

CSGE602040 • Struktur Data & Algoritma: Analisis Algoritma

Fakultas Ilmu Komputer • Universitas Indonesia

Slide acknowledgments:

Suryana Setiawan, Ade Azurat, Denny, Ruli Manurung



■ al•go•rithm

■ *n.*

1 *Math.* *a)* any systematic method of solving a certain kind of problem *b)* the repetitive calculations used in finding the greatest common divisor of two numbers (called in full Euclidean algorithm)

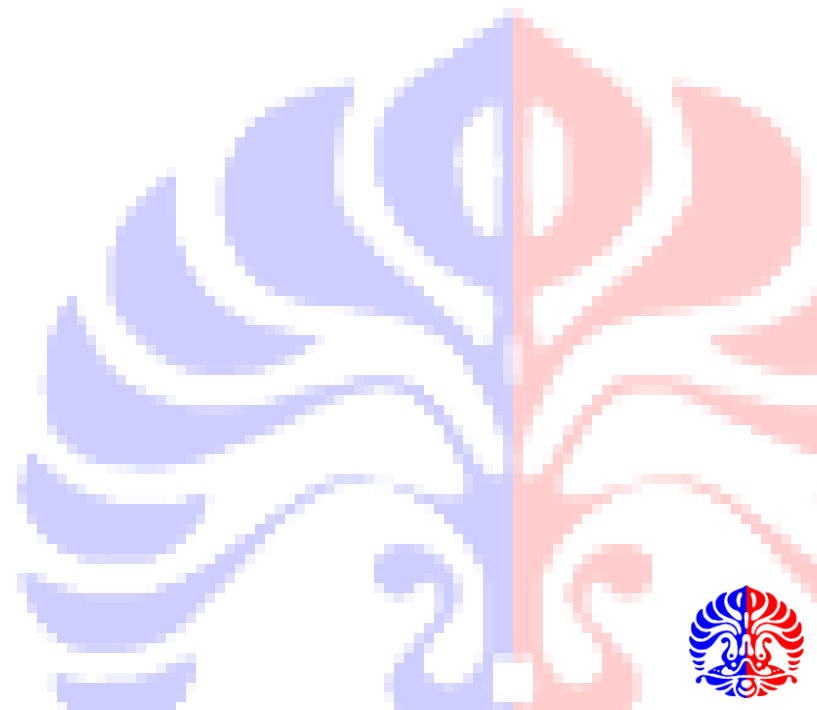
2 *Comput.* a predetermined set of instructions for solving a specific problem in a limited number of steps

- Suatu set instruksi yang harus diikuti oleh komputer untuk memecahkan suatu masalah.
- Program harus berhenti dalam batas waktu yang wajar (*reasonable*)
- Tidak terikat pada *programming language* atau bahkan paradigma pemrograman (mis. *Procedural* vs *Object-Oriented*)



Analisa Algoritma: Motivasi

- perlu diketahui berapa banyak *resource* (*time* & *space*) yang diperlukan oleh sebuah algoritma
- Menggunakan teknik-teknik untuk mengurangi waktu yang dibutuhkan oleh sebuah algoritma



Tujuan Pengajaran

- Mahasiswa dapat memperkirakan waktu yang dibutuhkan sebuah algoritma.
- Mahasiswa memahami kerangka matematik yang menggambarkan *running time*.
- Mahasiswa mengenali beberapa teknik sederhana yang dapat memperbaiki *running time*.
- Mahasiswa memahami bahwa suatu permasalahan dapat diselesaikan dengan beberapa algoritma dengan *running time* yang berbeda-beda.



Outline

- Apa itu analisa algoritma?- *what*
- Bagaimana cara untuk analisa/mengukur? - *how*
- Notasi Big-Oh
- Studi kasus: *The Maximum Contiguous Subsequence Sum Problem* (Nilai jumlah berurutan terbesar)
 - Algoritma 1:A cubic algorithm
 - Algoritma 2:A quadratic algorithm
 - Algoritma 3:A linear algorithm



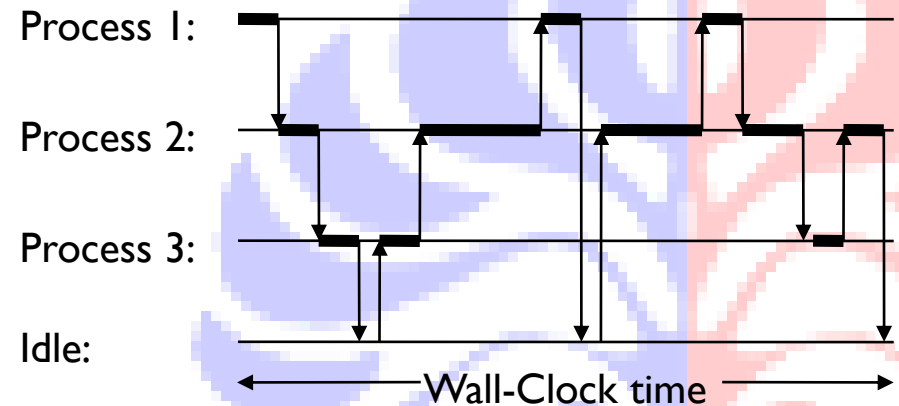
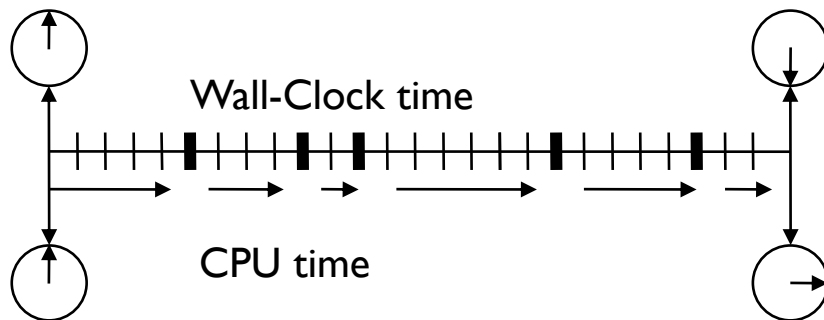
Analisa Algoritma: *What?*

