Basics of Complexity Analysis: The RAM Model and the Growth of Functions

Antonio Carzaniga

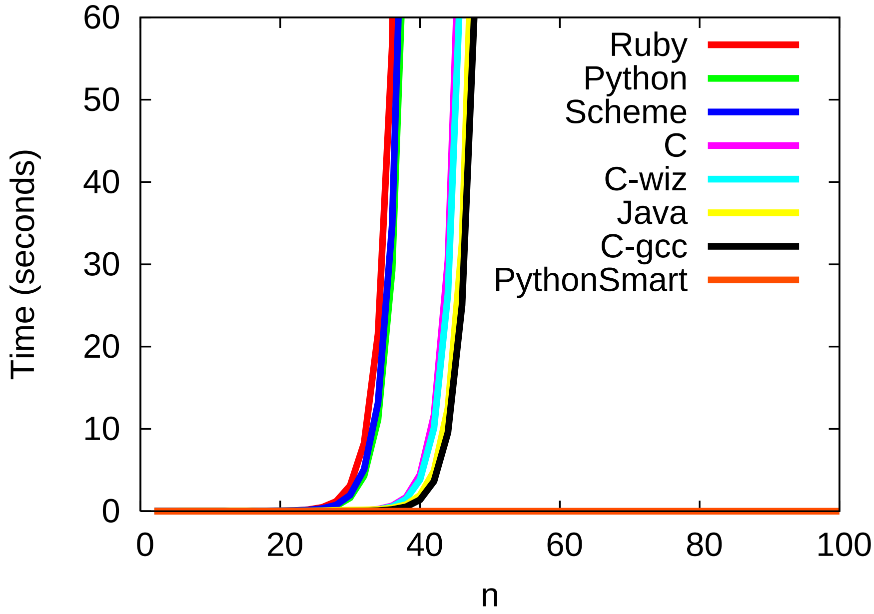
Faculty of Informatics
University of Lugano

Semptember 24, 2008

Outline

- Informal analysis of two Fibonacci algorithms
- The *random-access machine* model
- Measure of complexity
- Characterizing functions with their asymptotic behavior
- Big-*O*, omega, and theta notations

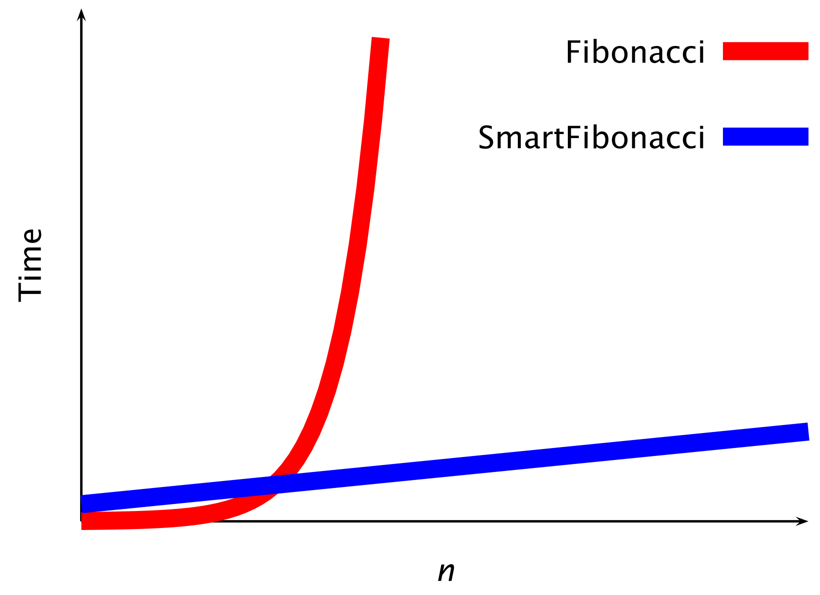
Slow vs. Fast Fibonacci



Slow vs. Fast Fibonacci

- We informally characterized our two Fibonacci algorithms
  - Fibonacci is *exponential* in *n*
  - SmartFibonacci is (almost) *linear* in *n*
- How do we characterize the complexity of algorithms?
  - in general
  - in a way that is specific of the *algorithms*
  - but *independent of any implementation detail*

Slow vs. Fast Fibonacci



A Model of the Computer

- An informal model of the *random-access machine (RAM)*
- *Basic types* in the RAM model
  - integer and floating-point numbers
  - limited size of each "word" of data (e.g., 64 bits)
- *Basic steps* in the RAM model
  - *operations involving basic types*
  - load/store: assignment, use of a variable
  - arithmetic operations: addition, multiplication, division, etc.
  - branch operations: conditional branch, jump
  - subroutine call
- A *basic step* in the RAM model takes a *constant time*

