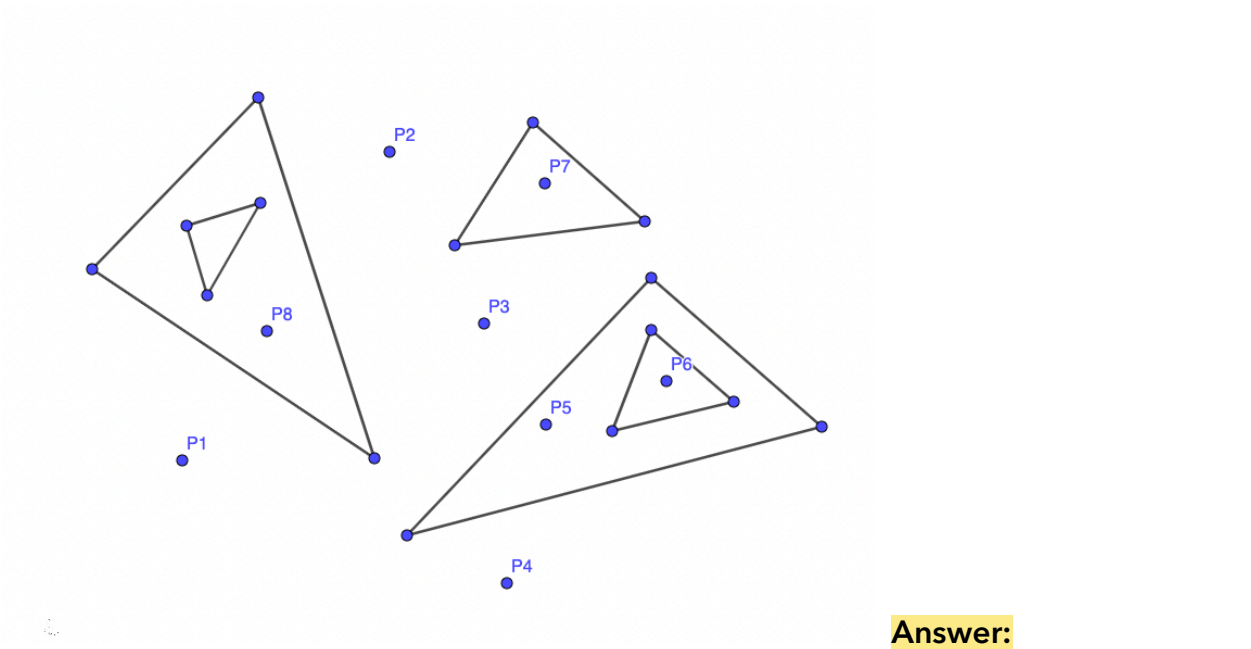
Assignment 4

Computational Geometry CS60064

Pratyush Jaiswal | 18EE35014
Ram Niwas Sharma | 18CS10044

Question 1

Let S be a set of n triangles in the plane as shown in the figure below. The boundaries of the triangles are disjoint, but one may completely enclose another. Let P be a set of n points in the plane. Outline an $O(n \log(n))$ algorithm that reports all points in P that lie outside all triangles in S.



Answer:

Algorithm

1. Store all the points in the plane, including the vertices of S and the points in P, with a flag that indicates whether a point is in P or which triangle the vertex belongs to. Sort these points according to their $x - coordinate$.

2. Moving along the sweeping plane we can get four types of events, *Neutral* : A point that lies in P , *Start Triangle* : leftmost vertex of triangle, *Mid Triangle* : middle vertex of the triangle or *End triangle* : rightmost vertex of triangle

3. Now, between consecutive events, if we have passed the start of the triangle but not the end, we can characterise it by two lines, one for top and one for bottom edge. If the current point lies betwwen these two lines then it lies inside the triangle.

4. During the sweep, we will keep the active lines in an AVL Tree sorted by their $y - coordinate$ of structure, allowing *Search*, *Insert* and *Delete* in $O\log(n)$ time. The ordering is defined since the triangles do not intersect.

5. If a triangle vertex is found to be inside another triangle, the lines belonging to the inside triangle has to be discarded.

