Homework 1: Convex Hulls, Plane Sweep, and More

Handed out Tuesday, Feb 14. Due at the start of class Tuesday, Feb 21. Late homeworks will not be accepted so turn in whatever you have finished. Unless otherwise specified, you may assume that points are in general position.

Problem 1. As mentioned in class, the convex hull is a somewhat non-robust shape descriptor, since if there are any distant outlying points, they will tend to dominate the shape of the hull. A more robust method is based on the following iterative approach. Given a planar point set P in general position (see Fig. 1(a)), let H_1 be the convex hull of P. Remove the vertices of H_1 from P and compute the convex hull of the remaining points, call it H_2. Repeat this until no more points remain, letting $H_1, \ldots, H_k$ denote the resulting hulls (see Fig. 1(b)). More formally, $H_i = \text{conv}(P \setminus (\bigcup_{j=1}^{i-1} \text{vert}(H_j)))$. The final result is a collection of nested convex polygons, where the last one may degenerate to a single line segment or a single point.

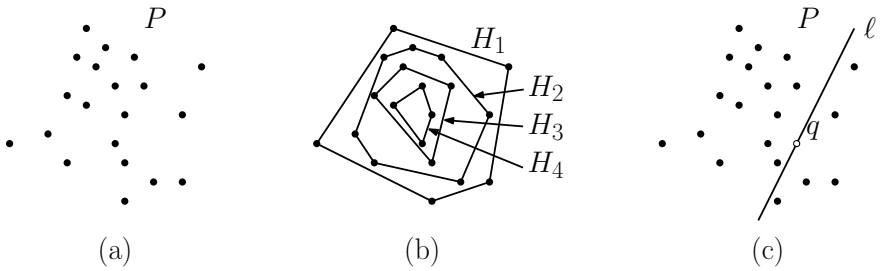


Figure 1: Repeated hulls.

(a) Assuming that the points are in general position in $\mathbb{R}^2$, as a function of n, what is the maximum number of hulls that can be generated by this process? (I am looking for an exact formula, not an asymptotic one. For every n, there should exist a point set that exactly achieves your bound.) Briefly justify your answer.

(b) Given a set P of n points in the plane, devise an $O(n^2)$ time algorithm to compute this iterated sequence of hulls. (FYI: $O(n \log n)$ is possible, but quite complicated.)

(c) Prove the following lemma: Given a planar point set P in general position, let k denote the number of hulls generated by the repeated hull process. There exists a point q in the plane (it need not be in P) such that, *every* closed halfplane whose bounding line ℓ passes through q contains at least k points of P. (Recall that a *closed halfplane* is the set of points lying on or to one side of a line. An example of such a point for $k = 4$ is shown in Fig. 1(c).)

Problem 2. You are given two sorted sets of numbers $A = \{a_1, \ldots a_n\}$ and $B = \{b_1, \ldots, b_n\}$. These two sets define a collection of n^2 sums, $S(A, B) = \{a_i + b_j \mid i, j \in \{1, \ldots, n\}\}$. For simplicity, let us make the "general position" assumption that $S(A, B)$ consists of exactly n^2 distinct values.

The *interval sum* problem is as follows. Given A and B, and given two values $s^- < s^+$, return a count of the number of elements of $S(A, B)$ that lie within the interval $[s^-, s^+]$. Present an efficient algorithm to solve the interval sum problem. Your algorithm should run in $O(n \log n)$ time.

(Hint: This problem can be solved by reducing it to inversion counting, but if you don't see the reduction, there are other ways of obtaining the desired running time.)

Problem 3. Suppose that you are given m convex polygons $P_1, \ldots, P_m$ in the plane. Let n_i denote the number of vertices on P_i, and let $\langle v_{i,1}, \ldots, v_{i,n_i} \rangle$ denote the vertices of this polygon listed in counterclockwise order, starting at the leftmost vertex of P_i (that is, the one with the smallest x-coordinate). Two polygons P_i and P_j are said to *intersect* if they contain any point in common (that is, either their boundaries intersect or one polygon is contained within the other). Present an efficient algorithm that determines whether any two polygons of the set intersect (see Fig. 2). (The output is either "yes" or "no.") Let $n = \sum_{i=1}^{m} n_i$ denote the total number of vertices. For full credit, you algorithm should run in $O(n \log m)$ time.

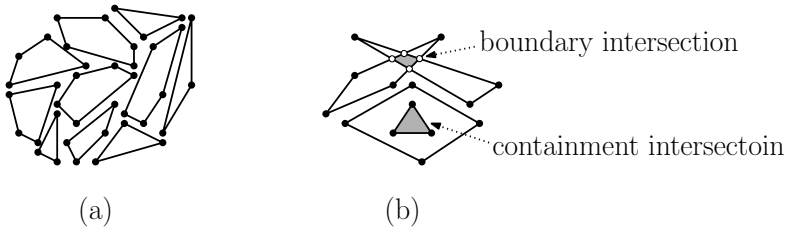


Figure 2: (a) a set of nonintersecting convex polygons; (b) a set of convex polygons that has some intersecting pairs.

Problem 4. Given a simple polygon P with n vertices, recall that the addition of any diagonal (an internal line segment joining two visible vertices of P) splits P into two simple polygons with n_1 and n_2 vertices respectively, where $n_1 + n_2 = n + 2$.

(a) Show that given any simple polygon P with $n \geq 4$ there exists a diagonal that splits P such that $\min(n_1, n_2) \geq \lfloor n/3 \rfloor$. (Hint: Consider the dual graph of a triangulation.)

(b) Show that the constant $1/3$ is the best possible, in that for any $c > 1/3$, there exists a polygon such that *any* diagonal chosen results in a split such that $\min(n_1, n_2) < cn$. (You can provide a drawing, but it should be clear how your drawing can be generalized to all sufficiently large values of n.)

Challenge Problem. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

Consider a polygonal curve P consisting of a sequence of vertices $\langle p_1, \ldots, p_n \rangle$. The curve is not closed, meaning that $p_1 \neq p_n$. The curve has the property that each vertex makes left-turn, that is, $\text{orient}(p_{i-1}, p_i, p_{i+1}) > 0$, for $2 \leq i \leq n-1$ (see Fig. 3). The problem is to design an efficient algorithm (ideally running in $O(n)$ time) that determines whether P is *simple*, meaning that no two nonadjacent edges of P intersect one another (see Fig. 3(a) and (b)).

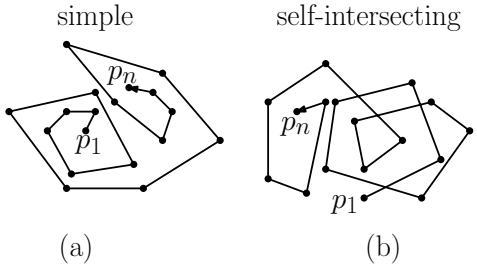


Figure 3: Left-turning polygonal curves.

(a) (Easiest) For $2 \leq i \leq n-1$, define θ_i be the counterclockwise angle from the directed vector $\overrightarrow{v_{i-1}v_i}$ to $\overrightarrow{v_iv_{i+1}}$. The *turning number* of P, denoted turn(P), is defined to be $(\sum_{i=2}^{n-2} \theta_i)/2\pi$. Derive an algorithm for testing the simplicity of P, assuming that turn(P) ≤ 1. (That is, the curve cannot make a spiral.)

(b) (Harder) Derive an algorithm for testing the simplicity of P, assuming that turn(P) $= O(1)$. (That is, the curve is not allowed to spiral more than a constant number of times.)

(c) (Hardest) Solve the general problem, without any constraint on the turning number.

Your algorithm should be self-contained, and in particular, it should not invoke any complex data structures or from the computational geometry literature. (There exist quite complex linear-time algorithms for determining whether an arbitrary polygonal curve is simple, but the intent here is to exploit the turning property to avoid the need for such sophisticated methods.)

(Hint: You may find it helpful to check out the definition of "winding number" on Wikipedia. For my solution, I found it simplified the description of the algorithm to assume that the curve has been preprocessed in $O(n)$ time into a more convenient form. For example, I assumed that the curve was translated and rotated so that p_1 and p_n both lie on the x-axis.)

Some tips about writing algorithms: Henceforth, whenever you are asked to present an "algorithm," you should present three things: the algorithm, an informal proof of its correctness, and a derivation of its running time. Remember that your description is intended to be read by a human, not a compiler, so conciseness and clarity are preferred over technical details. Unless otherwise stated, you may use any results from class, or results from any standard algorithms text. Nonetheless, be sufficiently complete that all critical issues are addressed, except for those that are obvious. (See the lecture notes for examples.)

Giving careful and rigorous proofs can be quite cumbersome in geometry, and so you are encouraged to use intuition and give illustrations whenever appropriate. Beware, however, that a poorly drawn figure can make certain erroneous hypotheses appear to be "obviously correct."

Throughout the semester, unless otherwise stated, you may assume that input objects are in *general position*. For example, you may assume that no two points have the same x-coordinate, no three points are collinear, no four points are cocircular. Also, unless otherwise stated, you may assume that any geometric primitive involving a constant number of objects each of constant complexity can be computed in $O(1)$ time.

A reminder: Remember that you are allowed to discuss general solution strategies with your classmates. Since exam problems are often modifications of homework problems, it is not a good idea to pass too much information to your friends, since this will deprive them from the exploration process of winnowing down the multitude of possible solution strategies to the final one. When it comes to writing your solution, you must work independently.

Occasionally student have told me that, in the process of trying to learn more about a problem, they have searched the Web. It sometimes happens that, as a result, they discover a fact that gives away the solution or provides a major boost. While I would encourage you to try to solve the problems on your own using just the material covered in class, I have no problem with using the Web if you get stuck. My only requirement is that you cite any resources that you used. I will not deduct points for help discovered on the Web, but it is your responsibility to express the solution in your own words and to fully understand it.

Homework 2: Linear Programming, Trapezoidal Maps, and More

Handed out Tuesday, Mar 6. Due at the start of class Thursday, Mar 15. Late homeworks will not be accepted so turn in whatever you have finished. Unless otherwise specified, you may assume that points are in general position, and if you are asked to present an $O(f(n))$ time algorithm, you may present a randomized algorithm whose expected running time is $O(f(n))$.

