Solutions to the Midterm Sample Problems

Solution 1:

(a) The factor 12 comes from the product of 4 and 3. The probability that a trapezoid is destroyed in the $i$th insertion is $4/i$, since each trapezoid is defined by at most four segments. When a trapezoid is destroyed, its leaf node is replaced by a search subtree of depth at most 3.

(b) The fact that $\ell$ intersects the triangle means that it lies above at least one point and below at least one. In the dual plane, this means that point $\ell^*$ lies in the region between the upper and lower envelopes of the three dual lines. Thus it lies in the shaded region showed in the figure below.

(c) The point $d$ lies in the interior of the triangle $a, b, c$ if and only if $Orient(a, b, d), Orient(b, c, d)$ and $Orient(c, a, d)$ are all positive.

(d) Observe that through the introduction of $h$ narrow “corridors,” each consisting of two parallel edges, we can connect the holes (directly or indirectly) to the exterior boundary, resulting in a simple polygon without holes and with $n + hk + 2h$ edges. Applying the result for simple polygons, we can triangulate this with $n + hk + 2h - 2$ triangles, after which we can remove the corridors.

(e) Excluding the leftmost trapezoid, charge each trapezoid to its leftmost vertex. Each of the $n$ segment left endpoints will be charged by two trapezoids (one above right and one below right) and each of the $n$ segment right endpoints will be charged by one trapezoid. The total number of charges is $3n$. Adding in the leftmost trapezoid implies that the total number is $3n + 1$.

(f) Charge each trapezoid to the vertex bounding its left wall. Each of the $I$ intersection points will be charged three times, one for the trapezoid above right, one for the trapezoid below right, and one for the trapezoid in between them. Thus, the total number is $3n + 3I + 1 = 3(n + I) + 1$. 

1
Solution 2:

**Alternative 1:** Sort the points by the $x$-coordinate. While there are points in $P$, compute the convex hull in $O(n)$ time using Graham scan, and remove the convex hull points from $P$. In the worst case, if there are three points in each slice, Graham scan is executed $\lceil n/3 \rceil$ times. Thus, the total time is $O(n \log n)$ for sorting, plus $O(n)$ for each of the $O(n)$ executions of Graham scan. The total time is $O(n \log n + n^2) = O(n^2)$.

**Alternative 2:** While there are points in $P$, compute the convex hull using Jarvis march, and remove the convex hull points from $P$. Let $h_i$, for $i$ from 1 to $k$ be the number of points in the $i$-th nested hull. Ignoring constant factors, the total running time is

$$\sum_{i=1}^k nh_i = n \sum_{i=1}^k h_i = O(n^2).$$

Solution 3:

(a) The maximum number of intersections is at most $2n_m$. To see this, observe that each of the $n_m$ edges of the monotone chain can intersect at most two edges of the convex chain. This bound is tight. To see this, consider a vertical segment that intersects the convex chain in two places. We can replace this vertical segment with an $x$-monotone chain that is arbitrarily close to it, which zig-zags nearly vertically up and down.

(b) The number of intersections is at most $n - 1$. To see this, consider $n$ vertical lines passing through each vertex of the two chains. This subdivides the plane into $n + 1$ vertical slabs. Ignoring the first and last (semi-infinite) slabs, the remaining $n - 1$ slabs can contain at most one segment of $P$ and one of $Q$. There can be at most one intersection in each slab, for a total of $n - 1$ intersections.

(c) No. They may intersect $\Theta(n^2)$ time, as shown in the following figure. Consider one chain zig-zagging up and down and another chain zig-zagging left and right.

Solution 4: This can be solved by linear programming in $O(n)$ time (randomized, in the expected case). Let $p_1, p_2, \ldots, p_n$ denote the upper endpoints of these segments and let $q_1, q_2, \ldots, q_n$ denote the lower endpoints. For such a line to exist it must be that all $p_i$’s lie above the line and the $q_i$’s lie below. Letting $y = ax + b$ denote the desired line equation, we have the following $2n$ linear constraints:

$$p_{i,y} \geq ap_{i,x} + b, \quad q_{i,y} \leq aq_{i,x} + b \quad \text{for } 1 \leq i \leq n.$$

We can use any objective function we want. If the resulting 2-dimensional LP, with variables $a$ and $b$ is feasible then such a stabbing line exists and otherwise it does not. This LP can be solved in $O(n)$ time.

