15-456/852 Computational Geometry, Fall 2017

Homework 1 (100 pts) **Due: Oct 6 by class time**

Gary Miller TA: Timothy Chu

Question	Points	Score
1	20	
2	25	
3	25	
4	25	
5	20	
6	30	
Total:	145	

(20) 1. **Projections of 3D Polygons into Two Dimensions**

Define the projection of a set S of points into xy plane, as the projection of each point into the xy plane. In particular, we can now talk about the projection of polytopes.

This problem can be thought of as the following: we have a polytope defined as the intersection of a bunch of half-spaces in 3D. It has a well defined projection onto the xy plane. This problem concerns finding what that projection is. The things we'd like to understand are: is the projection a polytope, and can we find a set of linear constraints for it?

1. Projecting 3D Polygons to 3D
   (a) (Writing polytopes in a way that makes projections easier) Show that you can write such a polytope in the form:

$$A^+[x, y, z] \leq b^+$$
$$A^-[x, y, z] \leq b^-$$
$$A^0[x, y, z] \leq b^0$$

   where the constraints A^+ all have a z coefficient of 1, the constraints A^- all have a z coefficient of -1, and the constraints A^0 all have a z coefficient of 0.

   (b) Consider the set of constraints defined by $A'[x, y] \leq b'$, where A' consists of all constraints generated by taking a constraint from A^+ and taking a constraint from A^-, and adding them. Then throw in all constraints in A^0. Show that it takes quadratic time in n to generate A'.

(c) Is A' always the projection of P onto the xy plane, for all A and b? Why or why not? If not, can you tweak this algorithm so that it generates the projection of P onto the xy plane in quadratic time? Is the projection always a polytope? (You can try playing around with this for some example polytopes).

2. Assume you have access to an oracle that solves 2-dimensional linear programs in expected linear time in the number of constraints. Use part (a) to find an expected quadratic time algorithm to solve a 3-dimensional linear program. Do not use the 3D linear-time algorithm mentioned in class.

3. * Use the ideas of projecting the feasible polytope to prove LP Duality. In particular, LP duality can be seen as projecting a polytope down to a 1-dimensional space containing c (for linear program $\{\max c^T x | Ax \leq b\}$). Projecting a polytope to 1-dimensional space can be done by projecting it successively into spaces of lower and lower dimension.

(25) 2. **Simple Paths and Convex Hull**

Suppose that $P = \{p_1, \ldots, p_n\}$ is a set of points in the plane. We say the the sequences of distinct points $Path = (p_1, \ldots, p_k)$ is a simple path if the line segments $l_i = [p_i p_{i+1}]$ are disjoint except for $l_i \cap l_{i+1} = p_{i+1}$. We may also allow $p_1 = p_k$ and in this case $l_{k-1} \cap l_1 = p_k$.

In the following questions we shall investigate the relation between finding a simple path of a set of points and finding their convex hull.

1. Design an algorithm for finding a simple path through all points in P. Make your algorithm as time efficient as possible.

2. In class we showed that computing the convex hull of n points in a comparison based model requires $\Omega(n \log n)$ time. Show that given a simple path for these points one can find the convex hull in $O(n)$ time.

   HINT:

   The idea is to run a variant of incremental convex hull where we add the points in the order they appear on the path. Suppose we are give a simple path $Path = (p_1, \ldots, p_n)$ on n distinct points and for simplicity no three are collinear. We start by constructing the triangle from the first three points and storing it as a doubly linked list of edges and recording which vertex is connected to the remain points on the path.

   Let $I = \{i \mid p_i \in CH(p_1, \ldots, p_i)\}$ We will for each $i \in I$ incrementally compute the convex hull of $(p_1, \ldots, p_i)$. Make sure your algorithm handles the case when the point p_{i+1} is interior to $CH(p_1, \ldots, p_i)$.

   Use amortized analysis to show that your algorithm runs in $O(n)$ time.

3. Show that in general any comparison based algorithm that finds a simple path of the points in P requires $\Omega(n \log n)$ comparisons.

(25) 3. **Star Shaped Polygon**

A polygon $\mathcal{P}$ is **star shaped** if there exists a point in the interior of $\mathcal{P}$ that can see all of the interior.

