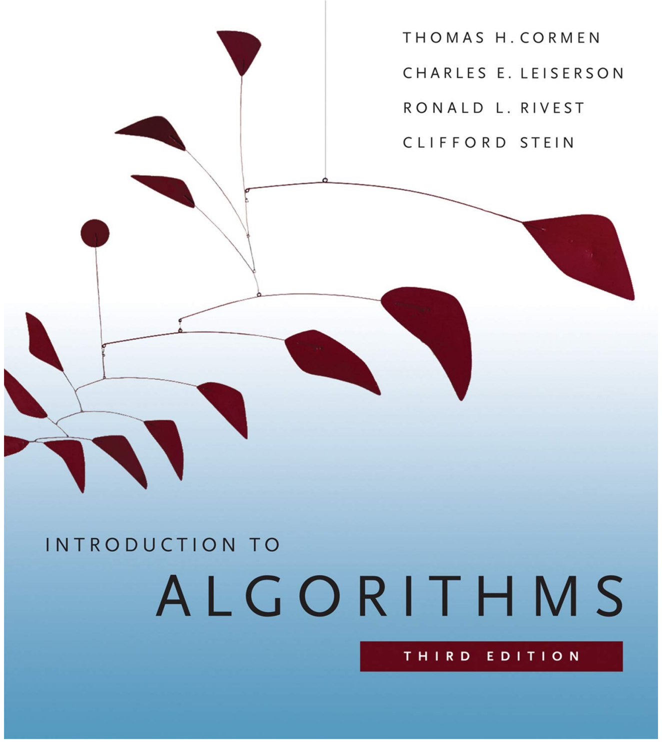
Algorithms



Chap 2: Asymptotic Notations

Outline

- Review of last lecture
- Order of growth
- Asymptotic notations
  - Big O, big Ω, Θ

How to express algorithms?

Increasing precision

English

Pseudocode

Real programming languages

Ease of expression

Describe the *ideas* of an algorithm in English.
Use pseudocode to clarify sufficiently tricky details of the algorithm.

Efficiency

- Correctness alone is not sufficient
  - Brute-force algorithms exist for most problems
  - E.g. use permutation to sort
  - Problem: too slow!
- How to measure efficiency?
  - Accurate running time is not a good measure
  - It depends on input, computer, and implementation, etc.

Machine-independent

- A generic uniprocessor random-access machine (RAM) model
  - No concurrent operations
  - Each simple operation (e.g. +, -, =, *, if, for) takes 1 step.
    - Loops and subroutine calls are *not* simple operations.
  - All memory equally expensive to access
    - Constant word size
    - Unless we are explicitly manipulating bits

Analysis of insertion Sort

```
InsertionSort(A, n) {
  for j = 2 to n {
      key = A[j]
      i = j - 1;
      while (i > 0) and (A[i] > key) {
            A[i+1] = A[i]
            i = i - 1
      }
      A[i+1] = key
  }
}
```

How many times will this line execute?

Analysis of insertion Sort

```
InsertionSort(A, n) {
  for j = 2 to n {
     key = A[j]
     i = j - 1;
     while (i > 0) and (A[i] > key) {
          A[i+1] = A[i]
          i = i - 1
     }
     A[i+1] = key
  }
}
```

How many times will this line execute?

Analysis of insertion Sort

Statement	cost	time
`InsertionSort(A, n) {`		
`for j = 2 to n {`	c_1	n
`key = A[j]`	c_2	(n-1)
`i = j - 1;`	c_3	(n-1)
`while (i > 0) and (A[i] > key) {`	c_4	S
`A[i+1] = A[i]`	c_5	(S-(n-1))
`i = i - 1`	c_6	(S-(n-1))
`}`	0	
`A[i+1] = key`	c_7	(n-1)
`}`	0	
`}`		

$S = t_2 + t_3 + \ldots + t_n$ where t_j is number of while expression evaluations for the j^{th} for loop iteration

Analyzing Insertion Sort

- $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 S + c_5(S - (n-1)) + c_6(S - (n-1)) + c_7(n-1)$
  $= c_8 S + c_9 n + c_{10}$
- What can S be?
  - Best case -- inner loop body never executed
    - $t_j = 1 \Rightarrow S = n - 1$
    - $T(n) = an + b$ is a linear function

