Index of /~cs61b/lectures/18

Name	Last modified	Size	Description
Parent Directory	24-Sep-98 21:22	-	

```
                         CS61b: Lecture 18
                         Monday, October 5
ANNOUNCEMENTS
=============
Congress has declared today National Kathy Yelick Day.  Please leave donations
  with the TAs at the door on your way out.

ASYMPTOTIC ANALYSIS (continued):  More Formalism
================================================

|-----------------------------------------------------------------------------|
| Omega(f(n)) is the set of all functions T(n) that satisfy:                  |
|                                                                             |
|   There exist positive constants d and N such that, for all n >= N,         |
|                              T(n) >= d f(n)                                 |
|-----------------------------------------------------------------------------|
^^^^^^^^^^  Compare with the definition of Big-Oh:  T(n) <= c f(n). ^^^^^^^^^^^

Omega is the reverse of Big-Oh.  If T(n) is in O(f(n)), f(n) is in Omega(T(n)).

  n    is  Omega(1)                  BECAUSE     1                 is  O(n).
  n^2  is  Omega(n)                  BECAUSE     n                 is  O(n^2).
  n^2  is  Omega(3 n^2 + n log n)    BECAUSE     3 n^2 + n log n   is  O(n^2).

Omega gives us a LOWER BOUND on an algorithm's worst-case performance,
  just as Big-Oh gives us an UPPER BOUND.  Big-Oh says, "Your algorithm never
  performs worse than this."  Omega says, "Don't expect your algorithm to
  behave better than this."
If we have both -- say, T(n) is in O(f(n)) and is also in Omega(g(n)) -- then
  T(n) is effectively sandwiched between the two, below f(n) and above g(n).
If f(n) and g(n) are the same, then we say that T(n) is in Theta(f(n)):

|-----------------------------------------------------------------------------|
| Theta(f(n)) is the set of all functions T(n) that are in both of:           |
|                                                                             |
|                        O(f(n)) and Omega(f(n)).                             |
|-----------------------------------------------------------------------------|

But how can a function be sandwiched between f(n) and f(n)?  Easy:  we choose
  different constants (c and d) for the upper bound and lower bound.
  For instance, here is a function T(n) in Theta(n):

                     c f(n) = 10 n
          ^                 /
          |                /               T(n)
          |               /              **
          |              /              *  *
          |             /   ***         *   **
```

```
|           /   *   *         *
|   ***    /   *     *       *
| **   ** /    *     *       *
|*       **   *       *      *
*       /  *  *        **   *
|      /    **           ***     ~~~
|     /                     ~~~~~
|    /                 ~~~~~
|   /             ~~~~~
|  /         ~~~~~     d f(n) = 2 n
| /     ~~~~~
|/ ~~~~~
O~~------------------------------> n
```

```
If we extend this graph infinitely far to the right, we find T(n) remains
  always sandwiched between the two linear functions, so T(n) is in Theta(n).
  This algorithm will never exhibit worse than linear performance, but can't
  be counted on to exhibit better than linear performance, either.
Theta is symmetric:  if f(n) is in Theta(g(n)), then g(n) is in Theta(f(n)).
  For instance, n^3 is in Theta(3 n^3 - n^2), and 3 n^3 - n^2 is in Theta(n^3).
  n^3 is not in Theta(n), and Theta(n) is not in Theta(n^3).

Recall that Big-Oh notation is misleading -- because n is in O(n^3), for
  instance.  Theta notation is not misleading in this way.
  n is NOT in Theta(n^3); but it is in Theta(n), Theta(3n + 7), etc.
Returning to the graph above, remember this caveat:  T(n) describes the
  WORST-CASE performance on an input of size n -- there may be SOME inputs
  of size n that run faster than linear!

ALGORITHM ANALYSIS
==================
Problem #1:  Given an array of n integers, find the smallest.
Algorithm #1:  Maintain a variable "min" that stores the smallest integer.
               Set "min" to the first element of the array.
               Scan through the rest of the array, updating "min" as necessary.

  Running time is Omega(n), because we scan through the whole array once.
  Running time is O(n), because all we do is scan through the whole array once.
  Therefore, running time is Theta(n).

Problem #2:  Given a set of n points, find the pair closest to each other.
Algorithm #2:  Calculate the distance between each pair; return the minimum.

  There are n (n - 1) / 2 pairs, so the running time is Theta(n^2).

  Hint:  The code for this algorithm gives a strong hint, because it contains
         a doubly-nested loop, with both loops iterating through input points.

         double minDistance = point[0].distance(point[1]);
         /* Visit a pair (i, j) of points. */
         for (int i = 0; i < numPoints; i++) {
           /* We require that j > i so that each pair is visited only once. */
           for (int j = i + 1; j < numPoints; j++) {
             double thisDistance = point[i].distance(point[j]);
             if (thisDistance < minDistance) {
               minDistance = thisDistance;
             }
           }
         }

Functions of Several Variables
```

Problem #3: Write a matchmaking program for w women and m men.
Algorithm #3: Compare each woman with each man. Decide if they're compatible.

If each comparison takes constant time then our running time, T(w, m), is in Theta(wm).

This means that there exist constants c, d, W, and M, such that d wm <= T(w, m) <= c wm for every w >= W and m >= M.

T is not in O(w^2), nor in O(m^2), nor in Omega(w^2), nor in Omega(m^2). Every one of these possibilities is eliminated either by choosing w >> m or m >> w. Conversely, w^2 is in neither O(wm) nor Omega(wm). You simply cannot compare the functions wm, w^2, and m^2.
If we expand our service to help form women's volleyball teams as well, the running time might become Theta(w^6 + wm).
This expression cannot be simplified; neither term dominates the other.

Problem #4: Recall from Project 1 that you have a database containing n music albums. You request a list of all albums whose titles begin with "The Best of"; suppose there are k such albums.
Algorithm #4: Search for the first matching album with binary search. Walk through the subset of k matching albums.

Each step of the binary search disqualifies half the albums from further consideration. Hence, after log n steps, there is only one album still under consideration. Next, the complete list of k matching albums is printed, each in constant time. The total running time is thus

Theta(log n + k).

Because k can be as large as n, it does not dominate the log n term. Because k can be as small as 1, the log n term does not dominate it. Hence, there is no simpler way to express the running time of this query. Algorithms like this are called "output-sensitive," because their performance depends partly upon the size of the output (k), which can vary greatly.

A Detail (Does anyone care? YOU decide.)

Neither of your textbooks tells you that Big-Oh, Omega, Theta, etc., are sets of functions; perhaps they presume you're not yet worthy of such sacred knowledge. But despite their misleading presentations, you should think of these notations as sets, and treat them so:

O(n^2) = { n^2, n, 1, 12 n^2 - 8 n + 4, ... so on forever }
Theta(n^2) = Theta(2 n^2) = Theta(4 n^2 - 6)
Omega(n^2) is a subset of Omega(n)
O(n^2) is a superset of O(n)
Theta(f(n)) = O(f(n)) intersect Omega(f(n))

When Weiss writes things like "T(N) = O(F(N))", he is turning you into bad mathematicians. You cannot equate a function with a set of functions. Following his perverse logic, we have N = O(N^2) and 3 N^2 = O(N^2), and therefore N = 3 N^2, which would clearly lead to moral relativism and the fall of Western civilization.