6. Handing each events :

   - *Neutral Point* : Just find the position of the point with respect to the currently active line. We find the pair of lines between which the point lies and then check if they belong to the same triangle, if not we have to report the point.
   - *Start Triangle* : Insert two new lines in the AVL Tree, which are the two leftmost edges of the triangle.
   - *Middle Triangle* : Activate the edge between the middle and rightmost vertex instead of the edge between the leftmost and middle vertex. In total, one deletion and one insertion operation are required.
   - *End Triangle*Remove the two rightmost edges of the triangle from the AVL Tree.

Time Complexity Analysis

1. Sorting all the points, (total n) takes $O(n\log(n)$

2. Checking the position of a point wrt a line can be done in $O(1)$ time, and finding the two closest lines by *Lower bound/ Upper bound* operation can be done in $O(\log(n)$ time. For n points it takes $O(n\log(n))$ time.

3. Inserting, searching and deleting an triangle in the AVL Tree takes $O(\log(n))$ time, so for atmost n triangles it takes $O(n\log(n))$ time.

4. Sorting all the points (total $2n$ points) and making a event queue takes $O(n\log n)$ time.

As a result, the entire method has a time complexity of $O(nlogn)$.

Question 2

Let $A(L)$ denote a simple arrangement of a set L of n lines. Describe an $O(n \log n)$-time algorithm for constructing the convex hull of all intersection points.

Answer:

Consider the set S of $n(n-1)/2$ points determined by the pairwise intersections of n lines in the plane.

Even though we are computing a property of $O(n^2)$ points, these points have a ·compact· description, namely the set of n straight lines which define them. In this note, we show that the Convex hull of the set S can be computed in time $O(n)$.

We now introduce some conventions and terminology which will be used further to prove. If W is any set of points, then we use $CH(W)$ to denote the convex hull of W. The storage description we use for $CH(W)$ is a list of the points of W that are on the convex hull, in counterclockwise cyclic order. A point $p \in W$ is a corner if it belongs to $CH(W)$ and it does not belong to the straight-lice segment which joins the point preceding it on $CH(W)$ to the one succeeding it on $CH(W)$. The result of deleting all the non–corner points from $CH(W)$ is called the edge hull of W and is denoted by $ECH(W)$. If all the points of W are in general position (i.e. if no three of them are aligned) then every point of the convex hull of W is a corner, and therefore in that case $ECH(W) = CH(W)$. However, the points of the set S under consideration are certainly not in general position, and therefore $ECH(S)$ and $CH(S)$ differ.

Computing $ECH(S)$

The input to the algorithm described in this ·section is the n lines whose pairwise intersections define the set S. The output of the algorithm is the edge-hull of S, $ECH(S)$. Since the algorithm is supposed to run in time $O(n)$ it is obvious that we cannot afford to explicitly generate the set S. Instead, the algorithm generates a set Q of n points which have the property that $ECH(Q) = ECH(S)$. Once we have such a set Q, $ECH(Q)$ can be computed in time $O(n \log n)$ by using any of the known convex hull algorithms.

Algorithm Edge-Hull

1. Sort the n input straight lines by decreasing slope. Let $L_0, .., L_{n-1}$ be the n lines listed by decreasing slope, i.e. if the equation of L_i is $y = a_i x + b_i$, then we have $a_0 > a_1 > ... > a_{n-1}$· This Step takes $O(n)$ time.
2. Let point q_i denote tbe intersection of lines L_i and $L_{i+1 \ mod \ n}$. Find the set $Q = q_o ...., q_{n-1}$· This takes $O(n)$ time.
3. Step 3. Compute $ECH(Q)$ using any of the known $O(n \log n)$ time convex hull algorithms (e.g. [G]). Then output $ECH(Q)$.

It is obvious that the above algorithm runs in $O(n \log n)$ time.

Correctness of the algorithm would immediately follow if we could prove that $ECH(Q) = ECH(S)$.

Proof $ECH(Q) = ECH(S)$

To prove this, it suffices to show that if p is a corner point of S then $p \in Q$. So let p be a corner point of S. Let L_i and L_j, $j > i$, be the two lines whose intersection in p. To prove that $p \in Q$, we must show that either $j = i + l$ (i.e.p=q,), or $j = n - 1$ and $i = O$ (i.e. $p = q_{n-1}$). We prove this by contradiction: Suppose to the contrary that $p \neq q_i, q_j$. Since $p \neq q_i$, there is at least one line L_k whose slope is between the slopes of L_j and L_i i.e. $a_i > a_k > a_j$. Let v and w be the points of intersection of L_k with L_i and $_j$ respectively.