Analysis in the RAM Model

SmartFibonacci(n)	*cost*	*times* ($n > 1$)
if $n = 0$	c_1	1
then return 0	c_2	0
elseif $n = 1$	c_3	1
then return 1	c_4	0
else $pprev \leftarrow 0$	c_5	1
$prev \leftarrow 1$	c_6	1
for $i \leftarrow 2$ **to** n	c_7	$n - 1$
do $f \leftarrow prev + pprev$	c_8	$n - 1$
$pprev \leftarrow prev$	c_9	$n - 1$
$prev \leftarrow f$	c_{10}	$n - 1$
return f	c_{11}	1

$$T(n) = c_1 + c_3 + c_5 + c_6 + c_{11} + (n-1)(c_7 + c_8 + c_9 + c_{10})$$

$$T(n) = nC_1 + C_2 \quad \Rightarrow T(n) \text{ is a } \textit{linear function} \text{ of } n$$

Input Size

- In general we measure the complexity of an algorithm as a function of the *size* of the input
  - size measured in bits
  - did we do that for SmartFibonacci?
- **Example:** given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, and a value x, output true if A contains x

```
Find(A, x)
for i ← 1 to length(A)
        do if A[i] = x
                then return true
return false
```

$$T(n) = Cn$$

Worst-Case Complexity

- In general we measure the complexity of an algorithm *in the worst case*
- **Example:** given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, output true if A contains two equal values $a_i = a_j$ (with $i \neq j$)

```
FindEquals(A)
for i ← 1 to length(A) − 1
      do for j ← i + 1 to length(A)
            do if A[i] = A[j]
                  then return true
return false
```

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

Constant Factors

- Does a load/store operation cost more than, say, an arithmetic operation?

$$\boxed{x \leftarrow 0} \text{ vs. } \boxed{y + z}$$

- *We do not care about the specific costs of each basic step*
  - these costs are likely to vary significantly with languages, implementations, and processors
  - so, we assume $c_1 = c_2 = c_3 = \cdots = c_i$
  - we also ignore the specific *value* c_i, and in fact *we ignore every constant cost factor*

Order of Growth

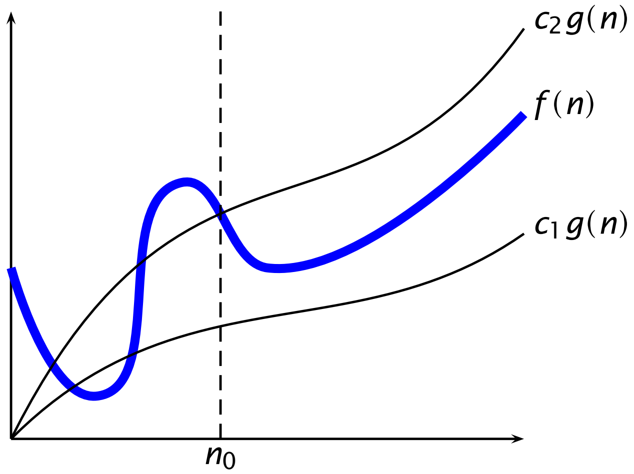
- We care only about the *order of growth* or *rate of growth* of $T(n)$
  - so we ignore lower-order terms

    E.g., in
    $$T(n) = an^2 + bn + c$$
    we only consider the n^2 term and say that $T(n)$ is a quadratic function in n

    We write
    $$\boxed{T(n) = \Theta(n^2)}$$
    and say that "$T(n)$ is theta of n-squared"

Θ-Notation

- Given a function $g(n)$, we define the *family of functions* $\Theta(g(n))$



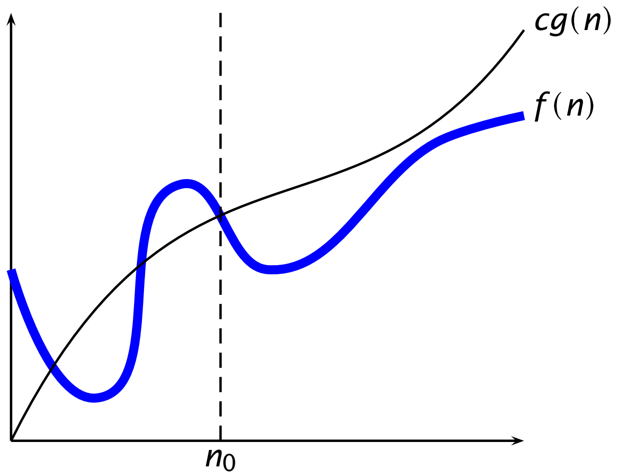
$$\Theta(g(n)) = \{f(n) : \exists c_1 > 0, \exists c_2 > 0, \exists n_0 > 0$$
$$: 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}$$

Examples

- $T(n) = n^2 + 10n + 100 \quad \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \quad \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \quad \Rightarrow T(n) = \Theta(n\sqrt{n})$
- $T(n) = 2^{\frac{n}{6}} + n^7 \quad \Rightarrow T(n) = \Theta(2^{\frac{n}{6}})$
- $T(n) = \frac{10+n}{n^2} \quad \Rightarrow T(n) = \Theta(\frac{1}{n})$
- $T(n) =$ complexity of SmartFibonacci $\quad \Rightarrow T(n) = \Theta(n)$
- We characterize the behavior of $T(n)$ *in the limit*
- The Θ-notation is an *asymptotic notation*

