**Computational Geometry**
**Summer Term 2025**
`scale.iti.kit.edu`

# Exercise Sheet 3

Submission due by 2025-06-19

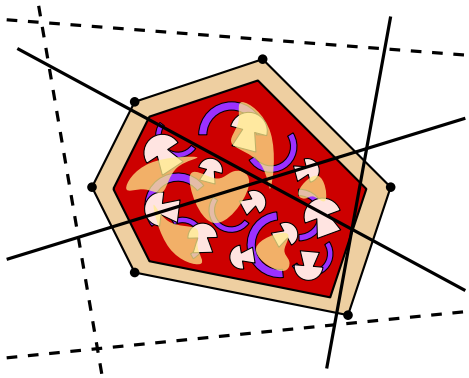
## Problem 1: Pizza-Guillotine

**5 + 5 + 5 = 15 points**

Many students complain that the local pizzeria delivers their baked goods uncut. To improve customer satisfaction, the pizza-cutting process is to be automated. A first prototype is a *pizza guillotine*. It contains an infinitely long blade that can in principle be freely translated and rotated in the plane, and then brought down onto the pizza. However, since the prototype is still in its early stages, the blade's orientation can only be changed using the `Reset` command, which sets the blade to a randomly chosen translation and orientation.

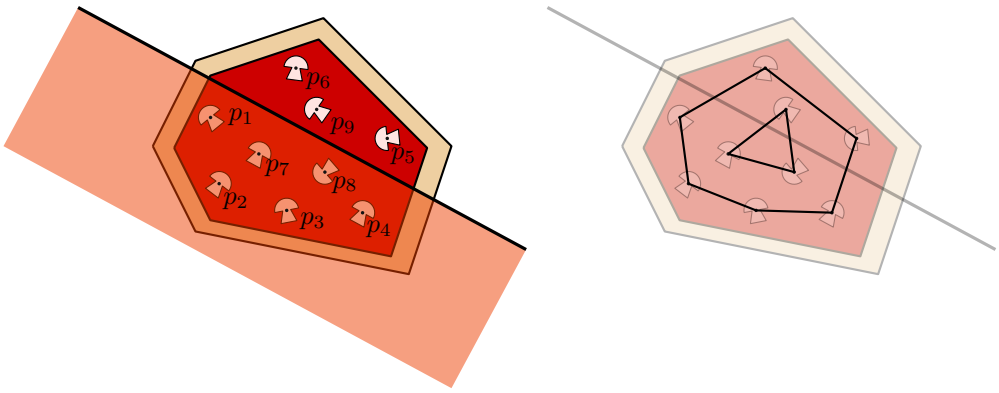
**Part (a)** To still be able to work with the prototype, the pizzeria agreed upon the following approach: First, it is tested whether the current blade orientation is unsuitable. If it is, a `Reset` is performed and the process is repeated. Otherwise, the cut is made. Since this process may be performed frequently, the test must be fast!

Unfortunately, it can happen that the blade, in its current orientation, is poorly positioned or even misses the pizza entirely. In this case, a new orientation must be tried via `Reset`. A pizza is represented as a convex polygon with $n$ points. Provide an algorithm that, after $\mathcal{O}(n)$ preprocessing time, can determine in $\mathcal{O}(\log n)$ time for a given blade orientation which points on the edge of the pizza the blade intersects.



**Part (b)** To save even more time, the oven and blade are to be combined so that the pizza is baked while it is being cut! For cost reasons, however, only one side of the blade can be equipped with the oven functionality.

There are $n$ ingredients on the pizza, represented by the points $p_1, \ldots, p_n$. In order to ensure, that they are not exposed to the heat multiple times, for each potential cut it must be determined which $k$ ingredients lie on the oven side of the blade.

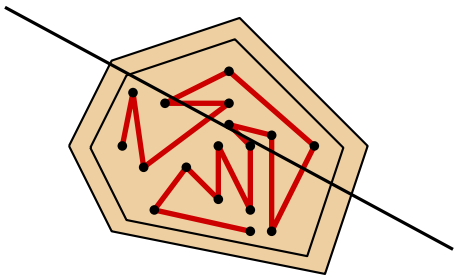



Provide an algorithm that, using the algorithm from part (a), can determine the $k$ ingredients on the oven side of the blade in $\mathcal{O}(\log n + k)$. The algorithm can use $\mathcal{O}(n)$ preprocessing time.