- Mengukur jumlah sumber daya (*time* dan *space*) yang diperlukan oleh sebuah algoritma
- Waktu yang diperlukan (*running time*) oleh sebuah algoritma cenderung *tergantung* pada *jumlah input* yang diproses.
 - *Running time* dari sebuah algoritma adalah *fungsi* dari *jumlah inputnya*
- Ada algoritma yang running time-nya tergantung pada *jumlah output*-nya (*output-sensitive*)
- Selalu *tidak terikat* pada *platform* (mesin + OS), bahasa pemrograman, kualitas kompilator atau bahkan *paradigma pemrograman* (mis. *Procedural* vs *Object-Oriented*)



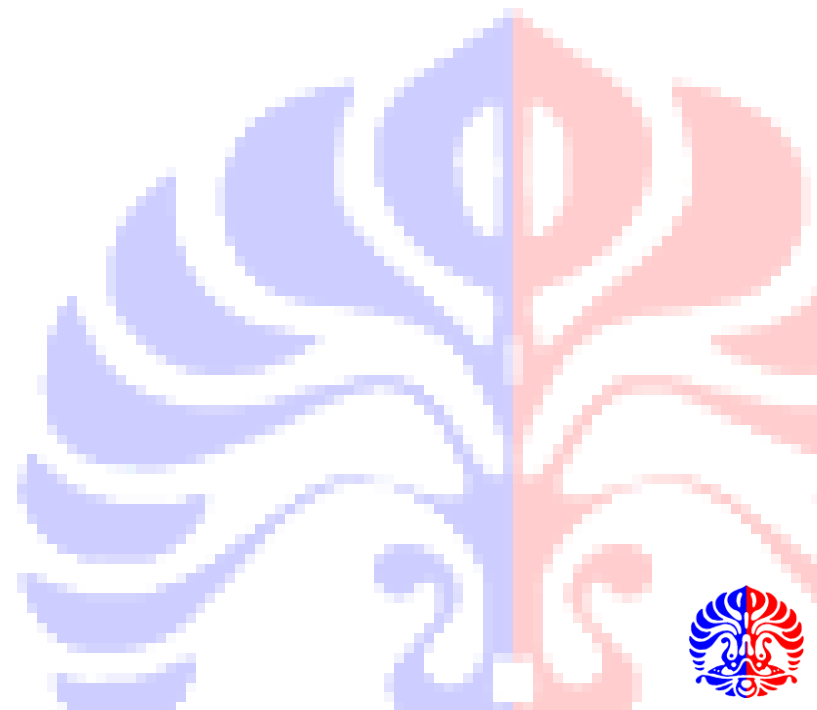
Analisa Algoritma: *How?*

- Bagaimana jika kita menggunakan jam?
 - Jumlah waktu yang digunakan **bervariasi** tergantung pada beberapa faktor lain: kecepatan mesin, sistem operasi (multi-tasking), kualitas kompiler, dan bahasa pemrograman.
 - Sehingga kurang memberikan gambaran yang tepat tentang algoritma



Analisa Algoritma: *How?*

- Notasi O (sering disebut sebagai “notasi big-Oh”)
 - Digunakan sebagai bahasa untuk membahas efisiensi dari sebuah algoritma: $\log n$, linier, $n \log n$, n^2 , n^3 , ...
 - Dari hasil *run-time*, dapat kita buat grafik dari waktu eksekusi dan jumlah data.



