{-6} \times 5^3 \times 80$

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

		input size (n)					
		5	10	15	20	$\ldots$	40
I-A	$10^{-6} \times n^3$	$0,01$				$\ldots$	
I-B	$10^{-4} \times n^3$	1				$\ldots$	
II-A	$10^{-6} \times 2^n$					$\ldots$	
II-B	$10^{-4} \times 2^n$					$\ldots$	

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

		input size (n)					
		5	10	15	20	$\ldots$	40
I-A	$10^{-6} \times n^3$	$0,01$				$\ldots$	
I-B	$10^{-4} \times n^3$	1				$\ldots$	
II-A	$10^{-6} \times 2^n$	$0,000032$				$\ldots$	
II-B	$10^{-4} \times 2^n$					$\ldots$	

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

		input size (n)					
		5	10	15	20	$\ldots$	40
I-A	$10^{-6} \times n^3$	$0,01$				$\ldots$	
I-B	$10^{-4} \times n^3$	1				$\ldots$	
II-A	$10^{-6} \times 2^n$	$0,000032$				$\ldots$	
II-B	$10^{-4} \times 2^n$	$0,0032$				$\ldots$	

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

		input size (n)					
		5	10	15	20	$\ldots$	40
I-A	$10^{-6} \times n^3$	$0,01$	$0,08$			$\ldots$	
I-B	$10^{-4} \times n^3$	1	8			$\ldots$	
II-A	$10^{-6} \times 2^n$	$0,000032$	$0,001$			$\ldots$	
II-B	$10^{-4} \times 2^n$	$0,0032$	$0,102$			$\ldots$	

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

		input size (n)					
		5	10	15	20	...	40
I-A	$10^{-6} \times n^3$	$0,01$	$0,08$	$0,27$		...	
I-B	$10^{-4} \times n^3$	1	8	27		...	
II-A	$10^{-6} \times 2^n$	$0,000032$	$0,001$	$0,03$		...	
II-B	$10^{-4} \times 2^n$	$0,0032$	$0,102$	$3,28$		...	

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

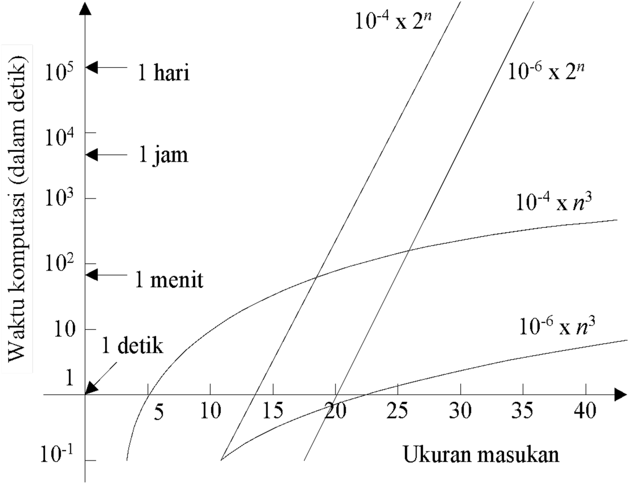
		input size (n)					
		5	10	15	20	$\ldots$	40
I-A	$10^{-6} \times n^3$	$0,01$	$0,08$	$0,27$	$0,64$	$\ldots$	
I-B	$10^{-4} \times n^3$	1	8	27	64	$\ldots$	
II-A	$10^{-6} \times 2^n$	$0,000032$	$0,001$	$0,03$	$1,05$	$\ldots$	
II-B	$10^{-4} \times 2^n$	$0,0032$	$0,102$	$3,28$	$104,9$	$\ldots$	

Ilustrasi efisiensi algoritma (2)

Running time (dalam detik) Algoritma I dan II pada komputer A dan B:

		input size (n)					
		5	10	15	20	$\ldots$	40
I-A	$10^{-6} \times n^3$	$0,01$	$0,08$	$0,27$	$0,64$	$\ldots$	$5,12$
I-B	$10^{-4} \times n^3$	1	8	27	64	$\ldots$	512
II-A	$10^{-6} \times 2^n$	$0,000032$	$0,001$	$0,03$	$1,05$	$\ldots$	$1.099.512$
II-B	$10^{-4} \times 2^n$	$0,0032$	$0,102$	$3,28$	$104,9$	$\ldots$	$109.951.163$

Ilustrasi efisiensi algoritma (3)



