

Chapter 3

Growth of Functions

Algorithm Analysis

School of CSEE

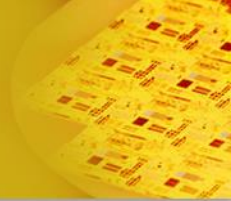
Growth of Function

- In this chapter we will study growth of function.
- 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.

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 | $f(n) = \theta(g(n))$ | $f(n) \approx c g(n)$ |
| BigOh | $f(n) = O(g(n))$ | $f(n) \leq c g(n)$ |
| Omega | $f(n) = \Omega(g(n))$ | $f(n) \geq c g(n)$ |
| Little Oh | $f(n) = o(g(n))$ | $f(n) < c g(n)$ |
| Little Omega | $f(n) = \omega(g(n))$ | $f(n) > c g(n)$ |

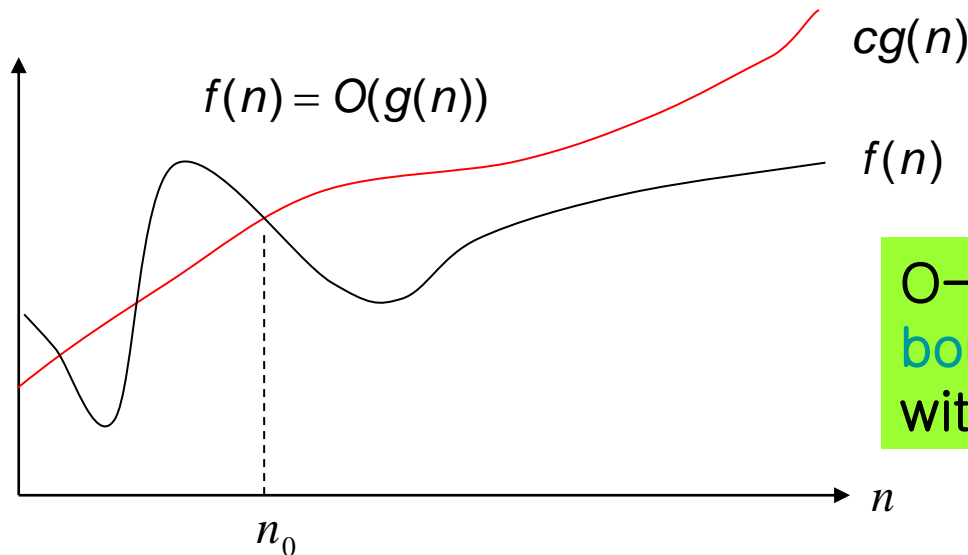
O-notation – upper bound

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

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

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

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



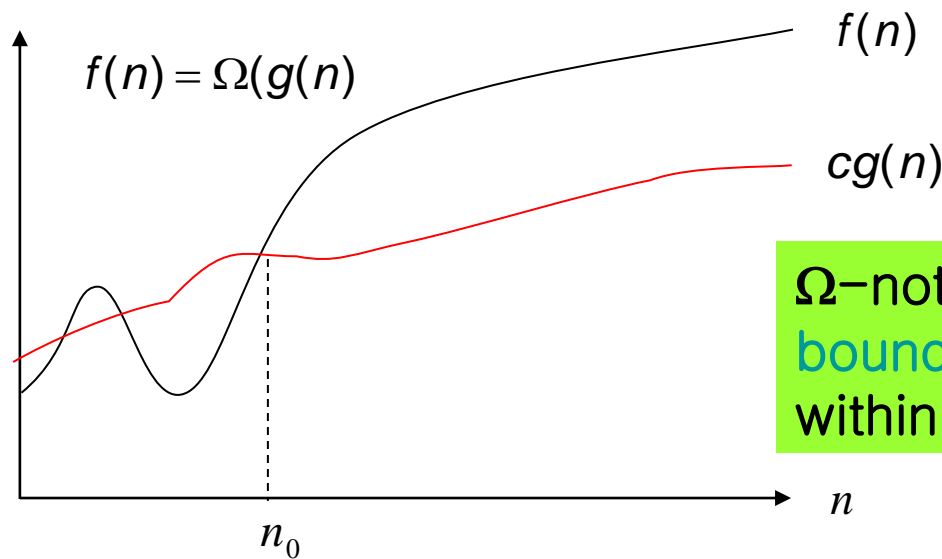
O-notation gives an upper bound on a function, to within a constant factor.