Problem 1. You are given two sets of points, red and blue, in the plane. Let $R = \{r_1, \ldots, r_n\}$ be the red points and $B = \{b_1, \ldots, b_n\}$ be the blue points. The problem is to determine a pair of parallel, nonvertical lines ℓ_R and ℓ_B such that all the points of R lie on or above ℓ_R, all the points of B lie on or below ℓ_B, and the signed vertical distance from ℓ_R to ℓ_B is as small as possible. (Note that if the points of R are above all the points of B, then ℓ_R will be above ℓ_B, and this signed distance will be negative.) Present an $O(n)$ time algorithm to solve this problem.

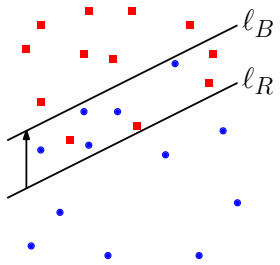


Figure 1: Problem 1.

Problem 2. You are given two convex polygons P^- and P^+, where P^- is enclosed within P^+. Let n be the total number of vertices between these two polygons. Present an efficient (ideally $O(n)$ time) algorithm that computes a convex polygon Q whose boundary is nested between P^- and P^+ (see Fig. 2). Your polygon Q should have the smallest number of sides possible. Let Q^* be the polygon nested between P^- and P^+ with the smallest number of sides. Prove that, if Q^* has k sides, then the polygon that your algorithm produces has at most $c \cdot k$ sides, for some constant c (which does not depend on n).

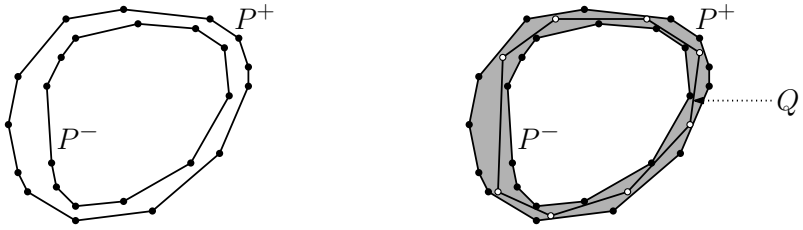


Figure 2: Problem 2.

Hint: Obtaining the exact minimum number of sides is possible, but quite difficult. All the simple algorithms I can think of achieve $c \leq 2$, and you will receive full credit if your algorithm does at least this well. The best that can be achieved using a simple approach produces a polygon with at most $k + 1$ sides.

Problem 3. The objective of this problem is to explore some properties of trapezoidal maps, which will be useful in the next problem.

Throughout this problem $S = \{s_1, \ldots, s_n\}$ is a set of n nonintersecting, nonvertical line segments in the plane. Let $T(S)$ denote the trapezoidal map of these segments. We say that a trapezoid $\Delta \in T(S)$ is *incident* on a segment $s \in S$ if s borders Δ from above or below, or if one of s's endpoints bounds Δ from the left or the right (see Fig. 3(a)). For $s \in S$, define $\deg(s)$ to be the number of trapezoids of $T(S)$ that are incident on s (in Fig. 3(a), $\deg(s) = 7$).

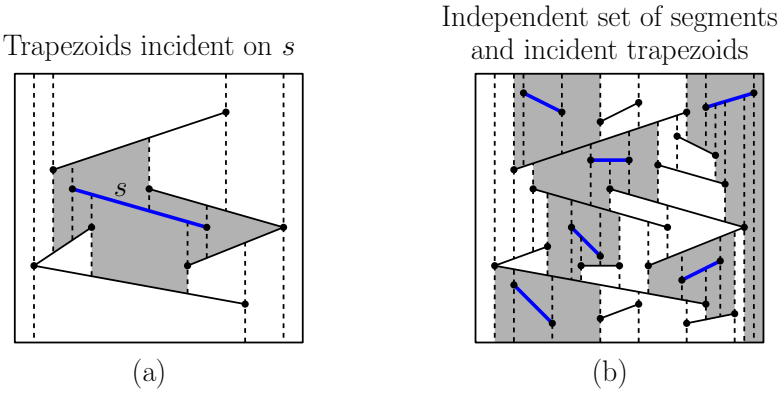


Figure 3: Problem 3.

(a) Given any set S of n segments, prove that there exists a constant c, such that, for all sufficiently large n, $\sum_{s \in S} \deg(s) \leq cn$.

(b) Let c be the constant derived in your solution to (a). We say that a segment $s \in S$ is *long* if $\deg(s) \geq 2c$, and otherwise we say that s is *short*. Let $S' \subseteq S$ be the set of short segments of S. Prove that there exists a constant c' (which may depend on c) such that, for all sufficiently large n, $|S'| \geq n/c'$.

(c) We say that two segments $s_i, s_j \in S$ are *adjacent* if there exists a trapezoid $\Delta \in T(S)$ that is incident on both s_i and s_j. Define an *independent set* of S to be a subset of S whose elements are pairwise non-adjacent (see Fig. 3(b)). Given the previous constants c and c', prove that there exists a constant $c'' > 1$ (depending on c and c') such that, for all sufficiently large n, S contains an independent set of size at least n/c''. (Hint: It suffices to consider just the short segments.)

Problem 4. Consider the set S of n segments from Problem 1 and let $T(S)$ denote the resulting trapezoidal map. The objective of this problem is to explore an alternative method for constructing a point-location data structure for $T(S)$. From Problem 1, we know that, for all sufficiently large n (that is, for all $n \geq n_0$, for some constant n_0) it is possible to compute an independent set of short segments $I \subseteq S$ whose size is at least n/c'', for some constant c''.

Let $S_0 = S$. For $j = 0, 1, 2, \ldots$, repeat the following until $|S_j| \leq n_0$. Let I_j be the independent set of Problem 1 for the set S_j. Set $S_{j+1} = S_j \setminus I_j$. (The first few phases are shown in Fig. 4.) Let k be the value of j when the process stops.

(a) Briefly justify each of the following claims:

(i) For $0 \leq j \leq k$, $|S_j| \leq n\big(1 - 1/c''\big)^j$ (The number of segments decreases geometrically)

(ii) $k = O(\log n)$ (The number of levels is logarithmic)

(iii) $\sum_{j=0}^{k} |S_j| = O(n)$ (The total space is linear)

(b) Describe a point-location data structure that determines which trapezoid of $T(S)$ contains a given query point q. Your data structure will have $k + 1$ *levels*, for $0 \leq j \leq k$. The job of level j is to determine which trapezoid of $T(S_j)$ contains q. This level may recursively invoke the search on

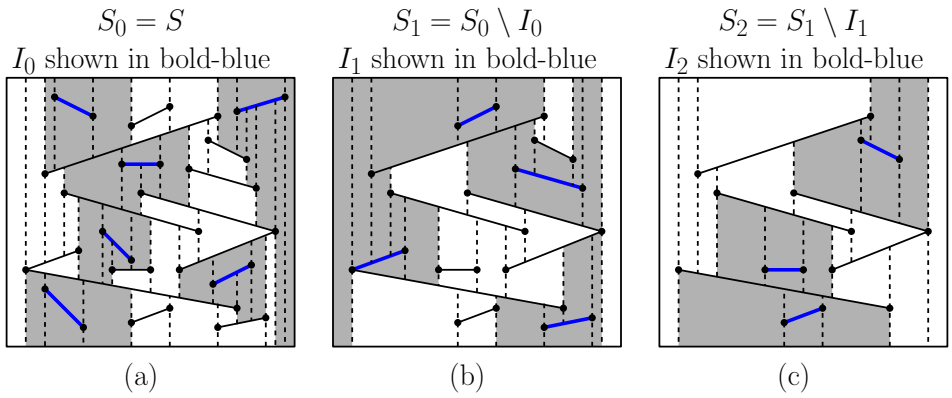


Figure 4: Problem 4.

deeper levels (e.g., level $j+1$), to achieve its objective. Explain both how to build the structure and how to use the structure to answer queries.

(Hint: It is not necessary to describe your algorithm to the same high degree of detail that we did for the history DAG in class. In particular, you may assume that the following primitive is available to you. Let T and T' be two collections of trapezoids that cover the same region of space, each containing $O(1)$ trapezoids. In constant time, it is possible to build a search structure which, assuming we know which trapezoid of T contains a query point q, in constant time we can determine which trapezoid of T' contains q.)

(c) Prove that your data structure uses $O(n)$ space and can answer point-location queries in $O(\log n)$ time.

Challenge Problem. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

Let P be a simple polygon with n sides. We say that two vertices v_i and v_j of P are *monotonically reachable* if there is an x-monotone path from v_i to v_j. (Fig. 5 shows a number of vertices of P that have x-monotone paths between them.) Present an $O(n \log n)$ time algorithm that computes a count of the total number of monotonically reachable pairs.

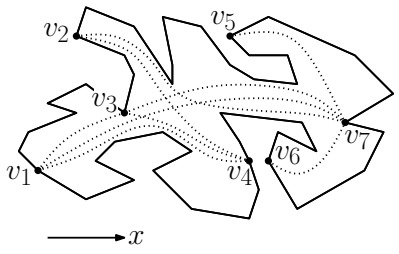


Figure 5: Challenge Problem. (Monotonically reachable pairs for the vertices shown are $\{v_1, v_3\}$, $\{v_1, v_4\}$, $\{v_1, v_7\}$, $\{v_2, v_4\}$, $\{v_2, v_7\}$, $\{v_3, v_4\}$, $\{v_3, v_7\}$, $\{v_5, v_7\}$, $\{v_6, v_7\}$.)

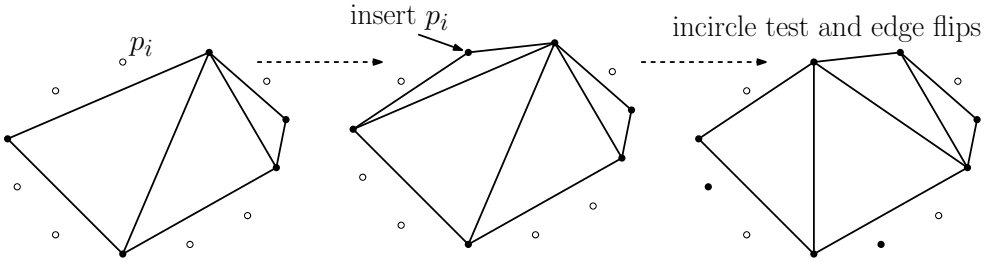
(Hint: This is not that hard. My algorithm starts by computing a trapezoidal decomposition or actually any triangulation. After this, it runs in $O(n)$ time.)

