

# Analisa dan Perancangan Algoritma

Ahmad Sabri, Dr  
Sesi 1: 9 Mei 2016



# Apakah algoritma itu?

- Asal istilah: Al Khwarizmi ( $\pm$  800 M), matematikawan dan astronomer Persia.




- Pengertian umum: "suatu urutan langkah-langkah atau instruksi untuk menyelesaikan suatu masalah".

# Kriteria algoritma yang "benar"



Urutan langkah/instruksi dapat dikatakan algoritma yang "benar" jika memenuhi kriteria berikut:

1. Untuk setiap input yang valid menghasilkan output
2. Efektif (tepat guna/menuju pada solusi yang dicari)
3. Jumlah langkah berhingga
4. Langkah-langkah berakhir pada kondisi tertentu (diperoleh/tidaknya solusi)



Masalah: memeriksa apakah integer  $n$  memiliki akar integer

Algoritma:

1. Input  $n$
2. Tetapkan  $i = 0$
3. Hitung  $h = i \times i$
4. Jika  $h = n$  maka tampilkan  $i$  dan berhenti
5.  $i = i + 1$
6. Kembali ke langkah 3

Apakah algoritma di atas adalah algoritma yang benar?

# Algoritma "terbaik"

- Sebuah permasalahan dapat diselesaikan oleh beberapa alternatif algoritma.
- Dari sekian banyak alternatif, bagaimanakah membandingkan kinerja beragam algoritma tersebut, sehingga dapat ditentukan manakah algoritma yang terbaik di antara yang ada?



# Algoritma terbaik

- Suatu algoritma harus *menghasilkan output* yang tepat guna (*efektif*) dalam waktu yang relatif singkat dan penggunaan sumber daya yang relatif sedikit (*efisien*) dengan *langkah yang berhingga* banyaknya dan *prosesnya berakhir* dalam kondisi memperoleh/tidak memperoleh solusi.



# Bagaimana mengetahui algoritma terbaik?



- Kinerja suatu algoritma diukur dengan indikator yang disebut "**kompleksitas komputasi**" dari algoritma tersebut (singkatnya disebut "**kompleksitas**")
- Untuk masalah yang sama, algoritma terbaik adalah algoritma yang terbukti memiliki kompleksitas terbaik. Dalam hal ini, algoritma tersebut dikatakan algoritma yang **paling efisien**.

# Analisis algoritma

- Usaha/proses untuk menemukan kompleksitas suatu algoritma dan/atau membandingkan sebuah algoritma dengan algoritma yang lain berdasarkan kompleksitasnya, disebut sebagai "analisis algoritma".
- Hasil dari analisis dari suatu algoritma dapat digunakan untuk memperbaiki algoritma tersebut sehingga memiliki kinerja yang lebih baik.





