Don’t panic!

You have at most three hours to complete this exam and upload your solutions. The clock started when you opened the Gradescope assignment. All solutions must be uploaded before 10pm (Central Standard Time) on Monday, March 8.

The exam is designed to be completed on paper in 90 minutes. There are four problems.

This is an open-book-but-closed-everything-else exam. You are welcome to use any of this semester’s official course materials (textbook, notes, lecture scribbles, homework solutions) and anything you wrote yourself before starting the exam. All other resources, including materials from other classes and/or previous semesters, other textbooks, other internet resources, and (most importantly) other people, are not permitted.

If you use a standard algorithm or data structure from this class, or from any prerequisite class (such as CS 374, CS 225, CS 173, or CS 125) as a black box, please do not give a detailed description; just name the algorithm or data structure. (For example: “balanced binary search tree” or “Gaussian elimination” or “Chan’s algorithm” or “the plane-sweep algorithm to count segment intersections”.)

Similarly, if you use a minor variant of an algorithm from this class or a prerequisite class, please describe only the necessary modifications.

You can implicitly assume general position without comment. You are welcome to use randomized algorithms, but none of the problems in this exam require them.

I cannot answer questions about the exam while the exam is in progress. If you need to make additional assumptions to solve an exam problem, please state those assumptions clearly in your solution. Do not discuss the exam with anyone until after Monday at 10pm.

The exam is not proctored. I am trusting you to take the exam by yourself, with no help from anyone, within the declared time limits. The exam is first and foremost a mechanism to give you honest feedback on your mastery of the course material; please treat it as such. All academic integrity policies are still in place.

Soon may the Wellerman come, to bring us sugar and tea and rum. One day, when the tonguin’ is done, we’ll take our leave and go.
1. Let $P$ be a set of points in the plane. A triangulation of $P$ is a planar straight-line graph whose vertices are the points $P$, whose outer face is the complement of the convex hull of $P$, and whose bounded faces are all triangles.

Describe and analyze an algorithm to compute a triangulation of a given set of $n$ points in the plane. For full credit, your algorithm should run in $O(n \log n)$ time.
2. Suppose you are given a set of $n$ “pyramids” in the plane. Each pyramid is a right isosceles triangle whose short edges have slope $\pm 1$ and whose long edge lies on the $x$-axis. Each pyramid is represented by the $x$- and $y$-coordinates of its topmost point.

The silhouette of these pyramids is the boundary of the union of the pyramids and the halfplane $y \leq 0$; see the red curve in the figure below. Your task is to compute this silhouette.

(a) Describe and analyze an algorithm that determines which pyramids are visible on the silhouette. These are the pyramids with black points in the figure above; the pyramids with white points are not visible.

(b) Once you know which pyramids are visible, how do you compute the actual shape of the silhouette? Briefly sketch and analyze an algorithm to compute the left-to-right sequence of silhouette vertices, including the vertices between pyramids and on the ground, given the output of your algorithm from part (a).
3. Suppose you are given a set $T$ of $n$ triangles, none of whose edges intersect, but some of which may be nested. The *nesting depth* of these triangles is the largest number $d$ such that some point lies inside $d$ triangles, but no point lies inside $d + 1$ triangles. For example, the figure below shows a set of triangles with nesting depth 4.

Describe and analyze an algorithm to compute the nesting depth of $T$. For full credit, your algorithm should run in $O(n \log n)$ time.
4. Describe and analyze an algorithm that determines whether a given point \( q \) lies in the interior of a given convex polygon \( P \). The input polygon \( P \) is represented as an array \( P[1 \ldots n] \) of its vertices in counterclockwise order, with the leftmost vertex stored at \( P[1] \). For full credit, your algorithm should run in \( O(\log n) \) time.