Computational Geometry (CS 236719)

http://www.cs.technion.ac.il/~barequet/teaching/cg/fa21

Chapter 1

Introduction

Copyright 2002-2015

Prof. Gill Barequet

Center for Graphics and Geometric Computing

Dept. of Computer Science

The Technion

Haifa

Thanks to **Michal Urbach-Aharon** who prepared the initial version of the presentations of this course

Staff (Fall 2021-22 תשפ״ב)

- **Lecturer: Prof. Gill Barequet**
  E-mail: **barequet**@cs.technion.ac.il
- **TA: Ms. Amani Shhadi**
  E-mail: **amani.shhadi**@cs.technion.ac.il
- Office hours: Any time (by appointment)
- Lecture: Monday 10:30-12:30
- Recitation: Monday 12:30-13:30
- Exams: Term A: Monday, January 31, 2022
  Term B: Hopefully no need to!

Bibliography

- **Computational Geometry: Algorithms and Applications,**

  M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf,

  3rd edition, Springer-Verlag, 2008.

- **Course slides**

Assessment

- **Three dry homework assignments** (~12.5%)
- **One wet (running) exercise** (~12.5%)
- **No midterm exam**
- **Final exam** (75%)

Syllabus

- Introduction
- Basic techniques
- Basic data structures
- Polygon triangulation
- Linear programming
- Range searching
- Point location
- Voronoi diagrams
- Duality and Arrangements
- Delaunay triangulations
- Applications and miscellaneous

Prerequisite course:
Data Structures
(Recommended but not mandatory:
Algorithms)

Questions?

Lecture Topics

- Sample problems
- Basic concepts
- Convex-hull algorithms

Sample Problems

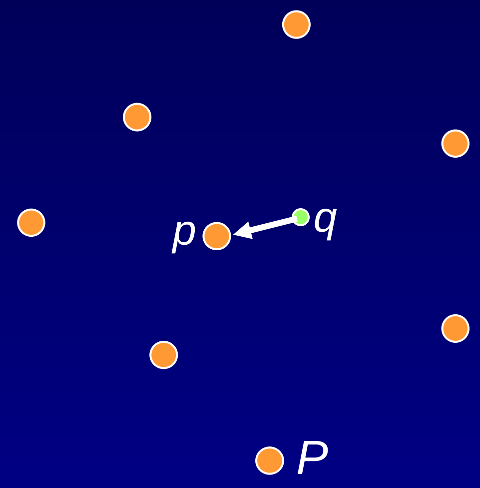
[Convex Hull demo]

[Voronoi Diagram demo]

[Visibility demo]

Nearest Neighbor

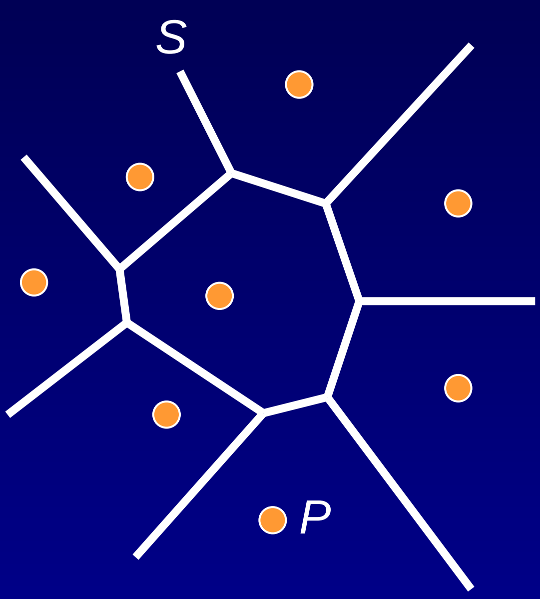
- Problem definition:
  - Input: A set of points (*sites*) *P* in the plane and a query point *q*.
  - Output: The point $p \in P$ closest to *q* among all points in *P*.



