CONVEX HULL IN 2D

Vera Sacristán

Discrete and Algorithmic Geometry Geometry
Facultat de Matemàtiques i Estadística
Universitat Politècnica de Catalunya

Computing the extreme points

Computing the extreme points

Characterization

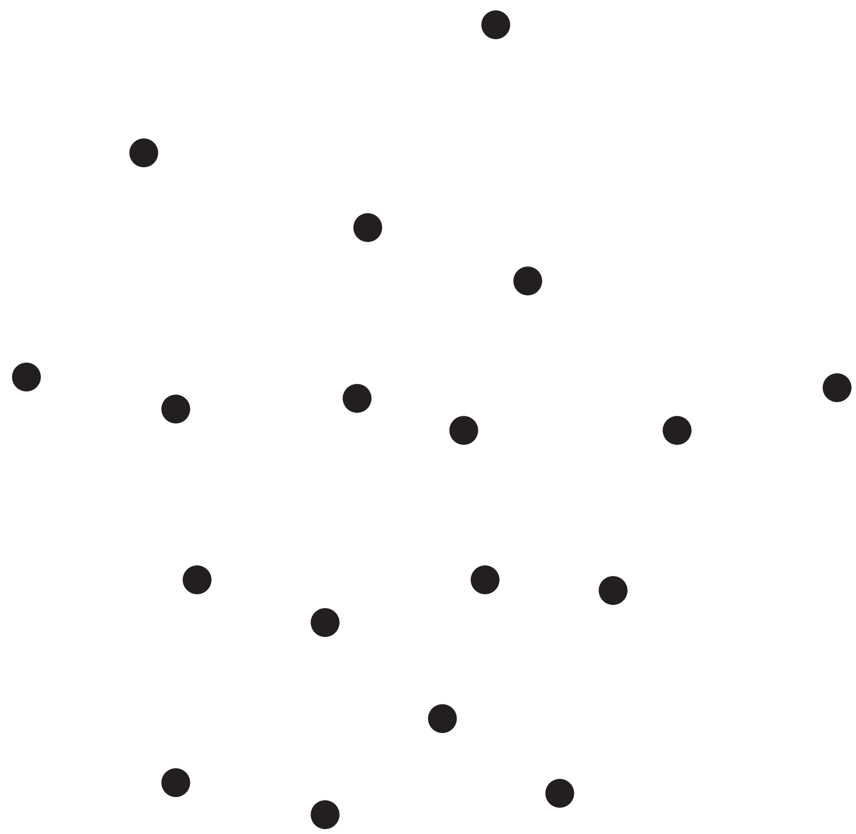
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

CONVEX HULL IN 2D

Computing the extreme points

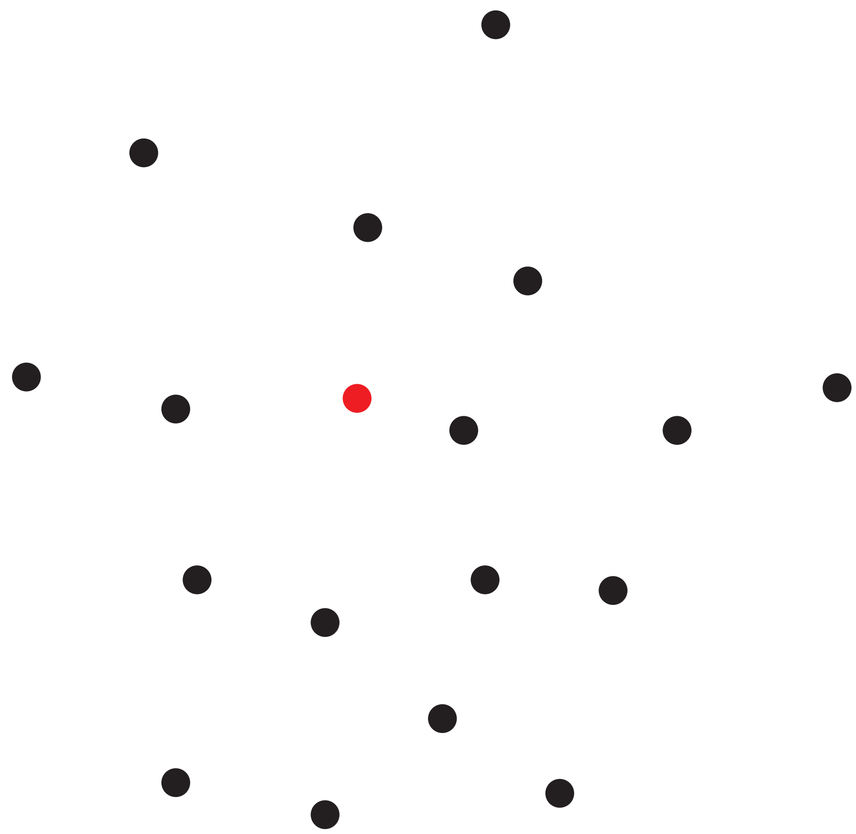
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.



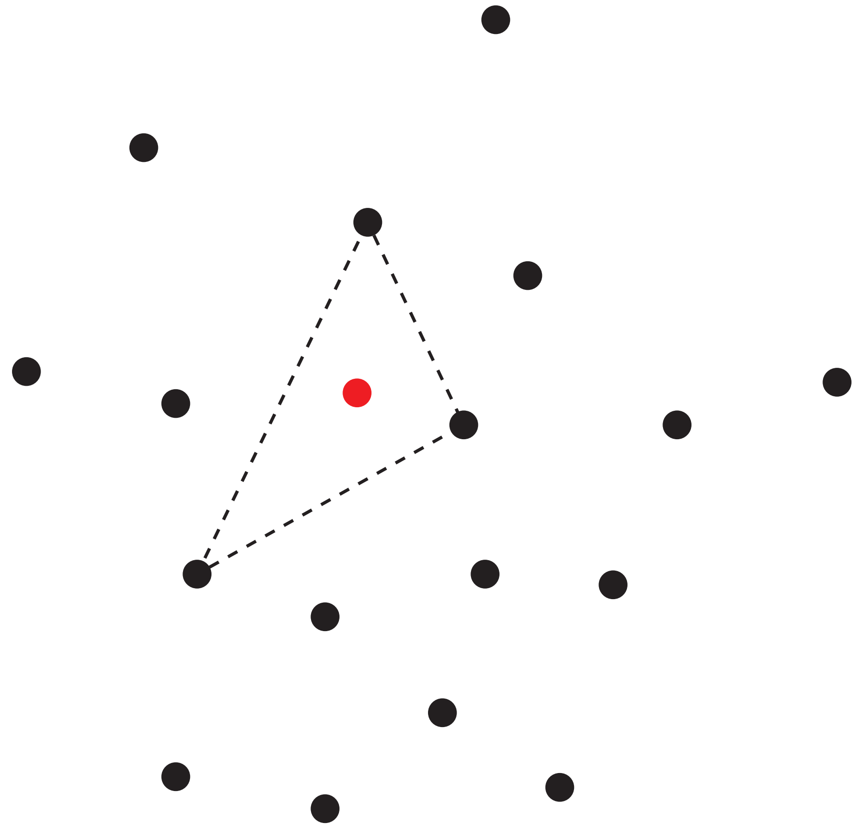
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.



Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.



CONVEX HULL IN 2D

Computing the extreme points

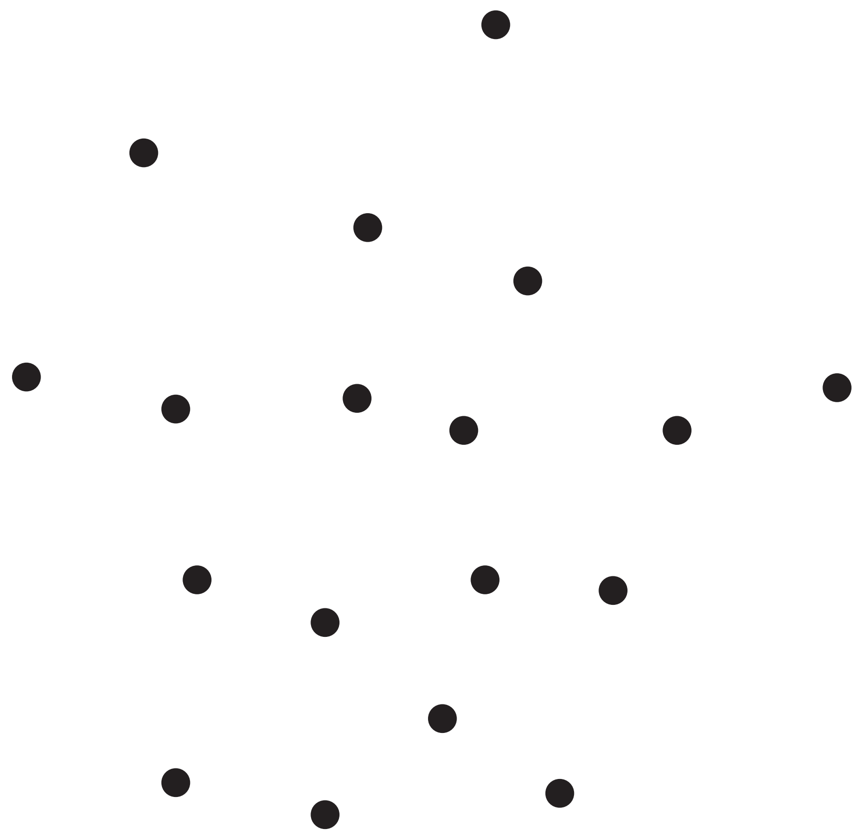
Characterization

Given $X = \{p_1, \dots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \dots, p_n$
Output: set of the extreme points

Procedure:
- For each i,
  - For each $j, k, l \neq i$,
  - If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
- Return the set of surviving p_i's.



Computing the extreme points

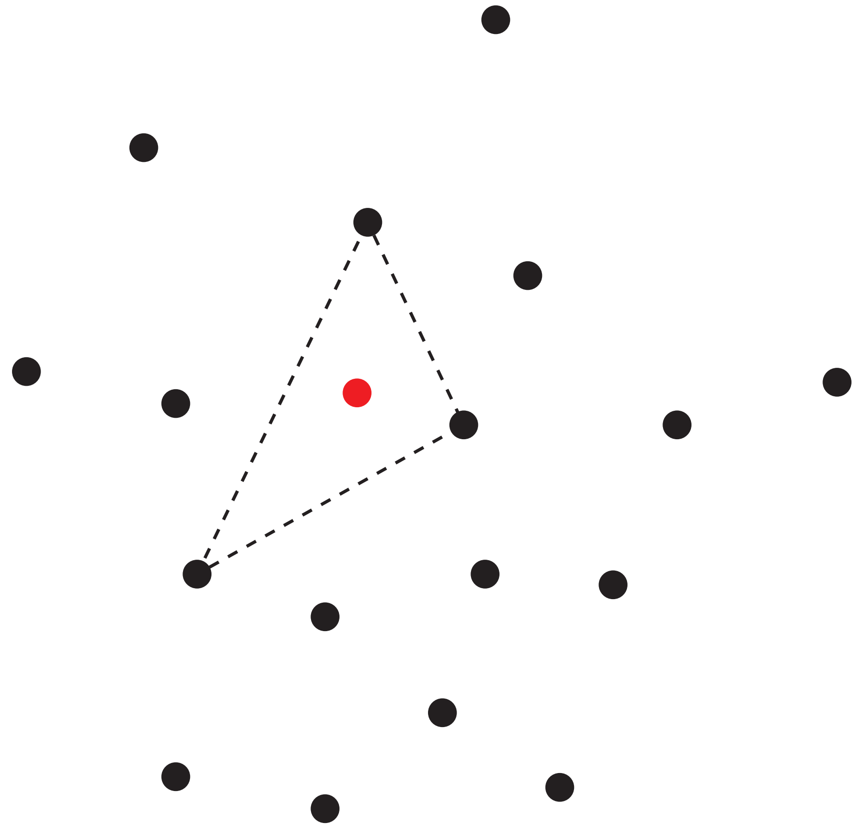
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



CONVEX HULL IN 2D

Computing the extreme points

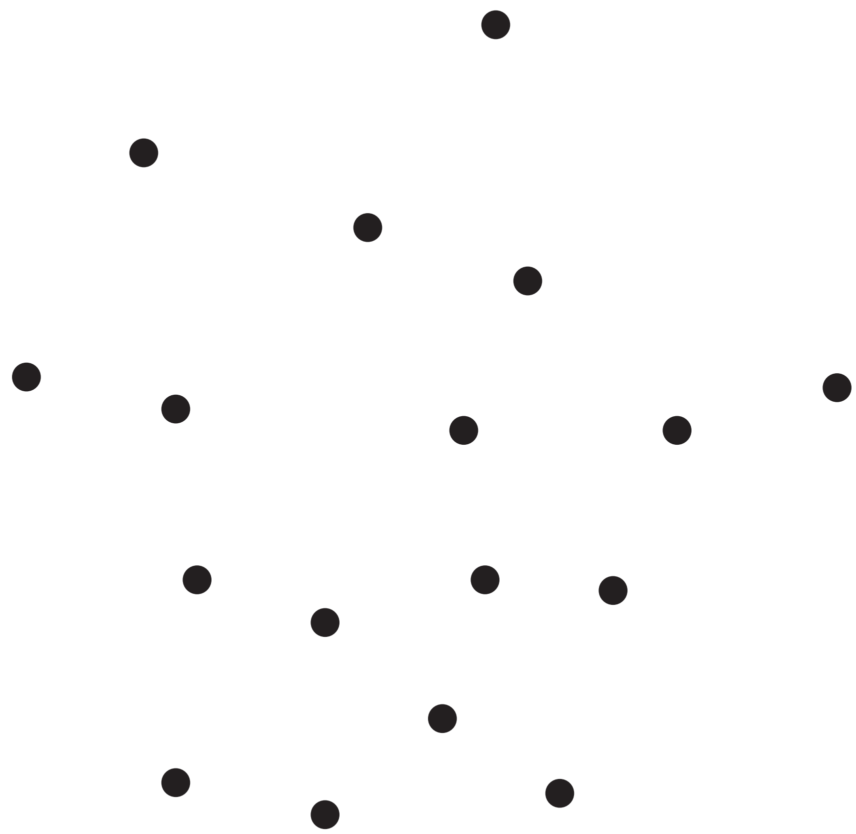
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



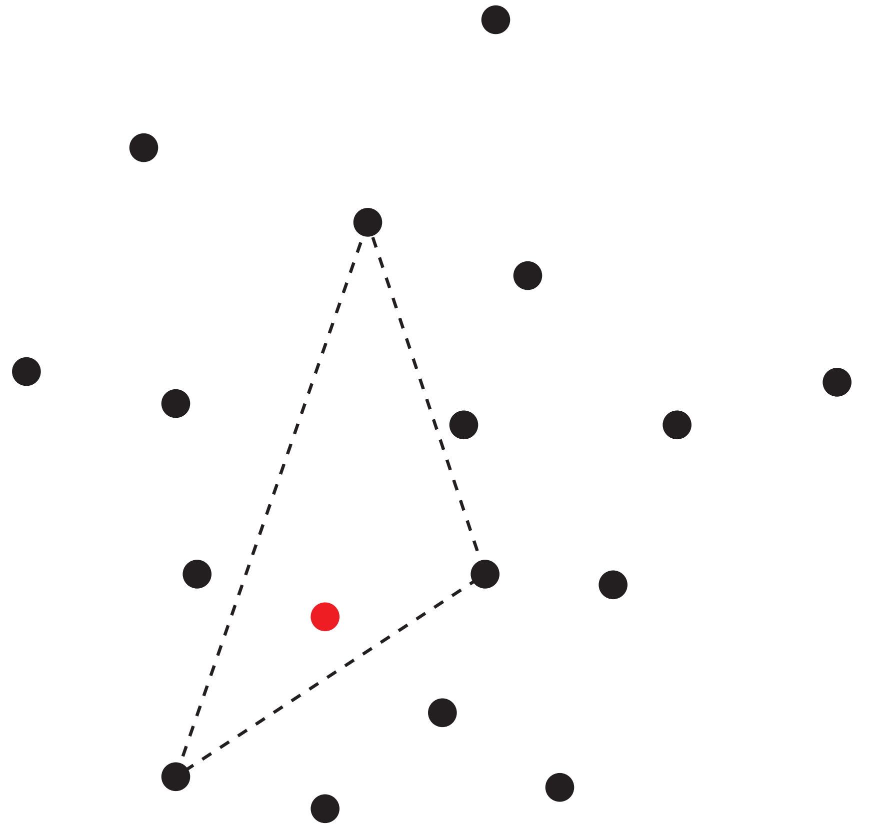
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
For each i,
For each $j, k, l \neq i$,
If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
Return the set of surviving p_i's.



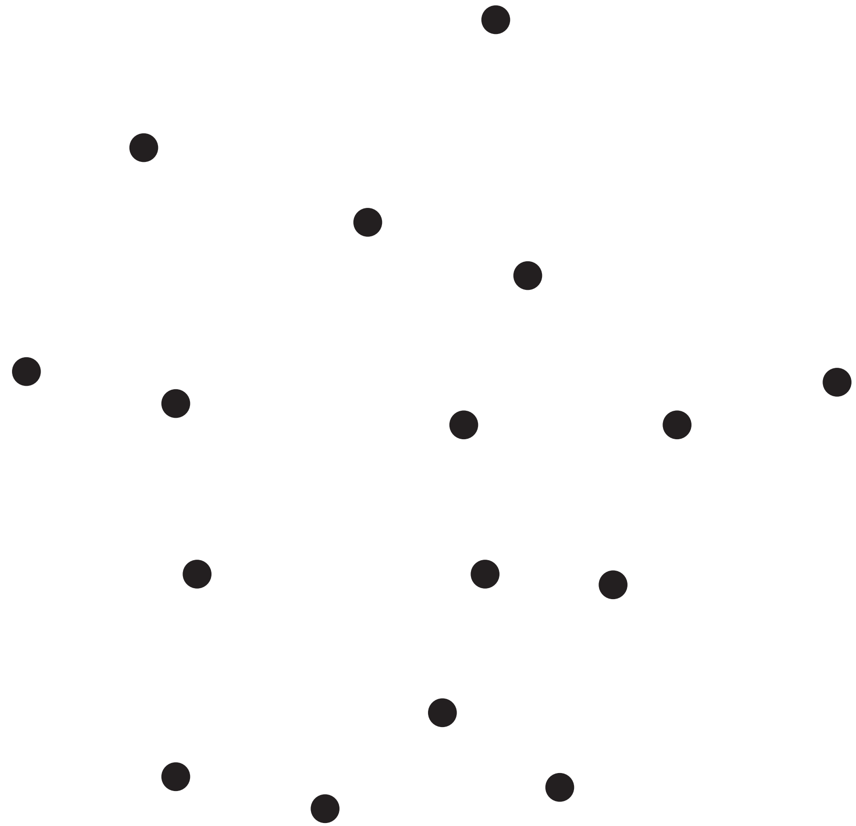
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



CONVEX HULL IN 2D

Computing the extreme points

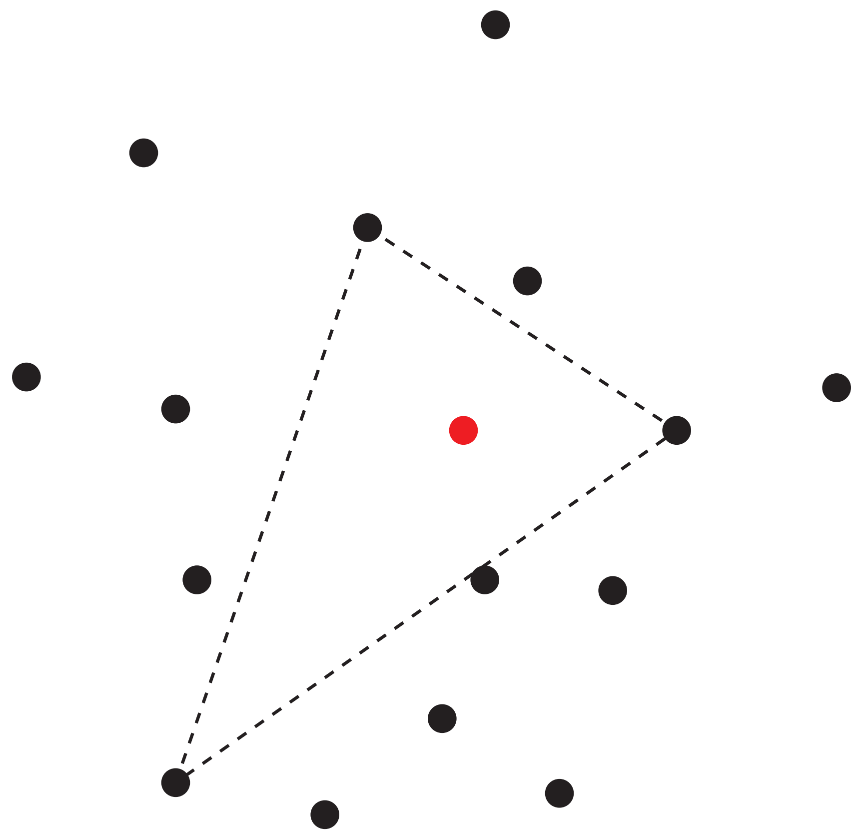
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
- For each i,
  - For each $j, k, l \neq i$,
  - If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
- Return the set of surviving p_i's.



CONVEX HULL IN 2D

Computing the extreme points

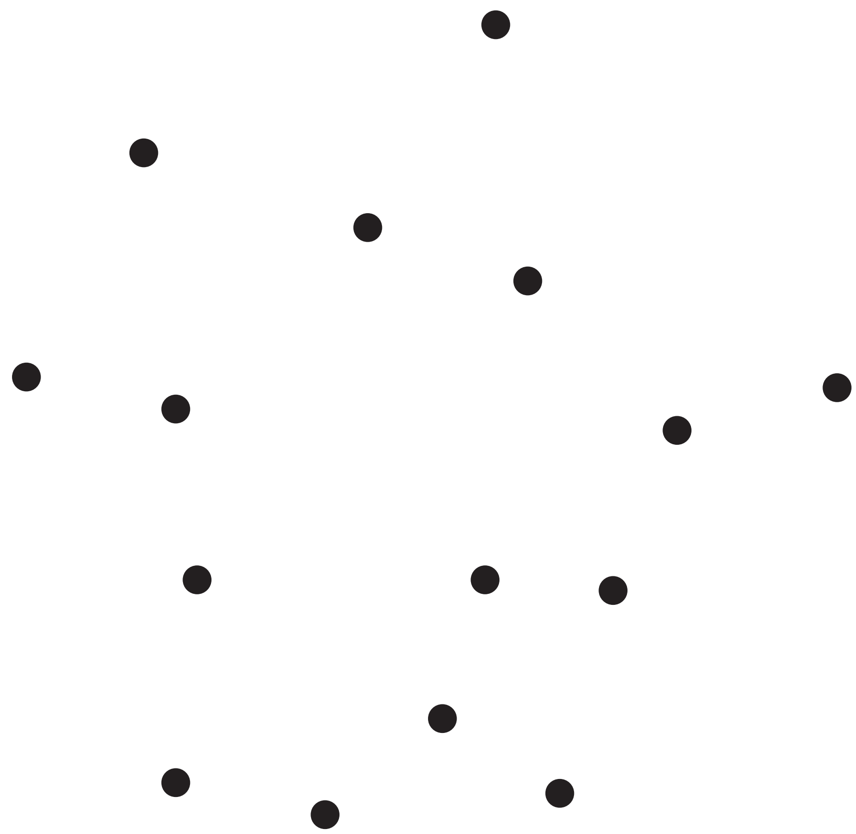
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
    For each i,
        For each $j, k, l \neq i$,
        If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
    Return the set of surviving p_i's.



Computing the extreme points

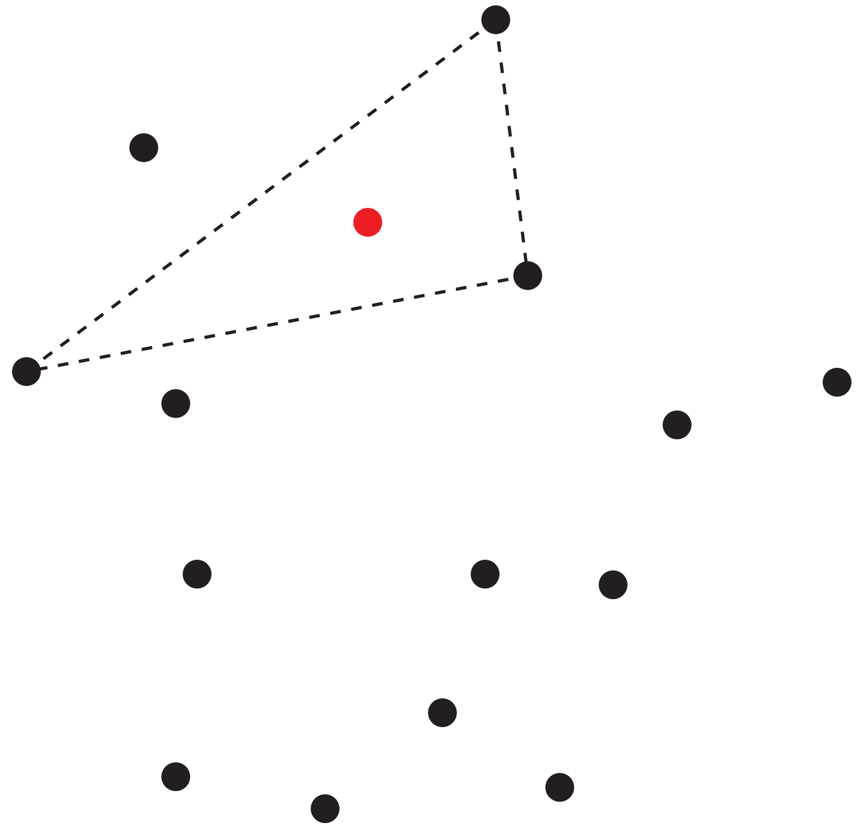
Characterization

Given $X = \{p_1, \dots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \dots, p_n$
Output: set of the extreme points

Procedure:
    For each i,
        For each $j, k, l \neq i$,
        If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
    Return the set of surviving p_i's.



CONVEX HULL IN 2D

Computing the extreme points

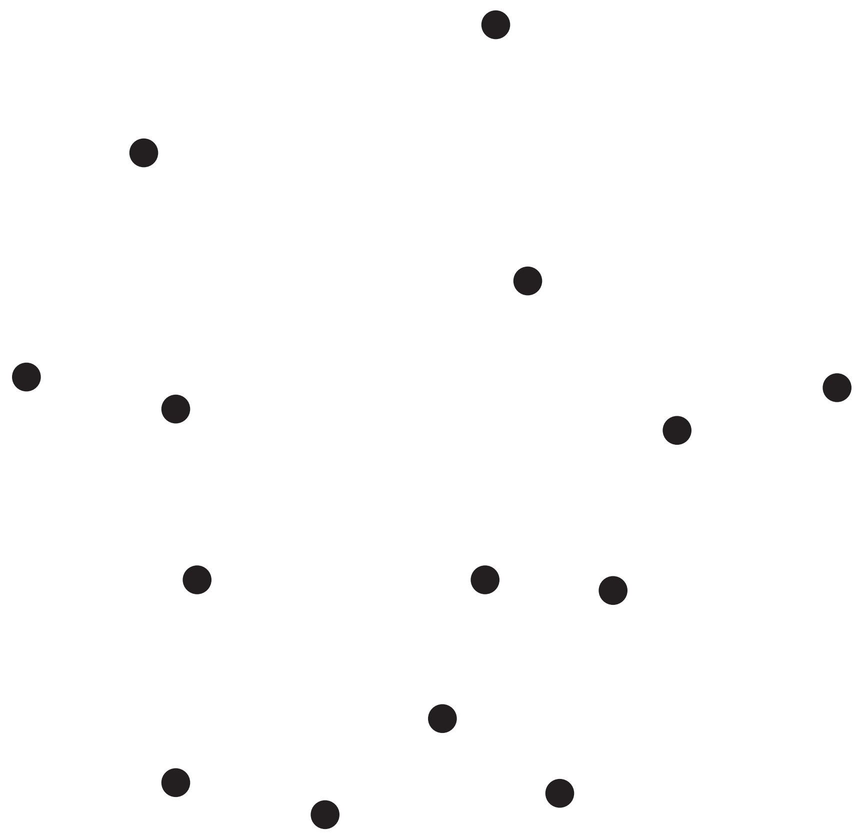
Characterization

Given $X = \{p_1, \dots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \dots, p_n$
Output: set of the extreme points

Procedure:
- For each i,
  - For each $j, k, l \neq i$,
  - If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
- Return the set of surviving p_i's.



Computing the extreme points

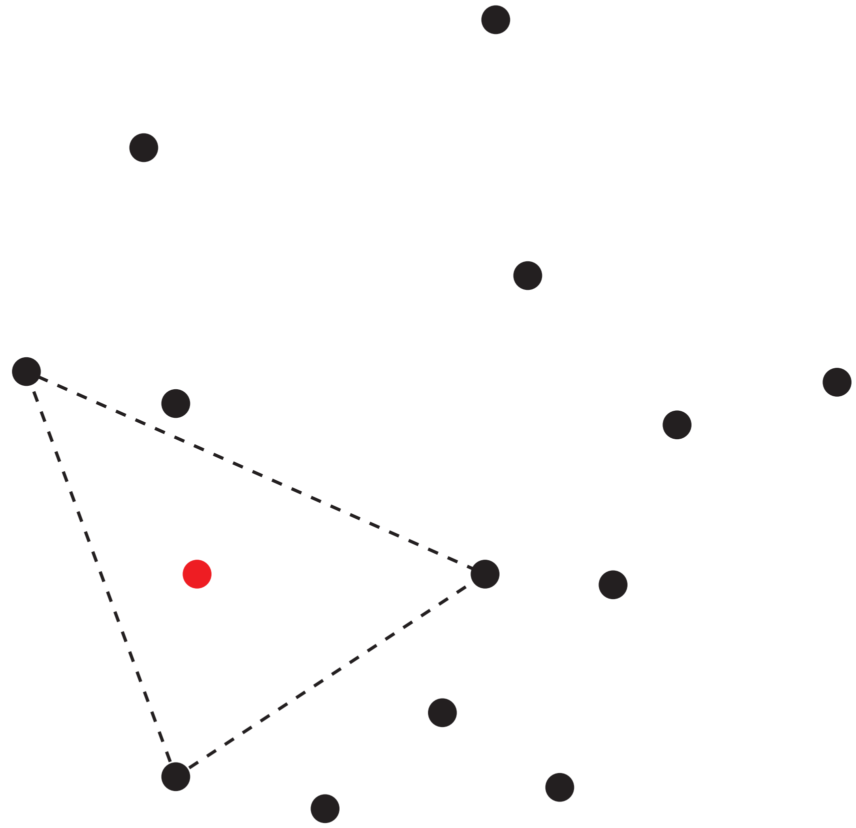
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
For each i,
For each $j, k, l \neq i$,
If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
Return the set of surviving p_i's.



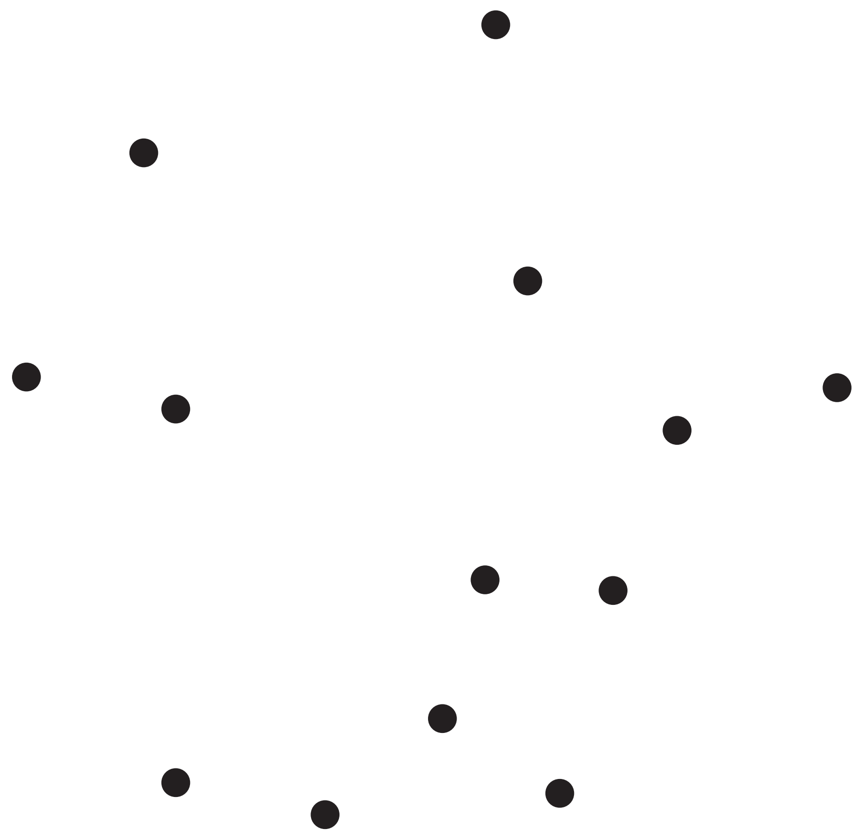
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



Computing the extreme points

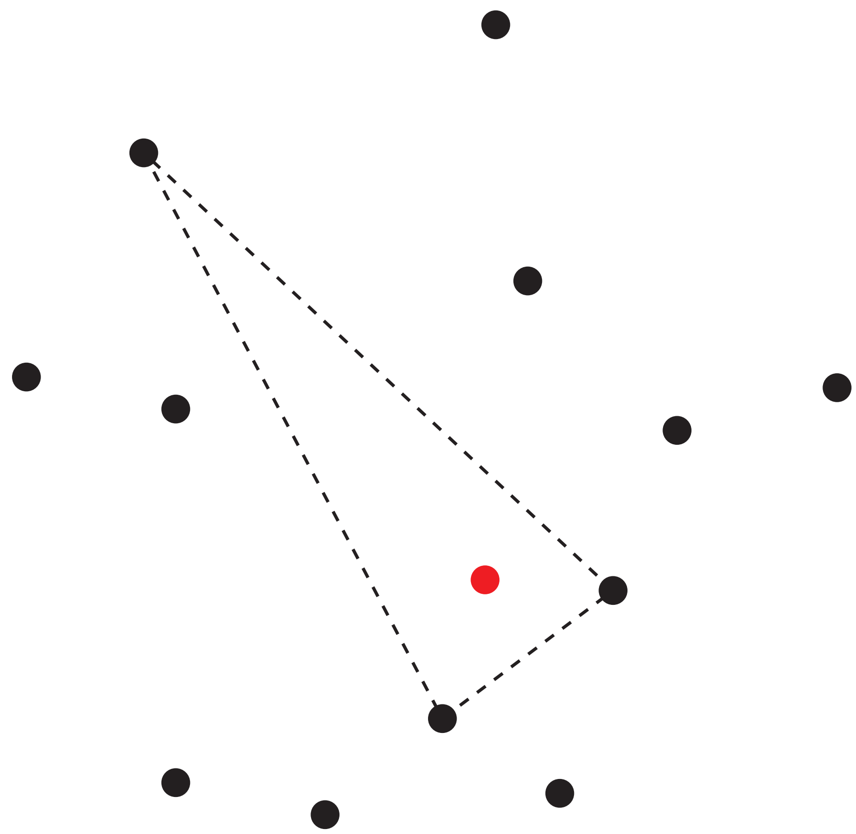
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
For each i,
For each $j, k, l \neq i$,
If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
Return the set of surviving p_i's.



CONVEX HULL IN 2D

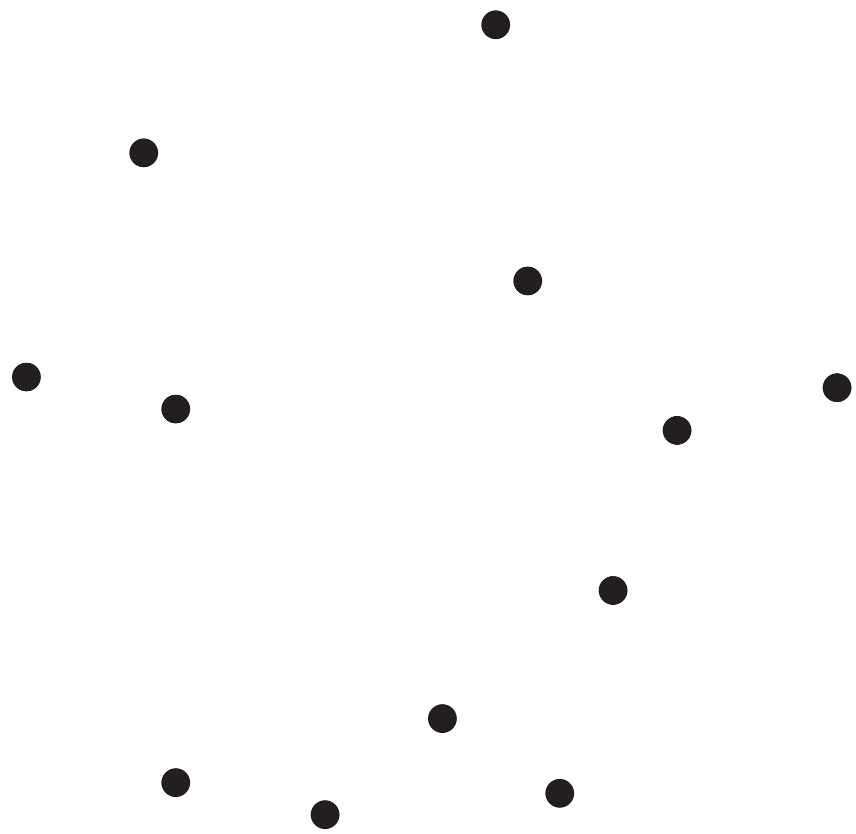
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



CONVEX HULL IN 2D

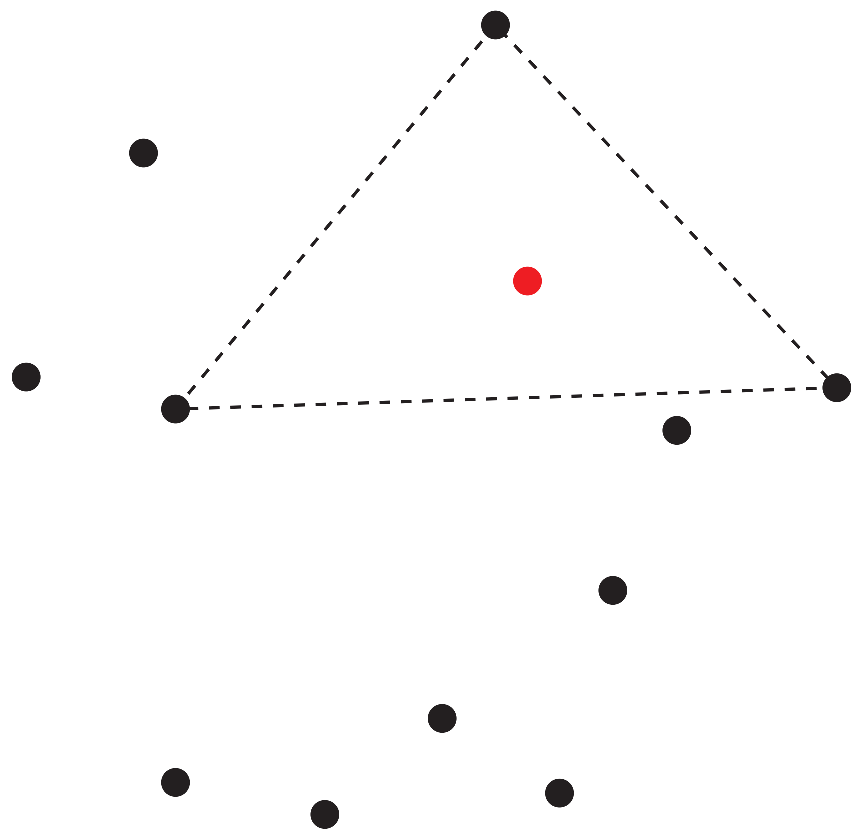
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



CONVEX HULL IN 2D

Computing the extreme points

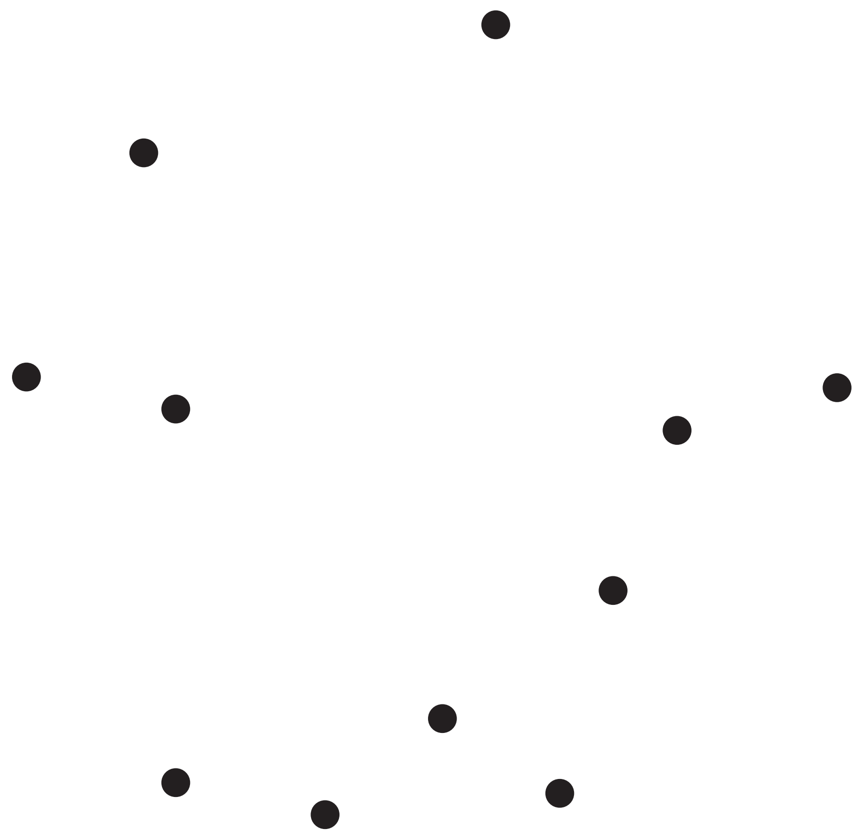
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
For each i,
For each $j, k, l \neq i$,
If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
Return the set of surviving p_i's.



Computing the extreme points

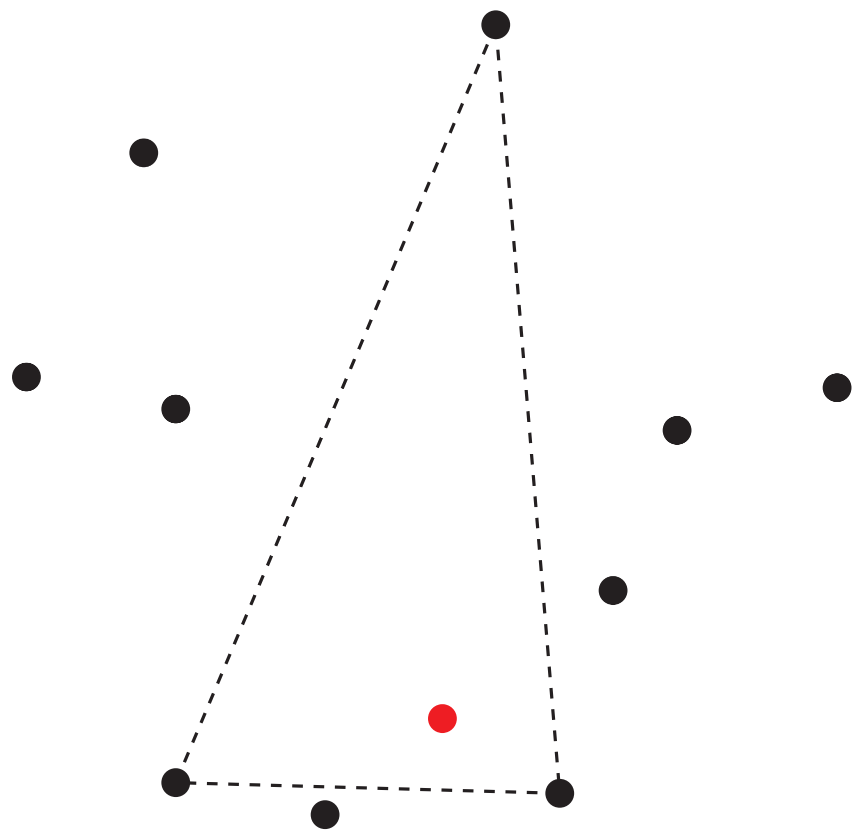
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
For each i,
For each $j, k, l \neq i$,
If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
Return the set of surviving p_i's.



Computing the extreme points

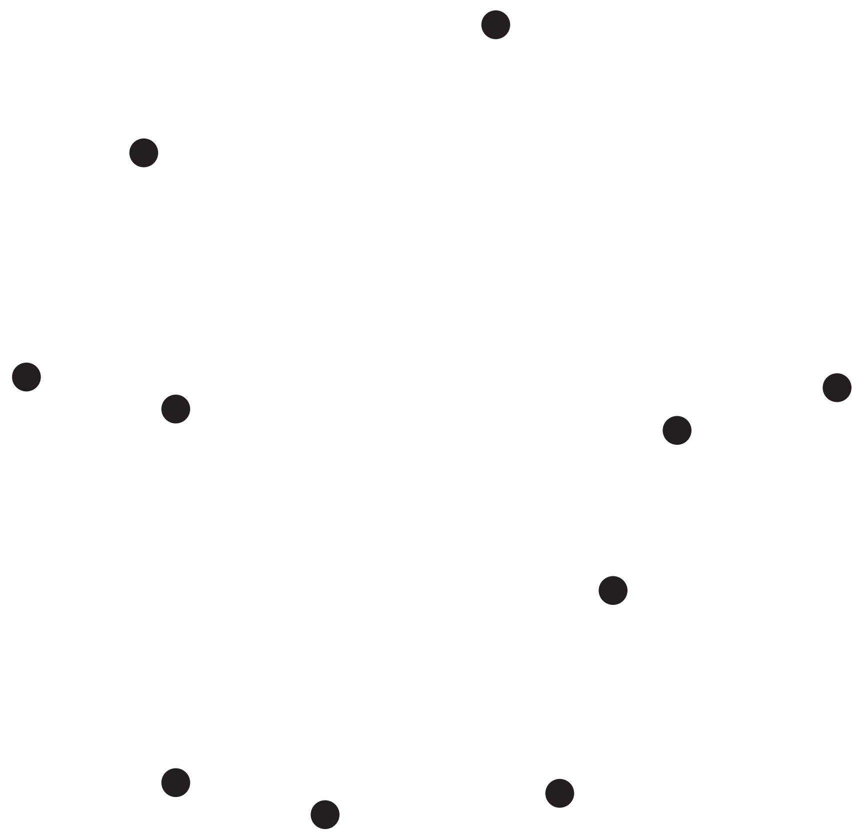
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
- For each i,
  - For each $j, k, l \neq i$,
  - If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
- Return the set of surviving p_i's.



Computing the extreme points

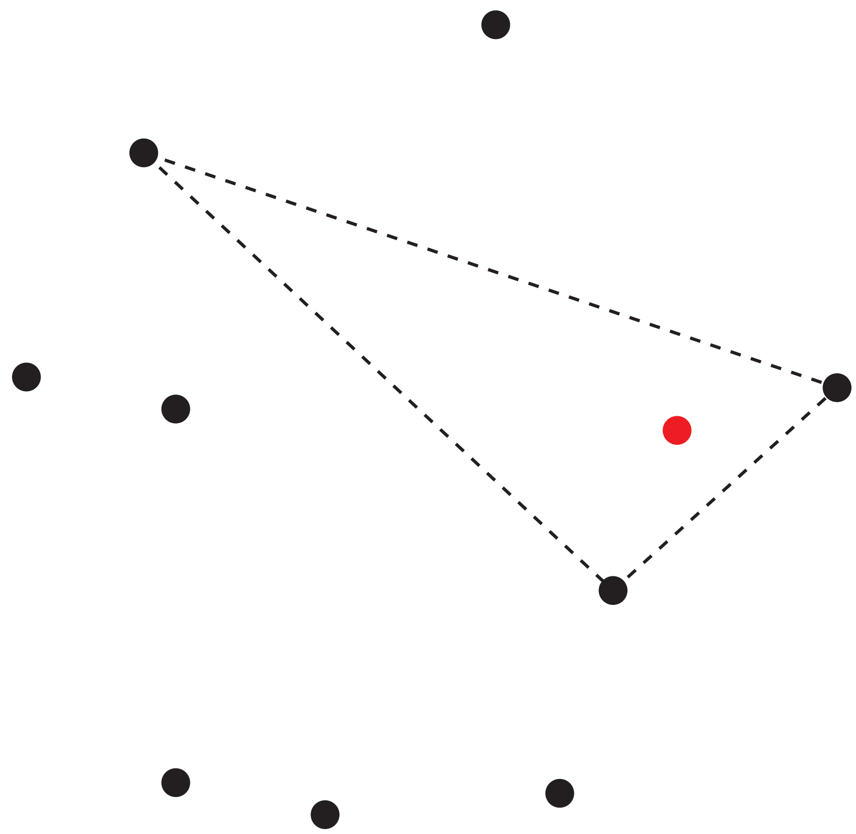
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



Computing the extreme points

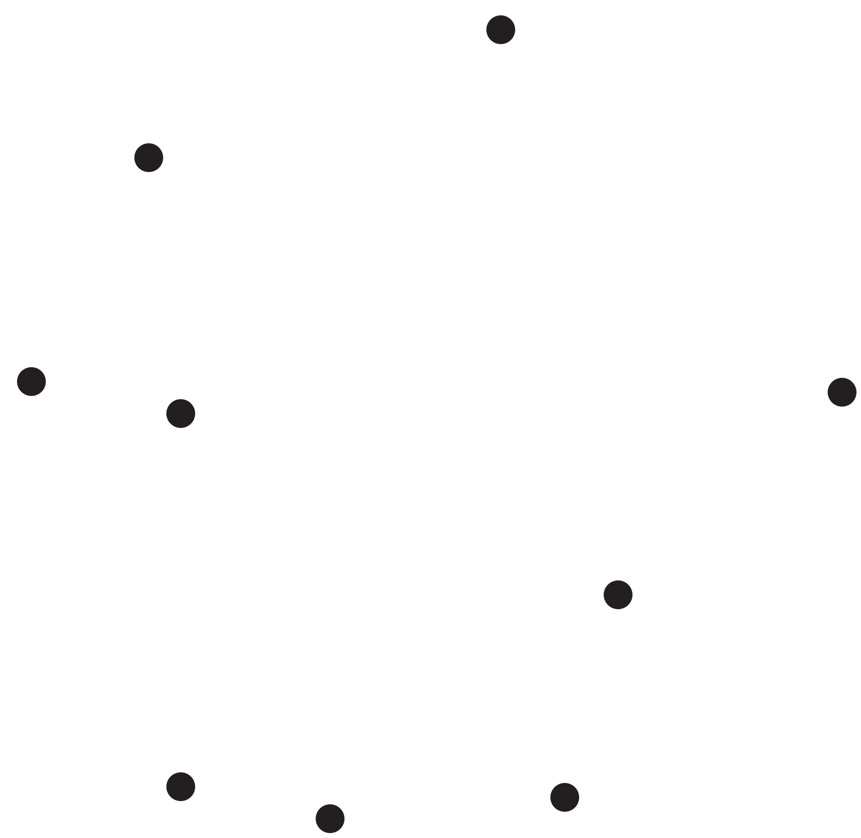
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



CONVEX HULL IN 2D

Computing the extreme points

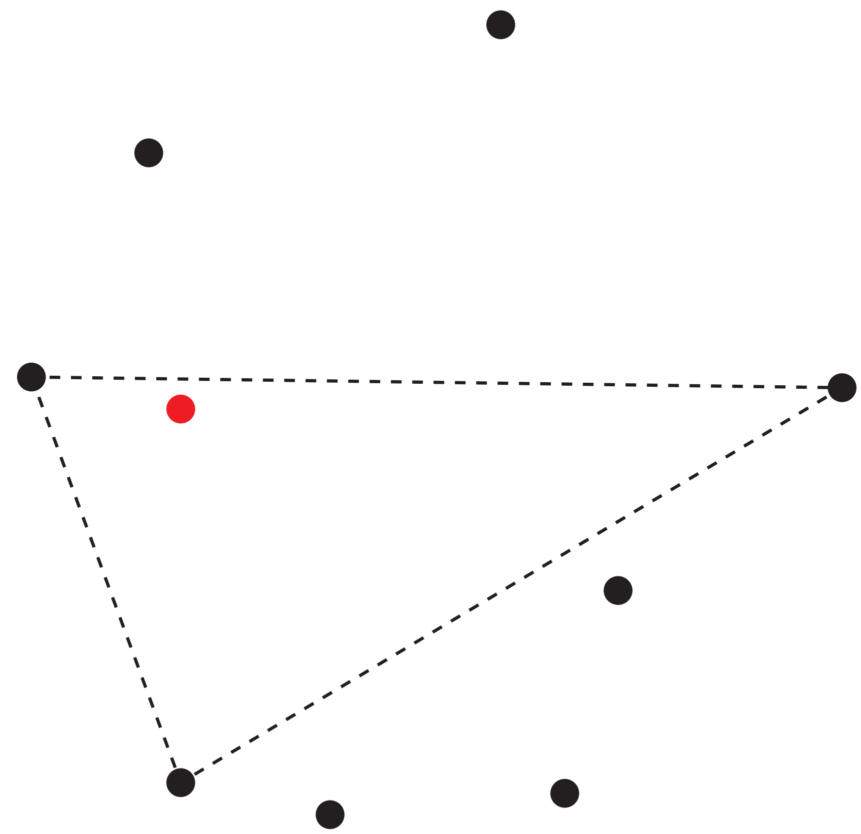
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



CONVEX HULL IN 2D

Computing the extreme points

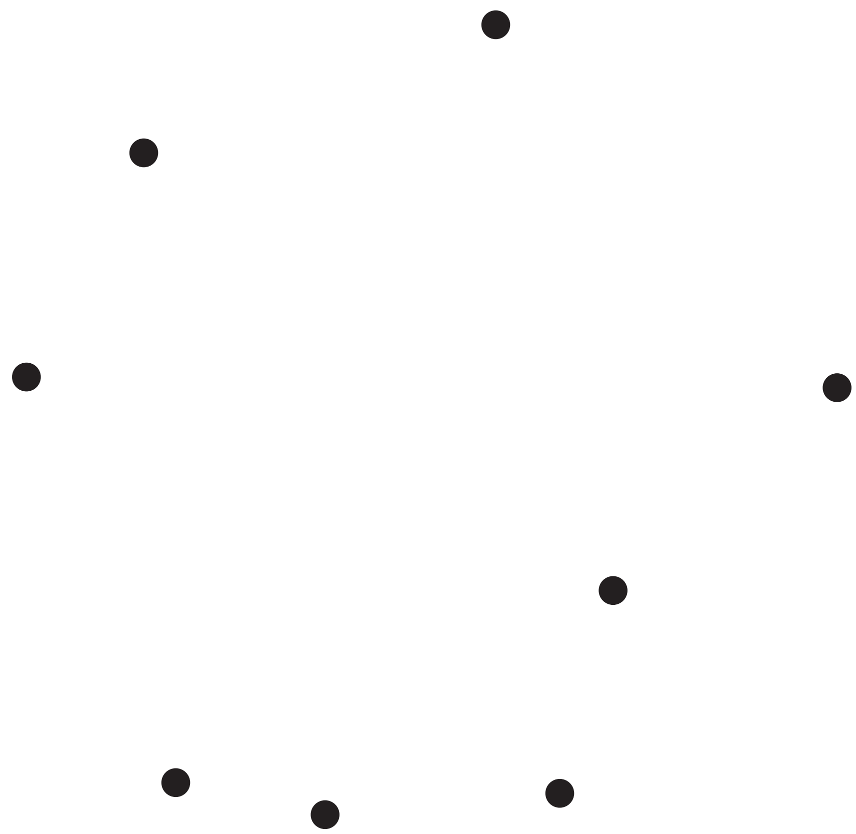
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
- For each i,
  - For each $j, k, l \neq i$,
  - If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
- Return the set of surviving p_i's.