Homework 3: Voronoi/Delaunay and Arrangements

Handed out Thu, Apr 12. Due at the start of class Tue, Apr 24. Late homeworks will not be accepted so turn in whatever you have finished. Unless otherwise specified, you may assume that points are in general position, and if you are asked to present an $O(f(n))$ time algorithm, you may present a randomized algorithm whose expected running time is $O(f(n))$.

Problem 1. Let $P = \langle p_1, p_2, \ldots, p_n \rangle$ be the vertices of a convex polygon, given in counterclockwise order. The purpose of this problem is to develop a randomized incremental Delaunay triangulation algorithm that runs in $O(n)$ expected time.

Unlike the algorithm presented in class, we *do not* put all the sites within a large bounding triangle. We start with the triangle defined by three random sites from P. The remaining sites are inserted in random order. Each new site p is connected to the convex hull by adding its two two tangent edges, thus creating a new triangle. As in the standard randomized algorithm, an incircle test is performed on this triangle to determine whether it is locally Delaunay. If it fails, an edge flip is performed. We continue performing incircle tests and edge flips until all the newly created triangles are locally Delaunay (see the figure below).



(a) Show that the expected number of edge-flips performed with each insertion is $O(1)$.

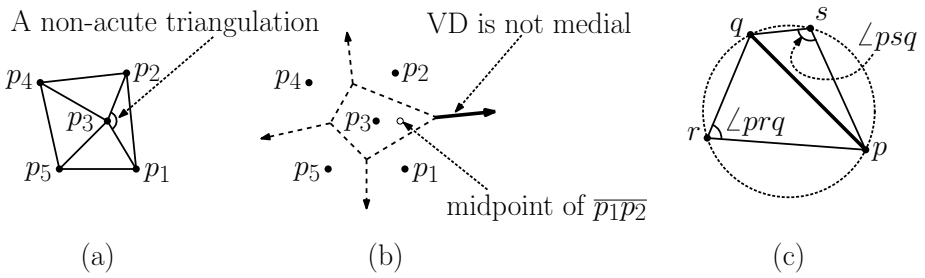
(b) Whenever a new site is added, we need to determine where along the current convex hull it is to be added. Explain how to do this in $O(1)$ expected time. You are allowed preprocess the sites in $O(n)$ time.

Hint: Your answer to (b) needs to be clear about the data structures involved and how much time is needed to perform each access. You may assume you have access to a black-box procedure that returns a random permutation of the integers from 1 to n in $O(n)$ time. It may be easier to consider the problem in reverse, namely start with all the vertices of P and remove them one-by-one in random order.

Problem 2. We start with two (seemingly unrelated) definitions. A triangulation of a set of points in the plane is *acute* if the angles of all its triangles are strictly acute, that is, less than $\pi/2$. (See the figure below (a) for an example of a triangulation that is *not* acute.) The Voronoi diagram of a set of points in the plane is said to be *medial* if each edge of the diagram contains in its interior the midpoint of the two defining sites for the edge. (See the figure below (b) for an example of a Voronoi diagram that is *not* medial.)

(a) Prove that any acute triangulation of a set of points in the plane is a Delaunay triangulation, that is, it satisfies the empty circumcircle property.

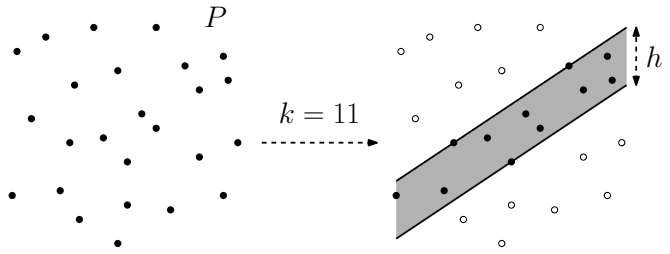
(b) Prove that if the Voronoi diagram of a set of points in the plane is medial, then the corresponding Delaunay triangulation is acute.



(a) (b) (c)

Hint: The following basic geometric fact may be useful. Let $\overline{pq}$ be a chord of a circle and let r and s be two points lying on the circle, one on either side of $\overline{pq}$. Then $\angle prq + \angle psq = \pi$. (See the figure above (c).)

Problem 3. The following exercise is motivated by a problem in statistical estimation. You are given a set of n points P in the plane and an integer k, where $3 \leq k \leq n$. Define a *k-corridor* to be the closed region of the plane bounded by two parallel lines that contains at least k points of P. The *height* of a corridor is the vertical distance h between these lines (see the figure below).



(a) Explain how to model the k-corridor problem in the dual setting. (Explain what a k-corridor of vertical height h corresponds to in the dual plane.)

(b) Present an $O(n^2 \log n)$ time algorithm for computing the minimum height k-corridor, which is given P and k as inputs. (**Hint:** Use plane sweep in the dual arrangement.)

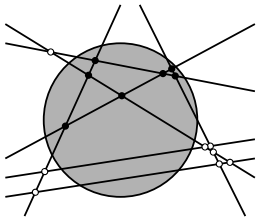
Problem 4. Let P be a set of n points in the plane. Define a *quadrilateral* to be a four-sided simple polygon.

(a) Present an $O(n^2)$ time algorithm that computes the minimum area quadrilateral whose vertices are drawn from P. (**Hint:** A quadrilateral consists of two triangles joined to opposite sides of a line segment $\overline{ab}$, where $a, b \in P$.)

(b) Part (a) places no constraints on the quadrilateral. A natural variant would be to compute the minimum area *convex* quadrilateral (but I do not know of an efficient solution to this problem). Instead, consider the following problem. Devise an $O(n^3 \log n)$ time algorithm that *counts* the number of convex quadrilaterals whose vertices are drawn from P. (**Hint:** $\Box abcd$ is a convex quadrilateral if and only if line segments $\overline{ac}$ and $\overline{bd}$ intersect. One approach is to enumerate all pairs $a, c \in P$ and then count the pairs of points $b, d \in P$ that satisfy this condition. This can be reduced to inversion counting in the dual.)

Challenge Problem. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

You are given a collection of n lines $L = \{\ell_1, \ldots, \ell_n\}$ in the plane, where, for $1 \leq i \leq n$, ℓ_i is represented by the pair (a_i, b_i) and is given by the equation $\ell_i : y = a_i x - b_i$. You are also given a circle whose center is at a given point $c = (c_x, c_y)$ and whose radius is given as $r > 0$. Give an $O(n \log n)$ time algorithm that counts the number of intersections of these lines that lie within the circle.



Homework 4: Geometric Searching and Approximations

Handed out Thu, April 26. Due at the start of class Thu, May 10. Late homeworks will not be accepted so turn in whatever you have finished. Unless otherwise specified, you may assume that points are in general position, and if you are asked to present an $O(f(n))$ time algorithm, you may present a randomized algorithm whose expected running time is $O(f(n))$.

Problem 1. In Homework 1, we saw how to compute a sequence of layers of convex hulls for a point set P. Use this structure to develop a data structure for answering halfplane range reporting queries for the set P. (Given a closed halfplane h, report all the points of P that lie within h).

Your data structure should use space $O(n)$ and should be able to answer a query in $O((k+1)\log n)$ time, where k is the number of points inside the query range. (Hint: The following utility may be helpful. Given an m-sided convex polygon H, preprocess H to answer the following queries in $O(\log m)$ time. Given a halfplane h determine either that h does not intersect H, or if it does, return any vertex of H that lies within h.)

Problem 2. The objective of this problem is to investigate the *VC-dimension* of some range spaces. Recall that a *range space* Σ is a pair $(X, \mathcal{R})$, where X is a (finite or infinite) set, called *points*, and $\mathcal{R}$ is a (finite or infinite) family of subsets of X, called *ranges*. (Please refer to Lecture 17 for the definition of range space, shattering, and VC-dimension.)

For each of the following range spaces, derive its VC-dimension and prove your result. (Note that in order to show that the VC-dimension is k, you need to give an example of a k-element subset that is shattered and prove that no set of size $k+1$ can be shattered.) Throughout, you may assume that points are in general position.

Example: Consider the range space $\Sigma = (\mathbb{R}^2, H)$ where H consists of all closed horizontal halfspaces, that is, halfplanes of the form $y \geq y_0$ or $y \leq y_0$. We claim that $\text{VC}(\Sigma) = 2$.

$\text{VC}(\Sigma) \geq 2$: Consider the points $a = (0, -1)$ and $b = (0, 1)$. The ranges $y \geq 2$, $y \geq 0$, $y \leq 0$ and $y \leq 2$ generate the subsets $\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$, respectively. Therefore, there is a set of size two that is shattered.

$\text{VC}(\Sigma) < 3$: Consider any three element set $\{a, b, c\}$ in the plane. Let us assume that these points have been given in increasing order of their y-coordinates. Observe that any horizontal halfplane that contains b, must either contain a or c. Therefore, no 3-element point set can be shattered.

(a) $\Sigma_{\mathcal{R}} = (\mathbb{R}^2, \mathcal{R})$, where $\mathcal{R}$ is the set of all closed axis-aligned rectangles.

(b) $\Sigma_{\mathcal{S}} = (\mathbb{R}^2, \mathcal{S})$, where $\mathcal{S}$ is the set of all closed axis-aligned squares.

(c) $\Sigma_{\mathcal{D}} = (\mathbb{R}^2, \mathcal{D})$, where $\mathcal{D}$ is the set of all closed circular disks in the plane.

Problem 3. The purpose of this problem is to consider a simple approximation algorithm to an important problem that arises in clustering. You are given n points P in the plane. Given any integer k, $1 \leq k \leq n$, define $b_k(P)$ to be the Euclidean ball of minimum radius that encloses k points of P (see Fig. 1). (Note that the center of $b_k(P)$ may be anywhere in $\mathbb{R}^2$, not necessarily at a point of P). Let $r_k(P)$ denote the radius of this ball. The objective of this problem is to derive an algorithm that computes a factor-2 approximation, that is, it computes a ball of radius at most $2r_k(P)$ that contains at least k points of P.