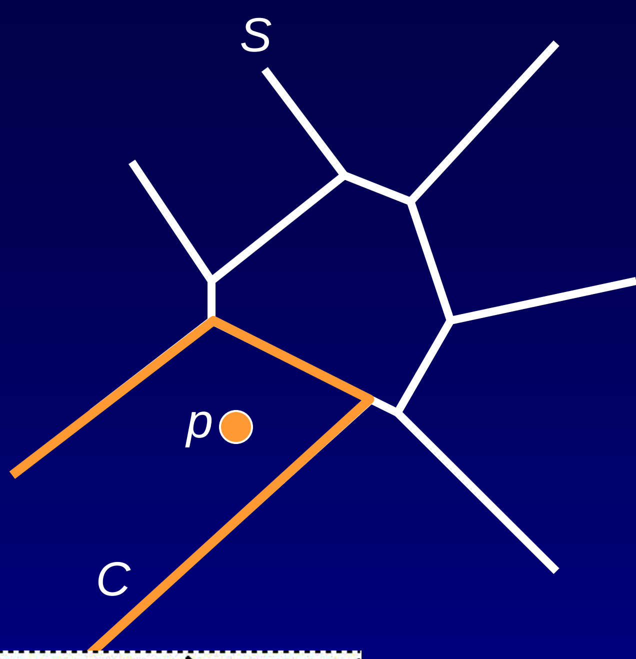
- Rules of the game:
  - One point set, multiple queries
- Application: Cellphones
  Store Locator

The Voronoi Diagram

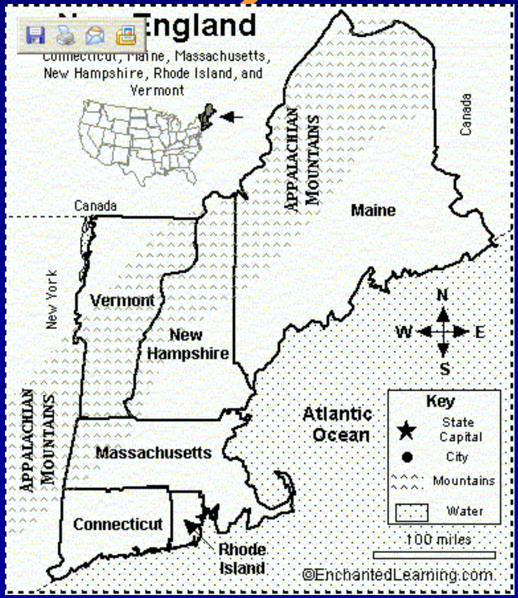
- Problem definition:
  - Input: A set of points (*sites*) *S* in the plane.
  - Output: A planar subdivision *S* into cells, one per site. The cell corresponding to $p \in P$ contains all the points to which *p* is the closest.



Point Location

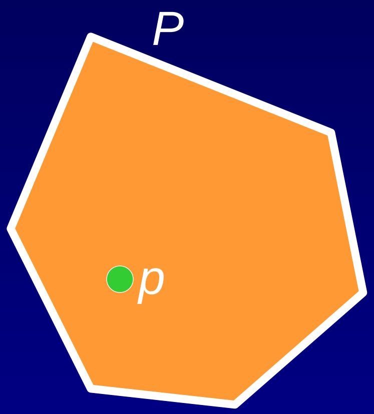


- **Problem definition:**
  - Input: A partition S of the plane into cells and a query point p.
  - Output: The cell $C \in S$ containing p.
- **Rules of the game:**
  - One partition, multiple queries
- **Applications:** Nearest neighbor
  State locator



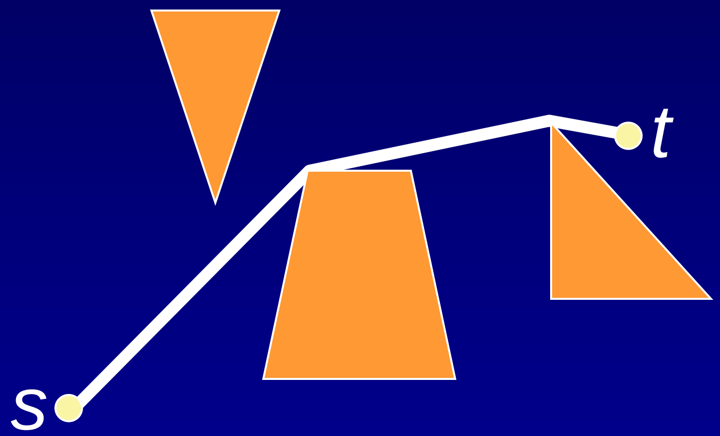
Point in Polygon

- Problem definition:
  - Input: A polygon *P* in the plane and a query point *p*.
  - Output: *true* if $p \in P$, else *false.*