Insertion sort vs merge sort

Dari kedua contoh yang ada, dapat kita lihat:

- Banyaknya eksekusi operasi dasar sangat berpengaruh pada *running time* algoritma tersebut
- Waktu eksekusi satu operasi dasar tidak terlalu berpengaruh jika kita berbicara mengenai data dengan ukuran masukan yang besar
- Oleh karenanya, kompleksitas waktu biasanya dihitung berdasarkan banyaknya operasi dasar dieksekusi pada suatu algoritma (tanpa melihat c_{op})

Latihan 1

Latihan 1

1. Buatlah program menggunakan algoritma sequential search dan binary search:
    - Input: bilangan bulat positif n
    - Bangkitkan bilangan bulat acak sebanyak n
    - Bangkitkan satu bilangan bulat x
    - Cari x pada bilangan-bilangan tersebut dengan menggunakan sequential search, catat *running time*-nya
    - Cari x pada bilangan-bilangan tersebut dengan menggunakan binary search, catat *running time*-nya
2. Buatlah tabel komparasi efisiensi untuk ukuran data (n):
    - 32
    - 1.024
    - 32.768

Recall: sequential search dan binary search

```
procedure SeqSearch(input A: Tabint; x, n: integer
                    output position: integer)
   position←1
   while (position <= n and A[position] ≠ x) do
      position ← position + 1
   if position > n then
      position ← 0
```

```
procedure binsearch(input A: Tabint; x, n: integer
                    output position: integer)

low, high, mid: integer

   low←1; high← n; position← 0;
   while (low <= high and position = 0) do
      mid ← ⌊(low + high)/2)⌋
      if x = A[mid] then
         position ← mid
      else if x < A[mid] then
         high ← mid − 1
      else
         low ← mid + 1
```

Contoh perhitungan komplesitas waktu

Pencarian elemen terbesar (1)

```
procedure CariElemenTerbesar(input a1, a2, ..., an: integer,
                             output maks: integer)

{ Mencari elemen terbesar dari sekumpulan elemen larik integer a1,
a2, ..., an.
  Elemen terbesar akan disimpan di dalam maks.
  Masukan: a1, a2, ..., an
  Keluaran: maks (nilai terbesar)
}
Deklarasi
 k : integer
Algoritma
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   - Cek bagian terpenting/yang paling banyak memakan waktu

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   - Cek bagian terpenting/yang paling banyak memakan waktu
   - Tentukan operasi dasar dari bagian tersebut

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?

   operasi perbandingan `ak > maks`

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan `ak > maks`
2. Berapa kali operasi dasar tersebut dioperasikan?

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan `ak > maks`
2. Berapa kali operasi dasar tersebut dioperasikan?
   - Cek variabel penentu iterasi

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan `ak > maks`
2. Berapa kali operasi dasar tersebut dioperasikan?
   - Cek variabel penentu iterasi
   - Cek berapa kali iterasi dapat dilakukan
     $k = 2, \ldots, n \rightarrow n - 1$ iterasi

Pencarian elemen terbesar (2)

```
procedure CariElemenTerbesar(a1, a2, ..., an)
 maks ← a1
 k ← 2
 while k <= n do
  if ak > maks then
   maks ← ak
  endif
  k ← k + 1
 endwhile
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan `ak > maks`
2. Berapa kali operasi dasar tersebut dioperasikan?
   $n - 1$ kali

Jadi, $T(n) = n - 1$

Selection sort (1)

```
procedure Urut (input a1,...,an : integer,
                             output a1,...,an : integer)

Deklarasi
   i, j, imaks, temp : integer

Algoritma
   for i ← n downto 2 do
      imaks ← 1
      for j ← 2 to i do
         if aj > aimaks then
            imaks ← j
         endif
      endfor
      temp ← ai
      ai ← aimaks
      aimaks ← temp
   endfor
```

Selection sort (2)

```
procedure Urut (a1,...,an : integer)
   for i ← n downto 2 do
      imaks ← 1
      for j ← 2 to i do
         if aj > aimaks then
            imaks ← j
         endif
      endfor
      temp ← ai
      ai ← aimaks
      aimaks ← temp
   endfor
```

Selection sort (2)

```
procedure Urut (a1,...,an : integer)
   for i ← n downto 2 do
      imaks ← 1
      for j ← 2 to i do
         if aj > aimaks then
            imaks ← j
         endif
      endfor
      temp ← ai
      ai ← aimaks
      aimaks ← temp
   endfor
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?

Selection sort (2)

