33 Computational Geometry

33.1 Line-segment properties

33.1-1

> Prove that if $p_1 \times p_2$ is positive, then vector p_1 is clockwise from vector p_2 with respect to the origin $(0, 0)$ and that if this cross product is negative, then p_1 is counterclockwise from p_2.

(Omit!)

33.1-2

> Professor van Pelt proposes that only the x-dimension needs to be tested in line 1 of ON-SEGMENT. Show why the professor is wrong.

$(0, 0), (5, 5), (10, 0)$.

33.1-3

> The ***polar angle*** of a point p_1 with respect to an origin point p_0 is the angle of the vector $p_1 - p_0$ in the usual polar coordinate system. For example, the polar angle of $(3, 5)$ with respect to $(2, 4)$ is the angle of the vector $(1, 1)$, which is 45 degrees or $\pi/4$ radians. The polar angle of $(3, 3)$ with respect to $(2, 4)$ is the angle of the vector $(1, 1)$, which is 315 degrees or $7\pi/4$ radians. Write pseudocode to sort a sequence $\langle p_1, p_2, \ldots, p_n \rangle$ of n points according to their polar angles with respect to a given origin point p_0. Your procedure should take $O(n \lg n)$ time and use cross products to compare angles.

(Omit!)

33.1-4

> Show how to determine in $O(n^2 \lg n)$ time whether any three points in a set of n points are colinear.

Based on exercise 33.1-3, for each point p_i, let p_i be p_0 and sort other points according to their polar angles mod π. Then scan linearly to see whether two points have the same polar angle. $O(n \cdot n \lg n) = O(n^2 \lg n)$.

33.1-5

> A ***polygon*** is a piecewise-linear, closed curve in the plane. That is, it is a curve ending on itself that is formed by a sequence of straight-line segments, called the ***sides*** of the polygon. A point joining two consecutive sides is a ***vertex*** of the polygon. If the polygon is ***simple***, as we shall generally assume, it does not cross itself. The set of points in the plane enclosed by a simple polygon forms the ***interior*** of the polygon, the set of points on the polygon itself forms its ***boundary***, and the set of points surrounding the polygon forms its ***exterior***. A simple polygon is convex if, given any two points on its

> boundary or in its interior, all points on the line segment drawn between them are contained in the polygon's boundary or interior. A vertex of a convex polygon cannot be expressed as a convex combination of any two distinct points on the boundary or in the interior of the polygon.

> Professor Amundsen proposes the following method to determine whether a sequence $\langle p_1, p_2, \ldots, p_{n-1} \rangle$ of n points forms the consecutive vertices of a convex polygon. Output "yes" if the set $\angle p_i p_{i+1} p_{i+2} : i = 0, 1, \ldots, n-1$, where subscript addition is performed modulo n, does not contain both left turns and right turns; otherwise, output "no." Show that although this method runs in linear time, it does not always produce the correct answer. Modify the professor's method so that it always produces the correct answer in linear time.

A line.

33.1-6

> Given a point $p_0 = (x_0, y_0)$, the ***right horizontal ray*** from p_0 is the set of points $p_i = (x_i, y_i) : x_i \geq x_0 \text{ and } y_i = y_0$, that is, it is the set of points due right of p_0 along with p_0 itself. Show how to determine whether a given right horizontal ray from p_0 intersects a line segment $\overline{p_1 p_2}$ in $O(1)$ time by reducing the problem to that of determining whether two line segments intersect.

$p_1.y = p_2.y = 0$ and $\max(p_1.x, p_2.x) \geq 0$.

or

$\text{sign}(p_1.y) \neq \text{sign}(p_2.y)$ and $\displaystyle p_1.y \cdot \frac{p_1.x - p_2.x}{p_1.y - p_2.y} \ge 0{equation}

33.1-7

> One way to determine whether a point p_0 is in the interior of a simple, but not necessarily convex, polygon P is to look at any ray from p_0 and check that the ray intersects the boundary of P an odd number of times but that p_0 itself is not on the boundary of P. Show how to compute in $\Theta(n)$ time whether a point p_0 is in the interior of an n-vertex polygon P. (Hint: Use Exercise 33.1-6. Make sure your algorithm is correct when the ray intersects the polygon boundary at a vertex and when the ray overlaps a side of the polygon.)

Based on exercise 33.1-6, use $p_i - p_0$ as p_i.

33.1-8

> Show how to compute the area of an n-vertex simple, but not necessarily convex, polygon in $\Theta(n)$ time. (See Exercise 33.1-5 for definitions pertaining to polygons.)