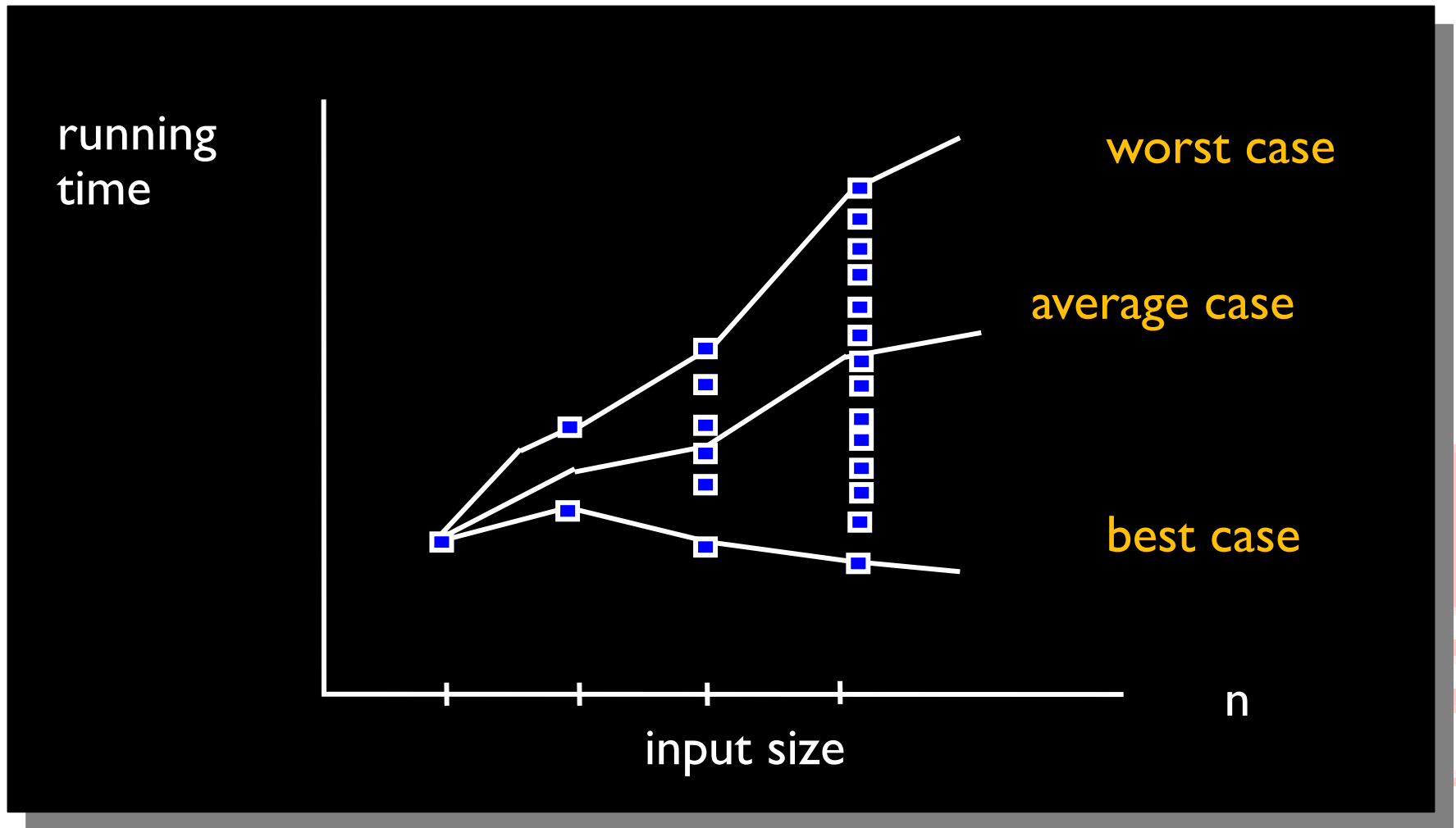
Analisa Algoritma: *How?*

- Contoh: Mencari elemen terkecil dalam sebuah array
 - Algoritma: *sequential scan / linear search*
 - Orde-nya: $O(n)$ – linear

```
public static int smallest (int a[])
{
    assert (array.length > 0);
    int elemenTerkecil = a[0];
    for (ii = 1; ii < a.length; ii++) {
        if (a[ii] < elemenTerkecil) {
            elemenTerkecil = a[ii];
        }
    }
    return elemenTerkecil ;
}
```

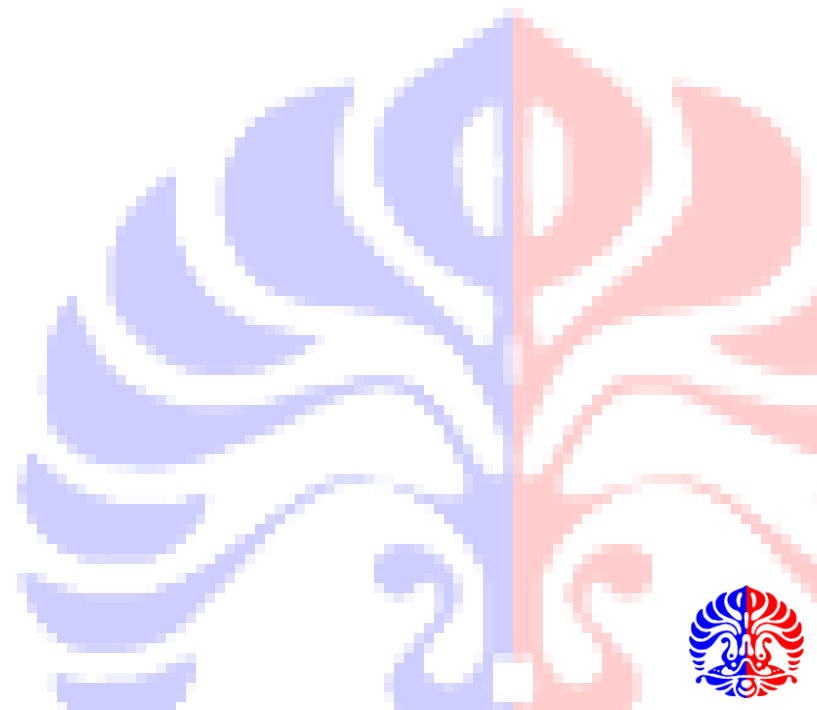


Analisa Algoritma: *How?*



Big Oh: contoh

- Sebuah fungsi kubik adalah sebuah fungsi yang suku dominannya (*dominant term*) adalah sebuah konstan dikalikan dengan n^3 .
- Contoh:
 - $10 n^3 + n^2 + 40 n + 80$
 - $n^3 + 1000 n^2 + 40 n + 80$
 - $n^3 + 10000$



Dominant Term

- Mengapa hanya suku yang memiliki pangkat tertinggi/dominan saja yang diperhatikan?
- Untuk n yang *besar*, suku dominan lebih mengindikasikan perilaku dari algoritma.
- Untuk n yang *kecil*, suku dominan tidak selalu mengindikasikan perilakunya, *tetapi* program dengan input kecil umumnya berjalan sangat cepat sehingga kita tidak perlu perhatikan.



Big Oh: issues

- Apakah fungsi linier selalu lebih kecil dari fungsi kubik?
 - Untuk n yang kecil, bisa saja fungsi linier $>$ fungsi kubik
 - Tetapi untuk n yang besar, fungsi kubik $>$ fungsi linier
 - Contoh:
 - $f(n) = 10 n^3 + 20 n + 10$
 - $g(n) = 10000 n + 10000$
 - $n = 10, f(n) = 10.210, g(n) = 110.000$
→ $f(n) < g(n)$
 - $n = 100, f(n) = 10.002.010, g(n) = 1.010.000$
→ $f(n) > g(n)$
 - Mengapa nilai konstan/koeffesien pada setiap suku tidak diperhatikan?
 - Nilai konstan/koeffesien tidak berarti pada mesin yang berbeda
- ∴ **Big Oh** digunakan untuk merepresentasikan laju pertumbuhan (*growth rate*)



