Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

Computational Geometry

Convex Hull
or
Mixing Things

Lecture #1

Thomas van Dijk

Winter Semester 2019/20

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

Computational Geometry

Convex Hull
or
Mixing Things

Lecture #1

Thomas van Dijk

Winter Semester 2019/20

Chair I

b) Knowledge-Based Systems

Prof. Dietmar Seipel

c) Theoretical Comp. Science

Prof. Christian Glaßer

a) Efficient Algorithms

Chair I

b) Knowledge-Based Systems
Prof. Dietmar Seipel
c) Theoretical Comp. Science
Prof. Christian Glaßer

a) Efficient Algorithms

Alexander Wolff
Professor

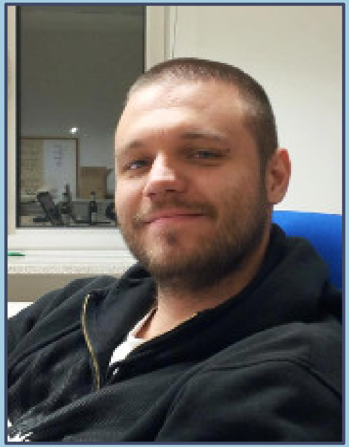
Philipp Kindermann
PostDoc

Oksana Firman

Steven Chaplick
PostDoc

Thomas van Dijk
PostDoc

Myroslav Kryven



André Löffler

Johannes Zink

Computational Geometry

Learning goals: At the end of this lecture you will be able to

Computational Geometry

Learning goals: At the end of this lecture you will be able to

– decide which algorithms help to solve a number of fundamental *geometric* problems,

Computational Geometry

Learning goals: At the end of this lecture you will be able to

- decide which algorithms help to solve a number of fundamental *geometric* problems,
- analyze new problems and find *efficient* solutions with the concepts of the lecture.

Computational Geometry

Learning goals: At the end of this lecture you will be able to

– decide which algorithms help to solve a number of fundamental *geometric* problems,

– analyze new problems and find *efficient* solutions with the concepts of the lecture.

Requirements: – Big-Oh notation (Landau); e.g., $O(n \log n)$

Computational Geometry

Learning goals: At the end of this lecture you will be able to

– decide which algorithms help to solve a number of fundamental *geometric* problems,

– analyze new problems and find *efficient* solutions with the concepts of the lecture.

Requirements:

– Big-Oh notation (Landau); e.g., $O(n \log n)$

– Some basic *Algorithms & Data Structures*

(Balanced) binary search tree, priority queue

Computational Geometry

Learning goals: At the end of this lecture you will be able to

- decide which algorithms help to solve a number of fundamental *geometric* problems,
- analyze new problems and find *efficient* solutions with the concepts of the lecture.

Requirements:

- Big-Oh notation (Landau); e.g., $O(n \log n)$
- Some basic *Algorithms & Data Structures*
  (Balanced) binary search tree, priority queue
- Some basic *Algorithmic Graph Theory*
  Breadth-first search, Dijkstra's algorithm

Computational Geometry

Learning goals: At the end of this lecture you will be able to

- decide which algorithms help to solve a number of fundamental *geometric* problems,
- analyze new problems and find *efficient* solutions with the concepts of the lecture.

Requirements:

- Big-Oh notation (Landau); e.g., $O(n \log n)$
- Some basic *Algorithms & Data Structures*
  (Balanced) binary search tree, priority queue
- Some basic *Algorithmic Graph Theory*
  Breadth-first search, Dijkstra's algorithm
- Oral exam at the end of the semester

Computational Geometry

Learning goals: At the end of this lecture you will be able to

– decide which algorithms help to solve a number of fundamental *geometric* problems,

– analyze new problems and find *efficient* solutions with the concepts of the lecture.

Requirements:

– Big-Oh notation (Landau); e.g., $O(n \log n)$

– Some basic *Algorithms & Data Structures*
  (Balanced) binary search tree, priority queue

– Some basic *Algorithmic Graph Theory*
  Breadth-first search, Dijkstra's algorithm

– Oral exam at the end of the semester

My vision:

- "hands-on"
- interactive

Content

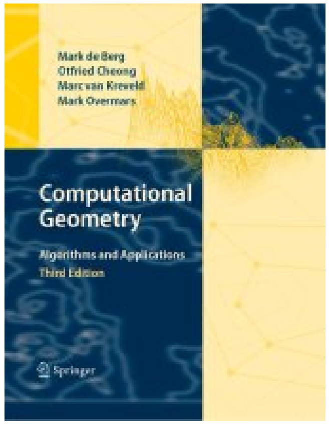
1. Convex Hull in 2D
2. Map Overlay
3. Polygon Triangulation
4. Linear Programming
5. Orthogonal Range Queries
6. Point Location
7. Voronoi Diagram
8. Delaunay Triangulation
9. Convex Hull in 3D
10. Motion Planning
11. Simplex Range Searching
12. Arrangements and Duality
13. Binary Space Partition
14. Randomized Triangulation

Content

1. Convex Hull in 2D
2. Map Overlay
3. Polygon Triangulation
4. Linear Programming
5. Orthogonal Range Queries
6. Point Location
7. Voronoi Diagram
8. Delaunay Triangulation
9. Convex Hull in 3D
10. Motion Planning
11. Simplex Range Searching
12. Arrangements and Duality
13. Binary Space Partition
14. Randomized Triangulation

New! Minimum Convex Partition

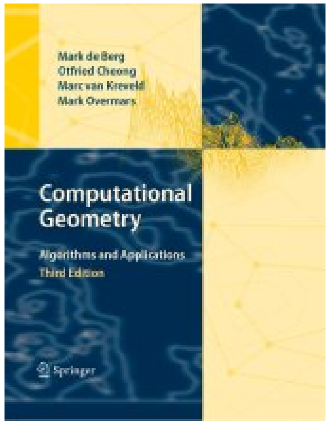
Literature



M. de Berg, O. Cheong, M. van Kreveld, M. Overmars: Computational Geometry: Algorithms & Applications. Springer, 3rd edition, 2008

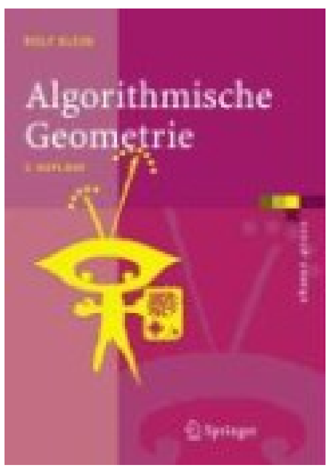
Main resource for this course!
Abbreviated as: Comp. Geom A&A

Literature



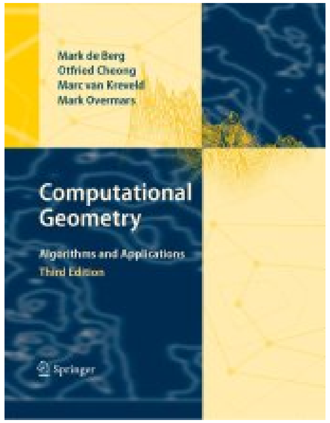
M. de Berg, O. Cheong, M. van Kreveld, M. Overmars:
Computational Geometry: Algorithms & Applications.
Springer, 3rd edition, 2008

Main resource for this course!
Abbreviated as: Comp. Geom A&A



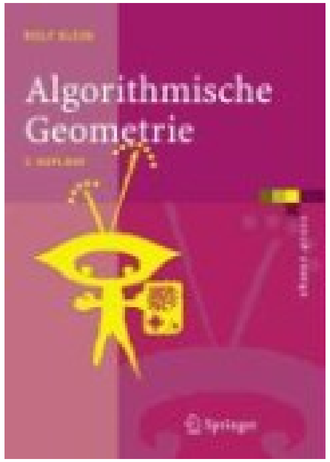
Rolf Klein:
Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen.
Springer, 2nd edition, 2005

Literature

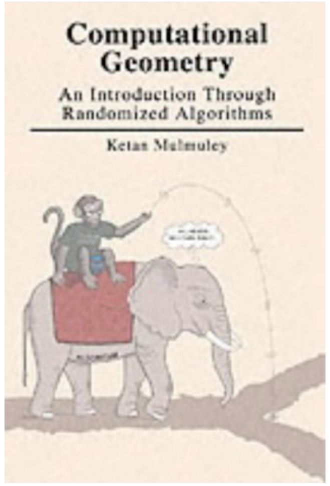


M. de Berg, O. Cheong, M. van Kreveld, M. Overmars:
Computational Geometry: Algorithms & Applications.
Springer, 3rd edition, 2008

Main resource for this course!
Abbreviated as: Comp. Geom A&A



Rolf Klein:
Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen.
Springer, 2nd edition, 2005



Ketan Mulmuley:
Computational Geometry: An Introduction Through Randomized Algorithms. Prentice Hall, 1st edition, 1993

Week 1

Convex Hull
or:
Mixing Things

[Comp. Geom A&A : Chapter 1, (see also Ch. 11)]

Mixing Things

Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

Mixing Things

Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 ?

Mixing Things

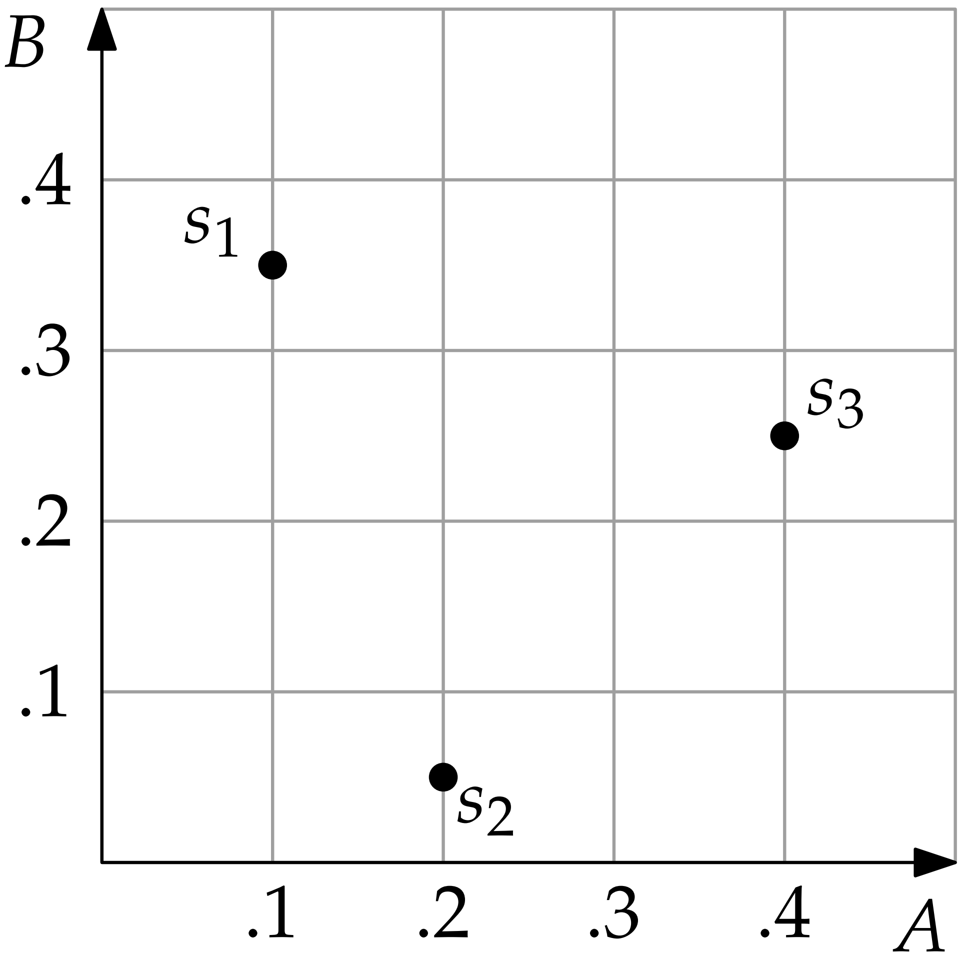
Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 ?



Mixing Things

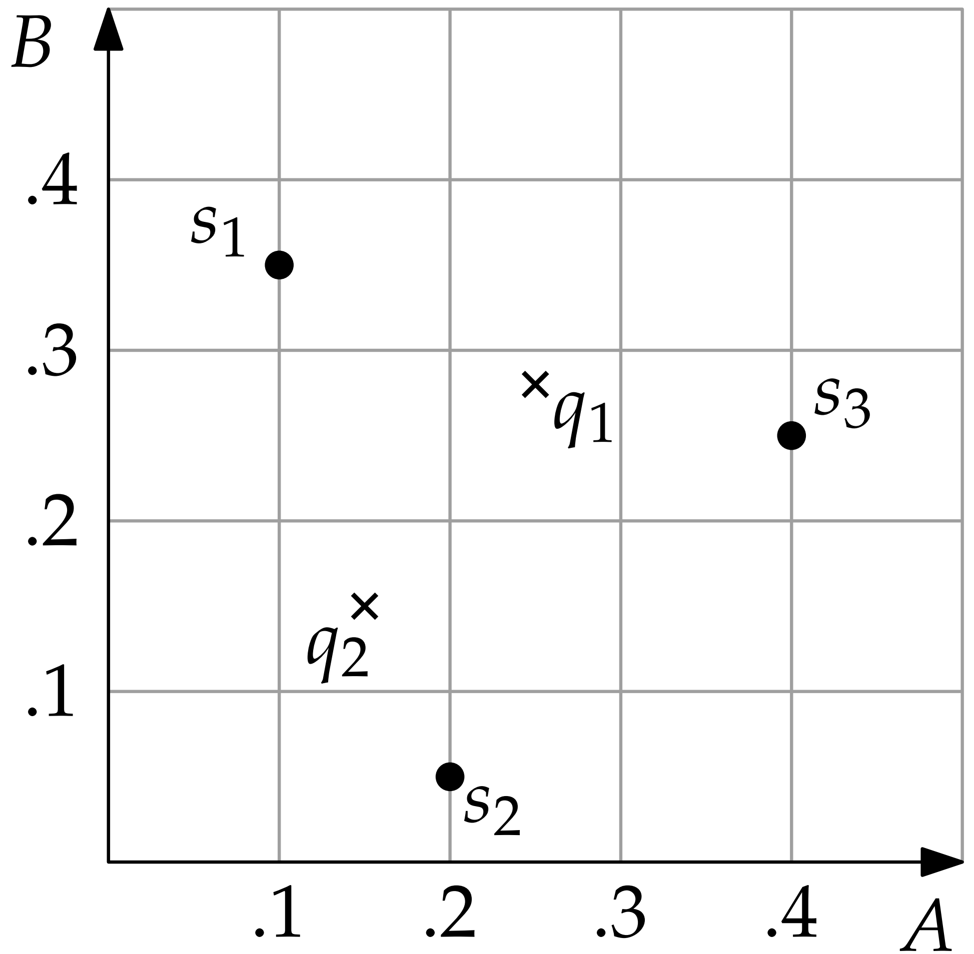
Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 ?



Mixing Things

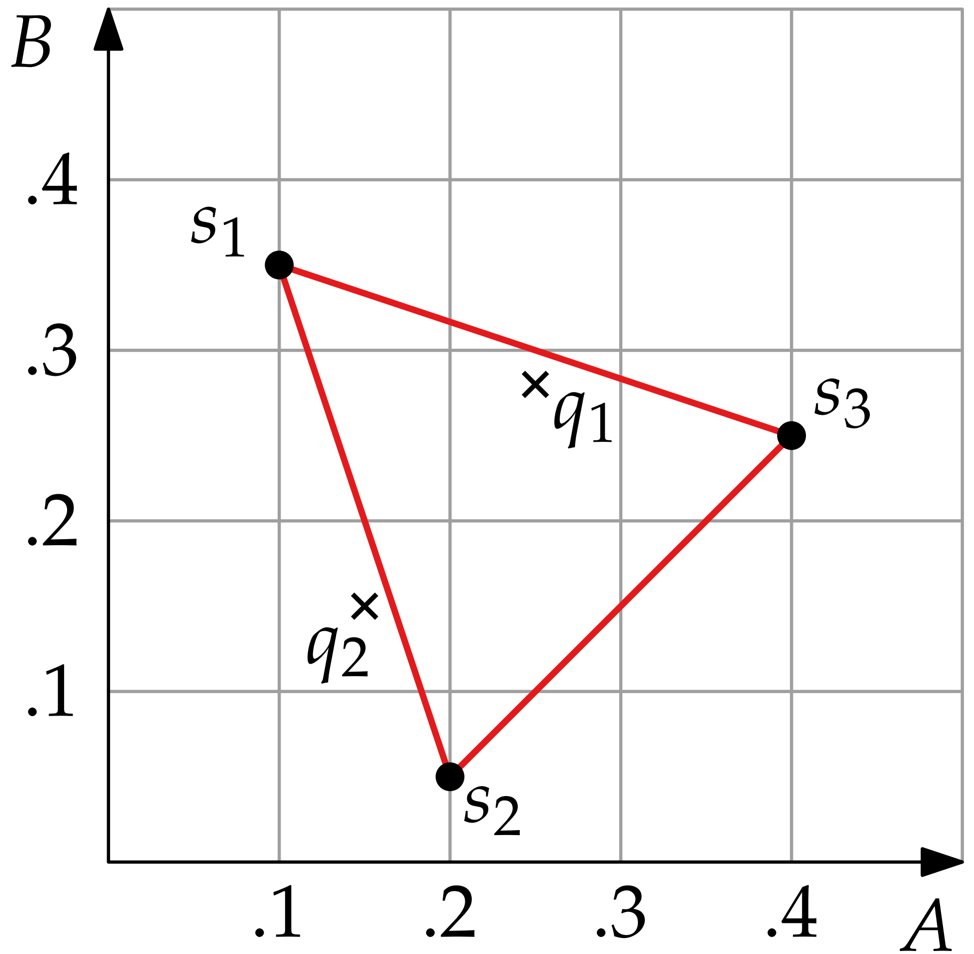
Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 ?



Mixing Things

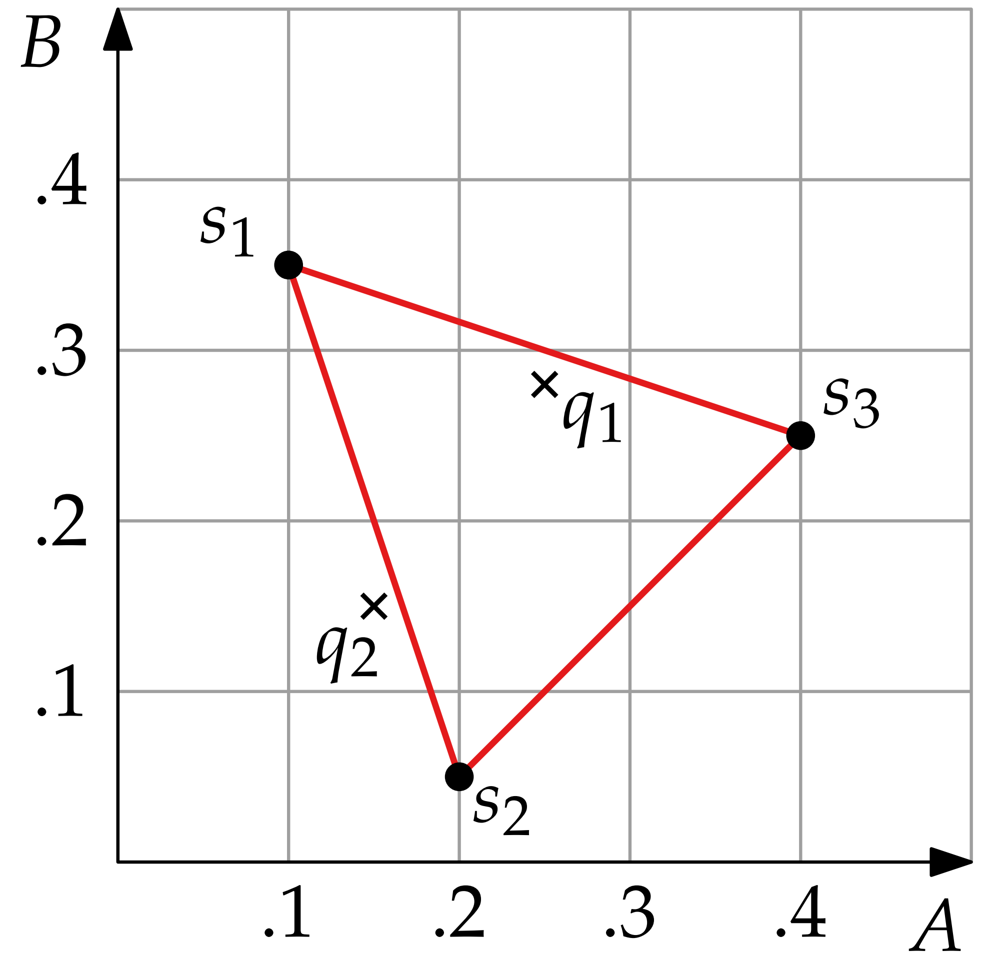
Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 ?



Observe: Given a set $S \subset \mathbb{R}^2$ of substances, we can mix a substance $q \in \mathbb{R}^2$ using the substances in S $\Leftrightarrow$

Mixing Things

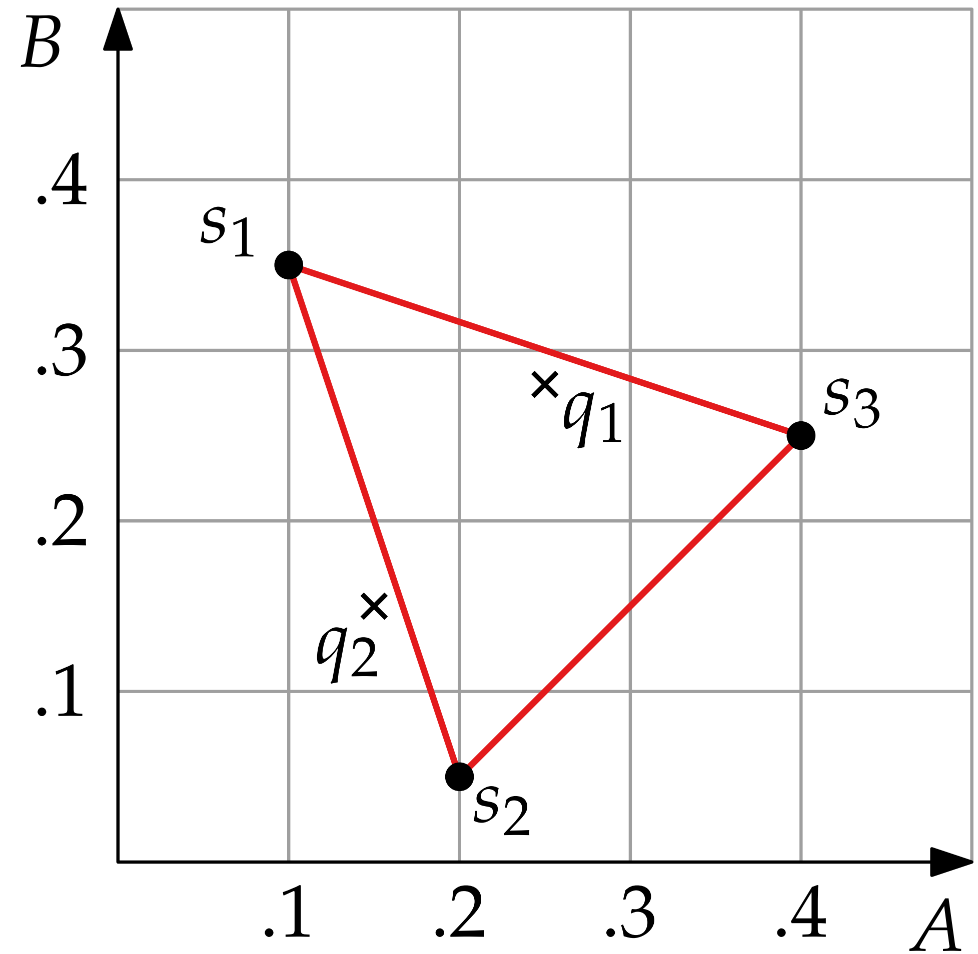
Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 ?



Observe: Given a set $S \subset \mathbb{R}^2$ of substances, we can mix a substance $q \in \mathbb{R}^2$ using the substances in S $\Leftrightarrow$ $q \in \mathrm{CH}(S)$.

Mixing Things

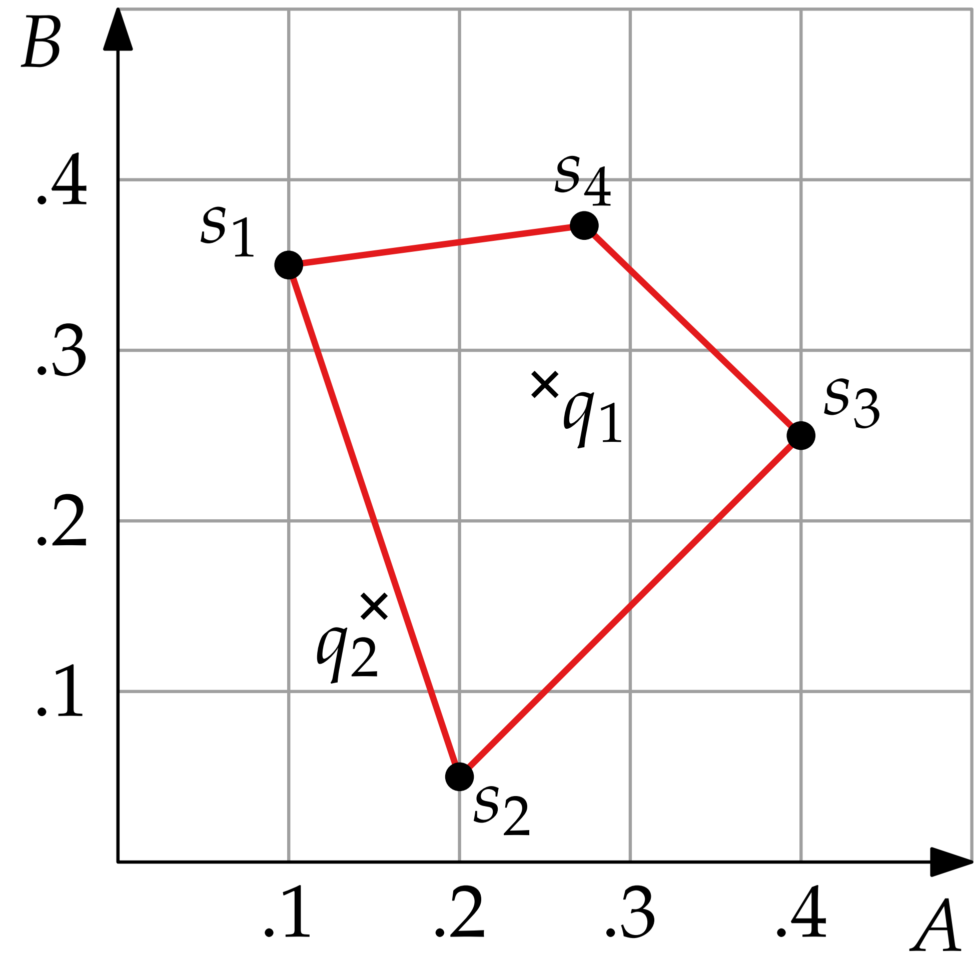
Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 **?**



Observe: Given a set $S \subset \mathbb{R}^2$ of substances, we can mix a substance $q \in \mathbb{R}^2$ using the substances in S $\Leftrightarrow$ $q \in \mathrm{CH}(S)$.