1. Give an $O(n)$ expected time algorithm to determine if a simple polygon of size n is star shaped.
2. Give a $O(\log n)$ time algorithm for determining if a point q is in a star shaped polygon $\mathcal{P}$. We assume that the the vertices of P are given in CW order and that we are also given a point p that can see all of the interior P.

(25) 4. **Circular Partition**

Given a set of red points R and a set of green point G in the plane give an algorithm to find a disk D such that $G \subset D$ and $R \cap D = \emptyset$ if one exists. Your algorithm should run in expected linear time in the size of R and G.

(20) 5. **Broken Simple Cycle and Convex Hull Algorithm**

Suppose $P = \{p_0, ..., p_n\}$ is a set of points in the plane. We say that sequences of distinct points $\{p_0, ..., p_n\}$ form a simple cycle if the line segments $l_i = [p_i, p_i + 1]$ are disjoint except for $l_i \cap l_{i+1}$. Furthermore we require $p_0 = p_n$ and $l_{n-1} \cup l_1 = p_1$.

Given a simple cycle, P, consider the following algorithm to compute $CH(P)$. Let S be a stack, initialized with s_0, s_1 where s_0 is the leftmost point of P and s_1 is the clockwise successor in P. You then successively process points p_i going clockwise around the cycle. (Assume $p_0 = s_0$, $p_1 = s_1$).

1. While the top of the stack is not s_0, take the next point in P, some p_i along with the top two points in the stack, s_{t-1}, s_t.
2. If $s_{t-1}s_tp_i$ form a right turn, add p_i to the stack and continue.
3. While $s_{t-1}s_tp_i$ form a left turn, pop the stack. Then add p_i.

Intuitively, what this algorithm does is go around the simple cycle clockwise and check whether you take a right turn from the top two points on the stack, to get to a given point - if you do, add the point on the stack. If the turn is left,pop the stack, and then add your current point to the stack. It may help to check that this algorithm works for a simple 5 -sided star.

Clearly, this algorithm terminates. What's more, it runs in $O(|P|)$! However, there's a bug in this algorithm - find a counter-example for which this algorithm fails to find the convex hull of P. Try to give a general description for the types of polygons this algorithm fails on.

(30) 6. *** Output Sensitive Convex Hull**

In class we analyzed two convex hull algorithms - merge hull and a randomized incremental algorithm, both of which ran in $O(n \log(n))$. We will now describe and analyze an output sensitive convex hull algorithm. If the number of vertices determining the boundary of $CH(P)$ is h, we want an algorithm that runs in $O(n \log(h))$.

- Let P be a convex polygon in $\mathbb{R}^2$ on $m > 1$ vertices and $p \in \mathbb{R}^2$ such that p is not interior to $CC(P)$. Define a tangent from p to P as a directed line l between $p \neq q \in P$ such that all $x \in P$ lie to the left of or on l. Prove that l is uniquely defined, and give an algorithm to find l in $O(\log(m))$ time. State any assumptions on how P is stored.

Suppose a 'little birdie' tells us the size h of the convex hull of a point set S, $|S| = n$. Consider the following divide and conquer convex hull algorithm. Our algorithm will divide S into n/h sets, each of size at most h and compute the convex hull of each of the smaller sets, and finally merge the results. We next give hints on an efficient merge algorithm.

- Suppose p_l is the leftmost point of S. Assuming the sub-hulls for each of our n/h pieces of size h have been computed, show how to find the next edge of the convex hull of S in $O((n/h) \log h)$ time. [Hint: think of a very simple search algorithm].
- What is the overall complexity of computing the convex hull of S by repeated application of the previous find-edge procedure? Be sure to explain your result.

Unfortunately we're basing all of our analysis on this 'little birdie' who so kindly told us the value of h. We will now see how to remove this dependence, and still meet the same bounds.

Suppose we start with h', an initial guess of h where $h' < h$. Upon running the divide and conquer algorithm described above with h' we will quickly see in the reconstruction step that we need more than h' edges in order to construct the global hull. Instead of running the reconstruction procedure for all h steps, let's terminate the algorithm once we observe that the global hull requires more than h' edges.

- Suppose that initially, $h' = 3$, and that each time we notice that h' is incorrect, we square h' and restart the algorithm. Clearly this new algorithm terminates, as in the final iteration, $h' < h^2$, and we shall fully reconstruct the global hull. Prove the running time of this procedure is $O(n \log(h))$.