

1. (a) **Warmup, not for submission:** Prove that  $\text{depth}(q, P) > 0$  if and only if  $q$  lies in the convex hull of  $P$ .

**Solution:** The following statements are all equivalent.

- $\text{depth}(q, P) = 0$ .
- $\text{depth}(\ell, P) = 0$  for some line  $\ell$  that contains  $q$  (by definition of  $\text{depth}$ ).
- $P$  lies in an open halfplane whose boundary line  $\ell$  contains  $q$  (by definition of  $\text{depth}$ ).
- $P$  lies in a closed halfplane that does not contain  $q$  (whose bounding line  $\ell'$  is parallel to  $\ell$  but closer to  $P$ ).
- The convex hull of  $P$  does not contain  $q$  (because the convex hull of  $P$  is the intersection of all halfplanes containing  $P$ .)

■

- (b) Describe and analyze an algorithm that computes  $\text{depth}(q, P)$ , given the point  $q$  and point set  $P$  as input.

**Solution:** Consider depth through the lens of point-line duality. For simplicity, I will assume that no two points in  $P \cup \{q\}$  lie on a vertical line and no three points in  $P \cup \{q\}$  lie on a common line.

For any dual point  $\ell^*$ , let  $\text{above}(\ell^*)$  and  $\text{below}(\ell^*)$  denote the number of lines in  $P^*$  that lie strictly above and below  $\ell^*$ , respectively. We immediately have  $\text{depth}(\ell, P) = \min\{\text{above}(\ell^*), \text{below}(\ell^*)\}$ . We can compute  $\text{above}(\ell^*)$  and  $\text{below}(\ell^*)$  by brute force in  $O(n)$  time.

Without loss of generality, suppose the dual line  $q^*$  is horizontal. Imagine translating a point  $\ell^*$  along  $q^*$  from left to right. Each time  $\ell^*$  crosses a line in  $P^*$  with positive slope,  $\text{above}(\ell^*)$  increases by 1 and  $\text{below}(\ell^*)$  decreases by 1. Symmetrically, each time  $\ell^*$  crosses a line in  $P^*$  with negative slope,  $\text{above}(\ell^*)$  decreases by 1 and  $\text{below}(\ell^*)$  increases by 1.

Thus, after sorting the intersection points of lines in  $P^*$  with  $q^*$ , we can compute  $\text{depth}(\ell, P)$  for every point  $\ell^* \in q^*$  in  $O(n)$  time. The entire algorithm takes  $O(n \log n)$  time. ■

- (c) Describe and analyze an algorithm that determines whether there is a point  $q$  such that  $\text{depth}(q, P) \geq k$ , given the set  $P$  and the integer  $k$  as input.

**Solution:** Define the *level* of each face of the arrangement of  $P^*$  as  $\text{above}(\ell^*)$  for any point  $\ell^*$  inside the face. For any integer  $0 \leq k \leq n$ , let  $\mathcal{A}_k$  denote the union of all (closed) cells with level less than  $k$ , and let  $\mathcal{B}_k$  denote the union of all (closed) cells with level greater than  $n - k$ . (In particular,  $\mathcal{A}_0 = \mathcal{B}_0 = \emptyset$ .) A point  $q$  has depth at least  $k$  if its dual line  $q^*$  lies below  $\mathcal{A}_k$  and below  $\mathcal{B}_k$ .

$\mathcal{A}_k$  and  $\mathcal{B}_k$  are unbounded polygons, each bounded by a monotone polygonal path. We can find the left “ends” of these polygonal paths in  $O(n)$  time by selecting the dual lines with  $k$ th largest and  $k$ th smallest slopes, and then trace each path in  $O(1)$  time per vertex. (In fact we only need to compute vertices at

levels  $k - 1$  and  $n - k + 1$ .) Then we can decide whether any line fits between those two paths using Seidel's linear-time linear-programming algorithm. Each boundary path trivially has at most  $O(n^2)$  vertices, so the entire algorithm runs in  $O(n^2)$  time. ■

