CMSC 754: Short Reference Guide

This document contains a short summary of information about algorithm analysis and data structures, which may be useful later in the semester.

Asymptotic Forms: The following gives both the formal "c and n_0" definitions and an equivalent limit definition for the standard asymptotic forms. Assume that f and g are nonnegative functions.

Asymptotic Form	Relationship	Limit Form	Formal Definition
$f(n) \in \Theta(g(n))$	$f(n) \equiv g(n)$	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	$\exists c_1, c_2, n_0, \forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n).$
$f(n) \in O(g(n))$	$f(n) \preceq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	$\exists c, n_0, \forall n \geq n_0,\ 0 \leq f(n) \leq cg(n).$
$f(n) \in \Omega(g(n))$	$f(n) \succeq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$	$\exists c, n_0, \forall n \geq n_0,\ 0 \leq cg(n) \leq f(n).$
$f(n) \in o(g(n))$	$f(n) \prec g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$	$\forall c, \exists n_0, \forall n \geq n_0,\ 0 \leq f(n) \leq cg(n).$
$f(n) \in \omega(g(n))$	$f(n) \succ g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	$\forall c, \exists n_0, \forall n \geq n_0,\ 0 \leq cg(n) \leq f(n).$

Polylog-Polynomial-Exponential: For any constants a, b, and c, where $b > 0$ and $c > 1$.

$$\log^a n \prec n^b \prec c^n.$$

Common Summations: Let c be any constant, $c \neq 1$, and $n \geq 0$.

Name of Series	Formula	Closed-Form Solution	Asymptotic
Constant Series	$\sum_{i=a}^{b} 1$	$= \max(b-a+1, 0)$	$\Theta(b-a)$
Arithmetic Series	$\sum_{i=0}^{n} i = 0 + 1 + 2 + \cdots + n$	$= \frac{n(n+1)}{2}$	$\Theta(n^2)$
Geometric Series	$\sum_{i=0}^{n} c^i = 1 + c + c^2 + \cdots + c^n$	$= \frac{c^{n+1}-1}{c-1}$	$\begin{cases} \Theta(c^n)\ (c>1) \\ \Theta(1)\ (c<1) \end{cases}$
Quadratic Series	$\sum_{i=0}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2$	$= \frac{2n^3+3n^2+n}{6}$	$\Theta(n^3)$
Linear-geom. Series	$\sum_{i=0}^{n-1} ic^i = c + 2c^2 + 3c^3 \cdots + nc^n$	$= \frac{(n-1)c^{(n+1)} - nc^n + c}{(c-1)^2}$	$\Theta(nc^n)$
Harmonic Series	$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$	$\approx \ln n$	$\Theta(\log n)$

Recurrences: Recursive algorithms (especially those based on divide-and-conquer) can often be analyzed using the so-called *Master Theorem*, which states that given constants $a > 0$, $b > 1$, and $d \geq 0$, the function $T(n) = aT(n/b) + O(n^d)$, has the following asymptotic form:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a. \end{cases}$$

Sorting: The following algorithms sort a set of n keys over a totally ordered domain. Let $[m]$ denote the set $\{0, \ldots, m\}$, and let $[m]^k$ denote the set of ordered k-tuples, where each element is taken from $[m]$.

A sorting algorithm is *stable* if it preserves the relative order of equal elements. A sorting algorithm is *in-place* if it uses no additional array storage other than the input array (although $O(\log n)$ additional space is allowed for the recursion stack). The *comparison-based algorithms* (Insertion-, Merge-, Heap-, and QuickSort) operate under the general assumption that there is a *comparator function* $f(x, y)$ that takes two elements x and y and determines whether $x < y$, $x = y$, or $x > y$.

Algorithm	Domain	Time	Space	Stable	In-place
CountingSort	Integers $[m]$	$O(n+m)$	$O(n+m)$	Yes	No
RadixSort	Integers $[m]^k$ or $[m^k]$	$O(k(n+m))$	$O(kn+m)$	Yes	No
InsertionSort	Total order	$O(n^2)$	$O(n)$	Yes	Yes
MergeSort	Total order	$O(n \log n)$	$O(n)$	Yes	No
HeapSort				No	Yes
QuickSort				Yes/No*	No/Yes

*There are two versions of QuickSort, one which is stable but not in-place, and one which is in-place but not stable.

Order statistics: For any k, $1 \leq k \leq n$, the kth smallest element of a set of size n (over a totally ordered domain) can be computed in $O(n)$ time.

Useful Data Structures: All these data structures use $O(n)$ space to store n objects.

Unordered Dictionary: (by randomized hashing) Insert, delete, and find in $O(1)$ expected time each. (Note that you can find an element exactly, but you cannot quickly find its predecessor or successor.)

Ordered Dictionary: (by balanced binary trees or skiplists) Insert, delete, find, predecessor, successor, merge, split in $O(\log n)$ time each. (Merge means combining the contents of two dictionaries, where the elements of one dictionary are all smaller than the elements of the other. Split means splitting a dictionary into two about a given value x, where one dictionary contains all the items less than or equal to x and the other contains the items greater than x.) Given the location of an item x in the data structure, it is possible to locate a given element y in time $O(\log k)$, where k is the number of elements between x and y (inclusive).

Priority Queues: (by binary heaps) Insert, delete, extract-min, union, decrease-key, increase-key in $O(\log n)$ time. Find-min in $O(1)$ time each. Make-heap from n keys in $O(n)$ time.

Priority Queues: (by Fibonacci heaps) Any sequence of n insert, extract-min, union, decrease-key can be done in $O(1)$ amortized time each. (That is, the sequence takes $O(n)$ total time.) Extract-min and delete take $O(\log n)$ amortized time. Make-heap from n keys in $O(n)$ time.

Disjoint Set Union-Find: (by inverted trees with path compression) Union of two disjoint sets and find the set containing an element in $O(\log n)$ time each. A sequence of m operations can be done in $O(\alpha(m, n))$ amortized time. That is, the entire sequence can be done in $O(m \cdot \alpha(m, n))$ time. (α is the *extremely* slow growing inverse-Ackerman function.)

Orientation Testing: For any constant dimension d, given any ordered $(d+1)$-tuple of points in $\mathbb{R}^d$, it can be determined in $O(1)$ time whether these points are (a) negatively oriented (clockwise), (b) positively oriented (counterclockwise) or (c) affinely dependent (collinear). This test can be applied for many other geometric predicates, such as determining whether two given line segments in the plane intersect, whether a given point lies within a given triangle, and whether a given point lies within the circumcircle of three other given points. (This will be discussed later in the semester.)

CMSC 754: Spring 2020 Dave Mount

Homework 1: Convexity and Hulls

Handed out Tuesday, Feb 11. Due at the start of class on Tuesday, Feb 18. Late homeworks are not accepted (unless an extension has been prearranged) so please turn in whatever you have completed by the due date. Unless otherwise specified, you may assume that all inputs are given in *general position*.

Problem 1. In this problem, we will consider a few applications of orientation testing. In each case, express the answer in terms of orientation tests in 2- or 3-dimensions.

(a) You are given four points a, b, c, and d in $\mathbb{R}^2$. Determine whether the line segments $\overline{ab}$ and $\overline{cd}$ intersect (see Fig. 1(a)). (You do *not* need to compute the intersection point itself. If they intersect, you may assume that they intersect in a single point in their interiors.)

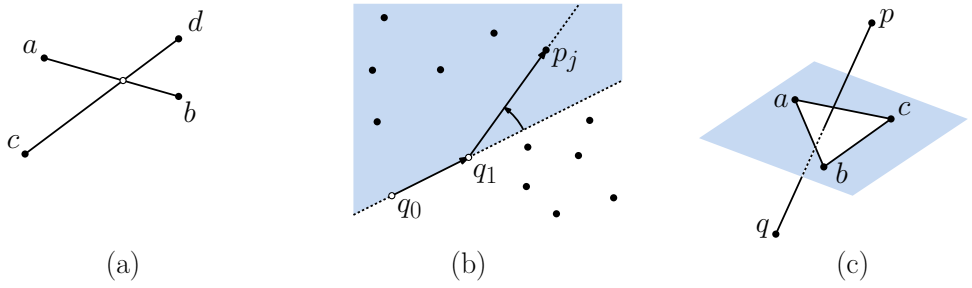


Figure 1: Orientation testing.

(b) You are given a set of points $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^2$ and two additional points q_0 and q_1. Among the points of P that lie strictly to the left of the directed line $\overrightarrow{q_0q_1}$, compute the point p_j that minimizes the counterclockwise angle between the rays $\overrightarrow{q_0q_1}$ and $\overrightarrow{q_1p_j}$ (see Fig. 1(b)). If no point of P lies to the left of $\overrightarrow{q_0q_1}$, your algorithm should report this. Your solution should employ $O(n)$ orientation tests. (Note: The notion of being "left" of a directed line means on the left side of the line with respect to an observer facing the line's direction.)

(c) You are given five points a, b, c, p and q in $\mathbb{R}^3$. Determine whether the line segment $\overline{pq}$ intersects the triangle $\triangle abc$ (see Fig. 1(c)). (You may assume that p and q do not lie in the plane spanned by $\triangle abc$ and that if there is an intersection, it occurs in the interior of the triangle.)

Problem 2. Consider a convex polygon P presented as a counterclockwise sequence of vertices $\langle p_1, \ldots, p_n \rangle$, and let q be a point that is external to P (see Fig. 2). Present an $O(\log n)$-time algorithm that computes a vertex p_j that defines a support line passing through q so that P lies to the left of the directed line $\overrightarrow{qp_j}$. For full credit, your algorithm should access points only through orientation tests.

Hint: Indexing and bisecting circular arrays can be a bit messy given issues with wrapping around. Given two vertices p_i and p_k, let $\text{ccw}(i, k)$ denote the subsequence of vertices along the boundary of P between p_i and p_k in counterclockwise order. You may assume that you have access to a function $\text{bisect}(i, k)$ which in $O(1)$ time returns an index m that splits this vertex chain into two subsequences $\text{ccw}(i, m)$ and $\text{ccw}(m, k)$ each having roughly an equal number of vertices.

Problem 3. The objective of this problem is to explore possible variations in the guessing sequence for Chan's convex hull algorithm for an n-element point set P. Let h denote the size of the final convex hull. Recall that $\text{ConditionalHull}(P, h^*)$ returns the convex hull of P if $h^* \geq h$ and "fail" otherwise, and the algorithm iteratively guesses values of h^* through repeated squaring. Here is a equivalent version of the one given in class.