We continue the proof assuming v is to the left of w(the argument when w is to the left of v is entirely symmetrical). Since p_{n-1} one (or both) of the following is true:

(i) $j \neq n - 1$, or

(ii) $i \neq 0$.

If (i) holds then we obtain a contradiction as follows. Line L_{n-1} must cross line L_i, somewhere, say at point s. If s is to the right of p then p is on the straight-line segment joining s to v. which contradicts the fact that p is a corner point. If on the other band, s is to the left of p. then L_{n-1} intersects L_j at a point t which is to the left of p. This implies that p is on the straight-line segment joining t to w which contradicts the fact that p is a corner point. Therefore (i) cannot occur.

The argument·for finding a contradiction when (ii) holds is similar to the above argument for (i), except that L_0 plays the role of L_{n-1}. Since either (i) or (ii) leads to a contradiction, it follows that our original assumption (that p ?Q) was wrong.

This completes the proof that every corner point belongs to Q, from which the correctness of the algorithm follows.

Computing $CH(S)$

The algorithm for computing $CH(S)$ works as follows. We compute $ECH(S)$ in time $O(n \log n)$, using the algorithm of the previous section. Then we mark every edge of $ECH(S)$ which is along with one of the input lines as being specia1. This takes $O(n)$ time. Then for every line L_i, we do the following. First, we find the intersection of L_j with the convex polygon $ECH(S)$. This takes $O()$ time by using the Chazelle-Dobkin algorithm for intersecting a line with a convex polygon. Let s_j and t_i, be the points of intersection of L_i with $ECH(S)$: If s_i (resp. t_i) is a corner, or the edge of $ECH(S)$ to which $s + i$ (resp. t_i) belongs is special, then add s_i, (resp. t_i) to a set H (H is initially empty). Since this is done for every L_i, the total cost of creating H is $O(n \log n)$ time, and $|H| < 2n$. The set H now contains all the points of S that appear on $CH(S)$, whether they are comers or not, and therefore $CH(H) = CH(S)$. Computing $CH(H) = CH(S)$ can now be done in time $O(n \log n)$ by using any of the known $O(n \log n)$ time convex-hull algorithms.

So the whole algorithm for computing the convex hull of the $n(n-1)/2$ points determined by be pairwise intersections of n input lines is having $O(n \log n)$ time complexity.

Question 3

Prove that the test for collinearity of three points in a set of n points on the plane, is as hard as checking whether there exist three distinct integers a, b, c among a set of n integers, such that $(a+b+c)=0$.

Answer:

Before jumping to this problem, let us recall some of the terms/definitions:

3SUM

Given n integers, we want to know if any 3 sum to 0. This problem is called **3SUM**. We can solve this in $O(n^3)$ by trying all triplets. We can also solve this in $O(n^2)$ by first computing all the pairwise sums, storing all those sums in a hash map and then check for all integers if there exists a pair with sum negation of the integer.

But we can do better, there exists an $O(n^2)$ deterministic algorithm. We can sort the array and then start two pointers, one from starting (say i) and other from ending (say j). And for every $i<k<j$, check the sum of the integers at i, j and k. If the sum is smaller than 0, move i to the right, if the sum is greater than zero, move j to the left and if the sum is zero, we have found our solution.

It is conjectured that no $O(n^{2-\epsilon})$ algorithm exists for this problem, for any $\epsilon>0$. This is called **3SUM-Conjecture**.

3SUM-Hardness

Definition

From the above discussion of **3SUM**, it leads to the notion of **3SUM-Hardness**. We call a problem **3SUM-hard**, if there exists an $O(n^{2-\epsilon})$ algorithm that can solve the problem, then there exists one for **3SUM** as well.

3SUM Reductions

This leads us to the notion of **3SUM-Reductions**. We know that have if A reduces to B, we can solve A using B. So, suppose we reduce an instance x in A to x' in B. If $n=|x|$, and $n'=|x'|$, then if we were to have a **3SUM reduction**, it is required that we make a $O(1)$-call reduction on $n'=O(n)$, and that our reduction is $O(n^{2-\epsilon})$ for some $\epsilon>0$.

So, if there is a *3SUM reduction* from A to B, then we have if A is **3SUM-hard** (such as *3SUM*), then B is also **3SUM-hard**.