- Rules of the game:
  - One polygon, multiple queries

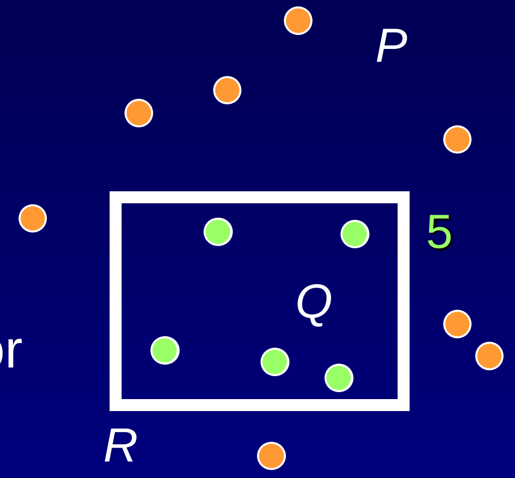
Shortest Path



- Problem definition:
  - Input: Obstacles locations and *query* endpoints *s* and *t*.
  - Output: The shortest path between *s* and *t* that avoids all obstacles.
- Rules of the game:
  - One obstacle set, multiple queries (*s*,*t*).
- Application: Robotics.

Range Searching and Counting

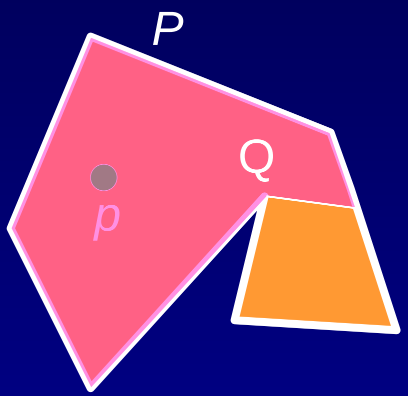
- Problem definition:
  - Input: A set of points *P* in the plane and a query rectangle *R*.
  - Output:
    (report) The subset $Q \subseteq P$ contained in *R*; or
    (count) The cardinality of *Q*.



- Rules of the game:
  - One point set, multiple queries.
- Application: Urban planning

Visibility

- Problem definition:
  - Input: A polygon *P* in the plane and a query point *p*.
  - Output: The polygon $Q \subseteq P$ containing all points in *P* visible to *p*.



- Rules of the game:
  - One polygon, multiple queries
- Applications: Security

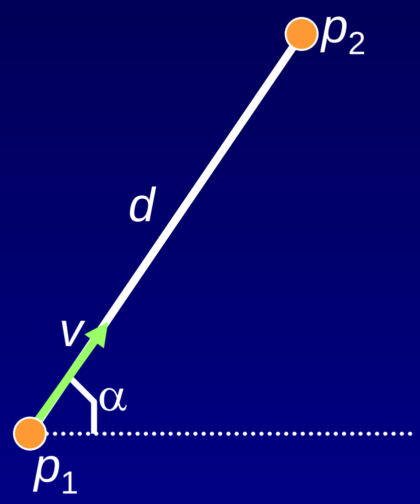
Questions?

Basic Concepts

Representing Geometric Elements

- Representation of a line segment by four real numbers:
  - Two endpoints (p_1 and p_2)
  - One endpoint (p_1), vector direction (v) and parameter interval length (d)

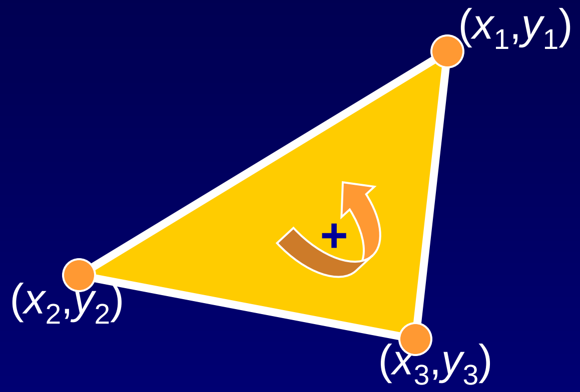
    (Question: where did the extra parameter come from?)
  - One endpoint (p_1), a slope (α), and length (d)
  - Other options…
  - Unique representation?



- Different representations may affect the running times of algorithms!

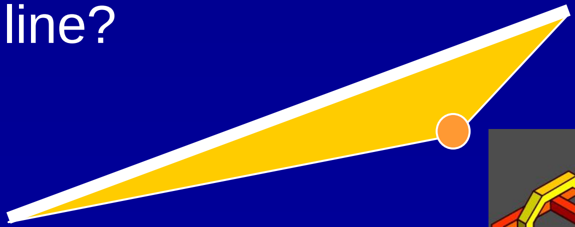
Orientation

$$\text{Area} = \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$



- The sign of the area indicates the orientation of the points.
- Positive area ≡ counterclockwise orientation ≡ left turn.
- Negative area ≡ clockwise orientation ≡ right turn.