O-Notation

- Given a function $g(n)$, we define the *family of functions* $O(g(n))$



$$O(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \\ : 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

Examples

- $T(n) = n^2 + 10n + 100 \quad \Rightarrow T(n) = O(n^2) \quad \Rightarrow T(n) = O(n^3)$
- $T(n) = n + 10 \log n \quad \Rightarrow T(n) = O(2^n)$
- $T(n) = n \log n + n\sqrt{n} \quad \Rightarrow T(n) = O(n^2)$
- $T(n) = 2^{\frac{n}{6}} + n^7 \quad \Rightarrow T(n) = O((1.5)^n)$
- $T(n) = \frac{10+n}{n^2} \quad \Rightarrow T(n) = O(1)$
- $f(n) = \Theta(g(n)) \Rightarrow T(n) = O(g(n))$
- $f(n) = \Theta(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
- $f(n) = O(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = O(h(n))$

Examples

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2\log_2 n})$? YES
- $f(n) = O(2^n) \Rightarrow f(n) = \Theta(n^2)$? NO
- $\sqrt{n} = O(\log^2 n)$? NO
- $n^2 + (1.5)^n = O(2^{\frac{n}{2}})$? NO

Example

- So, what is the complexity of FindEquals?

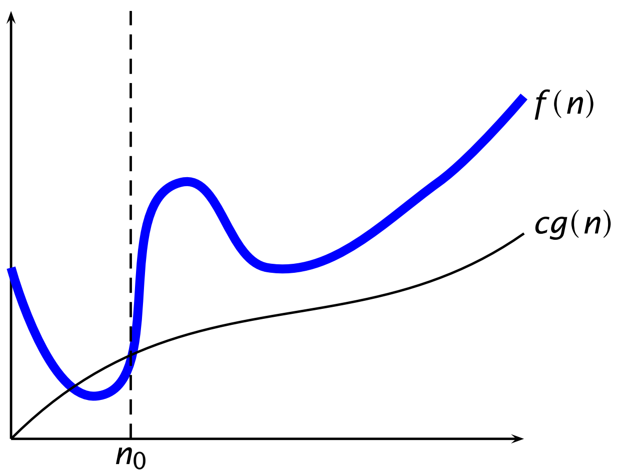
```
FindEquals(A)
for i ← 1 to length(A) − 1
    do for j ← i + 1 to length(A)
        do if A[i] = A[j]
            then return true
return false
```

$$T(n) = \Theta(n^2)$$

- $n = \mathit{length}(A)$
- we measure the *worst-case complexity*

Ω-Notation

- Given a function $g(n)$, we define the *family of functions* $\Omega(g(n))$



$$\Omega(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \\ : 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}$$

Θ, O, and Ω as Relations

- *Theorem:* for any two functions $f(n)$ and $g(n)$,
  $f(n) = \Omega(g(n)) \wedge f(n) = O(g(n)) \Leftrightarrow f(n) = \Theta(g(n))$

- The Θ-notation, Ω-notation, and O-notation can be viewed as the "asymptotic" $=$, $\geq$, and $\leq$ relations for functions, respectively

- The above theorem can be interpreted as saying

$$f \geq g \wedge f \leq g \Leftrightarrow f = g$$

- When $f(n) = O(g(n))$ we say that $g(n)$ is an *upper bound* for $f(n)$, and that $g(n)$ *dominates* $f(n)$

- When $f(n) = \Omega(g(n))$ we say that $g(n)$ is a *lower bound* for $f(n)$

Θ, O, and Ω as Anonymous Functions

- We can use the Θ-, O-, and Ω-notation to represent anonymous (unknown or unsecified) functions
  E.g.,

$$f(n) = 10n^2 + O(n)$$

  means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n.

- Examples

  $n^2 + 4n - 1 = n^2 + \Theta(n)$? YES

  $n^2 + \Omega(n) - 1 = O(n^2)$? NO

  $n^2 + O(n) - 1 = O(n^2)$? YES

  $n \log n + \Theta(\sqrt{n}) = O(n\sqrt{n})$? YES

o-Notation

- The upper bound defined by the O-notation may or may not be *asymptotically tight*

  E.g.,

  $n \log n = O(n^2)$ is not asymptotically tight

  $n^2 - n + 10 = O(n^2)$ is asymptotically tight

- We use the o-notation to denote upper bounds that are *not* asymtotically tight. So, given a function $g(n)$, we define the family of functions $o(g(n))$

$$o(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \\ : 0 \le f(n) < cg(n) \text{ for all } n \ge n_0\}$$

ω-Notation

- The lower bound defined by the Ω-notation may or may not be *asymptotically tight*

  E.g.,

  $2^n = \Omega(n \log n)$ is not asymptotically tight

  $n + 4n \log n = \Omega(n \log n)$ is asymptotically tight

- We use the ω-notation to denote lower bounds that are *not* asymtotically tight. So, given a function $g(n)$, we define the family of functions $\omega(g(n))$

$$\omega(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \\ : 0 \le cg(n) < f(n) \text{ for all } n \ge n_0\}$$