CONVEX HULL IN 2D

Vera Sacristán

ete and Algorithmic Geometry Geometry
acultat de Matemàtiques i Estadística
Universitat Politècnica de Catalunya

CONVEX HULL IN 2D

Computing the extreme points

Computing the extreme points

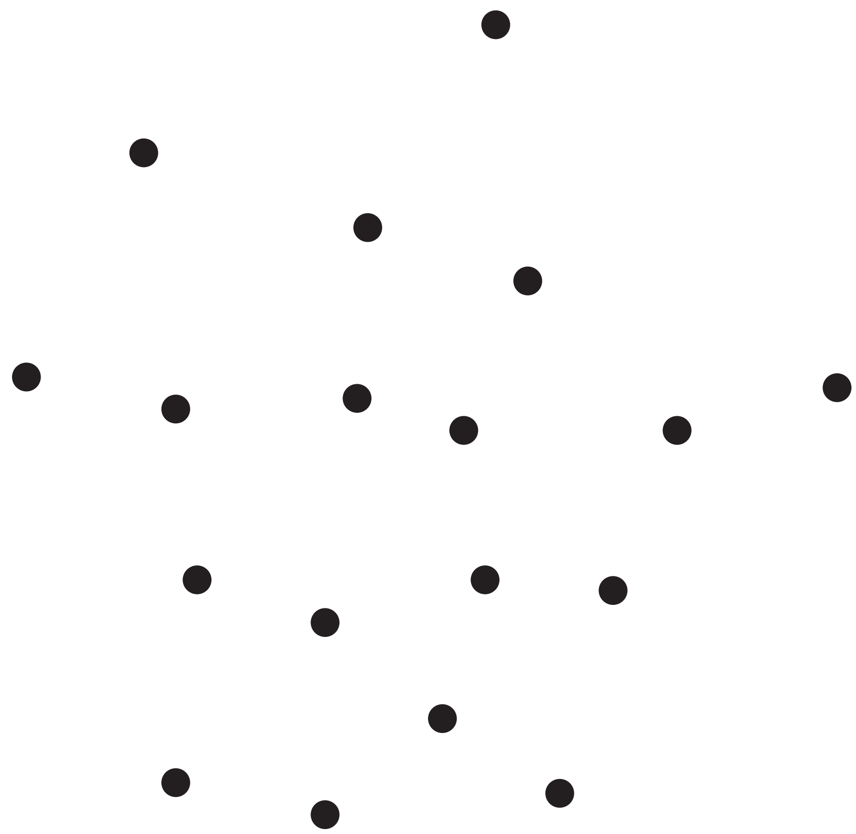
Characterization

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 $p_j p_k p_l$.

Computing the extreme points

Characterization

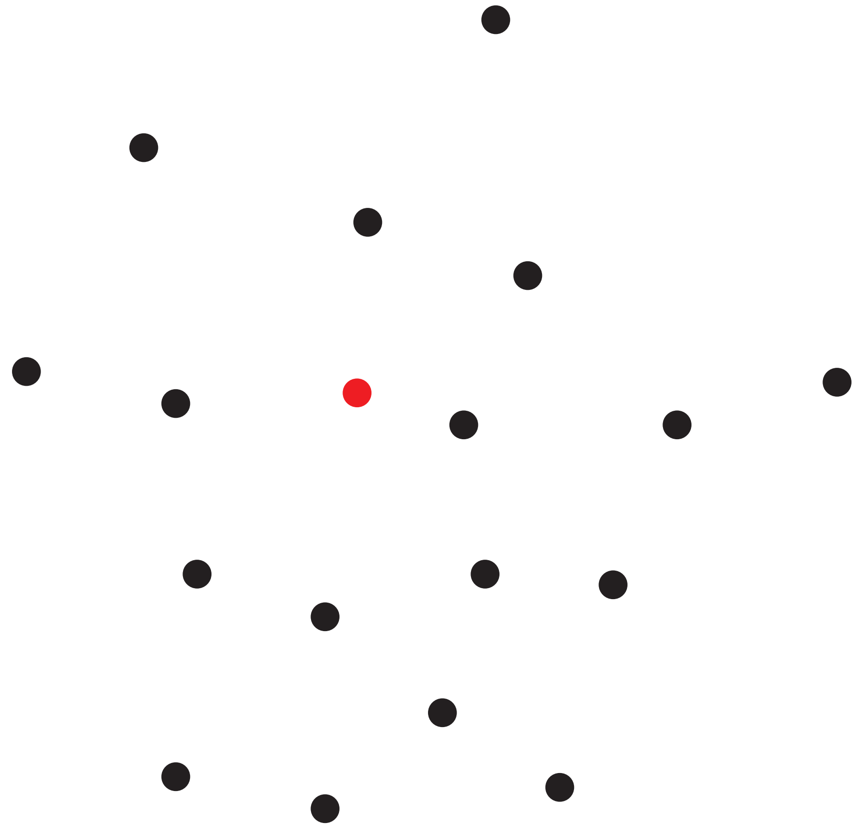
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.



Computing the extreme points

Characterization

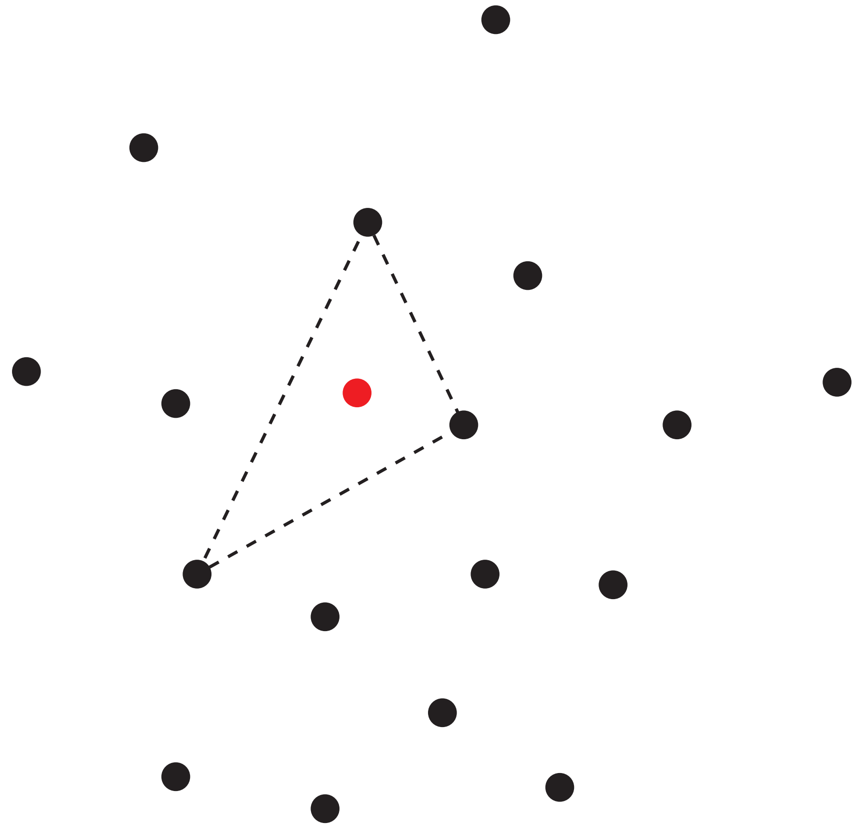
Given $X = \{p_1, \ldots, p_n\}$, the point p_i belongs to the boundary of the convex hull of X if and only if p_i does not lie in any of the triangles $p_j p_k p_l$.



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 $X = \{p_1, \ldots, p_n\}$, 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

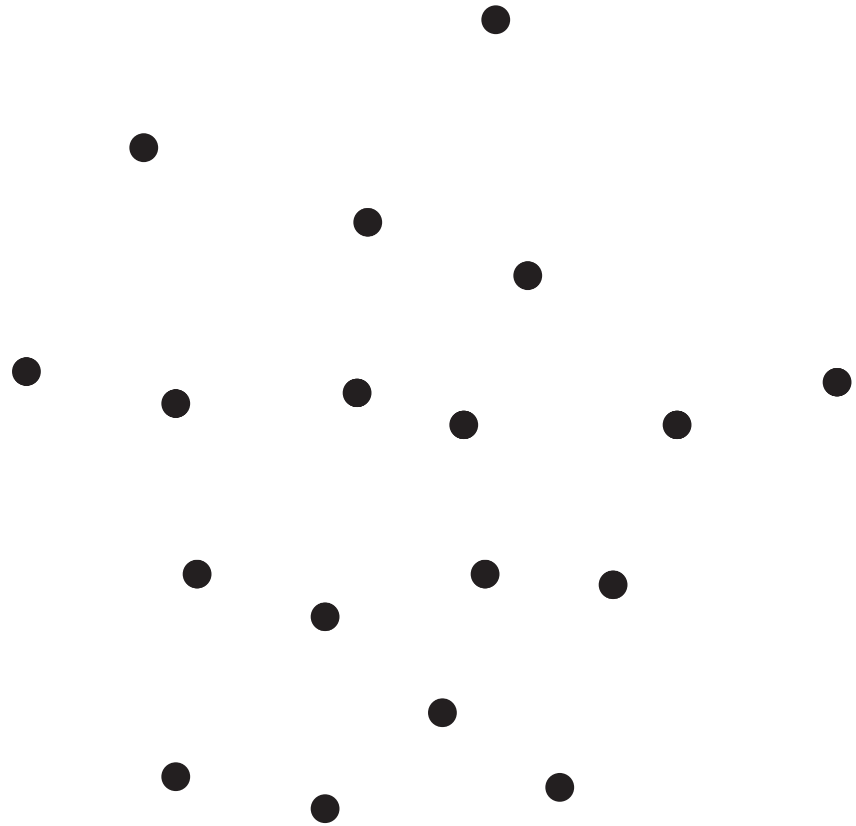
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

Re 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

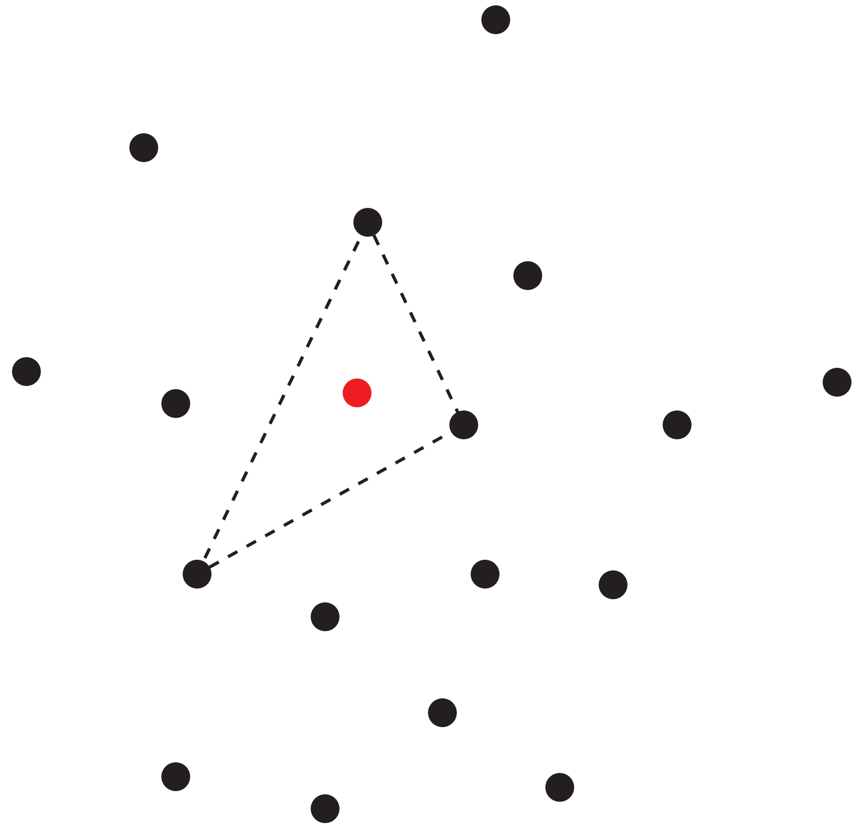
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

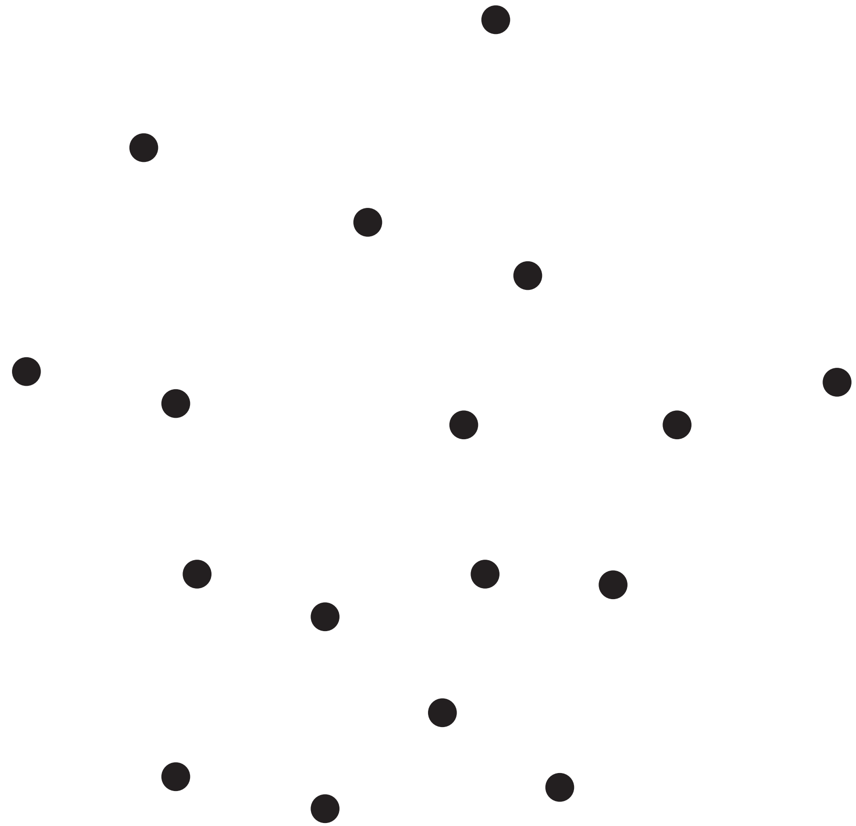
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R f 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 $p_j p_k p_l$.

Alg

Inpu

Outp ... extreme points

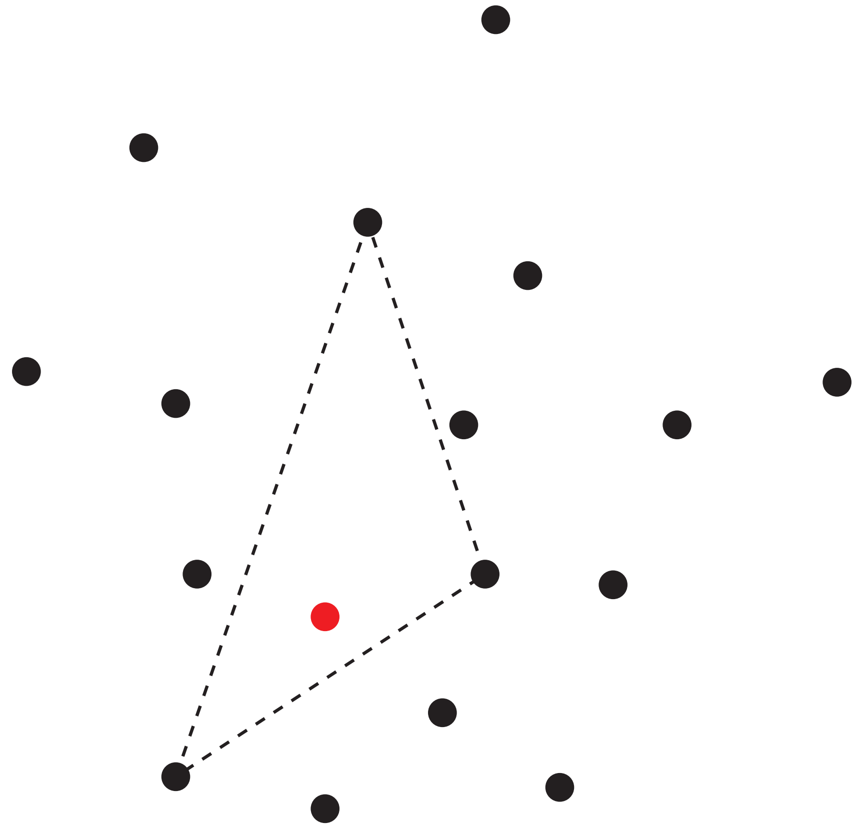
Proc

Fo

... $l \neq i$,

... e triangle p_j, p_k, p_l, eliminate p_i.

R ... f 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

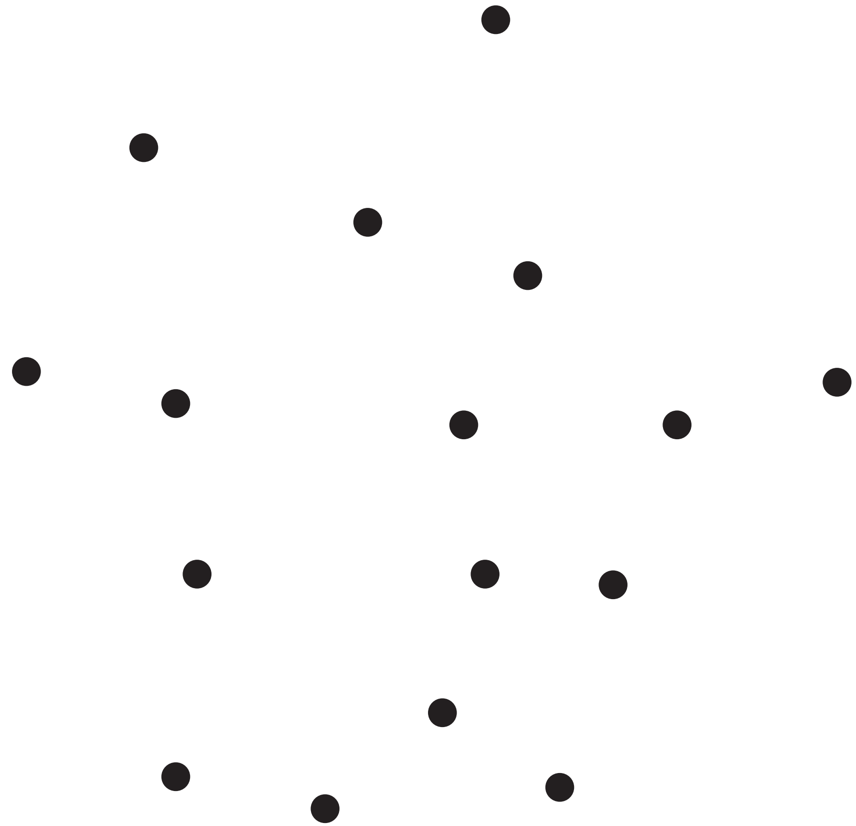
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

Re 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

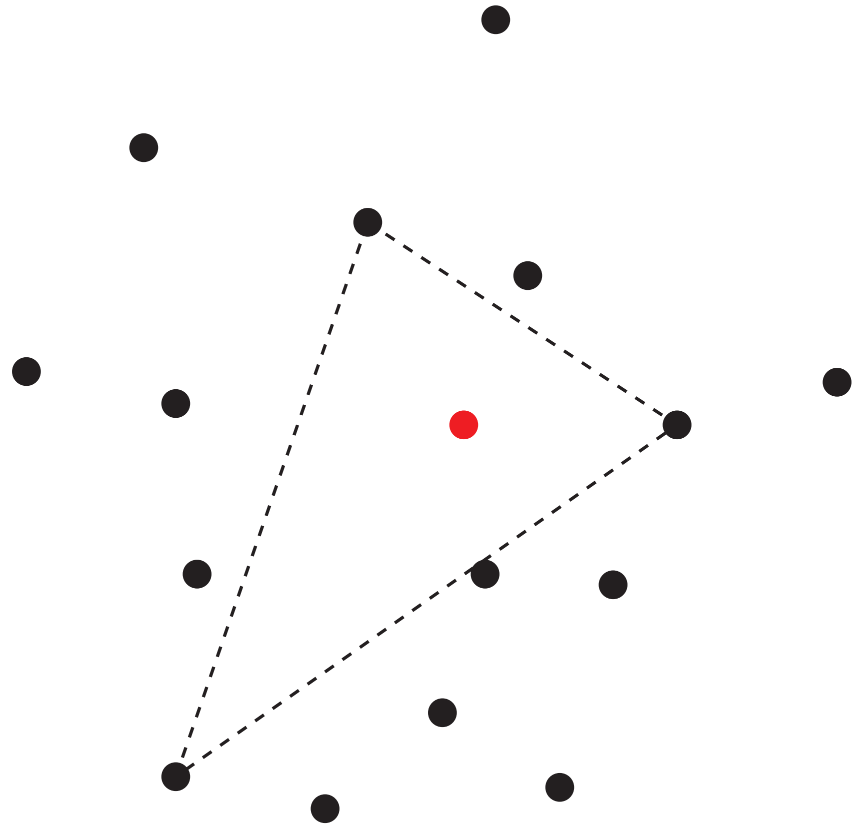
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

Re 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

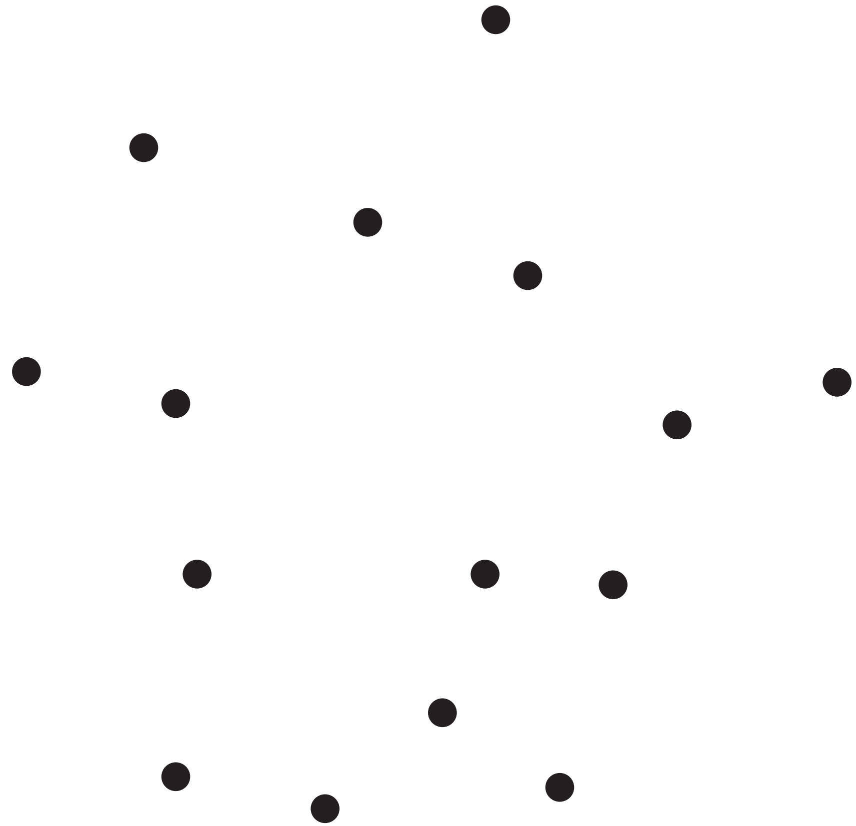
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R f 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

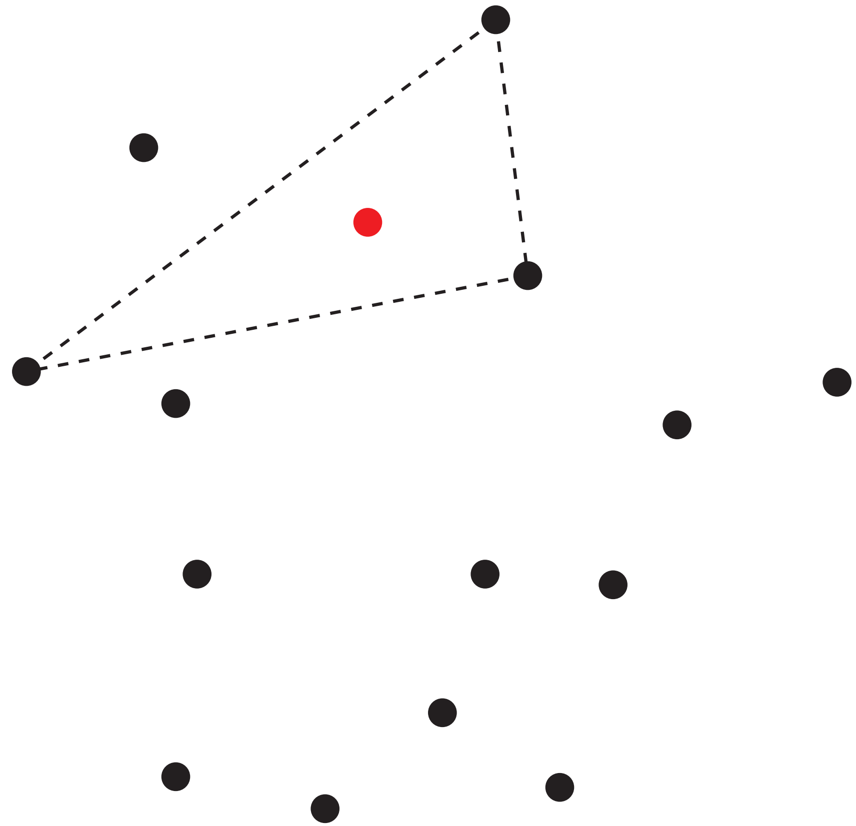
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R f 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 $p_j p_k p_l$.

