Asymptotic Growth Rate

Algorithm Theoretical Analysis

- Time for executing the basic operation
- Time for overhead instructions (e.g. Initialization)
- Time for the control instructions (e.g. incrementing an index to control a loop)
- These times are properties of an algorithm, not a problem.
- Compare every-case time complexities of two algos.
  - n for the 1st algorithm, n^2 for the 2^{nd} algorithm.
  - Time(Basic operation of 1^{st} algo.)=1000* Time(Basic operation of 2nd algo.)
  - Times to process an instance of size n: n*1000t (1^{st} algo.) and n^2t (2^{nd} algo.).
  - Which algorithm is more efficient? (1^{st} algorithm when n>1000)

Algorithm Theoretical Analysis

- Time complexity of n is more efficient than time complexity of n^2 for sufficiently large n.
- Compare every-case time complexities of two algos.
  - 100n for the 1st algorithm, $0.01n^2$ for the 2^{nd} algorithm.
  - 1^{st} algorithm is more efficient if n>10,000
  - If 1^{st} algorithm takes longer to process the basic operation, 1^{st} algorithm is more efficient if n> (some larger number than 10,000).
- Linear-time algo. (time complexity such as n and 100n)
- Quadratic-time algo. (time complexity such as n^2 and $0.1n^2$)

Introduction to Order

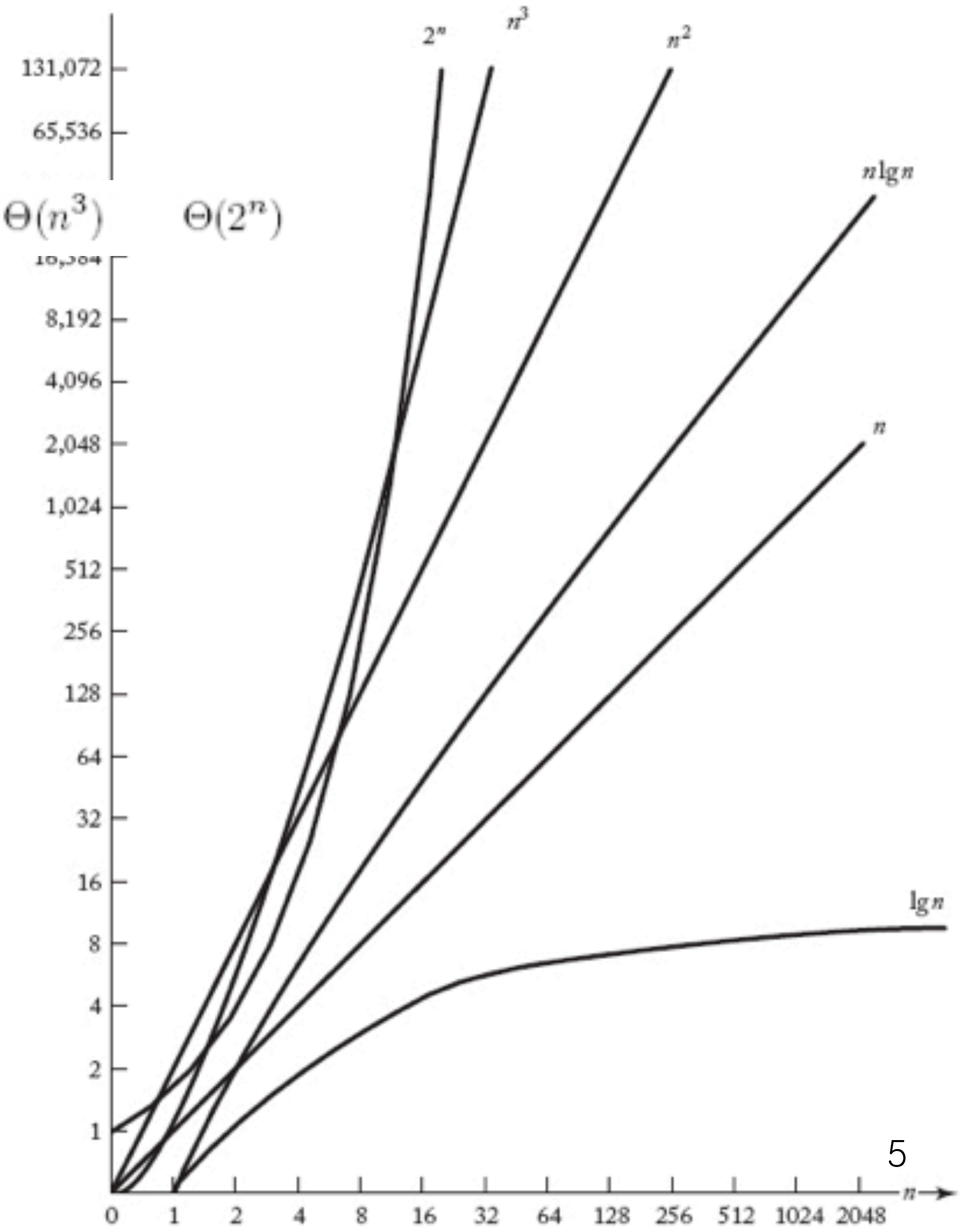
- Quadratic-time algo.
  - Pure quadratic: $5n^2$, $5n^2+100$
  - Complete quadratic: $0.1n^2+n+100$
  - we can classified both with pure quadratic function.
- The set that can be classified with pure quadratic functions is $\Theta(n^2)$.
- If a function is a member of the set $\Theta(n^2)$, we say the function is order of n^2.
- For example, $g(n)=5n^2+100n+60 \in \Theta(n^2)$, g(n) is order of n^2.
- For exchange sort, $T(n)=n(n-1)/2=n^2/2-n/2 \in \Theta(n^2)$

Introduction to Order

- Complexity categories.

$\Theta(\lg n)$ $\Theta(n)$ $\Theta(n \lg n)$ $\Theta(n^2)$ $\Theta(n^3)$ $\Theta(2^n)$

- The complexity category on the left will eventually lies beneath the right.
- More information in exact time complexity than its order only.
  - e.g. $100n$ and $0.01n^2$
  - if n <10,000 for all instances we implement algo. 2
  - Miss the info if only know the order $\Theta(n^2)$ and $\Theta(n)$.