CONVEX HULL IN 2D

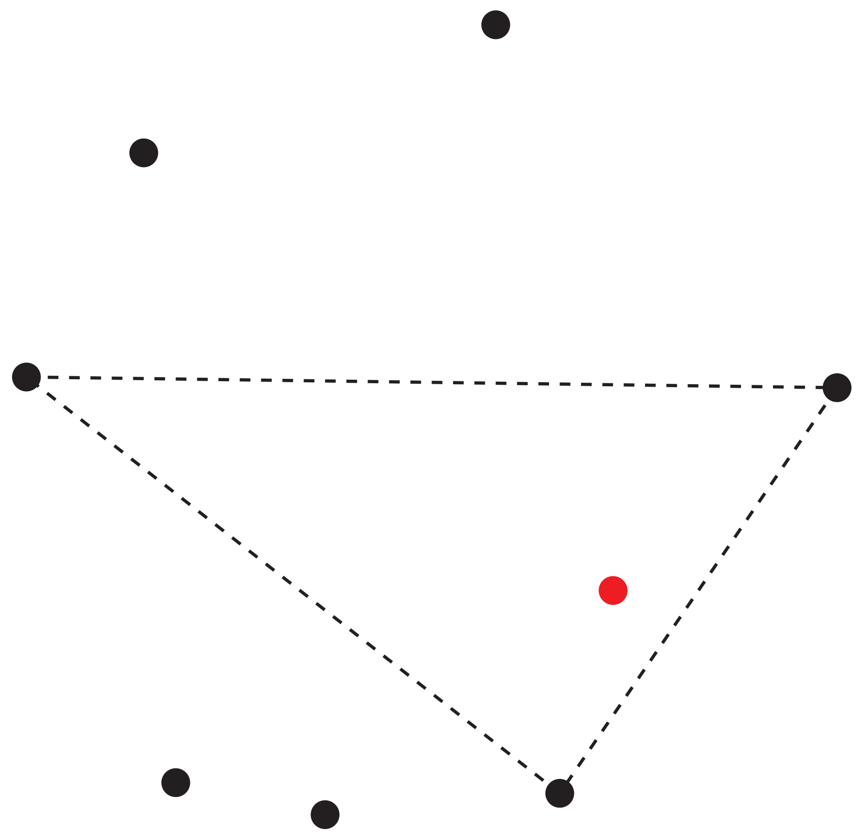
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



CONVEX HULL IN 2D

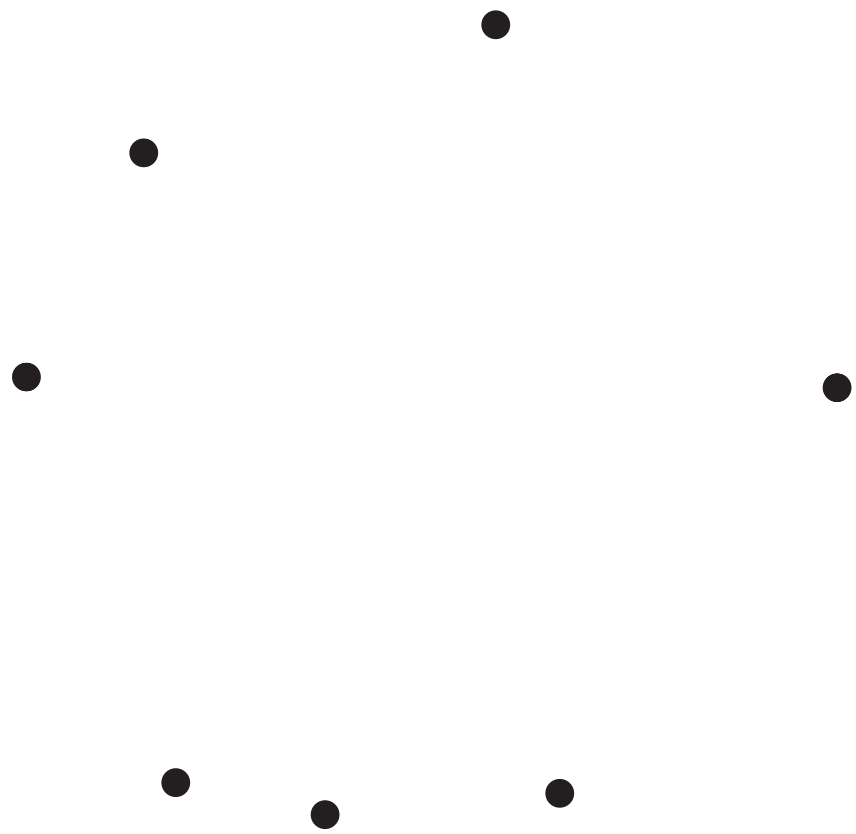
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
For each i,
For each $j, k, l \neq i$,
If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
Return the set of surviving p_i's.



CONVEX HULL IN 2D

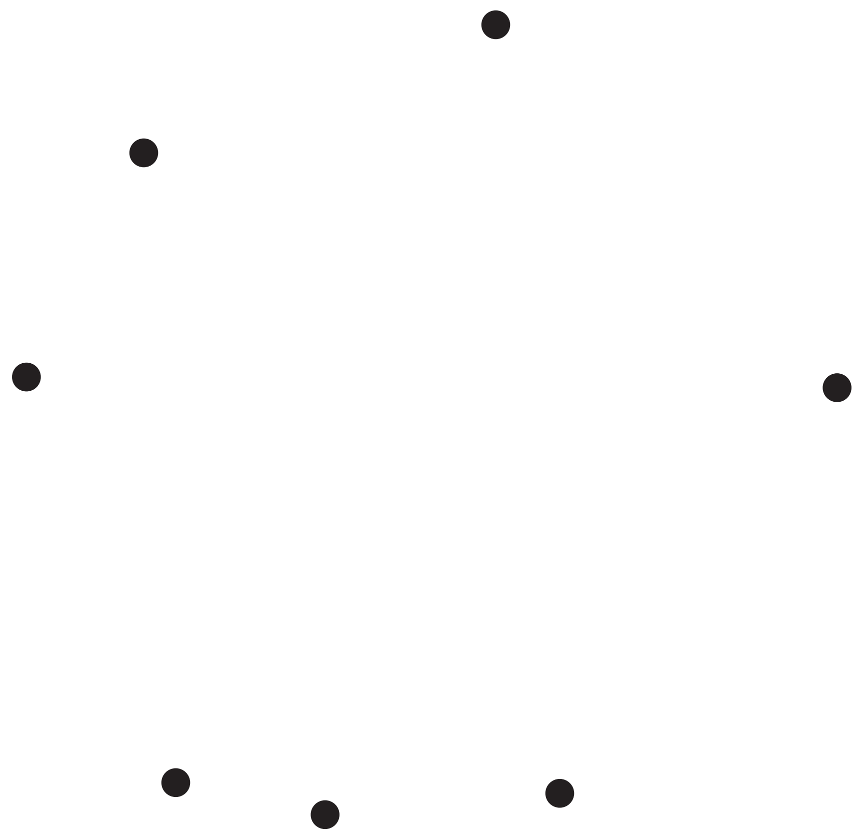
Computing the extreme points

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme points

Procedure:
  For each i,
    For each $j, k, l \neq i$,
    If p_i lies in the triangle p_j, p_k, p_l, eliminate p_i.
  Return the set of surviving p_i's.



Running time: $\Theta(n^4)$

Computing the extreme segments

Computing the extreme segments

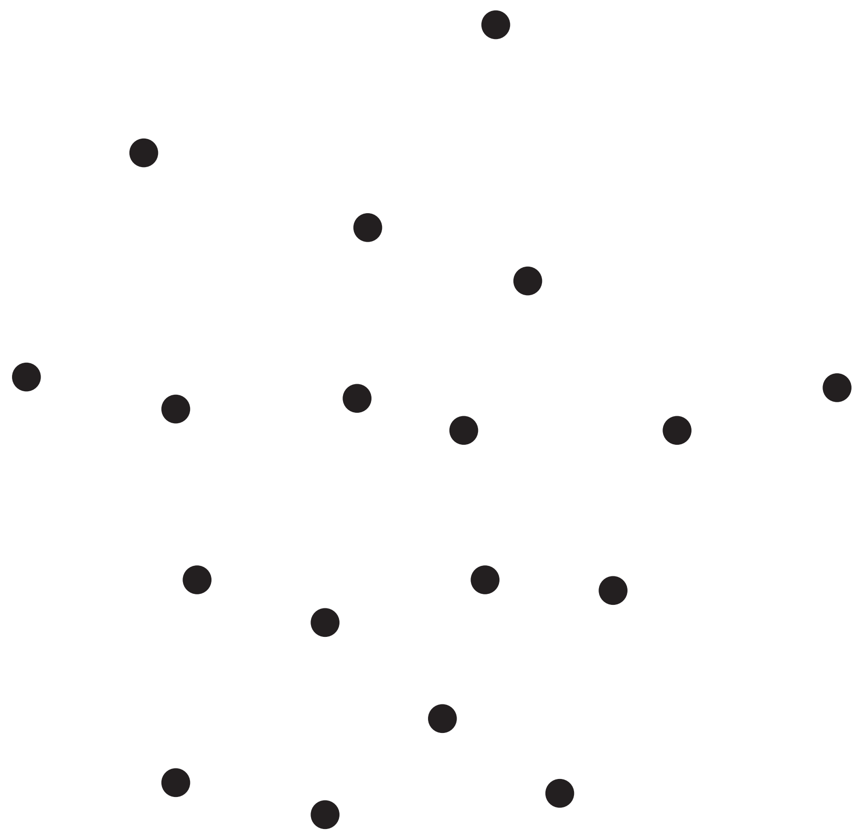
Characterization

Given $X = \{p_1, \dots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.

Computing the extreme segments

Characterization

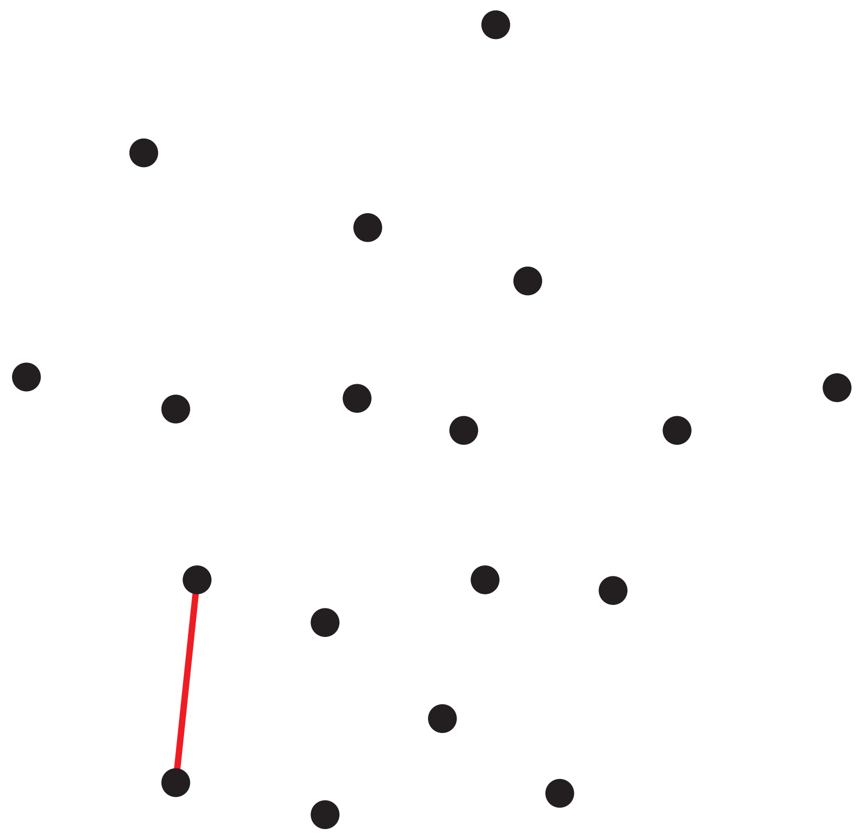
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

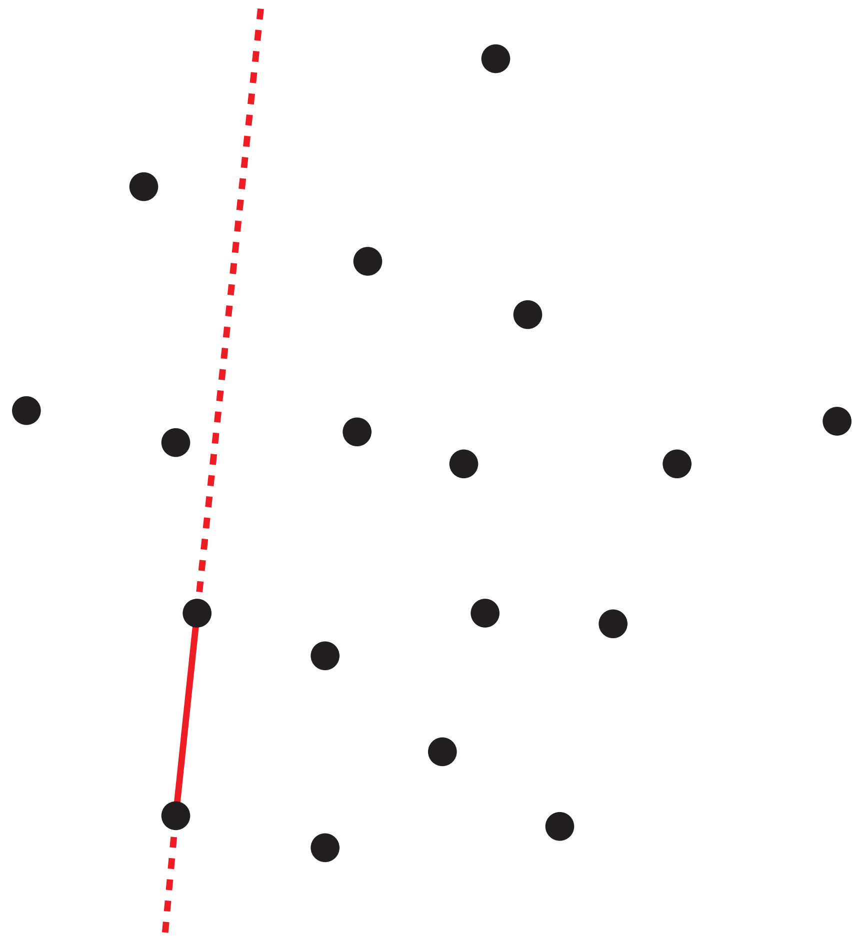
Given $X = \{p_1, \dots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

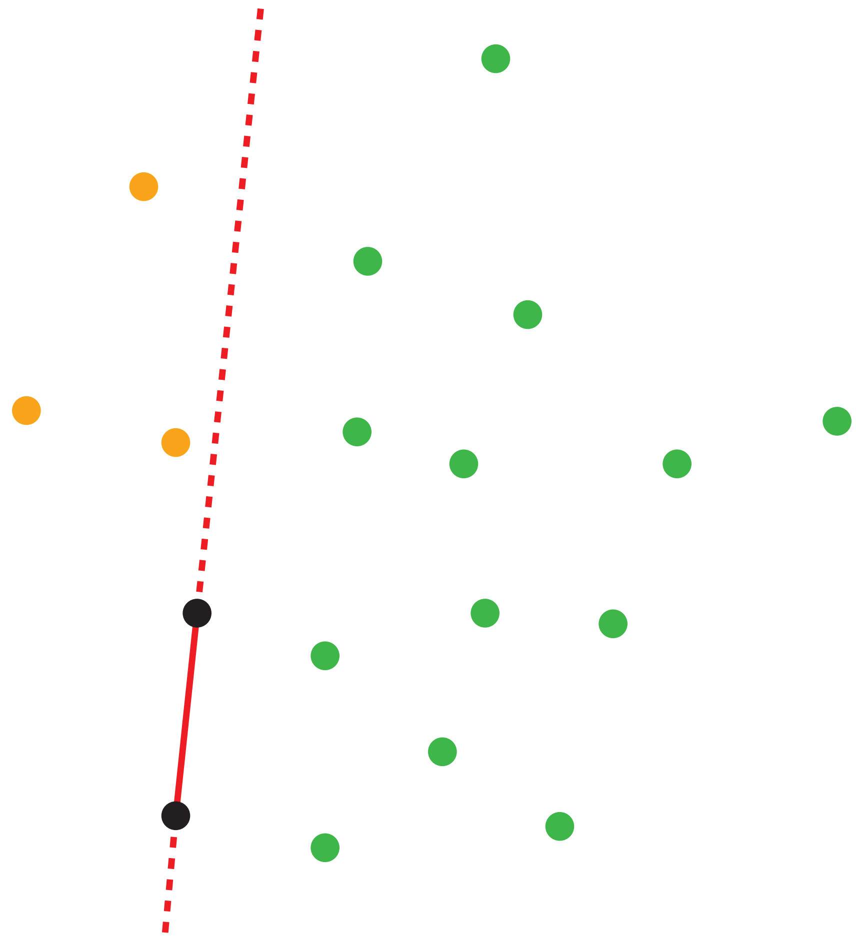
Given $X = \{p_1, \ldots, p_n\}$, the segment p_ip_j is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line p_ip_j.



Computing the extreme segments

Characterization

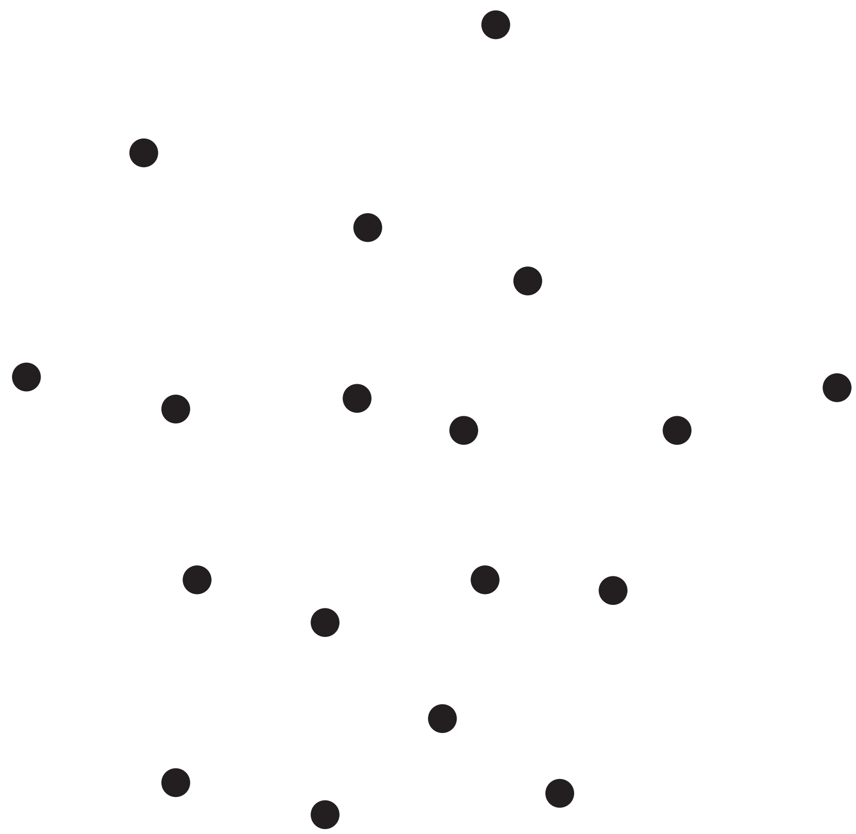
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

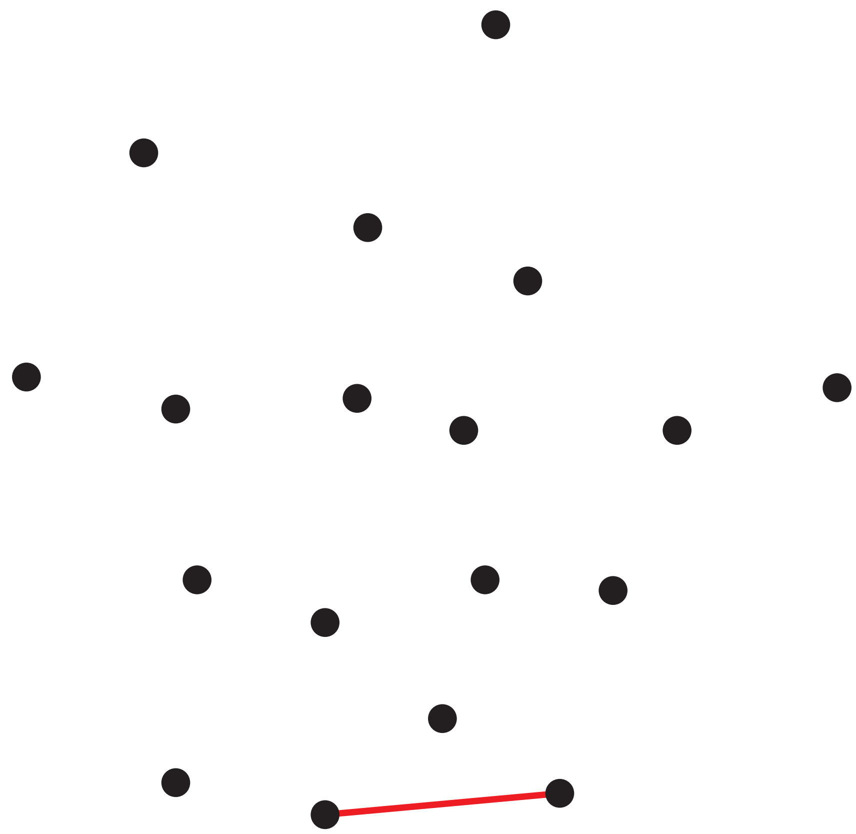
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

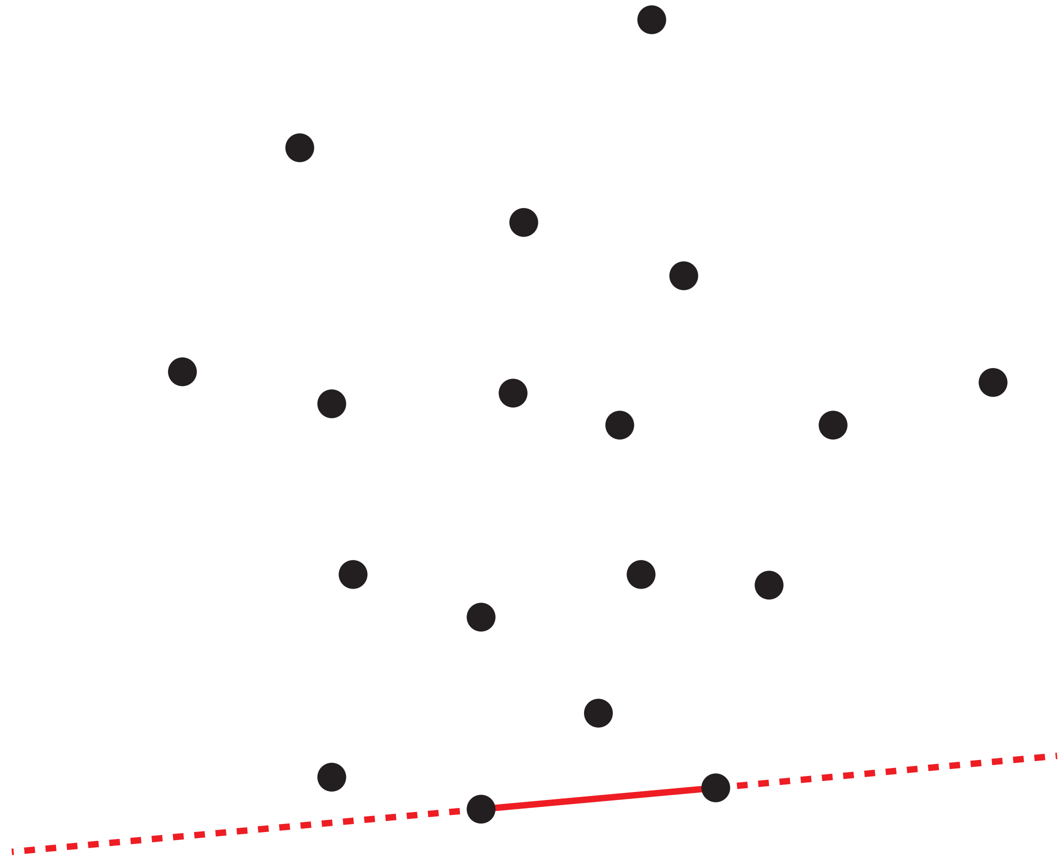
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

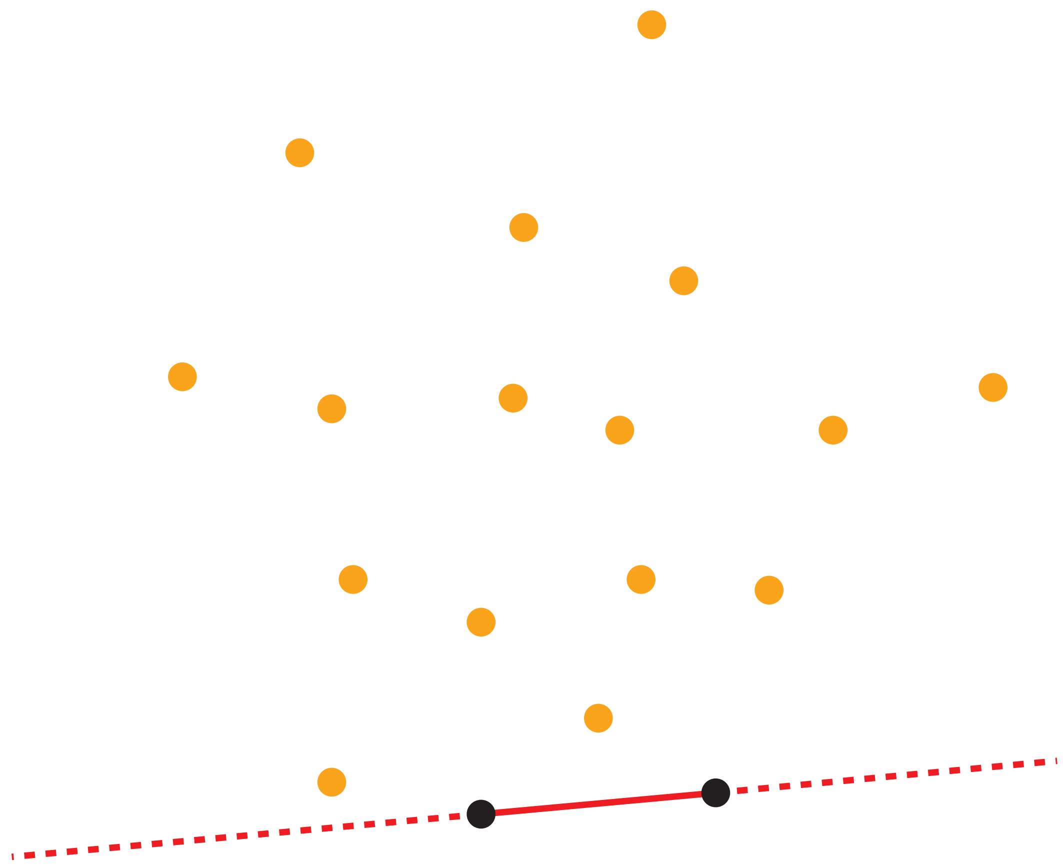
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



CONVEX HULL IN 2D

Computing the extreme segments

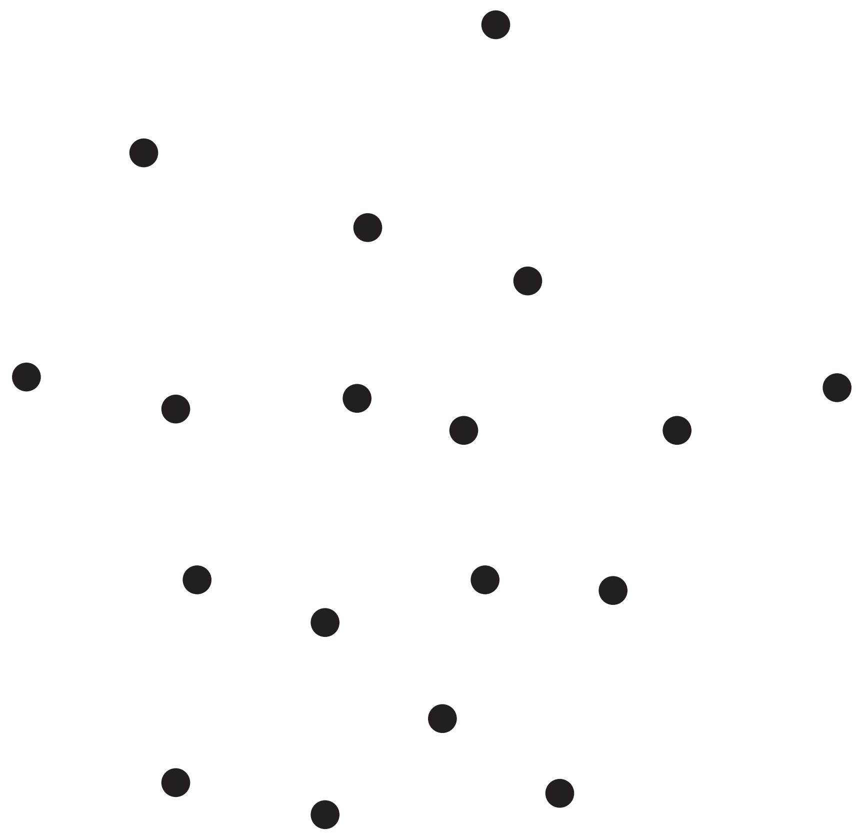
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme segments

Procedure:
For each i, j,
Check whether all p_k with $k \neq i, j$ lie in the same halfplane defined by $p_i p_j$.
In the affirmative, return the segment $p_i p_j$.



Computing the extreme segments

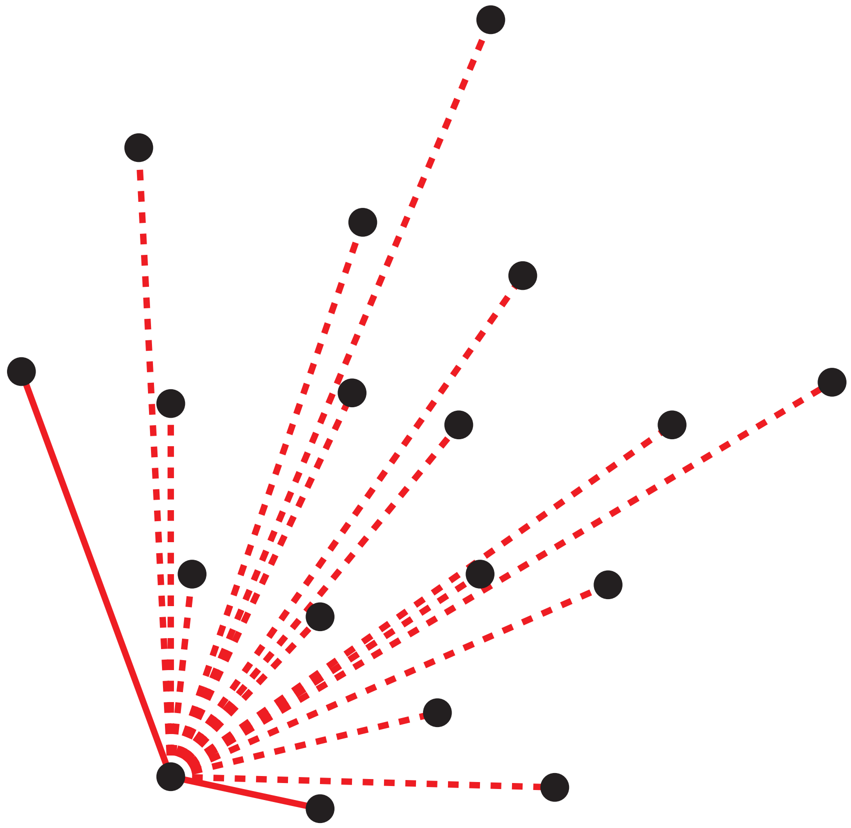
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme segments

Procedure:
For each i, j,
Check whether all p_k with $k \neq i, j$
lie in the same halfplane defined by $p_i p_j$.
In the affirmative, return the segment $p_i p_j$.



CONVEX HULL IN 2D

Computing the extreme segments

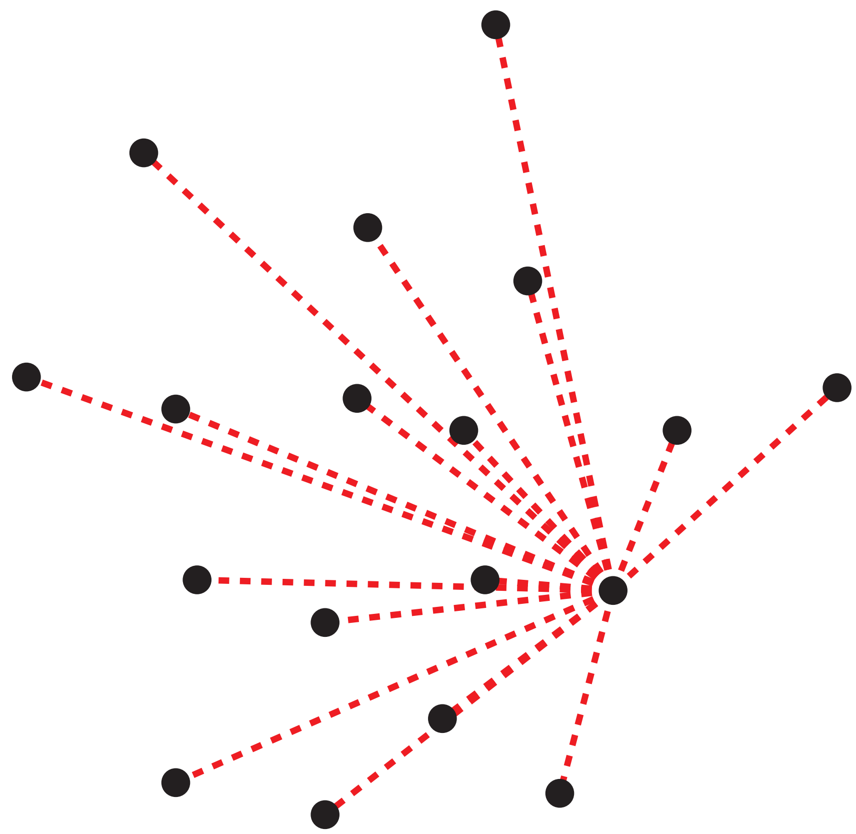
Characterization

Given $X = \{p_1, \dots, p_n\}$, the segment p_ip_j is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line p_ip_j.

Algorithm

Input: $p_1, \dots, p_n$
Output: set of the extreme segments

Procedure:
For each i, j,
Check whether all p_k with $k \neq i, j$
lie in the same halfplane defined by p_ip_j.
In the affirmative, return the segment p_ip_j.



Computing the extreme segments

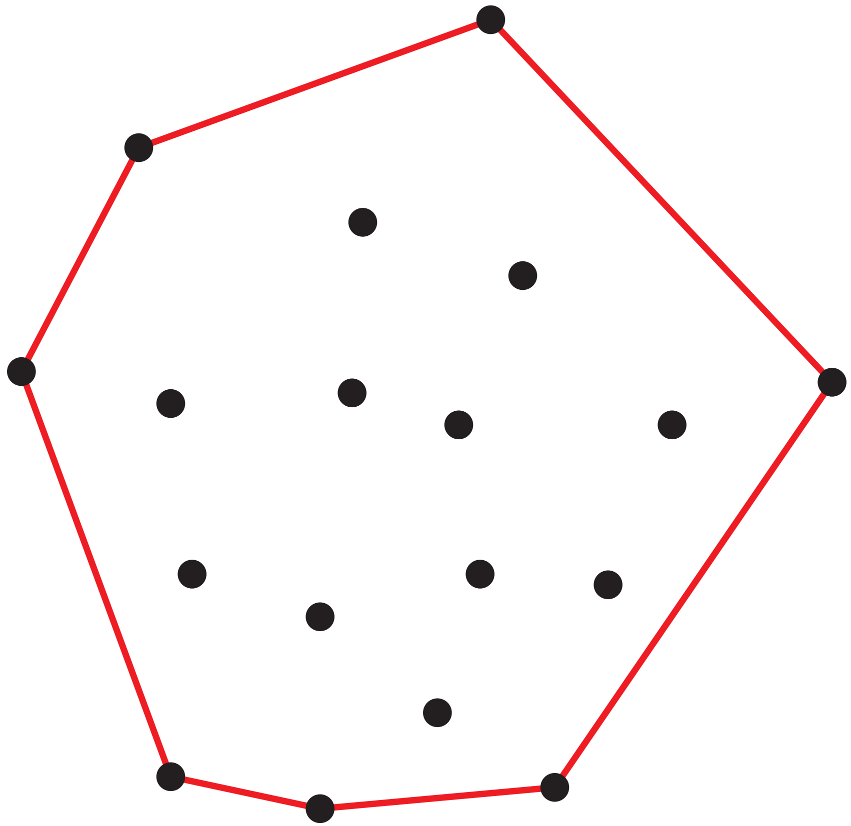
Characterization

Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme segments

Procedure:
For each i, j,
Check whether all p_k with $k \neq i, j$
lie in the same halfplane defined by $p_i p_j$.
In the affirmative, return the segment $p_i p_j$.



Computing the extreme segments

Characterization

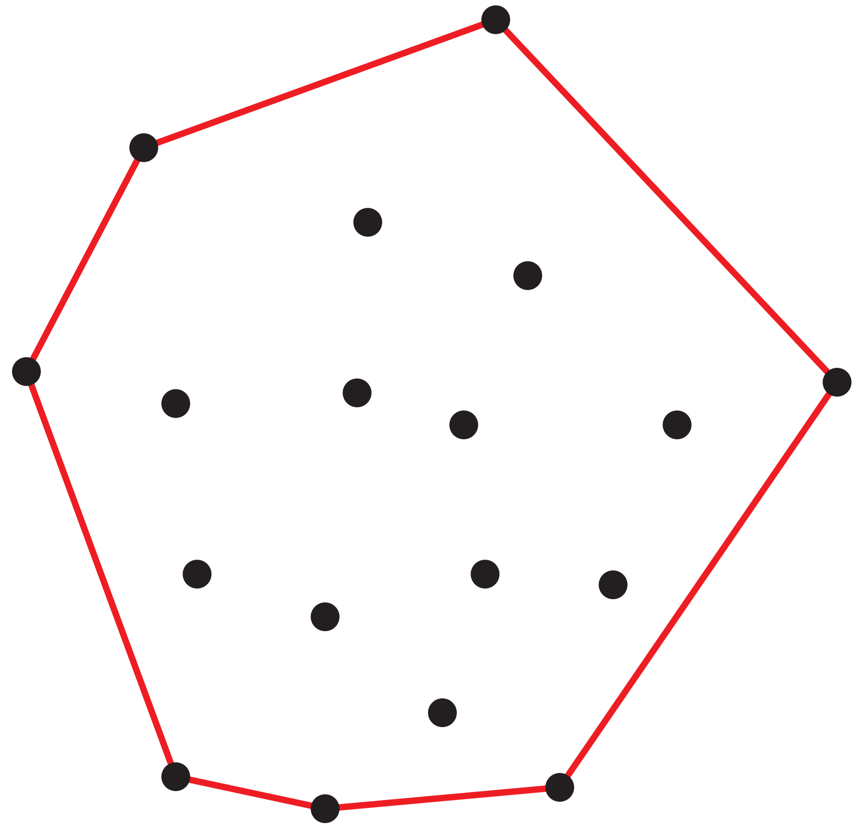
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.

Algorithm

Input: $p_1, \ldots, p_n$
Output: set of the extreme segments

Procedure:
For each i, j,
Check whether all p_k with $k \neq i, j$
lie in the same halfplane defined by $p_i p_j$.
In the affirmative, return the segment $p_i p_j$.

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



Computing the convex hull

CONVEX HULL IN 2D

Computing the convex hull **(sorted list of its vertices)**

Computing the convex hull

Input:
$P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ a set of n points in the plane

Output:
l, the list of the vertices of $ch(P)$ sorted in counterclockwise order

CONVEX HULL IN 2D

Computing the convex hull

Input:
$P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ a set of n points in the plane

Output:
l, the list of the vertices of $ch(P)$ sorted in counterclockwise order

Characterization
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an edge of the convex hull of X if and only if all the points p_k with $k \neq i, j$ lie to the left of the oriented line $p_i p_j$.

Jarvis march

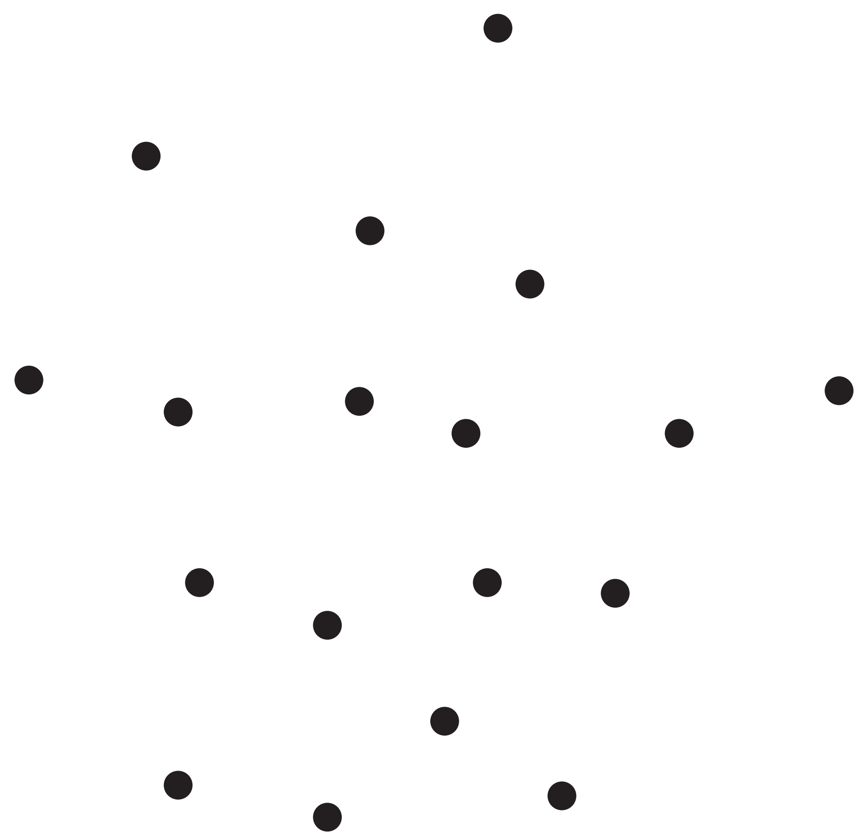
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l

CONVEX HULL IN 2D

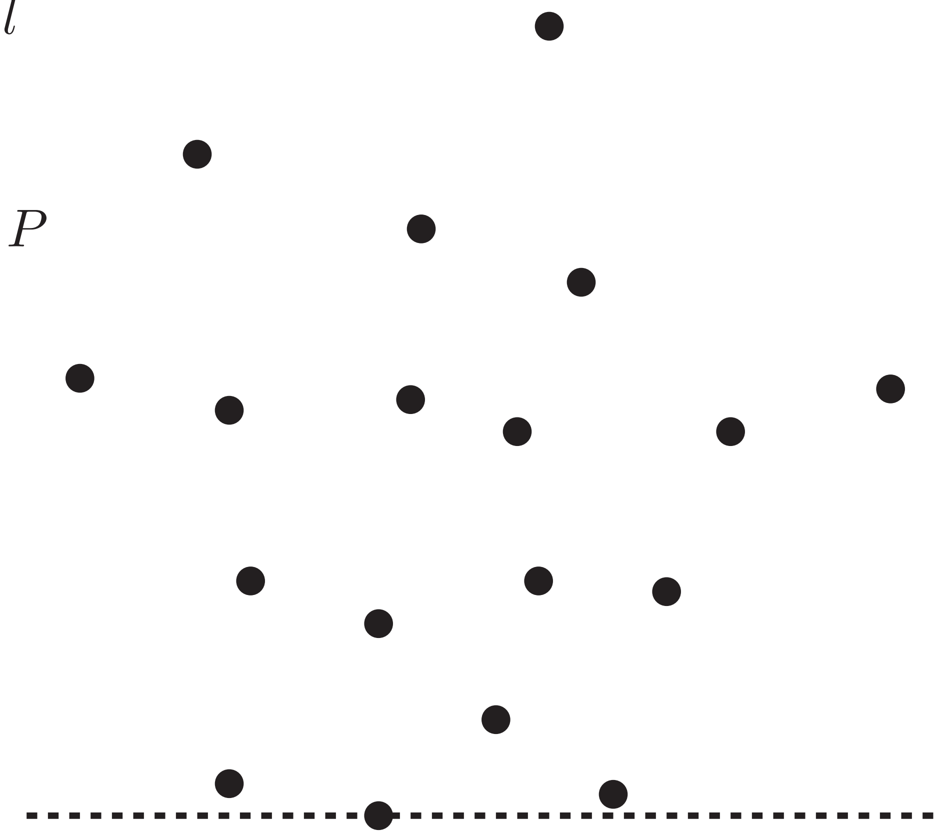
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



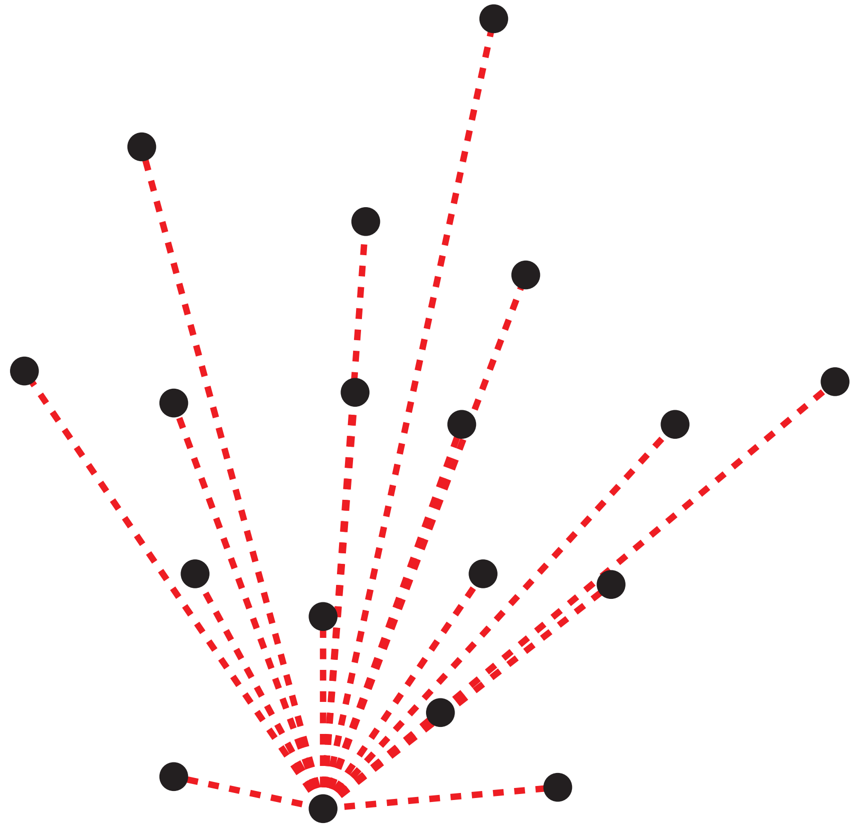
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



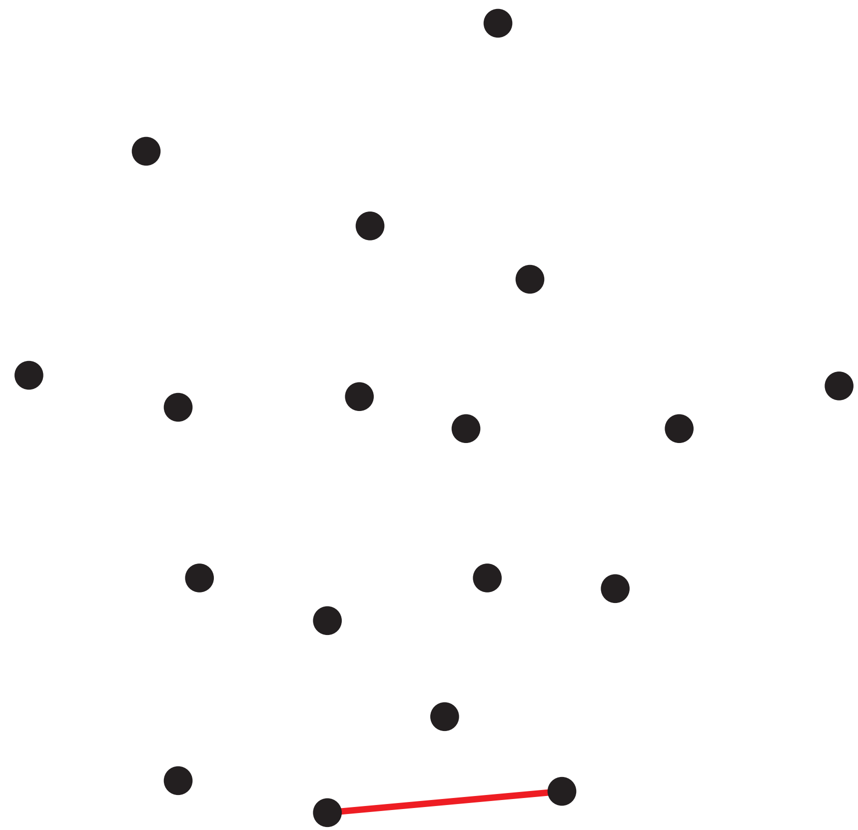
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