Half of the sum of the cross products of $\overline{p_1 p_i, p_1 p_{i+1}} \rceil \; i \in [2, n-1]$.

33.2 Determining whether any pair of segments intersects

33.2-1

> Show that a set of n line segments may contain $\Theta(n^2)$ intersections.

Star.

33.2-2

> Given two segments a and b that are comparable at x, show how to determine in $O(1)$ time which of $a \succeq_x b$ or $b \succeq_x a$ holds. Assume that neither segment is vertical.

(Omit!)

33.2-3

> Professor Mason suggests that we modify ANY-SEGMENTS-INTERSECT so that instead of returning upon finding an intersection, it prints the segments that intersect and continues on to the next iteration of the **for** loop. The professor calls the resulting procedure PRINT-INTERSECTING-SEGMENTS and claims that it prints all intersections, from left to right, as they occur in the set of line segments. Professor Dixon disagrees, claiming that Professor Mason's idea is incorrect. Which professor is right? Will PRINT-INTERSECTING-SEGMENTS always find the leftmost intersection first? Will it always find all the intersections?

No.

33.2-4

> Give an $O(n \lg n)$-time algorithm to determine whether an n-vertex polygon is simple.

Same as ANY-SEGMENTS-INTERSECT.

33.2-5

> Give an $O(n \lg n)$-time algorithm to determine whether two simple polygons with a total of n vertices intersect.

Same as ANY-SEGMENTS-INTERSECT.

33.2-6

> A ***disk*** consists of a circle plus its interior and is represented by its center point and radius. Two disks intersect if they have any point in common. Give an $O(n \lg n)$- time algorithm to determine whether any two disks in a set of n intersect.

Same as ANY-SEGMENTS-INTERSECT.

33.2-7

> Given a set of n line segments containing a total of k intersections, show how to output all k intersections in $O((n+k)\lg)$ time.

Treat the intersection points as event points.

33.2-8

> Argue that ANY-SEGMENTS-INTERSECT works correctly even if three or more segments intersect at the same point.

(Omit!)

33.2-9

> Show that ANY-SEGMENTS-INTERSECT works correctly in the presence of vertical segments if we treat the bottom endpoint of a vertical segment as if it were a left endpoint and the top endpoint as if it were a right endpoint. How does your answer to Exercise 33.2-2 change if we allow vertical segments?

(Omit!)

33.3 Finding the convex hull

33.3-1

> Prove that in the procedure GRAHAM-SCAN, points p_1 and p_m must be vertices of $CH(Q)$.

To see this, note that p_1 and p_m are the points with the lowest and highest polar angle with respect to p_0. By symmetry, we may just show it for p_1 and we would also have it for p_m just by reflecting the set of points across a vertical line.

To see a contradiction, suppose that we have the convex hull doesn't contain p_1. Then, let p be the point in the convex hull that has the lowest polar angle with respect to p_0. If p is on the line from p_0 to p_1, we could replace it with p_1 and have a convex hull, meaning we didn't start with a convex hull.

If we have that it is not on that line, then there is no way that the convex hull given contains p_1, also contradicting the fact that we had selected a convex hull

33.3-2

> Consider a model of computation that supports addition, comparison, and multiplication and for which there is a lower bound of $\Omega(n \lg n)$ to sort n numbers. Prove that $\Omega(n \lg n)$ is a lower bound for computing, in order, the vertices of the convex hull of a set of n points in such a model.

Let our n numbers be $a_1, a_2, \ldots, a_n$ and f be a strictly convex function, such as e^x. Let $p_i = (a_i, f(a_i))$. Compute the convex hull of $p_1, p_2, \ldots, p_n$. Then every point is in the convex hull. We can recover the numbers themselves by looking at the x-coordinates of the points in the order returned by the convex-hull algorithm,

which will necessarily be a cyclic shift of the numbers in increasing order, so we can recover the proper order in linear time.

In an algorithm such as GRAHAM-SCAN which starts with the point with minimum y-coordinate, the order returned actually gives the numbers in increasing order.

33.3-3

> Given a set of points Q, prove that the pair of points farthest from each other must be vertices of $\text{CH}(Q)$.