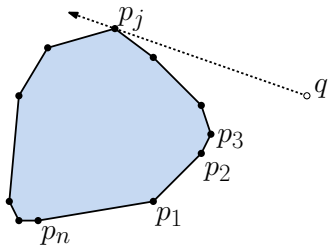


Figure 2: Orientation testing.

(1) $i \leftarrow 1$; status $\leftarrow$ fail
(2) while (status == fail)
  (a) $h^* \leftarrow \min(2^{2^i}, n)$
  (b) status $\leftarrow$ ConditionalHull(P, h^*)
  (c) $i \leftarrow i + 1$
(3) return status

What if we do not use repeated squaring? In each of the following cases, indicate what the running time of Chan's algorithm *would be* had we replaced the expression in line (2a). (Express your answer using O-notation as a function of n and h.) In each case, explain how you derived your answer.

(a) $h^* \leftarrow \min(i^2, n)$ (quadratic progression)
(b) $h^* \leftarrow \min(2^i, n)$ (repeated doubling)
(c) $h^* \leftarrow \min(\sqrt{2}^{\sqrt{2}^i}, n)$

You may assume any standard results on summations, e.g., the Wikipedia page on summations.

Problem 4. A *polygonal chain* in the plane is a sequence of vertices $C = \langle p_1, \ldots, p_n \rangle$, where each consecutive pair (p_i, p_{i+1}) is connected by a line segment, called an *edge*. Such a chain is said to be *strictly horizontally monotone* if any vertical line intersects the chain in a single point. A collection of chains is said to be *independent* if no two intersect each other (see Fig. 3).

Figure 3: A set of independent monotone polygonal chains.

Present an algorithm which, given a set of k strictly monotone polygonal chains $\mathcal{C} = \{C_1, \ldots, C_k\}$, determines whether they are independent. Your algorithm does not need to report the intersections, it just needs indicate whether any intersection exists. Let n_i denote the number of vertices in the ith chain, and let $n = \sum_{i=1}^{k} n_i$ be the total size of all the chains. Your algorithm should run in time $O(n \log k)$. (**Hint:** Use plane sweep, but do it efficiently.)

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.)

Given a set $P = \{p_1, \ldots, p_n\}$ of n points in the plane, the *minimum horizontal trapezoid* is a trapezoid of minimum area that contains P such that two of its sides are horizontal (see Fig. 4(b)). If there are multiple trapezoids of the same area, any one of them may be chosen.

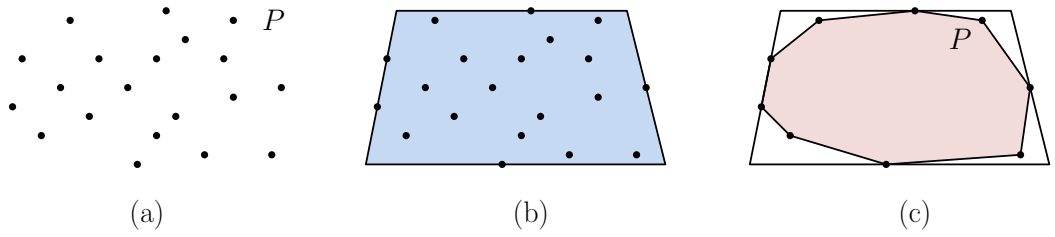


Figure 4: Challenge Problem: Minimum horizontal trapezoid

(a) Clearly, the horizontal sides of the trapezoid pass through topmost and bottommost points of P. What properties should the left and right sides have to yield the minimum area?

(b) Based on your answer to (a), present an $O(n \log n)$ time algorithm for computing the minimum horizontal trapezoid of P.

(c) Suppose that P is given as a convex polygon with n vertices (see Fig. 4(c)). Explain how to compute the minimum horizontal trapezoid in $O(\log n)$ time.

Some tips about writing algorithms: Throughout the semester, 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 textbook on algorithms and data structures. Also, you may use results from geometry that: (1) have been mentioned in class, (2) would be known to someone who knows basic geometry or linear algebra, or (3) is intuitively obvious. If you are unsure, please feel free to check with me.

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.

CMSC 754: Spring 2020 Dave Mount

Homework 2: Plane Sweep and Linear Programming

Handed out Thursday, Feb 27. Due at the start of class on Tuesday, Mar 10. Late homeworks are not accepted (unless an extension has been prearranged) so please turn in whatever you have completed by the due date. Unless otherwise specified, you may assume that all inputs are given in *general position*. Also, when asked to give an algorithm with running time $O(f(n))$, it is allowed to give a *randomized* algorithm with *expected* running time $O(f(n))$.

Problem 1. You are given an axis-parallel rectangle R and a collection of closed circular disks $D = \{d_1, \ldots, d_n\}$. Assume that each disk d_i is represented by its center point p_i, which lies within R, and its (positive real) radius r_i. Note that the disks may extend outside of R, and one disk may be contained within another.

(a) The elements of D are said to form a *packing* of R if every point of R lies within *at most* one disk of D (see Fig. 1(a)). Present a plane-sweep algorithm that determines whether D is a packing of R. **Hint:** $O(n \log n)$ time is possible.

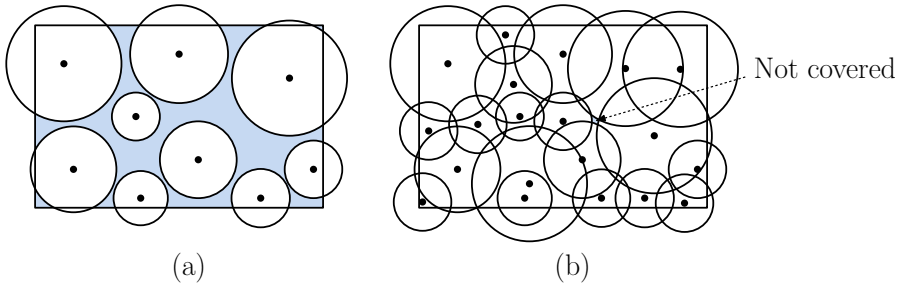


Figure 1: Problem 1: Packing and covering a rectangle

(b) The elements of D are said to form a *cover* of R if every point of R lies within *at least* one disk of D. (The disks of Fig. 1(b) fail to be a cover because of the small gap shown in the figure.) Present a plane-sweep algorithm that determines whether D is a cover of R.

Hint: $O((n + m) \log n)$ time is possible, where m is the number of intersection points lying within R between the boundaries of disks. The running time should *not* depend on the number of intersection points that lie outside of R.

In both cases, you may assume access to primitive operations on circles and disks (e.g., computing the intersection points of two circles or the containment of one disk within another). As always, briefly justify your algorithms' correctness and derive their running times.

Note on terminology: By the term *circle*, we mean the set of points of $\mathbb{R}^2$ that are equidistant from a center point. The associated *closed disk* is the set of points that lie at or within this distance, and the associated *open disk* is the set of points that lie strictly closer to the center. Given a center point $p \in \mathbb{R}^2$ and radius $r \geq 0$, we can define the circle, closed disk, and open disk as the set of points $q \in \mathbb{R}^2$ such that $\|q - p\| = r$, $\|q - p\| \leq r$ and $\|q - p\| < r$, respectively, where $\|q - p\|$ denotes the Euclidean distance between q and p.

Problem 2. Explain how to solve each of the following problems in linear (expected) time. Each can be modeled as a linear programming (LP) problem, perhaps with some additional pre- and/or post-processing. In each case, explain how the problem is converted into an LP instance and how the answer to the LP instance is used/interpreted to solve the stated problem.

(a) You are given two point sets $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$ in the plane. Let $p_i = (p_{i,x}, p_{i,y})$ and $q_i = (q_{i,x}, q_{i,y})$. You are told that these two sets are separated by a vertical line $x = x_0$, with P to the left and Q to the right (see Fig. 2(a)). Show how to efficiently compute the line equations (in the form $y = ax + b$) for each of the following "tangent lines" (that is, support lines) of both conv(P) and conv(Q).

(i) ℓ_1: Lies above both P and Q
(ii) ℓ_2: Lies below both P and Q
(iii) ℓ_3: Lies above P and below Q
(iv) ℓ_4: Lies below P and above Q

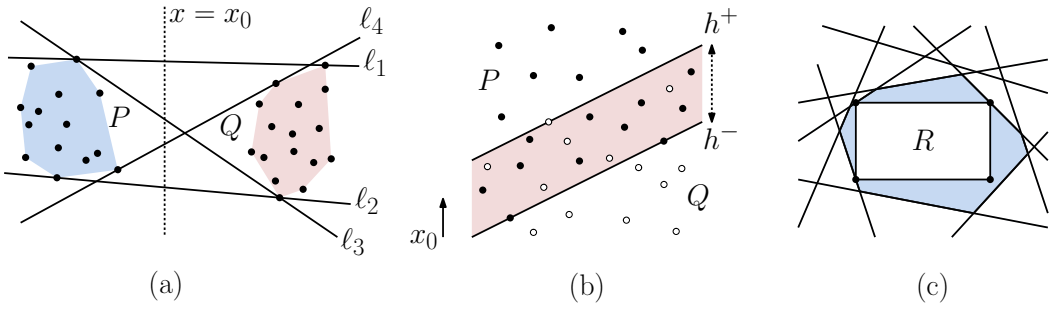


Figure 2: Problem 2: LP applications

Note that the convex hull is *not* given, just the points. In each case, your algorithm should run in time $O(n + m)$. (Briefly justify correctness and explain the running time in each case.)

(b) The following problem is related to the convex of *support vector machines* from machine learning. We are working in $\mathbb{R}^d$ (where d is a constant), where each point x is represented by a coordinate vector $(x_0, \ldots, x_{d-1})$. For the sake of illustration, let's think of the x_0 axis as pointing "upwards". Any nonvertical hyperplane can be expressed as d-vector of real coefficient $a = (a_0, \ldots, a_{d-1})$, by the equation $x_0 = a_0 + \sum_{i=1} d - 1 a_i x_i$.

You are given two sets of points $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$ in $\mathbb{R}^d$. Explain how to compute the equations of two parallel hyperplanes:

$$h^+ : x_0 = a_0^+ + \sum_{i=1}^{d-1} a_i x_i \qquad \text{and} \qquad h^- : x_0 = a_0^- + \sum_{i=1}^{d-1} a_i x_i,$$

such that every point of P lies on or above h^- and every point of Q lies on or below h^+ (see Fig. 2(b)).

