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: Fall 2023 Dave Mount

Homework 1: Convex Polygons and Plane Sweep

Handed out Tuesday, Sep 12. Due by the start of class on Thursday, Sep 21. (Submissions will be through Gradescope. Submission information will be forthcoming.) 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*. Listed point values are subject to change.

Problem 1. (10 points) You are given a set of four points in the plane $\{p_1, \ldots, p_4\}$, where $p_i = (x_i, y_i)$.

(a) (5 points) Present an efficient boolean function `quadSeq`(p_1, p_2, p_3, p_4) (in pseudocode) that determines whether this sequence of points define the vertices of a convex quadrilateral in cyclic order (either clockwise or counterclockwise).

(b) (5 points) Present an efficient boolean function `quadSet`(p_1, p_2, p_3, p_4) (in pseudocode) that determines whether this set of points define the vertices of a convex quadrilateral in any order.

In this problem you may *not* assume that points are in general position. If any three points are collinear, then both of the above functions should return `false` (see Fig. 1 for some examples.)

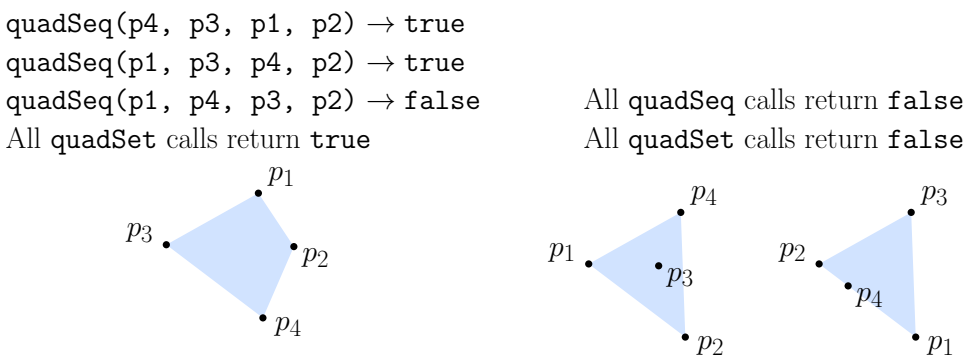


Figure 1: Examples of return results for `quadSeq` and `quadSet`.

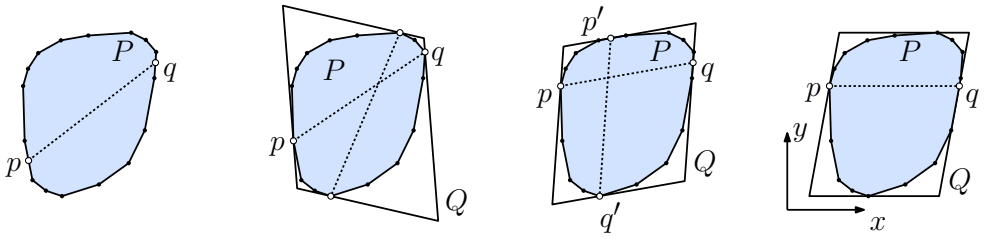
As with any procedure, you may create variables to store intermediate values or define additional subroutines. Explain how your procedure works in English and justify its correctness. For full credit, your procedure should be based solely on orientation tests involving these four points. (E.g., a comparison of the form "`if (orient(p1, p2, p4) < 0)`" is valid, but "`if (x1 < x2)`" is not.) You should not invoke any convex hull algorithms (like Graham's algorithm). Try to do it with the smallest possible number of orientation tests. In each case, indicate how many orientation tests your procedure makes in the worst case.

Hint: Your answer to (b) is allowed to invoke your answer to (a). For `quadSet`, I think that a good way to start would be to fix two points, say p_1 and p_2, and check whether the other

two points share the same orientation with respect to them, that is, "`if (orient(p1, p2, p3) == orient(p1, p2, p4))`".)

Problem 2. (15 points) Define a *chord* of a convex polygon to be any line segment $\overline{pq}$ where p and q are points on P's boundary (see Fig. 2(a)). Given a convex polygon P, we say that a parallelogram Q *circumscribes* P if it tightly encloses P. That is, P is contained within Q, and each of Q's sides touches P (see Fig. 2(b)). If Q circumscribes P, then for each pair of parallel sides of Q, there will be at least one chord common to both P and Q that is incident to the these two sides (as with $\overline{pq}$ in Fig. 2(b)). This is called a *common chord*.

A circumscribing parallelogram is *perfect* if for each pair of parallel sides of Q, there is a common chord incident to these two sides that is parallel to Q's other sides (see Fig. 2(c)). A circumscribing parallelograpm is *half-perfect* if there exists one such chord (see Fig. 2(d)).



(a) (b) (c) (d)

Figure 2: (a) A chord, (b) circumscribing parallelogram and common chords, (c) a perfect parallelogram (where chords $\overline{pq}$ and $\overline{p'q'}$ are parallel to Q's sides), and (d) a half-perfect parallelogram for P.

(a) (5 points) Prove that for any convex polygon P, there exists a half-perfect parallelogram where two of its sides are horizontal (that is, parallel to the x-axis) and the common chord is also horizontal (see Fig. 2(c)).

(**Hint:** Two approaches come to mind. One is based on varying the slopes of the non-horizontal sides of the parallelogram and showing that some slope yields a horizontal common chord. The other is based on varying the y-coordinate of the horizontal chord of P and showing that for some y-value, there a parallelogram that shares this chord.)

(b) (2 points) Prove that if P is in general position, the half-perfect parallelogram is unique. We will accept either of two answers. First, you can prove that multiple common chords (even if there is a unique half-perfect parallelogram) implies a violation of general position, or second, you can prove that multiple half-perfect parallelograms (even if there is a unique common chord) implies a violation of general position.

(c) (8 points) Present an $O(n)$ time algorithm which given a convex polygon P in general position, computes the half-perfect parallelogram described in parts (a) and (b). (If you can see how to do this faster, check out the Challenge Problem)

Problem 3. (10 points) The objective of this problem is to explore a natural adaptation of the notion of convex hulls to rectilinear objects. An *axis-parallel rectangle* is formally defined as

the Cartesian product of two nonempty, closed intervals, one along the x-axis and one along the y-axis, that is $r = [a, b] \times [c, d]$, where $a < b$ and $c < d$ (see Fig. 3(a)).

You are given a collection of n axis-parallel rectangles in the plane $R = \{r_1, \ldots, r_n\}$, where $r_i = [a_i, b_i] \times [c_i, d_i]$. Let $\mathcal{R}$ denote the union of these rectangles, that is, $\mathcal{R} = \bigcup_{i=1}^{n} r_i$. It will be convenient to assume that $\mathcal{R}$ is *path connected*, which means that any two points $p, q \in \mathcal{R}$ are connected by a path that lies entirely within $\mathcal{R}$ (see Fig. 3(b)).

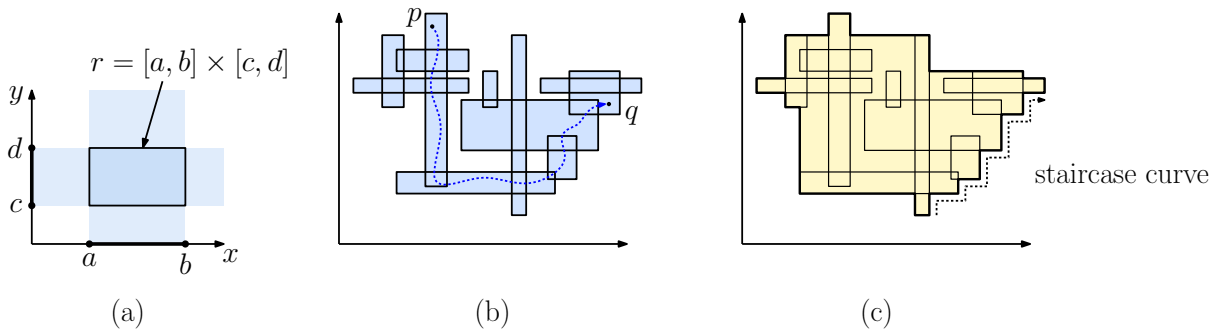


Figure 3: (a) An axis-parallel rectangle, (b) a path-connected set of rectangles, and (c) the orthogonal hull and one of its southeast staircase curve.

We say that a set H is *orthogonally convex* if the intersection of any horizontal or vertical line with H is either empty or a single line segment. The *orthogonal hull* of a set is defined to be the smallest orthogonally convex set that contains the set (see Fig. 3(c)).

(a) (8 points) Present an efficient algorithm which, given a set $R = \{r_1, \ldots, r_n\}$ of axis-parallel rectangles, computes the orthogonal hull of $\mathcal{R}$. The output of your algorithm consists of a counterclockwise sequence of edges that form the boundary of the hull. You may assume that the rectangles are in general position (and in particular, no two rectangles share the x- or y-coordinates) and their union is path connected. Explain your algorithm and derive its running time.
(**Hints:** It is a fact (which you do not need to prove) that the boundary of the orthogonal hull of a path-connected set of orthogonal rectangles can be divided into four "staircase curves", connecting the leftmost and rightmost vertical edges with the top and bottom horizontal edges (see Fig. 3(c)). Present an algorithm to build any one of these staircase curves, and argue that the others follow from symmetry. Finally, combine the four staircases to form the final output. Try plane sweep. You should aim for an overall running time of $O(n \log n)$.)

(b) (2 points) In the problem description, we made the convenience assumption that $\mathcal{R}$ is path connected. Show (by giving an example and an explanation) that if this were not true, the orthogonal hull might consist of multiple connected components.

Problem 4. (15 points) We are given a set of axis-parallel rectangles $R = \{r_1, \ldots, r_n\}$ in the plane, where $r_i = [a_i, b_i] \times [c_i, d_i]$. All the rectangles lie within a horizontal strip $\mathcal{S} = \{(x, y) \mid 0 \leq y \leq 1\}$ (see Fig. 4(a)).

In this application we imagine that water is flowing along the "stream" $\mathcal{S}$ from left to right, and it must flow around the rectangular "rocks" of R. The question is whether any water

can make it through, or do the rocks effectively block all the flow. Present an algorithm which, given R, returns `true` or `false`, depending on whether there exists a x-monotone path through S that avoids all the rectangles of R. (Such a path does exist in the example of Fig. 4(b) but not in the case of Fig. 4(c)).

Present an efficient algorithm that solves the problem. You may make the general position assumption that, except for $y = 0$ and $y = 1$, no two rectangles have the same x- or y-coordinates. Explain your algorithm and derive its running time. (**Hint:** Use plane sweep. Aim for a running time of $O((n + m) \log n)$, where m is the total number of edge-to-edge intersections among the sides of the rectangles. I believe that it is possible to improve this to $O(m + n \log n)$, but you make need to make some additional assumptions about your ordered dictionary.)

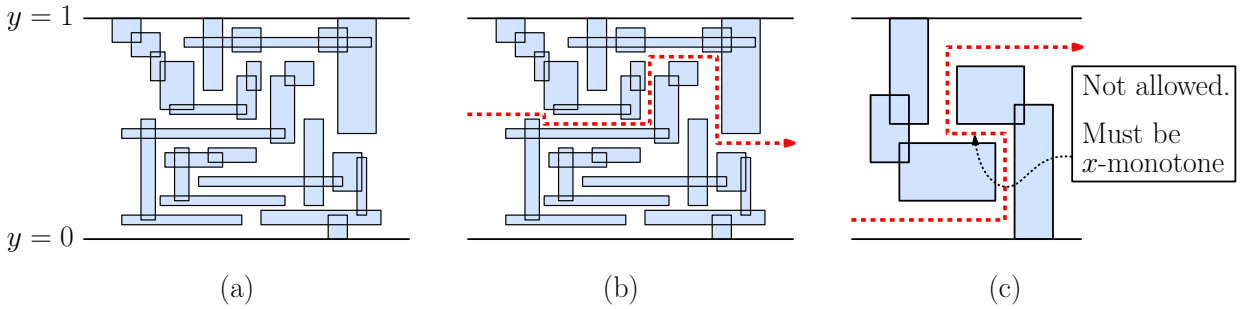


Figure 4: Does there exist an x-monotone path that avoids all the rectangles?.