Asymptotic Running Time

- The running time of an algorithm as input size approaches infinity is called the *asymptotic running time*
- We study different notations for asymptotic efficiency.
- In particular, we study tight bounds, upper bounds and lower bounds.

Outline

- Why do we need the different sets?
- Definition of the sets O (Oh), Ω (Omega) and Θ (Theta), o (oh), ω (omega)
- Classifying examples:
  - Using the original definition
  - Using limits

The functions

- Let $f(n)$ and $g(n)$ be *asymptotically nonnegative* functions whose domains are the set of natural numbers N={0,1,2,…}.
- A function $g(n)$ is *asymptotically nonnegative*, if $g(n) \geq 0$ for all $n \geq n_0$ where $n_0 \in$ N

Big Oh

- Big "oh" - asymptotic upper bound on the growth of an algorithm
- When do we use Big Oh?

1. To provide information on the maximum number of operations that an algorithm performs
   - Insertion sort is $O(n^2)$ in the worst case
     - This means that in the worst case it performs at most cn^2 operations where c is a positive constant
2. Theory of NP-completeness
   1. An algorithm is polynomial if it is $O(n^k)$ for some constant k
   2. P = NP if there is any polynomial time algorithm for any NP-complete problem

   Note: Theory of NP-completeness will be discussed much later in the semester

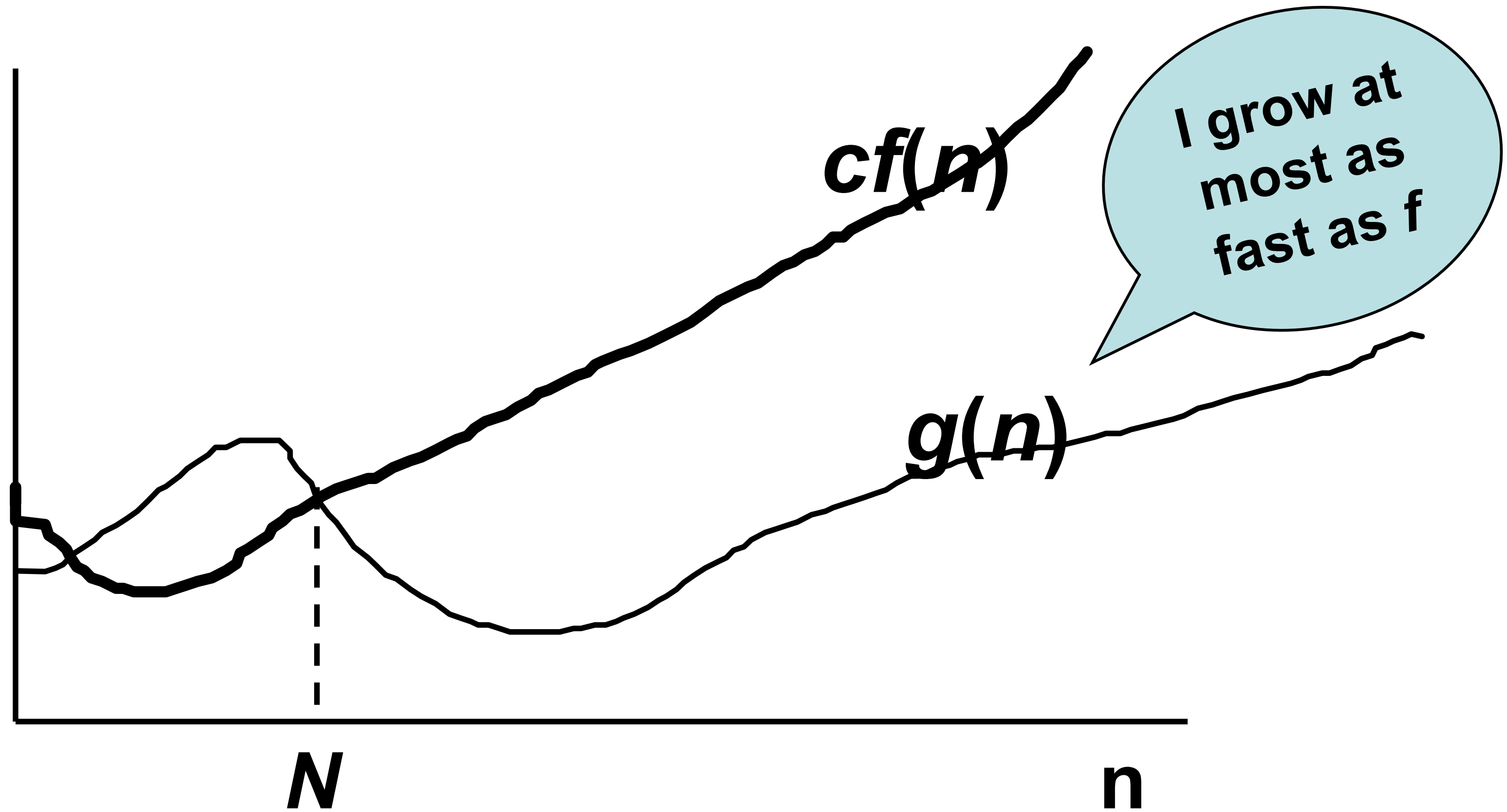
Definition of Big Oh

- $O(f(n))$ is the set of functions $g(n)$ such that: there exist positive constants c and N, for which