The *signed vertical distance* between h^+ and h^- is the vertical distance between these hyperplanes if h^+ lies above h^- and the negation of this distance otherwise. Your hyperplanes should be chosen to minimize the signed vertical distance between them. (Thus, if h^+ lies below h^-, you want to maximize the distance between them.)

Your algorithm should run in time $O(n + m)$. (Briefly justify your algorithm's correctness and explain its running time.)

(c) You are given a set of n halfplanes $H = \{h_1, \ldots, h_n\}$ in the plane. For concreteness, let us assume that each halfplane h_i is given by the inequality $a_i x + b_i y \leq 1$, for some real coefficients (a_i, b_i). Show how to compute the axis-parallel rectangle R of maximum perimeter that lies within the intersection of these halfplanes. (You may assume that R is given by the x- and y-coordinates of its left, right, top, and bottom sides.)

Your algorithm should run in time $O(n)$. (Briefly justify your algorithm's correctness and explain its running time.)

Problem 3. The purpose of this problem is to get practice with designing and analyzing randomized incremental algorithms. You are to devise a randomized algorithm for computing the upper hull of a set of points P in the plane. The approach is as follows:

(1) Deterministically, compute the leftmost and rightmost points, denoted v_1 and v_n, respectively. The initial upper hull is the segment $\overline{v_1, v_n}$.

(2) Randomly permute the remaining points, which we denote as $\langle p_2, p_3, \ldots, p_{n-1} \rangle$.

(3) For i running from 2 to $n-1$, add point p_i to the hull as follows (see Fig. 3(a)):

  (a) Determine the edge $\overline{v_j v_{j+1}}$ of the hull lying above or below p_i (see Fig. 3(b)).

  (b) If p_i lies below this edge, then ignore it.

  (c) Otherwise, walk outwards to the left and right of $\overline{v_j v_{j+1}}$ to compute points of tangency with respect to p_i, and delete any vertices that lie beneath these tangent lines. Add p_i to the hull.

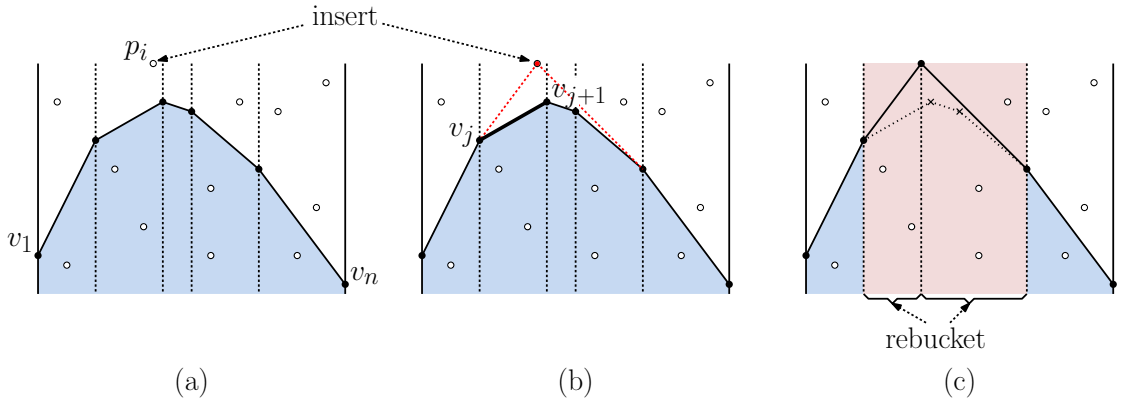


Figure 3: Incremental construction of the upper hull.

How do we determine the edge lying immediately above or below p_i? We maintain a collection of *buckets*, one per edge of the hull, where each contains all the points that lie vertically above or below this edge. Whenever a new point is added to the hull, the points lying within the bucket of a deleted edge need to be "rebucketed". We assume that the points within each bucket are sorted by their x-coordinates. We merge the points from the deleted buckets (in time proportional to the number of points) and then walk through this sorted list and assign these points to their new buckets.

(a) Show that (ignoring the time for rebucketing) the above algorithm runs in $O(n)$ time.

(b) Show that, assuming that points are inserted in random order, the expected number of times any point is rebucketed is $O(\log n)$. In particular, show that for any point p, the probability that p is rebucketed during the ith insertion is at most c/i, where c is a constant. What is the value of c?

It is interesting to note that, unlike the other convex hull algorithms we have seen so far, this one generalizes to arbitrary dimensions.

Problem 4. In this question, we will explore an alternate way in which to define point-line duality in the plane. (This generalizes to any dimension, but the plane is the simplest case to work with.) Any line ℓ in the x, y-plane that does not pass through the origin can be expressed by the equation $\ell : ax + by = 1$. Let us define the *polar dual* of ℓ, denoted ℓ^* to be the point (a, b). Similarly, given a point $p = (a, b)$, define its dual to be the line $p^* : ax + by = 1$. Clearly, this mapping is self-inverse, since $p^{**} = p$ and $\ell^{**} = \ell$.

(a) Given a line $\ell : ax + by = 1$, define its *(open) inner halfplane*, denoted $\ell_<$, to be set of points that lie on the same side of ℓ as the origin, that is $(x, y) \in \ell_<$ if $ax + by < 1$ (see Fig. 4(a)). Define $\ell_>$ analogously for points on the opposite side of ℓ, and define $\ell_\leq$ and $\ell_\geq$ analogously for the closed halfplanes. We say that point p is *inside* (resp., *outside*) ℓ if $p \in \ell_<$ (resp. $\ell_>$).

Prove that polar duality is "order reversing" by showing that point p lies inside/on/outside of ℓ if and only if ℓ^* lies inside/on/outside of p^*, respectively (see Fig. 4(b)).

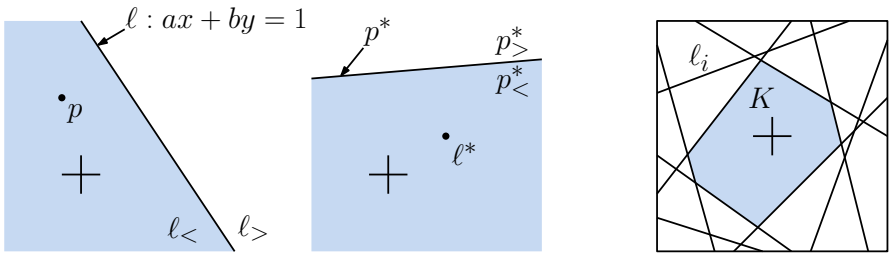


Figure 4: Polar dual.

(b) Prove that polar duality is "incidence preserving" by showing that ℓ_1 and ℓ_2 intersect at point p if and only if the dual line p^* passes through dual points ℓ_1^* and ℓ_2^*.

(c) Let $L = \{\ell_1, \ldots, \ell_n\}$ be a set of lines in $\mathbb{R}^2$ and let K be the intersection of the associated (closed) inner halfplanes, that is, $K = \bigcap_{i=1}^n \ell_\leq$ (see Fig. 4(c). Let's assume that K is not empty (which implies that it contains the origin).

Describe the relationship between K and the convex hull of the polar dual set of points $L^* = \{\ell_1^*, \ldots, \ell_n^*\}$? In particular, explain how the CCW order of edges around the boundary of K relates to the order of points around $\text{conv}(L^*)$. Justify your answer.

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.

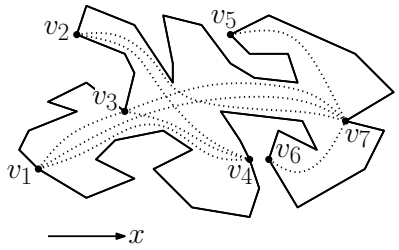


Figure 5: Some of the monotonically reachable pairs for the vertices shown: $\{v_1, v_3\}$, $\{v_1, v_4\}$, $\{v_1, v_7\}$, $\{v_2, v_4\}$, $\{v_2, v_7\}$, $\{v_3, v_4\}$, $\{v_3, v_7\}$, $\{v_5, v_7\}$, $\{v_6, v_7\}$.)

Hint: Note that the number of pairs can be quadratic in n. To achieve a running time of $O(n \log n)$ you cannot count pairs one by one, you need to count them in groups.

CMSC 754: Spring 2020 Dave Mount

Sample Problems for the Midterm Exam

The midterm exam will be this **Thursday, March 12 in class**. It will be *closed-book* and *closed-notes*, but 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.

Disclaimer: The following sample problems have been collected from old homeworks and exams. Because the material and order of coverage varies each semester, these problems *do not* necessarily reflect the actual length, coverage, or difficulty of the midterm exam.

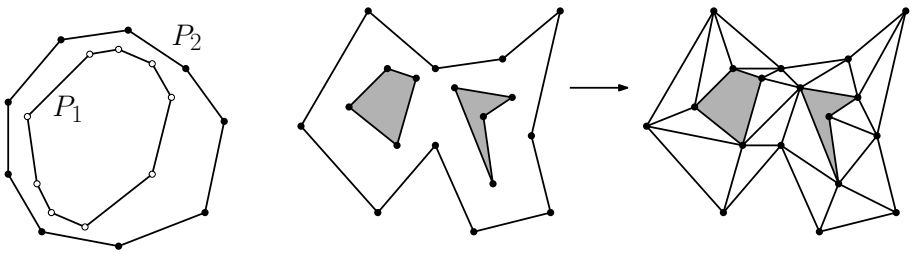
Problem 1. Give a short answer to each question (a few sentences suffice).

(a) Explain how to use *at most three* orientation tests to determine whether a point d lies within the interior of a triangle $\triangle abc$ in the plane. You do *not* know whether $\triangle abc$ is oriented clockwise or counterclockwise (but you may assume that the three points are not collinear).

(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) Recall that in a simple polygon, a *scan-reflex vertex* is a vertex having both incident edges on the same side of a vertical line passing through the vertex. Given a simple polygon P with n vertices and r scan-reflex vertices, what is the maximum and minimum number of diagonals that might be needed to decompose it into monotone pieces? Explain briefly.

(d) A convex polygon P_1 is enclosed within another convex polygon P_2 (see Fig. 1(a)). 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 sets of dual lines.



(a) (b)

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

(e) 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(b), $n = 10$, $h = 2$, and $k = 4$). As a function of n, h and k, how many triangles will such a triangulation have? Explain briefly.

(f) A trapezoidal map of n segments has roughly $6n$ vertices and roughly $3n$ trapezoids. Explain (e.g., via a charging argument) where the numbers 6 and 3 come from.

