

Computational Geometry

Introduction and Convex Hull

Thomas Bläsius

What Is Computational Geometry?

Wikipedia

- Computational geometry is a branch of computer science devoted to the study of algorithms which can be stated in terms of geometry.

What Is Computational Geometry?

Wikipedia

- Computational geometry is a branch of computer science devoted to the study of algorithms which can be stated in terms of geometry.
- Some purely geometrical problems arise out of the study of computational geometric algorithms, and such problems are also considered to be part of computational geometry.

What Is Computational Geometry?

Wikipedia

- Computational geometry is a branch of computer science devoted to the study of algorithms which can be stated in terms of geometry.
- Some purely geometrical problems arise out of the study of computational geometric algorithms, and such problems are also considered to be part of computational geometry.

The Things We Deal With

- points, lines, line segments, circles, polygons, . . .

What Is Computational Geometry?

Wikipedia

- Computational geometry is a branch of computer science devoted to the study of algorithms which can be stated in terms of geometry.
- Some purely geometrical problems arise out of the study of computational geometric algorithms, and such problems are also considered to be part of computational geometry.

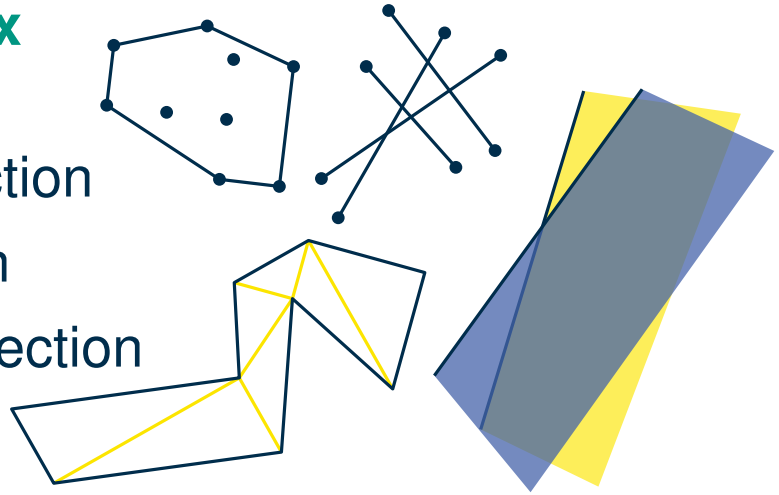
The Things We Deal With

- points, lines, line segments, circles, polygons, . . .
- but not: pixels

What Does That Mean Specifically?

Basic Toolbox

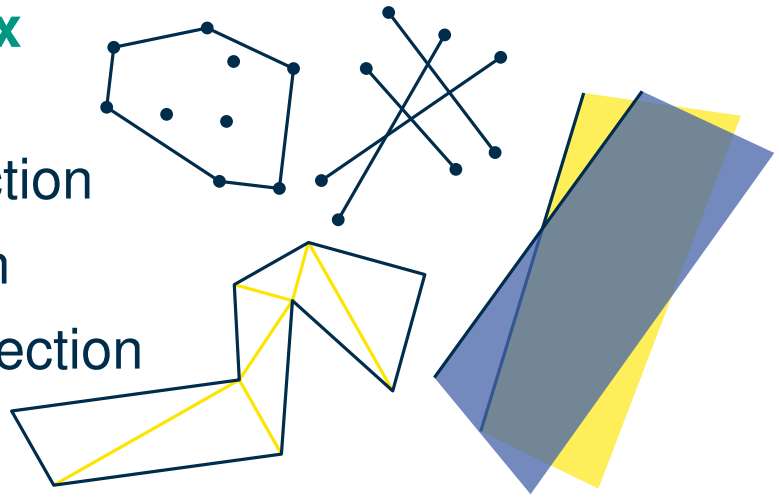
- convex hull
- line intersection
- triangulation
- plane intersection



What Does That Mean Specifically?

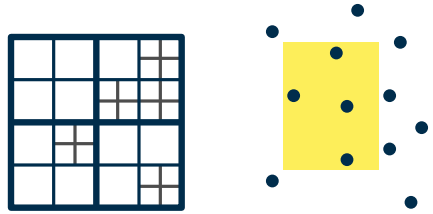
Basic Toolbox

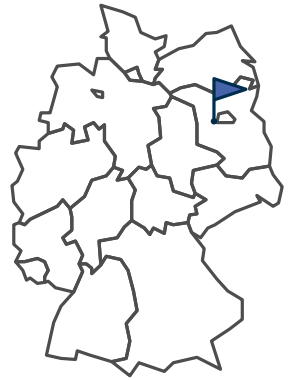
- convex hull
- line intersection
- triangulation
- plane intersection



Geometric Data Structures

- orthogonal range searching
- space partitioning
- point location

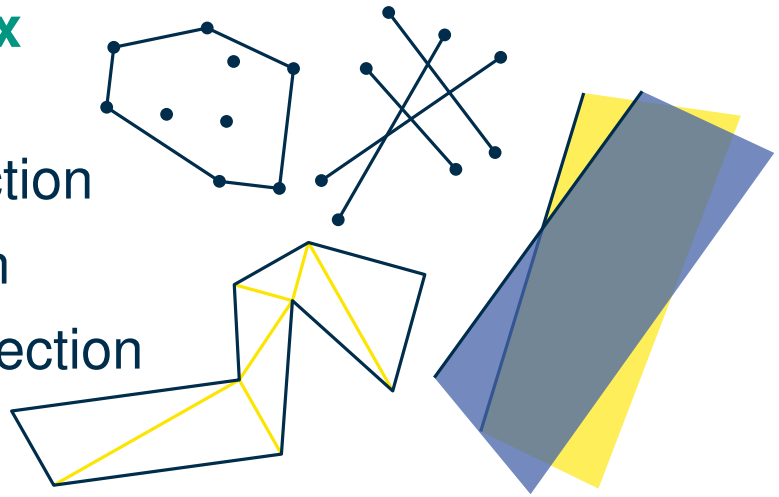




What Does That Mean Specifically?

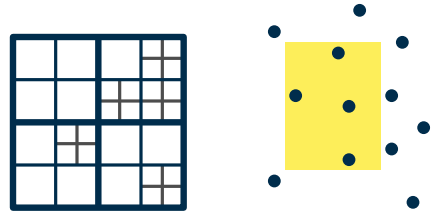
Basic Toolbox

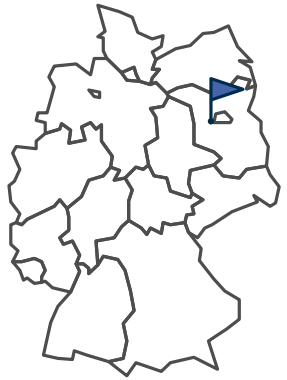
- convex hull
- line intersection
- triangulation
- plane intersection



Geometric Data Structures

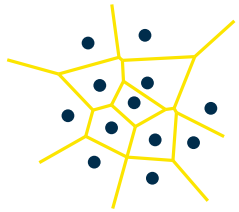
- orthogonal range searching
- space partitioning
- point location

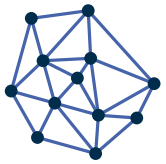


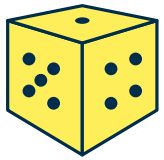


Advanced Toolbox

- Voronoi diagrams
- Delaunay triangulations
- randomized algorithms
- complexity



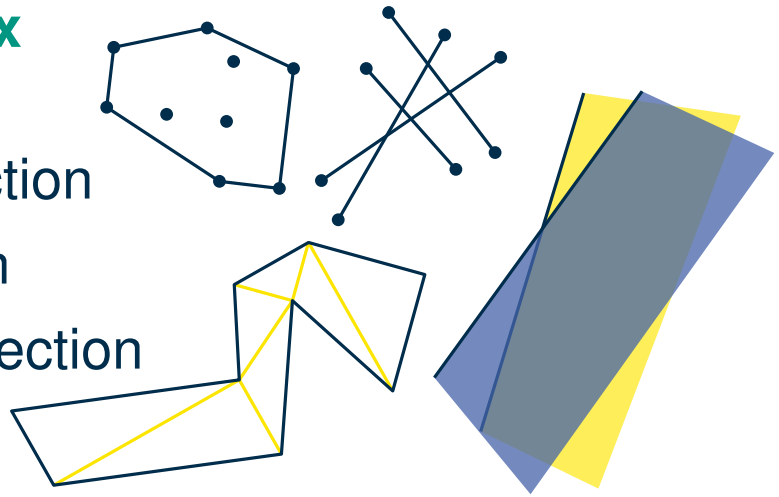




What Does That Mean Specifically?

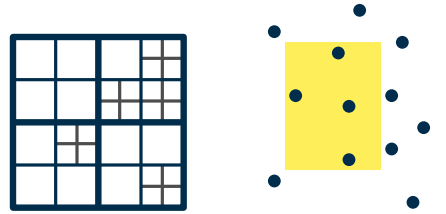
Basic Toolbox

- convex hull
- line intersection
- triangulation
- plane intersection



Geometric Data Structures

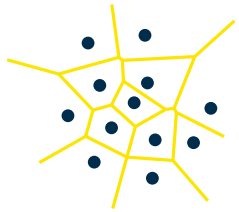
- orthogonal range searching
- space partitioning
- point location

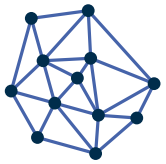


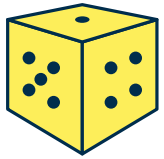


Advanced Toolbox

- Voronoi diagrams
- Delaunay triangulations
- randomized algorithms
- complexity

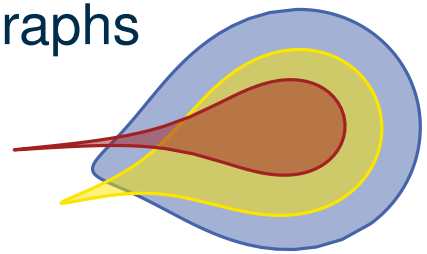


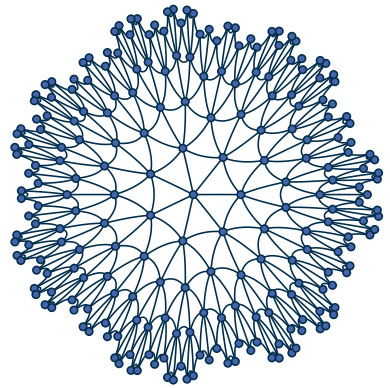




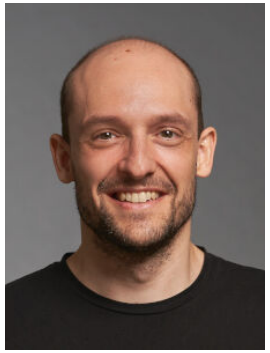
Related Topics

- What is geometry?
- hyperbolic geometry
- geometric graphs





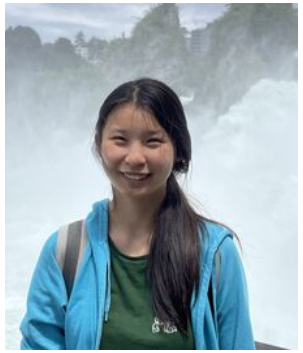
Before We Start



Thomas

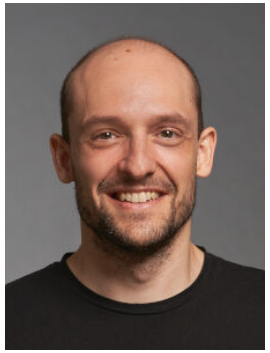


Elly



Wendy

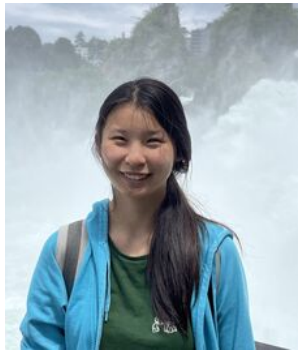
Before We Start



Thomas



Elly

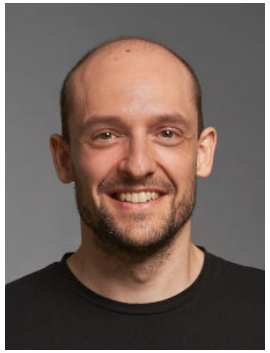


Wendy

?

You

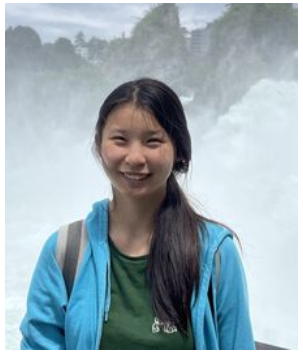
Before We Start



Thomas



Elly



Wendy

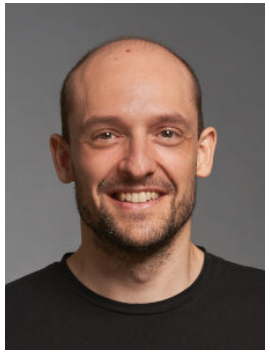


You

Materials & Infos

- slides, exercise sheets on our homepage: `https://scale.iti.kit.edu/teaching/2026ws/comput_geom/`

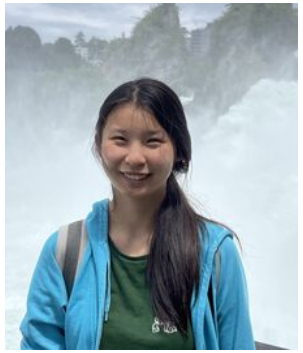
Before We Start



Thomas



Elly



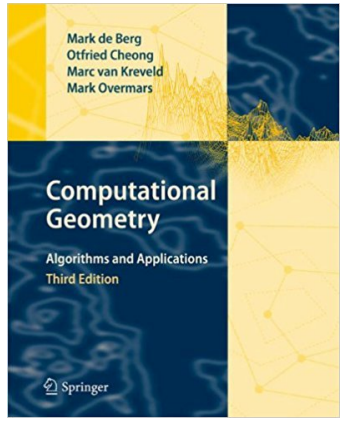
Wendy



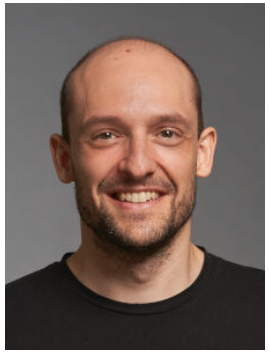
You

Materials & Infos

- slides, exercise sheets on our homepage: `https://scale.iti.kit.edu/teaching/2026ws/comput_geom/`
- Book: Computational Geometry



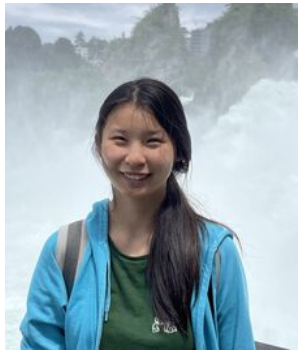
Before We Start



Thomas



Elly



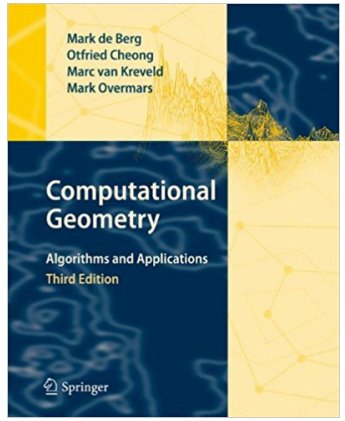
Wendy



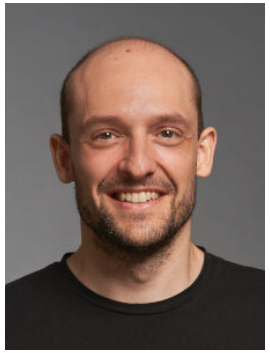
You

Materials & Infos

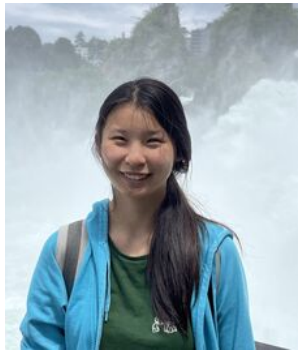
- slides, exercise sheets on our homepage: `https://scale.iti.kit.edu/teaching/2026ws/comput_geom/`
- Book: Computational Geometry
- Discord: `https://discord.gg/6p3nrfUb8S` (or if you are already on our server: send `!help join` to the scale-bot)



Before We Start


Thomas

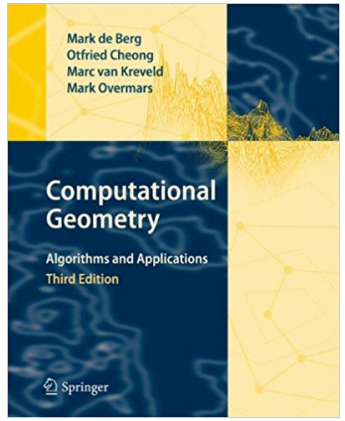

Elly


Wendy


You

Materials & Infos

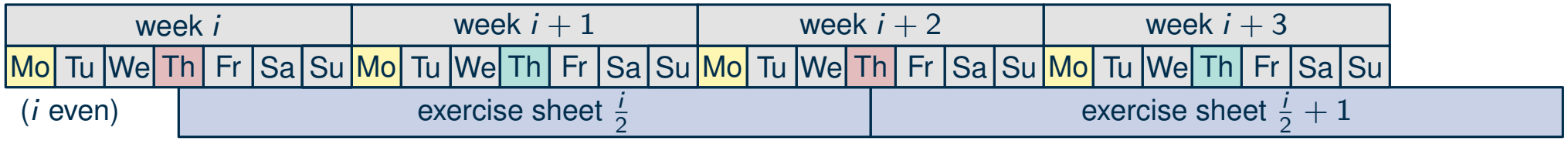
- slides, exercise sheets on our homepage: `https://scale.iti.kit.edu/teaching/2026ws/comput_geom/`
- Book: Computational Geometry
- Discord: `https://discord.gg/6p3nrfUb8S` (or if you are already on our server: send `!help join` to the scale-bot)