Algo

Inpu

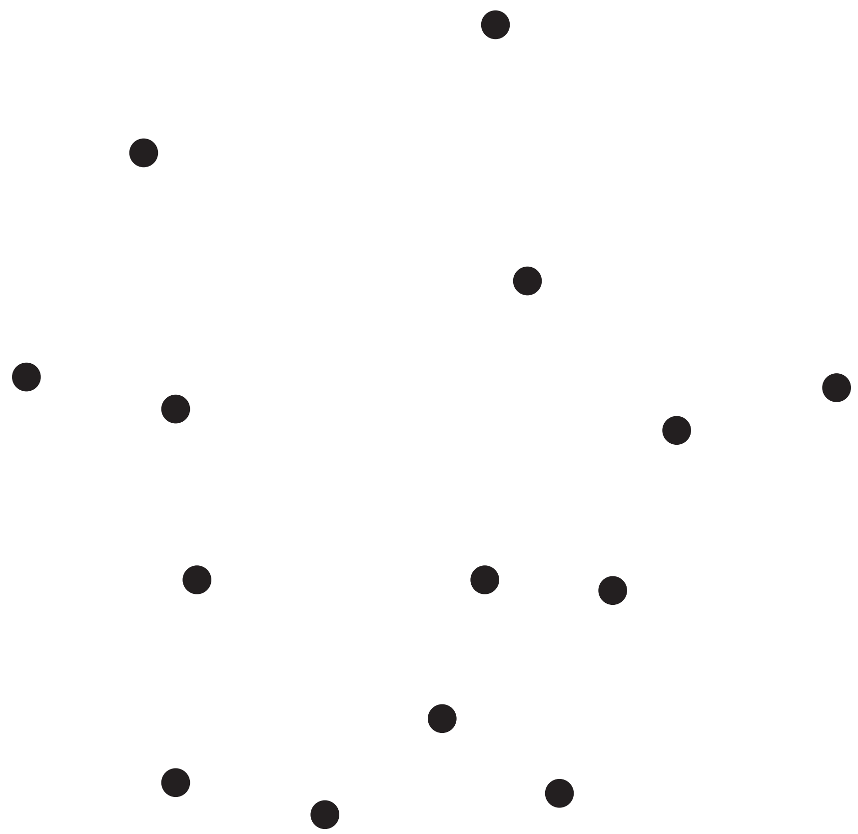
Outp ... extreme points

Proc

Fo ... $l \neq i$,

... e triangle p_j, p_k, p_l, eliminate p_i.

R ... of surviving p_i's.



Computing the extreme points

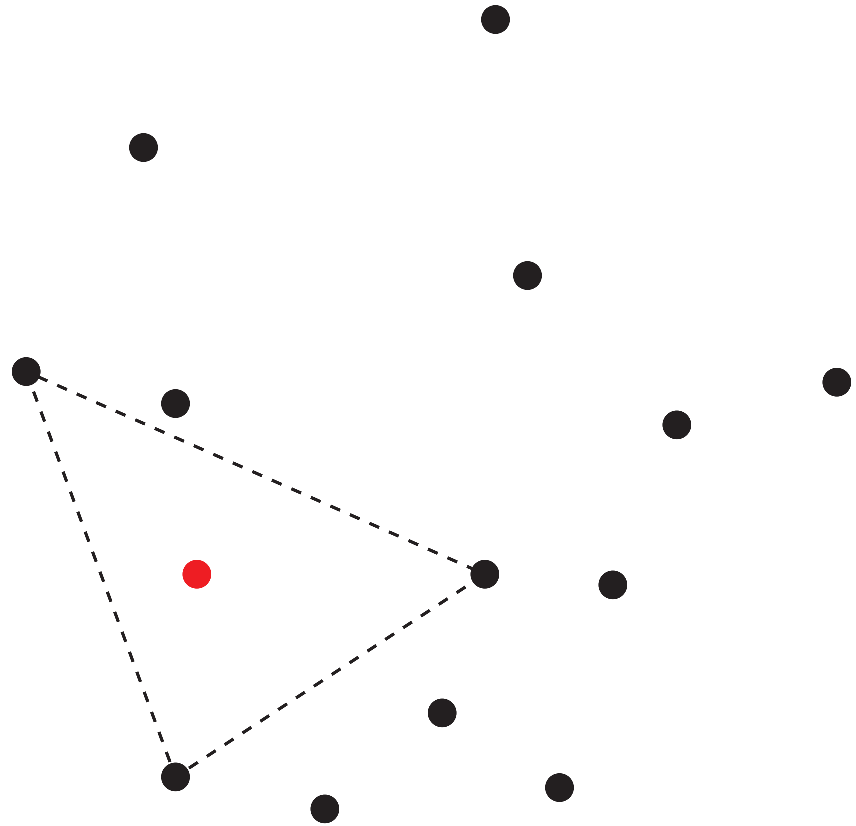
Characterization

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 $p_j p_k p_l$.

Algo

Inpu
Outp extreme points

Proc
Fo
$l \neq i$,
e triangle p_j, p_k, p_l, eliminate p_i.
Re f 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

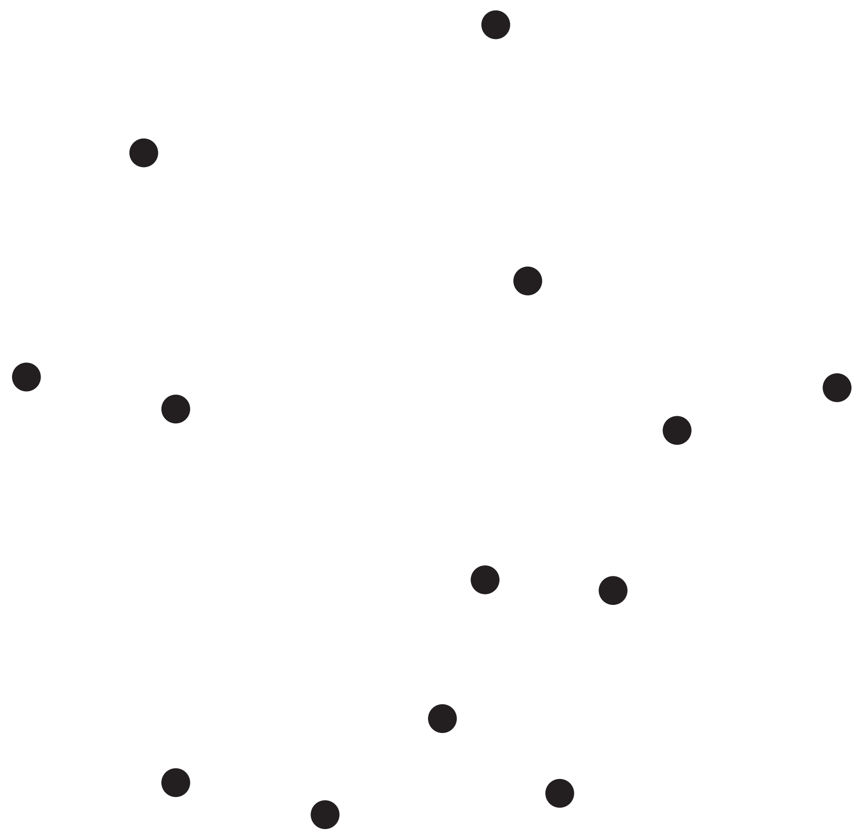
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

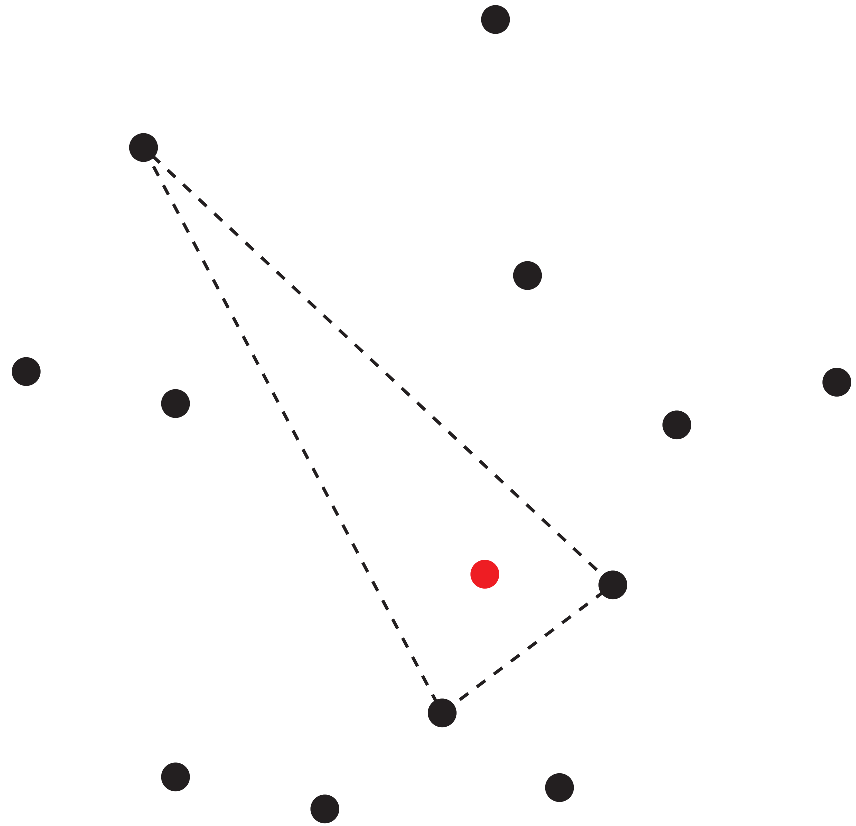
Re f surviving p_i's.



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 $p_j p_k p_l$.



Algo

Inpu

Outp extreme points

Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R f 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

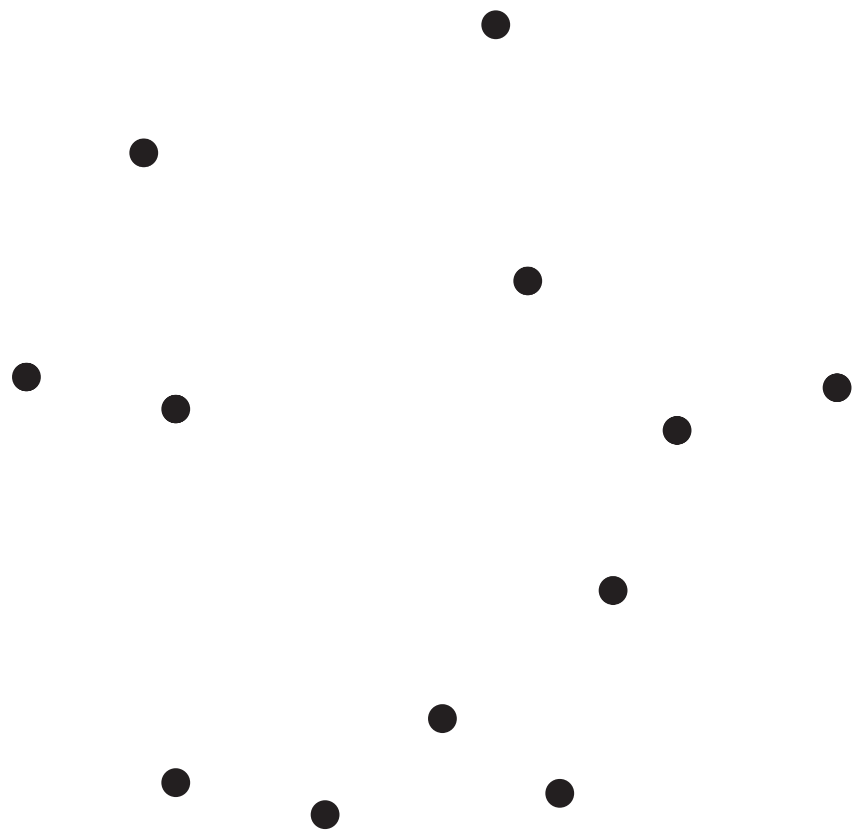
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

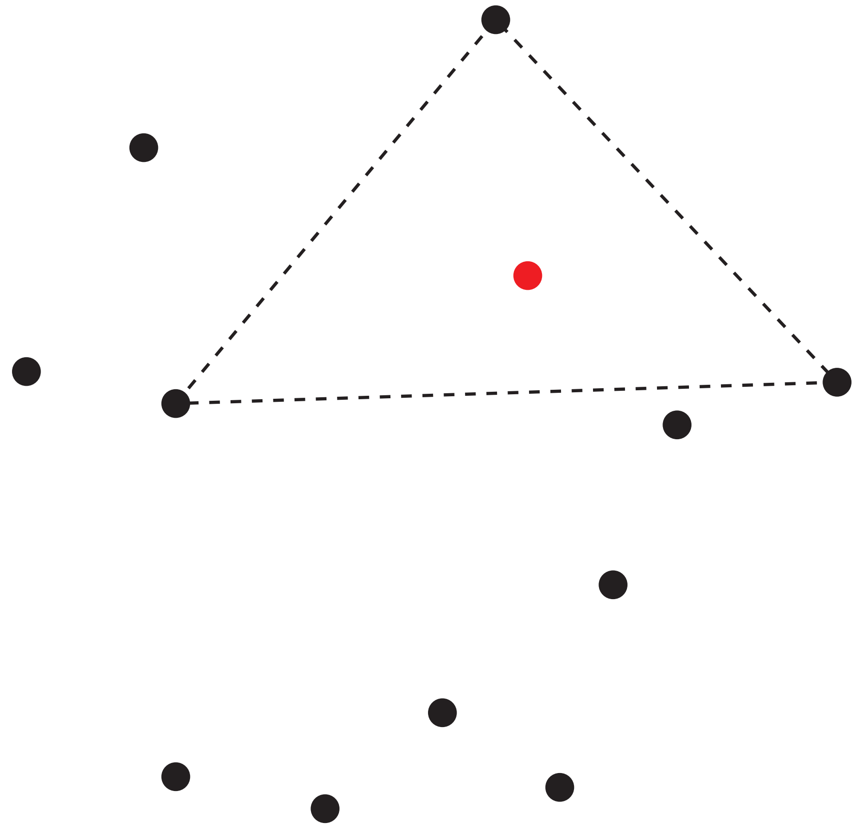
R f surviving p_i's.



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 $p_jp_kp_l$.



Alg

Inpu

Out extreme points

Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R f 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 $p_j p_k p_l$.

Alg

Inpu

Outp extreme points

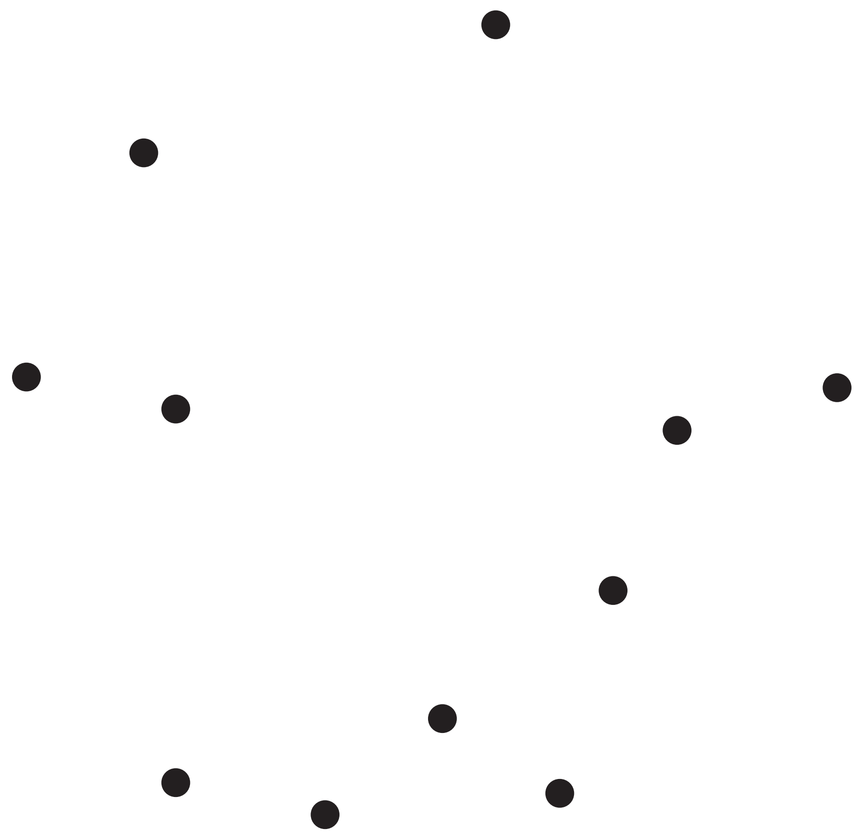
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R f 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

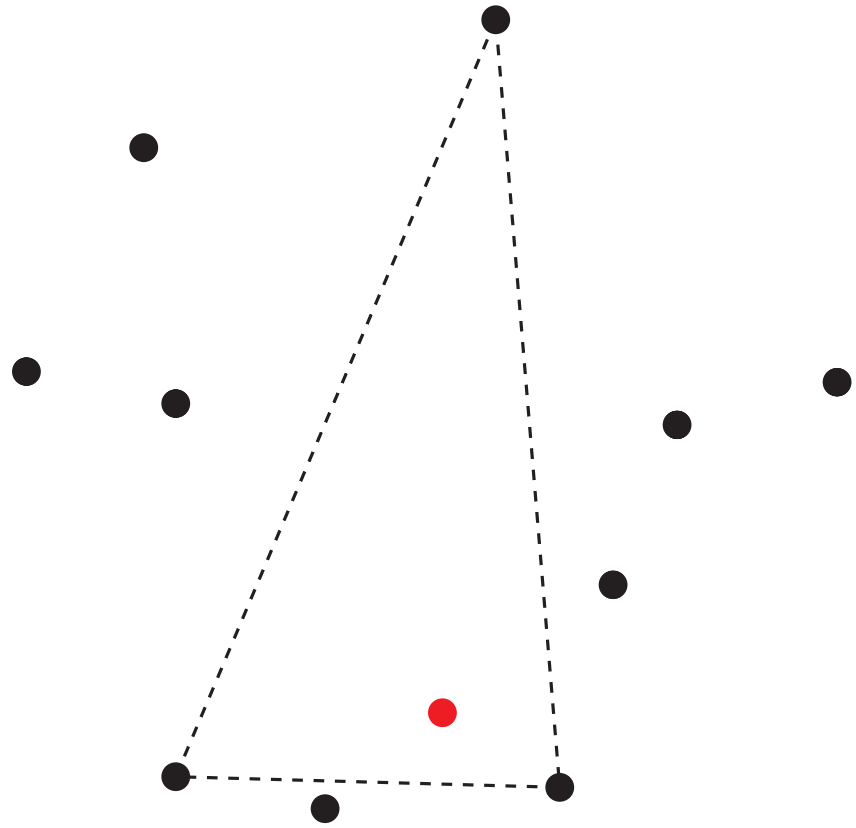
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

Re 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

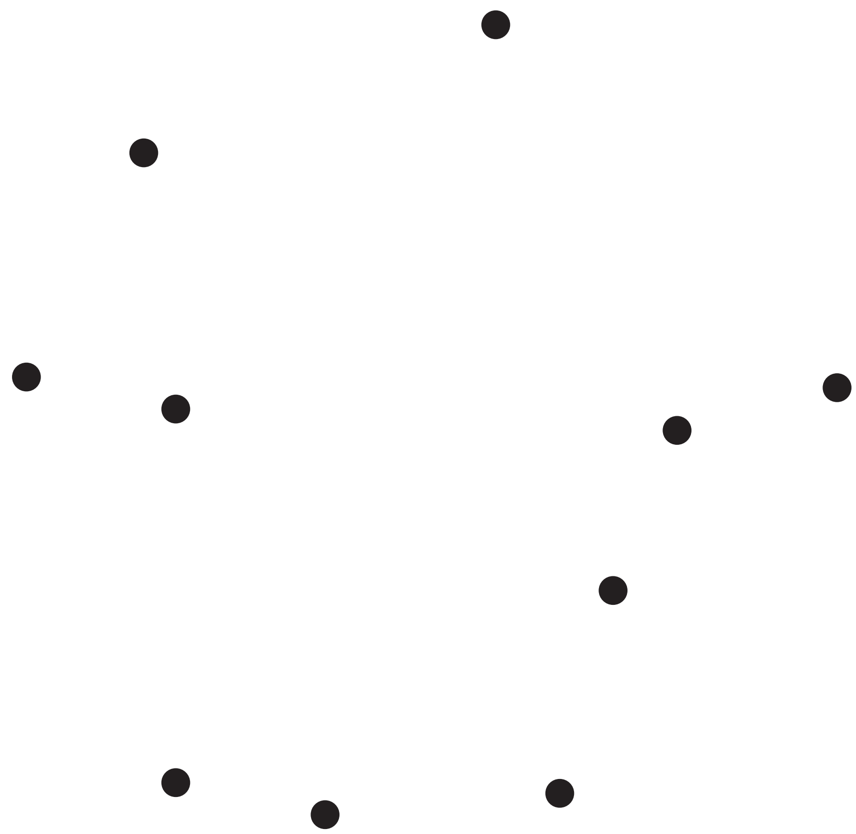
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R f 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 $p_j p_k p_l$.

Alg…

Inpu…

Outp… extreme points

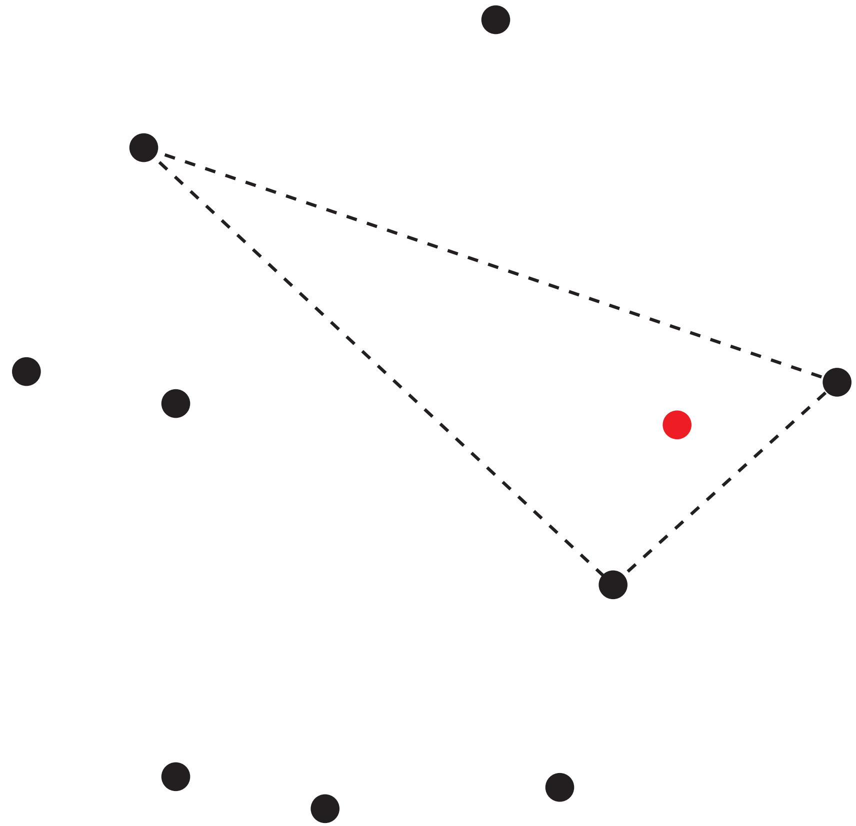
Proc…

Fo…

… $l \neq i$,

… e triangle p_j, p_k, p_l, eliminate p_i.

