Kompleksitas Algoritma dan Pemilihan Struktur Data yang Efisien

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Ikhtisar

Kompleksitas algoritma dan *asymptotic notation*

- Kompleksitas waktu dan memori

Perbandingan struktur data

- Array *vs.* list *vs.* tree *vs.* hash table

Pemilihan struktur data

Kenapa struktur data itu penting?

Struktur data dan algoritma adalah fondasi dari pemrograman komputer

Algorithmic thinking, problem solving dan struktur data sangatlah vital bagi para *software engineer*.

Analisis kompleksitas komputasi sangat penting untuk perancangan algoritma dan pemrograman yang efisien

Analisis Algoritma

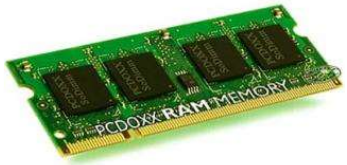
Kenapa kita sebaiknya menganalisis algoritma?

- Memprediksi "*resource*" yang dibutuhkan oleh algoritma terkait:
  - Waktu komputasi (konsumsi CPU)
  - Memori yang dibutuhkan (konsumsi RAM)
  - Konsumsi *bandwidth* komunikasi
- Ukuran waktu yang dibutuhkan dari suatu algoritma:
  - Jumlah primitif operasi yang dieksekusi (*machine independent steps*)
  - Juga diketahui dengan **kompleksitas algoritma** (*algorithm complexity*)**.**

Kompleksitas Algoritma

Apa yang diukur?

- Memori
- Waktu
- Jumlah step
- Jumlah operasi tertentu, seperti:
  1) banyaknya operasi yang melibatkan *disk*,
  2) banyaknya paket jaringan yang dibutuhkan
- *Asymptotic complexity*




Kompleksitas Waktu

Worst-case

Batas atas (*upper-bound*) dari waktu yang dibutuhkan untuk suatu ukuran input data

Average-case

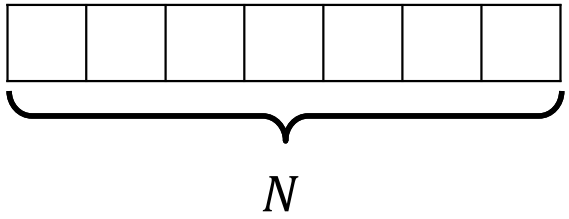
Waktu yang dibutuhkan dengan mengasumsikan kasus dapat terpecahkan secara "rerata" (tengah-tengan antara *worst-case* dan *best-case*)

Best-case

Batas bawah (*lower-bound*) dari waktu yang dibutuhkan untuk suatu ukuran input data

Contoh Kompleksitas Waktu

Pencarian sekuensial pada list dengan ukuran N



Worst-case
- Membandingkan sebanyak N kali

Average-case
- Membandingkan sebanyak $N/2$ kali

Best-case
- Membandingkan sebanyak 1 kali

Catatan: Kasus tersebut adalah kasus yang jumlah operasinya linear terhadap ukuran datanya

Kompleksitas Algoritma

Kompleksitas algoritma adalah estimasi mengenai jumlah langkah yang diperlukan oleh suatu algoritma terhadap variabel ukuran input data.

Jika $f(n)$ adalah fungsi jumlah langkah komputasi yang dibutuhkan terhadap data yang berukuran n, karakteristik $f(n)$ beberapa di antaranya dapat berupa:

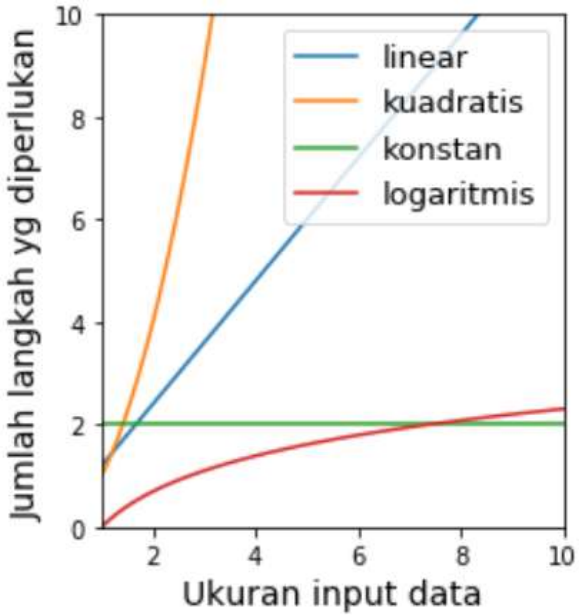
$f(n) = c_1 \rightarrow$ konstan

$f(n) = c_1 log(n) \rightarrow$ logaritmis

$f(n) = c_1 n + c_2 \rightarrow$ linear

$f(n) = c_1 n^2 + nc_2 + c_3 \rightarrow$ kuadratis

Di mana c_i adalah suatu konstanta yang menyatakan "*hardware **dependent** measuremet*".



Asymptotic Notation

Asymptotic notation adalah notasi yang digunakan untuk menyatakan kompleksitas algoritma. Notasi "Big O" adalah salah satu yang sering digunakan, yakni untuk ukuran "*worst-case*".

Diberikan $f(n)$, notasi Big O dapat diperoleh dengan: (i) ambil suku paling besar, (ii) hilangkan koefisien dari suku tersebut.

Karakteristik	**Jumlah langkah ($f(n)$)**	**(i) Suku paling besar**	**(ii) Hilangkan koefisiennya**	**Notasi Big O**
Konstan	c_1	c_1	1	$O(1)$
Logaritmis	$c_1 log(n)$	$c_1 log(n)$	$log(n)$	$O(\log(n))$
Linear	$c_1 n + c_2$	$c_1 n$	n	$O(n)$
Kuadratis	$c_1 n^2 + n c_2 + c_3$	$c_1 n^2$	n^2	$O(n^2)$

Notasi Big O tidak mengandung konstanta c_i, sehingga notasi ini adalah "*hardware* ***independent*** *measurement*".

Notasi Big O hanya mengambil suku terbesar karena definisi dari ukuran input datanya adalah $n \to \infty$

Macam-macam Kompleksitas

Kompleksitas	Notasi Big-O	Deskripsi
Konstan	O(1)	Jumlah operasinya konstan, tidak tergantung ukuran input data. Contoh n = 1.000.000 $\longrightarrow$ 1–2 operasi
Logaritmis	O(log n)	Jumlah operasinya proporsional dengan $\log_2(n)$, di mana n adalah ukuran input data. Contoh n = 1.000.000.000 $\longrightarrow$ 30 operasi
Linear	O(n)	Jumlah operasinya proporsional dengan ukuran input data. Contoh n = 10.000 $\longrightarrow$ 5.000 operasi

Macam-macam Kompleksitas

Kompleksitas	Notasi Big-O	Deskripsi
Kuadratis	$O(n^2)$	Jumlah operasinya proporsional dengan kuadrat dari ukuran input data. Contoh n = 500 $\rightarrow$ 250 000 operasi
Cubic	$O(n^3)$	Jumlah operasinya proporsional dengan pangkat tiga dari ukuran input data. Contoh n = 200 $\rightarrow$ 8.000.000 operasi
Eksponensial	$O(2^n)$, $O(k^n)$, $O(n!)$	Proporsional secara eksponensial. Contoh n = 20 $\rightarrow$ 1.048.576 operasi

AU6

Slide 11

AU6 Apa term bahasa Indonesia yg pas untuk ini?
Ardian Umam; 16/11/2019

Ilustrasi Kompleksitas Waktu dan Kecepatan

Kompleksitas	n=10	n=20	n=50	n=100	n=1000	n=10000	n=100000
O(1)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(log n)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(n)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(n*log n)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
O(n^2)	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	2 s	3–4 min
O(n^3)	< 1 s	< 1 s	< 1 s	< 1 s	20 s	5 hours	231 days
O(2^n)	< 1 s	< 1 s	260 days	hangs	hangs	hangs	hangs
O(n!)	< 1 s	hangs	hangs	hangs	hangs	hangs	hangs
O(n^n)	3–4 min	hangs	hangs	hangs	hangs	hangs	hangs