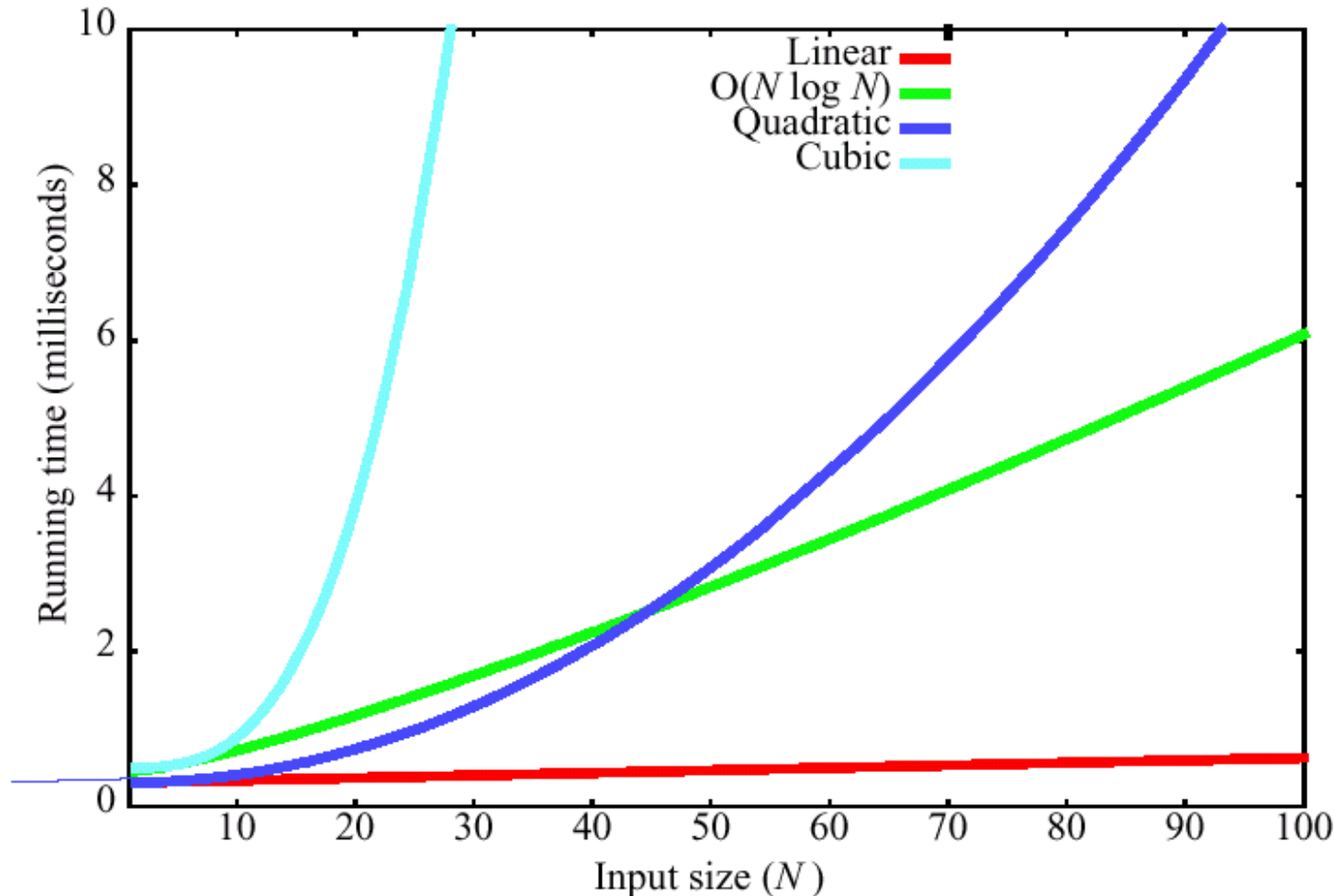
Orde Fungsi Running-Time

| Function | Name |
|------------|-------------|
| c | Constant |
| $\log N$ | Logarithmic |
| $\log^2 N$ | Log-squared |
| N | Linear |
| $N \log N$ | $N \log N$ |
| N^2 | Quadratic |
| N^3 | Cubic |
| 2^N | Exponential |
| N! | Factorial |