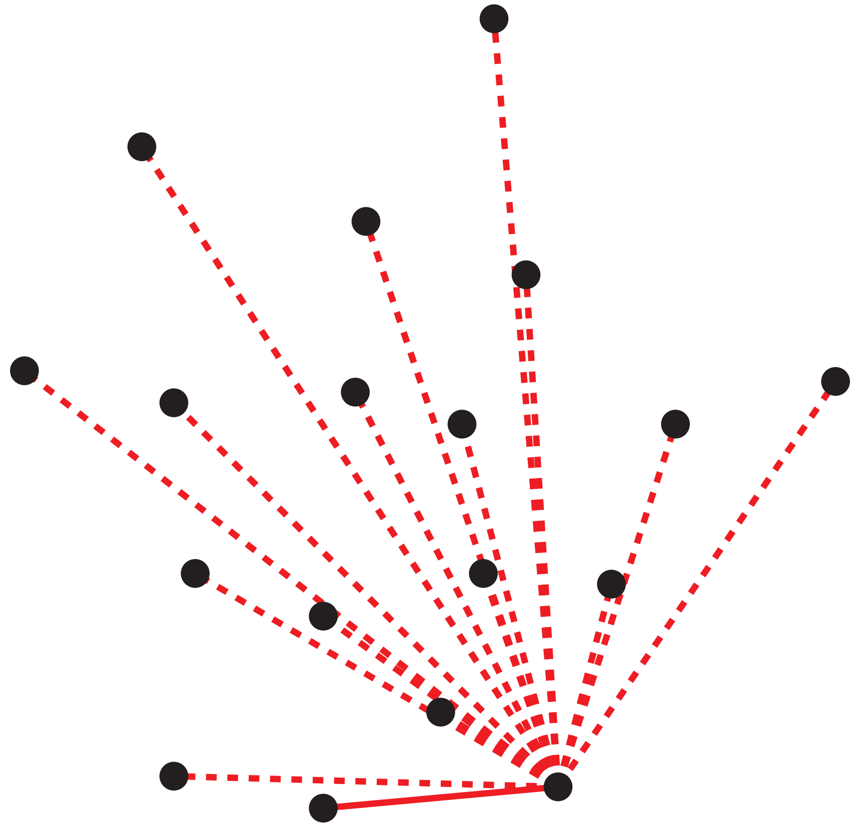
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \text{Last}(l) \neq \text{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



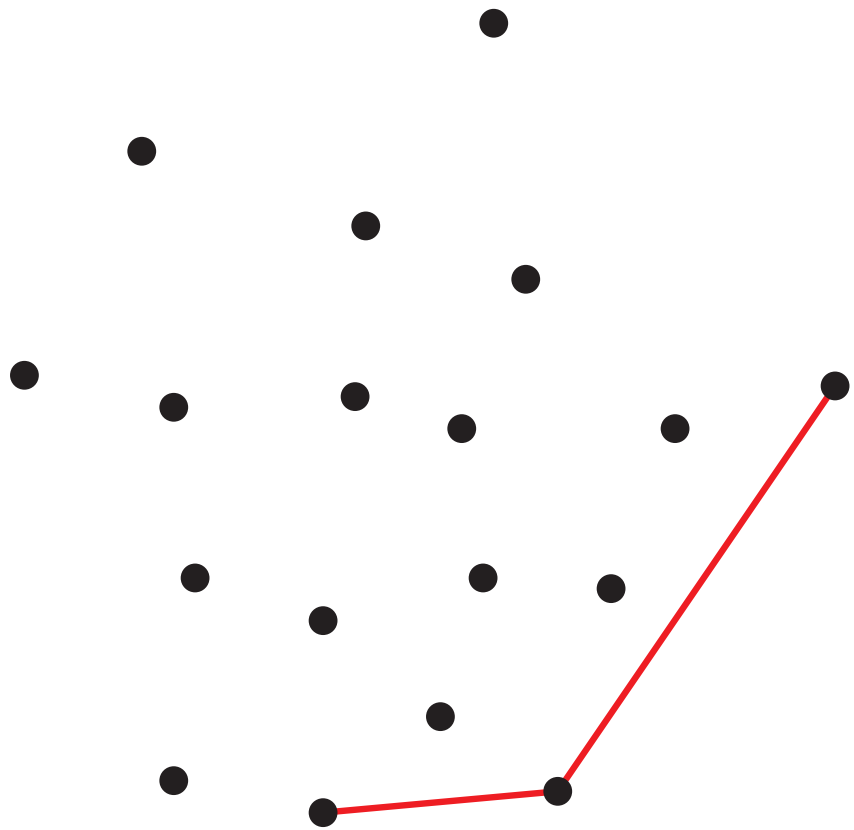
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \text{Last}(l) \neq \text{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



Jarvis march

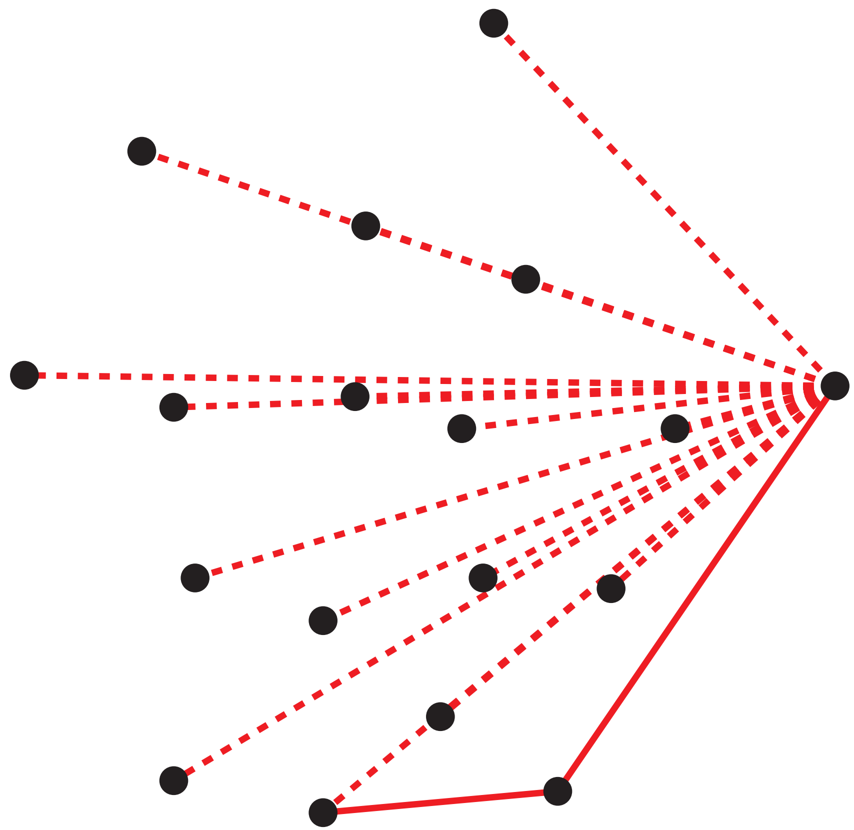
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



CONVEX HULL IN 2D

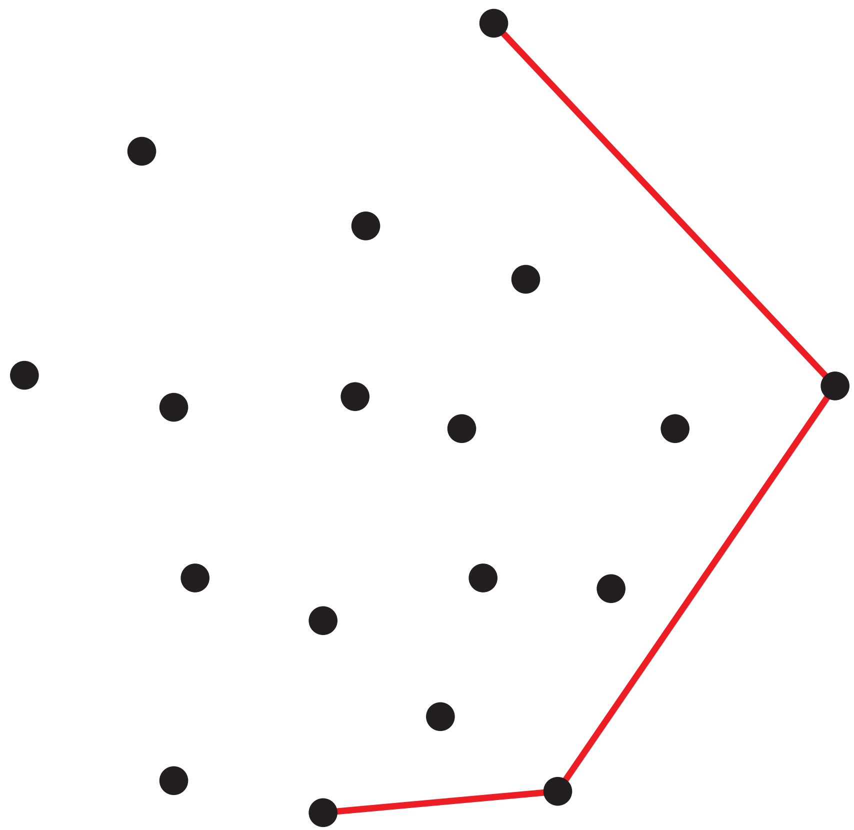
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



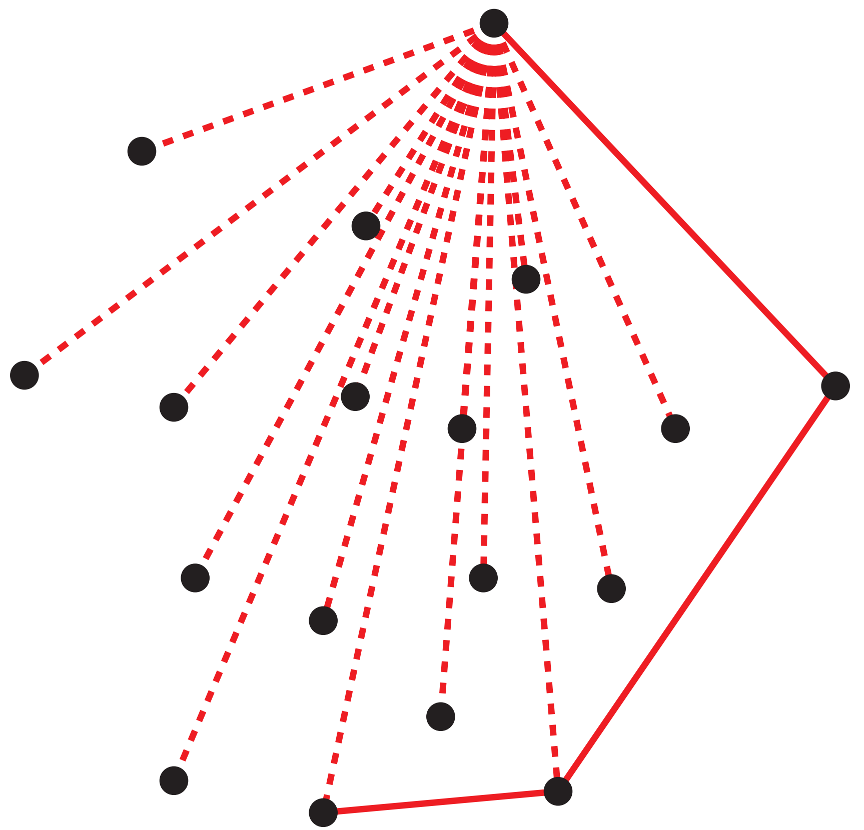
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



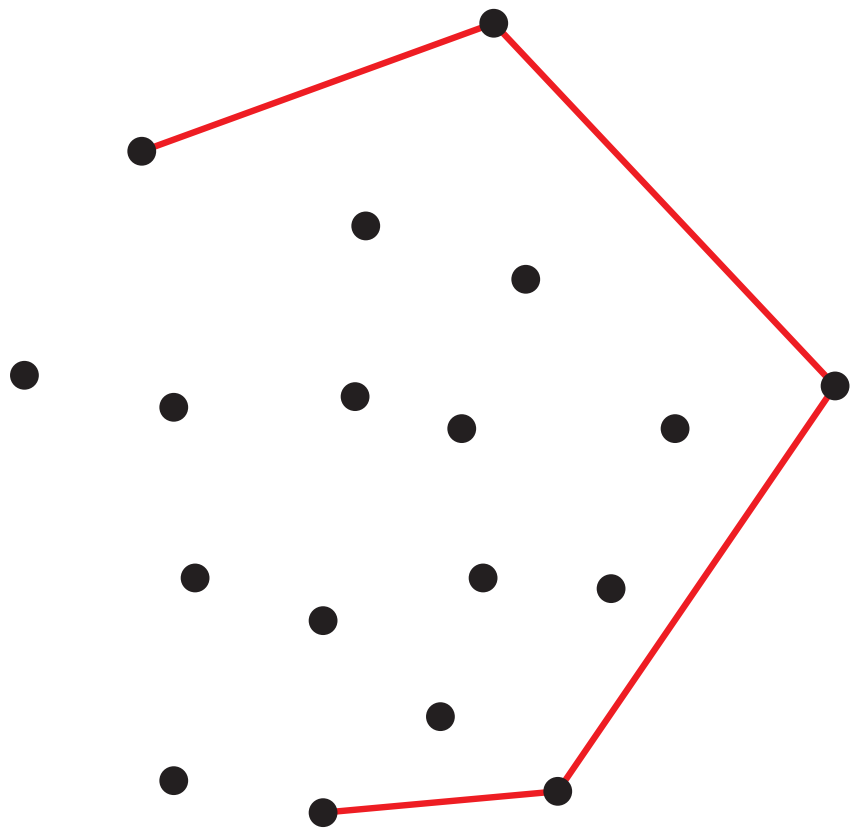
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



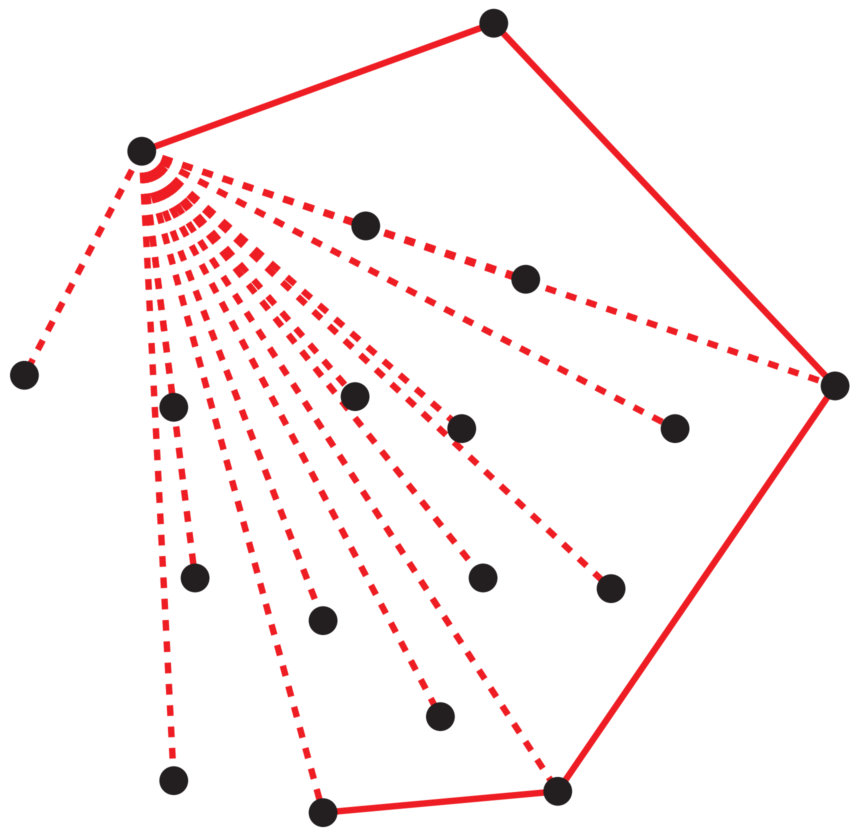
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \text{Last}(l) \neq \text{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



Jarvis march

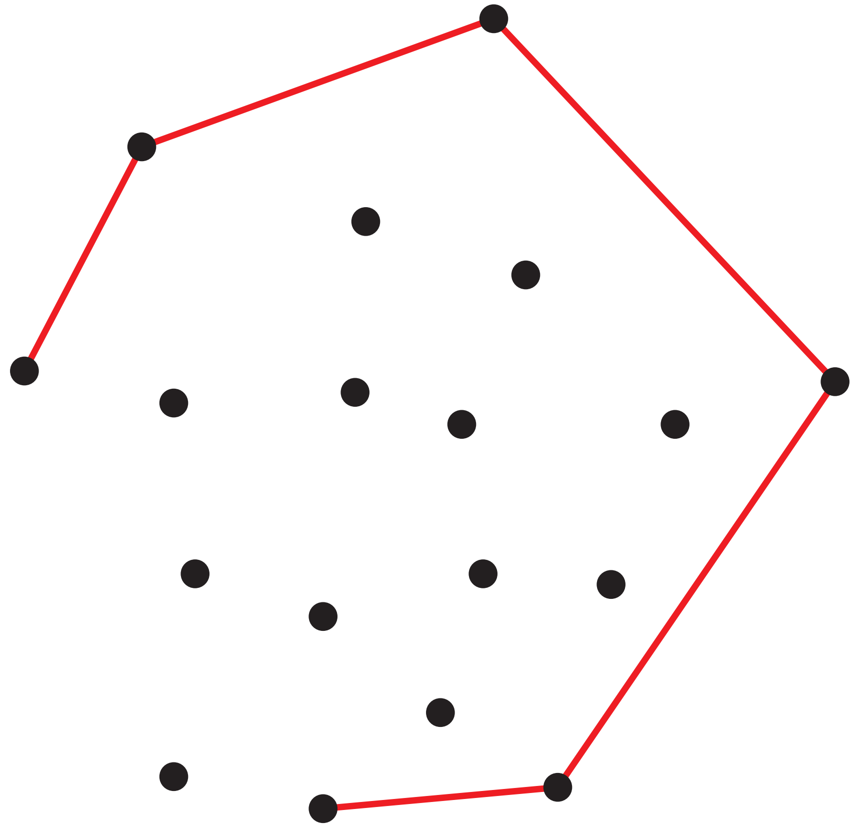
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



CONVEX HULL IN 2D

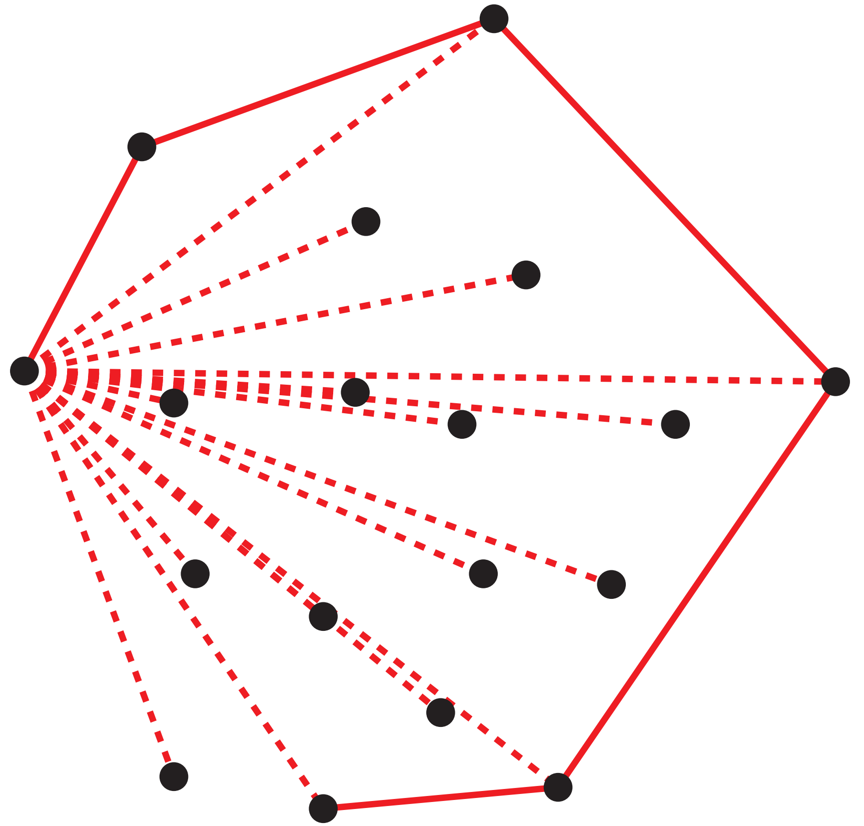
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \text{Last}(l) \neq \text{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



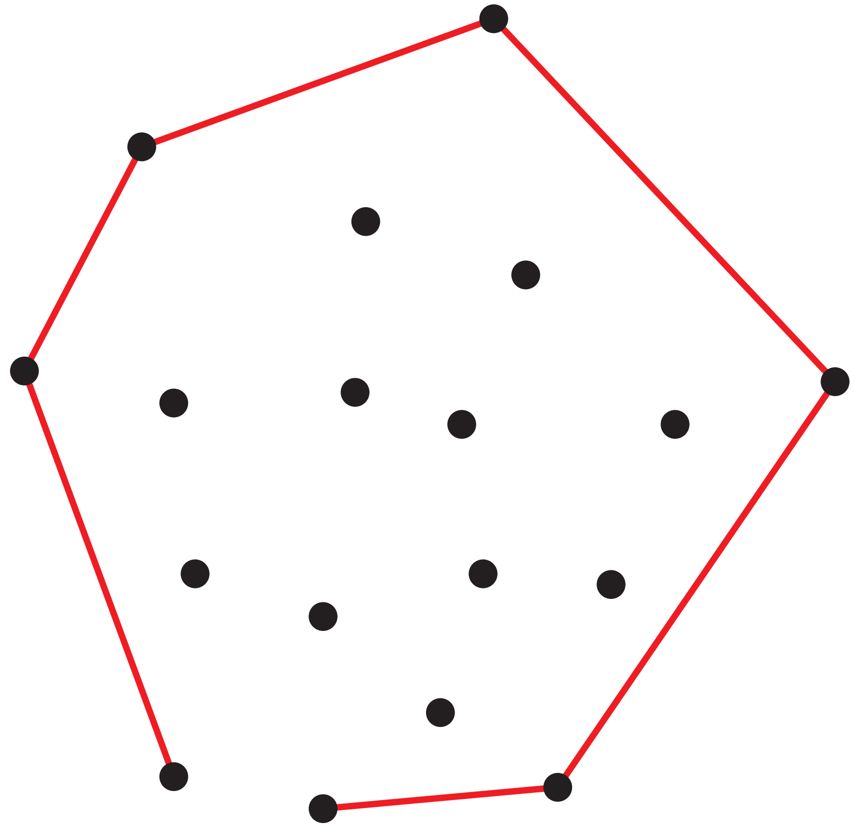
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



Jarvis march

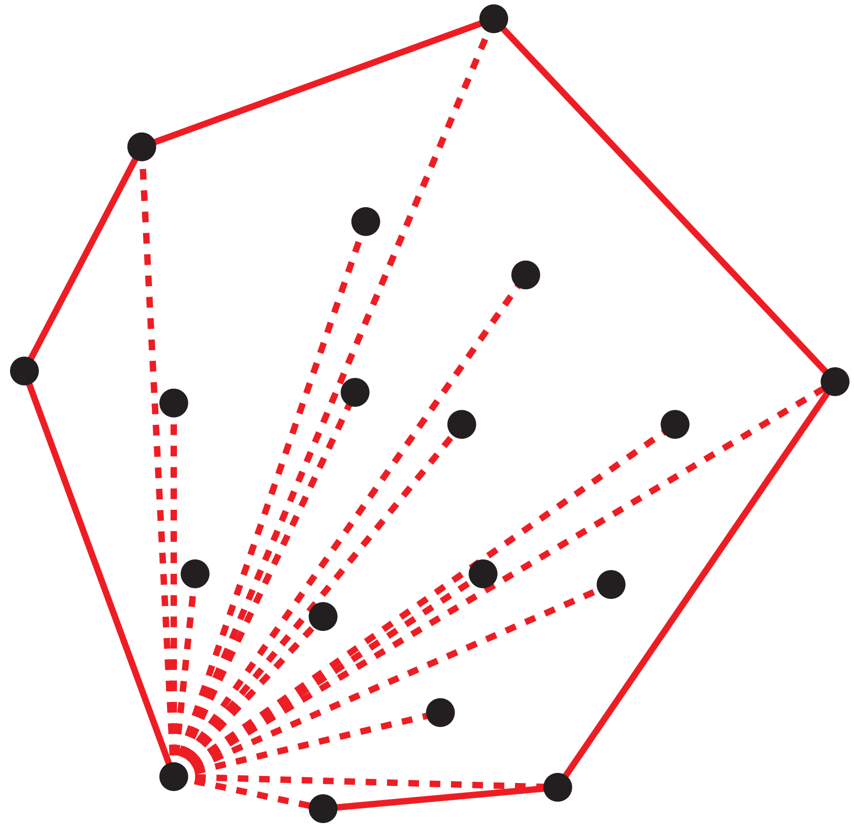
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \text{Last}(l) \neq \text{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



CONVEX HULL IN 2D

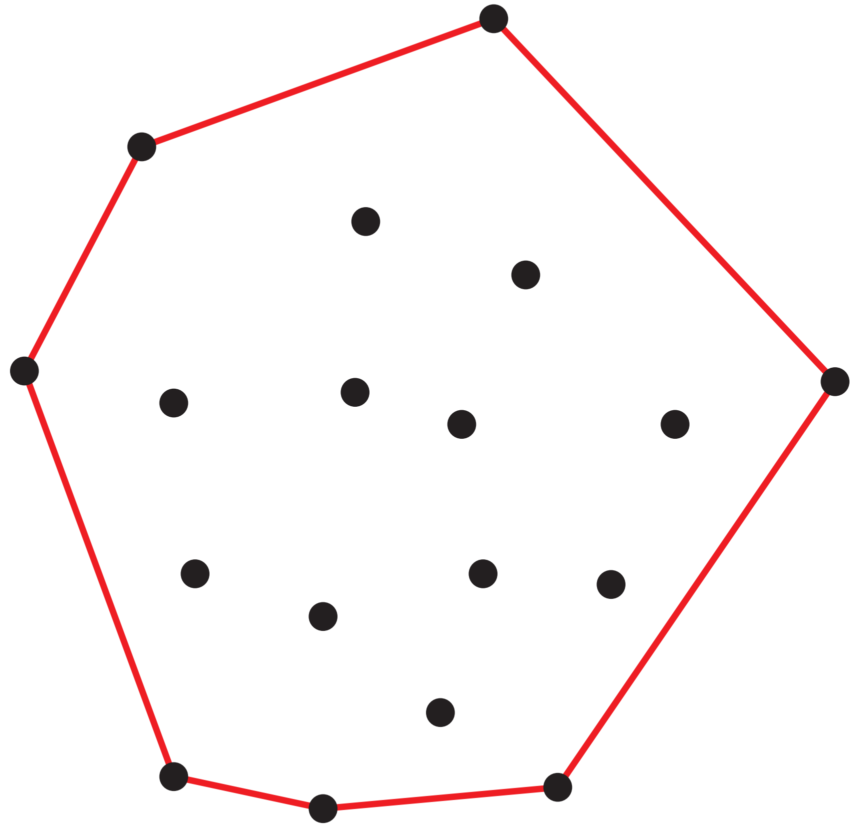
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



Jarvis march

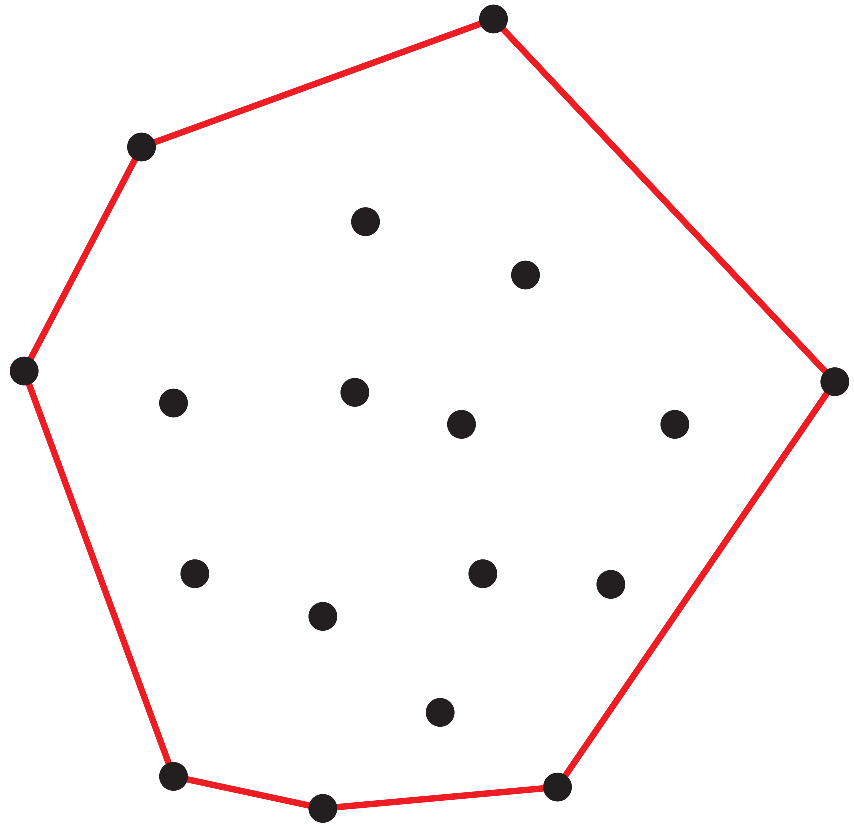
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \mathrm{Last}(l) \neq \mathrm{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



CONVEX HULL IN 2D

Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. While $v = \text{Last}(l) \neq \text{First}(l)$, do:
   (a) Detect the angularly rightmost point $p_j \in P$ with respect to v.
   (b) Add p_j to l
3. Return l



Time cost: $\Theta(hn) = O(n^2)$

QuickHull algorithm (by prune-and-search)

QuickHull algorithm (by prune-and-search)

Initialization

1. Find the extreme points in the horizontal and vertical directions.

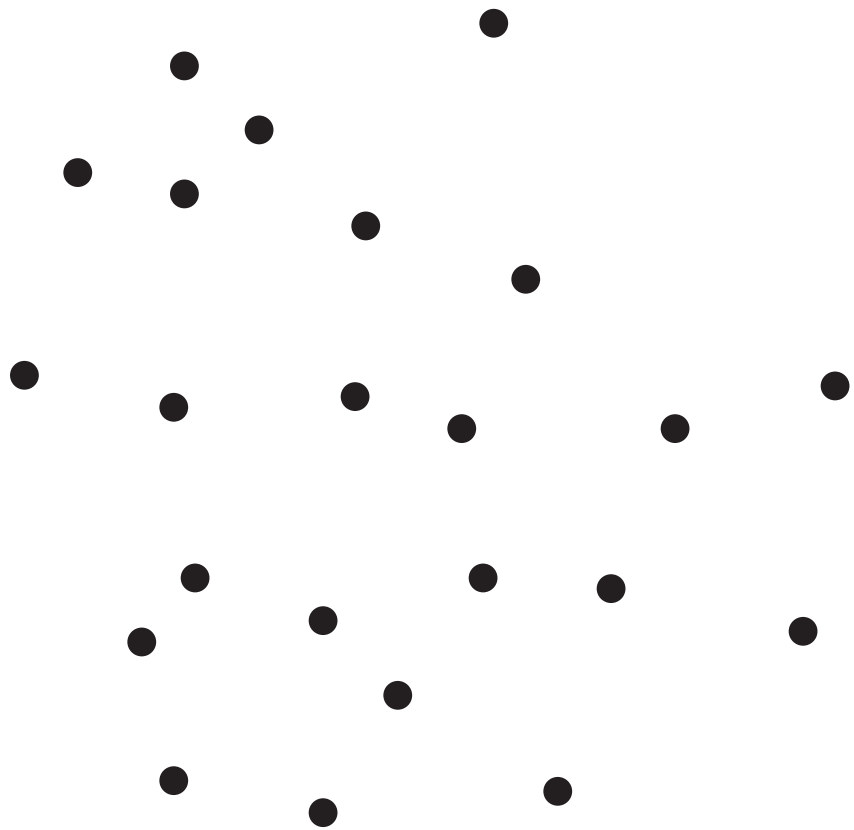
2. Compute the convex hull of these (between 2 and 8) points.

3. Test all the remaining points, and classify them according to their position (NE, SE, SW, NW) or eliminate them if they lie in the interior.

QuickHull algorithm (by prune-and-search)

Initialization

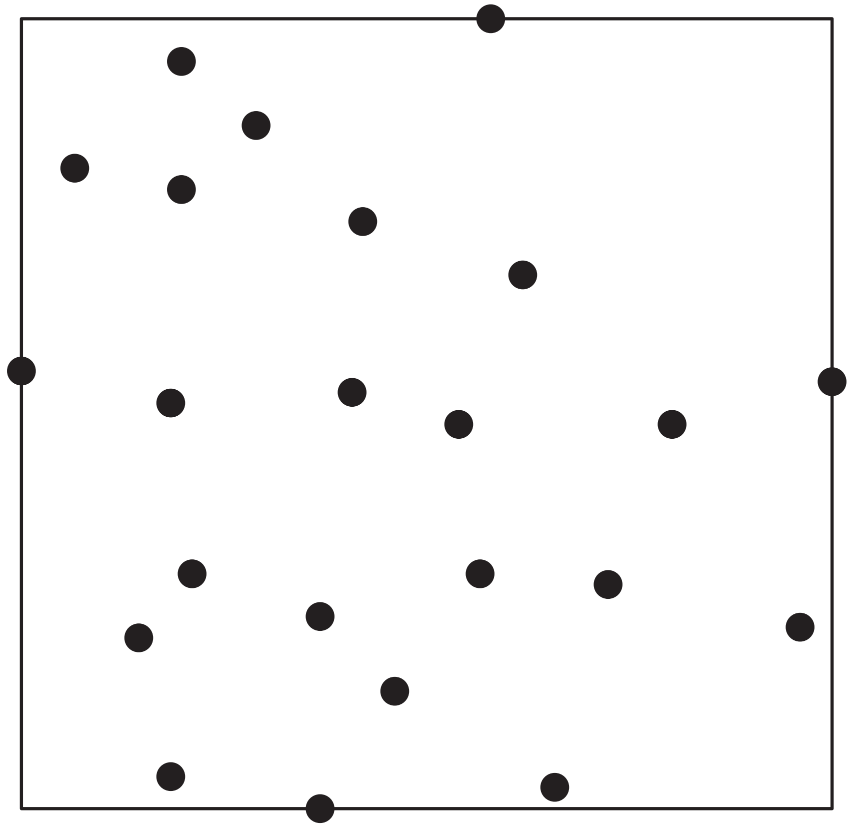
1. Find the extreme points in the horizontal and vertical directions.
2. Compute the convex hull of these (between 2 and 8) points.
3. Test all the remaining points, and classify them according to their position (NE, SE, SW, NW) or eliminate them if they lie in the interior.



QuickHull algorithm (by prune-and-search)

Initialization

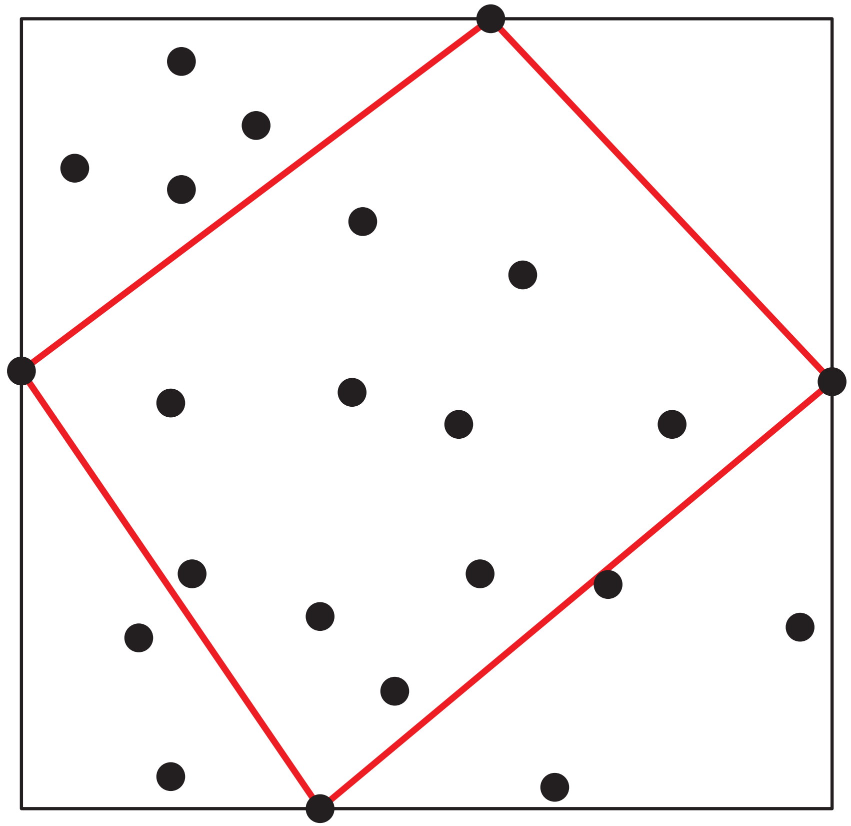
1. Find the extreme points in the horizontal and vertical directions.
2. Compute the convex hull of these (between 2 and 8) points.
3. Test all the remaining points, and classify them according to their position (NE, SE, SW, NW) or eliminate them if they lie in the interior.



QuickHull algorithm (by prune-and-search)

Initialization

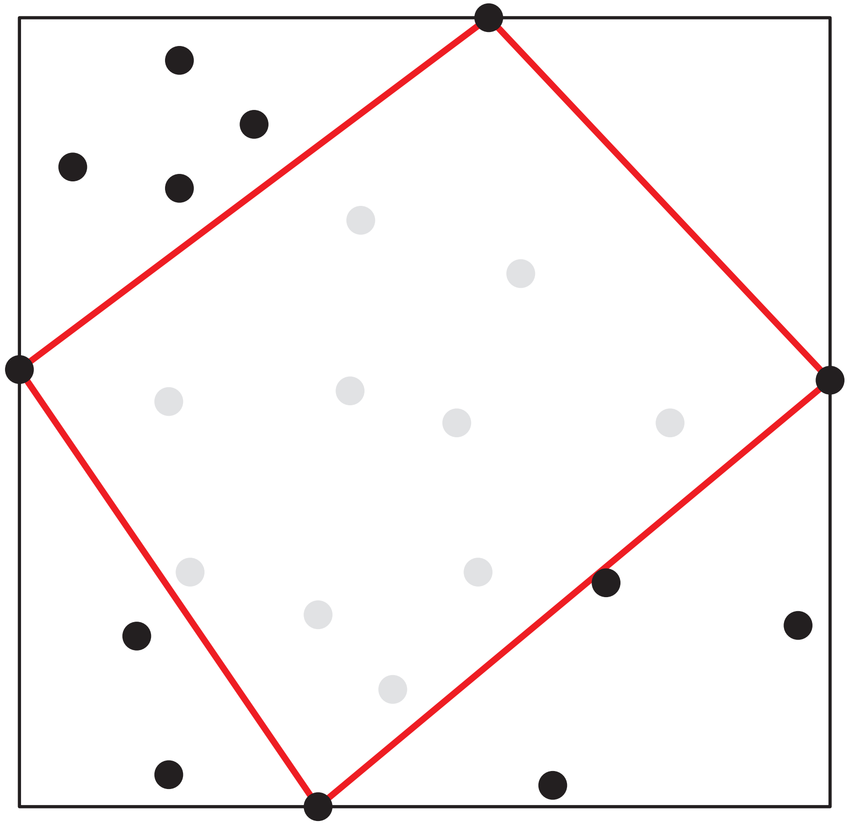
1. Find the extreme points in the horizontal and vertical directions.
2. Compute the convex hull of these (between 2 and 8) points.
3. Test all the remaining points, and classify them according to their position (NE, SE, SW, NW) or eliminate them if they lie in the interior.



QuickHull algorithm (by prune-and-search)

Initialization

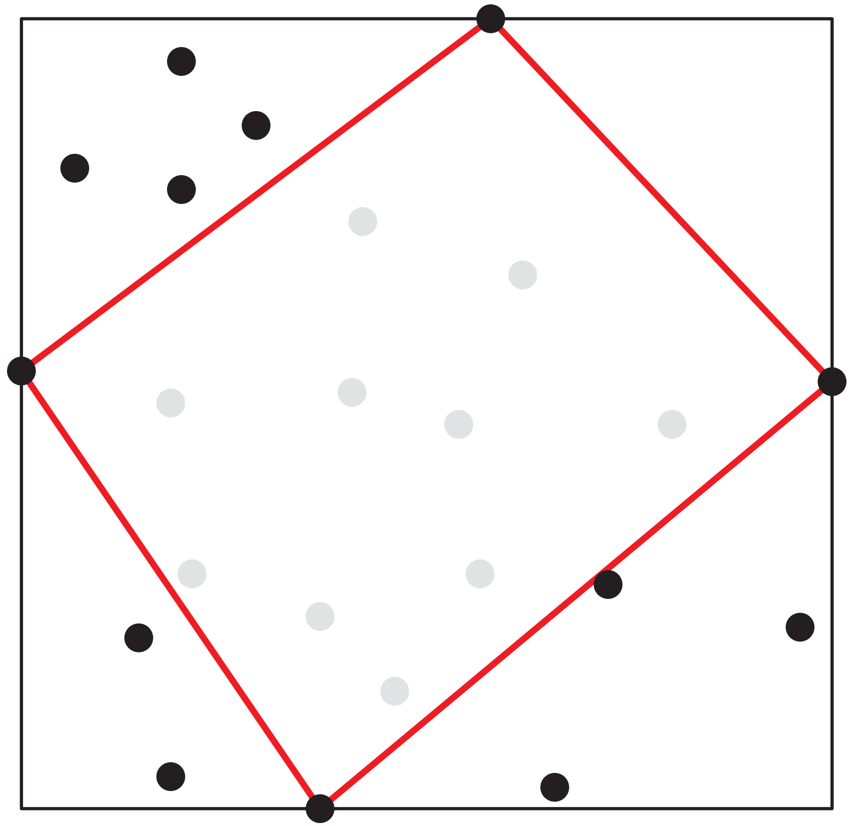
1. Find the extreme points in the horizontal and vertical directions.
2. Compute the convex hull of these (between 2 and 8) points.
3. Test all the remaining points, and classify them according to their position (NE, SE, SW, NW) or eliminate them if they lie in the interior.



QuickHull algorithm (by prune-and-search)

Initialization

1. Find the extreme points in the horizontal and vertical directions.
2. Compute the convex hull of these (between 2 and 8) points.
3. Test all the remaining points, and classify them according to their position (NE, SE, SW, NW) or eliminate them if they lie in the interior.



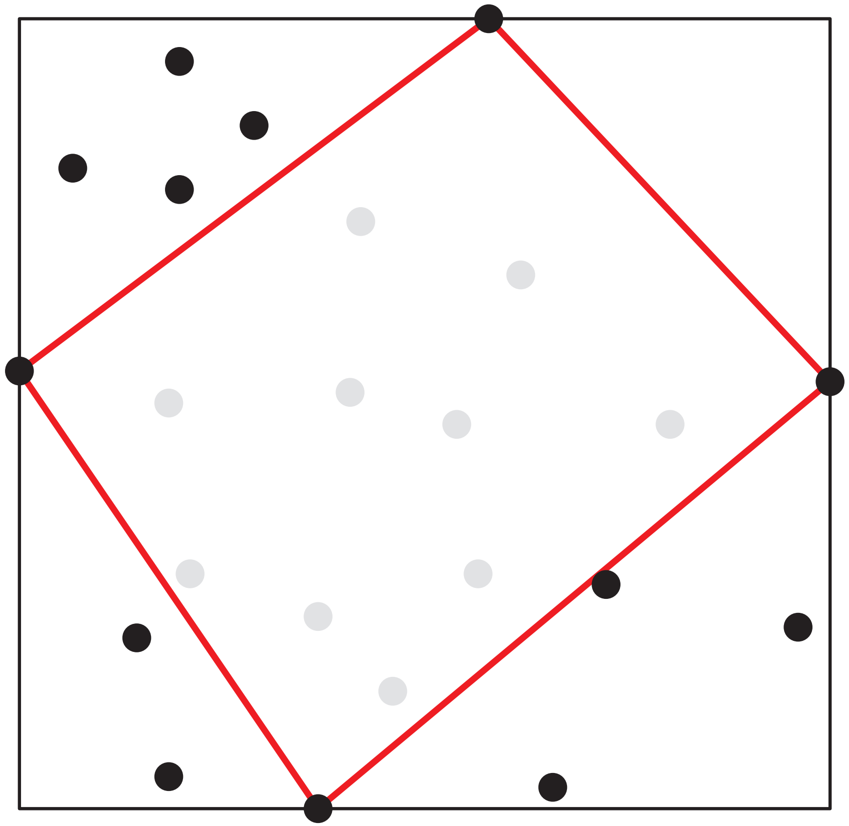
Running time of this step: $O(n)$

QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.

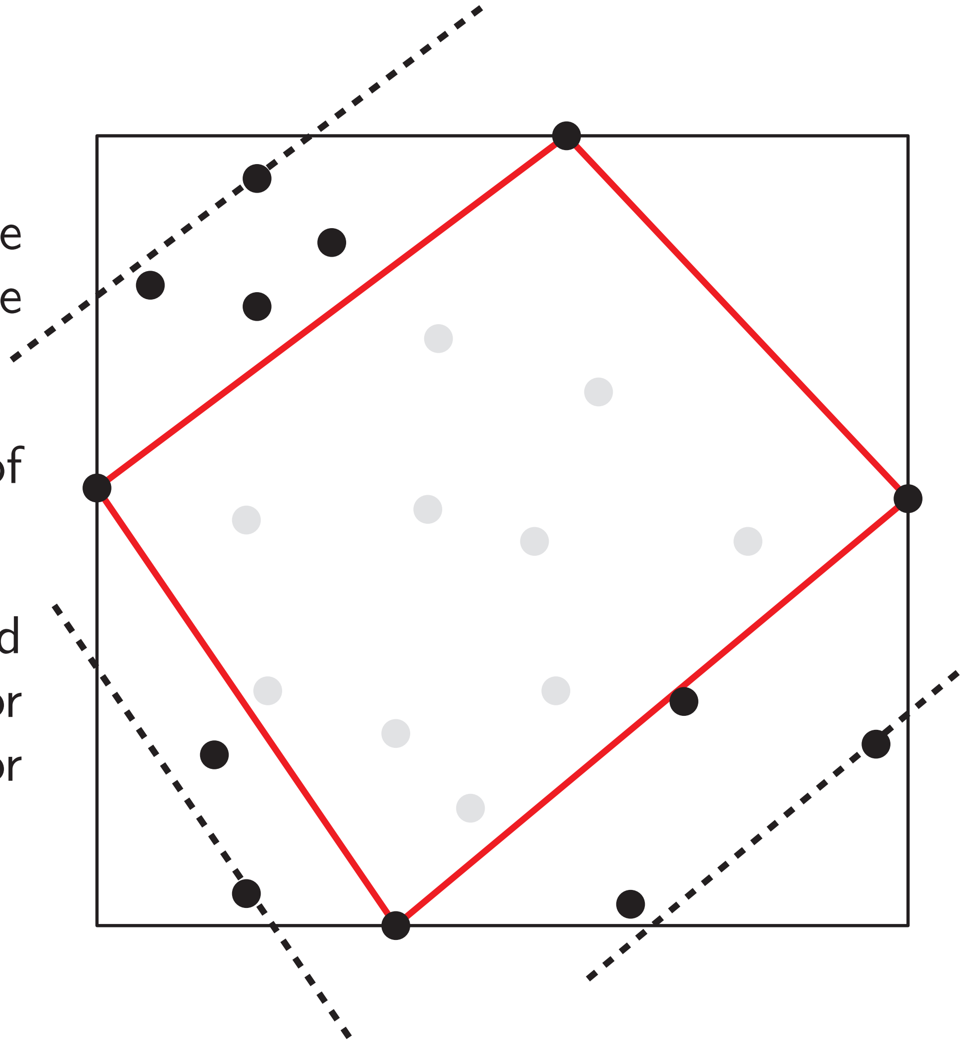


QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.



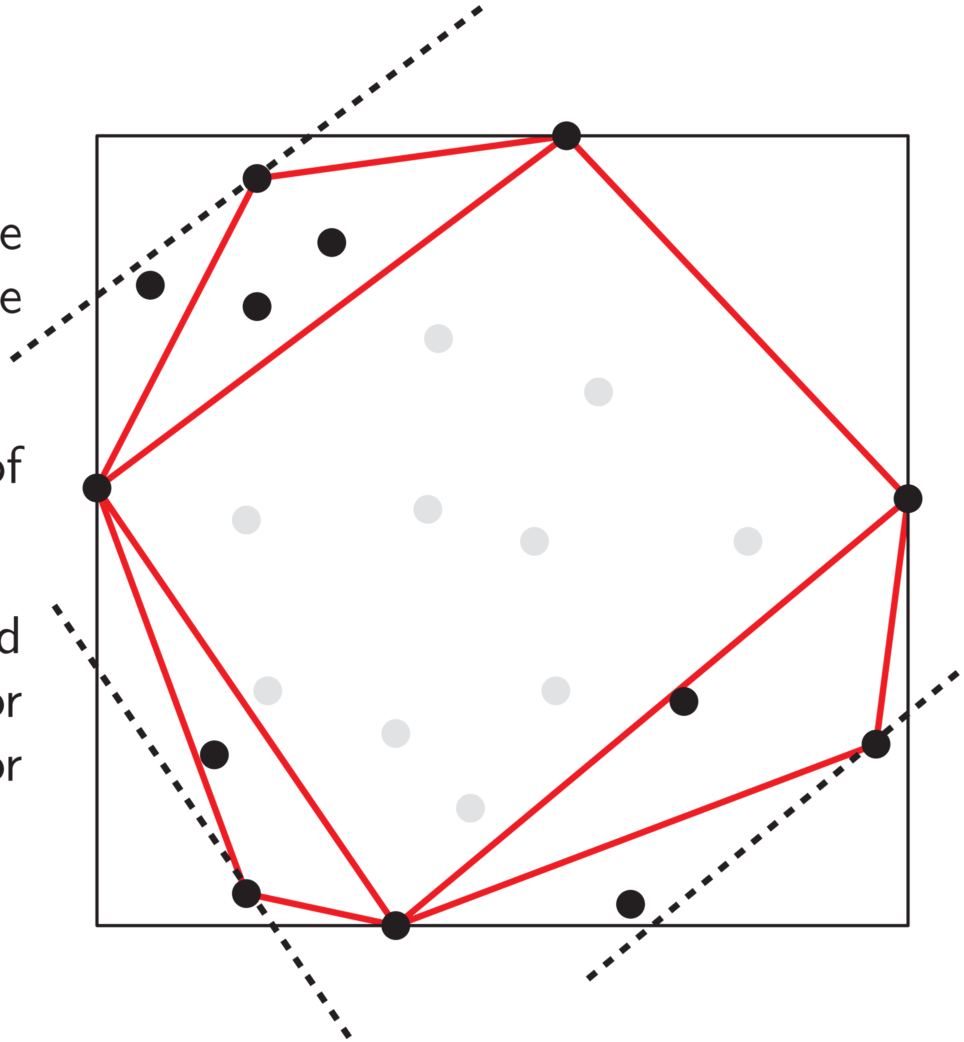
CONVEX HULL IN 2D

QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.

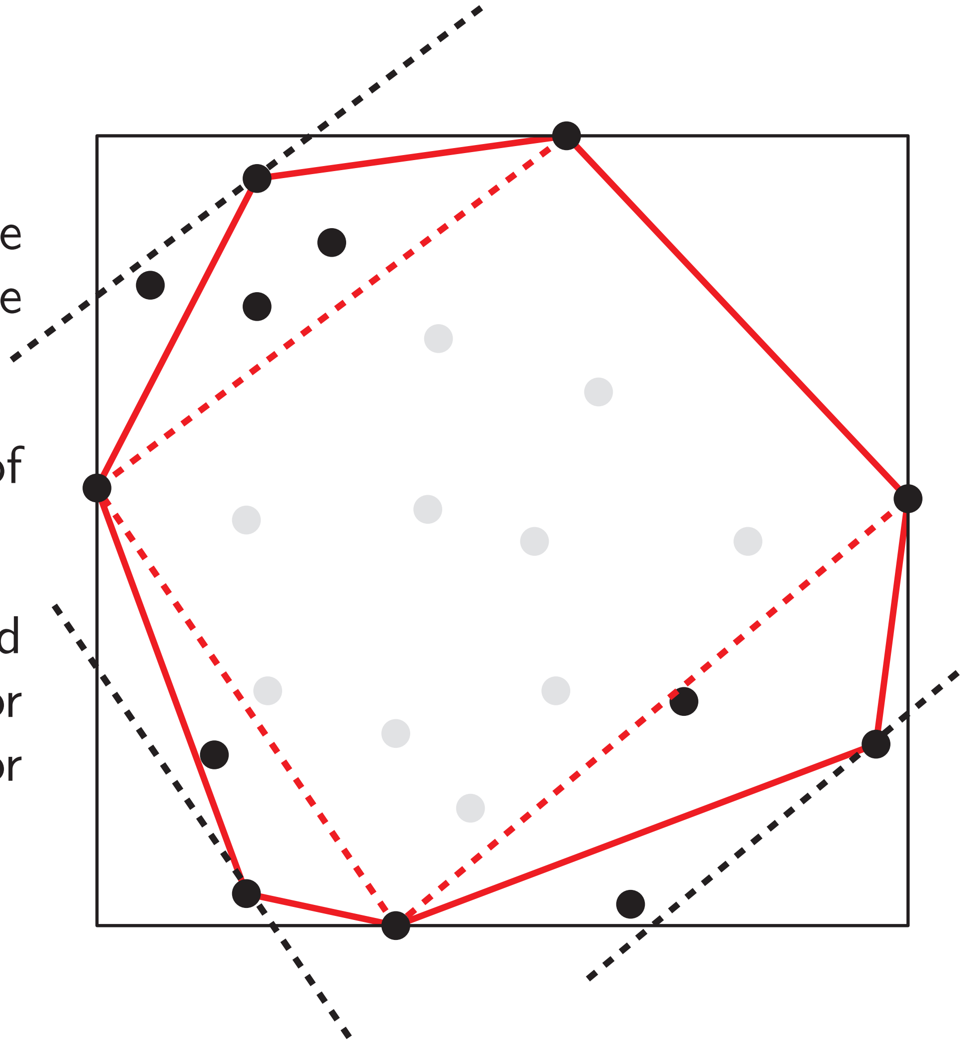


QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.