Suppose that p and q are the two furthest apart points. Also, to a contradiction, suppose, without loss of generality that p is on the interior of the convex hull. Then, construct the circle whose center is q and which has p on the circle. Then, if we have that there are any vertices of the convex hull that are outside this circle, we could pick that vertex and q, they would have a higher distance than between p and q. So, we know that all of the vertices of the convex hull lie inside the circle. This means that the sides of the convex hull consist of line segments that are contained within the circle. So, the only way that they could contain p, a point on the circle is if it was a vertex, but we suppsed that p wasn't a vertex of the convex hull, giving us our contradiction.

33.3-4

> For a given polygon P and a point q on its boundary, the ***shadow*** of q is the set of points r such that the segment $\overline{qr}$ is entirely on the boundary or in the interior of P. As Figure 33.10 illustrates, a polygon P is ***star-shaped*** if there exists a point p in the interior of P that is in the shadow of every point on the boundary of P. The set of all such points p is called the ***kernel*** of P. Given an n-vertex, star-shaped polygon P specified by its vertices in counterclockwise order, show how to compute $\text{CH}(P)$ in $O(n)$ time.

We simply run GRAHAM-SCAN but without sorting the points, so the runtime becomes $O(n)$. To prove this, we'll prove the following loop invariant: At the start of each iteration of the for loop of lines 7-10, stack S consists of, from bottom to top, exactly the vertices of $\text{CH}(Q_{i-1})$. The proof is quite similar to the proof of correctness. The invariant holds the first time we execute line 7 for the same reasons outline in the section. At the start of the ith iteration, S contains $\text{CH}(Q_{i-1})$. Let p_j be the top point on S after executing the while loop of lines 8-9, but before p_i is pushed, and let pk be the point just below p_j on S. At this point, S contains $\text{CH}(Q_j)$ in counterclockwise order from bottom to top. Thus, when we push p_i, S contains exactly the vertices of $\text{CH}(Q_j \cup \{p_i\})$.

We now show that this is the same set of points as $\text{CH}(Q_i)$. Let p_t be any point that was popped from S during iteration i and p_r be the point just below p_t on stack S at the time p_t was popped. Let p be a point in the kernel of P. Since the angle $\angle p_r p_t p_i$ makes a nonelft turn and P is star shaped, p_t must be in the interior or on the boundary of the triangle formed by p_r, p_i, and p. Thus, p_t is not in the convex hull of Q_i, so we have $\text{CH}(Q_i - \{p_t\}) = \text{CH}(Q_i)$. Applying this equality repeatedly for each point removed from S in the while loop of lines 8-9, we have $\text{CH}(Q_j \cup \{p_i\}) = \text{CH}(Q_i)$.

When the loop terminates, the loop invariant implies that S consists of exactly the vertices of $\text{CH}(Q_m)$ in counterclockwise order, proving correctness.

33.3-5

> In the ***on-line convex-hull problem***, we are given the set Q of n points one point at a time. After receiving each point, we compute the convex hull of the points seen so far. Obviously, we could run Graham's scan once for each point, with a total running time of $O(n^2 \lg n)$. Show how to solve the on-line convex-hull problem in a total of $O(n^2)$ time.

Suppose that we have a convex hull computed from the previous stage $\{q_0, q_1, \ldots, q_m\}$, and we want to add a new vertex, p in and keep track of how we should change the convex hull.

First, process the vertices in a clockwise manner, and look for the first time that we would have to make a non-left to get to p. This tells us where to start cutting vertices out of the convex hull. To find out the upper bound on the vertices that we need to cut out, turn around, start processing vertices in a clockwise manner and see the first time that we would need to make a non-right.

Then, we just remove the vertices that are in this set of vertices and replace the with p. There is one last case to consider, which is when we end up passing ourselves when we do our clockwise sweep.

Then we just remove no vertices and add p in in between the two vertices that we had found in the two sweeps. Since for each vertex we add we are only considering each point in the previous step's convex hull twice, the runtime is $O(nh) = O(n^2)$ where h is the number of points in the convex hull.

33.3-6 ⋆

> Show how to implement the incremental method for computing the convex hull of n points so that it runs in $O(n \lg n)$ time.

(Omit!)

33.4 Finding the closest pair of points

33.4-1

> Professor Williams comes up with a scheme that allows the closest-pair algorithm to check only 5 points following each point in array Y'. The idea is always to place points on line l into set P_L. Then, there cannot be pairs of coincident points on line l with one point in P_L and one in P_R. Thus, at most 6 points can reside in the $\delta \times 2\delta$ rectangle. What is the flaw in the professor's scheme?