Mixing Things

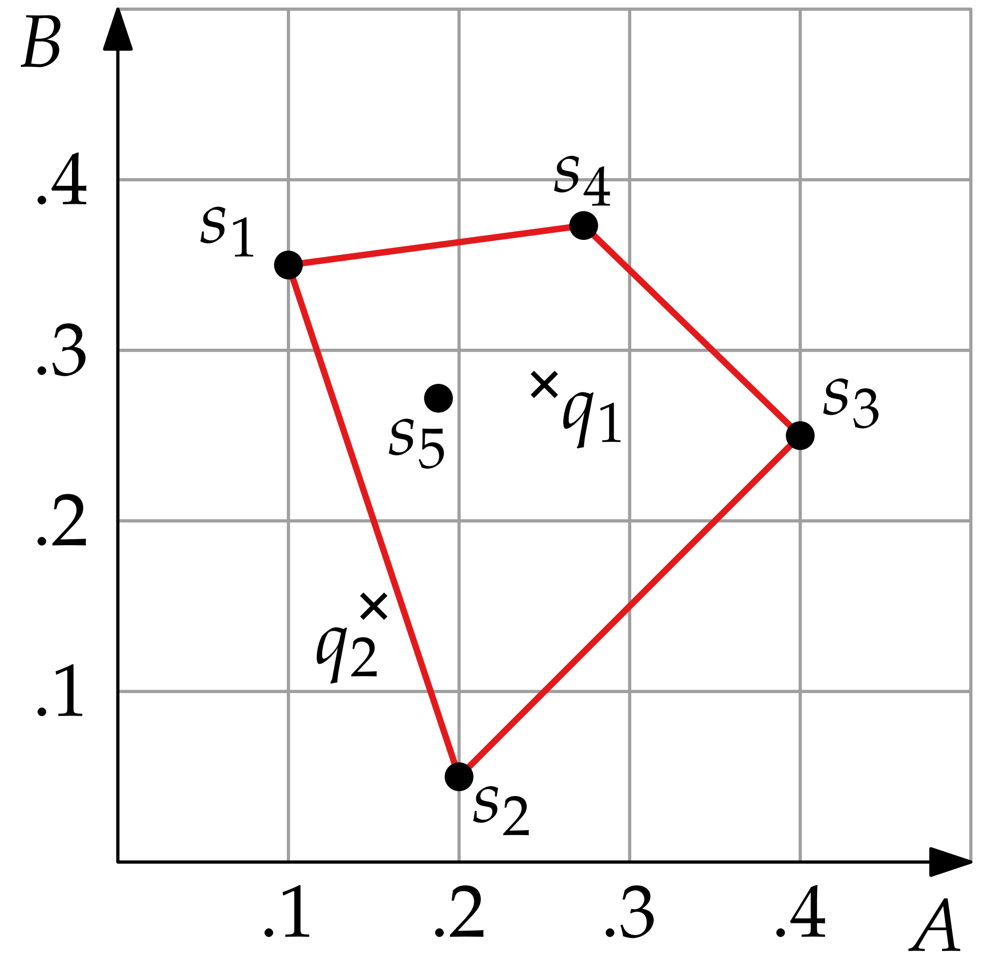
Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 **?**



Observe: Given a set $S \subset \mathbb{R}^2$ of substances, we can mix a substance $q \in \mathbb{R}^2$ using the substances in S $\Leftrightarrow$ $q \in \mathrm{CH}(S)$.

Mixing Things

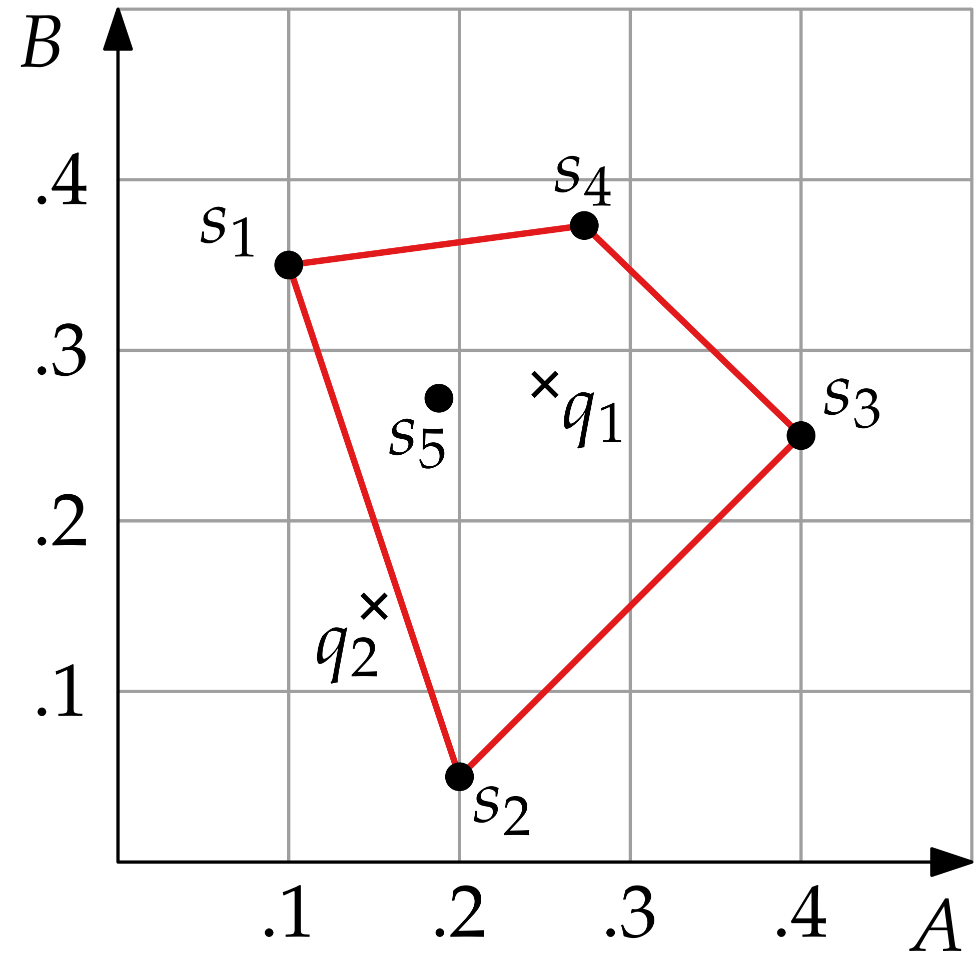
Given...

subst.	fract. A	fract. B
s_1	10 %	35 %
s_2	20 %	5 %
s_3	40 %	25 %

can we mix

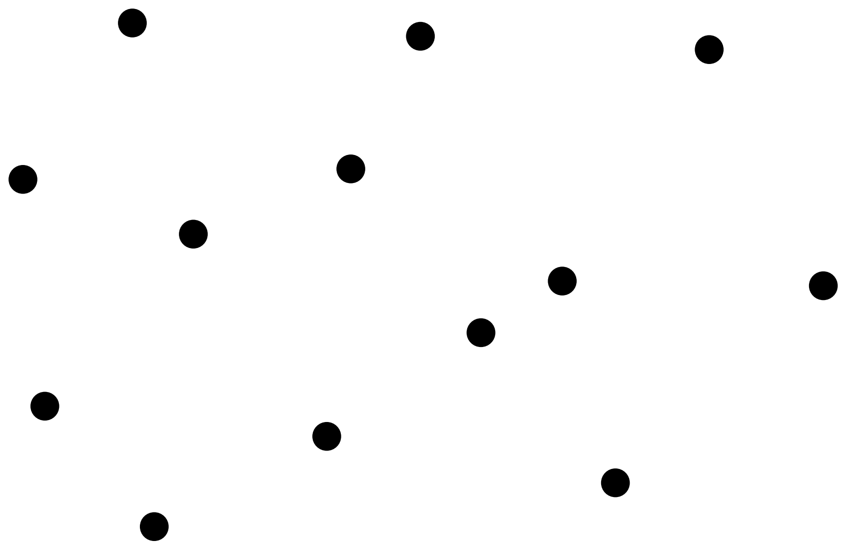
q_1	25 %	28 %
q_2	15 %	15 %

using s_1, s_2, s_3 **?**



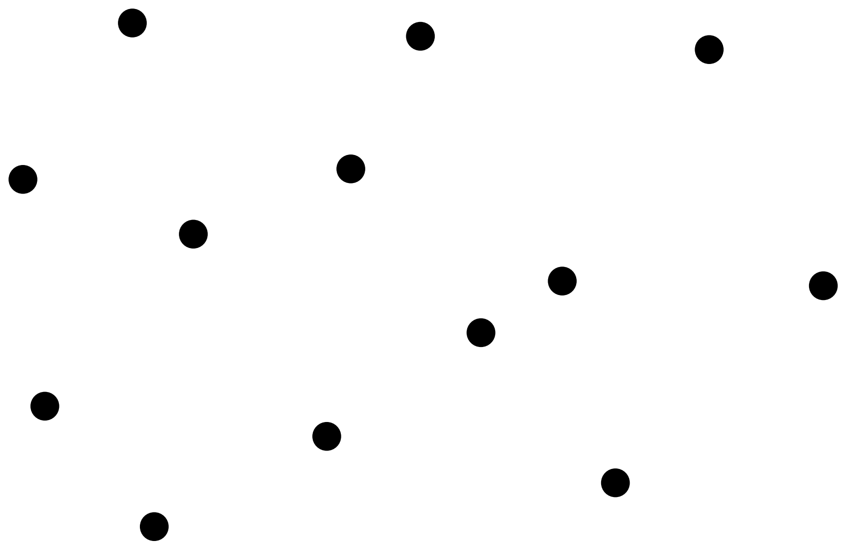
Observe: Given a set $S \subset \mathbb{R}^{\cancel{2}d}$ of substances, we can mix a substance $q \in \mathbb{R}^{\cancel{2}d}$ using the substances in S $\Leftrightarrow$ $q \in \mathrm{CH}(S)$.

Formally. . .



Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

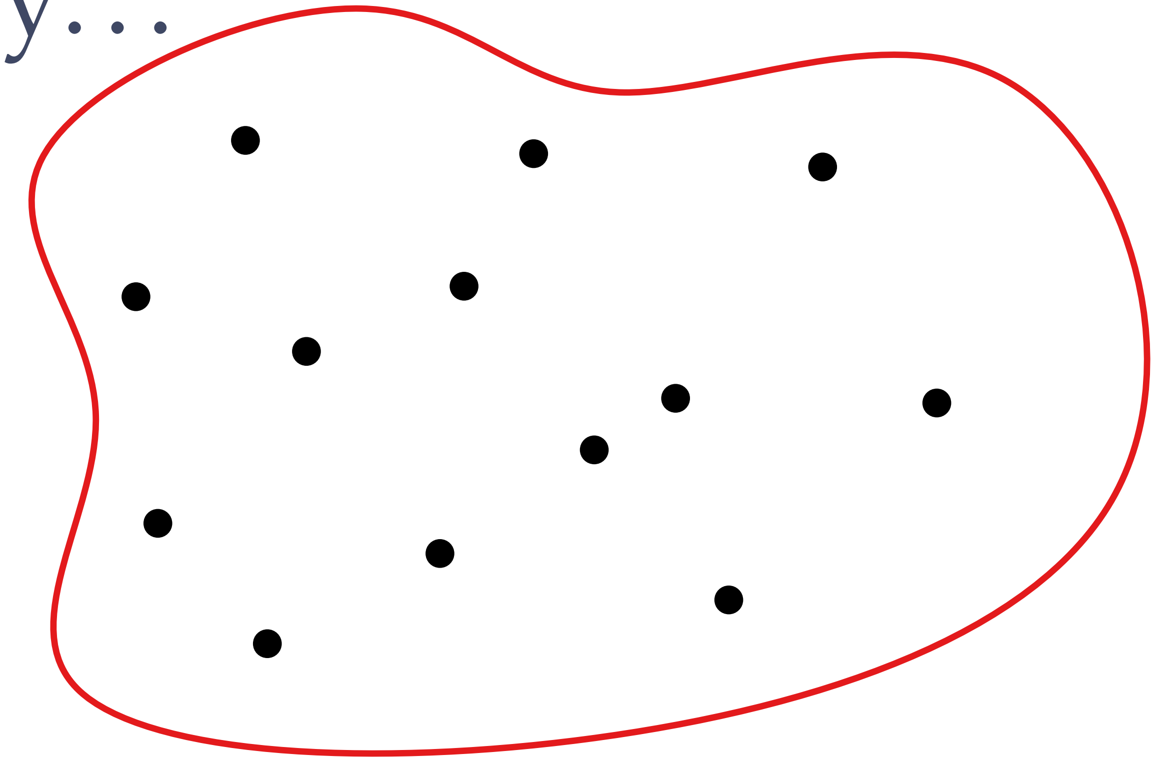
Formally. . .



Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach:

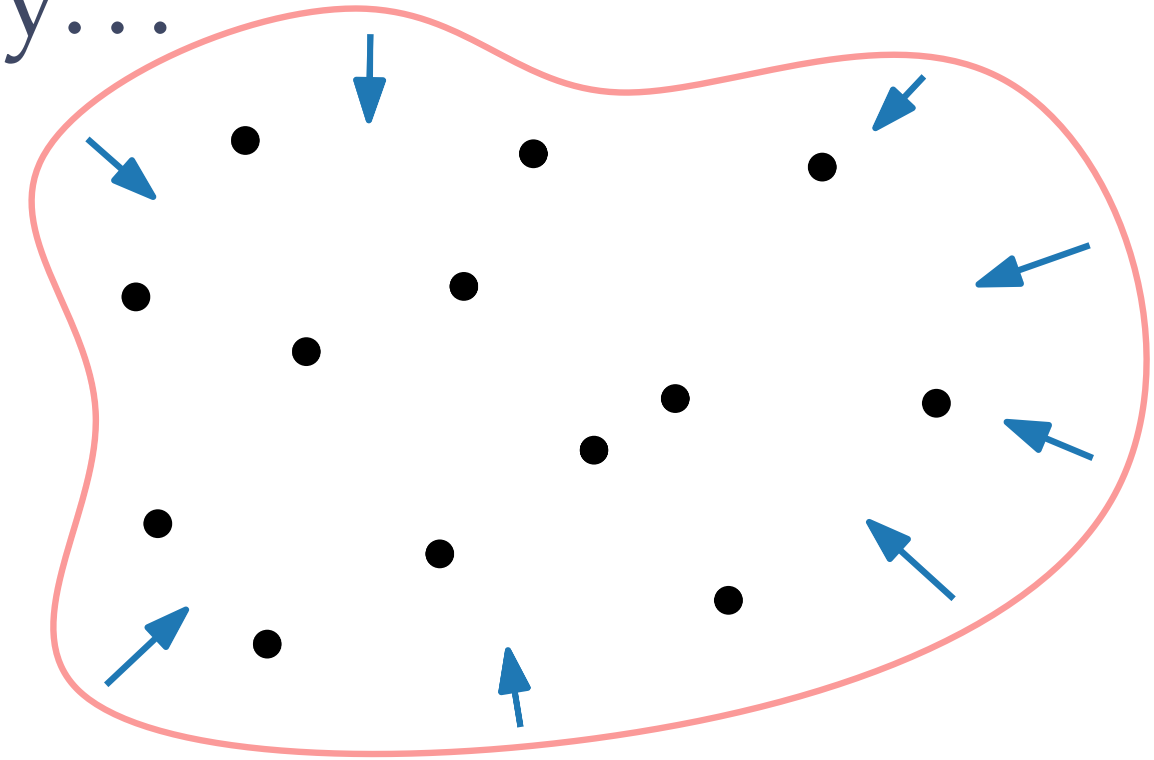
Formally...



Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach: – take (large enough) elastic rope

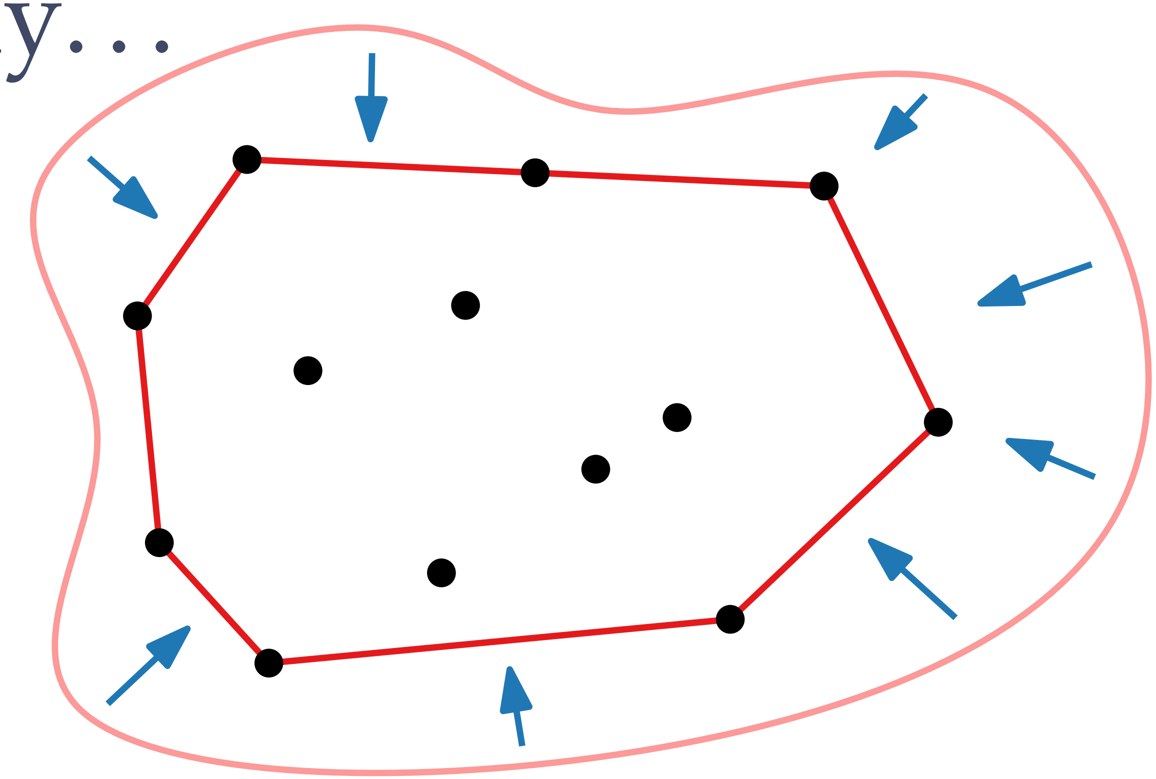
Formally. . .



Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach: – take (large enough) elastic rope
– stretch and let go

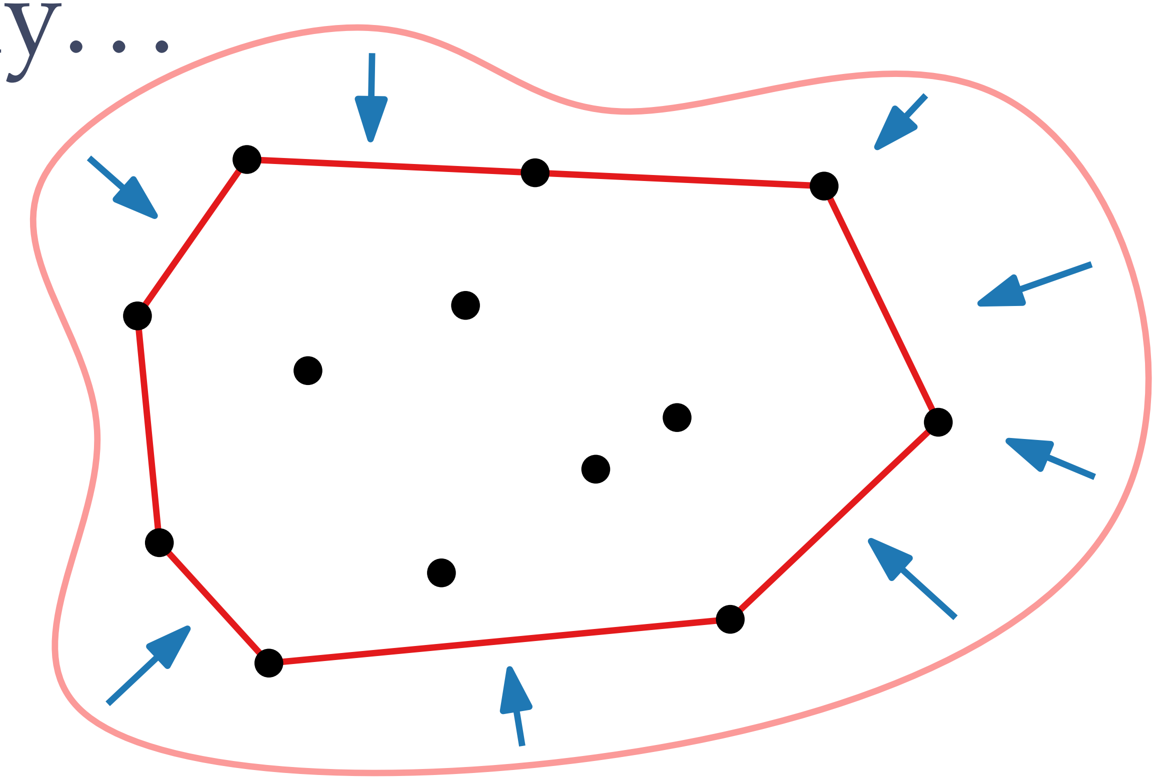
Formally…



Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach: – take (large enough) elastic rope
– stretch and let go

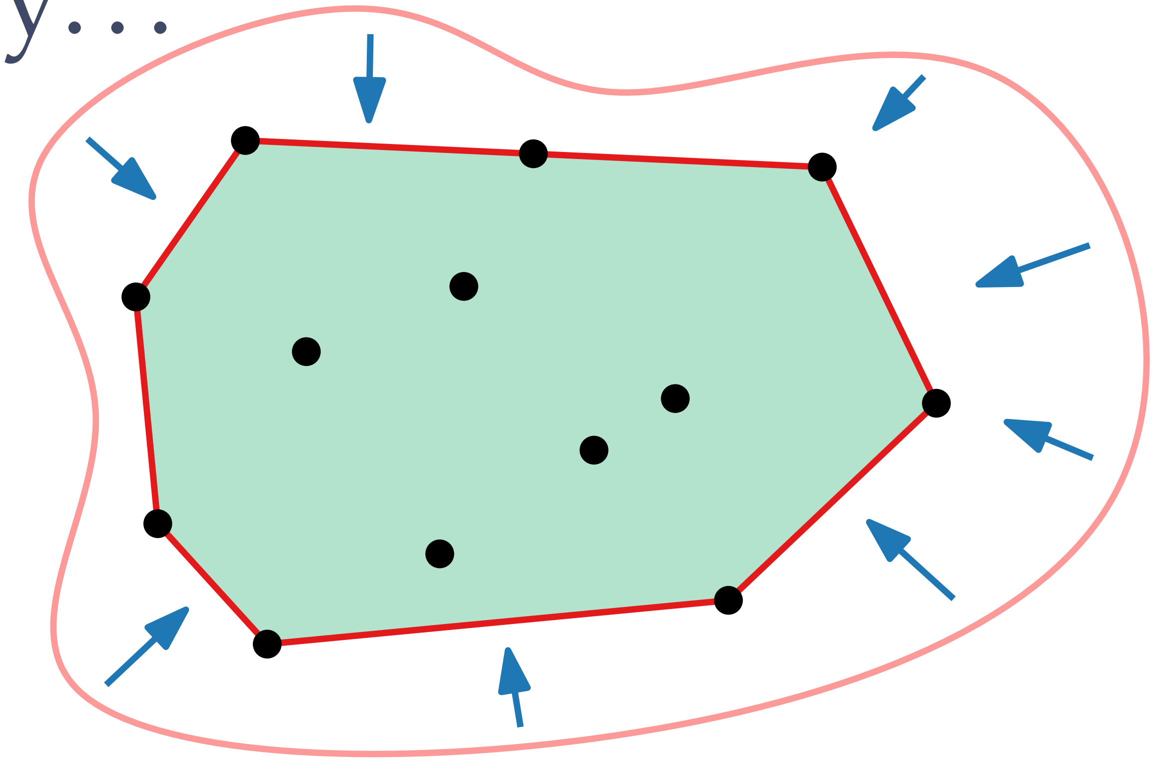
Formally...



Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach: – take (large enough) elastic rope
– stretch and let go
– take area inside (and on) the rope

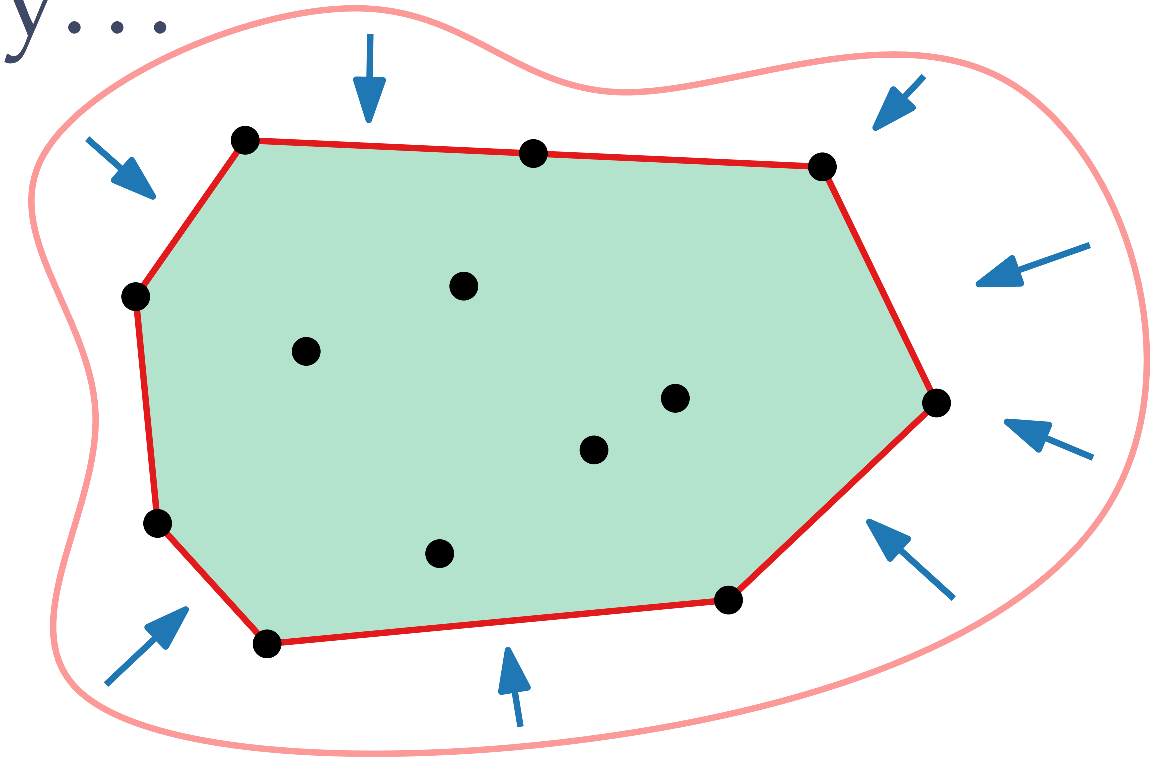
Formally. . .



Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach: – take (large enough) elastic rope
– stretch and let go
– take area inside (and on) the rope

Formally...

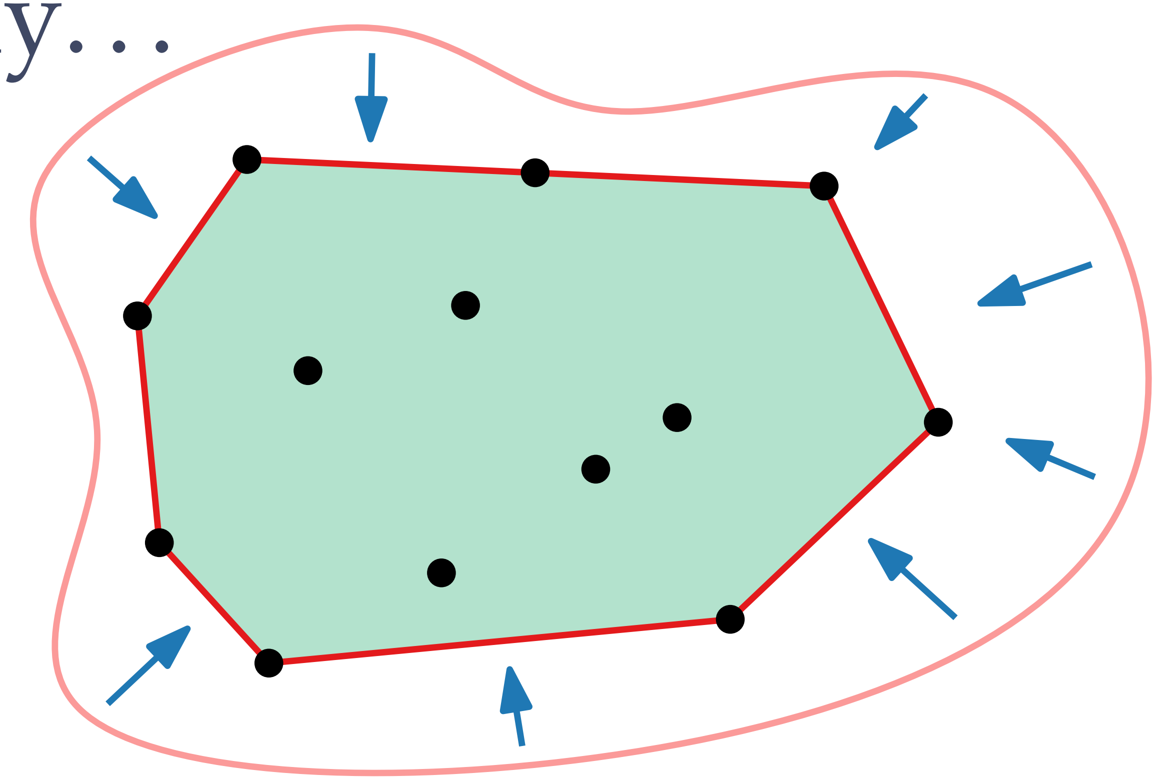


Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach: – take (large enough) elastic rope
– stretch and let go
– take area inside (and on) the rope

Maths approach:

Formally...