    $\Theta(n)$
  - Worst case -- inner loop body executed for all previous elements
    - $t_j = j \Rightarrow S = 2 + 3 + \dots + n = n(n+1)/2 - 1$
    - $T(n) = an^2 + bn + c$ is a quadratic function

    $\Theta(n^2)$
  - Average case
    - Can assume that on average, we have to insert A[j] into the middle of A[1..j-1], so $t_j = j/2$
    - $S \approx n(n+1)/4$
    - T(n) is still a quadratic function

    $\Theta(n^2)$

Analysis of insertion Sort

Statement	cost	time
`InsertionSort(A, n) {`		
`for j = 2 to n {`	c_1	n
`key = A[j]`	c_2	(n-1)
`i = j - 1;`	c_3	(n-1)
`while (i > 0) and (A[i] > key) {`	c_4	S
`A[i+1] = A[i]`	c_5	(S-(n-1))
`i = i - 1`	c_6	(S-(n-1))
`}`	0	
`A[i+1] = key`	c_7	(n-1)
`}`	0	
`}`		

What are the basic operations (most executed lines)?

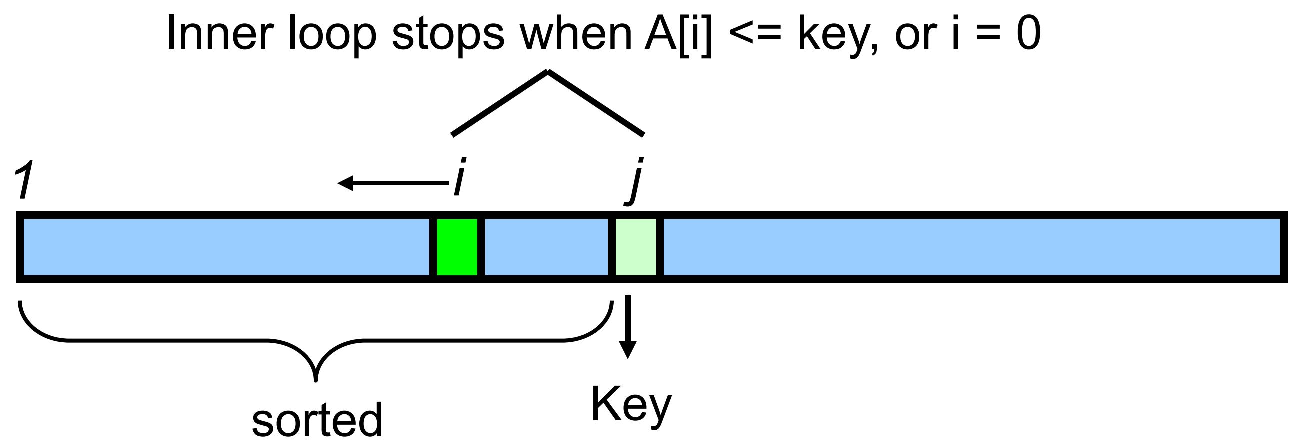
Analysis of insertion Sort

Statement	cost	time
`InsertionSort(A, n) {`		
`for j = 2 to n {`	c_1	n
`key = A[j]`	c_2	(n-1)
`i = j - 1;`	c_3	(n-1)
`while (i > 0) and (A[i] > key) {`	c_4	S
`A[i+1] = A[i]`	c_5	(S-(n-1))
`i = i - 1`	c_6	(S-(n-1))
`}`	0	
`A[i+1] = key`	c_7	(n-1)
`}`	0	
`}`		

Analysis of insertion Sort

Statement	cost	time
`InsertionSort(A, n) {`		
`for j = 2 to n {`	c_1	n
`key = A[j]`	c_2	(n-1)
`i = j - 1;`	c_3	(n-1)
`while (i > 0) and (A[i] > key) {`	c_4	S
`A[i+1] = A[i]`	c_5	(S-(n-1))
`i = i - 1`	c_6	(S-(n-1))
`}`	0	
`A[i+1] = key`	c_7	(n-1)
`}`	0	
`}`		