Kompleksitas Waktu dan Memori

- Kompleksitas juga dapat diekspresikan dalam formula multi variabel, contohnya:
  - Pengisian matriks berukuran $m \times n$, kompleksitasnya dapat dituliskan dengan $O(m \times n)$
  - Proses traversal DFS (*Depth First Search*) dalam suatu graf dengan n simpul dan m busur, kompeksitasnya dapat dituliskan dengan $O(m+n)$

- Konsumsi memori juga harus dipertimbangkan, contohnya:
  - Misalkan kompleksitas waktunya hanya $O(n)$ tetapi ternyata kompleksitas memorinya adalah $O(n^2)$.
  - Ukuran input $n = 50.000$ misalnya $\rightarrow$ OutOfMemoryException

Menganalisa Kompleksitas Algoritma

Contoh-contoh

Kompleksitas Konstan

```
void insertFirst (List *l, address p) {
    NEXT(p) = FIRST(*l); //O(1)
    FIRST(*l) = p; //O(1)
}
```

- **Kompleksitas**
  - $f(n) = O(1) + O(1)$
    $f(n) = c_1$ AU8 c_2
    $f(n) = c_3$
  - Suku paling besar = c_3
  - Notasi Big O $\rightarrow$ O(1)

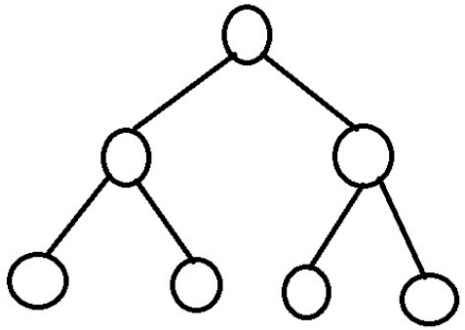
Slide 15

AU8 Penjumlahan kompleksitas konstan akan menghasilkan kompleksitas konstan juga

Ardian Umam; 17/11/2019

Kompleksitas Logaritmis

```
void insBTree(ElType x, BinTree *p) {
    if (isTreeEmpty(*p)) {
        CreateTree(x, NIL, NIL, p);
    }
    else {
        if (x>INFO(*p)) {
            insBTree(X,&RIGHT(*p));
        }
        else if (x<INFO(*p)) {
            insBTree(x,&LEFT(*p));
        }
    }
}
```



- **Kompleksitas** $\rightarrow O(\log n)$

Kompleksitas Linear

```
int findMaxElement(int[] array) {
    int i, max = array[0]; //O(1)
    for (i=0; i<array.length; i++) { // n kali
        if (array[i] > max) { //O(1)
            max = array[i]; //O(1)
        }
    }
    return max; //O(1)
}
```

AU3

- **Kompleksitas**
  - $f(n) = O(1) + n\big(O(1) + O(1)\big) + O(1)$
    $f(n) = c_1 + n(c_2 + c_3) + c_4$
    $f(n) = c_5 + c_6 n$
  - Suku paling besar = $c_6 n$ AU4
  - Notasi Big O $\rightarrow$ O(n)

Slide 17

AU3 Hanya copas dari materi Telerik (mulai dibandingkan dari indeks 0 lagi, bukan 1).
Ardian Umam; 16/11/2019

AU4 Penjumlahan kompleksitas konstan akan menghasilkan kompelsitas konstan juga
Ardian Umam; 16/11/2019

Kompleksitas Kuadratis (1)

```
void procedure1(int array[], int n) {
    //n = banyaknya elemen array
    int i, j;
    for (i=0; i<n; i++) {
       for (j = 0; j<n; j++) {
        //statement here
       }
    }
}
```

AU11

i	j	Jumlah langkah ($f(n)$)
0	0,1,2,…,n-1	n
1	0,1,2,…,n-1	n
2	0,1,2,…,n-1	n
⋮	⋮	⋮
n-1	0,1,2,…,n-1	n

- **Kompleksitas**
  - $f(n) = n + n + n + \cdots + n$

    $f(n) = n \times n = n^2$
  - Suku paling besar = n^2
  - Big O Notation $\rightarrow O(n^2)$

Slide 18

AU11 In C, array passed in a function/procedure will be treated as pointer. Thus, we cannot get arraySize by, e.g., (int)(sizeof(array) / sizeof(array[0]). One of alternatives is by passing the arraySize itself as parameter, in this example is "n".