In fact, the boundaries of  $\mathcal{A}_k$  and  $\mathcal{B}_k$  each have at most  $O(nk^{1/3})$  vertices,<sup>a</sup> so this algorithm actually runs in  $O(nk^{1/3}) = O(n^{4/3})$  time. Even this analysis is not known to be tight; the best worst-case lower bound known is only  $n \cdot e^{\Omega(\sqrt{\log k})}$ .<sup>b</sup>

Combining this decision algorithm with binary search gives us an algorithm to compute  $\text{depth}(q, P)$  in  $O(n^2 \log n)$  time, but a few simple observations improve the time to  $O(n^2)$ . First, we can compute the level of every vertex in the arrangement of  $P^*$  in  $O(n^2)$  time. Second, each vertex participates in at most two iterations of the binary search, so the total expected running time for all invocations of Seidel's algorithm is  $O(n^2)$ .

Dey's improved analysis of level complexity implies a running time of  $O(n^{4/3} \log n)$  for the basic binary search algorithm. The fastest algorithm for computing  $\text{depth}(q, P)$  runs in  $O(n \log n)$  expected time.<sup>c</sup>

<sup>a</sup>Tamal K. Dey. [Improved bounds for planar  \$k\$ -sets and related problems](#). *Discrete Comput. Geom.* 19(3):373–382, 1998.

<sup>b</sup>Gabriel Nivasch. [An improved, simple construction of many halving edges](#). *Discrete and Computational Geometry: Twenty Years Later*, Contemporary Mathematics 453, 299–305. American Mathematical Society, 2008.

<sup>c</sup>Timothy M. Chan. [An optimal randomized algorithm for maximum Tukey depth](#). *Proc. 15th SODA*, 430–436, 2004.

2. Suppose you are given a set  $L$  of  $n$  lines in the plane in general position.
- (a) Describe and analyze an algorithm to compute the leftmost vertex of the arrangement of  $L$ .

**Solution:** The arrangement vertex with smallest  $x$ -coordinate is dual to the line with smallest slope through two dual points in  $L^*$ . Those two points must be adjacent in  $x$ -coordinate order. We can prove this claim as follows: Fix three arbitrary points  $p, q, r$  with distinct  $x$ -coordinates, in order from left to right. If  $q$  lies above  $\overleftrightarrow{pr}$ , then  $\overleftrightarrow{qr}$  has smaller slope than  $\overleftrightarrow{pr}$ . On the other hand, if  $q$  lies below  $\overleftrightarrow{pr}$ , then  $\overleftrightarrow{pq}$  has smaller slope than  $\overleftrightarrow{pr}$ . In both cases,  $\overleftrightarrow{pr}$  is not the line with minimum slope.

Equivalently, the two lines that determine the rightmost arrangement vertex must be adjacent in slope order. We can prove this claim directly using a sweep-line argument: Imagine sweeping a vertical line  $\ell$  from left to right across the arrangement of  $L$ . Initially, when  $\ell$  is far to the left, the lines in  $L$  intersect  $\ell$  in slope order. Thus, as we move  $\ell$  to the right, the first change in the intersection order along  $\ell$  (the leftmost arrangement vertex) must be between two lines with adjacent slopes.

Thus, we can find the rightmost arrangement vertex in  $O(n \log n)$  time by sorting the lines in  $L$  by slope, computing the intersection point of each adjacent pair in sorted order, and returning the rightmost of those  $n - 1$  points. ■

- (b) Describe and analyze an algorithm to compute the smallest circle containing every vertex of the arrangement of  $L$ .

**Solution:** Let  $C$  denote the circle we are trying to compute, let  $v$  be any arrangement vertex that lies on  $C$ , and let  $\ell$  be the line tangent to  $C$  at  $v$ . After rotating the lines if necessary so that  $\ell$  is vertical, the argument in part (a) implies that  $v$  is the intersection of two lines that are adjacent in cyclic slope order, meaning either the slopes are adjacent, or one line has maximum slope and the other has minimum slope.