R… 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

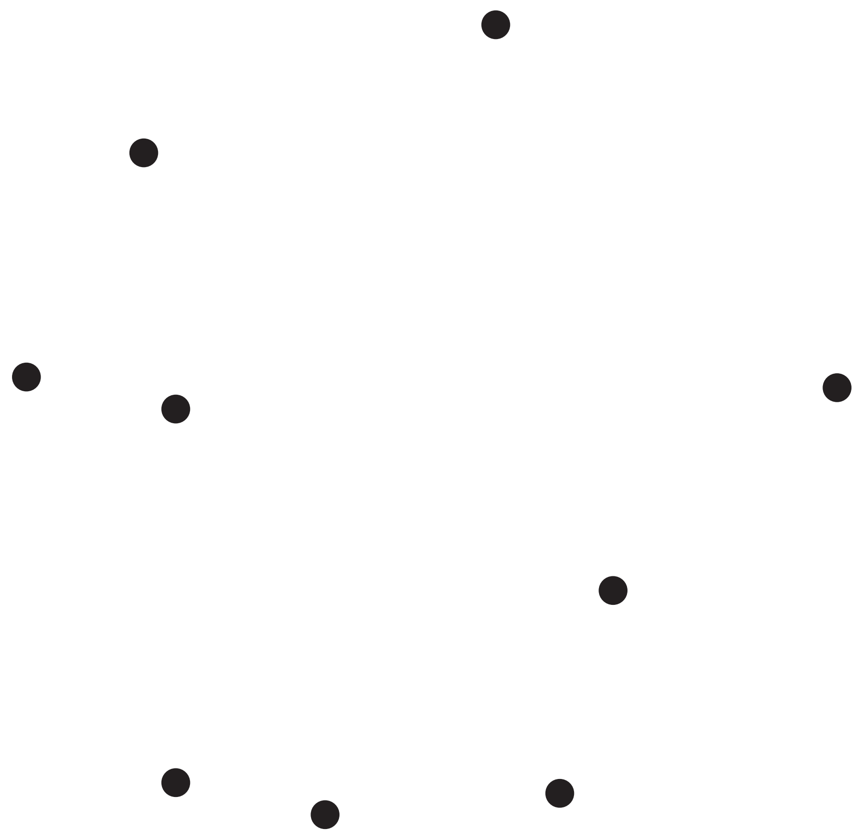
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R 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 $p_j p_k p_l$.

Alg

Inpu

Outp extreme points

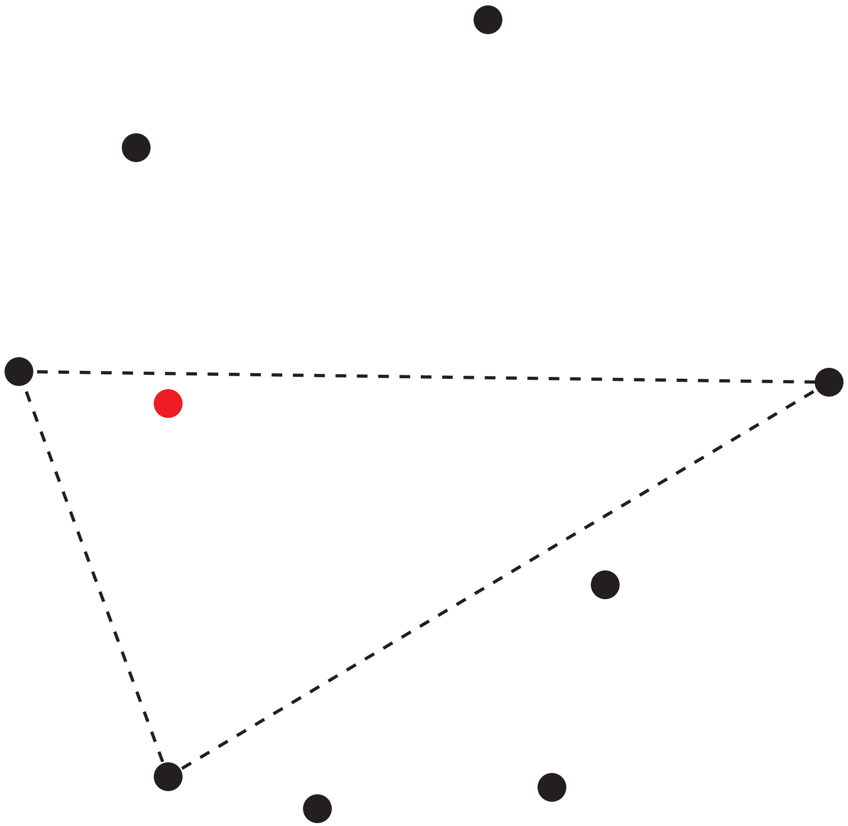
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R 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 $p_j p_k p_l$.

Alg

Inpu

Outp extreme points

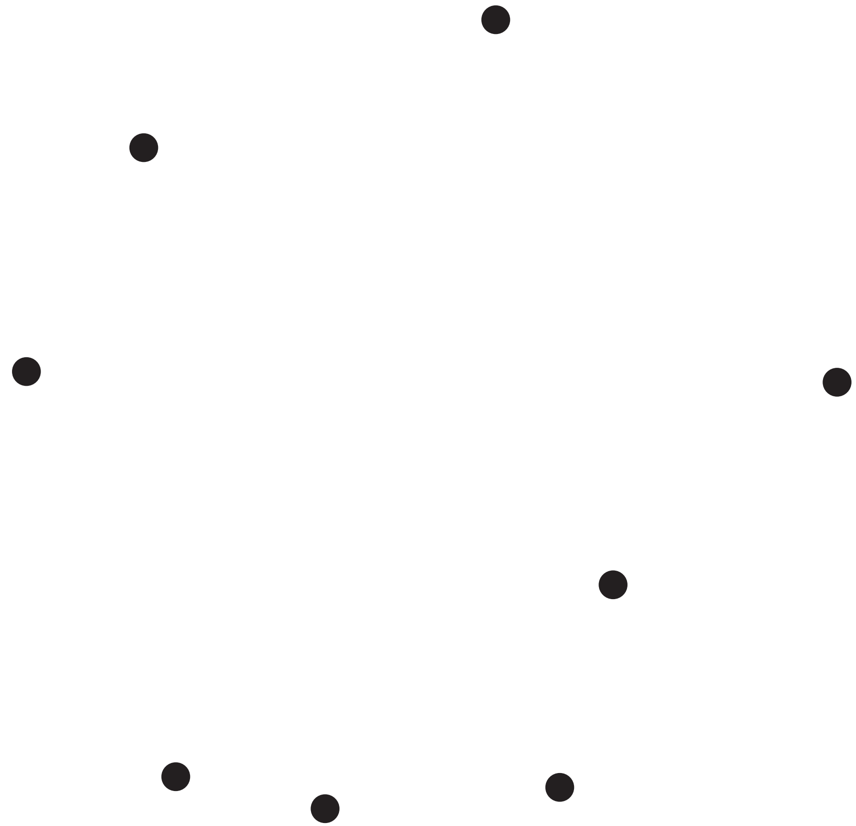
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

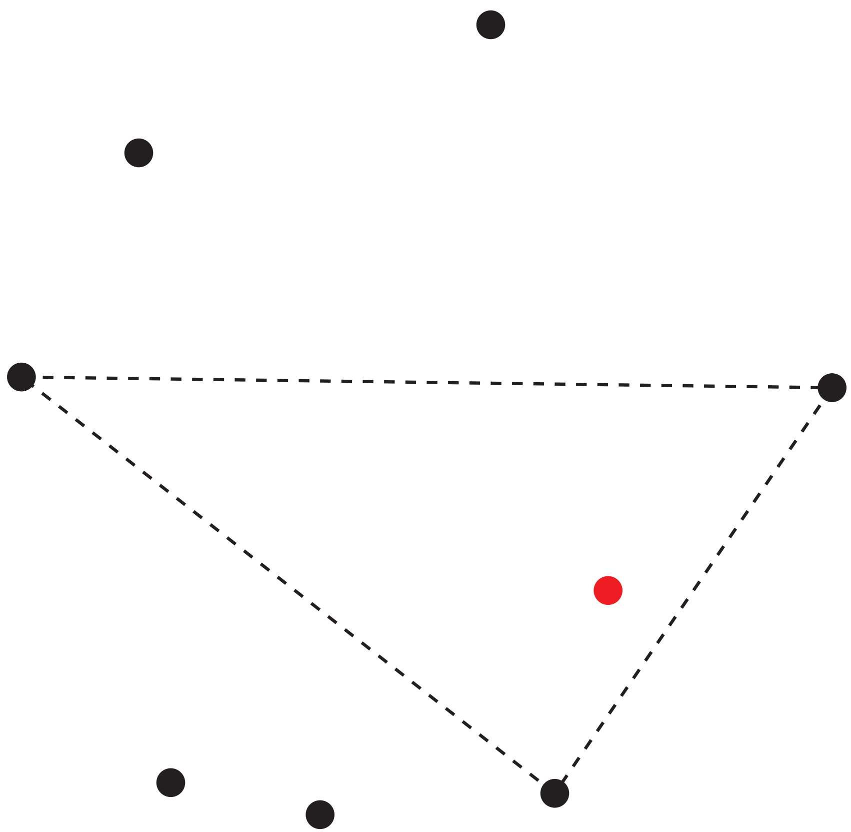
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

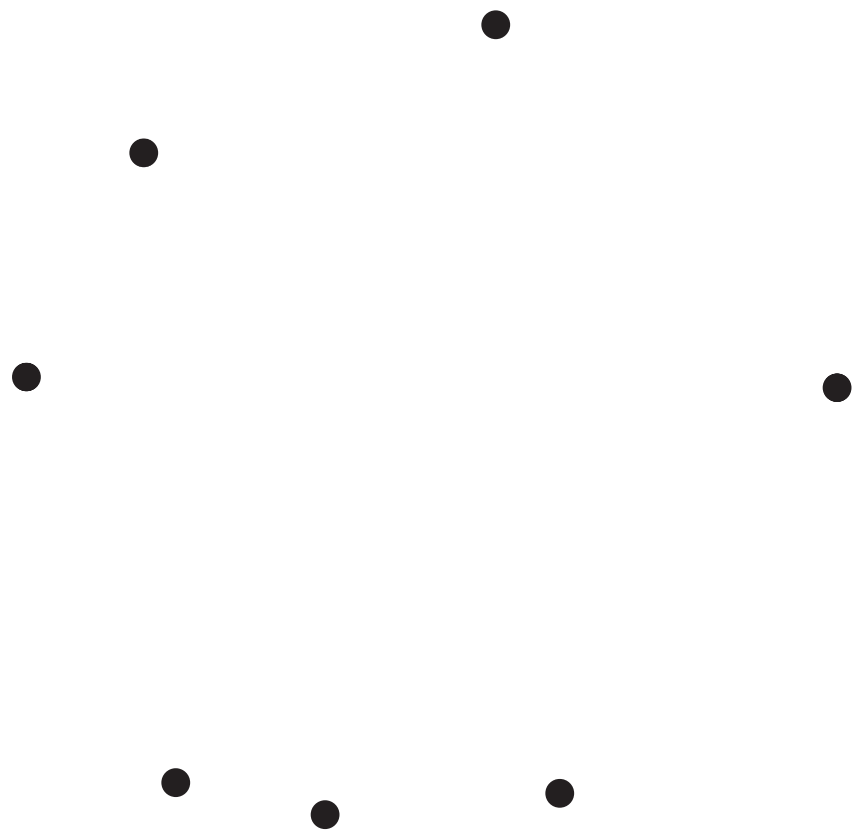
Proc

Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R f 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 $p_j p_k p_l$.

Algo

Inpu

Outp extreme points

Proc

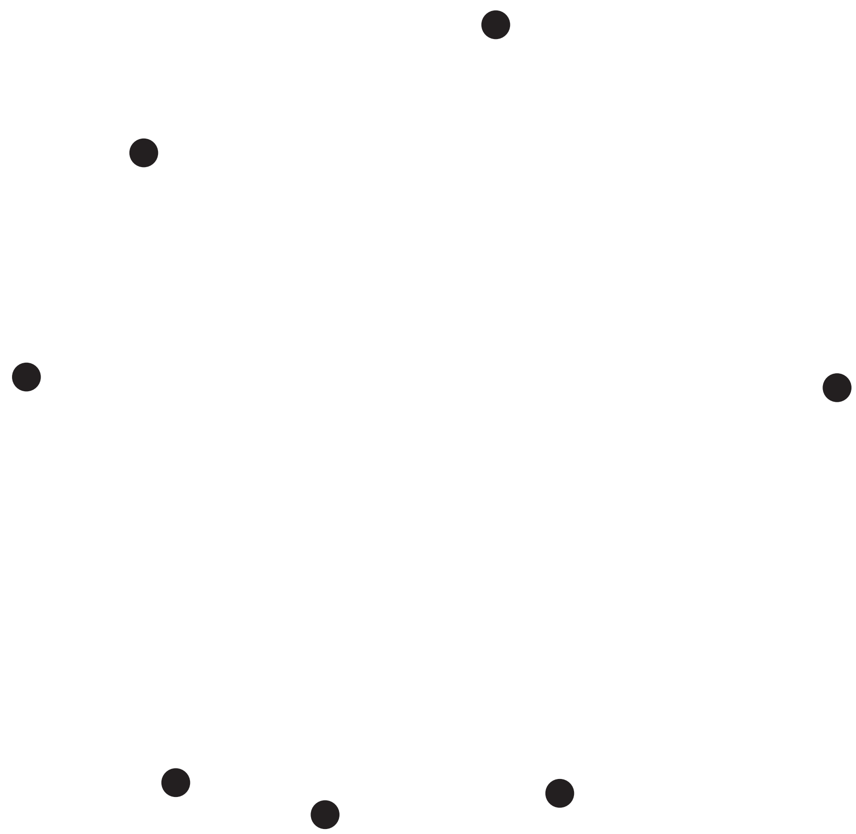
Fo

$l \neq i$,

e triangle p_j, p_k, p_l, eliminate p_i.

R of surviving p_i's.

Run $\Theta(n^4)$



Computing the extreme segments

Computing the extreme segments

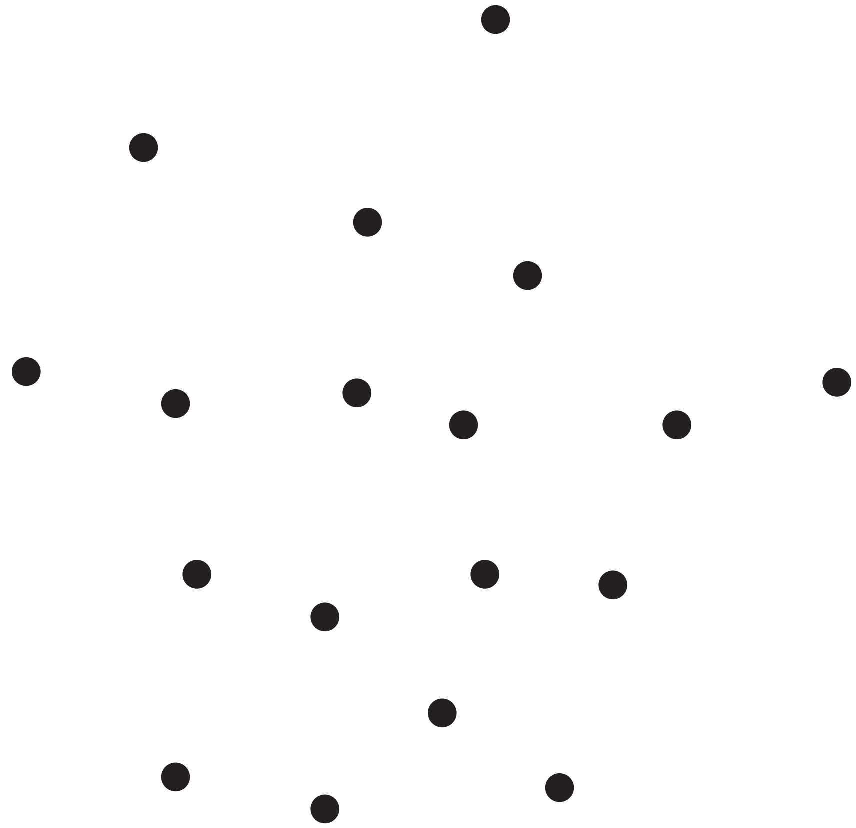
Characterization

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

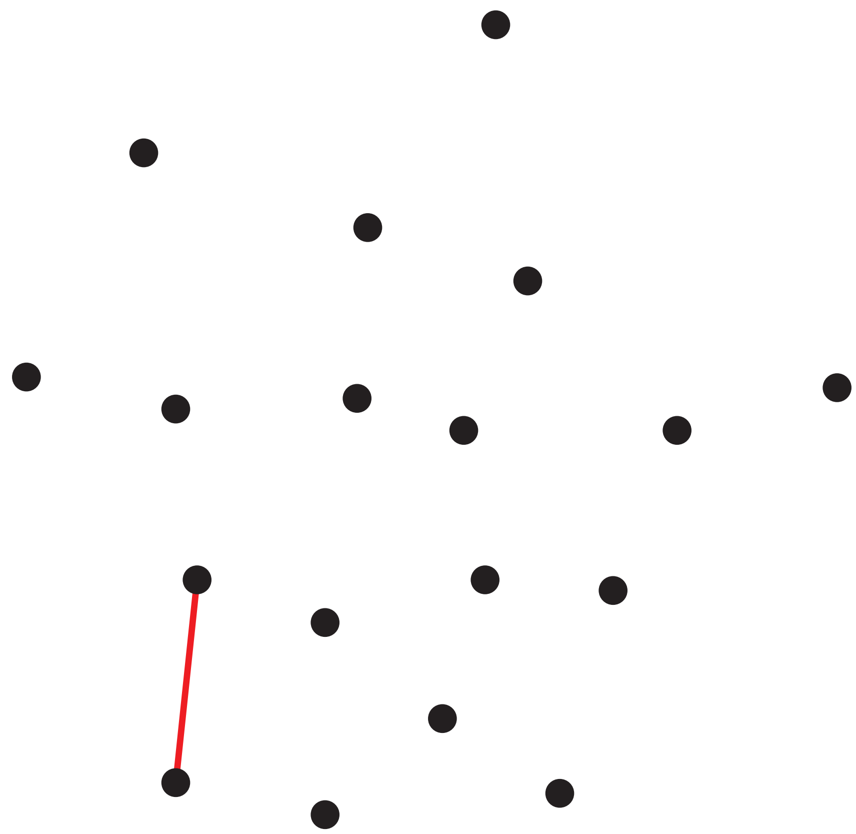
Given $X = \{p_1, \ldots, p_n\}$, the segment p_ip_j is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line p_ip_j.



Computing the extreme segments

Characterization

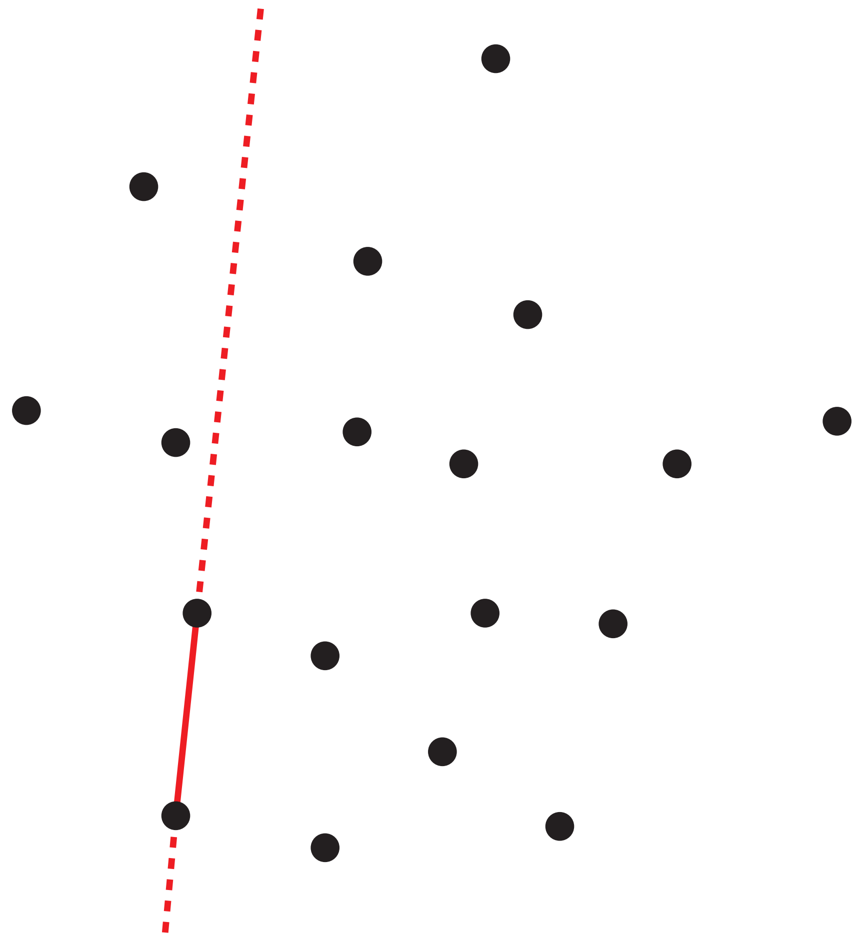
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

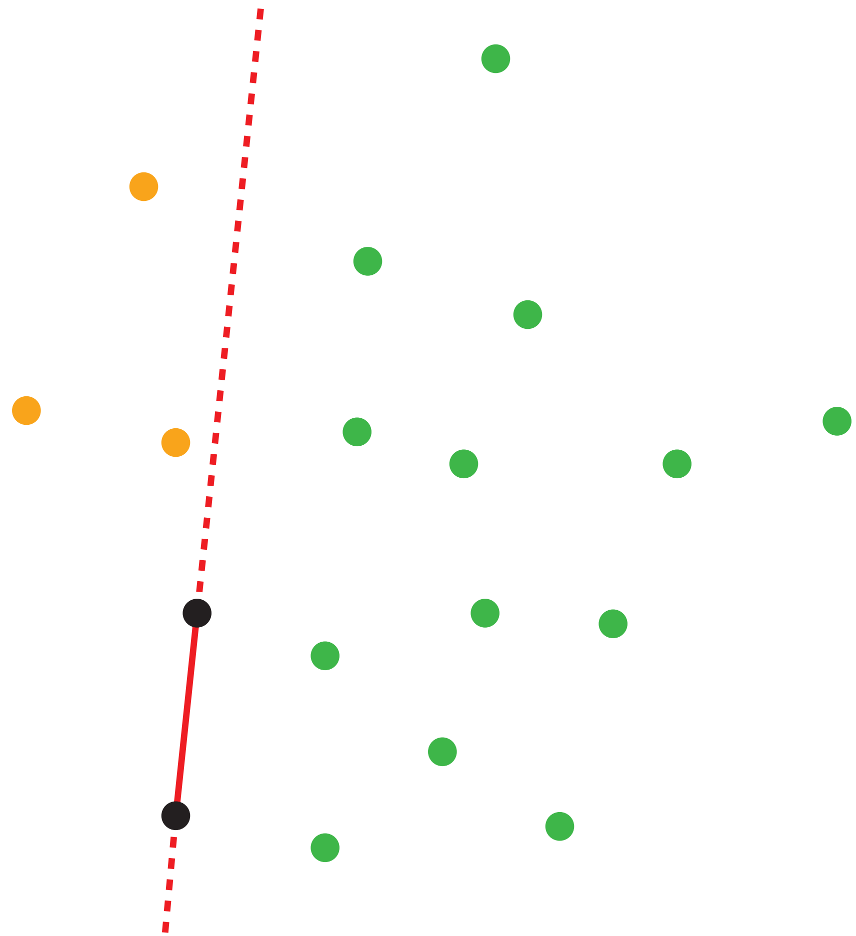
Given $X = \{p_1, \ldots, p_n\}$, the segment p_ip_j is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line p_ip_j.



Computing the extreme segments

Characterization

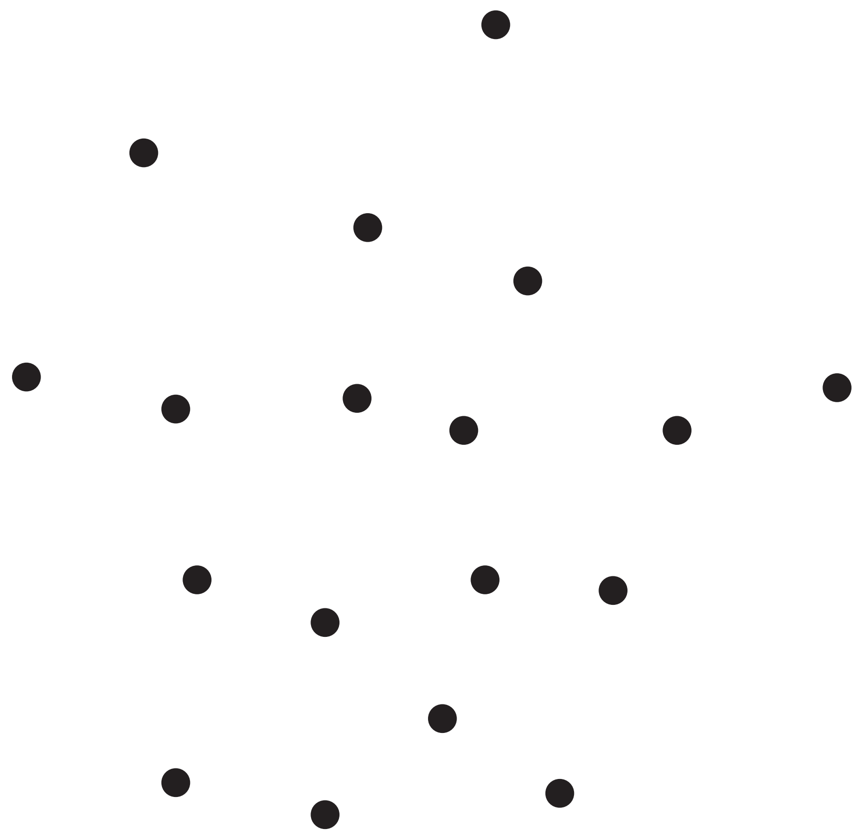
Given $X = \{p_1, \ldots, p_n\}$, the segment p_ip_j is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line p_ip_j.