The approximation algorithm is based on computing a small number of *candidate centers* $Q = \{q_1, \ldots, q_m\}$, such that at least one of these candidate centers is guaranteed to lie within $b_k(P)$.

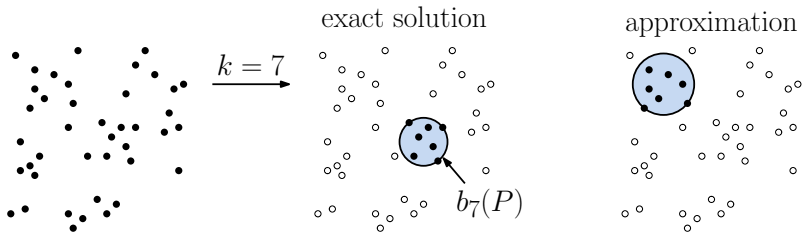


Figure 1: Problem 3.

For each candidate center q_i, we will determine the radius of the smallest ball centered at q_i that contains k points, and return the smallest such ball.

The candidate centers are constructed as follows. First, sort the points by their x-coordinates, letting $x_1 \leq \ldots \leq x_n$ be the resulting set. Let $k' = \lfloor (k-1)/2 \rfloor$. Let X be the set that results by taking every k'-th point in the sequence, that is $X = \{x_{k'}, x_{2k'}, \ldots, x_{zk'}\}$, where $z = \lfloor n/k' \rfloor$. Do the same for the y-coordinates by letting Y denote the result of taking every k'-th point of the y-coordinates in sorted order. Finally, let $Q = X \times Y$, be the set of points that result by taking any x-coordinate from X and any y-coordinate from Y.

(a) Prove that there exists a point $q \in Q$ such that $q \in b_k(P)$. (Hint: Show that *any* ball that fails to contain a point of Q can contain at most $2k'$ points of P.)

(b) For each $q_i \in Q$, let r_i denote the radius of the smallest ball centered at q_i that contains at least k points of P. Let $r_{\min} = \min_i r_i$ and let $b_{\min}$ denote the associated ball. Prove that $r_{\min} \leq 2r_k(P)$. (This establishes the fact that $b_{\min}$ is a factor-2 approximation to $b_k(P)$.)

(c) Show that the algorithm's overall running time is $O(n \log n + n^3/k^2)$. For what choices of k is the running time $O(n \log n)$?

Problem 4. At the start of the semester, we discussed how to efficiently compute statistics for a set of quadratic size in linear, or roughly linear time. For the last homework problem of the semester, we'll consider an approximation problem of this same ilk.

Given two disjoint point sets P and Q, each containing n points in the $\mathbb{R}^d$, and a fixed approximation parameter $\varepsilon > 0$, present an $O(n \log n)$ time algorithm that computes an ε-approximation to the average interpoint distance between these sets. More formally, define

$$\delta(P, Q) \;=\; \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \mathrm{dist}(p_i, q_j).$$

In $O(n \log n)$ time compute an value $\widehat{\delta}(P, Q)$ such that the relative error between $\delta(P, Q)$ and this approximate value is at most ε, that is,

$$\frac{\left|\delta(P, Q) - \widehat{\delta}(P, Q)\right|}{\delta(P, Q)} \;\leq\; \varepsilon.$$

Justify your algorithm's correctness. Express the hidden constant factor in the running time as a function of ε and dimension d. (Hint: You will need to explain how to modify the WSPD construction to this context.)

Challenge Problem 1. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

Modify your solution to Problem 1, so the running time is $O(k + \log n)$ with at most $O(n \log n)$ total space.

Hint: Apply fractional cascading to the following query problem. You are given a collection of m pairwise disjoint x-monotone polygonal chains that extend from $x = -\infty$ to $x = +\infty$. Let n denote the total number of vertices in all these chains. Preprocess these chains so that, given any query point q in the plane, it is possible to *report* all chain edges that are intersected by a ray shot vertically downwards from q. The query time should be $O(k + \log n)$, where k is the number of edges hit by the ray.

Note that a straightforward application of fractional cascading leads to $O(n \log n)$ space. If you are clever, it is possible to reduce the space to $O(n)$.

Challenge Problem 2. For many clustering problems, squared distances may more relevant than standard distances. Given two n-element point sets P and Q in $\mathbb{R}^d$, define

$$\delta^{(2)}(P, Q) \;=\; \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} (\text{dist}(p_i, q_j))^2.$$

Present an $O(n)$ time algorithm that computes $\delta^{(2)}(P, Q)$ exactly. (To keep the formulas simple, I will be satisfied if you do this $\mathbb{R}^2$.)

Hint: This does not require clever algorithm design, just algebra.

Sample Problems for the Midterm Exam

The following problems have been collected from old homeworks and exams. They do *not* reflect the actual length or difficulty of the midterm exam. The exam will be *closed-book* and *closed-notes*. You may use *one sheet of notes* (front and back). Unless otherwise stated, you may assume general position. If you are asked to present an $O(f(n))$ time algorithm, you may present a randomized algorithm whose expected running time is $O(f(n))$. For each algorithm you give, derive its running time and justify its correctness.

Problem 1. Give a short answer to each question (a few sentences suffice).

(a) In the analysis of the randomized incremental point location we argued that the expected depth of a random query point is at most $12 \ln n$. Based on your knowledge of the proof, where does the factor 12 arise? (Hint: It arises from two factors. You can explain either for partial credit.)

(b) In the primal plane, there is a triangle whose vertices are the three points p, q, and r and there is a line ℓ that intersects this triangle. What can you infer about the relationship among the corresponding dual lines p^*, q^*, r^*, and the dual point ℓ^*? Explain.

(c) Recall the orientation primitive $Orient(a, b, c)$, which given three points in the plane, returns a positive value if the sequence $\langle a, b, c\rangle$ is ordered counterclockwise, zero if the points are collinear, and negative if clockwise. Show how to use this primitive (one or more times) to determine whether a point d lies within the interior of the triangle defined by the points a, b, and c. (You may assume that a, b, and c are oriented counterclockwise.)

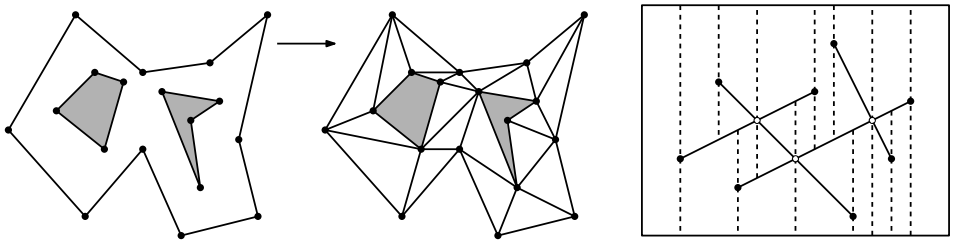


Figure 1: Problems 1(d) and 1(e).

(d) Any triangulation of any n-sided simple polygon has exactly $n - 2$ triangles. Suppose that the polygon has h polygonal holes each having k sides. (In Fig. 1, $n = 12$, $h = 3$, and $k = 4$). As a function of n, h and k, how many triangles will such a triangulation have?

(e) You are given a set of n disjoint line segments in the plane that have I intersection points (In Fig. 1, $n = 4$ and $I = 3$). Suppose that you build a trapezoidal map of the segments, but whenever two segments intersect, there are two bullet paths shot (up and down) from such a point. As a function of n and I, how many trapezoids are there in a trapezoidal map? Explain briefly. (Give an exact, not asymptotic, answer.)

Problem 2. The following problem is a two-dimensional version of the hidden-surface algorithm called *depth sorting* in computer graphics. Given a set of n line segments in the plane, such that no two segments intersect, order these segments so that any ray parallel to the x-axis, directed from left to right, intersects these segments in increasing order (see Fig. 2). As part of your answer, prove that such an ordering always exists. Give an algorithm for computing such an ordering in $O(n \log n)$ time. Explain briefly.

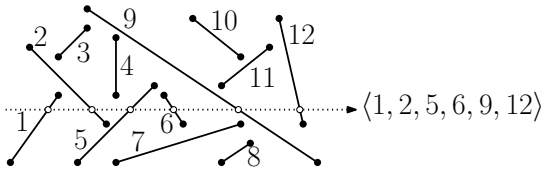


Figure 2: Problem 2.

Problem 3. For this problem give an exact bound for full credit and an asymptotic (big-Oh) bound for partial credit. Assume general position.

(a) You are given a convex polygon P in the plane having n_P sides and an x-monotone polygonal chain Q having n_Q sides (see Fig. 3(a)). What is the maximum number of intersections that might occur between the edges of these two polygons?

(b) Same as (a), but P and Q are both polygonal chains that are monotone with respect to the x-axis (see Fig. 3(b)).

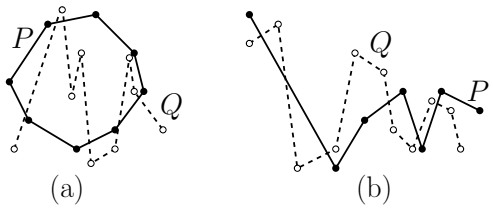


Figure 3: Problem 3.

(c) Same as (b), but P and Q are both monotone polygonal chains, but they may be monotone with respect to two different directions.

Problem 4. Consider the following randomized incremental algorithm, which computes the smallest rectangle (with sides parallel to the axes) bounding a set of points in the plane. This rectangle is represented by its lower-left point low and the upper-right point high.

(1) Let $P = \{p_1, p_2, \ldots, p_n\}$ be a random permutation of the points.

(2) Let $\text{low}[x] = \text{high}[x] = p_1[x]$. Let $\text{low}[y] = \text{high}[y] = p_1[y]$.

(3) For $i = 2$ through n do:

  (a) if $p_i[x] < \text{low}[x]$ then $(*)$ $\text{low}[x] = p_i[x]$.

  (b) if $p_i[y] < \text{low}[y]$ then $(*)$ $\text{low}[y] = p_i[y]$.

  (c) if $p_i[x] > \text{high}[x]$ then $(*)$ $\text{high}[x] = p_i[x]$.

(d) if $p_i[y] > \text{high}[y]$ then $(*)$ $\text{high}[y] = p_i[y]$.

