CS310 Computational Geometry
Spring 2019

Assignment:	**3**	**Professor:**	**Dianna Xu**
Due Date:	**2/26/19**	**E-mail:**	`dxu@cs.brynmawr.edu`
Office:	**Park 203**	**URL:**	`http://cs.brynmawr.edu/cs310`

Assignment 3

Tips about writing algorithms: Whenever you are asked to present an algorithm, you should present three things: the algorithm description (English and pseudocode), an analysis of its running time and an (informal) proof of its correctness. Remember that your description is intended to be read by a human, not a compiler, so conciseness and clarity are preferred over coding details. Unless otherwise stated, you may use any results from class, or results from any standard algorithms text. Nonetheless, be sufficiently complete so that a competent coder has all the details to implement your algorithm. 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.
Please consult this additional `algorithm write-up guidelines` from Analysis of Algorithm. The requirements for algorithm write-up is more rigid in that course (well it's the point of it). In this class, I do not require formal proof of correctness. However, I do require informal justifications because if you are unable to even informally argue correctness, it's likely wrong!

1. Prove that the intersection of two convex sets is again convex.

2. Exercises 2.19 and 2.20

3. Exercise 2.22

4. Exercise 2.38 and 2.43

5. The convex hull is a somewhat non-robust shape descriptor, since if there are any distant outlying points, they will tend to dominate the shape of the hull. A more robust method is based on the following iterative approach. Given a planar point set P in general position (see Fig. 1(a)), let H_1 be the convex hull of P. Remove the vertices of H_1 from P and compute the convex hull of the remaining points, call it H_2. Repeat this until no more points remain, letting $H_1, \ldots, H_k$ denote the resulting hulls (see Fig. 1(b)). More formally, $H_i = conv(P \setminus (\bigcup_{j=1}^{i-1} vert(H_j)))$. The final result is a collection of nested convex polygons, where the last one may degenerate to a single line segment or a single point.

   (a) Assuming that the points are in general position in $\mathbb{R}^2$, as a function of n, what is the maximum number of hulls that can be generated by this process? (I am looking for an exact formula, not an asymptotic one. For every n, there should exist a point set that exactly achieves your bound.) Briefly justify your answer.

   (b) Given a set P of n points in the plane, devise an $O(n^2)$ time algorithm to compute this iterated sequence of hulls. (FYI: $O(nlogn)$ is possible, but complicated.)

   (c) Prove the following lemma: Given a planar point set P in general position, let k denote the number of hulls generated by the repeated hull process. There exists a point q in the plane (it need not be in P) such that, every closed halfplane whose bounding line l passing through q contains at least k points of P. (Recall that a closed halfplane is the set of points lying on or to one side of a line. An example of such a point for $k = 4$ is shown in Fig. 1(c).)

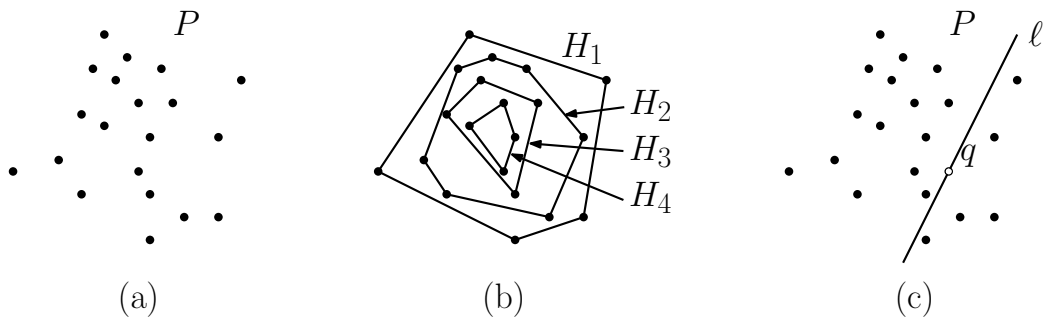

Figure 1: Nested hulls

6. Implementation Implement the naive convex hull algorithm by finding hull edges as described below.

Input xy-coordinates of n points in the plane. Two points a and b end up on the hull if and only if the directed line L_{ab} through a and b has every other point on or to the left of L_{ab}. Running this test over all pairs of points a and b will result in the set of points that lie on the hull, with lots of duplicates. Remove the duplicates and output (print out) the set, in ccw order. What is the complexity of this algorithm? Time your code on an input set with $1,000$, $10,000$ and $1,000,000$ points. How long did it take?

Implement the Graham scan convex hull algorithm and time the same input sets. What is the time difference? Does it justify the runtime analysis?

You may use any programming language of your choice. Submit a printed copy of your code, together with at least three test cases and printouts of sample runs on those test cases, together with some graphical representation of the points. You can either use additional plotting software after the fact, or use a programming language that supports easy graphical output (Procesing would be a fine example). The test cases should be chosen to clearly demonstrate that your code works. That is, do not include trivial cases and do include corner cases. For implementation, you may only assume that the input points do not contain duplicates. In other words, you need to take care of collinearity. To generate an input set, you can either use graph paper by hand for small sizes, or use a randomized generator (remember that straight-forward randomization tends to produce very uniformly distributed point sets, which may not be what you want). To produce worse-case test cases for convex hull algorithms, you need to find a way to generate points on or near a circle.

Also beware of numerical (in)stability, which is a common problem in computational geometry algorithm implementations.

Please do not include large test cases among the three that demo the correctness of your code. For that purpose, your input size probably doesn't need to be larger than 20.