Computing the extreme segments

Characterization

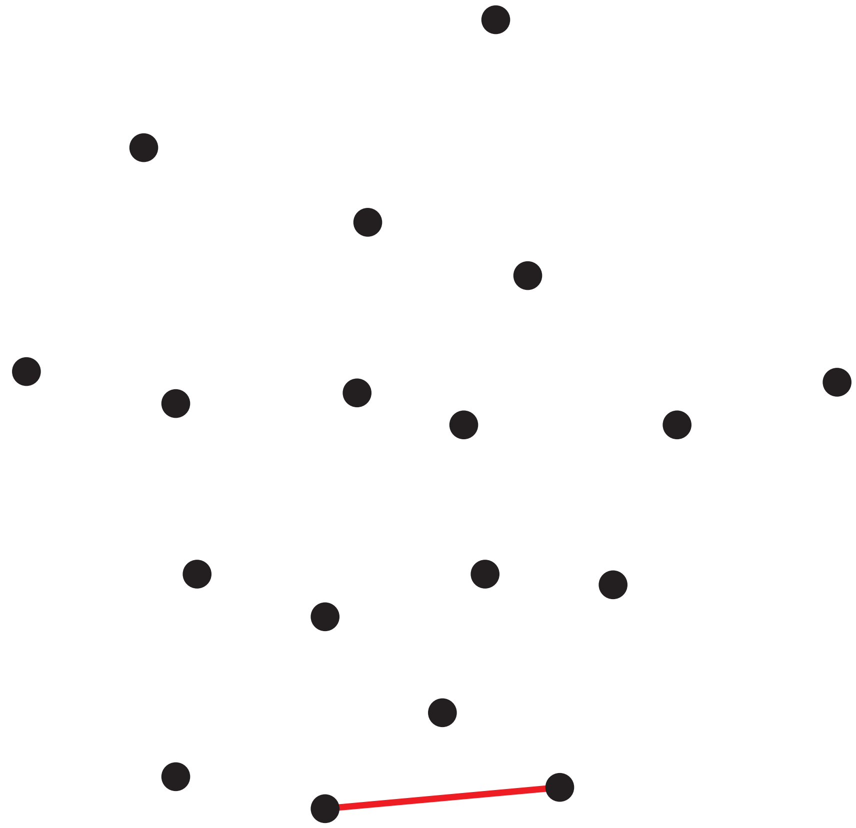
Given $X = \{p_1, \ldots, p_n\}$, the segment p_ip_j is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line p_ip_j.



Computing the extreme segments

Characterization

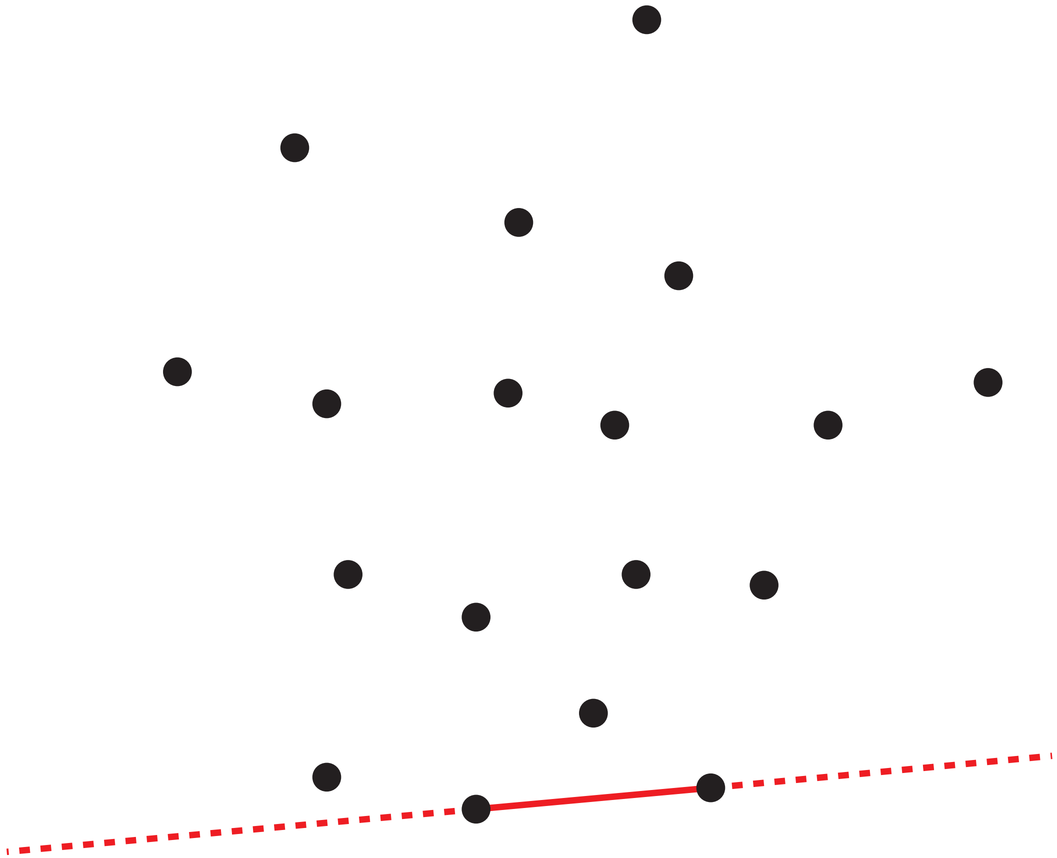
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

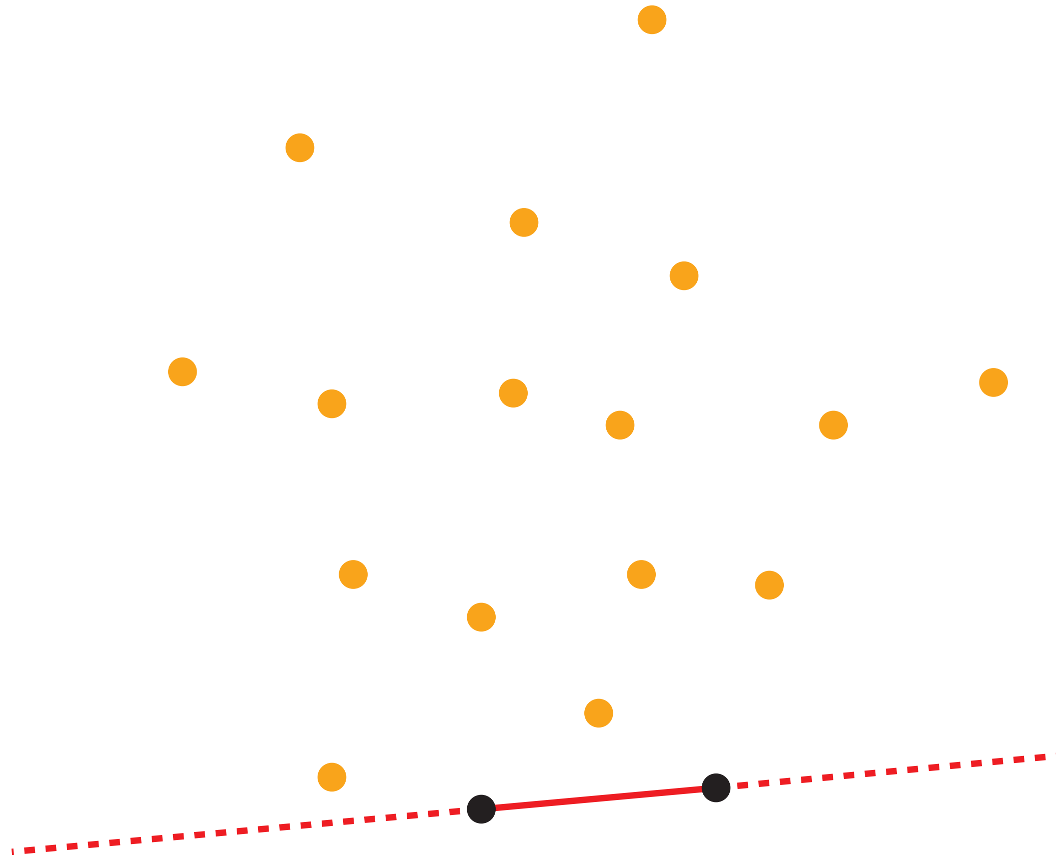
Given $X = \{p_1, \ldots, p_n\}$, the segment $p_i p_j$ is an extreme segment if and only if all the points p_k with $k \neq i, j$ lie in the same halfplane defined by the line $p_i p_j$.



Computing the extreme segments

Characterization

Given $X = \{p_1, \ldots, p_n\}$, the segment p_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 $p_i p_j$.

Algo

Inpu

Outp extreme segments

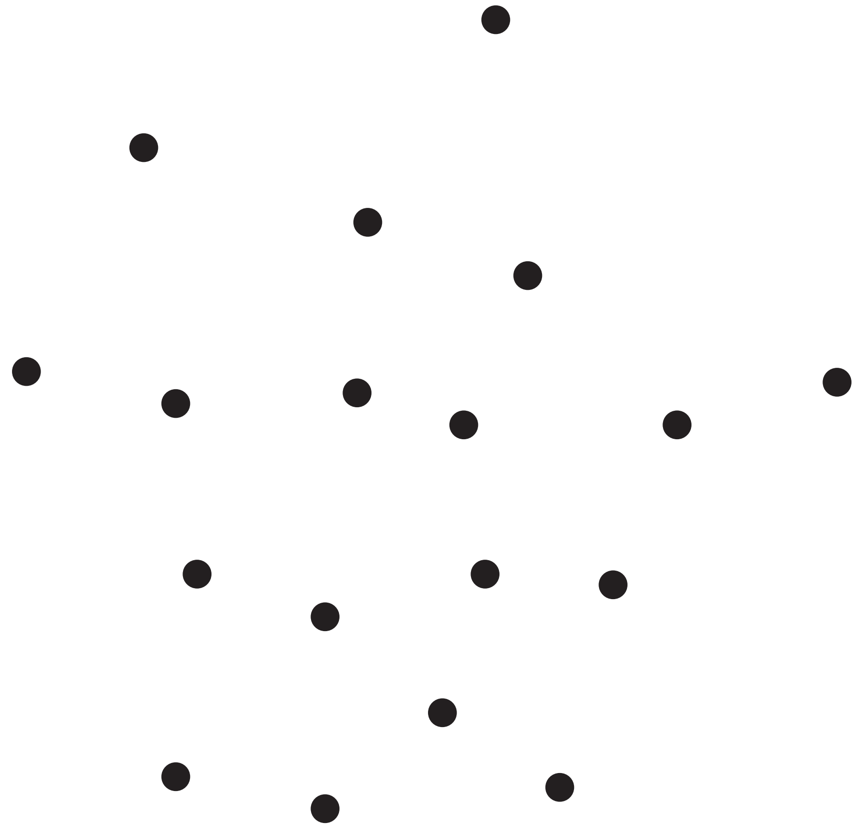
Proc

Fo

er all p_k with $k \neq i, j$

he halfplane defined by $p_i p_j$.

tive, return the segment $p_i p_j$.



CONVEX HULL IN 2D

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.

Algo

Inpu

Outp ... extreme segments

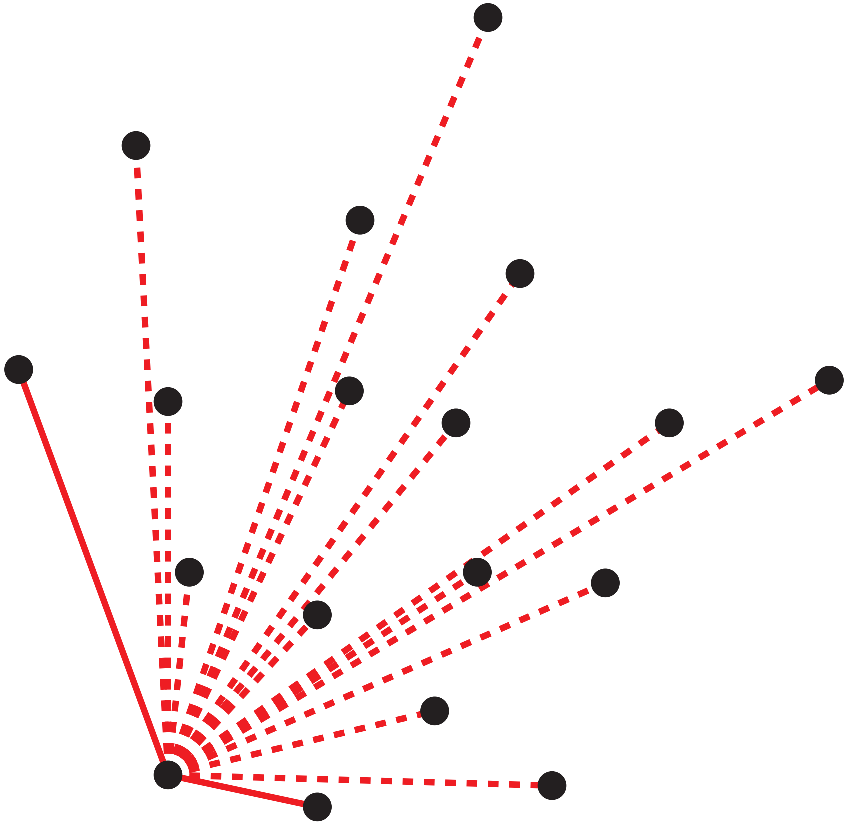
Proc

Fo

...r all p_k with $k \neq i, j$

...he halfplane defined by p_ip_j.

...tive, return the segment p_ip_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 $p_i p_j$.

Algo

Inpu

Outp extreme segments

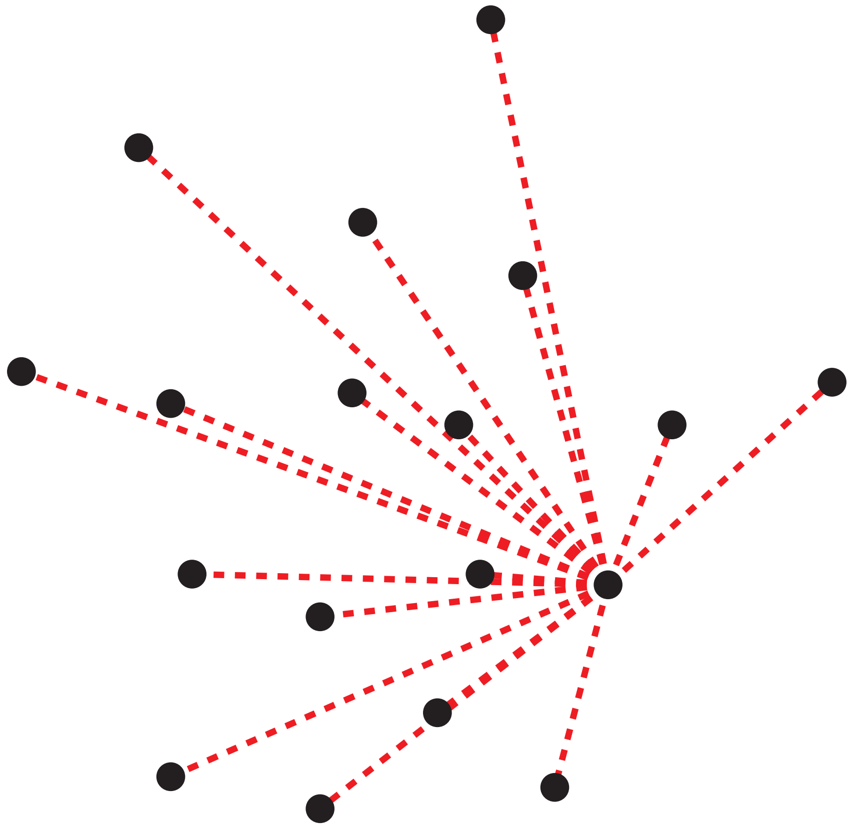
Proc

Fo

r all p_k with $k \neq i, j$

he halfplane defined by $p_i p_j$.

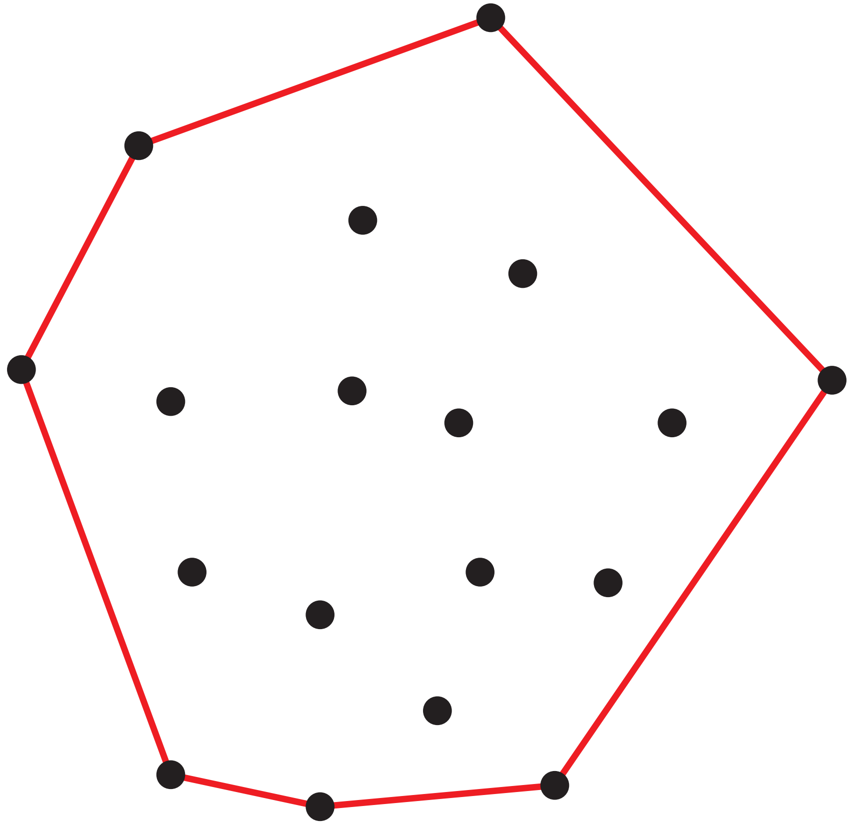
tive, return the segment $p_i p_j$.



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 $p_i p_j$.



Algo

Inpu

Outp extreme segments

Proc

Fo

er all p_k with $k \neq i, j$

he halfplane defined by $p_i p_j$.

tive, return the segment $p_i p_j$.

CONVEX HULL IN 2D

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.

Algo

Inpu

Outp extreme segments

Proc

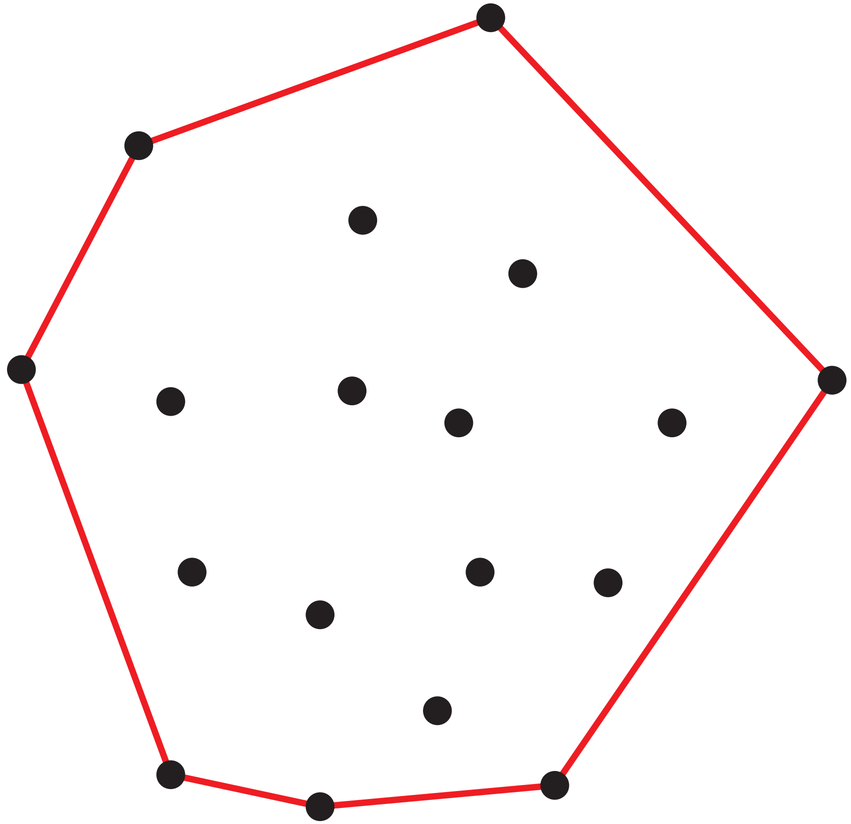
Fo

er all p_k with $k \neq i, j$

he halfplane defined by p_ip_j.

tive, return the segment p_ip_j.

Run $\Theta(n^3)$



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

O…

… of the vertices of $ch(P)$ sorted in counterclockwise order

CONVEX HULL IN 2D

Computing the convex hull

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

O

of the vertices of $ch(P)$ sorted in counterclockwise order

Cha

Give $\dots, p_n\}$, the segment $p_i p_j$ is an edge of the convex hull of X if and only if all the $k \neq i, j$ lie to the left of the oriented line $p_i p_j$.

CONVEX HULL IN 2D

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. ... $\mathrm{ast}(l) \neq \mathrm{First}(l)$, do:

   ... the angularly rightmost point $p_j \in P$ ... spect to v.

   ... to l

3. ...

Jarvis march

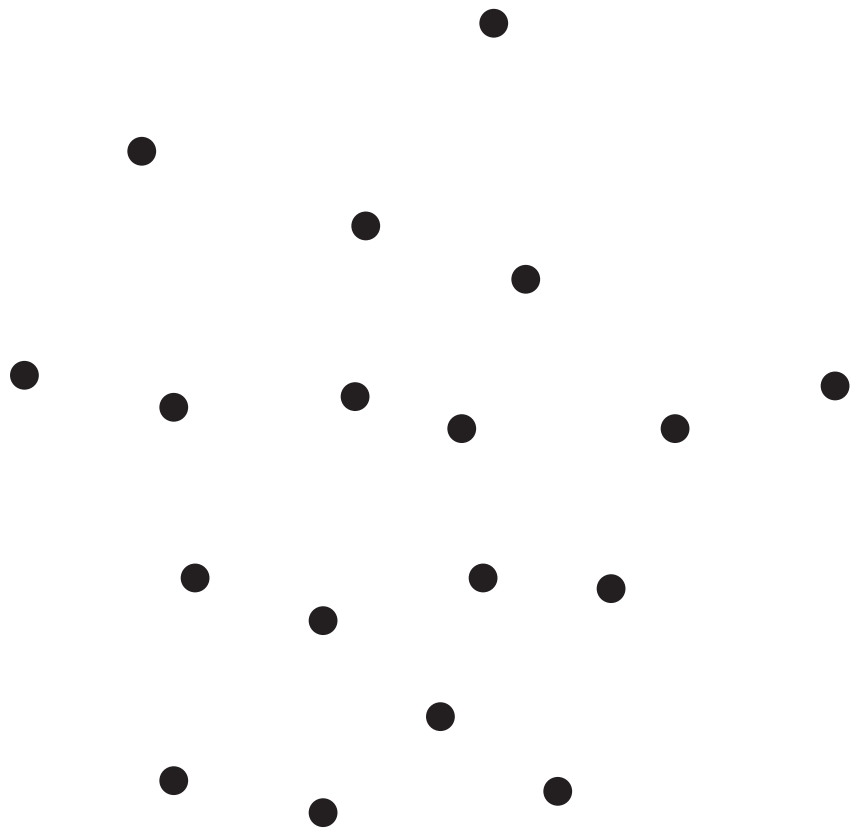
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 … $\mathrm{ast}(l) \neq \mathrm{First}(l)$, do:

… the angularly rightmost point $p_j \in P$ … spect to v.

