15-451/651 Algorithms, Fall 2017
Recitation #10 Worksheet

Points in Rectangles.

You are given a set P of n points (x_i, y_i) with integer coordinates and a set R of m axis-aligned rectangles specified by their four edges (i.e. you are given a_j, b_j, c_j, d_j, such that rectangle j consists of all points such that $a_j \leq x < b_j$ and $c_j \leq y < d_j$).

Give an algorithm that can determine the number of points in P that are contained in each rectangle in $O((n+m)\log m)$ time.

Solution: An important observation: we don't care what the particular y-coordinates of the points are, only their positions relative to the top and bottom of each rectangle (c_j and d_j). Therefore, we can use a technique called *index compression* to reduce the infinite y-axis into a set of at most $2m+1$ discrete buckets:

- Create a sorted list ℓ consisting of all values of c_j and d_j across all of the rectangles, removing duplicates. (This takes $O(m \log m)$ time to construct.)
- Define $\texttt{bucket}(y)$ as the minimum index i such that $\ell[i] > y$, or as $\texttt{length}(\ell)$ if y is larger than all of the elements of ℓ. (This takes $O(\log m)$ time since it can use a binary search.)

Now we can use a line-sweep approach. Construct a segment tree with $\texttt{length}(\ell)$ leaves, intially all 0, and an array A of length m (initially all 0) to store the results. Sort all of the points, rectangle left-sides, and rectangle-right sides by x-coordinate, breaking ties by putting rectangles before points but otherwise ordering arbitrarily. When we encounter a point (x, y), increment $\texttt{bucket}(y)$ in the segtree. When we encounter the left side of rectangle j, perform a range query to get the sum of the range $[\texttt{bucket}(c_i), \texttt{bucket}(d_i))$, and subtract the result from $A[j]$. When we encounter the right side of rectangle j, perform the same range query again, and this time add the result to $A[j]$. At the end of the sweep, $A[j]$ holds the number of points in rectangle j.

Width of a Set of Points.

You are given a set of n points on a plane. Define a *strip* as the area between two parallel lines, and the *width* of that strip to be the perpendicular distance between those two lines. In $O(n \log n)$ time, find a strip of minimum width that contains all of the points.

Solution: Note that we need only consider the convex hull of the points (since a strip containing the convex hull necessarily contains all of the points inside of that hull). Additionally, we need only consider strips that touch at least three points on the hull; any solution that has only two points touching the lines can be improved by rotating until three points are touching.

This gives a fairly natural algorithm: for each edge of the convex hull, determine the maximum distance across all of the points from the line containing this edge; this is the minimum strip width parallel to this edge. Taking the minimum across all of these options solves the problem. However, this takes $O(n^2)$ time.

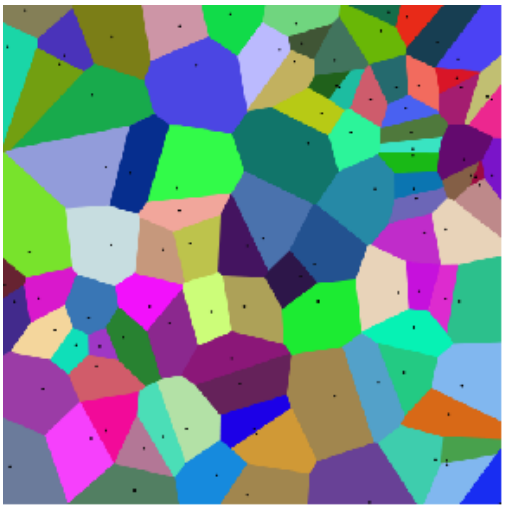
To go faster, we can make better use of the fact that the convex hull is convex. If we select an edge to be our "base" and rotate the hull such that the base edge is on the x-axis with the hull above it, we note that the y-coordinates of the remaining points form a convex function (which comes directly from the fact that the hull is convex). If we process the edges in order and keep track of the point furthest from the previous edge, we can reduce our scan around the polygon to linear time:

- Select some edge to go first, and determine the point furthest from this edge.
- Consider the next clockwise-adjacent edge. Note that the furthest-away point on the hull is either the same point as before or some point clockwise from that point. While the distance increases as we move our pointer clockwise, do so. (Correctness of this step is guaranteed by convexity; linear runtime is guaranteed by the fact that we will go around the entire polygon at most once.)
- Repeat until all edges have been considered.

This algorithm is $O(n \log n)$ to compute the convex hull and $O(n)$ to scan around the polygon, and is thus $O(n \log n)$ in total.

Painted Voronoi Diagram.

You are given a set P of n distinct points $p_1, p_2, \ldots p_n$ and n distinct colors $c_1, c_2, \ldots c_n$. In a *painted voronoi diagram*, the region of the unit square that is closer to point p_i than any other point in P is painted with color c_i. For example:



Consider the following painting algorithm:

> **Painting Algorithm:** Insert the points one at a time in order. When a point is inserted, paint the region of the unit square closer to this new point than any other point inserted so far.

Note that one unit of paint is needed to paint the entire square.

1. Devise a sequence of n points and prove that for this sequence the painting algorithm uses $\Omega(n)$ paint.

   Solution: Let $p_i = (\frac{1}{2}, \frac{i}{2n})$. When the points are added in order, each added point repaints at least the bottom half of the square. Thus, the algorithm uses at least $\frac{n}{2} \in \Omega(n)$ units of paint.

2. Suppose that we randomly permute the points before applying the painting algorithm. Show that the expected amount of paint used by this algorithm is at most $1 + \ln n$.

   Solution: We use backward analysis. Remove the points one at a time at random. When there are i points left, the area that is repainted when we remove the point is $\frac{1}{i}$ in expectation, since the total area of 1 is shared disjointly amongst the i points. Therefore, the total paint used is

$$\sum_{i=1}^{n} \frac{1}{i} = H_n \leq 1 + \ln n$$