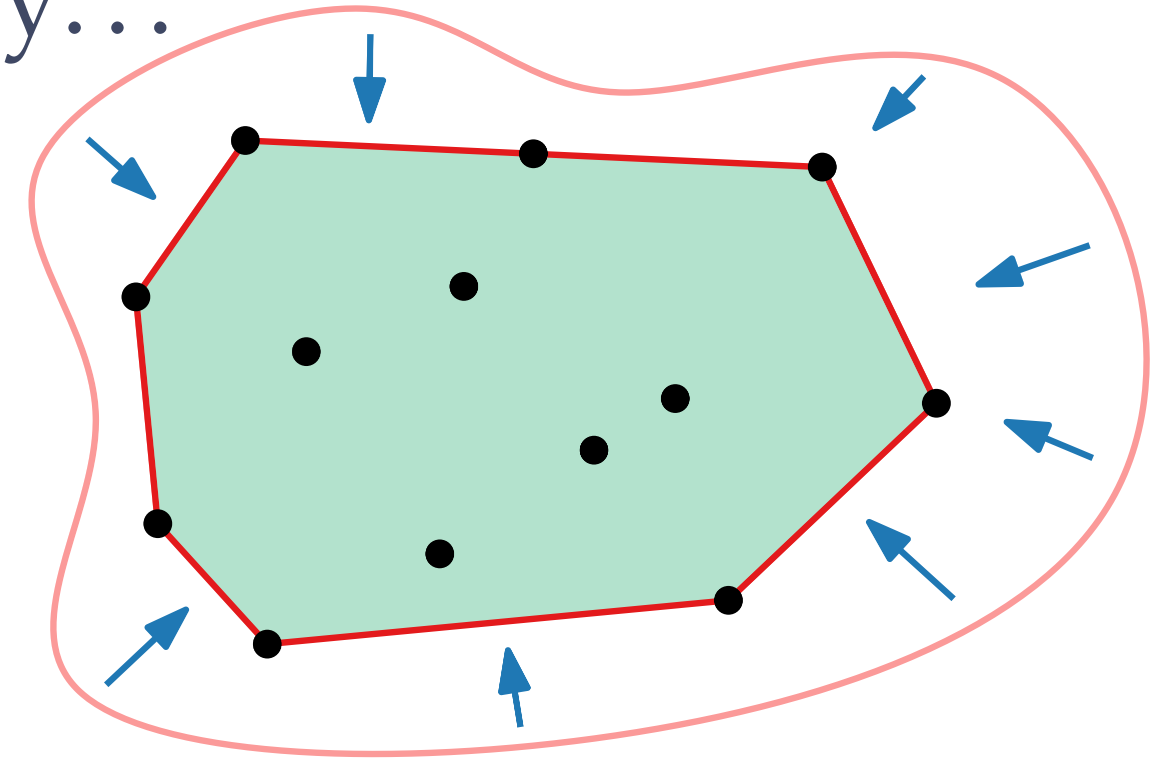


Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach: – take (large enough) elastic rope
– stretch and let go
– take area inside (and on) the rope

Maths approach: – define *convex*

Formally. . .



Given $S \subset \mathbb{R}^2$, how do we define the *convex hull* $\mathrm{CH}(S)$?

Physics approach: – take (large enough) elastic rope
– stretch and let go
– take area inside (and on) the rope

Maths approach: – define *convex*
– define $\mathrm{CH}(S) = \bigcap_{C \supseteq S\,:\, C \text{ convex}} C$

Towards Computation

$$\mathrm{CH}(S) \stackrel{\text{def}}{=} \bigcap_{C \supseteq S:\ C \text{ convex}} C$$

Problem with maths approach:

Towards Computation

$$\mathrm{CH}(S) \stackrel{\text{def}}{=} \bigcap_{C \supseteq S\,:\; C \text{ convex}} C$$

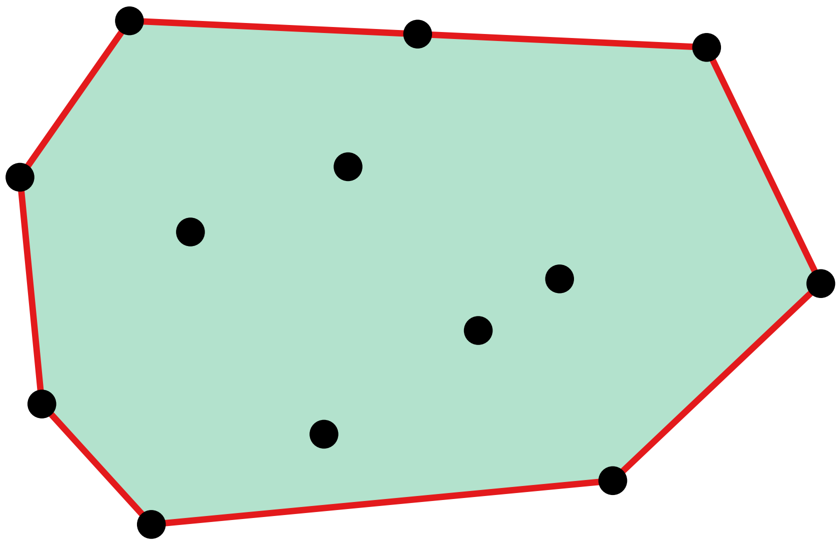
Problem with maths approach: *This set is* HUGE!

Towards Computation

$$\mathrm{CH}(S) \stackrel{\text{def}}{=} \bigcap_{C \supseteq S:\, C \text{ convex}} C$$

This set is HUGE!

Problem with maths approach:

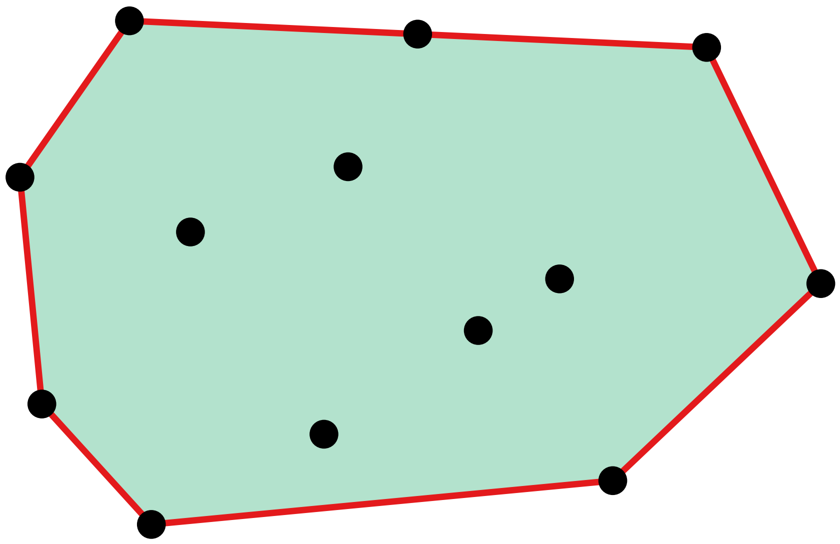


Maybe we can do with a little less?

Towards Computation

$$\mathrm{CH}(S) \stackrel{\text{def}}{=} \bigcap_{C \supseteq S:\, C \text{ convex}} C$$

Problem with maths approach: *This set is* HUGE!



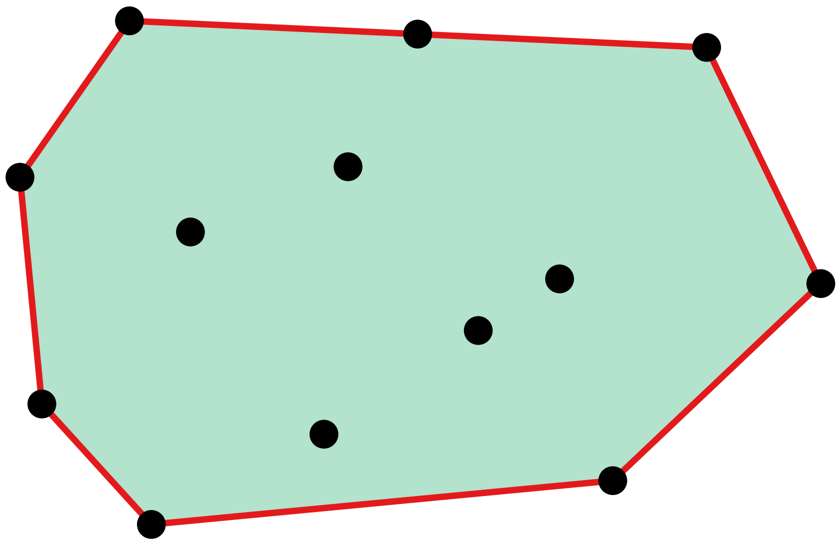
Maybe we can do with a little less?

Claim: $\mathrm{CH}(S) =$

Towards Computation

$$\mathrm{CH}(S) \stackrel{\text{def}}{=} \bigcap_{C \supseteq S:\, C \text{ convex}} C$$

Problem with maths approach: *This set is* HUGE!



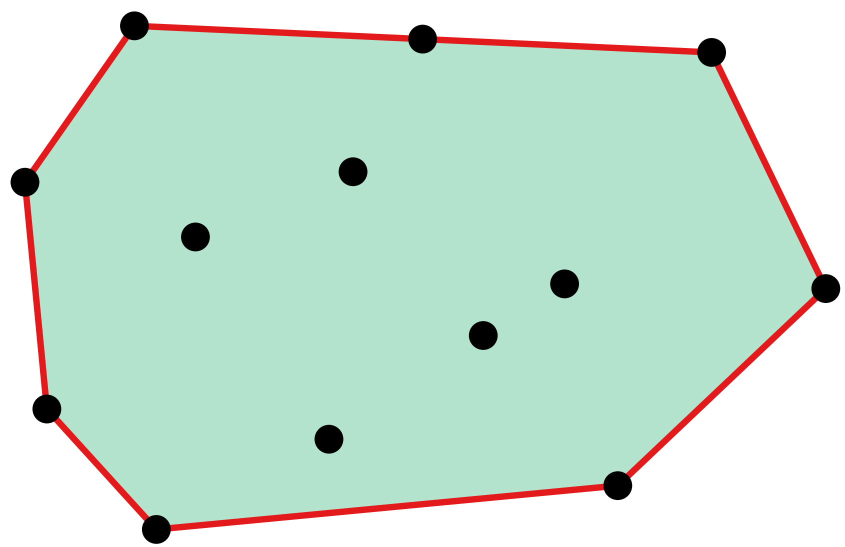
Maybe we can do with a little less?

Claim: $$\mathrm{CH}(S) = \bigcap_{\substack{H \supseteq S: \\ H \text{ closed halfplane}}} H$$

Towards Computation

$$\mathrm{CH}(S) \stackrel{\text{def}}{=} \bigcap_{C \supseteq S:\ C \text{ convex}} C$$

Problem with maths approach: *This set is* **HUGE!**



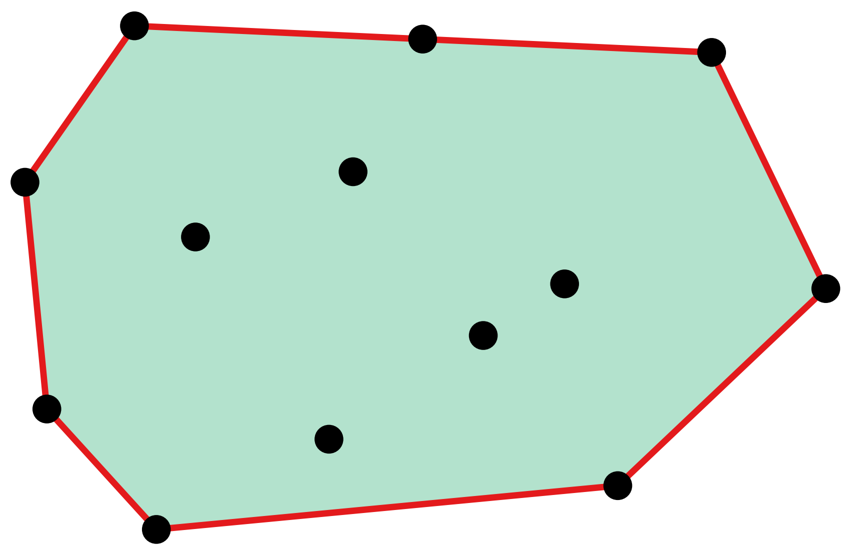
Maybe we can do with a little less?

Claim: $$\mathrm{CH}(S) = \bigcap_{\substack{H \supseteq S: \\ H \text{ closed halfplane}}} H =$$

Towards Computation

$$\mathrm{CH}(S) \stackrel{\text{def}}{=} \bigcap_{C \supseteq S:\ C \text{ convex}} C$$

Problem with maths approach: *This set is* HUGE!

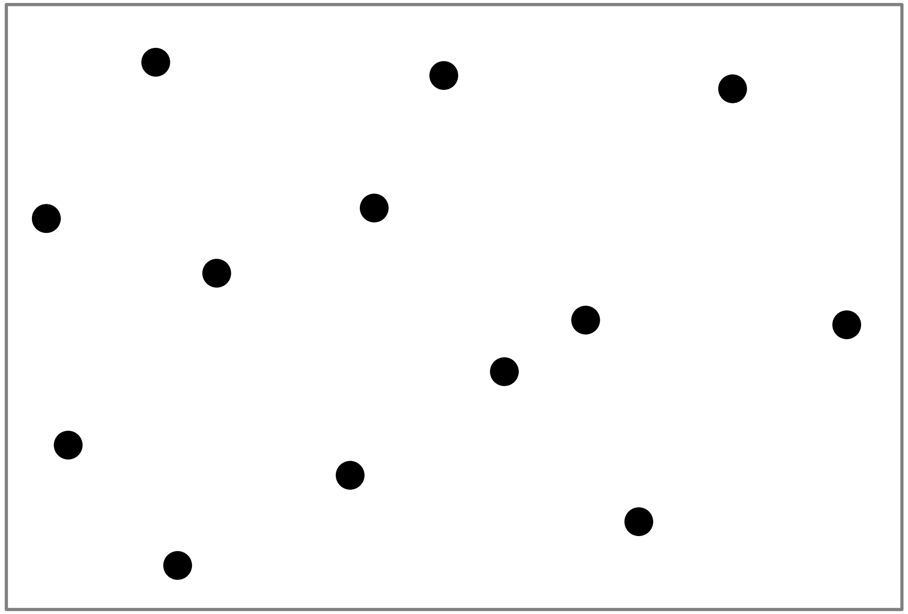


Maybe we can do with a little less?

Claim: $$\mathrm{CH}(S) = \bigcap_{\substack{H \supseteq S: \\ H \text{ closed halfplane}}} H = \bigcap_{\substack{H \supseteq S:\ H \text{ cl. halfplane}, \\ |\partial H \cap S| \geq 2}} H$$

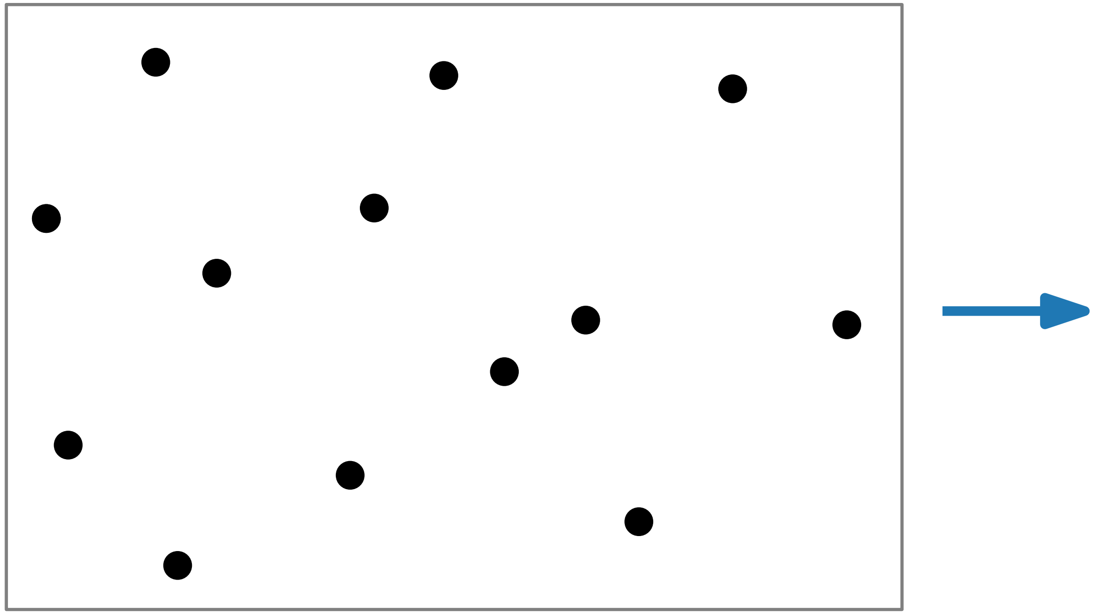
Algorithmic Approach

Input: set S of n points in the plane, that is, $S \subset \mathbb{R}^2$



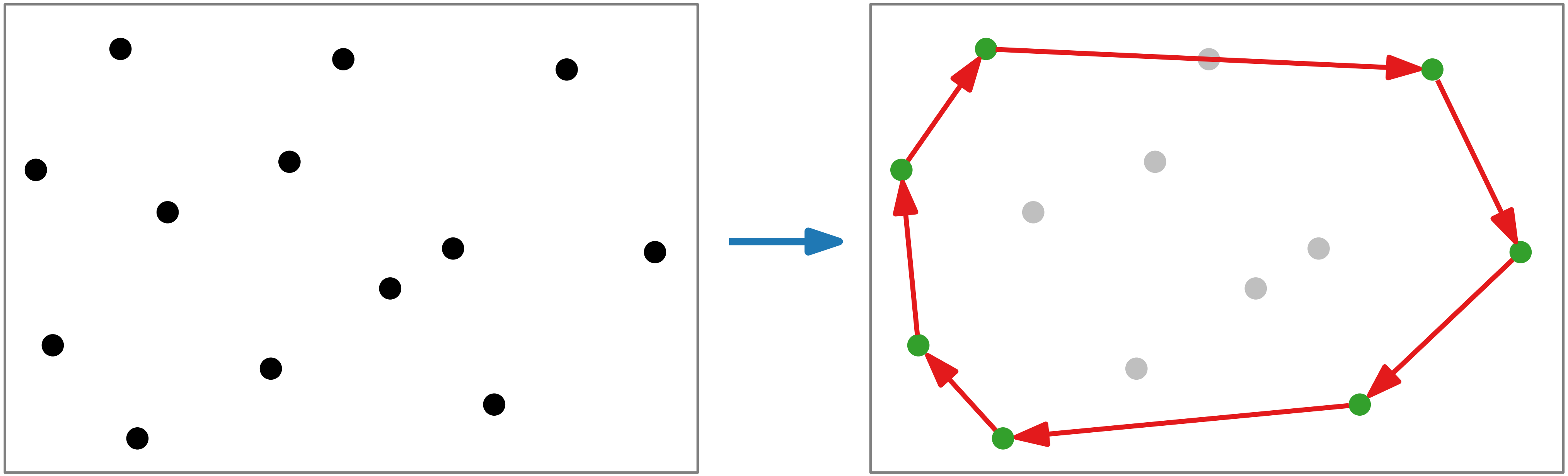
Algorithmic Approach

Input: set S of n points in the plane, that is, $S \subset \mathbb{R}^2$



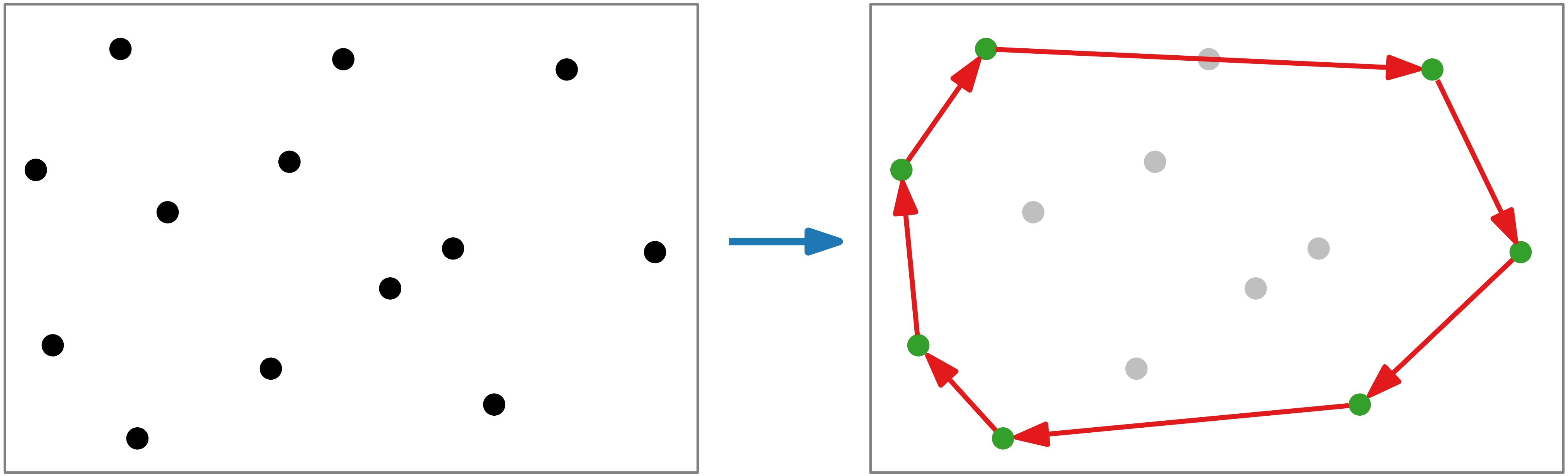
Algorithmic Approach

Input: set S of n points in the plane, that is, $S \subset \mathbb{R}^2$



Algorithmic Approach

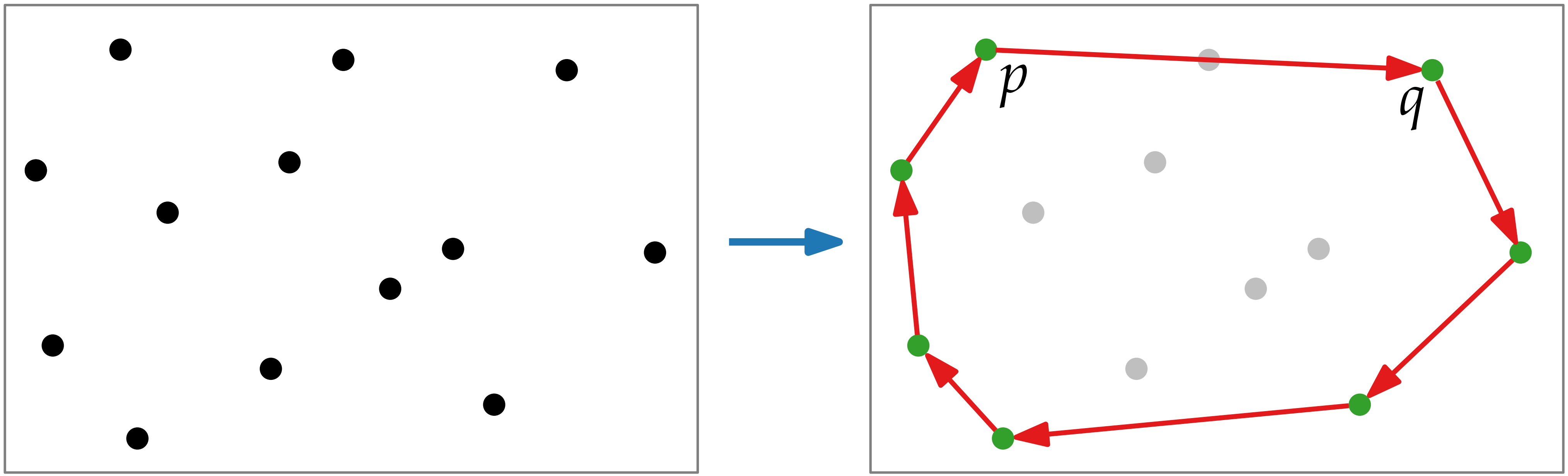
Input: set S of n points in the plane, that is, $S \subset \mathbb{R}^2$



Output: list of vertices of $\mathrm{CH}(S)$ in clockwise order

Algorithmic Approach

Input: set S of n points in the plane, that is, $S \subset \mathbb{R}^2$

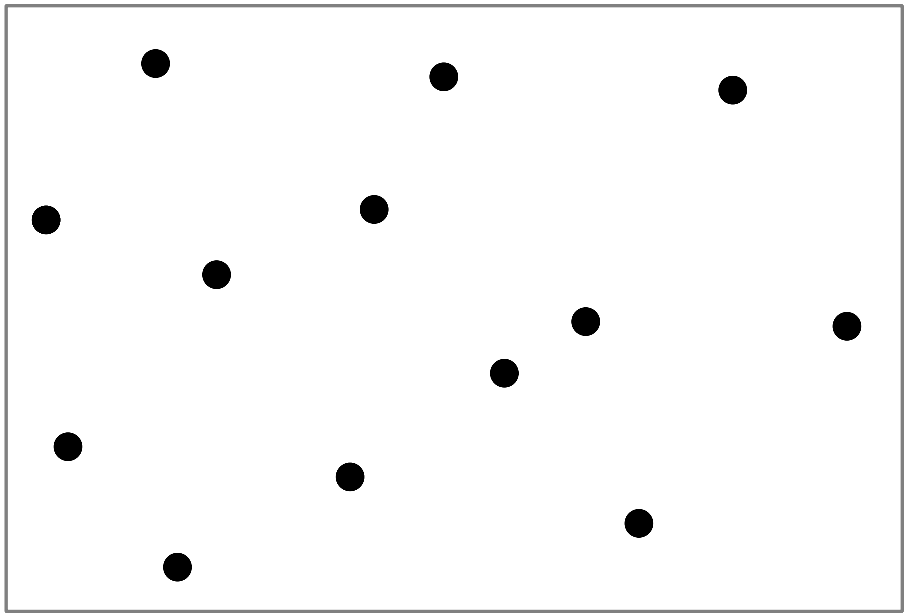


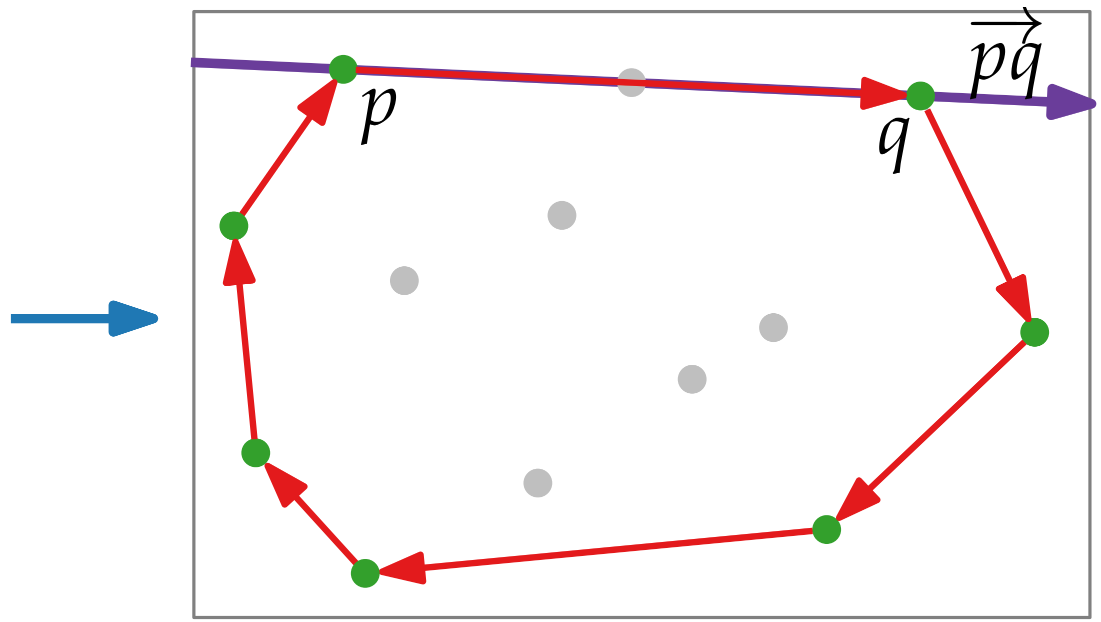
Output: list of vertices of $\mathrm{CH}(S)$ in clockwise order

Observation. (p, q) is an edge of $\mathrm{CH}(S)$ $\Leftrightarrow$

Algorithmic Approach

Input: set S of n points in the plane, that is, $S \subset \mathbb{R}^2$





Output: list of vertices of $\mathrm{CH}(S)$ in clockwise order

Observation. (p, q) is an edge of $\mathrm{CH}(S)$ $\Leftrightarrow$
each point in S lies
– strictly to the right of the directed line $\overrightarrow{pq}$ or
– on the line segment $\overline{pq}$

Finally, an Algorithm

FirstConvexHull(S)

$E \leftarrow \emptyset$
foreach $(p, q) \in S \times S$ with $p \neq q$ **do**
    valid $\leftarrow$ *true*
    foreach $r \in S$ **do**
        if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
            valid $\leftarrow$ *false*
    if *valid* **then**
        $E \leftarrow E \cup \{(p, q)\}$
from E construct sorted list L of vertices of CH(S)
return L

Finally, an Algorithm

FirstConvexHull(S)

$E \leftarrow \emptyset$
foreach $(p, q) \in S \times S$ with $p \neq q$ **do** // test: (p, q) edge of $\mathrm{CH}(S)$?
  valid $\leftarrow$ *true*
  foreach $r \in S$ **do**
    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
      valid $\leftarrow$ *false*
  if *valid* **then**
    $E \leftarrow E \cup \{(p, q)\}$
from E construct sorted list L of vertices of $\mathrm{CH}(S)$
return L

Finally, an Algorithm

```
FirstConvexHull(S)
  E ← ∅
  foreach (p, q) ∈ S × S with p ≠ q do   // test: (p, q) edge of CH(S)?
      valid ← true
      foreach r ∈ S do
          if not (r strictly right of →pq or r ∈ ‾pq) then
              valid ← false
      if valid then
          E ← E ∪ {(p, q)}
  from E construct sorted list L of vertices of CH(S)
  return L
```

Finally, an Algorithm

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ **do**

  $\mathit{valid} \leftarrow \mathit{true}$

  foreach $r \in S$ **do**

    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**

      $\mathit{valid} \leftarrow \mathit{false}$

  if valid **then**

    $E \leftarrow E \cup \{(p, q)\}$

from E construct sorted list L of vertices of CH(S)

return L

r strictly right of $\overrightarrow{pq}$

$$\Updownarrow$$

$$\begin{vmatrix} x_r & y_r & 1 \\ x_p & y_p & 1 \\ x_q & y_q & 1 \end{vmatrix} < 0$$

Finally, an Algorithm

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ **do**

  valid $\leftarrow$ *true*

  foreach $r \in S$ **do**

    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**

      valid $\leftarrow$ *false*

  if *valid* **then**

    $E \leftarrow E \cup \{(p, q)\}$

from E construct sorted list L of vertices of CH(S)

return L

r strictly right of $\overrightarrow{pq}$

$$\Updownarrow$$

$$\begin{vmatrix} x_r & y_r & 1 \\ x_p & y_p & 1 \\ x_q & y_q & 1 \end{vmatrix} < 0$$

Important:
Test takes $O(1)$ time!

