CSE 521: Design and Analysis of Algorithms — Winter 2006
Problem Set #2 — Instructor: Venkatesan Guruswami
Due on **January 31, 2006** (Tuesday) in class.

Instructions: Same as for Problem Set 1.

I'll repeat the piece of advice: Begin work on the problem set early and don't wait till the deadline is only a few days away.

Readings: Kleinberg and Tardos: Section 5.5, Chapter 6.

Each problem is worth 10 points unless noted otherwise. All problem numbers refer to the Kleinberg-Tardos textbook.

1. Modify Karatsuba's algorithm for integer multiplication by using divide-and-conquer based on splitting the integer into 3 pieces instead of two. Base your algorithm on an algorithm for multiplying two quadratic polynomials that uses evaluation and interpolation. It is possible to do this multiplication of polynomials with as few as 5 multiplications of coefficients. (It is not hard to find a way to do this with 6 multiplications but the resulting algorithm is less efficient than the basic Karatsuba algorithm.) What is the asymptotic running time of your algorithm? (Note: your running time must be better than the $O(n^{1.59})$ achieved by the basic Karatsuba algorithm.)

2. Chapter 6, Problem 16 (Reaching everyone in a tree in fewest number of rounds)

   There are many sunny days in Ithaca, NY; but this year, as it happens, the spring ROTC picnic at Cornell has fallen on a rainy day. The ranking officer decides to postpone the picnic, and must notify everyone by phone. Here is the mechanism she uses to do this.

   Each ROTC person on campus except the ranking officer reports to a unique *superior officer*. Thus, the reporting hierarchy can be described by a tree T, rooted at the ranking officer, in which each other node v has as a parent node u equal to his or her superior officer. Conversely, we will call v a *direct subordinate* of u.

   To notify everyone of the postponement, the ranking officer first calls each of her direct subordinates, one at a time. As soon as each subordinate gets the phone call, he or she must notify each of his or her direct subordinates one at a time. The process continues this way, until everyone has been notified. Note that each person in this process can only call direct subordinates on the phone.

   Now, we can picture this process as being divided into *rounds*: In one *round*, each person who has already learned of the postponement can call one of his or her direct subordinates on the phone. The number of rounds it takes for everyone to be notified depends on the sequence in which each person calls their direct subordinates.

   Give an efficient algorithm that determines the minimum number of rounds needed for everyone to be notified, and outputs a sequence of phone calls that achieves this minimum number of rounds.

   (Example omitted.)

3. Chapter 6, Problem 24 (Checking if a set of precincts is susceptible to gerrymandering)

   Gerrymandering, which has been in the news again lately, is the practice of carving up electoral districts in very careful ways so as to lead to outcomes that favor a particular political party. Recent count challenges to the practice have argued that through this calculated redistricting, large numbers of voters are being effectively (and intentionally) disenfranchised.

   Computers, it turns out, have been implicated as some of the main "villains" in much of the news coverage on this topic: it is only thanks to powerful software that gerrymandering grew from an activity carried out by a bunch of people with maps, pencil, and paper into the industrial-strength process that it is today. Why is gerrymandering a computational problem? Partly it's the database issues involved in tracking voter demographics down to the level of individual streets and houses; and partly it's the algorithmic issues involved in grouping voter into districts. Let's think a bit about what these latter issues look like.

   Suppose we have a set of n precincts $P_1, P_2, \ldots, P_n$, each containing m registered voters. We are supposed to divide these precincts into two *districts*, each consisting of $n/2$ of the precincts. Now, for each precinct, we have information on how many voters are registered to each of two political parties. (Suppose for simplicity that every voter is registered to one of these two.) We will say that the set of precincts is *susceptible* to gerrymandering if it's possible to perform the division into two districts in such a way that the same party holds a majority in both districts.

   Give an algorithm to determine whether a given set of precincts is susceptible to gerrymandering; the running time of your algorithm should be polynomial in n and m.

   (Example omitted.)

4. Let Σ be a finite alphabet. For two strings $x, y \in \Sigma^*$, define the *Insert-Delete distance* between x and y, denoted $\mathrm{ID}(x, y)$, to be the minimum number of insertions and deletions needed to convert x to y. For example if $\Sigma = \{a, b, c\}$, $x = abbc$ and $y = bbac$, then $\mathrm{ID}(x, y) = 2$ (we delete the first a and insert an a before the last c).

   (a) Give a polynomial time algorithm that on input two strings x, y, computes the distance $\mathrm{ID}(x, y)$ as well as a sequence of $\mathrm{ID}(x, y)$ insert/delete operations that can be used to convert x to y.

   (b) Let us say that a string $p = p_1p_2 \ldots p_k \in \Sigma^k$ is a substring of $z = z_1z_2 \ldots z_n \in \Sigma^n$ if there exist a sequence of indices $1 \le i_1 < i_2 < \cdots < i_k \le n$ such that for all j, $1 \le j \le k$, we have $z_{i_j} = p_j$. The *longest common substring* (LCS) of two strings x, y is the largest sequence L such that L is a substring of both x and y. Likewise, the *shortest common superstring* (SCS) of two strings x, y is the smallest sequence L such that both x and y are substrings of L.

   Design a polynomial time algorithm to find the LCS and SCS of two given strings. (Hint: It may be useful to figure out how, for two strings x, y, the quantities $\mathrm{ID}(x, y)$, length of $\mathrm{LCS}(x, y)$, and length of $\mathrm{SCS}(x, y)$ are related.)

5. (Chapter 6, Problem 29; somewhat tricky) Let $G = (V, E)$ be a graph with n nodes in which each pair of nodes is joined by an edge. There is a positive weight w_{ij} on each edge (i, j); and we will assume that these weights satisfy the *triangle inequality* $w_{ik} \le w_{ij} + w_{jk}$. For a subset $V' \subseteq V$, we will use $G[V']$ to denote the subgraph (with edge weights) induced on the nodes in V'.

We are given a set $X \subseteq V$ of k terminals that must be connected by edges. We say that a *Steiner tree* on X is a set Z so that $X \subseteq Z \subseteq V$, together with a spanning subtree of $G[Z]$. The *weight* of the Steiner tree is the weight of the tree T.

Show that there is a function $f(\cdot)$ and a *polynomial function* $p(\cdot)$ so that the problem of finding a minimum-weight Steiner tree on X can be solved in time $O(f(k) \cdot p(n))$.

Remark: Problems like above that admit an algorithm whose runtime is possibly non-polynomial only in a suitably identified size parameter k are called *fixed parameter tractable*. Such algorithms are very useful in efficiently solving instances of NP-hard problems which have a small value for the size parameter k. There is a whole theory of fixed parameter complexity that has been developed to formally address which problems might admit such algorithms.