Ω -notation – lower bound

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

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

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



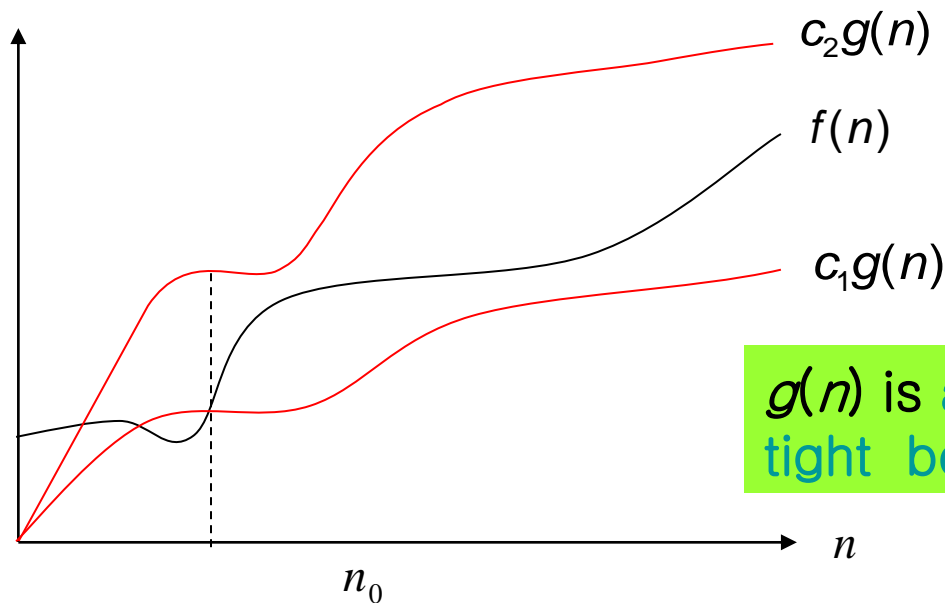
Ω -notation gives an **lower bound** on a function, to within a constant factor.

Θ -notation – tight (exact) bound

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

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

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 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

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

o-notation

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

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

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

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

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

$o(f)$ is usually called “little oh of f ”.

~~$$n^3 \in o(n^3)$$~~

ω -notation

- $\omega(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \text{ there exist a constant } n_0 > 0 \text{ such that } 0 \leq c g(n) < f(n) \text{ for all } n \geq n_0 \}$
 $f(n)$ is **asymptotically larger** than $g(n)$ if $f(n) = \omega(g(n))$

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

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

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 ”

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

Example

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

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

$$\Theta((8n^3 + n)^{1/2} + \log n)$$

$$= \Theta(2n^{3/2} + 6n \ln n)$$

$$= \Theta(n^{3/2}) \quad \leftarrow \text{simplest form}$$

$$= \Theta(n^{3/2} + 23n^{1/3})$$

Also specify O or θ

$$= \Theta\left(\frac{n^3 + n + 1}{4n^{3/2} + \ln n}\right)$$

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. $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$ (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*

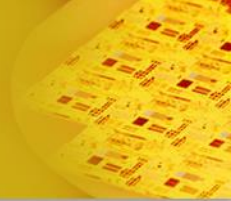
The time complexity of an *algorithm* is the largest time required on any input of size n .

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

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

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

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

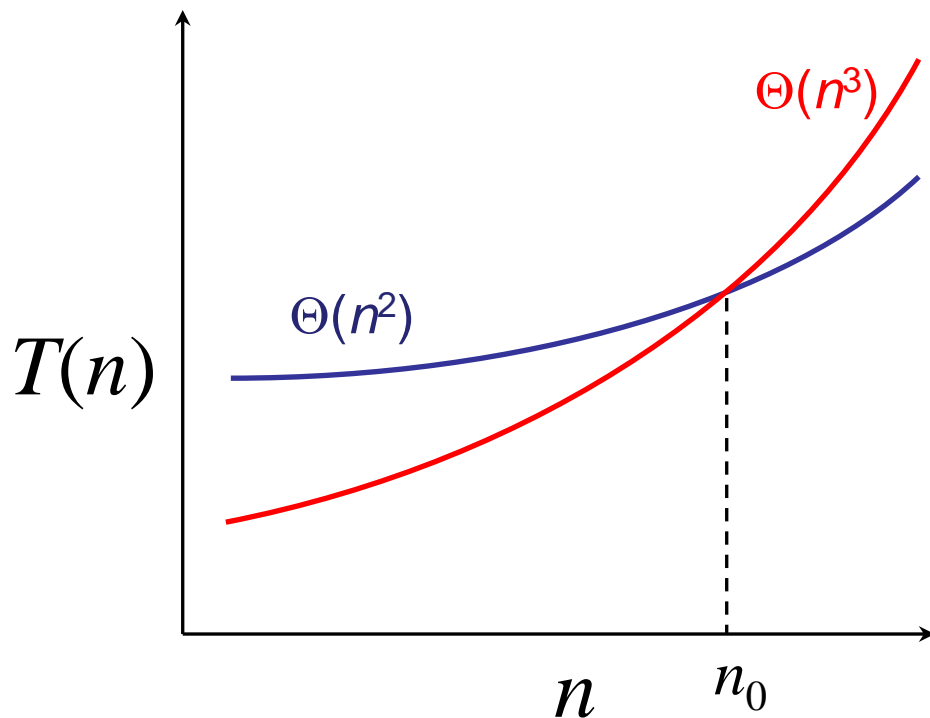
Time for several algorithms for a given problem