… to l

3



Jarvis march

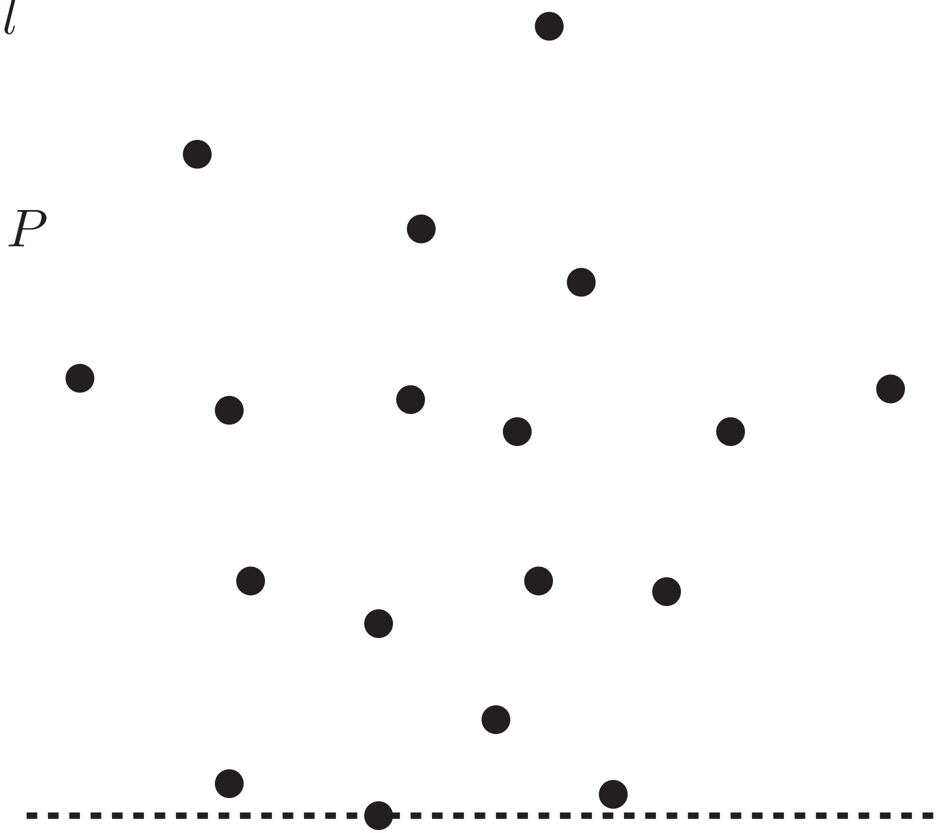
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2. ast$(l) \neq$ First(l), do:

   the angularly rightmost point $p_j \in P$ spect to v.

   to l

3.



Jarvis march

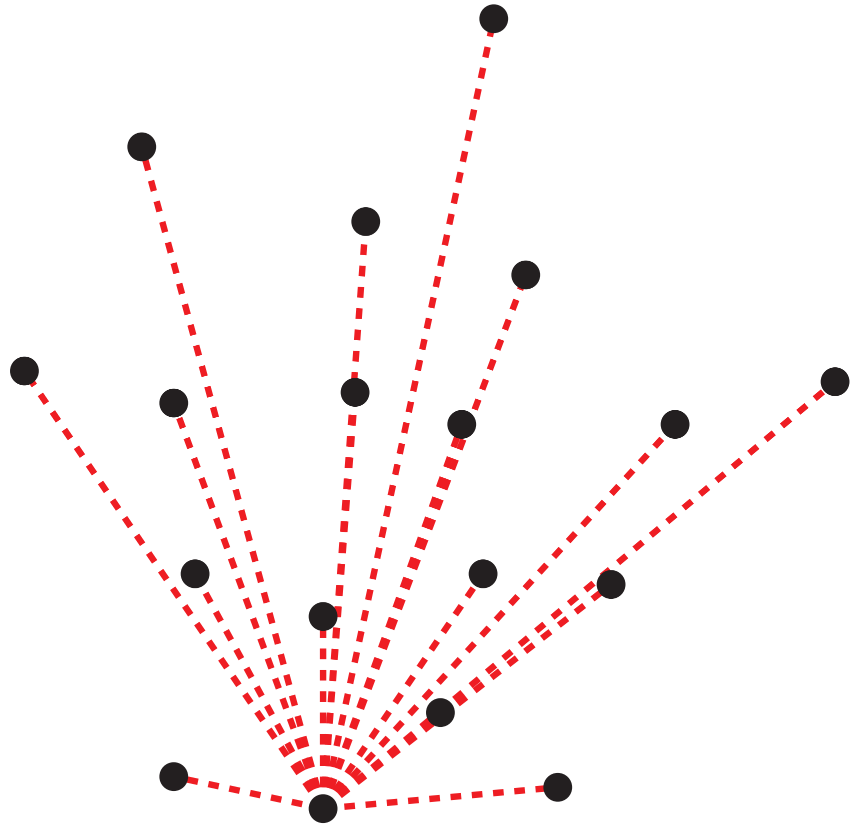
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 ... $\text{ast}(l) \neq \text{First}(l)$, do:

... the angularly rightmost point $p_j \in P$ ... spect to v.

... to l

3



Jarvis march

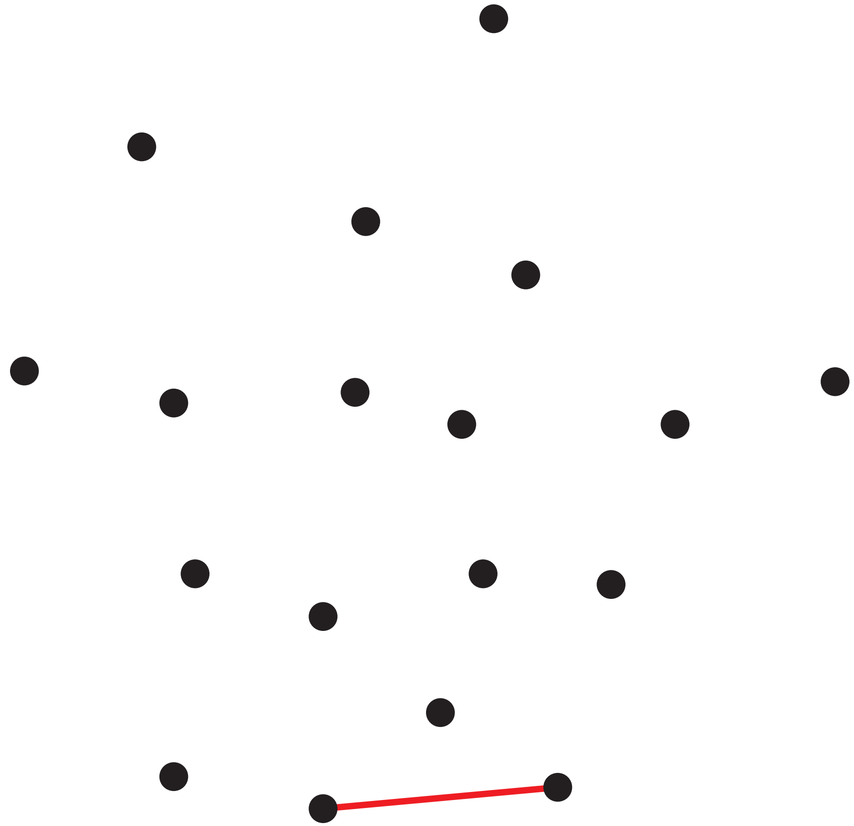
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 ast$(l) \neq \mathrm{First}(l)$, do:

the angularly rightmost point $p_j \in P$ spect to v.

to l

3

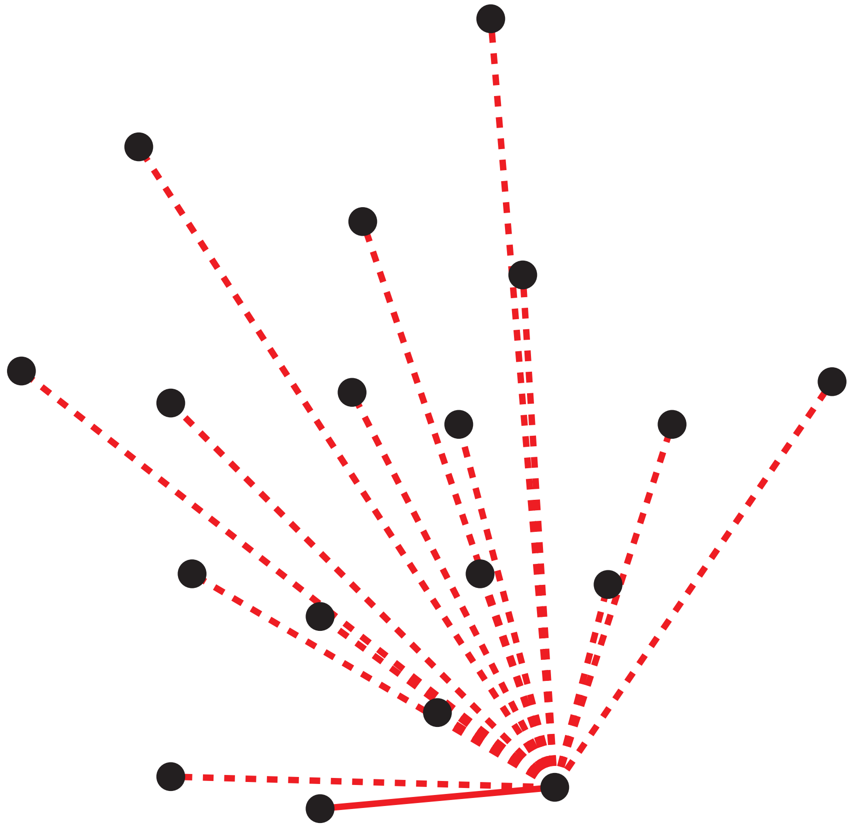


Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. ast$(l) \neq \text{First}(l)$, do:

   the angularly rightmost point $p_j \in P$ spect to v.

   to l
3.

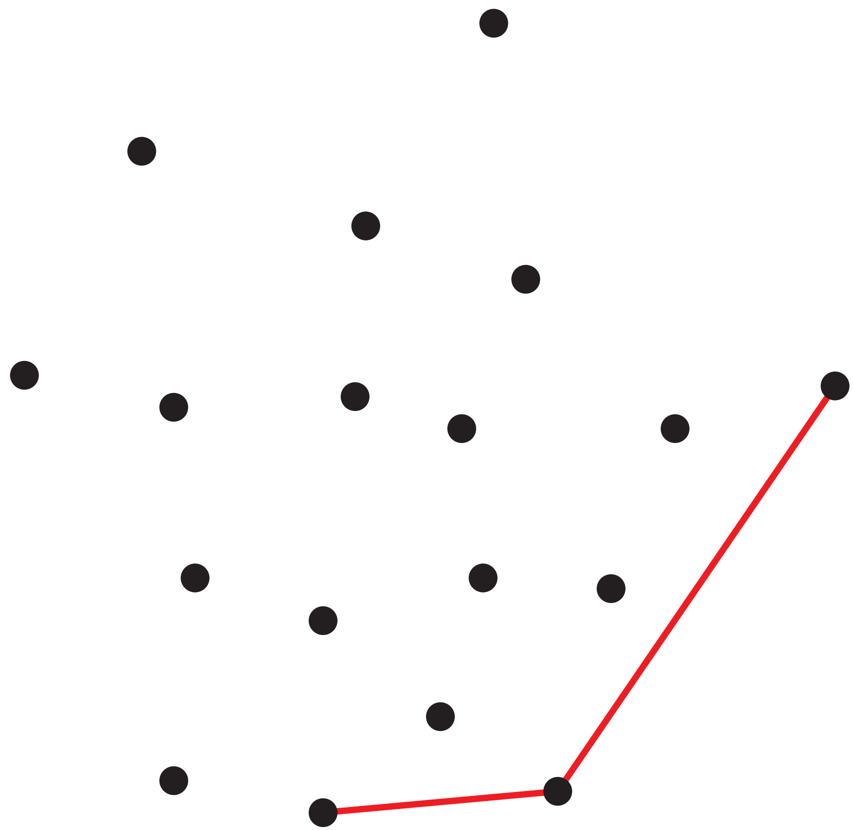


Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. ... $\mathrm{ast}(l) \neq \mathrm{First}(l)$, do:

   ... the angularly rightmost point $p_j \in P$ ... spect to v.

   ... to l
3. ...



Jarvis march

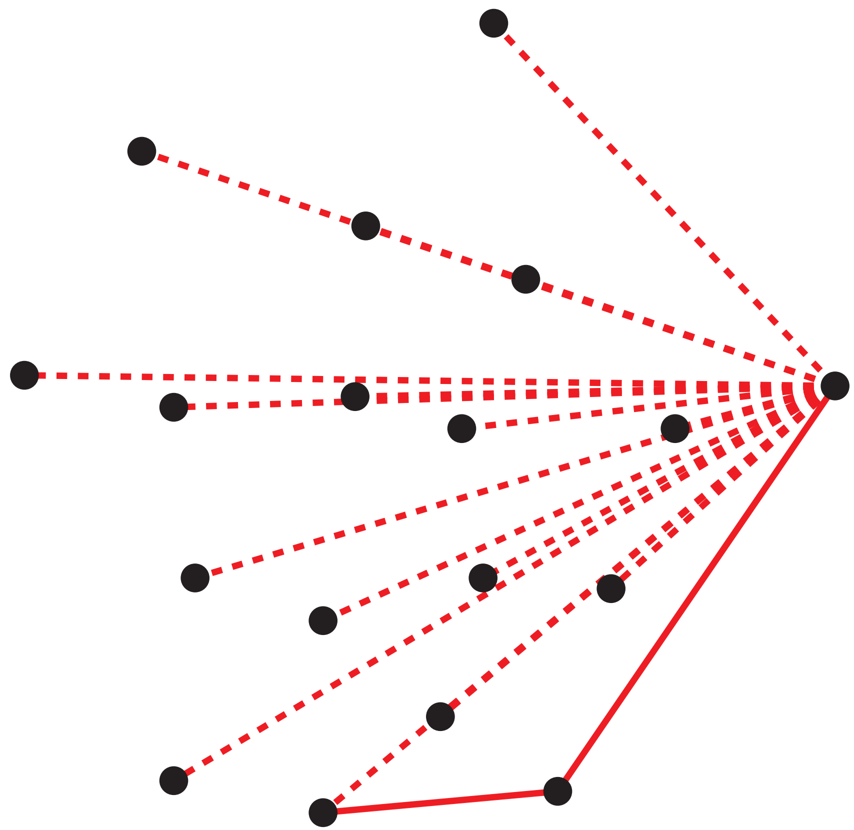
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2. ast$(l) \neq$ First(l), do:

   the angularly rightmost point $p_j \in P$ spect to v.

   to l

3.



Jarvis march

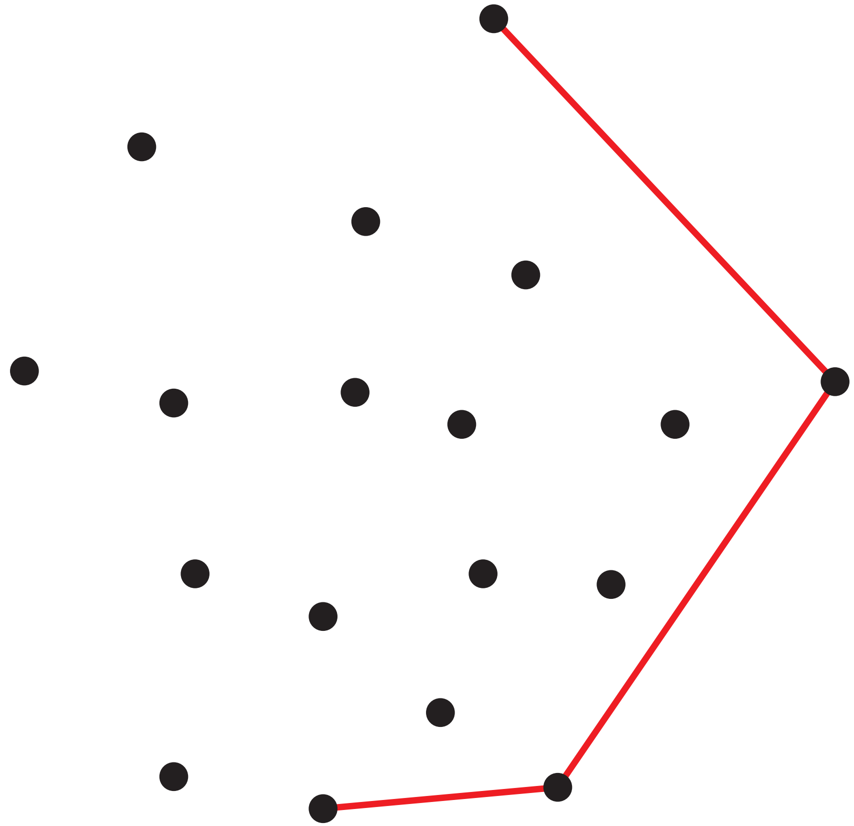
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 ast$(l) \neq \mathrm{First}(l)$, do:

the angularly rightmost point $p_j \in P$ spect to v.

to l

3



Jarvis march

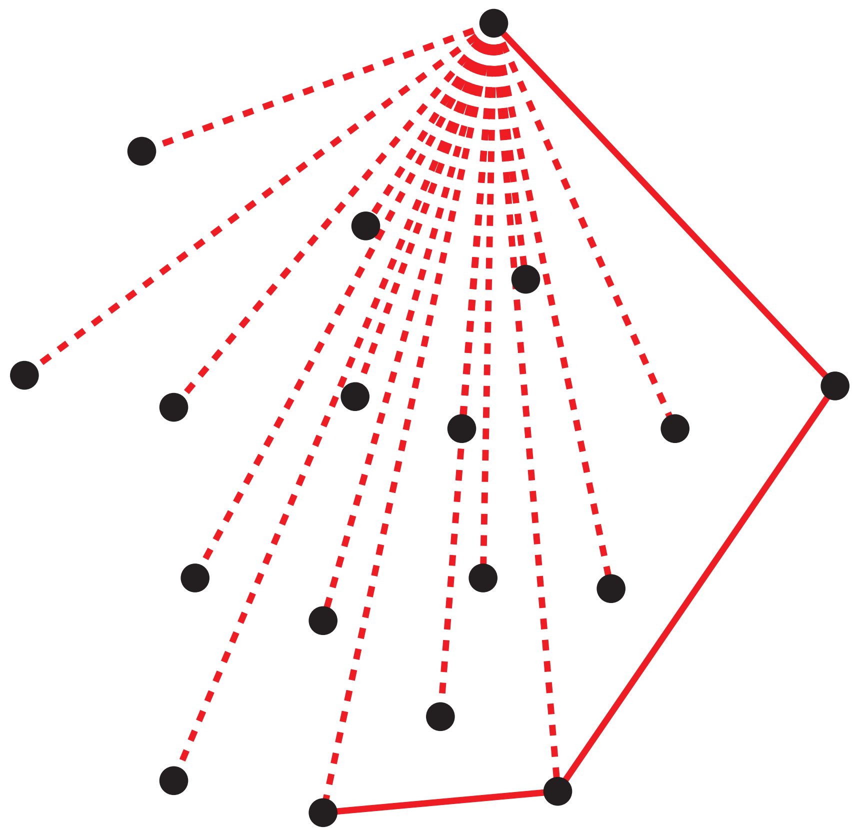
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 ... ast$(l) \neq \mathrm{First}(l)$, do:

... the angularly rightmost point $p_j \in P$ ... spect to v.

... to l

3



Jarvis march

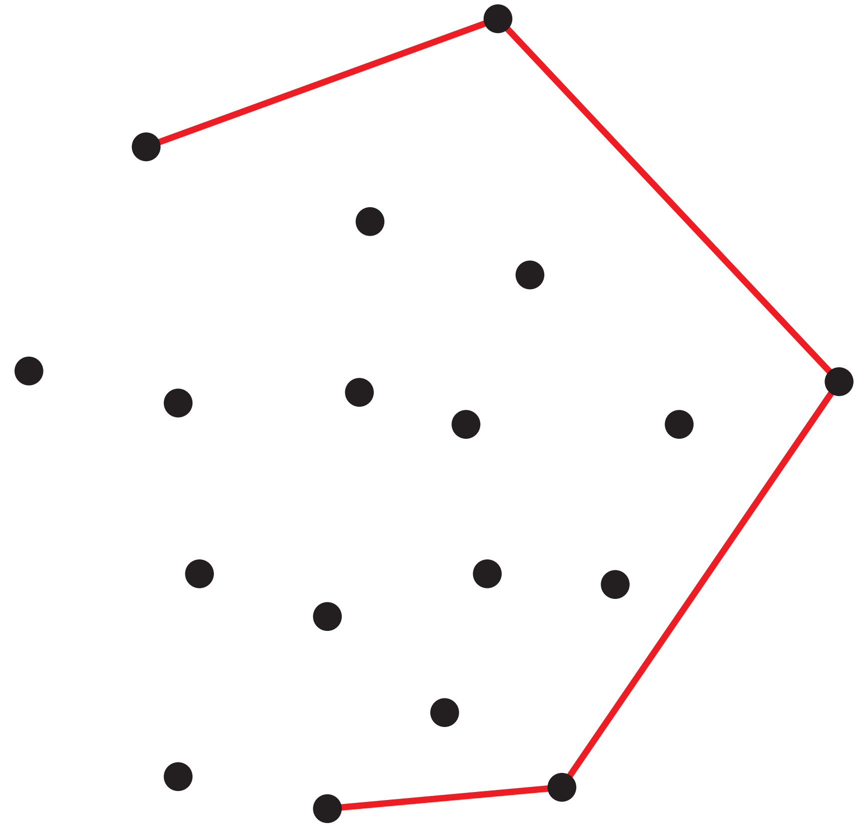
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 ... ast$(l) \neq \mathrm{First}(l)$, do:

... the angularly rightmost point $p_j \in P$ ... spect to v.

... to l

3

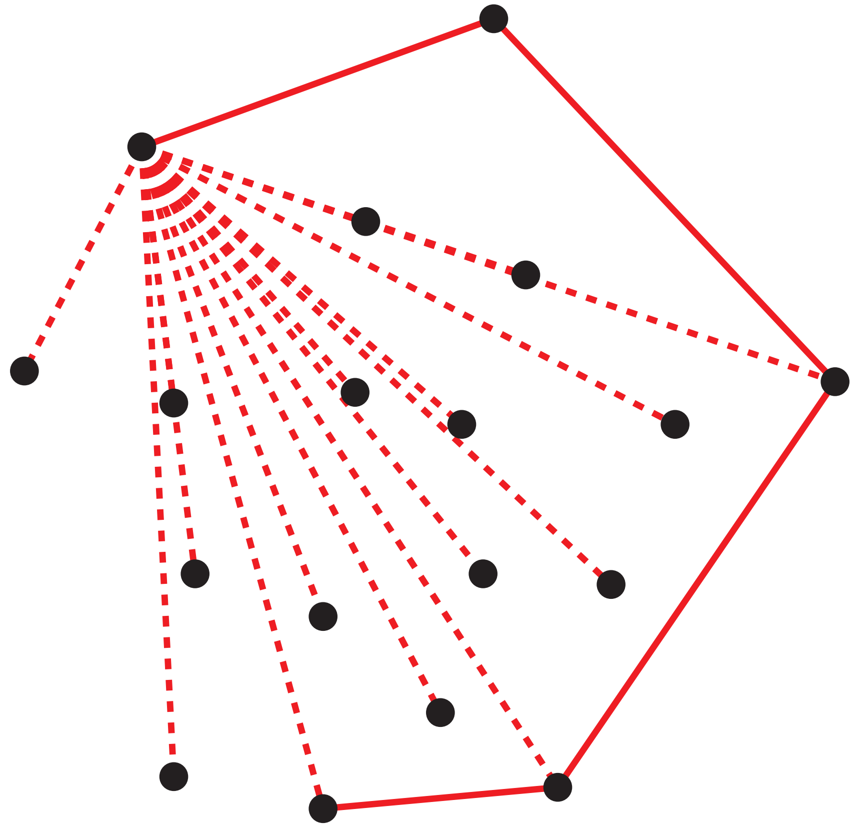


Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. ast$(l) \neq$ First(l), do:

   the angularly rightmost point $p_j \in P$ spect to v.

   to l
3.