Challenge Problem 1: Show that it is possible to solve Problem 2(c) in sublinear time. Explain your algorithm and derive its running time.

Challenge Problem 2: Given a convex polygon P, let Q be a parallelogram of minimum area that encloses P. Prove that Q must be perfect. (**Hint:** Show how to perturb any non-perfect parallelogram to form a new parallelogram enclosing P but with smaller area.)

Challenge Problem 3: Can you solve Problem 4 in $O(n \log n)$ time, irrespective of m? (I honestly do not know, so this may be quite challenging!)

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. (If the source is from outside of class, you must cite your sources.) 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: Fall 2023 Dave Mount

Homework 2: LP, Point Location, and Voronoi Diagrams

Handed out Tuesday, Oct 3. Due: **9:30am, Tuesday, Oct 17** (submission through Gradescope). No late homeworks will be accepted, 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. (20 points) Explain how to solve each of the following problems in linear (expected) time. Each can be modeled by reduction to linear programming (LP), perhaps involving multiple instances along with some additional pre- and/or post-processing. See the remarks at the end of the homework on how to present LP reductions.

Note that both problems involve obstacle avoidance. We allow the trajectory to pass through the boundary points of the obstacles. (This is necessary for LP, since it assumes that the constraints are closed halfspaces.)

(a) (10 points) In your new career as a professional miniature golf player, you are working on a program to compute your best initial shot on a tricky hole. Let's model this as a rectangle of height w and length ℓ whose lower left coordinate is the origin (see Fig. 1(a)). There are a series of line segment obstacles that need to be avoided, each growing perpendicularly out from one of the rectangle's sides. The input consists of points $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$ (both unsorted), where $a_i = (a_{i,x}, a_{i,y})$, and $b_j = (b_{j,x}, b_{j,y})$. The points of A are the endpoints of the segments ascending up from the rectangle's bottom edge and the points of B are the endpoints of the segment descending from the top edge. The ball must travel from the left edge to the right edge of the rectangle without intersecting any of the obstacles (see Fig. 1(b)).

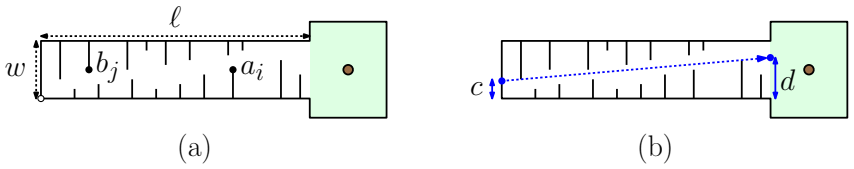


Figure 1: Miniature golf.

The final trajectory of the shot is given by two scalars, c and d, which denote the y-coordinates along the rectangle's left and right sides, respectively, through which the ball passes (see Fig. 1(a) lower). You do not expect to hit the hole on the first shot. Instead, you objective is to get the ball as close to the center as possible. In particular, you want the value of d where the ball exits the rectangle to be as close to $w/2$ as possible.

Present an algorithm which determines whether a straight-line path exists from the left to right edge of the rectangle that avoids all the obstacles. If such a path exists, determine the trajectory that places d as close as possible to $w/2$. Your algorithm should run in expected time $O(m + n)$.

(b) (10 points) After your golf career failed, you turn to (indoor) tennis. You want to determine how best to hit a lob shot, which travels high enough to avoid the players on the other side of the net but low enough to avoid hitting lighting fixtures hanging from the ceiling. Let's just consider the problem in two-dimensional space, where the floor is the x-axis, and the y-axis is vertical (see Fig. 2(a)). The lob needs to clear the net, which is of height h along the y-axis. The lob needs to pass above positions likely occupied by the opposing player, which are given as a set of points $P = \{p_1, \ldots, p_m\}$, and it must pass below a set of locations of light fixtures $Q = \{q_1, \ldots, q_n\}$, where $p_i = (p_{i,x}, p_{i,y})$, and $q_j = (q_{j,x}, q_{j,y})$. (Both sets are unsorted.) Finally, it needs to land within the court, which is of length ℓ.

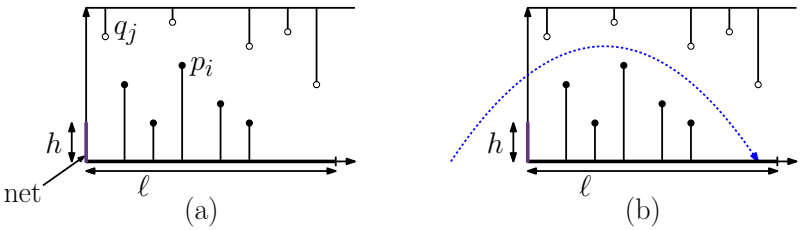


Figure 2: Tennis.

By standard physics, the shot will travel along a parabolic path, given by the equation $y = -gx^2 + bx + c$ of a parabola, where g is a fixed positive constant determined by the force of gravity (e.g., 32 ft/sec^2), and reals b and c are based on the initial location, direction, and speed with which the ball is hit. Present an algorithm which determines whether it is possible to find real values b and c to satisfy all the following requirements (see Fig. 2(b)). The ball's trajectory must:

(a) pass above the net of height h along the y-axis

(b) pass above all the opposing player points in P

(c) pass below all the lighting fixture points in Q

(d) land on the ground within the court (within distance ℓ of the net)

If such a shot is exists, return the one that travels as high as possible above the net along the y-axis. Your algorithm should run in expected time $O(m + n)$.

Problem 2. (10 points) Consider the segments shown in Fig. 3.

(a) (2 points) Show the (final) *trapezoidal map* for these segments, assuming they are inserted in the order $\langle s_1, s_2, s_3 \rangle$. (We have given you the map after inserting the first two segments, so you only need to show the result after inserting s_3.)

(b) (8 points) Show the *point-location data structure* resulting from the construction given in class, assuming the insertion order from part (a). (We have given you the point-location data structure after inserting s_1 and s_2, so you only need to show the result after inserting s_3.) We will give partial credit if your data structure works correctly, even though it does not match the construction given in class.

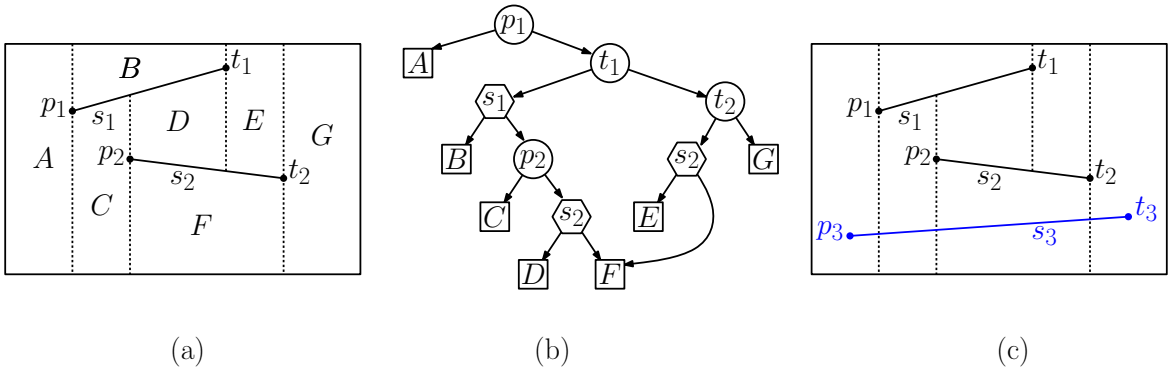


Figure 3: Trapezoidal map and point location.

Please follow the convention given in class for the node structure. (In particular, for y-nodes, the left (resp., right) child corresponds to the region above (resp., below) the segment.)

Problem 3. (10 points) After your tennis career failed, you have decided to return to miniature golf. Given the same hole structure as in Problem 1(a), you want to know which obstacle your ball will hit first for a given shot.

The setup is the same as before. The input consists of a $w \times \ell$ rectangle with origin at the lower-left corner and point sets A and B defining the lower and upper obstacle endpoints, of sizes m and n, respectively, where $a_i = (a_{i,x}, a_{i,y})$, and $b_j = (b_{j,x}, b_{j,y})$ (see Fig. 4(a)).

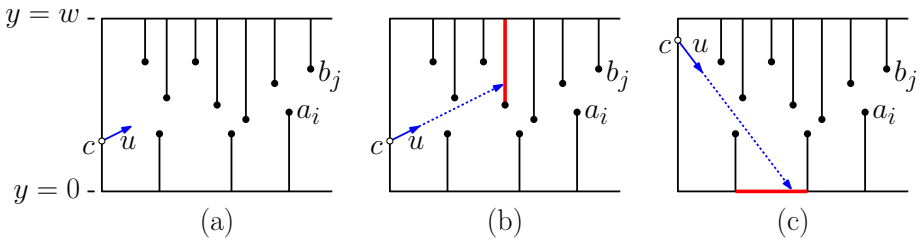


Figure 4: Golf shooting queries.

Your golf shot is determined by two quantities (see Fig. 4(a))

- the y-coordinate c along the left edge of the rectangle where your shot starts, and
- a directional vector $u = (u_x, u_y)$ of a ray along which the shot travels.

Your objective is to preprocess A and B into a data structure so that given the triple (c, u_x, u_y), you can efficiently determine which obstacle is hit first (or if no obstacle is hit, does the ball hit the upper, lower, or right side of the rectangle). You may assume that $0 \leq c \leq w$ and $u_x > 0$. Your data structure should use $O(n+m)$ space and answer queries in time $O(\log(n+m))$. It suffices to just explain how the reduction is performed and justify its correctness, query time, and space requirements. (You do not need to explain how to compute the segments nor how to construct the point-location data structure.)

Hint: Begin by extracting the equation for the line ℓ carrying the query ray. Then show that by applying duality, this problem can be reduced to a point-location in $\mathbb{R}^2$ where the segments are determined by the obstacle vertices A and B. Although I believe you can do this with one data structure, it may be simpler to describe two data structures, one for A and the other for B, and then combine the results.

Problem 4. (10 points) It is sometimes of interest to compute the Voronoi diagram of a set of sites, but we are only interested in a portion of the final diagram. In this problem, we'll consider how to compute the Voronoi diagram of a set of points in $\mathbb{R}^2$, but restricted to a given line ℓ. By rotating and translating space, we may assume that ℓ is aligned with the x-axis.

You are given a sequence of n sites in the plane $P = \langle p_1, \ldots, p_n \rangle$ sorted in increasing order of their x-coordinates (see Fig. 5(a)). Present an algorithm that computes the Voronoi diagram of P, but restricted only to x-axis. (We don't care about the portion of the diagram lying above or below the axis.)

Observe that the diagram is a sequence of intervals that subdivide the x-axis. The output consists of a sequence of (at most $n-1$) endpoints of the segments $\langle x_1, \ldots, x_m \rangle$, and each edge is labeled with the index of the associated site corresponding to this interval (see Fig. 5(b)). Your algorithm should run in $O(n)$ time. (**Hint:** Start by proving that the left-to-right order of the labels along the x-axis is consistent with the left-to-right order of the sites.)

