HANDONG GLOBAL UNIVERSITY

Chapter 3
Growth of Functions

Algorithm Analysis

School of CSEE

Growth of Function

time complexity

- In this chapter we will study growth of function.

time complexity

- The order of growth of the running time of an algorithm gives a simple characterization of the algorithm's efficiency and also allows us to compare the relative performance of alternative algorithms.

n^2 / n^3 / nlogn

Growth of Functions

- Although we can sometimes determine the exact running time of an algorithm, as we did for insertion sort in Chapter 2, the extra precision is not usually worth the effort of computing it.
- ***Asymptotic*** efficiency - how the running time increases with the size of the input *in the limit*.
- Focus on what's important by abstracting away *low-order terms* and *constant factors*.

Notations

Theta	$f(n) = \theta(g(n))$	$f(n) \approx c\ g(n)$ 가이 ∞(무한대)로 갔을 때 f(n)이 cg(n)처럼 비슷한 경우
BigOh # Upper Bound.	$f(n) = O(g(n))$	$f(n) \leq c\ g(n)$ f(n)이 아무리 커봤자 cg(n)보다 클 수 없는 경우
Big Omega #Lower Bound.	$f(n) = \Omega(g(n))$	$f(n) \geq c\ g(n)$ 가이 ∞(무한대)로 갔을 때 f(n)이 cg(n)보다 큰 경우
Little Oh	$f(n) = o(g(n))$	$f(n) < c\ g(n)$ f(n)과 cg(n)보다 크거나 같을 수 없는 경우
Little Omega	$f(n) = \omega(g(n))$	$f(n) > c\ g(n)$ f(n)이 cg(n)보다 작거나 같을 수 없는 경우

O-notation – upper bound

- $O(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le c\, g(n)$ for all $n \ge n_0 \}$

  ↳ time complexity에서 음수 고려 X. — set

  If $f(n) \in O(g(n))$, we write $f(n) = O(g(n))$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to c < \infty \quad \Rightarrow \quad f(n) = O(g(n))$$

(분모가 0 ↑ 0보다 큰 정수)

Example : $2n^2 + 1 = O(n^2)$, with c = ? and n_0 = ?

Q) $2n^2 = O(n^3)$, with $c=1$ and $n_0=2$

A) $2n^2 = cg(n)$

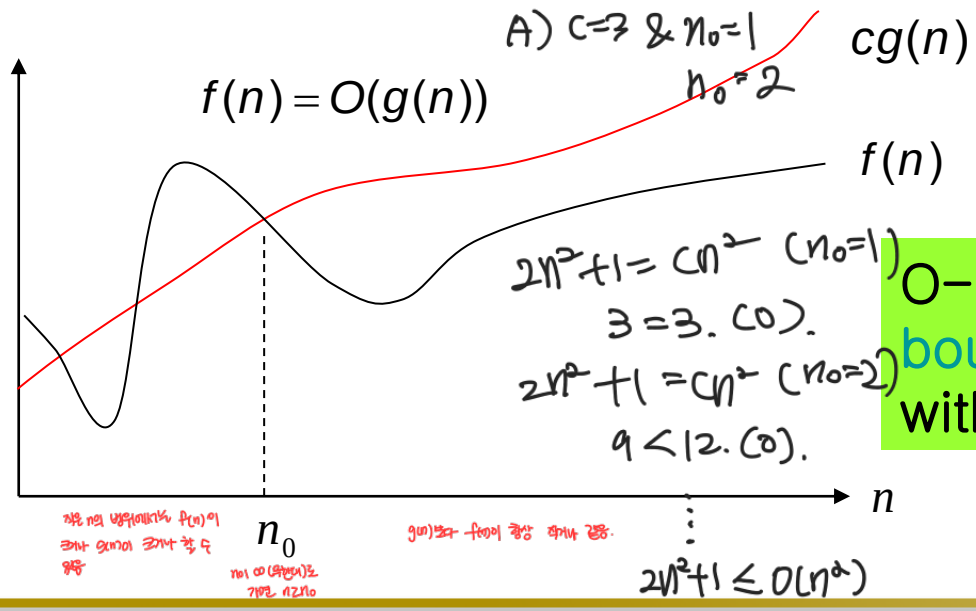
$2n^2 = cn^3$ ($c=1$, $n_0=2$ 대입)