Clearly this algorithm runs in $O(n)$ time. Prove that the total number of times that "then" clauses of statements 3(a)–(d) (each indicated with a $(*)$) are executed is $O(\log n)$ on average. (We are averaging over all possible random permutations of the points.) To simplify your analysis you may assume that no two points have the same x- or y-coordinates.

Problem 5. You are given a set of n vertical line segments in the plane. Present an efficient an algorithm to determine whether there exists a line that intersects all of these segments. An example is shown in the figure below. (Hint: $O(n)$ time is possible.) Justify your algorithm's correctness and derive its running time.

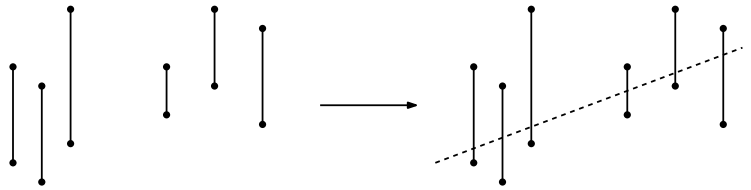


Figure 4: Problem 5.

Problem 6. Given a set of n points in the plane, a *triangulation* of these points is a planar straight line graph whose outer face is the convex hull of the point set, and each of whose internal faces is a triangle. There are many possible triangulations of a set of points. Throughout this problem you may assume that no three points are collinear.

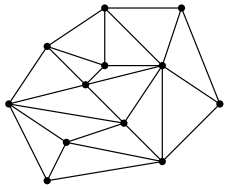


Figure 5: Problem 6.

(a) Among the n points, suppose that h lie on the convex hull of the point set. As a function of n and h, what is the number of triangles (internal faces) t in the triangulation. Show how you derived your answer. (It is a fact, which you do not need to prove, that the number of triangles depends only on n and h.) You may give an asymptotic answer for partial credit. (Hint: Remember Euler's formula, which states that a connected planar graph with v vertices, e edges, and f faces (including the external face) satisfies $v - e + f = 2$.)

(b) Describe an $O(n \log n)$ algorithm for constructing *any* triangulation (your choice) of a set of n points in the plane. Explain your algorithm and analyze its running time. You may assume that the result is stored in any reasonable representation.

Problem 7. A simple polygon P is *star-shaped* if there is a point q in the interior of P such that for each point p on the boundary of P, the open line segment $\overline{qp}$ lies entirely within

the interior of P (see Fig. 6). Suppose that P is given as a counterclockwise sequence of its vertices $\langle v_1, v_2, \ldots, v_n \rangle$. Show that it is possible to determine whether P is star-shaped in $O(n)$ time. (Note: You are *not* given the point q.) Prove the correctness of your algorithm.

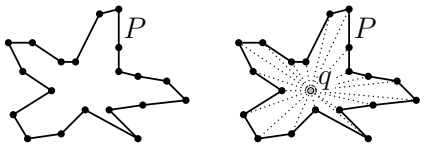


Figure 6: Problem 7.

Problem 8. You are given two sets of points in the plane, the red set R containing n_r points and the blue set B containing n_b points. The total number of points in both sets is $n = n_r + n_b$. Give an $O(n)$ time algorithm to determine whether the convex hull of the red set intersects the convex hull of the blue set. If one hull is nested within the other, then we consider them to intersect.

Problem 9. Define a *strip* to be the region bounded by two (nonvertical) parallel lines. The *width* of a strip is the vertical distance between the two lines.

(a) Suppose that a strip of vertical width w contains a set of n points in the primal plane (see Fig. 7). Dualize the points and the two lines. Describe (in words and pictures) the resulting configuration in the dual plane. Assume the usual dual transformation that maps point (a, b) to the line $y = ax - b$, and vice versa.

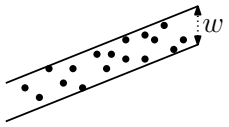


Figure 7: Problem 9.

(b) Give an $O(n)$ time algorithm, which given a set of n points in the plane, finds the nonvertical strip of minimum width that encloses all of these points.

Problem 10. Given a set of n points P in the plane, we define a subdivision of the plane into rectangular regions by the following rule. We assume that all the points are contained within a bounding rectangle. Imagine that the points are sorted in increasing order of y-coordinate. For each point in this order, shoot a bullet to the left, to the right and up until it hits an existing segment, and then add these three bullet-path segments to the subdivision (see Fig. 8(a)).

(a) Show that the resulting subdivision has size $O(n)$ (including vertices, edges, and faces).

(b) Describe an algorithm to add a new point to the subdivision and restore the proper subdivision structure. Note that the new point may have an arbitrary y-coordinate, but the subdivision must be updated as if the points had been inserted in increasing order of y-coordinate (see Fig. 8(b)).

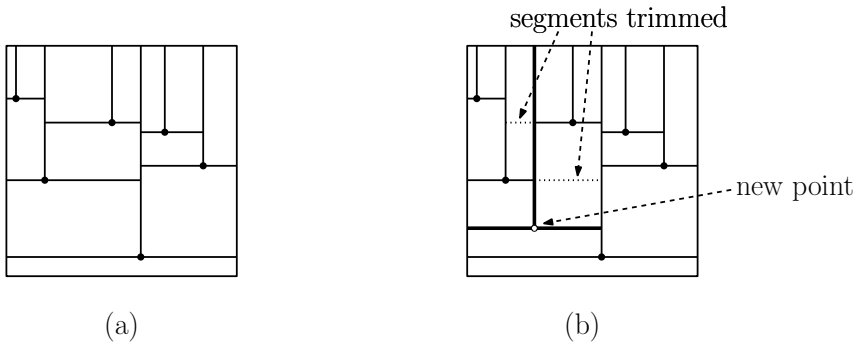


Figure 8: Problem 10.

(c) Prove that if the points are added in random order, then the expected number of structural changes to the subdivision with each insertion is $O(1)$.

Problem 11. You are given a collection of vertical line segments in the first quadrant of the x, y plane. Each line segment has one endpoint on the x-axis and the other endpoint has a positive y-coordinate. Imagine that from the top of each segment a horizontal bullet is shot to the left. The problem is to determine the index of the segment that is first hit by each of these bullet paths. If no segment is hit, return the index 0 (see Fig. 9).

The input is a sequence of top endpoints of each segment $p_i = (x_i, y_i)$, for $1 \leq i \leq n$, which are *sorted in increasing order* by x-coordinate. The output is the sequence of indices, one for each segment.

Present an $O(n)$ time algorithm to solve this problem. Explain how your algorithm works and justify its running time.

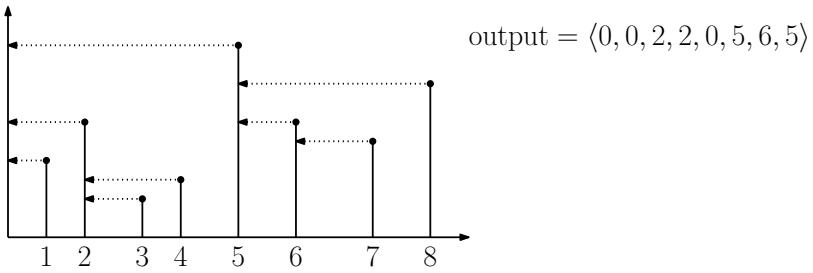


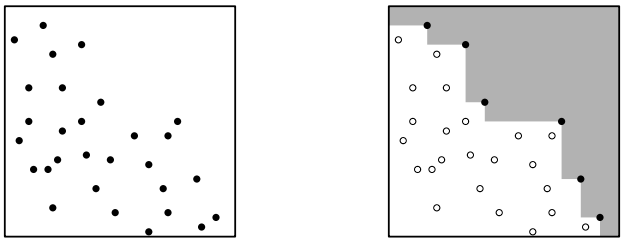
Figure 9: Problem 11.

Problem 12. The following problem asks you to solve two basic problems about a simple polygon in linear time. Ideally, your solution should not make use of any fancy data structures.

(a) You are given a cyclic sequence of vertices forming the boundary of a simple n-sided polygon P, but you are not told whether the sequence has been given in clockwise or counterclockwise order. (More formally, if the polygon's boundary were to be continuously morphed into circle, what would be the orientation of the circle?) Give an $O(n)$ time algorithm that determines which is the case. You may assume general position.

(b) You are given a simple n-sided polygon P where $n \geq 4$. Recall that a *diagonal* is a line segment joining two vertices such that the interior of the segment lies entirely within the polygon's interior. Give an $O(n)$ time algorithm that finds any diagonal in P.

Problem 13. Given two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in the plane, we say that p_2 *dominates* p_1 if $x_1 \leq x_2$ and $y_1 \leq y_2$. Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$, a point p_i is said to be *maximal* if it is not dominated by any other point of P (shown as black points in Fig. 10(b)).



(a) (b)

Figure 10: Problem 13.

Suppose further that the points of P have been generated by a random process, where the x-coordinate and y-coordinate of each point are independently generated random real numbers in the interval $[0, 1]$.

(a) Assume that the points of P are sorted in increasing order of their x-coordinates. As a function of n and i, what is the probability that p_i is maximal? (Hint: Consider the points p_j, where $j \geq i$.)

(b) Prove that the expected number of maximal points in P is $O(\log n)$.

Problem 14. Consider an n-sided simple polygon P in the plane. Let us suppose that the leftmost edge of P is vertical (see Fig. 11(a)). Let e denote this edge. Explain how to construct a data structure to answer the following queries in $O(\log n)$ time with $O(n)$ space. Given a ray r whose origin lies on e and which is directed into the interior of P, find the first edge of P that this ray hits. For example, in the figure below the query for ray r should report edge f. (Hint: Reduce this to a point location query in an appropriate planar subdivision.)

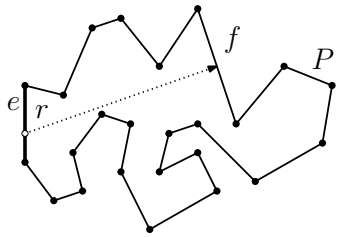
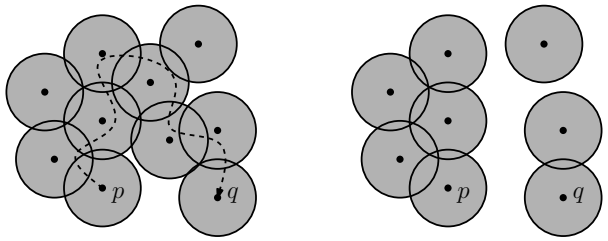


Figure 11: Problem 14.

