Homework 3 is due at the start of class (2:40 pm) on Wednesday, October 28, 2009.

You may use algorithms learned in class as subroutines without re-explaining them. For any problem that requests an algorithm that runs in $O(f(n))$ time for some function $f(n)$, expected $O(f(n))$ time will do.

[1] **Levels in arrangements** (2 points). Is it possible for an arrangement of lines, no two parallel, to have a vertex of level 2 but no vertex of level 1? Explain.

[2] **Stabbing vertical segments** (4 points). Let $S$ be a set of $n$ vertical line segments in the plane. Describe an algorithm that determines whether there exists a line that intersects every segment in $S$, and identifies such a line if one exists. For full points, your algorithm should run in linear time.

[3] **Axis-aligned ray-shooting queries** (3 points). Let $S$ be a set of segments in the plane, which can intersect only at their endpoints. Explain how to preprocess $S$ in $O(n \log n)$ time so that queries of the following form may be answered in $O(\log n)$ time: given a point $p$, give the first segment struck by each of four rays shot from $p$ directly up, down, to the left, and to the right. The query should report four segments, in that order. If a ray hits an endpoint of multiple segments, report any one of those segments. If a ray goes on forever without hitting a segment, report that. If $p$ lies on a segment of $S$, report that segment for all four rays.

[4] **Point in star-shaped polygon** (8 points). A simple polygon $P$ is called *star-shaped* if there exists a point $p \in P$ such that for every point $q \in P$, the line segment $pq$ lies in $P$. Hence, $p$ can “see” any point in $P$ by a line of sight entirely in $P$. You are given a simple star-shaped polygon $P$, described as an array of $n$ vertices in counterclockwise order about $P$.

   (i) Suppose you know a point $p$ that can see all of $P$. Describe an algorithm that determines whether a point $q$ is in $P$ in $O(\log n)$ time (with no preprocessing).

   (ii) Suppose you don’t know a point $p$ that can see all of $P$. Describe an optimal algorithm for finding one, and give its running time. (Hint: this is a one-liner.)

   (iii) Explain why no algorithm can find one faster. (Hint: this is not a reduction from sorting. Show that the algorithm must examine the entire input, or at least some constant fraction of it. A few examples where small differences between two polygons lead to completely different answers will help your explanation.)

   (iv) Can your algorithm from part (ii) also determine whether a simple polygon $P$ is star-shaped? Does your algorithm extend to three-dimensional polyhedra? Briefly justify your answers to both these questions.

[5] **Packing disks** (5 points). Let $D$ be a set of disks of radius $r$ in the plane, which represent atoms. (A disk is a set of points consisting of the points on a circle and all the points inside it.) The disks in $D$ may partly overlap each other (due to molecular bonds). Let $R$ be an axis-aligned rectangle.

   We wish to determine whether it is possible to place another disk $d$ of radius $s$ (representing a catalyst) such that $d$ lies inside $R$ and does not overlap any disk in $D$. (It is okay if $d$ touches a disk in $D$, or the boundary of $R$, at a single point.)

   Give an $O(n \log n)$-time algorithm for doing this, where $n$ is the number of disks in $D$. (Hint: what data structure makes this computation straightforward?)
[6] **Worst-case time for linear programming** (2 points). If you have bad luck with the random numbers, the worst-case running time for Seidel’s randomized linear programming algorithm in the plane is $\Theta(n^2)$. Suppose you are given a fixed set of $n$ distinct halfplanes and a linear objective function, such that the linear program is not unbounded. Is there always a way to order these halfplanes so that the algorithm requires $\Omega(n^2)$ time? If so, explain how to construct an ordering (given any fixed set of $n$ half-planes). If not, give an example (that extends to arbitrarily large $n$) in which all orderings lead to an $o(n^2)$ run time.

[7G] **Linear programs with multiple optima** (6 points). A linear program can have an infinite number of solutions. In general, the set of solutions is a $k$-face of the feasible region with $0 \leq k < d$, where $d$ is the dimensionality of the space.

You are given a linear program in which the number of variables $d$ is small enough to use Seidel’s algorithm. The algorithm finds the lexicographically maximum solution in expected $O(n)$ time. Suppose you want to find all the solutions. Pretend that numerical robustness is not an issue.

(i) Give a linear-time algorithm that determines whether there is more than one solution, and if so, identifies two vertices of the solution $k$-face. (Hint: I know several answers to this question, one of which fits in one sentence.)

(ii) Generalize your solution to part (i) so that, if you know $i \leq k$ affinely independent vertices of the solution $k$-face, you can find another one in $O(n)$ time.

(iii) Once you’ve found $k+1$ affinely independent optima, you’ll notice that they all have some active constraints in common. Use this operation to describe how to compute the solution $k$-face in $O(n^{\lfloor k/2 \rfloor})$ time.

[8G] **Dividing points** (5 points). Let $R$ and $G$ be two sets of points in the plane, called the red and green points, respectively. Let $n = |R| + |G|$ be the total number of red and green points.

(i) Describe an algorithm that preprocesses $R$ and $G$ in $O(n^2)$ time (or better) so that we can answer the following query in $O(n)$ time: given a query point $p$, return a line that passes through $p$ and has the same number of red points on one side of it as the number of green points on the other side of it; or report that no such line exists. For this problem, you are restricted to the decision-tree model of computation. (It would be easy to answer the query with radix sort and no preprocessing—except that the slope of a line through two points, being a rational number, isn’t really amenable to exact radix sort.)

(ii) Describe an algorithm that preprocesses $R$ and $G$ in $O(n^2 \log n)$ time so that we can answer the following query in $O(\log n)$ time: given a query line $\ell$, determine whether $\ell$ has the same number of red points on one side of it as the number of green points on the other side of it. (You’ll get a **bonus point** if you can explain how to do it with just $O(n^2)$ preprocessing time, and prove that your preprocessing is that fast.)