$8 = 8$

$2n^2 = cn^3$ ($c=1$, $n_0=3$ 대입)

$18 \not\le 2n$

(g(n)보다 작거나 같으면 안됨)



↑ upper bound, f(n)은 g(n)을 넘어갈 수 X.

특정 n에 대해 되면 f(n) ≤ cg(n)

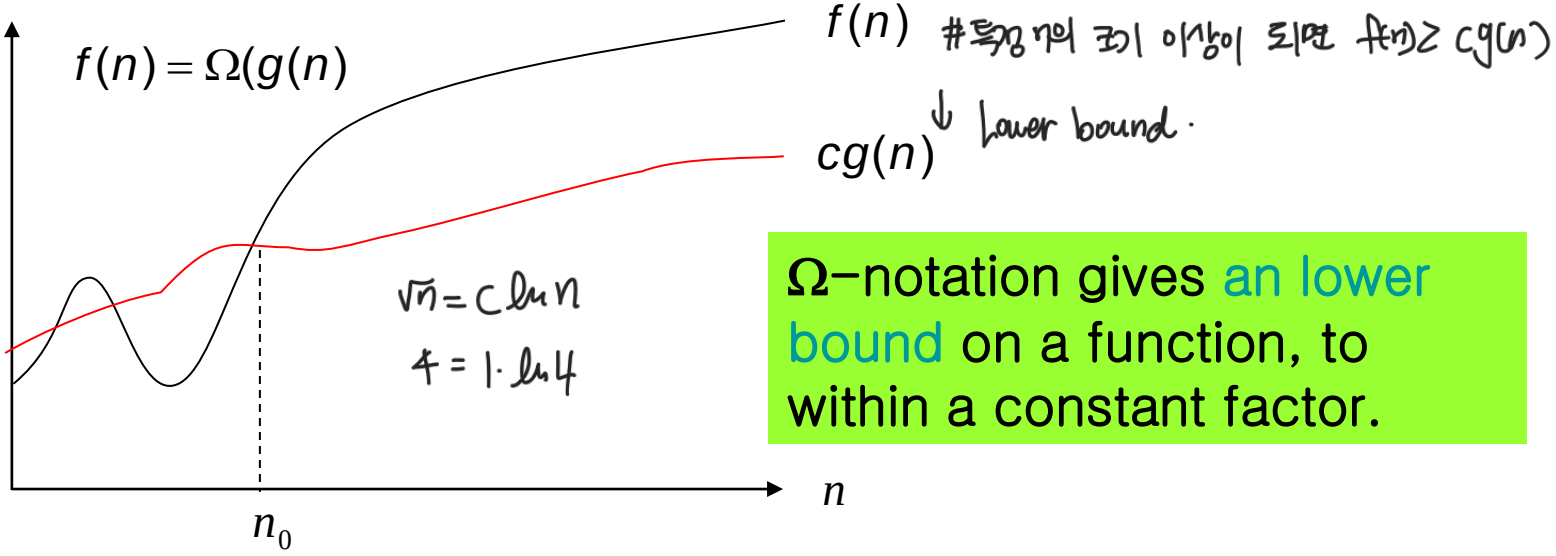
O–notation gives an upper bound on a function, to within a constant factor.

