

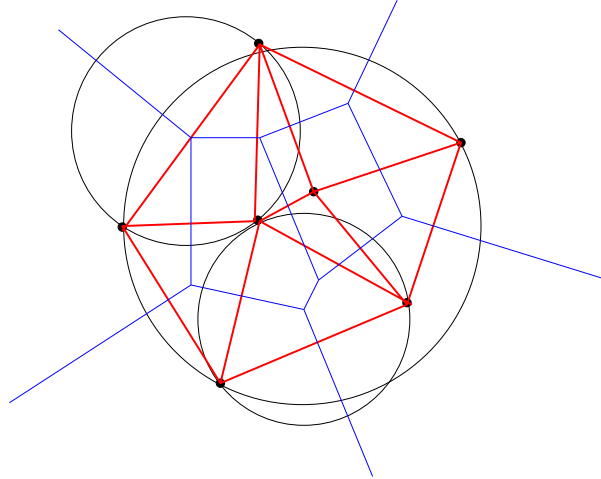
COMPUTATIONAL GEOMETRY Practice Final – Solution Notes

Statistics: $n = 36$, $\mu = 67.6$, median 67, $\sigma = 12.8$; score range: 42–93

(1). [12 points] For the set S of 7 points shown below, do the following:

(a). Draw the (Euclidean) Delaunay diagram. In order to assist you in making some decisions (in case you do not have a compass with you), I have drawn a few circles.

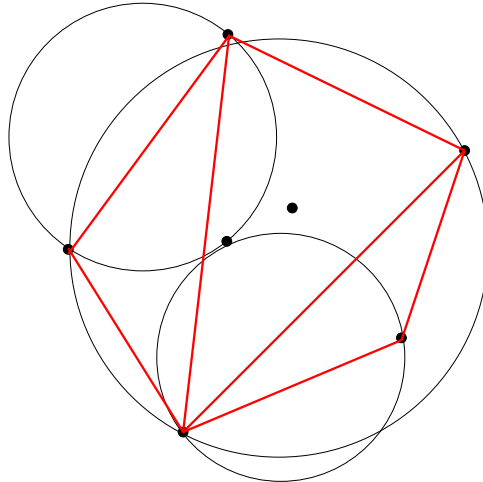
I draw the Delaunay diagram in red below; each pair of joined sites satisfies the empty circle property. In blue, I show the dual, Voronoi diagram.



(b). Sketch also the Voronoi diagram.

(c). Draw the furthest-site Delaunay diagram (dual to the furthest-site Voronoi diagram)

We show the diagram below, in red. We join any pair of sites that satisfies the “full circle” property (that there is a circle passing through that pair of sites, with all others strictly interior to the circle). Note that the only sites that participate in the diagram are those on the convex hull. (Do you see why?) This diagram is the upper convex hull of the lifted images of the points on the paraboloid of revolution.



(2). [30 points] For each of the computations below indicate how efficiently one can perform the calculation, in terms of $O(\dots)$ notation (e.g., $O(n)$, $O(\log n)$, $O(n^2)$, $O(n \log n)$, $O(k \log n)$, etc). Try to give the best (lowest) upper bound possible.

(a). Given an arbitrary set of n triangles in the plane (in general position), report all k points of intersection between pairs of boundary segments among the triangles.

There are $3n$ line segments that are edges of the n triangles. Using an optimal-time segment intersection algorithm (e.g., Chazelle-Edelsbrunner or Balaban) we can report all k points of intersection in time $O(k + n \log n)$ (using $O(n)$ space, using Balaban). (Note that the Bentley-Ottmann sweep gives a suboptimal time of $O((k + n) \log n)$.)

(b). Compute the Euclidean minimum spanning tree of n points in the plane.

We compute in time $O(n \log n)$ the Delaunay diagram of the n points (e.g., using Fortune’s sweep algorithm). Then, we know that the MST is a subgraph of the Delaunay (which has only $O(n)$ edges). Using Kruskal’s algorithm on the Delaunay, then, takes time $O(n \log n)$. So the total time is $O(n \log n)$, and this is best possible in the worst case (since one can use MST to sort n points along a line).

(c). Given a set S of n points in the plane, determine if the convex hull, $CH(S)$, is a triangle.

We simply do 3 iterations of gift wrapping at $O(n)$ per iteration, for a total time of $O(n)$.

(d). Given a set S of n points in the plane, determine if one of them (say, $p_1 \in S$) is a vertex of the convex hull, $CH(S)$.