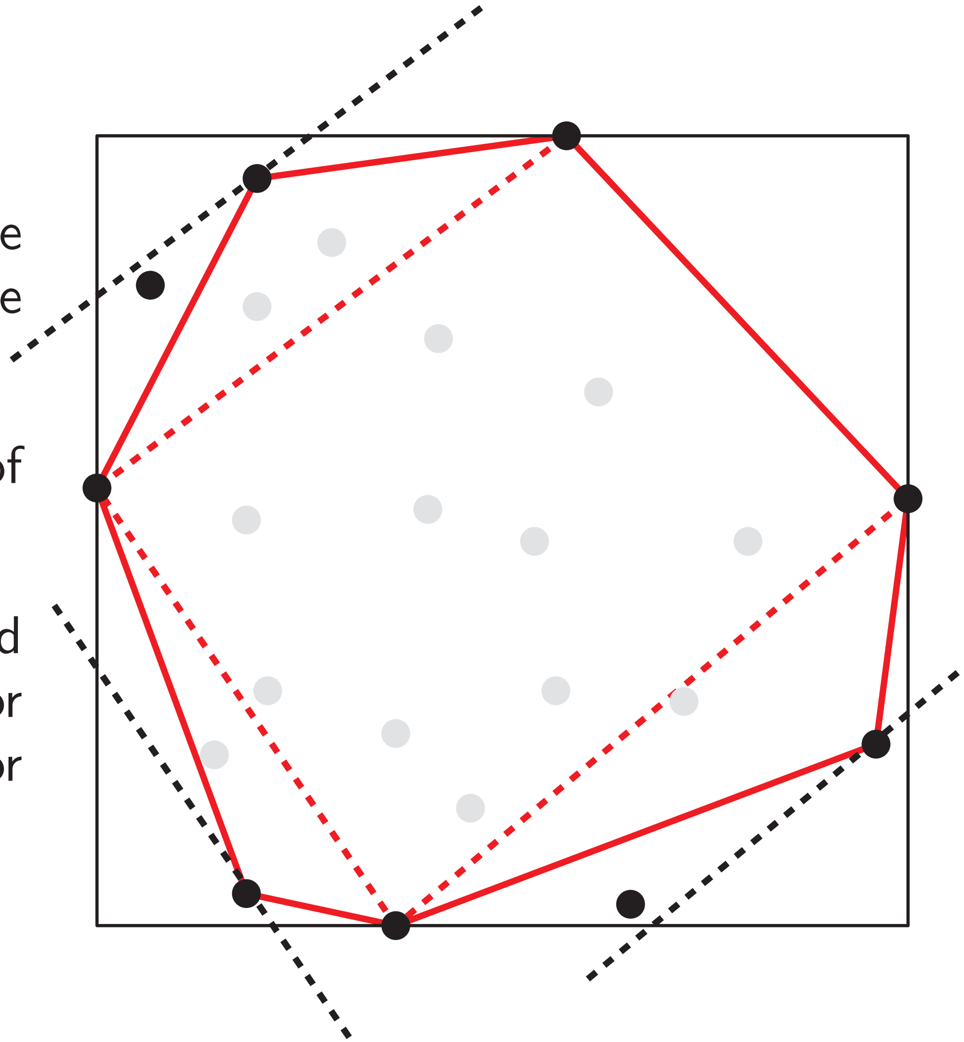


QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.



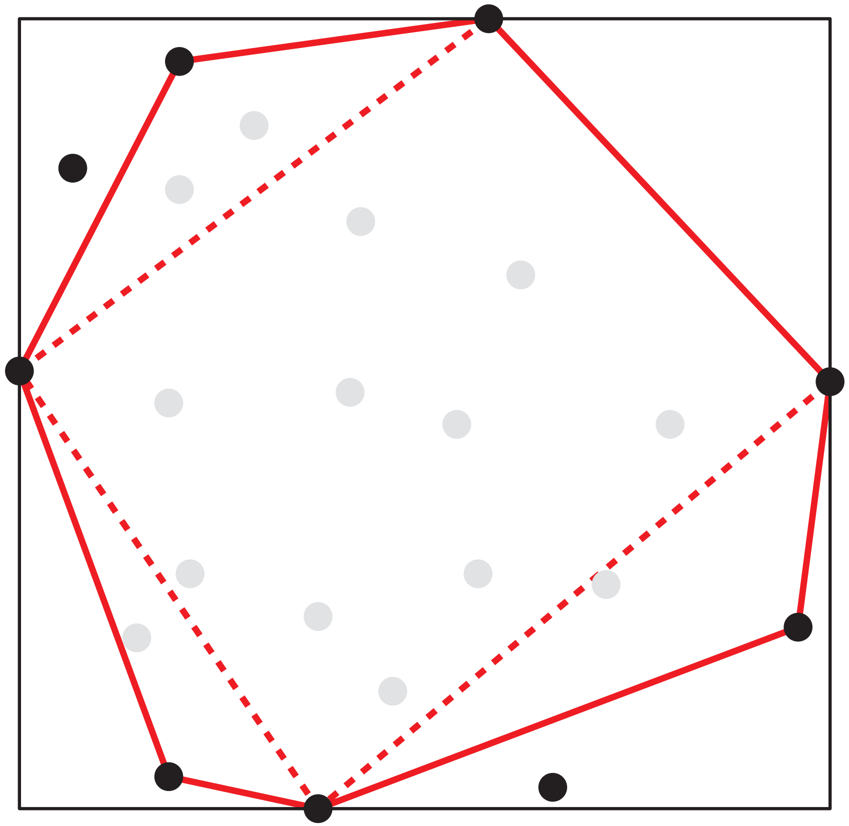
CONVEX HULL IN 2D

QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.



QuickHull algorithm (by prune-and-search)

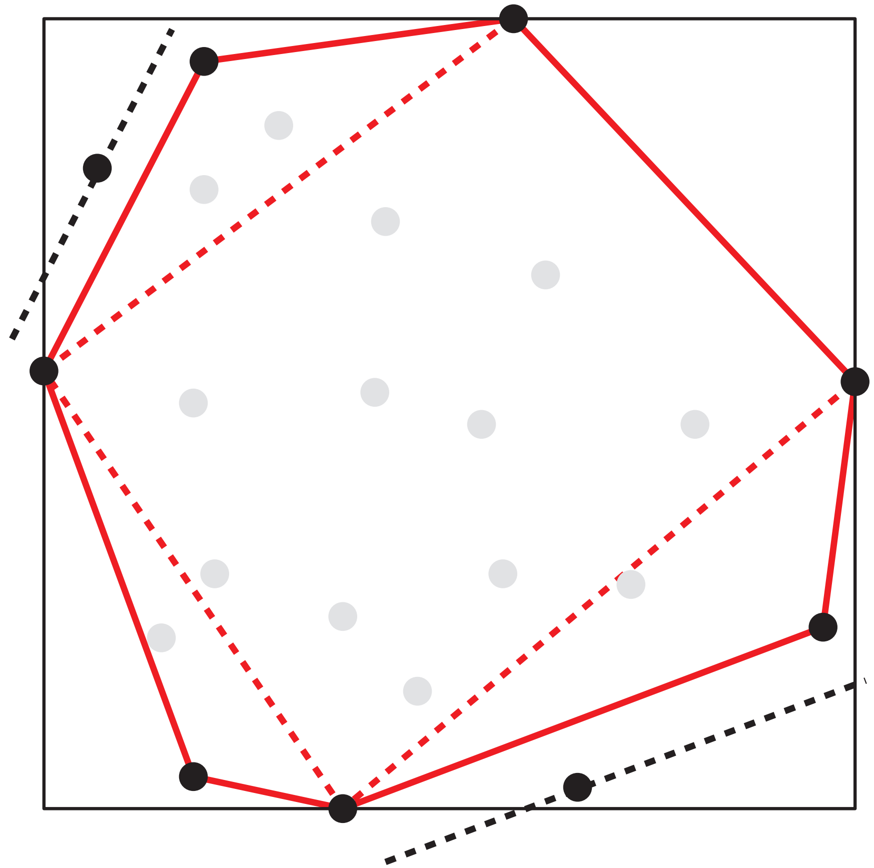
Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.

2. Connect the extreme point with te endpoints of the edge, and update the convex hull.

3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.



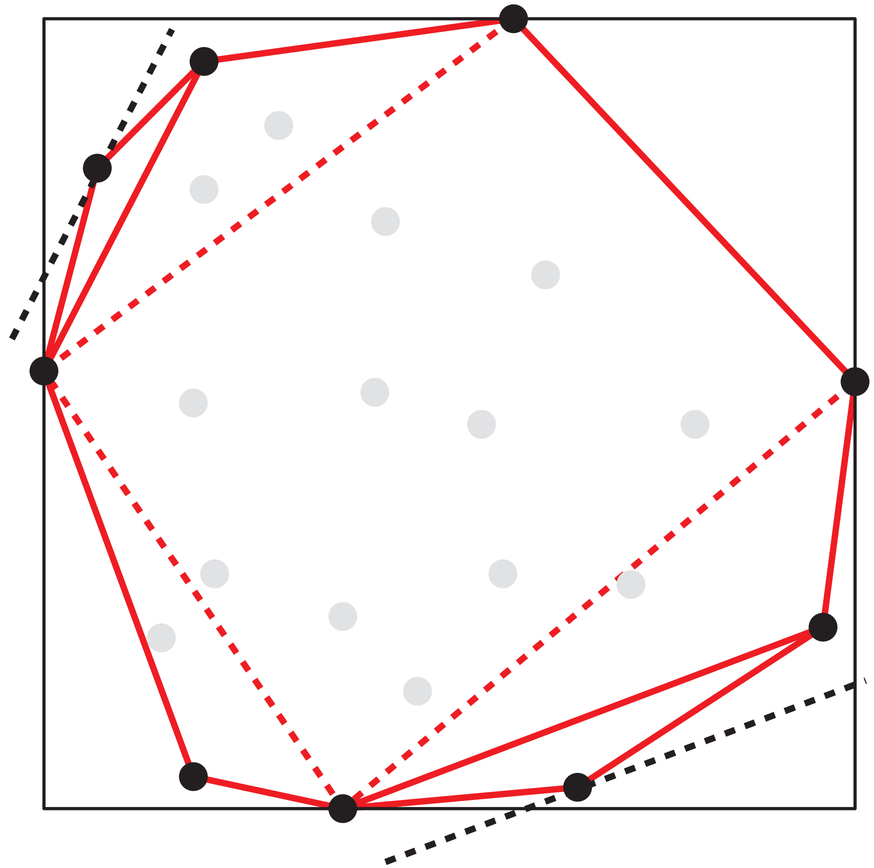
CONVEX HULL IN 2D

QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.

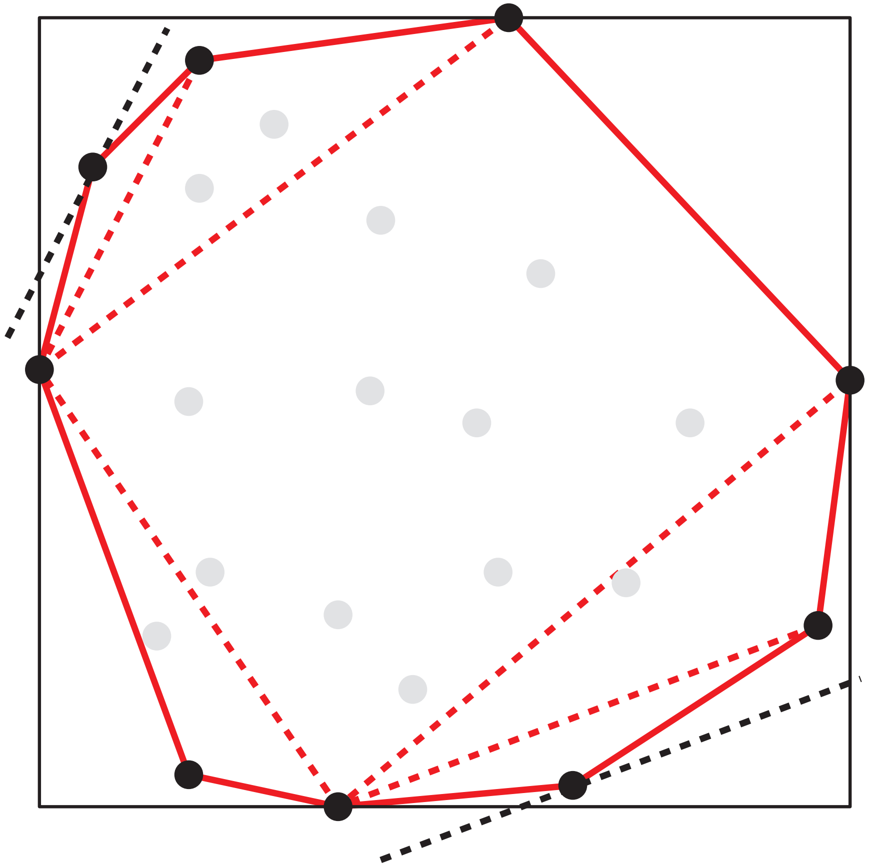


QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.



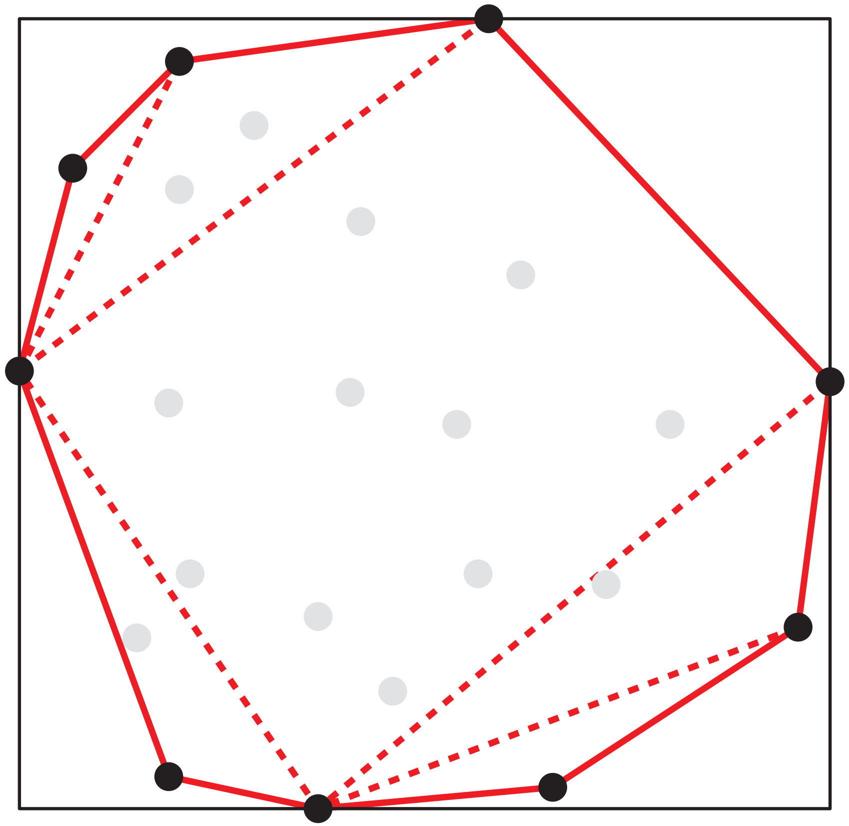
CONVEX HULL IN 2D

QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.



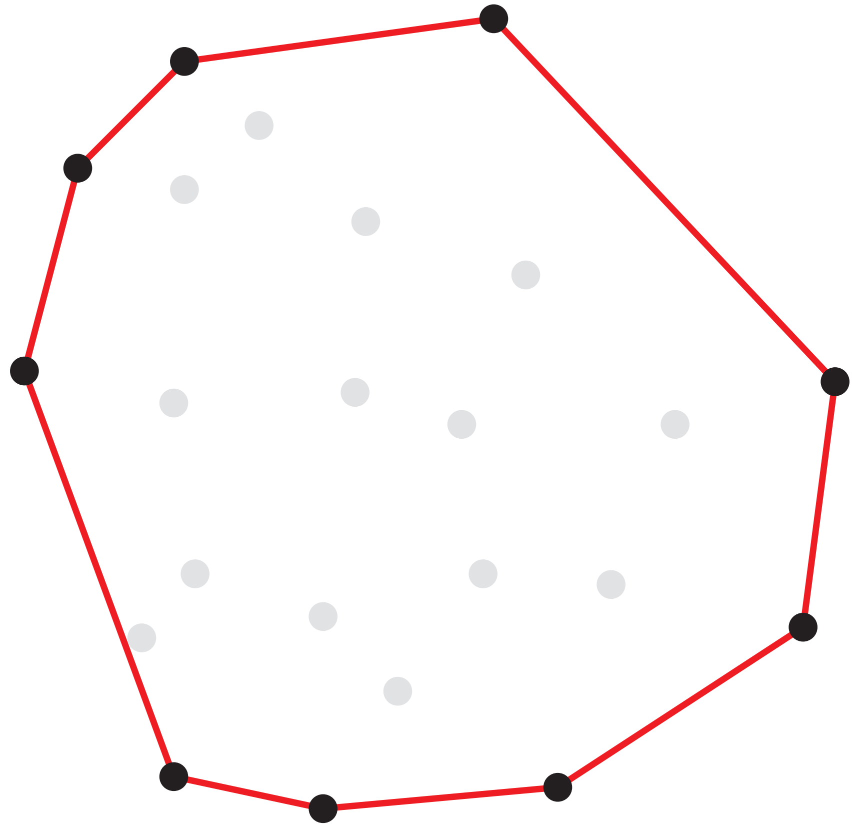
CONVEX HULL IN 2D

QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.

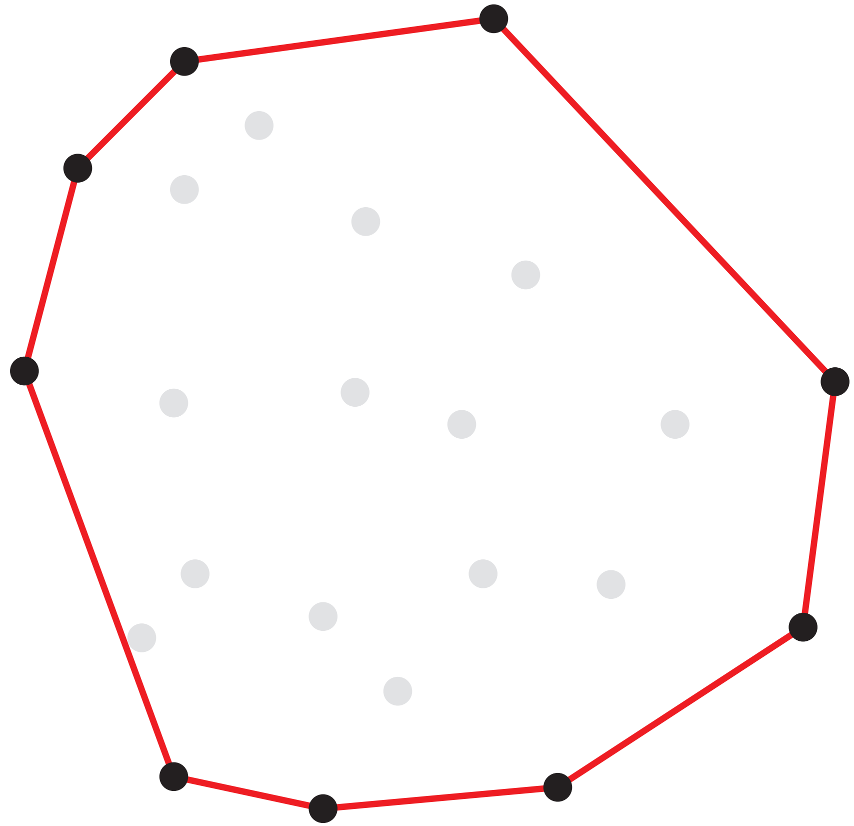


QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

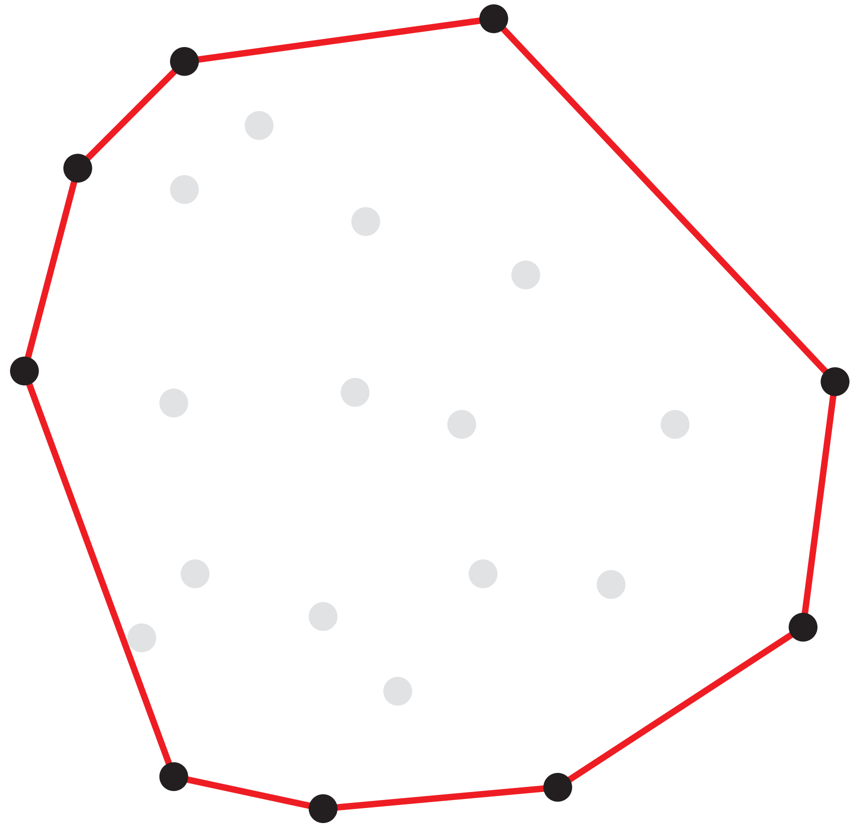
1. Among all points lying in each region, find the extreme point in the direction orthogonal to the edge that determines the region.
2. Connect the extreme point with te endpoints of the edge, and update the convex hull.
3. Test all the remaining points of each region, and classify them according to their position (left or right) or eliminate them if they lie in the interior of the newly created triangle.



Running time of this step: $O(n^2)$

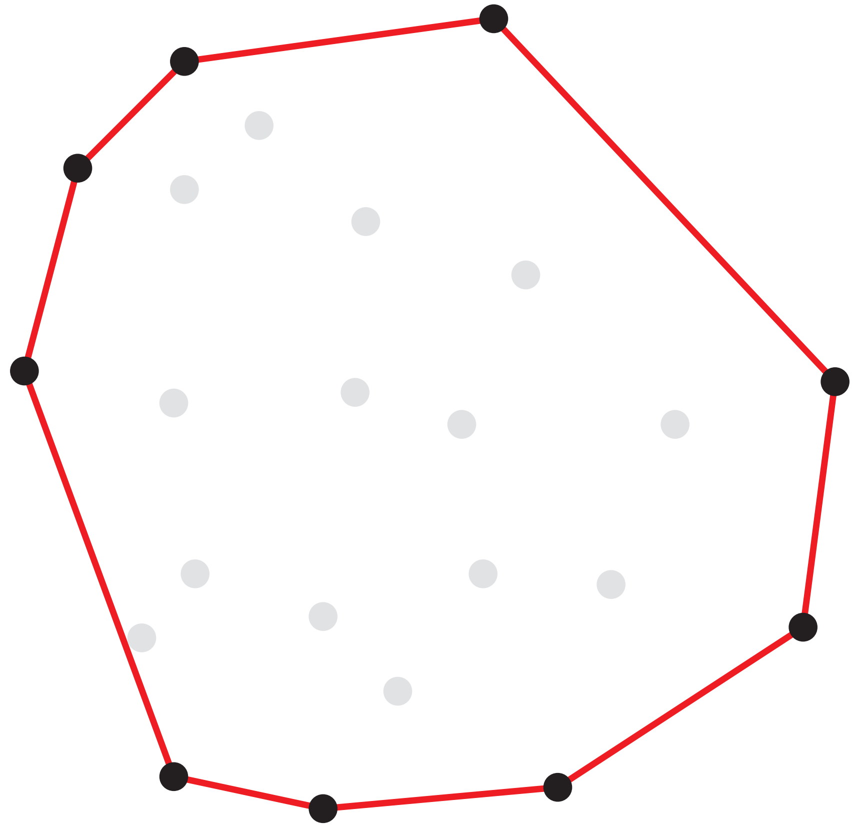
QuickHull algorithm (by prune-and-search)

Overall running time: $O(n^2)$



QuickHull algorithm (by prune-and-search)

Overall running time: $O(n^2)$



Nevertheless, the running time of this algorithm depends on the position of the input points.
For example:

- If the input points are in convex position, the running time is $\Theta(n^2)$.
- If the points are such that each prune step eliminates half of the current points, then the algorithm runs in $\Theta(n \log n)$ time.
- If the convex hull is triangular, the algorithm runs in $\Theta(n)$ time.

Graham's algorithm

Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

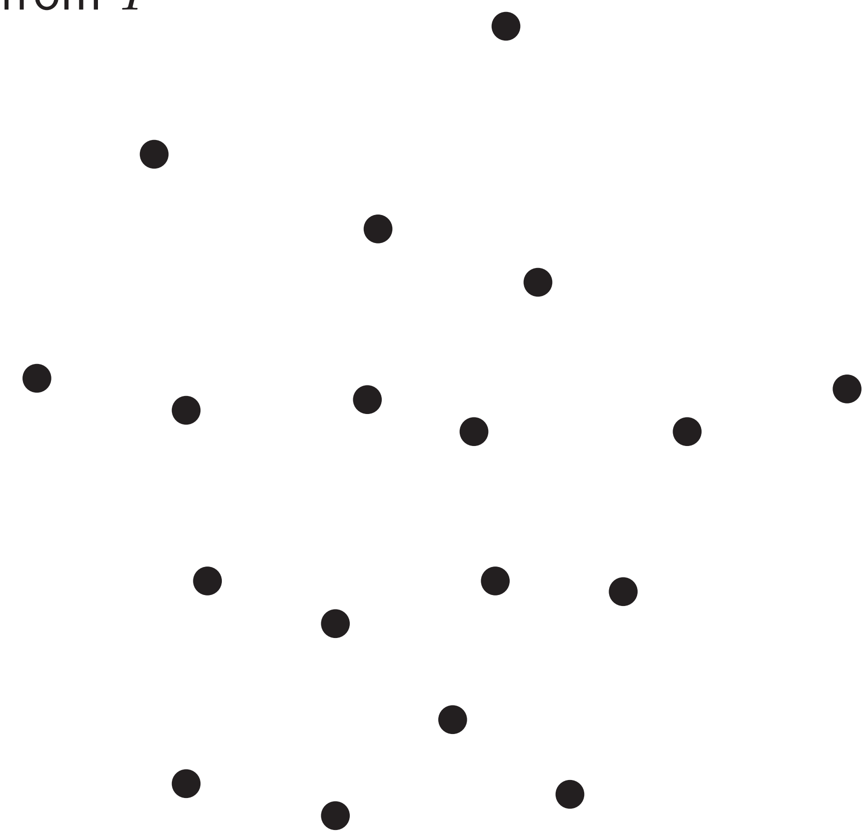
Return l

Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance
While there exist points $p_i \in P$ to be explored, do:
$p = \mathrm{top}(l)$
$p^- = \mathrm{previous}(\mathrm{top}(l))$
- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

Return l

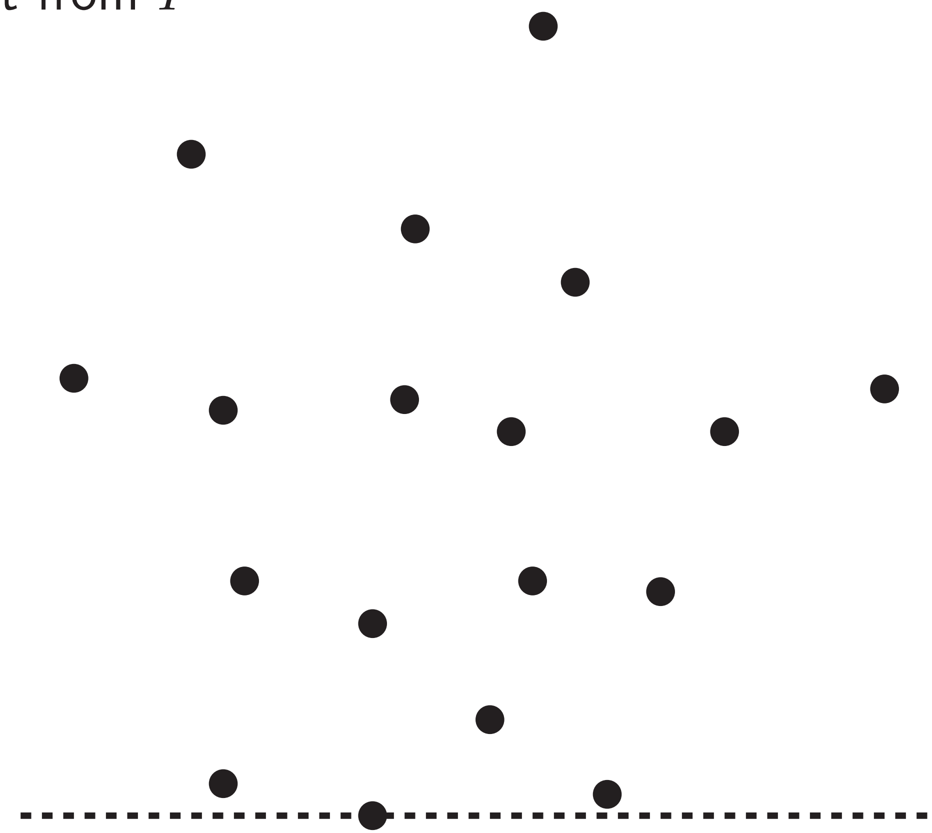


Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance
While there exist points $p_i \in P$ to be explored, do:
$p = \mathrm{top}(l)$
$p^- = \mathrm{previous}(\mathrm{top}(l))$
- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

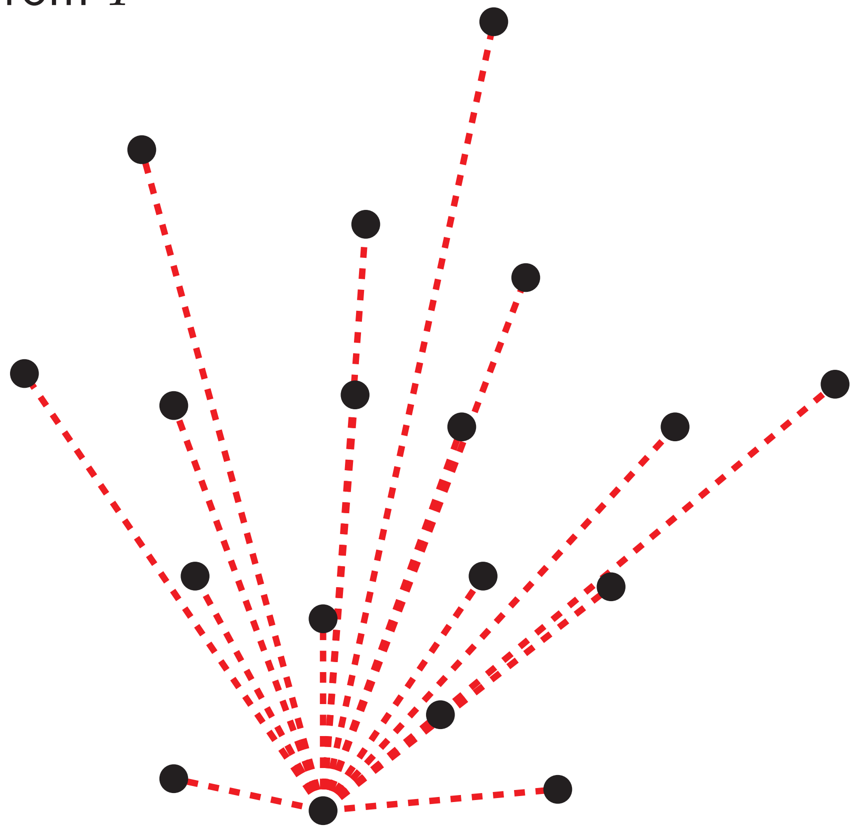
While there exist points $p_i \in P$ to be explored, do:

$p = \mathrm{top}(l)$

$p^- = \mathrm{previous}(\mathrm{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

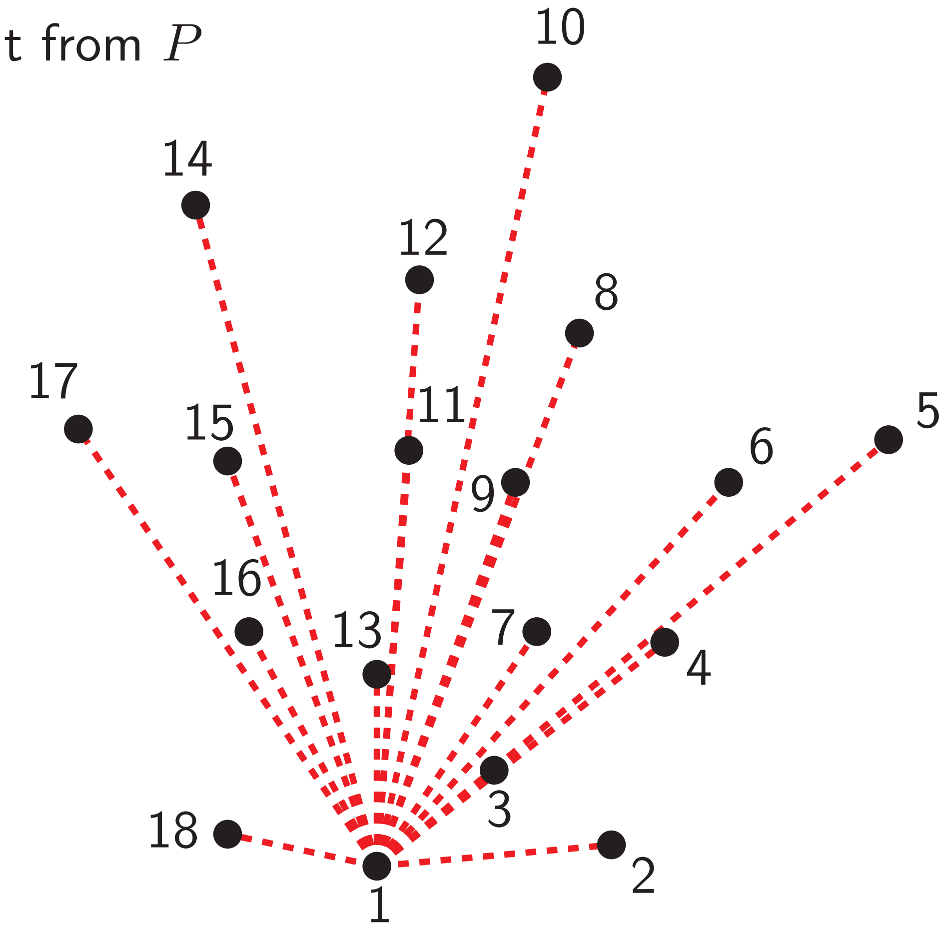
While there exist points $p_i \in P$ to be explored, do:

$p = \mathrm{top}(l)$

$p^- = \mathrm{previous}(\mathrm{top}(l))$

- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

Return l

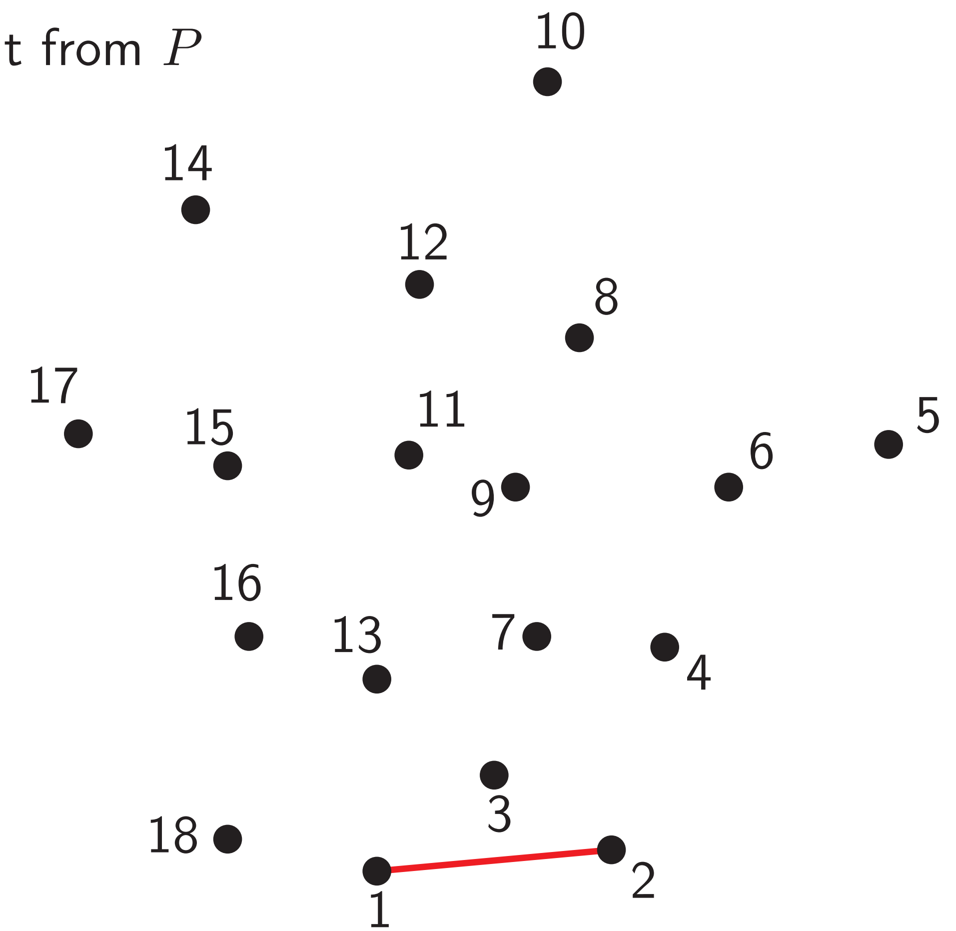


Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance
While there exist points $p_i \in P$ to be explored, do:
$p = \text{top}(l)$
$p^- = \text{previous}(\text{top}(l))$
- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

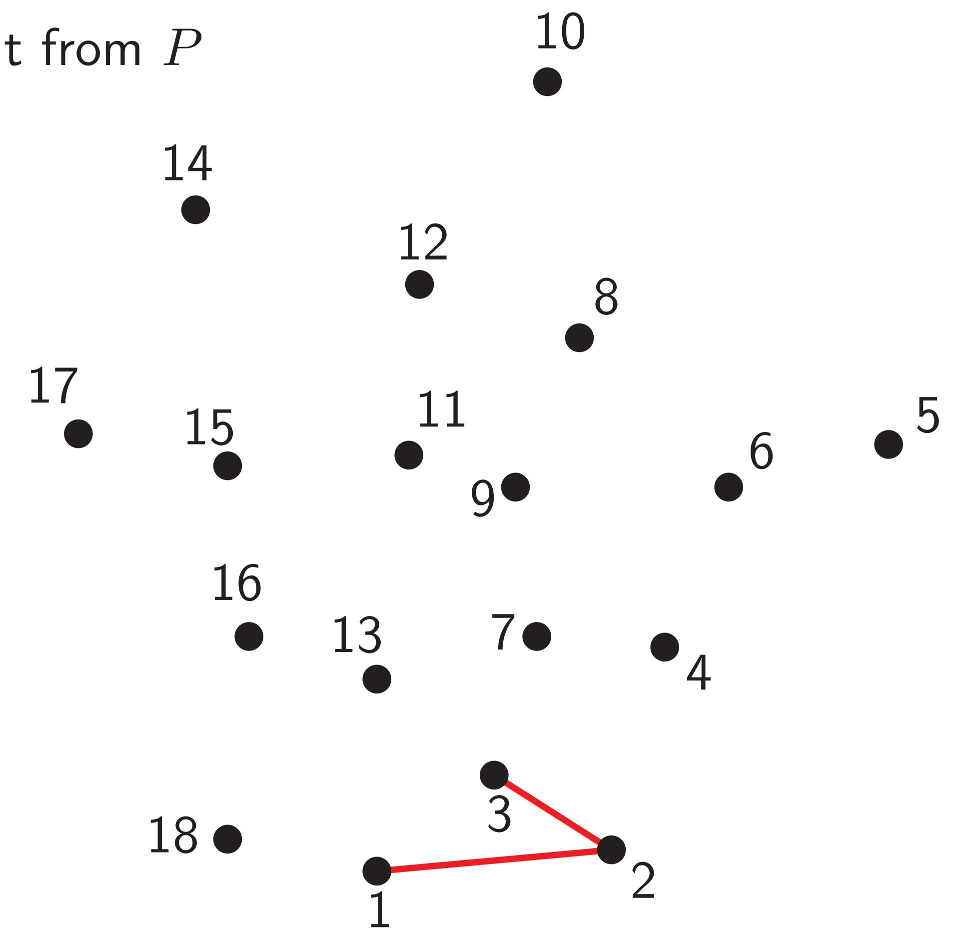
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l

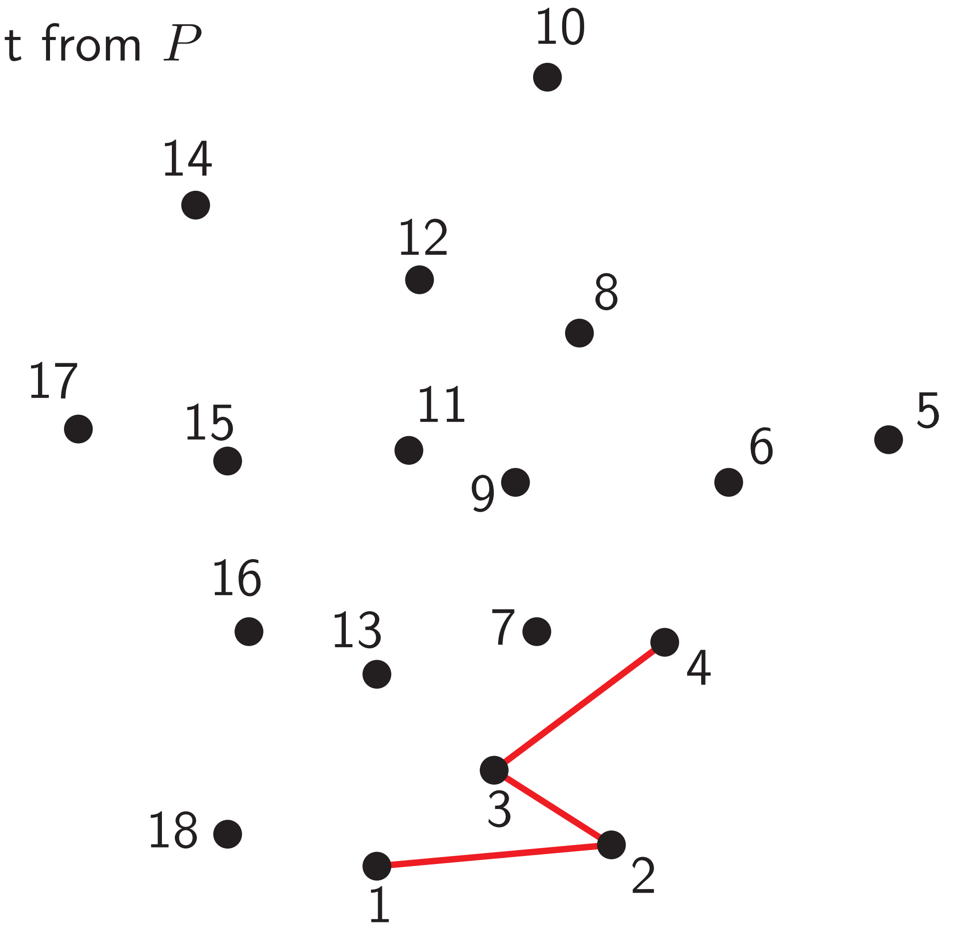


Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance
While there exist points $p_i \in P$ to be explored, do:
$p = \mathrm{top}(l)$
$p^- = \mathrm{previous}(\mathrm{top}(l))$
- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

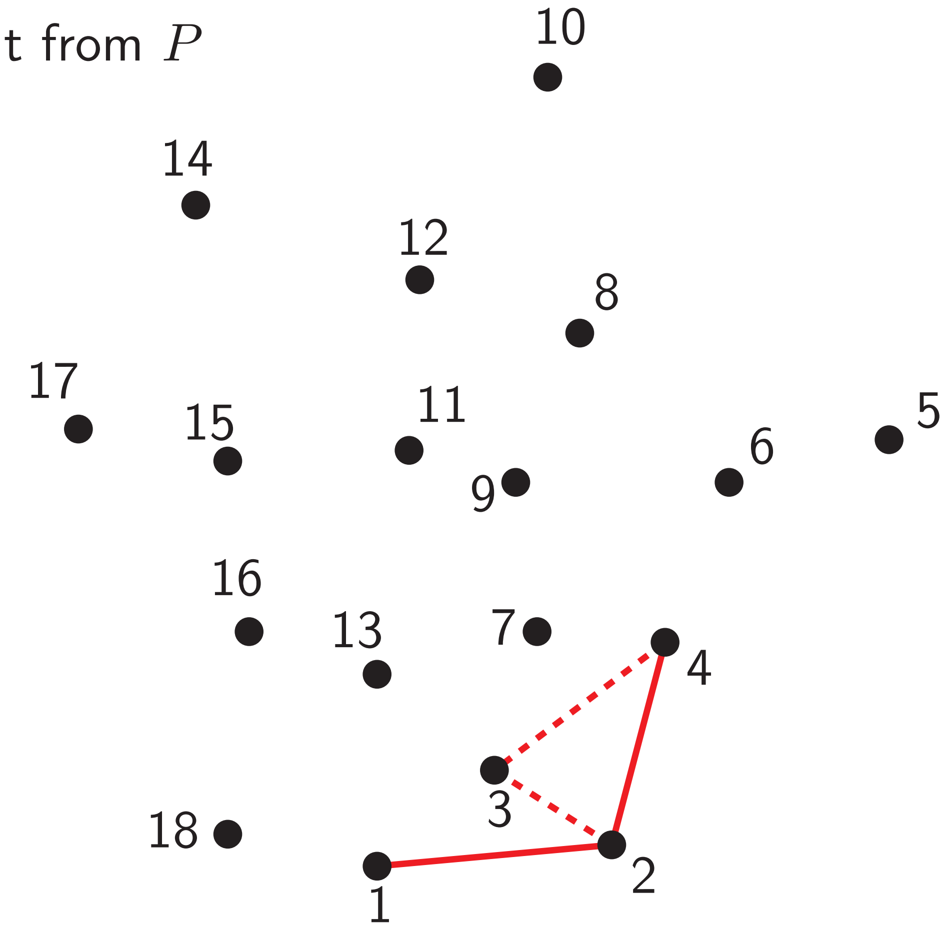
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

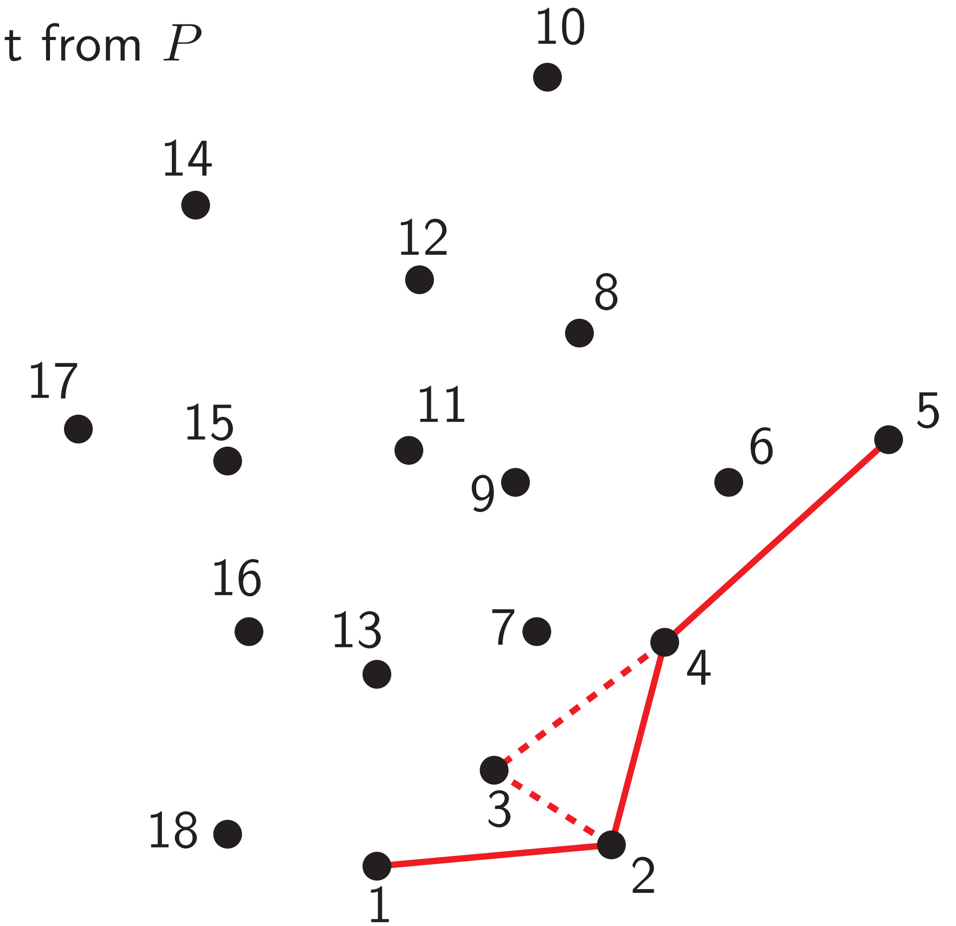
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

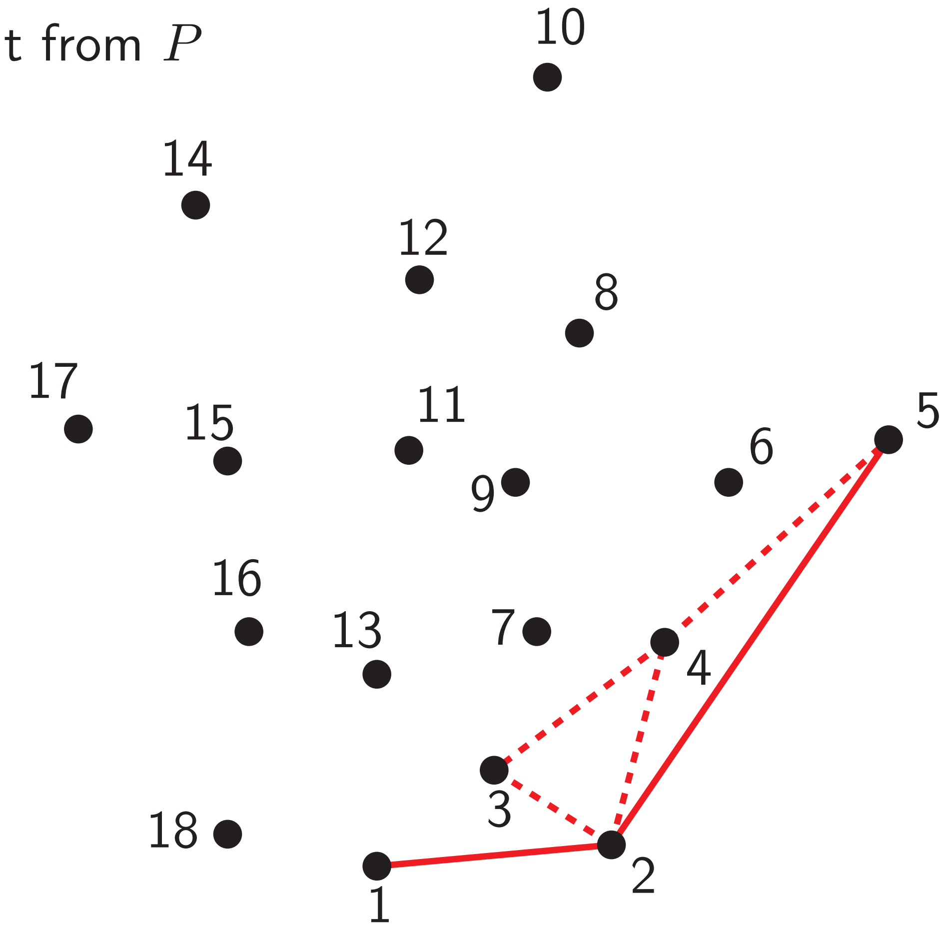
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