Ω-notation – lower bound

- $\Omega(g(n))$ = { $f(n)$: there exist positive constants c and n_0 such that $0 \le c\, g(n) \le f(n)$ for all $n \ge n_0$ }

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to c > 0 \quad \Rightarrow \quad f(n) = \Omega(g(n))$$

Example : $\sqrt{n} = \Omega(\lg n)$, with $c = 1$ and $n_0 = 16$



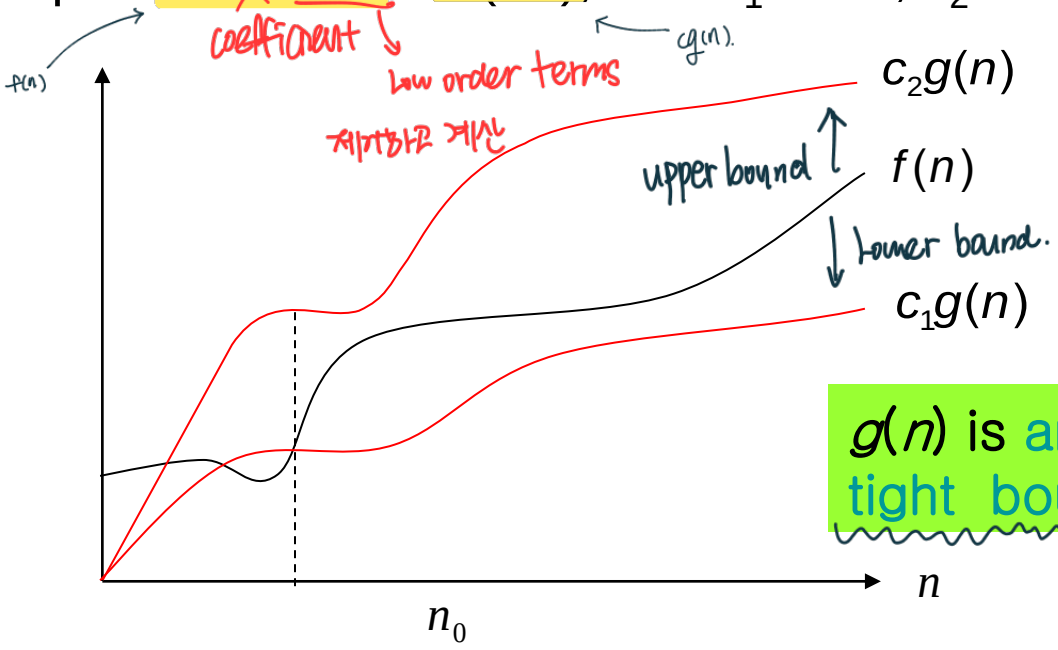
Ω−notation gives an lower bound on a function, to within a constant factor.

Θ-notation – tight (exact) bound

- $\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} \to c < \infty \Rightarrow \quad f(n) = \Theta(g(n))$$

Example : $n^2/2 - 2n = \Theta(n^2)$ with $c_1 = 1/4$, $c_2 = 1/2$, $n_0 = 8$



$g(n)$ is an asymptotically tight bound for $f(n)$.

Theorem and Rule

Theorem 3.1

For two functions *f*(*n*) and *g*(*n*), we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

L'Hôpital's Rule 로피탈 정리

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

$\lim_{n\to\infty} \frac{(\log n)'}{n'} = \lim_{n\to\infty} \frac{\frac{1}{n}}{1} = \lim_{n\to\infty} \frac{1}{n} = 0$

→ upper bound에 해당

$\log n = O(n)$

o-notation

little-o

- o($g(n)$) = { $f(n)$: for any positive constant $c > 0$, there exist a constant $n_0 > 0$ such that $0 \le f(n) < c\, g(n)$ for all $n \ge n_0$ }

  equal sign 빠짐 (from big-notation)

  $f(n)$ is ***asymptotically smaller*** than $g(n)$ if $f(n) = o(g(n))$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to 0 \Rightarrow \quad f(n) = o(g(n))$$

equal sign (=) 빠져서 OO.

$$n - 5,\ n,\ n^2,\ 10^{10}n^2 + 10^5 n + 10^9,\ n^2 - 9 \in o(n^3)$$

$o(f) = O(f) - \Theta(f)$.

equal.

o(f) is usually called "little oh of f".