***Note**: You may assume that the input is given as the nested convex hull of the points $p_1, \ldots, p_n$. This nested convex hull is obtained by first computing the convex hull of all points, and then iteratively computing the convex hull of points not included in any previously computed convex hull.*

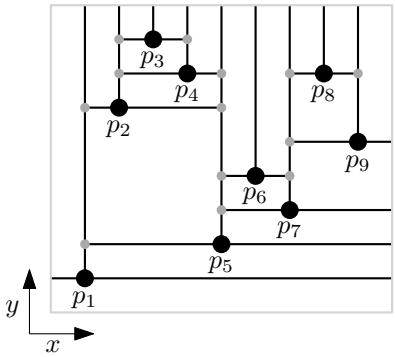
**Part (c)** One final issue in this (otherwise optimal) approach is the lifespan of the blade, which is heavily affected by the acidity of the tomato sauce. Now it should be taken into account that the sauce is "distributed" on the pizza by a robot, which pours it in the form of a polygonal chain consisting of $n$ points. If the blade intersects the sauce too much, a new orientation must be tried via `Reset`.

Provide an algorithm that, using the algorithm from part (a), can determine the $k$ intersection points of the blade with the polygonal chain in $\mathcal{O}((k+1)\log(n/(k+1)))$ time. The algorithm can use $\mathcal{O}(n \log n)$ preprocessing time.



## Problem 2: Half 2D Range Queries **5 points**

Given are points $p_1, \dots, p_n \in \mathbb{R}^2$ sorted by their $x$-coordinates. A data structure to efficiently answer half-range queries divides the plane into cells. From each point $p_i$, a vertical ray is shot upward, and then horizontal line segments are inserted from $p_i$ to the nearest rays on both sides whose starting points have smaller y-coordinates than $p_i$ itself.




Provide an algorithm that constructs the geometric graph representing this data structure in $\mathcal{O}(n)$ time. Prove that your algorithm does not exceed this time complexity.

## Problem 3: Range-Tree Implementation **5 bonus points**

*In this optional exercise, you have the opportunity to implement a range tree as seen in the lecture. The points you collect in this exercise do not count towards the total points of this sheet, but can be used to compensate for missing points on this or another sheet.*

Implement a 2D range tree using a programming language of your choice. The construction time should be in $\mathcal{O}(n \log^2 n)$, and the query time should be in $\mathcal{O}(\log^2(n) + k)$ per query. You should need no optimizations to solve our test cases, but you can implement optimizations (from the lecture or other sources) if you like.

Briefly explain in your submission how you implemented the range tree, e.g. which internal data structures are you using and which optimizations have you implemented. Further, put your solutions in the `.pdf` document using the format that is explained below, and submit your code as a separate file.

You can find our test cases as a `.zip` file on our website. It contains two folders: *sample* contains test cases with solutions and *test* only contains test cases. Submit your solutions for the test cases in the *test* folder.

**Input format:** The first line consists of two numbers, $n$ and $q$, the number of points and the number of queries. In each of the next $n$ lines, the $x$-coordinate and the $y$-coordinate of a point are given. Both are integers. Note that there might be multiple points with the same coordinates.

The next $q$ lines consist of four integers $\min_x$, $\max_x$, $\min_y$, $\max_y$ each describing a rectangular query.

**Output format:** For each query, we are looking for all points $(x, y)$ with $\min_x \leq x \leq \max_x$ and $\min_y \leq y \leq \max_y$. Since this set can be very large, we compress the solution in the following way: Let $P_i$ be the set of points within the given rectangle of query $i$. Compute three numbers for query $i$: $|P_i|$, the size of the solution, $x_i = (\sum_{(x,y)\in P_i} x) \mod (10^9 + 7)$, the sum over all $x$-coordinates in $P_i$ modulo $10^9 + 7$, and $y_i = (\sum_{(x,y)\in P_i} y) \mod (10^9 + 7)$, the sum over all $y$-coordinates in $P_i$ modulo $10^9 + 7$.

The output for a test case only consists of three numbers: The sum over all $|P_i|$ (modulo $10^9 + 7$), the sum over all $x_i$ (modulo $10^9 + 7$), and the sum over all $y_i$ (modulo $10^9 + 7$). We use the modulo to avoid any overflows (do not apply it too late in your calculations).