CS 498 TC ✧ Computational Geometry ✧ Spring 2022

Homework 1

Due Thursday, February 10, 2022 at 8pm

✧ Homework Policies ✧

- I expect to release a new homework set *roughly* every two weeks, with an *average* of five problems per set, but I also expect high variance on both counts. This homework is slightly shorter, so that I can squeeze in two homeworks before the midterm.
  - On average, 3-credit students should submit solutions for at least three problems in each homework set. (Only the top 60% of your problem scores count.)
  - On average, 4-credit students should submit solutions for at least four problems in each homework set. (Only the top 80% of your problem scores count.)
- You are welcome to use *any* resource at your disposal to help you solve homework problems. You do *not* need to cite this semester's course materials (linked from the schedule page), or sources for material typically covered in prerequisite classes (173, 225, and 374). Otherwise, you must cite every source that you use.
- Teams of up to three students can submit joint homework solutions. You are welcome to work on the homework in larger groups, but each team must write up solutions in their own words, giving proper credit to any other students (or groups) that helped them. Please list all team members at the top of the first page of each submitted solution.
- Submit your solutions on Gradescope, following the link on the course web page, as separate PDF files for each numbered problem. Exactly one member of each homework team should upload the team's solution and identify the other team members.
- Whenever a homework or exam problem asks for an algorithm, your solution should include the following:
  - If necessary, a concise specification of the problem your algorithm actually solves. (This could be more general than the actual homework/exam problem.)
  - If necessary, a concise description of the input and output representations.
  - A clear description of the actual algorithm, preferably in pseudocode.
  - A brief justification of your algorithm's correctness.
  - A brief analysis of the algorithm's running time.
- Unless the problem explicitly states otherwise, you may implicitly assume that inputs are in general position. Similarly, if a problem asks for a certain running time, a randomized algorithm with that *expected* running time is fine.
- Homework and exam scores will depend not only on correctness and speed, but also on clarity and style. Your solutions should be complete and mathematically precise, but at the same time, easy to understand and concise. For most numbered homework problems, a complete solution should fit on a single typeset page. Don't submit your first draft!

1. A *circular array* is a standard array that is being used to store a circular sequence of values. Every element $A[i]$ in a circular array $A[0..n-1]$ has a successor $A[(i+1) \bmod n]$ and a predecessor $A[(i-1) \bmod n]$.

   A *local minimum* in a circular array is any element that is strictly smaller than both its predecessor and its successor. Similarly, a *local maximum* is any element that is strictly greater than both its predecessor and its successor.

   Finally, a circular array A is *strictly bitonic* if no element of A is equal to its successor, and the smallest element of A is the only local minimum in A.

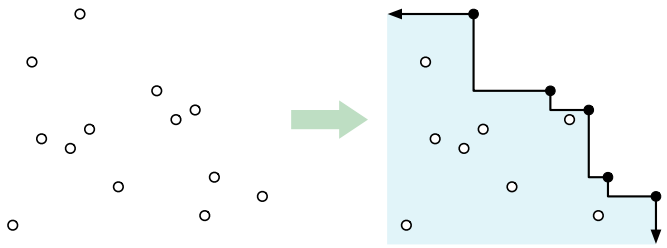
   (a) Prove that the largest element of a strictly bitonic array is also its only local maximum.

   (b) Describe an algorithm to find the smallest element in a strictly bitonic array $A[0..n-1]$ in $O(\log n)$ time.

   (c) Suppose we are given a convex polygon P (represented as usual as a circular array of vertex coordinates) and a line ℓ. Describe an algorithm that determines in $O(\log n)$ time whether ℓ intersects P. For full credit:

   - If P and ℓ intersect, your algorithm should return the intersection points.
   - Otherwise, your algorithm should return the closest point in P to ℓ.

2. Let P be a set of points in the plane. A point $p \in P$ is *Pareto-optimal* if no other point in P is both above and to the right of p. The Pareto-optimal points can be connected by horizontal and vertical lines into the *staircase* of P, with a Pareto-optimal point at the top right corner of every step.



The staircase of a set of points in the plane.

   Now suppose you are given a set P of n points in the plane, with distinct x- and y-coordinates.

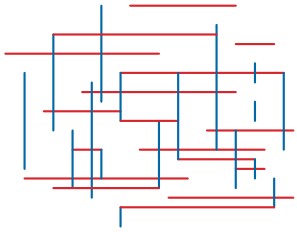
   (a) Suppose the points in P are given in order from left to right, that is, sorted by increasing x-coordinate. Describe an algorithm to compute the the staircase of P in $O(n)$ time.

   (b) Part (a) implies that when the the input array P is *not* sorted, we can still compute the staircase of P in $O(n \log n)$ time. Describe an algorithm to compute the the staircase of P in $O(n \log h)$ time, where h is the number of Pareto-optimal points in P. (For partial credit, describe an algorithm that runs in $O(nh)$ time.)

3. Suppose we are given a set H of horizontal line segments and a set V of vertical line segments. A *crossing point* is any point that lies on both a horizontal segment in H and a vertical segment in V.

   (a) Describe an algorithm that computes the number of crossing points. For full credit, your algorithm should run in $O(n \log n)$ time, regardless of the number of intersections. Assume that the segments H and V are in general position. *[Hint: Use a sweep-line algorithm. What operations does your sweep data structure need to support? Design a data structure that supports those operations!]*

   (b) Modify your algorithm from part (a) to handle ***arbitrary*** horizontal and vertical segments. In particular, your modified algorithm should handle segments with shared endpoints, overlapping collinear segments, and zero-length segments. Each crossing point should only be counted only once, even if it lies on multiple horizontal and/or vertical segments. Your modified algorithm should still run in $O(n \log n)$ time.

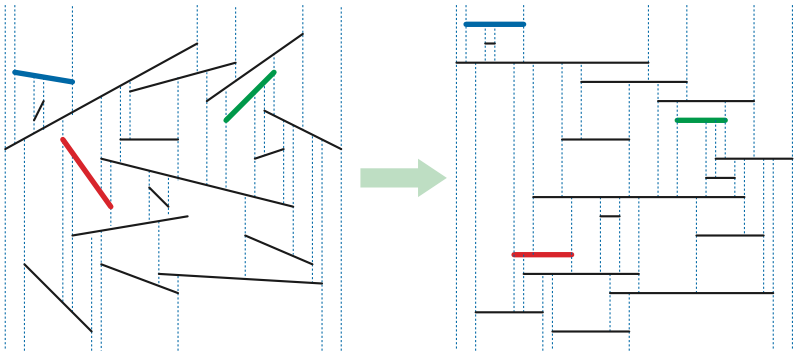


4. Suppose we are given a set $S = \{s_1, s_2, \ldots, s_n\}$ of disjoint line segments in the plane. Another set $R = \{r_1, r_2, \ldots, r_n\}$ of *horizontal* line segments is called a *rectification* of S if the following properties hold for all indices i and j:

   - The endpoints of s_i and r_i have the same x-coordinates. Thus, if a vertical line ℓ intersects s_i, then ℓ also intersects r_i.
   - Suppose vertical line ℓ intersects both s_i and s_j. Then $\ell \cap s_i$ is above $\ell \cap s_j$ if and only if $\ell \cap r_i$ is above $\ell \cap r_j$.

   Equivalently, R is a rectification of S if the trapezoidal decompositions of R and S are combinatorially isomorphic.

   Describe and analyze an algorithm to compute a rectification of S. *[Hint: First prove that S must contain at least one segment with no other segments in S directly above it.]*



Rectifying a set of line segments. Colors indicate corresponding segments.