  $0 \leq g(n) \leq cf(n)$ for all $n \geq N$

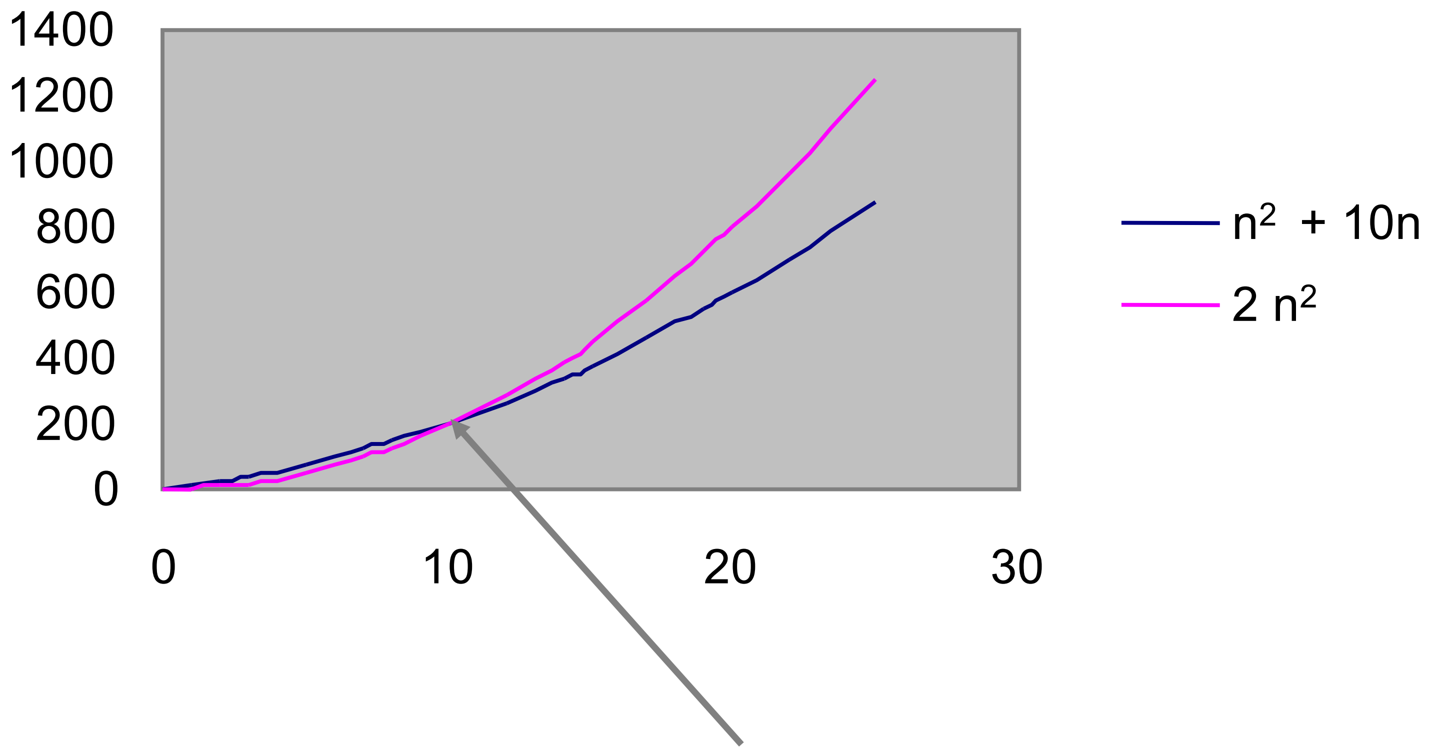
- $g(n) \in O(f(n))$: $f(n)$ is an *asymptotic upper bound* for $g(n)$

$g(n) \in O(f(n))$



$n^2 + 10\,n \in O(n^2)$ Why?

take c = 2
N = 10
n^2+10n <=2n^2 for all n>=10



Does $5n+2 \in O(n)$?

Proof: From the definition of Big Oh, there must exist $c > 0$ and integer $N > 0$ such that $0 \leq 5n+2 \leq cn$ for all $n \geq N$.

Dividing both sides of the inequality by $n > 0$ we get:

$$0 \leq 5+2/n \leq c.$$

- $2/n$ (> 0) becomes smaller as n increases
- For instance, let $N = 2$ and $c = 6$

There are many choices here for c and N.

Is $5n+2 \in O(n)$?

If we choose $N = 1$ then $5+2/n \leq 5+2/1= 7$. So any $c \geq 7$ works. Let's choose c = 7.

If we choose $c = 6$, then $0 \leq 5+2/n \leq 6$. So any $N \geq 2$ works. Choose $N = 2$.

In either case (we only need one!) , $c > 0$ and $N > 0$ such that $0 \leq 5n+2 \leq cn$ for all $n \geq N$. So the definition is satisfied and

$5n+2 \in O(n)$

Does $n^2 \in O(n)$? No.

We will prove by contradiction that the definition cannot be satisfied.

- Assume that $n^2 \in O(n)$. From the definition of Big Oh, there must exist $c > 0$ and integer $N > 0$ such that $0 \leq n^2 \leq cn$ for all $n \geq N$.
- Divide the inequality by $n > 0$ to get $0 \leq n \leq c$ for all $n \geq N$.
- $n \leq c$ cannot be true for any $n > \max\{c, N\}$. This contradicts the assumption. Thus, $n^2 \notin O(n)$.

Are they true? Why or why not?

- **$1{,}000{,}000\ n^2 \in O(n^2)$?**
- **True**

- **$(n - 1)n / 2 \in O(n^2)$?**
- **True**

- **$n / 2 \in O(n^2)$?**
- **True**

- **$\lg (n^2) \in O(\lg n)$?**
- **True**

- **$n^2 \in O(n)$?**
- **False**

Omega

Asymptotic lower bound on the growth of an algorithm or a problem

When do we use Omega?

1. To provide information on the minimum number of operations that an algorithm performs
   - Insertion sort is $\Omega(n)$ in the best case
     - This means that in the best case it performs at least cn operations where *c* is a positive constant
   - It is $\Omega(n^2)$ in the worst case
     - This means that in the worst case it performs at least cn^2 operations where *c* is a positive constant

Omega (cont.)

2. To provide information on a class of algorithms that solve a problem

- Sorting algorithms based on comparisons of keys are Ω(nlgn) in the worst case
  - This means that all sort algorithms based only on comparisons of keys have to do at least $cn\lg n$ operations
- Any algorithm based only on comparisons of keys to find the maximum of n elements is Ω(n) in every case
  - This means that all algorithms only based on key comparisons to find maximum have to do at least cn operations

Supplementary topic: Why $\Omega(n\lg n)$ for sorting?

