CSC165H, Mathematical expression and reasoning for computer science
week 10

17th November 2005

Gary Baumgartner and Danny Heap
heap@cs.toronto.edu
SF4306A
416-978-5899
http://www.cs.toronto.edu/~heap/165/S2005/index.shtml

More theorems

To show that $(f \in O(g) \wedge g \in O(h)) \Rightarrow f \in O(h)$, we need to find a constant $c \in \mathbb{R}^+$ and a constant $B \in \mathbb{N}$, that satisfy:

$$\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq ch(n).$$

Since we have constants that scale h to g and then g to f, it seems clear that we need their product to scale g to f. And if we take the maximum of the two starting points, we can't go wrong. Making this precise.

Theorem 1:
Assume $f \in O(g) \wedge g \in O(h)$.

So $f \in O(g)$.

So $g \in O(h)$.

So $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n > B \Rightarrow f(n) \leq cg(n)$. (by defn. of $f \in O(g)$).

Let $c_g \in \mathbb{R}^+, B_g \in \mathbb{N}$ be such that $\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq c_g g(n)$.

So $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow g(n) \leq ch(n)$. (by defn. of $g \in O(h)$).

Let $c_h \in \mathbb{R}^+, B_h \in \mathbb{N}$ be such that $\forall n \in \mathbb{N}, n \geq B_h \Rightarrow g(n) \leq c_h h(n)$.

Let $c = c_g c_h$. Let $B = \max(B_g, B_h)$.

Let $n \in \mathbb{N}$.

Suppose $n \geq B$.

Then $n \geq B_h$ (definition of max), so $g(n) \leq c_h h(n)$.
Then $n \geq B_g$ (definition of max), so
$f(n) \leq c_g g(n) \leq c_g c_h h(n)$.
So $f(n) \leq ch(n)$.

So $n \geq B \Rightarrow f(n) \leq ch(n)$.

Since n is an arbitrary natural number,
$\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq ch(n)$.

Since c is a positive real number, since B is a natural number,
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq ch(n)$.

So $f \in O(g)$, by definition.

So $(f \in O(g) \wedge g \in O(h)) \Rightarrow f \in O(g)$

To show that $g \in \Omega(f) \Leftrightarrow f \in O(g)$, it is enough to note the the constant, c, for one direction is positive, so its reciprocal will work for the other direction.[1]

Theorem 2:
$g \in \Omega(f)$
$\Leftrightarrow$ (definition)
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow g(n) \geq cf(n)$
$\Leftrightarrow$. (by letting $c' = 1/c$ and $B' = B$).
$\exists c' \in \mathbb{R}^+, \exists B' \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq c'g(n)$.
$\Leftrightarrow$ (definition)
$f \in O(g)$.

To show $g \in \Theta(f) \Leftrightarrow g \in O(f) \wedge g \in \Omega(f)$, it's really just a matter of unwrapping the definitions.

Theorem 3:

$g \in \Theta(f)$
$\Leftrightarrow$ (definition)
$\exists c_1, c_2 \in \mathbb{R}^+, B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \rightarrow c_1 f(n) \leq g(n) \leq c_2 f(n)$.

$\Leftrightarrow$ (combined inequality, and $B = \max(B_1, B_2)$).
$\exists c_1 \in \mathbb{R}^+, \exists B_1 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B_1 \Rightarrow g(n) \geq c_1 f(n) \wedge$
$\exists c_2 \in \mathbb{R}^+, \exists B_2 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B_2 \Rightarrow g(n) \leq c_2 f(n)$
$\Leftrightarrow$ (definition)
$g \in \Omega(f) \wedge g \in O(f)$.

Taxonomy of results

A lemma is a small result needed to prove something we really care about. A theorem is the main result that we care about (at the moment). A corollary is an easy (or said to be easy) consequence of another result. A conjecture is something suspected to be true, but not yet proven.

Here's an example of a conjecture whose proof has evaded the best minds for over 70 years. Maybe you'll prove it.

Define $f(n)$, for $n \in \mathbb{N}$ by:

$$f(n) = \begin{cases} n/2, & n \text{ even} \\ 3n+1, & n \text{ odd} \end{cases}$$

Let's call $f(f(n))$ $f^2(n)$, and $f(f^k(n))$ $f^{k+1}(n)$. Here's the conjecture: $\forall n \in \mathbb{N}, \exists k \in \mathbb{N}, f^k(n) = 1$. Easy to state, but (so far) hard to prove or disprove.

Here's an example of a corollary that recycles some of the theorems we've already proven (so we don't have to do the grubby work). To show $g \in \Theta(f) \Leftrightarrow f \in \Theta(g)$, I re-use theorems proved above and the commutativity of $\wedge$:

$g \in \Theta(f)$
$\Leftrightarrow$ (by Theorem 3)
$g \in O(f) \wedge g \in \Omega(f)$.
$\Leftrightarrow$ (by Theorem 2)
$g \in O(f) \wedge f \in O(g)$
$\Leftrightarrow$ (by commutativity of $\wedge$)
$f \in O(g) \wedge g \in O(f)$
$\Leftrightarrow$ (by Theorem 2)
$f \in O(g) \wedge f \in \Omega(g)$
$\Leftrightarrow$ (by Theorem 3)
$f \in \Theta(g)$.