Requirements

- good algorithmic understanding
- no (little) prior knowledge

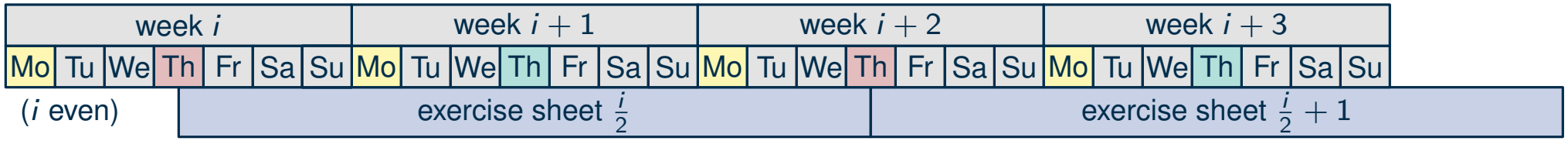
Rough Schedule



Lecture

- lecture with slides
- new topics

Rough Schedule



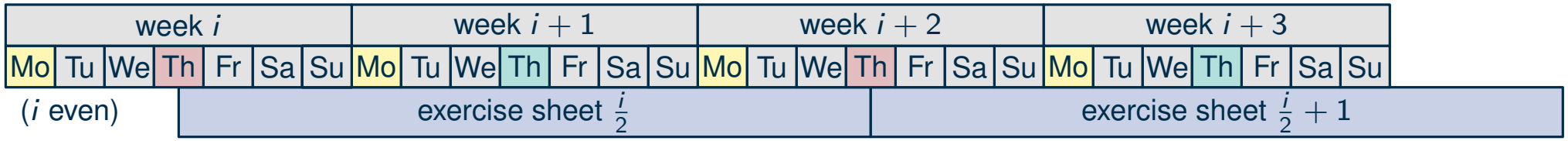
Lecture

- lecture with slides
- new topics

Exercise Sheet

- hand in in groups of two or three
- graded by us

Rough Schedule

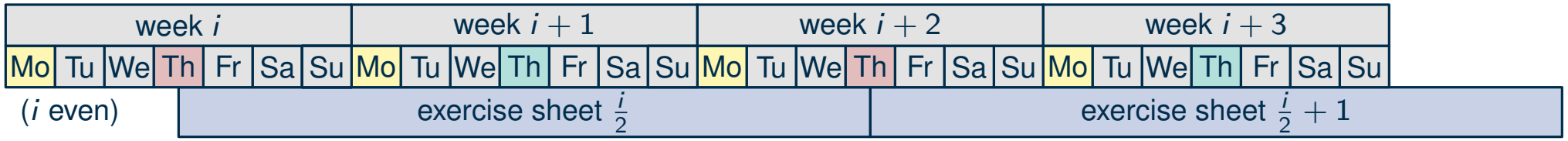


Lecture
- lecture with slides
- new topics

Exercise Sheet
- hand in in groups of two or three
- graded by us

Exercise Session (Week $i+1$)
- with Elly and Wendy
- recap
- support solving exercise sheets
- additional exercises

Rough Schedule



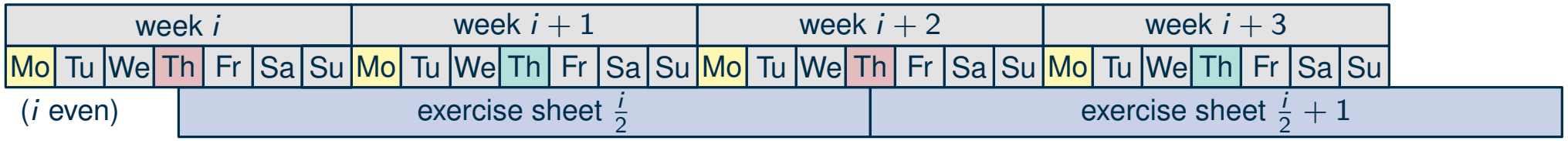
Lecture
- lecture with slides
- new topics

Exercise Sheet
- hand in in groups of two or three
- graded by us

Active Session
- train skills like: thinking, reading, explaining
- curiosities

Exercise Session (Week $i+1$)
- with Elly and Wendy
- recap
- support solving exercise sheets
- additional exercises

Rough Schedule



Lecture
- lecture with slides
- new topics

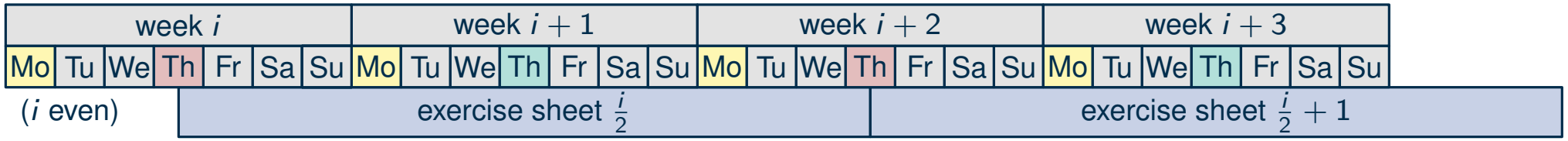
Exercise Sheet
- hand in in groups of two or three
- graded by us

Exam
- oral exam (20 min)
- admission only with exercise certificate

Active Session
- train skills like: thinking, reading, explaining
- curiosities

Exercise Session (Week $i+1$)
- with Elly and Wendy
- recap
- support solving exercise sheets
- additional exercises

Rough Schedule



Lecture
- lecture with slides
- new topics

Exercise Sheet
- hand in in groups of two or three
- graded by us

Exam
- oral exam (20 min)
- admission only with exercise certificate

Active Session
- train skills like: thinking, reading, explaining
- curiosities

Exercise Session (Week $i+1$)
- with Elly and Wendy
- recap
- support solving exercise sheets
- additional exercises

Two Thursday Slots
- one at 14:00
- another at 15:45
- try to split evenly

Exercise Certificate

Goal: $\frac{1}{2}$ of the points in total **and** $\frac{1}{4}$ on every exercise sheet

Exercise Certificate

Goal: $\frac{1}{2}$ of the points in total **and** $\frac{1}{4}$ on every exercise sheet

What If I Don't Find The Solution?

Exercise Certificate

Goal: $\frac{1}{2}$ of the points in total **and** $\frac{1}{4}$ on every exercise sheet

What If I Don't Find The Solution?

- you get points for explaining what you tried and why it did not work

Exercise Certificate

Goal: $\frac{1}{2}$ of the points in total **and** $\frac{1}{4}$ on every exercise sheet

What If I Don't Find The Solution?

- you get points for explaining what you tried and why it did not work
- and: there are many ways to get support
  - talk to your peers
  - ask in the exercise session, vial mail, or on discord

Exercise Certificate

Goal: $\frac{1}{2}$ of the points in total **and** $\frac{1}{4}$ on every exercise sheet

What If I Don't Find The Solution?

- you get points for explaining what you tried and why it did not work
- and: there are many ways to get support
  - talk to your peers
  - ask in the exercise session, vial mail, or on discord

What If I Can't Manage To Hand In An Exercise Sheet?

Exercise Certificate

Goal: $\frac{1}{2}$ of the points in total **and** $\frac{1}{4}$ on every exercise sheet

What If I Don't Find The Solution?

- you get points for explaining what you tried and why it did not work
- and: there are many ways to get support
  - talk to your peers
  - ask in the exercise session, vial mail, or on discord

What If I Can't Manage To Hand In An Exercise Sheet?

- sometimes, life can get in the way; talk to us, we'll find a solution
- do not hand in something AI generated!

we don't want to make your life hard and we also don't bite
we just want you to learn something and have fun doing so

Exercise Certificate

Goal: $\frac{1}{2}$ of the points in total **and** $\frac{1}{4}$ on every exercise sheet

What If I Don't Find The Solution?

- you get points for explaining what you tried and why it did not work
- and: there are many ways to get support
  - talk to your peers
  - ask in the exercise session, vial mail, or on discord

What If I Can't Manage To Hand In An Exercise Sheet?

- sometimes, life can get in the way; talk to us, we'll find a solution
- do not hand in something AI generated!

we don't want to make your life hard and we also don't bite
we just want you to learn something and have fun doing so

Our Goal

- you spend some time with the content of the lecture and write down your solution
- then, the exercise certificate should not be a big obstacle

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components A and B
- two springs:

	A	B
spring 1	35%	10%
spring 2	20%	16%

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components A and B
- two springs:

	A	B
spring 1	35%	10%
spring 2	20%	16%

Can we achieve 30% A and 12% B?

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components A and B
- two springs:

	A	B
spring 1	35%	10%
spring 2	20%	16%

Can we achieve 30% A and 12% B?

2 : 1

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components A and B
- two springs:

	A	B
spring 1	35%	10%
spring 2	20%	16%

Can we achieve 30% A and 12% B?

2 : 1

What about 22% A and 13% B?

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components A and B
- two springs:
- third spring:

	A	B
spring 1	35%	10%
spring 2	20%	16%
spring 3	15%	7%

Can we achieve 30% A and 12% B?

2 : 1

What about 22% A and 13% B?

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components A and B
- two springs:
- third spring:

	A	B
spring 1	35%	10%
spring 2	20%	16%
spring 3	15%	7%

Can we achieve 30% A and 12% B? 2 : 1

What about 22% A and 13% B? 1 : 3 : 1

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components A and B
- two springs:
- third spring:

	A	B
spring 1	35%	10%
spring 2	20%	16%
spring 3	15%	7%

Can we achieve 30% A and 12% B? 2 : 1

What about 22% A and 13% B? 1 : 3 : 1

What Is The Relation To Geometry?

Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components *A* and *B*
- two springs:

	A	*B*
spring 1	35%	10%
spring 2	20%	16%

- third spring:

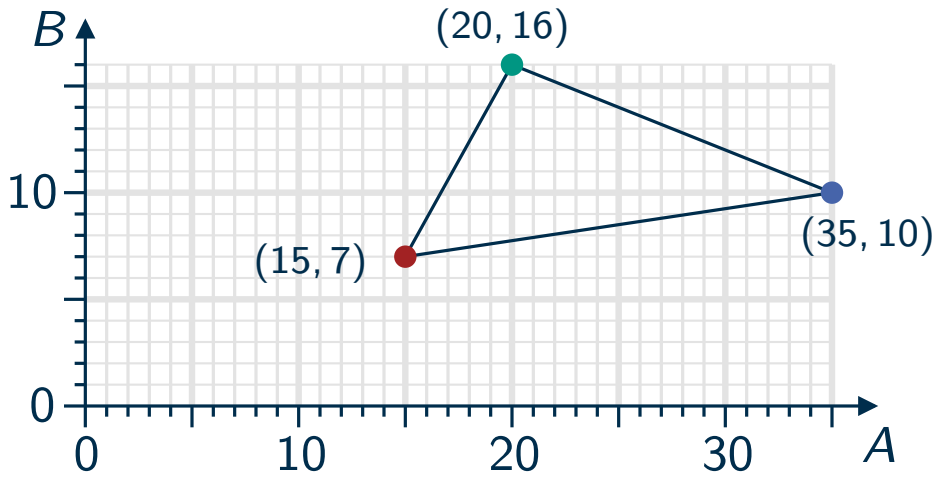
spring 3	15%	7%

What Is The Relation To Geometry?

- ratios can be interpreted as points

Can we achieve 30% *A* and 12% *B*? 2 : 1

What about 22% *A* and 13% *B*? 1 : 3 : 1



Motivation

Different Mixtures Of Oil

- the exact ratio between different components depends on the oil spring
- goal: mix oil from different springs, such that the result is easy to process

Example

- oil contains components *A* and *B*
- two springs:

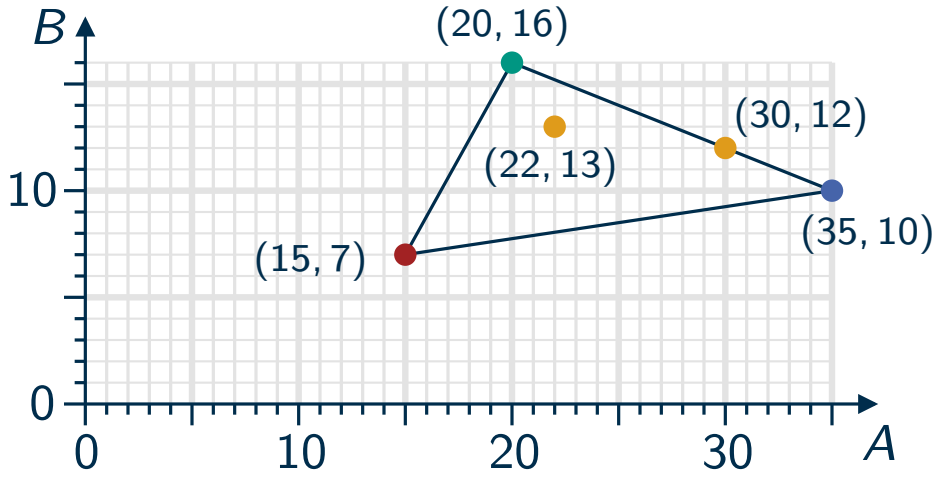
	A	*B*
spring 1	35%	10%
spring 2	20%	16%

- third spring:

spring 3	15%	7%

Can we achieve 30% *A* and 12% *B*? 2 : 1

What about 22% *A* and 13% *B*? 1 : 3 : 1



What Is The Relation To Geometry?

- ratios can be interpreted as points
- desired ratio is possible $\Leftrightarrow$ corresponding points lies “between” the available points

Convex Hull

Definition
A point set $P \subseteq \mathbb{R}^d$ is **convex** if for any two points $p, q \in P$, the line segment pq lies in P.

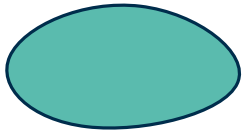


convex

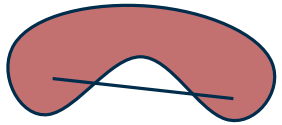
not convex

Convex Hull

Definition
A point set $P \subseteq \mathbb{R}^d$ is **convex** if for any two points $p, q \in P$, the line segment pq lies in P.

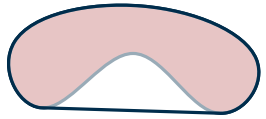


convex



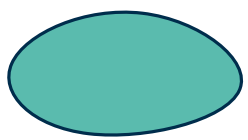
not convex

Definition
For $P \subseteq \mathbb{R}^d$, the **convex hull $\mathcal{CH}(P)$** is the minimal subset of $\mathbb{R}^d$ such that $\mathcal{CH}(P)$ is convex and $P \subseteq \mathcal{CH}(P)$.

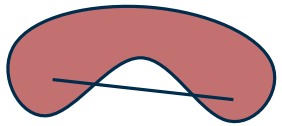


Convex Hull

Definition
A point set $P \subseteq \mathbb{R}^d$ is **convex** if for any two points $p, q \in P$, the line segment pq lies in P.

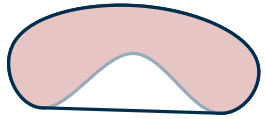


convex



not convex

Definition
For $P \subseteq \mathbb{R}^d$, the **convex hull $\mathcal{CH}(P)$** is the minimal subset of $\mathbb{R}^d$ such that $\mathcal{CH}(P)$ is convex and $P \subseteq \mathcal{CH}(P)$.

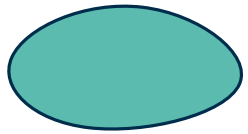


Equivalent Definitions

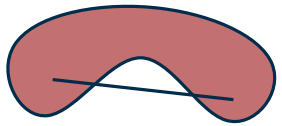
- intersection of all convex sets in $\mathbb{R}^d$ that contain P

Convex Hull

Definition
A point set $P \subseteq \mathbb{R}^d$ is **convex** if for any two points $p, q \in P$, the line segment pq lies in P.

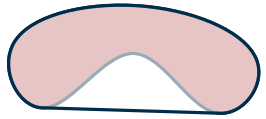


convex



not convex

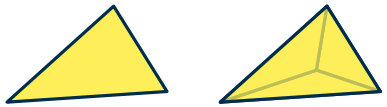
Definition
For $P \subseteq \mathbb{R}^d$, the **convex hull $\mathcal{CH}(P)$** is the minimal subset of $\mathbb{R}^d$ such that $\mathcal{CH}(P)$ is convex and $P \subseteq \mathcal{CH}(P)$.



Equivalent Definitions

- intersection of all convex sets in $\mathbb{R}^d$ that contain P
- union of all simplices with corners in P

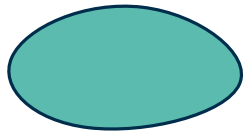
simplices in different dimensions:



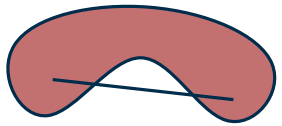
...

Convex Hull

Definition
A point set $P \subseteq \mathbb{R}^d$ is **convex** if for any two points $p, q \in P$, the line segment pq lies in P.

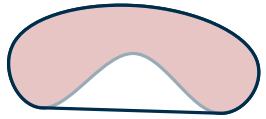


convex



not convex

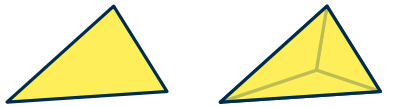
Definition
For $P \subseteq \mathbb{R}^d$, the **convex hull $\mathcal{CH}(P)$** is the minimal subset of $\mathbb{R}^d$ such that $\mathcal{CH}(P)$ is convex and $P \subseteq \mathcal{CH}(P)$.