$n^3 = O(n^3)$ (O)

$n^3 \in O(n^3)$
$n^3 \in \Theta(n^3)$
가능.
$n^3 \notin o(n^3)$ 임.

$n^3 \notin o(n^3)$

왜냐하면 n^3은 $\Theta(n^3)$이기 때문.
Θ부분을 제거해야함.

ω-notation

- $\omega(g(n))$ = { $f(n)$: for any positive constant $c > 0$, there exist a constant $n_0 > 0$ such that $0 \le c\, g(n) < f(n)$ for all $n \ge n_0$ }

  $f(n)$ is ***asymptotically larger*** than $g(n)$ if $f(n) = \omega(g(n))$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \infty \;\Rightarrow\; f(n) = \omega(g(n))$$

$$n^2, 10^{10}n^2 + 10^5 n + 10^9, n^3 - 9 \in \omega(n)$$

Note: $g(n) = o(f(n)) \Leftrightarrow f(n) = \omega(g(n))$

$\omega(f) = \Omega(f) - \theta(f)$.

$\omega(f)$ is usually called "little omega of f"

$n = \Omega(n)$ (O)

$n = \theta(n)$

$$n \in \omega(n)$$

Example

Two functions *f* and *g* have the same order if and only if

$$\Theta(f(n)) = \Theta(g(n))$$

$$\Theta((8n^3 + n)^{1/2} + \log n)$$
$$= \Theta(2n^{3/2} + 6n \ln n)$$
$$= \Theta(n^{3/2}) \quad \leftarrow \text{simplest form}$$
$$= \Theta(n^{3/2} + 23n^{1/3})$$
$$= \Theta\left(\frac{n^3 + n + 1}{4n^{3/2} + \ln n}\right)$$

Also specify O or θ

Proposition

Θ determines an equivalence relation on F – i.e., for $f(n), g(n) \in F$

1. $f(n) \in \Theta(f(n))$ **(reflexive)**
2. $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$ ∵ same order **(symmetric)**
3. $f(n) \in \Theta(g(n))$ **and** $g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$ **(transitive)**

Two functions *f* and *g* have the same order if $f(n) \in \Theta(g(n))$

* Read page 51 & 52

Time Complexity of an *Algorithm*

The time complexity of an algorithm is the largest time required on any input of size n.

$O(n^2)$: Upper bound on the worst case running time of an algorithm.

넘지 않는다.

$\Omega(n^2)$: Lower bound on the best case of running time of an algorithm.

보단 빠르다.

$\theta(n^2)$: Do both.

적절하게 작성하기

↓

$A: n^2+n$, $\theta(n^2) \rightarrow O(n^2) \rightarrow O(n^3) \rightarrow O(n!)$

very large version

Optimality of algorithm

- Each problem has inherent complexity; that is, there is some minimum amount of work required to solve it.
- Lower bound of problem : minimal number of operation needed to solve the problem
- When the worst-case time complexity of an algorithm matches lower bound of the problem, the algorithm is said to be optimal.
- "Optimal" does not mean "the best known"; it means "the best possible".