Finally, an Algorithm

FirstConvexHull(S)

$E \leftarrow \emptyset$
foreach $(p, q) \in S \times S$ with $p \neq q$ **do**
    $valid \leftarrow true$
    foreach $r \in S$ **do**
        if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
            $valid \leftarrow false$
    if $valid$ **then**
        $E \leftarrow E \cup \{(p, q)\}$
from E construct sorted list L of vertices of CH(S)
return L

r strictly right of $\overrightarrow{pq}$

$$\Updownarrow$$

$$\begin{vmatrix} x_r & y_r & 1 \\ x_p & y_p & 1 \\ x_q & y_q & 1 \end{vmatrix} < 0$$

Important:
Test takes $O(1)$ time!

Finally, an Algorithm

FirstConvexHull(S)

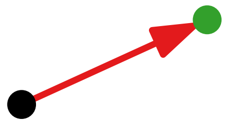
$E \leftarrow \emptyset$
foreach $(p, q) \in S \times S$ with $p \neq q$ **do**
  valid $\leftarrow$ *true*
  foreach $r \in S$ **do**
    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
      valid $\leftarrow$ *false*
  if *valid* **then**
    $E \leftarrow E \cup \{(p, q)\}$
from E construct sorted list L of vertices of CH(S)
return L

r strictly right of $\overrightarrow{pq}$

$$\Updownarrow$$

$$\begin{vmatrix} x_r & y_r & 1 \\ x_p & y_p & 1 \\ x_q & y_q & 1 \end{vmatrix} < 0$$

Important:
Test takes $O(1)$ time!



Finally, an Algorithm

FirstConvexHull(S)

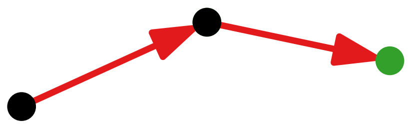
$E \leftarrow \emptyset$
foreach $(p,q) \in S \times S$ with $p \neq q$ **do**
  valid $\leftarrow$ *true*
  foreach $r \in S$ **do**
    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
      valid $\leftarrow$ *false*
  if *valid* **then**
    $E \leftarrow E \cup \{(p,q)\}$
from E construct sorted list L of vertices of CH(S)
return L

r strictly right of $\overrightarrow{pq}$

$\Updownarrow$

$$\begin{vmatrix} x_r & y_r & 1 \\ x_p & y_p & 1 \\ x_q & y_q & 1 \end{vmatrix} < 0$$

Important:
Test takes $O(1)$ time!



Finally, an Algorithm

FirstConvexHull(S)

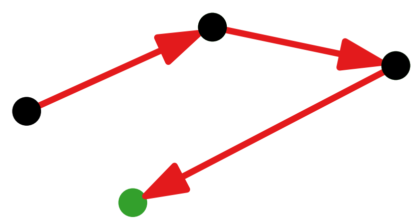
$E \leftarrow \emptyset$
foreach $(p, q) \in S \times S$ with $p \neq q$ **do**
  $valid \leftarrow true$
  foreach $r \in S$ **do**
    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
      $valid \leftarrow false$
  if $valid$ **then**
    $E \leftarrow E \cup \{(p, q)\}$
from E construct sorted list L of vertices of CH(S)
return L

r strictly right of $\overrightarrow{pq}$

$$\Updownarrow$$

$$\begin{vmatrix} x_r & y_r & 1 \\ x_p & y_p & 1 \\ x_q & y_q & 1 \end{vmatrix} < 0$$

Important:
Test takes $O(1)$ time!



Finally, an Algorithm

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p,q) \in S \times S$ with $p \neq q$ **do**

  valid $\leftarrow$ *true*

  foreach $r \in S$ **do**

    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**

      valid $\leftarrow$ *false*

  if *valid* **then**

    $E \leftarrow E \cup \{(p,q)\}$

from E construct sorted list L of vertices of CH(S)

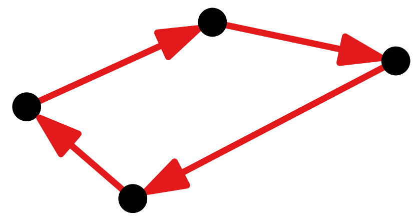
return L

r strictly right of $\overrightarrow{pq}$

$\Updownarrow$

$$\begin{vmatrix} x_r & y_r & 1 \\ x_p & y_p & 1 \\ x_q & y_q & 1 \end{vmatrix} < 0$$

Important:
Test takes $O(1)$ time!



Running Time Analysis

```
FirstConvexHull(S)
  E ← ∅
  foreach (p, q) ∈ S × S with p ≠ q do
    valid ← true
    foreach r ∈ S do
      if not (r strictly right of →pq or r ∈ ‾pq) then
        valid ← false
    if valid then
      E ← E ∪ {(p, q)}
  from E construct sorted list L of vertices of CH(S)
  return L
```

Running Time Analysis

```
FirstConvexHull(S)
  E ← ∅
  foreach (p, q) ∈ S × S with p ≠ q do
    valid ← true
    foreach r ∈ S do
      if not (r strictly right of →pq or r ∈ ‾pq) then
        valid ← false
    if valid then
      E ← E ∪ {(p, q)}
  from E construct sorted list L of vertices of CH(S)
  return L
```

Running Time Analysis

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ **do**

$\quad$ *valid* $\leftarrow$ *true*

$\quad$ **foreach** $r \in S$ **do**

$\quad\quad$ **if not** (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**

$\quad\quad\quad$ *valid* $\leftarrow$ *false*

$\quad\quad$ $\Theta(1)$

$\quad$ **if** *valid* **then**

$\quad\quad$ $E \leftarrow E \cup \{(p, q)\}$

from E construct sorted list L of vertices of CH(S)

return L

Running Time Analysis

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ **do**

  valid $\leftarrow$ *true*

  foreach $r \in S$ **do**

    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**

      valid $\leftarrow$ *false*

  $\Theta(1)$

  if *valid* **then**

    $E \leftarrow E \cup \{(p, q)\}$

$\Theta(n)$

from E construct sorted list L of vertices of CH(S)

return L

Running Time Analysis

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ **do** $(n^2 - n) \cdot$

valid $\leftarrow$ *true*

foreach $r \in S$ **do**

if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**

valid $\leftarrow$ *false*

$\Theta(1)$

$\Theta(n)$

if *valid* **then**

$E \leftarrow E \cup \{(p, q)\}$

from E construct sorted list L of vertices of CH(S)

return L

Running Time Analysis

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ **do** $(n^2 - n)\cdot$

valid $\leftarrow$ true

foreach $r \in S$ **do**

if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**

valid $\leftarrow$ false

$\Theta(1)$ $\Theta(n)$ $\Theta(n^3)$

if valid **then**

$E \leftarrow E \cup \{(p, q)\}$

from E construct sorted list L of vertices of CH(S)

return L

Running Time Analysis

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ **do** $(n^2 - n) \cdot$

 valid $\leftarrow$ *true*

 foreach $r \in S$ **do**

  if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**

   valid $\leftarrow$ *false*

 if *valid* **then**

  $E \leftarrow E \cup \{(p, q)\}$

from E construct sorted list L of vertices of CH(S)

return L

$\Theta(1)$ $\Theta(n)$ $\Theta(n^3)$ $O(n^2)$

Running Time Analysis

FirstConvexHull(S)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ **do** $(n^2 - n)\cdot$

  valid $\leftarrow$ *true*

  foreach $r \in S$ **do**

    if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then** $\Theta(1)$

      valid $\leftarrow$ *false*

  $\Theta(n)$

  if *valid* **then**

    $E \leftarrow E \cup \{(p, q)\}$

$\Theta(n^3)$

from E construct sorted list L of vertices of CH(S) $O(n^2)$

return L

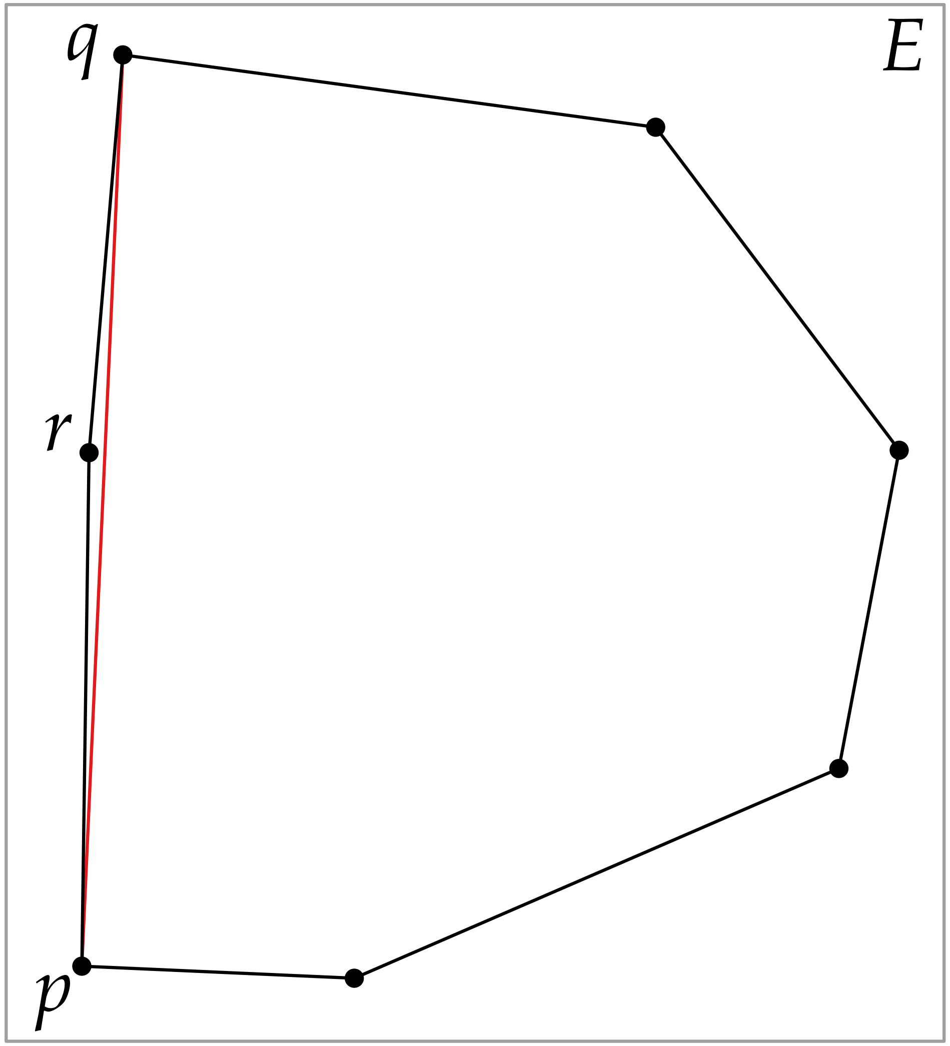
Lemma. We can compute the convex hull of n pts in the plane in $\Theta(n^3)$ time.

Discussion

if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
  $valid \leftarrow false$

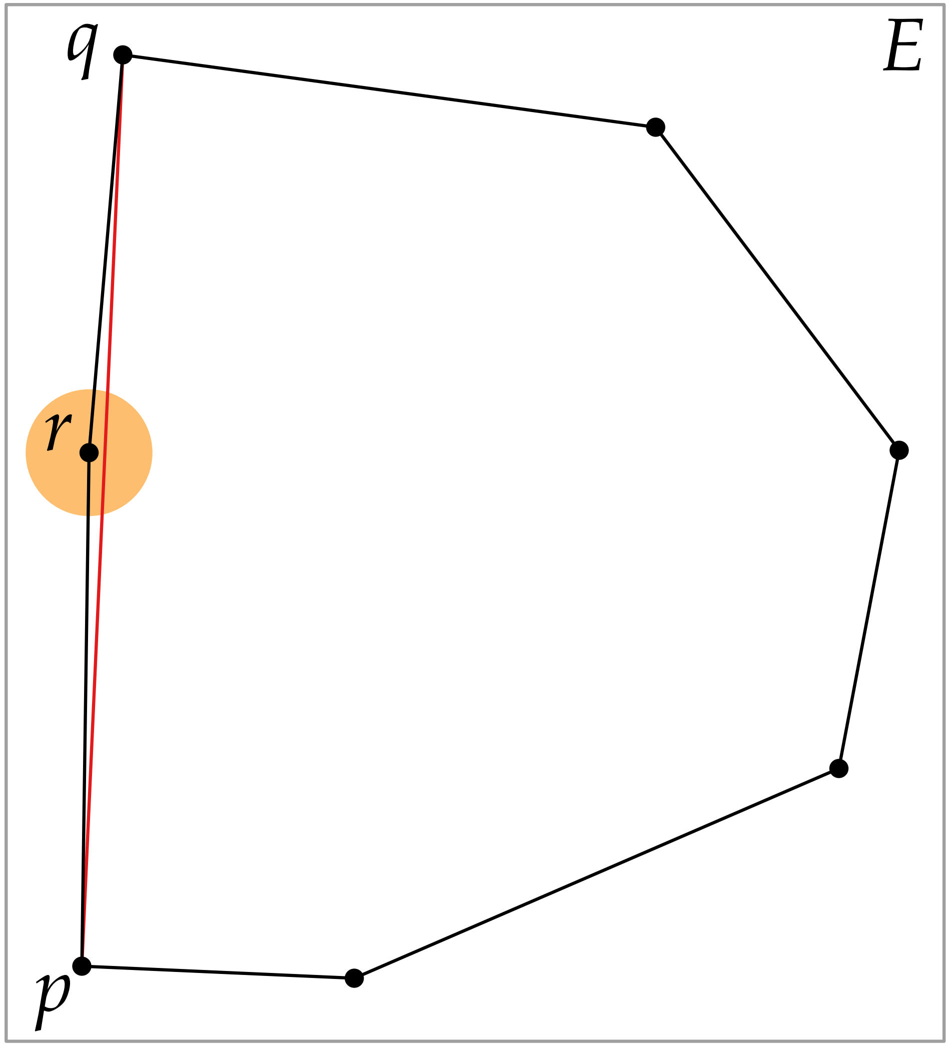
Discussion

if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
  $\textit{valid} \leftarrow \textit{false}$



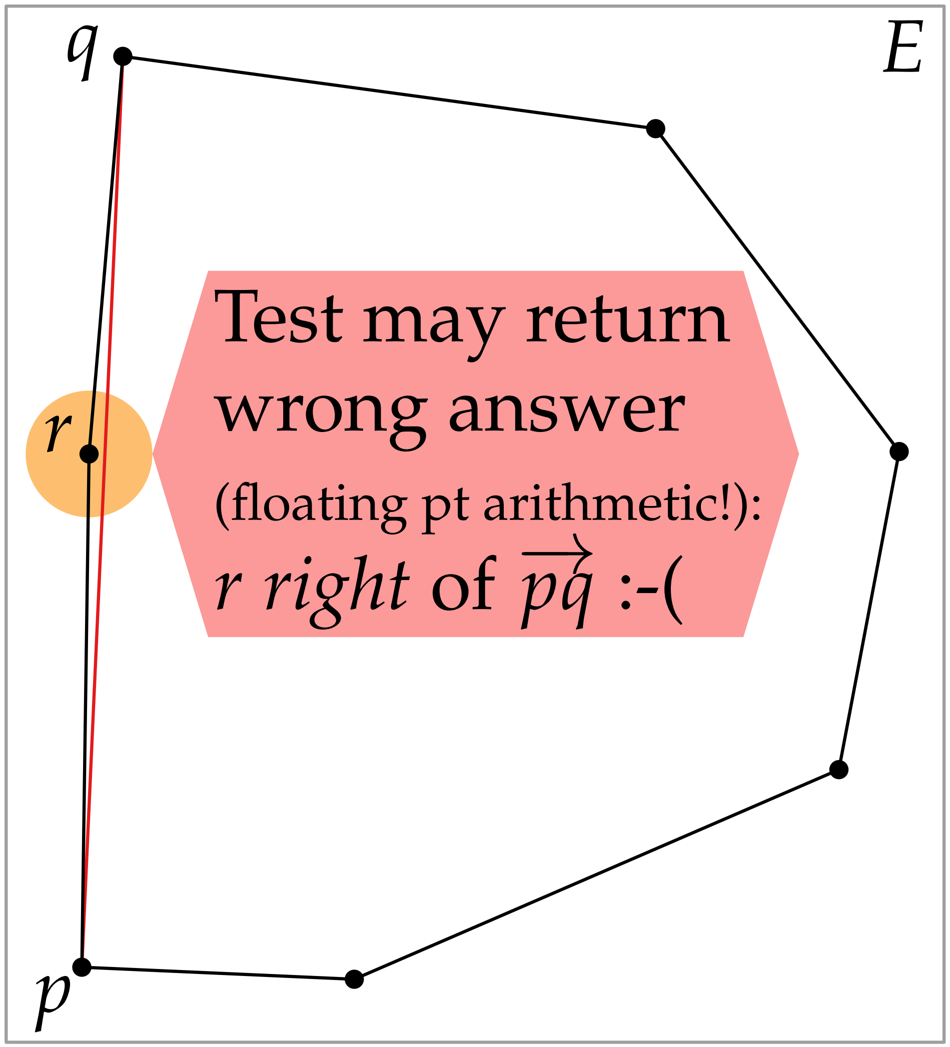
Discussion

if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
  $valid \leftarrow false$



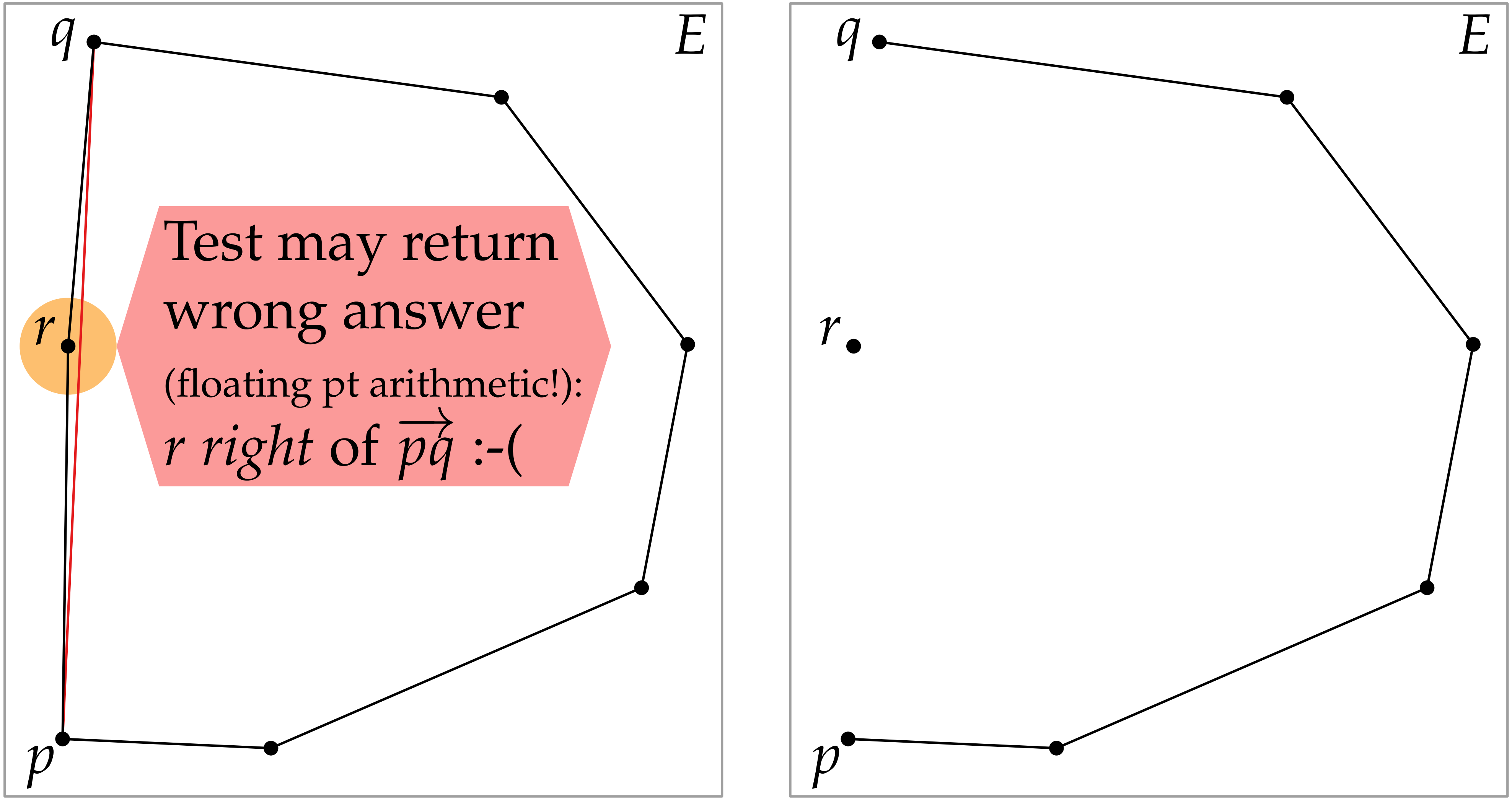
Discussion

if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
  $\quad$ *valid* $\leftarrow$ *false*



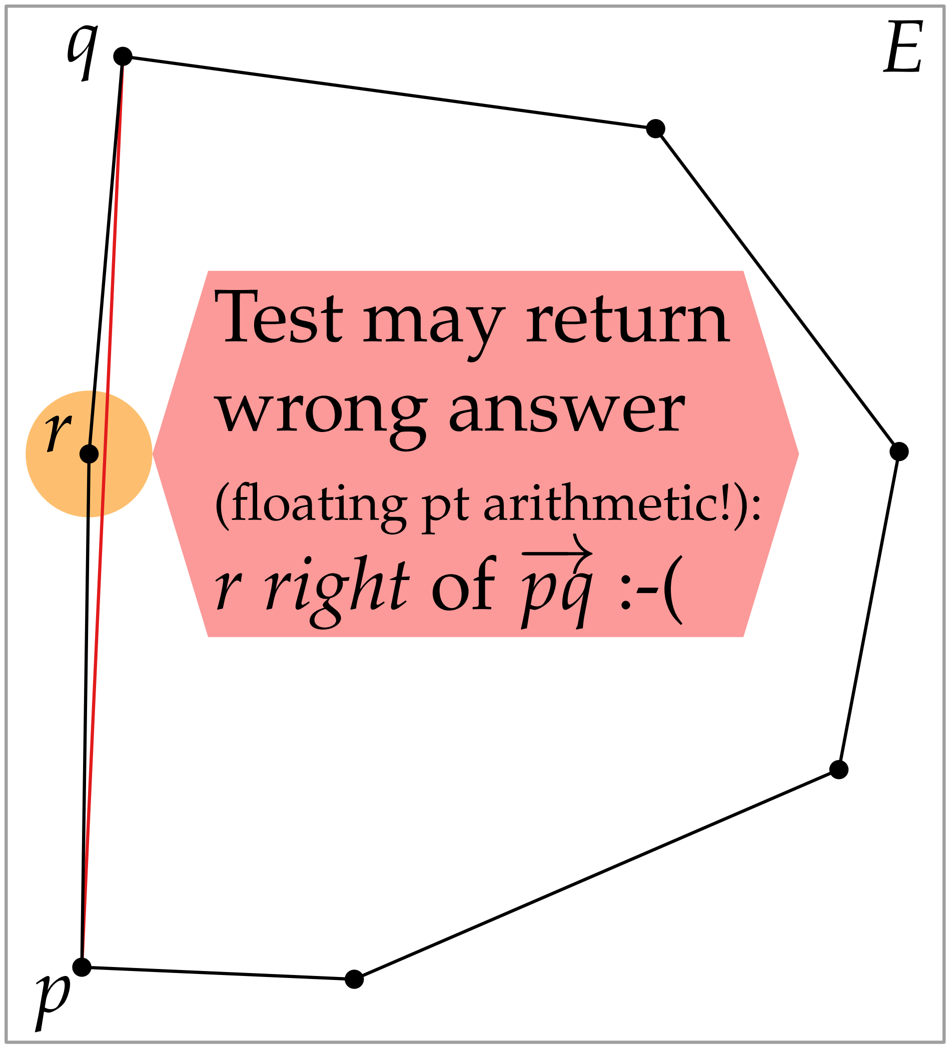
Discussion

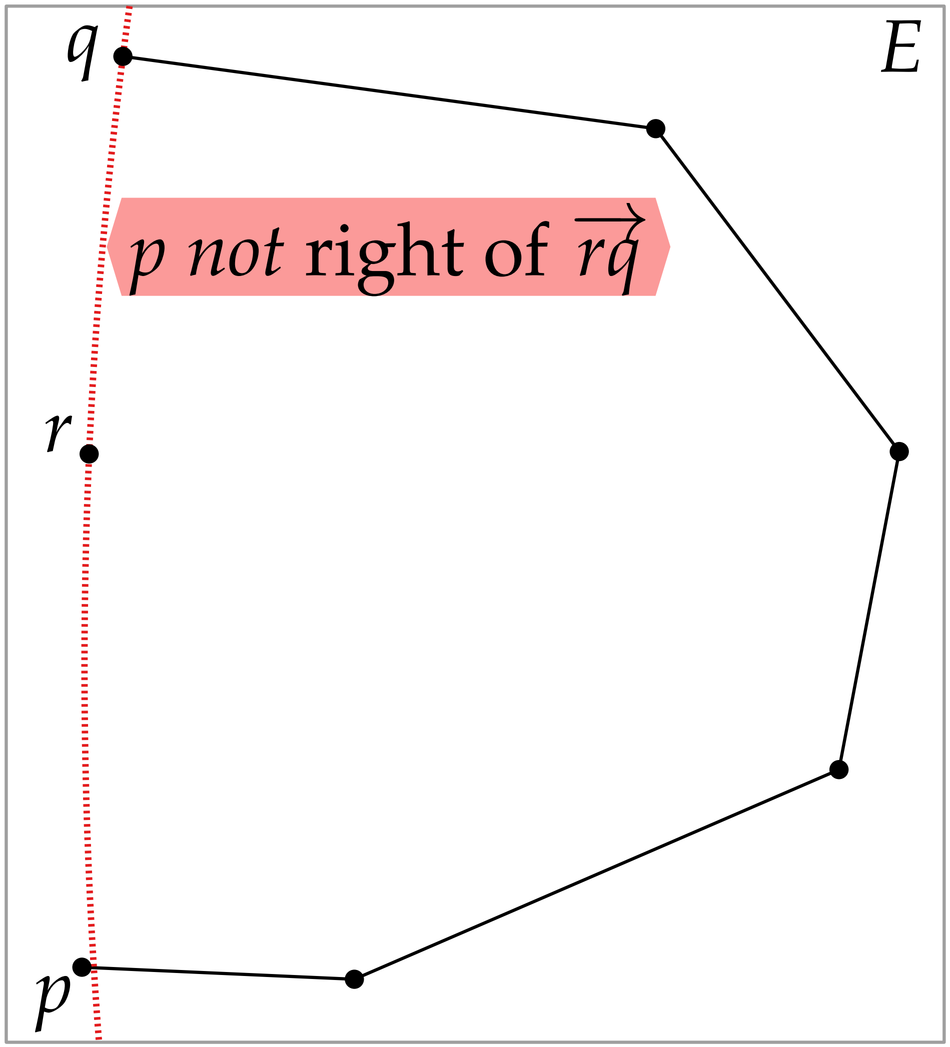
if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
  valid ← *false*