This takes time $O(n)$: One way to see this is to realize that p_1 is a vertex of $CH(S)$ if and only if there exists a line, ℓ , with p_1 on one side of ℓ and all $n - 1$ other points on the other side of ℓ . Thus, the problem is a linear programming feasibility question, in 2 variables. By Megiddo or Dyer algorithms, LP in fixed dimension can be solved in worst-case linear time.

An alternative direct method is to solve it as a 1-dimensional LP, realizing that we can assume that ℓ passes through p_1 , so we only need to find the slope of ℓ . This amounts to an incremental $O(n)$ time algorithm to keep track of a convex wedge, with apex at p_1 , that contains all other points. With each new point p_i , we check (in time $O(1)$) if it lies within the wedge; if it does, we do nothing; if it lies within the negative of the wedge, then we have found a witness triangle containing p_1 , proving it is not a vertex of $CH(S)$; otherwise, we can enlarge the wedge to include p_i , and continue.

(e). Given a set \mathcal{L} of n lines, build a data structure to support efficient queries of the form: How many sides does the cell containing query point q have in the arrangement induced by \mathcal{L} ? State the preprocessing time, storage space, and query time.

(i). Preprocessing time is $O(n^2)$: Build the arrangement $A(\mathcal{L})$, in time $O(n^2)$ (using the Zone Theorem). Label each cell with how many sides it has. (This can be done during the incremental construction, with each insertion of a line in time $O(n)$; or, we can do a breadth-first search later, in the DCEL, in time proportional to the size of the arrangement ($O(n^2)$)). Preprocess $A(\mathcal{L})$ for point location queries: since each cell is a convex polygon (we can triangulate each of them, if we want a full triangulation), we can apply an optimal algorithm, such as Kirkpatrick’s, to preprocess in time linear in the size ($O(n^2)$) of the subdivision.

(ii). Storage space (memory usage) is $O(n^2)$: We store the entire arrangement in a DCEL and in a point location hierarchy, in space proportional to the size of the subdivision, $O(n^2)$.

(iii). Query time is $O(\log n)$: The query consists of doing a point location, and then reporting the cardinality stored with the cell in which the query point is located.

(f). Given a list of points, (p_1, p_2, \dots, p_n) , in the plane, defining a polygonal chain (with edges $p_i p_{i+1}$), determine if the chain crosses itself at any point.

We run Chazelle’s triangulation algorithm, in time $O(n)$, which does simplicity testing too (it exits with a witness pair of crossing segments if the input chain/polygon crosses itself).

(3). [10 points] We want to solve the following query problem: Given a set S of n disjoint line segments in the plane, determine the first segment that is stabbed by a vertical ray running from a point $q = (q_x, q_y)$ vertically upwards to infinity. Describe briefly a data structure and a method for this problem. Try to be as efficient as possible in both space and query time.

(i). Preprocessing time is $O(n \log n)$: Sweep with a vertical line, building a vertical trapezoidalization of the segments. (Events occur at segment endpoints; sweep line status maintains the segments intersected by the sweep line, sorted by y .) Preprocess the resulting trapezoidalization for point location queries (e.g., using Kirkpatrick), in time $O(n)$.

(ii). Storage space (memory usage) is $O(n)$: The vertical trapezoidalization has size $O(n)$, as does the point location data structure.

(iii). Query time is $O(\log n)$: Point location. Once we know the trapezoid containing the query point, we know the segment that forms the “top” of the trapezoid (we store this information when we build the DCEL).

(4). [9 points] Suppose I claim to have an algorithm, ALG , which computes the convex hull of n points in $4D$, and does so in the best possible worst-case time (in terms of n). How fast is ALG (in terms of n)? Give your answer in terms of big-Oh notation.

The convex hull in $4D$ can be found in time $O(n^2)$; more generally, in dimension 2 or 3, it takes time $O(n \log n)$; in dimension $d \geq 4$, convex hull takes worst-case time $O(n^{\lfloor d/2 \rfloor})$, which is worst-case optimal.

Explain briefly how you could use ALG to compute the Delaunay triangulation (“tetrahedralization”) of n points in $3D$.

Use the lifting map to take each point (x_i, y_i, z_i) to point $(x_i, y_i, z_i, x_i^2 + y_i^2 + z_i^2) \in \mathbb{R}^4$, and find the convex hull of the lifted points. The lower convex hull (consisting of facets whose outer normal has a negative inner product with $(0, 0, 0, -1)$) yields the Delaunay diagram of the points in $3D$.