In particular, when we select line l, we may be unable perform an even split of the vertices. So, we don't neccesarily have that both the left set of points and right set of points have fallen to roughly half. For example, suppose that the points are all arranged on a vertical line, then, when we recurse on the the left set of points, we haven't reduced the problem size at all, let alone by a factor of two. There is also the issue in this setup that you may end up asking about a set of size less than two when looking at the right set of points.

33.4-2

> Show that it actually suffices to check only the points in the 5 array positions following each point in the array Y'.

Since we only care about the shortest distance, the distance δ' must be strictly less than δ. The picture in Figure 33.11(b) only illustrates the case of a nonstrict inequality. If we exclude the possibility of points whose x coordinate differs by exactly δ from l, then it is only possible to place at most 6 points in the $\delta \times 2\delta$ rectangle, so it suffices to check on the points in the 5 array positions following each point in the array Y'.

33.4-3

> We can define the distance between two points in ways other than euclidean. In the plane, the L_m-***distance*** between points p_1 and p_2 is given by the expression $(|x_1 - x_2|^m + |y_1 - y_2|^m)^{1/m}$. Euclidean distance, therefore, is L_2-distance. Modify the closest-pair algorithm to use the L_1-distance, which is also known as the ***Manhattan distance***.

In the analysis of the algorithm, most of it goes through just based on the triangle inequality. The only main point of difference is in looking at the number of points that can be fit into a $\delta \times 2\delta$ rectangle. In particular, we can cram in two more points than the eight shown into the rectangle by placing points at the centers of the two squares that the rectangle breaks into. This means that we need to consider points up to 9 away in Y' instead of 7 away. This has no impact on the asymptotics of the algorithm and it is the only correction to the algorithm that is needed if we switch from L_2 to L_1.

33.4-4

> Given two points p_1 and p_2 in the plane, the L_∞-distance between them is given by $\max(|x_1 - x_2|, |y_1 - y_2|)$. Modify the closest-pair algorithm to use the L_∞-distance.

We can simply run the divide and conquer algorithm described in the section, modifying the brute force search for $|P| \leq 3$ and the check against the next 7 points in Y' to use the L_∞ distance. Since the L_∞ distance between two points is always less than the euclidean distance, there can be at most 8 points in the $\delta \times 2\delta$ rectangle which we need to examine in order to determine whether the closest pair is in that box. Thus, the modified algorithm is still correct and has the same runtime.

33.4-5

> Suppose that $\Omega(n)$ of the points given to the closest-pair algorithm are covertical. Show how to determine the sets P_L and P_R and how to determine whether each point of Y is in P_L or P_R so that the running time for the closest-pair algorithm remains $O(n \lg n)$.

We select the line l so that it is roughly equal, and then, we won't run into any issue if we just pick an arbitrary subset of the vertices that are on the line to go to one side or the other.

Since the analysis of the algorithm allowed for both elements from P_L and P_R to be on the line, we still have correctness if we do this. To determine what values of Y belong to which of the set can be made easier if we select our set going to P_L to be the lowest however many points are needed, and the P_R to be the higher

points. Then, just knowing the index of Y that we are looking at, we know whether that point belonged to P_L or to P_R.

33.4-6

> Suggest a change to the closest-pair algorithm that avoids presorting the Y array but leaves the running time as $O(n \lg n)$. (Hint: Merge sorted arrays Y_L and Y_R to form the sorted array Y.)

In addition to returning the distance of the closest pair, the modify the algorithm to also return the points passed to it, sorted by y-coordinate, as Y. To do this, merge Y_L and Y_R returned by each of its recursive calls. If we are at the base case, when $n \leq 3$, simply use insertion sort to sort the elements by y-coordinate directly. Since each merge takes linear time, this doesn't affect the recursive equation for the runtime.

Problem 33-1 Convex layers

> Given a set Q of points in the plane, we define the ***convex layers*** of Q inductively. The first convex layer of Q consists of those points in Q that are vertices of $CH(Q)$. For $i > 1$, define Q_i to consist of the points of Q with all points in convex layers $i, 2, \ldots, i-1$ removed. Then, the ith convex layer of Q is $CH(Q_i)$ if $Q_i \neq \emptyset$ and is undefined otherwise.
>
> **a.** Give an $O(n^2)$- time algorithm to find the convex layers of a set of n points.
>
> **b.** Prove that $\Omega(n \lg n)$ time is required to compute the convex layers of a set of n points with any model of computation that requires $\Omega(n \lg n)$ time to sort n real numbers.

(Omit!)