- **Question:** How can this be used to determine whether a given point is “above” or “below” a given line?
  (Hint: … or a line segment?)
  (Degenerate instances?)



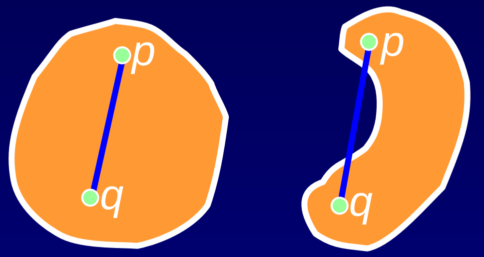
Complexity (reminder)

Symbol	Definition	"Nickname"
$f(n) = O(g(n))$	$\exists N,C\ \forall n>N\ f(n)/g(n) \le C$	"≤"
$f(n) = o(g(n))$	$\lim_{n\to\infty} f(n)/g(n) = 0$	"<"
$f(n) = \Theta(g(n))$	$f(n) = O(g(n))$ *and* $g(n) = O(f(n))$	"="
$f(n) = \Omega(g(n))$	$g(n) = O(f(n))$	"≥"
$f(n) = \omega(g(n))$	$g(n) = o(f(n))$	">"

Convex Hull Algorithms

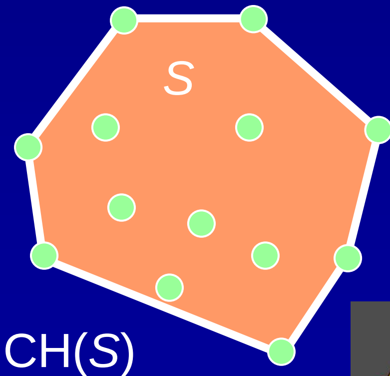
Convexity and Convex Hull

- Definition: A set S is *convex* if for any pair of points $p,q \in S$, the entire line segment $pq \subseteq S$.



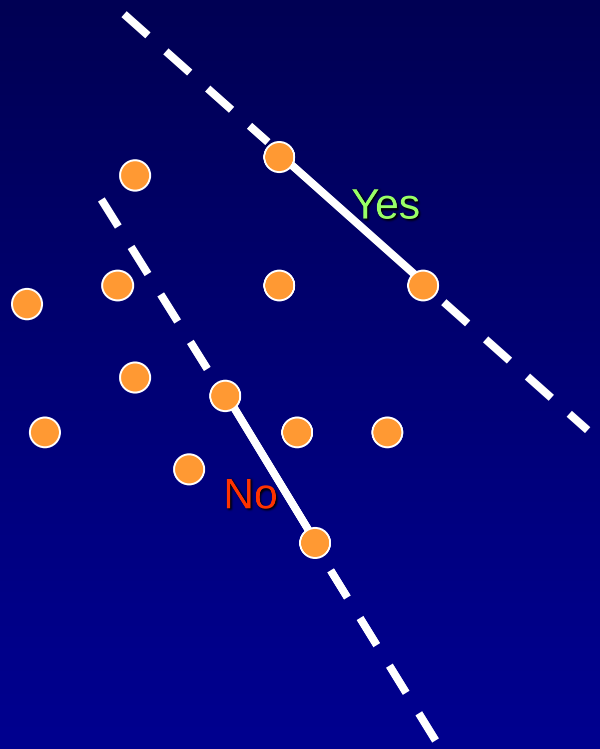
convex non-convex

- The *convex hull* (קְמוֹר) of a set S is the minimal convex set that contains S.
- Another (equivalent) definition: The intersection of **all** convex sets that contain S.
- **Question:** Why should the boundary of the convex hull of a point set be a polygon whose vertices are a subset of the points?



Convex Hull: Naive Algorithm

- Description:
  - For each pair of points construct its connecting segment and *supporting line*.
  - Find all the segments whose supporting lines divide the plane into two halves, such that one half plane contains *all* the other points.
  - Construct the convex hull out of these segments.