Example

Time for several algorithms for a given problem

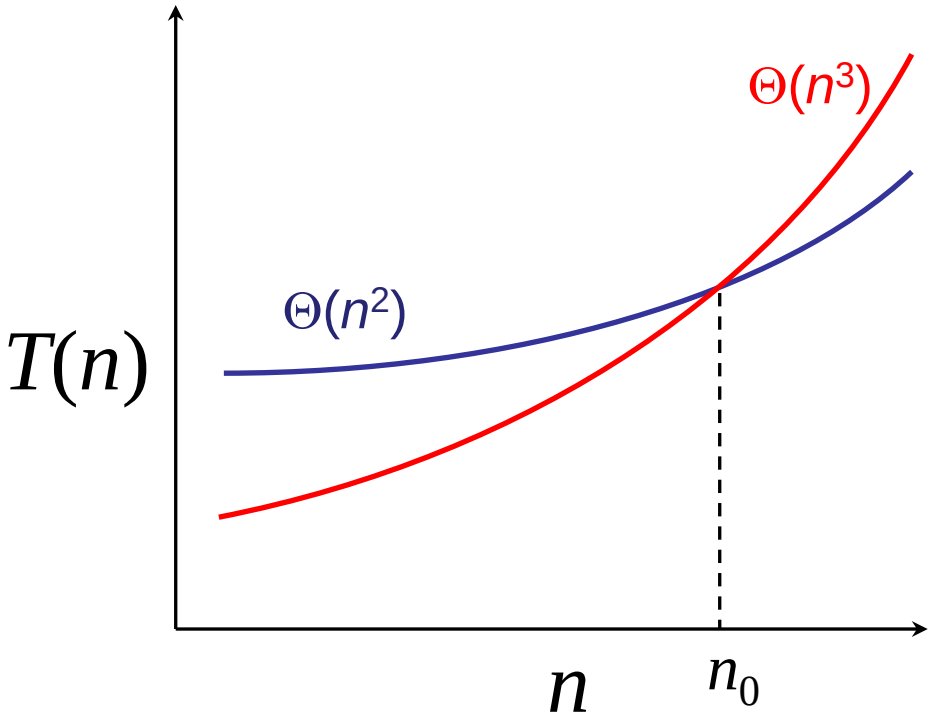
- ✓ Algorithm1 : $\theta(n^2)$
- ✓ Algorithm2 : $\theta(n^2 \lg n)$
- ✓ Algorithm3 : $\theta(n^3)$

➔ If someone proves the given problem needs at least n^2 basic operations, then the algorithm1 is optimal.

이거보다 더 빠르게 수행할 수 없기 때문에 optimal하다.