```
procedure Urut (a1,...,an : integer)
   for i ← n downto 2 do
       imaks ← 1
       for j ← 2 to i do
          if aj > aimaks then
             imaks ← j
          endif
       endfor
       temp ← ai
       ai ← aimaks
       aimaks ← temp
   endfor
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   - Cek bagian terpenting/yang paling banyak memakan waktu

Selection sort (2)

```
procedure Urut (a1,...,an : integer)
   for i ← n downto 2 do
       imaks ← 1
       for j ← 2 to i do
          if aj > aimaks then
             imaks ← j
          endif
       endfor
       temp ← ai
       ai ← aimaks
       aimaks ← temp
   endfor
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   - Cek bagian terpenting/yang paling banyak memakan waktu
   - Tentukan operasi dasar dari bagian tersebut

Selection sort (2)

```
procedure Urut (a1,...,an : integer)
   for i ← n downto 2 do
       imaks ← 1
       for j ← 2 to i do
          if aj > aimaks then
             imaks ← j
          endif
       endfor
       temp ← ai
       ai ← aimaks
       aimaks ← temp
    endfor
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?

   operasi perbandingan $a_j > a_{imaks}$

   operasi pertukaran a_j dan a_{imaks}

Selection sort (2)

```
procedure Urut (a1,...,an : integer)
   for i ← n downto 2 do
       imaks ← 1
       for j ← 2 to i do
          if aj > aimaks then
             imaks ← j
          endif
       endfor
       temp ← ai
       ai ← aimaks
       aimaks ← temp
    endfor
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?

   operasi perbandingan $a_j > a_{imaks}$

   operasi pertukaran a_j dan a_{imaks}

2. Berapa kali operasi dasar tersebut dioperasikan?

Selection sort (2)

```
procedure Urut (a1,...,an : integer)
   for i ← n downto 2 do
       imaks ← 1
       for j ← 2 to i do
          if aj > aimaks then
             imaks ← j
          endif
       endfor
       temp ← ai
       ai ← aimaks
       aimaks ← temp
    endfor
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan $a_j > a_{imaks}$
   operasi pertukaran a_j dan a_{imaks}
2. Berapa kali operasi dasar tersebut dioperasikan?
   - Cek variabel penentu iterasi

Selection sort (2)

```
procedure Urut (a1,...,an : integer)
   for i ← n downto 2 do
       imaks ← 1
       for j ← 2 to i do
          if aj > aimaks then
             imaks ← j
          endif
       endfor
       temp ← ai
       ai ← aimaks
       aimaks ← temp
    endfor
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?

   operasi perbandingan $a_j > a_{imaks}$

   operasi pertukaran a_j dan a_{imaks}

2. Berapa kali operasi dasar tersebut dioperasikan?
   - Cek variabel penentu iterasi
   - Cek berapa kali iterasi dapat dilakukan

Operasi perbandingan

```
for i ← n downto 2 do
```

$\longrightarrow i = n, n-1, n-2, \ldots, 2$

```
    for j ← 2 to i do
```

$$
\begin{array}{lll}
\longrightarrow i = n & \Rightarrow j = 2, 3, \ldots, n & \Rightarrow n-1 \text{ kali perbandingan} \\
i = n-1 & \Rightarrow j = 2, 3, \ldots, n-1 & \Rightarrow n-2 \text{ kali perbandingan} \\
i = n-2 & \Rightarrow j = 2, 3, \ldots, n-2 & \Rightarrow n-3 \text{ kali perbandingan} \\
\vdots & & \\
i = 3 & \Rightarrow j = 2, 3 & \Rightarrow 2 \text{ kali perbandingan} \\
i = 2 & \Rightarrow j = 2 & \Rightarrow 1 \text{ kali perbandingan}
\end{array}
$$

TOTAL: $n-1+n-2+\ldots+2+1 = \dfrac{n(n-1)}{2}$ kali perbandingan

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

Operasi pertukaran