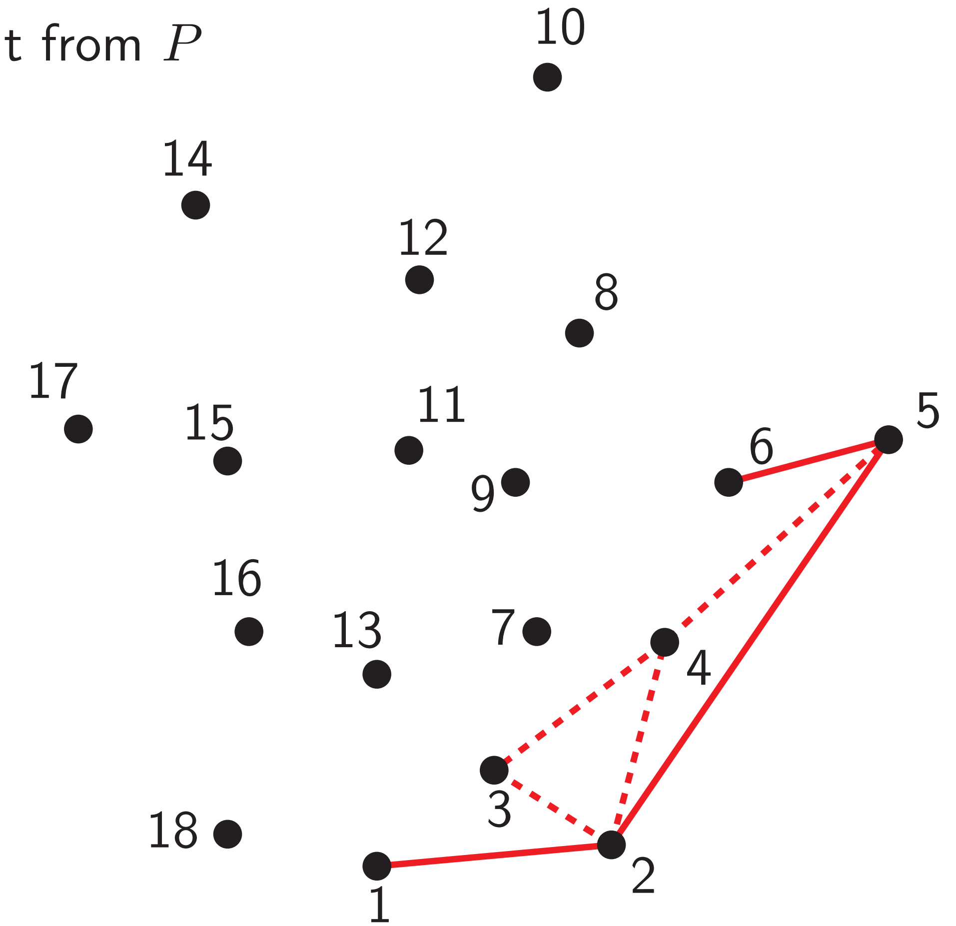
While there exist points $p_i \in P$ to be explored, do:

$p = \mathrm{top}(l)$

$p^- = \mathrm{previous}(\mathrm{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



CONVEX HULL IN 2D

Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

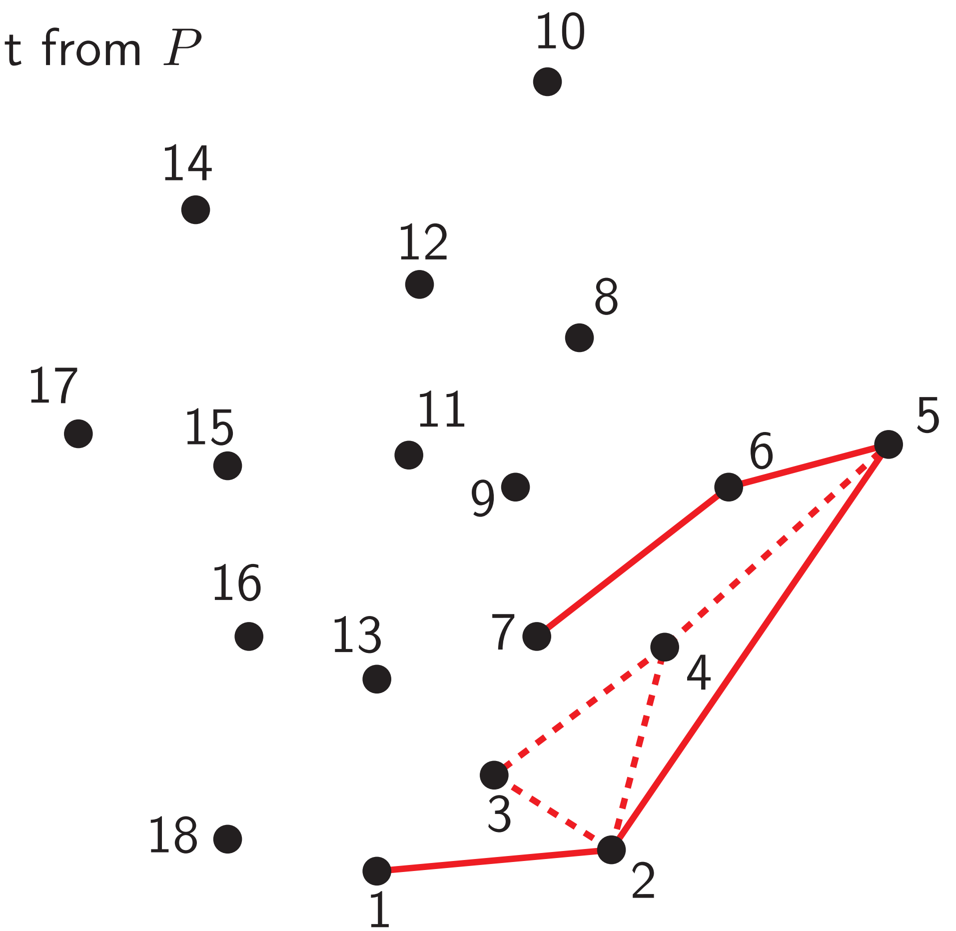
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



CONVEX HULL IN 2D

Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

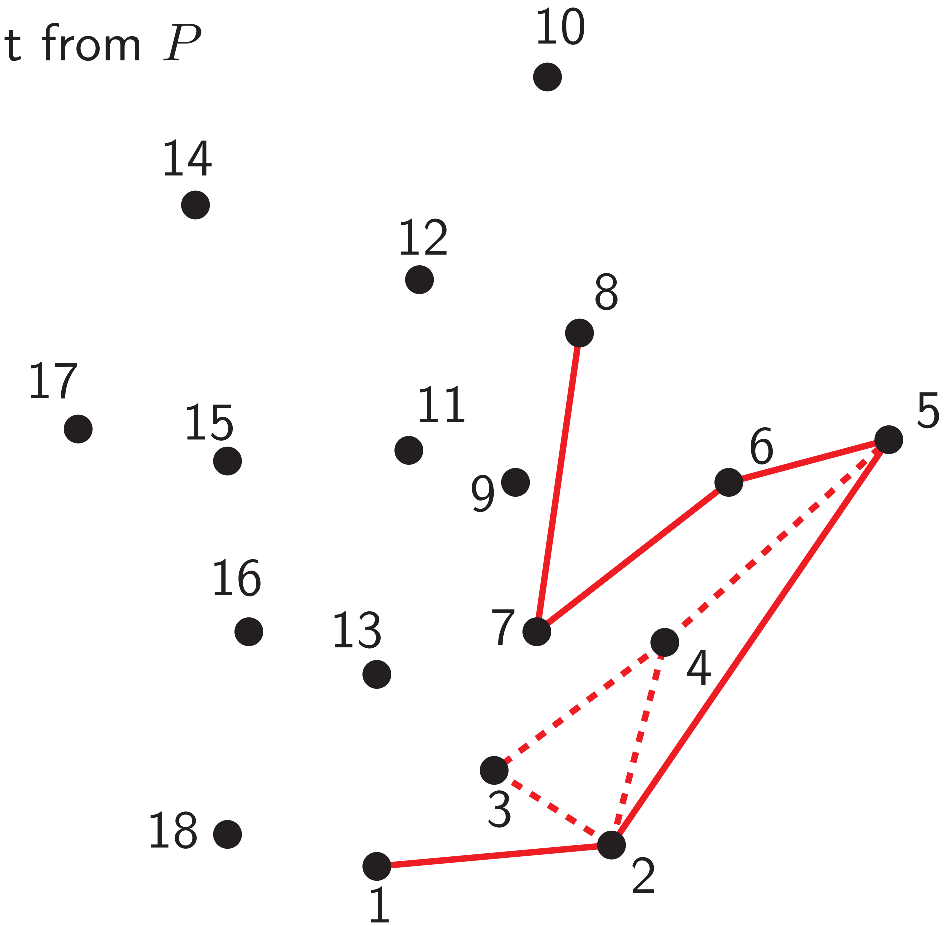
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

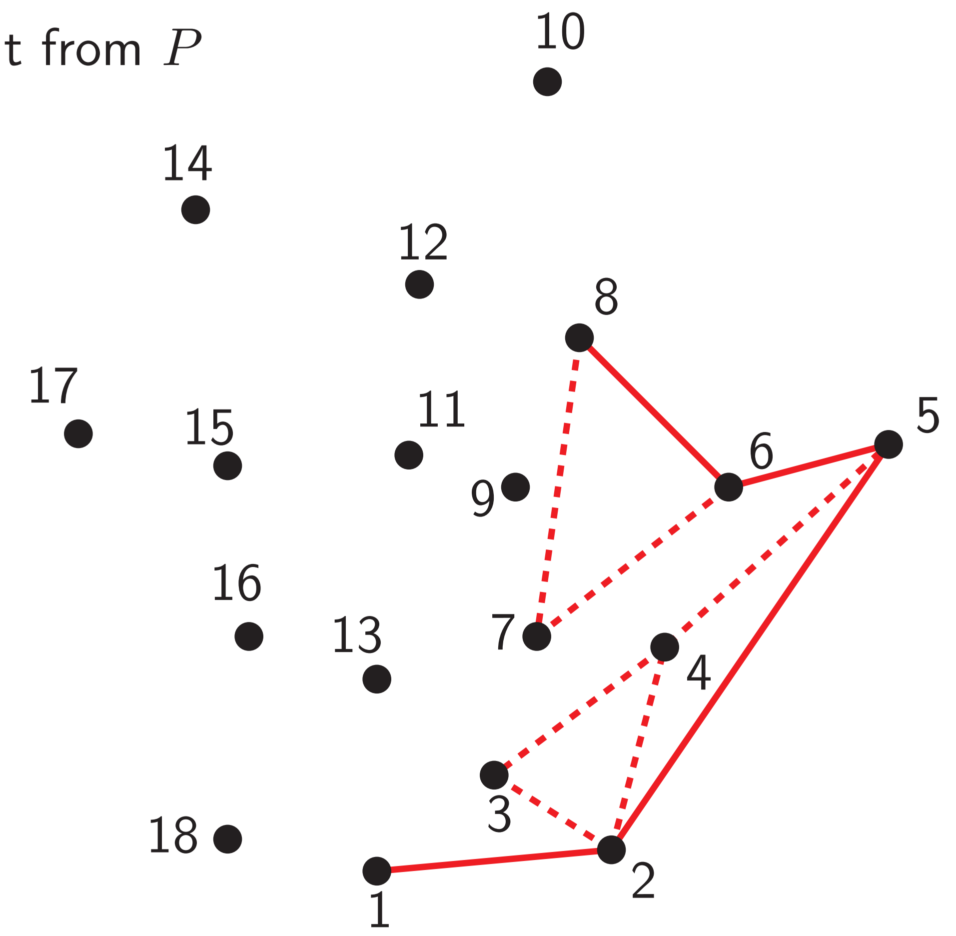
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

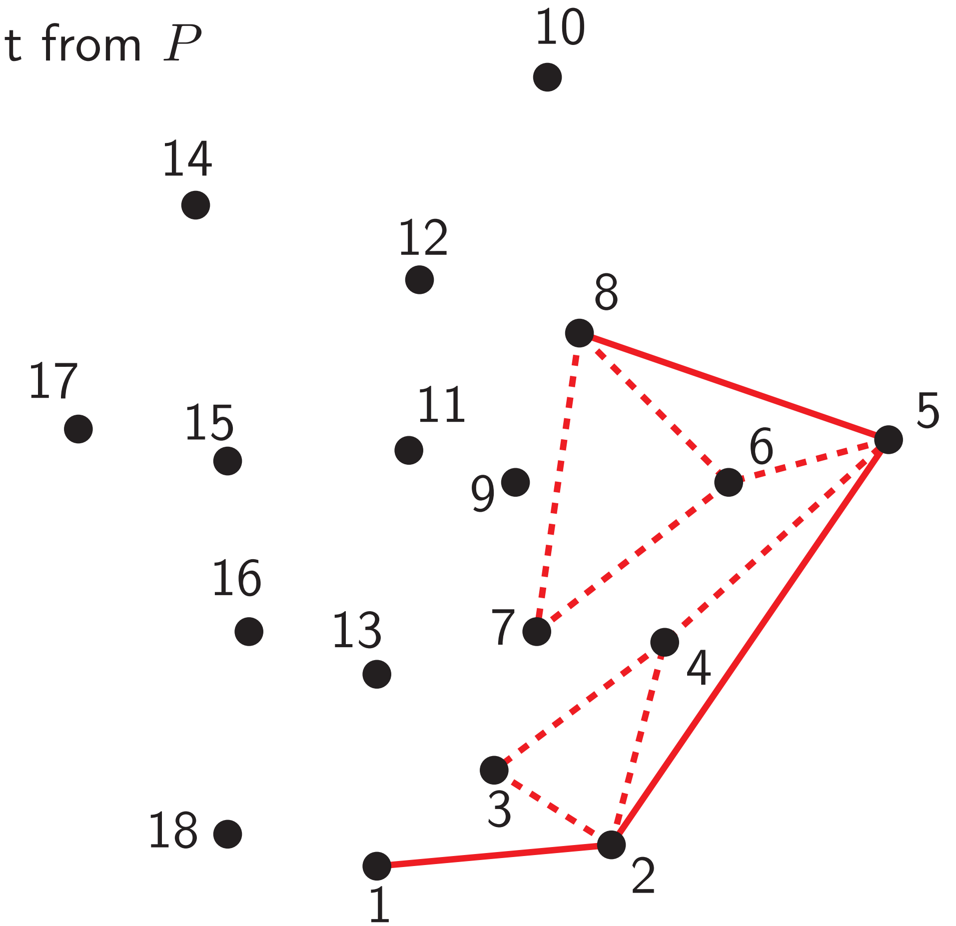
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

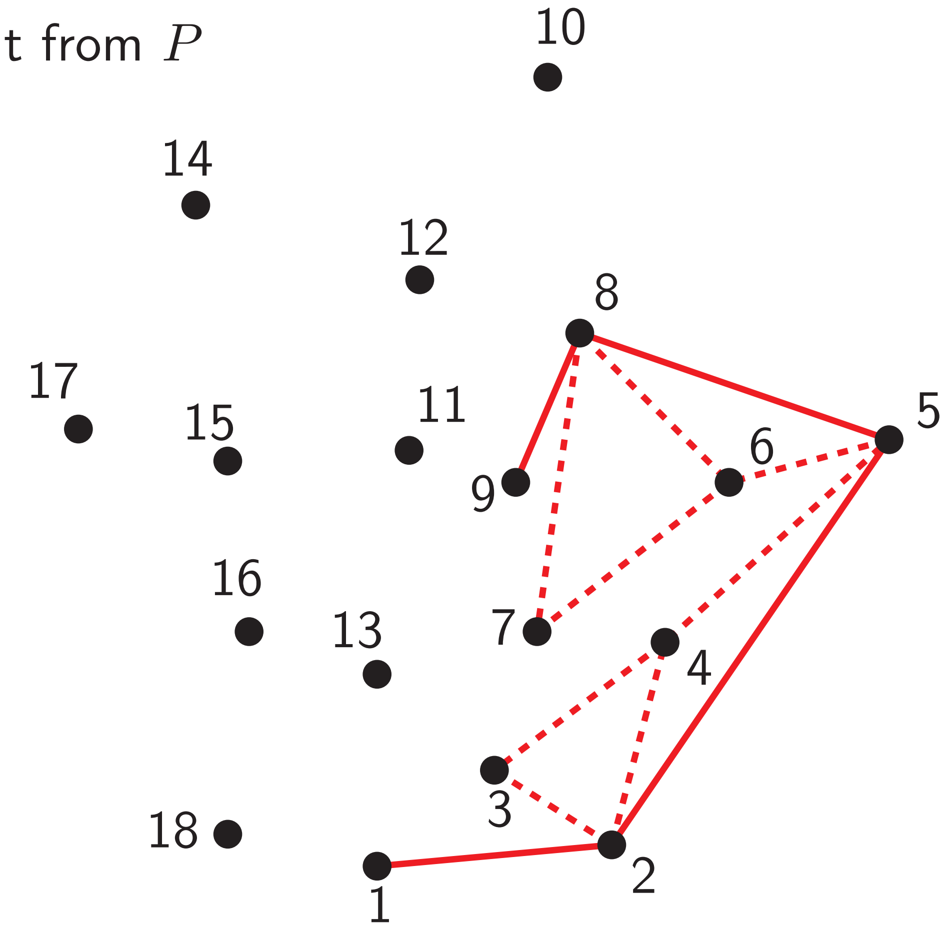
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



CONVEX HULL IN 2D

Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

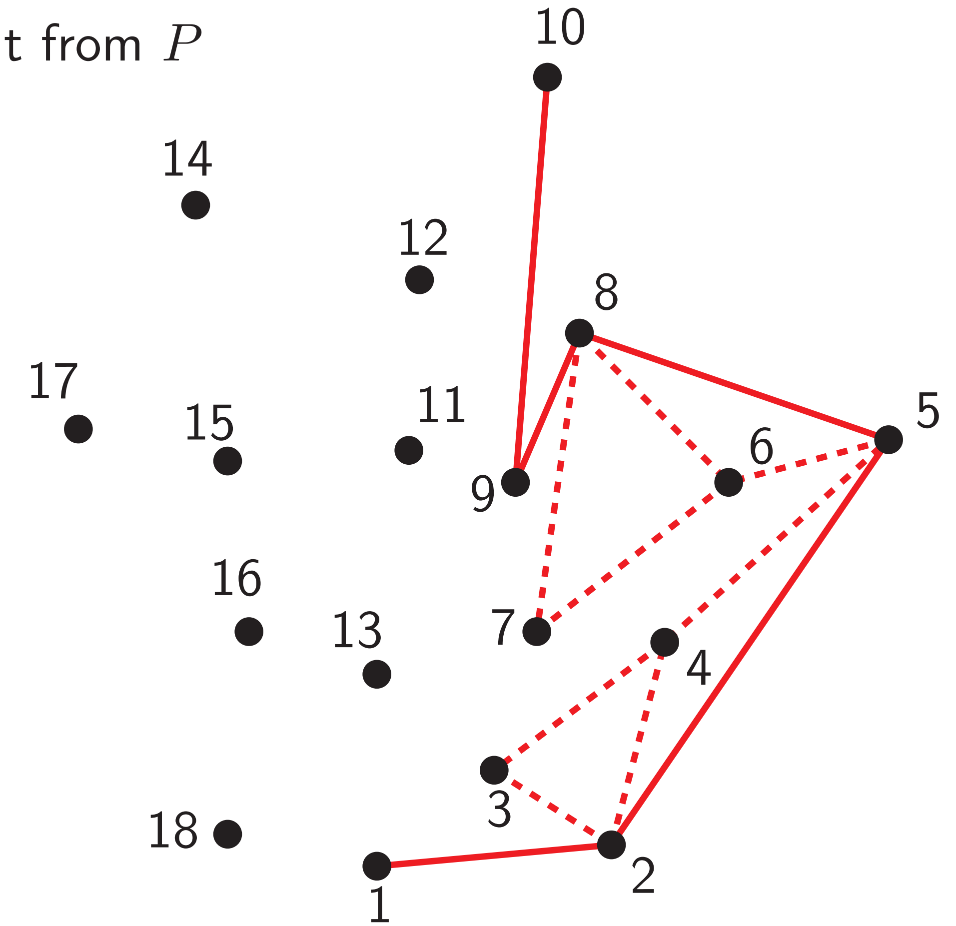
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

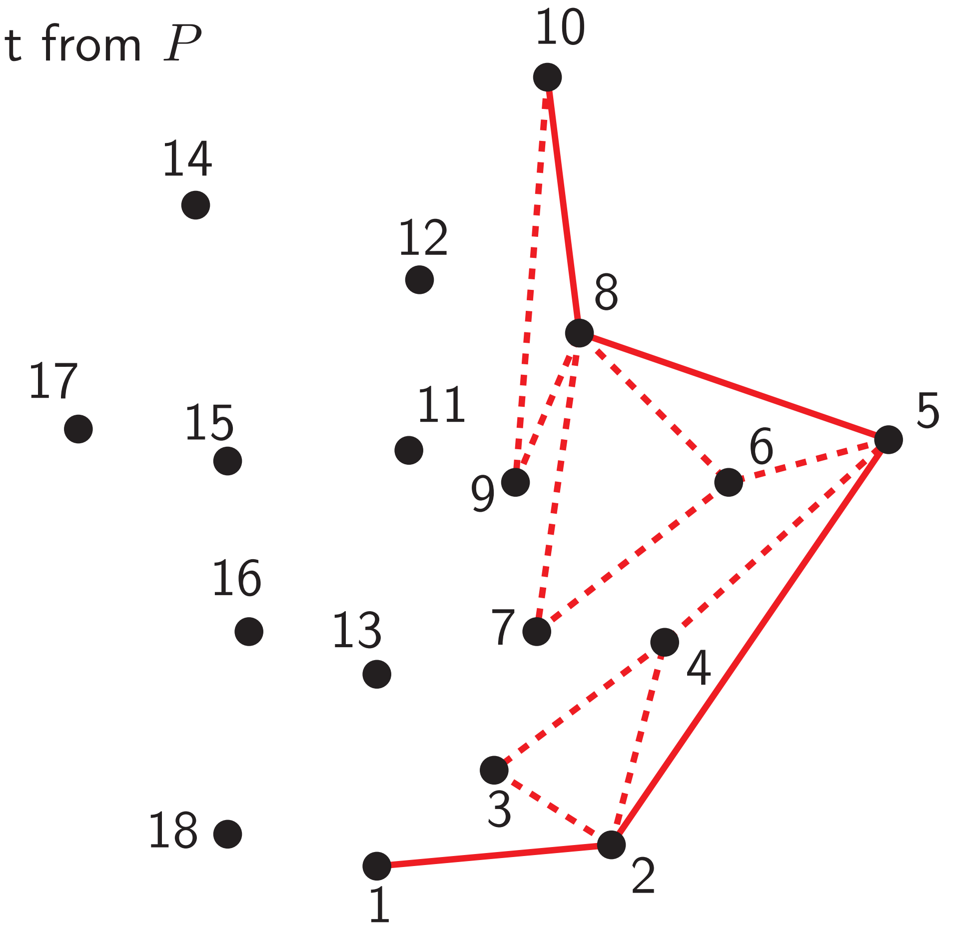
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

Return l

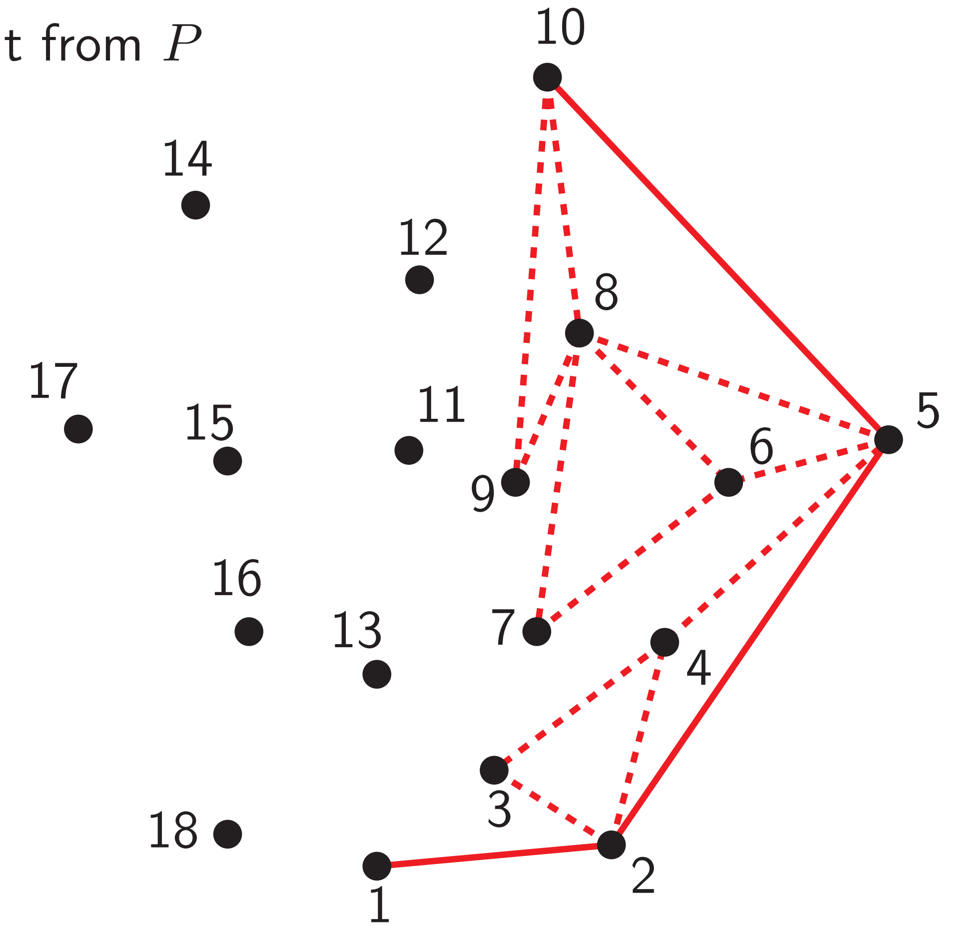


Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance
While there exist points $p_i \in P$ to be explored, do:
$p = \mathrm{top}(l)$
$p^- = \mathrm{previous}(\mathrm{top}(l))$
- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

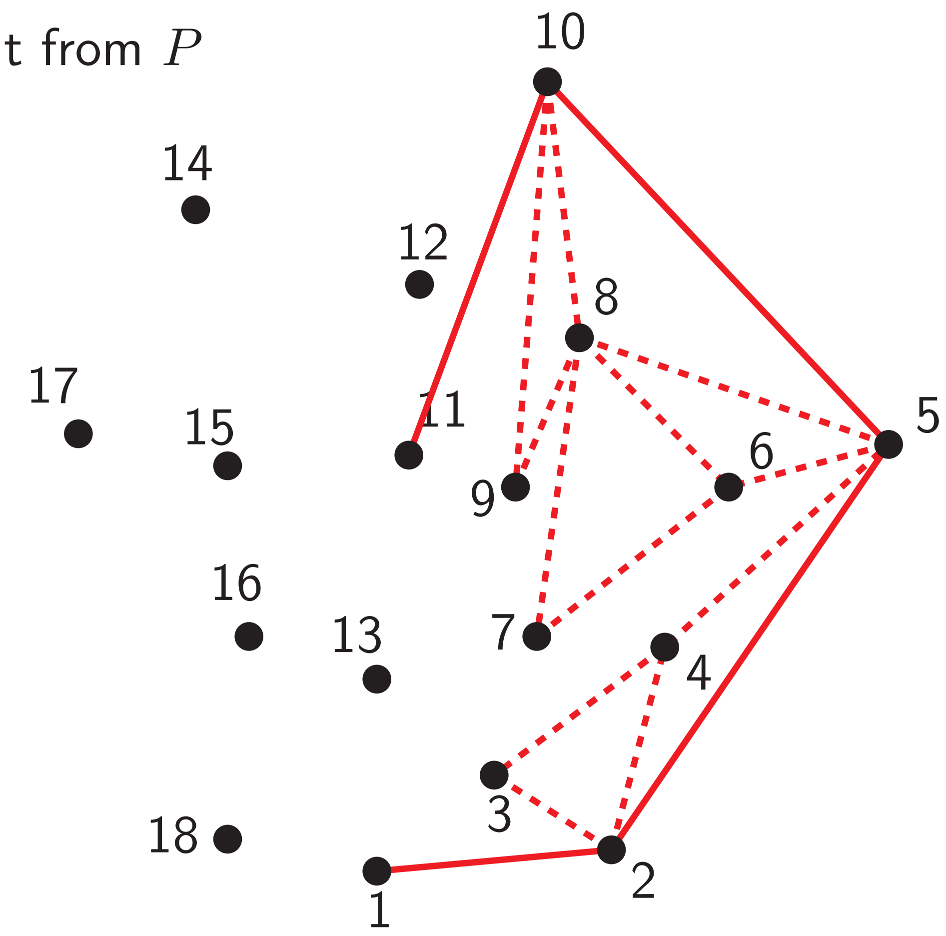
While there exist points $p_i \in P$ to be explored, do:

$p = \mathrm{top}(l)$

$p^- = \mathrm{previous}(\mathrm{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

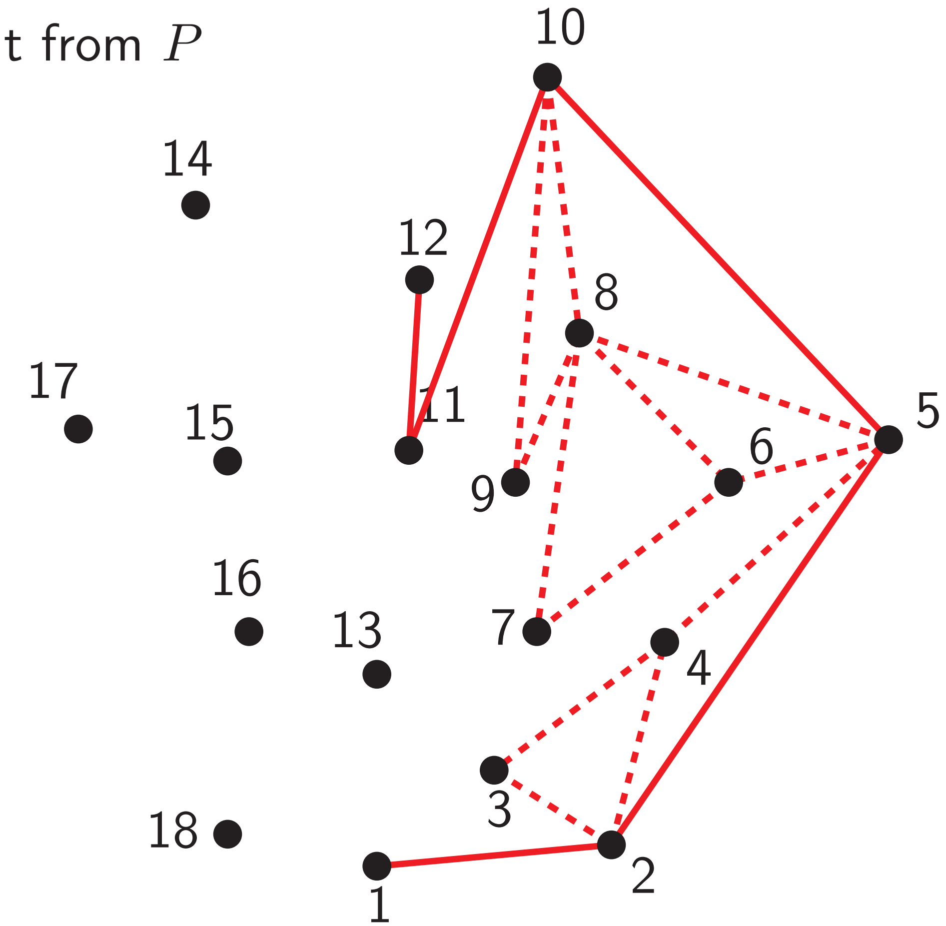
While there exist points $p_i \in P$ to be explored, do:

$p = \mathrm{top}(l)$

$p^- = \mathrm{previous}(\mathrm{top}(l))$

- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

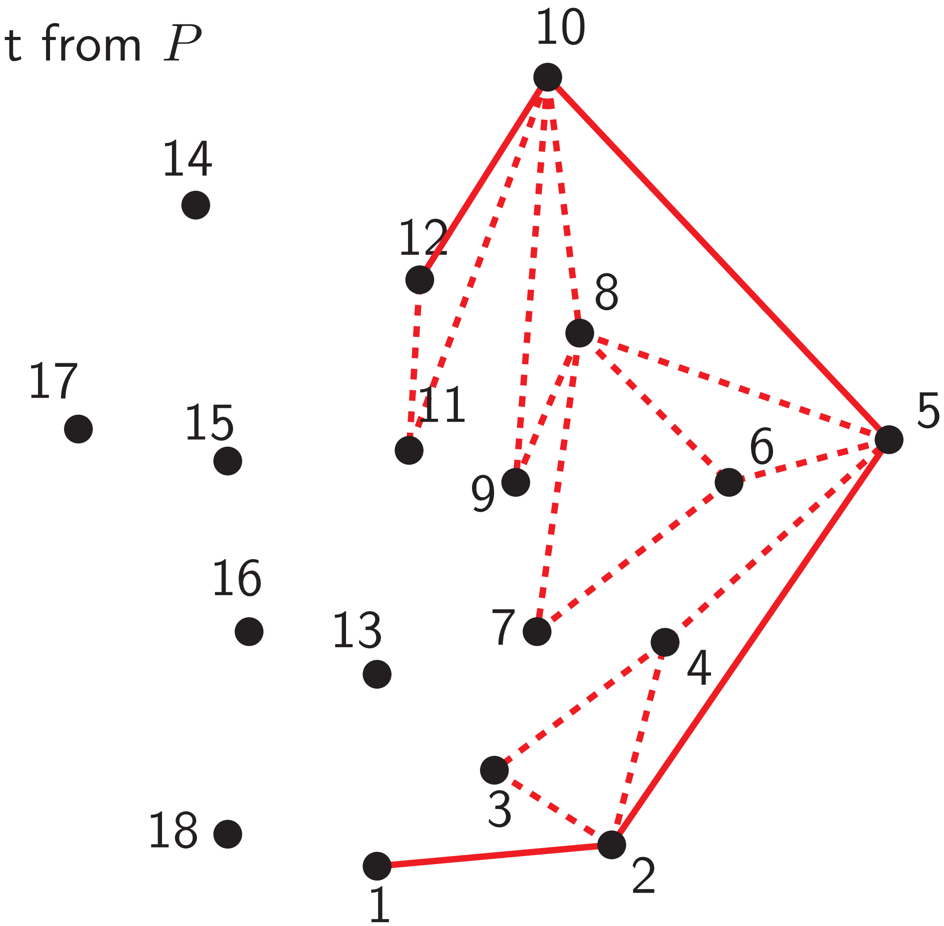
While there exist points $p_i \in P$ to be explored, do:

$p = \mathrm{top}(l)$

$p^- = \mathrm{previous}(\mathrm{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

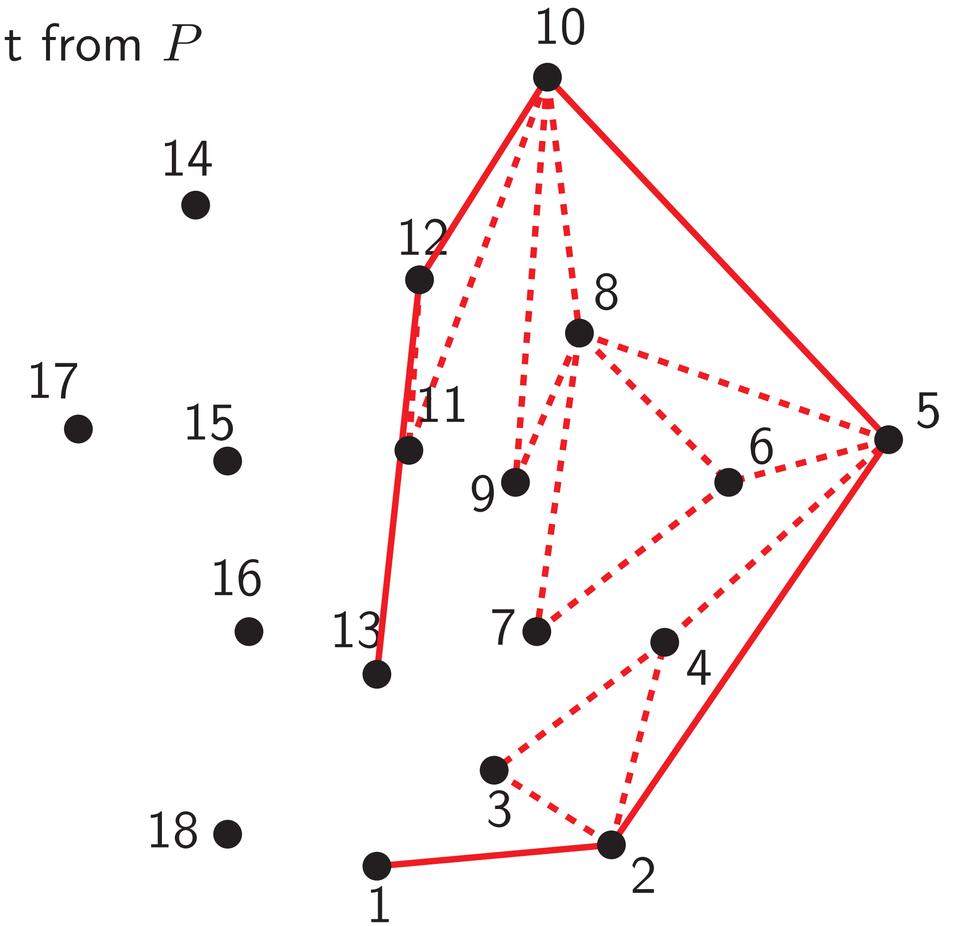
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

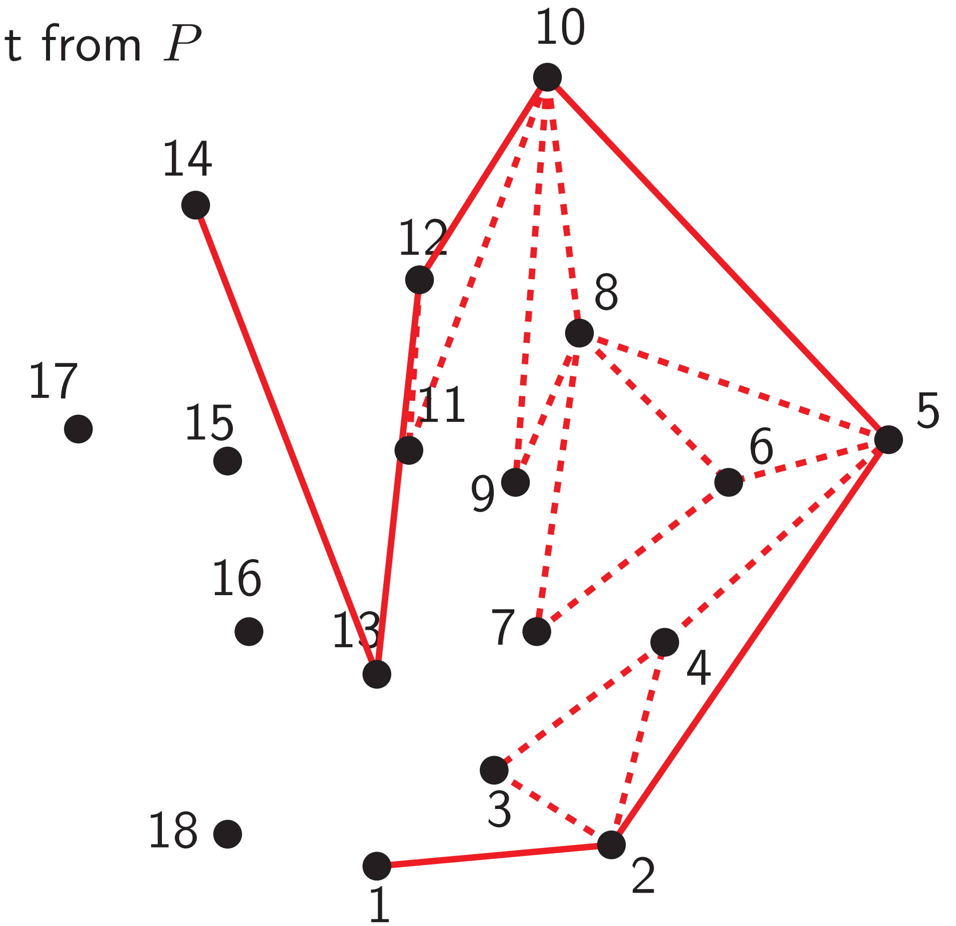
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

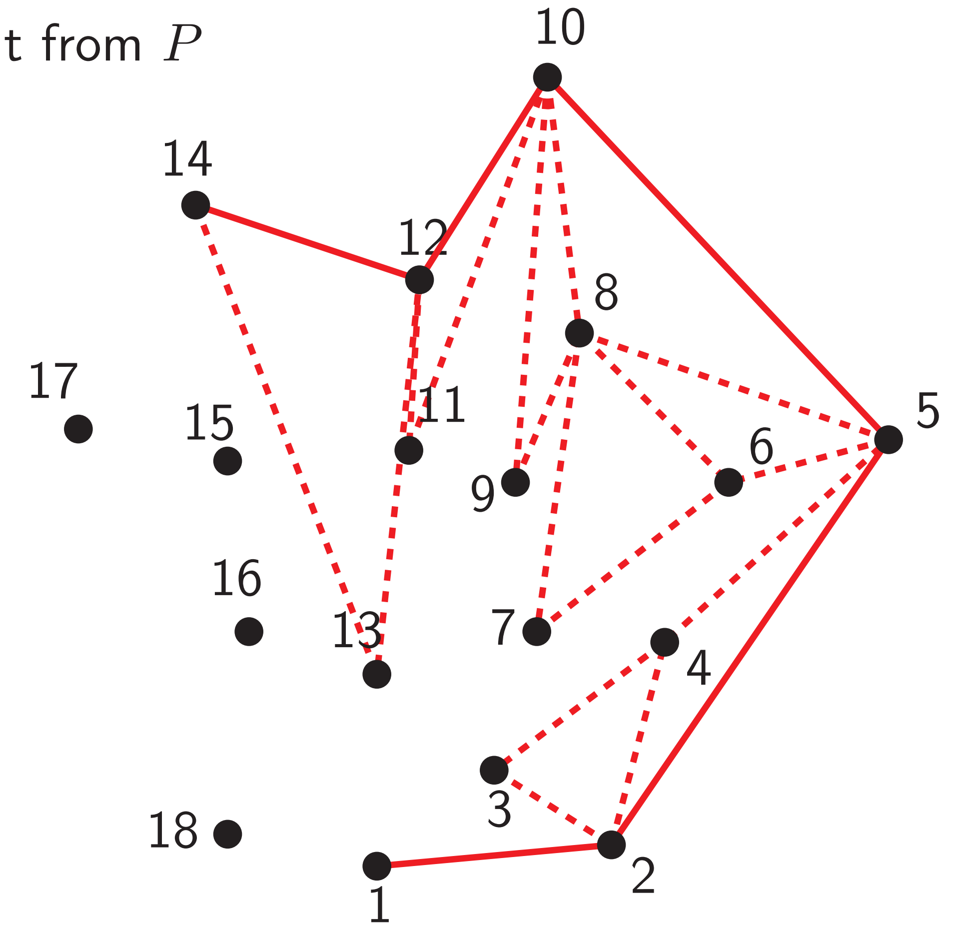
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

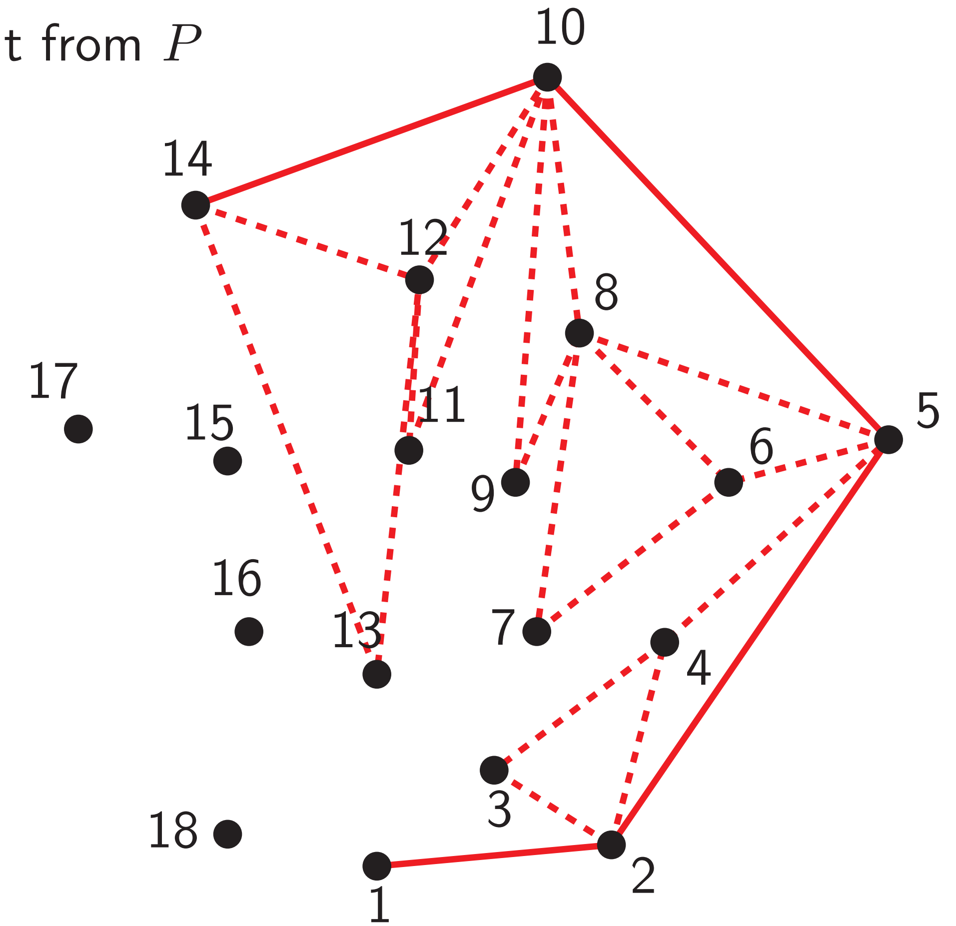
While there exist points $p_i \in P$ to be explored, do:

$p = \mathrm{top}(l)$

$p^- = \mathrm{previous}(\mathrm{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

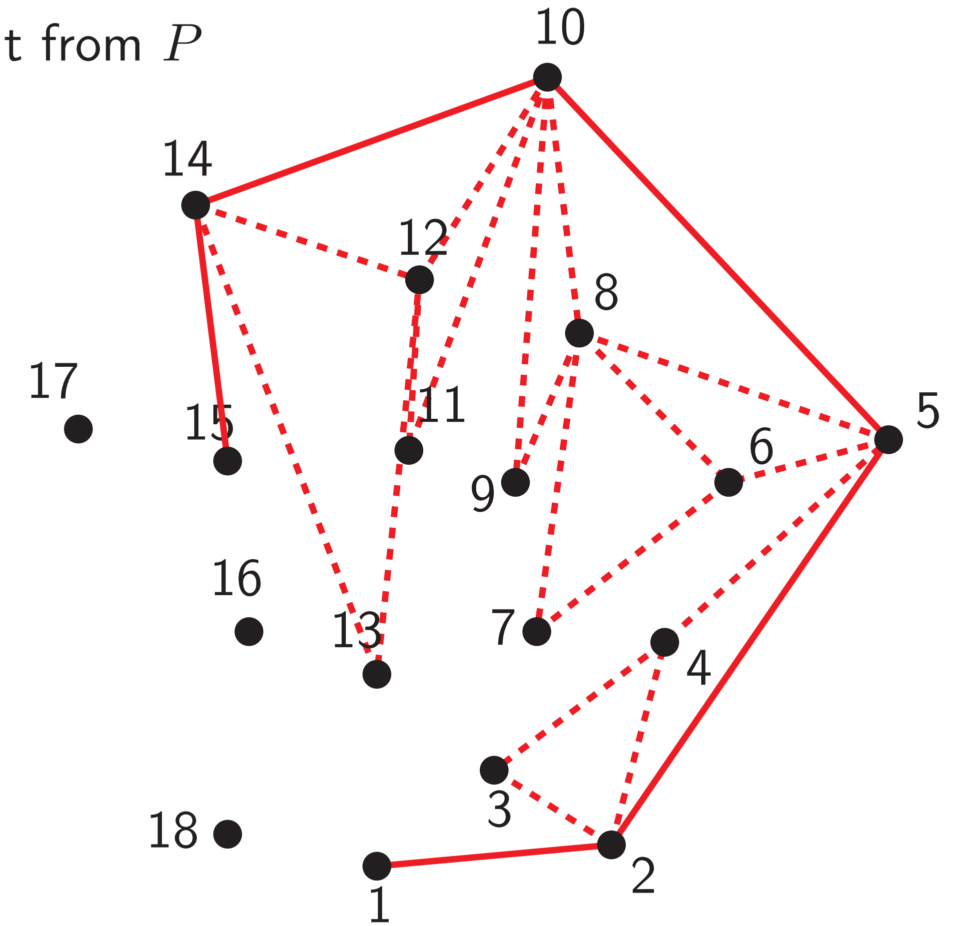
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

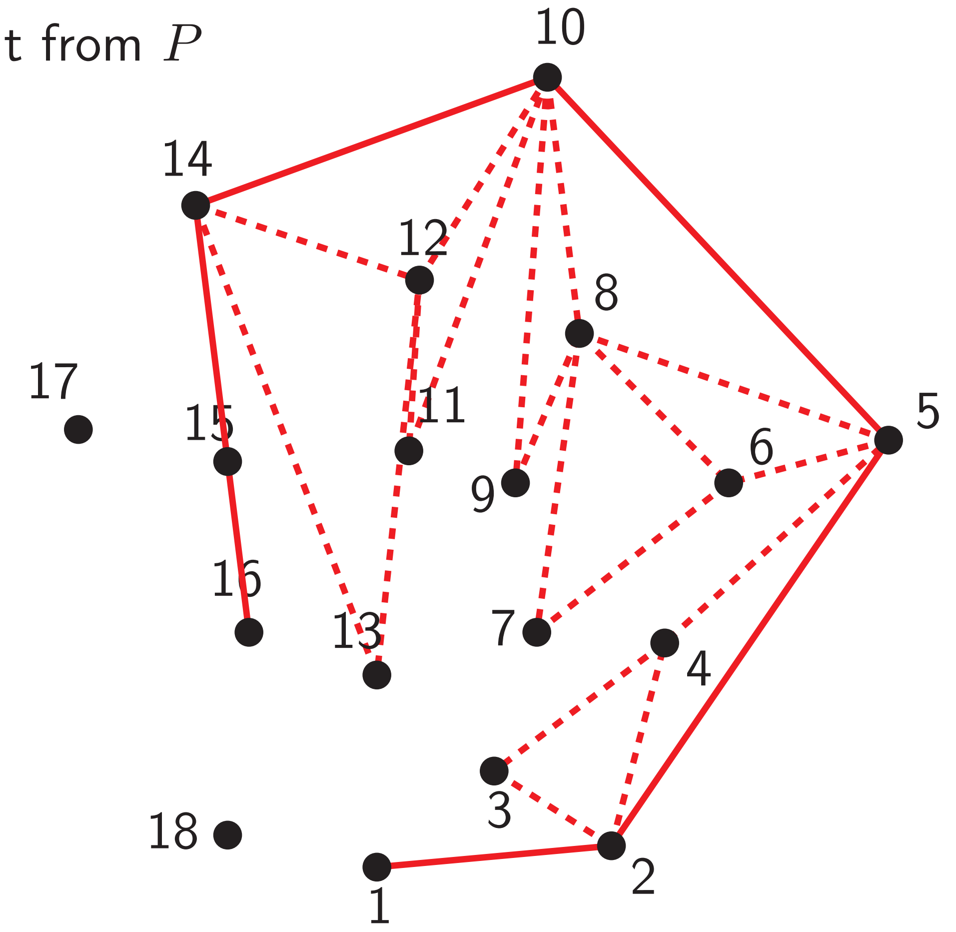
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