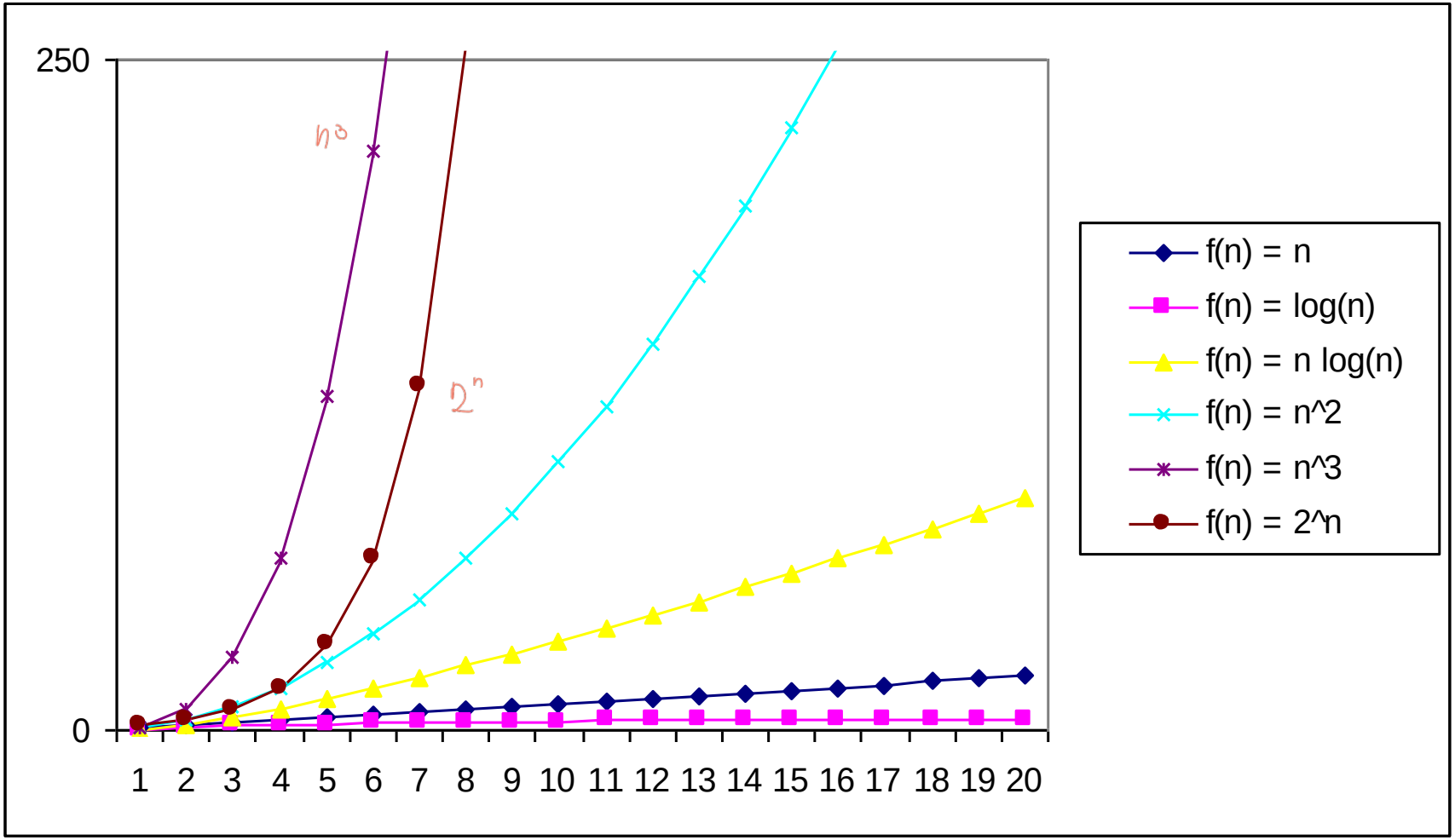
Asymptotic performance

When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.