Solution 5:
(a) By Euler’s formula we have $n - e + f = 2$, where $e$ is the number of edges and $f$ is the number of faces. Since $f$ includes the external face, we have $f = t + 1$. If we take each triangle and multiply by three, we count every edge on the convex hull once, and every edge inside the hull twice. Since there are $h$ edges on the hull, this implies that $3t = 2e - h$ or equivalently $e = (3t + h)/2$. Substituting into Euler’s formula we now have $n - \frac{3t + h}{2} + (t + 1) = 2$. Solving for $t$ we get $t = 2n - h - 2$.

(b) This algorithm is a simple greedy algorithm that works by plane sweep. First we sort the points according to their $x$-coordinates, and then insert them one-by-one into the triangulation. The main invariant is that after the first $i - 1$ points have been inserted, we have triangulated their convex hull. We maintain the vertices along the right chain of the hull. As the $i$th point is added, we join it to every visible point on the current hull by a line segment. We first join $p_i$ to the previous point $p_{i-1}$ (note that this point must be on the hull), and then scan both clockwise and counterclockwise connecting $p_i$ to successive points until we reach a tangent point on each side of the hull. As each new connection is added, we can update the associated representation in constant time. As in the analysis of Graham’s scan, the time to process each point is proportional to the number of triangles created. By Euler’s formula this is $O(n)$. So the total time is dominated by the $O(n \log n)$ needed to sort the points. See the figure below.

Solution 6: Let us make the usual general position assumptions. The $O(n)$ solution operates by linear programming in the plane. The key observation is that the two convex hulls do not intersect then there is a line $y = ax - b$ that separates them. Either the red points lie entirely above this line and the blue points beneath, or vice versa. We show how to test the first condition in $O(n)$ time (and the other part is symmetrical). The unknowns are $a$ and $b$. Each point $(x_i, y_i)$, in the red set must satisfy $y_i \geq ax_i - b$, and each point in the blue set must satisfy $y_i \leq ax_i - b$. This defines $n$ linear inequalities, that is, $n$ halfplanes in the $(a, b)$ coordinate plane. Any feasible point $(a, b)$ gives the coefficients of the separating line. Such a feasible point can be found in $O(n)$ time by linear programming. If a feasible point is found then the convex hulls do not intersect. If no feasible point exists, we repeat the process, but this time determining whether there is a line such that the blue points lie above it and the red points lie beneath it. Again, if there is such a line, the hulls do not intersect. Otherwise, we infer that it is impossible to separate the two hulls by a line, and this implies that the hulls overlap one another.

Solution 7: Recall that the edges of $P$ are oriented in counterclockwise order around $P$’s boundary. For each edge $e_i$, let $h_i$ denote the halfplane whose supporting line passes through this edge, and which lies to the left of this directed edge. Let $H$ denote the resulting set of $n$ halfspaces. (Each can be expressed as $\text{orient}(v_i, v_{i+1}, q) > 0$.) We assert that $P$ is star-shaped if and only if the intersection of the halfspaces of $H$ is nonempty, and $q$ may be taken to be any point in the
interior of this intersection.\textsuperscript{1} Assuming this is true, it follows that we can test this in \( O(n) \) time by reduction to linear programming. (Since this just a feasibility test, the objective function is arbitrary.)

To prove the assertion, we claim that a point \( q \) can “see” every point \( p \) on the boundary of \( P \) if and only if \( q \) lies in the interior of the intersection of the halfspaces of \( H \). Clearly, if \( q \) lies outside any halfspace of \( h_i \in H \), it cannot see the edge \( e_i \) (because then \( q \) would lie “behind” \( e_i \)). Conversely, if a point \( q \) lies within the intersection of halfplanes of \( H \) then we claim that for any point \( p \) on the boundary of \( P \) the open line segment \( \overline{qp} \) lies entirely within the interior of \( P \). Suppose not. By hypothesis, \( q \) must lie within the halfspace associated with the edge on which \( p \) lies. Since both \( p \) and \( q \) lie within \( P \), the segment \( \overline{qp} \) must cross the boundary of \( P \) at some last point \( r \) before \( p \). Let \( e_i \) be the edge on which \( r \) lies. Clearly \( p \) can see \( e_i \), and so \( p \) lies within \( h_i \). But, this implies that \( q \) does not lie within \( h_i \), contradicting the hypothesis that \( q \) lies in the intersection of \( H \).