Equivalent Definitions

- intersection of all convex sets in $\mathbb{R}^d$ that contain P
- union of all simplices with corners in P
- set of all points that are convex combinations of points in P

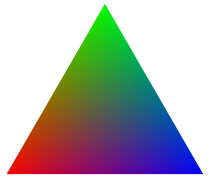
simplices in different dimensions:



...

convex combination: $\sum_{i=1}^{n} a_i \cdot p_i$ with $p_i \in P$, $a_i \in \mathbb{R}$, $a_i \geq 0$, and $\sum_{i=1}^{n} a_i = 1$

you might know this from the barycentric coordinate system



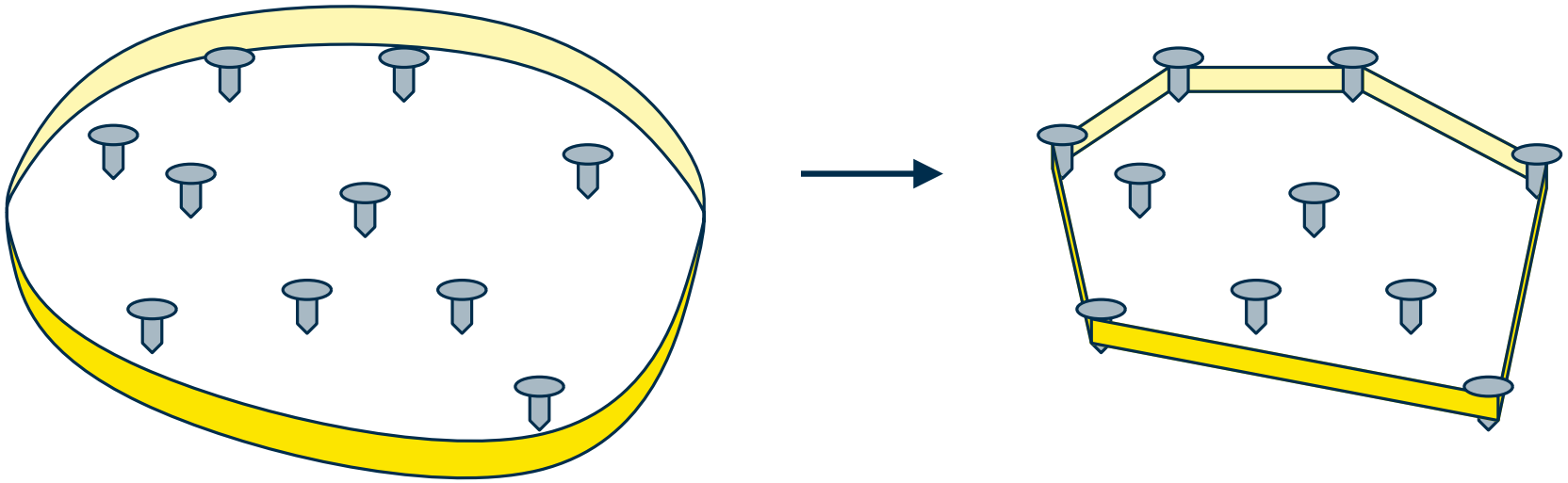
Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Intuition

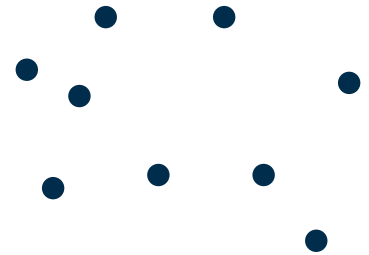


Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position

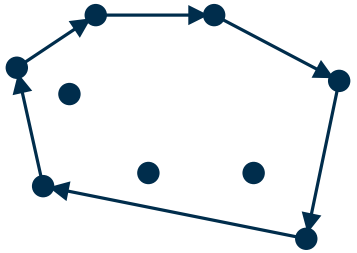


Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points

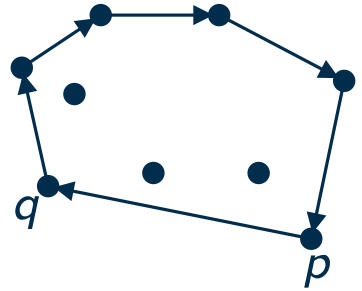


Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points



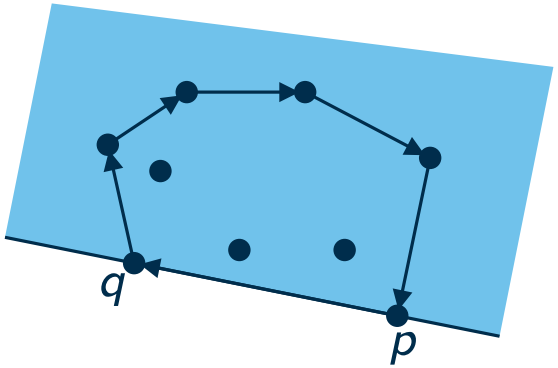
When is pq an edge of $\mathcal{CH}(P)$?

Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points
- pq edge of $\mathcal{CH}(P) \Leftrightarrow$ all points of P lie in the half space right of pq



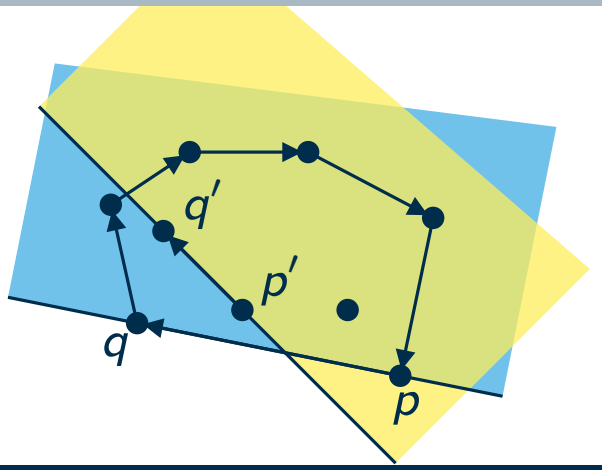
When is pq an edge of $\mathcal{CH}(P)$?

Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points
- pq edge of $\mathcal{CH}(P) \Leftrightarrow$ all points of P lie in the half space right of pq



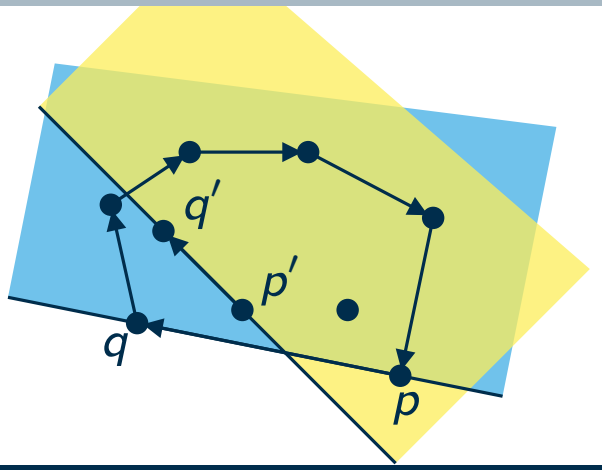
When is pq an edge of $\mathcal{CH}(P)$?

Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points
- pq edge of $\mathcal{CH}(P) \Leftrightarrow$ all points of P lie in the half space right of pq



Trivial Algorithm

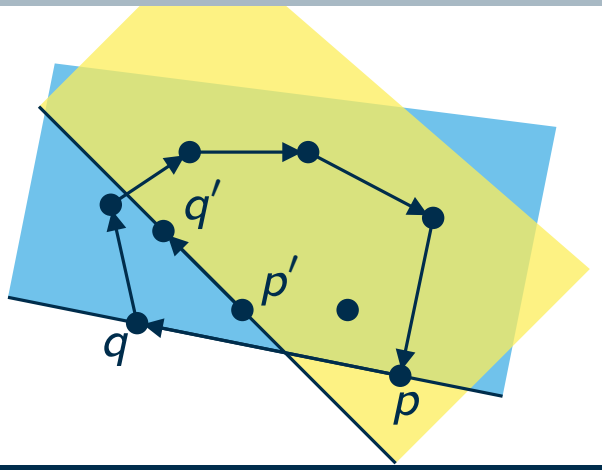
When is pq an edge of $\mathcal{CH}(P)$?

Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points
- pq edge of $\mathcal{CH}(P) \Leftrightarrow$ all points of P lie in the half space right of pq



When is pq an edge of $\mathcal{CH}(P)$?

Trivial Algorithm

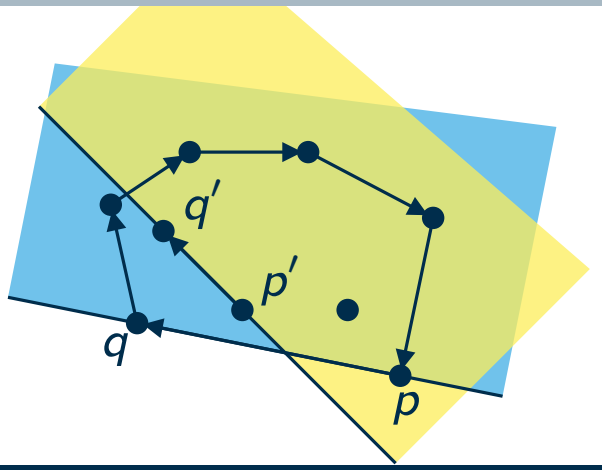
- iterate over all pairs of points $(p, q) \in P \times P$ (oriented)
  - check if all points of P lie to the right of pq
  - if yes: save the edge pq

Convex Hull – Trivial Algorithm

Convex Hull Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points
- pq edge of $\mathcal{CH}(P) \Leftrightarrow$ all points of P lie in the half space right of pq



When is pq an edge of $\mathcal{CH}(P)$?

Trivial Algorithm

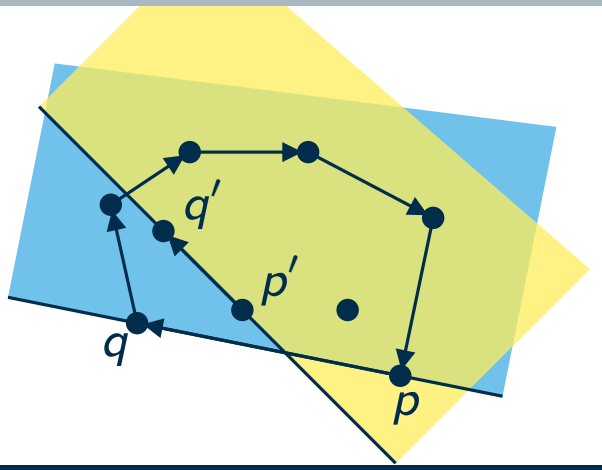
- iterate over all pairs of points $(p, q) \in P \times P$ (oriented)
  - check if all points of P lie to the right of pq
  - if yes: save the edge pq
- construct the polygon (as sequence of points) from the saved edges

Convex Hull – Trivial Algorithm

CONVEX HULL Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points
- pq edge of $\mathcal{CH}(P) \Leftrightarrow$ all points of P lie in the half space right of pq



When is pq an edge of $\mathcal{CH}(P)$?

Trivial Algorithm

- iterate over all pairs of points $(p, q) \in P \times P$ (oriented)
  - check if all points of P lie to the right of pq
  - if yes: save the edge pq
- construct the polygon (as sequence of points) from the saved edges

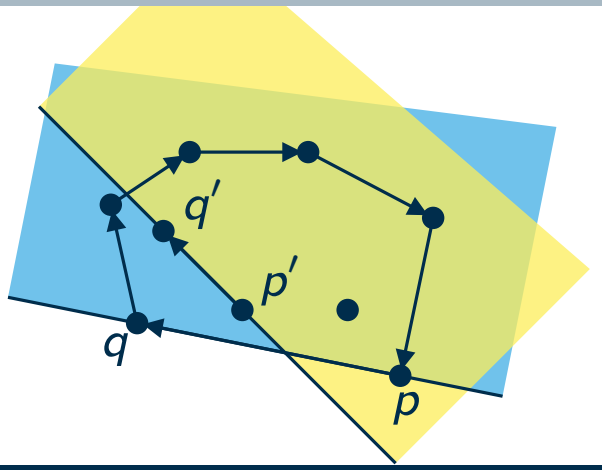
Running Time:

Convex Hull – Trivial Algorithm

Convex Hull Problem (2D): Given n points $P \subseteq \mathbb{R}^2$, compute the convex hull $\mathcal{CH}(P)$.

Notes And General Observations

- assumption: points are in general position
- boundary of $\mathcal{CH}(P)$ is a polygon $\rightarrow$ output is a sequence of points
- pq edge of $\mathcal{CH}(P) \Leftrightarrow$ all points of P lie in the half space right of pq



When is pq an edge of $\mathcal{CH}(P)$?

Trivial Algorithm

- iterate over all pairs of points $(p, q) \in P \times P$ (oriented)
  - check if all points of P lie to the right of pq
  - if yes: save the edge pq
- construct the polygon (as sequence of points) from the saved edges

Running Time: $\Theta(n^3)$

Convex Hull – Trivial Algorithm

Trivial Algorithm

- iterate over all pairs of points $(p, q) \in P \times P$ (oriented)
  - check if all points of P lie to the right of pq
  - if yes: save the edge pq
- construct polygon (sequence of points) from the saved edges

Problems

- the algorithm is slow

Convex Hull – Trivial Algorithm

Trivial Algorithm

- iterate over all pairs of points $(p, q) \in P \times P$ (oriented)
  - check if all points of P lie to the right of pq
  - if yes: save the edge pq
- construct polygon (sequence of points) from the saved edges

Problems

- the algorithm is slow
- the algorithm is not robust

Convex Hull – Trivial Algorithm

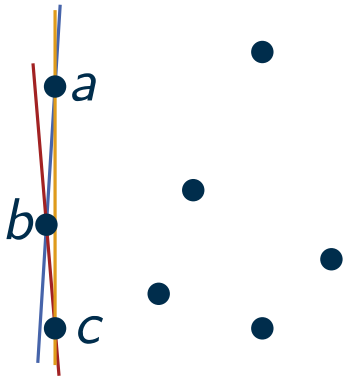
Trivial Algorithm

- iterate over all pairs of points $(p, q) \in P \times P$ (oriented)
  - check if all points of P lie to the right of pq
  - if yes: save the edge pq
- construct polygon (sequence of points) from the saved edges

Problems

- the algorithm is slow
- the algorithm is not robust

Example For Lacking Robustness



- three decisions “lies to the right of” are close

Convex Hull – Trivial Algorithm

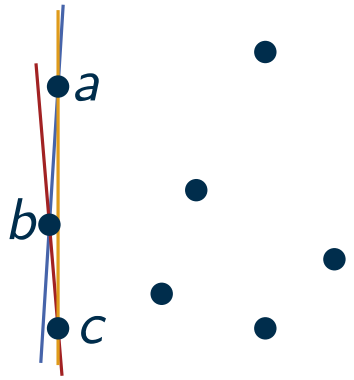
Trivial Algorithm

- iterate over all pairs of points $(p, q) \in P \times P$ (oriented)
  - check if all points of P lie to the right of pq
  - if yes: save the edge pq
- construct polygon (sequence of points) from the saved edges

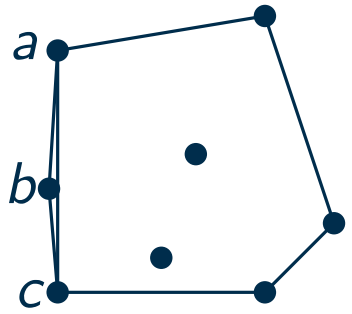
Problems

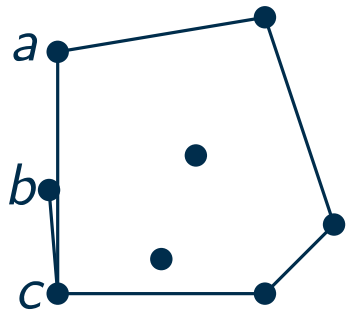
- the algorithm is slow
- the algorithm is not robust

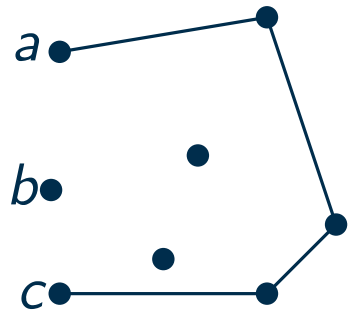
Example For Lacking Robustness



- three decisions “lies to the right of” are close
- wrong decision $\rightarrow$ output maybe not a polygon







Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

Idea: Iterative Approach

- add points one after another
- update convex hull in each step

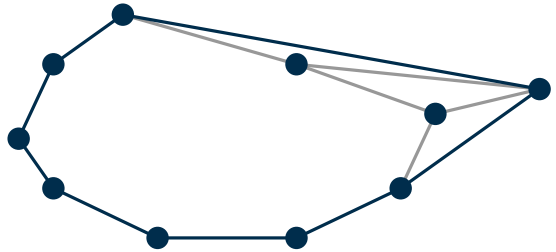


Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

Idea: Iterative Approach

- add points one after another
- update convex hull in each step

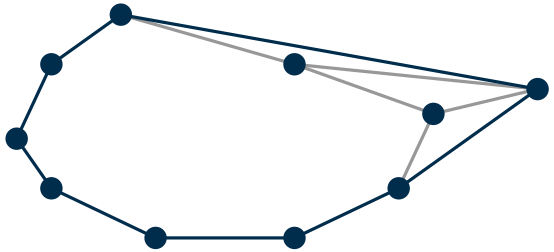


Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns

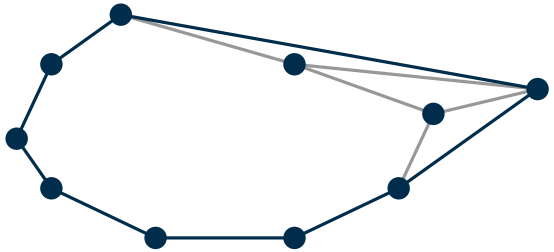


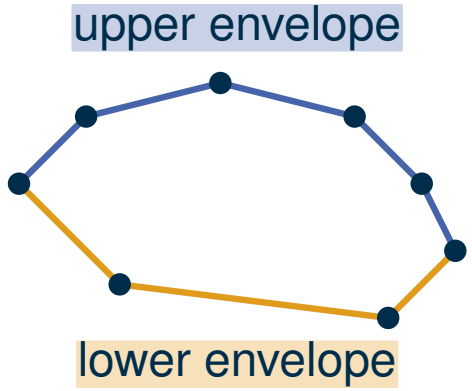
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