Problem 33-2 Maximal layers

> Let Q be a set of n points in the plane. We say that point (x, y) ***dominates*** point (x', y') if $x \geq x'$ and $y \geq y'$. A point in Q that is dominated by no other points in Q is said to be ***maximal***. Note that Q may contain many maximal points, which can be organized into ***maximal layers*** as follows. The first maximal layer L_1 is the set of maximal points of Q. For $i > 1$, the ith maximal layer L_i is the set of maximal points in $Q - \cup_{j=1}^{i-1} L_j$.
>
> Suppose that Q has k nonempty maximal layers, and let y_i be the y-coordinate of the leftmost point in L_i for $i = 1, 2, \ldots, k$. For now, assume that no two points in Q have the same x- or y-coordinate.
>
> **a.** Show that $y_1 > y_2 > \cdots > y_k$.
>
> Consider a point (x, y) that is to the left of any point in Q and for which y is distinct from the y-coordinate of any point in Q. Let $Q' = Q \cup \{(w, y)\}$.
>
> **b.** Let j be the minimum index such that $y_j < y$, unless $y < y_k$, in which case we let $j = k + 1$. Show that the maximal layers of Q' are as follows:

- If $j \le k$, then the maximal layers of Q' are the same as the maximal layers of Q, except that L_j also includes (x, y) as its new leftmost point.
- If $j = k + 1$, then the first k maximal layers of Q' are the same as for Q, but in addition, Q' has a nonempty $(k + 1)$st maximal layer: $L_{k+1} = \{(x, y)\}$.

c. Describe an $O(n \lg n)$-time algorithm to compute the maximal layers of a set Q of n points. (Hint: Move a sweep line from right to left.)

d. Do any difficulties arise if we now allow input points to have the same x- or y-coordinate? Suggest a way to resolve such problems.

(Omit!)

Problem 33-3 Ghostbusters and ghosts

A group of n Ghostbusters is battling n ghosts. Each Ghostbuster carries a proton pack, which shoots a stream at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The Ghostbusters decide upon the following strategy. They will pair off with the ghosts, forming n Ghostbuster-ghost pairs, and then simultaneously each Ghostbuster will shoot a stream at his chosen ghost. As we all know, it is very dangerous to let streams cross, and so the Ghostbusters must choose pairings for which no streams will cross.

Assume that the position of each Ghostbuster and each ghost is a fixed point in the plane and that no three positions are colinear.

a. Argue that there exists a line passing through one Ghostbuster and one ghost such that the number of Ghostbusters on one side of the line equals the number of ghosts on the same side. Describe how to find such a line in $O(n \lg n)$ time.

b. Give an $O(n^2 \lg n)$-time algorithm to pair Ghostbusters with ghosts in such a way that no streams cross.

(Omit!)

Problem 33-4 Picking up sticks

Professor Charon has a set of n sticks, which are piled up in some configuration. Each stick is specified by its endpoints, and each endpoint is an ordered triple giving its (x, y, z) coordinates. No stick is vertical. He wishes to pick up all the sticks, one at a time, subject to the condition that he may pick up a stick only if there is no other stick on top of it.

a. Give a procedure that takes two sticks a and b and reports whether a is above, below, or unrelated to b.

> **b.** Describe an efficient algorithm that determines whether it is possible to pick up all the sticks, and if so, provides a legal order in which to pick them up.

(Omit!)

Problem 33-5 Sparse-hulled distributions

> Consider the problem of computing the convex hull of a set of points in the plane that have been drawn according to some known random distribution. Sometimes, the number of points, or size, of the convex hull of n points drawn from such a distribution has expectation $O(n^{1-\epsilon})$ for some constant $\epsilon > 0$. We call such a distribution ***sparse-hulled***. Sparse-hulled distributions include the following:
>
> - Points drawn uniformly from a unit-radius disk. The convex hull has expected size $\Theta(n^{1/3})$.
> - Points drawn uniformly from the interior of a convex polygon with k sides, for any constant k. The convex hull has expected size $\Theta(\lg n)$.
> - Points drawn according to a two-dimensional normal distribution. The convex p hull has expected size $\Theta(\sqrt{\lg n})$.
>
> **a.** Given two convex polygons with n_1 and n_2 vertices respectively, show how to compute the convex hull of all $n_1 + n_2$ points in $O(n_1 + n_2)$ time. (The polygons may overlap.)
>
> **b.** Show how to compute the convex hull of a set of n points drawn independently according to a sparse-hulled distribution in $O(n)$ average-case time. ($Hint$: Recursively find the convex hulls of the first $n/2$ points and the second $n/2$ points, and then combine the results.)

(Omit!)