- ✓ 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.

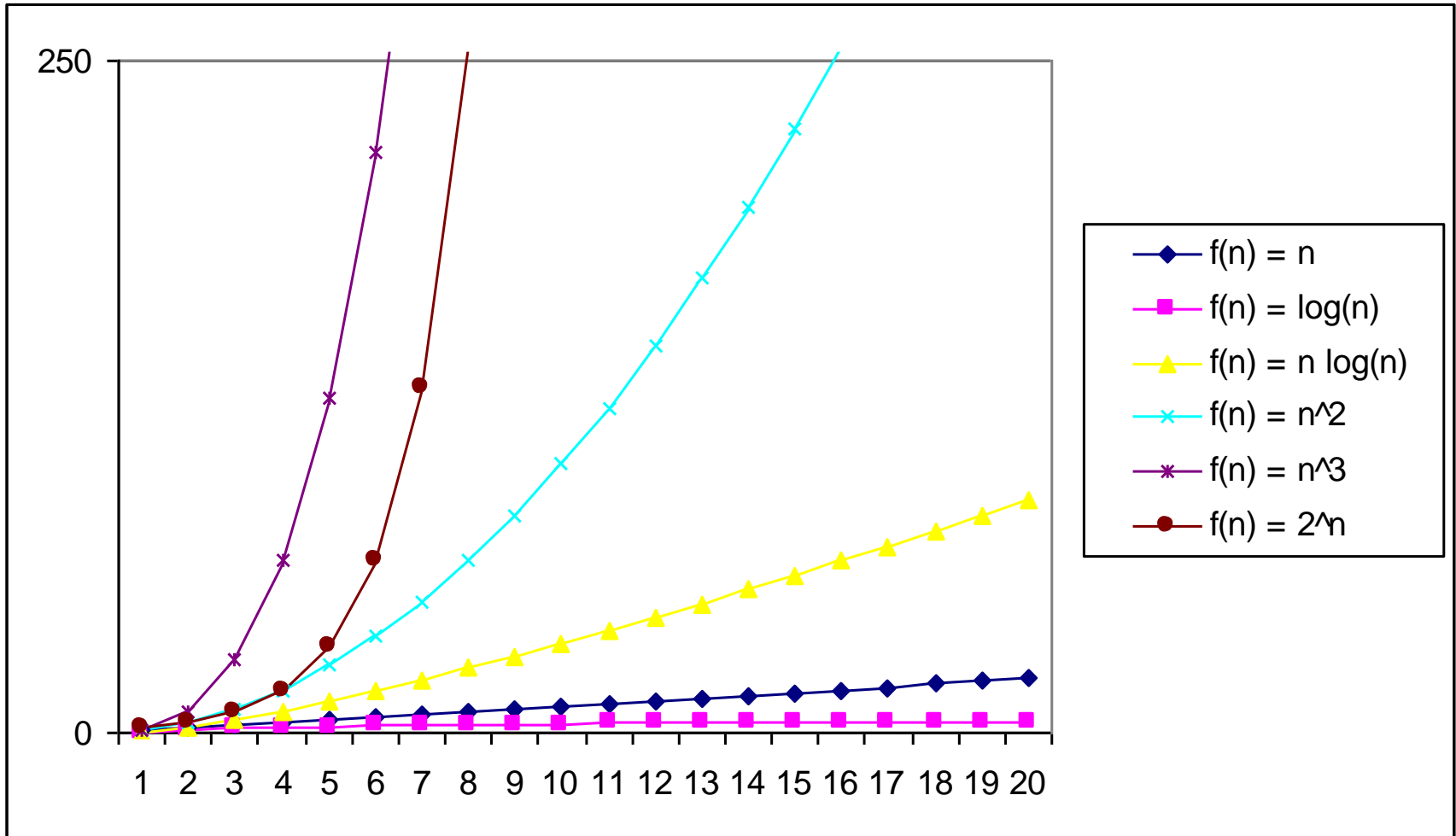
Asymptotic performance

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