Ardian Umam; 18/11/2019

Kompleksitas Kuadratis (2)

```
void bubbleSort(int arr[], int n) {
   //n = banyaknya elemen array
   int i, j;
   for (i = 0; i < n-1; i++){
       for (j = 0; j < n-i-1; j++) {
           if (arr[j] > arr[j+1]) {
              swap(&arr[j], &arr[j+1]);//O(1)
           }
       }
   }
}
```

```
void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
```

i	j	**Jumlah langkah ($f(n)$)**
0	0,1,2,..,n-2	n-1
1	0,1,2,...,n-3	n-2
⋮	⋮	⋮
n-3	0,1	2
n-2	0	1

- **Kompleksitas**
  - $f(n) = 1 + 2 + \cdots + (n-1)$

    $\mathrm{f}(n) = \frac{(\mathrm{n}-1)(1+(\mathrm{n}-1)}{2}$

    $\mathrm{f}(n) = \frac{n^2-n}{2} = \frac{1}{2}n^2 - \frac{1}{2}\mathrm{n}$
  - Suku paling besar = $\frac{1}{2}n^2$
  - Big O Notation $\rightarrow O(n^2)$

Slide 19

AU9 In C, array passed in a function/procedure will be treated as pointer. Thus, we cannot get arraySize by, e.g., (int)(sizeof(array) / sizeof(array[0]). One of alternatives is by passing the arraySize itself as parameter, in this example is "n".
Ardian Umam; 18/11/2019

AU10 Didapatkan dari penjumlahan deret aritmatika
Ardian Umam; 18/11/2019

AU14 Menggunakan "for" ini sebenarnya "worst case", lbh pas jika menggunakan while (bisa berhenti ketika dalam satu iterasi tidak ada lagi yang diswap).
Ardian Umam; 19/11/2019

Kompleksitas Multi-variabel: m×n (1)

```
long SumMN(int n, int m) {
    int x, y;
    long sum = 0;
    for (x=0; x<n; x++) {
        for (int y=0; y<m; y++)
            sum += x*y;
        }
    }
    return sum;
}
```

- Kompleksitas
  - $f(m,n) = \sim m \times n$
  - Suku paling besar = $m \times n$
  - Notasi Big O $\longrightarrow$ O($m \times n$)

Kompleksitas Multi-variabel: m×n (2)

```
long SumMN(int n, int m)
{
    long sum = 0; int x,y;
    for (x=0; x<n; x++) {
        for (int y=0; y<m; y++) {
            if (x==y) {
                for (int i=0; i<n; i++) {
                    sum += i*x*y;
                }
            }
        }
    }
    return sum;
}
```

- Kompleksitas
  - $f(m,n) = \sim m \times n + min(m,n) \times n$
  - Suku paling besar = $m \times n$
  - Notasi Big O $\longrightarrow$ O($m \times n$)

Kompleksitas Eksponensial

```
int Fibbonaci(int n) {
    if (n <= 1) {
        return n;
    }
    else {
        return Fibbonaci (n-1) + Fibbonaci (n-2);
    }
}
```

- **Kompleksitas**
  - Kita anggap $\boldsymbol{F(n-2) \approx F(n-1)}$ → asumsi *upper bound* AU12
  - $F(n) = F(n-1) + F(n-2)$
    $F(n) = 2F(n-1)$
    $F(n) = 4F(n-2)$
    $F(n) = 8F(n-3)$
    $F(n) = 2^k F(n-k)$, fungsi berhenti saat $n - k = 0 \ \rightarrow k = n$
    $F(n) = 2^n F(0)$
  - Notasi Big O $\rightarrow O(2^n)$

Slide 22

AU12 Karena F(n-2) sebenarnya lebih rendah kompleksitasnya dibandingkan F(n-1), sehingga asumsi ini adalah asumsi upper bound.

