Due Date: October 27, 2005

Problem 1: Let $R$ be a set of $n$ rectangles in the plane. Describe an algorithm that reports all $k$ pairs of intersecting rectangles in time $O(n \log n + k)$ time.

(Hint: Use a sweep-line algorithm and maintain a segment tree.)

Solution by Mason Matthews. Since queries on a segment tree take $O(\log n)$ time, I will not use the segment tree as described in the notes. Instead, I will sweep a vertical line from left to right (in order of increasing $x$ coordinate) and maintain a binary tree that stores the horizontal edges of the rectangles.

First, sort the vertical edges of the rectangles by $x$-coordinate. This takes $O(n \log n)$ time. Next, consider them in increasing order. If a left vertical edge corresponding to rectangle $r_i$ is reached, add $r_i$'s horizontal (top and bottom) edges to the binary tree. Then use the vertical edge as a query on the tree. Output a pairing with any rectangle $r_j$ if one of $r_j$'s horizontal edges intersect the vertical query edge.

When the right edge of a rectangle is reached, remove its horizontal edges from the tree and repeat the query. Each query takes $O(\log n + k_i)$ time, where $k_i$ is the number of rectangles that intersect with the $i$th vertical edge. Since there are $2n$ total queries, the line sweep takes $O(n \log n + k)$ time. Therefore, the total running time of this algorithm is $O(n \log n + k)$.

Problem 2: Show that the space requirement of the 2-dimensional orthogonal range searching can be improved to $O(n)$, provided we allow query time to be $O(n^\epsilon)$, for any arbitrarily small constant $\epsilon > 0$. Of course, the constant of proportionality depends on $\epsilon$. What is the preprocessing time?

(Hint: Store the secondary structures only at certain levels of the primary tree.)

Solution. Create the the range tree as usual but do not construct secondary trees $T_{\text{assoc}}(v)$ at every node of the primary tree $T$. We create secondary trees for nodes at only a constant number of levels. Create secondary trees for nodes at every $1/\delta$ levels of $T$ where $\delta > 0$ is a constant.

The primary tree requires $O(n)$ storage and we create secondary trees at a constant number of levels with secondary trees on the same level requiring $O(n)$ storage in total. The total storage requirement is therefore $O(n)$. The preprocessing time remains the same at $O(n \log n)$.

There are two steps in a query. First, search for the $x$-coordinate interval of the range query in the primary tree $T$. This takes $O(\log n)$ time returning $O(\log n)$ nodes of $T$. For each node $v$ returned, we perform a query on the $y$-coordinate of the range query in the secondary structure $T_{\text{assoc}}(v)$ at $v$. However, $v$ may not contain a secondary structure so we must descend down the subtree $S_v \subseteq T$ rooted at $v$ until we find a level of $S_v$ with secondary trees. For each node $v$, we have to descend at most $\delta \log n$ levels. Once we have found the level containing secondary trees, we
have to perform a query on each of the nodes at this level. There are \(2^\delta \lg n = n^\delta\) nodes at this level each with a secondary tree of size \(O(n/n^\delta) = O(n^{1-\delta})\). A query at every node in this level takes time \(O(n^\delta \lg n^{1-\delta} + k_v)\) where \(k_v\) is the number of points returned. We do this \(\lg n\) times for each node \(v\) returned by the first step. The total query time is

\[
O(n^\delta \lg n \lg n^{1-\delta} + k) = O(n^\delta \lg n \lg n + k) = O(n^\delta + k) \text{ for sufficiently large } n = O(n^\delta)
\]

**Problem 3:** A circular disk of radius \(r\) centered at point \(c \in \mathbb{R}^2\) is the set \(D = \{x \mid \|x - c\| \leq r\}\). Let \(D = \{D_1, \ldots, D_n\}\) be a set of \(n\) circular disks in the plane. Let \(U\) be the union of the disks in \(D\). Show that \(U\) has \(O(n)\) vertices. Describe an algorithm for computing \(U\).

(Hint: Show that each \(D_i\) can be mapped to a halfspace \(H_i\) in \(\mathbb{R}^3\) so that each point in \(U\) maps to \(\cap_i H_i\).)