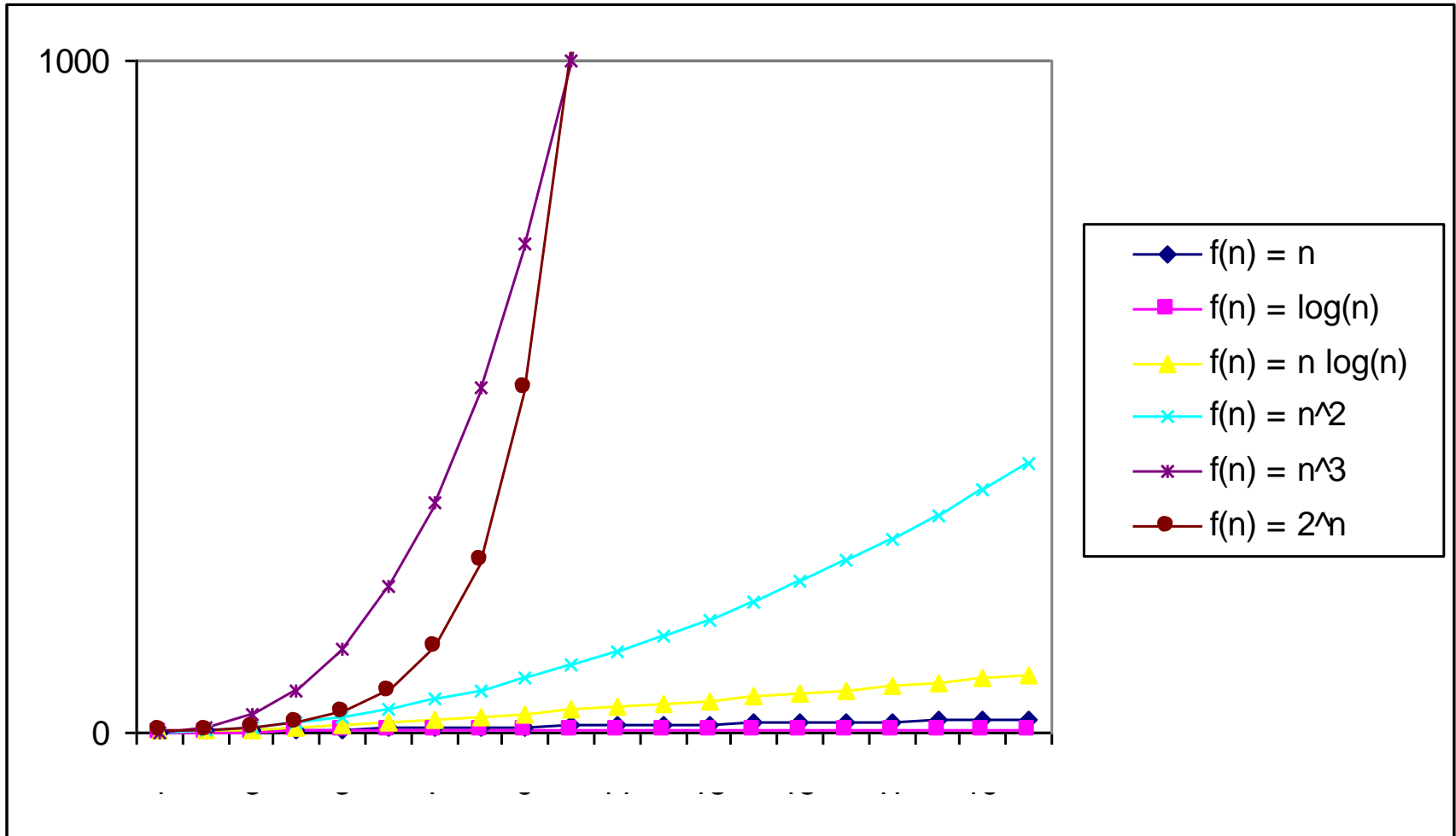


- Asymptotic analysis is a useful tool to help to structure our thinking toward better algorithm.
- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing

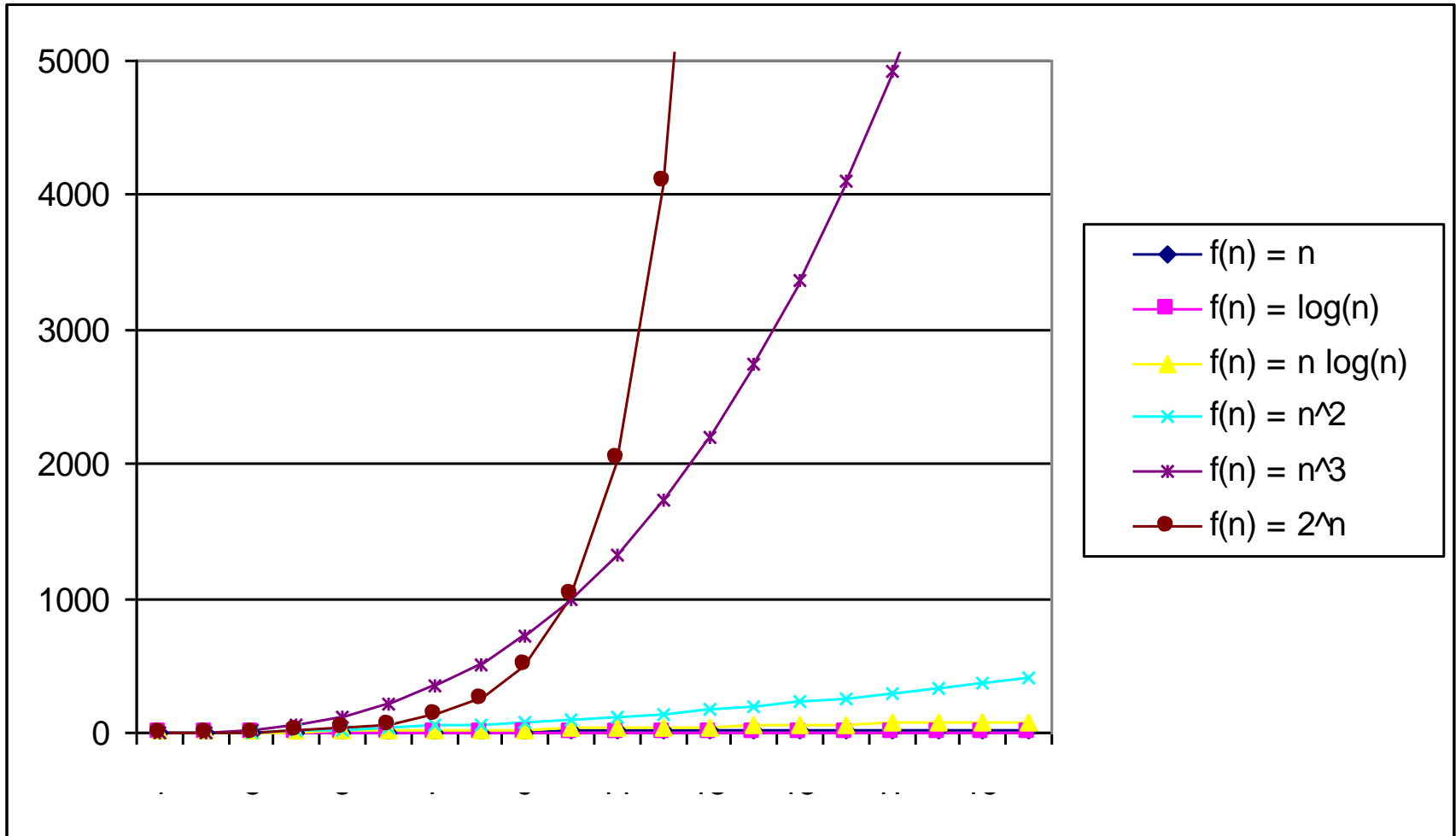
Practical Complexity



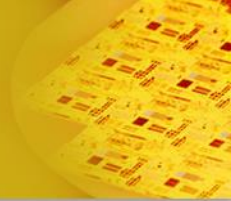
Practical Complexity



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:

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

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

Time complexity comparison

| n | Cray-1 Fortran^a 3n³ 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 Pearls* by John Bentley.