Big-O Notation and Complexity Analysis

Jonathan Backer
backer@cs.ubc.ca

Department of Computer Science
University of British Columbia

May 28, 2007

Problems

Reading:

- CLRS: "Growth of Functions" 3
- GT: "Algorithm Analysis" 1.1-1.3

Course is about solving problems with algorithms: Find a function from a set of valid *inputs* to a set of *outputs*. An instance of a problem is just one specific input.

Sorting

- Input: A sequence of n values $a_1, a_2, \ldots, a_n$.
- Output: A permutation $b_1, b_2, \ldots, b_n$ of $a_1, a_2, \ldots, a_n$ such that $b_1 \leq b_2 \leq \ldots \leq b_n$.
- Instance: $3, 8, 2, 5$.

Compiling a Program

- Input: A sequence of characters (file).
- Output: A sequence of bytes (either an executable file or error messages).

Algorithms

An algorithm is a *finite* set of instructions such that

- each step is precisely stated (e.g. english instructions, pseudo-code, flow charts, etc.),
- the result of each step is uniquely defined and depends only on the input and previously executed steps, and
- it stops after finitely many steps on every instance of the problem (i.e. no infinite loops).

Algorithms (cont'd)

What follows is a pseudo-code description of the insertion sort algorithm. We more interested in clarity than syntax.

```
InsertionSort(A)
   for j ← 2 to A.length-1 do
      key ← A[j]
      i ← j-1
      while (i ≥ 0 and key < A[i]) do
         A[i+1] ← A[i]
         i ← i-1
      A[i+1] ← key
```

Just like Java, we pass parameters by value for simple types and reference for arrays and objects.

Analysis

We analyse the behaviour of algorithms. That is, we will *prove*

- an algorithm is correct (i.e. it always terminate and returns the right result)
- a bound on its best-/worst-/average-case time or space complexity

A machine model captures the relevant properties of the machine that the algorithm is running on. We usually use the Integer-RAM model with

- constant time memory access,
- sequential instruction execution,
- a single processor, and
- memory cells that hold integers of **arbitrary size**.

Time Complexity

We count the # of elementary steps, which depends on the instance of the problem. We look at

- worst-case: usual,
- best-case: typically not very useful, and
- average-case: hard and really depends on input distribution (i.e what is average?).

Search an unsorted list of n numbers.

- Worst-case: n comparisons (not in the list).
- Best-case: 1 comparison (won the lottery!).
- Average-case: $n/2$ comparisons, if all numbers are different, the number occurs in the array, and each permutation of the array is equally likely.

Time Complexity (cont'd)

Why analyse the worst-case? Because it

- provides an upperbound,
- may frequently occur, and
- is often related to the average-case (but it is much easier to prove).

How do we count elementary steps?

- It is hard to do exactly (even in worst-case).

So we use asymptotic notation because it

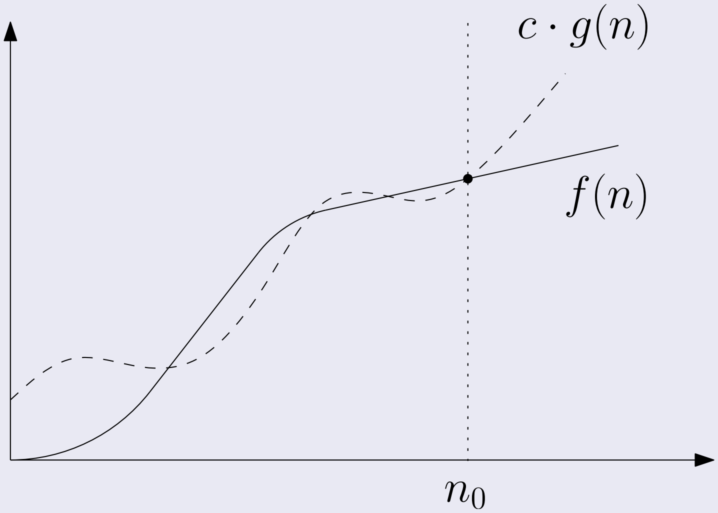
- focuses on behaviour in the limit (where things break) and
- is independent of underlying technology (e.g. machine, OS, compiler, etc.).