- n numbers to sort with no further information or assumption about them ➔ there are n! permutations
- One comparison has only two outcomes
- So, lg(n!) comparisons are required in the worst case
- n! is approximately equal to $(n/e)^n$

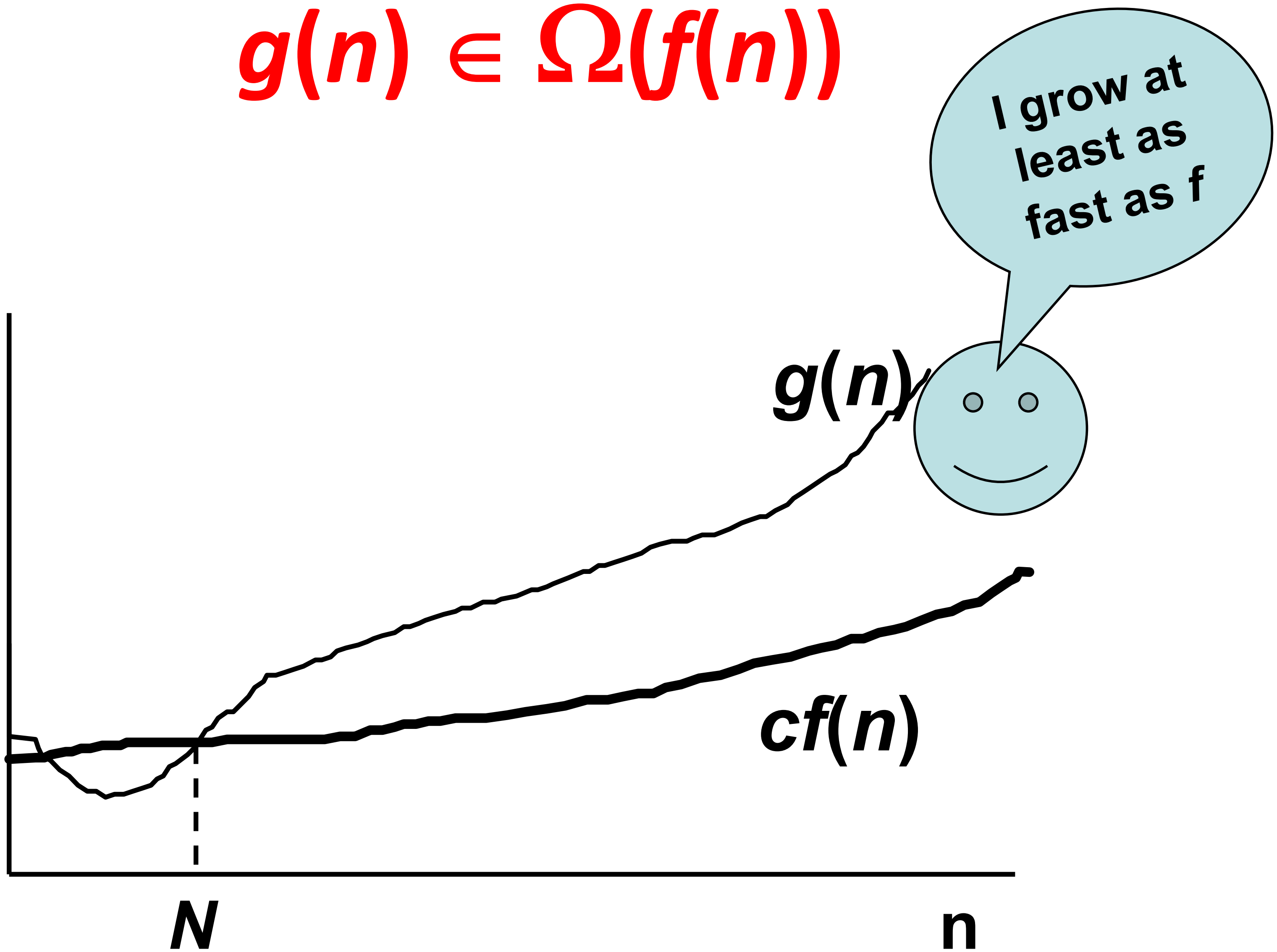
Definition of the set Omega

- $\Omega(f(n))$ is the set of functions $g(n)$ such that there exist positive constants c and N for which

$$0 \le cf(n) \le g(n) \quad \text{for all} \quad n \ge N$$

- $g(n) \in \Omega(f(n))$: $f(n)$ is an *asymptotic lower bound* for $g(n)$

$$g(n) \in \Omega(f(n))$$



Is $5n$-20$\in \Omega$ (n)?

Proof: From the definition of Omega, there must exist c > 0 and integer N>0 such that $0 \leq cn \leq 5n\text{-}20$ for all $n \geq N$

Dividing the inequality by $n > 0$ we get: $0 \leq c \leq 5\text{-}20/n$ for all $n \geq N$.

$20/n \leq 20$, and 20/n becomes smaller as n grows.

There are many choices here for c and N.

Since $c > 0$, $5 - 20/n > 0$ and $N > 4$.

If we choose c=1, then $5 - 20/n \geq 1$ and $N \geq 5$ Choose $N = 5$.

If we choose c=4, then $5 - 20/n \geq 4$ and $N \geq 20$. Choose $N = 20$.

In either case (we only need one!) we have c>o and N>0 such that $0 \leq cn \leq 5n\text{-}20$ for all $n \geq N$. So $5n\text{-}20 \in \Omega(n)$.

Are they true?