Ardian Umam; 18/11/2019

Membandingkan Struktur Data

Contoh-contoh

Perbandingan Kompleksitas Struktur Data

Struktur data	Add	Find	Delete	Get-by-index
Array	O(n)	O(n)	O(n)	O(1)
Linked list	O(1)	O(n)	O(n)	O(n)
Stack	O(1)	–	O(1)	–
Queue	O(1)	–	O(1)	–
Hash table	O(1)	O(1)	O(1)	–
Balanced Search Tree	O(log n)	O(log n)	O(log n)	O(log n)

Pemilihan struktur Data

- Array

  Digunakan ketika yang dibutuhkan adalah akses elemen berdasarkan indeks

- Linked lists (dicatat first dan last)

  Digunakan ketika yang dibutuhkan adalah akses elemen ujung-ujung dari list saja

- Hash table-based dictionary

  Digunakan ketika dibutuhkan: (i) penambahan pasangan key-value secara cepat, dan (ii) pencarian berdasarkan key secara cepat

  Elemen di dalam hash table tidak memiliki urutan tertentu

Pemilihan struktur Data

- Balanced search tree-based dictionary

  Digunakan ketika dibutuhkan: (i) penambahan pasangan key-value secara cepat, (ii) pencarian berdasarkan key secara cepat dan (iii) enumerasi yang tersortir berdasarkan key

- Hash table-based set

  Digunakan untuk menjaga value-nya memiliki nilai unik

  Elemennya tidak memiliki urutan tertentu

Ringkasan

Kompleksitas algoritma adalah estimasi mengenai jumlah langkah yang diperlukan oleh suatu algoritma terhadap variabel ukuran input data.

- Kompleksitasnya dapat berupa logaritmis, linear, log n, kuadrat, kubik, eksponensial, dll.
- Dapat digunakan untuk mengestimasi kecepatan dari suatu code sebelum dieksekusi.

Struktur data yang berbeda memiliki efisiensi yang berbeda pada operasi-operasi yang berbeda.

- Operasi add, find, delete paling cepat adalah struktur data hash table, yakni O(1) untuk semua operasi.

Slide 27

AU13 Masih blm menggunakan istilah Indonesia untuk cubic.
Ardian Umam; 18/11/2019

2SAR1 sudah yee
23516003 Satrio Adi Rukmono; 25/11/2019

Ada diskusi / pertanyaan?

Latihan

1. A text file **students.txt** holds information about students and their courses in the following format:

```
Kiril   | Ivanov    | C#
Stefka  | Nikolova  | SQL
Stela   | Mineva    | Java
Milena  | Petrova   | C#
Ivan    | Grigorov  | C#
Ivan    | Kolev     | SQL
```

Using **SortedDictionary<K,T>** print the courses in alphabetical order and for each of them prints the students ordered by family and then by name:

```
C#: Ivan Grigorov, Kiril Ivanov, Milena Petrova
Java: Stela Mineva
SQL: Ivan Kolev, Stefka Nikolova
```

Slide 29

AU13 Masih blm menggunakan istilah Indonesia untuk cubic.
Ardian Umam; 18/11/2019

Latihan

2. A large trade company has millions of articles, each described by barcode, vendor, title and price. Implement a data structure to store them that allows fast retrieval of all articles in given price range [x…y]. ***Hint***: use **OrderedMultiDictionary<K,T>** from Wintellect's Power Collections for .NET.

3. Implement a data structure **PriorityQueue<T>** that provides a fast way to execute the following operations: add element; extract the smallest element.

4. Implement a class **BiDictionary<K1,K2,T>** that allows adding triples {key1, key2, value} and fast search by key1, key2 or by both key1 and key2. **Note**: multiple values can be stored for given key.

Latihan

5. A text file **phones.txt** holds information about people, their town and phone number:

```
Mimi Shmatkata           | Plovdiv   | 0888 12 34 56
Kireto                   | Varna     | 052 23 45 67
Daniela Ivanova Petrova  | Karnobat  | 0899 999 888
Bat Gancho               | Sofia     | 02 946 946 946
```

Duplicates can occur in people names, towns and phone numbers. Write a program to execute a sequence of commands from a file **commands.txt**:

- **find(name)** – display all matching records by given name (first, middle, last or nickname)
- **find(name, town)** – display all matching records by given name and town