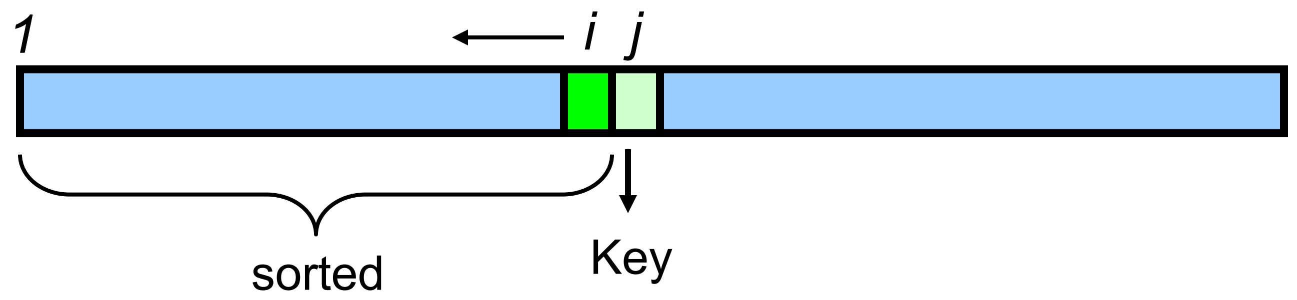
What can S be?



- $S = \sum_{j=1..n} t_j$
- Best case:
- Worst case:
- Average case:

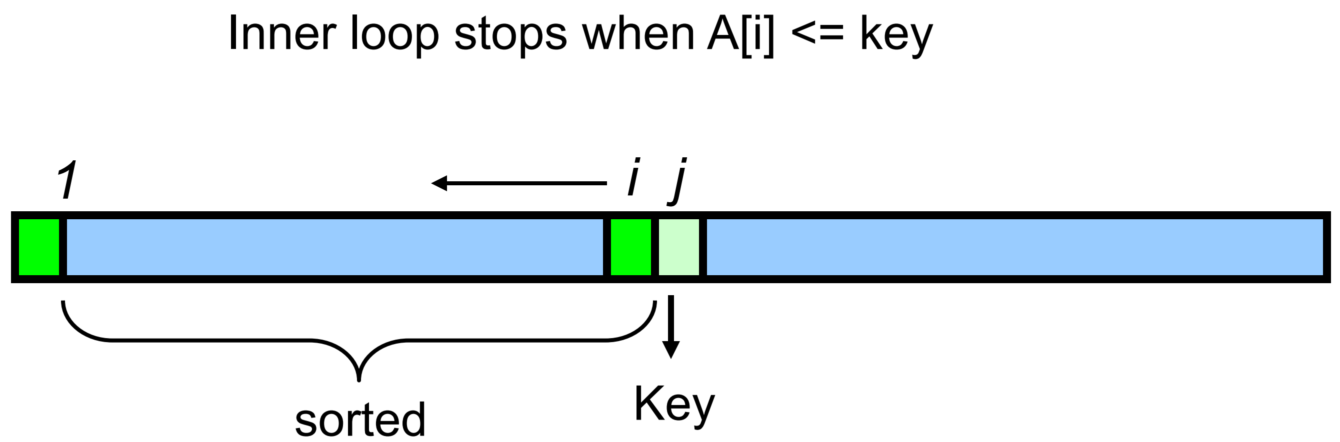
Best case

Inner loop stops when A[i] <= key, or i = 0



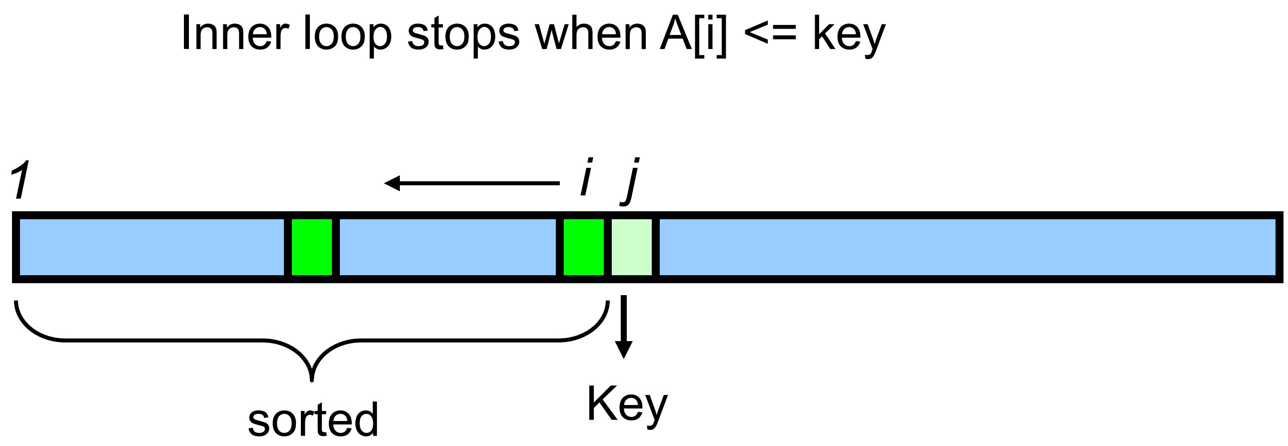
- Array already sorted
- $S = \sum_{j=1..n} t_j$
- $t_j = 1$ for all j
- $S = n.$ $T(n) = \Theta(n)$

Worst case



- Array originally in reverse order sorted
- $S = \sum_{j=1..n} t_j$
- $t_j = j$
- $S = \sum_{j=1..n} j = 1 + 2 + 3 + \ldots + n = n\,(n+1) / 2 = \Theta\,(n^2)$

Average case



- Array in random order
- $S = \sum_{j=1..n} t_j$
- $t_j = j / 2$ on average
- $S = \sum_{j=1..n} j/2 = ½ \sum_{j=1..n} j = n (n+1) / 4 = \Theta (n^2)$

What if we use binary search?

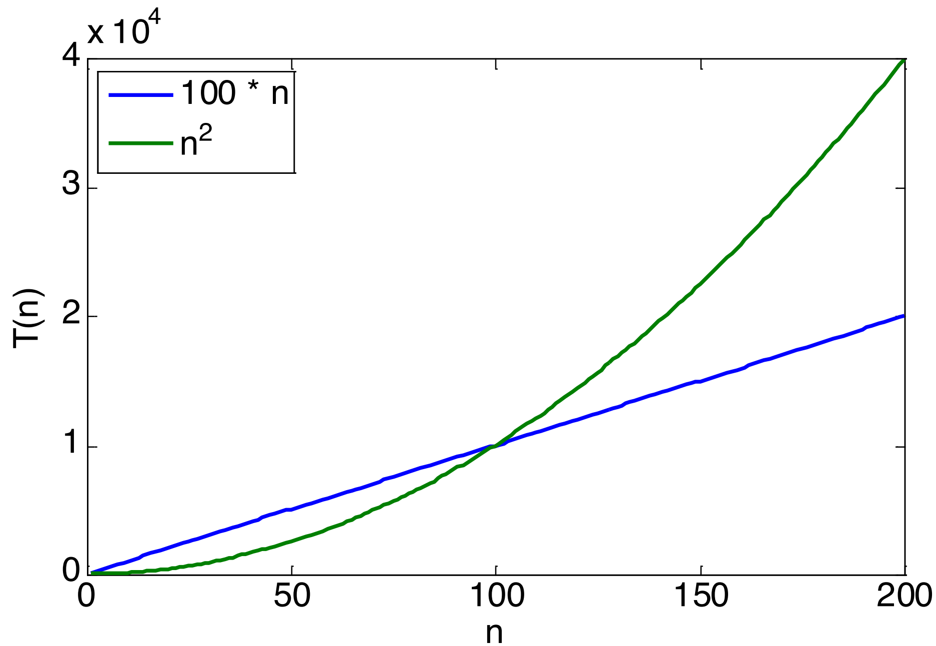
Asymptotic Analysis

- Running time depends on the size of the input
  - Larger array takes more time to sort
  - T(n): the time taken on input with size n
  - Look at ***growth*** of $T(n)$ as $n \to \infty$.