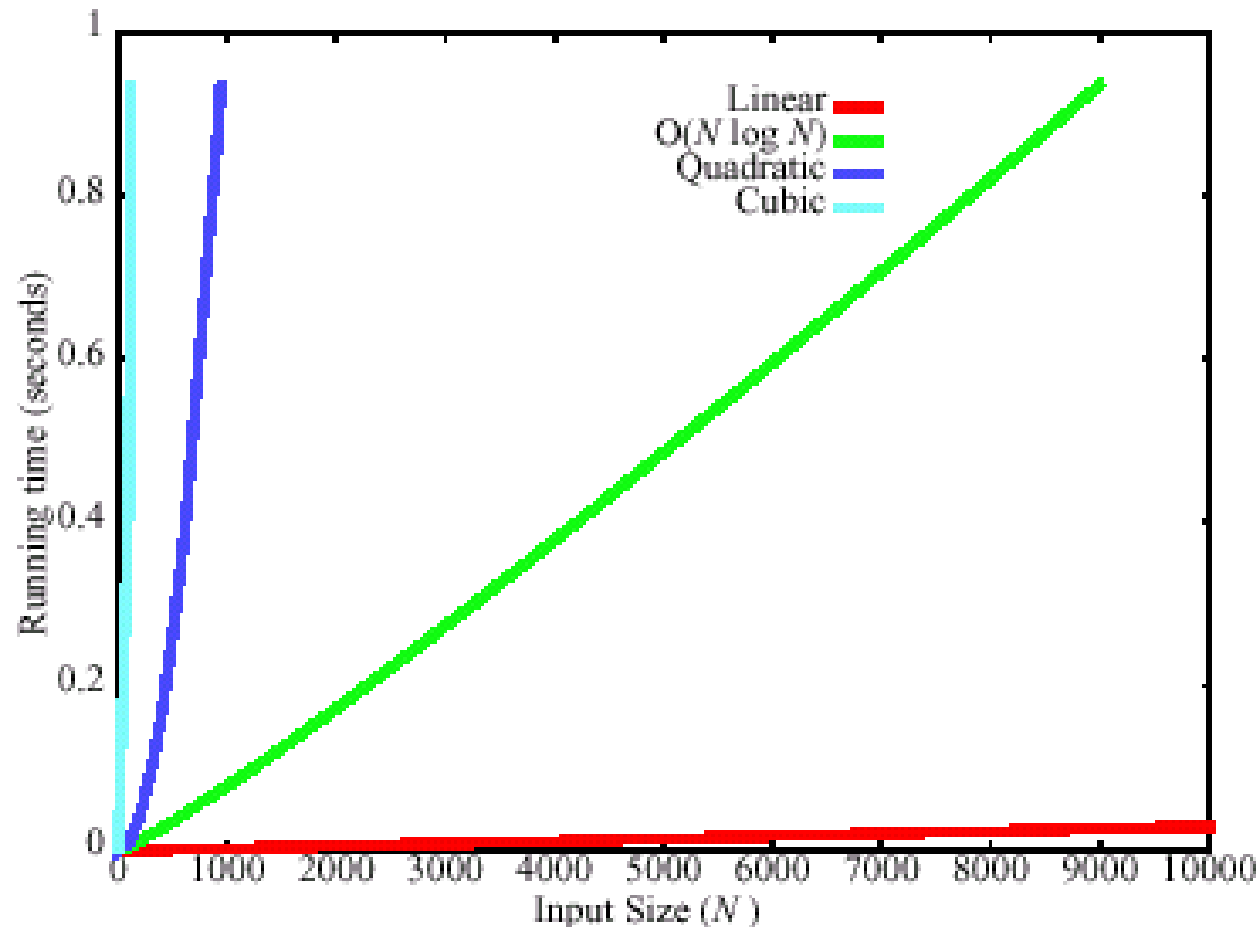
Fungsi Running-Time

- Untuk input yang sedikit, beberapa fungsi lebih cepat dibandingkan dengan yang lain.



Fungsi Running-Time (2)

- Untuk input yang besar, beberapa fungsi *running-time* sangat lambat - tidak berguna.



Contoh Algoritma

- Mencari dua titik yang memiliki jarak terpendek dalam sebuah bidang (koordinat X-Y)
 - Masalah dalam komputer grafis.
 - Algoritma *brute force*:
 - hitung jarak dari semua pasangan titik
 - cari jarak yang terpendek
 - Jika jumlah titik adalah n , maka jumlah semua pasangan adalah $C(n, 2) = n * (n - 1) / 2$
 - Orde-nya: $O(n^2)$ - kuadratik
 - Ada solusi yang $O(n \log n)$



Contoh Algoritma

- Tentukan apakah ada tiga titik dalam sebuah bidang yang segaris (*colinier*).
 - Algoritma *brute force*:
 - periksa semua tripel titik yang terdiri dari 3 titik.
 - Jumlah pasangan: $C(n, 3) = n * (n - 1) * (n - 2) / 6$
 - Orde-nya: $O(n^3)$ - kubik. Sangat tidak berguna untuk 10.000 titik.
 - Ada algoritma yang kuadratik.



Studi Kasus

- Mengamati sebuah masalah dengan beberapa solusi.
- Masalah *Maximum Contiguous Subsequence Sum*
 - Diberikan (angka integer negatif dimungkinkan) A_1, A_2, \dots, A_N , cari nilai maksimum dari $(A_i + A_{i+1} + \dots + A_j)$.
- *Maximum contiguous subsequence sum* adalah nol jika semua integer adalah negatif.
- Contoh (*maximum subsequences* digarisbawahi)
 - $2-, \underline{11}, 4-, \underline{13}, 4-, 2$
 - $1, 3-, \underline{4}, 2-, \underline{1-}, 6$



Brute Force Algorithm (1)

- Algoritma:
 - Hitung jumlah dari semua *sub-sequence* yang mungkin
 - Cari nilai maksimumnya
- Contoh:
 - jumlah *subsequence* (*start*, *end*)
 - (0,0), (0,1), (0,2), ..., (0,5)
 - (1,1), (1,2), ..., (1,5)
 - ...
 - (5,5)

| | | | | | |
|----|----|----|----|----|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
| -2 | 11 | -4 | 13 | -4 | 2 |



Brute Force Algorithm (2)

```
public static int maxSubSum1 (int [] A)
{
    int maxSum = 0;
    for (int ii = 0; ii < A.length; ii++) {
        for (int jj = ii; jj < A.length; jj++) {
            int thisSum = 0;
            for (int kk = ii; kk <= jj; kk++) {
                thisSum += A[kk];
            }
            if (thisSum > maxSum) {
                maxSum = thisSum;
            }
        }
    }
    return maxSum;
}
```



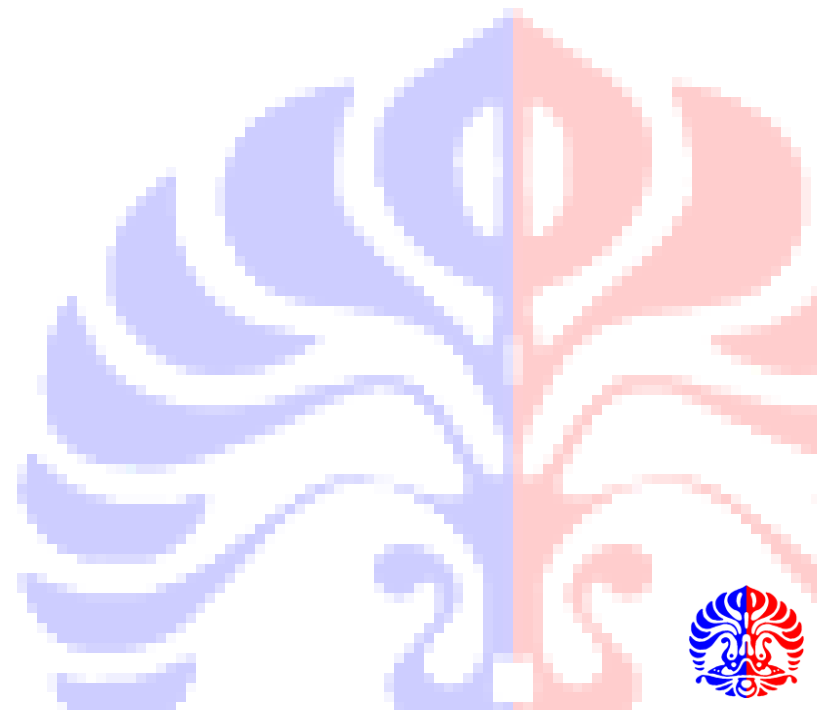
↑
ii

↑
jj



Analisa

- Iterasi (kk) sebanyak N dalam iterasi (jj) sebanyak N dalam iterasi (ii) sebanyak N artinya: $O(N^3)$, atau algoritma kubik.
- *Slight over-estimate* yang dihasilkan dari perhitungan iterasi yang kurang dari N adalah tidak terlalu penting ($kk \leq jj$).