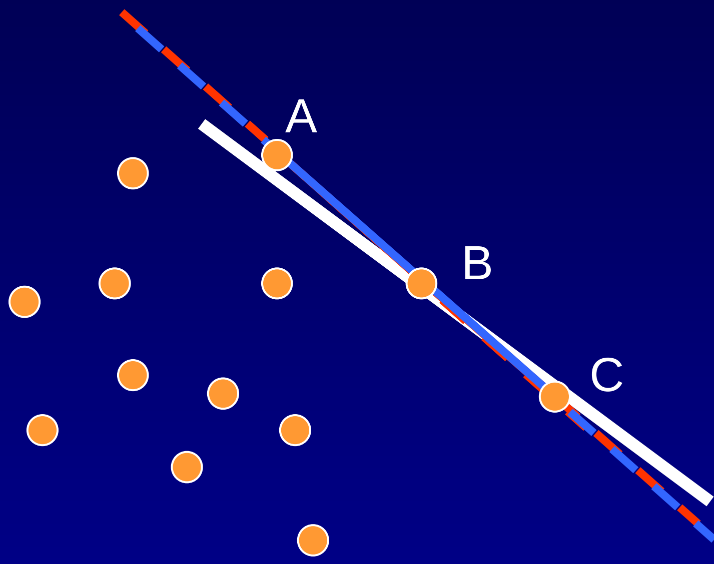


- Time complexity (for *n* points):
  - Number of point pairs: $$\binom{n}{2} = \Theta(n^2)$$
  - Check all points for each pair: $\Theta(n)$
  - Total: $\Theta(n^3)$
- Space complexity: $\Theta(n)$

Possible Pitfalls

- Degenerate cases, e.g., 3 collinear points, may harm the correctness of the algorithm. All, or none, of the segments AB, BC and AC will be included in the convex hull.

  Question: How can we solve the problem?

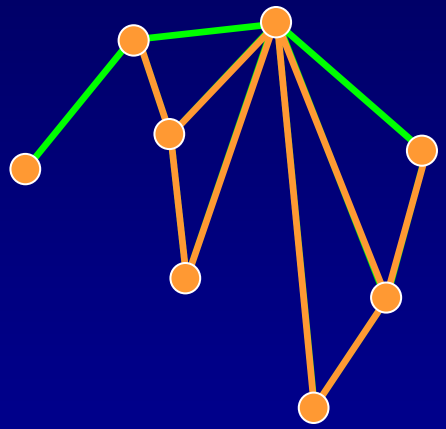


- Numerical problems: We might conclude that *none* of the three segments (or a wrong pair of them) belongs to the convex hull.

- **Question:** How is collinearity detected?

Convex Hull: Graham's Scan

- Algorithm:
  - Sort the points according to their *x* coordinates.
  - Construct the upper boundary by scanning the points in the sorted order and performing only "right turns" (trim off "left turns").
  - Construct the lower boundary in the same manner.
  - Concatenate the two boundaries.
- Time Complexity: O(*n* log *n*) (only!)
- May be implemented using a stack



- **Question:** How do we check for a "right turn"?

The Algorithm

- Input: Point set $\{p_i\}$.
- Sort the points in increasing order of x coordinates:

  $p_1, \ldots, p_n$.
- Push(S,p_1); Push(S,p_2);
- For i = 3 to n do
  - While Size(S) $\geq$ 2 and Orient(p_i,top(S),second(S)) $\leq$ 0 do Pop(S);
  - Push(S,p_i);
- Output S.

Graham's Scan: Time Complexity

- Sorting: O(n log n)
- If D_i is the number of points popped on processing p_i,

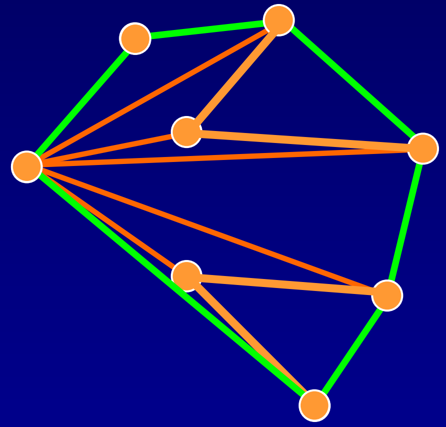
$$\text{time} = \sum_{i=1}^{n} (D_i + 1) = n + \sum_{i=1}^{n} D_i$$

- Naively, the last term can be quadratic in n; But…
- Each point is pushed on the stack only **once.**
- Once a point is popped, it cannot be popped again.
- Hence, $\sum_{i=1}^{n} D_i \le n$.