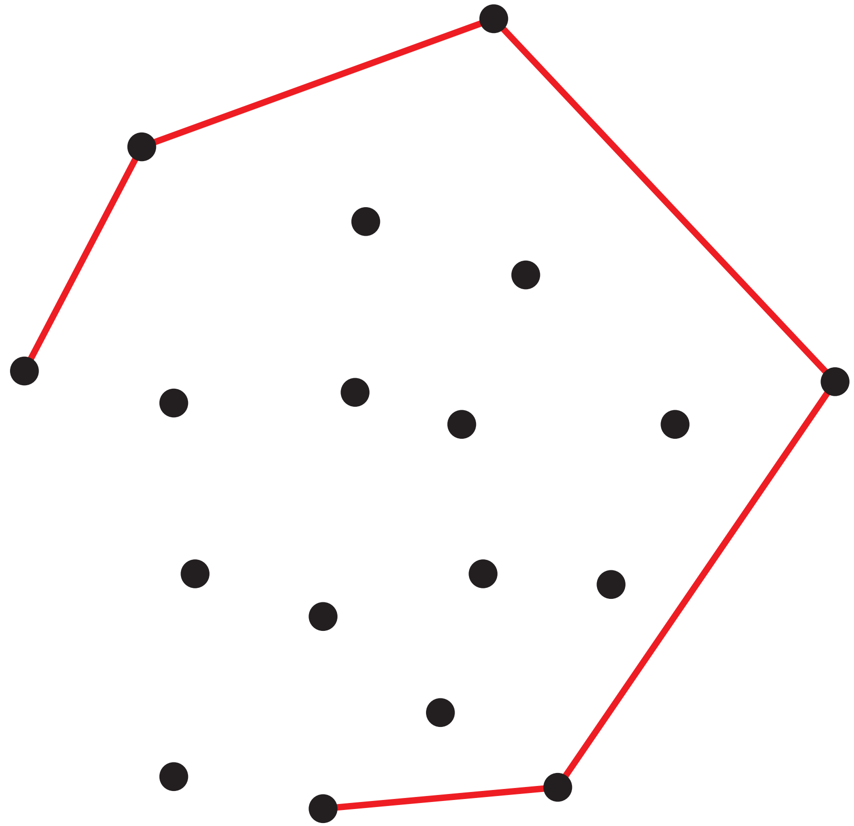


Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. ast$(l) \neq \mathrm{First}(l)$, do:

   the angularly rightmost point $p_j \in P$
   spect to v.

   to l
3.



Jarvis march

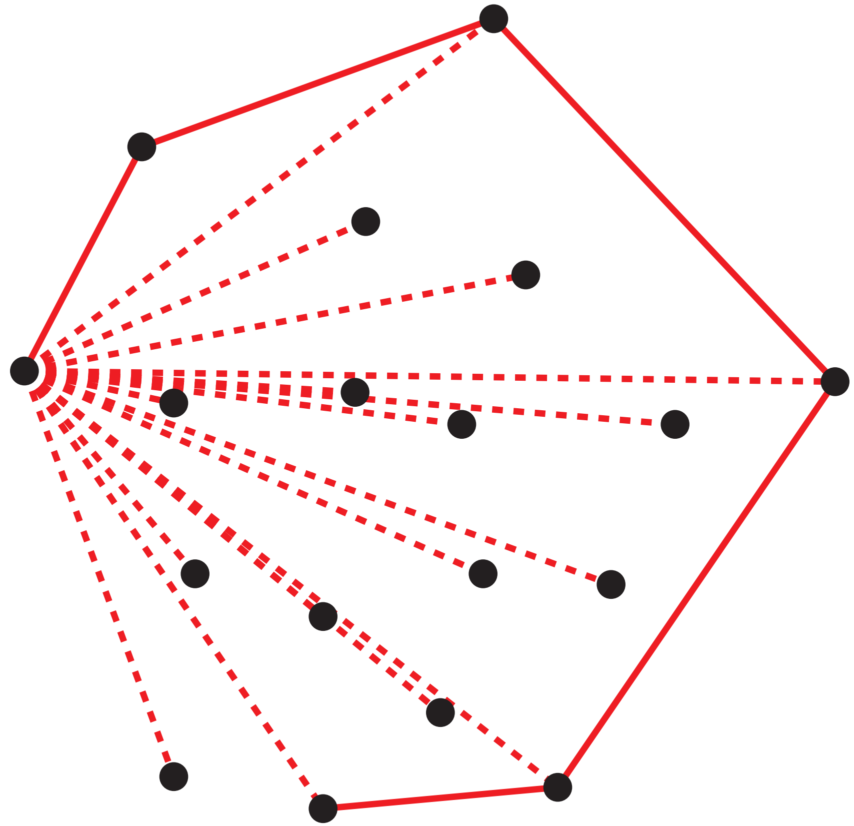
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2. ast$(l) \neq \text{First}(l)$, do:

   the angularly rightmost point $p_j \in P$ spect to v.

   to l

3.



Jarvis march

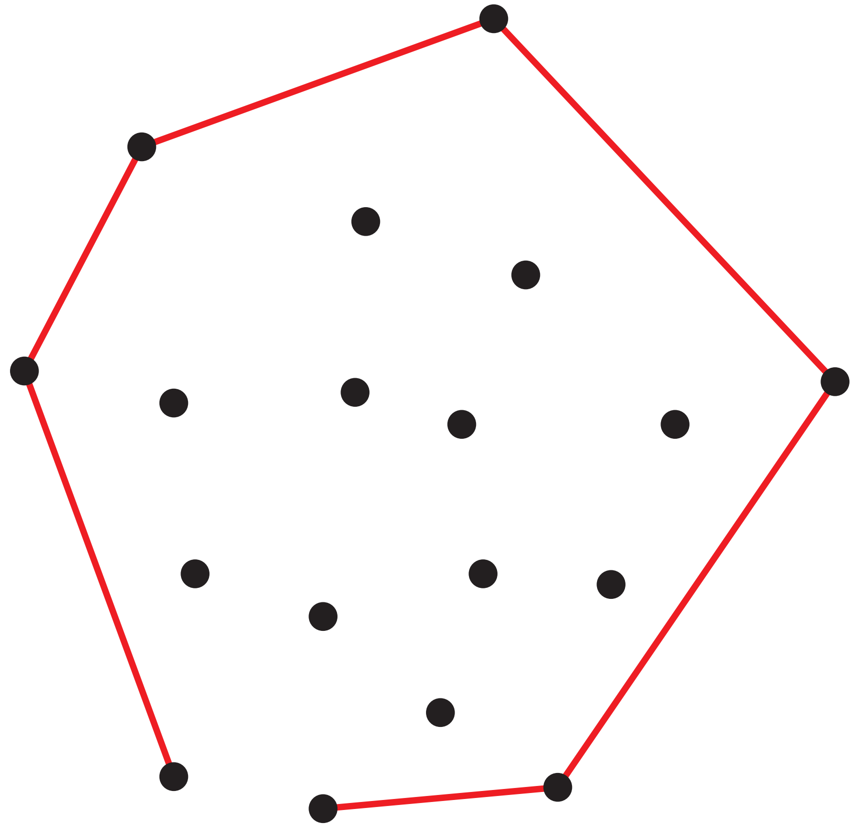
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 … $\text{Last}(l) \neq \text{First}(l)$, do:

… the angularly rightmost point $p_j \in P$ … spect to v.

… to l

3



Jarvis march

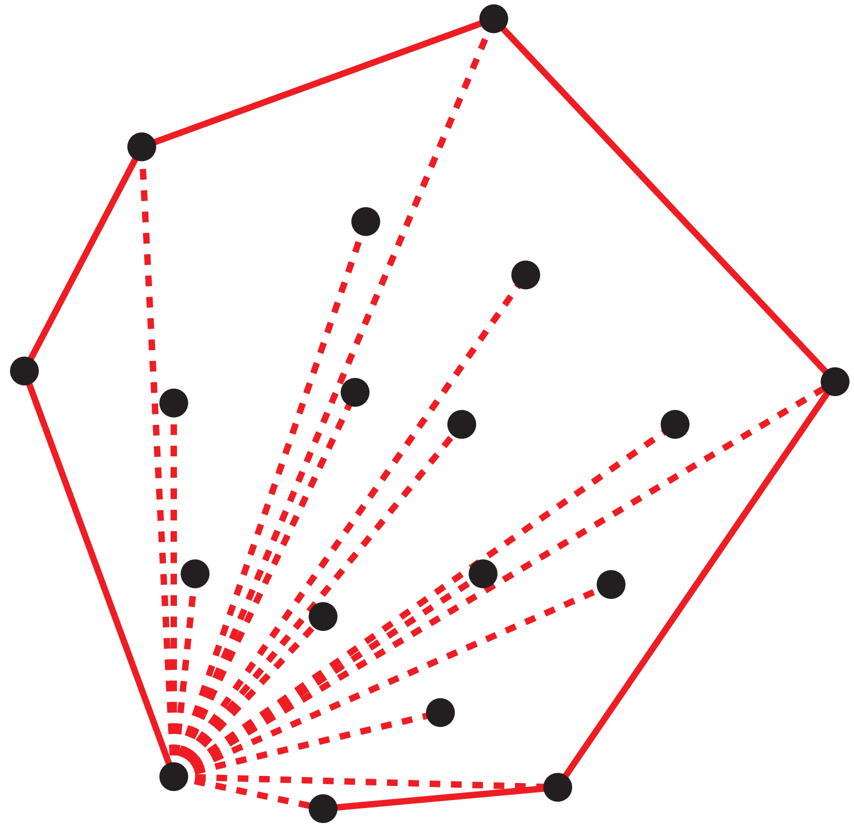
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 ast$(l) \neq$ First(l), do:

the angularly rightmost point $p_j \in P$ spect to v.

to l

3



Jarvis march

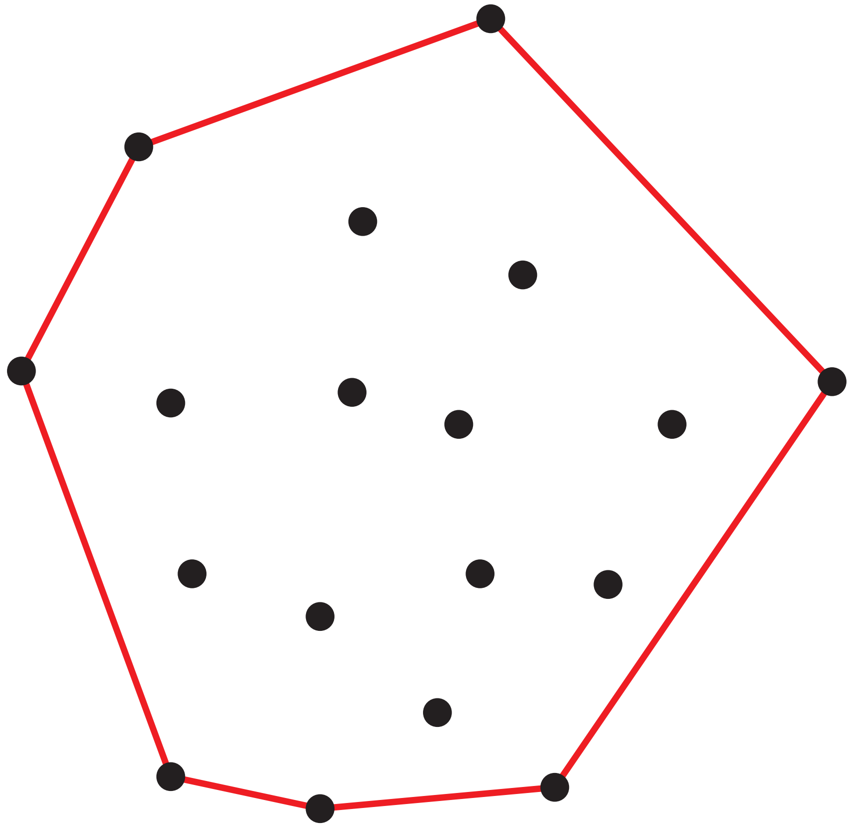
1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l

2 ... ast$(l) \neq$ First(l), do:

... the angularly rightmost point $p_j \in P$ ... spect to v.

... to l

3



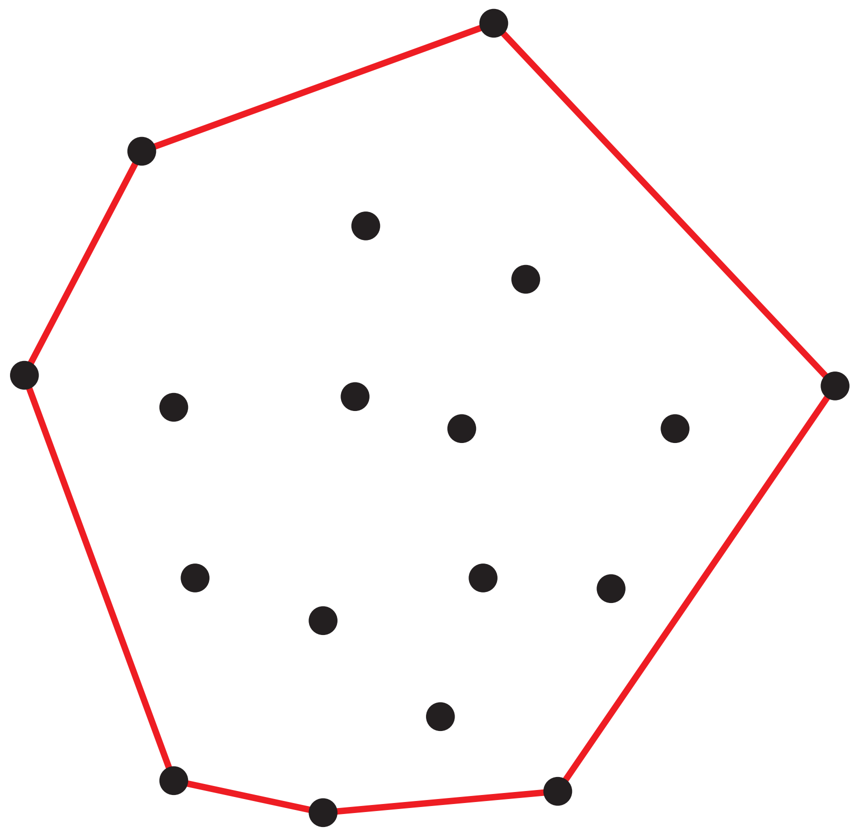
Jarvis march

1. Find a vertex of $ch(P)$ (for example, the lexicographically smaller point $p_i \in P$) and add it to l
2. ast$(l) \neq$ First(l), do:

   the angularly rightmost point $p_j \in P$ spect to v.

   to l
3.

Tim $n) = O(n^2)$



QuickHull algorithm (by prune-and-search)

QuickHull algorithm (by prune-and-search)

Initialization

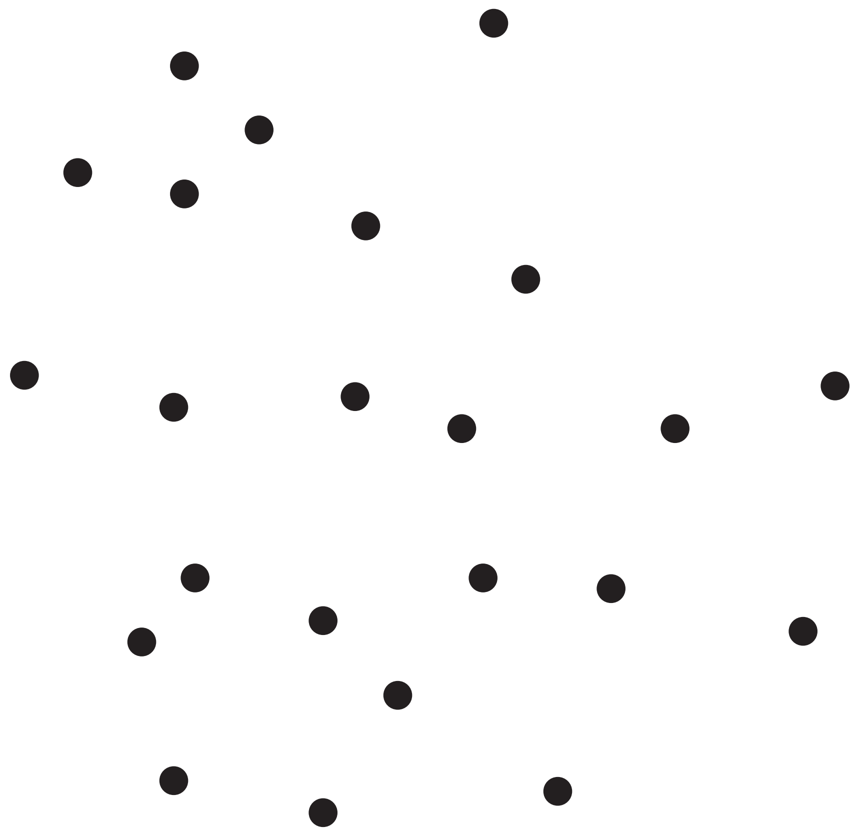
1. Find the extreme points in the horizontal and ...ctions.
2. ...e convex hull of these (between 2 ...s.
3. ...remaining points, and classify them ... their position (NE, SE, SW, NW) ... them if they lie in the interior.

CONVEX HULL IN 2D

QuickHull algorithm (by prune-and-search)

Initialization

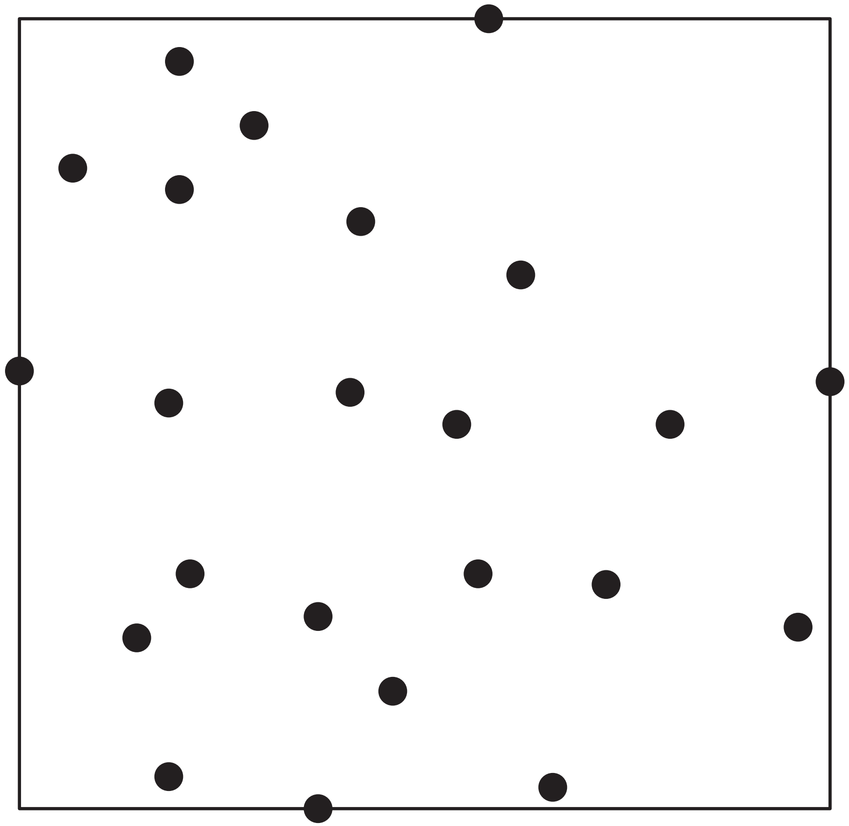
1. Find the extreme points in the horizontal and ...tions.
2. ...e convex hull of these (between 2 ...s.
3. ... remaining points, and classify them ... their position (NE, SE, SW, NW) ... them if they lie in the interior.



QuickHull algorithm (by prune-and-search)

Initialization

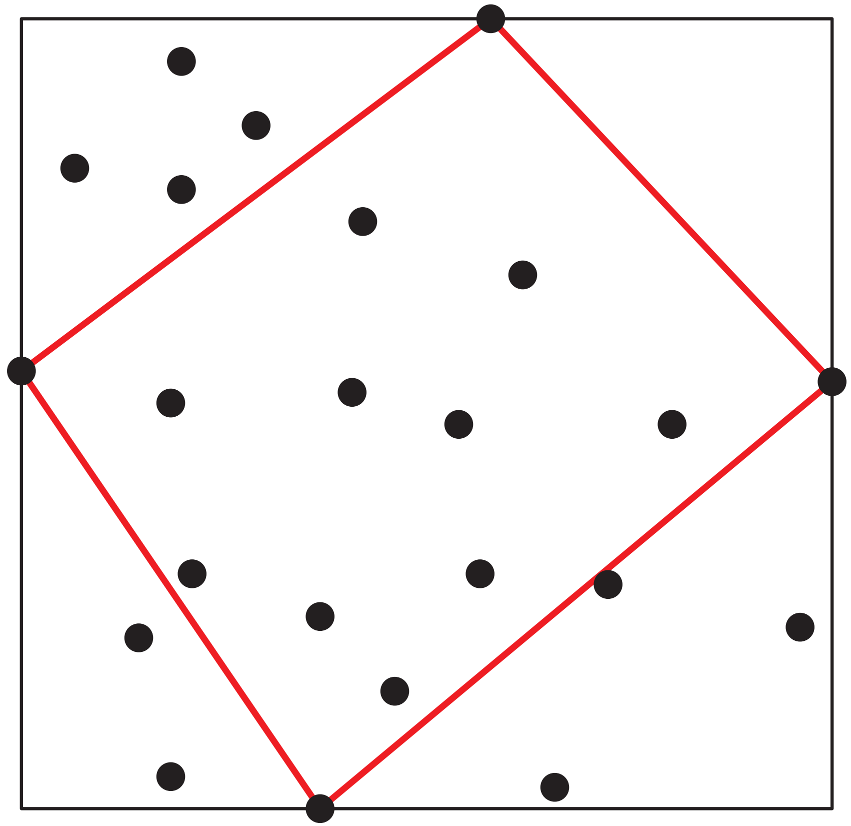
1. Find the extreme points in the horizontal and …ctions.
2. …e convex hull of these (between 2 …s.
3. …remaining points, and classify them … their position (NE, SE, SW, NW) … them if they lie in the interior.



QuickHull algorithm (by prune-and-search)

Initialization

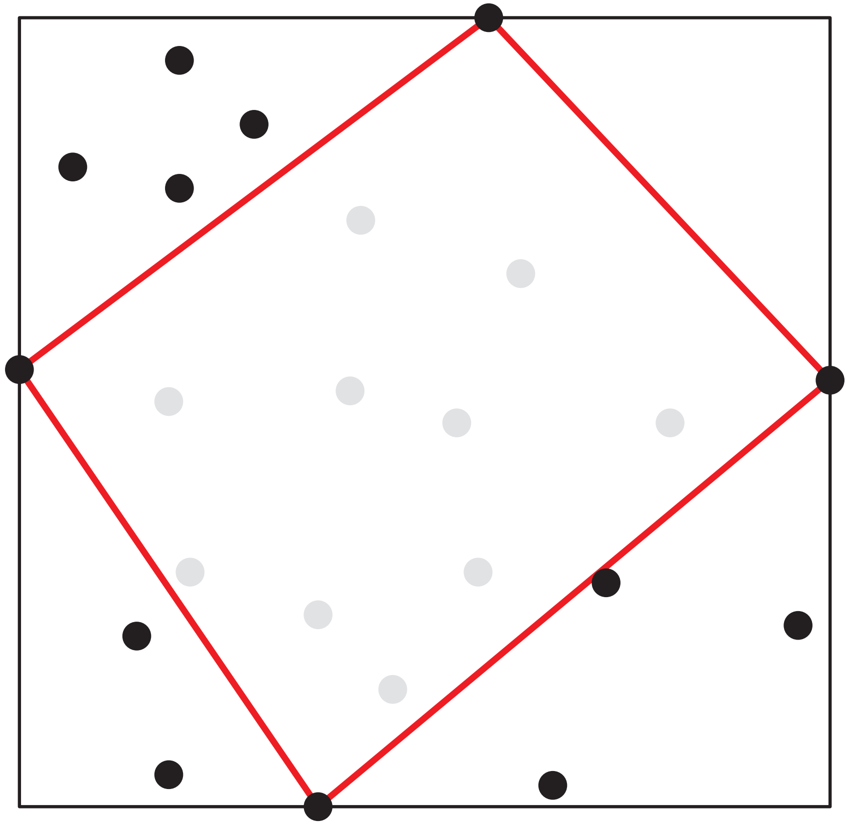
1. Find the extreme points in the horizontal and ...tions.
2. ...e convex hull of these (between 2 ...s.
3. ... remaining points, and classify them ... their position (NE, SE, SW, NW) ... them if they lie in the interior.



QuickHull algorithm (by prune-and-search)

Initialization

1. Find the extreme points in the horizontal and … ctions.
2. … e convex hull of these (between 2 … s.
3. … remaining points, and classify them … their position (NE, SE, SW, NW) … them if they lie in the interior.