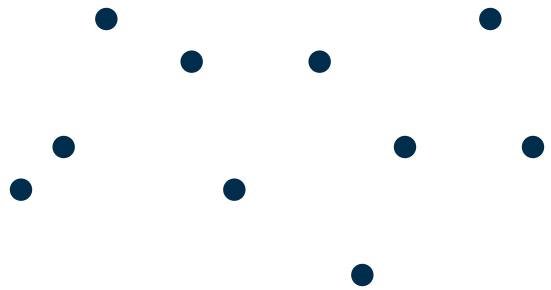
Andrews Monotone Chain Algorithm

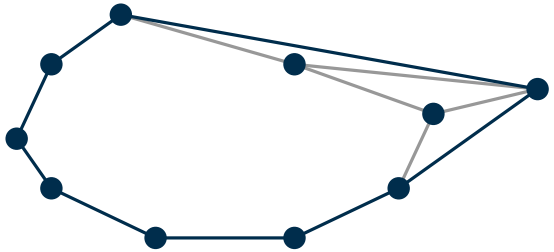
(variant of the Graham Scan)

Idea: Iterative Approach

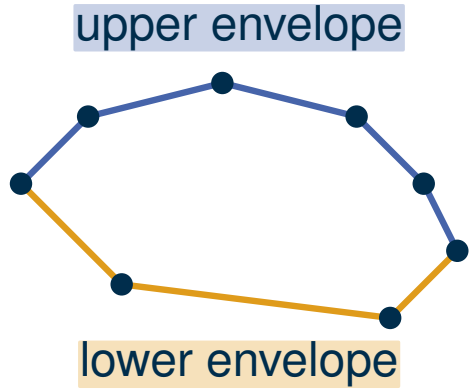
- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope

Example





Andrews Algorithm



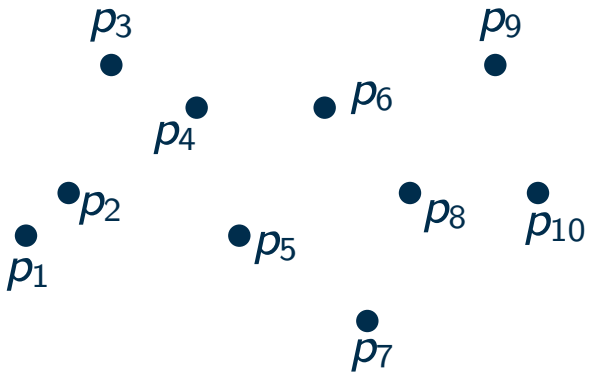
Andrews Monotone Chain Algorithm

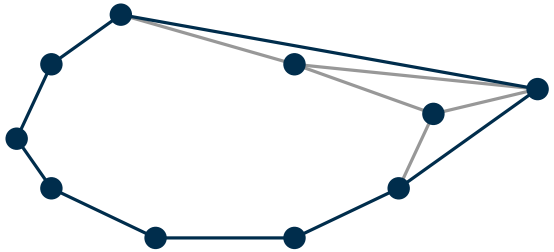
(variant of the Graham Scan)

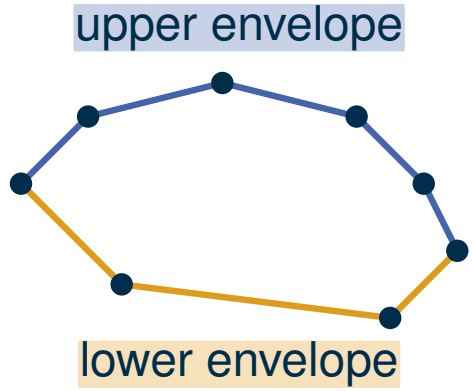
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope

Example







Andrews Algorithm

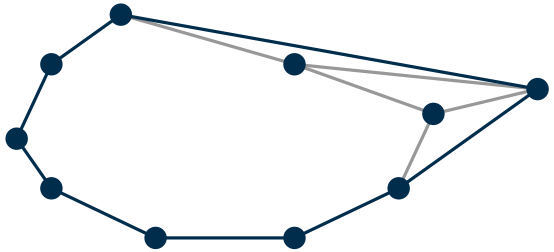
- sort P (left to right): $p_1, \ldots, p_n$

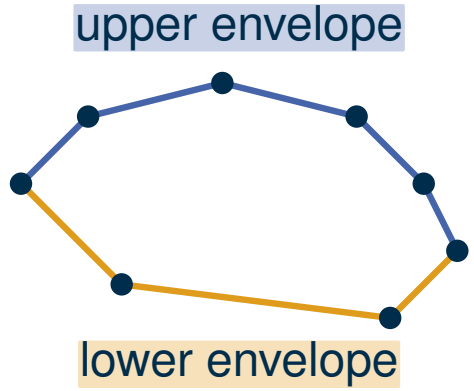
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

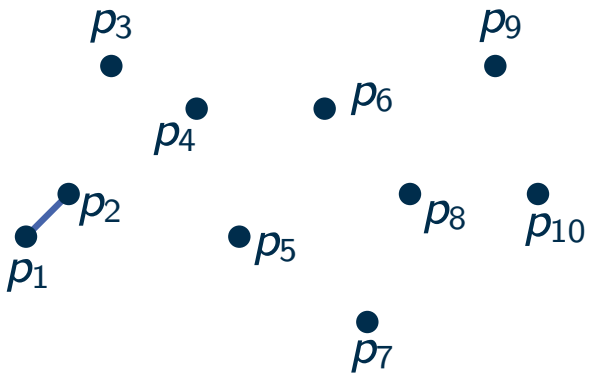
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_2

Andrews Algorithm

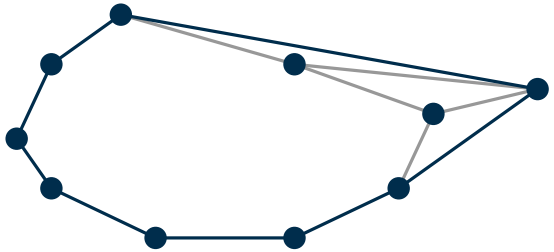
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L

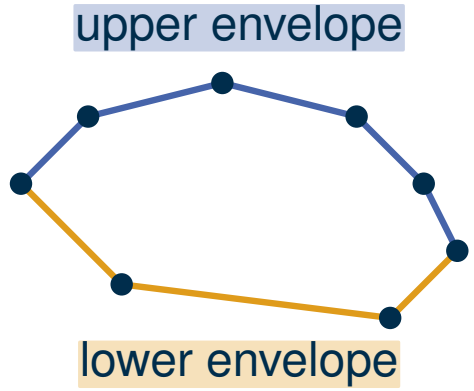
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

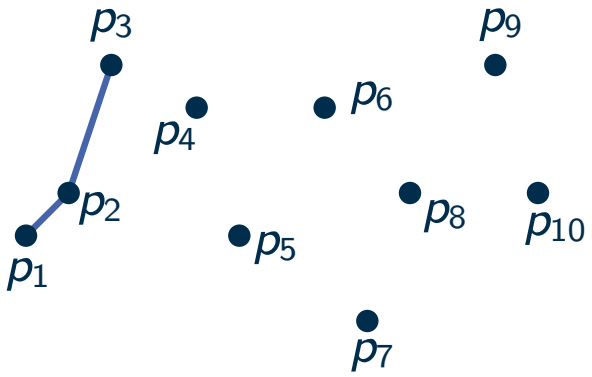
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_2 p_3

Andrews Algorithm

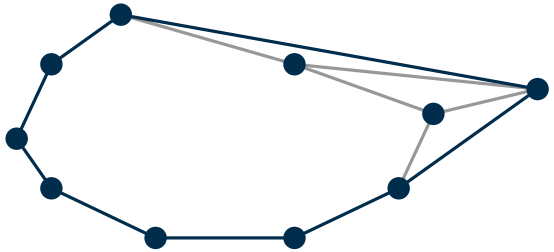
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L

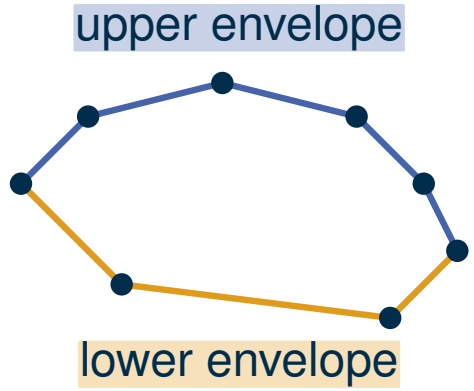
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

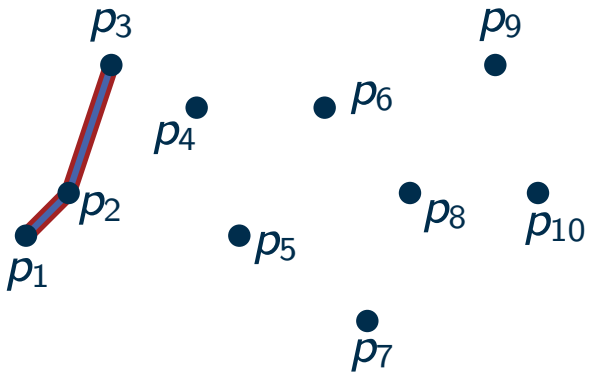
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_2 p_3

Andrews Algorithm

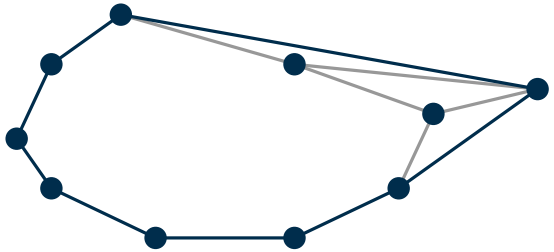
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

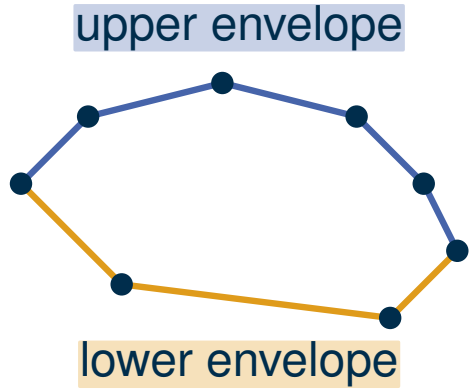
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

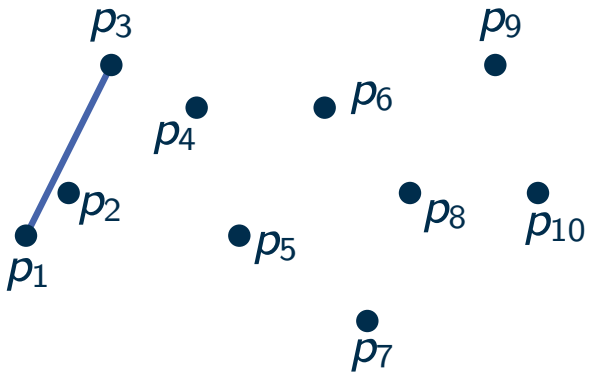
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3

Andrews Algorithm

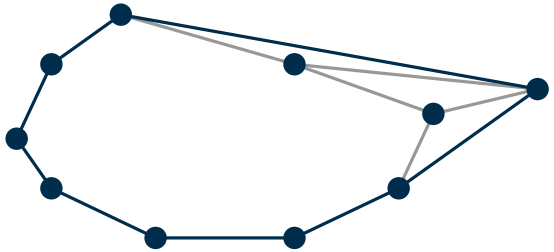
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

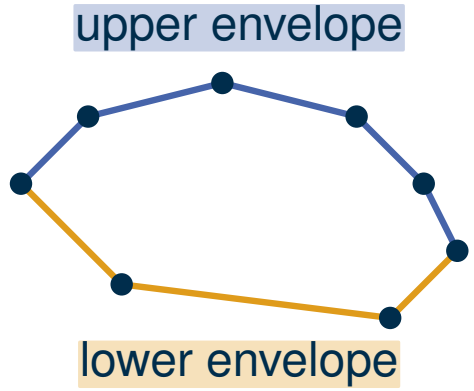
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

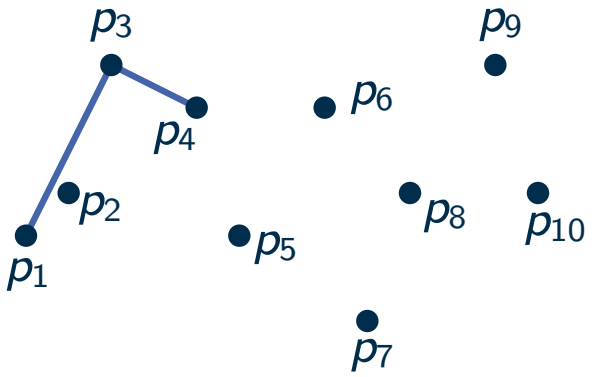
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_4

Andrews Algorithm

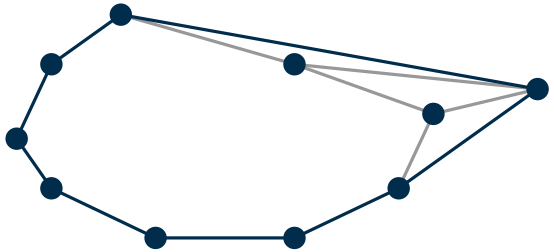
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

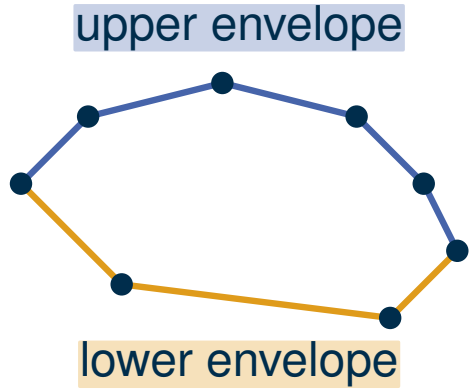
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

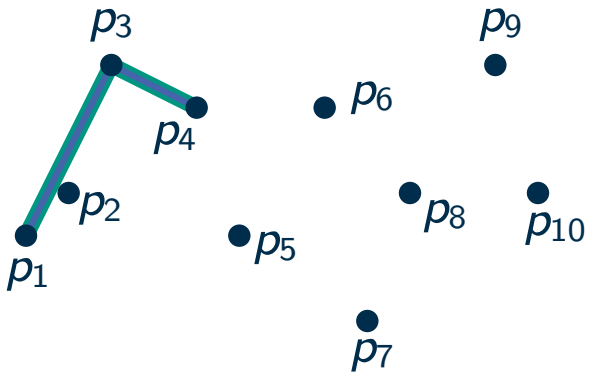
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_4

Andrews Algorithm

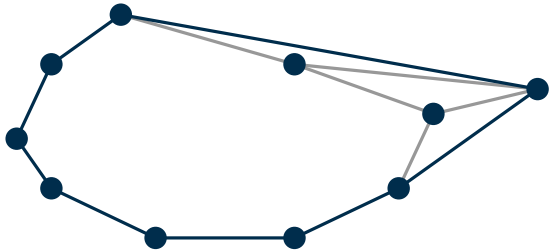
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

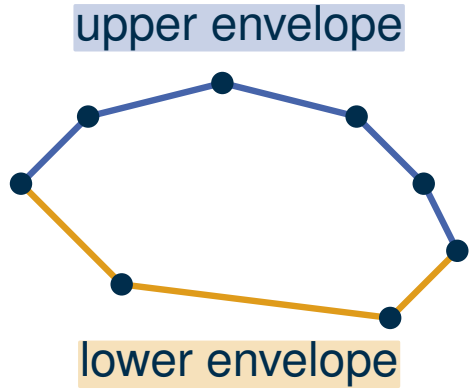
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

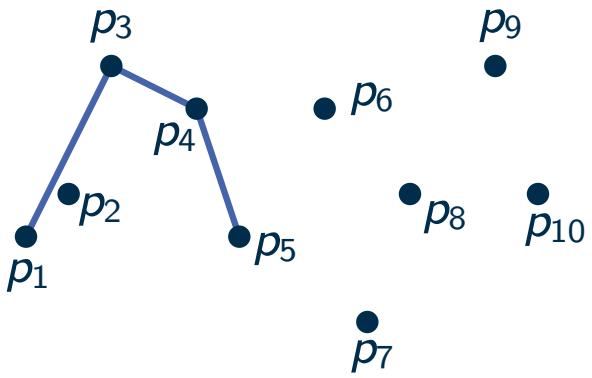
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_4 p_5

Andrews Algorithm

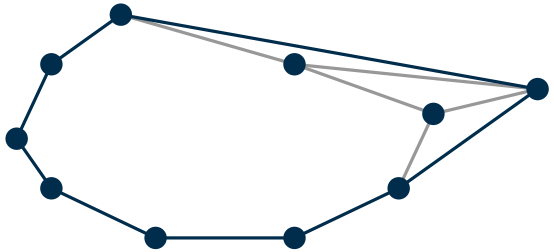
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

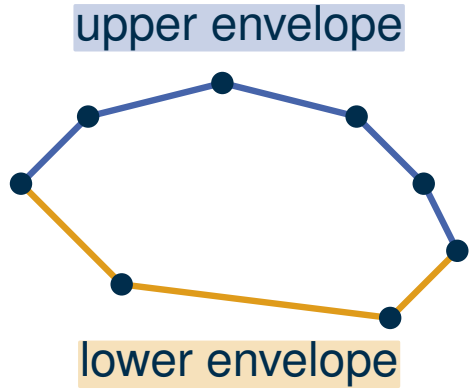
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

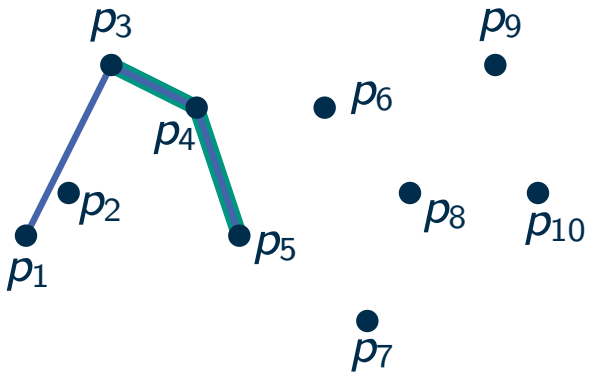
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_4 p_5