Problem 15. You are given a set of n sites P in the plane. Each site of P is the center of a circular disk of radius 1. The points within each disk are said to be *safe*. We say that P is *safely connected* if, given any $p, q \in P$, it is possible to travel from p to q by a path that travels only in the safe region. (For example, the disks of Fig. 12(a) are connected, but the disks of Fig. 12(b) are not.)

Present an $O(n \log n)$ time algorithm to determine whether such a set of sites P is safely connected. Justify the correctness of your algorithm and derive its running time.



(a) (b)

Figure 12: Problem 15.

Problem 16. In class we argued that the number of parabolic arcs along the beach line in Fortune's algorithm is at most $2n - 1$. The goal of this problem is to prove this result in a somewhat more general setting.

Consider the beach line at some stage of the computation, and let $\{p_1, p_2, \ldots, p_n\}$ denote the sites that have been processed up to this point in time. Label each arc of the beach line with its the associated site. Reading the labels from left to right defines a string. (In Fig. 13 below the string would be $p_2p_1p_2p_5p_7p_9p_{10}$.)

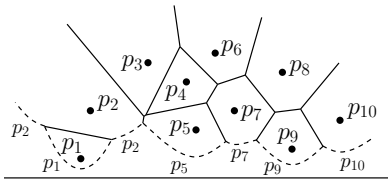


Figure 13: Problem 16.

(a) Prove that for any i, j, the following alternating subsequence *cannot* appear anywhere within such a string:

$$\ldots p_i \ldots p_j \ldots p_i \ldots p_j \ldots$$

(b) Prove that any string of n distinct symbols that does not contain any repeated symbols ($\ldots p_i p_i \ldots$) and does not contain the alternating sequence[1] of the type given in part (a) cannot be of length greater than $2n - 1$. (Hint: Use induction on n.)

[1]Sequences that contain no forbidden subsequence of alternating symbols are famous in combinatorics. They are known as *Davenport-Schinzel sequences*. They have numerous applications in computational geometry, this being one.

CMSC 754: Midterm Exam

This exam is closed-book and closed-notes. You may use one sheet of notes, front and back. Write all answers in the exam booklet. If you have a question, either raise your hand or come to the front of class. Total point value is 100 points. Good luck!

In all problems, unless otherwise stated, you may assume that points are in *general position*. You may make use of any results presented in class and any well known facts from algorithms or data structures. If you are asked for an $O(T(n))$ time algorithm, you may give a *randomized* algorithm with *expected* time $O(T(n))$.

Problem 1. (25 points; 4-6 points each) Give a short answer (a few sentences) to each question.

(a) You wish to use an orientation primitive to determine whether a point d lies within the interior of a triangle $\triangle abc$ in the plane. Your friend tells you that it suffices to test whether $\text{Orient}(a, b, d)$, $\text{Orient}(b, c, d)$ and $\text{Orient}(c, a, d)$ are all of the *same sign*. You realize that you don't know whether the vertices a, b, and c are given in clockwise or counterclockwise order. Is your friend's solution correct? Explain briefly.

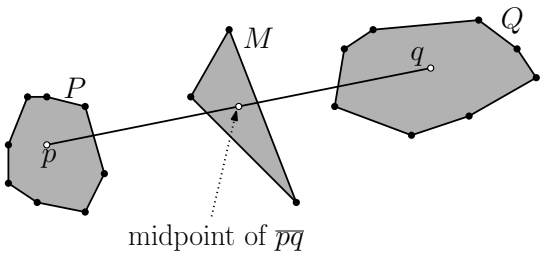
(b) In the algorithm presented in class for decomposing a simple polygon into monotone pieces, what was the definition of helper(e) and (in a few words) what role did it play in the algorithm?

(c) A convex polygon P_1 is enclosed within another convex polygon P_2. Suppose you dualize the vertices of each of these polygons (using the dual transform given in class, where the point (a, b) is mapped to the dual line $y = ax - b$). What can be said (if anything) about the relationships between the resulting two dual sets of lines.

(d) In a Voronoi diagram of a set of sites $P = \langle p_1, \ldots, p_n \rangle$ in the plane, you observe that the Voronoi edge between sites p_i and p_j is semi-infinite (that is, one end of the edge goes to infinity). What can be said about the relationship between these two sites and the rest of the point set? Explain.

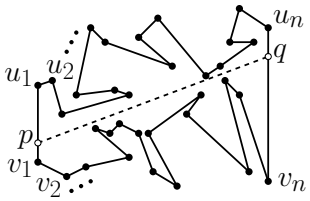
(e) Give a short proof of the following claim: Given a set of n points in the plane $P = \langle p_1, \ldots, p_n \rangle$ that have been sorted by their x-coordinates, the maximum slope determined by any two points of the set will be achieved by a pair of consecutive points. Thus, to compute the maximum slope, it suffices to compute the slopes of $\{p_i, p_{i+1}\}$ for $1 \leq i < n$.

Problem 2. (15 points) You are given three convex polygons in the plane P, Q, and M. Let n denote the total number of vertices in all three polygons. Each is given as a sequence of vertices in counterclockwise order. Present an $O(n)$ time algorithm to determine whether there exists a line segment $\overline{pq}$ such that $p \in P$, $q \in Q$, and the midpoint of p and q lies within M (see the figure below).



Problem 3. (20 points)

You are given a simple polygon P with the property that its leftmost and rightmost edges are vertical. Let $\langle u_1, \ldots, u_n \rangle$ denote the sequence of vertices on the polygonal chain joining the two upper endpoints of the leftmost and rightmost edges, and let $\langle v_1, \ldots, v_n \rangle$ denote the sequence of vertices of the polygonal chain joining the two lower endpoints (see the figure to the right).



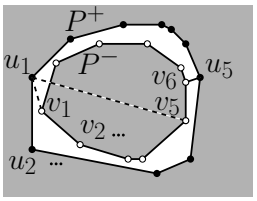
Present an efficient algorithm to determine whether there exists a point p on the leftmost edge and a point q on the rightmost edge such that the line segment $\overline{pq}$ lies entirely within P. (The line segment $\overline{pq}$ is allowed to pass through vertices of P, but it cannot intersect the exterior of P.)

Derive your algorithm's running time and justify its correctness.

Hint: I have intentionally left the running time unspecified. Your score will depend in part on the asymptotic efficiency of your algorithm.

Problem 4. (20 points)

You are given two convex polygons, each as a counterclockwise sequence of vertices, $P^+ = \langle u_1, \ldots, u_n \rangle$ and $P^- \langle v_1, \ldots, v_m \rangle$, where $P^- \subseteq P^+$. Given any two vertices $u_i \in P^+$ and $v_j \in P^-$, we say that these vertices are *visible* if the line segment between them lies entirely within the region $P^+ \setminus P^-$. (For example, in the figure to the right, u_1 and v_1 are visible, but u_1 and v_5 are *not* visible.)

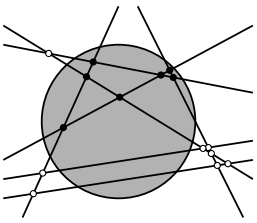


Present an efficient algorithm that computes the *closest pair* of visible vertices (v_i, u_j), $1 \le i \le n$ and $1 \le j \le m$. (In the figure, the final answer is (u_5, v_6).) Distance is measured as the Euclidean distance between the points.

Hint: $O(n + m)$ time is possible. I'll give 1/2 partial credit for an $O((n + m) \log(n + m))$ time solution, and 1/3 partial credit for an output sensitive algorithm whose running time is $O(V)$, where V is the number of visible pairs.

Problem 5. (20 points)

You are given a collection of n lines $L = \{\ell_1, \ldots, \ell_n\}$ in the plane, where, for $1 \le i \le n$, ℓ_i is represented by the pair (a_i, b_i) and is given by the equation $\ell_i : y = a_i x - b_i$. You are also given a circle whose center is at a given point $c = (c_x, c_y)$ and whose radius is given as $r > 0$. The problem is to count the number of intersections of these lines that lie within the circle.



For full credit, give an $O(n \log n)$ time algorithm. For 1/4 partial credit, you may give an $O((n + I) \log n)$ time solution, where I is the number of line intersections that occur *inside* the circle. (For example, in the figure $I = 7$, since it does not count intersections outside the circle.)

Sample Problems for the Final Exam

The following problems have been collected from old homeworks and exams. They do not necessarily reflect the actual difficulty or coverage of questions on the final exam. The final will be comprehensive, but will emphasize material since the midterm.

The final exam will be Mon, May 14, 8:00-10:00am. (Set an alarm!) The exam will be closed-book and closed-notes. You may use *two* sheets of notes (front and back). In all problems, unless otherwise stated, you may assume general position, and you may use of any results presented in class or any well-known result from algorithms and data structures.

Problem 1. Give a short answer (a few sentences) to each question.

(a) A *dodecahedron* is a convex polyhedron that has 12 faces, each of which is a 5-sided pentagon. How many vertices and edges does the dodecahedron have? Show how you derived your answer.

(b) A kd-tree of n points in the plane defines a subdivision of the plane into n cells, each of which is a rectangle. Is this subdivision a *cell complex*? Explain briefly.

(c) Given a kd-tree with n points in the plane, what is the (asymptotic) maximum number of cells that might be stabbed by an line that is not axis-parallel? Explain briefly.

(d) What is a *zone* in an arrangement? Given an arrangement A of n lines in the plane and given an arbitrary line ℓ, what is the (asymptotic) maximum complexity (number of edges) of the zone of ℓ relative to A? *(No explanation needed.)*

(e) Which of the following statements regarding the Delaunay triangulation (DT) of a set of points in the plane are true? *(No explanation needed.)* Among all triangulations...

(i) ...the DT minimizes the maximum angle.

(ii) ...the DT maximizes the minimum angle.

(iii) ...the DT has the minimum total edge length.

(iv) The largest angle in a DT cannot exceed 90 degrees.

(f) An arrangement of n lines in the plane has exactly n^2 edges. How many edges are there in an arrangement of n planes in 3-dimensional space? (Give an exact answer for full credit or an asymptotically tight answer for partial credit.) Explain briefly.

