Analysis of Algorithms: Homework 1

COP 4531 (Section 1/2) CGS 5427
Piyush Kumar

Spring 2008

Problem 1: Consider the following basic problem. You are given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You would like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ – that is, the sum $A[i] + A[i + 1] + \ldots + A[j]$. (The value of array entry $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.) Here's a simple algorithm to solve this problem:

```
For i = 1, 2, ... , n
  For j = i + 1, i + 2, ... , n
    Add up array entries A[i] through A[j]
    Store the result in B[i; j]
  Endfor
Endfor
```

1. For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n. (i.e. a bound on the number of operations performed by the algorithm.)
2. For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)
3. Although the algorithm you analyzed in parts (1) and (2) is the most natural way to solve the problem – after all, it just iterates through the relevant entries of the array B, filling in a value for each – it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where: $\mathbf{lim}_{n\to\infty} \frac{g(n)}{f(n)} = 0$..

Problem 2: You are doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the highest safe rung. One possible approach is to start at the bottom and drop a jar from the first rung, then the second rung, then the third rung, and so on until the jar breaks. This approach only breaks one jar, but takes $\Theta(n)$ drops in the worst case. If we want to save time by making fewer drops, we could try binary search: drop a jar from the middle rung, see if it breaks, then recursively try the $n/4$ or $3n/4$ rung depending on the outcome. This takes $\Theta(logn)$ drops but breaks $\Theta(logn)$ jars in the worst case. We want a compromise that doesn't break so many jars but doesn't take a lot of drops.

1. Describe a strategy to find the highest safe rung that is guaranteed to break at most 2 jars, and takes $\Theta(\sqrt{n})$ drops in the worst case.
2. Describe another strategy that is guaranteed to break at most 3 jars, and takes $\Theta(n^{1/3})$ drops in the worse case.

Problem 3: Assume you have functions f and g such that $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample:

1. $\log_2 f(n)$ is $O(\log_2 g(n))$
2. $2^{f(n)}$ if $O(2^{g(n)})$
3. $f(n)^2$ is $O(g(n)^2)$.

1 Ruberic

The students were graded based on correctness of solution to the problems. For example in Problem 3, the students were required to provide counterexamples to the false statements (1) and (2) and a simple one line proof for (3). Each subproblem carried 10 points each. Formal proofs were given 10 points whereas informal proofs or incorrectly written proofs earned partial credit.