CS 274
Computational Geometry (Spring 2003)
Homework 3

Homework 3 is due **at the start of class (3:30 pm) on Thursday, March 13, 2003**.

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

[1] Axis-aligned ray-shooting queries (4 points). Let S be a set of segments in the plane, which intersect only at their endpoints. Explain how to preprocess S in $\mathcal{O}(n \log n)$ time so that queries of the following form may be answered in $\mathcal{O}(\log n)$ 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 (for that ray). 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.

[2] Worst-case time for linear programming (2 points). 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 half-planes and a linear objective function, such that the linear program is not unbounded. Is there always a way to order these half-planes so that the algorithm requires at least $\Omega(n^2)$ time? If so, explain how to construct an ordering (given *any* fixed set of n half-planes). If not, give an example (for arbitrary n) in which all orderings lead to an $o(n^2)$ run time.

[3] Two-dimensional linear programming with preprocessing (3 points). Suppose you are given a set of linear constraints that define a feasible region that is bounded (and therefore is a convex polygon), and you are given the vertices V of the feasible region. Explain how to preprocess the feasible region so that, given any objective vector c, you can solve the linear program in $\mathcal{O}(\log n)$ time (not counting preprocessing time). Hint: you've (more or less) done this before.

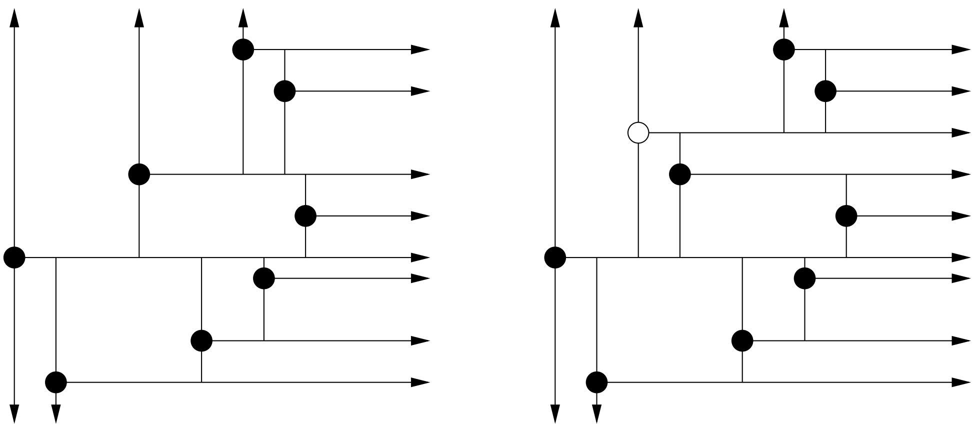
[4] Red-black closest pair (4 points). Let P and Q be two point sets in the plane, and let $p \in P$ and $q \in Q$ be two points from these sets that minimize the Euclidean distance $|pq|$. True or false: $\overline{pq}$ is an edge of DT$(P \cup Q)$. Explain.

[5] Stabbing vertical segments (3 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.

[6] Linear programs with multiple solutions (6 points). Suppose we want to find *all* solutions to a three-dimensional linear program with n constraints. A linear program can have an infinite number of solutions, of course, but we can express the set of solutions with a finite data structure.

(i) Describe an algorithm for finding these solutions, and explain how they might be expressed.
(ii) Argue that $\Omega(n \log n)$ is a lower bound for the worst-case complexity of any algorithm solving this problem in the decision-tree model of computation.

[7] An incremental algorithm (8 points). Given a set P of n points in the plane, where no two points have the same x- or y-coordinate, define a planar subdivision as follows. Imagine all the points are sorted lexicographically. For each point in this order, shoot one bullet up, one bullet down, and one bullet to the right until they hit existing segments, then add these three bullet-path segments/rays to the subdivision.



the subdivision after point insertion

(i) Give tight upper bounds on the number of vertices, edges, and faces of the subdivision. Vertices include points in P, segment intersections, and one "vertex at infinity." Faces include unbounded faces.

(ii) Describe an algorithm to add a new point to the subdivision and restore the proper subdivision structure. The new point may have an arbitrary x-coordinate (as long as it's different from the other points), but the subdivision must be updated as if the points were processed in lexicographic order. (See the figure.) Assume the subdivision is represented by a DCEL.

(iii) Prove that if a set of points are inserted in random order (starting from scratch), the expected number of structural changes to the subdivision is $\mathcal{O}(1)$ per insertion. Give the smallest upper bounds you can on the expected number of new vertices created, the expected number of old vertices eliminated, the expected number of new faces created, and the expected number of old faces eliminated during a vertex insertion.

(iv) Explain how to locate each point in $\mathcal{O}(\log n)$ expected time so it may be inserted. Prove that your method is that fast.