\begin{center}
\textbf{Solution 8:} \hspace{1em} For \( 2 \leq i \leq n \), consider the resulting rectangle just after the \( i \)-th point has been inserted. We will compute a bound on the probability that any one of the four “then” clauses has just been executed. Consider the “then” clause in step 3(a). Any of the \( i \) points so far are equally likely to have been the last point added. The last point will cause this statement to be executed if and only if it has the smallest \( x \)-coordinate among the first \( i \) points. The probability that a randomly chosen point has then smallest \( x \)-coordinate is \( 1/i \). Thus, the probability that the last point caused this “then” clause to be executed is \( 1/i \). The expected number of times that this statement is executed over the entire course of the algorithm is

\[ \sum_{i=2}^{n} \frac{1}{i} \approx \ln n. \]

By symmetry, the total number of times all four “then” clauses are executed is roughly \( 4 \ln n \) which is \( O(\log n) \).

\textbf{Solution 9:}

(a) Let \( \ell_0 : ax - b_0 \) and \( \ell_1 : ax - b_1 \) be the lower and upper lines defining the strip, respectively. Let \( p_1, \ldots, p_n \) denote the points. Since the two lines have the same slope, their duals share the same \( a \) coordinate, and hence they define a vertical segment in the dual plane. The width of the strip is \( b_1 - b_0 = w \), which is the length of vertical line segment. All the points lie above \( \ell_0 \) and below \( \ell_1 \), which by the order-reversing property implies that each of the resulting \( n \) dual lines passes below \( \ell_0 \) and above \( \ell_1 \).

\textsuperscript{1}It might seem at first that the assertion is trivial, but if we changed things slightly so that \( P \) is an open polygonal curve, the assertion would be false.
(b) Let the bottom line be given by the equation \( y = ax - b_0 \) and the upper line be given by the equation \( y = ax - b_1 \). Thus our goal is to find \( a, b_0 \) and \( b_1 \). Each point \( p_i = (x_i, y_i) \) must lie within the strip from which we have the \( 2n \) linear constraints

\[
y_i \leq ax_i - b_1 \quad y_i \geq ax_i - b_0.
\]

These are linear constraints in \( a, b_0 \) and \( b_1 \). We want to minimize the width of the strip, which is equivalent to minimizing the linear function \( b_1 - b_0 \). Thus, this is a linear programming problem involving \( 2n \) constraints, and linear objective function. This can be solved in \( O(n) \) expected time using the randomized LP algorithm.

Solution 10:

(a) Each insertion gives rise to three bullet paths, thus creating three new edges and splitting three existing edges. Since we start with four edges for the bounding rectangle, the total number of edges in the final subdivision is \( 6n + 4 \). The insertion of each point creates three new faces and replaces one for a net increase of two, so there are \( 2n + 1 \) (internal) faces. By Euler’s formula there are \( 4n + 4 \) vertices.

(b) First locate the rectangle of the subdivision containing the new point. (This can be done by a history graph approach.) Insert two horizontal edges connecting the point to the left and right vertical sides of this rectangle. Then walk a bullet path up from this point to the top edge of the rectangle. For every horizontal edge that it intersects, trim this edge either on the left or right side, depending on where the originating point is located, so it terminates at this vertical line.

(c) We will describe only the modifications to a standard backward analysis. Observe that the number of structural changes is proportional to the number of new faces created. Let’s say that a face \( \Delta \) depends on point \( p_i \), if the insertion of \( p_i \) last would have caused \( \Delta \) to be created. Our goal is to count the average number of faces that depend on each point. Instead, we count the average number of points that each face depends on. Each face is bounded by at most four bullet paths, and the points giving rise to these bullet paths are the points on which this face depends. The total number of faces is \( 2n + 1 \), implying that the total number of face-point dependencies is at most \( 8n + 4 \). Thus, the average number of face dependencies per point is at most \( 8 + (4/n) \), which is roughly 8 for large \( n \).