Discussion

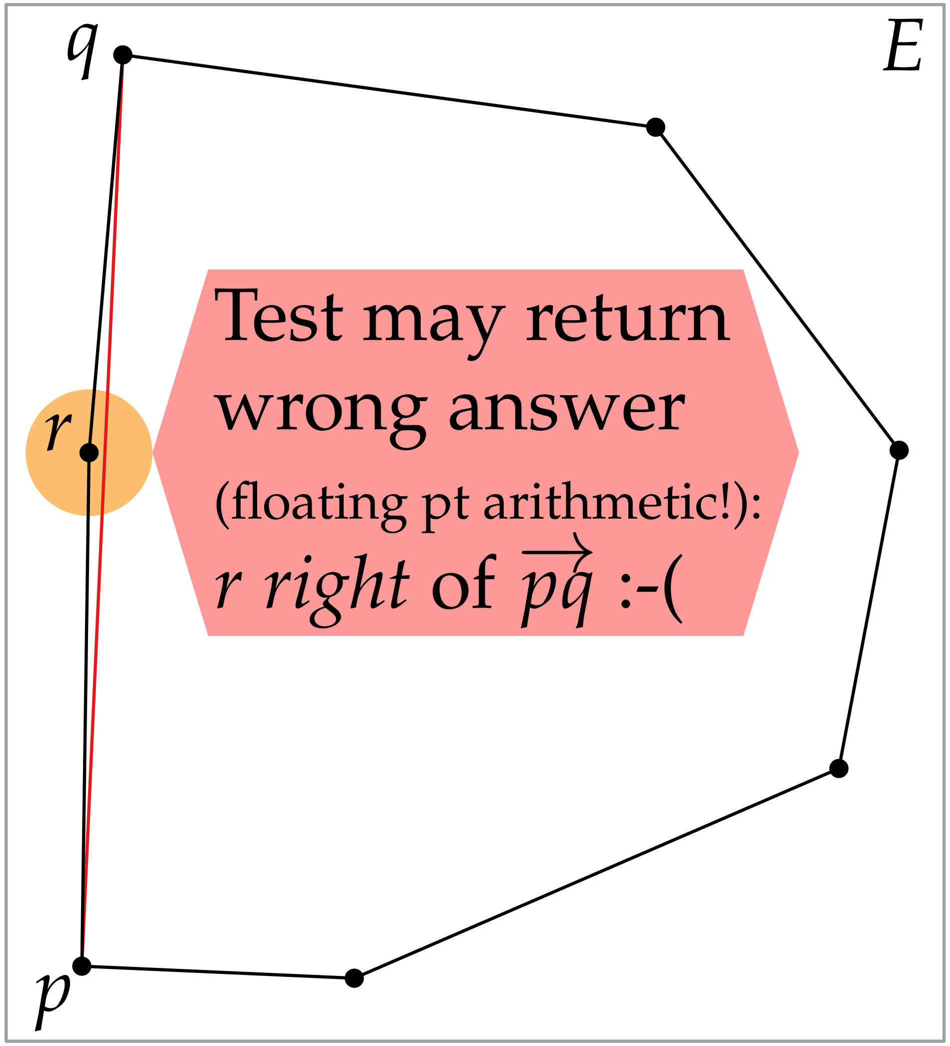
if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
valid $\leftarrow$ *false*

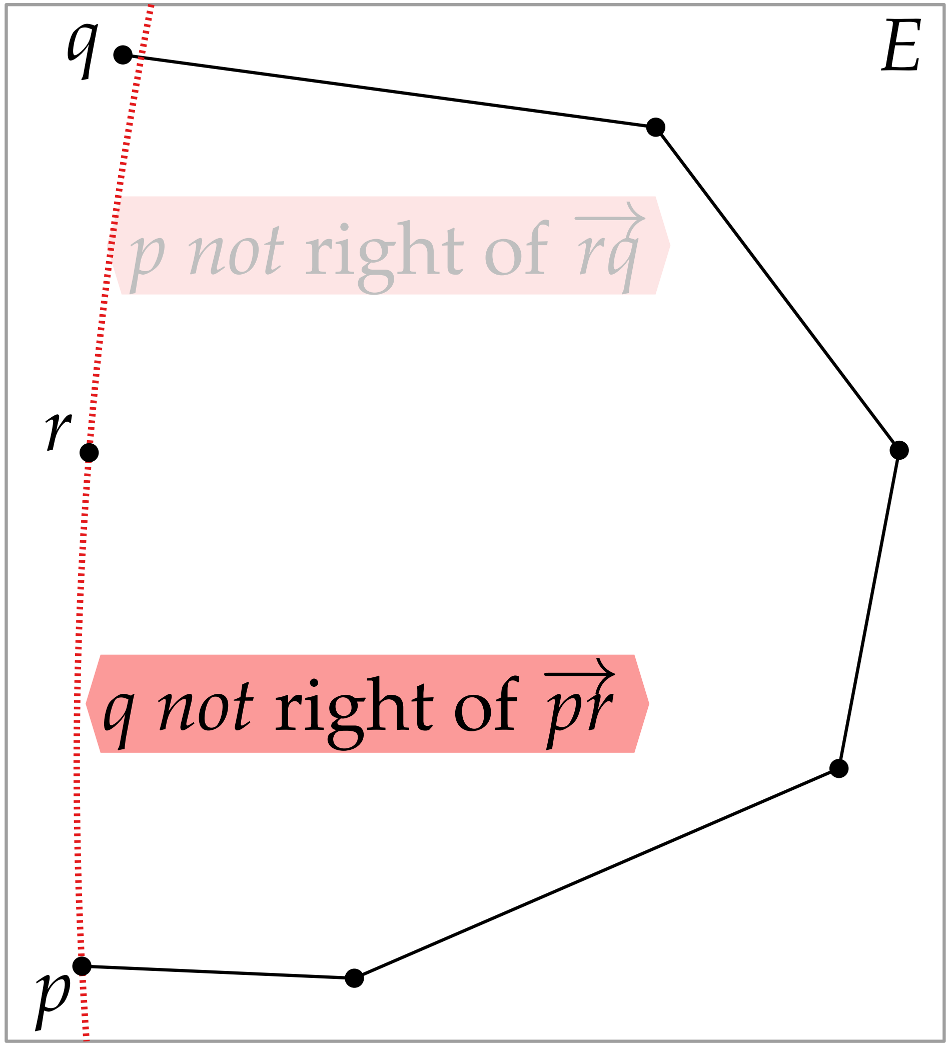




Discussion

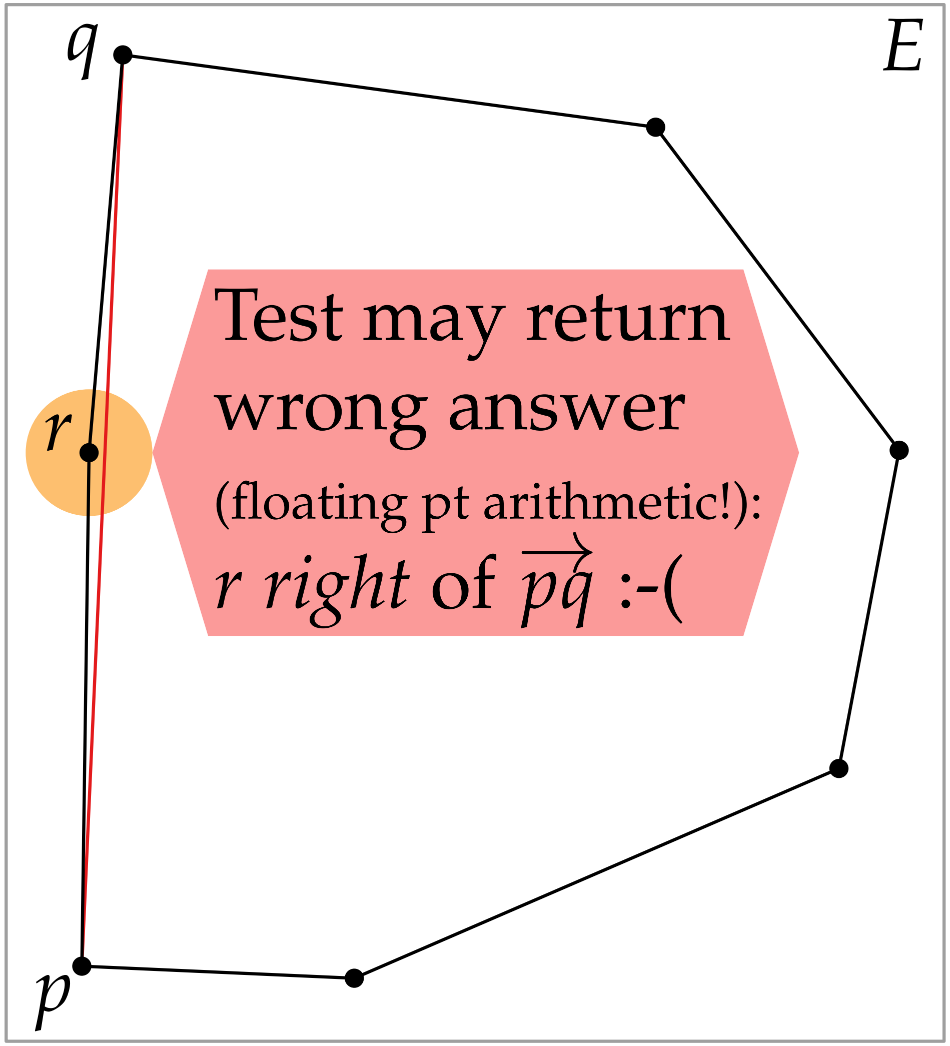
if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
  valid $\leftarrow$ *false*

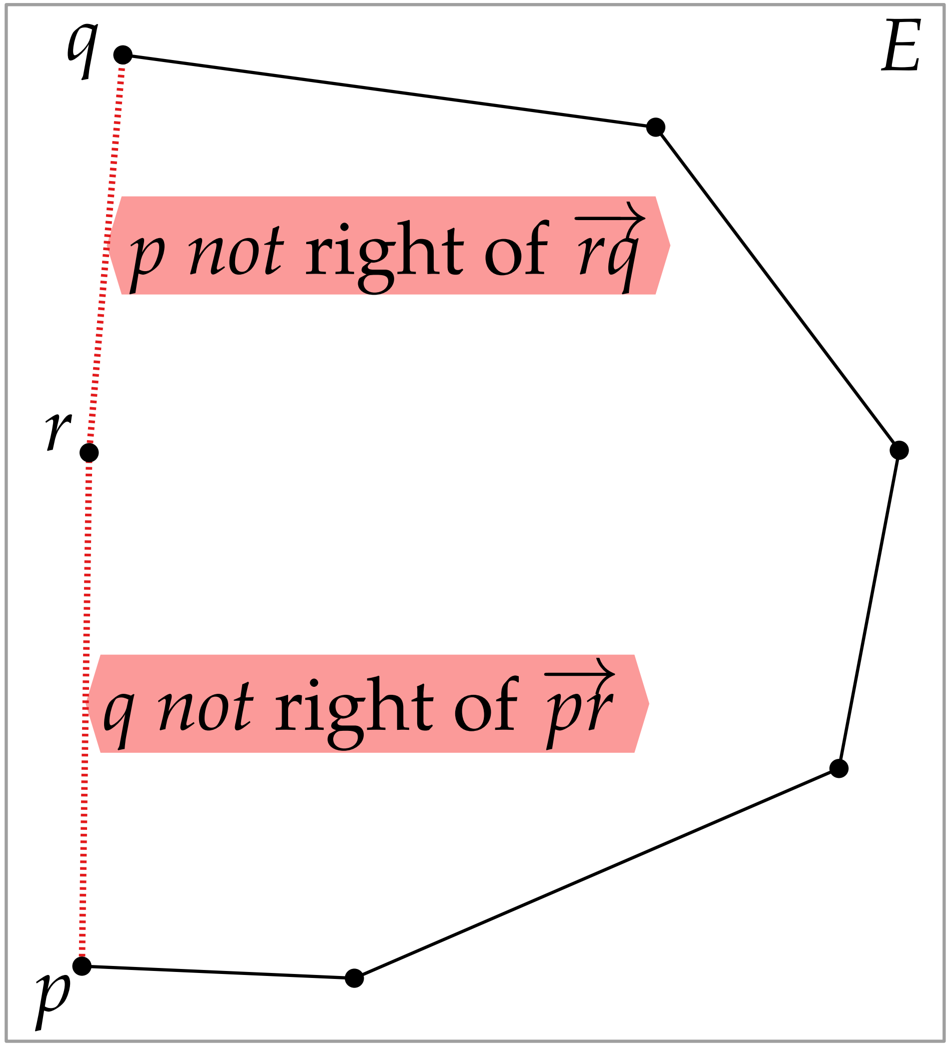




Discussion

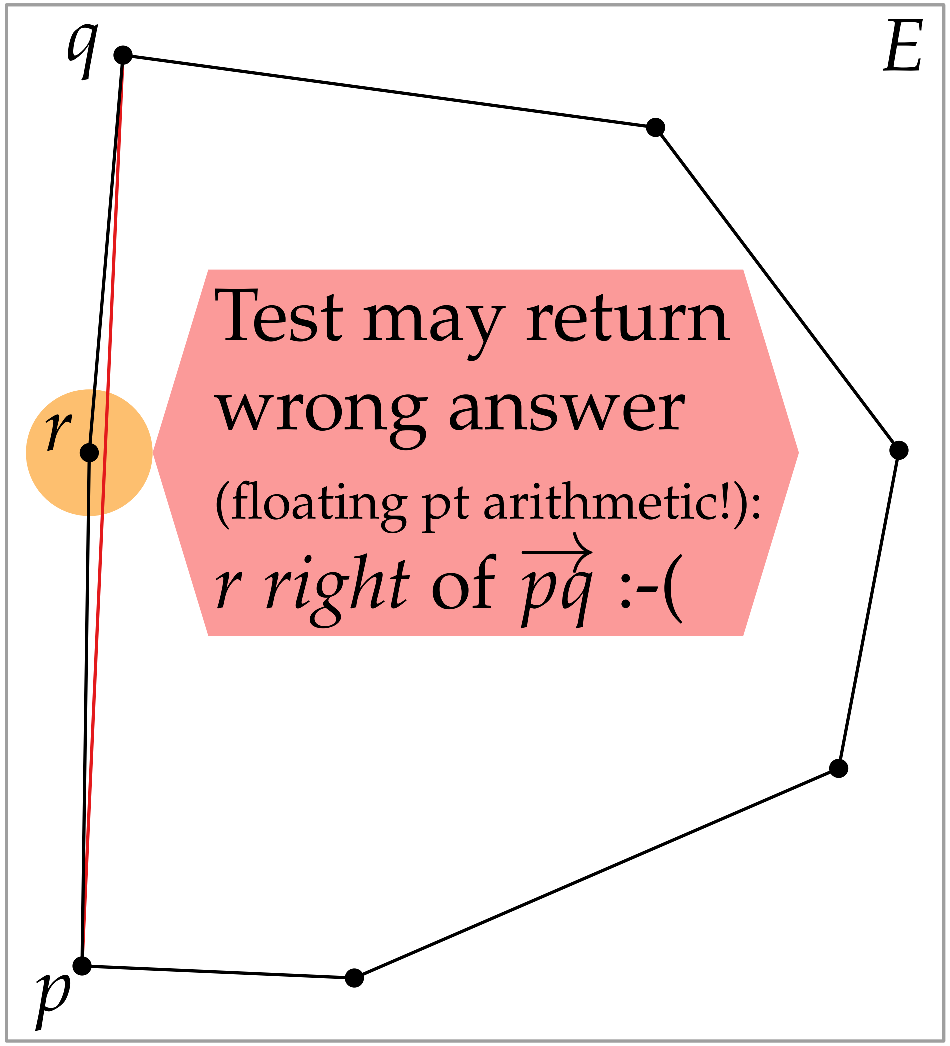
if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
  valid $\leftarrow$ *false*

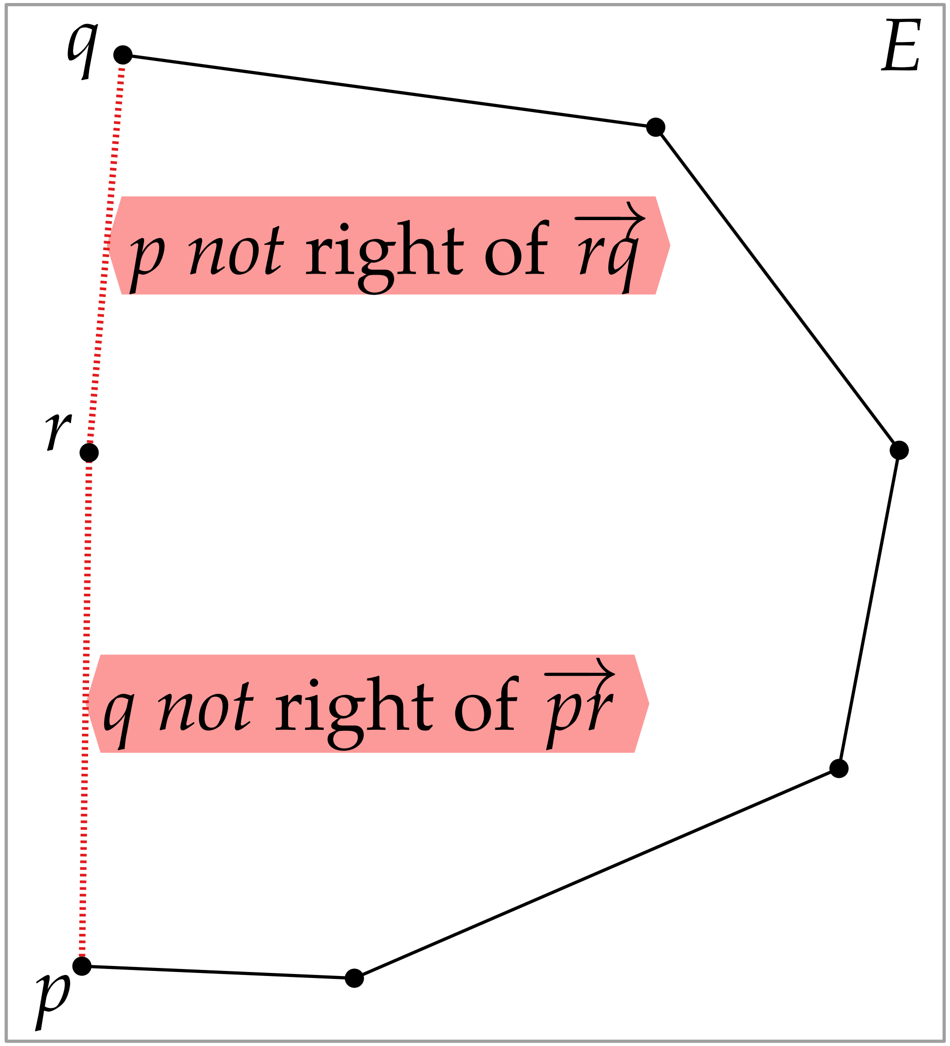




Discussion

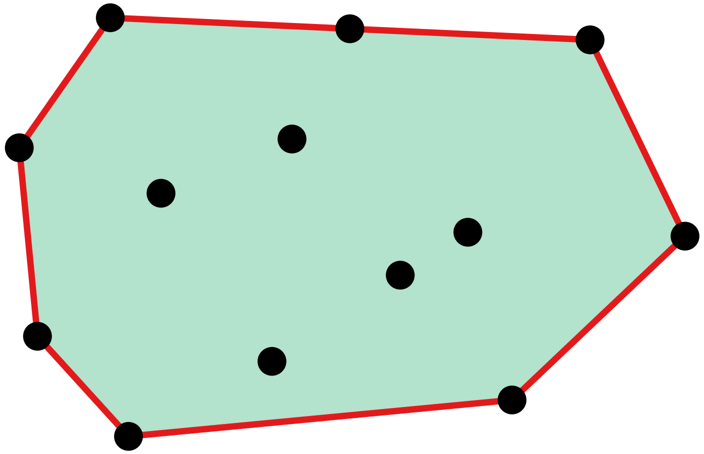
if not (r strictly right of $\overrightarrow{pq}$ **or** $r \in \overline{pq}$) **then**
  valid $\leftarrow$ *false*





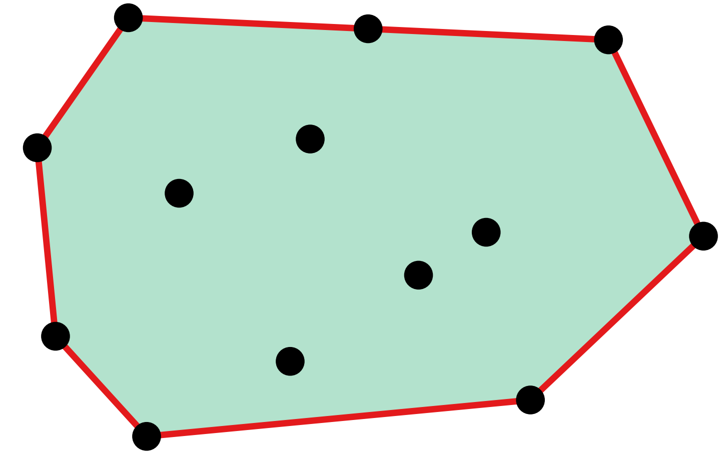
Observation. Algorithm FirstConvexHull is not *robust*.