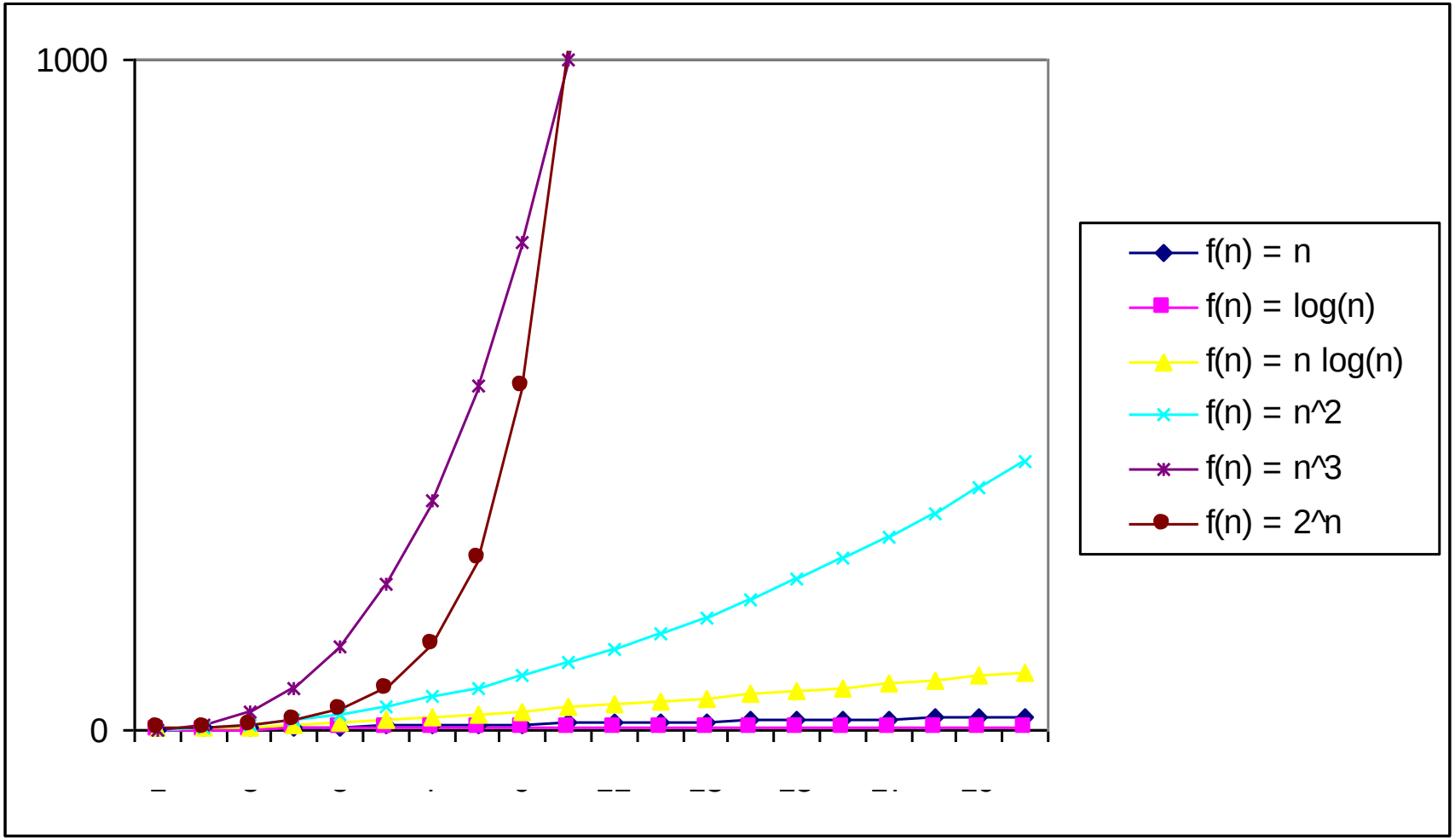


- Asymptotic analysis is a useful tool to help to structure our thinking toward better algorithm.
- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing

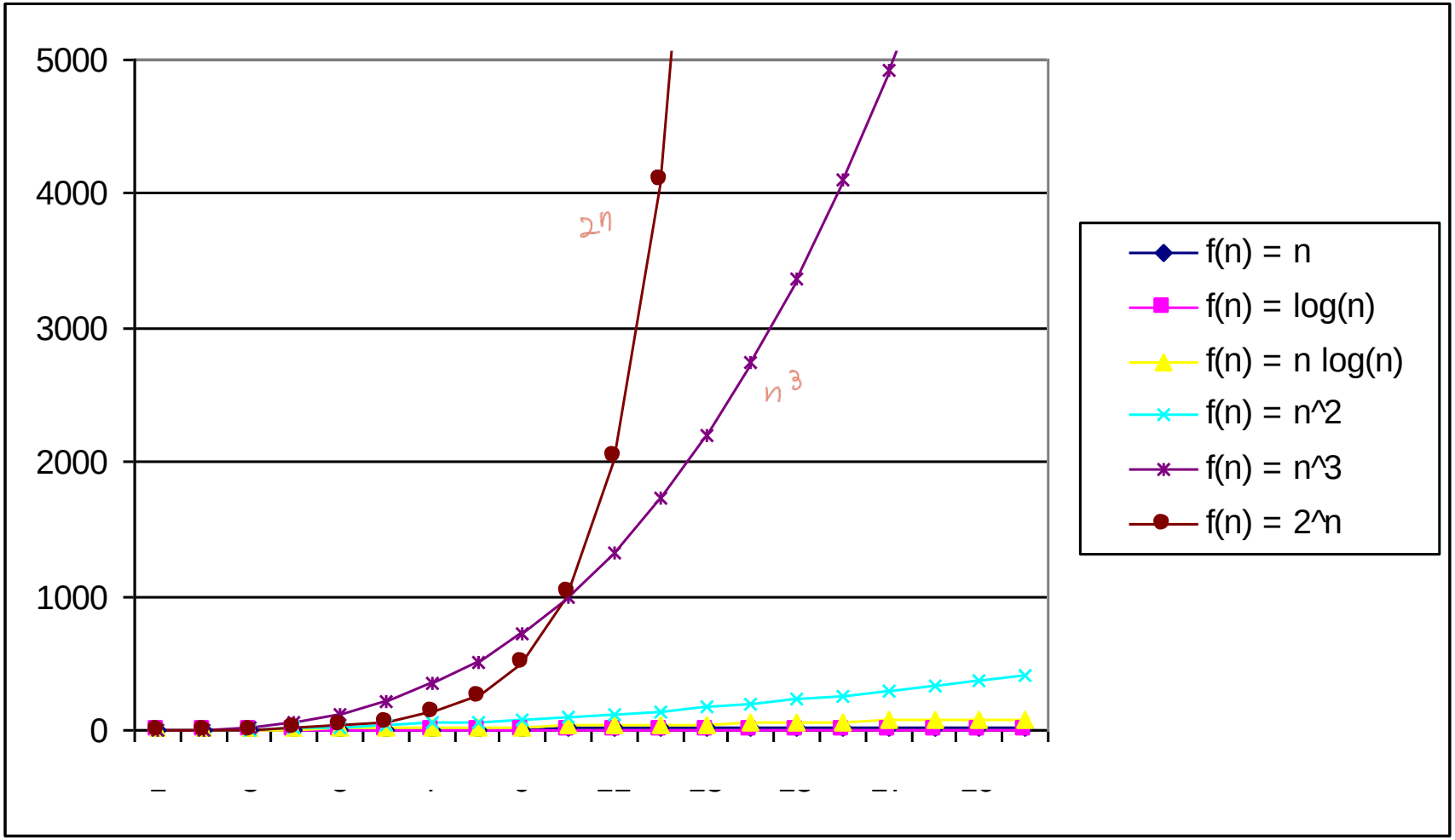
Practical Complexity



Practical Complexity



Practical Complexity



Hierarchy of Orders

Given function $f(n)$ and $g(n)$, we say that $f(n)$ has smaller order than $g(n)$, if $O(f(n))$ is strictly contained in $O(g(n))$.

,i.e., $O(f(n)) \subset O(g(n))$.

Example:

$$O(1) \subset O(\log n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \log n)$$
$$\subset O(n^2) \subset O(n^3) \subset O(2^n) \subset O(n2^n) \subset O(n!)$$

Where does $O(n^{1/1000})$ belong?

Time complexity comparison

빠른 software (algorithm)을 만드는 것은 중요함.

n	Cray-1 Fortran[a] $3n^3$ nanoseconds	TRS-80 Basic[b] 19,500,000n nanoseconds
10	3 microseconds	.2 seconds
100	3 milliseconds	2.0 seconds
1,000	3 seconds	20.0 seconds
2,500	50 seconds	50.0 seconds
10,000	49 minutes	3.2 minutes
1,000,000	95 years	5.4 hours

[a] Cray-1 is a trademark of Cray Research, Inc.

[B] TRS-80 is a trademark of Tandy Corporation.

Time complexity comparison

Algorithm	1	2	3	4	5
Time function (microsec.)	33n	46n lg n	$13n^2$	$3.4n^3$	2^n
Input size(n)	Solution time				
10	.0033 sec.	.0015 sec.	.0013 sec.	.0034 sec.	.001 sec.
100	.003 sec.	.03 sec.	.13 sec.	3.4 sec.	$4 \cdot 10^{16}$ yr.
1,000	.033 sec.	.45 sec.	13 sec.	.96 hr.	
10,000	.33 sec.	6.1 sec.	22 min.	39 days	
100,000	3.3 sec.	1.3 min.	1.5 days	108 yr.	
Time allowed	Maximum solvable input size (approx.)				
1 second	30,000	2,000	280	67	20
1 minute	1,800,000	82,000	2,200	260	26

Table is adapted from *Programming Peals* by John Bently.