Graham's Scan: Rotational Variant

- Algorithm:
  - Find a point, p_0, which **must** be on the convex hull (e.g., the leftmost point).
  - Sort the other points by the *angle* of the rays shot to them from p_0.

    Question: Is it necessary to compute the actual angles? If not, how can we sort?
  - Construct the convex hull using one traversal of the points.
- Time Complexity: $O(n \log n)$

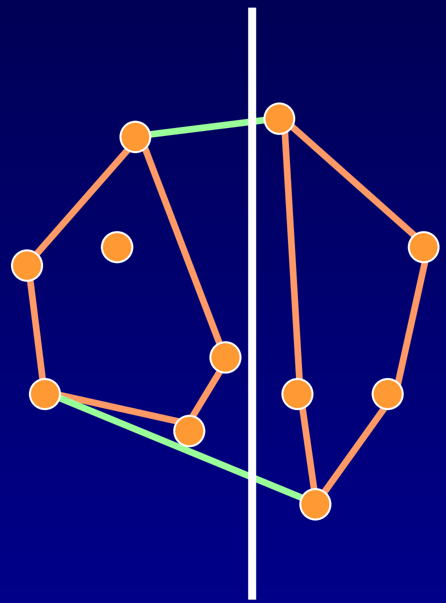


- **Question:** What are the pros and cons of this algorithm relative to the previous one?

Convex Hull: Divide and Conquer

- Algorithm:
  - Find a point with a median x coordinate (time: O(n))
  - Compute the convex hull of each half (recursive execution)
  - Combine the two convex hulls by finding common tangents.

    Question: How can this be done in ***O(n)*** time?



- Time Complexity:

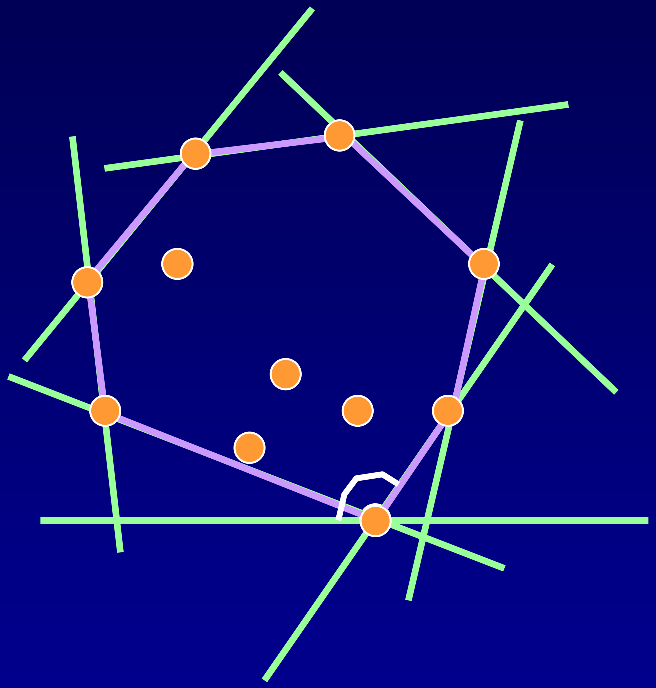
  O(n log n)

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Convex Hull: Gift Wrapping

- Algorithm:
  - Find a point p_1 on the convex hull (e.g., the lowest point).
  - Rotate counterclockwise a line through p_1 until it touches one of the other points (start from a horizontal orientation).

    Question: How is this done?



  - Repeat the last step for the new point.
  - Stop when p_1 is reached again.
- Time Complexity: $O(nh)$, where n is the input size and h is the output (hull) size.
- Since $3 \leq h \leq n$, time is $\Omega(n)$ and $O(n^2)$.

General Position

- When designing a geometric algorithm, we first make some simplifying assumptions (that depend on the problem and on the algorithm!), e.g.:
  - No 3 collinear points;
  - No two points with the same *x* coordinate.
- Later, we consider the general case:
  - How should the algorithm react to degenerate cases?
  - Will the correctness be preserved?
  - Will the running time remain the same?

Lower Bound for Convex Hull

- A reduction from Sorting to convex hull:
  - Given n real values x_i, generate n points on the graph of a convex function, e.g., a parabola, (x_i, x_i^2).
  - Compute the polygon C, the convex hull of the points.
  - The order of the points on C is the same order as that of the x_i.
- Hence, Complexity(CH)=$\Omega(n \log n)$
- Due to the existence of $O(n \log n)$-time algorithms, Complexity(CH)= $\Theta(n \log n)$