Algorithms and Complexity
Semester 2, 2012

Problem Set 1

J. Mestre

You should solve all problems above the line before the tutorial. Tutors will call out students to solve these problems in front of the class during the tutorials.

You should attempt to solve some of the problems below the line. Tutors will go over these problems during the tutorial.

Problem 1: Sort the following functions in increasing order of asymptotic growth:

$$n^2,\ \log n,\ n^{1/\log n},\ 2^n,\ n^3,\ n\log n,\ \sqrt{n}.$$

Problem 2: Given an array A consisting of n integers $A[0], A[2], \ldots, A[n-1]$, we want to compute the upper triangle matrix

$$C[i][j] = \frac{A[i] + A[i+1] + \cdots + A[j]}{j - i + 1}$$

for $0 \le i \le j < n$. Consider the following algorithm for computing C:

Algorithm 1 SUMMING-UP(A)

```
1. for i = 0, ..., n − 1 do
2.     for j = i, ..., n − 1 do
3.         add up entries A[i] through A[j] and divide by j − i + 1
4.         store result in C[i][j]
5. return C
```

Your task is to prove an upper bound on the running time of SUMMING-UP:

1. First show that the algorithm performs at most n^2 iterations.
2. Then show that the time spent per iteration is at most $O(n)$.
3. Finally, put these observation together to show that the running time $O(n^3)$.

Problem 3: Your task is to prove a lower bound on the running time of SUMMING-UP.

1. First show that each iteration where $i < \frac{1}{4}n$ and $j > \frac{3}{4}n$ takes $\Omega(n)$ time.
2. Then show that there are $\Omega(n^2)$ pairs (i, j) such that $i < \frac{1}{4}\,n$ and $j > \frac{3}{4}\,n$.
3. Finally, put these observations together to show that the running time is $\Omega(n^3)$.

Problem 4: Come up with a more efficient algorithm for computing the above matrix $C[i][j] = \frac{A[i]+A[i+1]+\cdots+A[j]}{j-i+1}$ for $0 \le i \le j < n$. Your algorithm should run in $O(n^2)$ time. Prove that this is optimal in the sense that any algorithm that performs the task must take $\Omega(n^2)$ time.

Problem 5: Give a formal proof of the transitivity of the O-notation. That is, for function f, g, and h show that if $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.

Problem 6: There is a naive implementation of priority queue based on arrays where `insert`(x) takes $O(1)$ time, and `get_min`() takes $O(n)$ time. There is also another implementation based on heaps where both operations run in $O(\log n)$ time. In each case n the size of the queue at the time the operation is performed.

Suppose we have an algorithm that performs X number of `insert`() operations and a Y number of `get_min`() operations. Suppose we only care about the time spent on priority queue operations. When does it make sense to switch from a heap-based implementation to a an array-based implementation? Assume that all inserts are preformed before the get minimum operations.

Problem 7: Your task is to implement two versions of the priority queue data structure.

1. Implement a priority queue data structure using arrays
2. Implement a priority queue data structure using heaps

Perform doubling experiments using the sorting algorithm described in class and both version of the data structure.

Problem 8 (Advanced): Design an experiment to verify that the running time of heap sort is $\Omega(n \log n)$.

Problem 9 (Advanced): Come up with an implementation of the Gale-Shapley algorithm that runs in $O(n^2)$ time.