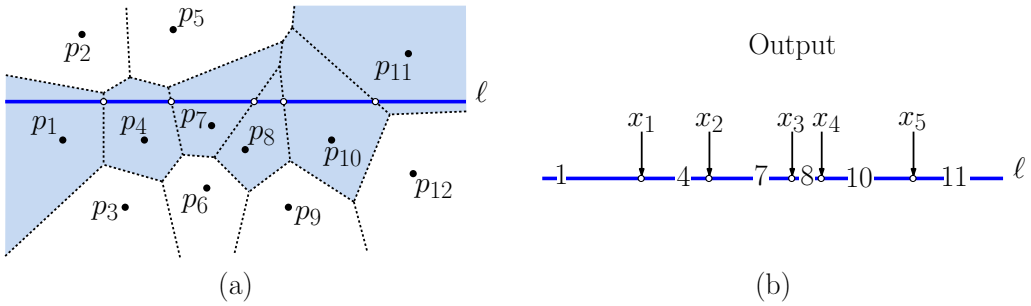


Figure 5: Restriction of a Voronoi diagram to a line.

Challenge Problem. You are given a collection of n nonintersecting circular disks in the plane, each of radius r_1 called *obstacles*. Let $P = \{p_1, \ldots, p_n\}$ denote their center points. You are also given a radius value r_2 and two points s and t (see Fig. 6(a)).

Present an efficient algorithm which, given the obstacles along with s, t, and r_2, determines whether it is possible to move the disk of radius r_2 from s to t without intersecting any of the obstacle disks (see Fig. 6(b)). Ideally, your algorithm should run in $O(n \log n)$ time. If there does not exist such a path (including the case where the initial or final positions are invalid), then indicate this. Otherwise, your algorithm should output any such path, say, as a polygonal chain from s to t. (**Hint:** Use Voronoi diagrams.)

Guidance for Writing LP Reductions: In an linear programming (LP) reduction, you should explain the following:

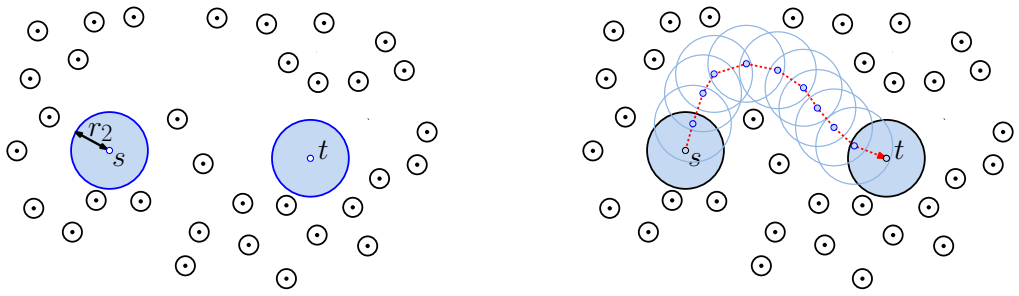


Figure 6: Can you move a disk of radius r_2 from s to t while avoiding the obstacles?

- how the solution space is modeled as a vector (and in what dimension),
- what are the constraints,
- what is the objective function (and express it as a vector)
- how to interpret the result (including the cases feasible, unbounded, infeasible)

Here is an example.

Sample Problem: Present an efficient algorithm which given two sets of points $R = \{r_1, \ldots, r_n\}$ and $B = \{b_1, \ldots, b_n\}$, both in $\mathbb{R}^3$, determines whether their exists a plane h in $\mathbb{R}^3$ such that all the points of R lie on or above h and all the points of B lie on or below h.

Sample solution: We reduce the problem to linear programming in $\mathbb{R}^3$. Let's assume that each $r_i \in R$ is given in coordinate form as $(r_{i,x}, r_{i,y}, r_{i,z})$ and similarly for B. Let's model h by the equation $z = ax + dy + e$, for some real parameters a, d, and e. To enforce the condition that each r_i lies on or above h and each b_j lies on or below it, we add the constraints

$$\begin{aligned} r_{i,z} &\geq ar_{i,x} + dr_{i,y} + e, \qquad \text{for } 1 \leq i \leq n \\ b_{j,z} &\leq ab_{j,x} + db_{j,y} + e, \qquad \text{for } 1 \leq j \leq n. \end{aligned}$$

We then invoke LP with $2n$ constraints in $\mathbb{R}^3$ (with the variables (a, d, e)). Since this is a yes-no answer, we don't really care about the objective function. We can set it arbitrarily, for example, "maximize e" (which is equivalent to using the objective vector $c = (0, 0, 1)$).

If we wish to be even more formal (which is not usually required), we can express the LP in standard form as maximizing c^{T} form as $Ax \leq b$, where x is the symbolic vector $(a, d, e) \in \mathbb{R}^3$, and A and b can be expressed as

$$\begin{bmatrix} r_{1,x} & r_{1,y} & 1 \\ \vdots & \vdots & \vdots \\ r_{m,x} & r_{m,y} & 1 \\ -b_{1,x} & -b_{1,y} & -1 \\ \vdots & \vdots & \vdots \\ -b_{n,x} & -b_{1,y} & -1 \end{bmatrix} \begin{bmatrix} a \\ d \\ e \end{bmatrix} \leq \begin{bmatrix} r_{1,z} \\ \vdots \\ r_{m,z} \\ -b_{1,z} \\ \vdots \\ -b_{n,z} \end{bmatrix}$$

We interpret the LP's result as follows. If the result is "infeasible", then we know that no such plane exists. If the answer is "feasible" or "unbounded", then we assert that such a plane exists (assuming general position). This is clearly true if the result is "feasible", since we can just take h to be the plane associated with the optimum vertex (a, d, e). If the result is "unbounded", then the plane is vertical, but there exists a perturbation such that R lies above and B lies below.

CMSC 754: Fall 2023 Dave Mount

Sample Problems for the Midterm Exam

The midterm exam will be this **Thursday, Oct 19 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.

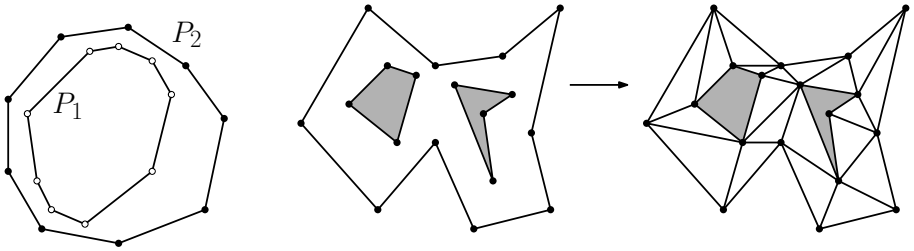
Problem 0. Expect a problem asking you to work through all or part of an algorithm that was presented in class on a specific example.

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) 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.

(c) 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: (a) Duals of nested polygons and (b) triangulating a polygon with holes.

(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(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.

(e) You are given a set of n disjoint line segments in the plane that have m intersection points. (For example, in Fig. 2, $n = 4$ and $m = 3$). 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 m, what is the maximum number of trapezoids in the final trapezoidal map? Explain briefly. (Give an exact, not asymptotic, answer.)

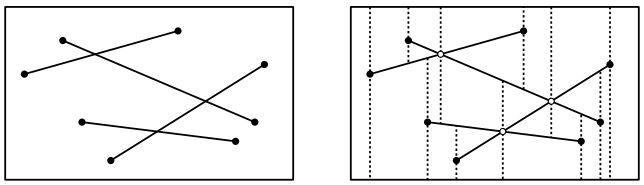


Figure 2: Trapezoidal map with intersections.

(f) 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).

(g) 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?

(h) 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.)

(i) In Fortune's algorithm, suppose that three points p_i, p_j, and p_k are involved in a vertex event (also known as a circle event). Let $c = (c_x, c_y)$ denote the center of their circumcircle, and let r denote its radius. What is the y-value where the vertex event is processed?

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

(i) The Euclidean minimum spanning tree of P is a subgraph of the DT.

(ii) Among all triangulations of P, the DT maximizes the minimum angle.

(iii) Among all triangulations of P, the DT minimizes the maximum angle.

(iv) Among all triangulations of P, the DT minimizes the total sum of edge lengths.

(v) The DT is a t-spanner, for some constant t.

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. 3(a)). What is the maximum number of intersections that might occur between the edges of these two polygons?

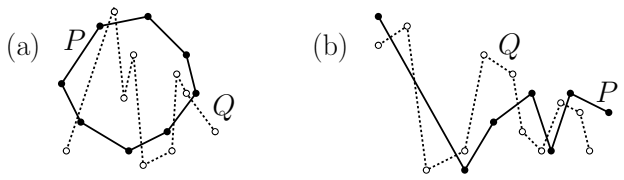


Figure 3: Maximum number of intersections.

(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)).

(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. 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. 4). 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.

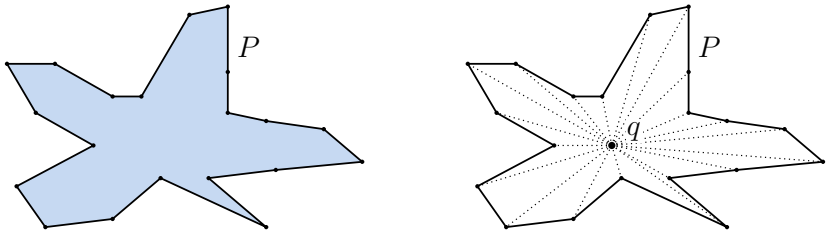


Figure 4: Determining whether a polygon is star-shaped.

Problem 4. This problem is inspired from applications in surveillance. Given a simple polygon P, we say that two points p and q are *visible* to each other if the open line segment between them lies entirely within P's interior. We allow for p and q to lie on P's boundary, but the segment between them cannot pass through any vertex of P (see Fig. 5(a)).

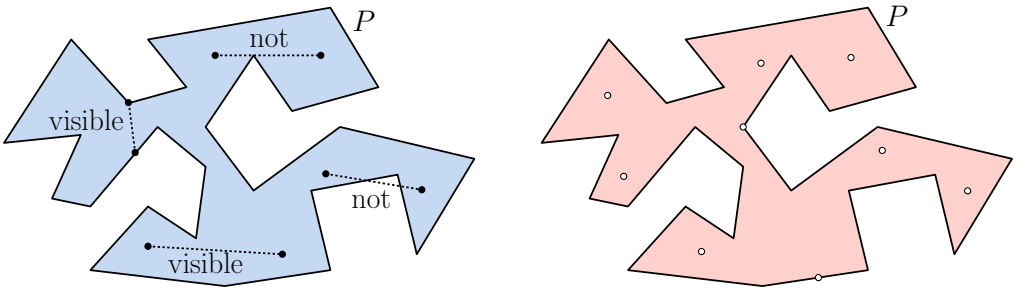


Figure 5: (a) Visibility and (b) a guarding set of size 9 for P.

A *guarding set* for P is any set of points G, called *guards*, lying in P (either on its boundary or in its interior) such that every point in P's interior is visible to at least one guard of G. Note that guards may be placed on vertices, along edges, or in P's interior (see Fig. 5(b)).

Prove that there exists a constant $c \geq 1$ such that (for all sufficiently large n) every n vertex simple polygon P has a guarding set of size at most n/c. For full credit, show that $c = 3$ works.

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. 6). 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. 6 the maximum depth is 3, as realized by the small triangular face in the middle.)