# Mengapa perlu analisis algoritma?



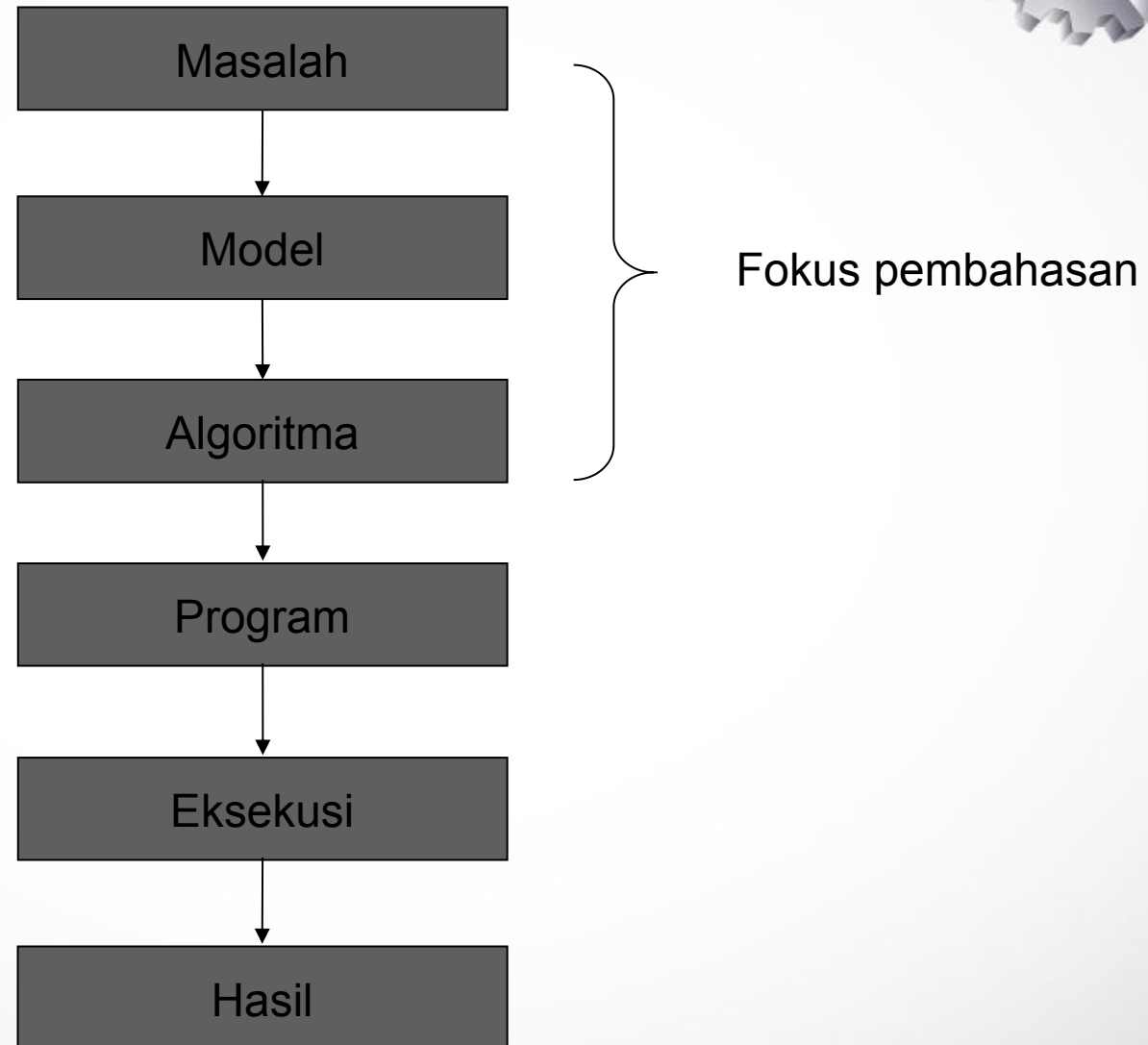
- Sebagai alat perbandingan algoritma, sehingga dapat diperoleh algoritma terbaik
- Hasil dari analisis dari suatu algoritma dapat digunakan untuk memperbaiki algoritma tersebut sehingga memiliki kinerja yang lebih baik.

# Namun...

- Terdapat permasalahan yang algoritma paling efisiennya belum diketahui/ditemukan
- Walaupun demikian, belum terbukti ada/tidaknya algoritma paling efisien untuk permasalahan demikian



# Skema penyelesaian masalah





- Rancanglah algoritma untuk menjumlahkan bilangan dari 1 sampai  $n$
- Input: integer  $n$ , Output: jumlah bilangan dari 1 sampai  $n$

# Studi tentang algoritma



1. Merencanakan algoritma  
membuat model penyelesaian masalah
2. Merepresentasikan algoritma  
langkah-langkah penyelesaian model dengan  
menggunakan **diagram alur** ataupun **pseudo code**
3. Menentukan validitas algoritma  
outputnya adalah solusi masalah
4. Menganalisa algoritma  
menentukan tingkat efisiensi algoritma dari sisi waktu  
maupun penggunaan memori
5. Menguji program dari algoritma

# Analisis Algoritma

Kompleksitas algoritma mencakup 2 hal:

1. Waktu yang digunakan
2. Memori yang digunakan



# Waktu yang digunakan

Bergantung pada:

- 1.banyaknya langkah
- 2.besar dan jenis data yang digunakan  
(termasuk input data)
- 3.jenis operasi
- 4.komputer dan kompiler yang digunakan



# Memori yang digunakan

Banyaknya langkah yang digunakan dan jenis variabel/data yang dipakai dalam algoritma berbanding lurus dengan penggunaan memori.





# Kompleksitas waktu



Kompleksitas waktu adalah sebuah fungsi  $f(n)$  yang mewakili waktu tempuh dan kebutuhan memori dengan ukuran  $n$  input data

# Notasi Big-O



$f(n)$  merupakan big O dari  $g(n)$ , dinotasikan sebagai  $O(g(n))$  jika dan hanya jika terdapat dua konstanta bulat positif  $c$  dan  $n_0$  sedemikian sehingga

$$0 \leq f(n) \leq cg(n) \text{ untuk setiap } n \geq n_0$$



## Contoh

- $2n + 10$  adalah  $O(n)$ 
  - $2n + 10 \leq cn$
  - $(c - 2)n \geq 10$
  - $n \geq 10/(c - 2)$
  - Ambil  $c = 3$  and  $n_0 = 10$
- $n^2$  bukan  $O(n)$
- $7n-2$  adalah  $O(n)$
- $3n^3 + 20n^2 + 5$  adalah  $O(n^3)$
- $3 \log n + \log \log n$  adalah  $O(\log n)$