- **1,000,000 $n^2 \in \Omega(n^2)$ why /why not?**
  - **true**
- **$(n - 1)n / 2 \in \Omega(n^2)$ why /why not?**
  - **true**

- **$n / 2 \in \Omega(n^2)$ why /why not?**
  - **(false)**
- **$\lg(n^2) \in \Omega(\lg n)$ why /why not?**
  - **(true)**
- **$n^2 \in \Omega(n)$ why /why not?**
  - **(true)**

Theta

- Asymptotic tight bound on the growth rate of an algorithm
  - Insertion sort is $\Theta(n^2)$ in the worst and average cases
    - This means that in the worst case and average case insertion sort performs cn^2 operations
  - Binary search is $\Theta(\lg n)$ in the worst and average cases
    - The means that in the worst case and average case binary search performs $c\lg n$ operations

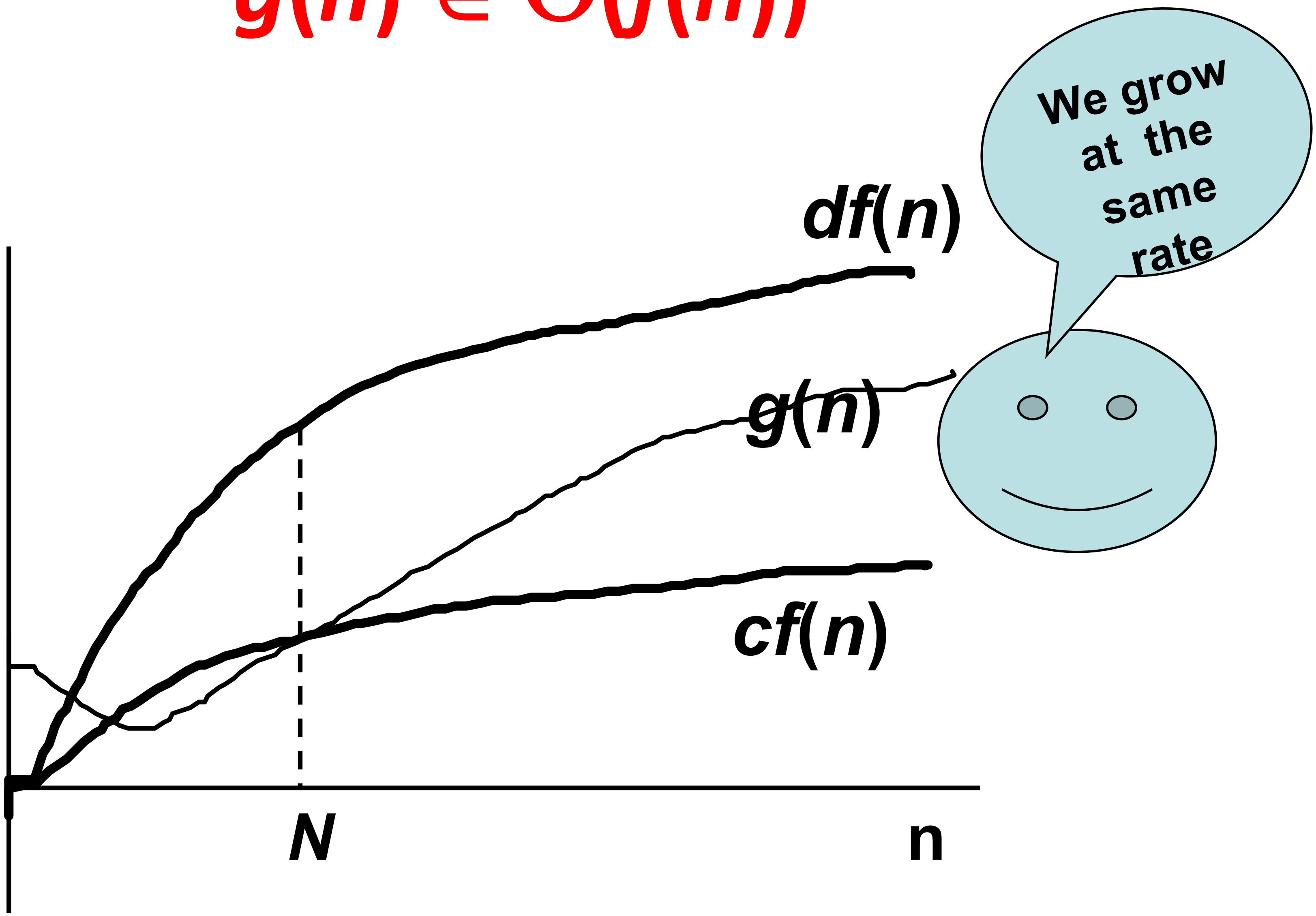
Definition of the set Theta

- $\Theta(f(n))$ is the set of functions $g(n)$ such that there exist positive constants c, d, and N for which