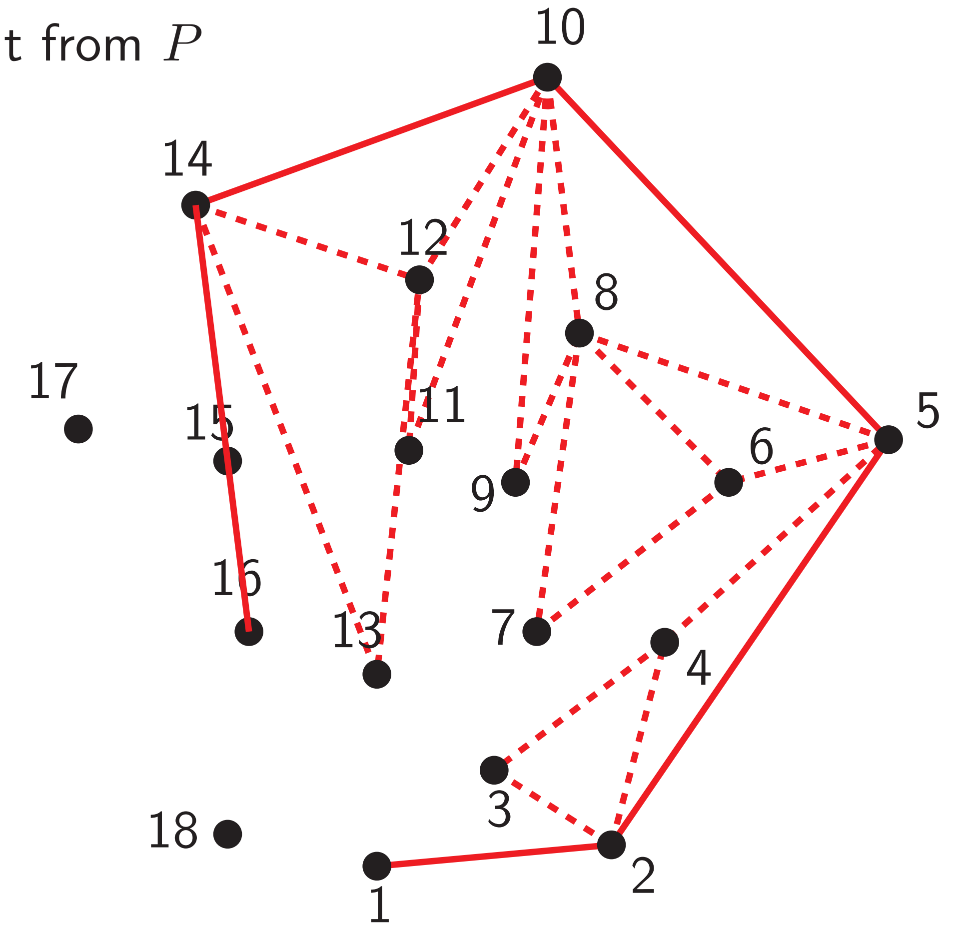
CONVEX HULL IN 2D

Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance
While there exist points $p_i \in P$ to be explored, do:
$p = \text{top}(l)$
$p^- = \text{previous}(\text{top}(l))$
- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

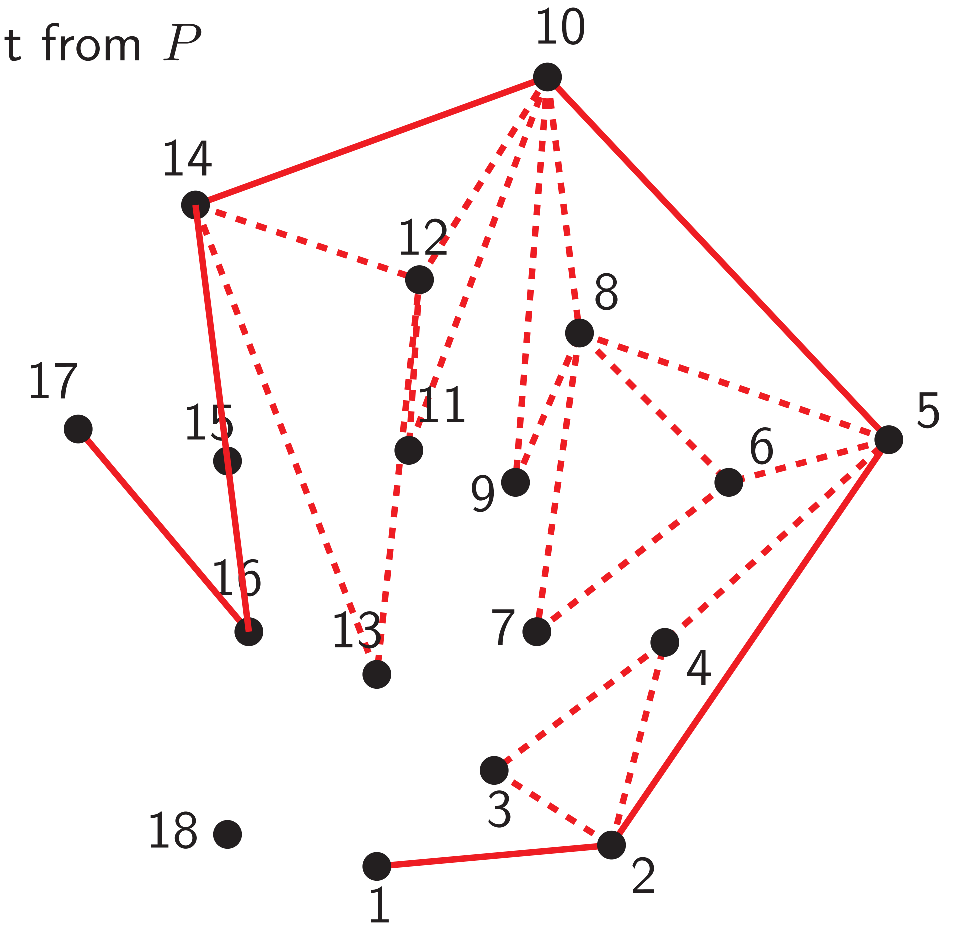
While there exist points $p_i \in P$ to be explored, do:

$p = \mathrm{top}(l)$

$p^- = \mathrm{previous}(\mathrm{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

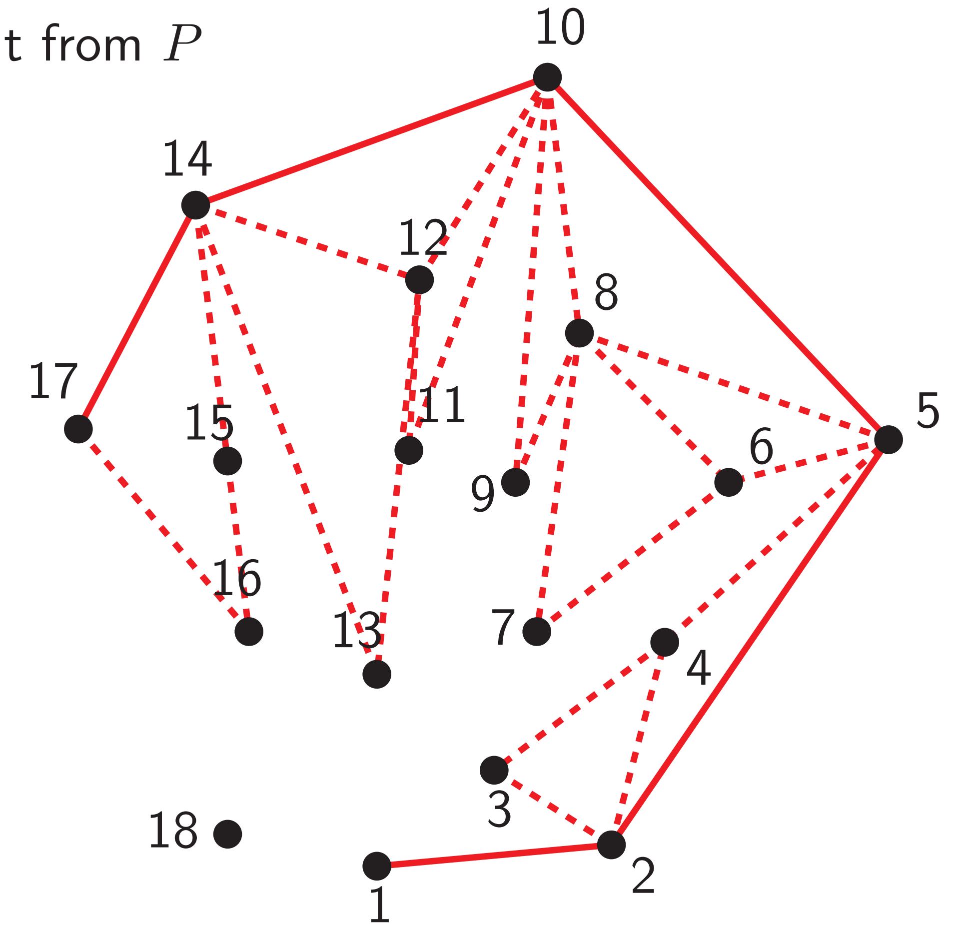
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l

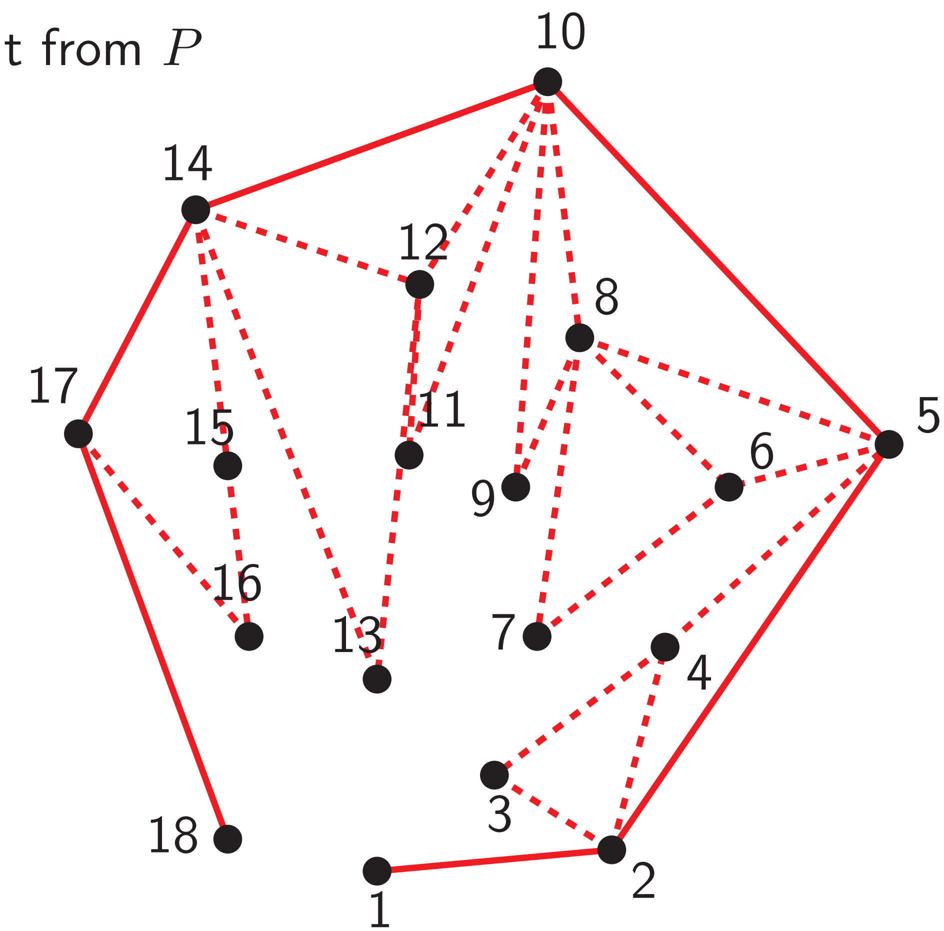


Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance
While there exist points $p_i \in P$ to be explored, do:
$p = \mathrm{top}(l)$
$p^- = \mathrm{previous}(\mathrm{top}(l))$
- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

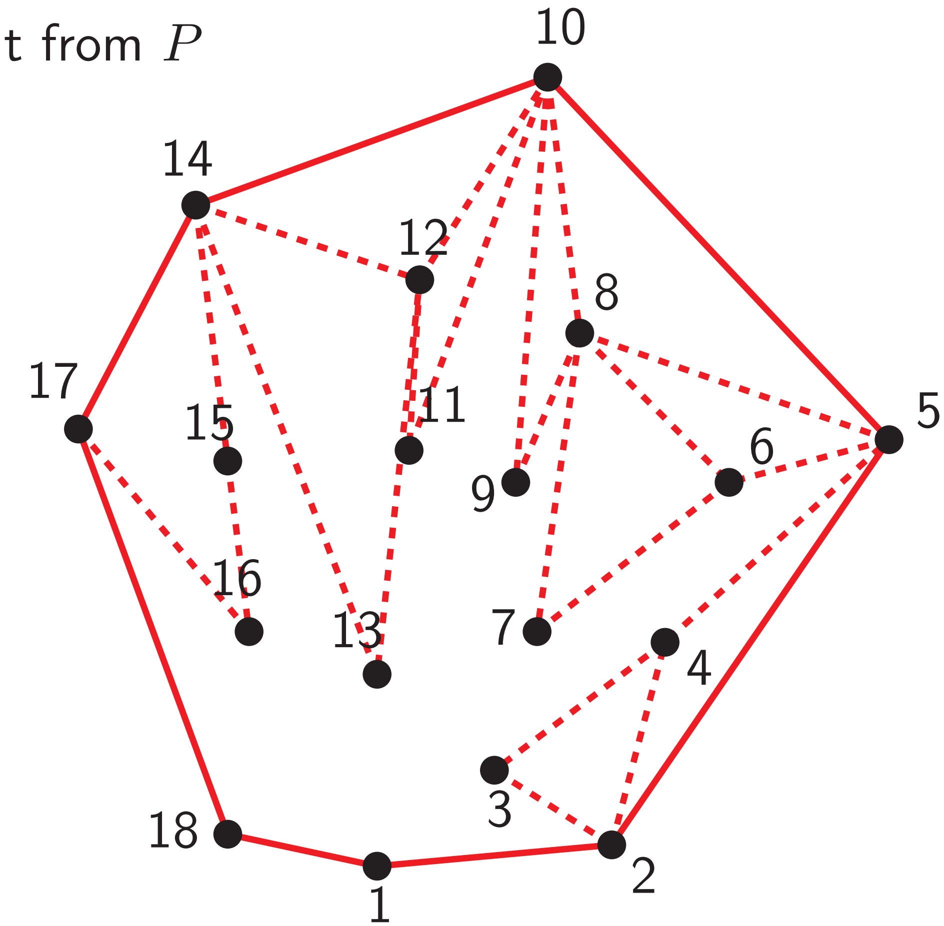
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



Graham's algorithm

Initialization

- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

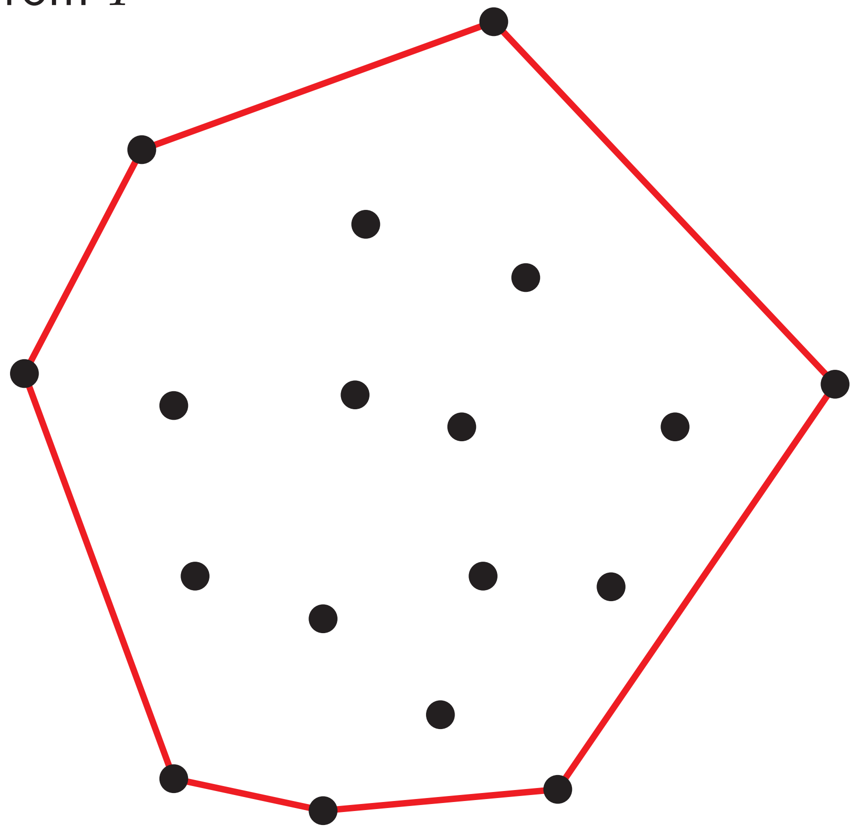
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
    - Push p_i in l
    - Advance i
- Else:
    - Pop p from l

Return l



Graham's algorithm

Initialization
- Find a vertex v of $ch(P)$, push it in l and delete it from P
- Angularly sort the points around v
- Push the first point in l and delete if from P

Advance

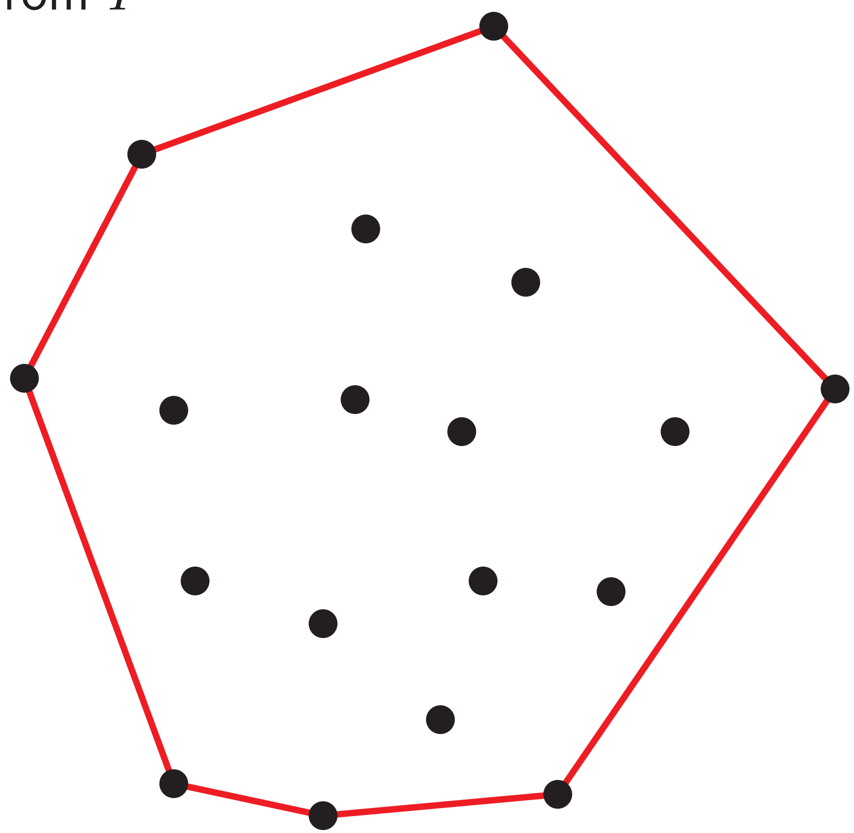
While there exist points $p_i \in P$ to be explored, do:

$p = \text{top}(l)$

$p^- = \text{previous}(\text{top}(l))$

- If $p^- p p_i$ is a left turn:
  - Push p_i in l
  - Advance i
- Else:
  - Pop p from l

Return l



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

Incremental algorithm

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

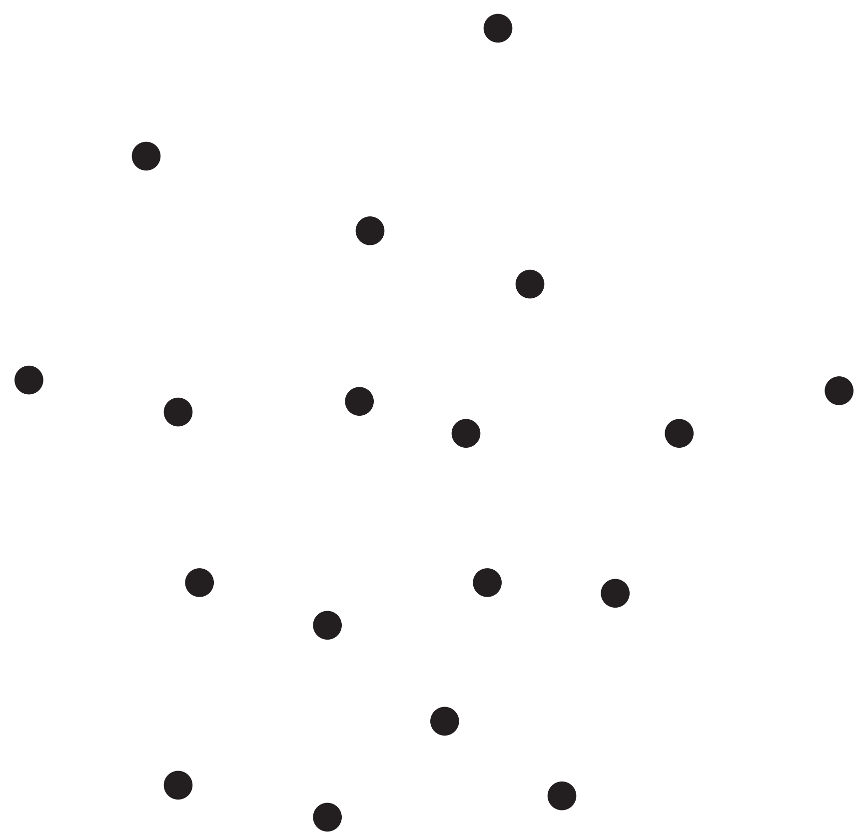
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

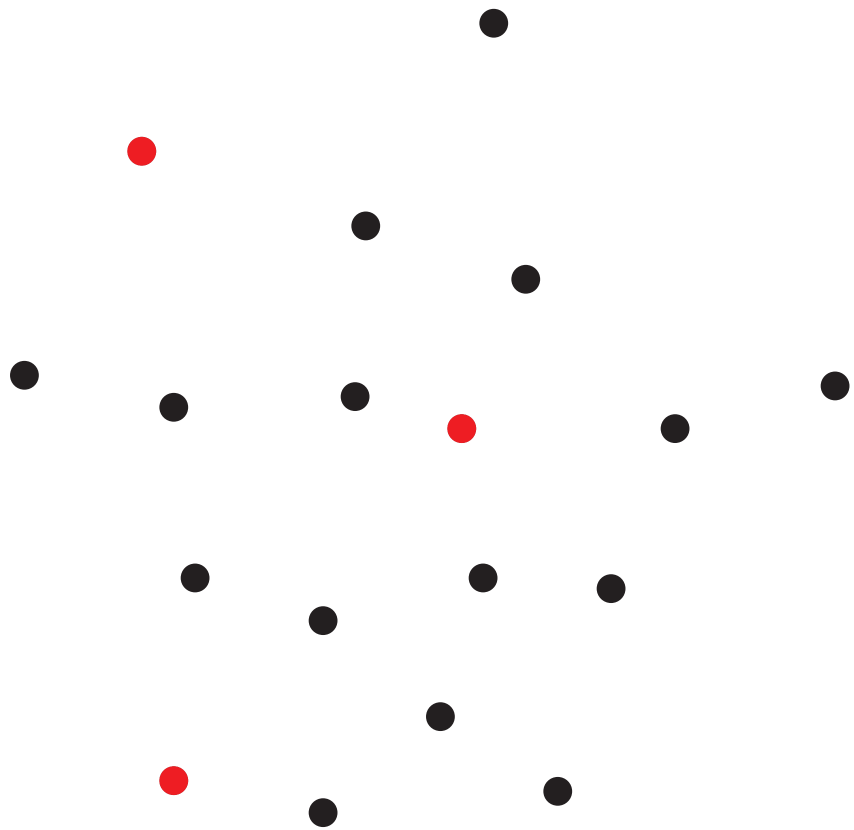
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

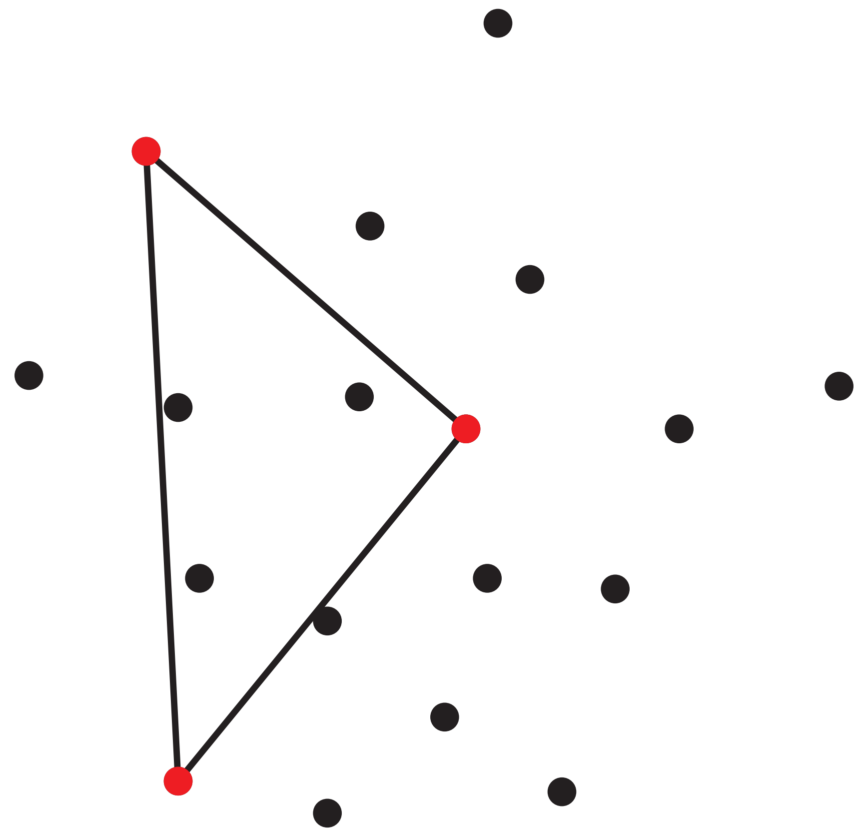
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

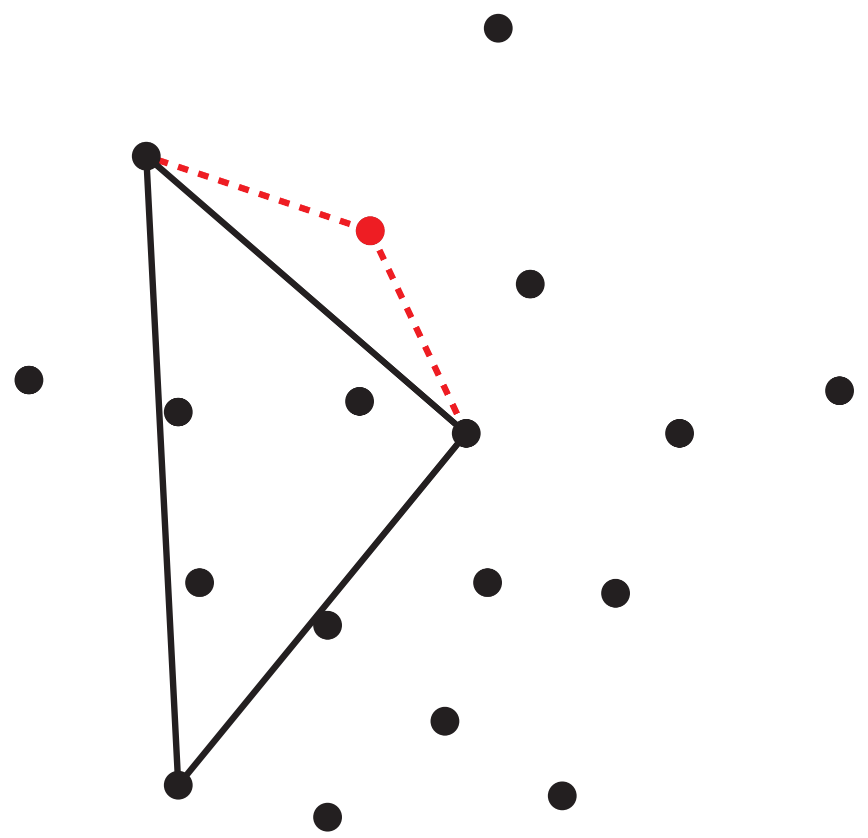
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

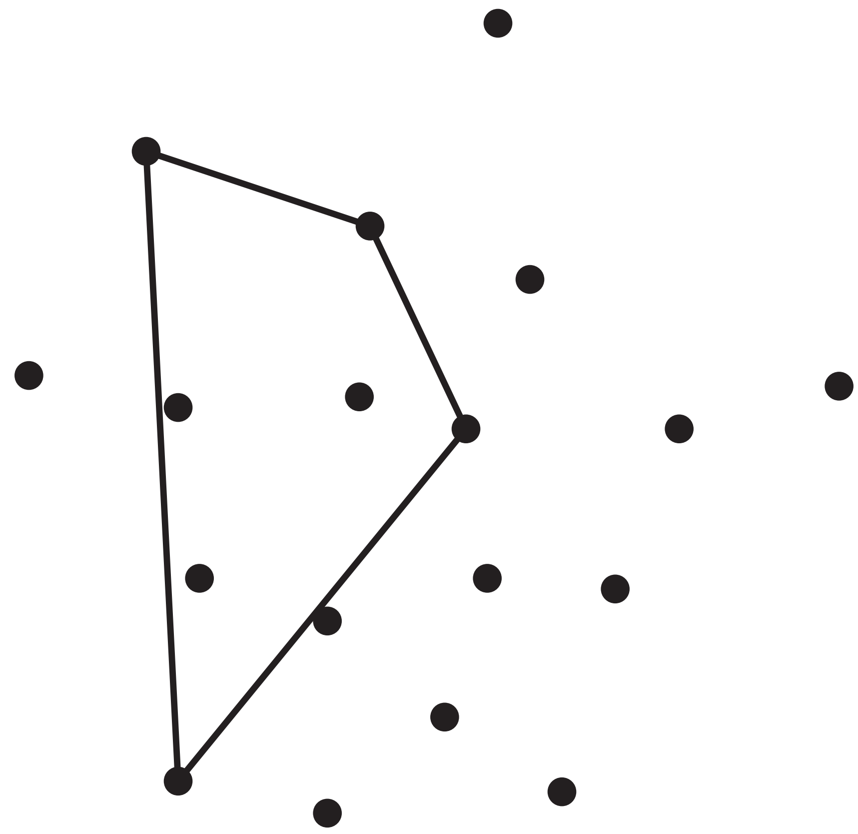
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

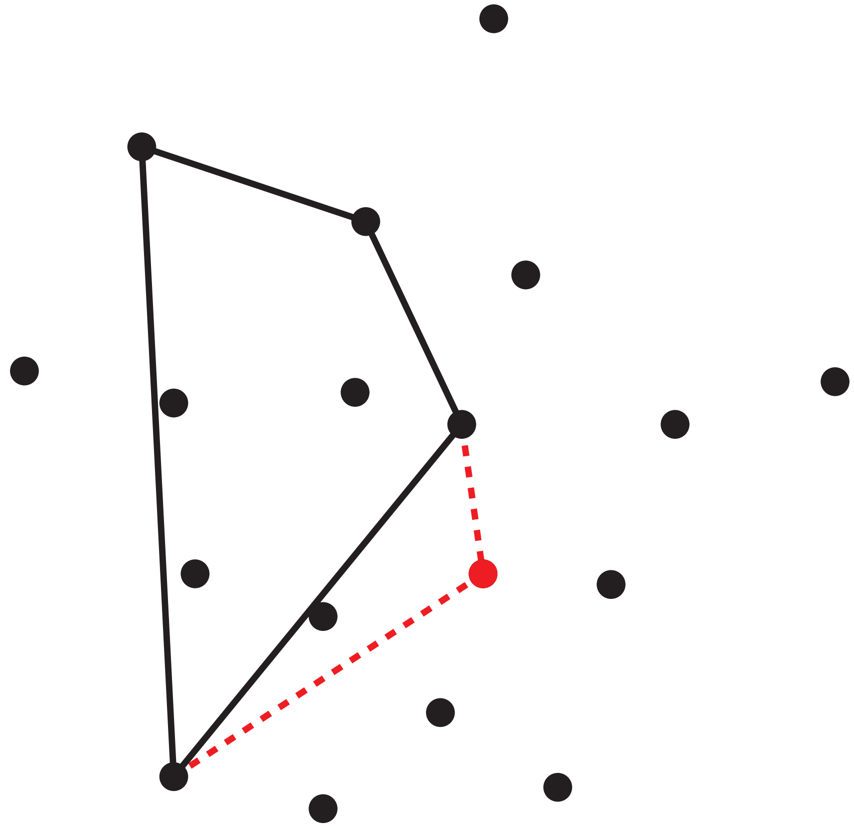
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

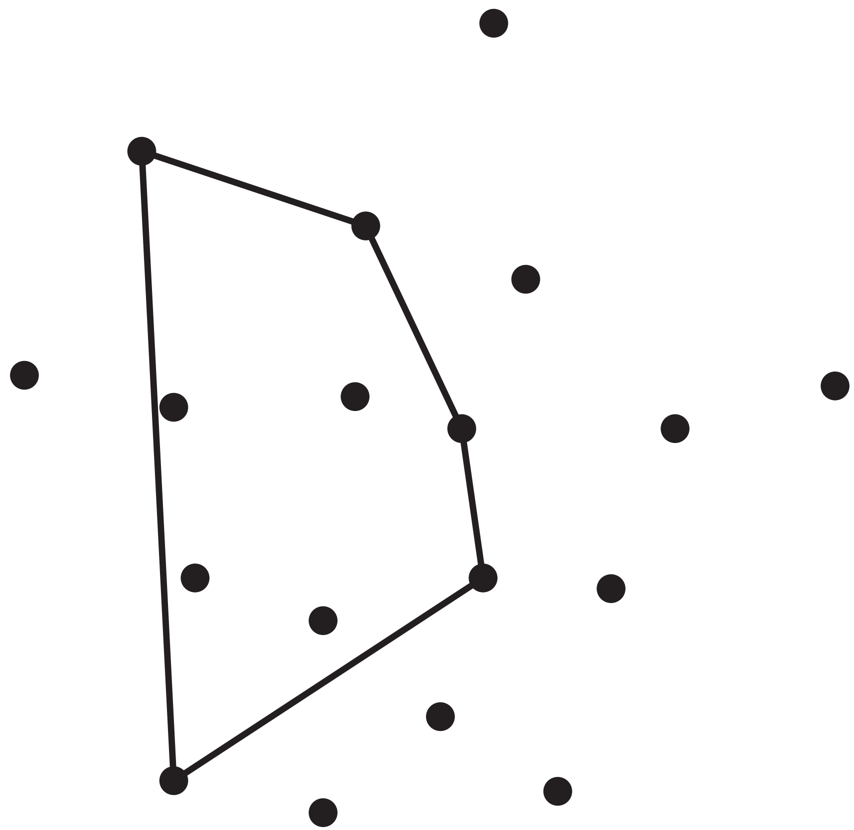
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

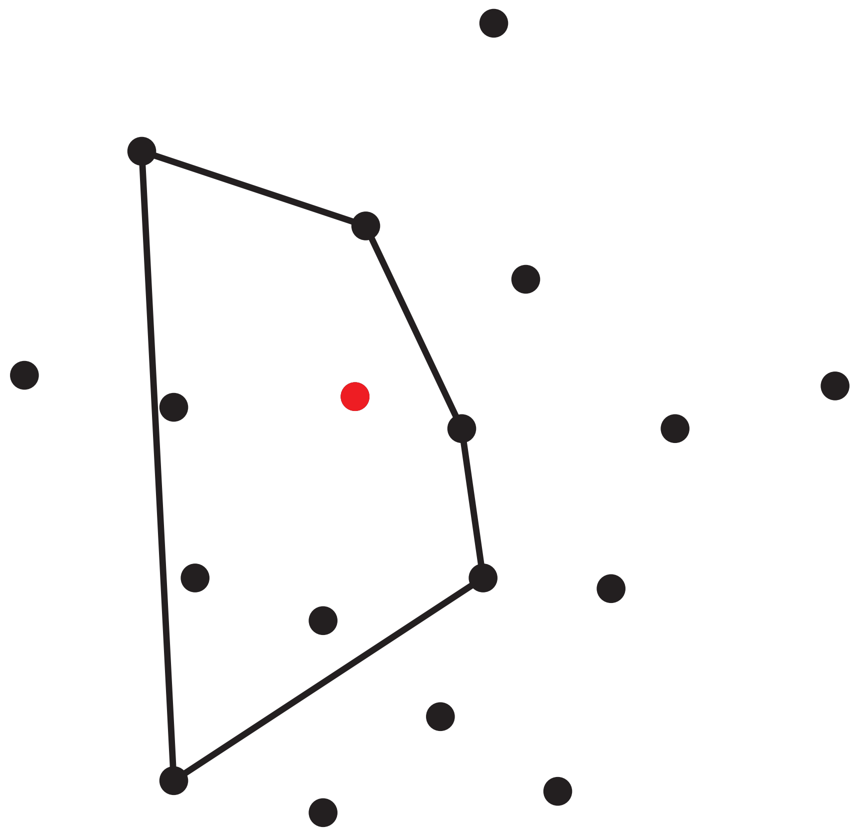
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

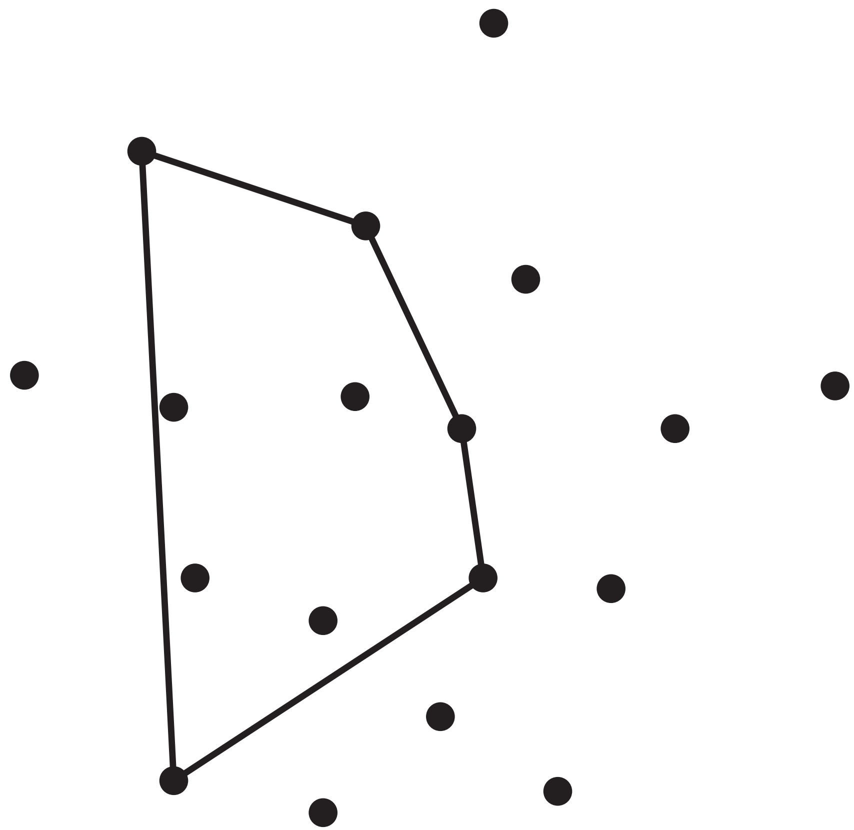
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

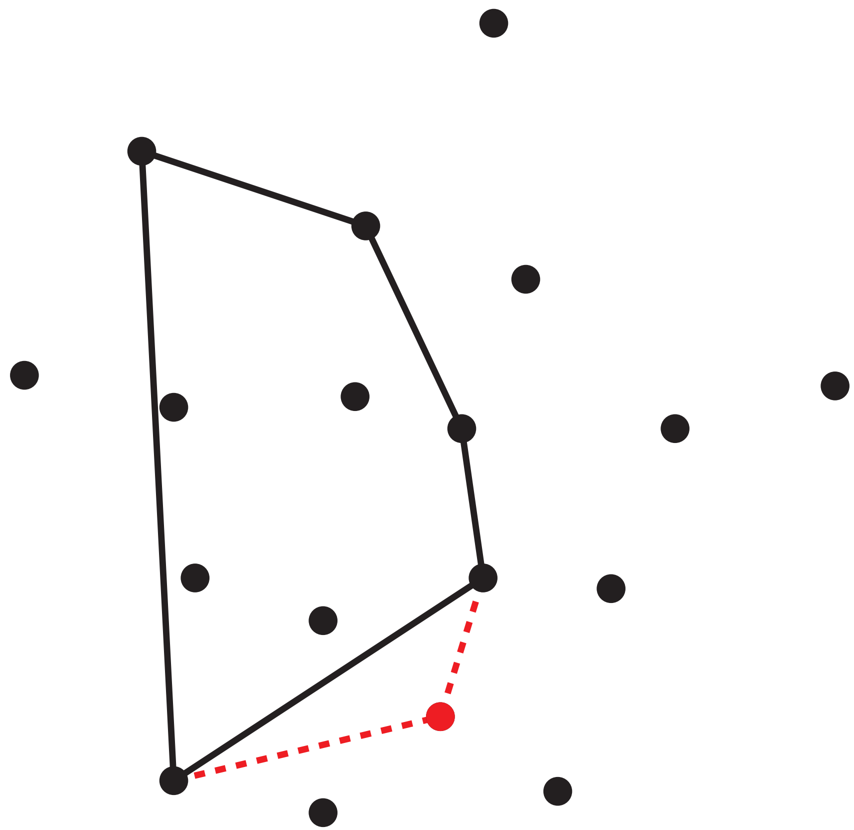
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l

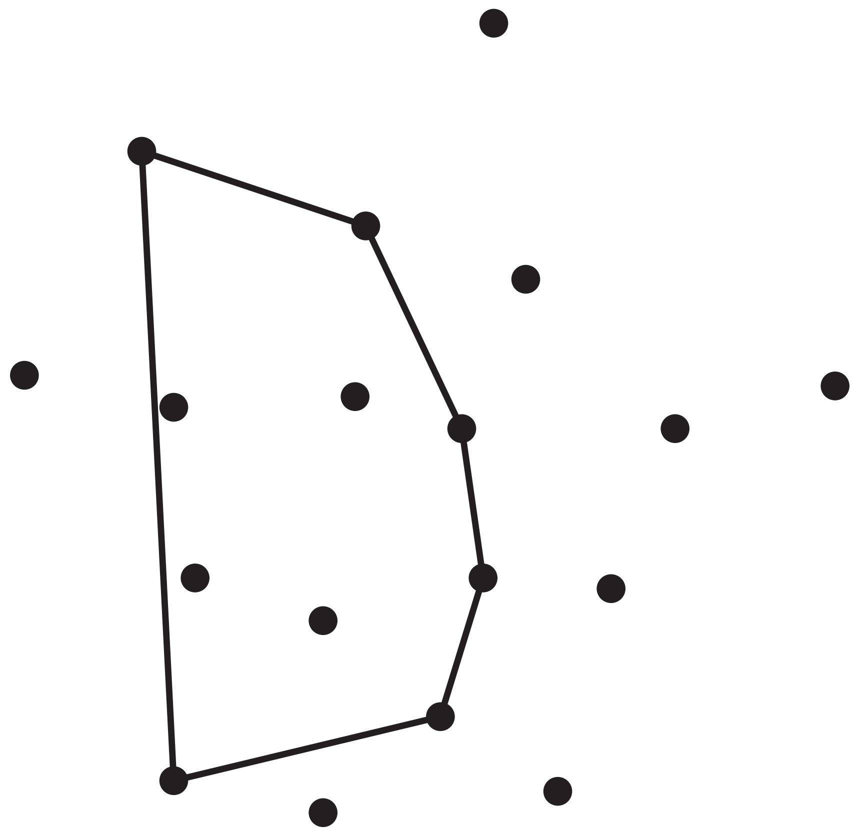


Incremental algorithm

Initialization
$l = p_1, p_2, p_3$

Advance
From $i = 4$ to n, do:
If p_i lies in the exterior of the polygon defined by l:
- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \dots, p_r$ in l with the chain p_l, p_i, p_r

Return l

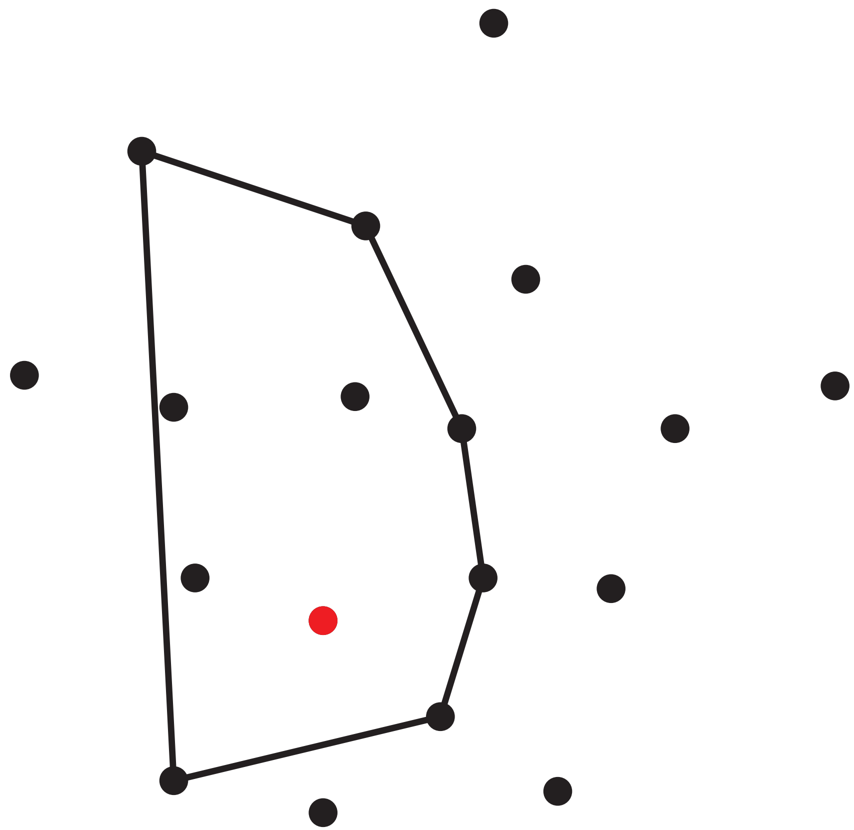


Incremental algorithm

Initialization
$l = p_1, p_2, p_3$

Advance
From $i = 4$ to n, do:
If p_i lies in the exterior of the polygon defined by l:
- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \dots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

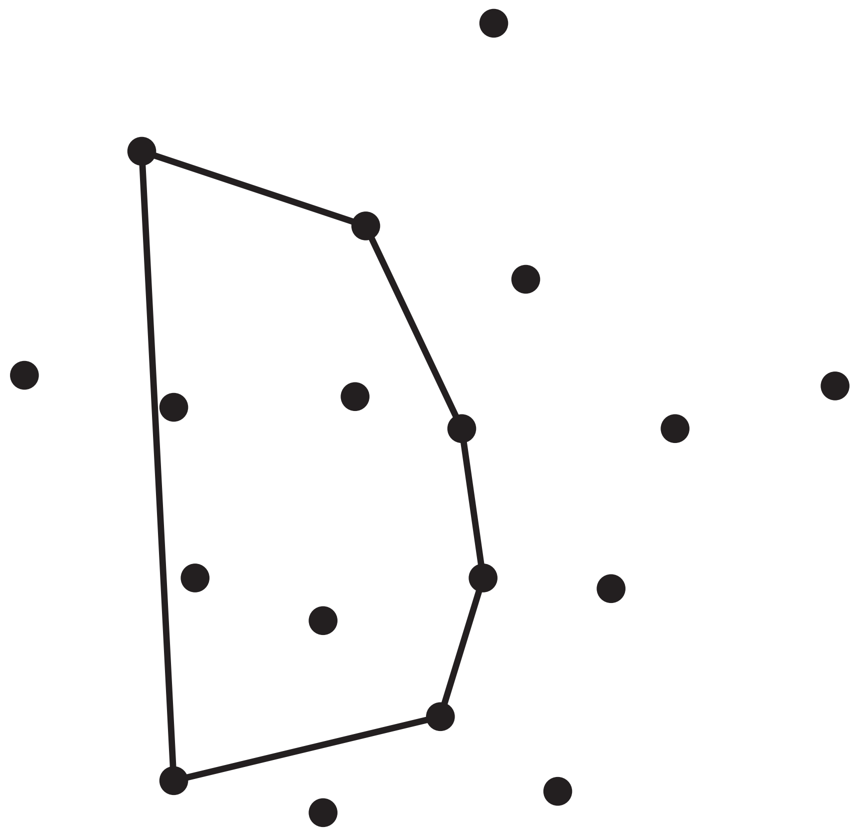
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

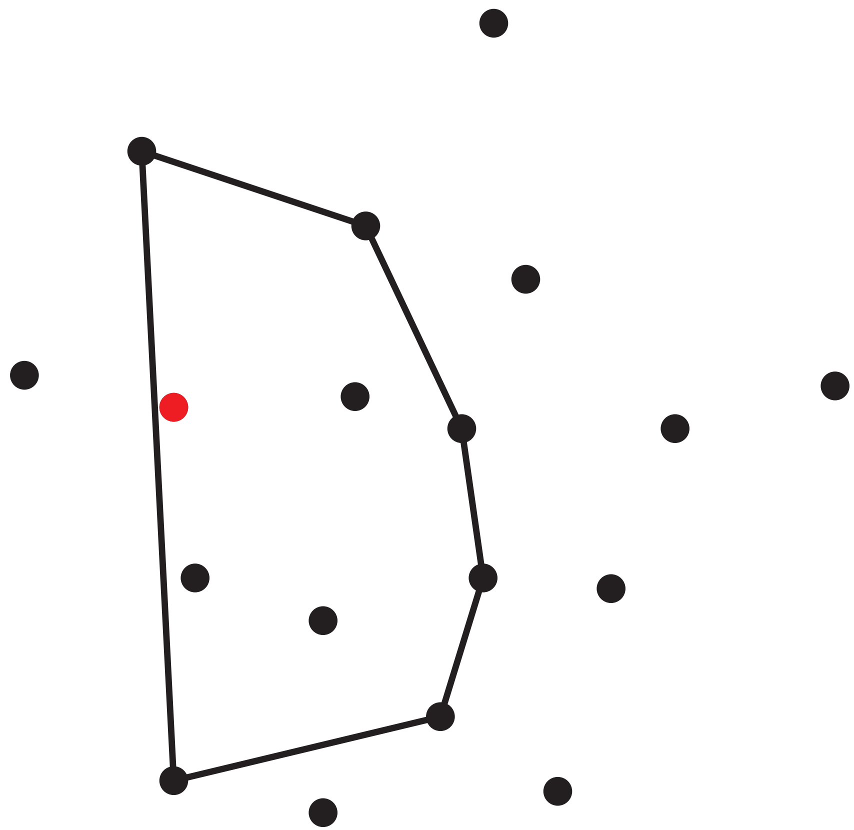
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

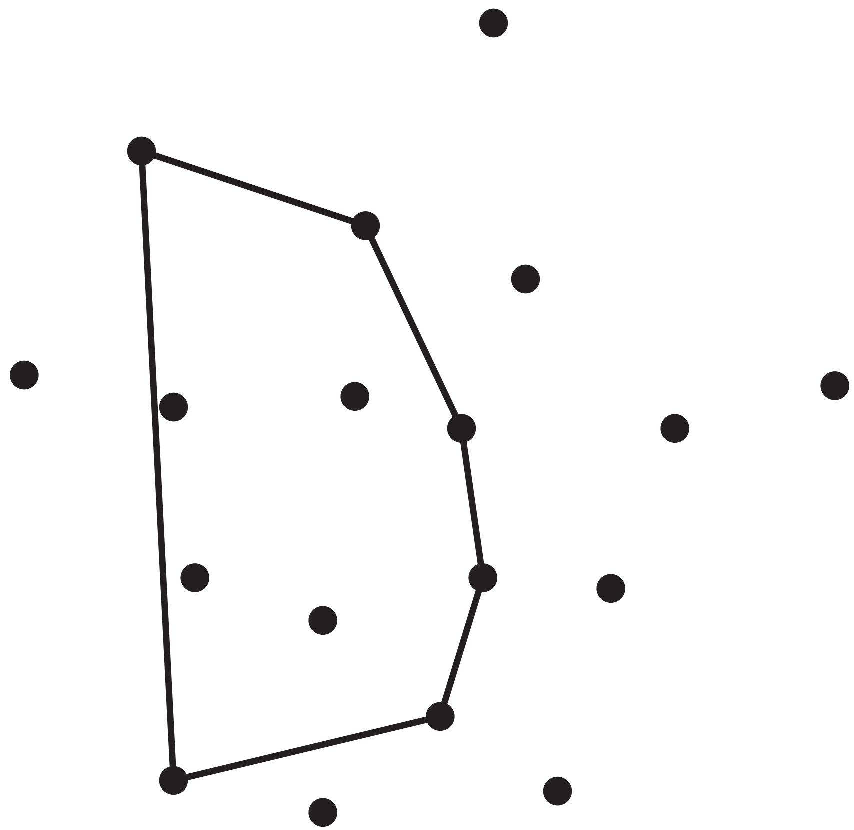
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