Andrews Algorithm

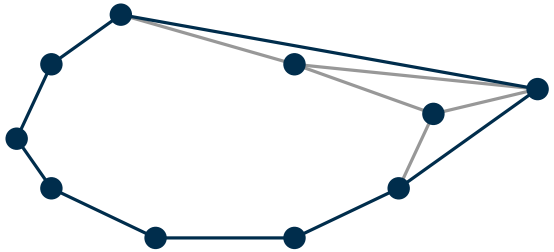
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

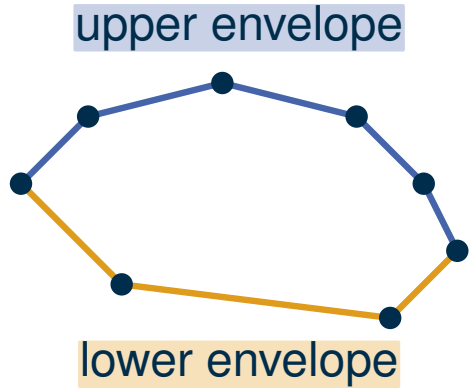
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

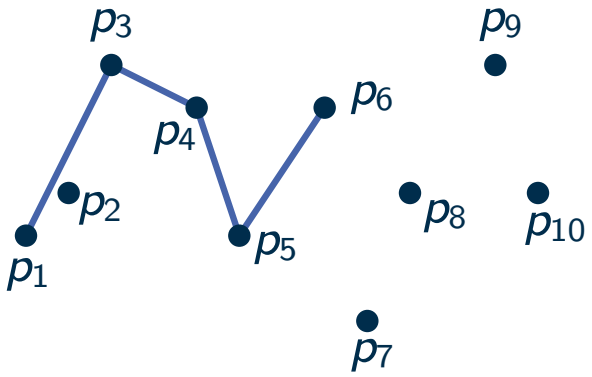
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_4 p_5 p_6

Andrews Algorithm

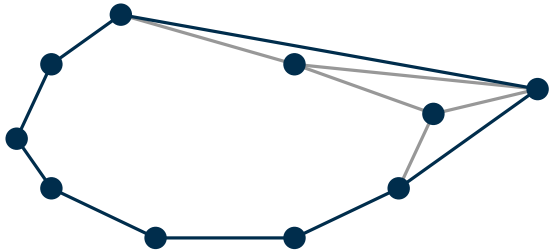
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

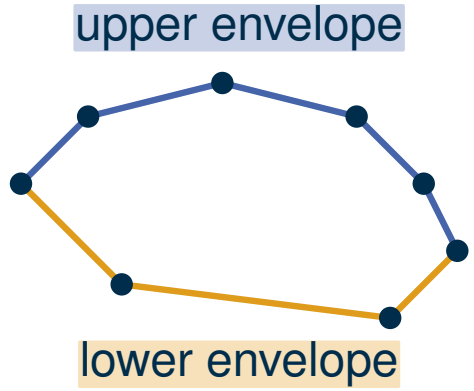
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

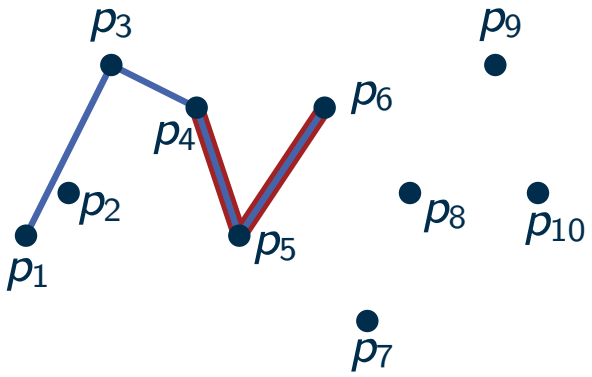
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_4 p_5 p_6

Andrews Algorithm

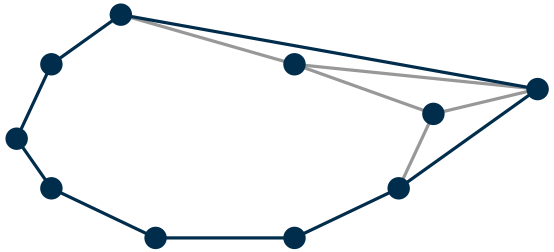
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

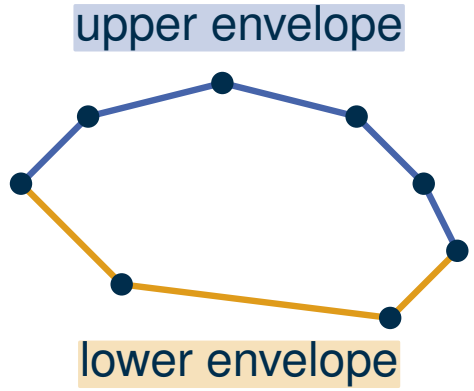
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

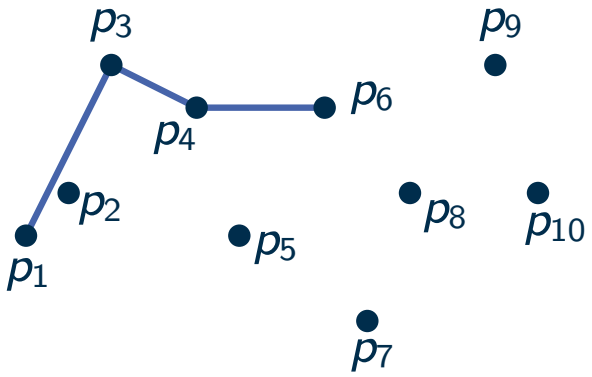
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_4 p_6

Andrews Algorithm

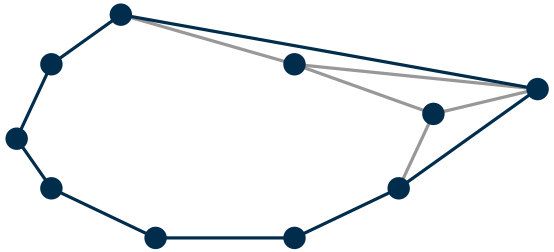
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

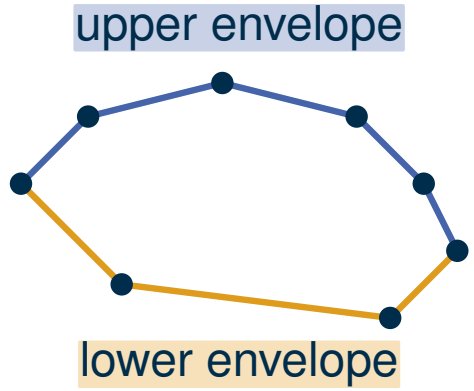
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

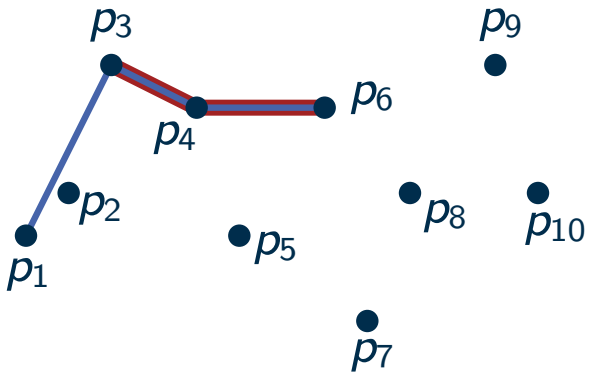
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_4 p_6

Andrews Algorithm

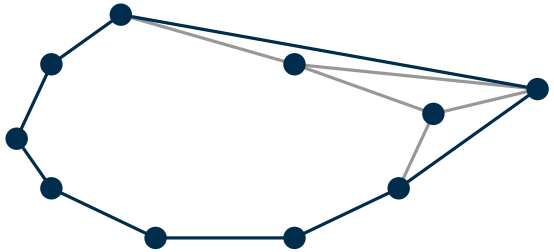
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

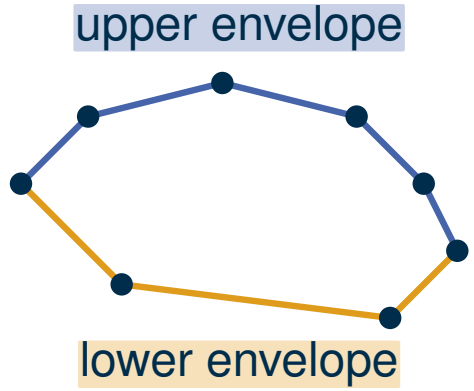
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

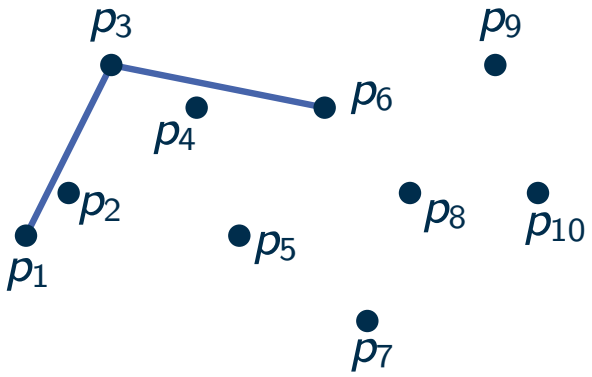
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6

Andrews Algorithm

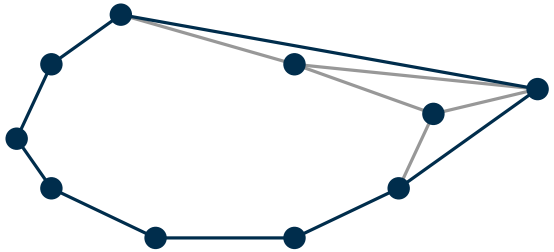
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

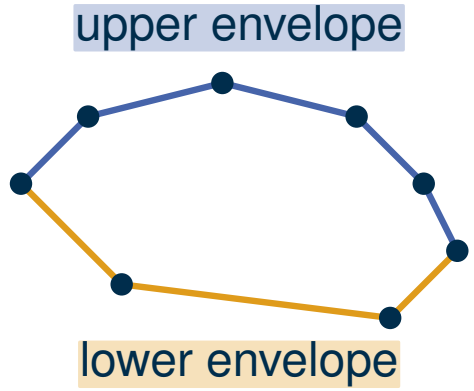
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

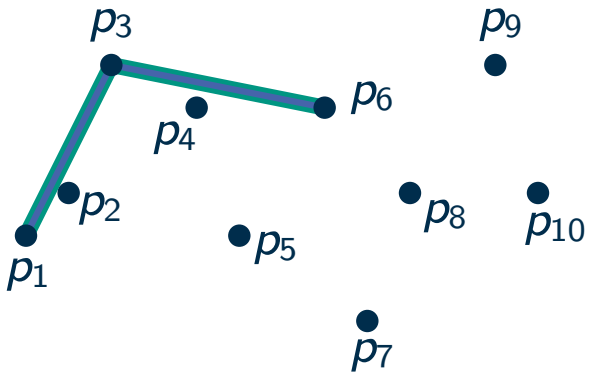
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6

Andrews Algorithm

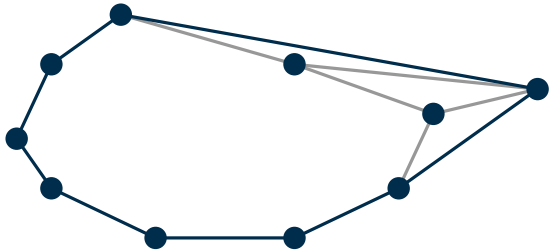
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

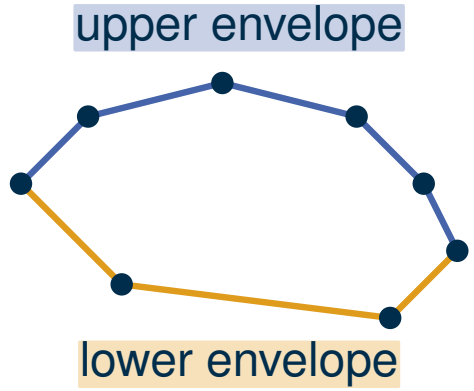
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

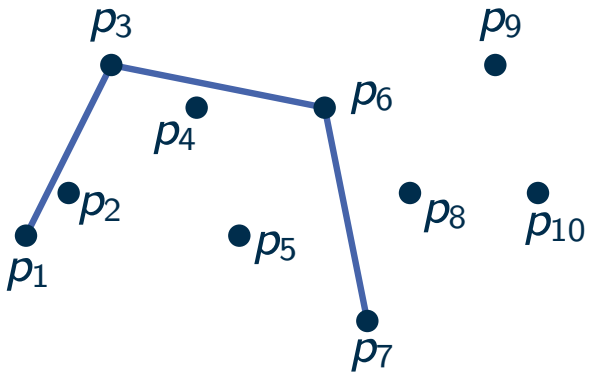
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_7

Andrews Algorithm

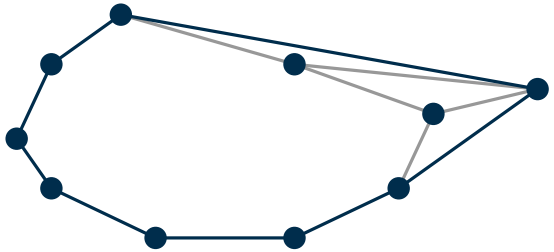
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

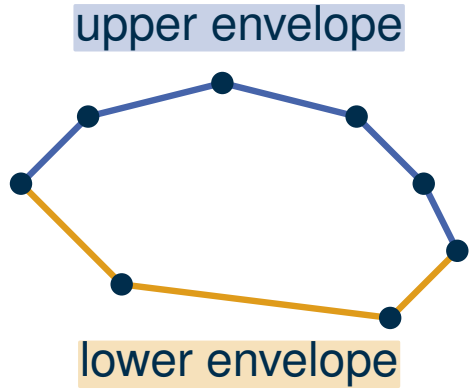
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

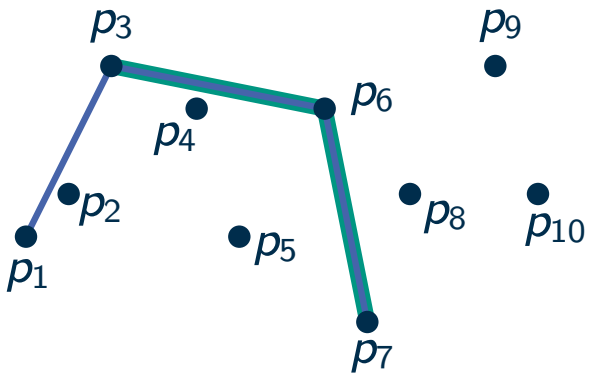
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_7

Andrews Algorithm

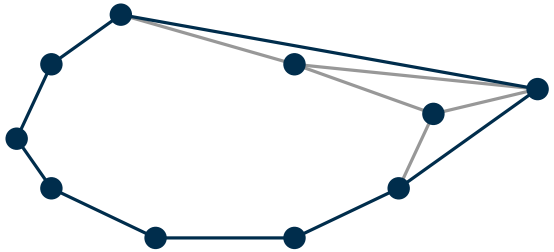
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

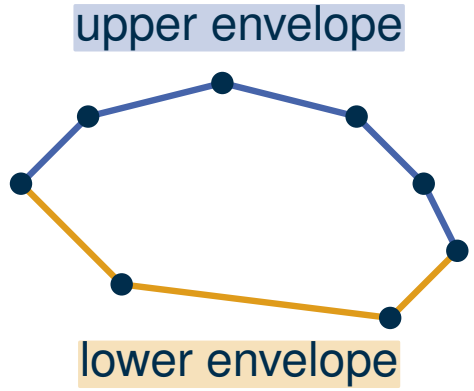
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

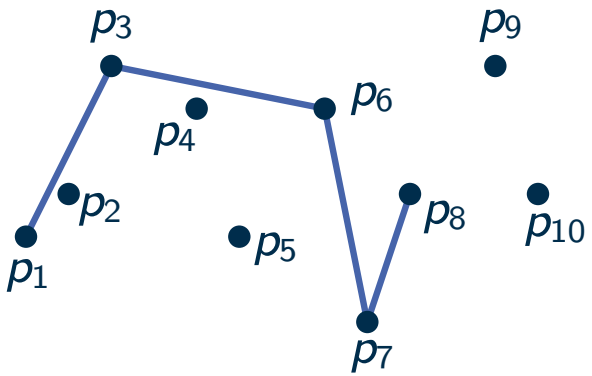
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_7 p_8

Andrews Algorithm

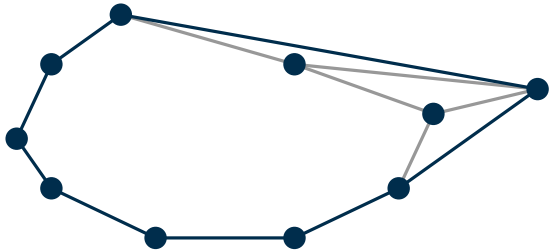
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

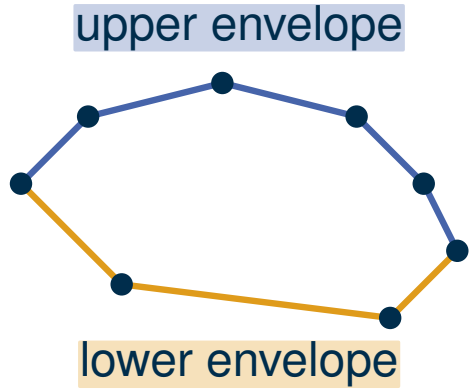
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

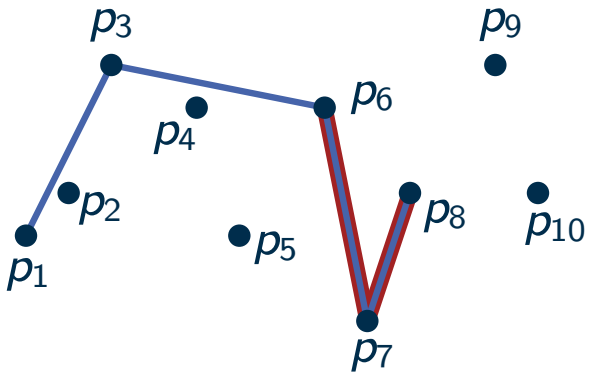
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_7 p_8