Problem 2. You are given a set of n triangles in the plane $\mathcal{T} = \{T_1, \ldots, T_n\}$, where triangle T_i has vertices $\langle a_i, b_i, c_i \rangle$. Present an algorithm that computes a line ℓ that stabs the greatest number of triangles of $\mathcal{T}$. (For example, in Fig. 1 there exists a line that intersects 4 of the 5 triangles.) Your algorithms should run in $O(n^2)$ time and use at most $O(n^2)$ space. (Hint: Given any triangle, identify the set of points in the dual plane corresponding to lines that stab this triangle.)

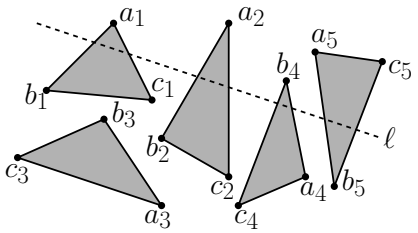


Figure 1: Problem 2.

Problem 3. Given a set P of n points in the plane and an integer k ($1 \leq k \leq n-1$), a *k-set* is defined to be a subset of $P' \subseteq P$, where $|P'| = k$ and $P' = P \cap h$ for some halfplane h. That is, a k-set is any set of k points that can be separated from the rest of P by a single line. (In Fig. 2(a) we show two examples of 3-sets, $\{a, b, c\}$ and $\{a, d, h\}$.)

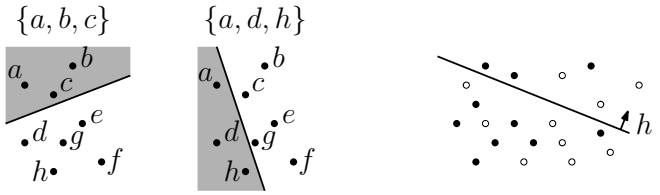


Figure 2: Problem 3.

(a) Given what you know about point-line duality, explain how a k-set manifests itself in the dual plane, in terms of the arrangement of the dual lines P^*.

(b) Present an $O(n^2 \log n)$ time and $O(n)$ space algorithm which, given two planar point sets P and Q, each of size n, and an integer k, determines whether there is there a halfspace h, such that $|P \cap h| = |Q \cap h| = k$. (See Fig. 2(b) for the special case $k = 3$.)

Problem 4. Consider a set P of n points in the plane. The distances between each pair of distinct points defines a set of $\binom{n}{2}$ *interpoint distances*. Present an efficient algorithm to compute an approximation the *second largest* interpoint distance.

More formally, your algorithm is given a set P of n points in the plane and a constant approximation parameter $\varepsilon > 0$. Let Δ denote the true second largest interpoint distance among the points of P. Your algorithm may output any value Δ' where

$$\frac{\Delta}{1+\varepsilon} \;\leq\; \Delta' \;\leq\; (1+\varepsilon)\Delta.$$

(Hint: Use WSPDs.)

Problem 5. You are given a set P of n points in the plane and a path π that visits each point exactly once. (This path may self-intersect. See Fig. 3.)

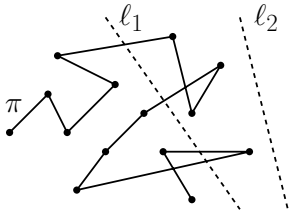


Figure 3: Problem 5.

(a) Explain how to build a data structure from P and π of space $O(n)$ so that given any query line ℓ, it is possible to determine in $O(\log n)$ time whether ℓ intersects the path. (For example, in Fig. 3 the answer for ℓ_1 is "yes," and the answer for ℓ_2 is "no.")

(b) This is a generalization of part (a). Explain how to build a data structure from P and π so that given any line ℓ, it is possible to report all the segments of π that intersect ℓ. The space should be at most $O(n \log n)$ and the query time should be at most $O(k \log^2 n)$, where k is the number of segments reported. (Hint: Even if you did not solve (a), you can solve part (b) by applying the data structure of part (a) as a "black box.")

Problem 6. You are given a set $P = \{p_1, p_2, \ldots, p_n\}$ of n points in the plane. Consider all the $\binom{n}{2}$ lines passing through each pair of distinct points $p_i, p_j \in P$, and let $\ell_{\max}$ to be the line of this set with the maximum slope. We are interested in computing $\ell_{\max}$.

(a) What is the interpretation of $\ell_{\max}$ in the dual plane?

(b) Give an $O(n \log n)$ algorithm that computes $\ell_{\max}$. Justify your algorithm's correctness.

Problem 7. In this problem you may assume the follow rather remarkable result, which we will not prove.

Theorem: For any set P of n points in the plane, there exists a spanning tree T of P, whose edges are straight line segments, such that any line in the plane, crosses at most $O(\sqrt{n})$ edges of T.

(a) Prove that the above theorem is tight in the sense that there exists a set of n points in the plane such that for any spanning tree T on these points, there exists a line (depending on T) that intersects at least $\Omega(\sqrt{n})$ edges of T. (Hint: For my proof, I arranged the points in a grid, and only needed to consider horizontal and vertical lines.)

(b) (Using the above theorem) prove that there exists a path π (as opposed to a tree) consisting of straight line segment that spans all the points of P, such that any line intersects at most $O(\sqrt{n})$ edges of π.

(c) Assuming that n is even, (using the above theorem) prove that there exists a matching M of the points of P, such that any line intersects at most $O(\sqrt{n})$ edges of M. (Recall that a *matching* is defined to be a collection of line segments joining pairs of points of P such that each point P is incident to exactly one segment of the collection.)

Problem 8. Consider the following two geometric graphs defined on a set P of points in the plane. For each graph, indicate whether it is a subgraph of the Delaunay triangulation. If it is, provide a short proof. If not, give a small counterexample. (Any counterexample should be in general position.)

(a) *Box Graph:* Given two points $p, q \in P$, define $\text{box}(p, q)$ to be the square centered at the midpoint of $\overline{pq}$ having two sides parallel to the segment $\overline{pq}$ (see part (a) of the figure below). The edge (p, q) is in the box graph if and only if $\text{box}(p, q)$ contains no other point of P (see the figure (b)).

(b) *Diamond Graph:* Given two points $p, q \in P$, define $\text{diamond}(p, q)$ to be the square having $\overline{pq}$ as a diagonal (see (c) below). The edge (p, q) is in the diamond graph if and only if $\text{diamond}(p, q)$ contains no other point of P (see (d)).

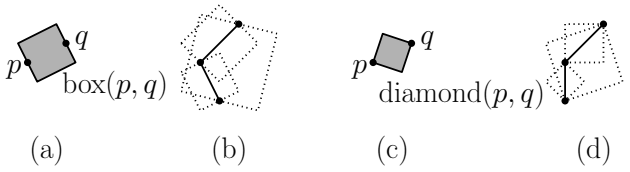


Figure 4: Problem 8.

Problem 9. The objective of this problem is to compute the discrepancy of a set of points in the plane, but this time with respect to a different set, namely, the set of axis-parallel rectangles. Let P denote a set of n points in the unit hypercube $U = [0,1]^2$. Given any axis-parallel rectangle R define $\mu(R)$ to be the area of $R \cap U$ and define $\mu_P(R) = |P \cap R|/|P|$ to be the fraction of points of P lying within R (see Fig. 5). Define the discrepancy of P with respect to R to be $\Delta_P(R) = |\mu(R) - \mu_P(R)|$, and define the *rectangle discrepancy* of P, denoted $\Delta(P)$ to be the maximum (or more accurately, the supremum) of $\Delta_P(R)$ over all axis-parallel rectangles R in U.

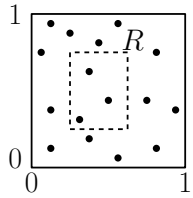


Figure 5: Problem 9.

Present an $O(n^4)$ time and $O(n^2)$ space algorithm for computing rectangle discrepancy of P by answering the following parts. Throughout you may assume that the points of P are in general position, but the axis-parallel rectangles that are used in the computation of the discrepancy are arbitrary.

(a) Establish a *finiteness criterion* for this problem by showing that there exists a set of at most $O(n^4)$ rectangles such that $\Delta(P)$ is given by one of these rectangles. Call these the *canonical rectangles* for P.

(b) Develop a rectangle range counting data structure of size $O(n^2)$ that can be used to compute the number of points of P lying within any canonical rectangle in $O(1)$ time. (Hint: The answer to the query will involve both addition and subtraction.) Because the rectangle query is canonical, you should not assume general position. Your procedure should allow the option of either including or excluding points on the boundary of the rectangle.

(c) Using your solution to (b), show how to compute the discrepancy for P in $O(n^4)$ time and $O(n^2)$ space.

Problem 10. You are given a set P of n points in $\mathbb{R}^d$ and an approximation factor $\varepsilon > 0$. An (exact) *distance query* is defined as follows. You are given a real $z > 0$, and you are to return a count of all the pairs of points $(p, q) \in P \times P$, such that $\|pq\| \geq z$. In an *ε-approximate distance query*, your count *must* include all pairs (p, q) where $\|pq\| \geq z(1+\varepsilon)$ and it *must not* include any pairs (p, q) where $\|pq\| < z/(1+\varepsilon)$. Pairs of points whose distances lie between these two bounds may or may not be counted, at the discretion of the algorithm.

Explain how to preprocess P into a data structure so that ε-approximate distance counting queries can be answered in $O(n/\varepsilon^d)$ time and $O(n/\varepsilon^d)$ space. (Hint: Use a well-separated pair decomposition. Explain clearly what separation factor is used and any needed modification to the WSPD construction.)

Problem 11. A number of problems do not, at first, appear to be orthogonal range searching, but they can be reduced to orthogonal range searching. This may involve performing multiple queries, the use of multi-level search structures, or transforming objects into higher dimensional spaces.

For each of the following query problems, explain how to answer it by reduction to one or more instances of orthogonal range searching. In each case, indicate the query time and space of your data structure, assuming that your structure is based on orthogonal range trees. (You do not need to assume that fractional cascading is used, but if you do assume this, please state this fact clearly.)

(a) The data consists of a set P of n points in the plane. A query is a rectangle in which two sides have slope $+1$ and two sides have slope -1. You are to report the points of P that lie within the rectangle.

(b) The data is the same as in (a). A query is a triangle whose left side is vertical, whose bottom edge is horizontal, and whose third side has a slope of -1. You are to report the points of P that lie within this triangle.