Now using the above discussions, we can come to our problem. So from above ***3SUM Reductions*** concept, if we make a reduction from ***3SUM*** to this original problem, it is proved that both have same hardness that is ***3SUM-hard***.

As displayed in the below figure, we map from x to (x,x^3) for each x in the 3SUM problem, where we assume the three chosen numbers are distinct. We can show that three points on the curve will lie on a line if and only if there are three integers whose sum is

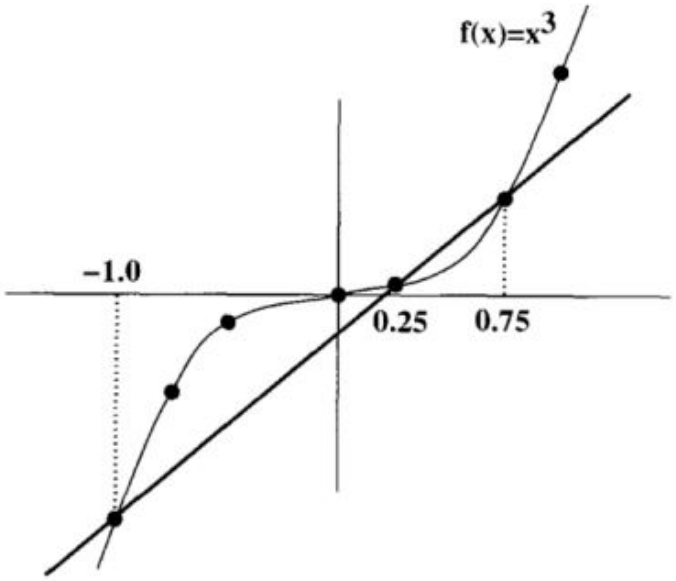


Figure 1: Reduction from 3SUM to Collinearity

zero in the original set.

Also we can notice from the slopes taking two points at a time

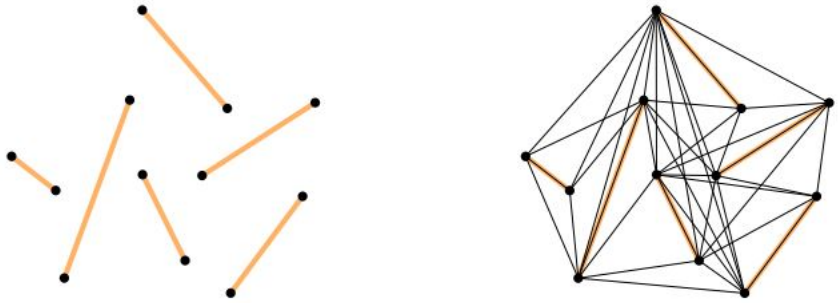
$$\frac{b^3 - a^3}{b - a} = \frac{c^3 - a^3}{c - a}$$
$$b^2 + ab + a^2 = c^2 + ac + a^2$$
$$(b - c)(a + b + c) = 0$$
$$a + b + c = 0 \qquad \text{\{Since all integers are distinct\}}$$

So it can be clearly seen that the reduction is possible from *3SUM* to *Collinearity* problem. So from above ***3SUM Reductions*** concept, it is proved that both have same hardness that is ***3SUM-hard***.

Question 4

Consider the visibility graph $G(V, E)$ of a set of n disjoint line-segments on the plane. Assume for simplicity that no three end-points of line-segments are collinear, and no line segment is horizontal or vertical. Each end-point defines a vertex in V. An edge (v_1, v_2) appears in E when the two end-points corresponding to v_1 and v_2 either belong

to the same segment, or are visible on the plane. An example of visibility graph for a set of six line-segments is shown below.



4(a)

Show that G admits a cycle through all vertices.

Answer:

We know that it is not always possible to form (the boundary of) a simple polygon from a given set of disjoint line segments. We can take three parallel chords of a convex polygon shown below to prove the above statement. However, if we do not insist that

Figure 2: Sets of disjoint line segments that do not allow certain polygons.

the segments appear on the boundary (which is given in the question about all line segments), but allow them to be diagonals or epigonals, then it is always possible. In other words, the segment endpoint visibility graph of disjoint line segments is **Hamiltonian**, unless all segments are collinear. And we know that the Hamiltonian Graph in a graph is a cycle which contains every vertex of G.

So it proved that G admits a cycle through all vertices.

Reference: http://www.uop.edu.pk/ocontents/Hamiltonian%20Graphs.pdf