Running Time yang sesungguhnya

- Untuk $N = 100$, waktu sebenarnya adalah 0.47 detik pada sebuah komputer.
- Dapat menggunakan informasi tersebut, untuk memperkirakan waktu untuk input yang lebih besar:

$$T(N) = cN^3$$

$$T(10N) = c(10N)^3 = 1000cN^3 = 1000T(N)$$

- Ukuran input meningkat dengan kelipatan 10 kali, yang artinya meningkatkan *running time* dengan kelipatan 1000 kali.
- Untuk $N = 1000$, perkiraan waktu 470 detik. (waktu sebenarnya 449 detik).
- Untuk $N = 10,000$, perkiraan waktu 449000 detik (6 hari).



Bagaimana memperbaikinya?

- Membuang sebuah iterasi; Tidak selalu bisa.
- Dalam contoh sebelumnya dimungkinkan: iterasi paling dalam (*kk*) tidak diperlukan karena informasi nya terbuang.
- Nilai **thisSum** untuk nilai **jj** selanjutnya dapat dengan mudah diperoleh dari nilai **thisSum** yang sebelumnya:
 - Yang diperlukan: $A_{ii} + A_{ii+1} + \dots + A_{jj-1} + A_{jj}$
 - Yang telah dihitung: $A_{ii} + A_{ii+1} + \dots + A_{jj-1}$
 - Yang dibutuhkan adalah yang telah dihitung + A_{jj}
- Dengan kata lain:
 - **$\text{sum}(ii, jj) = \text{sum}(ii, jj - 1) + A_{jj}$**



The Better Algorithm

```
public static int maxSubSum2 (int [ ] A)
{
    int maxSum = 0;
    for (int ii = 0; ii < A.length; ii++)
    {
        int thisSum = 0;
        for (int jj = ii; jj < A.length; jj++)
        {
            thisSum += A[jj];
            if (thisSum > maxSum)
                maxSum = thisSum;
        }
    }
    return maxSum;
}
```



Analisa

- Dengan logika yang sama: Saat ini *running time* adalah *quadratic*, or $O(N^2)$
- Perkirakan waktu eksekusi untuk input berukuran ribuan. Algoritma ini masih dapat digunakan untuk ukuran input 10 ribu.
- Ingat, bahwa algoritma kubik sudah tidak *practical* lagi bila digunakan untuk ukuran input tersebut.



Actual running time

- Untuk $N = 100$, waktu sebenarnya adalah 0.011 detik pada sebuah komputer.

- Perkirakan waktu sebenarnya untuk input lebih besar:

$$T(N) = cN^2$$

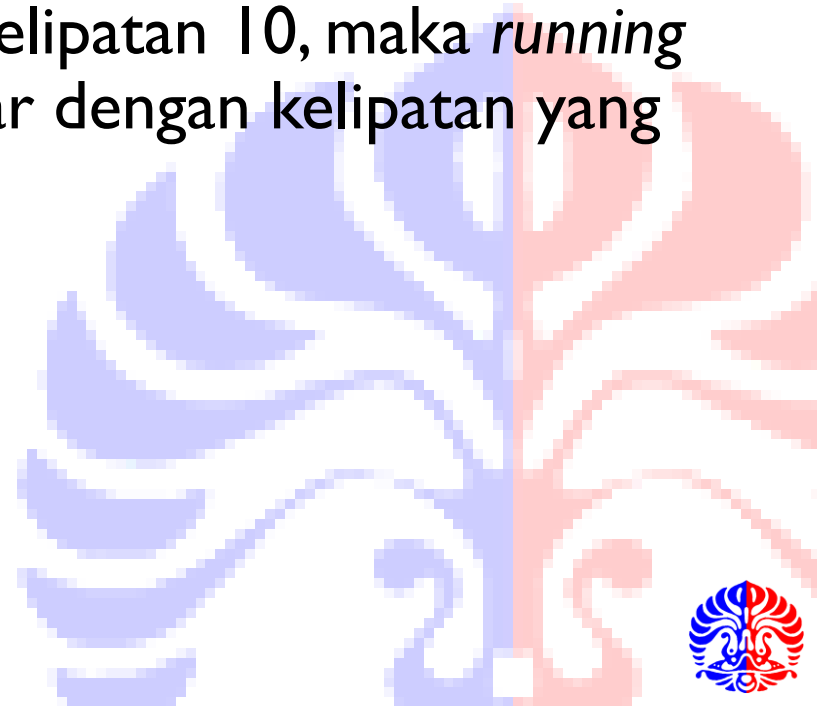
$$T(10N) = c(10N)^2 = 100cN^2 = 100T(N)$$

- Input diperbesar dengan kelipatan 10 artinya *running time* akan membesar dengan kelipatan 100.
- Untuk $N = 1000$, perkiraan *running time* adalah 1.11 detik. (waktu sebenarnya 1.12 detik).
- Untuk $N = 10,000$, perkiraan 111 detik (= actual).