  $0 \le cf(n) \le g(n) \le df(n)$ for all $n \ge N$

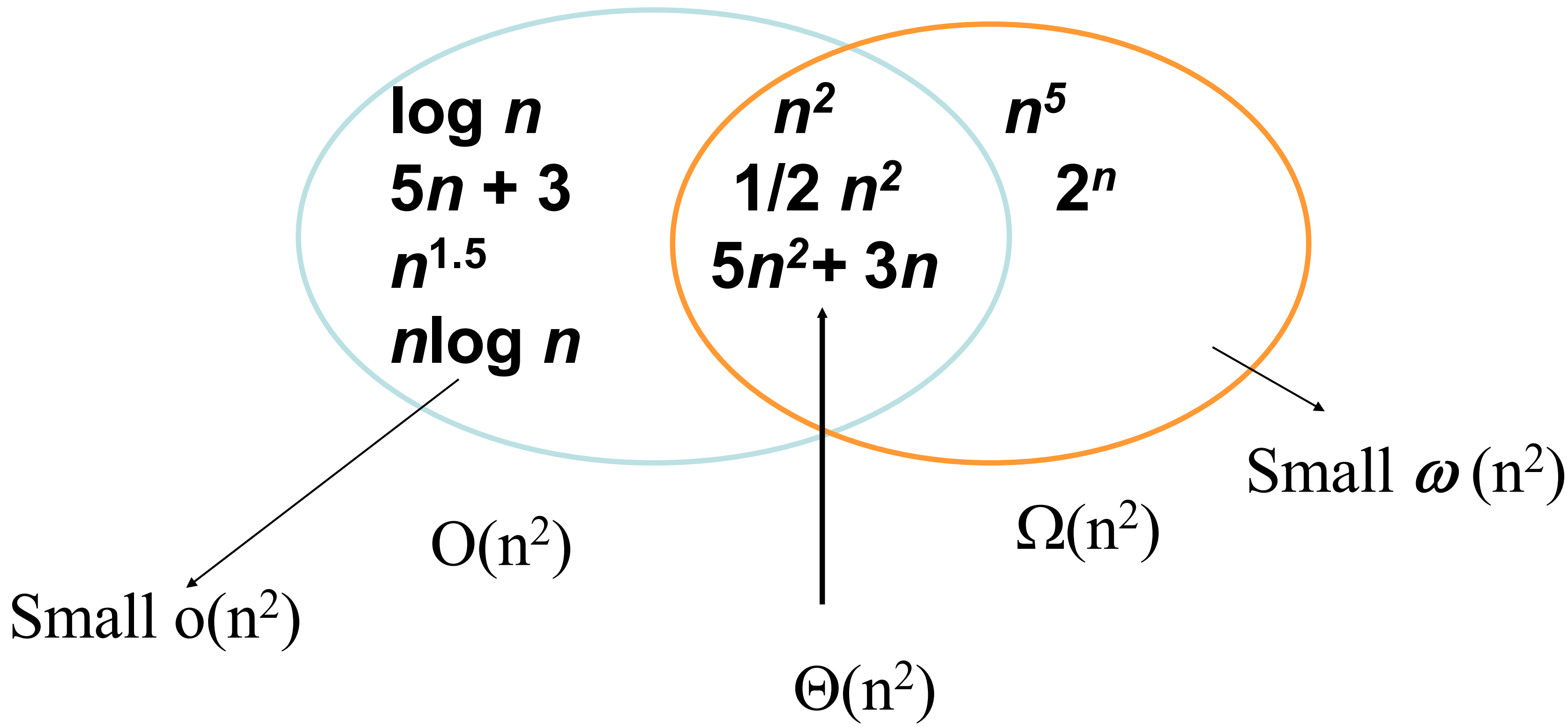
- $g(n) \in \Theta(f(n))$: $f(n)$ is an asymptotic tight bound for $g(n)$

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



Another Definition of Theta

$$\Theta(f(n)) = O(f(n)) \bigcap \Omega(f(n))$$



$$\text{Does } \frac{1}{2}n^2 - 3n = \Theta(n^2)?$$

- We use the last definition and show:

1.

$$\frac{1}{2}n^2 - 3n = O(n^2)$$

2.

$$\frac{1}{2}n^2 - 3n = \Omega(n^2)$$

$$\text{Does } \frac{1}{2}n^2 - 3n = O(n^2)?$$

From the definition there must exist $c > 0$, and $N > 0$ such that

$$0 \le \frac{1}{2}n^2 - 3n \le cn^2 \text{ for all } n \ge N.$$

Dividing the inequality by $n^2 > 0$ we get:

$$0 \le \frac{1}{2} - \frac{3}{n} \le c \text{ for all } n \ge N.$$

Clearly any $c \ge 1/2$ can be chosen

Choose $c = 1/2$.

$$0 \le \frac{1}{2} - \frac{3}{n} \le \frac{1}{2} \text{ for all } N \ge 6. \text{ Choose } N = 6$$

Does $\frac{1}{2}n^2 - 3n = \Omega(n^2)$?

There must exist $c > 0$ and $N > 0$ such that

$0 \le cn^2 \le \frac{1}{2}n^2 - 3n$ for all $n \ge N$

Dividing by $n^2 > 0$ we get

$$0 \le c \le \frac{1}{2} - \frac{3}{n}.$$

Since $c > 0$, $0 < \frac{1}{2} - \frac{3}{N}$ and $N > 6$.

Since $3/n > 0$ for finite n, $c < 1/2$. Choose $c = 1/4$.

$\frac{1}{4} \le \frac{1}{2} - \frac{3}{n}$ for all $n \ge 12$.

So $c = 1/4$ and $N = 12$.

More Θ

- **1,000,000 $n^2 \in \Theta(n^2)$ why /why not?**
  - **True**
- **$(n - 1)n / 2 \in \Theta(n^2)$ why /why not?**
  - **True**
- **$n / 2 \in \Theta(n^2)$ why /why not?**
  - **False**
- **lg $(\mathrm{n}^2) \in \Theta$ (lg n) why /why not?**
  - **True**
- **$n^2 \in \Theta(n)$ why /why not?**
  - **False**

small o

- $o(f(n))$ is the set of functions $g(n)$ which satisfy the following condition:
- *g(n)* is *o(f(n))*: For *every* positive real constant c, there exists a positive integer *N*, for which

$$g(n) \le cf(n) \text{ for all } n \ge N$$

small o

- Little "oh" - used to denote an upper bound that is not asymptotically tight.
  - n is in $o(n^3)$
  - n is not in $o(n)$

small omega

- $\omega(f(n))$ is the set of functions $g(n)$ which satisfy the following condition:
- $g(n)$ is $\omega(f(n))$: For *every* positive real constant c, there exists a positive integer N, for which

$$g(n) \geq cf(n) \text{ for all } n \geq N$$

small omega and small o

- $g(n) \in \omega(f(n))$

  if and only if

  $f(n) \in o(g(n))$

Limits can be used to determine Order

$$\text{if } \lim_{n\to\infty} f(n)/g(n) = \begin{cases} c \text{ then } f(n) = \Theta(g(n)) \text{ if } c > 0 \\ 0 \text{ then } f(n) = o(g(n)) \\ \infty \text{ then } f(n) = \omega(g(n)) \end{cases}$$

- We can use this method if the limit exists

Example using limits

$$5n^3 + 3n \in \omega(n^2)$$

$$\lim_{n \to \infty} \frac{5n^3 + 3n}{n^2} = \lim_{n \to \infty} \frac{5n^3}{n^2} + \lim_{n \to \infty} \frac{3n}{n^2} = \infty$$