Solution 11: The algorithm is based on a plane sweep. We process the points in order from left to right. At each point we maintain a list of indices of vertical segments that are visible from
$x = +\infty$. Observe that these segments form a monotonically decreasing staircase, which we store in a stack.

On processing the $i$th segment, whose upper endpoint is $y_i$, we pop the stack zero or more times until the top of the stack has an index $j$ such that $y_j > y_i$. We set the $i$th bullet label to $j$, and we push $i$ onto the stack. Because all the segments that were popped had smaller $y$-coordinates than $y_i$, it follows they are now all hidden from $x = +\infty$ by the $i$th segment, and hence the invariant is maintained. Each segment is pushed onto the stack once and can be popped from the stack at most once, and so the total running time is $O(n)$. (Observe that the actual values of the $x$- and $y$-coordinates were not relevant to the solution, only their relative orders.)

**Solution 12:**

(a) There are a number of solutions. The key is to find three vertices whose orientation is the same as the boundary orientation. One approach is to find three vertices on the convex hull. This can be done, for example by finding the leftmost and rightmost vertices, call them $p$ and $q$, and then finding the vertices that maximize the distance both above and below the line $pq$. This procedure is guaranteed to produce at least three distinct points (possibly four), all of whom are on the hull. (Notice that the simpler alternative of computing the points with the minimum and maximum $x$-coordinates and minimum and maximum $y$-coordinates may generally only produce two distinct points.)

Our approach is a bit easier. We compute the leftmost vertex (that is, the one with the minimum $x$-coordinate) in $O(n)$ time. It is easy to see that this vertex has an internal angle that is less than 180 degrees (assuming general position). Let $q$ be this vertex and let $p$ be its immediate predecessor and let $r$ be its immediate successor. Then the orientation of the polygon is given by the orientation of this triple, $O_r(p, q, r)$, which is computable in $O(1)$ time.

(b) As with part (a) we begin by computing the leftmost vertex of the polygon. As before, let $q$ denote this vertex and let $p$ and $r$ denote its immediate predecessor and successor, respectively. In $O(n)$ time we determine whether any other vertex of the polygon lies within the triangle $\triangle pqr$. If not, then $\overline{pr}$ is the desired diagonal. Otherwise, among all such vertices, select the one whose $x$-coordinate is closest to $q$’s. Let $s$ denote this vertex. We assert that $\overline{qs}$ is the desired diagonal. To see this, suppose to the contrary that some edge $\overline{tu}$ intersected this segment. Then either $t$ or $u$ would have a smaller $x$-coordinate than $s$, and (because the polygon is simple) this vertex would lie within the triangle $\triangle pqr$, contradicting the choice of $s$.

**Solution 13:**
(a) Let $m(i)$ denote the probability that $p_i$ is maximal. This point is maximal if and only if $p_i$ has the largest $y$-coordinate among all the points $\{p_i, p_{i+1}, \ldots, p_n\}$, which have larger $x$-coordinates. This set contains $n - i + 1$ elements. Since the $y$-coordinates are independent of the $x$-coordinates, the probability that any one of these points (and $p_i$ in particular) has the largest $y$-coordinate is $1/(n - i + 1)$. Therefore, $m(i) = 1/(n - i + 1)$.

(b) Let $E(n)$ denote the expected number of maxima from a set of size $n$, and let $m(i)$ denote the probability that $p_i$ is maximal. If we consider all $n$ points, with probability $m(i)$ we get 1 more maximal point and with probability $1 - m(i)$ we do not. Thus, the expected number of maximal points is given by the formula

$$E_n = \sum_{i=1}^{n} (m(i) \cdot 1 + (1 - m(i)) \cdot 0) = \sum_{i=1}^{n} m(i) = \sum_{i=1}^{n} \frac{1}{n - i + 1}.$$

By reversing the order of summation and using basic facts about the Harmonic series we have

$$E_n = \sum_{i=1}^{n} \frac{1}{n - i + 1} = \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1).$$

Therefore, the expected number of maxima is very nearly $\ln n$, which is $O(\log n)$. 

7