Big-O Notation

Intuition: f is upperbounded by a multiple of g in the limit.

Definition

Let $g : \mathbb{N} \rightarrow \mathbb{R}$. Then $f : \mathbb{N} \rightarrow \mathbb{R}$ is in $O(g(n))$ if and only if $\exists c \in \mathbb{R}^+$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$.



Constructive Big-O Proofs

Proofs following the definition. Called constructive because we construct specific c and n_0.

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

Take $c = 1,\ n_0 = 2$.

$$\begin{aligned} 2n &\le 2 \cdot n \\ &\le n \cdot n \\ &= n^2 \end{aligned}$$

Or take $c = 2,\ n_0 = 1$.

$$\begin{aligned} 2n &\le 2 \cdot n \cdot n \\ &\le 2n^2 \end{aligned}$$

Show $7n^2 + 5 \in O(n^3/6)$

Take $c = 72,\ n_0 = 1$.

$$\begin{aligned} 7n^2 + 5 &\le 7n^2 + 5n^2 \\ &\le 12n^2 \\ &\le 12n^3 \\ &\le 72 \cdot \frac{n^3}{6} \end{aligned}$$

Big-O Proofs by Contradiction

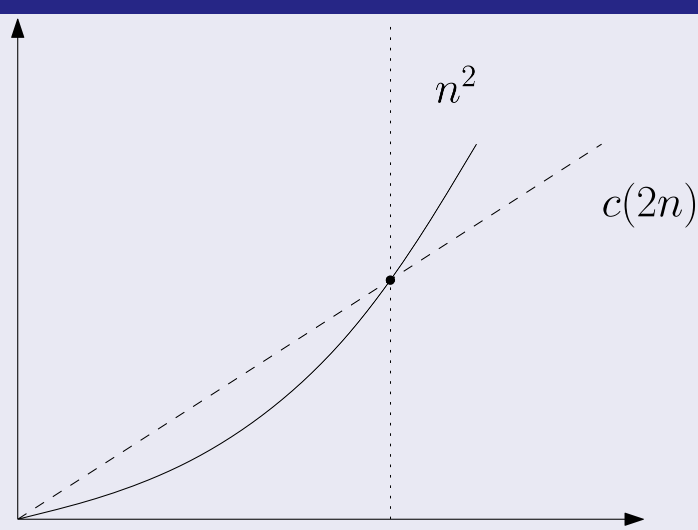
Typically used to prove that $f(n) \notin O(g(n))$.

Show $n^2 \notin O(2n)$

Suppose $n^2 \in O(2n)$ and consider any $n > \max\{n_0, 2c\}$. Then

$$\begin{aligned} n^2 &= n \cdot n \\ &> (2c) \cdot n \\ &= c \cdot (2n) \end{aligned}$$

which is a contradiction.



Big-O Ignores Constant Factors

Theorem

If $f(n) \in O(g(n))$, then $x \cdot f(n) \in O(g(n))$ for every (constant) $x \in \mathbb{R}^+$.

Proof.

Since $f(n) \in O(g(n))$, consider c, n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. Let $b = c \cdot x$. Then, for $n > n_0$

$$\begin{aligned} x \cdot f(n) &\leq x \cdot c \cdot g(n) \\ &= b \cdot g(n) \end{aligned}$$

Hence $x \cdot f(n) \in O(g(n))$. □

Big-O Ignores Lower Order Terms

Theorem

If $f(n) \in O(g(n))$ and $h(n) \in O(f(n))$, then $f(n) + h(n) \in O(g(n))$.

Proof.

Consider a, l_0 such that $f(l) \leq a \cdot g(l)$, for $l \geq l_0$. Consider b, m_0 such that $h(m) \leq b \cdot f(m)$, for $m \geq m_0$. Let $c = a \cdot (1 + b)$ and $n_0 = \max\{l_0, m_0\}$. Then for $n \geq n_0$

$$\begin{aligned} f(n) + h(n) &\leq f(n) + b \cdot f(n) \\ &= (1 + b) \cdot f(n) \\ &\leq (1 + b) \cdot a \cdot g(n) \\ &= c \cdot g(n) \end{aligned}$$