Algoritma linear

- Algoritma Linear merupakan yang terbaik.
- Ingat: linear artinya $O(N)$.
- *Running time* yang linear selalu sebanding dengan ukuran input. Sulit untuk membuat algoritma yang lebih baik dari linear (kecuali ada rumusan tertentu).
- Jika input membesar dengan kelipatan 10, maka *running time* juga hanya akan membesar dengan kelipatan yang sama.



| | | | |
|---------------|-----|-------------|-----|
| i | j | $j+1$ | q |
| < 0 | | $S_{j+1,q}$ | |
| $< S_{j+1,q}$ | | | |

- $A_{i,j}$ adalah kumpulan bilangan mulai dari urutan i hingga urutan j .
- $S_{i,j}$ adalah jumlah dari kumpulan bilangan tersebut.
- Theorema: Untuk $A_{i,j}$ dengan $S_{i,j} < 0$. Jika $q > j$, maka $A_{i,q}$ bukanlah deretan terurut yang terbesar.



Bukti Theorema

■ Bukti:

- $S_{i,q} = S_{i,j} + S_{j+1,q}$

- $S_{i,j} < 0 \Rightarrow S_{i,q} < S_{j+1,q}$

| | | | |
|---------------|-----|-------------|-----|
| i | j | $j+1$ | q |
| < 0 | | $S_{j+1,q}$ | |
| $< S_{j+1,q}$ | | | |



Program – versi 1

```
static public int
maximumSubSequenceSum3 (int a[])
{
    int maxSum = 0;
    int thisSum = 0;
    for (int jj = 0; jj < a.length; jj++) {
        thisSum += a[jj];
        if (thisSum > maxSum) {
            maxSum = thisSum;
        } else if (thisSum < 0) {
            thisSum = 0;
        }
    }
    return maxSum;
}
```



Program – versi 2

```
static public int
maximumSubSequenceSum3b (int a[])
{
    int thisSum = 0, maxSum = 0;
    for (int ii = 0; ii < a.length; ii++) {
        thisSum = Math.max(0, a[ii] + thisSum);
        maxSum  = Math.max(thisSum, maxSum);
    }
    return maxSum;
}
```



Running Time

Observed Running Time (in seconds)
for various max contiguous subsequence sum algorithms

| | Figure 5.4 | Figure 5.5 | Figure 7.20 | Figure 5.8 |
|---------|------------|------------|---------------|------------|
| N | $O(N^3)$ | $O(N^2)$ | $O(N \log N)$ | $O(N)$ |
| 10 | 0.000009 | 0.000004 | 0.000006 | 0.000003 |
| 100 | 0.002580 | 0.000109 | 0.000045 | 0.000006 |
| 1,000 | 2.281013 | 0.010203 | 0.000485 | 0.000031 |
| 10,000 | NA | 1.2329 | 0.005712 | 0.000317 |
| 100,000 | NA | 135 | 0.064618 | 0.003206 |



Running Time: on different machines

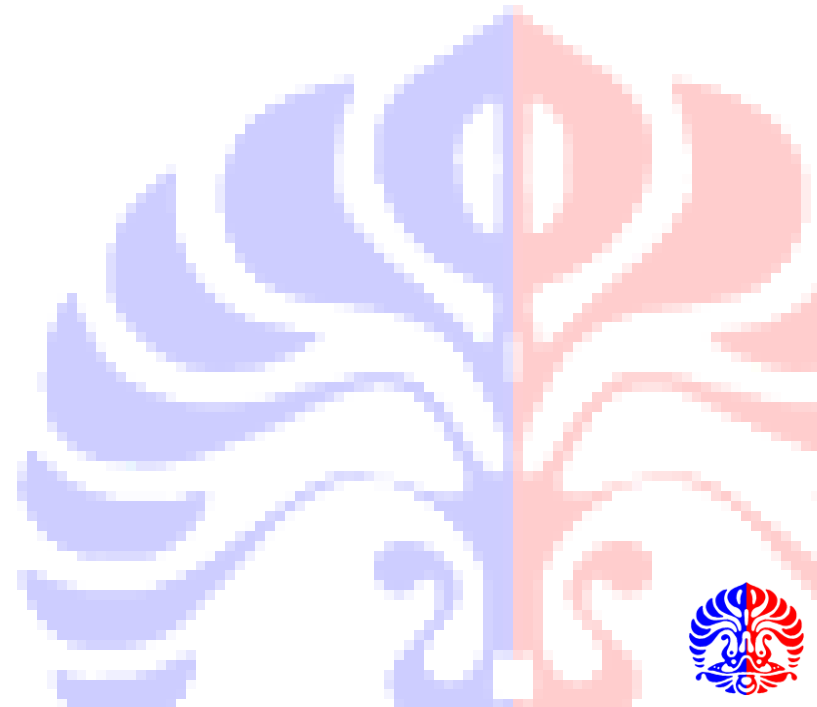
- Algoritma Kubik pada mesin Alpha 21164 at **533 Mhz** menggunakan **C compiler** (mewakili mesin yang cepat – secara relatif)
- Algoritma Linear pada mesin Radio Shack TRS-80 Model III (a 1980 personal computer with a Z-80 processor running at **2.03 Mhz**) using **interpreted Basic** (mewakili mesin lambat)

| n | Alpha 21164A, C compiled, Cubic Algorithm | TRS-80, Basic interpreted, Linear Algorithm |
|------------------|--|--|
| 10 | 0.6 microsecs | 200 milisecs |
| 100 | 0.6 milisecs | 2.0 secs |
| 1,000 | 0.6 secs | 20 secs |
| 10,000 | 10 mins | 3.2 mins |
| 100,000 | 7 days | 32 mins |
| 1,000,000 | 19 yrs | 5.4 hrs |



Running Time: Moral Of The Story

- Bahkan teknik programming yang terbaik tak akan dapat membuat sebuah algoritma yang tidak efisien menjadi cepat.
- Sebelum kita menginvestasikan waktu untuk mencoba mengoptimisasi program, kita harus pastikan algoritma-nya sudah yang paling efisien.



Algoritma Logaritme

■ Definisi Formal

- Untuk setiap $B, N > 0$, $\log_B N = K$, if $B^K = N$.
- Jika (base) B diabaikan, maka default-nya adalah 2 dalam konteks ilmu komputer (binary representation).

