Prof. Dr. Jasmin Blanchette
Dr. Martin Desharnais-Schäfer
Dr. Michael Kirsten
Elisabeth Lempa

Ludwig-Maximilians-Universität München
Institut für Informatik
Discussion on 17.12.2025
Homework due on 07.01.2026 at 16:00

Possible solution for Exercise Sheet 10 in

Scientific and Technical English for Computer Scientists

The exercise sheets consist of in-class exercises and homework. The in-class exercises take place during the second half of the lecture time slots. The homework, which is optional and ungraded, can be submitted via the "Homework" section in Moodle. The homework is subject to peer review.

Unless indicated otherwise, generative artificial intelligence assistants such as ChatGPT may be used, as long as you acknowledge how you use them as specified by the Institute's policy on plagiarism.[1] However, you may not use such tools to generate peer reviews for you. In addition, we strongly recommend that you do not use them to generate entire solutions, since that would defeat the purpose of the exercises.

Homework 10-2 ***Incorporating Feedback*** For your bachelor's thesis about a topic in computational geometry, you have written the following background section. It is designed to give a brief introduction to computational geometry problems and to serve as a motivating example for the algorithms you present in later sections.

> Imagine you are walking on a university campus and suddenly you realize that you need to make an urgent phone call. There are many public phones on campus, and of course you want to go to the nearest one. But which one is the nearest? It would be helpful to have a map on which you could look up the nearest public phone, no matter where you are on campus. The map should show a subdivision of the campus into regions, and for each region indicate the nearest public phone. What would these regions look like? And how could we compute them?
>
> This is just one example of a geometric problem requiring carefully designed geometric algorithms for their solution. The field of computational geometry emerged in the 70ies and deals with such geometric problems. It can be defined as the systematic study of algorithms and data structures for geometric objects.
>
> To illustrate the issues that arise in developing a geometric algorithm, this section deals with one of the first problems that was studied in computational geometry: the computation of planar convex hulls.

[1] `https://www.medien.ifi.lmu.de/lehre/Plagiate-IfI.pdf`

A subset S of the plane is called convex if and only if for any pair of points $p, q \in S$, the line segment $\overline{pq}$ is completely contained in S. The *convex hull* $\mathcal{CH}(S)$ of a set S is the smallest convex set that contains S. To be more precise, it is the intersection of all convex sets that contain S.

How do we compute the convex hull? Before we can answer this question, we must ask another question: What does it mean to compute the covnex hull?

As we have seen, the convex hull of P is a convex polygon. A natural way to represent a polygon is by listing its vertices in clockwise order, starting with an arbitrary one. So the problem we want to solve is this: Given a set $P = \{p_1, p_2, ..., p_n\}$ of points in the plane, compute a list that contains those points from P that are the vertices of $\mathcal{CH}(P)$, listed in clockwise order.

Algorithm 1 ConvexHull

Input:
A set P of points in the plane.
Output:
A list L containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

```
1: E ← ∅
2: for all ordered pairs (p, q) ∈ P × P with p not equal to q do
3:     valid ← true
4:     for all points r ∈ P not equal to p or q do
5:         if r lies to the left of the directed line from p to q then
6:             valid ← false.
7:         end if
8:     end for
9:     if valid then
10:        add the directed edge p→q to E.
11:    end if
12: end for
13: From the set E of edges construct a list L of vertices of CH(P), sorted in clockwise
    order.
```

Analyzing the time complexity of CONVEXHULL is easy. In the outer loop, we check $n^2 - n$ pairs of points. For each pair, we look at the $n - 2$ other points to see whether they all lie on the right side. This will take $O(n^3)$ time in total. An algorithm with a cubique running time is too slow to be of practical use for anything but the smallest input sets. To improve upon, this we apply a standard algorithmic design technique: an incremental algorithm.

Your supervisor and your cosupervisor have given you the following feedback in response to this section. Please edit the section in light of the feedback.

"The prose in the introduction is too flowery. All in all, you should try to write more formally."

"Public phones are no longer relevant."

"Algorithm 1 is a float, but it's not referenced in the text."

"I really like the beginning. It was a good informal motivation and fun to read."

"There is a jarring break between the first part with the example and the second part with the algorithm. You should write some transition text."

"There are some typos and spelling errors, like 'covnex'. You should get into the habit of using a spellchecker."

POSSIBLE SOLUTION:

Imagine you are walking on a university campus and suddenly you realize that you need to charge your smartphone. There are many power outlets on campus, and of course you want to go to the nearest one. But which one is the nearest? It would be helpful to have a map on which you could look up the nearest power outlet, no matter where you are on campus. The map should show a subdivision of the campus into regions, and for each region indicate the nearest power outlet. What would these regions look like? And how could we compute them?

This is just one example of a geometric problem requiring carefully designed geometric algorithms for their solution. In the 1970s, the field of computational geometry emerged, dealing with such geometric problems. It can be defined as the systematic study of algorithms and data structures for geometric objects.

To illustrate the issues that arise in developing a geometric algorithm, we will give as a motivating example one of the first problems that was studied in computational geometry: the computation of planar convex hulls.

A subset S of the plane is called convex if and only if for any pair of points $p, q \in S$, the line segment $\overline{pq}$ is completely contained in S. The *convex hull* $\mathcal{CH}(S)$ of a set S is the smallest convex set that contains S. To be more precise, it is the intersection of all convex sets that contain S.

As we have seen, the convex hull of P is a convex polygon. A natural way to represent a polygon is by listing its vertices in clockwise order, starting with an arbitrary one. So the problem we want to solve is this: Given a set $P = \{p_1, p_2, \ldots, p_n\}$ of points in the plane, compute a list that contains those points from P that are the vertices of $\mathcal{CH}(P)$, listed in clockwise order. Algorithm 1 is a naive approach to solve the problem.

Algorithm 1 ConvexHull

Input:
A set P of points in the plane.

Output:
A list L containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

```
1: E ← ∅
2: for all ordered pairs (p, q) ∈ P × P with p not equal to q do
3:     valid ← true
4:     for all points r ∈ P not equal to p or q do
5:         if r lies to the left of the directed line from p to q then
6:             valid ← false.
7:         end if
8:     end for
9:     if valid then
10:        add the directed edge pq→ to E.
11:    end if
12: end for
13: From the set E of edges construct a list L of vertices of CH(P), sorted in clockwise order.
```

Analyzing the time complexity of CONVEXHULL is easy. In the outer loop, we check $n^2 - n$ pairs of points. For each pair, we look at the $n - 2$ other points to see whether they all lie on the right side. This will take $O(n^3)$ time in total. An algorithm with a cubic running time is too slow to be of practical use for anything but the smallest input sets. To improve upon this, we apply a standard algorithmic design technique: an incremental algorithm.

(Adapted from Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars, *Computational Geometry: Algorithms and Applications*, Springer, 2008.)