    "Asymptotic Analysis"
- Size of input is generally defined as the number of input elements
  - In some cases may be tricky

Asymptotic Analysis

- Ignore actual and abstract statement costs
- *Order of growth* is the interesting measure:
  - Highest-order term is what counts
    - As the input size grows larger it is the high order term that dominates



Comparison of functions

	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10	33	10^2	10^3	10^3	10^6
10^2	6.6	10^2	660	10^4	10^6	10^{30}	10^{158}
10^3	10	10^3	10^4	10^6	10^9		
10^4	13	10^4	10^5	10^8	10^{12}		
10^5	17	10^5	10^6	10^{10}	10^{15}		
10^6	20	10^6	10^7	10^{12}	10^{18}		

For a super computer that does 1 trillion operations per second, it will be longer than 1 billion years

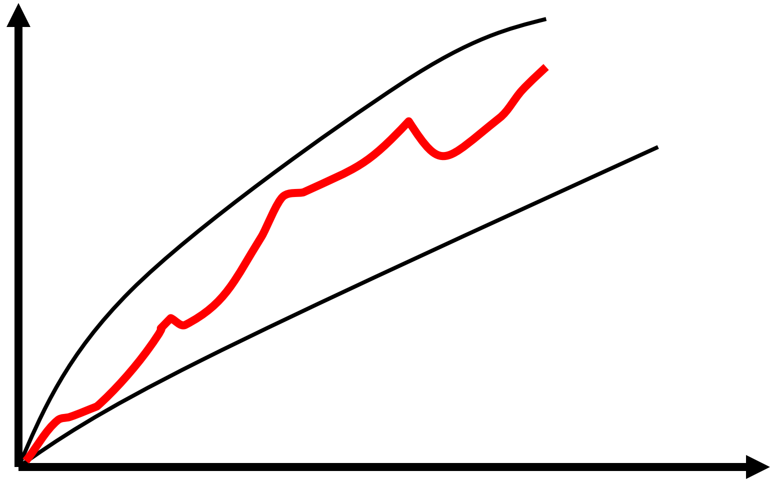
Order of growth

$1 << \log_2 n << n << n\log_2 n << n^2 << n^3 << 2^n << n!$

(We are slightly abusing of the “<<“ sign. It means a smaller order of growth).

Exact analysis is hard!

- Worst-case and average-case are difficult to deal with precisely, because the details are very complicated



It may be easier to talk about upper and lower bounds of the function.

Asymptotic notations

- O: Big-Oh
- Ω: Big-Omega
- Θ: Theta
- o: Small-oh
- ω: Small-omega

Big O

- Informally, O (g(n)) is the set of all functions with a smaller or same order of growth as g(n), within a constant multiple
- If we say f(n) is in O(g(n)), it means that g(n) is an asymptotic upper bound of f(n)
  - Intuitively, it is like f(n) $\leq$ g(n)
- What is O(n^2)?
  - The set of all functions that grow slower than or in the same order as n^2

So:

$n \in O(n^2)$

$n^2 \in O(n^2)$

$1000n \in O(n^2)$

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

$100n^2 + n \in O(n^2)$

But:

$1/1000\ n^3 \notin O(n^2)$

Intuitively, O is like ≤

small o

- Informally, o (g(n)) is the set of all functions with a strictly smaller growth as g(n), within a constant multiple
- What is $o(n^2)$?
  - The set of all functions that grow slower than n^2

So:

$1000n \in o(n^2)$

But:

$n^2 \notin o(n^2)$

Intuitively, o is like <

Big Ω

- Informally, Ω (g(n)) is the set of all functions with a larger or same order of growth as g(n), within a constant multiple
- $f(n) \in \Omega(g(n))$ means g(n) is an asymptotic lower bound of f(n)
  - Intuitively, it is like $g(n) \leq f(n)$

Intuitively, Ω is like ≥

So:

$n^2 \in \Omega(n)$

$1/1000\ n^2 \in \Omega(n)$

But:

$1000\ n \notin \Omega(n^2)$

small ω

- Informally, ω (g(n)) is the set of all functions with a larger order of growth as g(n), within a constant multiple

So:

$n^2 \in \omega(n)$

$1/1000\ n^2 \in \omega(n)$

$n^2 \notin \omega(n^2)$

Intuitively, ω is like >

Theta (Θ)

- Informally, Θ (g(n)) is the set of all functions with the same order of growth as g(n), within a constant multiple
- $f(n) \in \Theta(g(n))$ means g(n) is an asymptotically tight bound of f(n)
  - Intuitively, it is like $f(n) = g(n)$
- What is $\Theta(n^2)$?
  - The set of all functions that grow in the same order as n^2

So:

$n^2 \in \Theta(n^2)$

$n^2 + n \in \Theta(n^2)$

$100n^2 + n \in \Theta(n^2)$

$100n^2 + \log_2 n \in \Theta(n^2)$

But:

$n\log_2 n \notin \Theta(n^2)$

$1000n \notin \Theta(n^2)$

$1/1000\ n^3 \notin \Theta(n^2)$

Intuitively, Θ is like =

Tricky cases

- How about sqrt(n) and $\log_2 n$?
- How about $\log_2 n$ and $\log_{10} n$
- How about 2^n and 3^n
- How about 3^n and $n!$?

Big-Oh

- Definition:

  There exist

  For all

  $O(g(n)) = \{f(n): \exists$ positive constants c and n_0 such that $0 \le f(n) \le cg(n)\ \forall\ n > n_0\}$

- $\lim_{n\to\infty} g(n)/f(n) > 0$ (if the limit exists.)
- Abuse of notation (for convenience):

  $f(n) = O(g(n))$ actually means $f(n) \in O(g(n))$

Big-Oh

- Claim: $f(n) = 3n^2 + 10n + 5 \in O(n^2)$
- Proof by definition

  To prove this claim by definition, we need to find some positive constants c and n_0 such that $f(n) <= cn^2$ for all $n > n_0$.

  (Note: you just need to find one concrete example of c and n0 satisfying the condition.)

  $3n^2 + 10n + 5 \leq 10n^2 + 10n + 10$

  $\leq 10n^2 + 10n^2 + 10n^2, \forall\ n \geq 1$

  $\leq 30\ n^2, \forall\ n \geq 1$

  Therefore, if we let c = 30 and $n_0 = 1$, we have $f(n) \leq c\ n^2, \forall\ n \geq n_0$.

  Hence according to the definition of big-Oh, $f(n) = O(n^2)$.
- Alternatively, we can show that

  $\lim_{n\to\infty} n^2 / (3n^2 + 10n + 5) = 1/3 > 0$

Big-Omega

- Definition:

  $\Omega(g(n)) = \{f(n): \exists$ positive constants c and n_0 such that $0 \le cg(n) \le f(n)\ \forall\ n>n_0\}$

- $\lim_{n\to\infty} f(n)/g(n) > 0$ (if the limit exists.)
- Abuse of notation (for convenience):

  $f(n) = \Omega(g(n))$ actually means $f(n) \in \Omega(g(n))$

Big-Omega

- Claim: $f(n) = n^2 / 10 = \Omega(n)$

- Proof by definition:

  $f(n) = n^2 / 10$, $g(n) = n$

  Need to find a c and a n_o to satisfy the definition of $f(n) \in \Omega(g(n))$, i.e., $f(n) \geq cg(n)$ for $n > n_0$

  $n \leq n^2 / 10$ when $n \geq 10$

  If we let $c = 1$ and $n_0 = 10$, we have $f(n) \geq cn$, $\forall\ n \geq n_0$. Therefore, according to definition, $f(n) = \Omega(n)$.

- Alternatively:

  $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} (n/10) = \infty$

Theta

- Definition:
  - $\Theta(g(n)) = \{f(n):\ \exists$ positive constants c_1, c_2, and n_0 such that $0 \le c_1\, g(n) \le f(n) \le c_2\, g(n)$, $\forall\ n \ge n_0\}$
- $\lim_{n\to\infty} f(n)/g(n) = c > 0$ and $c < \infty$
- $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
- Abuse of notation (for convenience):

  $f(n) = \Theta(g(n))$ actually means $f(n) \in \Theta(g(n))$

  $\Theta(1)$ means constant time.