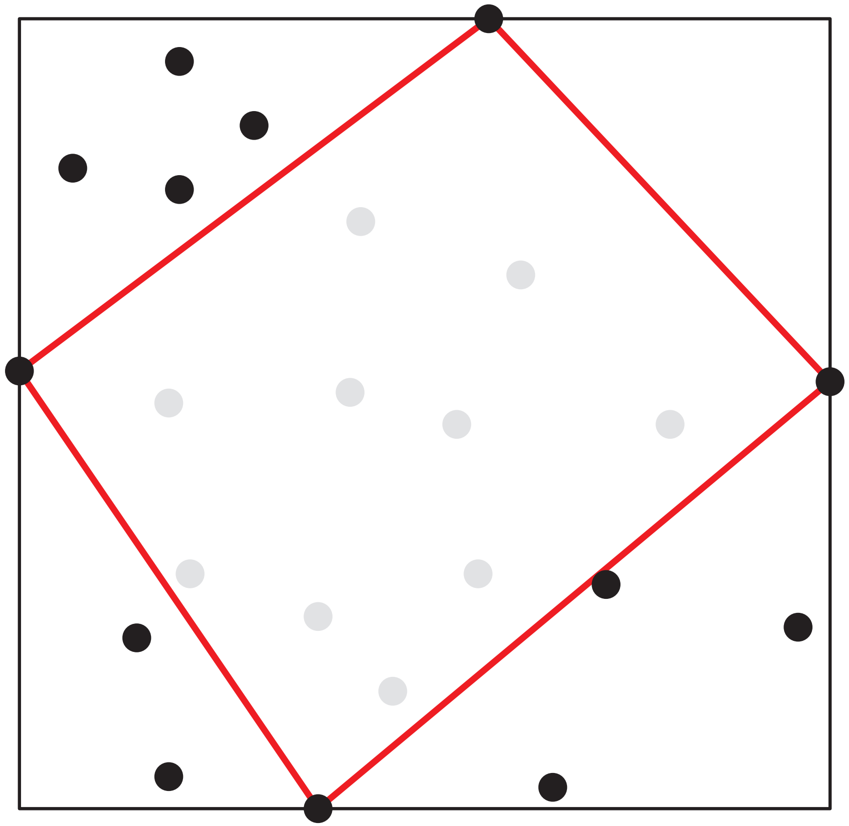


QuickHull algorithm (by prune-and-search)

Initialization

1. Find the extreme points in the horizontal and … tions.
2. … e convex hull of these (between 2 … s.
3. … remaining points, and classify them … their position (NE, SE, SW, NW) … them if they lie in the interior.

Run… this step: $O(n)$



QuickHull algorithm (by prune-and-search)

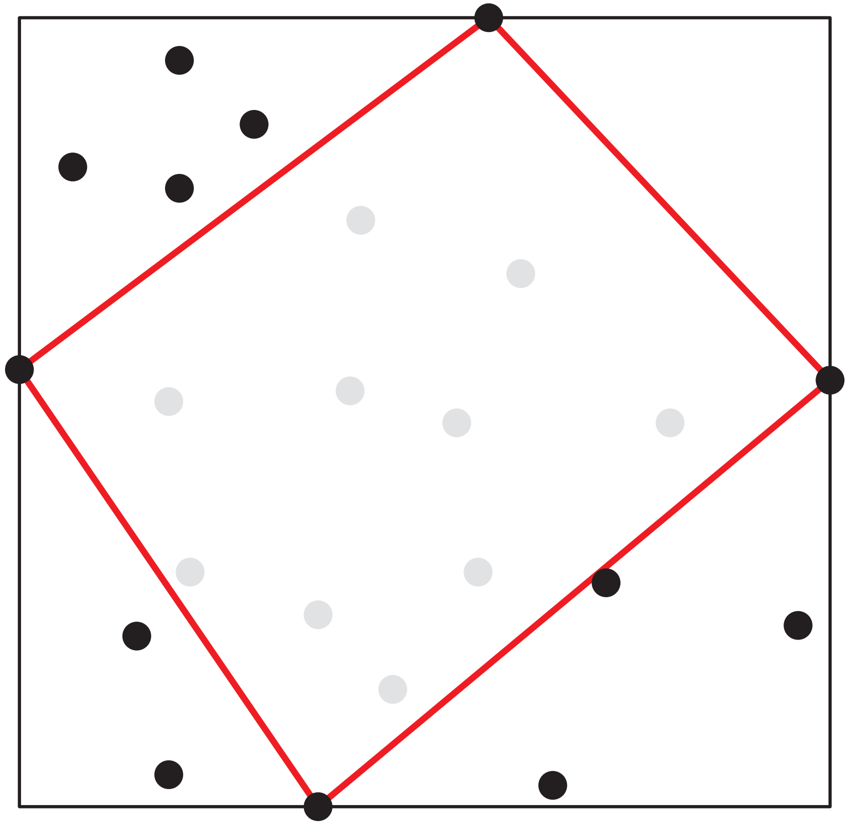
Advance

Recursively, do:

1. ...oints lying in each region, find the ...t in the direction orthogonal to the ...termines the region.

2. ... extreme point with te endpoints of ...d update the convex hull.

3. ...emaining points of each region, and ... according to their position (left or ...inate them if they lie in the interior ... created triangle.



QuickHull algorithm (by prune-and-search)

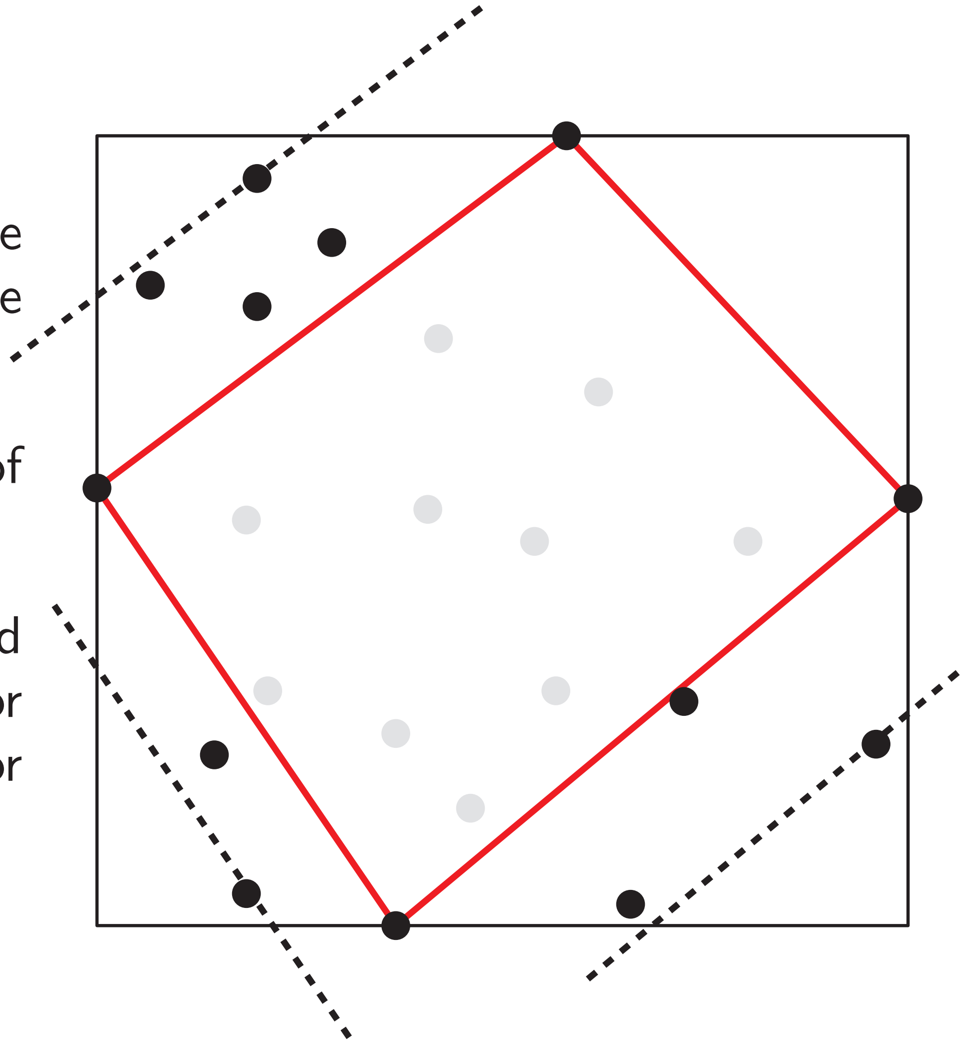
Advance

Recursively, do:

1. … oints lying in each region, find the … t in the direction orthogonal to the … termines the region.

2. … extreme point with te endpoints of … d update the convex hull.

3. … emaining points of each region, and … according to their position (left or … inate them if they lie in the interior … created triangle.



QuickHull algorithm (by prune-and-search)

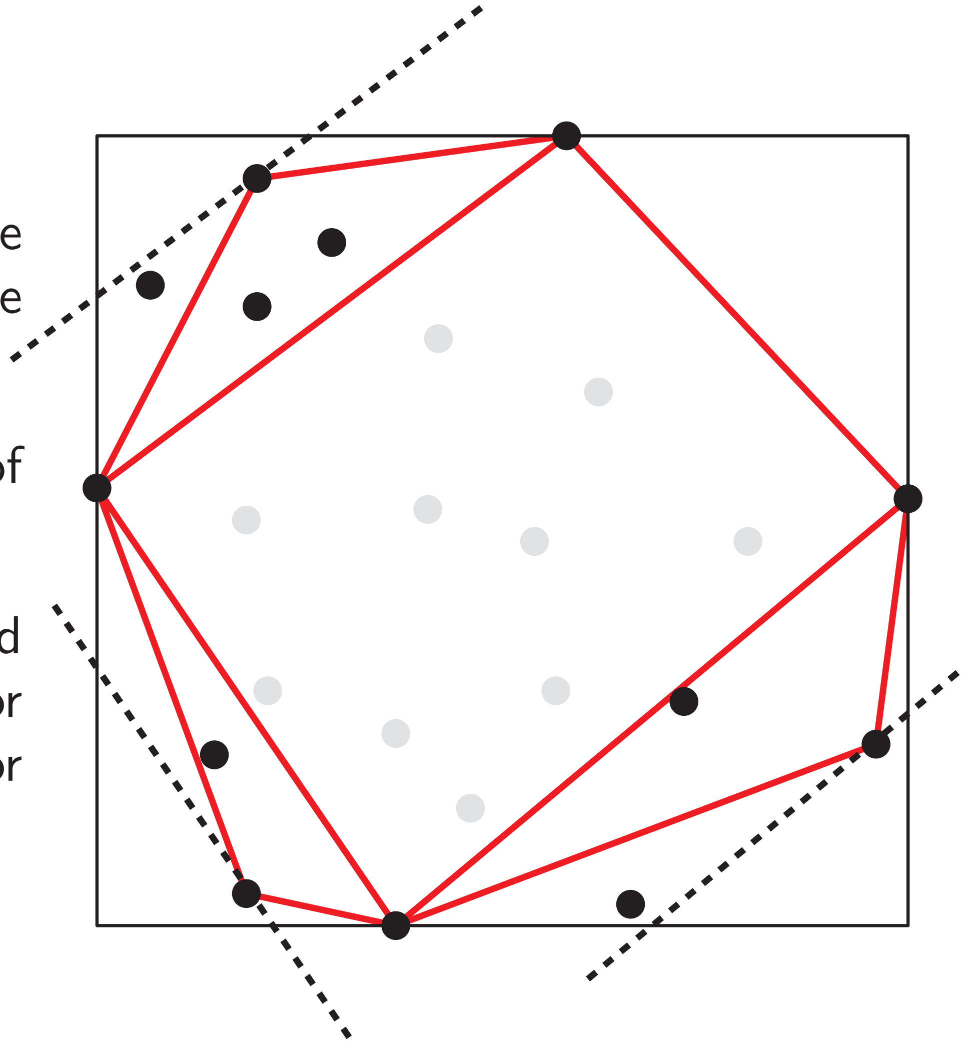
Advance

Recursively, do:

1. ...oints lying in each region, find the ...t in the direction orthogonal to the ...termines the region.

2. ... extreme point with te endpoints of ...d update the convex hull.

3. ...emaining points of each region, and ... according to their position (left or ...inate them if they lie in the interior ... created triangle.



QuickHull algorithm (by prune-and-search)

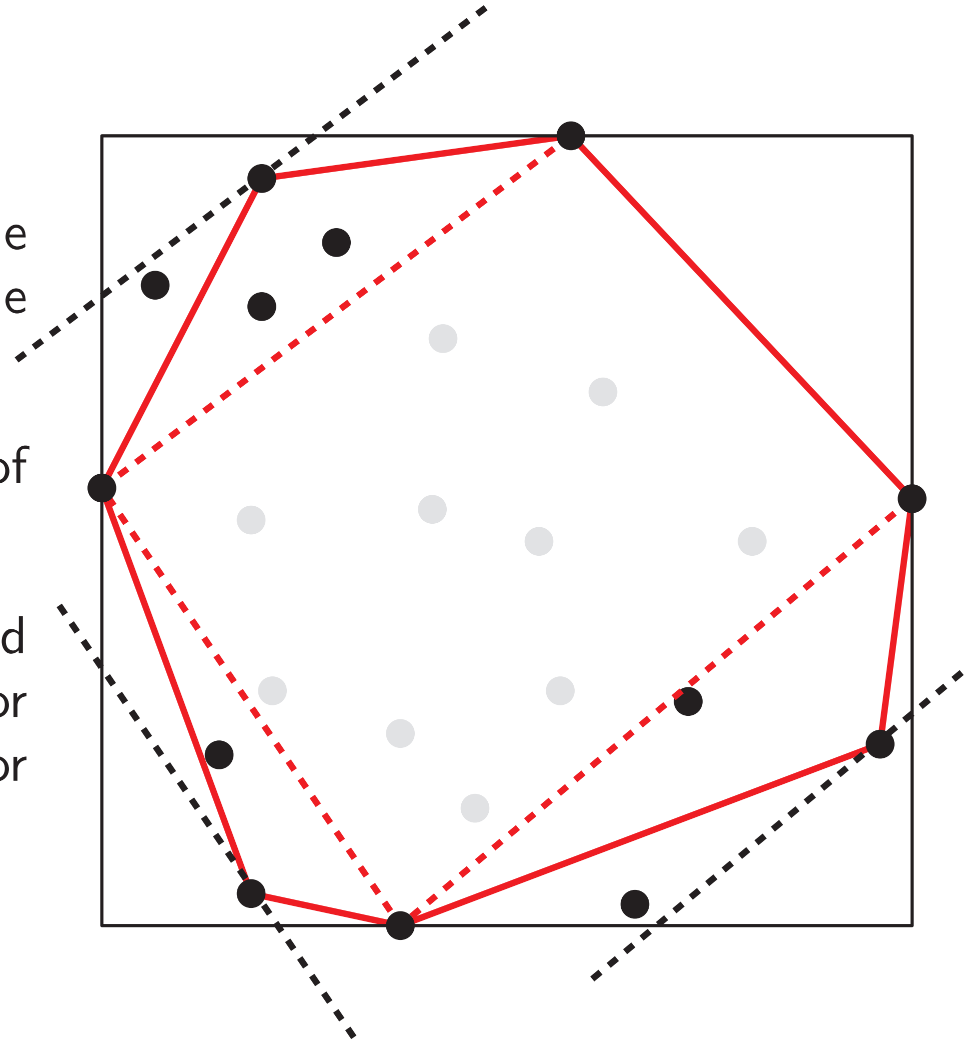
Advance

Recursively, do:

1. ...oints lying in each region, find the ...t in the direction orthogonal to the ...termines the region.

2. ... extreme point with te endpoints of ...d update the convex hull.

3. ...emaining points of each region, and ... according to their position (left or ...inate them if they lie in the interior ... created triangle.



CONVEX HULL IN 2D

QuickHull algorithm (by prune-and-search)

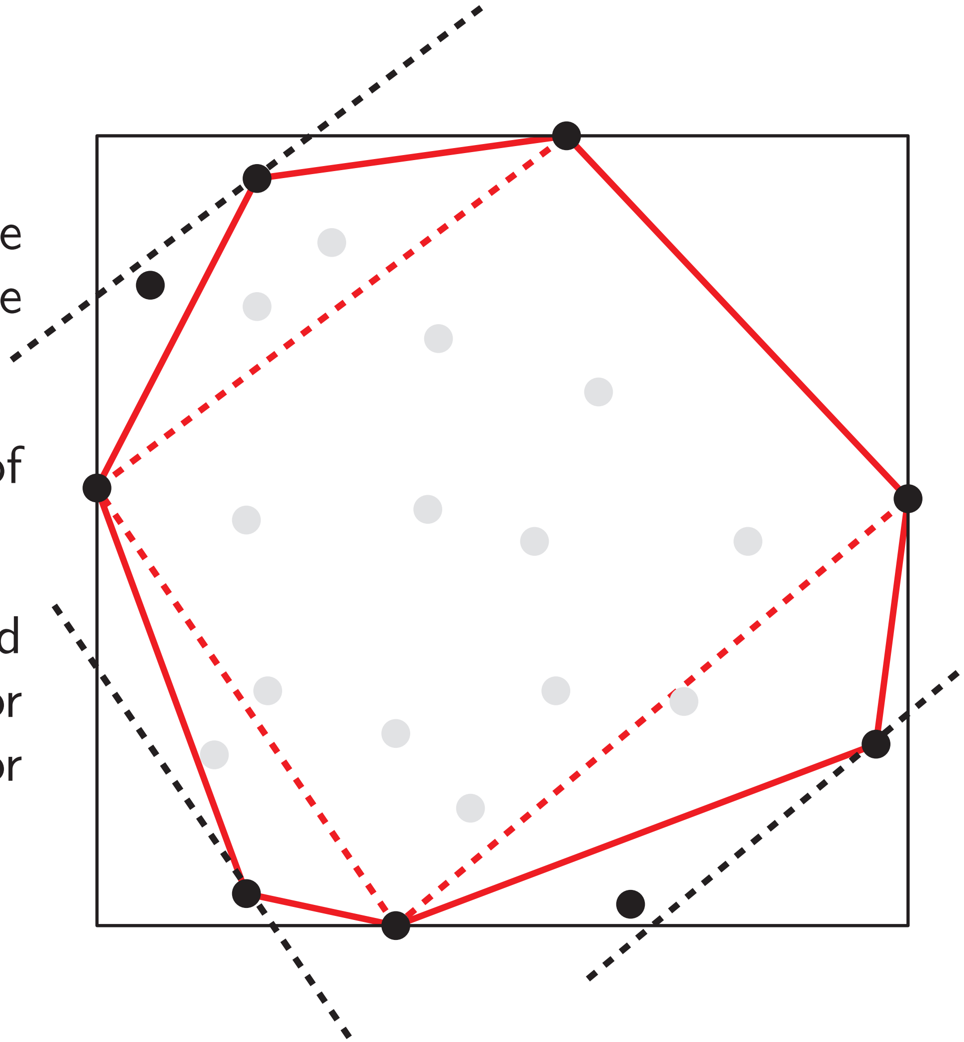
Advance

Recursively, do:

1 ... oints lying in each region, find the ... nt in the direction orthogonal to the ... termines the region.

2 ... extreme point with te endpoints of ... d update the convex hull.

3 ... remaining points of each region, and ... n according to their position (left or ... inate them if they lie in the interior ... created triangle.

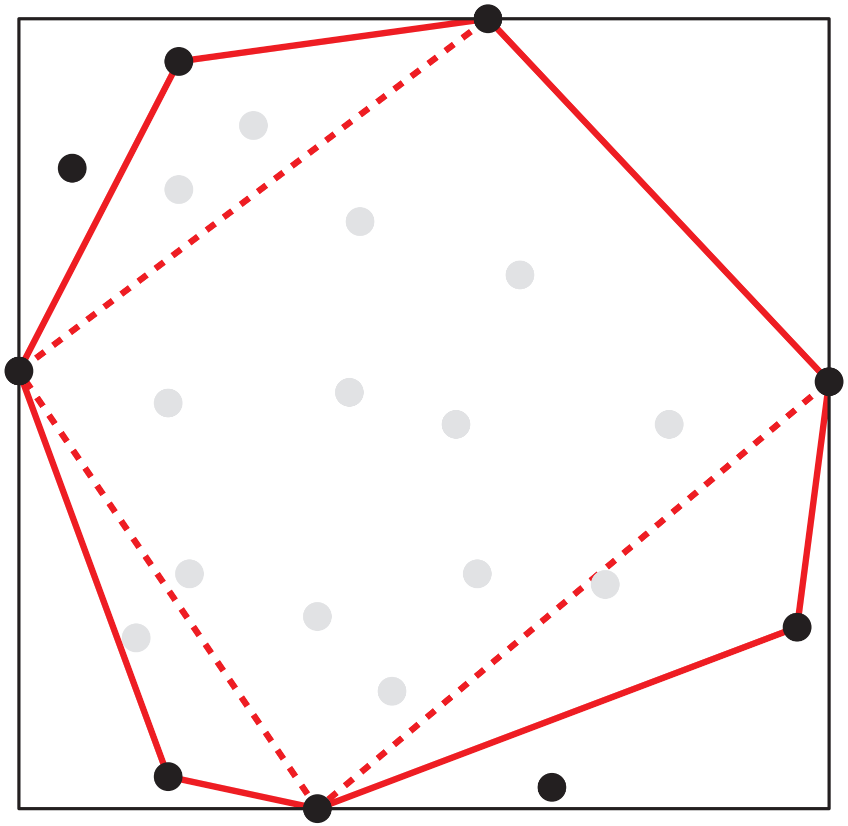


QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. ...oints lying in each region, find the ...t in the direction orthogonal to the ...termines the region.
2. ... extreme point with te endpoints of ...d update the convex hull.
3. ...emaining points of each region, and ... according to their position (left or ...inate them if they lie in the interior ... created triangle.



QuickHull algorithm (by prune-and-search)

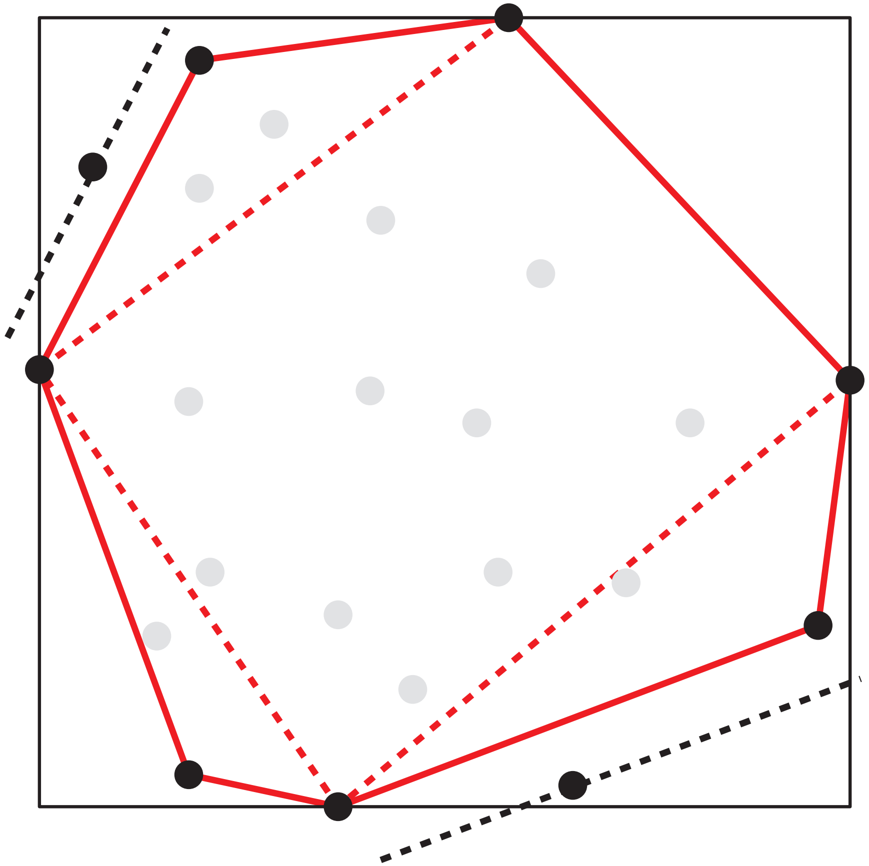
Advance

Recursively, do:

1 … oints lying in each region, find the … t in the direction orthogonal to the … termines the region.

2 … extreme point with te endpoints of … d update the convex hull.

3 … emaining points of each region, and … according to their position (left or … inate them if they lie in the interior … created triangle.