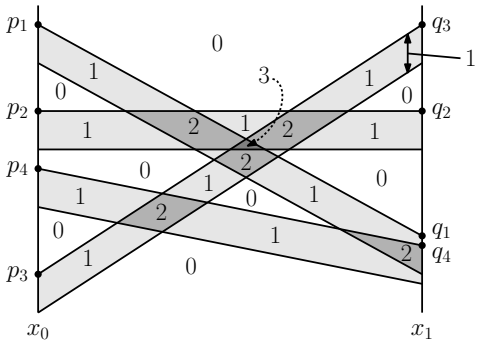


Figure 6: Maximum depth in a set of slabs.

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. 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^-. A *transversal* is defined to be a line $\ell : y = ax + b$ that intersects all of these segments (see Fig. 7).

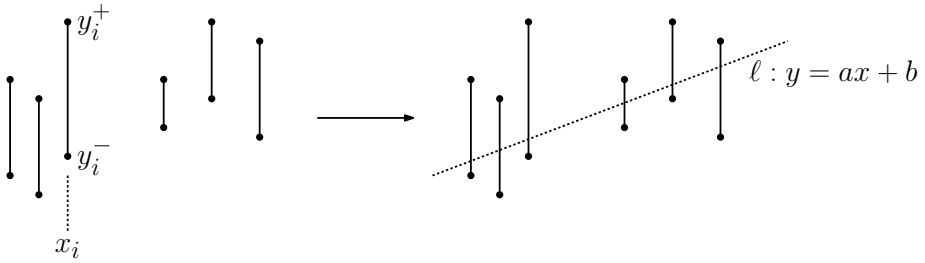


Figure 7: Existence of a transversal.

Present an efficient an algorithm, which given S determines whether there exists a transversal. (Hint: $O(n)$ time is possible.) Justify your algorithm's correctness and derive its running time.

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.)

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. 8(a)).

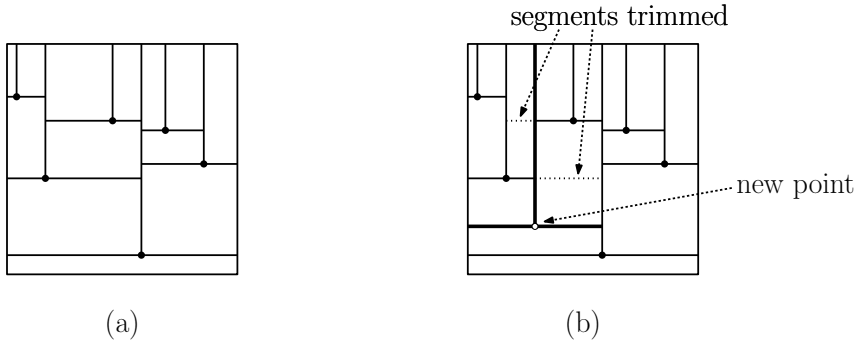


Figure 8: Building a subdivision.

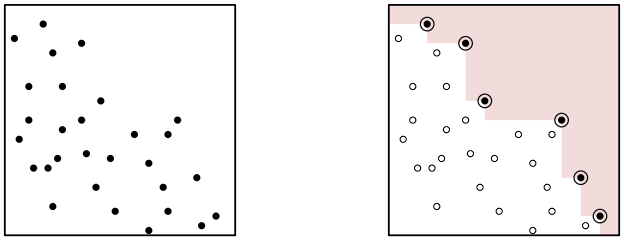
(a) Show that the resulting subdivision has size $O(n)$ (including vertices, edges, and faces).

(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. 8(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. 9(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) (b)

Figure 9: Paresto maxima.

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

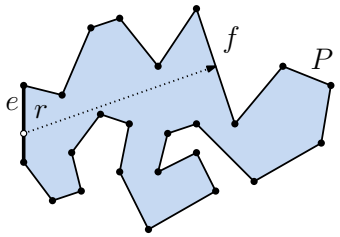


Figure 10: Ray-shooting queries.

Problem 11. 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. 11(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. 11(b)).

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

(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

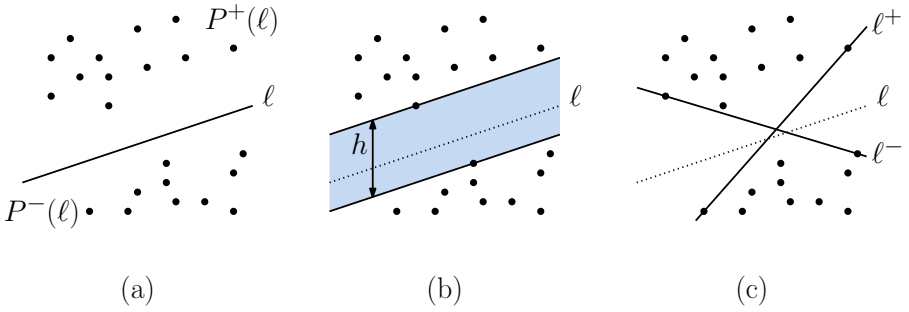


Figure 11: Separation queries.

$P^+(\ell)$ from $P^-(\ell)$ (see Fig. 11(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 12. 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. 12.)

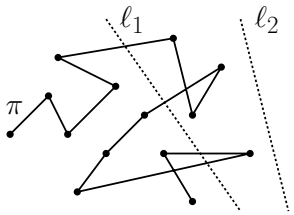


Figure 12: Path crossing queries.

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. 12 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 13. 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 $\text{box}(p, q)$ to be the square centered at the midpoint of $\overline{pq}$ having two sides parallel to the segment $\overline{pq}$ (see Fig. 13(a)). The edge (p, q) is in the box graph if and only if $\text{box}(p, q)$ contains no other point of P (see Fig. 13(b)). Show that the box graph is a subgraph of the Delaunay triangulation of P.

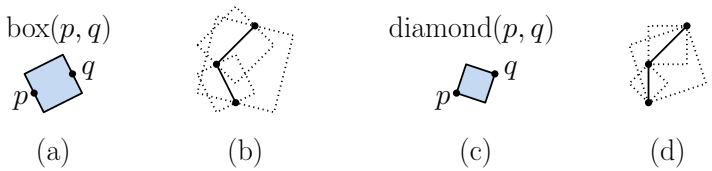


Figure 13: The box and diamond graphs.

(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. 13(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. 13(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 14. 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. 14(a) are connected, but the disks of Fig. 14(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.

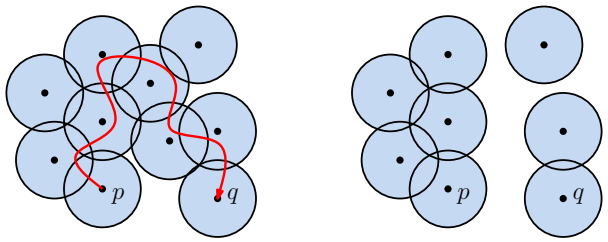


Figure 14: Safe connectivity.

Problem 15. 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. 15 below the string would be "$p_2p_1p_2p_5p_7p_9p_{10}$".)

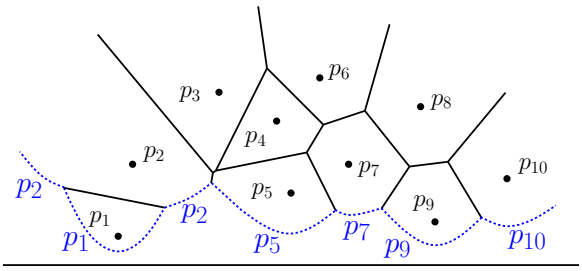


Figure 15: Beach-line complexity.

(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 ($\ldots p_i p_i \ldots$) 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

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 $O(T(n))$. If you are asked to give an algorithm, also explain how it works and derive its running time. (Unless otherwise stated, formal proofs of correctness are not required.)

Problem 1. (20 points) Short-answer questions.

(a) (4 points) Given a planar point set with n points and h vertices on the convex hull. What must be true about the relationship between h and n for Jarvis's algorithm to run at least as fast as Graham's algorithm? (State your answer asymptotically.)

(b) (4 points) Given a simple polygon with $n \geq 3$ vertices, what is the maximum number of *reflex vertices* it might have. (Remember that a reflex vertex is one whose interior angle is greater than 180°.) For full credit, give an exact answer as a function of n.

(c) (4 points) Consider Fortune's algorithm to compute the Voronoi diagram of n sites in the plane. What is the maximum number of arcs that any one site can contribute to the beach line?

(d) (8 points) Which of the following assertions about the Delaunay triangulation (DT) of a set P of n points in the plane are true?

(1) The closest pair of points are connected by an edge in the DT.
(2) The farthest pair of points cannot be connected by an edge in the DT.
(3) Among all triangulations of P, the DT maximizes the minimum angle.
(4) Among all triangulations of P, the DT minimizes the maximum angle.
(5) Among all triangulations of P, the DT minimizes the total sum of edge lengths.

Problem 2. (15 points) Recall the plane-sweep algorithm for decomposing a simple polygon into monotonic pieces.

(a) (8 points) Give the values of helper(e_i), for $i = 1, \ldots, 4$ in Fig 1(a).

(b) (7 points) Show all the new diagonals that would be added as a result of the event at vertex v_i, for $i = 1, \ldots, 4$ in Fig 1(b).

Problem 3. (15 points) We are given a collection of n (non-vertical) line segments $S = \{s_1, \ldots, s_n\}$ in the plane *in no particular order*. Each segment is given by its endpoints $s_i = \overline{p_i q_i}$, where p_i has the smaller x-coordinate. We want to know whether altogether, these segments form the edges of a single simple polygon (see Fig. 2(a)). That is, we need to check the following things:

- No two segments intersect, except at their endpoints (see Fig. 2(b)).

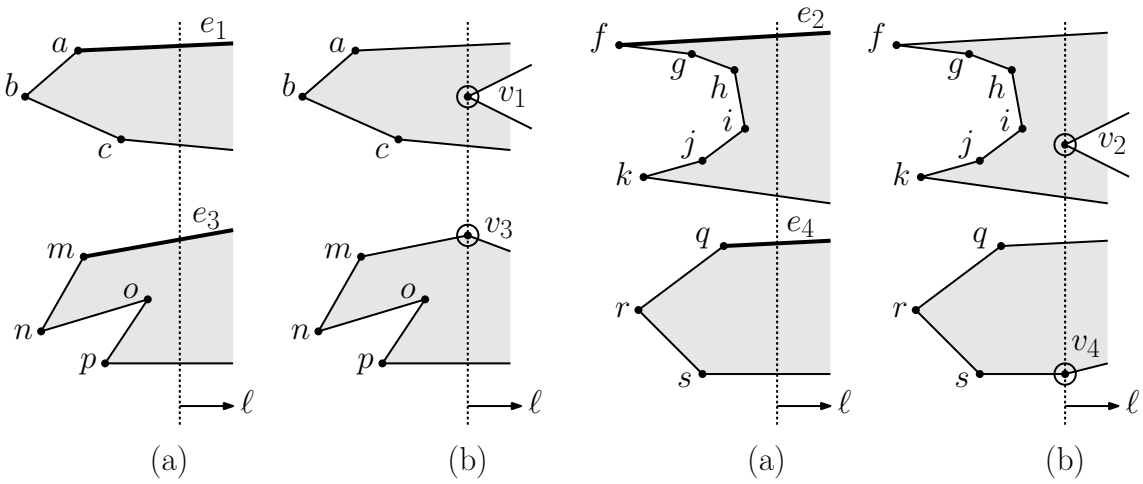


Figure 1: Plane-sweep monotone decomposition.

- Each vertex should have degree two, meaning that there are exactly two segments sharing each endpoint (see Fig. 2(b)).
- The segments form a single closed loop (see Fig. 2(c)).

