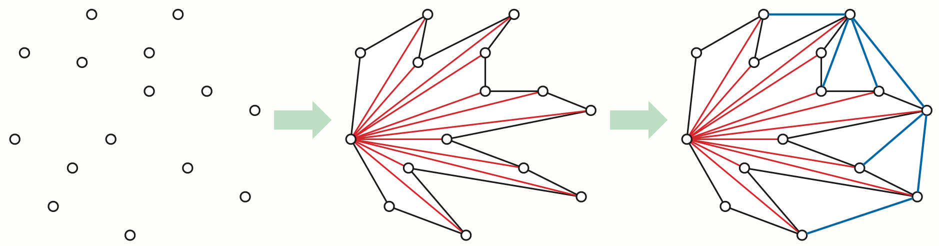
1. Describe and analyze an algorithm to compute a triangulation of a given set of n points in the plane. For full credit, your algorithm should run in $O(n \log n)$ time.

Solution: We make two modifications to the "Graham's scan" convex hull algorithm:

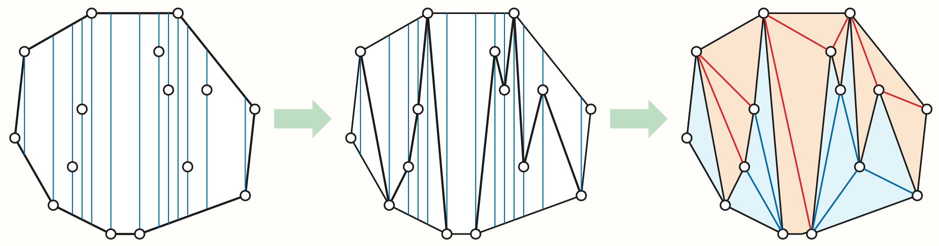
- After sorting around the leftmost point p_ℓ to build the original star-shaped polygon Q, we triangulate Q with diagonals from p_ℓ to every other vertex.
- Whenever we remove a reflex vertex q from Q, we add a diagonal between the predecessor and successor of q.

The algorithm runs in $O(n \log n)$ time.



■

Solution: Compute the convex hull of the input points, treat the interior points as degenerate holes, and use the algorithm from Homework 2.1 to triangulate polygons with holes. The algorithm runs in $O(n \log n)$ time.



More explicitly: After computing the convex hull of P, we proceed as follows. First decompose the convex hull into trapezoids by vertical lines through each point in P, using a standard sweep algorithm. Then insert a diagonal inside any boring trapezoid; these boring diagonals decompose the convex hull of P into monotone mountains with total complexity $O(n)$. Finally, we triangulate the resulting monotone mountains using the three-penny algorithm.

(This algorithm is morally equivalent to a variant of Graham's scan that connects the points in order from from left to right, and then uses the 3-penny algorithm to compute the upper and lower convex hulls.) ■

Rubric: 10 points. These are not the only correct solutions.

2. Suppose you are given a set of n "pyramids" in the plane. Each pyramid is a right isosceles triangle whose short edges have slope ± 1 and whose long edge lies on the x-axis. Each pyramid is represented by the x- and y-coordinates of its topmost point. The *silhouette* of these pyramids is the boundary of the union of the pyramids and the halfplane $y \le 0$.

   (a) Describe and analyze an algorithm that determines which pyramids are visible on the silhouette.

   Solution: Rotate the world 1/8 turn clockwise, or equivalently, multiply all coordinate vectors by the matrix $\left(\begin{smallmatrix} 1 & 1 \\ -1 & 1 \end{smallmatrix}\right)$. In the new coordinate frame, compute the Pareto-optimal points in $O(n \log n)$ time, using the algorithm from Homework 1.1. ■

   Rubric: 7 points. Yes, this is enough detail for full credit. No penalty for implicitly assuming all input points lie above the x-axis. This is not the only correct solution.

   (b) *Briefly sketch* and analyze an algorithm to compute the left-to-right sequence of silhouette vertices, including the vertices between pyramids and on the ground, given the output of your algorithm from part (a).

   Solution: Assume the tops of the visible pyramids are given an array sorted from left to right. Between the peaks of two adjacent pyramids, the silhouette has either two vertices (on the x-axis, at the base of each pyramid) or only one (where the right side of pyramid i intersects the left side of pyramid $i+1$). We can distinguish those two cases and compute the intermediate vertex or vertices in $O(1)$ time. The entire algorithm runs in $O(n)$ time. ■

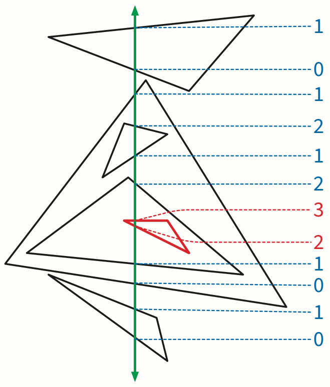
   Rubric: 3 points.

3. Suppose you are given a set T of n triangles, none of whose edges intersect, but some of which may be nested. Describe and analyze an algorithm to compute the nesting depth of T. For full credit, your algorithm should run in $O(n \log n)$ time.

Solution (sweep and maintain depths): We modify the standard Shamos-Hoey sweep algorithm for detecting segment intersections. As in that algorithm, we store the sequence of intersections between the vertical sweep line ℓ and the triangles in a balanced binary search tree.

Every node v in the binary search tree stores an integer $v.depth$, which is the number of triangles containing the point just below the crossing stored at v. Whenever the sweep line ℓ passes the leftmost vertex of a triangle, we insert two adjacent crossings v and w (with v above w) into the binary search tree, and we assign their depths as follows:

- If v has no successor (so v is the highest crossing), we set $v.depth \leftarrow 1$; otherwise, we set $v.depth \leftarrow succ(v).depth + 1$.
- In both cases, we set $w.depth \leftarrow v.depth - 1$.