(c) The data consists of a set L of n nonvertical lines in the plane. A query consists of a triple of numbers (t_0, t_1, t_2). You are to report the lines of L that pass above the point $(0, t_0)$, below $(1, t_1)$, and above $(2, t_2)$.

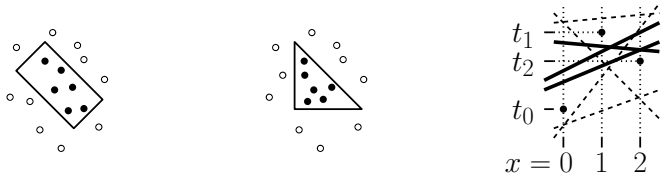


Figure 6: Problem 11.

Problem 12. Consider a set P of n points in the plane. For $k \leq \lfloor n/2 \rfloor$, point q (which may or may not be in P) is called a *k-splitter* if every line L passing through q has at least k points of P lying on or above it and at least k points on or below it. (For example the point q in Fig. 7 is a 3-splitter, since every line passing through q has at least 3 points of P lying on either side. But it is not a 4-splitter since a horizontal line through q has only 3 points below it.)

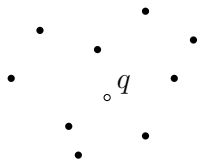


Figure 7: Problem 12.

(a) Show that for all (sufficiently large) n there exists a set of n points with no $\lfloor n/2 \rfloor$-splitter.

(b) Prove that there exists a k-splitter if and only if in the dual line arrangement, levels $\mathcal{L}_k$ and $\mathcal{L}_{n-k+1}$ can be separated by a line.

(c) Prove that any set of n points in the plane has a $\lfloor n/3 \rfloor$-splitter.

(d) Describe an $O(n^2)$ algorithm for computing a $\lfloor n/3 \rfloor$-splitter for point set P.

CMSC 754: Final Exam

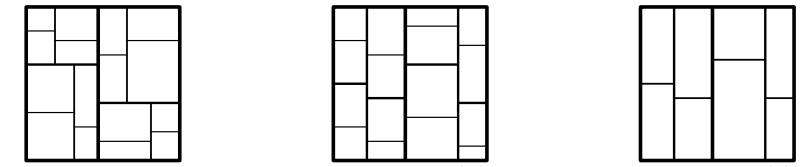
This exam is closed-book and closed-notes. You may use two sheets of notes, front and back. Write all answers in the exam booklet. If you have a question, either raise your hand or come to the front of class. Total point value is 100 points. Good luck!

In all problems, unless otherwise stated, you may assume that points are in *general position*. You may make use of any results presented in class and any well known facts from algorithms or data structures. If you are asked for an $O(T(n))$ time algorithm, you may give a *randomized* algorithm with *expected* time $O(T(n))$.

Problem 1. (25 points; 4–8 points each) Give a short answer (a few sentences) to each question.

(a) Given a set P of n points in the plane, what is the maximum number of edges in P's Voronoi diagram? (For full credit, express your answer up to an additive constant.)

(b) Suppose we were to build a kd-tree for n points in the plane, but rather than alternating splitting along x and then y, we perform median splits for the x-coordinates for *two consecutive levels* of the tree followed by median splits for the y-coordinates for the next *two consecutive levels*. This same pattern repeats *every four levels*. Would asymptotic query time to answer an orthogonal range search query in the resulting be the same, larger, or smaller compared to a standard kd-tree? (No explanation required.)

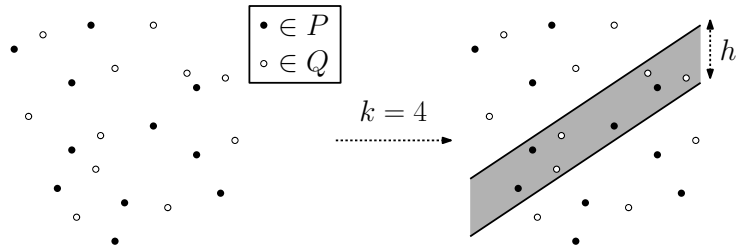


standard kd-tree Problem 1(b) Problem 1(c)

(c) Same as (b), but suppose that we perform median splits for the x-coordinates for two consecutive levels followed by a *single* median split for the y-coordinates. This pattern repeats *every three levels*.

(d) True or false: Given a planar point set P, if p and q are the closest points of P then $\overline{pq}$ is an edge of P's Delaunay triangulation?

Problem 2. (20 points) You are given two sets of points P and Q in the plane, each containing n elements. Given an integer k, $2 \leq k \leq n$, define a *balanced k-corridor* to be the closed region of the plane bounded by two parallel lines that contains exactly k points of P and k points of Q. The *height* of a corridor is the vertical distance h between these lines (see the figure below).



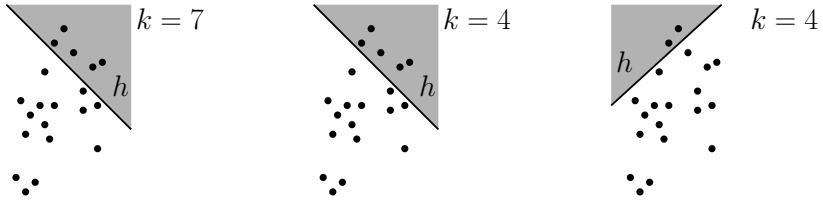
(*continued on next page*)

(a) Explain what a balanced k-corridor of vertical height h corresponds to in the dual plane.

(b) What conditions will be satisfied by a balanced k-corridor of *minimum height* in the dual setting? Explain why.

(c) Present an $O(n^2 \log n)$ time algorithm, which given inputs P, Q, and k, computes the minimum height balanced k-corridor for P and Q. (You will receive more partial credit if you present a correct algorithm that runs in $O(kn^2 \log n)$ time than an incorrect algorithm.)

Problem 3. (25 points; 4–8 points each) In this problem we will consider *limited halfplane range counting*, which means that we are given an upper bound k on the number of points we wish to count. Given a halfplane h, the query returns a count of the number of points that lie in h, unless that number exceeds k, in which case we return a special value "too many!" (see the figure below).



ans: 5 ans: "too many" ans: 2

For each of the problems below, explain how to preprocess the input set of size of n points into a data structure so that queries can be answered in $O(\log n)$ time. If there are multiple solutions, choose one that minimizes the asymptotic space requirements. Briefly justify the space and query time used by your solution. (I don't care about preprocessing time.)

You may use any data structure presented in class, either in the lecture notes or in a homework solution, but be sure to explain any adaptations you may need to make.

(a) Let k be a *fixed integer constant* (independent of n). The input consists of a set P of n points in the plane. The query is a closed upper halfplane h, that is, $h = \{(x, y) : y \geq ax - b\}$. If $|P \cap h| \leq k$, then the answer to the query is the count $|P \cap h|$. Otherwise, the query returns the special value "too many!"

(b) Same as problem (a), but now k is an *arbitrary integer* between 0 and $n - 1$, which is *fixed at preprocessing time*. (Express space requirements as a function of both n and k.)

(c) Same as problem (b), but now k is an arbitrary integer between 0 and $n - 1$, which is *given as part of the query*.

(d) Same as problem (c), but now the slope and y-intercept of the line that defines h are *rational numbers* whose numerators and denominators are integers in the range $[-100, +100]$.

Hint: The following facts may be useful. Given an arrangement of n lines the plane, the combinatorial complexity of: (a) any zone is $O(n)$, (b) the kth level of the arrangement is $O(nk^{1/3})$, (c) the total complexity of the first k levels is $O(nk)$.

(*continued on next page*)

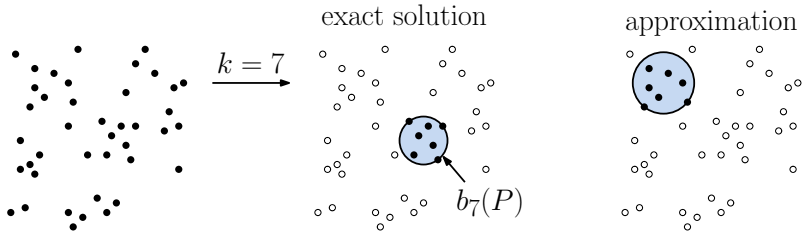
Problem 4. (15 points) Given a set of points P in space and a parameter $t > 1$, called the *stretch factor*, a *t-spanner* is a subgraph G of the complete Euclidean such that the graph distance between any two points exceeds the Euclidean distance by a factor of at most t. In this problem we will consider a variant, where, instead of considering Euclidean distances, we will instead use *squared Euclidean distances*.

More formally, consider a connected, undirected graph G whose vertices are the points of P. The weight $w(u, v)$ of an edge (u, v) in G is the squared Euclidean distance between u and v, which we denote by $\|uv\|^2$. Given any $x, y \in P$, let $\delta_G(x, y)$ denote the cost of a shortest path between x and y in G, using these weights. G is called a *squared t-spanner* if $\delta_G(x, y) \leq t \cdot \|xy\|^2$ for all $x, y \in P$.

Present an algorithm, which, given an n-element point set P in $\mathbb{R}^d$ and a spanner factor $t > 1$, computes a squared t-spanner for P. Prove that your algorithm's correctness. As a function of n, t, and dimension d, how many edges does your spanner contain?

Hint: Algebraic manipulation doesn't help here. The construction is the same as the WSPD-based spanner construction given in class, but the separation factor will be different. *Be careful in working through the inequalities.* It's easy to get lost!

Problem 5. (15 points) In Homework 4, we explored a factor-2 approximation algorithm for the following problem. You are given n points P in the plane. Given any integer k, $1 \leq k \leq n$, compute the ball of minimum radius that contains at least k points of P. (Recall that this ball may be centered at any point of $\mathbb{R}^2$.) Recall that $b_k(P)$ denotes this ball and $r_k(P)$ denotes its radius.



The objective of this problem is to improve this result to an ε-approximation. In particular, given P, k and $\varepsilon > 0$, your algorithm should return a ball that contains at least k points of P and whose radius is no greater than $(1+\varepsilon)r_k(P)$. The running time of your algorithm should be $O((1/\varepsilon^2)(n \log n + n^3/k^2))$.

To save time, you need only describe the modifications and additions relative to the solution given in Homework 4. Justify your algorithm's correctness, and derive its running time.