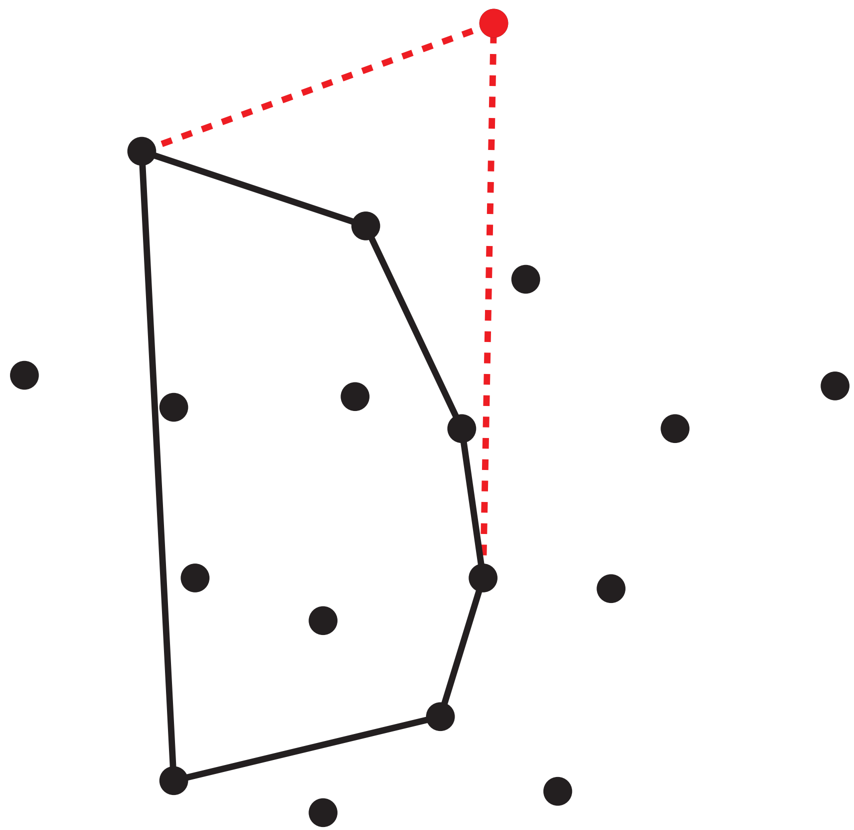
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \dots, p_r$ in l with the chain p_l, p_i, p_r

Return l

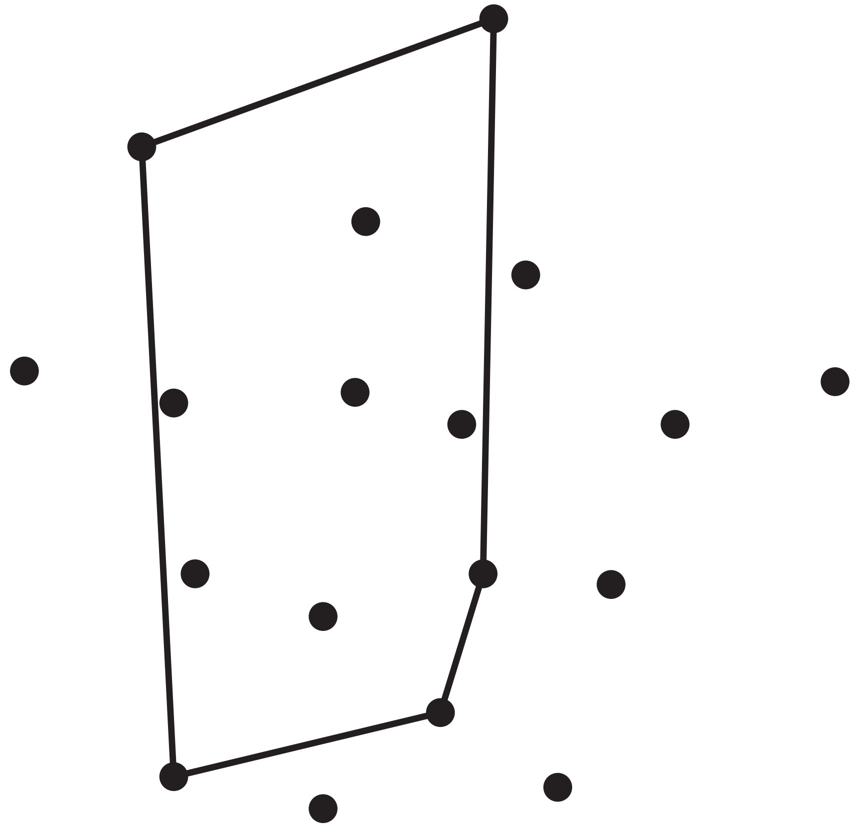


Incremental algorithm

Initialization
$l = p_1, p_2, p_3$

Advance
From $i = 4$ to n, do:
If p_i lies in the exterior of the polygon defined by l:
- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

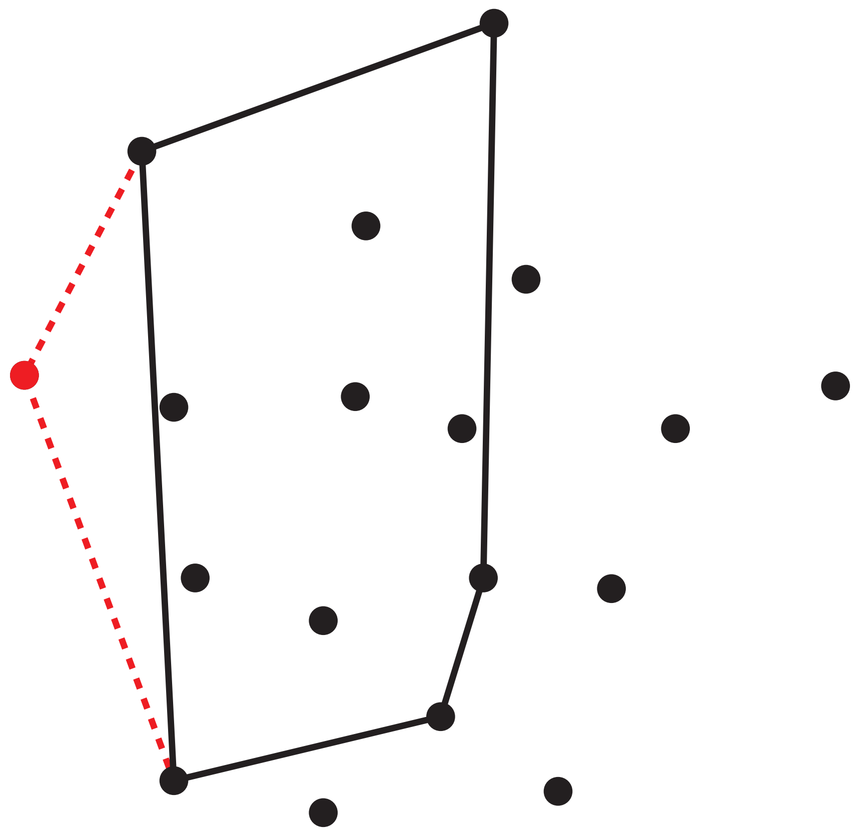
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

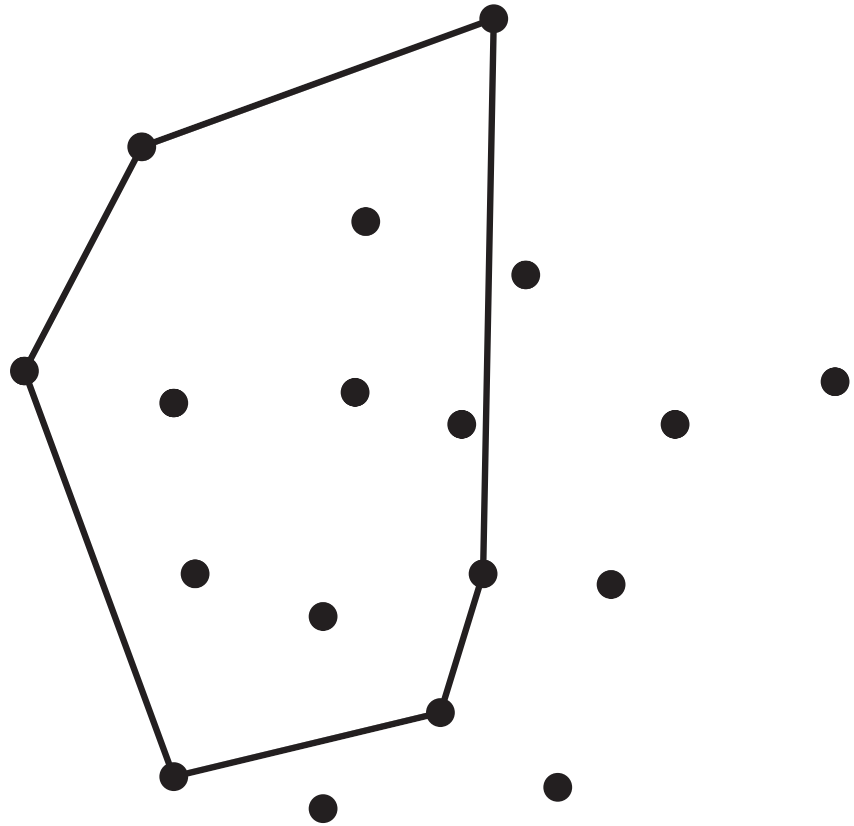
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

  $l = p_1, p_2, p_3$

Advance

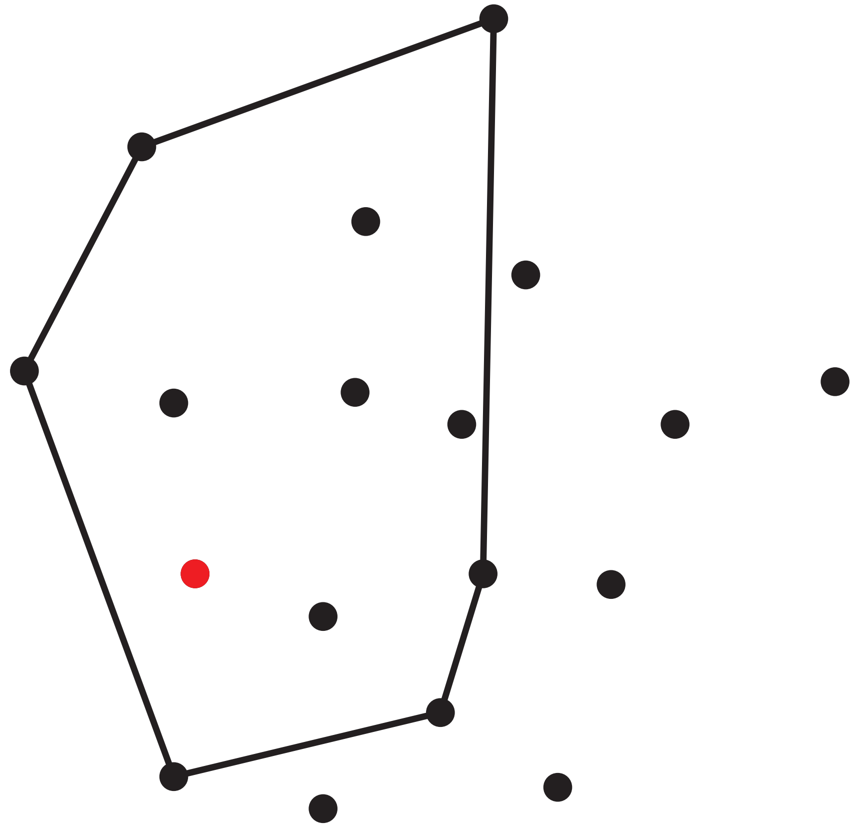
  From $i = 4$ to n, do:

    If p_i lies in the exterior of the polygon defined by l:

      - Compute the points p_l and p_r defining the supporting lines from p_i to the polygon

      - Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

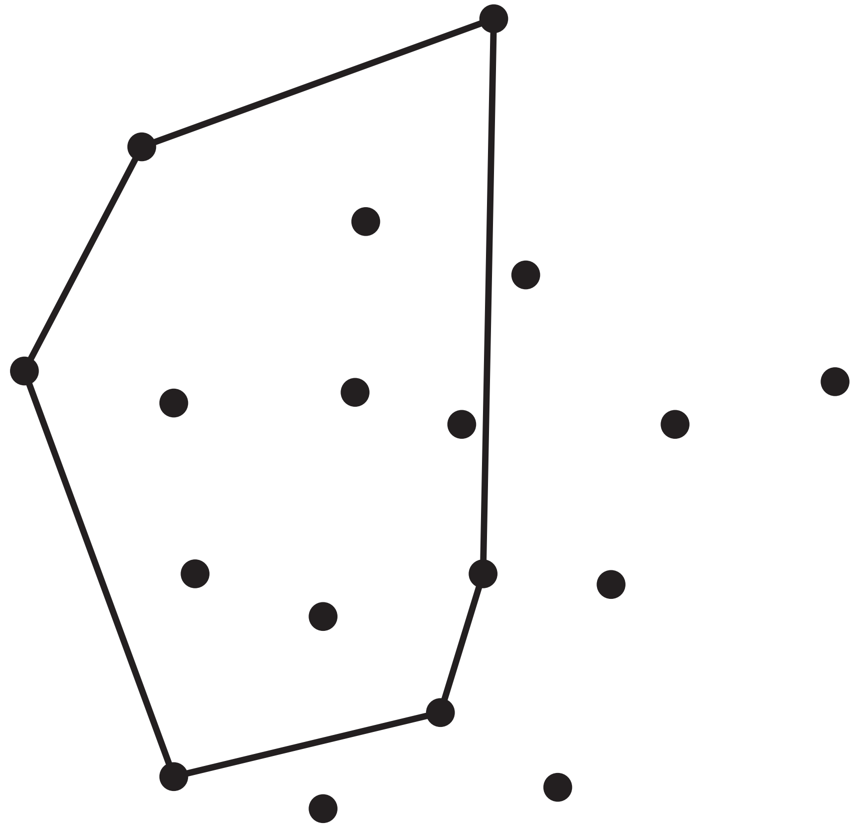
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

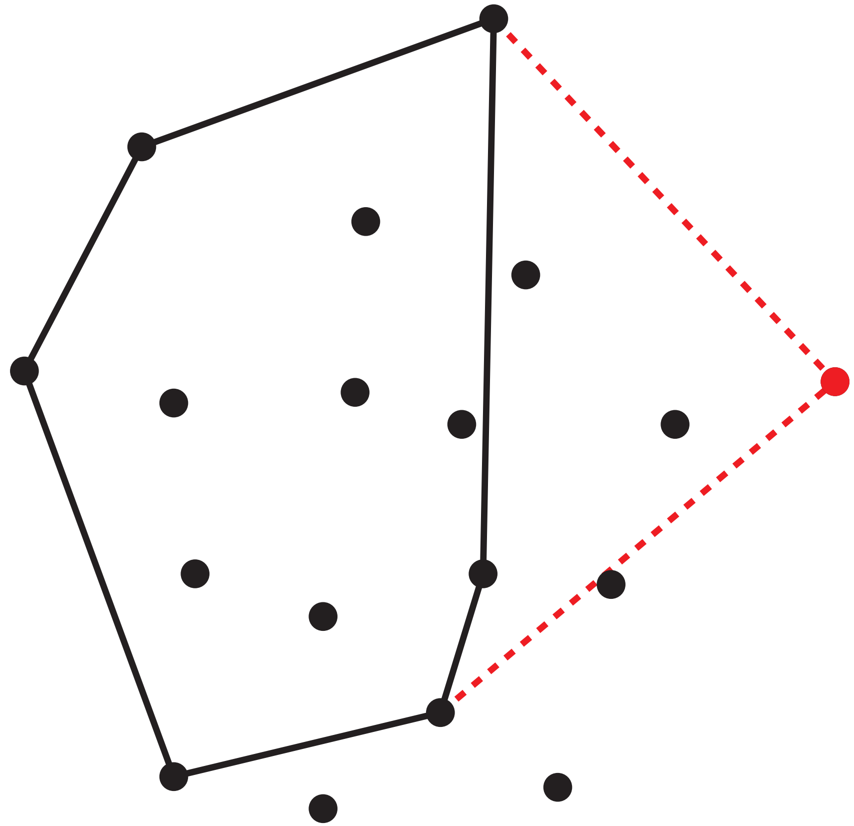
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

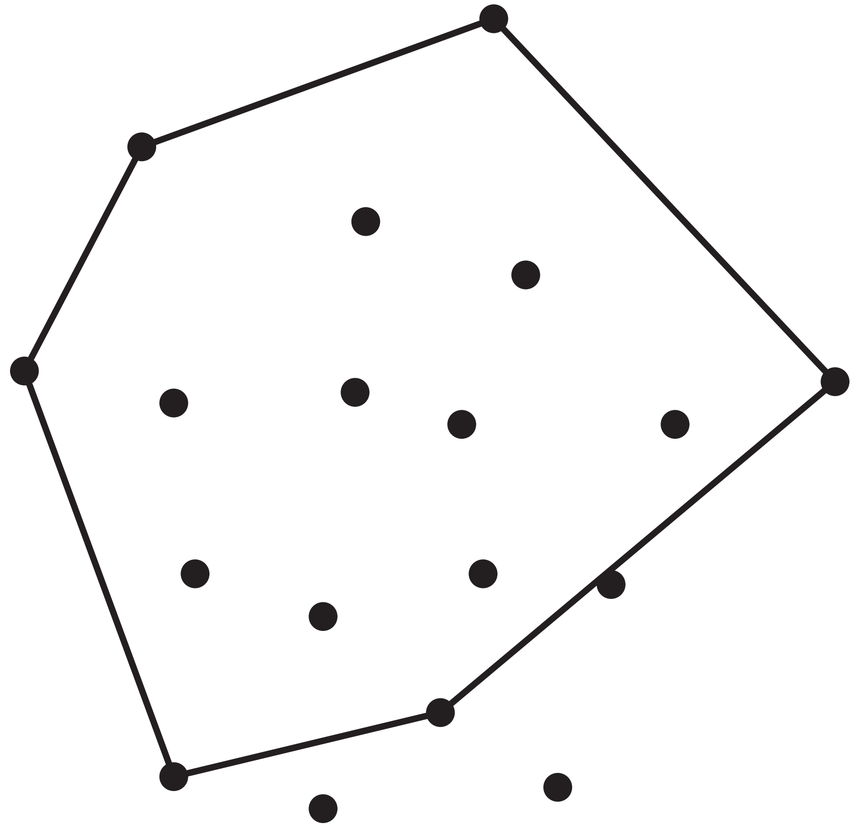
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

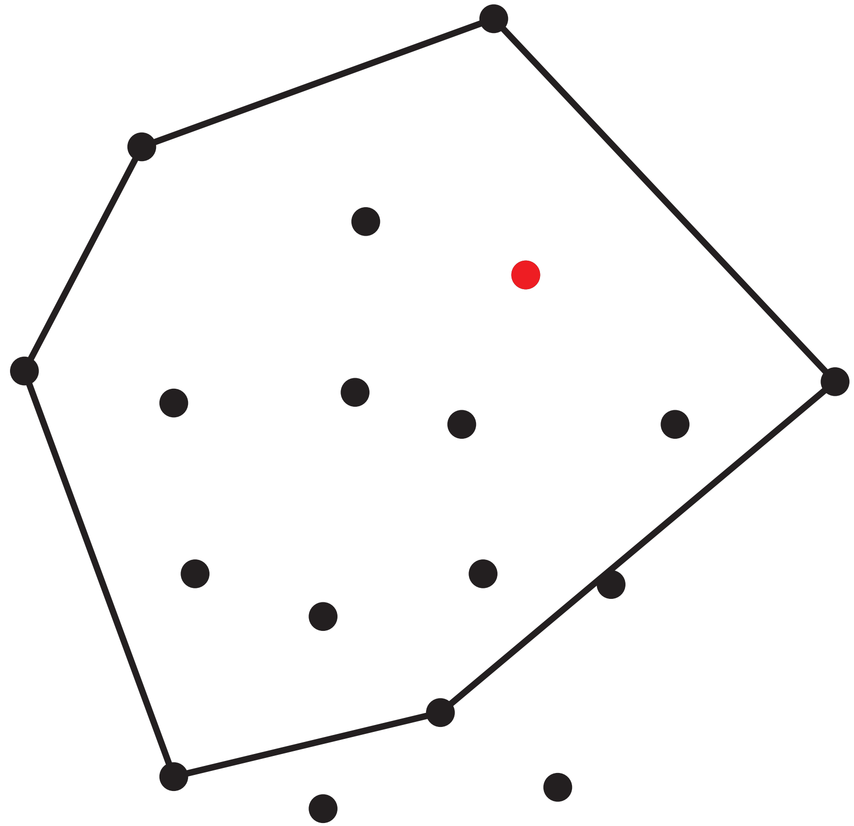
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

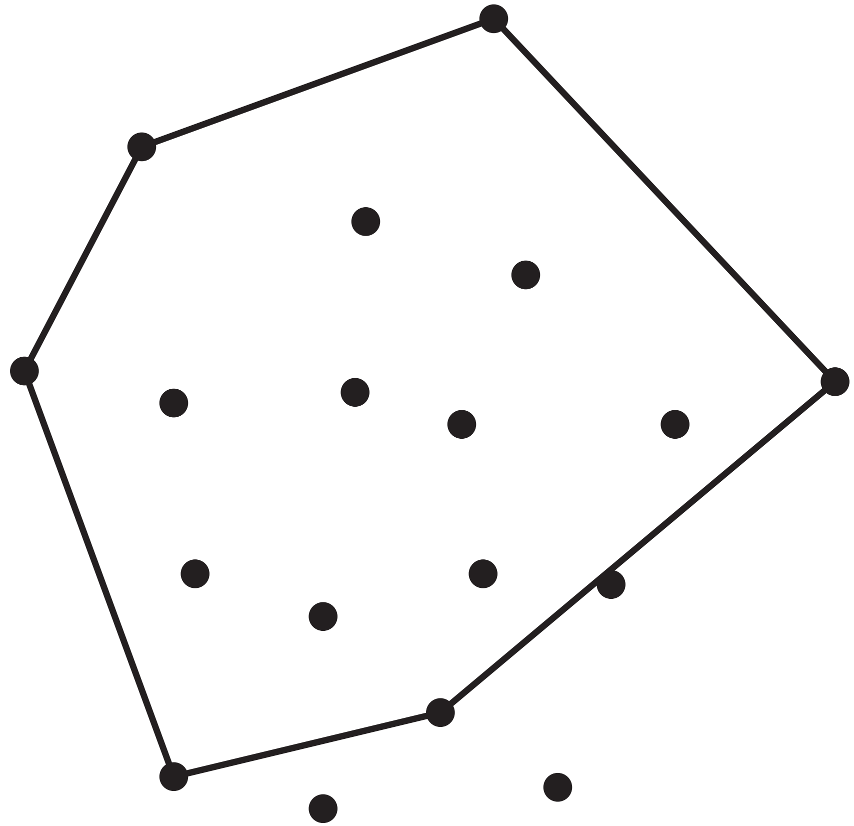
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

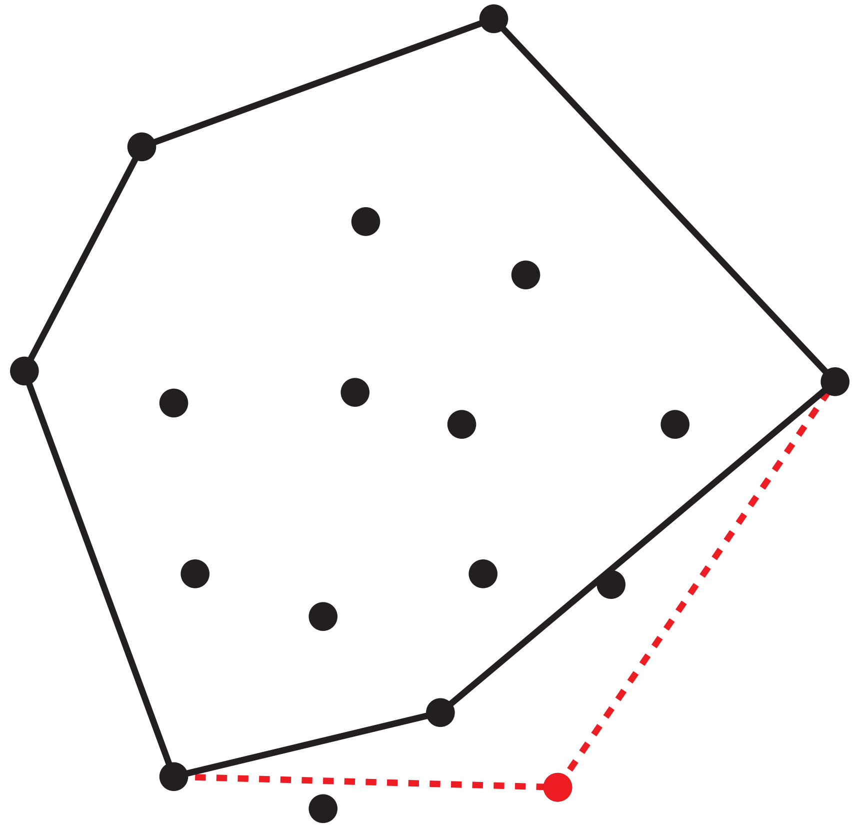
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

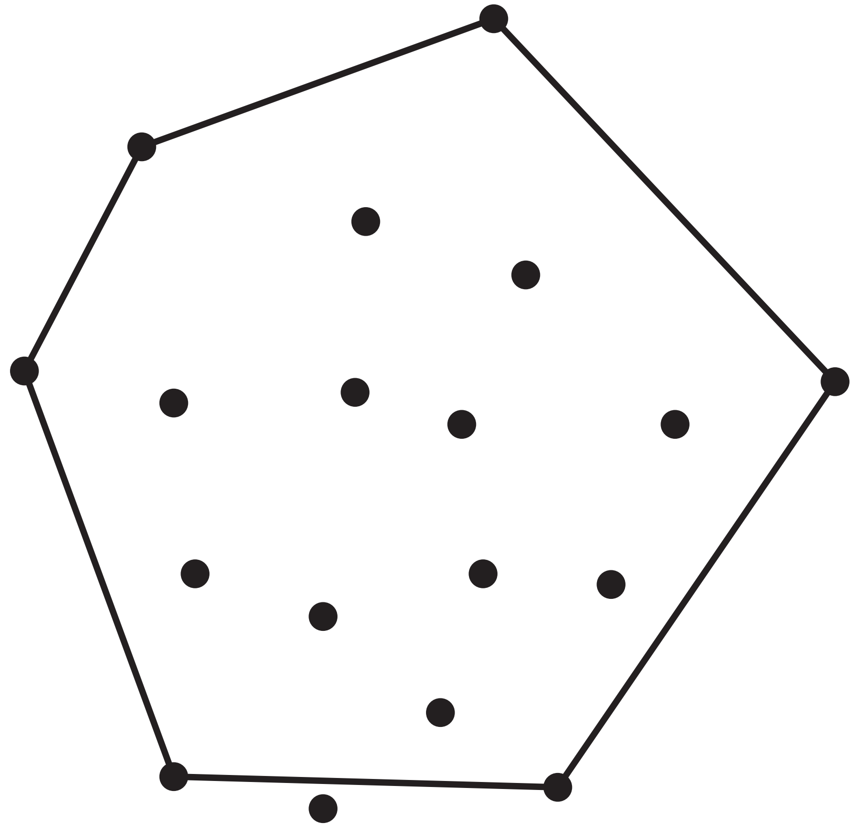
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

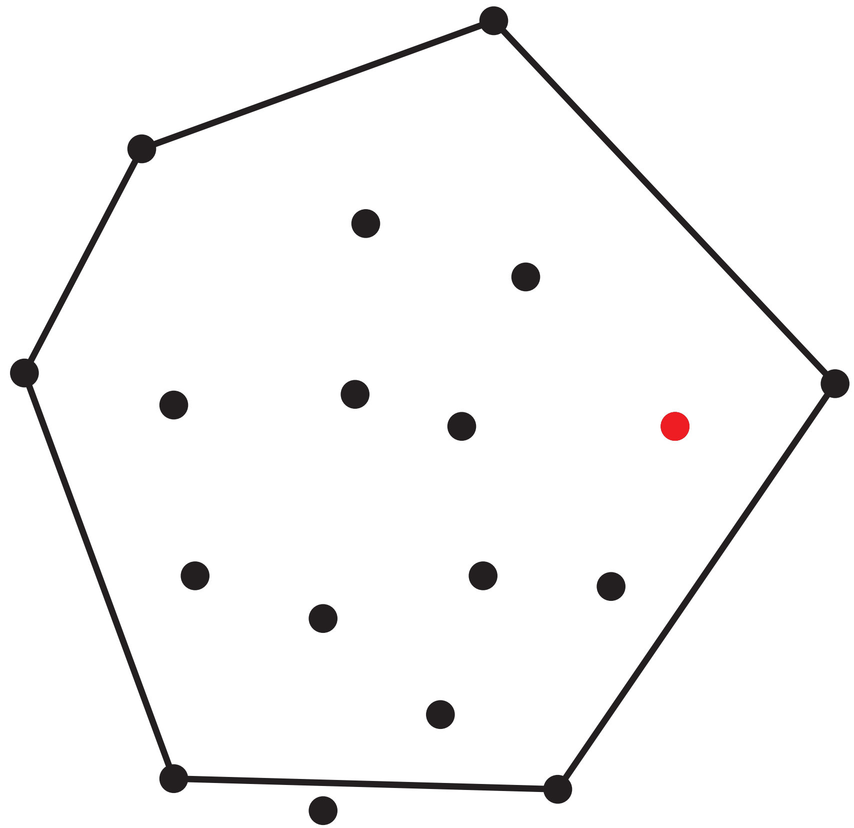
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

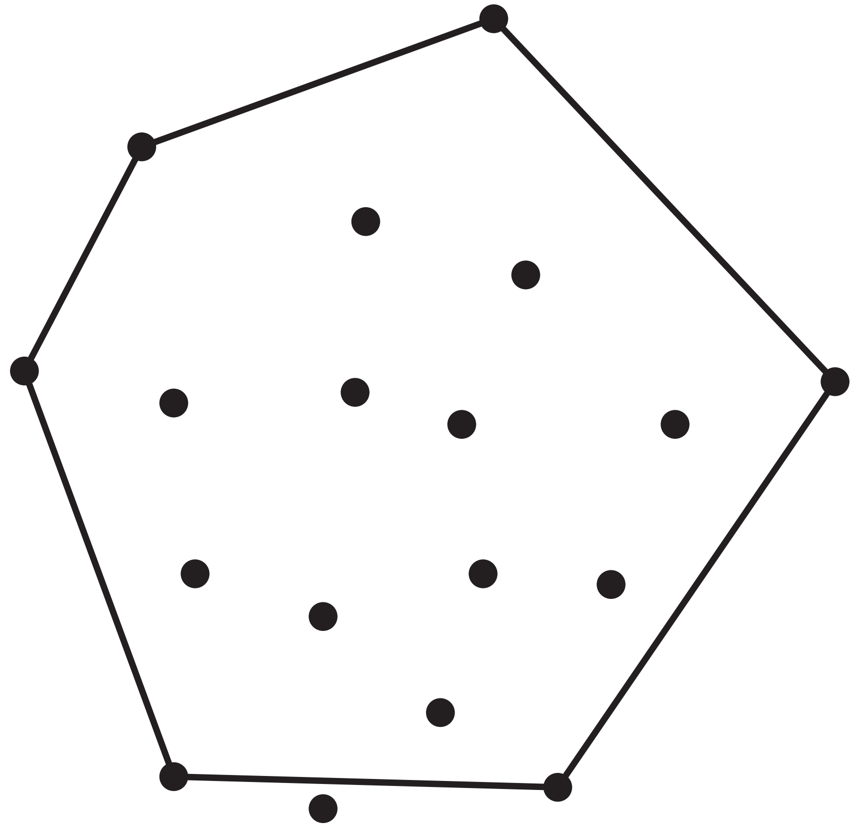
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

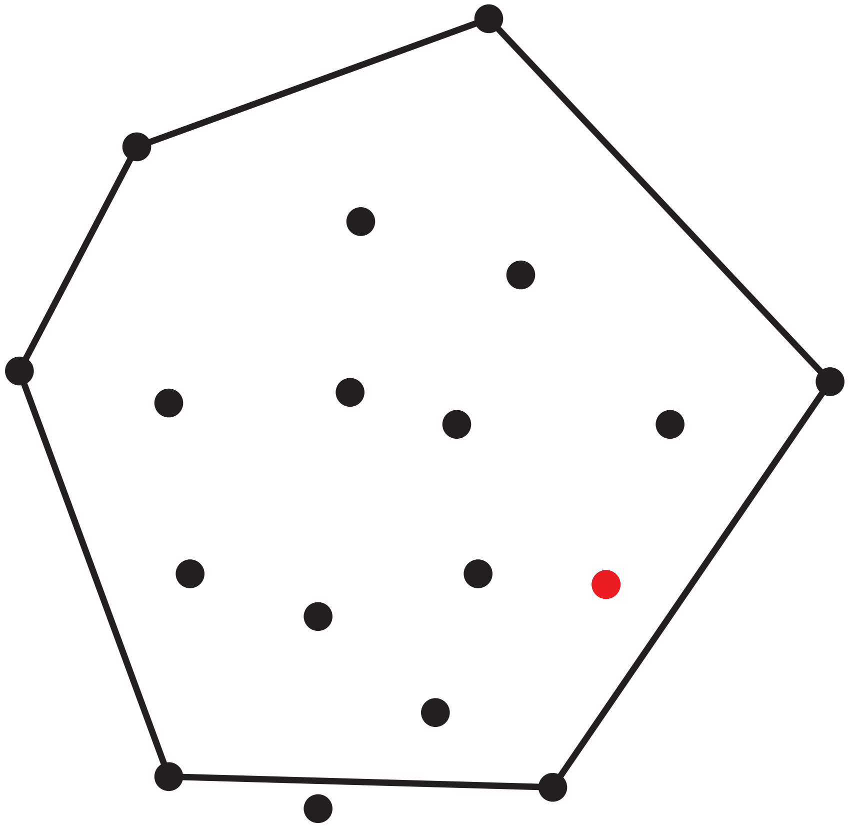
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

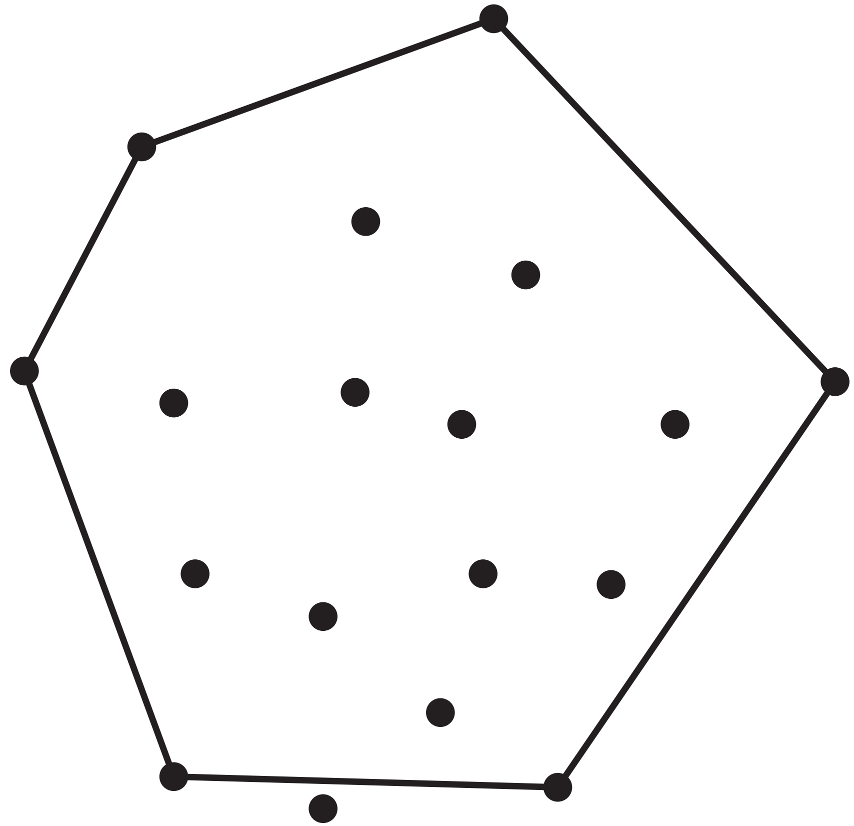
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l

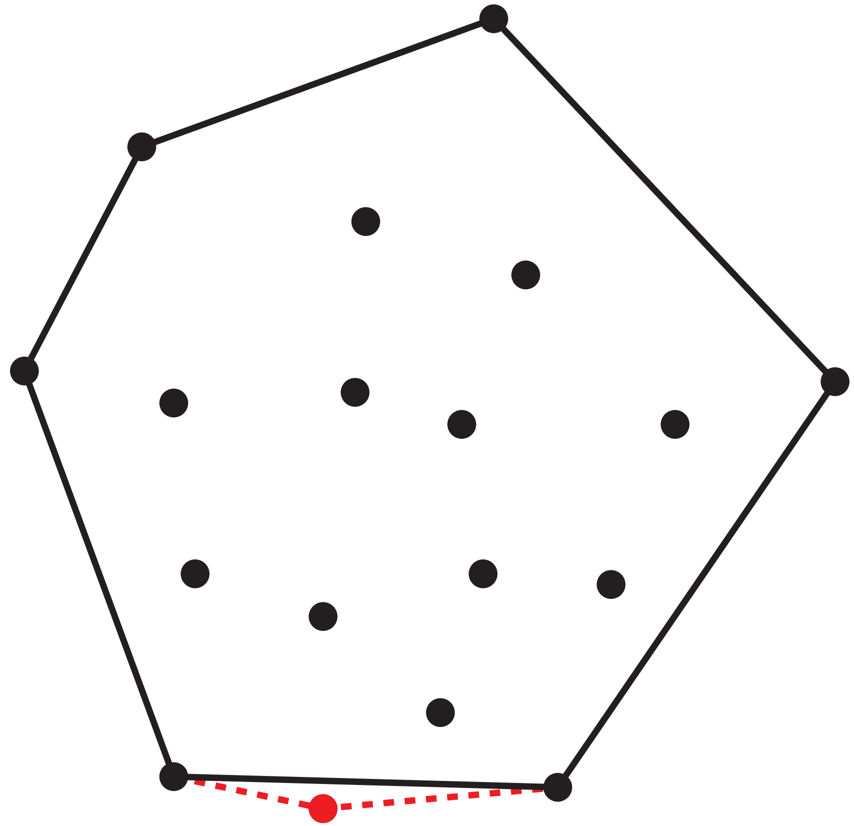


Incremental algorithm

Initialization
$l = p_1, p_2, p_3$

Advance
From $i = 4$ to n, do:
If p_i lies in the exterior of the polygon defined by l:
- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

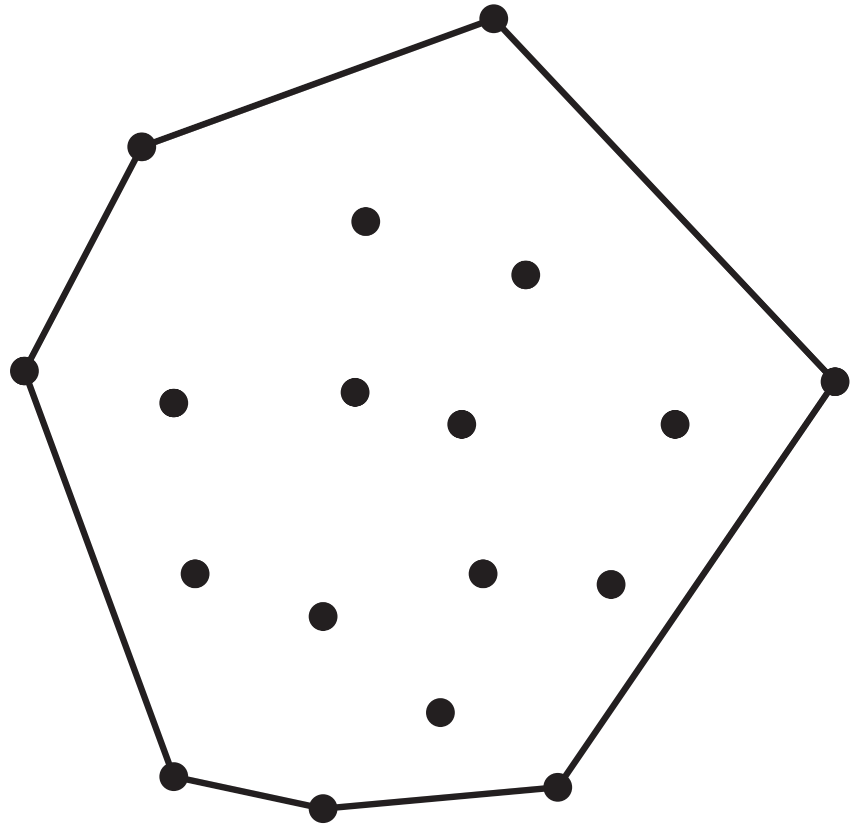
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

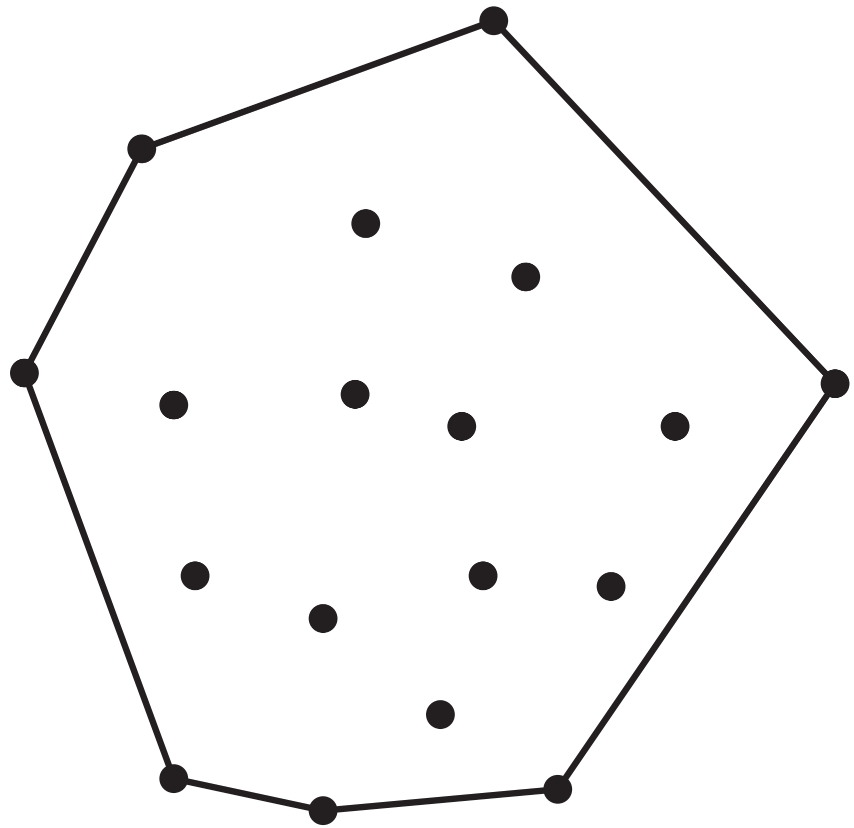
Advance

From $i = 4$ to n, do:

If p_i lies in the exterior of the polygon defined by l:

- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



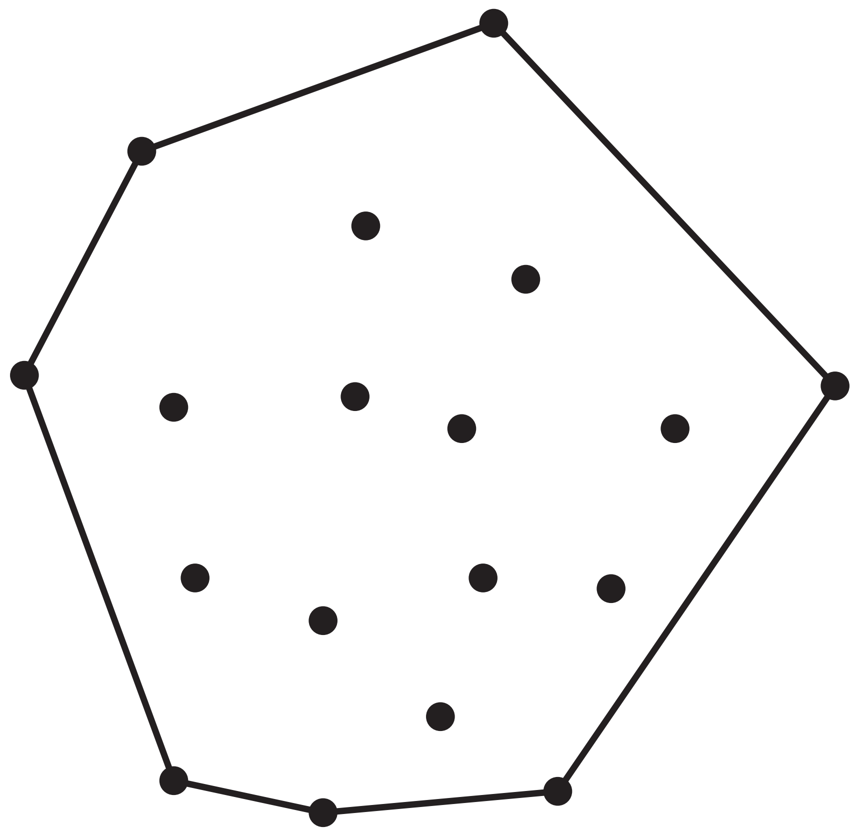
Running time: $O(n \log n)$

Incremental algorithm

Initialization
$l = p_1, p_2, p_3$

Advance
From $i = 4$ to n, do:
If p_i lies in the exterior of the polygon defined by l:
- Compute the points p_l and p_r defining the supporting lines from p_i to the polygon
- Replace the chain $p_l, \ldots, p_r$ in l with the chain p_l, p_i, p_r

Return l



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

By storing l in a structure allowing binary search and updatings (insertions and deletions) in $O(\log n)$ time.