New Ideas (Graham Scan)



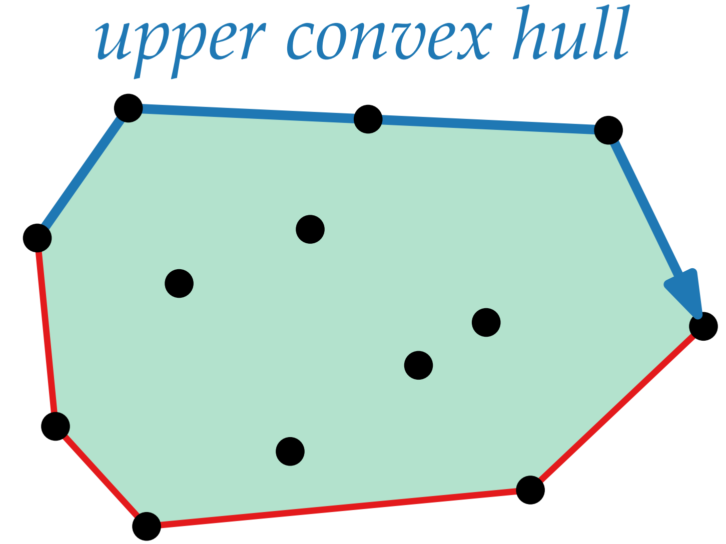
New Ideas (Graham Scan)

- split computation in two



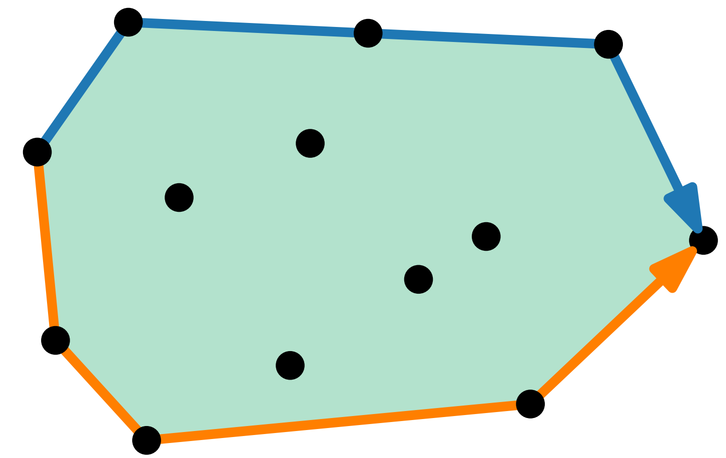
New Ideas (Graham Scan)

- split computation in two



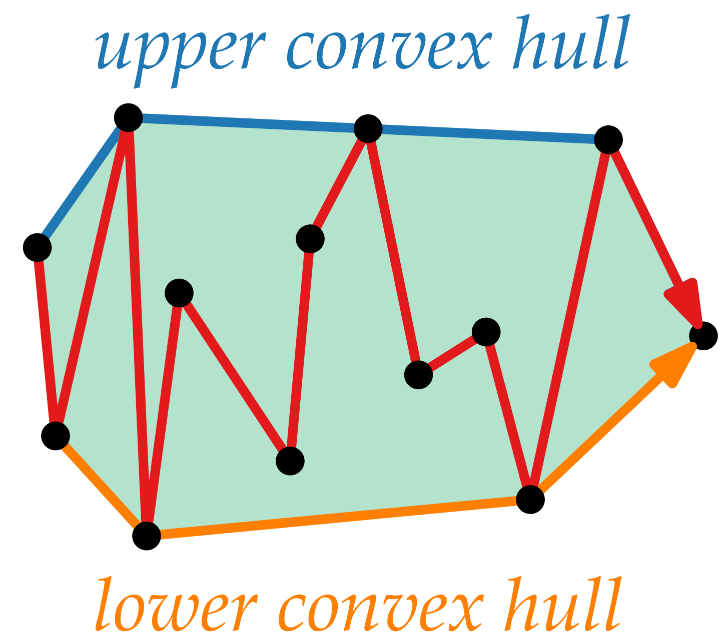
New Ideas (Graham Scan)

- split computation in two



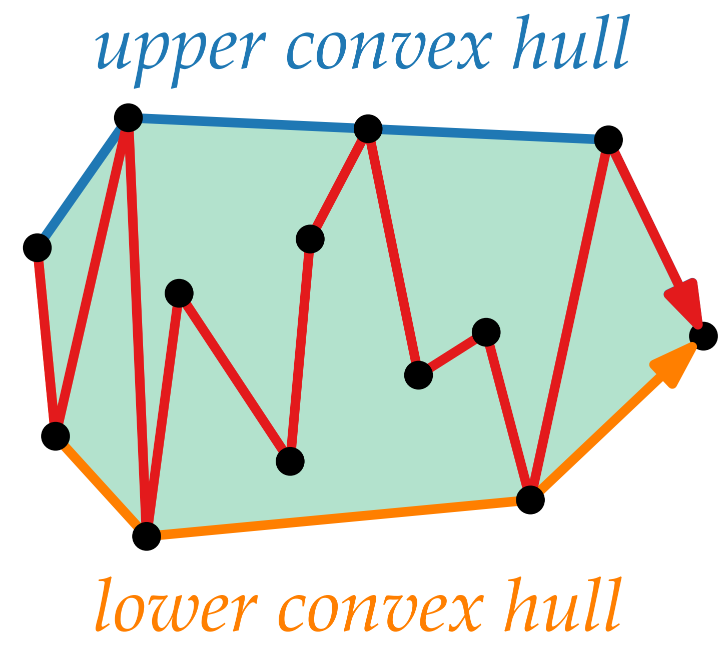
New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order



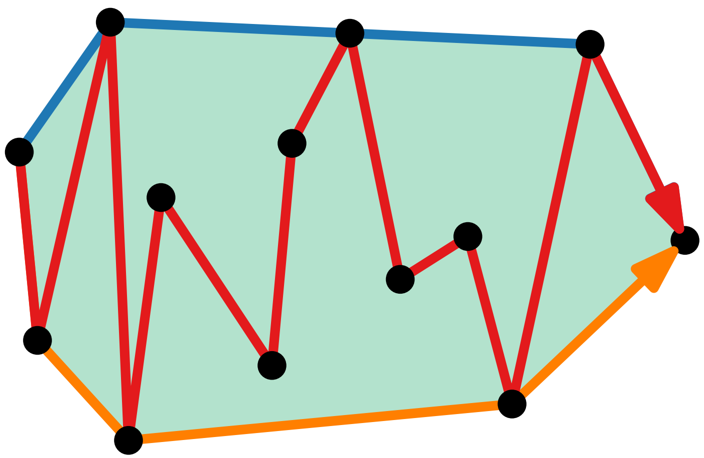
New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order
- proceed incrementally



New Ideas (Graham Scan)

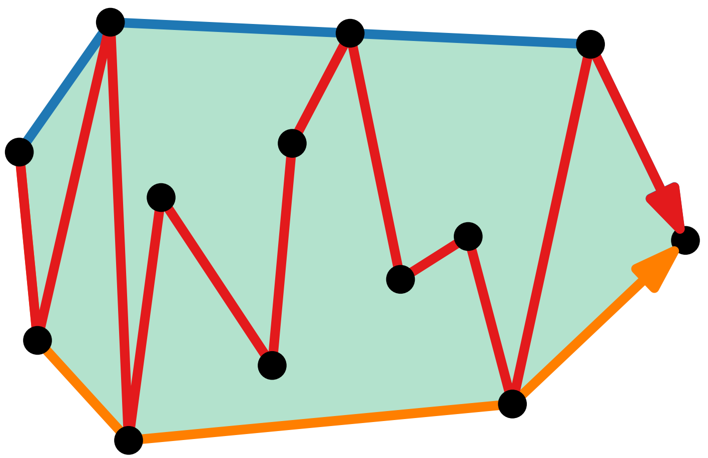
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do**
    L.append(p_i)
  return L

New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically

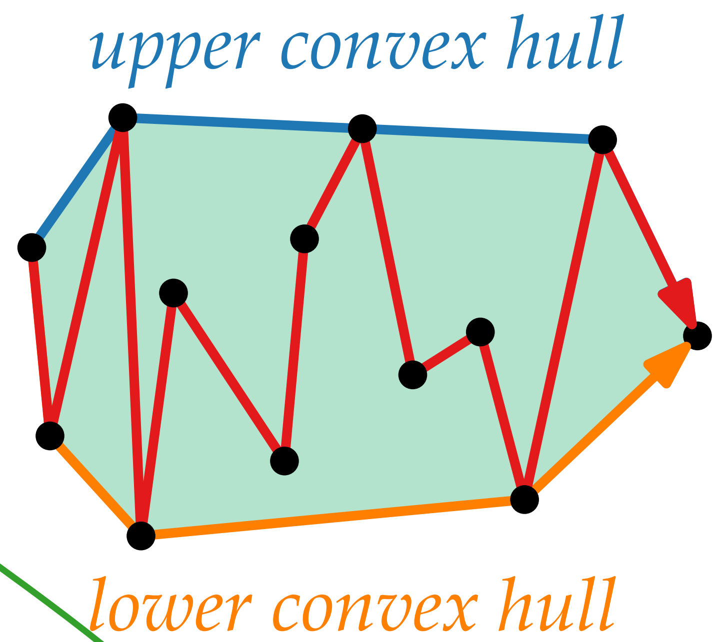
$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do**

  L.append(p_i)

return L

New Ideas (Graham Scan)

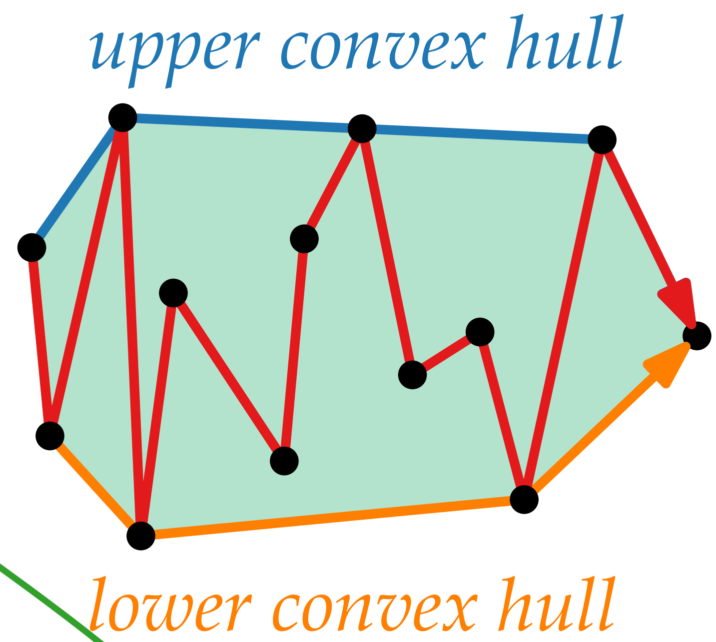


- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
$L \leftarrow \langle p_1, p_2 \rangle$
for $i \leftarrow 3$ **to** n **do**
    L.append(p_i)
return L

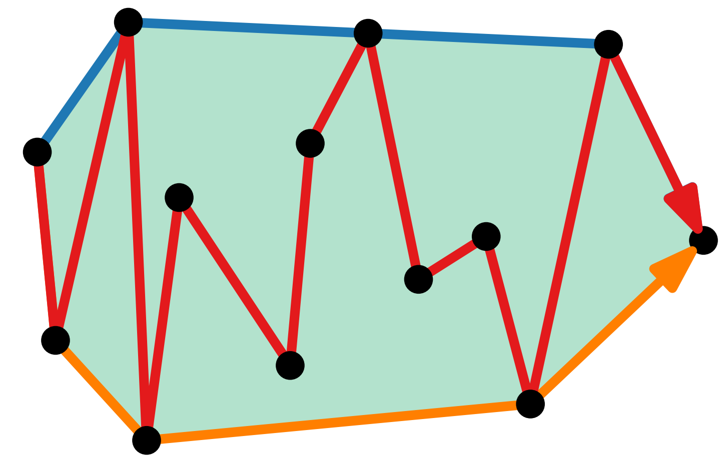
New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

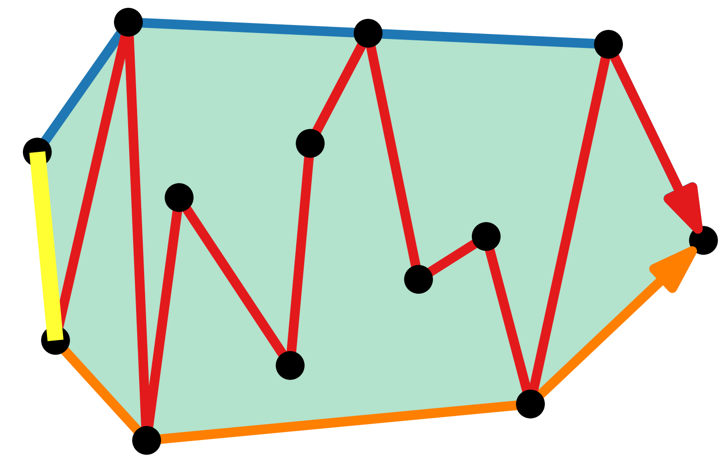


- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

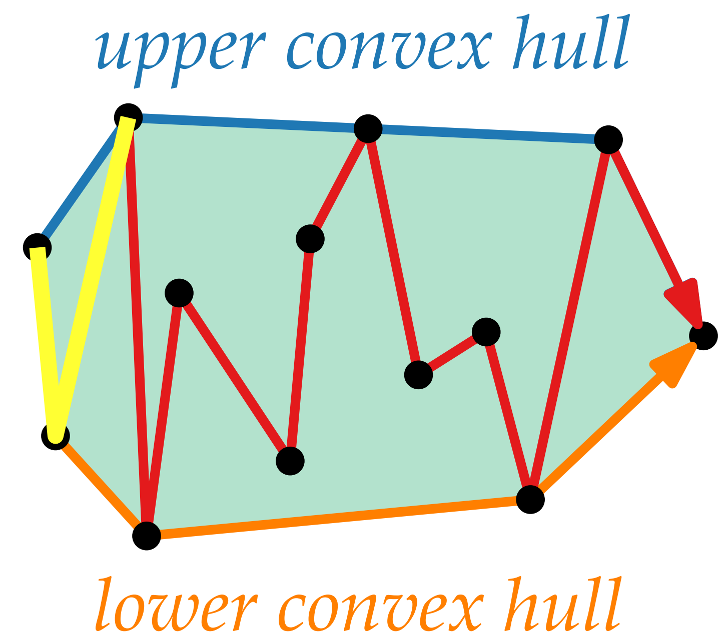
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

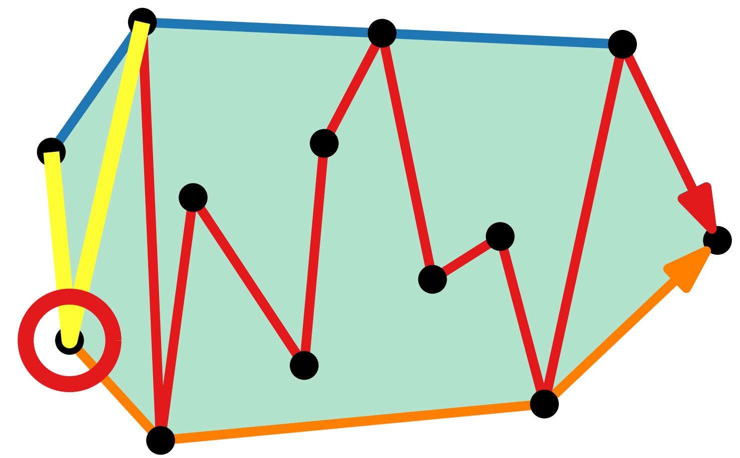
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

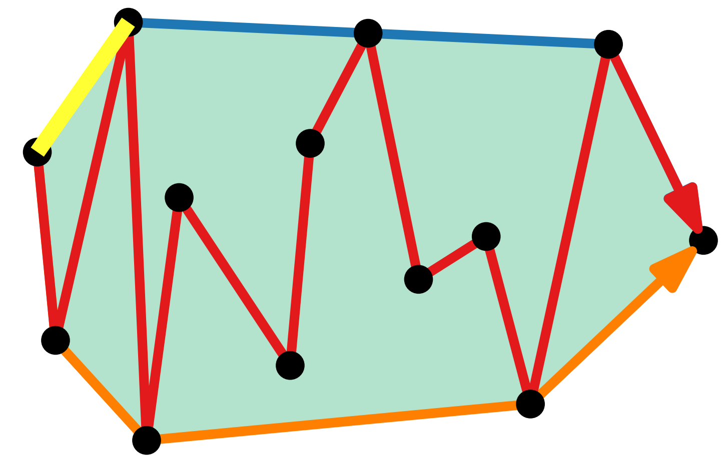
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order
- proceed incrementally

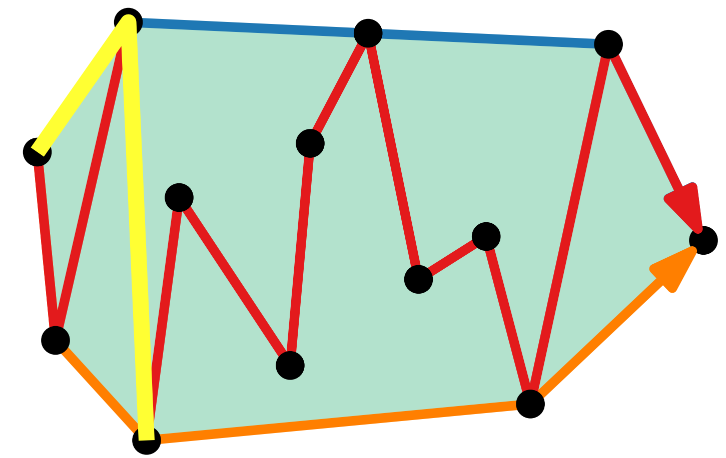



```
UpperConvexHull(S: set of pts in the plane)
  ⟨p_1, p_2, ..., p_n⟩ ← sort S lexicographically
  L ← ⟨p_1, p_2⟩
  for i ← 3 to n do  // compute upper convex hull of {p_1, p_2, ..., p_i}
      L.append(p_i)

  return L
```

New Ideas (Graham Scan)

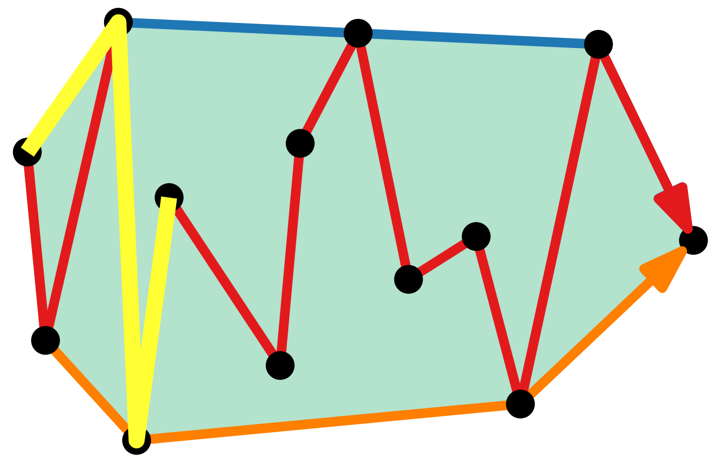
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
$L \leftarrow \langle p_1, p_2 \rangle$
for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
L.append(p_i)
return L

New Ideas (Graham Scan)

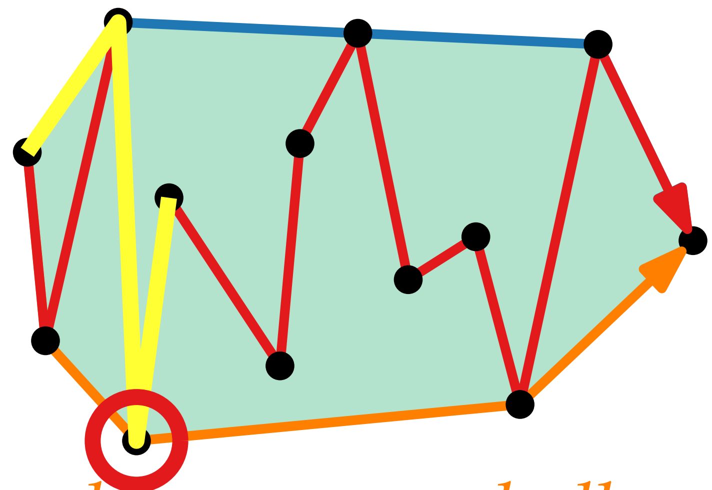
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

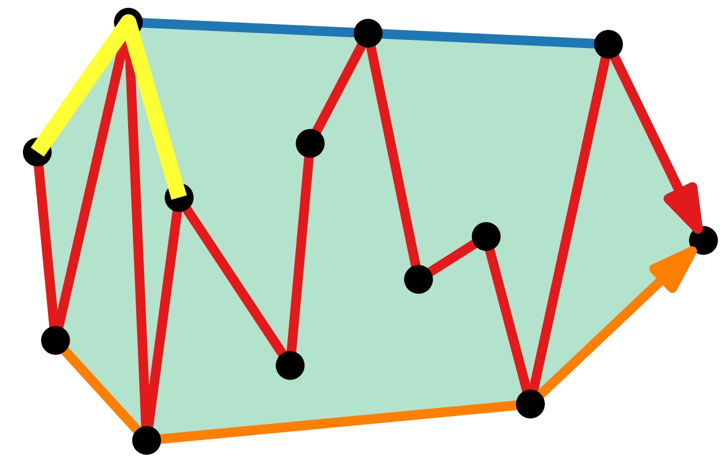
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

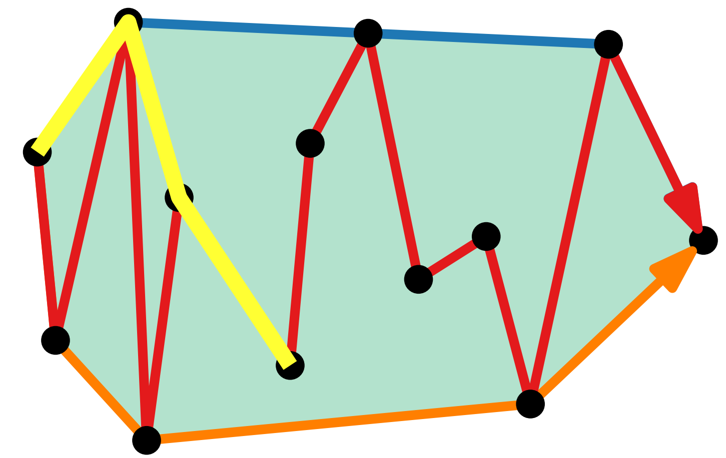
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

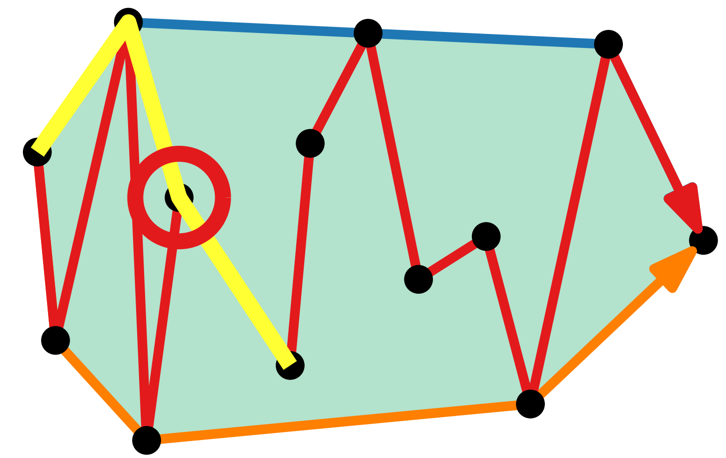
New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

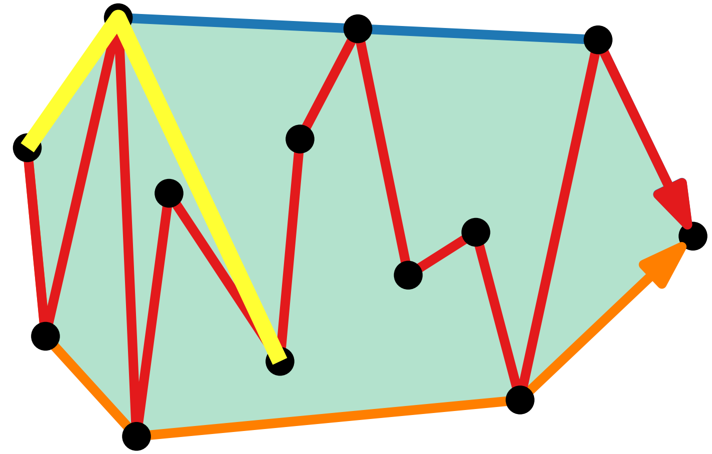


- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

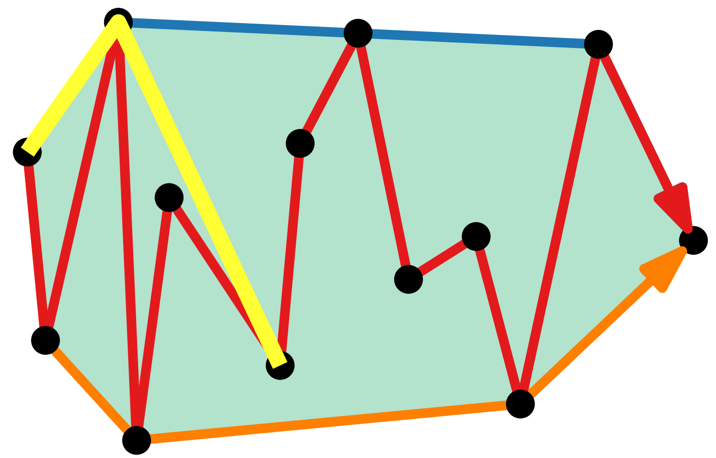
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
  return L

New Ideas (Graham Scan)

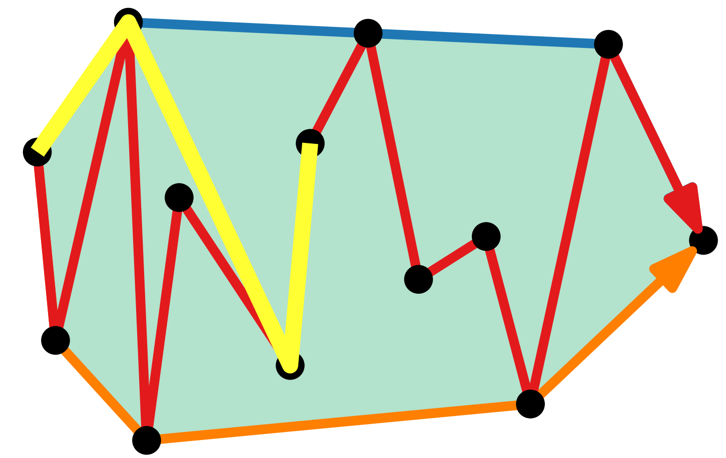
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

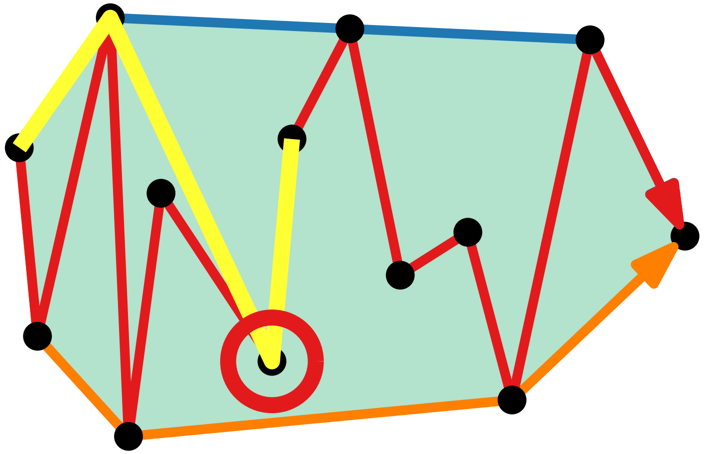
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
$L \leftarrow \langle p_1, p_2 \rangle$
for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
  L.append(p_i)
  while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
    remove second last pt from L
return L

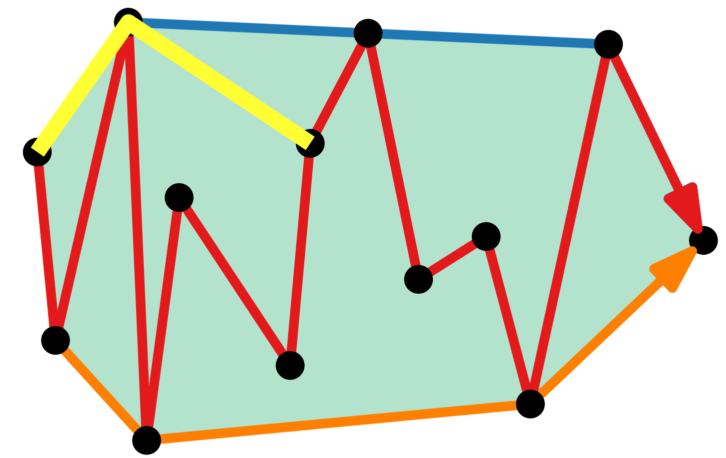
New Ideas (Graham Scan)



- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

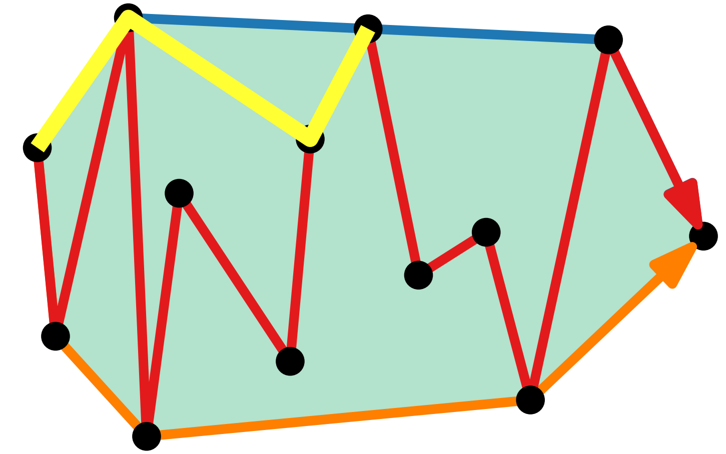


- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
$L \leftarrow \langle p_1, p_2 \rangle$
for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
        remove second last pt from L
return L

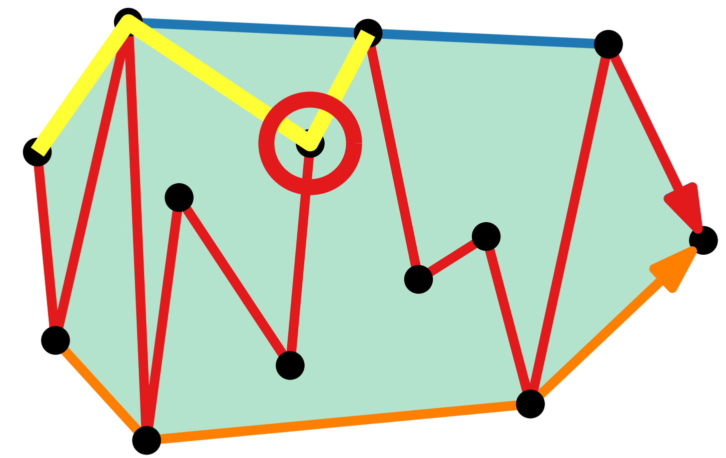
New Ideas (Graham Scan)



- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

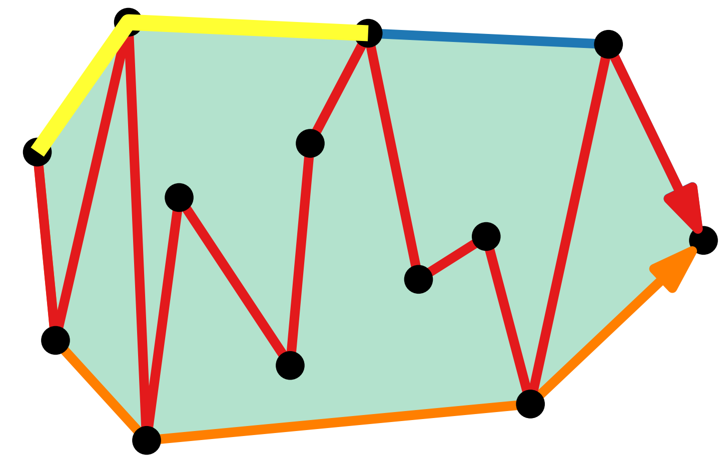


- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

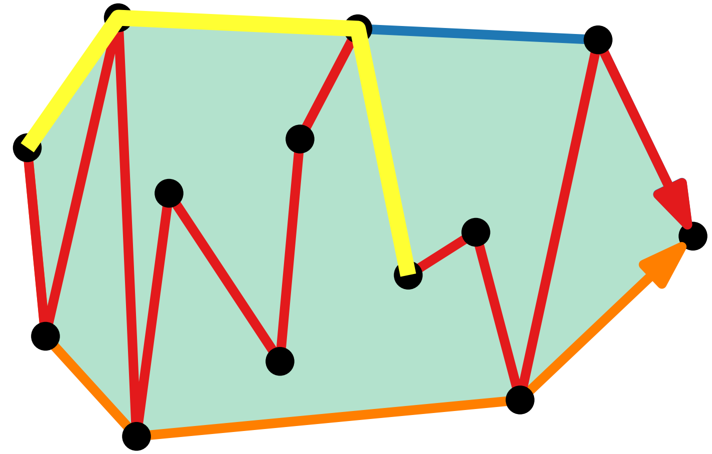
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

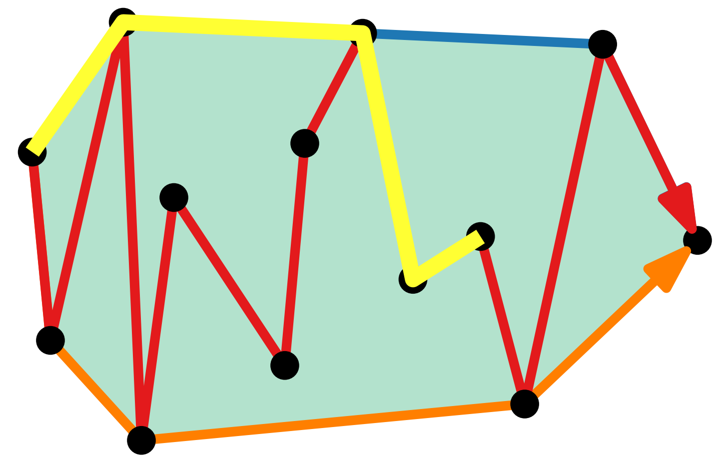
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

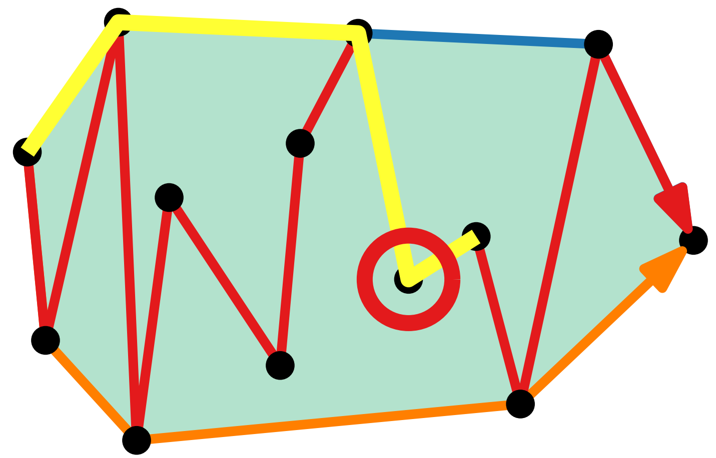
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

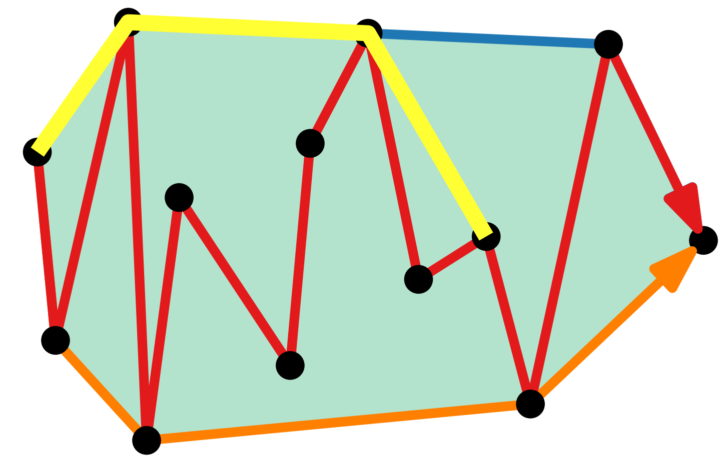
New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

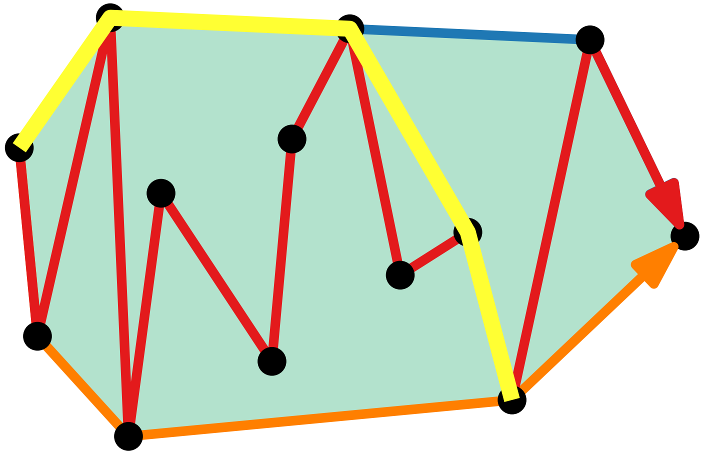
New Ideas (Graham Scan)



- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

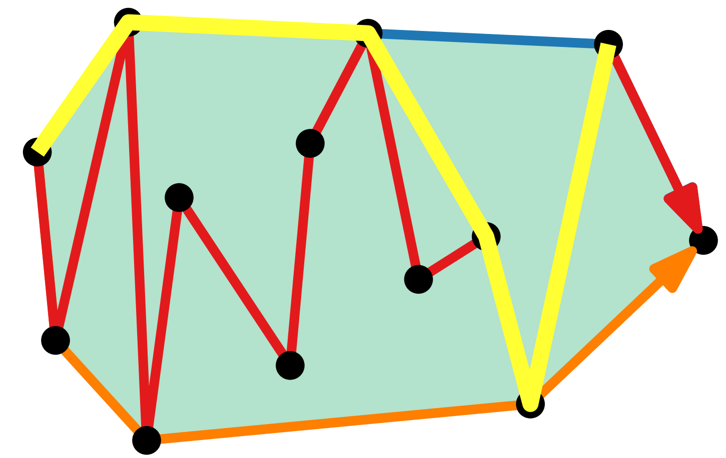


- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

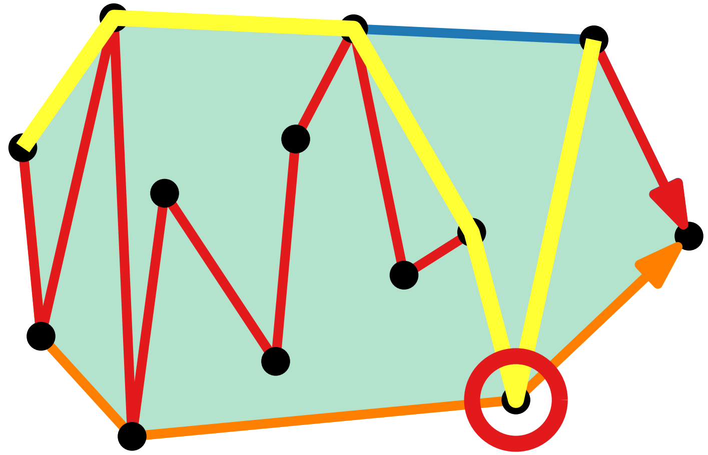
New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

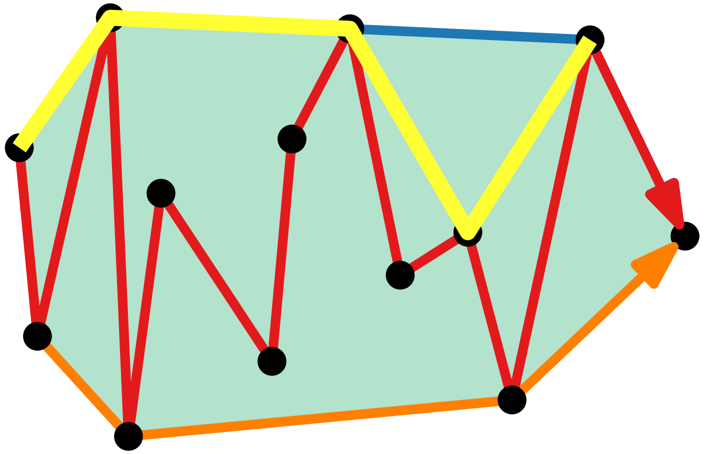


- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)

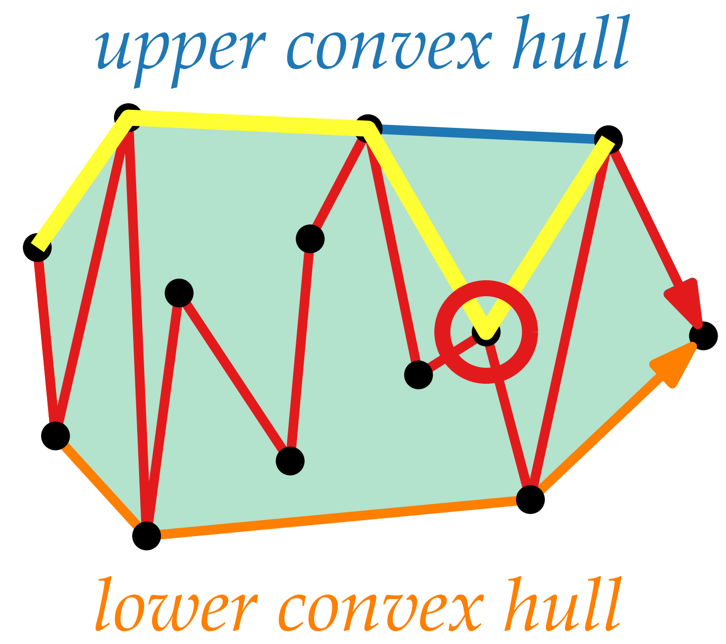
- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

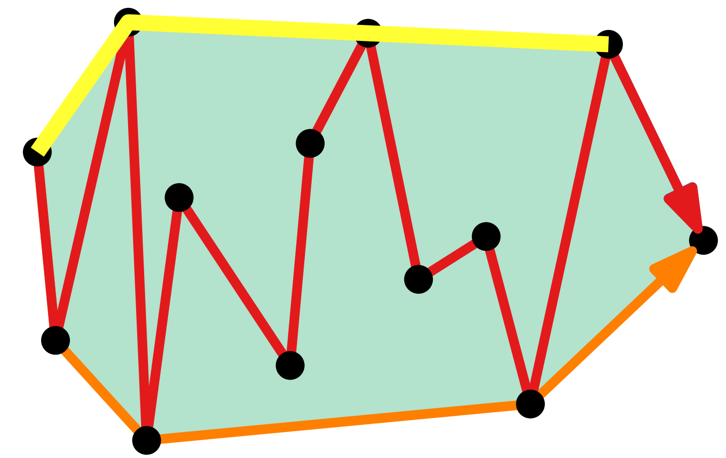
New Ideas (Graham Scan)

- split computation in two
- bring pts in lexicographic order
- proceed incrementally



UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

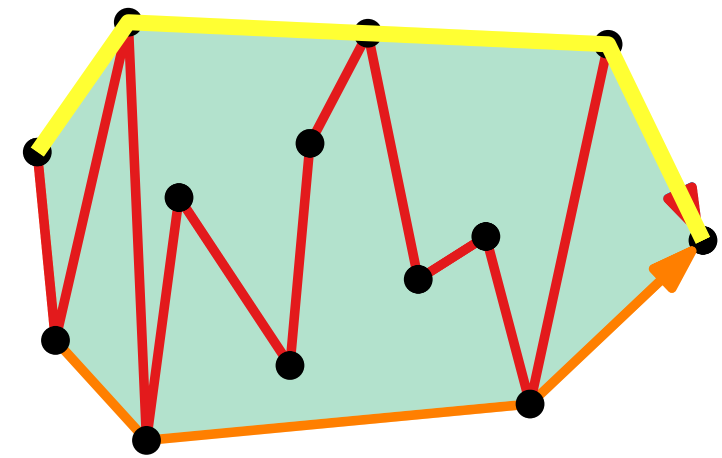
New Ideas (Graham Scan)



- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do** // compute upper convex hull of $\{p_1, p_2, \ldots, p_i\}$
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

New Ideas (Graham Scan)



- split computation in two
- bring pts in lexicographic order
- proceed incrementally

```
UpperConvexHull(S: set of pts in the plane)
  ⟨p_1, p_2, ..., p_n⟩ ← sort S lexicographically
  L ← ⟨p_1, p_2⟩
  for i ← 3 to n do  // compute upper convex hull of {p_1, p_2, ..., p_i}
      L.append(p_i)
      while |L| > 2 and last 3 pts in L make no right turn do
          remove second last pt from L
  return L
```

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do**
    L.append(p_i)
    while $|L| > 2$ **and** last 3 pts in L make no right turn **do**
      remove second last pt from L
  return L

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically $O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do**

L.append(p_i)

while $|L| > 2$ **and** last 3 pts in L make no right turn **do**

remove second last pt from L

return L

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically $\qquad O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do** $\qquad (n-2)\cdot$

L.append(p_i)

while $|L| > 2$ **and** last 3 pts in L make no right turn **do**

remove second last pt from L

return L

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically $\quad O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do** $\quad (n-2)\cdot$

L.append(p_i)

while $|L| > 2$ **and** last 3 pts in L make no right turn **do**

remove second last pt from L $\quad$?

return L

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically $\quad O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do** $\quad (n-2)\cdot$

L.append(p_i)

while $|L| > 2$ **and** last 3 pts in L make no right turn **do**

remove second last pt from L $\quad$ **?**

return L

Amortized analysis:

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically $O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do** $(n-2)\cdot$

L.append(p_i)

while $|L| > 2$ **and** last 3 pts in L make no right turn **do**

remove second last pt from L ?

return L

Amortized analysis:

– each pt $p_2, \ldots, p_{n-1}$ pays 1 € for its potential removal later on

Running Time Analysis

```
UpperConvexHull(S: set of pts in the plane)
    ⟨p_1, p_2, ..., p_n⟩ ← sort S lexicographically            O(n log n)
    L ← ⟨p_1, p_2⟩
    for i ← 3 to n do                                           (n − 2)·
        L.append(p_i)
        while |L| > 2 and last 3 pts in L make no right turn do
            remove second last pt from L                        ?
    return L
```

Amortized analysis:

– each pt $p_2, \ldots, p_{n-1}$ pays 1 € for its potential removal later on

– this pays for the total effort of all executions of the while loop

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically $O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do** ~~$(n-2)\cdot$~~

L.append(p_i)

while $|L| > 2$ **and** last 3 pts in L make no right turn **do**

remove second last pt from L ~~?~~

return L

Amortized analysis:

– each pt $p_2, \ldots, p_{n-1}$ pays 1 € for its potential removal later on

– this pays for the total effort of all executions of the while loop

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically $\quad O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do** $\quad$ ~~$(n-2)$~~ $\cdot$

  L.append(p_i) $\quad O(n)$

  while $|L| > 2$ **and** last 3 pts in L make no right turn **do**

    remove second last pt from L $\quad$ ~~?~~

return L

Amortized analysis:

– each pt $p_2, \ldots, p_{n-1}$ pays 1 € for its potential removal later on

– this pays for the total effort of all executions of the while loop

Running Time Analysis

UpperConvexHull(S: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort S lexicographically $O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ **to** n **do** ~~$(n-2)\cdot$~~

L.append(p_i) $O(n)$

while $|L| > 2$ **and** last 3 pts in L make no right turn **do**

remove second last pt from L ~~?~~

return L

Amortized analysis:

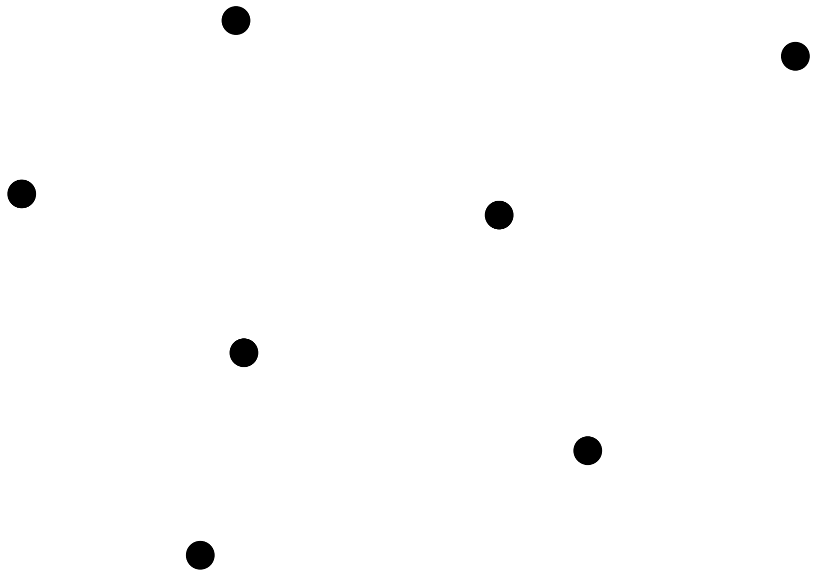
– each pt $p_2, \ldots, p_{n-1}$ pays 1 € for its potential removal later on

– this pays for the total effort of all executions of the while loop

Theorem. We can compute the convex hull of n pts in the plane in $O(n \log n)$ time – in a robust way.

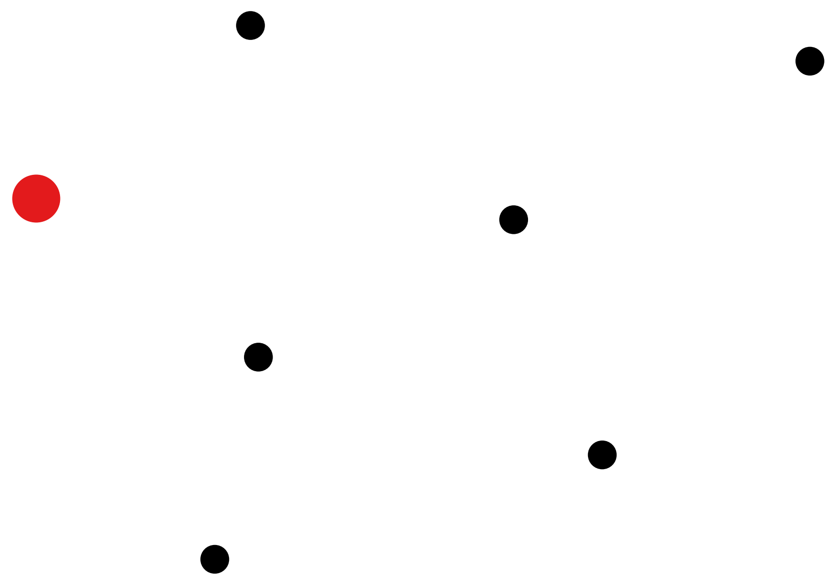
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



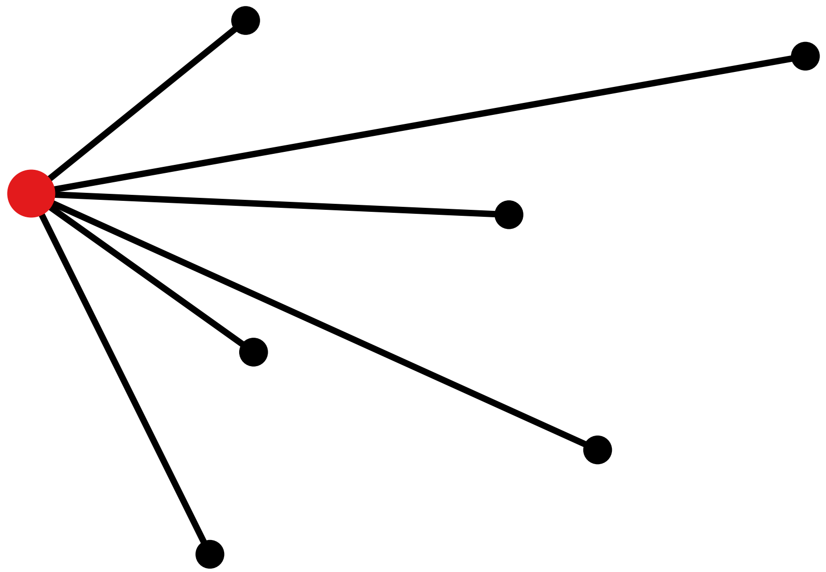
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



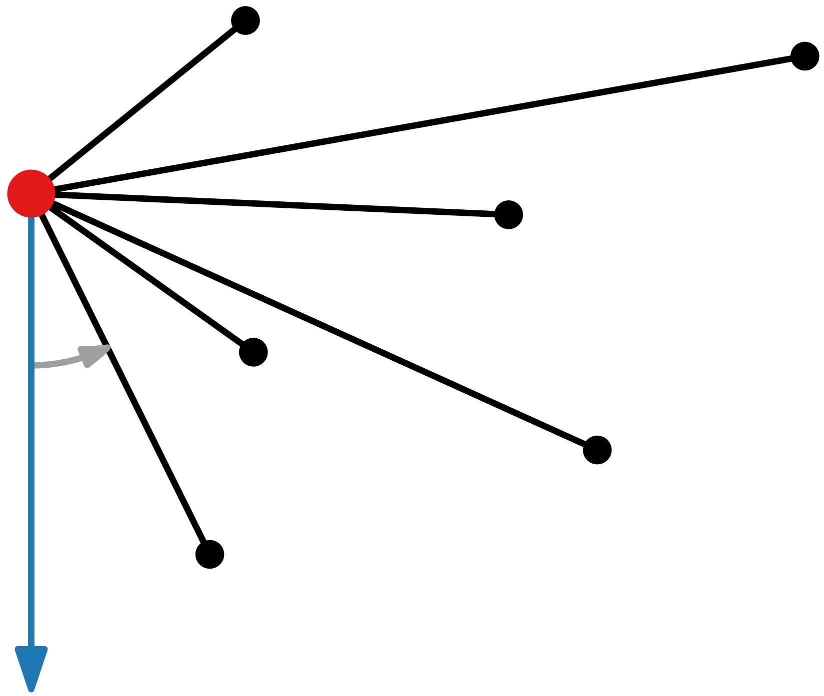
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



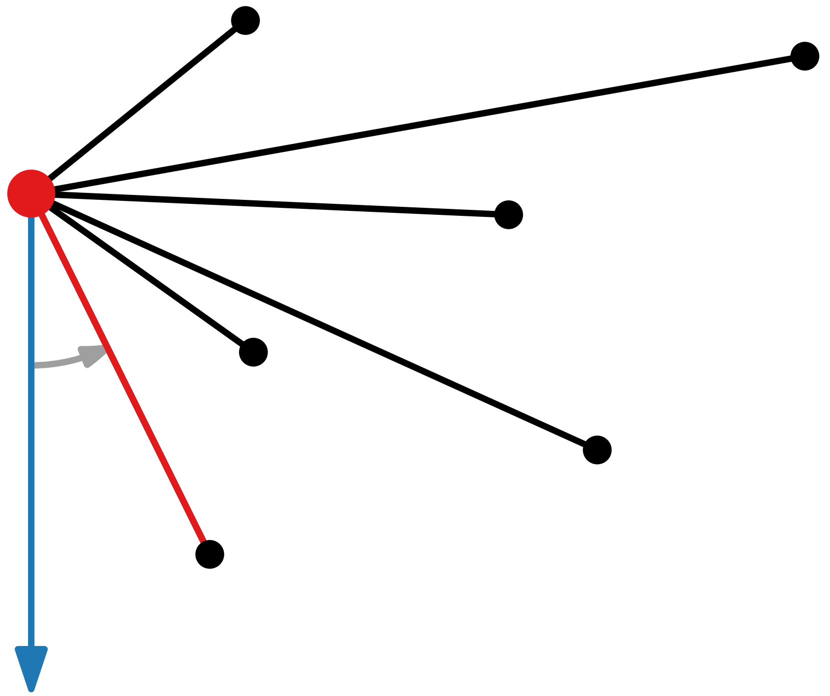
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



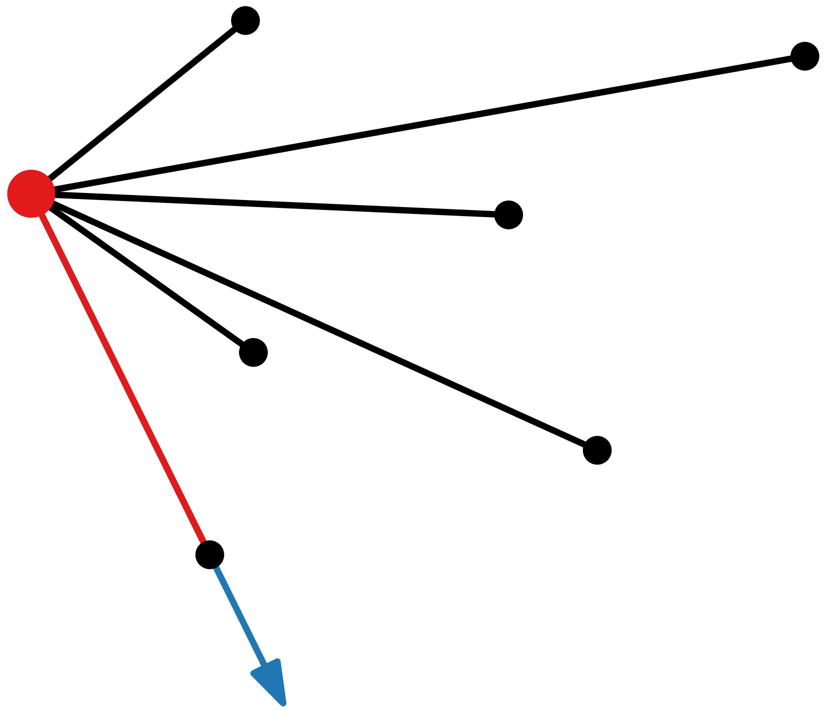
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



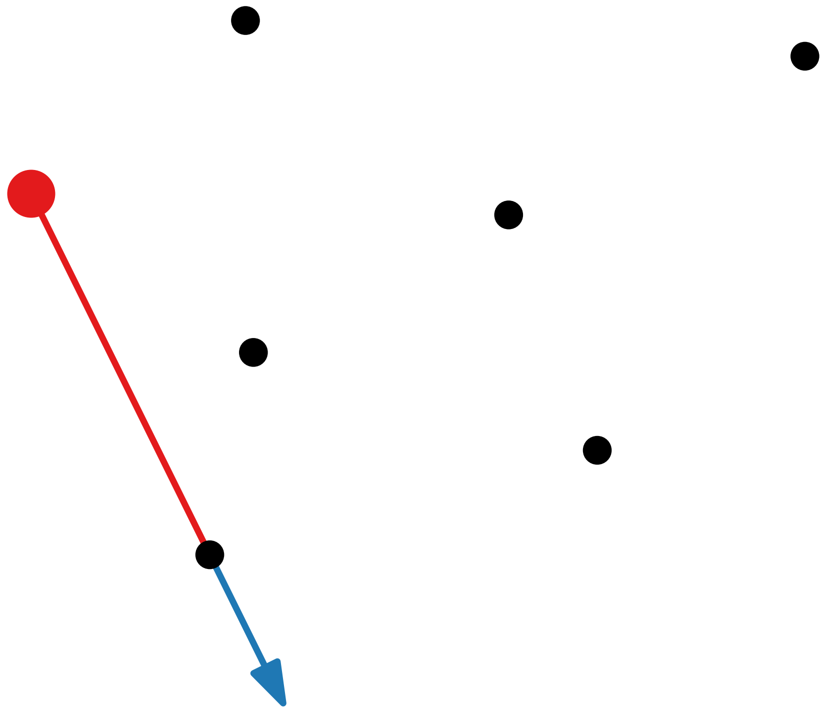
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