# Aturan Big-O



- Jika  $f(n)$  adalah polinomial berderajat  $d$ , maka  $f(n)$  adalah  $O(n^d)$ , yaitu
  1. Abaikan suku dengan derajat kurang dari  $d$
  2. Abaikan koefisien

# Notasi lain untuk kompleksitas



- **big-Omega**
  - $f(n)$  adalah  $\Omega(g(n))$  jika terdapat konstanta  $c > 0$  dan sebuah integer  $n_0 \geq 1$  sedemikian sehingga  $f(n) \geq c \cdot g(n)$  untuk  $n \geq n_0$
- **big-Theta**
  - $f(n)$  adalah  $\Theta(g(n))$  terdapat konstanta  $c' > 0$  dan  $c'' > 0$  dan sebuah integer  $n_0 \geq 1$  sedemikian sehingga  $c' \cdot g(n) \leq f(n) \leq c'' \cdot g(n)$  untuk  $n \geq n_0$
- **little-o**
  - $f(n)$  adalah  $o(g(n))$  jika, untuk sebarang konstanta  $c > 0$ , terdapat integer  $n_0 \geq 0$  sedemikian sehingga  $f(n) \leq c \cdot g(n)$  untuk  $n \geq n_0$
- **little-omega**
  - $f(n)$  adalah  $\omega(g(n))$  jika, untuk sebarang konstanta  $c > 0$ , terdapat integer  $n_0 \geq 0$  sedemikian sehingga  $f(n) \geq c \cdot g(n)$  untuk  $n \geq n_0$

# Contoh kasus: Insertion sort

- Step 1 – If it is the first element, it is already sorted. return 1;
- Step 2 – Pick next element
- Step 3 – Compare with all elements in the sorted sub-list
- Step 4 – Shift all the elements in the sorted sub-list that is greater than the value to be sorted
- Step 5 – Insert the value
- Step 6 – Repeat until list is sorted





```
for i ← 1 to length(A)
  j ← i
  while j > 0 and A[j-1] > A[j]
    swap A[j] and A[j-1]
    j ← j - 1
  end while
end for
```



- Urutkanlah barisan berikut dengan menggunakan algoritma insertion sort
- 3 7 4 8 5 2 6 1
- 6 5 3 1 8 7 2 4
- 1 2 3 4 5 6 7 8
- 8 7 6 5 4 3 2 1



# Insertion Sort: Number of Comparisons



# of Sorted Elements	Best case	Worst case
0	0	0
1	1	1
2	1	2
...	...	...
n-1	1	n-1
	<hr/>	<hr/>
	<b>n-1</b>	<b><math>n(n-1)/2</math></b>
	<hr/>	<hr/>

Remark: we only count comparisons of elements in the array.

# Kompleksitas Insertion sort

- Best case (array sudah terurut):  $O(n)$
- Worst case (array terurut secara terbalik):  $O(n^2)$
- Average case:  $O(n^2)$



# Bubble Sort



- Simplest sorting algorithm
- Idea:
  - 1. Set flag = false
  - 2. Traverse the array and compare pairs of two consecutive elements
    - 1.1 If  $E1 \leq E2$   $\rightarrow$  OK (do nothing)
    - 1.2 If  $E1 > E2$  then Swap( $E1, E2$ ) and set flag = true
  - 3. repeat 1. and 2. while flag=true.



```
procedure bubbleSort( A : list of sortable items )  
  n = length(A)  
  repeat  
    swapped = false  
    for i = 1 to n-1 inclusive do  
      if A[i-1] > A[i] then  
        swap(A[i-1], A[i])  
        swapped = true  
      end if  
    end for  
    n = n - 1  
  until not swapped  
end procedure
```



- Urutkanlah barisan berikut dengan menggunakan algoritma bubble sort
- 3 7 4 8 5 2 6 1
- 6 5 3 1 8 7 2 4
- 1 2 3 4 5 6 7 8
- 8 7 6 5 4 3 2 1

# Kompleksitas Bubble sort

- Best case (array sudah terurut):  $O(n)$
- Worst case (array terurut secara terbalik):  $O(n^2)$
- Average case:  $O(n^2)$



# Membangkitkan permutasi



- Rancanglah algoritma untuk membentuk seluruh permutasi yang mungkin dari himpunan  $\{1, 2, \dots, n\}$ , dimulai dari permutasi  $12\dots n$
- Setiap permutasi dihasilkan dengan jumlah komputasi yang sama (kompleksitas  $O(1)$ )
- Kompleksitas  $O(N)$ , di mana  $N$  adalah banyaknya permutasi dengan panjang  $n$

# Steinhaus Johnson Trotter algorithm



1. Generate initial permutation  $12 \dots n$ , and set all symbols movable, and to have direction left.
2. Swap the largest movable symbol  $L$  with the adjacent symbol according to the current direction of  $L$ .
3. Reverse the direction of all symbol larger than  $L$ .
4. Back to step 2 until there is no movable symbol.