■ Contoh:

- $\log 32 = 5$ (karena $2^5 = 32$)
- $\log 1024 = 10$
- $\log 1048576 = 20$
- $\log 1 \text{ milyar} = \text{sekitar } 30$
- logaritme berkembang jauh lebih lambat dari N dan lebih lambat dari akar kuadrat dari N .



Contoh Algoritma Logaritme

■ BITS IN A BINARY NUMBER

- Berapa banyak bits dibutuhkan untuk merepresentasikan bilangan bulat?

■ REPEATED DOUBLING

- Mulai dari $X = 1$, berapa kali X harus dikalikan dua agar mendekati nilai N ?

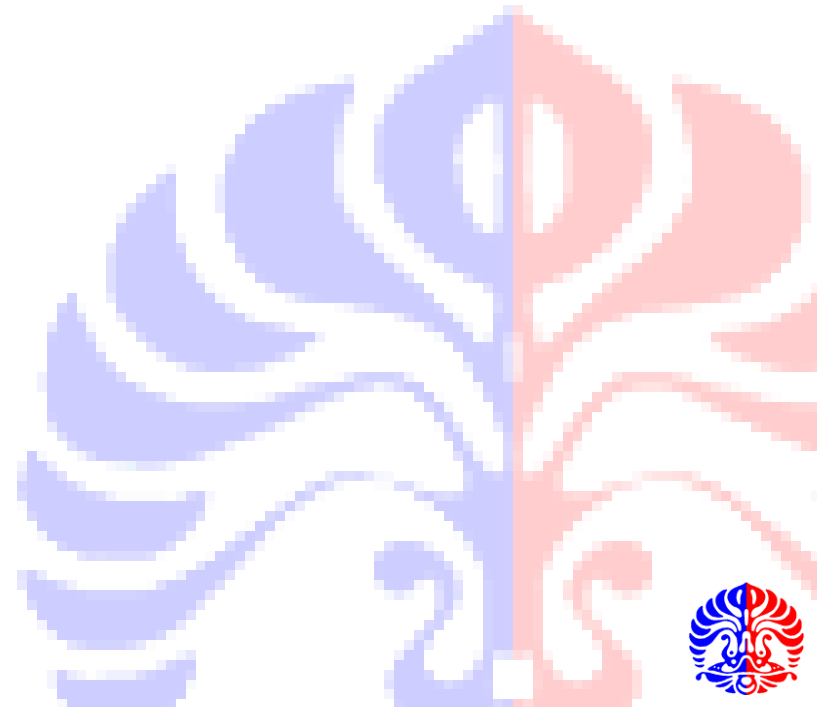
■ REPEATED HALVING

- Mulai dari $X = N$, jika N dibagi dua terus menerus, berapa kali iterasi agar membuat N lebih kecil atau sama dengan 1 (Halving rounds up).
- Jawaban untuk seluruh pertanyaan diatas adalah: $\log N$ (dibulatkan).



Repeated Halving Principle

- Sebuah algoritma adalah $O(\log N)$ jika membutuhkan waktu konstan untuk membagi input permasalahan dan mengerjakan masing-masing-nya secara rekursif. (biasanya $1/2$).
- Penjelasan: Akan terjadi sebanyak $\log N$ dari proses konstan tersebut.



Big-Oh Rules (1)

- Ekspresi matematika untuk menyatakan tingkat laju pertumbuhan relatif:
 - **DEFINITION:** (Big-Oh) $T(N) = O(F(N))$ jika ada konstan positif c dan N_0 sehingga $T(N) \leq c F(N)$ untuk $N \geq N_0$.
 - **DEFINITION:** (Big-Omega) $T(N) = \Omega(F(N))$ jika ada konstan c dan N_0 sehingga $T(N) \geq c F(N)$ untuk $N \geq N_0$.
 - **DEFINITION:** (Big-Theta) $T(N) = \Theta(F(N))$ jika dan hanya jika $T(N) = O(F(N))$ dan $T(N) = \Omega(F(N))$.
 - **DEFINITION:** (Little-Oh) $T(N) = o(F(N))$ jika dan hanya jika $T(N) = O(F(N))$ dan $T(N) \neq \Theta(F(N))$.



Big-Oh Rules (2)

■ Arti dari beberapa fungsi tingkat laju.

| | |
|-----------------------|---|
| $T(N) = O(F(N))$ | Growth of $T(N)$ is \leq growth of $F(N)$ |
| $T(N) = \Omega(F(N))$ | Growth of $T(N)$ is \geq growth of $F(N)$ |
| $T(N) = \Theta(F(N))$ | Growth of $T(N)$ is $=$ growth of $F(N)$ |
| $T(N) = o(F(N))$ | Growth of $T(N)$ is $<$ growth of $F(N)$ |

■ Jika ada lebih dari satu parameter, maka aturan tersebut berlaku untuk setiap parameter.

■ $4n \log(m) + 50n^2 + 500m + 1853$

→ $O(n \log(m) + n^2 + m)$

■ $4m \log(m) + 50n^2 + 500m + 853$

→ $O(m \log(m) + n^2)$



Latihan

- Urutkan fungsi berikut berdasarkan laju pertumbuhan (growth rate)
 - $N, \sqrt{N}, N^{1.5}, N^2, N \log N, N \log \log N, N \log^2 N, N \log (N^2), 2/N, 2^N, 2^{N/2}, 37, N^3, N^2 \log N$
- $A^5 + B^5 + C^5 + D^5 + E^5 = F^5$
 - $0 < A \leq B \leq C \leq D \leq E \leq F \leq 75$
 - Hanya memiliki satu solusi. Berapa?



Summary

■ Algoritma

- Suatu set instruksi (*unambiguous*) yang harus diikuti oleh komputer untuk memecahkan suatu masalah, dan harus berhenti (*terminate*) dalam batas waktu yang wajar (*reasonable*)

■ Analisa algoritma

- banyak *resource* (*time* & *space*) yang diperlukan oleh sebuah algoritma berdasarkan besarnya input
- menggunakan Big Oh notation – dominant terms, upper bound
 - Bukan menggunakan *wall clock*
- *best case, average case, worst case*