(5). [12 points] We want to solve the following query problem: Given a set S of n disjoint line segments in the plane, determine how many segments are stabbed by a query line ℓ .

(a). Describe briefly a data structure and a method for this problem. Try to be as efficient as possible, especially in query time.

(i). Preprocessing time is $O(n^2)$: Dualize the n segments (each becomes a “double wedge”) and construct the arrangement of the double wedges, in time $O(n^2)$. Label each face of the arrangement with the number of double wedges present at points in that face. (This too takes time $O(n^2)$, since we can do BFS in the arrangement, or maintain these cardinalities as we incrementally construct the arrangement.) Preprocess the arrangement for point location queries (again, in time $O(n^2)$).

(ii). Storage space (memory usage) is $O(n^2)$: We store the arrangement of the n double wedges ($2n$ lines).

(iii). Query time is $O(\log n)$: We do point location query, and then report the cardinality stored with the face containing the query point.

(b). Answer the question now assuming that the query line is known to be horizontal or vertical.

It suffices to assume that the query line is vertical; we separately do it for horizontal query lines.

Thus, the problem becomes a 1D problem: Just project the segments to the x -axis. Sort the endpoints (time $O(n \log n)$), and walk through the sorted intervals, labeling each with the cardinality of how many segments cover the interval.

(i). Preprocessing time is $O(n \log n)$

(ii). Storage space (memory usage) is $O(n)$

(iii). Query time is $O(\log n)$

(6). [9 points] Let $S = \{p_1, \dots, p_n\}$ be a set of n points in the plane. Describe briefly how you would preprocess S into a data structure that will support very efficient queries of the form: For query point q , report all points of S that are within L_1 distance d of q . (Recall that the L_1 distance between p and q is given by $d_1(p, q) = |p_x - q_x| + |p_y - q_y|$.)

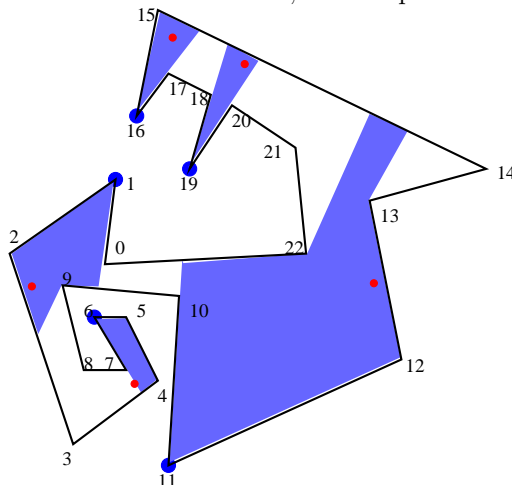
An L_1 disk of radius d is simply a square, of side length $d\sqrt{2}$, at angle 45 degrees with respect to the x -axis. Thus, we simply preprocess the n points for orthogonal range queries (in the rotated coordinate frame), in time $O(n \log n)$, space $O(n \log n)$, which allows queries in time $O(\log n + k)$, where k is the number of points within L_1 distance d of the query point.

(7). [9 points] Let $L = \{\ell_1, \dots, \ell_n\}$ be a set of n lines in the plane. Describe briefly how you would preprocess L into a data structure that will support queries of the following form: For query line ℓ , report the points where ℓ crosses the lines ℓ_i , and give these points in order of increasing x -coordinate.

We build the arrangement of L , in time $O(n^2)$; store it in a DCEL ($O(n^2)$ storage). Then, for a query line, ℓ , we can walk through the DCEL to determine the zone of ℓ , walking around each cell we cross, identifying the points where ℓ crosses the lines, in order by x . Thus, a query takes time $O(n)$.

(8). [9 points] (a). By inspection, obtain the point guard number for P , allowing guards to be placed at any point (interior or boundary) of the polygon. Justify your answer! (Give an argument that fewer guards cannot suffice.)

By inspection, we obtain 5 guards that suffice to see P . In order to prove that at least five guards are needed, we can place witness points at vertices 1, 6, 11, 16, and 19; the 5 visibility polygons from these points are pairwise-disjoint (no overlaps), so at least one guard must be placed in each of these 5 visibility polygons, proving that $g(P) \geq 5$. In the figure below, guard points are shown in small red circles, witness points are shown in large blue circles.



(b). *How many (vertex) guards does the Art Gallery Theorem guarantee are sufficient for this polygon?*

The Art Gallery Theorem guarantees that $\lfloor n/3 \rfloor$ guards are sufficient. Here, $n = 23$, so we know that 7 guards are sufficient (though, they are not necessary in this case).