L'Hopital's Rule

If $f(x)$ and $g(x)$ are both differentiable with derivatives $f'(x)$ and $g'(x)$, respectively, and if

$$\lim_{x \to \infty} g(x) = \lim_{x \to \infty} f(x) = \infty \text{ then}$$

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

whenever the limit on the right exists

Example using limits

$$10n^3 - 3n \in \Theta(n^3) \ \text{ since,}$$

$$\lim_{n\to\infty} \frac{10n^3 - 3n}{n^3} = \lim_{n\to\infty} \frac{10n^3}{n^3} - \lim_{n\to\infty} \frac{3n}{n^3} = 10$$

$$n \log_e n \in o(n^2) \text{ since,}$$

$$\lim_{n\to\infty} \frac{n \log_e n}{n^2} = \lim_{n\to\infty} \frac{\log_e n}{n} = ? \quad \text{Use L'Hopital's Rule:}$$

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

$$y = \log_a x$$

$$y^{(k)} = \frac{(-1)^{k-1}(k-1)!}{x^k \ln a}$$ (kth order differentiation of y)

Example using limit

$$\lg n \in o(n)$$

$$\lg n = \frac{\ln n}{\ln 2} \text{ and } (\lg n)' = \left(\frac{\ln n}{\ln 2}\right)' = \frac{1}{n \ln 2}$$

$$\lim_{n\to\infty} \frac{\lg n}{n} = \lim_{n\to\infty} \frac{(\lg n)'}{n'} = \lim_{n\to\infty} \frac{1}{n \ln 2} = 0$$

Example using limits

$n^k \in o(2^n)$ where k is a positive integer

$$2^n = e^{n \ln 2}$$

$$\left(2^n\right)' = \left(e^{n \ln 2}\right)' = \ln 2 e^{n \ln 2} = \ln 2 (2^n)$$

$$Note: (e^x)' = x'(e^x)$$

$$\lim_{n \to \infty} \frac{n^k}{2^n} = \lim_{n \to \infty} \frac{kn^{k-1}}{2^n \ln 2} = \lim_{n \to \infty} \frac{k(k-1)n^{k-2}}{2^n \ln^2 2} = \ldots =$$

$$= \lim_{n \to \infty} \frac{k!}{2^n \ln^k 2} = 0$$

Analogy between asymptotic comparison of functions and comparison of real numbers.

$$f(n) = O(g(n)) \quad \approx \quad a \le b$$

$$f(n) = \Omega(g(n)) \quad \approx \quad a \ge b$$

$$f(n) = \Theta(g(n)) \quad \approx \quad a = b$$

$$f(n) = o(g(n)) \quad \approx \quad a < b$$

$$f(n) = \omega(g(n)) \quad \approx \quad a > b$$

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

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

Asymptotic Growth Rate
Part II

Order of Algorithm

- Property
  - Complexity Categories:

$\theta(\lg n)$ $\theta(n)$ $\theta(n \lg n)$ $\theta(n^2)$ $\theta(n^j)$ $\theta(n^k)$ $\theta(a^n)$ $\theta(b^n)$ $\theta(n!)$

Where $k>j>2$ and $b>a>1$. If a complexity function $g(n)$ is in a category that is to the left of the category containing $f(n)$, then $g(n) \in o(f(n))$

Comparing ln *n* with n^k ($k > 0$)

- Using limits we get:

$$\lim_{n\to\infty}\frac{\ln n}{n^k}=\lim_{n\to\infty}\frac{1}{kn^k}=0$$

- So $\ln n = o(n^k)$ for any $k > 0$
- When the exponent k is very small, we need to look at very large values of n to see that $n^k > \ln n$

Values for $\log_{10} n$ *and* $n^{0.01}$

		0.01	
	n	log n	$n^{.01}$
	1	0	1
	1.00E+10	10	1.258925
	1.00E+100	100	10
	1.00E+200	200	100
	1.00E+230	230	199.5262
	1.00E+240	240	251.1886

Values for $\log_{10} n$ *and* $n^{0.001}$

n	$\log n$	$n^{.001}$
1	0	1
1.00E+10	10	1.023293
1.00E+100	100	1.258925
1E+1000	1000	10
1E+2000	2000	100
1E+3000	3000	1000
1E+4000	4000	10000

Lower-order terms and constants

- Lower order terms of a function do not matter since lower-order terms are dominated by the higher order term
- Constants (multiplied by highest order term) do not matter, since they do not affect the asymptotic growth rate
- All logarithms with base b >1 belong to $\Theta(\lg n)$, since

$$\log_b n = \frac{\lg n}{\lg b} = c \lg n \text{ where } c \text{ is a constant}$$

General Rules

- We say a function $f(n)$ is polynomially bounded if $f(n) = O(n^k)$ for some positive constant k
- We say a function $f(n)$ is polylogarithmic bounded if $f(n) = O(\lg^k n)$ for some positive constant k
- Exponential functions
  - grow faster than positive polynomial functions
- Polynomial functions
  - grow faster than polylogarithmic functions

