1. (a) Describe and analyze a *self-contained*[1] algorithm to compute the area of a simple polygon in $O(n)$ time.

   (b) Describe and analyze an algorithm to compute the area of a (not necessarily convex) three-dimensional *polyhedron* in $O(n)$ time. Here, a polyhedron is an object whose boundary is made up of (not necessarily convex) simple polygons. Any two boundary polygons meet at a single edge, a single vertex, or not at all; exactly two boundary polygons meet at each edge; and the union of the boundary polygons is connected. Assume the polyhedron is represented in some reasonable data structure.

2. Recall that a *chord* of a simple polygon is a line segment between two vertices that lies entirely inside the polygon except at its endpoints. A chord is *balanced* if it splits the polygon into two pieces, each of which has at least $n/3$ vertices.

   (a) Prove that every n-gon with $n > 3$ has a balanced chord. *[Hint: This is easy.]*

   (b) Describe and analyze a *self-contained* algorithm to find a balanced chord in $O(n)$ time. *[Hint: This is not so easy.]*

   (c) Use the algorithm in part (b) to build yet another algorithm to triangulate a polygon in $O(n \log n)$ time. *[Hint: This is trivial.]*

3. In class, we defined trapezoidal decompositions only for sets of disjoint segments, but we can easily generalize the definition to allow for intersections: add walls above and below every intersection point. This problem asks you to describe algorithms for computing trapezoidal decompositions in this more general setting.

   (a) Describe and analyze a randomized incremental algorithm that runs in $O(n \log n + k)$ time, where k is the number of intersecting pairs.

   (b) Describe and analyze a sweep-line algorithm that runs in $O(n \log n + p \log n)$ time, where p is the number of intersection points. Do *not* assume general position! Note that p can be much smaller than k. *[Hint: $O(n \log n + k \log n)$ time is pretty easy. But how do you quickly handle lots of segments crossing at a common point? There is an algorithm in the textbook, but it can be improved!]*

   *(c) Describe and analyze an algorithm that runs in $O(n \log n + p)$ time.

   In all three cases, you only need to describe differences from the algorithms and analysis presented in class.

[1] Here and in question 2, "self-contained" means you may not invoke Chazelle's linear-time triangulation algorithm.

4. A *triangulation* of a non-convex polyhedron is a decomposition of the polyhedron into simplices, such that any two simplices meet in a (possibly empty) face of both, and so that any vertex of a simplex is also a vertex of the original polyhedron. In class, we saw that Schönhardt's twisted prism cannot be triangulated without adding new vertices, which are usually called *Steiner points*.

   (a) Prove that any non-convex polyhedron with n faces can be triangulated using $O(n^2)$ Steiner points. *[This bound is tight!]*

   (b) Prove that Schönhardt's twisted prism can be triangulated using a single Steiner point in the interior of a concave edge.

   *(c) Prove (preferably by describing an algorithm) or disprove (via a counterexample) that every non-convex three-dimensional polyhedron can be triangulated if Steiner points are added *only along its edges*.

   $\bigstar$(d) How does this result generalize to higher dimensions? Can any non-convex d-dimensional polyhedron be triangulated using Steiner points only on its edges? Only on its ridges? Only on its $\lfloor d/2 \rfloor$-faces?