Thus, we can compute the circle  $C$  in  $O(n \log n)$  time by sorting the lines in  $L$  by slope, computing the intersection point of each adjacent pair in cyclic order, and finally computing the smallest circle that contains those  $n$  intersection points using Welzl's MINIDISK algorithm. ■

3. In class we saw the classical *funnel* algorithm to compute shortest paths inside a triangulated simple polygons. How would you modify this algorithm to find shortest paths in a polygon with holes?
  - (a) Describe and analyze an algorithm to compute the shortest path between two given points in the interior of a polygon with *one* hole. [Hint: Which way does the path go around the hole?]
  - (b) Describe and analyze an algorithm to compute the shortest path between two given points in the interior of a polygon with *two* holes.
  - (c) **Extra credit:** Describe and analyze an algorithm to compute shortest paths in a polygon with  $h$  holes; analyze your algorithm as a function of both  $n$  (the total number of vertices) and  $h$  (the number of holes).

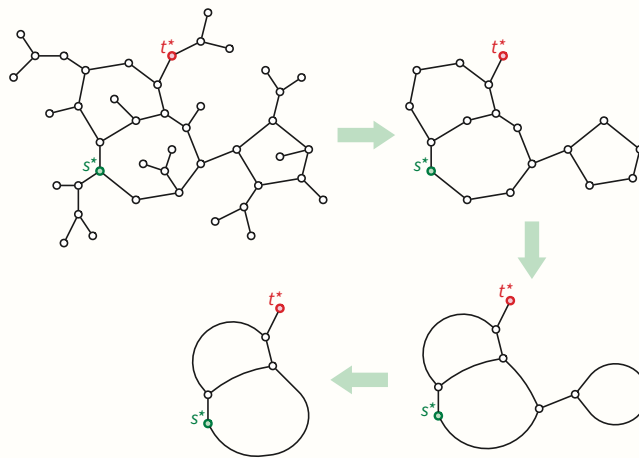
In all cases, you can assume that you are given a triangulation of the input polygon. For full credit, your algorithms should run in  $O(n)$  time (for any constant  $h$ ).

**Solution:** I'll describe a solution to part (c); solutions for (a) and (b) follow immediately as special cases.

Suppose we are given a triangulation  $T$  of a polygon  $P$  with  $h$  holes, along with two points  $s$  and  $t$  in  $P$ . The shortest path from  $s$  to  $t$  crosses each diagonal in the triangulation at most once. Thus, the sequence of triangles that the shortest path intersects describes a simple path (no repeated vertices or edges) in the dual graph  $T^*$ .

At a high level, our algorithm considers all simple paths in  $T^*$  with the correct endpoints; for each such path, we run the standard funnel algorithm to find the shortest path from  $s$  to  $t$  through the corresponding sleeve of triangles in  $T$ . The running time of the algorithm is  $O(Nn)$ , where  $N$  is the number of dual paths the algorithm considers. I will prove that  $N = O(8^h)$ .

To efficiently enumerate simple paths in  $T^*$ , we first *reduce*  $T^*$  as follows. Let  $s^*$  and  $t^*$  denote the nodes in  $T^*$  dual to the triangles containing  $s$  and  $t$ , respectively.



- First we repeatedly remove vertices of degree 1 *except*  $s^*$  and  $t^*$ , until no such vertices remain. This operation is sometimes called a *leaf reduction*.

- Then we repeatedly replace paths through degree-2 vertices *except*  $s^*$  and  $t^*$  with single edges, until no such paths remain. (This operation is sometimes called a *series reduction*. As we perform series reductions, we maintain the path in  $T^*$  corresponding to each edge the remaining graph.)
- The final phase is optional. A *bridge* is an edge whose deletion disconnects the graph. If the remaining graph contains a bridge that does *not* separate  $s$  and  $t$ , we delete the bridge and the component not containing  $s$  and  $t$ , and then (if possible) perform another series reduction at the remaining endpoint of the bridge.