Inserting v, inserting w, and computing the successor of v each take $O(\log n)$ time, after which computing the depths take $O(1)$ time. All other details of the sweep algorithm are unchanged.

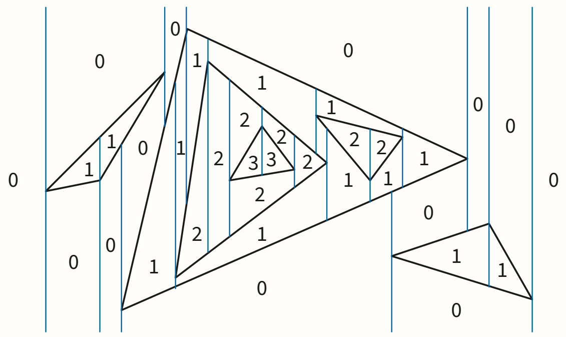
At the end of the sweep, the algorithm returns the largest depth ever assigned to any node in the binary search tree. The entire algorithm runs in $O(n \log n)$ time. ■

Solution (trapezoidalize and traverse dual graph): Our algorithm works in two stages. In the first stage, we construct a trapezoidal decomposition G of the $3n$ edges of T, using the sweep algorithm described in class and used in Homework 2.1. Alternatively, we could also use the randomized incremental algorithm described in class last week. Both algorithms run in $O(n \log n)$ time.

In the second stage, we compute the depth of every trapezoid in the decomposition G by traversing its dual graph G^*, which is implicitly represented by the doubly-connected edge list of G. Specifically, the leftmost unbounded face of G has depth 0, and for every pair of trapezoids t and t' that share an edge of G, we have

$$depth(t) = \begin{cases} depth(t') & \text{if } t \cap t' \text{ is part of a vertical wall} \\ depth(t')+1 & \text{if } t \cap t' \text{ is part of a triangle } \triangle \text{ with } t \text{ inside } \triangle \\ depth(t')-1 & \text{otherwise} \end{cases}$$

We can distinguish between these three cases in $O(1)$ time. Thus, we can compute the depth of every trapezoid in $O(n)$ time using a whatever-first-search of the dual graph G^*, starting at (the node dual to) the leftmost unbounded trapezoid.



Finally, the algorithm returns the maximum depth assigned to any trapezoid. The entire algorithm runs in $O(n \log n)$ time. ■

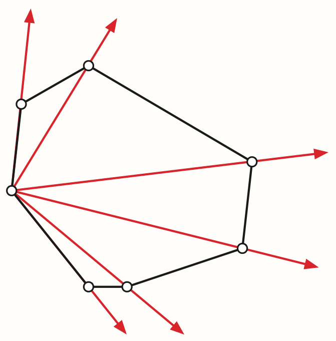
Rubric: 10 points. These are not the only correct solutions. This was intended to be the hardest question on the exam.

4. Describe and analyze an algorithm that determines whether a given point q lies in the interior of a convex polygon P. The input polygon P is represented as an array of vertices in counterclockwise order. For full credit, your algorithm should run in $O(\log n)$ time.

Solution (wedges): If the input point q lies inside P, it must also lie inside the triangle with vertices $P[1]$, $P[i]$, and $P[i+1]$, for some unique index i. We can find this index i as follows:

- First check that q is inside the wedge bounded by the rays $P[1]{\rightarrow}P[2]$ and $P[1]{\rightarrow}P[n]$ using two orientation tests. If q is outside this wedge, return FALSE.
- Then find the unique index i such that q is between the rays $P[1]{\rightarrow}P[i]$ and $P[1]{\rightarrow}P[i+1]$, using a variant of binary search that uses orientation tests on triples $(P[1], P[i], q)$ instead of simple comparisons.
- Finally, to check whether q is inside the triangle $P[1], P[i], P[i+1]$, perform one more orientation test on the triple $(P[i], P[i+1], q)$.

The algorithm runs in $O(\log n)$ time.



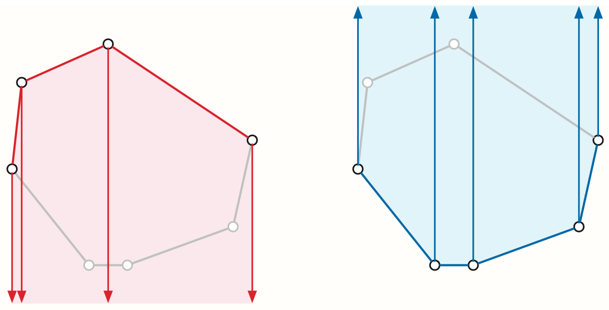
■

Solution (implicit trapezoidal decomposition): First we find the rightmost vertex $P[r]$ of P using a variant of binary search, as in Chan's algorithm and in Homework 1.3.

If either $q,x < P[1].x$ or $q.x > P[r].x$, we can immediately return FALSE.

The vertices of the lower hull $P[1\,..\,r]$ are sorted from left to right. We perform a binary search to find an index $1 \le i \le r-1$ such that $P[i].x \le q.x \le P[i+1].x$. Then if the triple $P[i]$, $P[i+1]$, q is oriented clockwise, we return FALSE. Otherwise, we know that q is above the lower hull of P.

The vertices of the upper hull $P[r\,..\,n]$ are sorted from right to left. Finally, we perform a binary search to find an index $r \le i \le n$ such that $P[i].x \ge q.x \ge P[i+1].x$. Then if the triple $P[i]$, $P[i+1]$, q is oriented clockwise, we return FALSE; otherwise, we return TRUE.



■

Rubric: 10 points. This is not the only correct solution.