CSE 202 Homework 1
Divide and conquer; comparison algorithms
Fall 2002
Due Thursday, October 17

For each of the algorithm problems, design as asymptotically efficient an algorithm as possible. Give a correctness argument (explanation, if it is relatively simple, or proof if not) and time analysis. You may use any well-known algorithm or data structure, or algorithm from the text or from class, as a sub-routine without needing to provide details.

**Base Conversion**. Give an algorithm that inputs an array of \( n \) base \( b_1 \) digits representing a positive integer in base \( b_1 \) and outputs an array of base \( b_2 \) digits representing the same integer in base \( b_2 \). Get as close as possible to linear time. Assume \( b_1, b_2 \) are fixed constants.

**Binary Tree Isomorphism** Two rooted trees \( T_1 \) and \( T_2 \) are isomorphic if there is a 1-1 onto map \( f : T_1 \to T_2 \) so that \( f(\text{root}_1) = \text{root}_2 \) and \( p_2(f(x)) = f(p_1(x)) \), for every \( x \in T_1 \) except root\(_1\). (Here, root\(_1\) is the root of \( T_1 \), root\(_2\) is the root of \( T_2 \), and \( p_1, p_2 \), represent the parents in the respective trees.) Give an efficient algorithm to determine whether two \( n \) node rooted binary trees are isomorphic. (On the calibration homework, an \( O(n^2) \) algorithm was given, so you should try to do better than \( O(n^2) \). Note that \( f \) is NOT GIVEN AS INPUT—your job is to decide whether such an isomorphism exists (so your output will be a Boolean), Binary tree means that each node has at most two children, say left\(_1\) and right\(_1\) where one or more could be null. Do not assume the trees are balanced.)

**Weighted Median**, Problem 9-2, part c., p. 194

**\( k \)-Almost Sorted Arrays** Call an array \( A \) \( k \)-almost sorted if the \( l \)th largest element is in positions \( l-k, ... l+k \) in the input array. Give an \( O(n \log k) \) algorithm to sort a \( k \)-almost sorted array. (5 points) Prove that, for any \( 4 \leq k \leq n \), ANY algorithm to sort a \( k \)-almost sorted array requires time \( \Omega(n \log k) \). (The lower bound must hold for any algorithm, and it must hold for \( k \) any function of \( n \), such as \( k = \log n \).) (5 points)

**Implementation: Integer Multiplication** Implement the \( O(n \log^3) \) divide-and-conquer algorithm for integer multiplication, but with a threshold, below which naive “gradeschool” multiplication is used. Experimentally determine the optimal threshold. For what values of \( n \) do you see an improvement in the time using divide-and-conquer, both using no threshold and using the optimal threshold?