The entire reduction process can be performed in  $O(n)$  time. Every simple path from  $s^*$  to  $t^*$  in the reduced dual graph  $R$  corresponds to a simple path from  $s^*$  to  $t^*$  in  $T^*$  and vice versa.

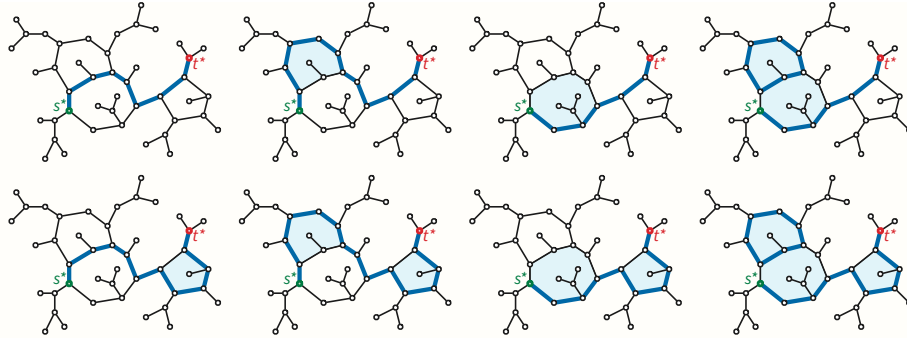
The reduced dual graph  $R$  is a planar graph with at most  $h + 1$  faces—at most one bounded face for each hole in  $P$ , plus the unbounded outer face. (Some hole cycles might have been discarded when we deleted the bridges.) Every vertex has degree 3 except possibly  $s^*$  and  $t^*$ , which might have degree 1 or 2. Routine calculations with Euler's formula now imply that  $R$  has at most  $2h + 2$  vertices and at most  $3h + 1$  edges.

Every simple path in  $R$  uses a subset of the edges; it follows immediately that  $R$  contains at most  $2^{3h+1} = O(8^h)$  simple paths from  $s^*$  to  $t^*$ . Moreover, we can enumerate these paths in  $O(8^h h)$  time by brute force: For every subset  $S$  of edges of  $R$ , test whether  $S$  is a simple path from  $s^*$  to  $t^*$  in  $O(h)$  time.

For each simple path in  $R$ , we recover the corresponding path in  $T^*$  and run the funnel algorithm in the corresponding sleeve in  $T$  in  $O(n)$  time. Finally, we return the shortest of all the funnel paths. The overall algorithm runs in  $O(8^h n)$  time, which is  $O(n)$  time for any constant  $h$ . ■

**Solution:** We can improve the previous solution by counting paths in  $T^*$  differently. The *symmetric difference* of two subgraphs of  $T^*$  is the subgraph of edges that appear in exactly one of those two subgraphs. The symmetric difference of any two paths from  $s^*$  to  $t^*$  in  $T^*$  is a subgraph of  $T^*$  in which every vertex has even degree; this subgraph must be the union of edge-disjoint simple cycles. Any union of disjoint cycles is the boundary of the union of a subset of the bounded faces of  $R$ , namely, the faces with odd depth/winding number. It follows that the number of simple paths from  $s^*$  to  $t^*$  is at most  $2^h$ . (It should be easy to see that this bound is tight in the worst case, but not *always* tight.)

We can enumerate these paths by brute force in  $O(2^h n)$  time as follows. For each subset of bounded faces, compute the boundary of its union, compute the symmetric difference of that boundary with some fixed path from  $s^*$  to  $t^*$ , and finally test whether that symmetric difference is connected (and therefore a simple path), all in  $O(n)$  time. (Using the reduced dual graph  $R$  would improve the running time of this part of the algorithm from  $O(2^h n)$  to  $O(2^h h)$ .)



Eight potential paths from  $s^*$  to  $t^*$ . Only six of them are actually paths.

Finally, for each simple path from  $s^*$  to  $t^*$  in  $T^*$ , we can run the funnel algorithm in the corresponding sleeve in  $T$  in  $O(n)$  time. The overall algorithm runs in  $O(2^h n)$  *time*, which is  $O(n)$  time for any constant  $h$ . ■