Problem 2. 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. 2(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. 2(b)).

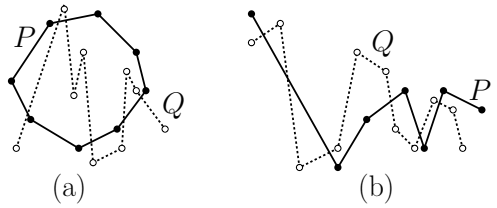


Figure 2: Maximum number of intersections.

(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 3. 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 the "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 4. You are given a set of n vertical line segments in the plane $S = \{s_1, \ldots, s_n\}$, where each segment s_i is described by three values, its x-coordinate x_i, its upper y-coordinate y_i^+ and its lower y-coordinate y_i^-. Present an efficient an algorithm to determine whether there exists a line $\ell : y = ax + b$ that intersects all of these segments (see Fig. 3). Such a line is called a *transversal*. (Hint: $O(n)$ time is possible.) Justify your algorithm's correctness and derive its running time.

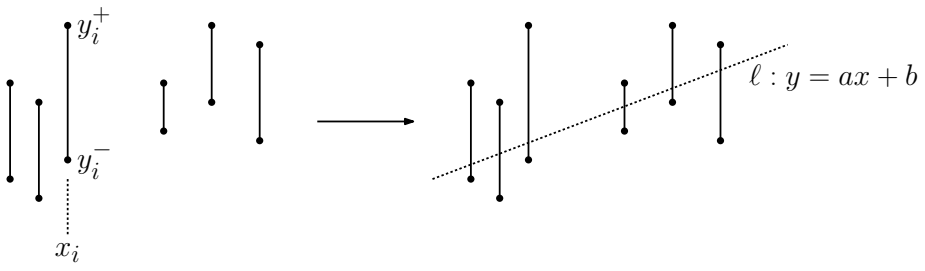


Figure 3: Existence of a transversal.

Problem 5. A *slab* is the region lying between two parallel lines. You are given a set of n slabs, where each is of vertical width 1 (see Fig. 4). Define the *depth* of a point to be the number of slabs that contain it. The objective is to determine the maximum depth of the slabs using plane sweep. (For example, in Fig. 4 the maximum depth is 3, as realized by the small triangular face in the middle.)

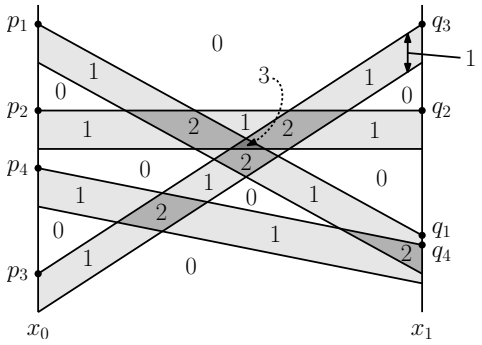


Figure 4: Maximum depth in a set of slabs.

We assume that the slabs lie between two parallel lines at $x = x_0$ and $x = x_1$. The ith slab is identified by the segment $\overline{p_i q_i}$ that forms its upper side (and the lower side is one unit below this). Let I denote the number of intersections between the line segments (both upper and lower) that bound the slabs. Present an $O((n + m) \log n)$-time algorithm to determine the maximum depth. (Hint: Use plane-sweep.)

Problem 6. 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. 5). 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.

Problem 7. 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. (Hint: It may be easier to consider the question in its inverse form, do the convex hulls *not* intersect.)

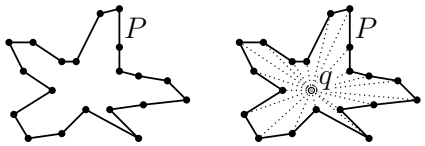


Figure 5: Determining whether a polygon is star-shaped.

Problem 8. 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. 6(a)).

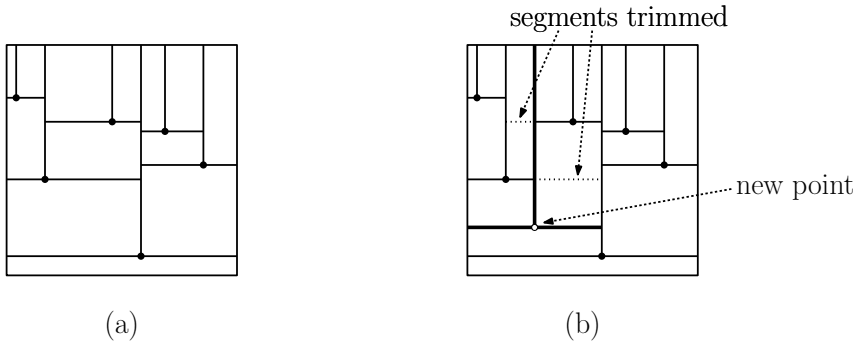


Figure 6: Building a subdivision.

(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. 6(b)).

(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 9. 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 *Pareto maximal* if it is not dominated by any other point of P (shown as black points in Fig. 7(b)).

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 $j \geq i$.)

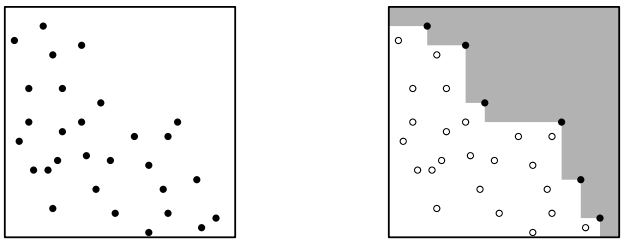


Figure 7: Pareto maxima.

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

Problem 10. Consider an n-sided simple polygon P in the plane. Let us suppose that the leftmost edge of P is vertical (see Fig. 8(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.)

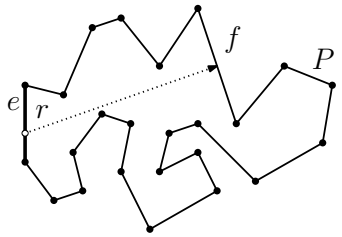


Figure 8: Ray-shooting queries.

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 inputs 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. (20 points) Consider the two segments s_1 and s_2 shown in Fig. 1(a).

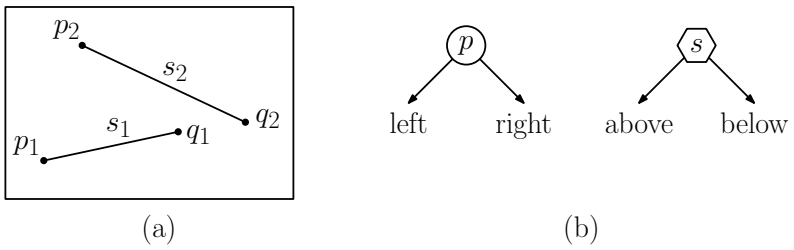


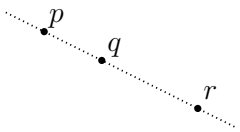
Figure 1: Trapezoidal map and point location.

(a) (5 points) Show the (final) *trapezoidal map* for these two segments, assuming the insertion order $\langle s_1, s_2 \rangle$. Draw over the figure below.

(b) (15 points) Show the *point-location data structure* resulting from the construction given in class, assuming the insertion order $\langle s_1, s_2 \rangle$. Recall the nodes types in Fig. 1(b). (Note: We will give 50% partial credit if your data structure works correctly, even if it is not the same as the one from class.)

Problem 2. (20 points; 4–6 points each) Short-answer questions.

(a) You have three distinct, collinear points in the plane that appear in the left-to-right order p, q, r (see the figure below). Assume the dual transformation given in class, which maps the point (a, b) to the line $y = ax - b$. What can you assert about the relationship between the dual lines p^*, q^* and r^*? (Be as specific as possible.)



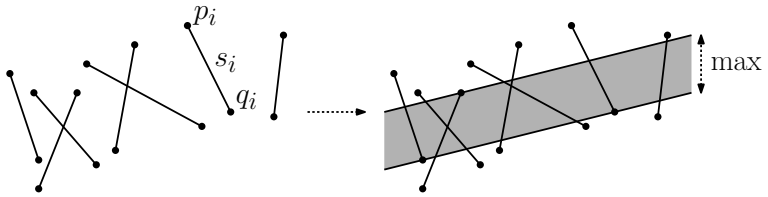
(b) You have two convex polygons P and Q, each having exactly n vertices. No two edges of P and Q are collinear. As a function of n, what is the maximum number of times the boundaries of P and Q can intersect? (No proof needed.) For $n = 5$, give an example (two convex pentagons) that illustrates your bound.

(c) Consider the linear-programming algorithm given in class for n constraints in dimension 2. In class we showed that the *expected-case* running time of the algorithm is $O(n)$. What is the *worst-case* running time of the algorithm? Briefly justify your answer (in a sentence or two).

(d) It is a fact that if P is a uniformly distributed random set of n points in a circular disk in the plane, the expected number of vertices of P's convex hull is $\Theta(n^{1/3})$. That is, the lower and upper bounds are both within some constant of $n^{1/3}$ for large n.

What is the average-case running time of Jarvis's algorithm for such an input? (If you forgot the running time of Jarvis's algorithm, we will give it to you for a 50% penalty on this problem.)

Problem 3. (15 points) You are given a set of line segments $S = \{s_1, \ldots, s_n\}$ in the plane, where $s_i = \overline{p_i, q_i}$ (see the figure below). A *slab* is the region of the plane between two parallel lines. The *vertical width* of the slab is the length of the slab's intersection with the y-axis. Present an efficient algorithm to compute the slab of maximum vertical width that stabs all the segments, so that for all i, p_i lies above the slab and q_i lies below the slab. If no such slab exists, your algorithm should indicate this. (**Hint:** This is possible in $O(n)$ time, but $O(n \log n)$ is acceptable for partial credit.)



Problem 4. (20 points) You are given an axis-parallel rectangle R in the plane and a set of line segments $S = \{s_1, \ldots, s_n\}$ that lie within R. The segments may intersect only at their endpoints, and their endpoints may lie on the boundary of R (see Fig. 2(a)).

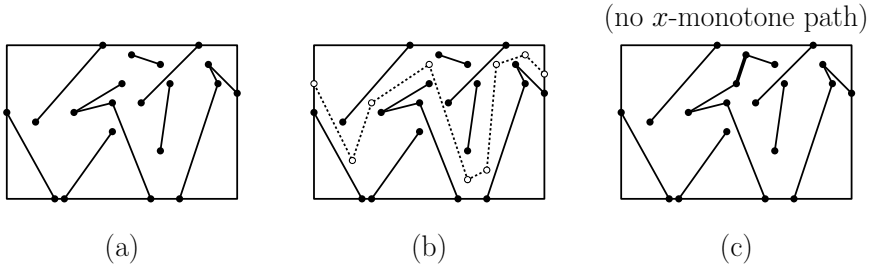


Figure 2: Monotone path.