```
for i ← n downto 2 do
```

$\longrightarrow i = n, n-1, \ldots, 2 \qquad \Rightarrow n-1$ kali pertukaran

$T(n) = n - 1$

Kompleksitas Waktu Selection Sort

Jadi, algoritma pengurutan maksimum membutuhkan $n(n–1)/2$ buah operasi perbandingan elemen dan $n–1$ buah operasi pertukaran.

Worst case, best case, average case

- Pada kedua contoh sebelumnya, banyaknya iterasi tidak ditentukan pada data dari masukan yang diberikan
- Jika banyaknya iterasi bergantung pada data dari masukan yang ada, maka kita dapat membagi kompleksitas waktu menjadi:
  - Best case ($T_{min}(n)$)
    kebutuhan waktu untuk kasus terbaik (kebutuhan waktu minimum)
  - Worst case ($T_{max}(n)$)
    kebutuhan waktu untuk kasus terburuk (kebutuhan waktu maksimum)
  - Average case ($T_{avg}(n)$)
    kebutuhan waktu untuk kasus rata-rata (kebutuhan waktu rata-rata)

Sequential Search (1)

```
procedure PencarianBeruntun (input a1,...,an : integer, x : integer,
                             output idx : integer)

{Mencari x di dalam elemen a1,a2,...,an. Elemen terbesar akan
 disimpan di dalam maks. Lokasi (indeks elemen) tempat x ditemukan
 diisi ke dalam idx. Jika idx tidak ditemukan, idx diisi dengan 0.
 Masukan: a1,a2,...,an
 Keluaran: idx}

Deklarasi
   k : integer

Algoritma
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                          { k = n or ak = x }
   if ak = x then                    { x ditemukan }
      idx←k
   else                              { x tidak ditemukan }
      idx←0
   endif
```

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                             { k = n or ak = x }
   if ak = x then                       { x ditemukan }
      idx←k
   else                                 { x tidak ditemukan }
      idx←0
   endif
```

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                        { k = n or ak = x }
   if ak = x then                  { x ditemukan }
      idx←k
   else                            { x tidak ditemukan }
      idx←0
   endif
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                              { k = n or ak = x }
   if ak = x then                        { x ditemukan }
      idx←k
   else                                  { x tidak ditemukan }
      idx←0
   endif
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   - Cek bagian terpenting/yang paling banyak memakan waktu

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                        { k = n or ak = x }
   if ak = x then                  { x ditemukan }
      idx←k
   else                            { x tidak ditemukan }
      idx←0
   endif
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   - Cek bagian terpenting/yang paling banyak memakan waktu
   - Tentukan operasi dasar dari bagian tersebut

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                          { k = n or ak = x }
   if ak = x then                    { x ditemukan }
      idx←k
   else                              { x tidak ditemukan }
      idx←0
   endif
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?

   operasi perbandingan $k < n$ and $a_k \neq x$)

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                            { k = n or ak = x }
   if ak = x then                      { x ditemukan }
      idx←k
   else                                { x tidak ditemukan }
      idx←0
   endif
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan $k < n$ and $a_k \neq x$)
2. Berapa kali operasi dasar tersebut dioperasikan?

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                        { k = n or ak = x }
   if ak = x then                  { x ditemukan }
      idx←k
   else                            { x tidak ditemukan }
      idx←0
   endif
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan $k < n$ and $a_k \neq x$)
2. Berapa kali operasi dasar tersebut dioperasikan?
   - Cek variabel penentu iterasi

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                         { k = n or ak = x }
   if ak = x then                   { x ditemukan }
      idx←k
   else                             { x tidak ditemukan }
      idx←0
   endif
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan $k < n$ and $a_k \neq x$)
2. Berapa kali operasi dasar tersebut dioperasikan?
   - Cek variabel penentu iterasi
   - Cek berapa kali iterasi dapat dilakukan

Sequential Search: kompleksitas waktu

```
procedure PencarianBeruntun (a1,...,an : integer, x : integer)
  k ← 1
   while k < n and (ak ≠ x) do
      k← k + 1
   endwhile                        { k = n or ak = x }
   if ak = x then                  { x ditemukan }
      idx←k
   else                            { x tidak ditemukan }
      idx←0
   endif
```

Untuk mencari kompleksitas waktu:

1. Apa operasi dasar dari algoritma tersebut?
   operasi perbandingan $k < n$ and $a_k \neq x$)
2. Berapa kali operasi dasar tersebut dioperasikan?
   - Cek variabel penentu iterasi
   - Cek berapa kali iterasi dapat dilakukan
     Perhatikan bahwa banyaknya iterasi akan bergantung dari urutan masukan karena $a_k \neq x$

Sequential Search: kompleksitas waktu

```
k ← 1
while k < n and (a_k ≠ x) do
```

Sequential Search: kompleksitas waktu

```
k ← 1
while k < n and (a_k ≠ x) do
```