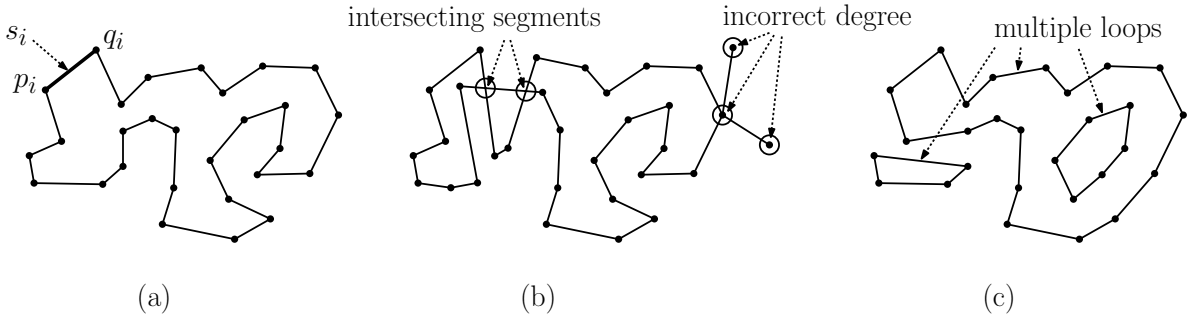


Figure 2: Does a set of segments define a simple polygon?

Present an efficient algorithm to check that the segments form a simple polygon. If the segments violate any of the above conditions, print the first such violation and terminate. Your algorithm should run in $O(n \log n)$ time. (Partial credit will be given if you correctly detect any of the three violations.)

Problem 4. (25 points) Explain how to solve each of the following problems in linear (expected) time. Each can be modeled by reduction to linear programming (LP), perhaps involving multiple instances along with some additional pre- and/or post-processing.

(a) (15 points) In your new career as an archer (shooting arrows), you want to shoot a single arrow through a series of targets. You may stand anywhere you want on the positive y-axis and may shoot in any direction you like. The targets are vertical line segments all in the positive x, y-plane (see Fig. 3(a)). There are n targets. The ith target is specified by giving its center point $c_i = (c_{i,x}, c_{i,y})$ and its height h_i.

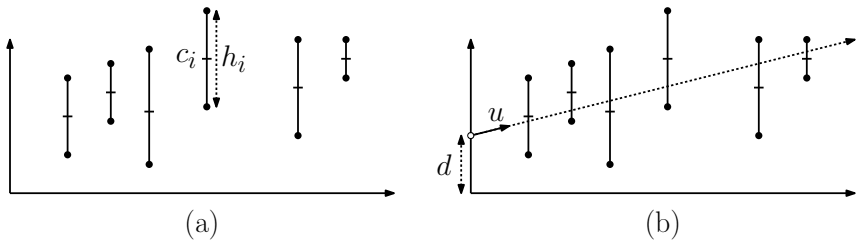


Figure 3: Hitting all targets with a single shot.

Present an algorithm that determines whether there exists a single (straight-line) shot that passes through all n targets. If *any* such a shot exists, output the height d where the shot originates on the y-axis and a directional vector $u = (u_x, u_y)$ that indicates the direction of the shot (see Fig. 3(b)). If there is no shot, indicate this. Your algorithm should run in $O(n)$ time.

(b) (10 points) Consider the same problem as (a) but with the following modification. As in (a), your shot must hit all the targets, but in addition it should come as close as possible (on average) to hitting the centers of the targets. More formally, given any shot, let p_i denote the point where the shot crosses the ith target (see Fig. 4). The objective is to minimize the *absolute value* of $\frac{1}{n}\sum_{i=1}^{n}(p_{i,y} - c_{i,y})$. Explain your algorithm and derive/explain its running time.

Hint: This is a bit tricky, and if you don't see the answer right away, you may want to come back to this later.

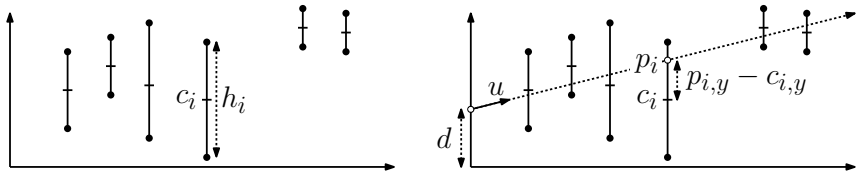


Figure 4: Hitting all targets close to the centers.

Problem 5. (25 points) Our objective is to develop a data structure to determine the first target (if any) that is missed by a shot in the archery problem (Problem 4). The data structure is constructed based on the target centers c_i and their heights h_i (see Fig. 5(a)). A query is given the y-coordinate d along the y-axis where the shot starts and a directional vector u (see Fig. 5(b)).

(a) (15 points) Present a data structure which, given d and $u = (u_x, u_y)$ determines the first target, if any, that is *missed* by the shot. You may assume that $d \geq 0$ and $u_x > 0$. Your data structure should use $O(n)$ space and answer queries in time $O(\log n)$.

(b) (5 points) Suppose that your shot hits all the targets. Assuming that you maintain the same directional vector for the shot, what is the minimum and maximum range of d values such that the shot hits all the targets? Explain how to modify your solution to (a) to answer this in the same space and query time bounds (see Fig. 6(b)).

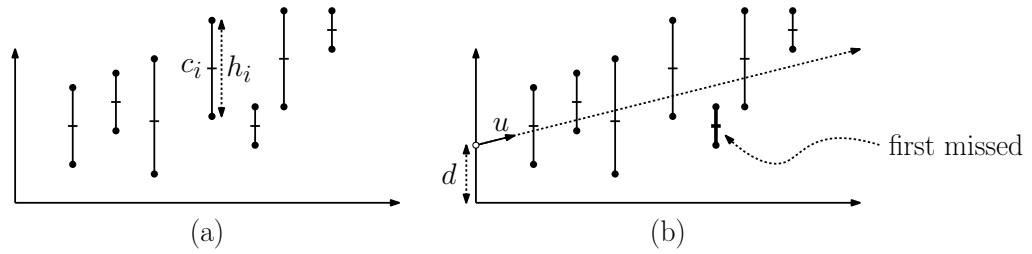


Figure 5: Arrow-shooting queries.

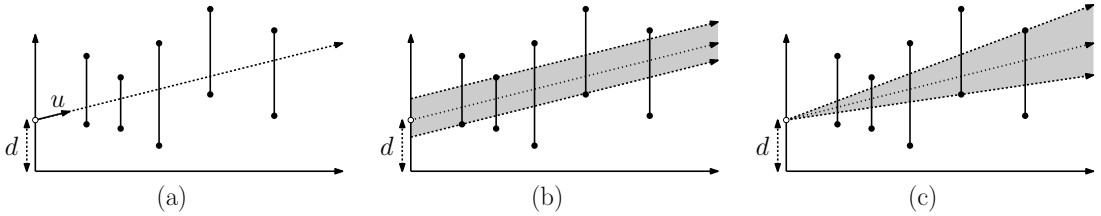


Figure 6: Arrow-shooting variants.

(c) (5 points) Suppose that your shot hits all the targets. Assuming that you maintain the same d value on the y-axis for the shot, what is the minimum and maximum range slopes such that the shot hits all the targets? Explain how to modify your solution to (a) to answer this in the same space and query time bounds (see Fig. 6(c)).

Hint: This is a bit tricky, and if you don't see the answer right away, you may want to come back to this later.

CMSC 754: Fall 2023 Dave Mount

Homework 3: Voronoi/Delaunay, Arrangements, and Search

Handed out Thu, Nov 2. Due at the start of class on **Tue, Nov 28**. 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 $O(f(n))$. If your algorithm involves a plane sweep of a line arrangement and runs in time $O(n^2 \log n)$, you may assume that there exists a topological plane sweep variant that runs in time $O(n^2)$.

Problem 1. (10 points) This problem demonstrates an important application of computational geometry to the field of amphibian navigation. A frog who is a afraid of the water wants to cross a stream that is defined by two horizontal lines $y = y^-$ and $y = y^+$. On the surface there are n circular lily pads whose centers are at the points $P = \{p_1, \ldots, p_n\}$. The frog crosses by hopping across the circular lily pads. The lily pads grow at the same rate, so that at time t they all have radius t (see Fig. 1(a)). Help the frog out by presenting an algorithm that in $O(n \log n)$ time determines the earliest time t^* such that there exists a path from one side of the stream to the other by traveling along the lily pads (see Fig. 1(b)).

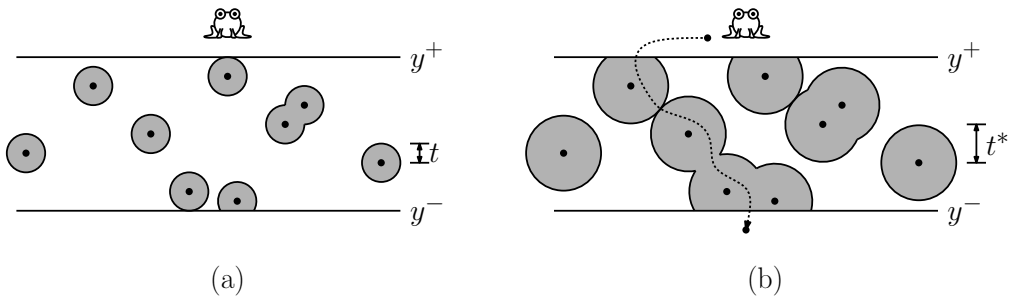


Figure 1: Frog crossing.

Problem 2. (15 points) Define a *slab* to be the region bounded by two parallel (nonvertical) lines, and define its *height* to be the vertical distance between these lines. You are given three sets of points R, W, and B (red, white, and blue) in $\mathbb{R}^2$. A *3-color slab* is a pair of parallel lines such that the closed region bounded between these two lines contains exactly one point from each of R, W, and B in its interior (see Fig. 2). (We do not care what are the colors of the points that lie on the two lines that bound the slab.)

Update (11/8): It was pointed out that with the above definition, an optimal solution may fail to exist. (This is illustrated in Fig. 2(c), where the blue slab is a valid 3-color slab, but it can be made progressively higher by rotating the two bounding lines. When the slab reaches maximum height, shown in yellow, the blue point now lies on the slab's upper boundary, and hence it is not a valid 3-color slab!) To circumvent this issue, we allow solutions like the one shown above, where the slab itself may not be valid, but an infinitesimal perturbation would result in a valid 3-color slab.

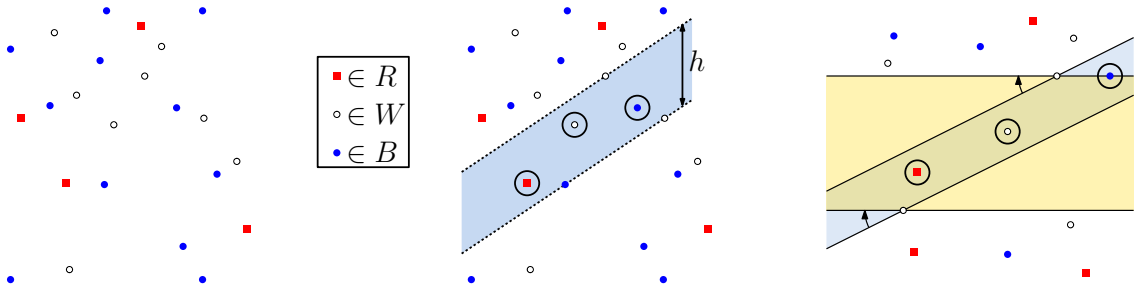


Figure 2: A 3-color slab of height h.

(a) (5 points) Assuming the standard dual transformation (mapping point (a, b) to line $y = ax - b$), explain what a 3-color slab of height h corresponds to in the dual setting.