Running time of programs

For any program P and any input x, let $t_P(x)$ denote the number of "steps" P takes on input x. We need to specify what we mean by a "step." Consider the following (somewhat arbitrary) accounting for common program steps:

METHOD CALL: 1 step + steps to evaluate each argument, + steps to execute the method.

RETURN STATEMENT: 1 step + steps to evaluate return value.

IF STATEMENT: 1 step + steps to evaluate condition

ASSIGNMENT STATEMENT: 1 + steps to evaluate each side

ARITHMETIC, COMPARISON, BOOLEAN OPERATORS: 1 + steps to evaluate each operand.

ARRAY ACCESS: 1 + steps to evaluate index

MEMBER ACCESS: 2 steps

CONSTANT, VARIABLE EVALUATION: 1 step

Example: Linear Search

// A is an array, x is an element to search for.
// Return an index i such that $A[i] = x$
// if there is no such index, return -1
// Convention: array indices start at 0

```
LS (A,x)
1. i = 0;                          // 3 steps (evaluate variable, constant, assignment)
2.   while (i < A.length) {         // 5 steps (while, A.length (2), i, <)
3.         if (A[i] == x) {         // 5 steps (A[i], ==, x, if)
4.           return i;              // 2 steps (return, i)
5.         }
6.         i = i+1;                 // 5 steps (i, assignment, i, +, 1)
7.   }
8. return -1;                      // 2 steps (return, -1)
9. }
```

Let's trace a function call, $t_{LS}([2,4,6,8],4)$:

1. 3 steps (i=0)

2. 5 steps ($0 < 4$)

3. 5 steps ($A[0] == 4$)

6. 5 steps (i = 1)

2. 5 steps ($1 < 4$)

3. 5 steps ($A[1] == 4$)

4. 2 (return 1)

So $t_{LS}([2,4,6,8],4) = 30$. Notice that if the first index where x is found is j, then $t_{LS}(A,x)$ will count lines 2, 3, and 6 for each index from 0 to $j-1$ (j indices), and then count lines 2,3,4 when for index j, and so $t_{LS}(A,x)$ will be $3+15j+12 = 15(j+1)$.

If x doesn't appear in A, then $t_{LS}(A,x) = 3+15A.length+7 = 15A.length+10$, because line 1 executes once, lines 2,3, and 6 execute for each index from 0 to $A.length-1$ (A.length indices), and then lines 2 and 8 execute.

We want a measure that depends on the size of the input, not the particular input. There are three standard ways:

BEST-CASE COMPLEXITY: $\min(t_p(x))$, where x is an input of size n. In other words, $\min\{t_p(x) | x \in I \wedge size(x) = n\}$.

WORST-CASE COMPLEXITY: $max(t_p(x))$, where x is an input of size n. In other words, $\max\{tp(x) | x \in I \wedge size(x) = n\}$.

AVERAGE-CASE COMPLEXITY (ASSUMING ALL INPUTS EQUALLY LIKELY):

$$A_p(n) = \frac{\sum_{x \text{ of size } n} t_p(x)}{\text{number of inputs of size } n}$$

Best-case: useless. Average-case: difficult to compute. Worst-case: easier to compute, and gives a guarantee.

What is meant by "input size"? This depends on the algorithm. For linear search, the number of elements in the array is a reasonable parameter. Technically (CSC363, etc.) the size is the number of bits required to represent the input in binary. In practice we use the number of elements of input (length of array, number of nodes in a tree, etc.)

Best-case for linear search?[2] Worst-case for linear search?[3] Average-case for linear search?[4] Now we can use the machinery of big-Oh. Suppose U is an upper bound on the worst-case running time of some algorithm P, denoted $T_P(n)$:

$t_P \in O(U)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow T_P(n) \leq cU(n)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow \max\{t_p(x) | x \in I \wedge size(x) = n\} \leq cU(n)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow \forall x \in I, size(x) = n \Rightarrow t_p(x) \leq cU(n)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall x \in I, size(x) \geq B \Rightarrow t_p(x) \leq cU(size(x))$

(I is the set of all inputs for P). So to show that $T_P \in O(U(n))$, you need to find constants c and B and show that for an arbitrary input x of size n, P takes at most $cU(n)$ steps.

In the other direction, L is a lower bound on the worst-case running time of algorithm P:

$T_P \in \Omega(L)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow \max\{t_p(x) | x \in I \wedge size(x) = n\} \geq cL(n)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow \exists x \in I, size(x) = n \wedge t_p(x) \geq cL(n)$

So, to prove that $T_p \in \Omega(L)$, we have to find constants c, B and for arbitrary n, find an input x of size n, for which we can show that P takes at least $cL(n)$ steps on input x

Notes

[1]Let's try the symmetrical presentation of bi-implication.

[2]15 steps, when $A[0] == x$.

[3]$15n + 10$, where $n = A.length$.

[4]Inputs at index 0 through $n - 1$, plus missing value equally likely $(n + 1)$ input categories, so

$$\begin{aligned}\frac{\left(\sum_{i=0}^{n-1} 15(i+1)\right) + 15n + 10}{n+1} &= \frac{15n(n+1)/2 + 15n + 10}{n+1} \\ &= \frac{7.5n(n+1) + 15n + 10}{n+1} \\ &= \frac{7.5n(n+1) + 15n + 10}{n+1} = 7.5n + \frac{15n+10}{n+1}.\end{aligned}$$