Theta

- Claim: $f(n) = 2n^2 + n = \Theta(n^2)$
- Proof by definition:
  - Need to find the three constants c_1, c_2, and n_0 such that

    $c_1 n^2 \le 2n^2 + n \le c_2 n^2$ for all $n > n_0$
  - A simple solution is $c_1 = 2$, $c_2 = 3$, and $n_0 = 1$

- Alternatively, $\lim_{n \to \infty} (2n^2+n)/n^2 = 2$

More Examples

- Prove $n^2 + 3n + \lg n$ is in $O(n^2)$
- Want to find c and n_0 such that

  $n^2 + 3n + \lg n <= cn^2$ for $n > n_0$
- Proof:

$n^2 + 3n + \lg n <= 3n^2 + 3n + 3\lg n$ for $n > 1$

$<= 3n^2 + 3n^2 + 3n^2$

$<= 9n^2$

Or $n^2 + 3n + \lg n <= n^2 + n^2 + n^2$ for $n > 10$

$<= 3n^2$

More Examples

- Prove $n^2 + 3n + \lg n$ is in $\Omega(n^2)$
- Want to find c and n_0 such that

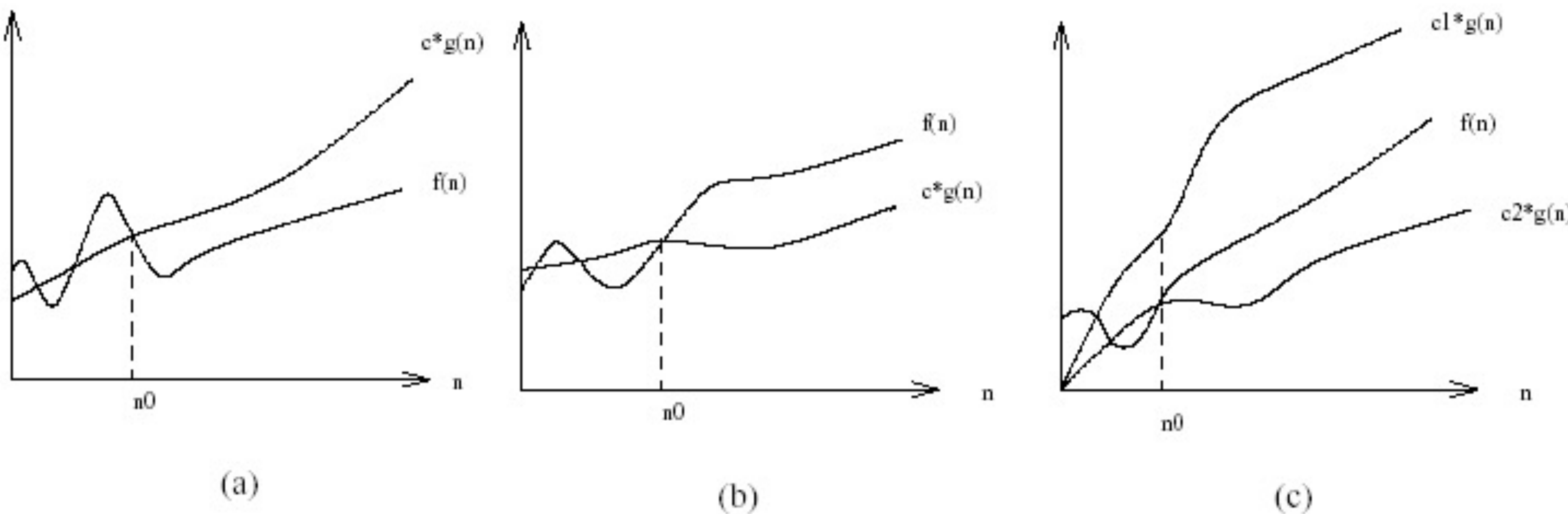
$$n^2 + 3n + \lg n >= cn^2 \text{ for } n > n_0$$

$$n^2 + 3n + \lg n >= n^2 \text{ for } n > 0$$

$n^2 + 3n + \lg n = O(n^2)$ and $n^2 + 3n + \lg n = \Omega(n^2)$

$=> n^2 + 3n + \lg n = \Theta(n^2)$

O, Ω, and Θ



The definitions imply a constant n_0 *beyond which* they are satisfied. We do not care about small values of n.