(b) (5 points) There are generally (uncountably) many 3-color slabs. What local conditions must a 3-color slab satisfy in order to have maximum height? (You may assume that the points are in general position. You may also assume that the given point set has no 3-color slab of infinite height.)

(c) (5 points) Present an algorithm, which given inputs R, W, and B, determines whether there exists a 3-color slab. If not, indicate this. If so, compute the 3-color slab of maximum height. Derive your algorithm's running time and justify its correctness. (**Hint:** For full credit, aim for a running time of $O(n^2 \log n)$ time with $O(n)$ space, where $n = |R| + |W| + |B|$. You may make the same assumption as in part (b).)

Problem 3. (10 points) While kd-trees and range trees are good for axis-aligned queries, they do not perform very well when the query shape is more general. In a *halfplane range counting query*, we are given a point set $P = \{p_1, \ldots, p_n\} \in \mathbb{R}^2$ to be preprocessed into a data structure. Given any halfplane, expressed say by a linear inequality, $H = \{(x, y) : ax + by \leq c\}$, the objective is to count how many points of P lie within H, that is, $|P \cap H|$ (see Fig. 3(a)).

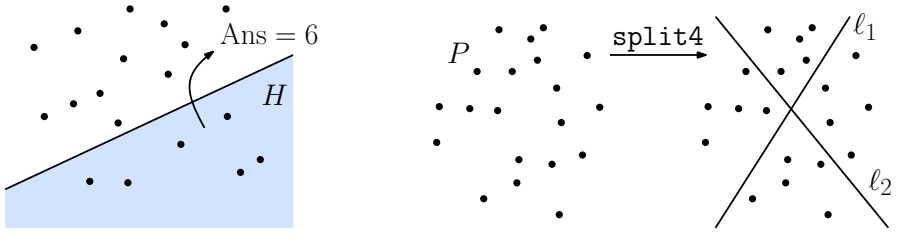


Figure 3: Halfplane range counting.

(a) (5 points) Show that kd-trees do not provide an efficient worst-case solution for halfplane range queries. Give an example of a point set P and a halfplane H such that any (reasonable) query algorithm will require $O(n)$ time to answer this query. (**Hint:** Design

the point set along with a hyperplane so that the line bounding H intersects a constant fraction of the leaves of the kd-tree. We will accept a drawing as a solution, but make it large enough that it is clear how to generalize it to arbitrarily large values of n.)

(b) (5 points) Let's explore a simple (but not optimal) approach for answering halfplane range counting queries. Let us suppose that we have as a "black-box" subroutine, called `split4`, which given any n-element point set P in $\mathbb{R}^2$, generates two lines ℓ_1 and ℓ_2, that partitions P into four subsets (depending on which sides of these lines the points lie) such that each set has at most $\lceil n/4 \rceil$ points (see Fig. 3(b)). (Such a subroutine exists. See the challenge problem.)

Given a point set P, explain how to use the `split4` function to generate a partition tree for P. Show that tree has space $O(n)$ and height $O(\log n)$. Prove that the tree supports halfplane range counting queries in time $O(n^{\log_4 3}) \approx O(n^{0.792})$. (**Hint**: Each node of the tree will have four children. The Master Theorem may be helpful in analyzing the query time.)

Problem 4. (15 points) In this problem, we will consider how to use/modify range trees to answer a number of queries. In each case, the input is an n-element point set P in $\mathbb{R}^2$, and for each explain what points are stored in the range tree, what the various layers of the range tree are, and how queries are answered. Finally, justify your algorithm's correctness and derive its storage and running time as a function of n. You may make the general-position assumption that no point of P lies on the boundary of any range.

(a) (5 points) A *sheared rectangle* is defined by two points $q^- = (x^-, y^-)$ and $q^+ = (x^+, y^+)$. The range shape is a parallelogram that has two horizontal sides and two sides with a slope of -1 (see Fig. 3(a)). The upper left corner is q^- and the lower right corner is q^+. The answer to a *sheared rectangle query* is the number of points of P that lie within the parallelogram.

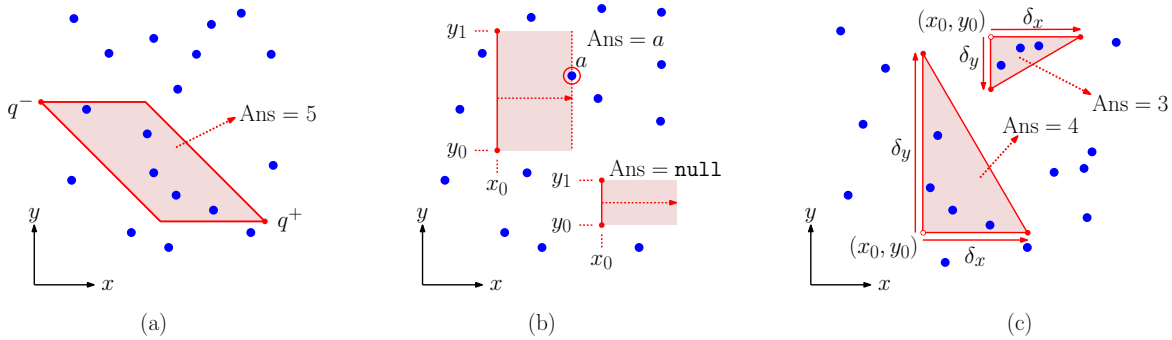


Figure 4: Using range trees to answer various queries.

(b) (5 points) In a *vertical segment-sliding query* (VSS), you are given a vertical line segment with x-coordinate x_0 and endpoints at y-coordinates y_0 and y_1, where $y_0 < y_1$. The answer to the query is the first point that is hit if the segment is translated to the right (see Fig. 4(b)). If the segment can be slid infinitely far to the right without hitting a point of P, the query returns the special value `null`.

(c) (5 points) A *30-60 right triangle* is a right triangle where vertices have angles 30, 60, and 90 degrees. A 30-60 right triangle range is such a triangle, where two of the sides are parallel to the coordinate axes. In a *30-60 triangle query*, you are given such a triangle, and the answer is the number of points of P lying within the triangle (see Fig. 4(c)). The triangle is specified by giving the coordinates (x_0, y_0) of the vertex forming the right angle, and two additional nonzero scalars δ_x and δ_y (which may be positive or negative). The other two vertices are located at $(x_0 + \delta_x, y_0)$ and $(x_0, y_0 + \delta_y)$. Note that there are eight distinct triangle shapes, depending on the signs of δ_x and δ_y and which is larger. Since we are only interested in 30-60 triangles, you may assume that $|\delta_x/\delta_y| = \sqrt{3}$ or $1/\sqrt{3}$.

Hint: In all cases, it is possible to answer queries in $O(\log^c n)$ time, for some constant c. It is sufficient to adapt the standard range-tree approach, and in particular, you do not need to use cascading search.

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. Present a polynomial time algorithm for the function `split4`. That is, given a set P of n points in the plane, your algorithm will output two lines ℓ_1 and ℓ_2 that partition P into four subsets, each of size at most $\lceil n/4 \rceil$.

Hint: First, translate the points so the y-axis partitions P into two subsets of equal size. The y-axis is the line ℓ_1. All you need is to show how to compute ℓ_2. For that, it may be helpful to consider the problem in the dual setting.

Challenge Problem 2. The data structure described in 3(b) can be improved upon. Show how to modify the data structure of 3(b) by replacing the 4-ary tree with a binary tree. Show that, up to constant factors, the query time $T(n)$ satisfies the following recurrence:

$$T(n) \;=\; T(n/2) + S(n/2) + 1 \qquad \text{and} \qquad S(n) \;=\; T(n/2) + 1.$$

Here $T(n)$ denotes the query time for a subtree containing n points, and $S(n)$ denotes the query time for a subtree with the added condition that for one of its two children, all the points lie within H or none of them do. Derive a solution to this recurrence. (**Hint:** Somewhat surprisingly, the answer is related to the Fibonacci sequence.)

CMSC 754: Fall 2023 Dave Mount

Homework 4: WSPDs, Sampling, and Motion Planning

Handed out Tue, Nov 28. Due, Thu, Dec 7, 9:30am. 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. (15 points) This problem involves a couple problems related to computing the statistics on a sets of n points P in $\mathbb{R}^d$. Each can be solved with the use of WSPDs. In each case, assume that you have access to a subroutine that can compute an s-WSPD of size $O(s^d n)$ for any set of n points in $\mathbb{R}^d$ in time $O((n \log n) + s^d n)$. As we did in class, assume that the WSPD is represented as a set of pairs of nodes from a quadtree for P. Each node of the quadtree can store auxiliary information about the points lying within it subtree (e.g., a representative point, the number of points, or any additional statistics involving just these points).

In your answer, you should explain (i) what separation value do you use, (ii) what auxiliary information is stored in each node of the quadtree to help you, (iii) prove your algorithm's correctness, and (iv) indicate its running time.

(a) (5 points) Given a point set $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ and $0 < \varepsilon < 1$ the *average interpoint distance* is defined to be

$$\Delta(P) \;=\; \frac{1}{\binom{n}{2}} \sum_{1 \le i < j \le n} \|p_i - p_j\|.$$

Computing this naively would take $O(n^2)$ time. Present an efficient WSPD-based algorithm that computes an ε-approximation to $\Delta(P)$. It computes a value $\widetilde{\Delta}(P)$ such that

$$\frac{1}{1+\varepsilon}\Delta(P) \;\le\; \widetilde{\Delta}(P) \;\le\; (1+\varepsilon)\Delta(P).$$

(b) (10 points) Solve (a), but this time with the generalization that we want to compute the average of the kth powers of the distances. Given an integer $k \ge 1$, define

$$\Delta^{[k]}(P) \;=\; \frac{1}{\binom{n}{2}} \sum_{1 \le i < j \le n} \|p_i - p_j\|^k.$$

(This is also known as the *kth moment* of the interpoint distance.) Assuming that k is a constant (independent of d or n) present an efficient WSPD-based algorithm that computes an ε-approximation to $\Delta(P)$. It computes a value $\widetilde{\Delta}^{[k]}(P)$ such that

$$\frac{1}{1+\varepsilon}\Delta^{[k]}(P) \;\le\; \widetilde{\Delta}^{[k]}(P) \;\le\; (1+\varepsilon)\Delta^{[k]}(P).$$

(**Hint:** The structure is essentially the same as part (a). It will be helpful to have an approximation for $(1+\varepsilon)^k$, assuming $0 < \varepsilon < 1$. For this, it is useful to recall the *Binomial Theorem*, $(a+b)^k = \sum_{i=0}^{k} \binom{k}{i} a^i b^{k-i}$.

Problem 2. (10 points) While coresets provide close approximations to some statistic of interest, it is often useful to have an easily computable lower bound or upper bound, that is just within a constant fraction of the exact value. In this problem, we'll consider a crude lower bound for the weight of the Euclidean minimum spanning tree in $\mathbb{R}^d$. Recall that in dimensions 3 and higher, computing the Euclidean minimum spanning tree exactly takes nearly quadratic time.

You are given a set of n points $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$. Let $\text{emst}(P)$ denote the weight of P's Euclidean minimum spanning tree. We will compute our lower bound as follows. Place the points of P in a unit square grid, that is, a grid of hypercubes, each of side length 1 (see Fig. 1(a)). Let $N(P)$ denote the number of grid squares that contain at least one point of P.