Recall that a polygonal chain is *x-monotone* if any vertical line intersects the chain in at most one point. Present an algorithm which, given R and S, determines whether there exists an x-monotone polygonal chain, that does not intersect the interior of any segment, that starts anywhere on the left side of R, and goes to anywhere on the right side of R. Your algorithm *does not* need to output the path, simply "yes" or "no." (For example, the input from Fig. 2(b) the answer is "yes" as illustrated by the dotted path, and for the input from

Fig. 2(c) the output is "no".) Briefly justify your algorithm's correctness and derive its running time.

(**Hints:** For full credit, your algorithm should run in $O(n \log n)$ time. You may modify any algorithm from class.)

Problem 5. (25 points) A polygonal chain in the plane is *cyclically monotone* with respect to a point O if every ray emanating from O intersects the chain in a single point (see Fig. 3(a)). Henceforth, let us take O to be the origin of the coordinate system.

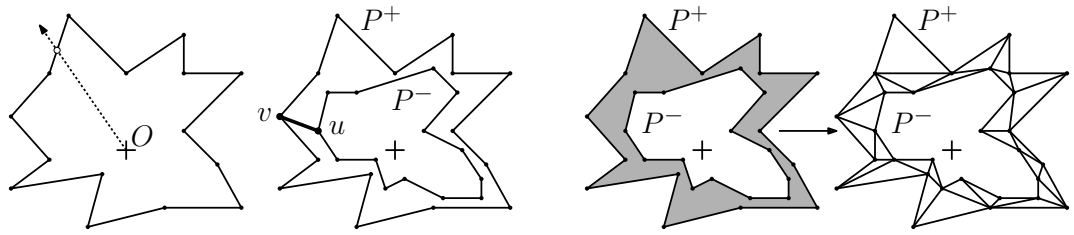


Figure 3: Triangulating and cyclically monotone region.

You are given two simple polygons, P^- and P^+, whose boundaries are cyclically monotone. These two chains do not intersect each other, and P^- is nested within P^+. Let n denote the total number of vertices on P^- and P^+.

(a) (10 points) Define a *cross diagonal* to be a line segment $\overline{uv}$, where u is a vertex of P^-, v is a vertex of P^+, and this segment lies entirely within the region between P^- and P^+ (see Fig. 3(b)). Present an efficient algorithm, which given just P^- and P^+, computes any one cross diagonal $\overline{uv}$. Briefly justify your algorithm's correctness and derive its running time. (**Hint:** This is possible in $O(n)$ time, but $O(n \log n)$ is acceptable for partial credit.)

(b) (15 points) Give an algorithm to triangulate the region between P^- and P^+ (see Fig. 3(c)). Briefly justify your algorithm's correctness and derive its running time. (**Hint:** This is possible in $O(n)$ time, but $O(n \log n)$ is acceptable for partial credit. It may help to assume that you are given a cross diagonal to start. You may modify an algorithm given in class.)

CMSC 754: Spring 2020 Dave Mount

Homework 3: Voronoi Diagrams, Delaunay Triangulations, and More

Handed out Thu, April 16. Due **Thu, April 30, 9:30am**. (**Updated!**) Late homeworks are not accepted (unless an extension has been prearranged) so please turn in whatever you have completed by the due date. Unless otherwise specified, you may assume that all inputs are in *general position*. Whenever asked to give an algorithm running in $O(f(n))$ time, you may give a *randomized algorithm* whose expected running time is $O(f(n))$.

Problem 1. In class we proved that the Delaunay triangulation of a set of sites in the plane maximizes the minimum angle. (It is the max-min angle triangulation.) Unfortunately, it is not the best triangulation for the following two criteria.

(a) Give an example of a set of point sites in the plane such that the Delaunay triangulation of this set *does not* minimize the sum of edge lengths, among all possible triangulations. In other words, the Delaunay triangulation is *not* the minimum-weight triangulation.

(b) Give an example of a set of point sites in the plane such that the Delaunay triangulation of this point set *does not* minimize the maximum angle, among all possible triangulations. In other words, the Delaunay triangulation is *not* the min-max angle triangulation.

In each case briefly explain your construction. Your example should be in general position (e.g., no four sites should be cocircular).

Hint: In both cases, it is possible build a counterexample consisting of just four points that are nearly co-circular. It suffices to present a single, specific example.

Problem 2. The Delaunay triangulation of a convex polygon is defined to be the Delaunay triangulation whose sites are the vertices of the polygon. As usual, let us assume that the vertices $V = \{v_1, \ldots, v_n\}$ of the polygon are presented in counterclockwise order around the polygon. Also we assume that $n \geq 3$ and no three vertices are collinear. In this problem, we will analyze a randomized incremental algorithm for constructing the Delaunay triangulation of a convex polygon.

The algorithm is similar to the randomized incremental algorithm given in class, but with the following differences. First, we *do not* use any sentinel sites, just the points themselves. We begin by permuting the points randomly. Let $P = \langle p_1, \ldots, p_n \rangle$ denote the sequence of vertices after this permutation has been applied. We start with the triangle $\triangle p_1 p_2 p_3$. Then, we go through the points p_4 through p_n, adding each one and updating the Delaunay triangulation as we go. The insertion process for p_i involves the following steps:

(i) Determine the edge ab of the current convex hull that is visible to p_i (see Fig. 1(a)).

(ii) Connect p to the convex hull by adding the edges ap_i and $p_i b$ to the convex hull (see Fig. 1(b)).

(iii) As in Lecture 13, perform repeated edge flips until all the triangles incident to p_i satisfy the local Delaunay condition (see Fig. 1(c)).

Answer the following questions about this algorithm:

(a) Prove that in any triangulation of an n-sided convex polygon, the number of triangles is $n - 2$ and the number of edges (including the edges of the convex hull) $2n - 3$.

(b) What is the average degree of a vertex in the Delaunay triangulation of n? (Include the two edges of the convex hull that are incident to this vertex.) Derive your answer.

(c) Apply a backwards analysis to bound the expected number of edge flips performed when the ith site is inserted into the diagram (for $4 \leq i \leq n$).

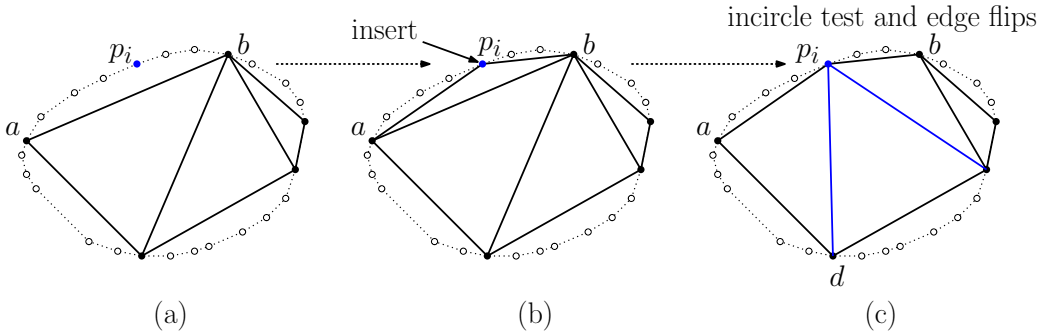


Figure 1: Delaunay triangulation of a convex point set.

(d) In step (i) of the above algorithm, we need to determine the edge ab of the convex hull that is visible to the new site p_i. Describe a method for answering these queries and derive its expected running time. (Hint: As we did in the standard Delaunay Triangulation algorithm, apply some form of bucketing combined with a backwards analysis.) $O(n \log n)$ total expected time is possible, but if you think you can do this in $O(n)$ expected time, see the challenge problem.

Problem 3. In this problem, we will derive a complete version of the empty-circumcircle test for four points in $\mathbb{R}^2$, which considers both the affine and circular parts of the test. Let a, b, and c be three distinct, non-collinear points in the plane, and let us assume that they have been labeled so that $\text{Orient}(a, b, c) > 0$ (that is, they are given in counterclockwise order). Let d be any point in plane. Throughout this problem, you may assume general position: No three points are collinear and no four points are cocircular.

You may access the points *only* through the two determinant-based functions $\text{Orient}(a, b, c)$ (from Lecture 2) and $\text{inCircle}(a, b, c, d)$ (from Lecture 13).

(a) Consider the subdivision of the plane induced by the three lines defining the edges of triangle $\triangle abc$ (see Fig. 2(a)). Present a short code fragment that uses orientation tests to label a point d according to the region in which it lies. For example, given the point d shown in the figure, your code fragment should output the label "2". (Hint: It may simplify your code to think of the labels as binary numbers, e.g., $2 = 010_2$.)

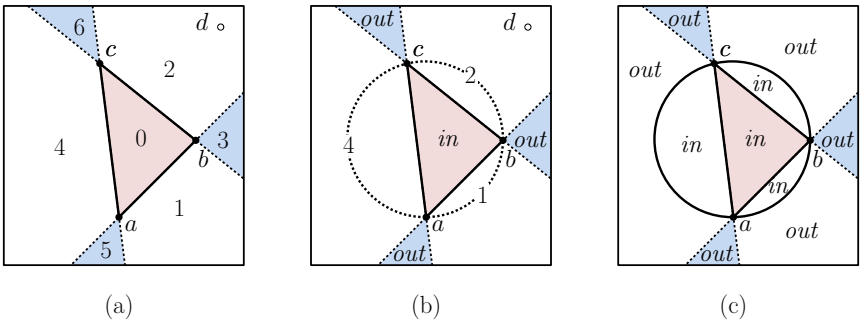


Figure 2: InCircle testing.

(b) Clearly, if d lies in region 0 it is inside the circumcircle of $\triangle abc$, and if it lies in any of the regions 3, 5, or 6, it is outside this circumcircle. Explain how to apply the inCircle function from Lecture 13 to classify d as lying inside or outside the circumcircle of $\triangle abc$ for the remaining regions 1, 2, and

4. (Hint: Recall that this test is based on the sign of a determinant, which was derived under the assumption that the arguments are presented in counterclockwise order.)

Problem 4. Computational biologist often compute physical properties of large molecules. One of these properties is something called the *accessible surface area* of the molecule. We will consider this problem in a 2-dimensional setting.