$\longrightarrow k = 1, 2, \ldots n - 1 \Rightarrow n - 1$ kali

Sequential Search: kompleksitas waktu

```
k ← 1
while k < n and (a_k ≠ x) do
```

$\longrightarrow k = 1, 2, \ldots n-1 \Rightarrow n-1$ kali

ATAU

i kali (ketika ditemukan $\mathtt{a_i} = \mathtt{x}$)

Sequential Search: kompleksitas waktu

$\mathtt{k} \leftarrow \mathtt{1}$

<u>while</u> $\mathtt{k} < \mathtt{n}$ <u>and</u> $(\mathtt{a_k} \neq \mathtt{x})$ <u>do</u>

$\longrightarrow k = 1, 2, \ldots n - 1 \Rightarrow n - 1$ kali

ATAU

i kali (ketika ditemukan $\mathtt{a_i} = \mathtt{x}$)

Best case: berapakah nilai i paling minimum?

Sequential Search: kompleksitas waktu

```
k ← 1
while k < n and (ak ≠ x) do
```

$\longrightarrow k = 1, 2, \ldots n - 1 \Rightarrow n - 1$ kali

ATAU

i kali (ketika ditemukan $\mathtt{a_i} = \mathtt{x}$)

Best case: $a_1 = x$ (1 kali perbandingan) $\Rightarrow T_{min}(n) = 1$

$$T_{avg}(n) = \frac{1 + 2 + \ldots + n}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}$$

Sequential Search: kompleksitas waktu

`k ← 1`

`while k < n and (a_k ≠ x) do`

$\longrightarrow k = 1, 2, \ldots n-1 \Rightarrow n-1$ kali

ATAU

i kali (ketika ditemukan $\mathtt{a_i} = \mathtt{x}$)

Best case: $a_1 = x$ (1 kali perbandingan) $\Rightarrow T_{min}(n) = 1$

Worst case: berapakah nilai i paling maksimum/tidak ada yang memenuhi?

$$T_{avg}(n) = \frac{1 + 2 + \ldots + n}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}$$

Sequential Search: kompleksitas waktu

```
k ← 1
while k < n and (ak ≠ x) do
```

$\longrightarrow k = 1, 2, \ldots n-1 \Rightarrow n-1$ kali

ATAU

i kali (ketika ditemukan $\mathtt{a_i} = \mathtt{x}$)

Best case: $a_1 = x$ (1 kali perbandingan) $\Rightarrow T_{min}(n) = 1$

Worst case: $a_n = x$ atau x tidak ditemukan $\Rightarrow T_{max}(n) = n$

$$T_{avg}(n) = \frac{1 + 2 + \ldots + n}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}$$

Sequential Search: kompleksitas waktu

```
k ← 1
while k < n and (a_k ≠ x) do
```

$\longrightarrow k = 1, 2, \ldots n-1 \Rightarrow n-1$ kali

ATAU

i kali (ketika ditemukan $\mathtt{a_i} = \mathtt{x}$)

Best case: $a_1 = x$ (1 kali perbandingan) $\Rightarrow T_{min}(n) = 1$

Worst case: $a_n = x$ atau x tidak ditemukan $\Rightarrow T_{max}(n) = n$

Average case: rata-rata $T(n)$ di berbagai kasus

$$T_{avg}(n) = \frac{1 + 2 + \ldots + n}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}$$

Sequential Search: kompleksitas waktu

```
k ← 1
while k < n and (ak ≠ x) do
```

$\longrightarrow k = 1, 2, \ldots n-1 \Rightarrow n-1$ kali

ATAU

i kali (ketika ditemukan $\mathtt{a_i} = \mathtt{x}$)

Best case: $a_1 = x$ (1 kali perbandingan) $\Rightarrow T_{min}(n) = 1$

Worst case: $a_n = x$ atau x tidak ditemukan $\Rightarrow T_{max}(n) = n$

Average case:

$a_1 = x \Rightarrow T(n) = 1$

$a_2 = x \Rightarrow T(n) = 2$

$\vdots$

$a_n = x$ atau x tidak ditemukan $\Rightarrow T(n) = n$

$$T_{avg}(n) = \frac{1 + 2 + \ldots + n}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}$$

Referensi

- R. Neapolitan, K. Naimipour. Foundations of Algorithms –5th Edition, Jones and Bartlett Learning, 2014.
- I. Parberry. Lecture Notes on Algorithm.

THANK YOU