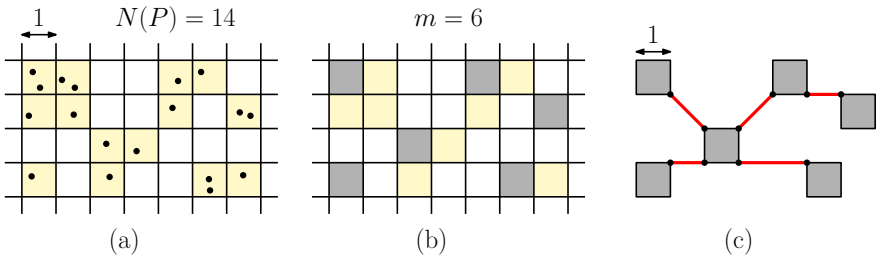


Figure 1: Lower bound for Euclidean minimum spanning tree.

Prove that there exist positive constants a_d and c_d (depending on the dimension d, but not on n), such that if $N(P) \geq a_d$ then

$$\text{emst}(P) \;\geq\; \frac{N(P)}{c_d}.$$

It is not necessary to derive the best values of a_d and c_d, but as a hint, both quantities vary exponentially with d.

Hint: This can be done in two parts. For the first part, let $G(P)$ denote the grid squares that contain at least one point of P, meaning that $N(P) = |G(P)|$. (These are the yellow squares in Fig. 1(a)). Show that if $N(P)$ is sufficiently large (depending on d) there is a subset of $G(P)$ containing at least a constant fraction of squares, so that no two of these squares have boundaries that touch each other. (These are the gray squares in Fig. 1(b)). For the second part, show that if you are given any set of m unit grid squares, where no two have boundaries that touch, then any spanning tree connecting these squares has weight at least $m - 1$ (see Fig. 1(c)). Finally, explain how these two facts can be combined to prove the lower bound on $\text{emst}(P)$.

Problem 3. (15 points) 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*.

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.

(a) (5 points) $\Sigma = (\mathbb{R}^2, \mathcal{U})$, where $\mathcal{U}$ consists of all orthogonal U-shaped ranges. An *orthogonal U-shaped range* is defined by three numbers (x_0, x_1, y_0). It is the region bounded between the two vertical lines $x_0 \leq x \leq x_1$ and above a horizontal line $y \geq y_0$ (see Fig. 2(a)).

(b) (10 points) $\Sigma = (\mathbb{R}^2, \mathcal{V})$, where $\mathcal{V}$ consists of all orthogonal V-shaped ranges. An *orthogonal V-shaped range* is defined by four numbers (x_0, x_1, a, b). It is the region bounded between the two vertical lines $x_0 \leq x \leq x_1$ and above a line $y \geq ax + b$ (see Fig. 2(b)).

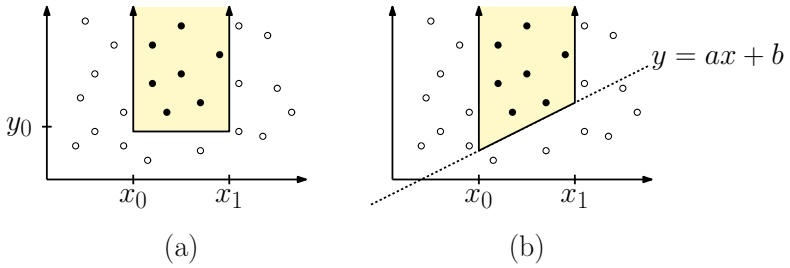


Figure 2: VC-Dimension of some range spaces.

Problem 4. (10 points) Let us consider a motion planning problem in the plane involving a set of rectangular obstacles and a translating "L"-shaped robot (see Fig. 3(a)). The robot $\mathcal{R}$ is an "L"-shaped polygon as shown in Fig. 3(b). Its reference point is its lower-left corner. The obstacles consist of a collection of non-intersecting rectangles $\{O_1, \ldots, O_n\}$, where O_i has lower-left corner p_i and horizontal length ℓ_i and vertical height h_i (see Fig. 3(c)).

You are given a start point s and target point t. Assuming the robot's reference point starts at s, is there an obstacle-avoiding motion that terminates with the endpoint at t? In this problem will explore a solution to this problem.

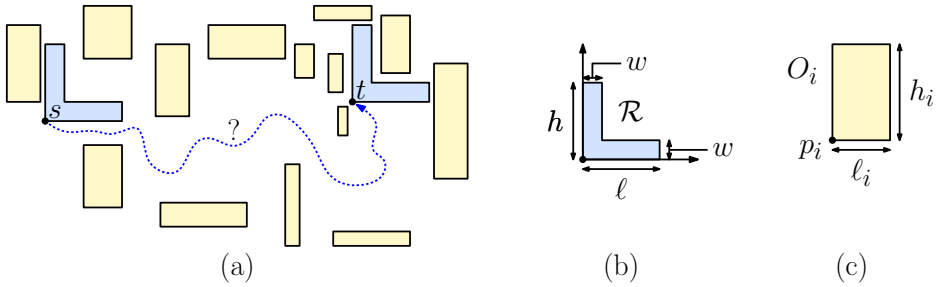


Figure 3: L-shaped robot motion planning.

(a) (5 points) Given an obstacle O_i, provide a clear description and a picture of the associated C-obstacle (its shape, side lengths, and location).

(b) (5 points) Given the starting and target points s and t, sketch an algorithm for determining whether there is a collision-free translational motion of the robot with its reference point starting at s and ending at t.

A high-level sketch of the algorithm is sufficient. To make your life simpler, you may assume that you are given a procedure that will input the C-obstacles from part (a), compute their union, and returns a convenient decomposition of free-space (e.g., as a trapezoidal map). Your algorithm should be efficient, in the sense that its running time should be roughly proportional to the combinatorial complexity of the free space, that is, the union of the C-obstacles. (See the Challenge Problem.)

Challenge Problem. Recalling the setup in Problem 4, what is the worst-case combinatorial complexity of the free space, that is, the union of all the C-obstacles? You may select any values you like for the robot's dimensions (h, ℓ, and w), and you may place the n rectangles anywhere you like. Justify your answer.

Sample Problems for the Final Exam

The final exam will be **Thu, Dec 14, 8:00am-10:00am** in class. The exam will be closed-book and closed-notes. You may use *two* sheets of notes (front and back). 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 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) What was the importance of the Zone Theorem in our incremental algorithm for building line arrangements in the plane?

(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) Let P and Q be two simple polygons in $\mathbb{R}^2$, where P has m vertices and Q has n vertices. What is the maximum number of vertices on the boundary of the Minkowski sum $P \oplus Q$ (asymptotically) assuming:

(i) P and Q are both convex

(ii) P is convex but Q is arbitrary

(iii) P and Q are both arbitrary

(g) In each of the following cases, what is the asymptotic worst-case complexity (number of vertices) on the boundary of the union of n of the following objects in $\mathbb{R}^2$:

(i) axis-parallel squares

(ii) axis-parallel rectangles (of arbitrary heights and widths)

(iii) rectangles (of arbitrary heights and widths which need not be axis parallel)

(iv) axis-parallel rectangles, where the width to height ratio is either 4×1 or 1×4.

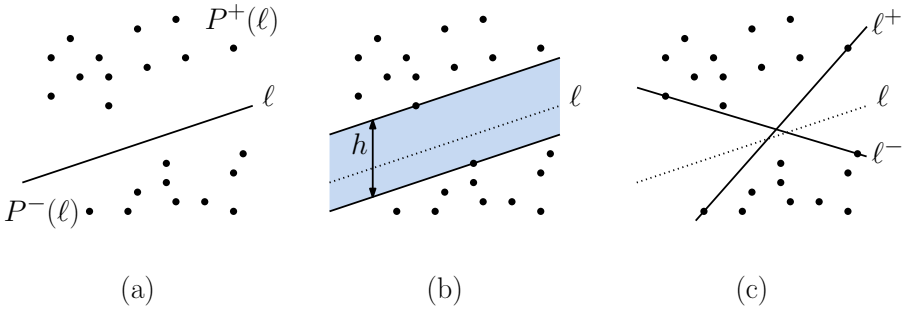


Figure 1: Query problem.

Problem 2. You are given a set P of n points in $\mathbb{R}^2$. A nonvertical line ℓ 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)).

Given the point set P, 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, but it should be constructable in polynomial time in n.)

(a) The input to the query is a nonvertical line ℓ. 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).

(b) Again, the input to the query is a nonvertical line ℓ. The answer to the query are 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. 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.

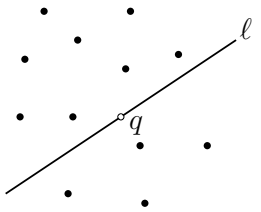


Figure 2: Point q is 4-deep within P.

For example, the point q in Fig. 2 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) 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) 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) 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 4. 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.)

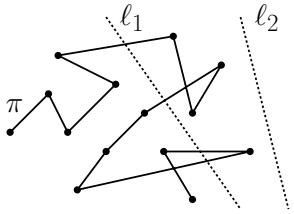


Figure 3: Path crossing queries.

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. 3 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 5. 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. 4(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. 4(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. 4(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. 4(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.)

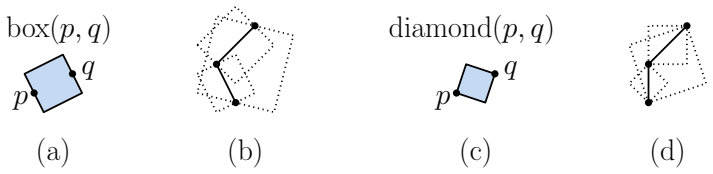


Figure 4: The box and diamond graphs.

Problem 6. Consider the range space $\Sigma = (\mathbb{R}^2, \mathcal{T})$, where $\mathcal{T}$ is the set of all right triangles whose two legs are parallel to the coordinate axes, so that the right angle is in the lower-left corner of the triangle (see Fig. 5).

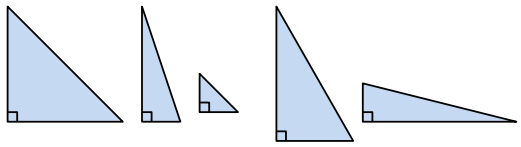


Figure 5: Set system of axis-aligned right triangles.

(a) Give an example of a 4-element point set P in $\mathbb{R}^2$ that is shattered by Σ, and demonstrate why it is shattered. (For preciseness, indicate the coordinates of the points, but you can present a drawing to illustrate how to shatter them.)

(b) Prove that no 5-element point set in $\mathbb{R}^2$ is shattered by Σ.

(c) What is the VC-dimension of Σ?

(d) Given a point set P in $\mathbb{R}^2$ with n points, give a (tight) asymptotic upper bound on the number of distinct subsets of P determined by Σ. (Using the notation given in class, this is $|\mathcal{T}_{|P}|$.)

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.

Problem 8. 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 $O((n \log n)/\varepsilon^d)$.

Problem 9. 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 $\delta > 0$, and you are to return a count of all the pairs of points $(p, q) \in P \times P$, such that $\|pq\| \geq \delta$. In an *ε-approximate distance query*, your count *must* include all pairs (p, q) where $\|pq\| \geq \delta(1+\varepsilon)$ and it *must not* include any pairs (p, q) where $\|pq\| < \delta/(1+\varepsilon)$. 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 10. This problem considers motion planning in a dynamic setting, which is inspired by various old video games. You are given a *robot* that consists of a line segment of unit length that resides on the x-axis. The robot can move left or right (but not up or down) at a speed of up to one unit per second. You are given two real values x^- and x^+, and the robot must remain entirely between these two values at all times (see Fig. 6). The robot's *reference point* is its left endpoint, and at time $t = 0$, the left endpoint is located at x^-. (You may assume that $x^+ > x^- + 1$.)

