UC Berkeley—CS 170 Problem Set 1
Lecturer: David Wagner Due on January 28 at 3:30 p.m.

Problem Set 1 for CS 170

Note

When asked for an algorithm you must give (1) a brief informal description of the algorithm, (2) a precise description using pseudo-code, (3) an argument for termination and correctness of the algorithm, and (4) an analysis of the running time of the algorithm. Be clear about what the input to the algorithm is, how you measure the size of the input, and what constitutes a "step" in your running-time analysis.

Problem 0. [Any questions?] (5 points)

What's the one thing you'd most like to see explained better in lecture or discussion sections? A one-line answer would be appreciated.

(Sometimes we botch the description of some concept, leaving people confused. Sometimes we omit things people would like to hear about. Sometimes the book is very confusing on some point. Here's your chance to tell us what those things were.)

Problem 1. [Asymptotic order of growth] (20 points)

Here is a list of functions of one variable, n.

$$\frac{n^5}{100}, 5, (\lg n)^2, \frac{n^2}{1+n}, 2^{2\lg n}, n^2 \lg n, n^2 + 27n, 2^n$$

Place them in a list from left to right, so that if $f(n)$ is any function in the list and $g(n)$ any function to its right, we have $f(n) = O(g(n))$. You do not need to prove your answer.

(For instance, if the list were $5n, 23, n^2$, the correct answer would be $23, 5n, n^2$, since $23 = O(5n)$ and $5n = O(n^2)$.)

Note added 1/23: The notation $\lg n$ refers to the binary logarithm (base 2), i.e., $\lg n = \log_2 n$. For example, $\lg 16 = 4$.

Problem 2. [Big-O notation] (20 points)

For each of the following statements, say whether they are true or false. If true, give a proof of the statement; if false, give a disproof of the statement.

(a) True or false? If $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ are two positive functions satisfying $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

(b) True or false? If $f : \mathbb{N} \to \mathbb{N}$ is any positive function, then either (1) there exists a constant $c > 0$ so that $f(n) = O(n^c)$, or (2) there exists a constant $\alpha > 1$ so that $f(n) = \Omega(\alpha^n)$.

Here $\mathbb{N} = \{1, 2, \ldots\}$ represents the set of natural numbers.

Note added 1/24: Problem 2 was changed to consider positive-valued functions only, since CLR's definition of big-O notation doesn't apply to functions that may be negative. (One person proposed $f(x) = -1$ as a counterexample to 2(b), and I hope this revision makes clear that I'm not going to accept that proposal.)

Problem 3. [Recurrence relations] (25 points)

Give asymptotically tight solutions for $T(n)$. (In other words, find an explicit formula $f(n)$ so that $T(n) = \Theta(f(n))$, and prove it.)

(a) Use a recursion tree (CLRS Exercise 4.2-4) to guess a solution, and prove your guess by induction:

$$T(n) \;=\; T(n-d) + T(d) + c \cdot 2^n$$

for constants $d \geq 1$ and $c > 0$. Assume that $T(n) = \Theta(1)$ for $n \leq d$.

(b) Use a recursion tree (CLRS Exercise 4.2-4) to guess a solution, and prove your guess by induction:

$$T(n) \;=\; T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + c$$

for constant $c > 0$. Assume that $T(n) = \Theta(1)$ for $n \leq 10$.

Problem 4. [Algorithm design] (30 points)

You are given an array $A[1..n]$ of pairwise distinct integers. Give an algorithm that finds the second smallest element in A using $n + O(\log n)$ many comparisons between array elements.

Bonus question: With how many comparisons can you find the third smallest element in A?