Andrews Algorithm

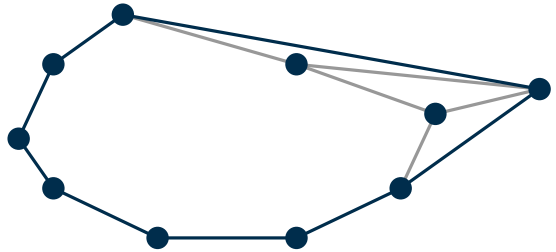
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

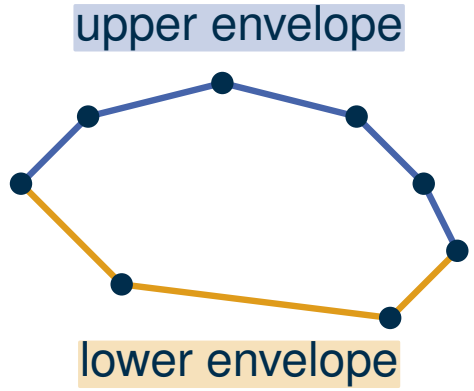
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

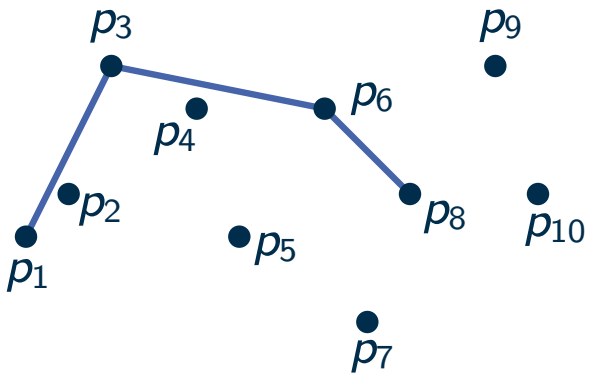
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_8

Andrews Algorithm

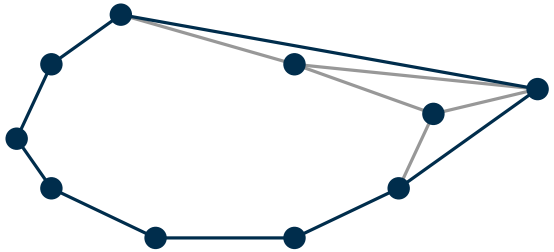
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

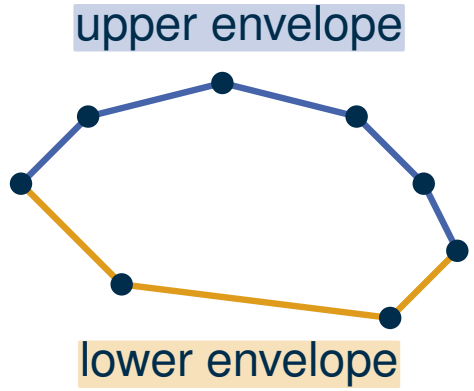
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

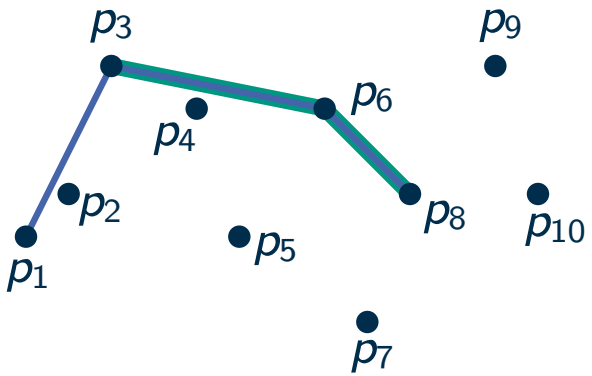
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_8

Andrews Algorithm

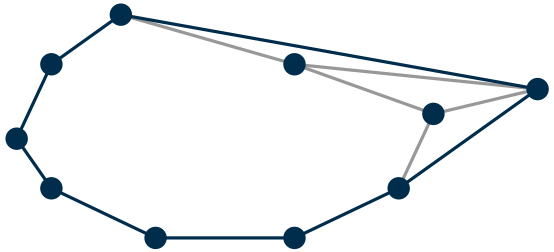
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

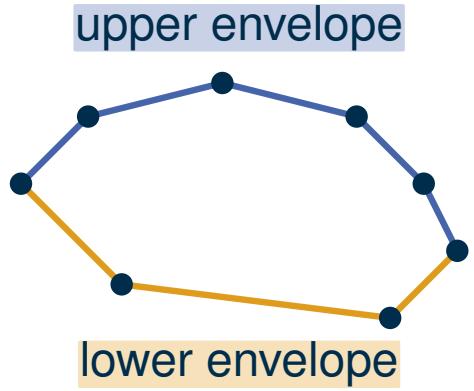
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

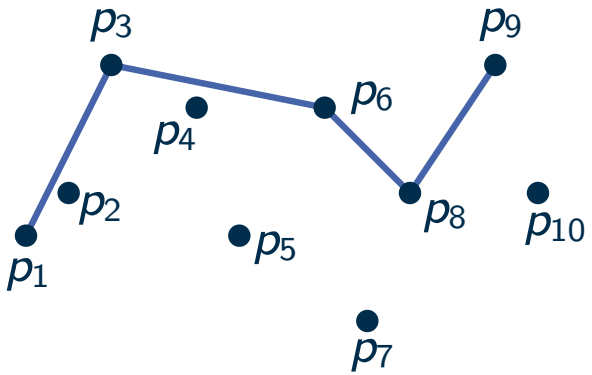
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_8 p_9

Andrews Algorithm

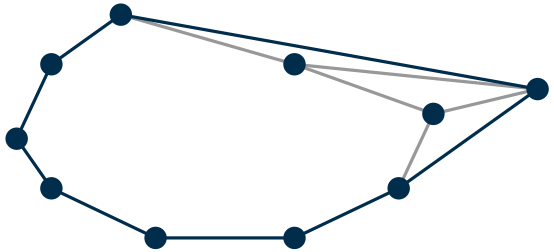
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

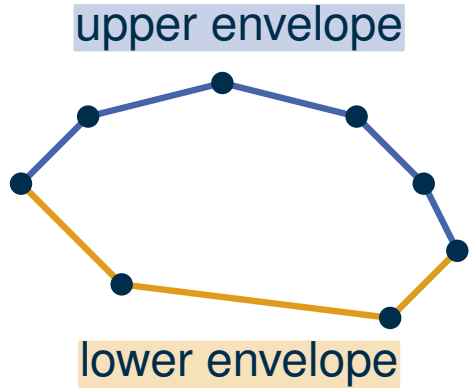
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

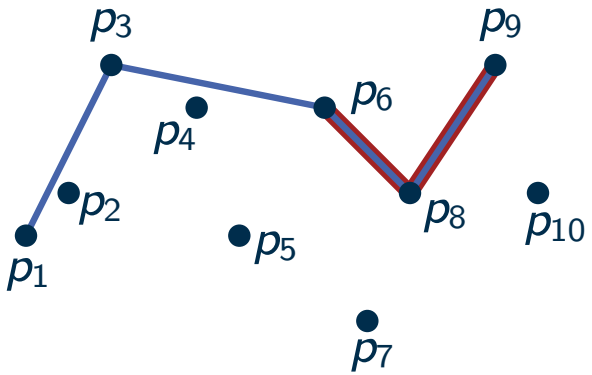
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_8 p_9

Andrews Algorithm

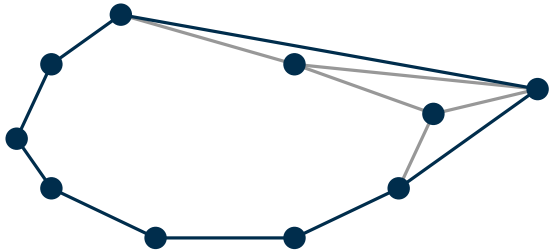
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

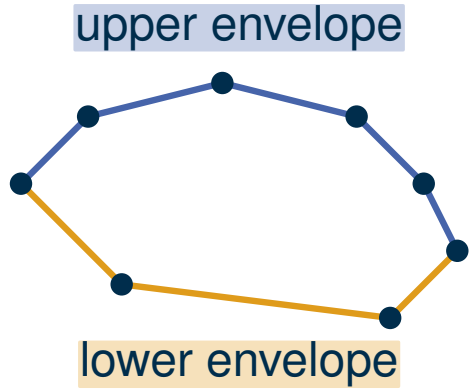
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

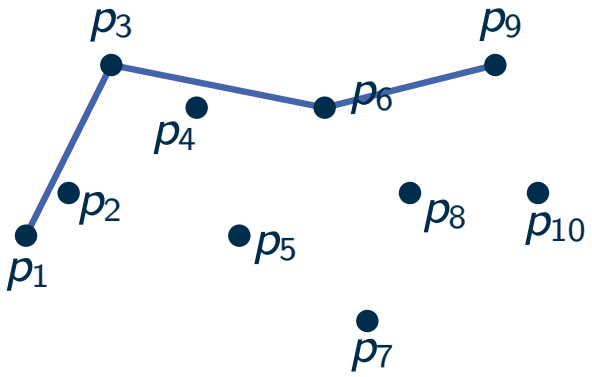
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_9

Andrews Algorithm

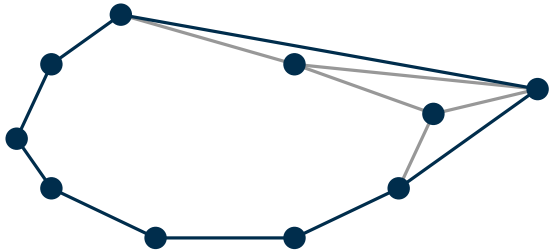
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

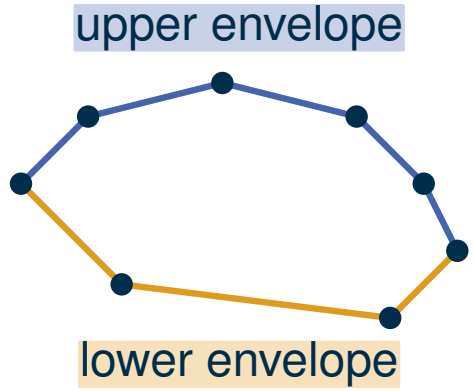
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

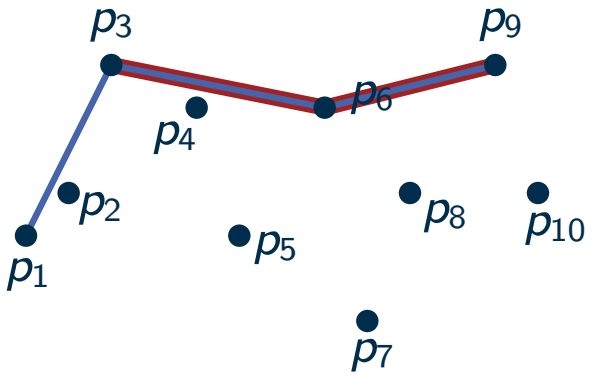
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_6 p_9

Andrews Algorithm

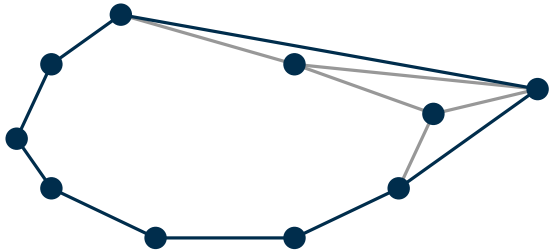
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

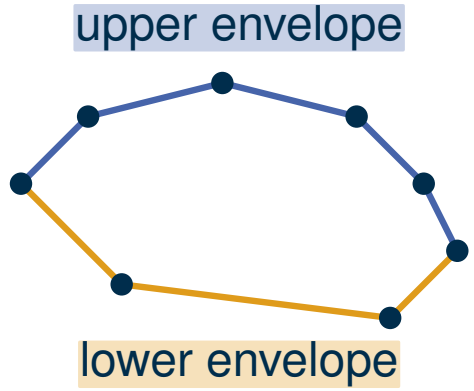
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

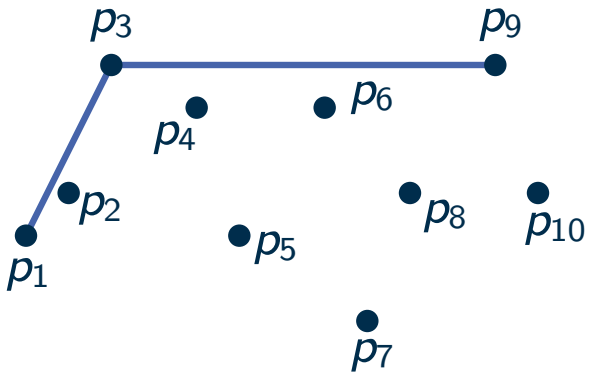
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_9

Andrews Algorithm

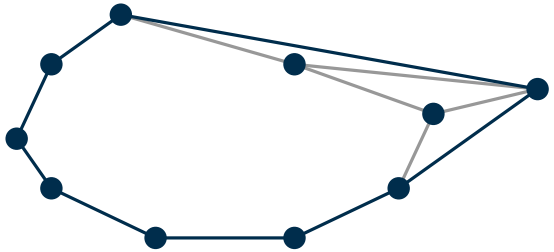
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

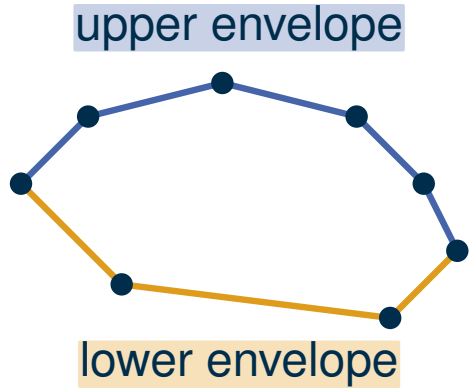
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

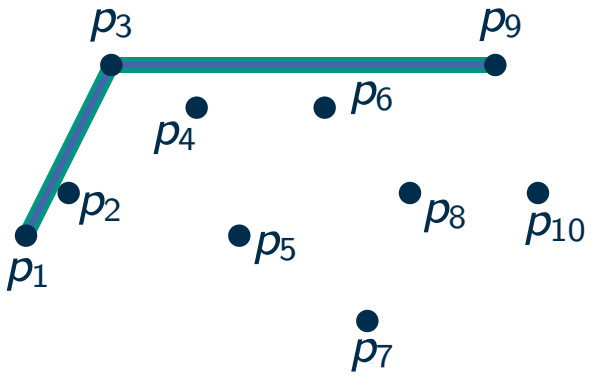
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_9

Andrews Algorithm

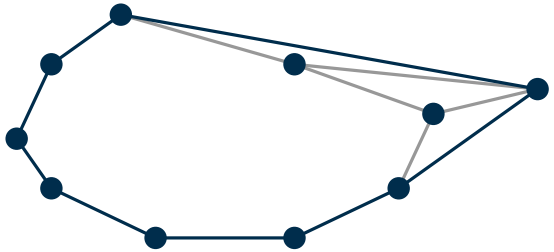
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

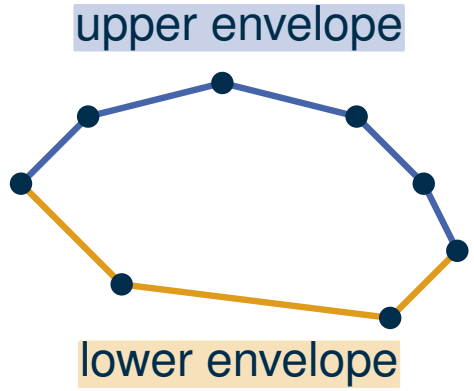
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

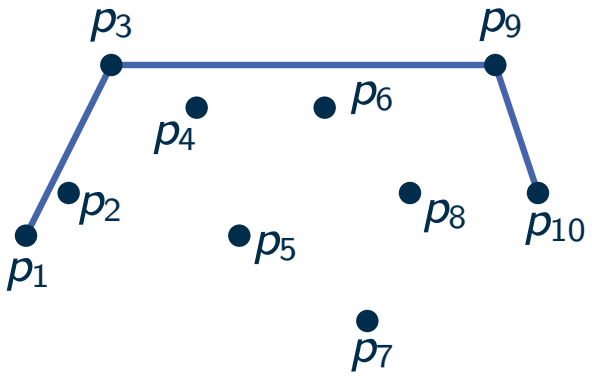
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_9 p_{10}

Andrews Algorithm

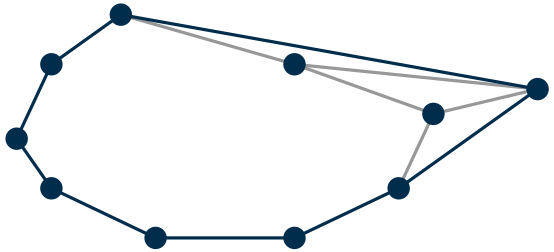
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

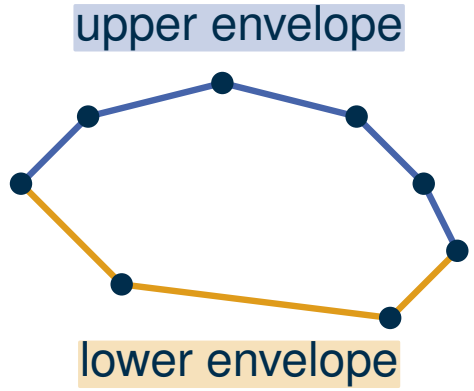
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