**Solution by Mason Matthews.** If we assume that the center of a disk is given by the pair \((\mu_x, \mu_y)\) and its radius by \(r\), then the formula for the disk is \((x - \mu_x)^2 + (y - \mu_y)^2 \leq r^2\). To represent this as a halfspace in \(\mathbb{R}^3\), we need to linearize the equation. If we define \(z = x^2 + y^2\) and perform some simple algebra, we have:

\[
(x - \mu_x)^2 + (y - \mu_y)^2 \leq r^2
\]

\[
x^2 - 2x\mu_x + \mu_x^2 + y^2 - 2y\mu_y + \mu_y^2 \leq r^2
\]

\[
z \leq 2x\mu_x + 2y\mu_y - \mu_x^2 - \mu_y^2 + r^2
\]

where \(r, \mu_x, \text{ and } \mu_y\) are constants. If this is the \(i\)th disk, then the inequality above defines \(H_i\).

Consider \(\cap_i H_i^C\), the intersection of the complements of all these halfspaces. This will be a convex polyhedron \(C\) which is unbounded above. Since a vertex in \(U\) is the intersection of two disk boundaries (say of disks \(j\) and \(k\)), it is represented in \(\mathbb{R}^3\) by the intersection of two hyperplanes. Let us call this intersection of the two hyperplanes line \(l_{ij}\). However, recall that our points must lie on the paraboloid \(z = x^2 + y^2\). This paraboloid can intersect with a given \(l_{ij}\) once, twice, or never (implying at most 2 intersections between a pair of circles). Since there are only \(O(n)\) edges on a convex polyhedron in \(\mathbb{R}^3\), there are \(O(n)\) intersections between circles, and therefore \(O(n)\) vertices in \(U\). It is also worth noting that the edges in \(U\) correspond to the intersection of the paraboloid with the faces of \(C\).

An algorithm for computing this union follows from these concepts. First, compute each of the \(H_i^C\) halfspaces and find their intersection. This is equivalent to computing the convex hull, which takes \(O(n \lg n)\) time in \(\mathbb{R}^3\). Next, for each edge in the convex hull, check for intersections with the parabola \(z = x^2 + y^2\). For each intersection, use the equation for the corresponding hyperplane to
map it back into \( \mathbb{R}^2 \). This will provide the vertices of \( U \), and this can be done in \( O(n) \) total time. Edges can be computed in \( U \) in \( O(n \lg n) \) time as well. For each vertex in \( U \) (point on an edge in \( C \)), consider the two faces in \( C \) surrounding it. For each face, the other edge intersecting the parabola can be found in \( O(\lg n) \) time. Following these pairs can yield the cycles in the union.

The total running time of this algorithm is \( O(n \lg n) \).

**Problem 4:** The farthest neighbor Voronoi diagram of a set \( S \) of points in \( \mathbb{R}^d \), denoted by \( \text{Vor}_f(S) \), is the decomposition of \( \mathbb{R}^d \) into maximal connected regions so that the farthest point of \( S \) from any point within each region (under the Euclidean metric) is the same.

(i) Show that \( \text{Vor}_f(S) \) in the plane is a tree.

**Solution by Mason Matthews.** Suppose that there exists a bounded region \( r \) in \( \text{Vor}_f(S) \). There exists a point \( s_i \in S \) such that all points in \( r \) are farther from \( s_i \) than any other points in \( S \). Choose any point in \( r \) and follow the line that moves directly away from \( s_i \). Since \( r \) is bounded, following this line will lead to a different region. This implies that another point \( s_j \) is now farther away than \( s_i \). However, since we increased our distance from \( s_i \) at the fastest possible rate, the distance from \( s_j \) could never have overtaken it. Therefore, our assumption is false, and there are no bounded regions in \( \text{Vor}_f(S) \). Since all regions are unbounded, there are no cycles in the graph of \( \text{Vor}_f(S) \). Therefore, \( \text{Vor}_f(S) \) is a tree.

(ii) What is the complexity of \( \text{Vor}_f(S) \) in \( \mathbb{R}^d \)?

**Solution.** Given \( n \) points \( S \) in \( \mathbb{R}^d \), linearize each point \( s = (s_1, \ldots, s_d) \in S \) to a hyperplane \( e(s) \) in \( \mathbb{R}^d \). The hyperplane \( e(s) \) is described as follows.

\[
x_{d+1} = 2s_1x_1 + 2s_2x_2 + \cdots + 2s_dx_d - s_1^2 - s_2^2 - \cdots - s_d^2
\]

Let \( E(s) \) denote the halfspace above \( e(s) \). The farthest neighbor Voronoi diagram is the vertical projection of \( \bigcap_{s \in S} E(s) \) onto the hyperplane \( x_{d+1} = 0 \). By the Upper Bound Theorem, the complexity of \( \text{Vor}_f(S) \) is \( O(n[^{[n/2]}]) \).