More properties

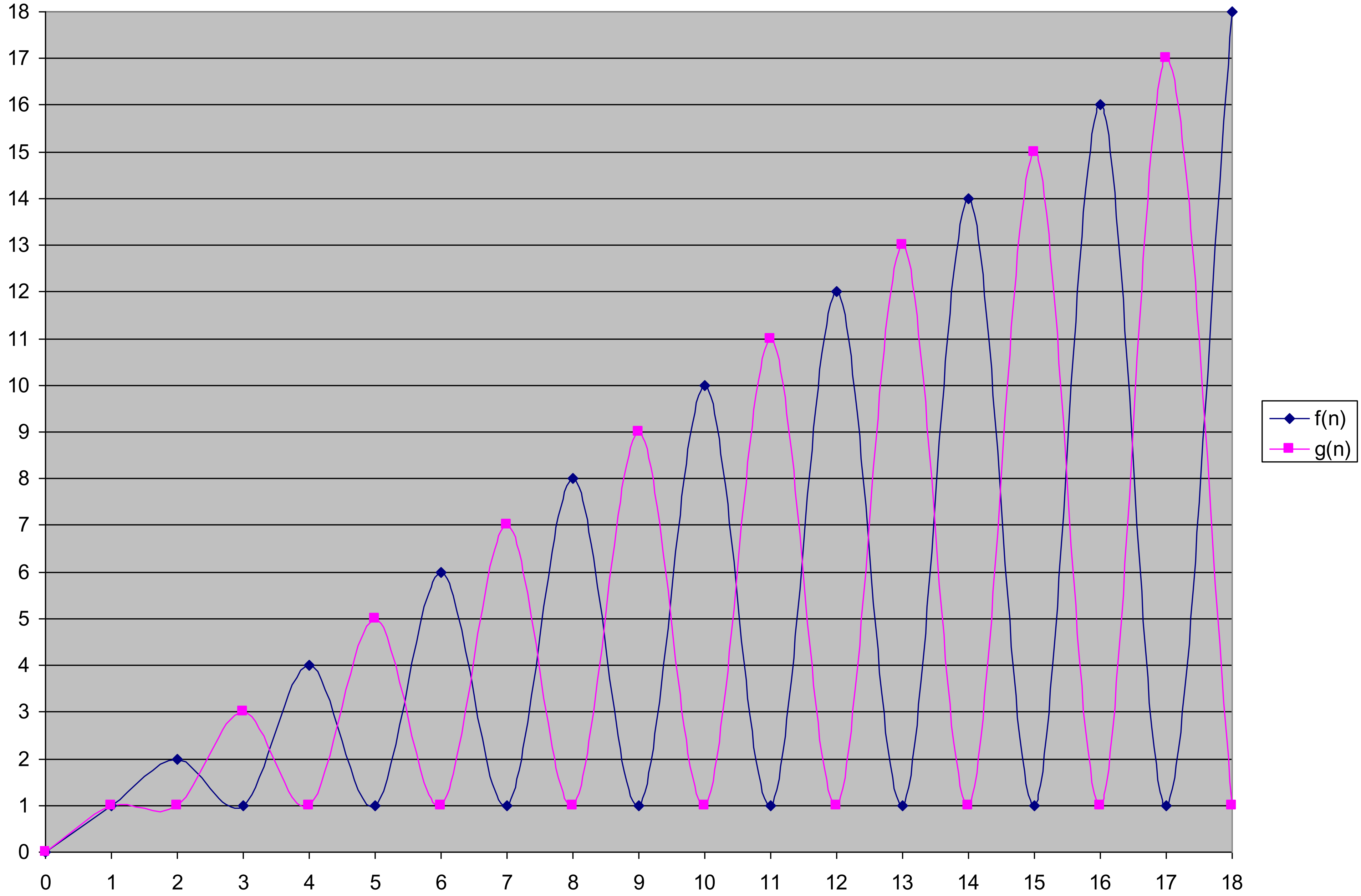
- The following slides show
  - An example in which a pair of functions are not comparable in terms of asymptotic notation
  - How the asymptotic notation can be used in equations
  - Theta, Big Oh, and Omega define a transitive and reflexive order.
  - Theta also satisfies symmetry, while Big Oh and Omega satisfy transpose symmetry

An example

The following functions are not asymptotically comparable:

$$f(n) = \begin{cases} n \text{ for even } n \\ 1 \ \text{ for odd } n \end{cases} \qquad g(n) = \begin{cases} 1 \text{ for even } n \\ n \ \text{ for odd } n \end{cases}$$

$$f(n) \notin O(g(n)), \text{and } f(n) \notin \Omega(g(n)),$$



18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
f(n)
g(n)

Transitivity:

If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ then $f(n) = \Theta(h(n))$.

If $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.

If $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ then $f(n) = \Omega(h(n))$.

If $f(n) = o(g(n))$ and $g(n) = o(h(n))$ then $f(n) = o(h(n))$.

If $f(n) = \omega(g(n))$ and $g(n) = \omega(h(n))$ then $f(n) = \omega(h(n))$

Reflexivity:

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

$$f(n) = O(f(n)).$$

$$f(n) = \Omega(f(n)).$$

"o" is not reflexive

"ω" is not reflexive

Symmetry and Transpose symmetry

- Symmetry:
  $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$

- Transpose symmetry:
  $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$
  $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$

Relate o to Other Asymptotic Notation

Theorem 1

If $g(n) \in o(f(n))$, then

$$g(n) \in O(f(n)) - \Omega(f(n)).$$

That is, $g(n)$ is in $O(f(n))$ but is not in $\Omega(f(n))$.

Proof: Because $g(n) \in o(f(n))$, for every positive real constant c there exists an N such that, for all $n \geq N$,

$$g(n) \leq c \times f(n),$$

which means that the bound certainly holds for some c. Therefore,

$$g(n) \in O(f(n)).$$

Relate o to Other Asymptotic Notation

Theorem 1

If $g(n) \in o(f(n))$, then

$$g(n) \in O(f(n)) - \Omega(f(n)).$$

That is, $g(n)$ is in $O(f(n))$ but is not in $\Omega(f(n))$.

Proof (continued): We will show that $g(n)$ is not in $\Omega(f(n))$ using proof by contradiction. If $g(n) \in \Omega(f(n))$, then there exists some real constant $c > 0$ and some N_1 such that, for all $n \geq N_1$,

$$g(n) \geq c \times f(n).$$

But, because $g(n) \in o(f(n))$, there exists some N_2 such that, for all $n \geq N_2$,

$$g(n) \leq \frac{c}{2} \times f(n).$$

Both inequalities would have to hold for all $n \geq max(N_1, N_2)$. This contradiction proves that $g(n)$ cannot be in $\Omega(f(n))$.

Is $O(g(n)) = \Theta(g(n)) \cup o(g(n))$?

No. We prove it by a counter example.

Consider the following functions:

$$g(n) = n$$

and

$$f(n) = \begin{cases} n & \text{if } n \text{ is odd} \\ 1 & \text{if } n \text{ is even} \end{cases}$$

Conclusion:

$f(n) \in O(n)$ but $f(n) \notin \Theta(n)$ and $f(n) \notin \mathbf{o}(n)$

Therefore, $O(g(n)) \neq \Theta(g(n)) \cup \mathbf{o}(g(n))$