Divide-and-conquer algorithm

Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

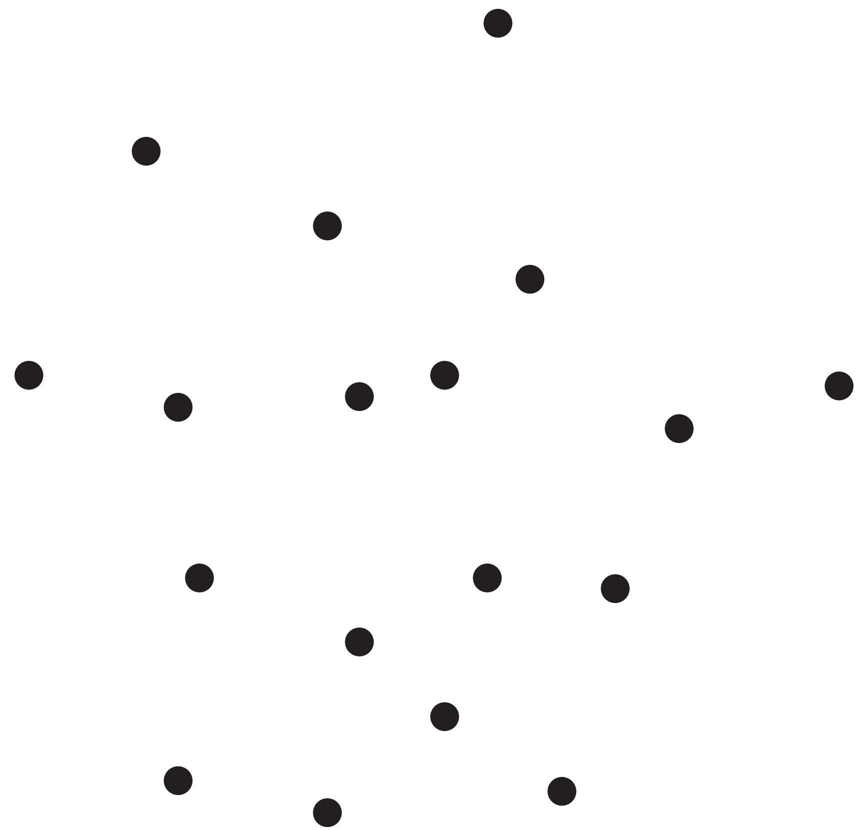
Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae



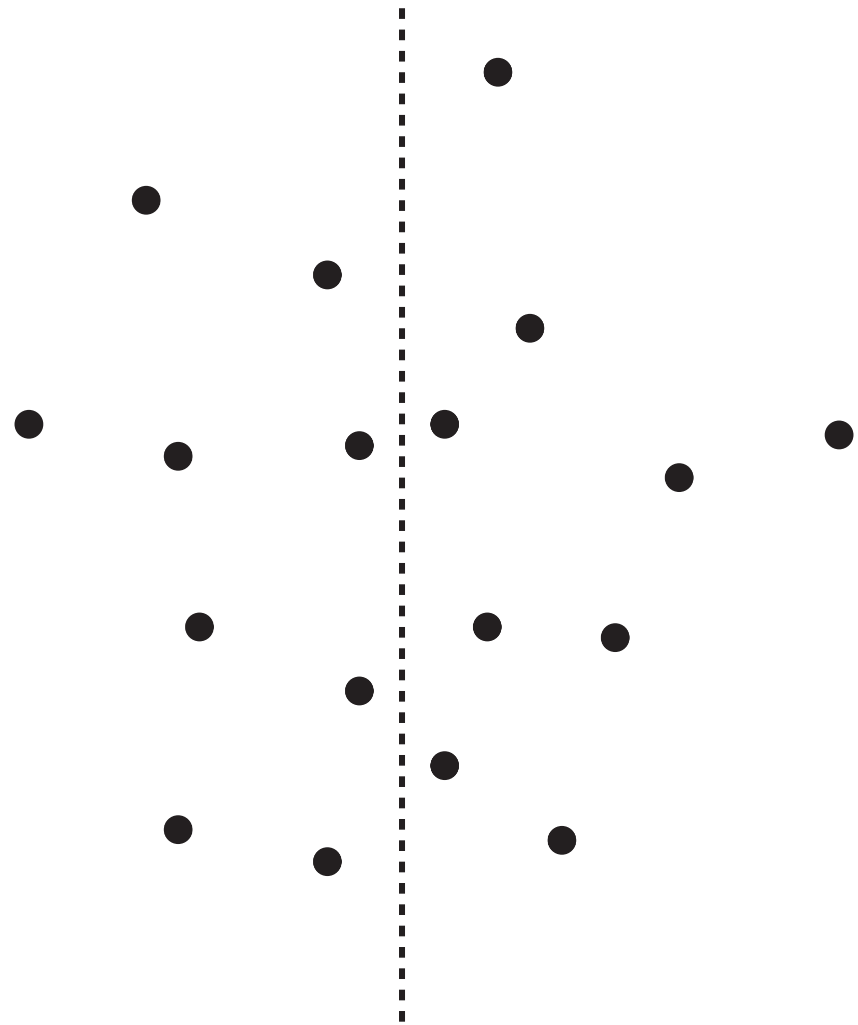
Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae



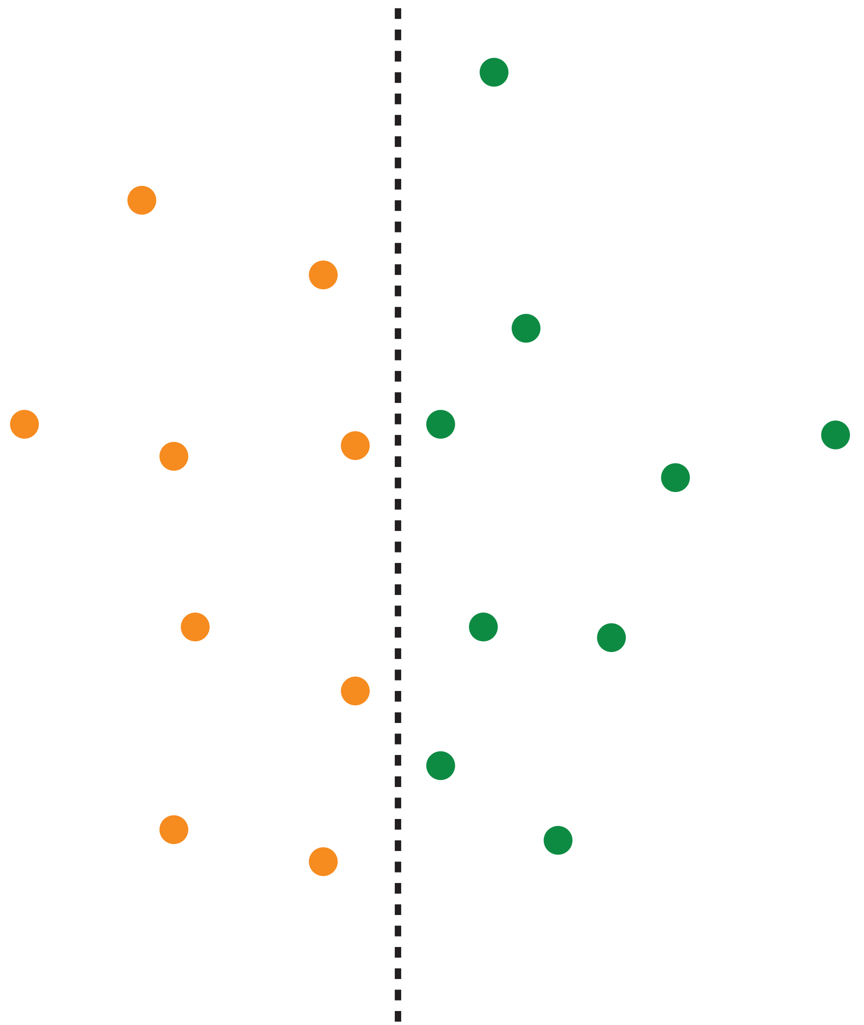
Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae



Divide-and-conquer algorithm

Initialization

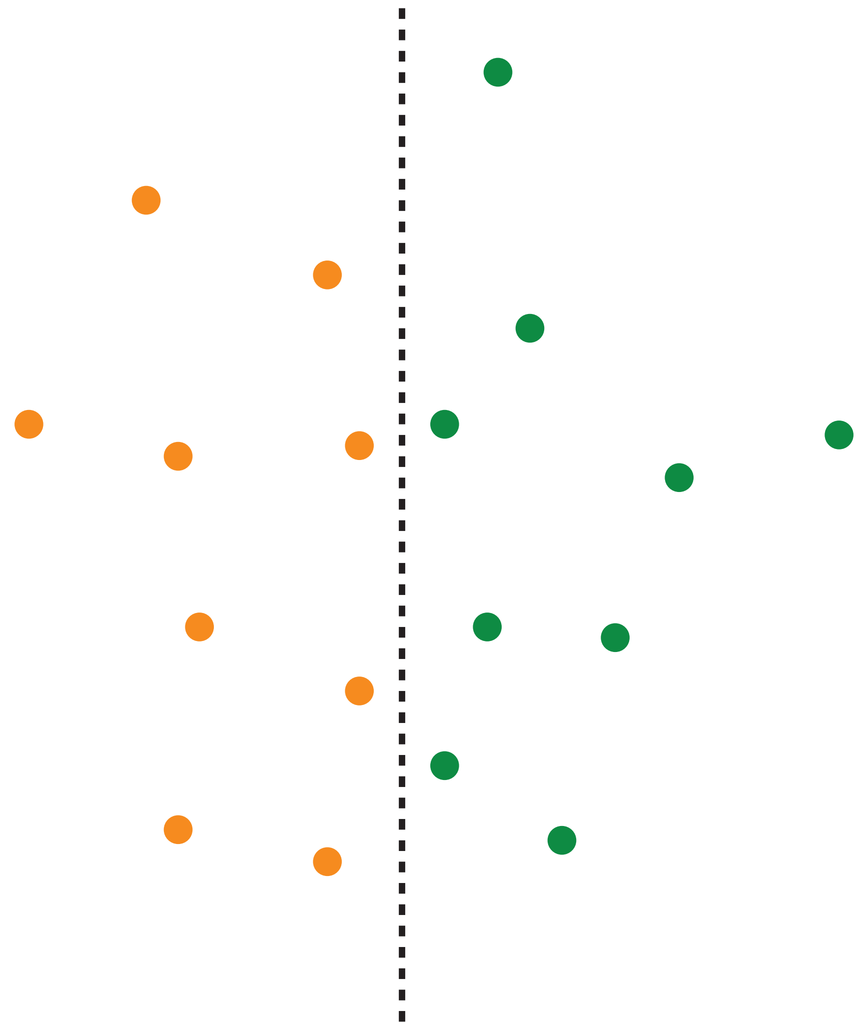
1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

Recursion

1. Recursively compute the convex hull of the two subsets



Divide-and-conquer algorithm

Initialization

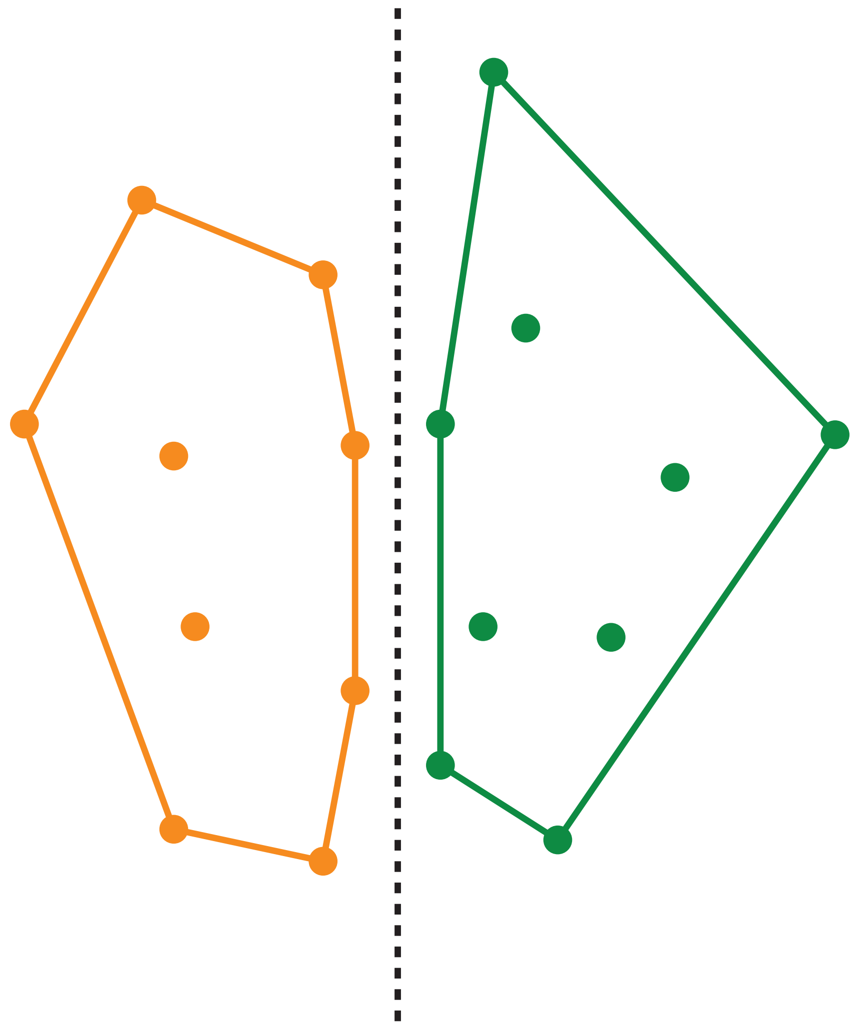
1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

Recursion

1. Recursively compute the convex hull of the two subsets



Divide-and-conquer algorithm

Initialization

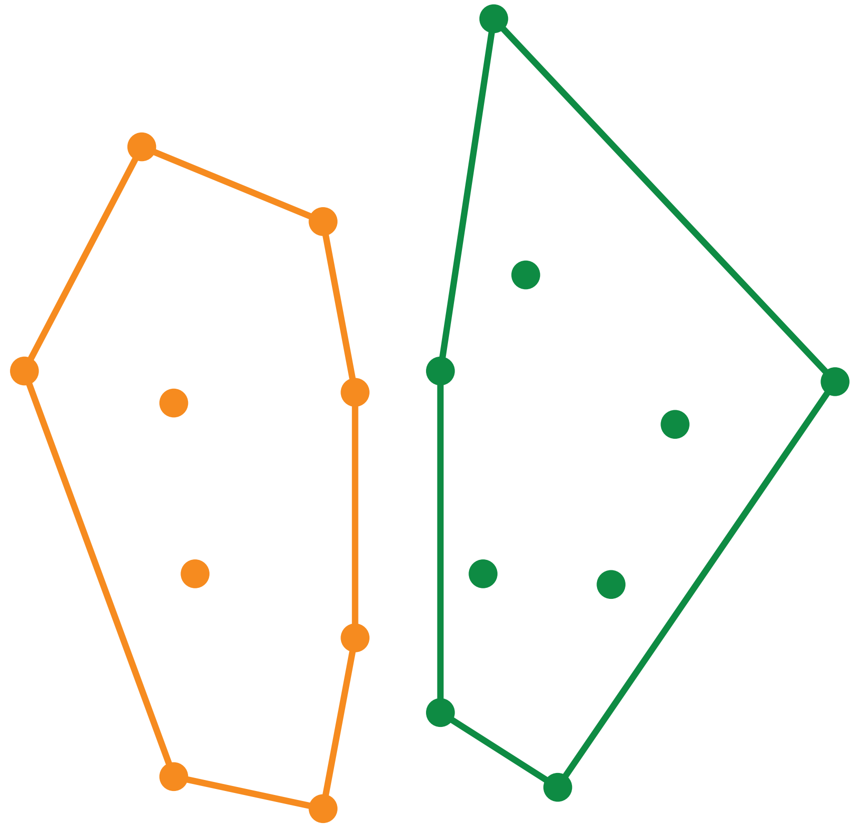
1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

Recursion

1. Recursively compute the convex hull of the two subsets



CONVEX HULL IN 2D

Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

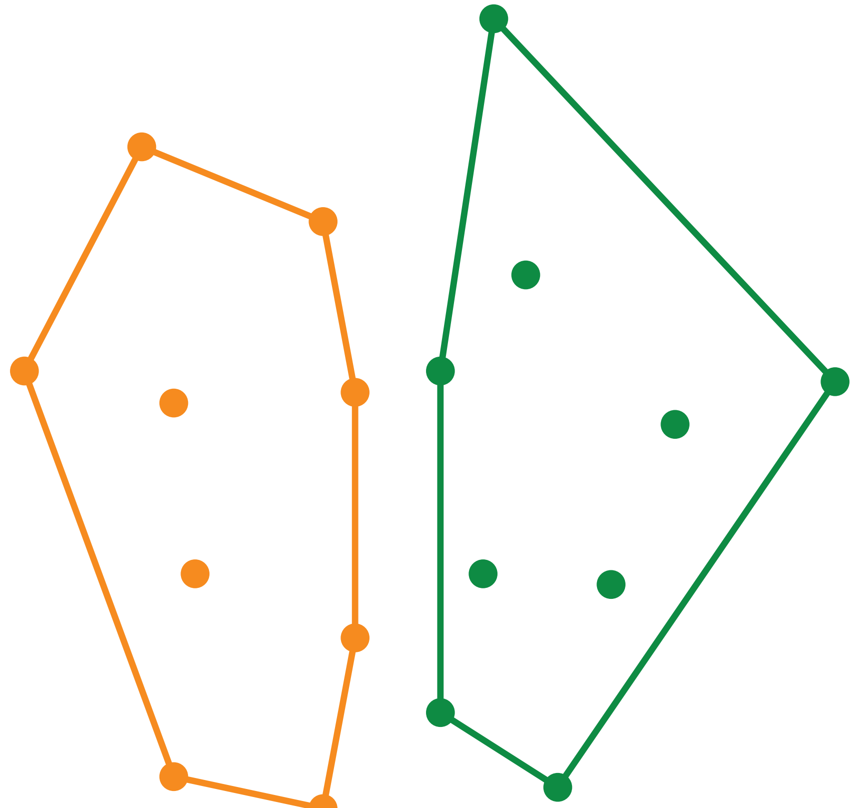
1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

Recursion

1. Recursively compute the convex hull of the two subsets

Merge

1. Compute the external common tangents of the two convex polygons
2. Delete the interior chains of the two polygons and join the external chains through the supporting segments



Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

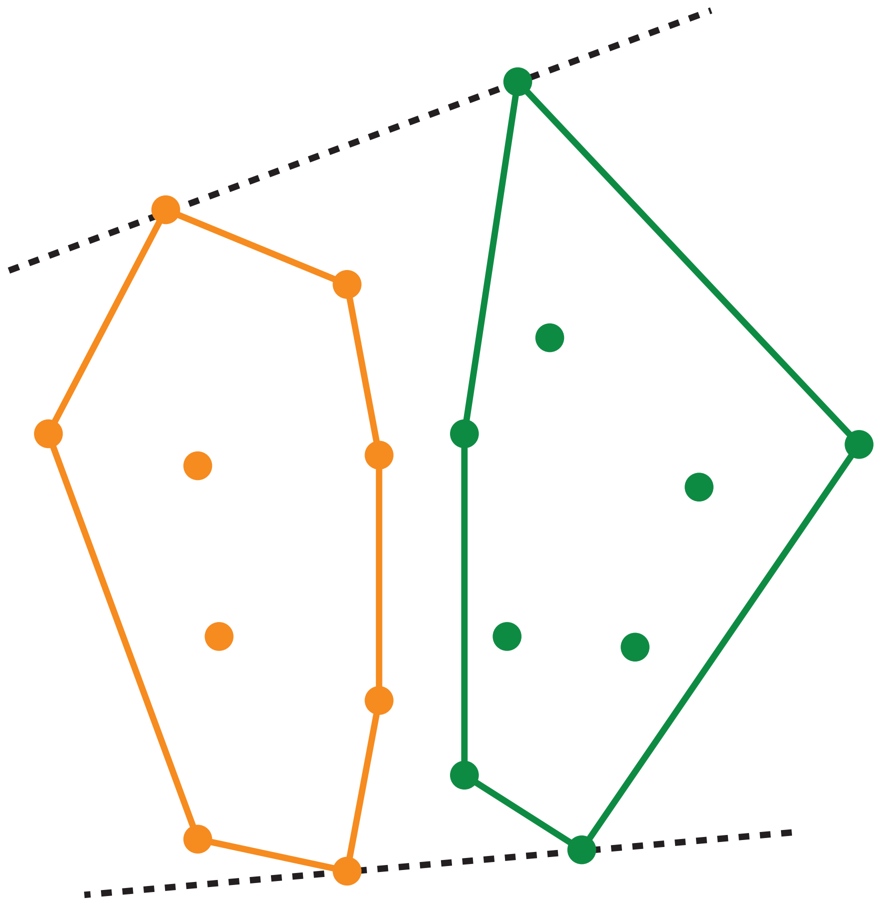
Recursion

1. Recursively compute the convex hull of the two subsets

Merge

1. Compute the external common tangents of the two convex polygons

2. Delete the interior chains of the two polygons and join the external chains through the supporting segments



CONVEX HULL IN 2D

Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

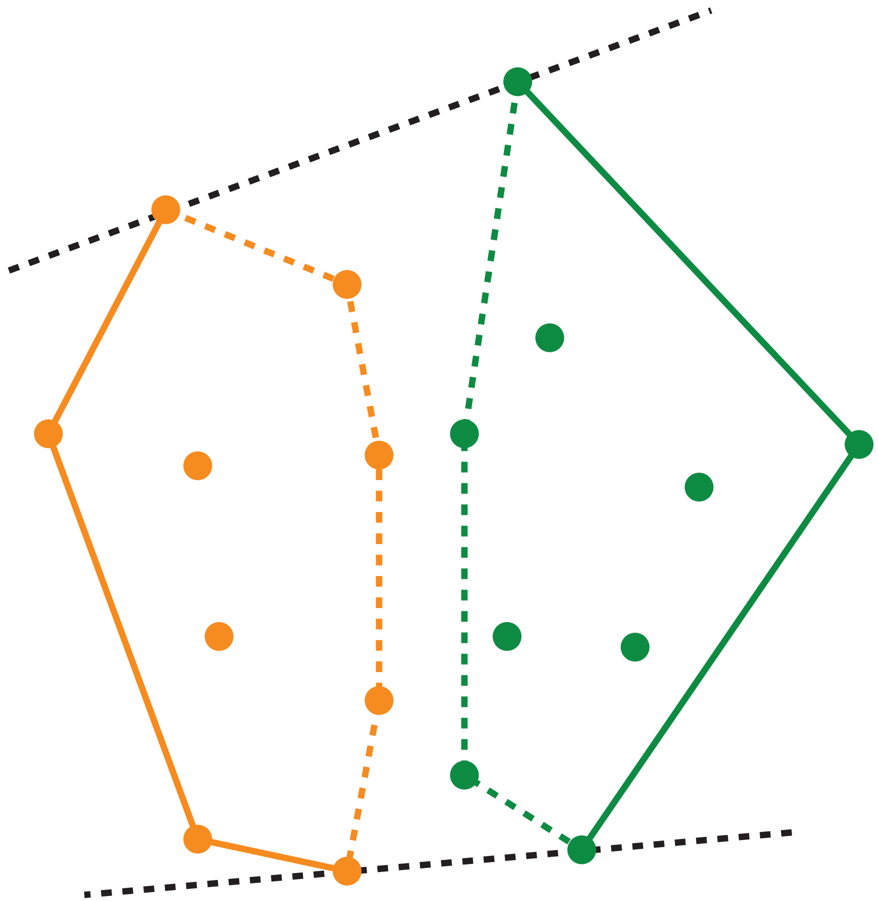
1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

Recursion

1. Recursively compute the convex hull of the two subsets

Merge

1. Compute the external common tangents of the two convex polygons
2. Delete the interior chains of the two polygons and join the external chains through the supporting segments



Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

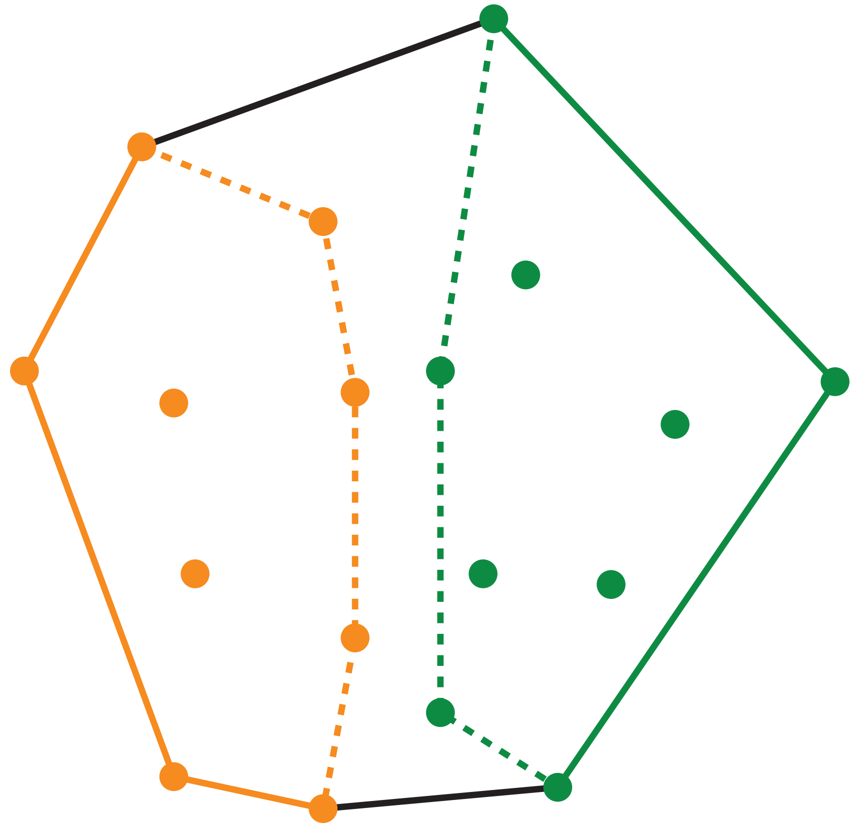
Recursion

1. Recursively compute the convex hull of the two subsets

Merge

1. Compute the external common tangents of the two convex polygons

2. Delete the interior chains of the two polygons and join the external chains through the supporting segments



CONVEX HULL IN 2D

Divide-and-conquer algorithm

Initialization

1. Sort the points by abscissae

Division

1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

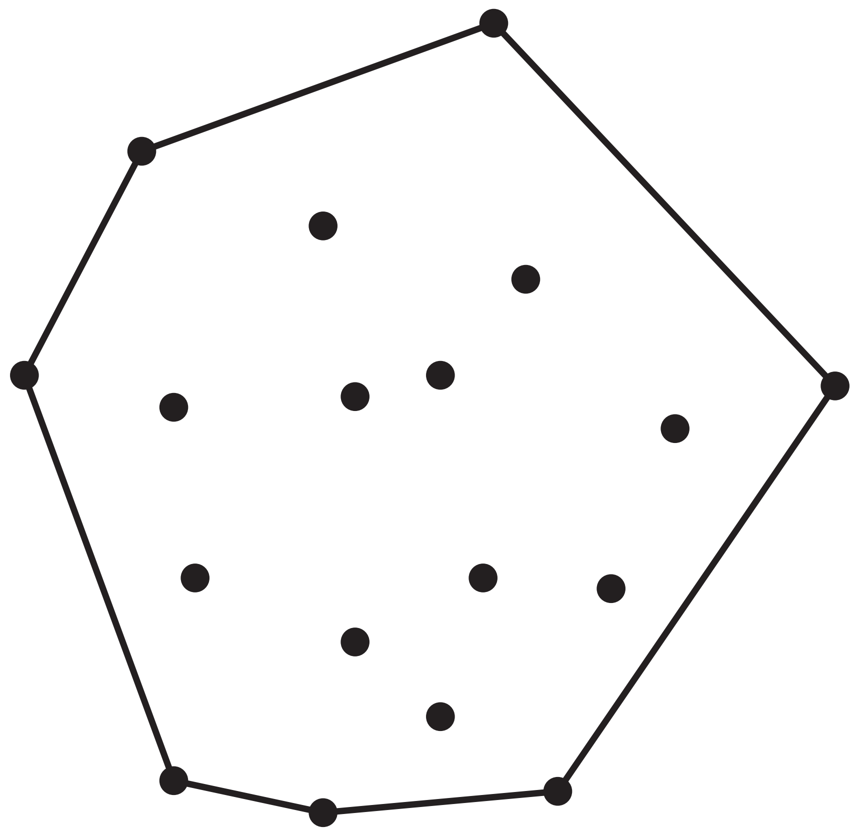
Recursion

1. Recursively compute the convex hull of the two subsets

Merge

1. Compute the external common tangents of the two convex polygons

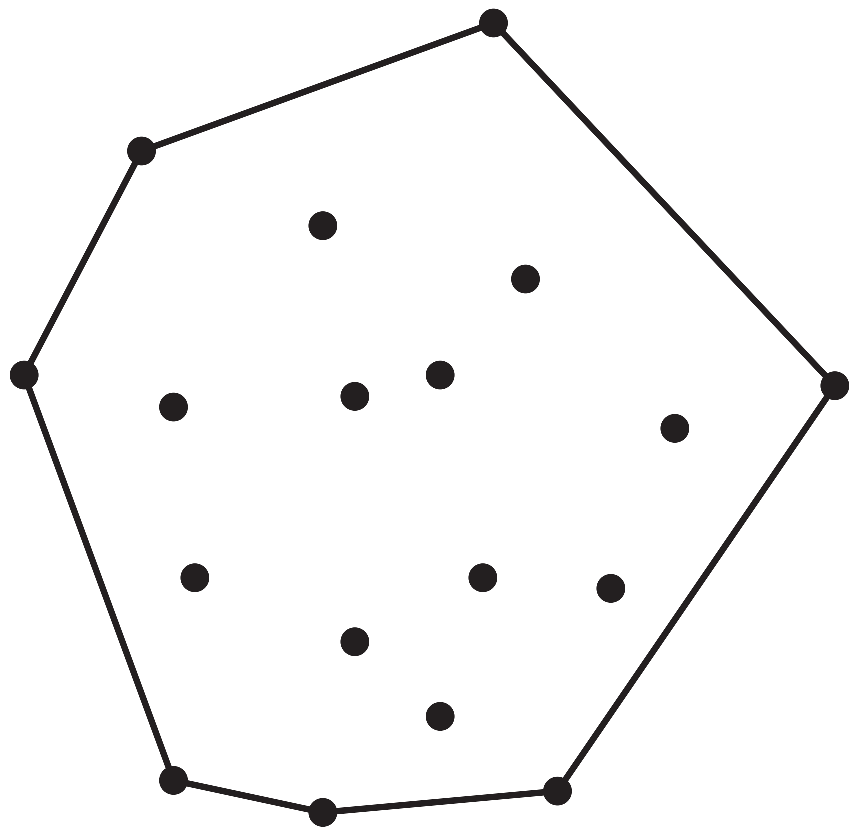
2. Delete the interior chains of the two polygons and join the external chains through the supporting segments



Divide-and-conquer algorithm

Running time

Initialization: $O(n \log n)$ (only once)



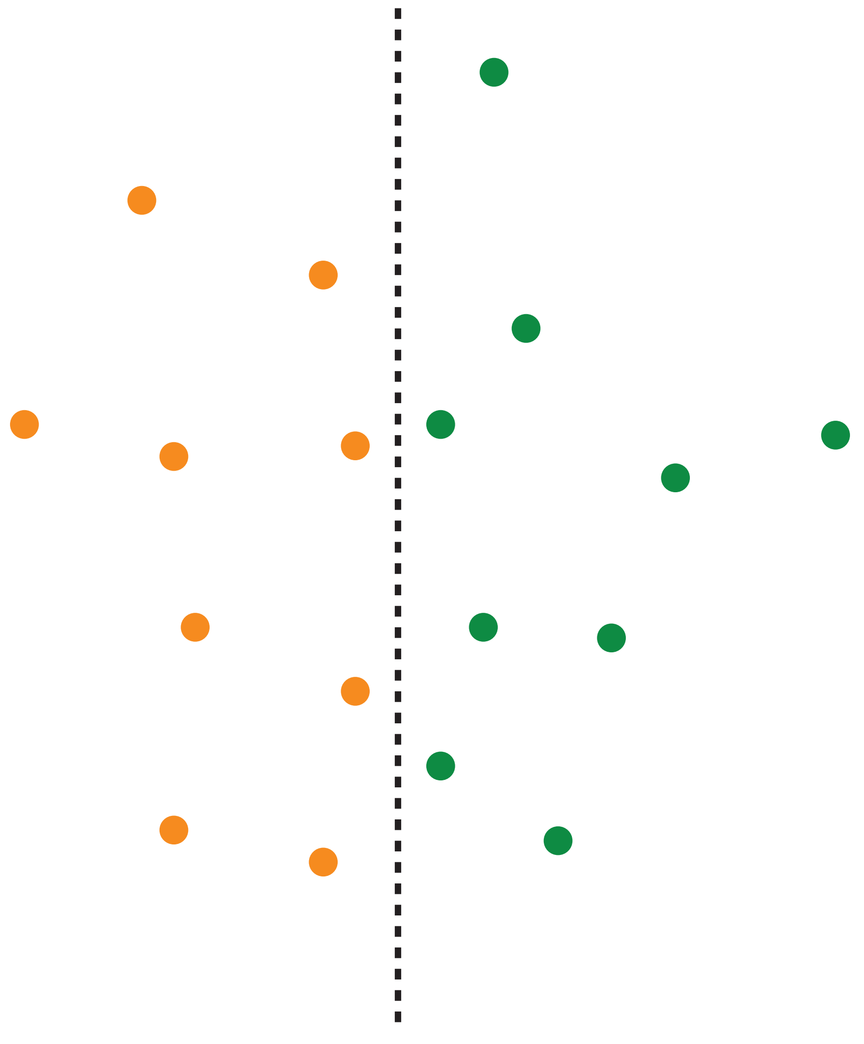
CONVEX HULL IN 2D

Divide-and-conquer algorithm

Running time

Initialization: $O(n \log n)$ (only once)

Division: $O(n)$



CONVEX HULL IN 2D

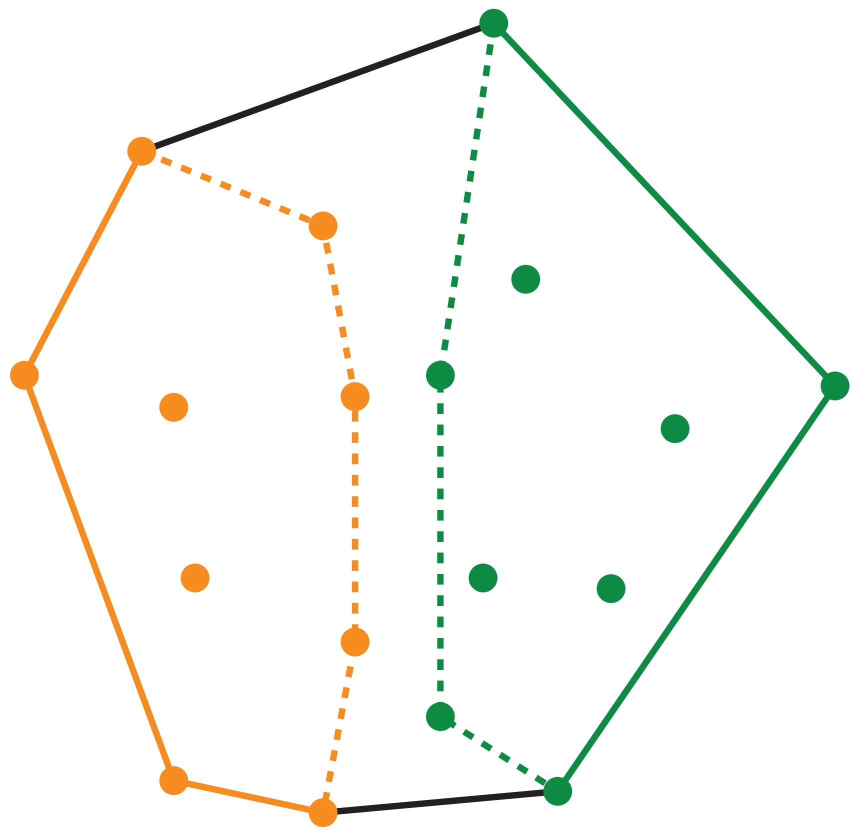
Divide-and-conquer algorithm

Running time

Initialization: $O(n \log n)$ (only once)

Division: $O(n)$

Merge: $O(n)$



Divide-and-conquer algorithm

Running time

Initialization: $O(n \log n)$ (only once)

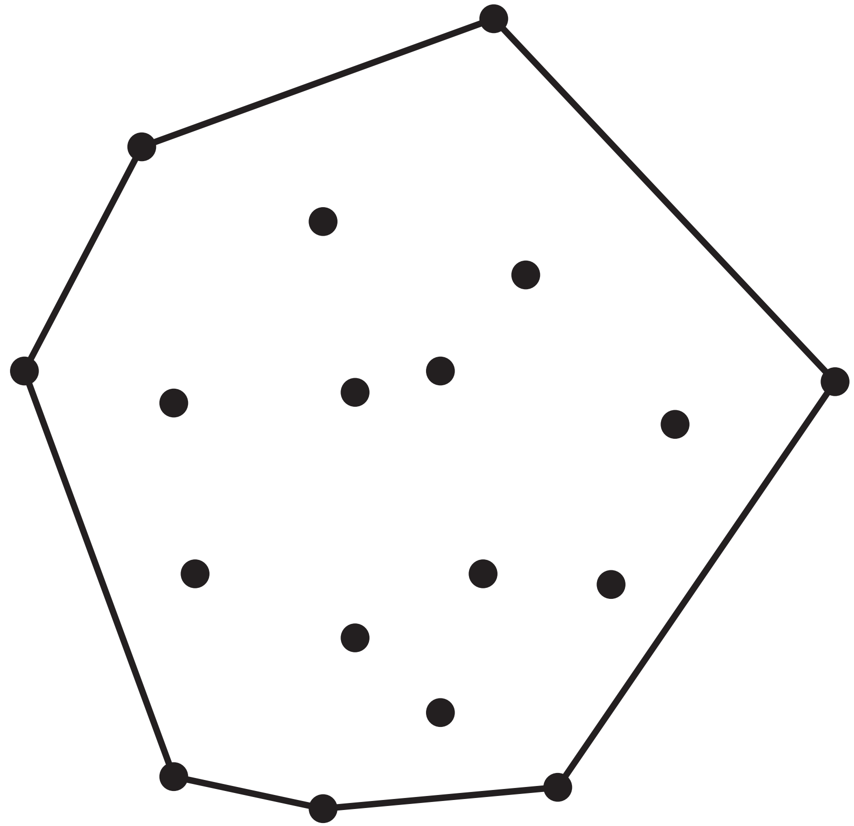
Division: $O(n)$

Merge: $O(n)$

Advance:

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

Overall: $O(n \log n)$



Lower bound

Lower bound

Input: n **real numbers**

$x_1, \ldots, x_n$ real numbers

Lower bound

Input: n real numbers

$x_1, \ldots, x_n$ real numbers

CONVEX HULL IN 2D

Lower bound

Input: n **real numbers**

$x_1, \ldots, x_n$ real numbers

Input: n **points**

$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$

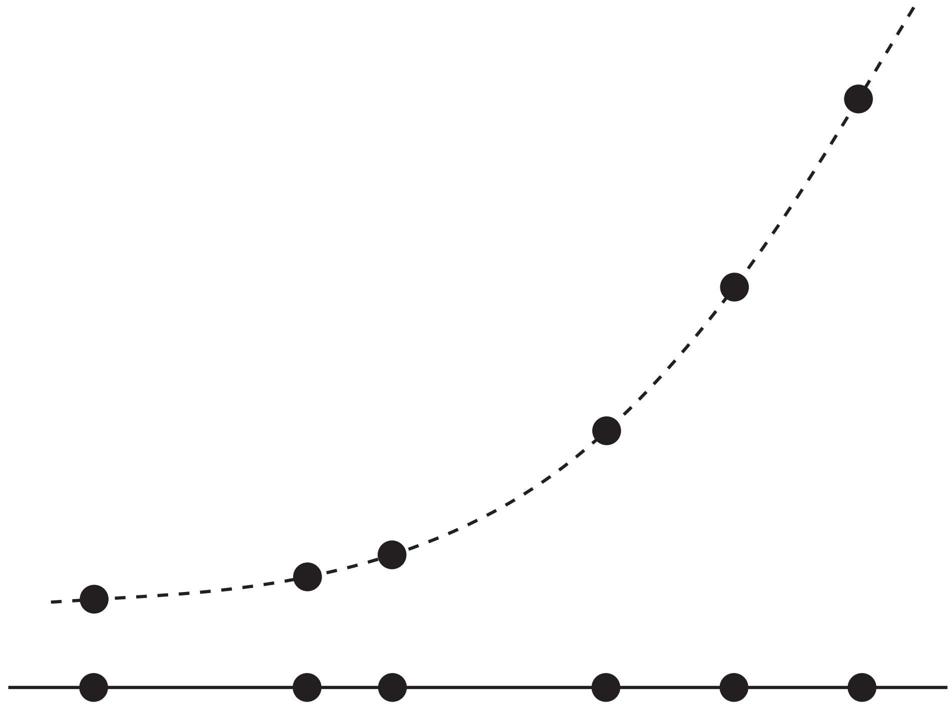
Lower bound

Input: n **real numbers**

$x_1, \ldots, x_n$ real numbers

Input: n **points**

$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$



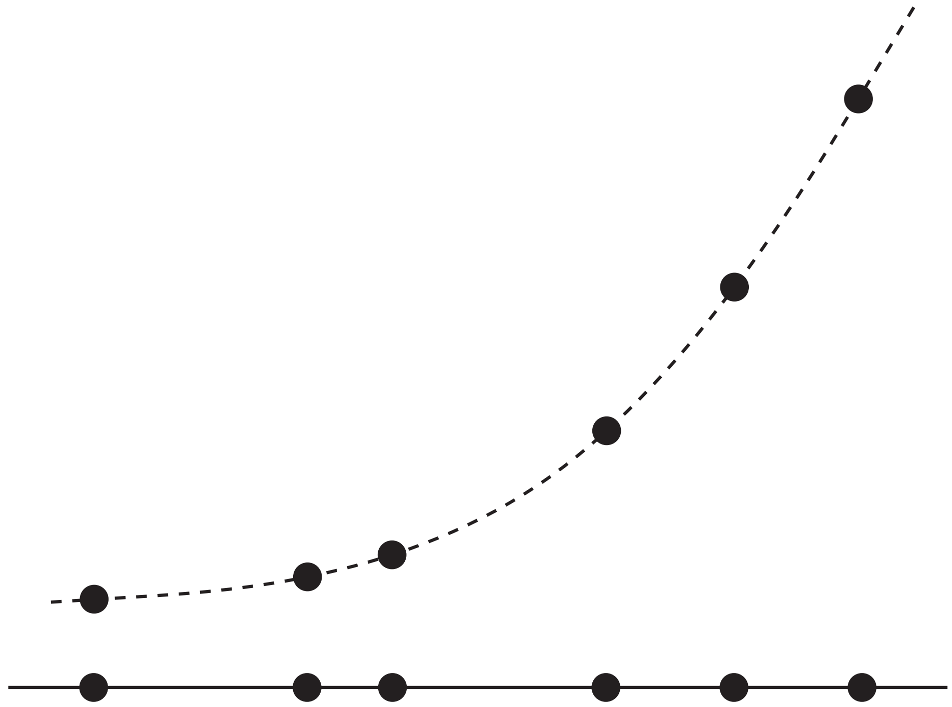
CONVEX HULL IN 2D

Lower bound

Input: n **real numbers**
$x_1, \ldots, x_n$ real numbers

Input: n **points**
$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$

Output: convex hull of the points
Sorted list of the vertices of the convex hull



CONVEX HULL IN 2D

Lower bound

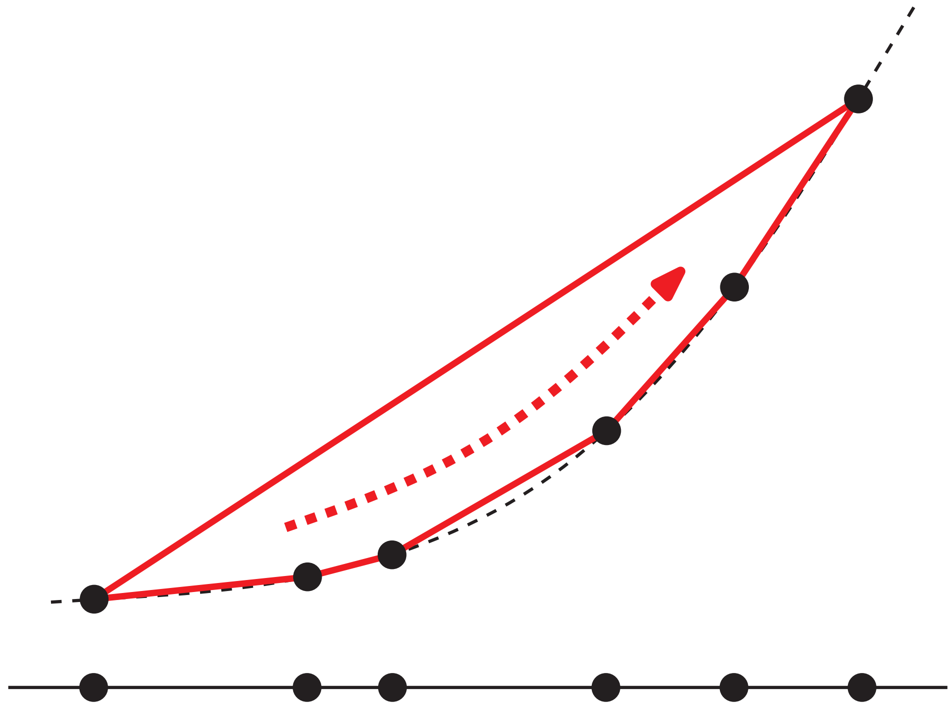
Input: n real numbers
$x_1, \ldots, x_n$ real numbers

↓

Input: n points
$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$

↓

Output: convex hull of the points
Sorted list of the vertices of the convex hull



CONVEX HULL IN 2D

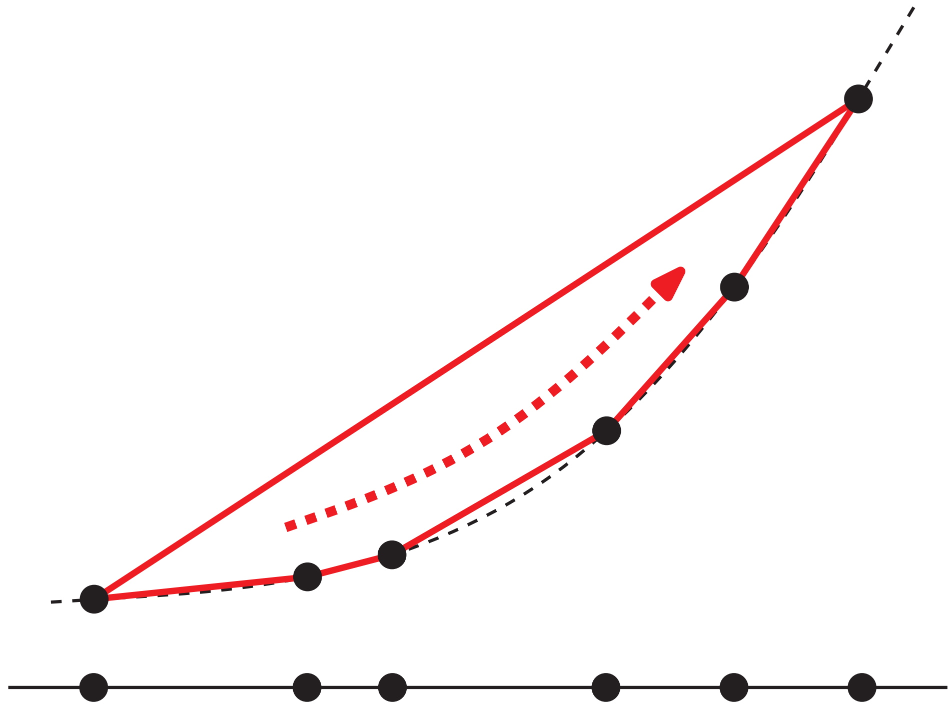
Lower bound

Input: n **real numbers**
$x_1, \ldots, x_n$ real numbers

Input: n **points**
$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$

Output: convex hull of the points
Sorted list of the vertices of the convex hull

Output: sorting the numbers
Sorted list of the numbers $x_1, \ldots, x_n$



CONVEX HULL IN 2D

Lower bound

Input: n **real numbers**
$x_1, \ldots, x_n$ real numbers

↓

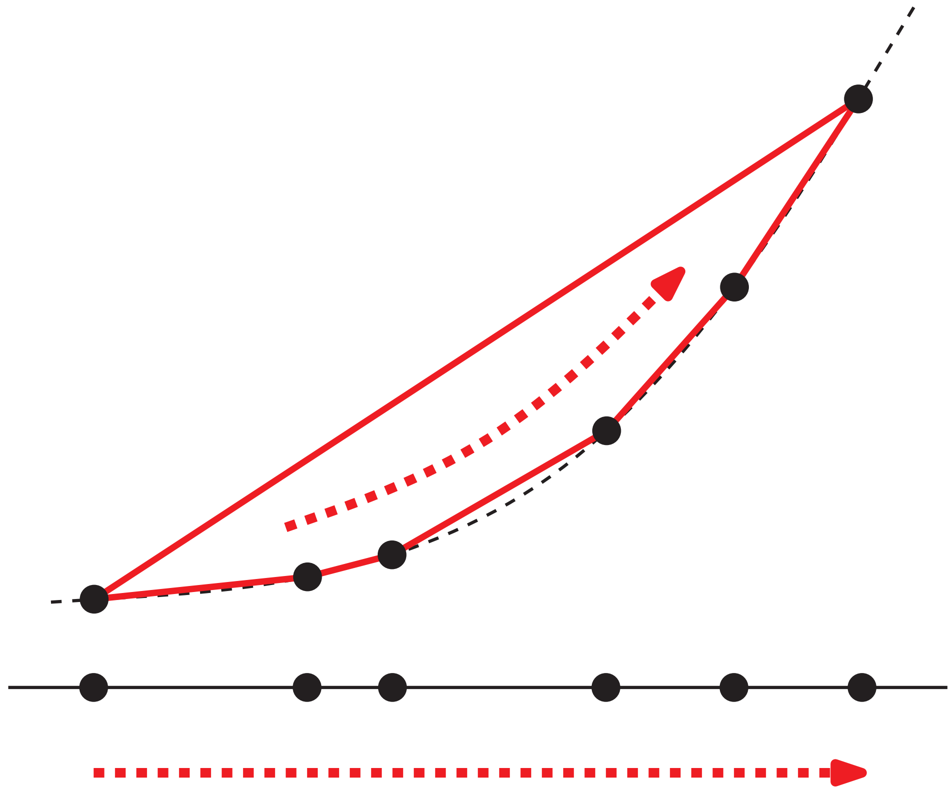
Input: n **points**
$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$

↓

Output: convex hull of the points
Sorted list of the vertices of the convex hull

↓

Output: sorting the numbers
Sorted list of the numbers $x_1, \ldots, x_n$



CONVEX HULL IN 2D

Lower bound

Input: n **real numbers**
$x_1, \dots, x_n$ real numbers

↓ $O(n)$

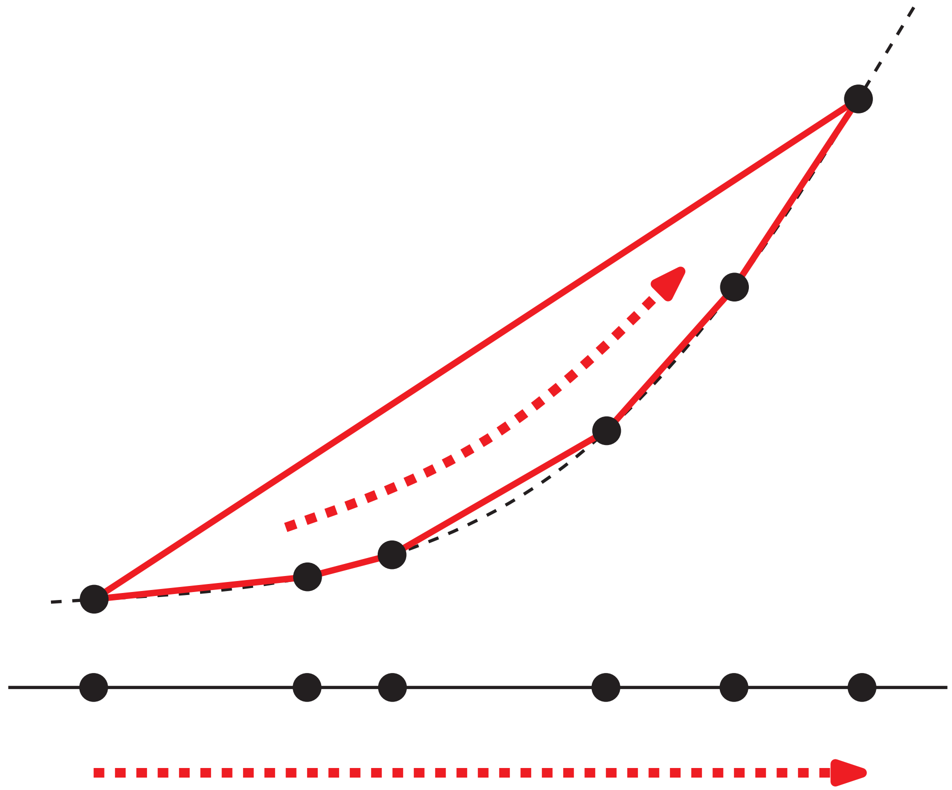
Input: n **points**
$p_1, \dots, p_n$, with $p_i = (x_i, x_i^2)$

↓

Output: convex hull of the points
Sorted list of the vertices of the convex hull

↓ $O(n)$

Output: sorting the numbers
Sorted list of the numbers $x_1, \dots, x_n$



CONVEX HULL IN 2D

Lower bound

Input: n real numbers
$x_1, \ldots, x_n$ real numbers

$\downarrow$ $O(n)$

Input: n points
$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$

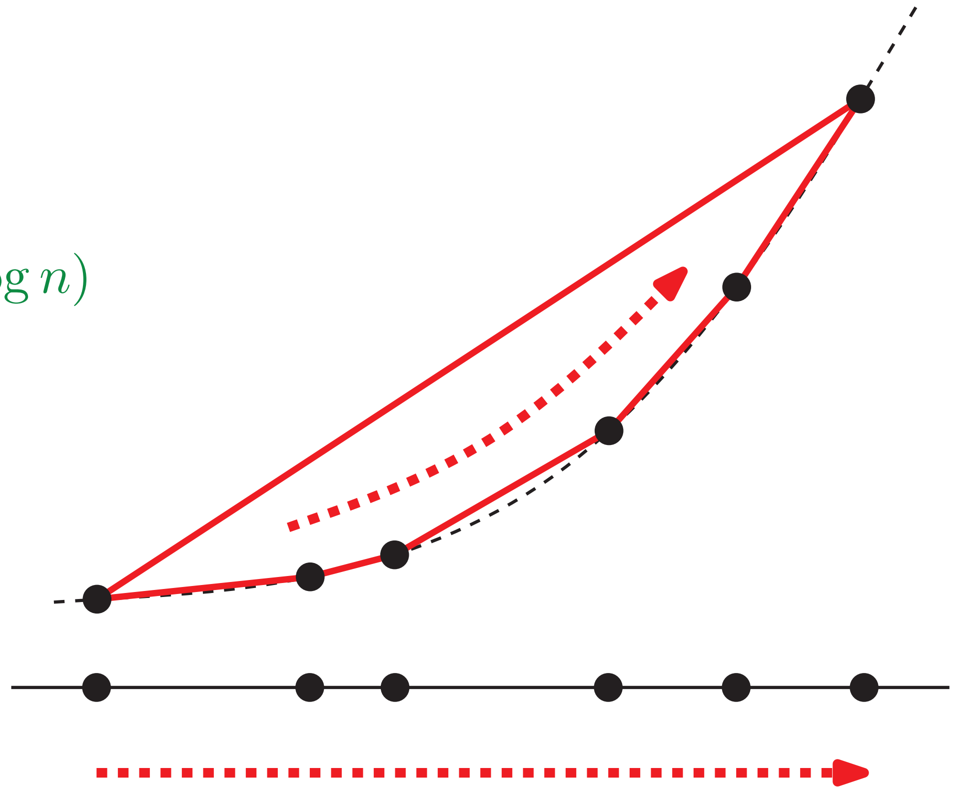
$\downarrow$

Output: convex hull of the points
Sorted list of the vertices of the convex hull

$\downarrow$ $O(n)$

Output: sorting the numbers
Sorted list of the numbers $x_1, \ldots, x_n$

$\Omega(n \log n)$



Lower bound

Input: n real numbers
$x_1, \ldots, x_n$ real numbers

$\downarrow O(n)$

Input: n points
$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$

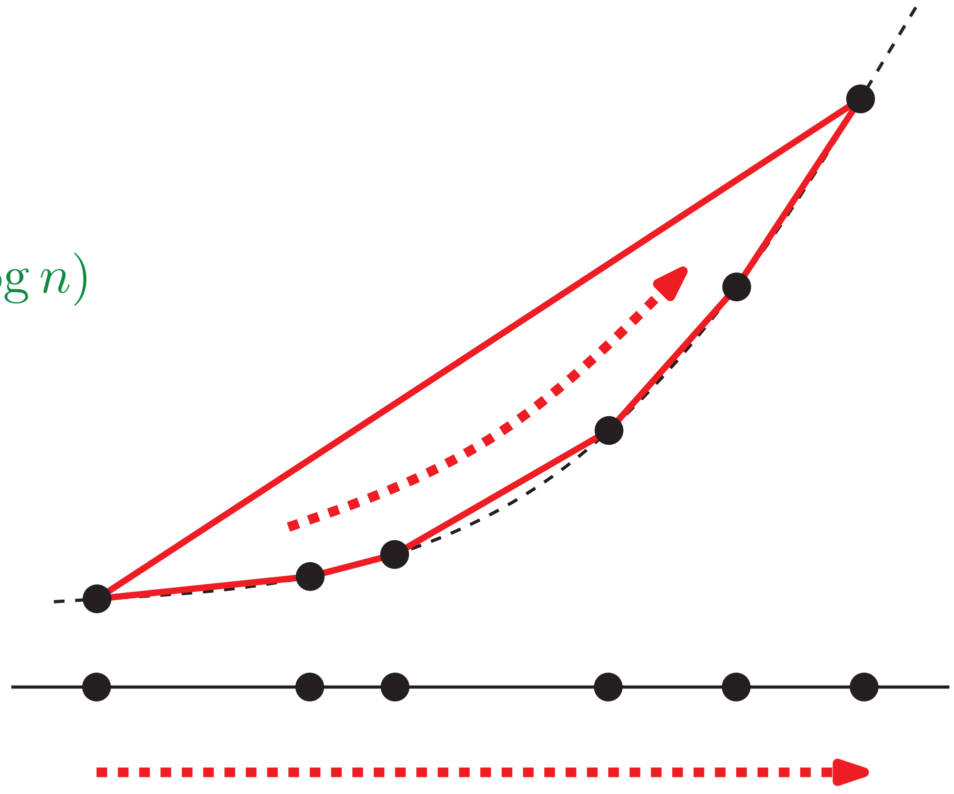
$\downarrow \Omega(n \log n)$

Output: convex hull of the points
Sorted list of the vertices of the convex hull

$\downarrow O(n)$

Output: sorting the numbers
Sorted list of the numbers $x_1, \ldots, x_n$

(Overall: $\Omega(n \log n)$)



Extension: convex hull of a simple polygon

- Is it possible to design an $o(n \log n)$ time algorithm by exploiting the order of the vertices of the polygon?
- Is it possible, for example, to apply Graham's algorithm using the order of the vertices of the polygon?

Extension: convex hull of a simple polygon

- Is it possible to design an $o(n \log n)$ time algorithm by exploiting the order of the vertices of the polygon?
- Is it possible, for example, to apply Graham's algorithm using the order of the vertices of the polygon?

Extension: do the previous strategies extend to the 3D case?

- Is it possible to design an 3-dimensional gitf wrap convex hull algorithm?
- Is it possible to design a 3-dimensional incremental convex hull algorithm?
- Is it possible to design a 3-dimensional divide-and-conquer convex hull algorithm?

FURTHER READING

- J. O'Rourke, **Computational Geometry in C (2nd ed.)**, Cambridge University Press, 1998.
- F. Preparata, M. Shamos, **Computational Geometry: An introduction (revised ed.)**, Springer, 1993.

...AND PLAYING

In 2D:

http://www.dma.fi.upm.es/docencia/segundociclo/geomcomp/convex.html

In 3D:

http://www.cse.unsw.edu.au/~lambert/java/3d