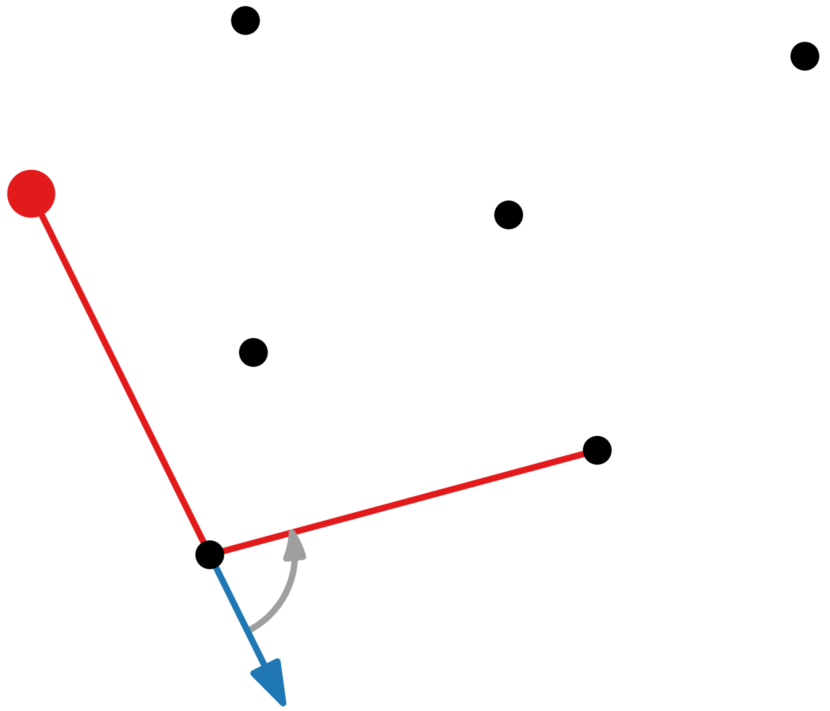
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



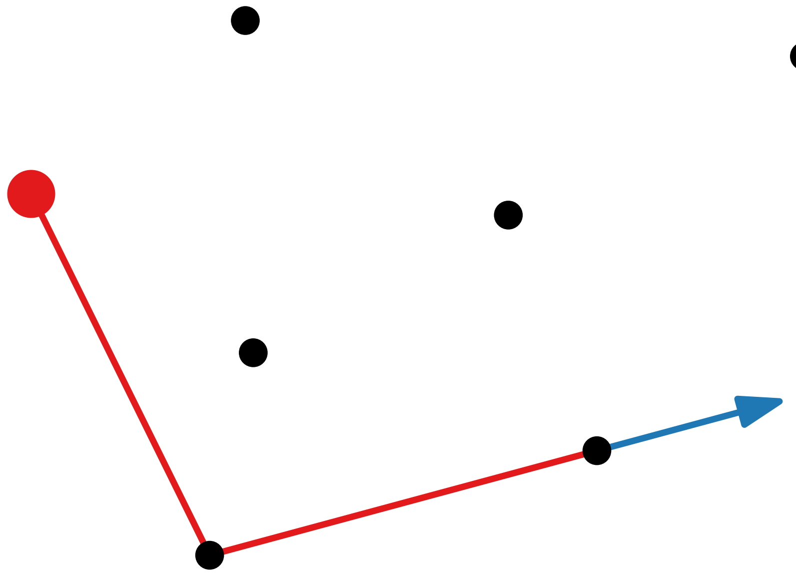
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



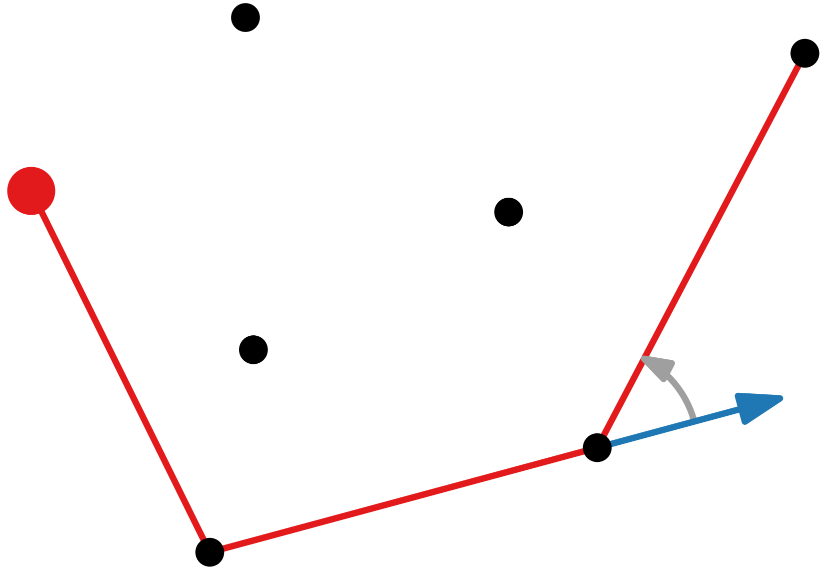
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



Output-Sensitive Algorithms

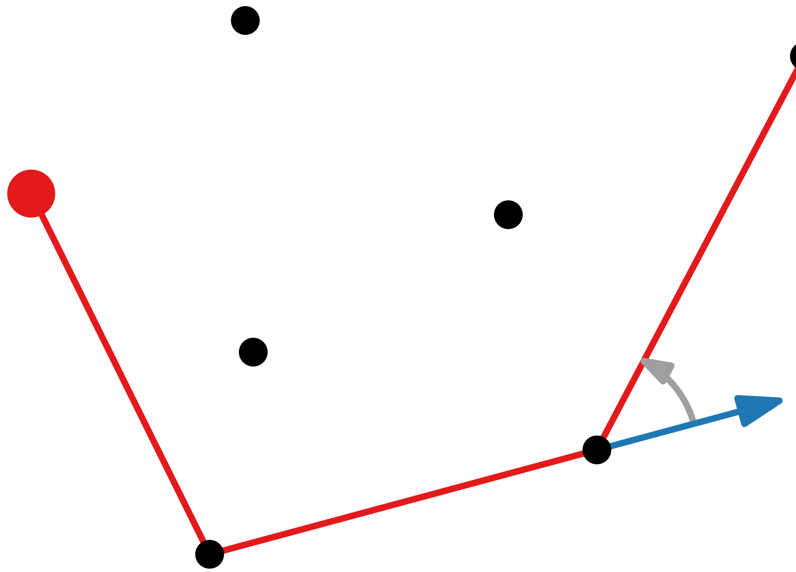
- Jarvis' gift-wrapping algorithm (aka Jarvis' march)



Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)

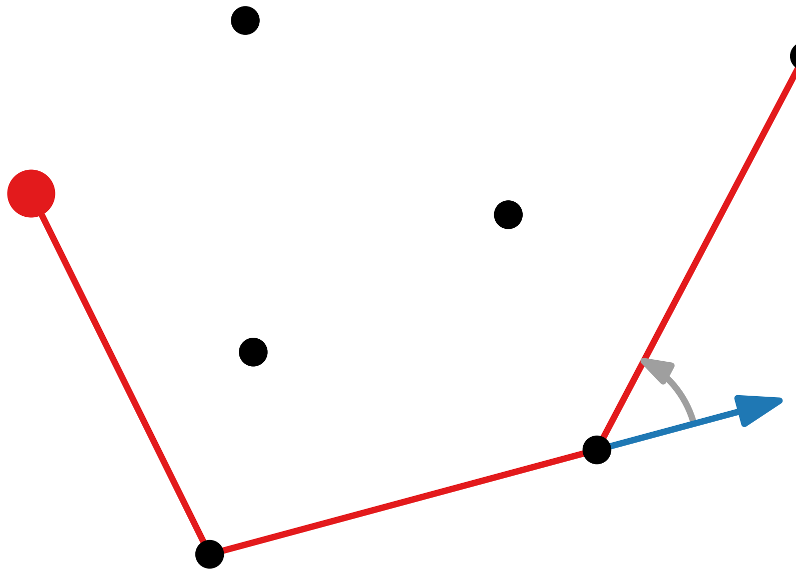
Runtime?



Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)

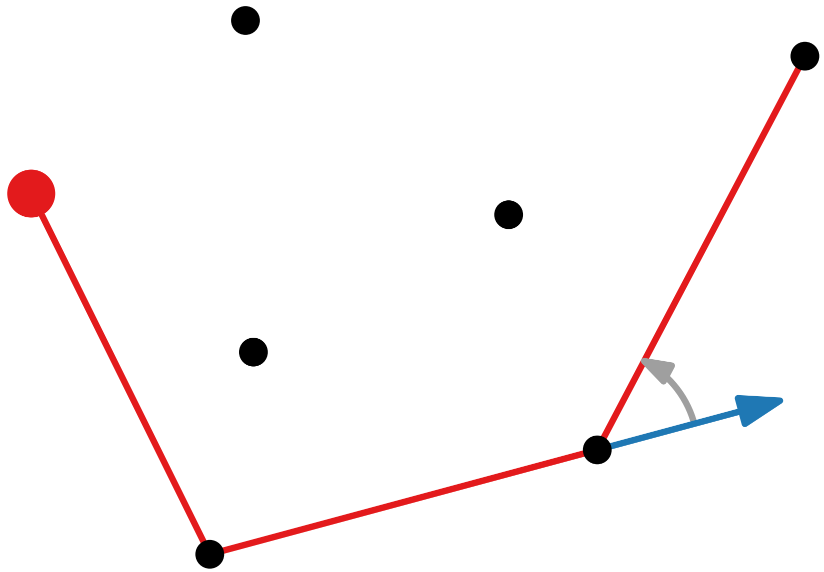
Runtime? $O(n \cdot h)$



Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)

Runtime? $O(n \cdot h)$

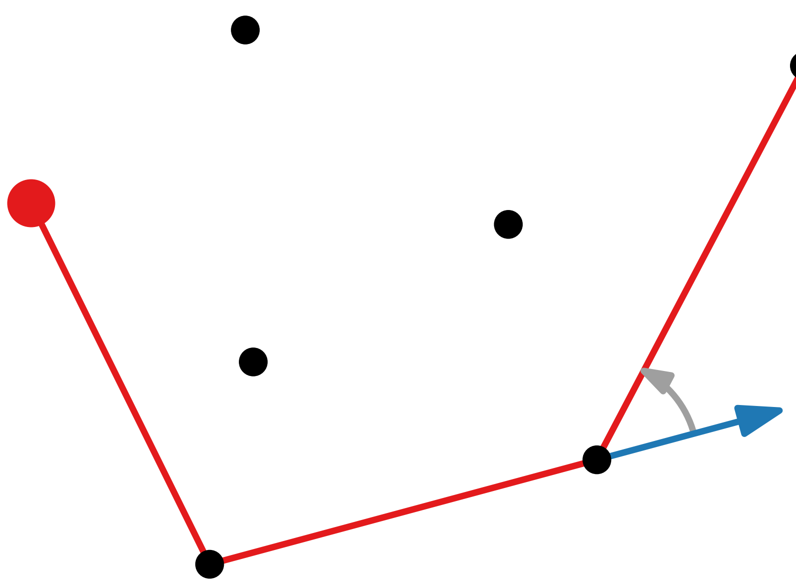


$\ldots$ where $h = |\mathrm{CH}(S)| =$ size of the output

Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm (aka Jarvis' march)

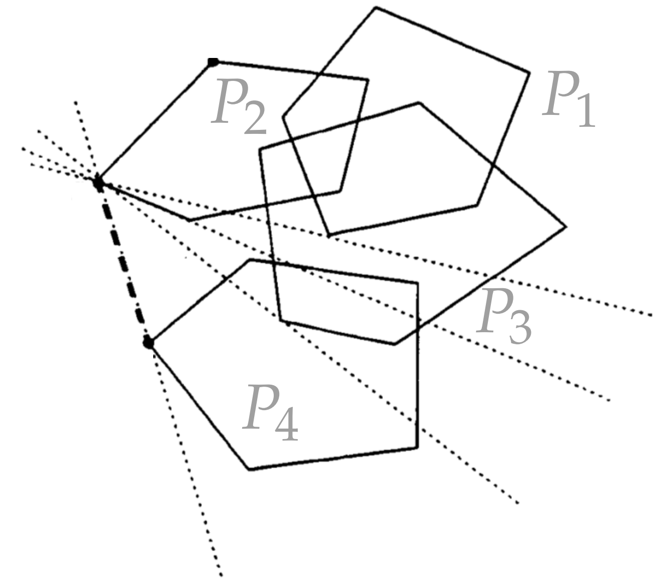
Runtime? $O(n \cdot h)$



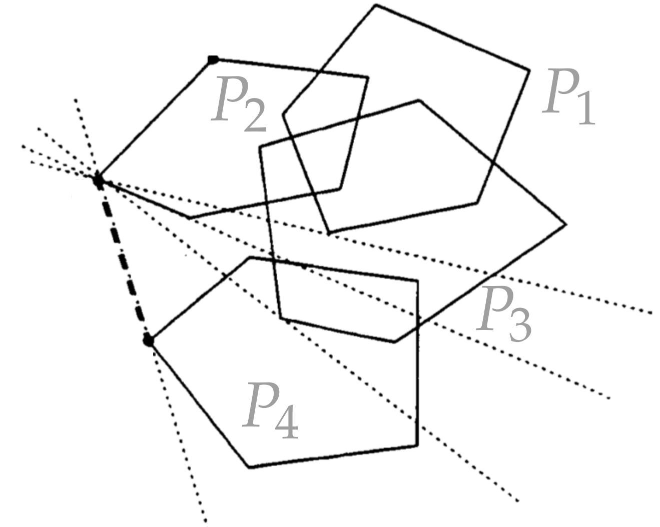
- Chan's exponential-search algorithm $O(n \log h)$

... where $h = |\mathrm{CH}(S)|$ = size of the output

Chan's Algorithm



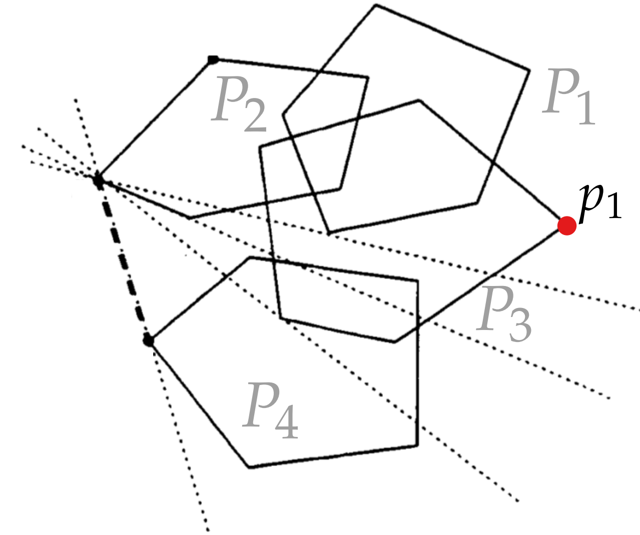
Chan's Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \le m \le n$, and $H \ge 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order [in $O(m \log m)$ time]
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P

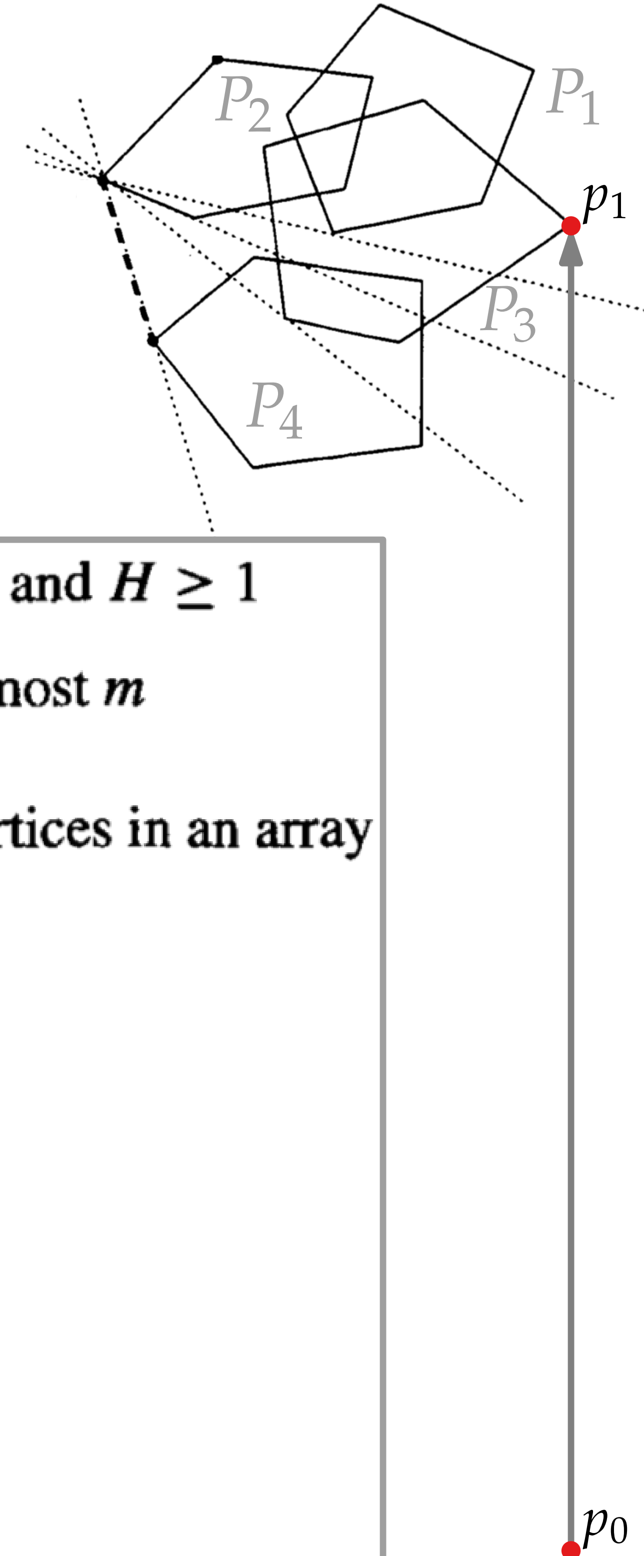
Chan's Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \le m \le n$, and $H \ge 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order [in $O(m \log m)$ time]
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P

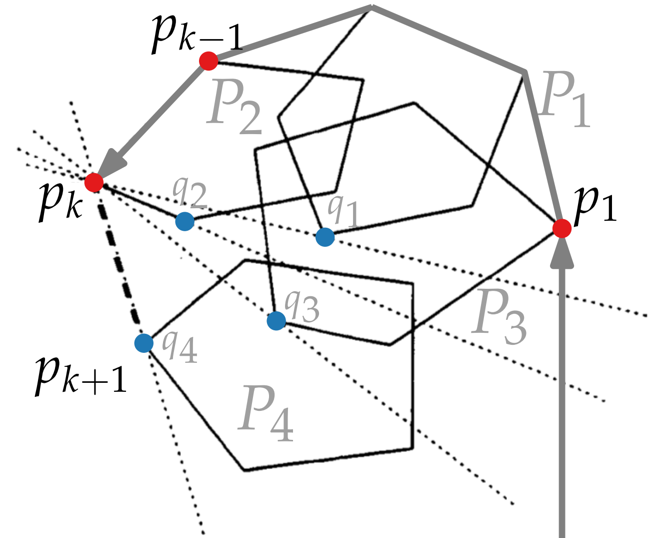
Chan's Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

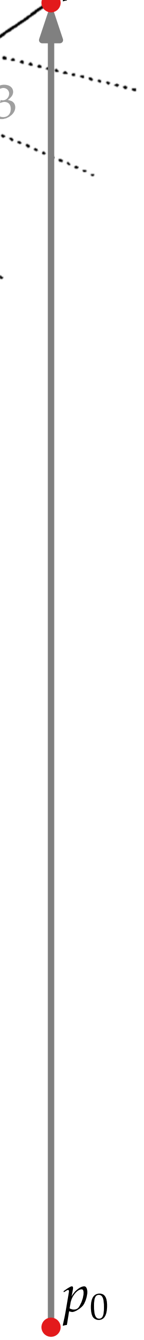
1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order [in $O(m \log m)$ time]
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P

Chan's Algorithm

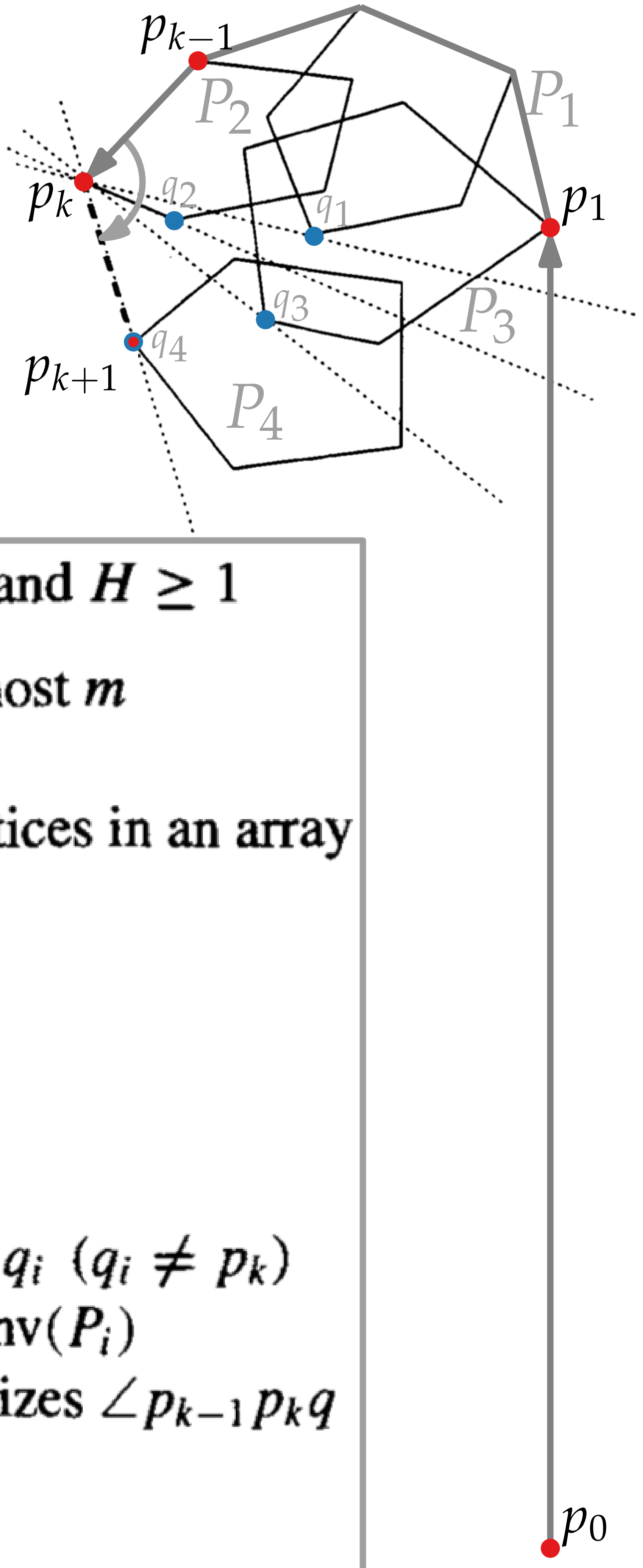


Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \le m \le n$, and $H \ge 1$

partition P into subsets $P_1, \dots, P_{\lceil n/m \rceil}$ each of size at most m
for $i = 1, \dots, \lceil n/m \rceil$ do
    compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order   [in $O(m \log m)$ time]
$p_0 \leftarrow (0, -\infty)$
$p_1 \leftarrow$ the rightmost point of P
for $k = 1, \dots, H$ do
    for $i = 1, \dots, \lceil n/m \rceil$ do
        compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \ne p_k)$ by performing a binary search on the vertices of conv(P_i)
    $p_{k+1} \leftarrow$ the point q from $\{q_1, \dots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
    if $p_{k+1} = p_1$ then return the list $\langle p_1, \dots, p_k \rangle$
return *incomplete*



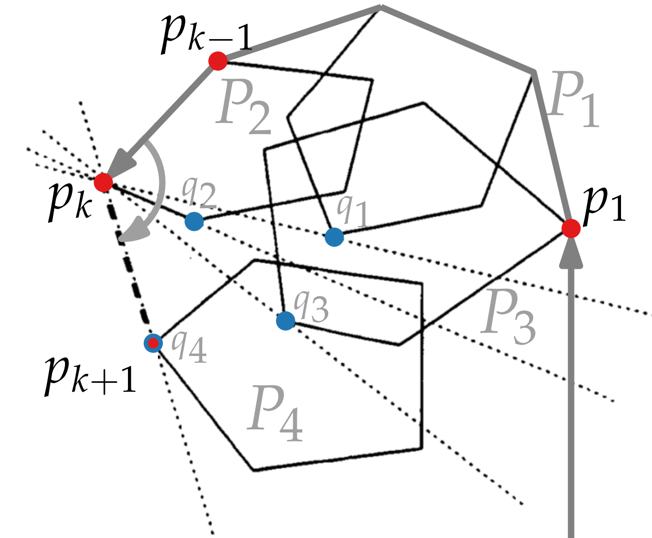
Chan's Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \le m \le n$, and $H \ge 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order [in $O(m \log m)$ time]
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of conv(P_i)
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

Chan's Algorithm



Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. $\quad$ compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order $\qquad$ [in $O(m \log m)$ time]
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. $\quad$ for $i = 1, \ldots, \lceil n/m \rceil$ do
8. $\quad\quad$ compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ ($q_i \neq p_k$) by performing a binary search on the vertices of conv(P_i)
9. $\quad p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. $\quad$ if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

Chan's Algorithm

[Text copied on October 17, 2017 from: https://en.wikipedia.org/wiki/Chan's_algorithm]

Initially, we assume that the value of h is known and make a parameter $m = h$. This assumption is not realistic, but we remove it later. The algorithm starts by arbitrarily partitioning P into at most $1 + \frac{n}{m}$ subsets Q with at most m points each. Then, it computes the convex hull of each subset Q using an $O(n \log n)$ algorithm – **Graham's scan**. Note that, as there are $O(n/m)$ subsets of $O(m)$ points each, this phase takes $O(n/m) \cdot O(m \log m) = O(n \log m)$ time.

The second phase consists of executing the **Jarvis' march** algorithm algorithm and using the precomputed convex hulls to speed up the execution. At each step in Jarvis's march, we have a point p_i in the convex hull, and need to find a point $p_{i+1} = f(p_i, P)$ such that all other points of P are to the right of the line $p_i p_{i+1}$. If we know the convex hull of a set Q of m points, then we can compute $f(p_i, Q)$ in $O(\log m)$ time, by using binary search. We can compute $f(p_i, Q)$ for all the $O(n/m)$ subsets Q in $O(n/m \log m)$ time. Then, we can determine $f(p_i, P)$ using the same technique as normally used in Jarvis's march, but only considering the points that are $f(p_i, Q)$ for some subset Q. As Jarvis's march repeats this process $O(h)$ times, the second phase also takes $O(n \log m)$ time, and therefore $O(n \log h)$ time if $m = h$.
By running the two phases described above, we can compute the convex hull of n points in $O(n \log h)$ time, assuming that we know the value of h. If we make $m < h$, we can abort the execution after $m + 1$ steps, therefore spending only $O(n \log m)$ time (but not computing the convex hull). We can initially set m as a small constant (we use 2 for our analysis, but in practice numbers around 5 may work better), and increase the value of m until $m > h$, in which case we obtain the convex hull as a result.
If we increase the value of m too slowly, we may need to repeat the steps mentioned before too many times, and the execution time will be large. On the other hand, if we increase the value of m too quickly, we risk making m much larger than h, also increasing the execution time. Similar to strategy used by Chazelle and Matoušek's algorithm, Chan's algorithm squares the value of m at each iteration, and makes sure that m is never larger than n. In other words, at iteration t (starting at 1), we have $m = \min\left(n, 2^{2^t}\right)$. The total running time of the algorithm is

$$\sum_{t=1}^{\lceil \log\log h \rceil} O\left(n \log(2^{2^t})\right) = O(n) \sum_{t=1}^{\lceil \log\log h \rceil} O(2^t) = O\left(n \cdot 2^{1+\lceil \log\log h \rceil}\right) = O(n \log h).$$

To generalize this construction for the 3-dimensional case, an $O(n \log n)$ algorithm to compute the 3-dimensional convex hull should be used instead of Graham scan, and a 3-dimensional version of Jarvis's march needs to be used. The time complexity remains $O(n \log h)$.