We model the atoms of our molecule as a set of unit disks, each of radius of 1, centered at a given set of points $P = \{p_1, \ldots, p_n\}$. Let b_i denote the unit disk centered at point p_i. Let $M(P) = \bigcup_{i=1}^{n} b_i$ denote the union of these disks (shaded in blue in Fig. 3(a)). Let $\partial M(P)$ denote its boundary. Observe that $\partial M(P)$ is composed of circular arcs, and it may have multiple connected components (all of which contribute to the accessible surface).

(a) Present an $O(n \log n)$-time algorithm, which given a set P of n points in the plane, computes the length of accessible surface (see the solid curve in Fig. 3(a)).

Note: I am mostly interested in how to identify the circular arcs that make up the boundary of $M(P)$, not the actual perimeter value. To simplify matters, you may assume that you have access to a black-box function that computes the length of a given circular arc.

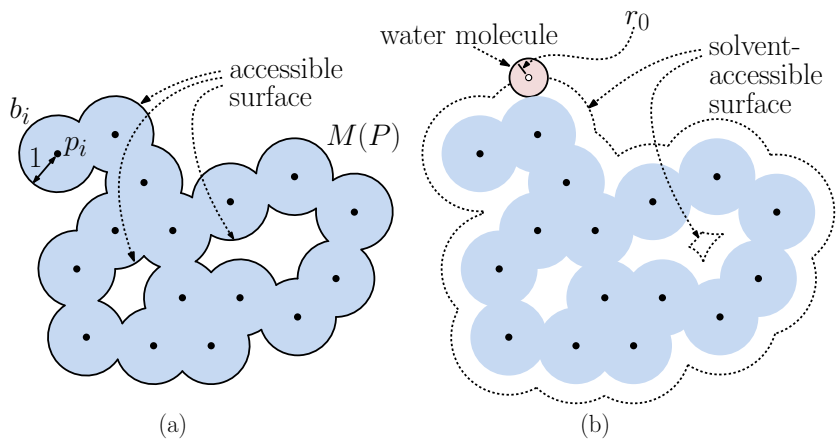


Figure 3: Accessible surface.

(b) In most applications in drug design, the molecule resides in liquid solvent, such as water. A statistic that is more relevant is something called the *solvent-accessible surface*. We model a water molecule as a disk of some radius r_0. Imagine that we roll such a disk around the entire boundary of $M(P)$ and trace out the path taken by the center of this disk (the dashed curve in Fig. 3(b)). As with the accessible surface, this boundary may have multiple components, all of which contribute to the final result.

Present an $O(n \log n)$-time algorithm, which given P and r_0, computes the length of the solvent-accessible surface (the dashed curve in Fig. 3(b)).

Hint: This should be an easy extension to (a).

Problem 5. In linear classification in machine learning it is desirable to determine whether two point sets in $\mathbb{R}^d$ can be partitioned by a hyperplane. We will consider the problem in a 2-dimensional setting. When a perfect partition is not possible, one approach is to find a line that achieves the best possible split. Let A and B be two point sets in the plane. Given a nonvertical line $\ell : y = ax - b$, let ℓ^+ and ℓ^- denote the halfplanes lying above and below ℓ, respectively. Define ℓ's *separation defect* to be

$$\delta(\ell) \;=\; \min\big(|A \cap \ell^+| + |B \cap \ell^-|,\;\; |A \cap \ell^-| + |B \cap \ell^+|\big).$$

So, if ℓ separates A from B the separation defect is zero. Otherwise, it is equal to the number of points that fall on the "wrong" side of the separating line. (For example, the line shown in Fig. 4 has separation defect of three.) In this problem, we will derive a roughly quadratic-time algorithm to compute a line of minimum separation defect.

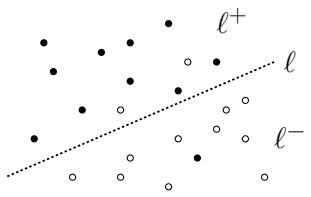


Figure 4: Problem 5: The line ℓ has separation defect 3.

(a) Given two point sets A and B in $\mathbb{R}^2$ and a line ℓ, what is the meaning of ℓ's separation defect in the dual setting? Explain briefly. Assume the standard dual transformation $(a, b) \leftrightarrow y = ax - b$.

(b) Letting $n = |A| + |B|$, present an $O(n^2 \log n)$-time algorithm to compute a line of minimum separation defect. Briefly explain your algorithm and derive its running time. (Note: There may generally be many lines achieving the minimum defect. Your algorithm may output any one of them. If a point lies on ℓ you have a choice of which halfplane to assign it. I will let you decide how to handle this. You may either have your algorithm make the choice explicitly to minimize the defect, or you can just assume that ℓ is chosen so that no point of either set lies on it.)

(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.)

Challenge Problem 1. Show that Problem 2(d) (locating points in the incremental DT algorithm for convex hulls) can be solved in $O(n)$ time in expectation.

Challenge Problem 2. Prove that if a, b, and c are given in counterclockwise order, the InCircle determinant is positive if and only if d lies inside the circumcircle of $\triangle abc$. (This implies that the affine test used in Problem 3(a) is not needed.)

CMSC 754: Spring 2020 Dave Mount

Homework 4: Arrangements and Approximations

Handed out Fri, May 1. Due at the start of class on Tuesday, May 12. Late homeworks are not accepted (unless an extension has been prearranged) so please turn in whatever you have completed by the due date. Unless otherwise specified, you may assume that all inputs are in *general position*. Whenever asked to give an algorithm running in $O(f(n))$ time, you may give a *randomized algorithm* whose expected running time is $O(f(n))$.

Problem 1. Given a set P of n points in $\mathbb{R}^2$ in general position, and given a nonvertical line ℓ, project the points orthogonally onto ℓ and consider their left-to-right order (see Fig. 1). The result is called an *allowable permutation*. (Let us assume that ℓ is chosen so no two points have the same projection.)

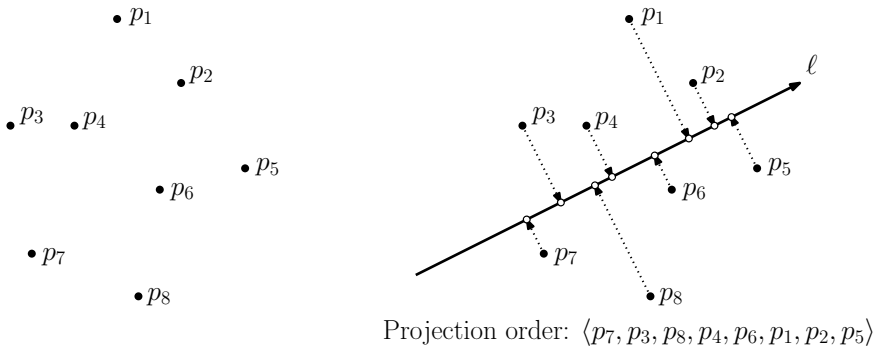


Figure 1: Allowable permutation

In general there are an exponential number ($n!$) of distinct permutations of n points. Prove that the number of allowable permutations is only $O(n^2)$. (Hint: Explain how each allowable permutation is manifested in the dual arrangement.)

Problem 2. You are given three sets of points R, G, and B (red, green, and blue) in $\mathbb{R}^2$. A *tricolor slab* is a pair of parallel lines such that the closed region bounded between these two lines contains at least one point from each of R, G, and B. Define the slab's *height* to be the vertical distance between these lines (see Fig. 2).

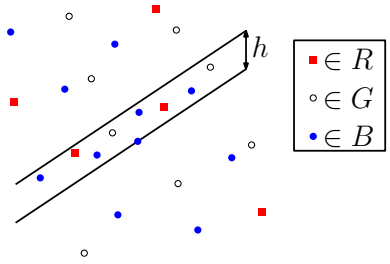


Figure 2: Tricolor slab of height h.

(a) There are infinitely many a tricolor slabs. Assuming that the point set $R \cup G \cup B$ is in general position, prove that a tricolor slab of minimum height has a point of each color on its boundary, with two points on one line and one point on the other.

(b) Assuming the standard dual transformation (mapping point (a, b) to line $y = ax - b$), explain what a tricolor slab of minimum vertical height h corresponds to in the dual setting.

(c) Present an algorithm, which given inputs R, G, and B, computes the tricolor slab of minimum vertical height. Your algorithm should run in $O(n^2 \log n)$ time, where $n = |R| + |G| + |B|$. Derive your algorithm's running time and justify its correctness.

Problem 3. You are given two sets of points in $\mathbb{R}^d$, called R (for red) and B (for blue). A *bichromatic pair* is any pair of points (p, q), where $p \in R$ and $q \in B$. The *bichromatic diameter* is defined to be the bichromatic pair of maximum distance: $\max_{p \in R, q \in B} \|p - q\|$.

Given $\varepsilon > 0$, an *ε-approximation* to the bichromatic diameter is a pair $p' \in R$ and $q' \in B$, such that

$$\frac{\|p - q\|}{1 + \varepsilon} \;\leq\; \|p' - q'\| \;\leq\; \|p - q\|,$$

where p and q are the true bichromatic diameter.

Present an efficient algorithm which, given R, B, and $\varepsilon > 0$, computes an ε-approximation to the bichromatic diameter. Your algorithm should run in time $O(n \log n + n/\varepsilon^d)$ time, where $n = |R| + |B|$. Derive your algorithm's running time and justify its correctness.

Problem 4. You are given a set $P = \{p_1, \ldots, p_n\}$ of n points in $\mathbb{R}^d$. Given a positive real r, the *proximity index*, denoted $\Psi_P(r)$ is the number of pairs of points of P that lie within distance r of each other. (Formally, $\Psi_P(r)$ is the number of pairs $1 \leq i < j \leq n$ such that $\|p_i - p_j\| \leq r$.)

Given P and r, we can easily compute $\Psi_P(r)$ in $O(n^2)$ time by inspecting all pairs of points, but since n is very large, this is too slow. Given a constant $\varepsilon > 0$, an *ε-approximation* to $\Psi_P(r)$ is an integer X from 0 to $\binom{n}{2}$, such that

$$\Psi_P\left(\frac{r}{1 + \epsilon}\right) \;\leq\; X \;\leq\; \Psi_P((1 + \epsilon)r).$$

Present an algorithm running in time $O(n \log n + n/\epsilon^d)$ which, given P, r, and ε, computes such an approximation. (Hint: You may assume that, when a quadtree is computed, each node u of the quadtree can be associated with an integer $\mathrm{wt}(u)$, which indicates the number of points of P lying within u's subtree.)