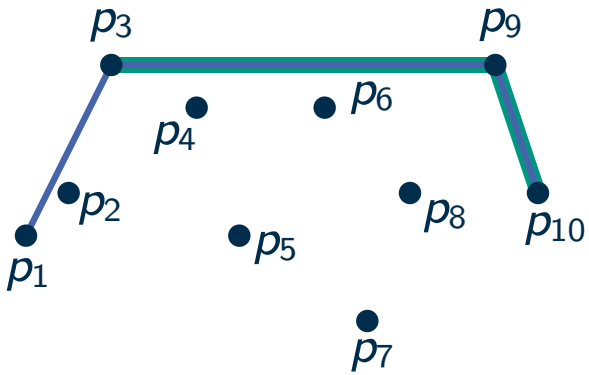
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_9 p_{10}

Andrews Algorithm

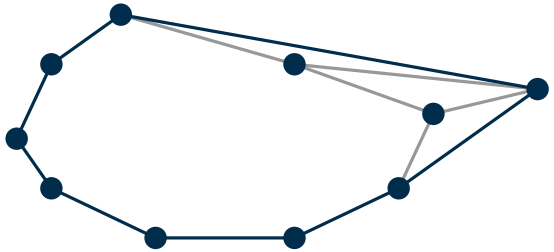
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point

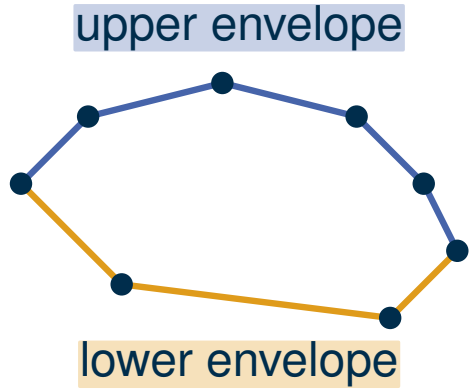
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

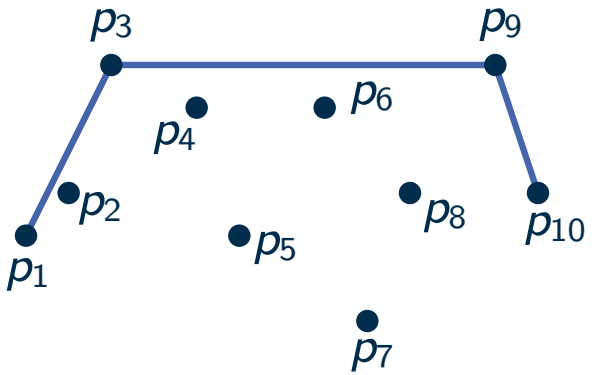
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_9 p_{10}

Andrews Algorithm

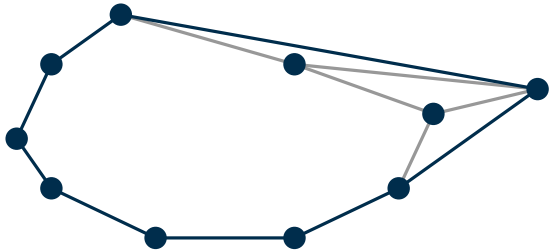
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

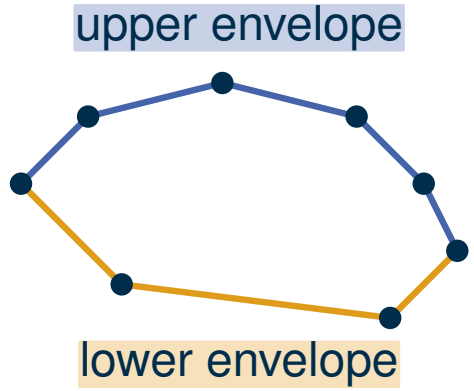
Andrews Monotone Chain Algorithm

(variant of the Graham Scan)

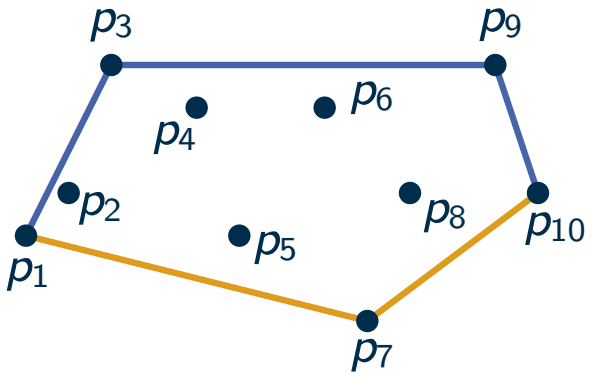
Idea: Iterative Approach

- add points one after another
- update convex hull in each step
- observe: convex hull makes only right turns
- order: from left to right
- for now: only the upper envelope





Example



L: p_1 p_3 p_9 p_{10}

analogously: lower envelope

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Andrews Algorithm – Analysis

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time:

Andrews Algorithm – Analysis

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time:

$O(n \log n)$

$O(1)$

$O(??)$

$O(1)$

$O(??)$

$O(n)$

Andrews Algorithm – Analysis

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time:

$O(n \log n)$

$O(1)$

$O(??)$

$O(1)$

$O(??)$ ← happens at most once to each point

$O(n)$

Andrews Algorithm – Analysis

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$ — $O(n \log n)$
- insert p_1 and p_2 into a L — $O(1)$
- for each remaining point p_i: — $O(n)$
  - append p_i to the back of L — $O(1)$
  - while last three points form a left bend: remove the second-to-last point — $O(1)$ (amortized)
    ← happens at most once to each point
- L is the upper envelop — $O(n)$

Running Time: $O(n \log n)$

Andrews Algorithm – Analysis

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time: $O(n \log n)$

Andrews Algorithm – Analysis

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time: $O(n \log n)$

Special Case: Same x-Coordinate



Andrews Algorithm – Analysis

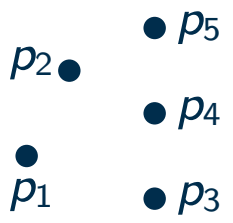
Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time: $O(n \log n)$

Special Case: Same x-Coordinate

- lexicographic order (first x, then y)
- make consistent with lower envelope



Andrews Algorithm – Analysis

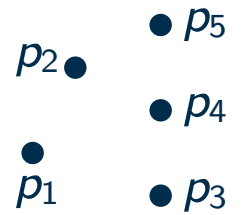
Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

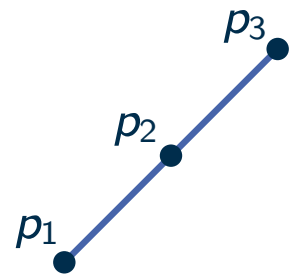
Running Time: $O(n \log n)$

Special Case: Same x-Coordinate

- lexicographic order (first x, then y)
- make consistent with lower envelope



Special Case: Collinear Points



Andrews Algorithm – Analysis

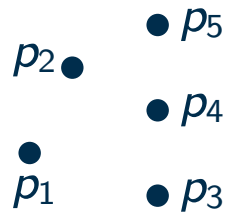
Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time: $O(n \log n)$

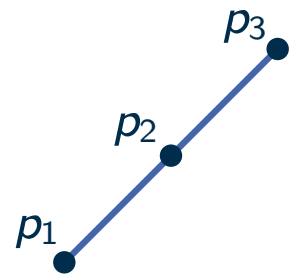
Special Case: Same x-Coordinate

- lexicographic order (first x, then y)
- make consistent with lower envelope



Special Case: Collinear Points

- p_2 should not be part of the output
- check for right instead of left bend



Andrews Algorithm – Analysis

Andrews Algorithm

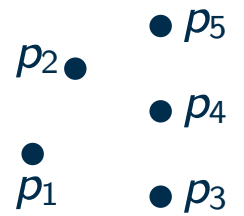
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Robustness

Running Time: $O(n \log n)$

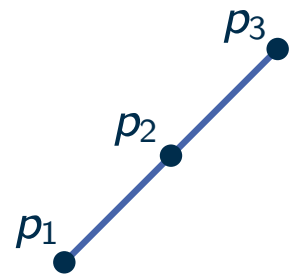
Special Case: Same x-Coordinate

- lexicographic order (first x, then y)
- make consistent with lower envelope



Special Case: Collinear Points

- p_2 should not be part of the output
- check for right instead of left bend



What if a check for left bend goes wrong?

Andrews Algorithm – Analysis

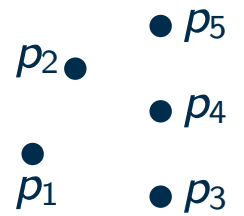
Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time: $O(n \log n)$

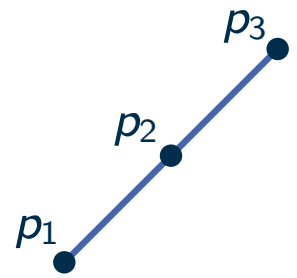
Special Case: Same x-Coordinate

- lexicographic order (first x, then y)
- make consistent with lower envelope



Special Case: Collinear Points

- p_2 should not be part of the output
- check for right instead of left bend



Robustness

- resulting polygon maybe has a slight left bend
- a point may lie slightly outside the resulting polygon

What if a check for left bend goes wrong?

Andrews Algorithm – Analysis

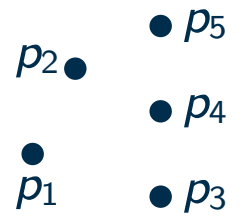
Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Running Time: $O(n \log n)$

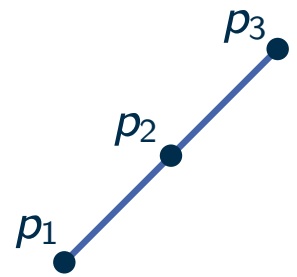
Special Case: Same x-Coordinate

- lexicographic order (first x, then y)
- make consistent with lower envelope



Special Case: Collinear Points

- p_2 should not be part of the output
- check for right instead of left bend



Robustness

- resulting polygon maybe has a slight left bend
- a point may lie slightly outside the resulting polygon
- but: the result is always a polygon that is similar to $\mathcal{CH}(P)$

What if a check for left bend goes wrong?

Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Lemma
In the end, L is the upper envelope of P.

Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L

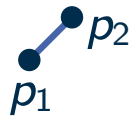
Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



Andrews Algorithm – Correctness

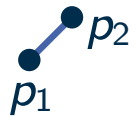
Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

After Step i: L Goes From p_1 To p_i

Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



Andrews Algorithm – Correctness

Andrews Algorithm

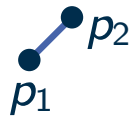
- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

After Step i: L Goes From p_1 To p_i

- obvious, as the last point is never deleted

Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



Andrews Algorithm – Correctness

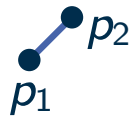
Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

After Step i: L Has Only Right Bends

Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



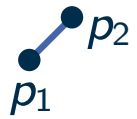
Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

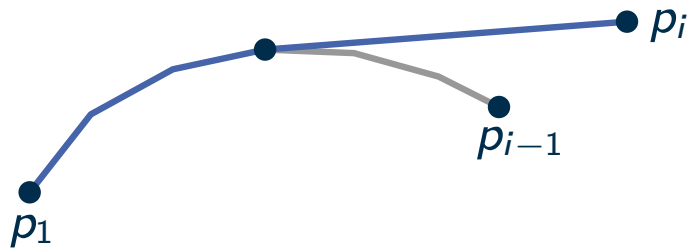
Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



After Step i: L Has Only Right Bends

- after step i, L consists of two parts
  - prefix of the polygon L from the previous step $i-1$
  - edge to p_i



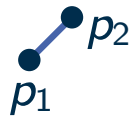
Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

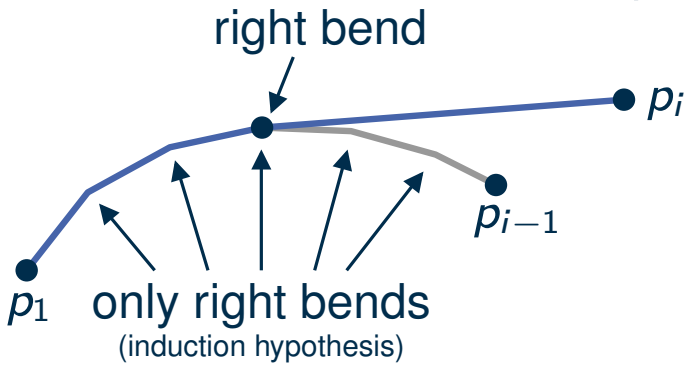
Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



After Step i: L Has Only Right Bends

- after step i, L consists of two parts
  - prefix of the polygon L from the previous step $i-1$
  - edge to p_i



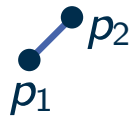
Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Lemma
In the end, L is the upper envelope of P.

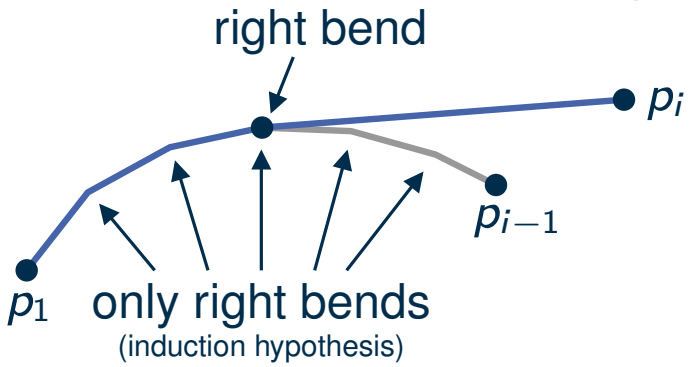
- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



After Step i: L Has Only Right Bends

- after step i, L consists of two parts
  - prefix of the polygon L from the previous step $i-1$
  - edge to p_i

$\Rightarrow$ only right bends



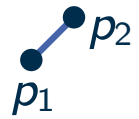
Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



After Step i: Every Point In $P_i \setminus L$ Lies Below L

Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

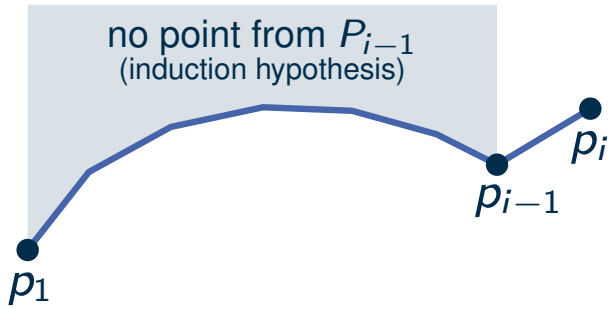
Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



After Step i: Every Point In $P_i \setminus L$ Lies Below L

- still true after inserting p_i



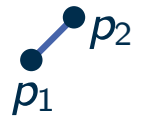
Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

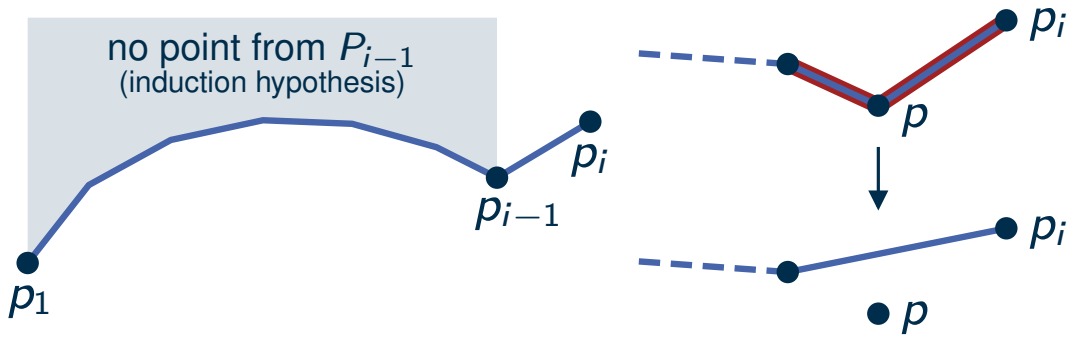
Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)



After Step i: Every Point In $P_i \setminus L$ Lies Below L

- still true after inserting p_i
- removing a point p from L moves L further up
- and afterwards, p itself lies below L



Andrews Algorithm – Correctness

Andrews Algorithm

- sort P (left to right): $p_1, \ldots, p_n$
- insert p_1 and p_2 into a L
- for each remaining point p_i:
  - append p_i to the back of L
  - while last three points form a left bend: remove the second-to-last point
- L is the upper envelop

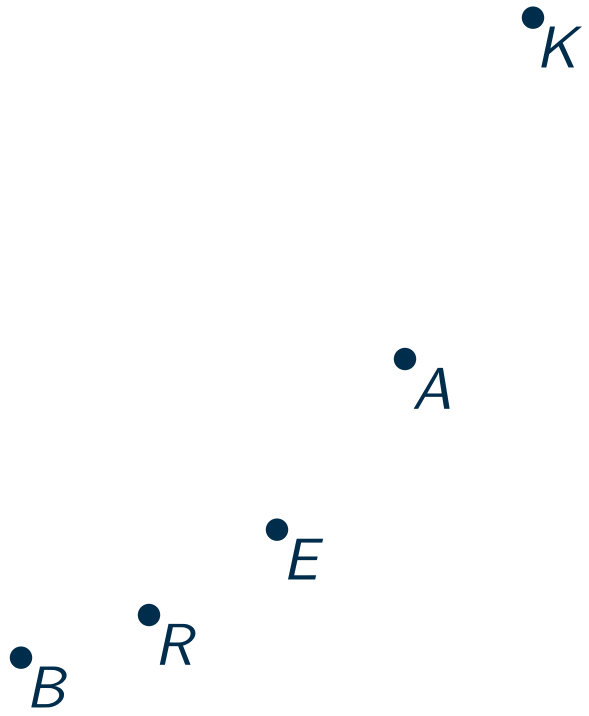
Lemma
In the end, L is the upper envelope of P.