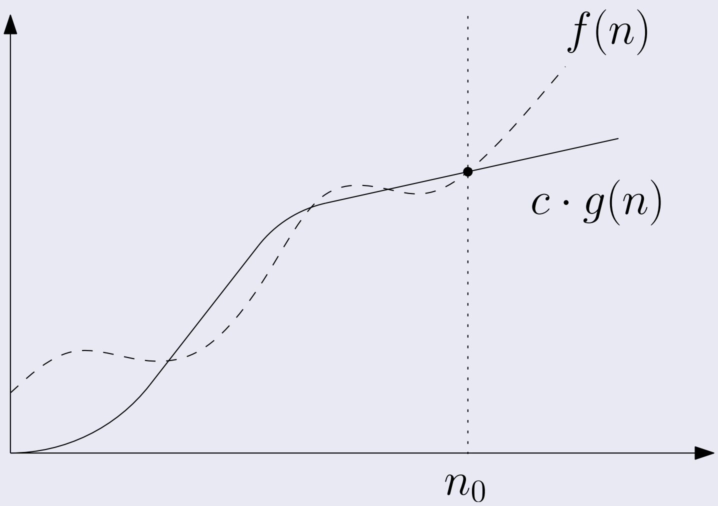
So $f(n) + h(n) \in O(g(n))$ □

Big-Ω Notation

Intuition: f is lowerbounded by a multiple of g in the limit.

Definition

Let $g : \mathbb{N} \to \mathbb{R}$. Then $f : \mathbb{N} \to \mathbb{R}$ is in $\Omega(g(n))$ if and only if $\exists c \in \mathbb{R}^+$ and $n_0 \in \mathbb{N}$ such that $f(n) \geq c \cdot g(n)$, $\forall n \geq n_0$.



Other Asymptotic Notations

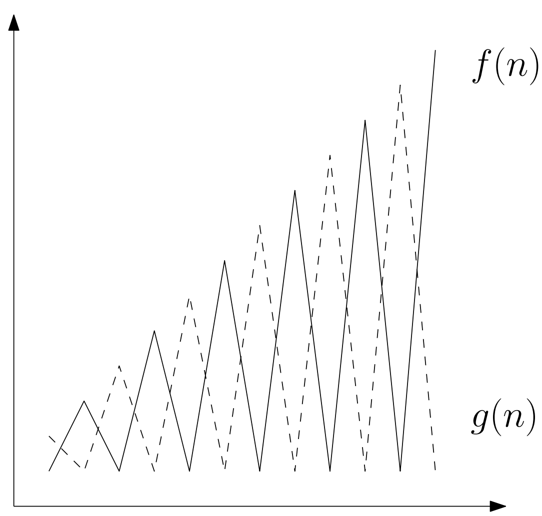
Definition

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

There is a correspondence:

$<$	$\leq$	$=$	$\geq$	$>$
o	O	θ	Ω	ω

Except that not every pair of functions is comparable.



Limits

Theorem

Let $f, g : \mathbb{N} \to \mathbb{R}^+$. Suppose $L = \lim_{n\to\infty} f(n)/g(n)$ exists. Then

- $f(n) \in \omega(g(n))$, *if* $L = +\infty$
- $f(n) \in \Theta(g(n))$, *if* $L \in \mathbb{R}^+$
- $f(n) \in o(g(n))$, *if* $L = 0$

Show $\sqrt{n} \in \omega(\log n)$

$$\lim_{n\to\infty} \frac{\sqrt{n}}{\log n} = \frac{\infty}{\infty} \text{ so use L'Hopital's Rule}$$

$$= \lim_{n\to\infty} \frac{\frac{1}{2}n^{-\frac{1}{2}}}{n^{-1}} = \lim_{n\to\infty} \frac{1}{2}\sqrt{n} = \infty$$

Time Complexity (Redux)

How do you determine the run-time of an algorithm?

- Pick a barometer: An operation performed a # of times proportional to the (worst case) running time.
- Count how many times the barometer is performed.

Example

```
for i ← 1 to n do
    for j ← 1 to 2^i do
        print "Hi!"
```

$$T(n) = \sum_{i=1}^{n} 2^i = \frac{2^{n+1}-1}{2-1} - 1 = 2^{n+1} - 2 \in \Theta(2^n)$$

Geometric Series: $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$