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

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.

Challenge Problem 1. Modify your solution to Problem 2 (tricolor slab), so that instead of minimizing the vertical distance between the lines of the slab, it minimizes the *perpendicular distance* between these lines. The running time should be the same.

Challenge Problem 2. You are given two sets of points B and R (called, blue and red, respectively) in $\mathbb{R}^2$, each of size n. Give an $O(n^2 \log n)$ time algorithm which determines whether there exists a line ℓ such that the orthogonal projections of the points of $B \cup R$ onto this line alternate between blue and red (see Fig. 3). (It does not matter which color the sequence begins or ends with, just that there is no consecutive red-red or blue-blue in the projected order.)

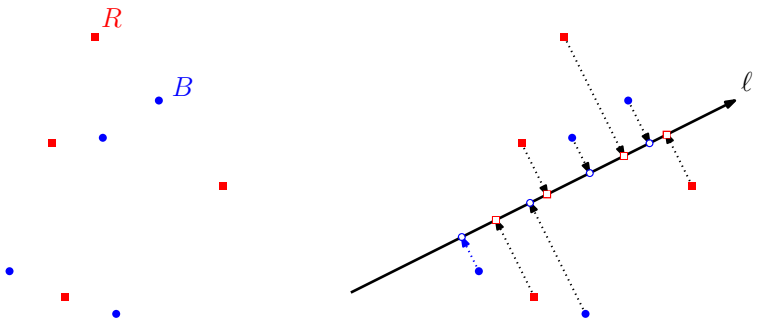


Figure 3: Red-blue alternating projection

CMSC 754: Spring 2020 Dave Mount

Sample Problems for the Final Exam

The final exam will be released **Fri, May 15, 8:00am EDT** (on the class handouts page) and will be due through Gradescope upload by **Mon, May 18, 8:00am EDT**.

Disclaimer: 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.

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 0. You should expect a problem that involves working through some algorithm covered in class on a given input set.

Problem 1. Give a short answer (a few sentences) to each question. Unless explicitly requested, explanations are not required, but may be given for partial credit.

(a) A *dodecahedron* is a convex polyhedron that has 12 faces, each of which is a 5-sided pentagon. Every vertex has degree 3. How many vertices and edges does the dodecahedron have? Show how you derived your answer.

(b) 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.)

(c) When the ith site is added to the Delaunay triangulation using the randomized incremental algorithm, what is the worst-case number of edges that can be incident on the newly added site? What can you say about the expected-case number of such edges (assuming that points are inserted in random order)?

(d) For each of the following assertions about the Delaunay triangulation of a set P of n points in the plane, which are True and which are False?

(i) The Delaunay triangulation is a t-spanner, for some constant t

(ii) The Euclidean minimum spanning tree of P is a subgraph of the Delaunay triangulation

(iii) Among all triangulations of P, the Delaunay triangulation maximizes the minimum angle

(iv) Among all triangulations of P, the Delaunay triangulation minimizes the maximum angle

(v) Among all triangulations of P, the Delaunay triangulation minimizes the total sum of edge lengths

(e) 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 half credit.) Explain briefly.

(f) Given an n-element point set P in $\mathbb{R}^d$ and a scalar $s \geq 1$, it is known that an s-WSPD consists of $O(s^d n)$ pairs. Given any point $p \in P$, what is the *maximum* number of pairs $\{P_u, P_v\}$ of the decomposition such that either $p \in P_u$ or $p \in P_v$. (Express your answer as a function of s, d, and n. Explain briefly.)

Problem 2. You are given a set P of n points in $\mathbb{R}^2$. Present data structures for answering the following two queries. In each case, the data structure should use $O(n^2)$ space, it should answer queries in $O(\log n)$ time. (You do not need to explain how to build the data structure.)

(a) The input to the query is a nonvertical line ℓ. Such a line partitions P into two (possibly empty) subsets: $P^+(\ell)$ consists of the points lie on or above ℓ and $P^-(\ell)$ consists of the points of P that lie strictly below ℓ (see Fig. 1(a)). The answer is the maximum vertical distance h between two lines parallel to h that lie between $P^+(\ell)$ and $P^-(\ell)$ (see Fig. 1(b)).

For simplicity, you may assume that neither set is empty (implying that h is finite).

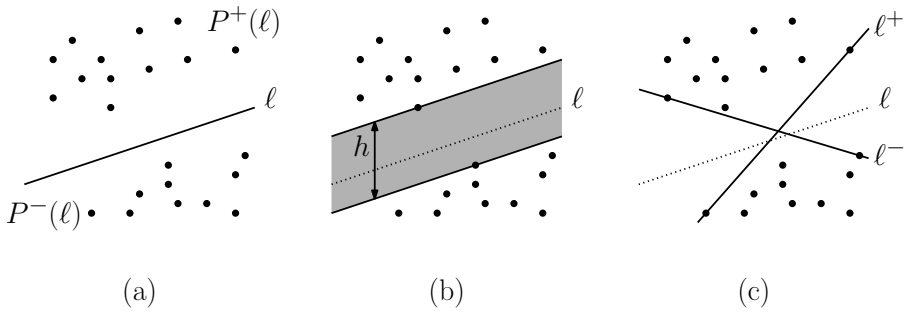


Figure 1: Separation queries.

(b) Again, the input to the query is a nonvertical line ℓ. The answer to the query consists of the two lines ℓ^- and ℓ^+ of minimum and maximum slope, respectively, that separate $P^+(\ell)$ from $P^-(\ell)$ (see Fig. 1(c)). You may assume that $P^+(\ell)$ from $P^-(\ell)$ are *not* separable by a vertical line (implying that these two slopes are finite).

Problem 3. 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. 2.)

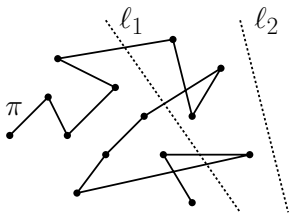


Figure 2: Path crossing queries.

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. 2 the answer for ℓ_1 is "yes," and the answer for ℓ_2 is "no.") (Hint: Duality is involved, but the solution requires a bit of lateral thinking.)

Problem 4. Consider the following two geometric graphs defined on a set P of points in the plane.

(a) *Box Graph:* Given two points $p, q \in P$, define box(p, q) to be the square centered at the midpoint of $\overline{pq}$ having two sides parallel to the segment $\overline{pq}$ (see Fig. 3(a)). The edge (p, q) is in the box graph if and only if box(p, q) contains no other point of P (see Fig. 3(b)). Show that the box graph is a subgraph of the Delaunay triangulation of P.

(b) *Diamond Graph:* Given two points $p, q \in P$, define diamond(p, q) to be the square having $\overline{pq}$ as a diagonal (see Fig. 3(c)). The edge (p, q) is in the diamond graph if and only if diamond(p, q) contains no other point of P (see Fig. 3(d)). Show that the diamond graph may not be a subgraph of the Delaunay triangulation of P. (Hint: Give an example that shows that the diamond graph is not even planar.)

Problem 5. 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. 4(a) are connected, but the disks of Fig. 4(b) are not.)

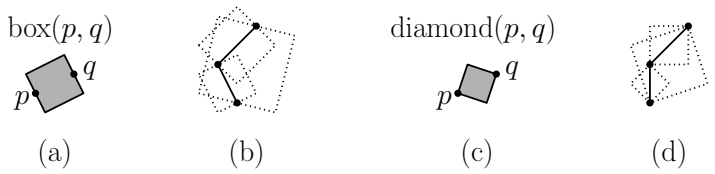


Figure 3: The box and diamond graphs.

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.

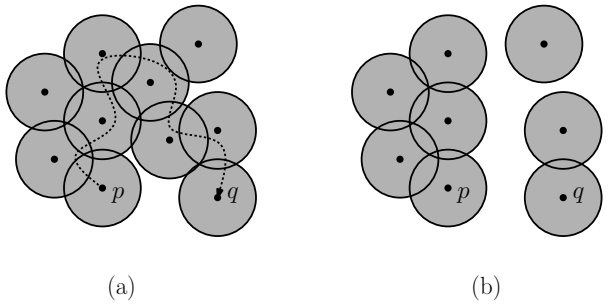


Figure 4: Safe connectivity.

Problem 6. 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 $\{p_1, \ldots, p_n\}$ 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. 5 below the string would be $p_2p_1p_2p_5p_7p_9p_{10}$.)

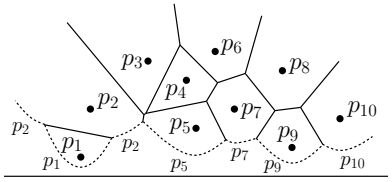


Figure 5: Beach-line complexity.

(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 ($\ldots p_ip_i \ldots$) 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.

Problem 7. Given a set of n points P in $\mathbb{R}^d$, and given any point $p \in P$, its *nearest neighbor* is the closest point to p among the remaining points of P. Note that nearest neighbors are not reflexive, in the sense that if p is the nearest neighbor of q, then q is not necessarily the nearest neighbor of p. Given an approximation factor $\varepsilon > 0$, we say that a point $p' \in P$ is an *ε-approximate nearest neighbor* to p if $\|pp'\| \leq (1+\varepsilon)\|pp''\|$, where p'' is the true nearest neighbor to p.

Show that in $O(n \log n + (1/\varepsilon)^d n)$ time it is possible to compute an ε-approximate nearest neighbor for every point of P. Justify the correctness of your algorithm. Hint: This can be solved using either WSPDs or spanners.

Note: There exists an algorithm that runs in $O(n \log n)$ time that solves this problem exactly, but it is considerably more complicated than the one I have in mind here.

CMSC 754: Final Exam

Throughout, 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 $O(T(n))$. Total point value is 100 points. **Good luck!**

Problem 1. (30 points) Give a short answer (a few sentences at most) to each question. Except where requested, explanations are not required.

(a) (3 points) You are given four points a, b, c, d in $\mathbb{R}^2$. Using just orientation tests, show how to test whether these points are the vertices of a convex quadrilateral listed in counterclockwise order.

(b) (3 points) In the plane-sweep algorithm for line-segment intersection, we were meticulous in deleting intersection events from the priority queue whenever the two segments generating this event were no longer consecutive on the sweep line. Why did we do this?

(c) (3 points) Let P be a simple polygon with n sides, where n is a large number. As a function of n, what is the maximum number of *scan reflex vertices* that it might have? Draw an example to illustrate.

