CS 274
Computational Geometry (Spring 2003)
Homework 4

Homework 4 is due **at the start of class (3:30 pm) on Thursday, April 24, 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] Levels in arrangements (2 points). Is it possible for an arrangement of n lines, no two parallel, to have no vertex of level $n-2$? Explain.

[2] Line-polygon intersection (4 points). You are given a convex polygon P, described as an array of n vertices in counterclockwise order. Assume you know the indices of the leftmost and rightmost vertices, and there is only one of each (i.e. there is no tie for least or greatest x-coordinate). Give an algorithm for testing in $\mathcal{O}(\log n)$ time whether an arbitrary line ℓ intersects P (with no preprocessing). (Hint: duality might help you better understand the problem.)

[3] Point in star-shaped polygon (8 points). A 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 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 $\mathcal{O}(\log n)$ 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. A few examples where small differences between two polygons lead to completely different answers will help your explanation.)
(iv) Give an algorithm that determines whether a simple polygon P is star-shaped. Does your algorithm extend to polyhedra?

[4] 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 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 $\mathcal{O}(n \log n)$-time algorithm for doing this. (Hint: what data structure makes this computation straightforward?)

[5] Counting problems in orthogonal range search (5 points). The orthogonal range search problem we have studied is the *reporting* problem, where we want to list all the points in a query box. Here we are interested in the *counting* problem, where we merely want to state how many points are in a query box. Call this number k. Since we are not listing the points, there is no $\Omega(k)$ lower bound on query time.

Suppose you have a two-dimensional layered range tree (with fractional cascading) representing a point set P in the plane. Without modifying the data structure, explain how you can report the number of points of P in a query box $B = [x, x'] \times [y, y']$ in $\mathcal{O}(\log n)$ time.

(For partial credit, explain how to report the number of points in $B = [y, y']$ from a 1-dimensional range tree in $\mathcal{O}(\log n)$ time.)

[6] Adding points to a convex hull (6 points). Let S and T be two sets of points, having s points and t points respectively. Suppose we are given the convex hull $\mathrm{conv}(S)$ expressed as a facet graph, as produced by the Clarkson–Shor incremental convex hull construction algorithm, and our task is to incrementally insert the points in T and thus construct $\mathrm{conv}(S \cup T)$.

We use the following algorithm: first, for each point p in T, find a facet of $\mathrm{conv}(S)$ visible from p (or verify that none exists) by brute force (checking each point against each facet), and thereby build a conflict graph in $\mathcal{O}(ts^{\lfloor d/2 \rfloor})$ time. Second, incrementally insert the vertices of T in random order (with each permutation being equally likely) using the Clarkson–Shor algorithm.

If we were constructing $\mathrm{conv}(S \cup T)$ from scratch, with the insertion order wholly randomized, it would take at worst $\mathcal{O}((s+t)^{\lfloor d/2 \rfloor}) = \mathcal{O}(s^{\lfloor d/2 \rfloor} + t^{\lfloor d/2 \rfloor})$ time. However, because S is *not* a random subset of $S \cup T$, the running time for this algorithm can be worse. An adversary could choose S in such a way that inserting the vertices of T is slow.

Prove that the expected running time to incrementally insert the vertices of T is in $\mathcal{O}(t^{\lfloor d/2 \rfloor} + ts^{\lfloor d/2 \rfloor} + ts^{\lfloor d/2 \rfloor - 1} \log t)$. (For asymptotic purposes, treat d as a small fixed constant.)

Some hints:

- You may safely assume that the time spent making structural changes does not exceed the time spent updating the visibilities (conflict graph), and analyze only the latter.
- The technique presented in class (and Section 9 of Seidel's paper) for analyzing the expected running time works here as well, with the following difference: the sets S and T might be chosen (by an adversary) so the uninserted vertices of T can see a larger-than-usual number of facets, and the inserted vertices of T have much higher average degree than the vertices of S. This is why the complexity can exceed $\mathcal{O}((s+t)^{\lfloor d/2 \rfloor})$.
- If you get the right summation at the end, it will include a binomial. Expand it (or at least the first few and last few terms) before trying to evaluate the summation. You may find the following facts helpful.
- $\sum_{i=1}^{n} \frac{1}{i} \in \mathcal{O}(\log n)$.
- $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$.