- show: L connects p_1 with p_n, such that
  - L makes only right bends
  - every point in $P \setminus L$ lies below L
- induction over i for $P_i = \{p_1, \ldots, p_i\}$
- correct after the initialization ($i = 2$)

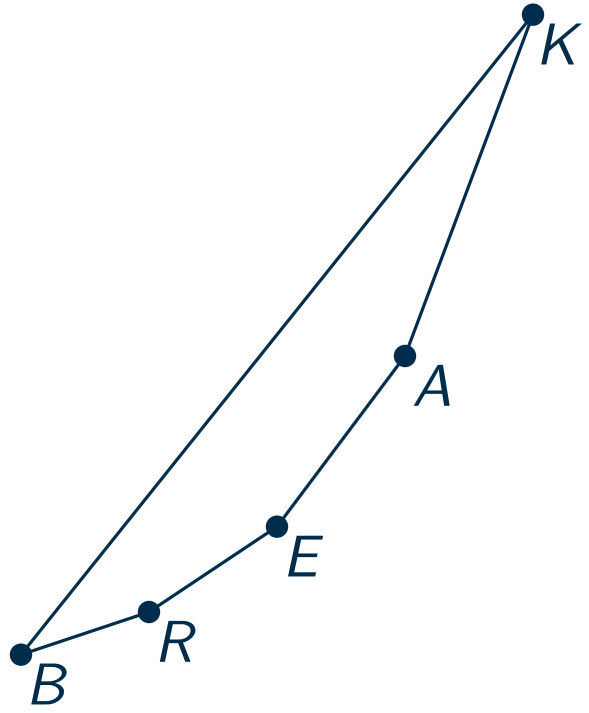


Theorem
Andrews algorithm computes the convex hull of n points in $O(n \log n)$ time.

Compute The Convex Hull



Compute The Convex Hull



Can We Be Faster?

Can We Be Faster?

Theorem
If the convex hull of n points can be computed in time $f(n)$, then we can sort n numbers in $O(f(n) + n)$ time.

Proof

Can We Be Faster?

Theorem
If the convex hull of n points can be computed in time $f(n)$, then we can sort n numbers in $O(f(n) + n)$ time.

Proof

- given: n numbers $a_1, \ldots, a_n$
- construct n points $P = \{p_1, \ldots, p_n\}$

Can We Be Faster?

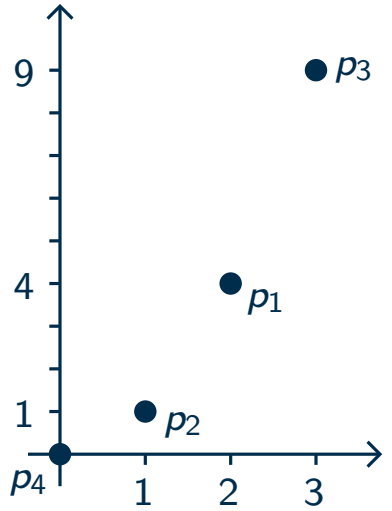
Theorem
If the convex hull of n points can be computed in time $f(n)$, then we can sort n numbers in $O(f(n) + n)$ time.

Proof

- given: n numbers $a_1, \ldots, a_n$
- construct n points $P = \{p_1, \ldots, p_n\}$ with $p_i = (a_i, a_i^2)$

Example

$a_1 = 2$, $a_2 = 1$, $a_3 = 3$, $a_4 = 0$



Can We Be Faster?

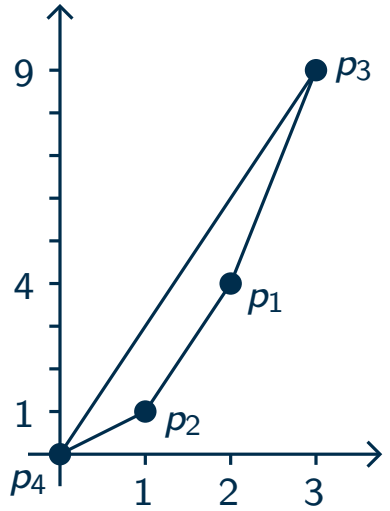
Theorem
If the convex hull of n points can be computed in time $f(n)$, then we can sort n numbers in $O(f(n) + n)$ time.

Proof

- given: n numbers $a_1, \ldots, a_n$
- construct n points $P = \{p_1, \ldots, p_n\}$ with $p_i = (a_i, a_i^2)$
- $\mathcal{CH}(P)$ contains the points sorted by a_i

Example

$a_1 = 2, a_2 = 1, a_3 = 3, a_4 = 0$



Can We Be Faster?

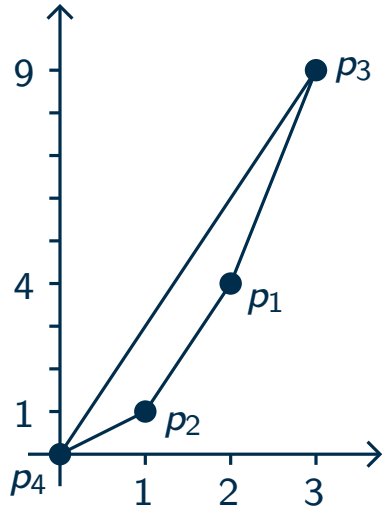
Theorem
If the convex hull of n points can be computed in time $f(n)$, then we can sort n numbers in $O(f(n) + n)$ time.

Proof

- given: n numbers $a_1, \ldots, a_n$
- construct n points $P = \{p_1, \ldots, p_n\}$ with $p_i = (a_i, a_i^2)$
- $\mathcal{CH}(P)$ contains the points sorted by a_i
- order can be obtained in $O(n)$ from $\mathcal{CH}(P)$

Example

$a_1 = 2, a_2 = 1, a_3 = 3, a_4 = 0$



Can We Be Faster?

Theorem
If the convex hull of n points can be computed in time $f(n)$, then we can sort n numbers in $O(f(n) + n)$ time.

Proof

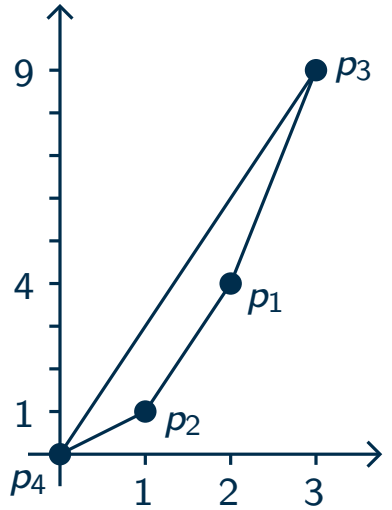
- given: n numbers $a_1, \ldots, a_n$
- construct n points $P = \{p_1, \ldots, p_n\}$ with $p_i = (a_i, a_i^2)$
- $\mathcal{CH}(P)$ contains the points sorted by a_i
- order can be obtained in $O(n)$ from $\mathcal{CH}(P)$

Lower Bound

- comparison based sorting: $\Omega(n \log n)$
- Andrews algorithm is optimal
  (unless you want to do crazy stuff with numbers)

Example

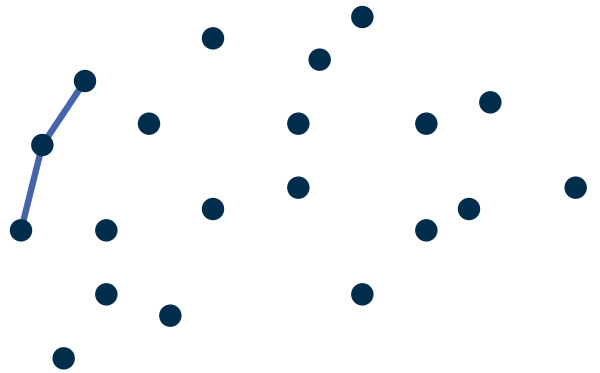
$a_1 = 2, a_2 = 1, a_3 = 3, a_4 = 0$



Gift Wrapping (Jarvis March)

Alternative Approach

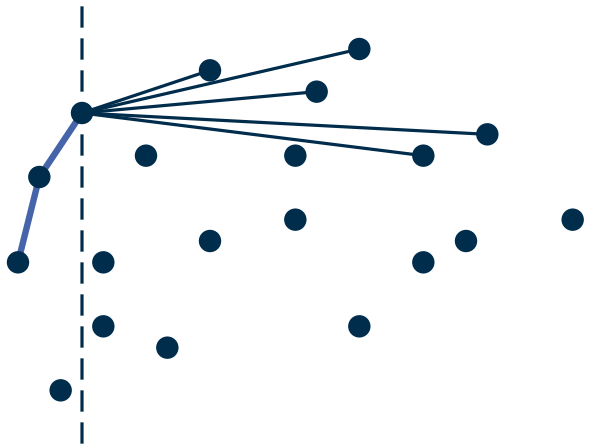
- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope



Gift Wrapping (Jarvis March)

Alternative Approach

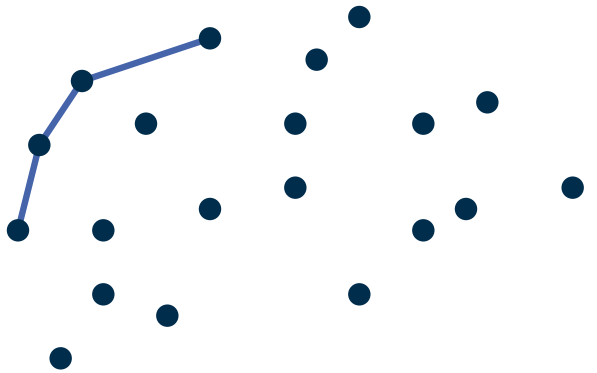
- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope
- choose point with the smallest angle



Gift Wrapping (Jarvis March)

Alternative Approach

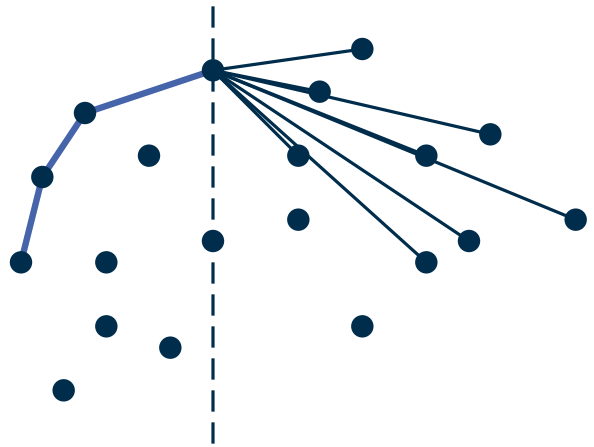
- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope
- choose point with the smallest angle



Gift Wrapping (Jarvis March)

Alternative Approach

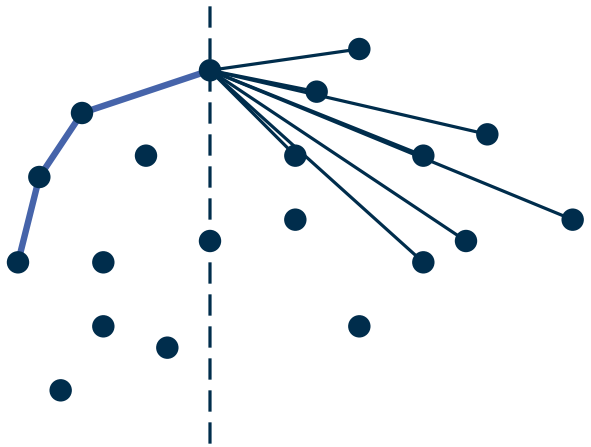
- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope
- choose point with the smallest angle



Gift Wrapping (Jarvis March)

Alternative Approach

- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope
- choose point with the smallest angle



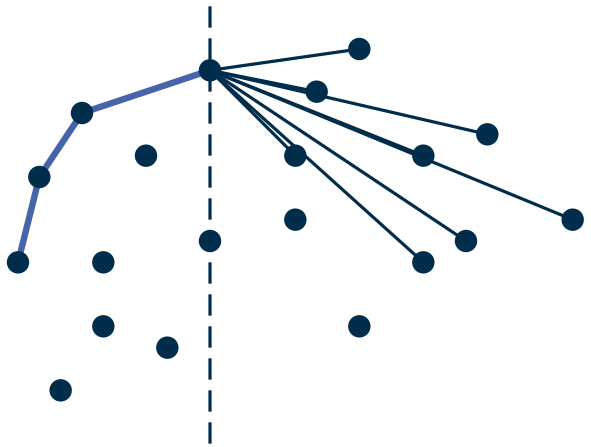
Running Time

- each step: find minimum $\rightarrow O(n)$

Gift Wrapping (Jarvis March)

Alternative Approach

- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope
- choose point with the smallest angle



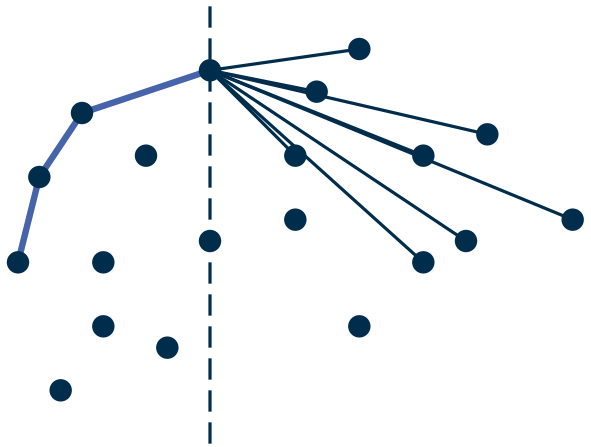
Running Time

- each step: find minimum $\rightarrow O(n)$
- h steps, for $h = |\mathcal{CH}(P)|$

Gift Wrapping (Jarvis March)

Alternative Approach

- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope
- choose point with the smallest angle



Running Time

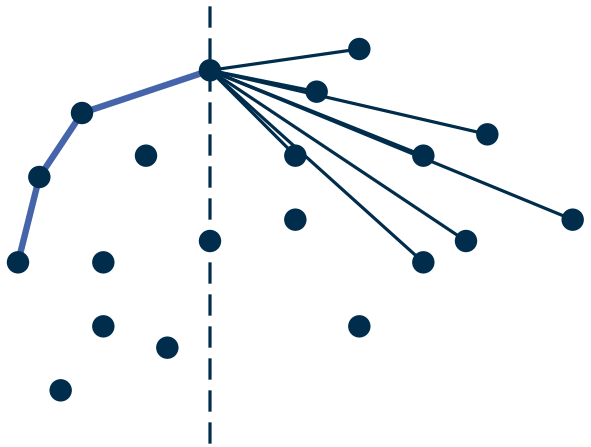
- each step: find minimum $\rightarrow O(n)$
- h steps, for $h = |\mathcal{CH}(P)|$

Theorem
The Gift Wrapping algorithm computes the convex hull of n points P in $O(hn)$ time, where h is the number of points of $\mathcal{CH}(P)$.

Gift Wrapping (Jarvis March)

Alternative Approach

- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope
- choose point with the smallest angle



Running Time

- each step: find minimum $\rightarrow O(n)$
- h steps, for $h = |\mathcal{CH}(P)|$

Theorem
The Gift Wrapping algorithm computes the convex hull of n points P in $O(hn)$ time, where h is the number of points of $\mathcal{CH}(P)$.

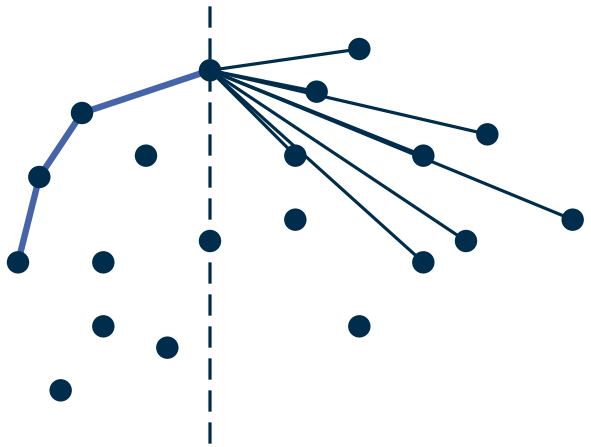
Comment

- such an algorithm is called **output sensitive**

Gift Wrapping (Jarvis March)

Alternative Approach

- assumption: we already know parts of the upper envelope
- goal: find the next point on the upper envelope
- choose point with the smallest angle



Running Time

- each step: find minimum $\rightarrow O(n)$
- h steps, for $h = |\mathcal{CH}(P)|$

Theorem
The Gift Wrapping algorithm computes the convex hull of n points P in $O(hn)$ time, where h is the number of points of $\mathcal{CH}(P)$.

Comment

- such an algorithm is called **output sensitive**
- beats the lower bound on certain instances (small h)

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$
- $\Omega(n \log n)$ lower bound

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$
- $\Omega(n \log n)$ lower bound
- output sensitive algorithm with running time $O(hn)$

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$
- $\Omega(n \log n)$ lower bound
- output sensitive algorithm with running time $O(hn)$
- robustness is an important aspect in computational geometry

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$
- $\Omega(n \log n)$ lower bound
- output sensitive algorithm with running time $O(hn)$
- robustness is an important aspect in computational geometry
- initially assuming general position helps with algorithm design

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$
- $\Omega(n \log n)$ lower bound
- output sensitive algorithm with running time $O(hn)$
- robustness is an important aspect in computational geometry
- initially assuming general position helps with algorithm design

What Else Is There?

- one can achieve running time $O(n \log h)$

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$
- $\Omega(n \log n)$ lower bound
- output sensitive algorithm with running time $O(hn)$
- robustness is an important aspect in computational geometry
- initially assuming general position helps with algorithm design

What Else Is There?

- one can achieve running time $O(n \log h)$
- higher dimensions

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$
- $\Omega(n \log n)$ lower bound
- output sensitive algorithm with running time $O(hn)$
- robustness is an important aspect in computational geometry
- initially assuming general position helps with algorithm design

What Else Is There?

- one can achieve running time $O(n \log h)$
- higher dimensions
- convex hull of a simple polygon can be computed in $O(n)$ time

Wrap-Up

What Have We Learned Today?

- algorithm for computing the convex hull in time $O(n \log n)$
- $\Omega(n \log n)$ lower bound
- output sensitive algorithm with running time $O(hn)$
- robustness is an important aspect in computational geometry
- initially assuming general position helps with algorithm design

What Else Is There?

- one can achieve running time $O(n \log h)$
- higher dimensions
- convex hull of a simple polygon can be computed in $O(n)$ time

Reminder: hand in exercises in groups of two or three