(**Hint:** Recall the definition from Lecture 5. No proof is needed. It is okay if your answer is off by a constant additive term. For full credit, the multiplicative factor on n should be tight.)

(d) (3 points) We claimed that the worst-case depth of the point-location data structure based on trapezoidal maps is $\Omega(n)$. Draw an example of a set of line segments and an insertion order for these segments such that the depth of the point-location data structure is $\Omega(n)$. (It should be clear how to generalize your example to work with arbitrarily large values of n.)

(e) (6 points)

(i) Define the *zone* of an arrangement of lines in the plane.

(ii) State the *Zone Theorem*.

(iii) What was the importance of the zone theorem in our algorithm for building line arrangements in the plane?

(f) (12 points) Consider the simplicial complex shown in Fig. 1(a), consisting of four 0-simplices $(0, 1, 2, 3)$, five 1-simplices $(01, 03, 12, 13, 23)$, and two 2-simplices $(013, 123)$. Consider the 1-chains $c_1 = \{01, 12, 23\}$ and $c_2 = \{12, 13, 23\}$ (see Fig. 1(b)).

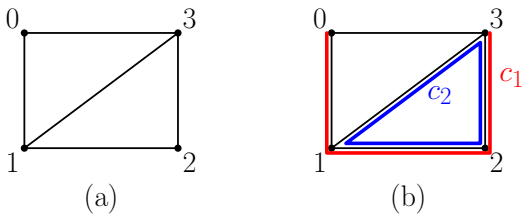


Figure 1: Chains in a simplicial complex. Recall the definitions from Lecture 18.

(i) Express c_1 and c_2 as 0-1 vectors over the basis $\langle 01, 03, 12, 13, 23\rangle$.

(ii) Express $c_1 + c_2$ as a 0-1 vector over this same basis.

(iii) What is the chain c_3 such that $c_1 + c_3 = c_2$? (Express it as a 0-1 vector over this same basis.)

(iv) What are the boundaries of c_1 and c_2? (**Hint:** Each is a 0-chain, which can be expressed as a 0-1 vector over the basis $\langle 0, 1, 2, 3 \rangle$.

Problem 2. (10 points) You are given three n-element point sets in $\mathbb{R}^2$, $R = \{r_1, \ldots, r_n\}$, called *red*, $G = \{g_1, \ldots, g_n\}$, called *green*, and $P = \{p_1, \ldots, p_n\}$, called *purple*. For each of the following two problems, present a reduction to linear programming in a space of constant dimension. Indicate which variables are used in the LP formulation, what the constraints are, and what the objective function is. Indicate what to do if the LP returns an answer that is infeasible or unbounded (if that is possible).

(a) (5 points) A (linear) *slab* is a region of the plane bounded by two parallel lines, $y = ax + b^+$ and $y = ax + b^-$. Given R, G, and P, compute the slab (if it exists) of minimum vertical height such that all the points of R lie strictly above the slab, all the points of G lie within the slab, and all the points of P lie strictly below the slab (see Fig. 2(a)). If no such slab exists, you should detect and report this.

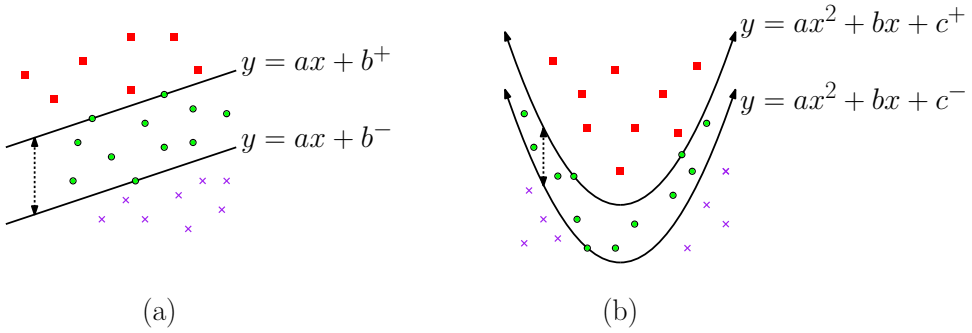


Figure 2: Linear and parabolic slabs.

(b) (5 points) A *parabolic slab* is the region of the plane bounded between two "parallel" parabolas, $y = ax^2 + bx + c^+$ and $y = ax^2 + bx + c^-$. Given R, G, and P, compute the parabolic slab of minimum vertical distance such that all the points of R lie strictly above the slab, all the points of G lie within the slab, and all the points of P lie strictly below the slab (see Fig. 2(b)). If no such parabolic slab exists, you should detect and report this.

Problem 3. (25 points) You are given two sets of points R and B in $\mathbb{R}^2$ (called red and blue, respectively). Let n denote the total number of points in both sets. Let us assume that the point sets are disjoint, that is, $R \cap B = \emptyset$. Given an arbitrary point $q \in \mathbb{R}^2$ (not in R or B), we *color* it (red or blue) depending on whether its closest point in $R \cup B$ is red or blue. (In Fig. 3(a), q is colored red, since its nearest neighbor is red). If q is equidistant to its closest points in R and B, it may be assigned either color.

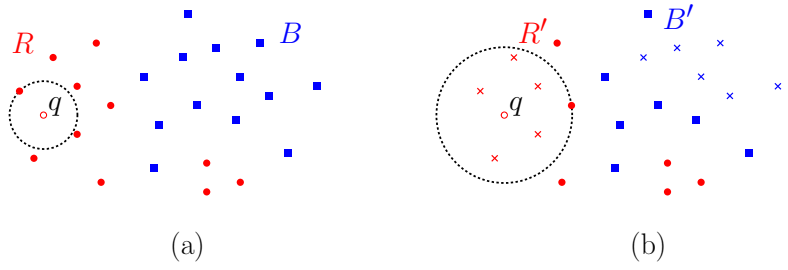


Figure 3: Coloring queries via nearest neighbors and compatible subsets.

(a) (5 points) Present a data structure with $O(n)$ storage which is given R and B as inputs, and answers the following *color queries*: given any $q \in \mathbb{R}^2$, return q's color. It should be possible to build your data structure in $O(n \log n)$ time, and it should answer queries in $O(\log n)$ time. (**Hint:** Don't do this from scratch. Combine structures we have seen in this semester.)

(b) (5 points) In some applications, n may be extremely large, and so it is desirable to reduce the sizes of R and B as much as possible. We say that two subsets $R' \subseteq R$ and $B' \subseteq B$ are *compatible* with R and B if every point $q \in \mathbb{R}^2$ is assigned the same color with respect to R' and B' that it would have been assigned with respect to the original sets R and B. (We have removed all the points marked with "$\times$" in Fig. 3(b). Observe that point q is still correctly colored by its nearest neighbor in the reduced point set.)

Present an efficient algorithm that, given R and B as input, produces the compatible subsets $R' \subseteq R$ and $B' \subseteq B$ having the smallest possible number of points. Just give the algorithm. Correctness and running time will be covered below. (**Hint:** Voronoi diagrams and/or Delaunay triangulations may be helpful.)

(c) (5 points) Prove that your algorithm produces a compatible subset by showing that for every query point $q \in \mathbb{R}^2$, the color of its nearest neighbor in $R' \cup B'$ is the same as its nearest neighbor in $R \cup B$.

(d) (5 points) Assuming that the points of $R \cup B$ are in general position, prove that your algorithm produces a minimum-sized subset by showing that for any smaller subset, there exists a point $q \in \mathbb{R}^2$ that will be incorrectly colored based on its nearest neighbor. (The general-position assumption is critical!)

(e) (5 points) Derive the running time of your algorithm. (**Hint:** $O(n \log n)$ time is possible. Partial credit will be given for a slower algorithm.)

Problem 4. (20 points) Consider an n-element point set $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^2$, and an arbitrary point $q \in \mathbb{R}^2$ (which is not in P). We say that q is *k-deep* within P if any line ℓ passing through q has at least k points of P on or above the line and at least k points of P on or below it.

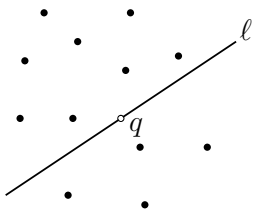


Figure 4: Point q is 4-deep within P.

For example, the point q in Fig. 4 is 4-deep, because any line passing through q has at least four points of P on either side of it (including lying on the line itself).

(a) (5 points) Assuming we use the usual dual transformation, which maps point $p = (a, b)$ to line $p^* : y = ax - b$, explain what it means for a point q to be k-deep within P (in terms of the dual line q^* and the dual arrangement $\mathcal{A}(P^*)$).

(b) (5 points) Present an efficient algorithm which, given P and q, determines the maximum value k such that q is k-deep within P. (**Hint:** $O(n \log n)$ time is possible. I will accept a slower algorithm for partial credit.)

(c) (10 points) Present an efficient algorithm which, given P and an integer k, determines whether there exists a point q that is k-deep within P. (**Hint:** First consider what this means in the dual setting. $O(n^2 \log n)$ time is possible. I will accept a slower algorithm for partial credit.)

For parts (b) and (c) briefly justify your algorithm's correctness and derive its running time.

Problem 5. (15 points) A set P of n points in $\mathbb{R}^d$ determines a set of $\binom{n}{2}$ different distances. Define $\Delta(P)$ to be this set of distances $\{\|p_i - p_j\| : 1 \leq i < j \leq n\}$. Given an integer k, where $1 \leq k \leq \binom{n}{2}$, we are interested in computing the kth smallest distance from this set. Normally, this would take $O(n^2)$ time, so let's consider a fast approximation algorithm.

Let $\delta(P, k)$ denote the exact kth smallest distance in $\Delta(P)$. Given $\varepsilon > 0$, a distance value x is an ε-approximation to $\delta(P, k)$ if

$$\frac{\delta(P,k)}{1+\varepsilon} \leq x \leq (1+\varepsilon)\delta(P,k).$$

Present an efficient algorithm to compute such a value x. Justify your algorithm's correctness and derive its running time. (**Hint:** Use well-separated pair decompositions. You may assume that, when the quadtree is computed, each node u of the quadtree is associated with an integer $\text{wt}(u)$, which indicates the number of points of P lying within u's subtree.)

You may assume that d and ε are constants (independent of n). I know of an algorithm that runs in time $O(n \log n + n/\varepsilon^d)$ time, but I will accept for full credit an algorithm that runs in time $O((n \log n)/\varepsilon^d)$.)