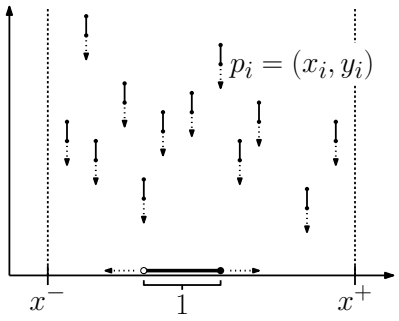


Figure 6: Robot motion planning.

You are also given a set of *missiles* in the form of n vertical line segments, each of length 0.2, that fall down from the sky at a rate of two units per second. Each of these vertical segments is specified by the coordinates of its lower endpoint at time $t = 0$. So, if $p_i = (x_i, y_i)$ is the starting position of the ith missile, then at time t is its lower endpoint is located at $(x_i, y_i - 2t)$, and its upper endpoint is at $(x_i, y_i - 2t + 0.2)$. You may assume that $x^- \leq x_i \leq x^+$.

The question is whether it is possible for the robot to move in a manner to avoid all the missiles. We will explore an algorithm for solving this problem.

(a) A natural way to define the robot's configuration at any time is as a pair (t, x), where

t is the current time, and x is the location of the robot's left endpoint. Based on this, what is the C-obstacle associated with a missile whose starting position is p_i (as defined above)? In other words, describe the set of robot configurations (t, x) such that the robot intersects this missile. (Please provide low-level details, as opposed, say, to expressing this as a Minkowski sum.)

(b) Provide a complete characterization of the properties of a path in configuration space (assuming it exists) that corresponds to a motion plan for the robot that satisfies the robot's speed constraints and avoids all the missiles. Be sure to include constraints on the path's starting and ending positions and include the robot's maximum speed.

CMSC 754: Final Exam

In all problems, unless otherwise stated, you may assume that inputs are in *general position*. If you are asked for an $O(T(n))$ time algorithm, you may give a *randomized* algorithm with *expected* time $O(T(n))$. If you are asked to give an algorithm, also explain how it works and derive its running time. (Unless otherwise stated, formal proofs of correctness are not required.)

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

(a) (5 points) Given a simple polygon with n sides, what can be said about the number of triangles in any triangulation? Either state the exact number as a function of n, or provide upper and lower bounds.

(b) (8 points) In class we showed that the *trapezoidal map* of n nonintersecting line segments has at most $6n + 4$ vertices and at most $3n + 1$ trapezoids. Suppose instead that there are k instances where two segments intersect. (In Fig. 1 there are $n = 3$ segments and $k = 2$ intersection points.) At each intersection point, we shoot two bullet paths, up and down. As a function of n and k, what is the (exact) number of vertices and trapezoids in the resulting trapezoidal map? (No explanation needed.)

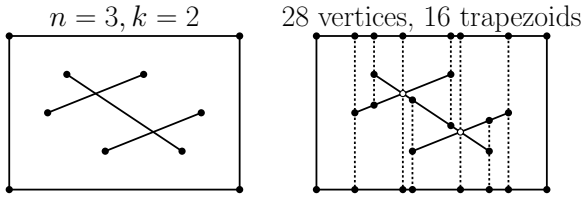


Figure 1: A trapezoidal map with $n = 3$ segments and $k = 2$ intersection points.

(c) (4 points) In our backward analysis of randomized incremental algorithms, we were careful to argue that the structure being analyzed does *not* depend on the order of insertion. What aspect of the analysis would *fail* if the structure did depend on the order of insertion? (Explain briefly.)

(d) (5 points) Consider the portion of the line arrangement shown in Fig. 2. Illustrate the *zone* of the line ℓ in this arrangement.

(e) (8 points) Suppose you have a workspace with n disjoint obstacles, each of which is a k-sided convex polygon, and you are given a robot that translates in this workspace, which is modeled as an m-sided convex polygon.

(i) As a function of n, k, and m, give an upper bound on the total number of vertices in all the C-obstacles? (You may express your answer using big-Oh notation.) Explain briefly.

(ii) The C-obstacles may overlap. As a function of n, k, and m, give an upper bound on the total number of vertices in the *union* of the C-obstacles? (Again, you can use big-Oh notation.) Explain briefly.

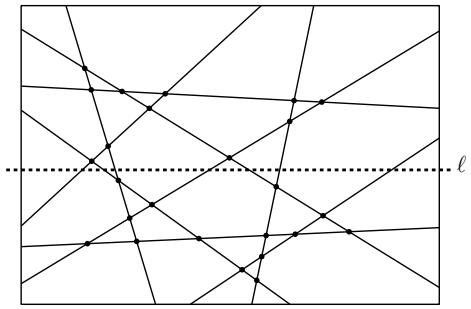


Figure 2: The zone of an arrangement.

Problem 2. (20 points) Present LP solutions to the following problems. In each case, explain how the problem is formulated as an instance of LP (and what the dimension of the space is), and how the result of the LP (feasible, infeasible, unbounded) is to be interpreted in answering the problem.

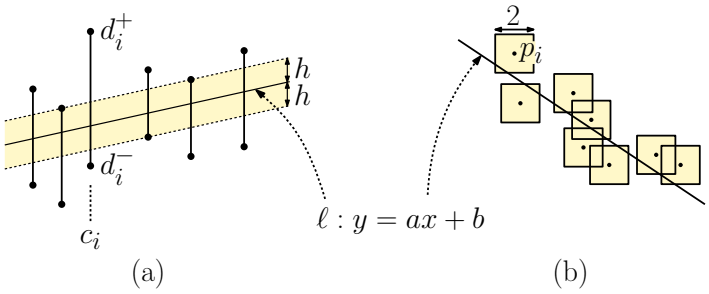


Figure 3: Stabbing segments and squares.

(a) (10 points) 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 c_i, its upper y-coordinate d_i^+ and its lower y-coordinate d_i^-. Compute a line $\ell : y = ax + b$ that intersects all of these segments. If it exists, return the line that maximizes the vertical spacing h above and below the line (see Fig. 3(a)).

(b) (10 points) You are given a collection of n axis-aligned squares in the plane, each of side length 2. The squares are centered at the points $P = \{p_1, \ldots, p_n\}$, where $p_i = (c_i, d_i)$ (see Fig. 3(b)). Determine whether there exists a line $\ell : y = ax + b$ that intersects all of these squares. If it exists, return any such line. **Hint:** You may need to make multiple ($O(1)$) calls to LP.

Problem 3. (30 points) For each of the following search problems, explain how to map this to a data structure that we have seen this semester. In each case, explain which data structure you will use (e.g., point-location, kd-tree, range tree, etc.), what information is stored in the data structure, how much space it uses, and what the query time is. There may be multiple options, and if so, select one that is most efficient.

The goal is to achieve space that is either linear in n or slightly higher (e.g., $O(n)$, $O(n \log n)$, or $O(n \log^2 n)$), and to achieve a query time that $O(\log n)$ or slightly higher (e.g., $O(\log^2 n)$, or $O(\log^3 n)$). We will give partial credit for a correct answer, even if the space or query time is not optimal.

Hint: Keep your answers brief. We are looking for a reduction to known data structure, perhaps with additional minor modifications.

(a) (10 points) The data is a set of n points P in $\mathbb{R}^2$. The query is an axis-aligned rectangle $Q = [x_0, x_1] \times [y_0, y_1]$. The answer to the query is the area of the smallest axis-aligned rectangle that contains all the points of $P \cap Q$ (see Fig. 4(a)).

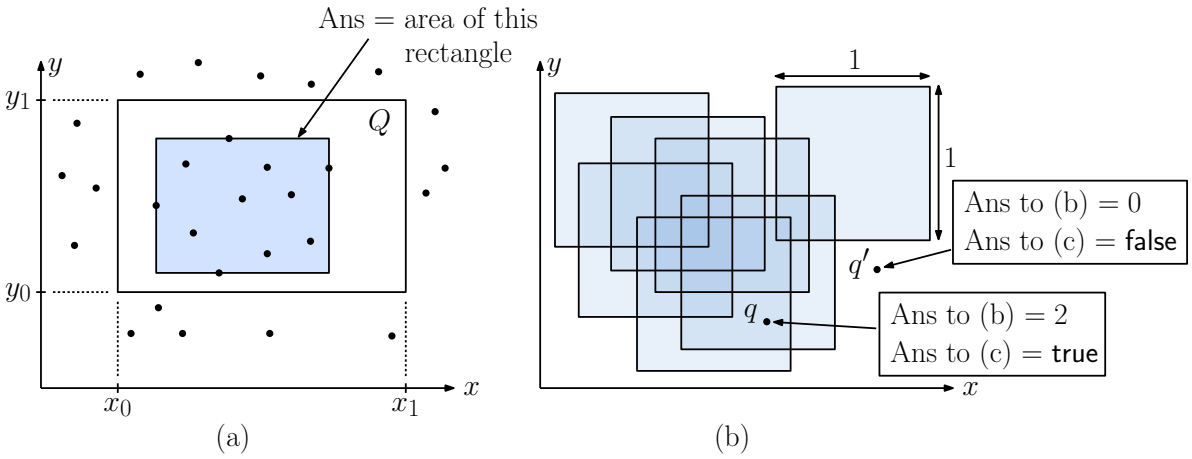


Figure 4: Geometric queries.

(b) (10 points) The data is a set S of n axis-aligned unit squares in $\mathbb{R}^2$ (that is, each has side length 1). The query is a point $q = (q_x, q_y)$. The answer to the query is the number of squares of S that contain q (see Fig. 4(b)).

(c) (10 points) The data set is again a set S of n axis-aligned unit squares in $\mathbb{R}^2$ (that is, each has side length 1). The query is a point $q = (q_x, q_y)$. The answer to the query is true if q lies in the union of the squares and false otherwise (see Fig. 4(b)).

Problem 4. (20 points) 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.

Hint: Proving the upper bound on the VC-dimension can involve multiple cases. Do your best to explain what the relevant cases are, but don't bother proving each one formally. For each case, you can give a short one-sentence explanation (or draw a picture) to explain which subsets cannot be generated.

(a) (8 points) $\Sigma = (\mathbb{R}^2, \mathcal{S})$, where $\mathcal{S}$ consists of parallelograms having two horizontal sides and two sides that have a slope of 1 (see Fig. 5(a)).

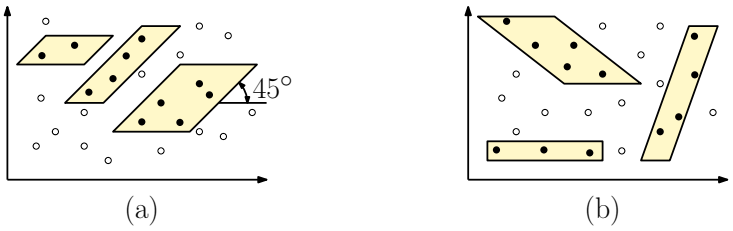


Figure 5: VC-Dimension of some range spaces.

(b) (12 points) $\Sigma = (\mathbb{R}^2, \mathcal{P})$, where $\mathcal{P}$ consists of parallelograms having two horizontal sides and two sides of arbitrary (nonzero) slope (see Fig. 5(b)).

Problem 5. (20 points) You are given a set P of n points in $\mathbb{R}^d$ and a real $\delta > 0$. The objective of a *δ-distance query* is to return a count of all the pairs of distinct points $x, y \in P$, such that $\|x - y\| \leq \delta$. Given $0 < \varepsilon < 1$, in an *ε-approximate δ-distance query*, your count *must* include all pairs such that $\|x - y\| \leq \delta$, and it *must not* include any pair such that $\|x - y\| > (1 + \varepsilon)\delta$. 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 what modifications are needed to the WSPD construction. It is not necessary to derive the tightest possible bound on the separation factor, but formally justify your choice.