Using limits to compare orders of growth

- $$\lim_{n\to\infty} f(n)\,/\,g(n) = \begin{cases} 0 & \longrightarrow\ f(n) \in o(g(n)) \\ c \;\; > 0 & \longrightarrow\ f(n) \in \Theta\,(g(n)) \\ \infty & \longrightarrow\ f(n) \in \omega\,(g(n)) \end{cases}$$

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

$$f(n) \in \Theta\,(g(n)) \longrightarrow f(n) \in O(g(n))$$

$$f(n) \in \Theta\,(g(n)) \longrightarrow f(n) \in \Omega(g(n))$$

$$f(n) \in \omega\,(g(n)) \longrightarrow f(n) \in \Omega(g(n))$$

logarithms

- compare $\log_2 n$ and $\log_{10} n$

- $\log_a b = \log_c b \,/\, \log_c a$
- $\log_2 n = \log_{10} n \,/\, \log_{10} 2 \sim 3.3 \log_{10} n$
- Therefore $\lim(\log_2 n \,/\, \log_{10} n) = 3.3$
- $\log_2 n = \Theta\,(\log_{10} n)$

- Compare 2^n and 3^n

- $\lim\limits_{n\to\infty} 2^n / 3^n = \lim\limits_{n\to\infty}(2/3)^n = 0$
- Therefore, $2^n \in o(3^n)$, and $3^n \in \omega(2^n)$

- How about 2^n and 2^{n+1}?

  $2^n / 2^{n+1} = ½$, therefore $2^n = \Theta\ (2^{n+1})$

L' Hopital's rule

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

Condition:

If both lim f(n) and lim g(n) are ∞ or 0

- You can apply this transformation as many times as you want, as long as the condition holds

- Compare $n^{0.5}$ and $\log n$

- $\lim_{n \to \infty} n^{0.5} / \log n = ?$

- $(n^{0.5})' = 0.5\, n^{-0.5}$
- $(\log n)' = 1 / n$
- $\lim (n^{-0.5} / 1/n) = \lim(n^{0.5}) = \infty$
- Therefore, $\log n \in o(n^{0.5})$
- In fact, $\log n \in o(n^{\varepsilon})$, for any $\varepsilon > 0$

Stirling's formula

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n = \sqrt{2\pi}\, n^{n+1/2} e^{-n}$$

$n! \approx$ (constant) $n^{n+1/2} e^{-n}$

- Compare 2^n and $n!$

$$\lim_{n\to\infty}\frac{n!}{2^n}=\lim_{n\to\infty}\frac{c\sqrt{n}n^n}{2^n e^n}=\lim_{n\to\infty}c\sqrt{n}\left(\frac{n}{2e}\right)^n=\infty$$

- Therefore, $2^n = o(n!)$

- Compare n^n and $n!$

$$\lim_{n\to\infty}\frac{n!}{n^n}=\lim_{n\to\infty}\frac{c\sqrt{n}n^n}{n^n e^n}=\lim_{n\to\infty}\frac{c\sqrt{n}}{e^n}=0$$

- Therefore, $n^n = \omega(n!)$

- How about log (n!)?

$$\log(n!) = \log \frac{c\sqrt{n}n^n}{e^n} = C + \log n^{n+1/2} - \log(e^n)$$

$$= C + n\log n + \frac{1}{2}\log n - n$$

$$= C + \frac{n}{2}\log n + (\frac{n}{2}\log n - n) + \frac{1}{2}\log n$$

$$= \Theta(n\log n)$$

More advanced dominance ranking

$$n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\epsilon} \gg n \log n \gg n \gg \sqrt{n} \gg$$
$$\log^2 n \gg \log n \gg \log n / \log \log n \gg \log \log n \gg \alpha(n) \gg 1$$

Asymptotic notations

- O: Big-Oh
- Ω: Big-Omega
- Θ: Theta
- o: Small-oh
- ω: Small-omega
- Intuitively:

O is like $\leq$	Ω is like $\geq$	Θ is like =
o is like <	ω is like >	