CS 274
Computational Geometry (Spring 2017)
Homework 5

Homework 5 is due **at the start of class (2:40 pm) on Monday, May 1, 2017**.

You may use algorithms learned in class as subroutines without re-explaining them.

[1] General-dimensional Voronoi diagrams (4 points). The worst-case complexity of a Voronoi diagram of n sites is $\Theta(n^2)$ in three dimensions and $\Theta(n^{\lceil d/2 \rceil})$ in E^d.

(i) Prove that a planar cross-section of a d-dimensional Voronoi diagram—in other words, the planar subdivision formed by intersecting a 2-flat with a Voronoi diagram—has complexity no greater than $\mathcal{O}(n)$. Hint: what do you know about the properties of Voronoi cells?
(ii) Does a planar cross-section of a d-dimensional Delaunay triangulation also always have a complexity in $\mathcal{O}(n)$? Explain.

[2G] Minkowski sums (6 points). Suppose we want to automatically machine a two-dimensional part by cutting away the region around it with a circular drill bit. Let D be a disk whose center is the origin and whose radius r is the radius of the drill bit. When machining a shape S, we may place the center of the drill bit at any point not in the interior of $S \oplus D$.

We wish to argue that for any n-vertex polygon P (not necessarily convex, possibly with polygonal holes), the worst-case complexity of $P \oplus D$ is in $\mathcal{O}(n)$. The complexity is the total number of vertices, line segments, and circular arcs needed to represent the boundary of $P \oplus D$. Line segments are induced by edges of P, offset sideways by the radius r of D. Circular arcs (of radius r) are induced by vertices of P.

As the boundary representation of $P \oplus D$ is a planar graph, it suffices to bound the number of vertices. There are three types of vertices that can appear: intersections between two line segments, intersections between two arcs, and intersections between a line segment and an arc.

(i) Demonstrate that a single edge of P can induce up to $\Theta(n)$ line segments in $P \oplus D$, and that a single vertex of P can induce up to $\Theta(n)$ circular arcs. (A couple of simple figures should suffice to illustrate these two facts—no words are necessary.)
(ii) An edge e of P can induce a sequence of line segments in $P \oplus D$, offset a distance of r from e. Suppose two line segments of $P \oplus D$, s_1 and s_2, intersect at a vertex v. Show that v terminates one of the sequences of line segments induced by some edge of P. Use this fact to show that $P \oplus D$ can have only $\mathcal{O}(n)$ vertices where pairs of line segments meet. (For clarity, please use "edge" to denote an edge of P, and "segment" to denote an edge of $P \oplus D$.)
(iii) Show that $P \oplus D$ can have only $\mathcal{O}(n)$ vertices where pairs of arcs meet. The easiest way to do this is to explain the relationship between these vertices and the edges of a Voronoi diagram.

Note: the number of segment-arc intersections can be bounded by the same reasoning as part (iii), by using the properties of the Voronoi diagram of the line segments. (See Section 7.3 of the third edition of the Dutch Book if you're curious what a Voronoi diagram of line segments is.)

[3] A three-dimensional BSP tree for point location (4 points). Let S be a set of n axis-aligned rectangular prisms in E^3 with disjoint interiors. We wish to build a BSP tree to perform point location queries: given a point $p \in E^3$, return a rectangular prism that contains p, or report that there is none. (If p lies on the boundaries of two or more rectangular prisms, it doesn't matter which one we return.) We choose a permutation of the prisms' $6n$ rectangular faces uniformly at random, then use that ordering of the faces to construct an autopartition. Show that for the BSP tree thus built,

(i) point location queries run in expected $\mathcal{O}(\log n)$ time and
(ii) the expected size of the tree is in $\mathcal{O}(n \log n)$.

[4G] Overlapping intervals (8 points).

(i) Let S be a set of n intervals. Describe a data structure that uses $\mathcal{O}(n)$ storage and returns the *number* of intervals in S that intersect any query interval I in $\mathcal{O}(\log n)$ time. (Note that I didn't say "$\mathcal{O}(k+\log n)$ time.") Hint: I can think of three very different ways to do this. One way uses two different types of tree; one way uses two of the same type. The simplest way doesn't use any fancy data structures at all.

(ii) Explain how to use a two-dimensional layered range tree to answer the same counting query, also in $\mathcal{O}(\log n)$ time. (Assume you have a correct solution to Problem [3] of Homework 4; there's no need to explain that solution again. Observe that a two-dimensional range tree requires $\Theta(n \log n)$ space, so this isn't an answer to part (i).)

(iii) Let D be a disk, and let L be a set of l lines in the plane. Describe an algorithm that returns, in $\mathcal{O}(l \log l)$ time, the number of pairs of lines that intersect each other inside D. For simplicity, you may assume that no two lines intersect each other precisely on the boundary of D. Hint: This can be done with either the answer to part (i) or the range tree in part (ii) as a subroutine, but the latter is easier. For **1 bonus point**, do it with only the former.

[5G] Point-rectangle pairs (8 points). Let P be a set of m real numbers, and let R be a set of n intervals. By building an interval tree or a segment tree on R, we can report all pairs $\langle p \in P, r \in R \rangle$ such that $p \in r$ in $\mathcal{O}(k + (m+n) \log n)$ time (including preprocessing), where k is the number of such pairs.

Observe that k could be as large as $\Theta(mn)$. Suppose that instead of reporting pairs individually, we are allowed to report pairs of subsets of the form $\langle \{p_1, p_2, \ldots, p_i\}, \{r_1, r_2, \ldots, r_j\} \rangle$, meaning that each point $p_1 \ldots p_i$ lies in every interval $r_1 \ldots r_j$. These *subset pairs* sometimes make it possible to express the results in $o(k)$ space, and always in $o(mn)$ space. Every intersecting real-interval pair $\langle p_i, r_j \rangle$ should appear in *exactly* one of the subset pairs in the output.

(i) Describe an algorithm that runs in $\mathcal{O}((m+n) \log n)$ time (including all preprocessing) and reports the real-interval pairs in $\mathcal{O}(\min\{k, (m+n) \log n\})$ space. (This is the amount of space taken by the subset pairs; the algorithm may use more space.)

(ii) Improve your answer so that it runs in $\mathcal{O}(k' + m + n \log n)$ time, where k' is the size of the output. Hint: you will need radix sort, and you will need to find a way to group points together when traversing the tree.

(iii) Let P be a set of m points in the plane, and let R be a set of n axis-parallel rectangles (possibly intersecting). We want to report all pairs $\langle p \in P, r \in R \rangle$ such that $p \in r$. Suppose there are k such pairs. Again we use subset pairs to reduce output size and running time. Describe an algorithm that reports the point-rectangle pairs in $k' \in \mathcal{O}(\min\{k, (m+n) \log^2 n\})$ space and runs in $\mathcal{O}(k' + m \log n + n \log^2 n)$ time. Explain the running time.
You'll get **2 bonus points** if you can do it in $\mathcal{O}(k' + m + n \log^2 n)$ time. (I don't actually know if this is possible. It would take very careful organization of the batches of points to avoid getting charged $\log n$ for each point that is contained in few or no rectangles.)