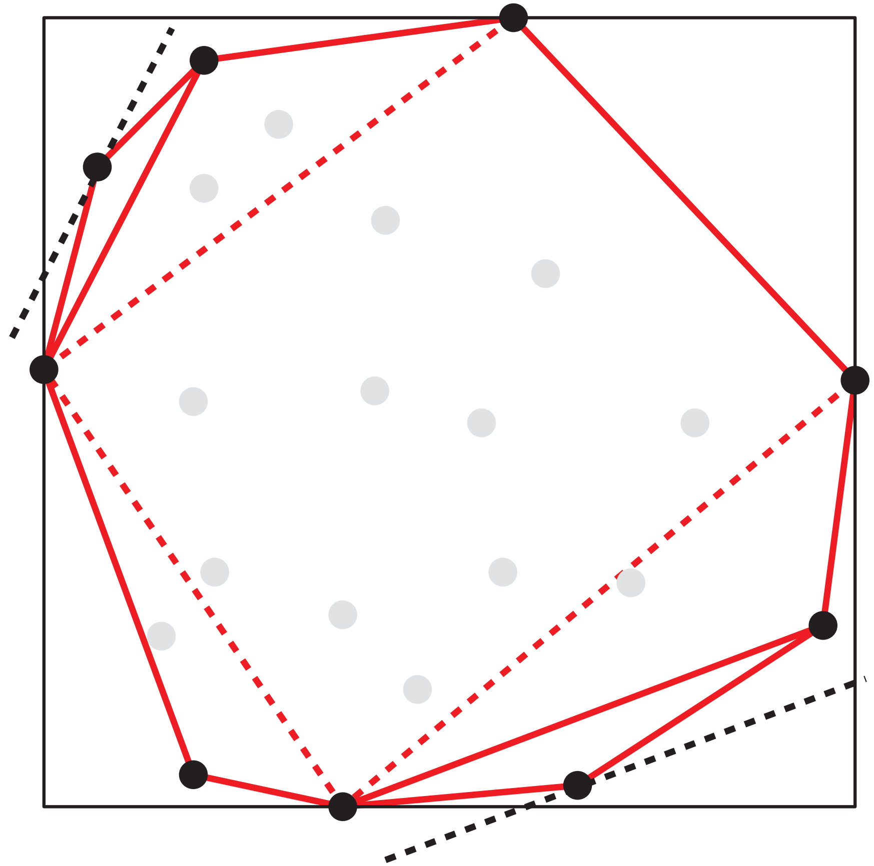


QuickHull algorithm (by prune-and-search)

Advance

Recursively, do:

1. …oints lying in each region, find the …t in the direction orthogonal to the …termines the region.
2. … extreme point with te endpoints of …d update the convex hull.
3. …emaining points of each region, and … according to their position (left or …inate them if they lie in the interior … created triangle.



QuickHull algorithm (by prune-and-search)

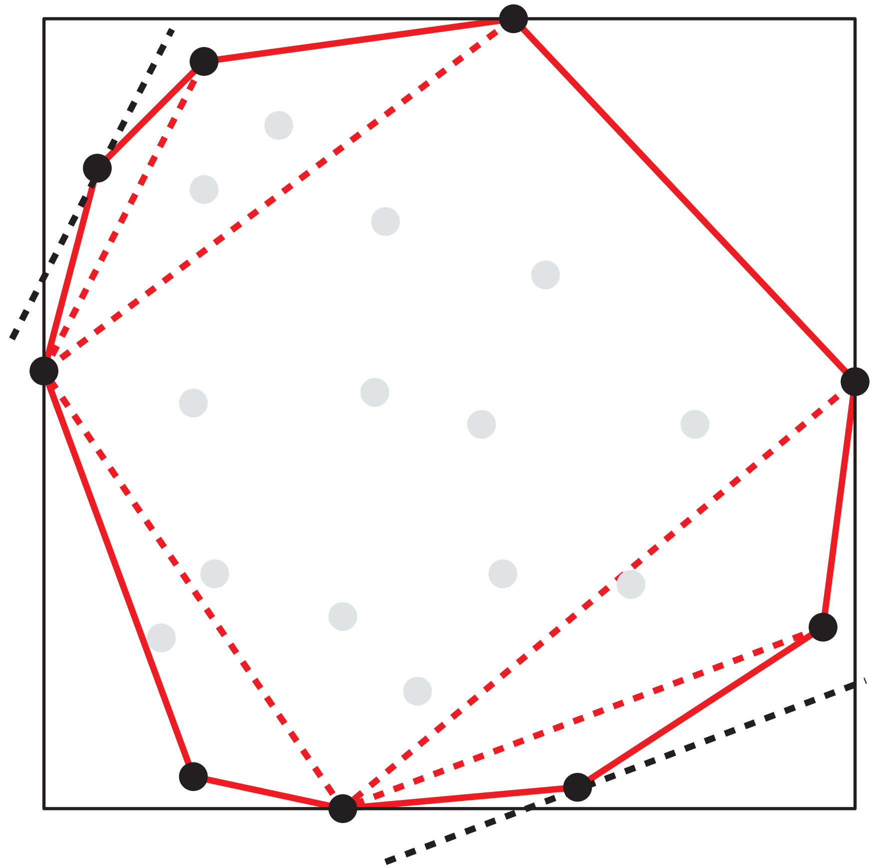
Advance

Recursively, do:

1. ...oints lying in each region, find the ...t in the direction orthogonal to the ...termines the region.

2. ... extreme point with te endpoints of ...d update the convex hull.

3. ...emaining points of each region, and ... according to their position (left or ...inate them if they lie in the interior ... created triangle.



QuickHull algorithm (by prune-and-search)

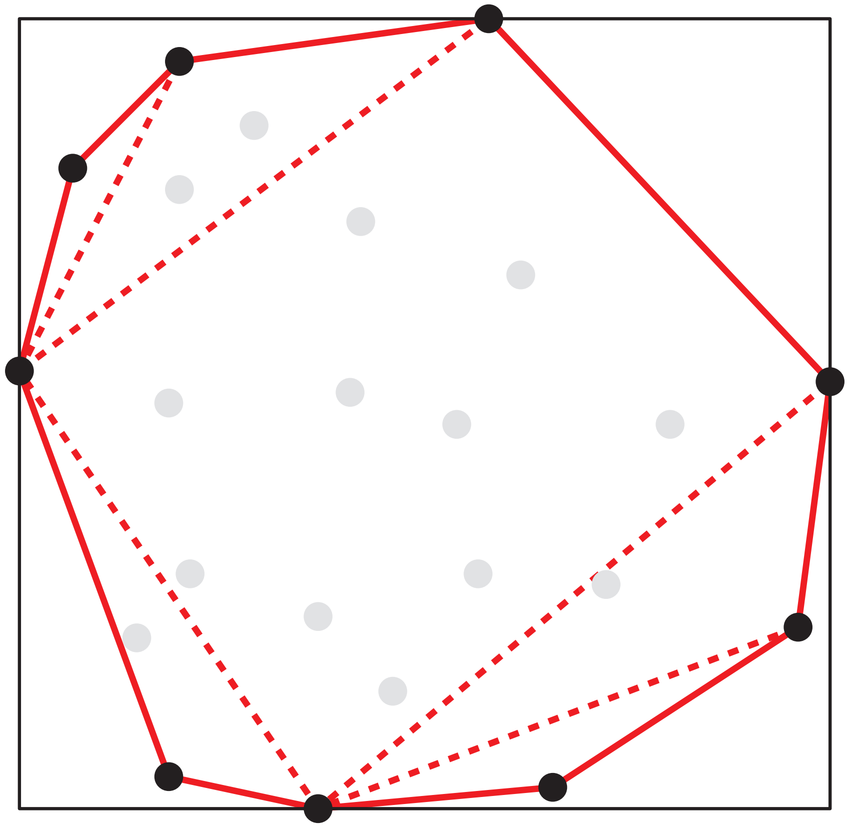
Advance

Recursively, do:

1. oints lying in each region, find the t in the direction orthogonal to the termines the region.

2. extreme point with te endpoints of d update the convex hull.

3. emaining points of each region, and according to their position (left or inate them if they lie in the interior created triangle.



QuickHull algorithm (by prune-and-search)

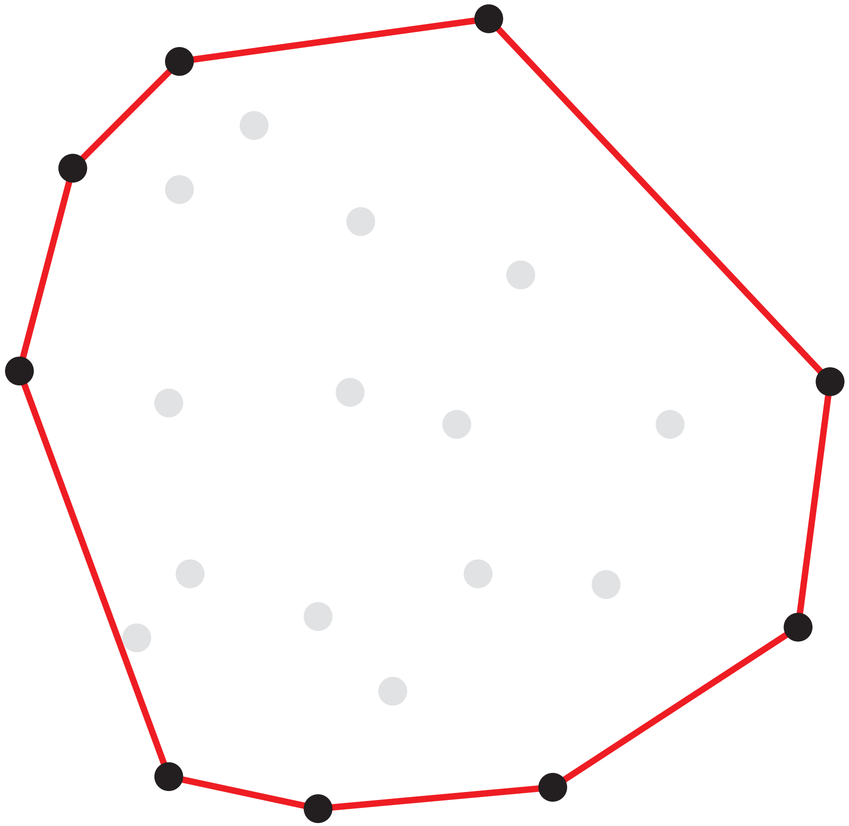
Advance

Recursively, do:

1. ...oints lying in each region, find the ...t in the direction orthogonal to the ...termines the region.

2. ... extreme point with te endpoints of ...d update the convex hull.

3. ...emaining points of each region, and ... according to their position (left or ...inate them if they lie in the interior ... created triangle.



CONVEX HULL IN 2D

QuickHull algorithm (by prune-and-search)

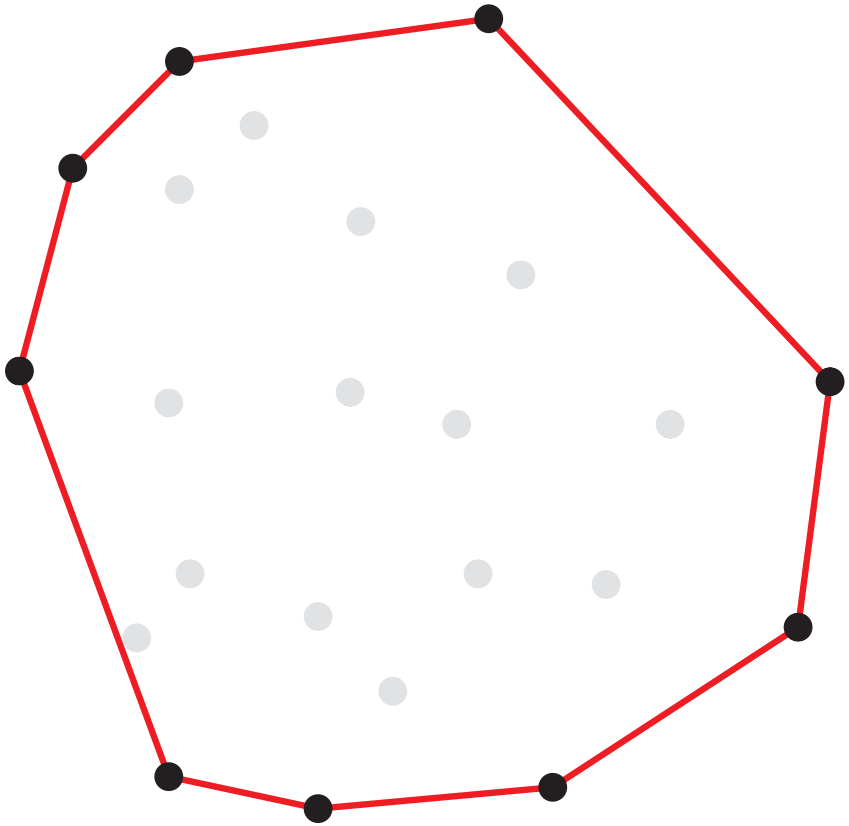
Advance

Recursively, do:

1. ...oints lying in each region, find the ...t in the direction orthogonal to the ...termines the region.

2. ... extreme point with te endpoints of ...d update the convex hull.

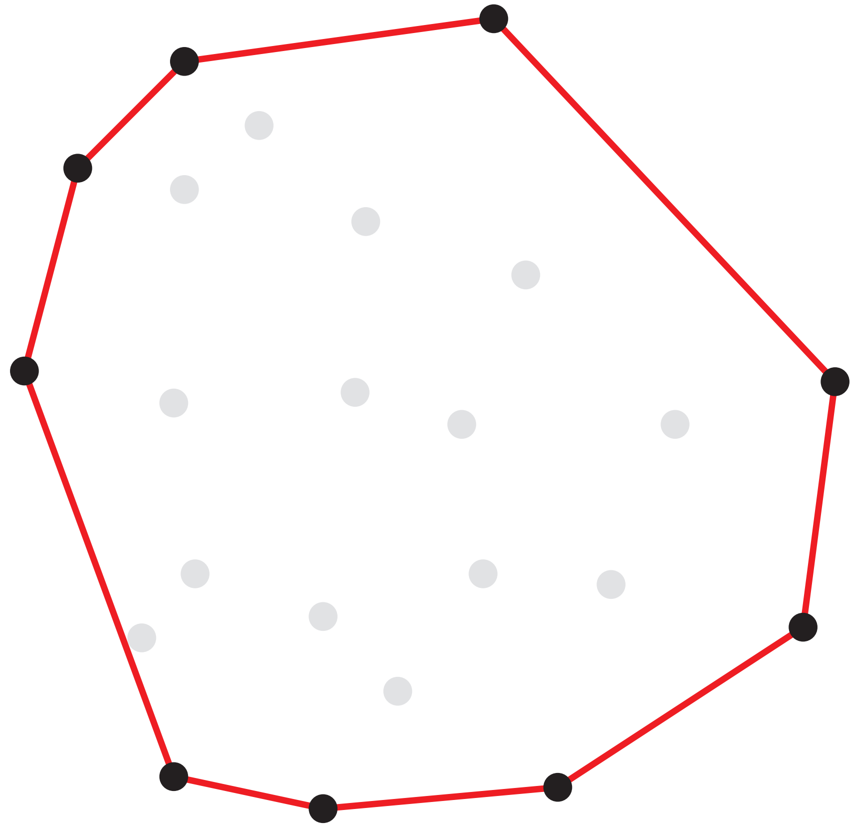
3. ...emaining points of each region, and ... according to their position (left or ...inate them if they lie in the interior ... created triangle.



Run... this step: $O(n^2)$

QuickHull algorithm (by prune-and-search)

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

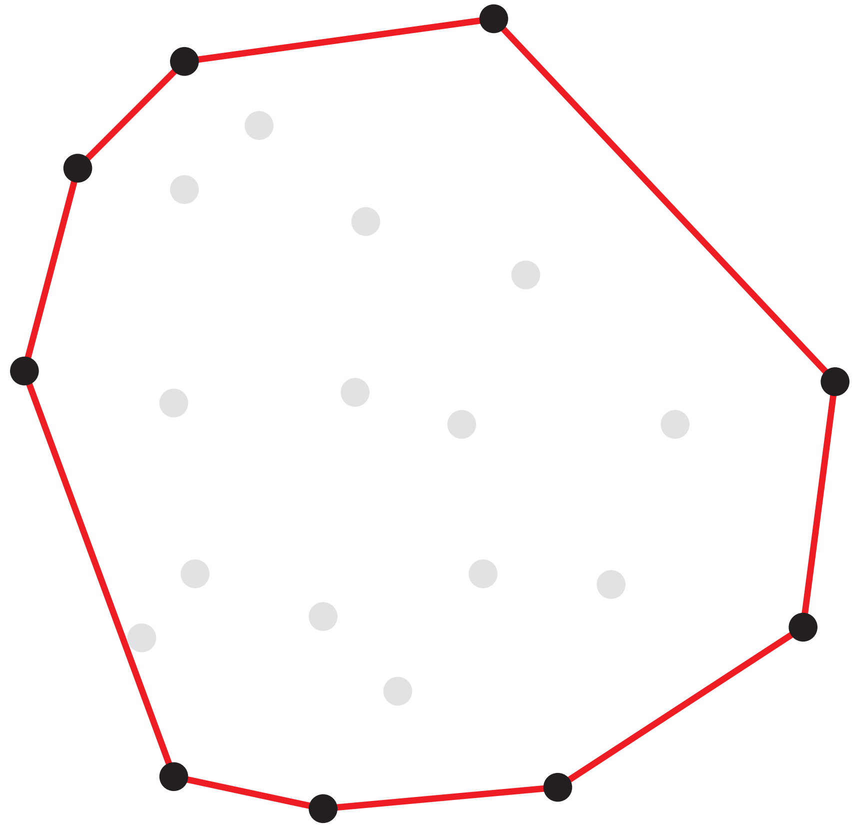


QuickHull algorithm (by prune-and-search)

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

Neve… running time of this algorithm de-pend… ion of the input points.
For …

- … points are in convex position, the … is $\Theta(n^2)$.
- … are such that each prune step elim-… f the current points, then the algo-… $\Theta(n \log n)$ time.
- … hull is triangular, the algorithm runs … e.



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

left turn:

in l

e i

from l

Retu

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
- … point in l and delete if from P

Adva…

W… t points $p_i \in P$ to be explored, do:

…s(top(l))

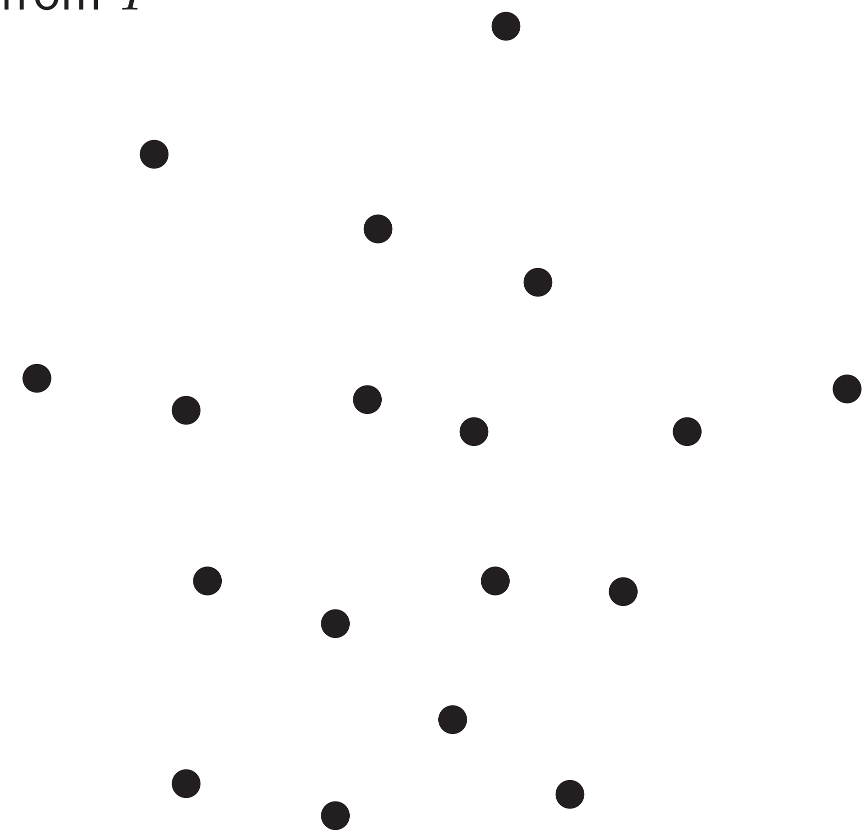
… left turn:

… in l

…e i

…from l

Retu…



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

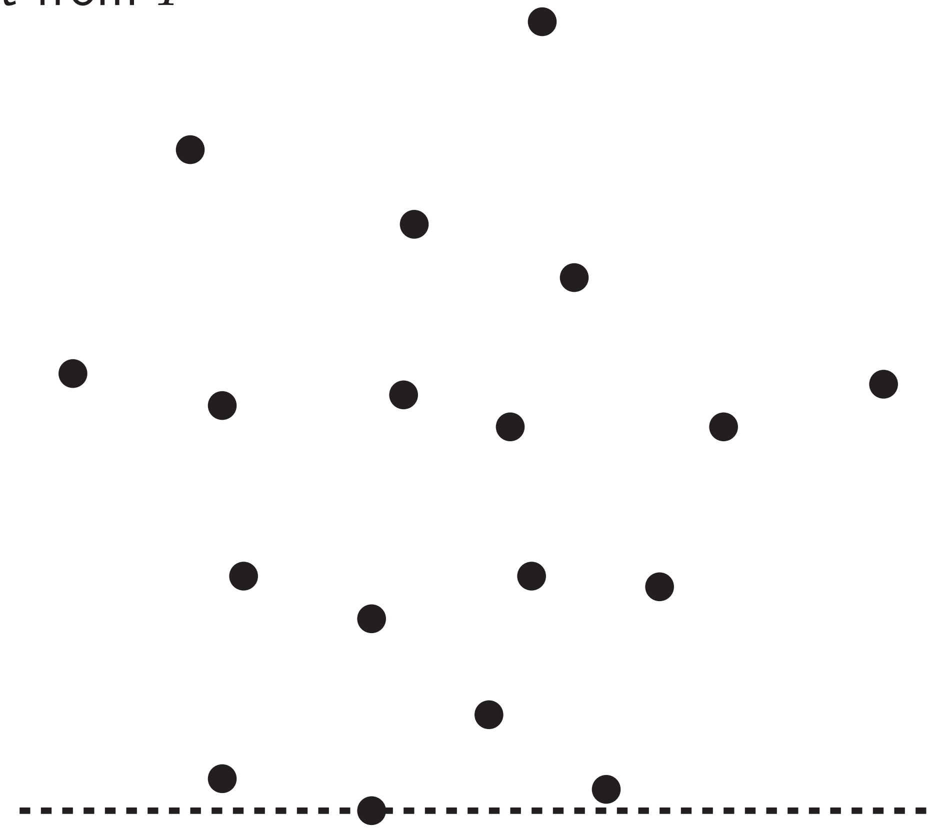
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s$(\text{top}(l))$

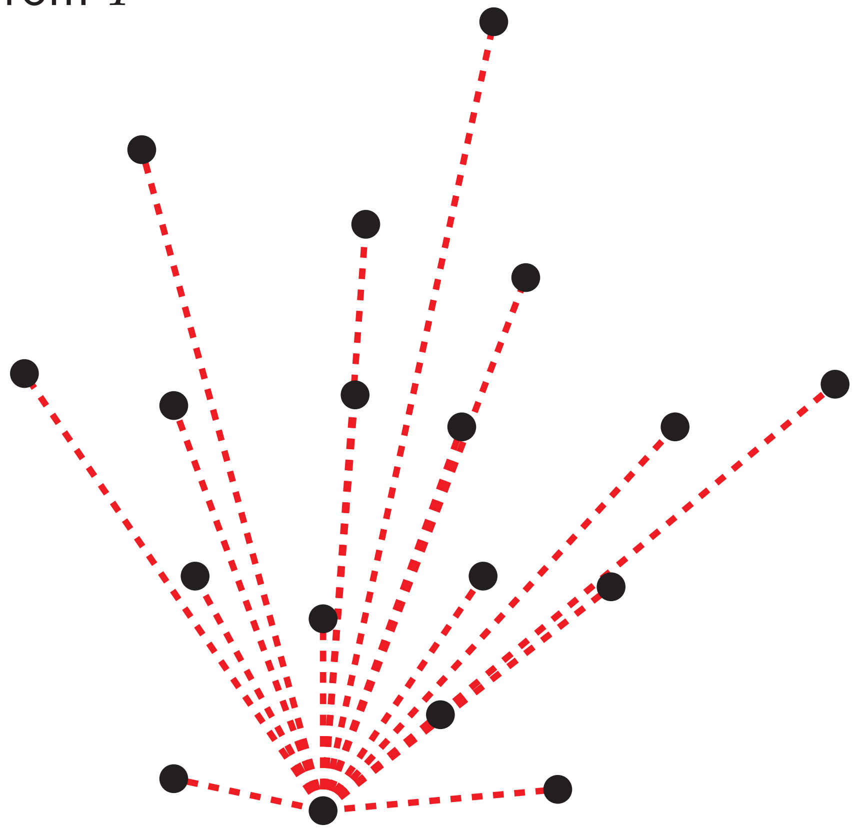
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adv

W t points $p_i \in P$ to be explored, do:

s(top(l))

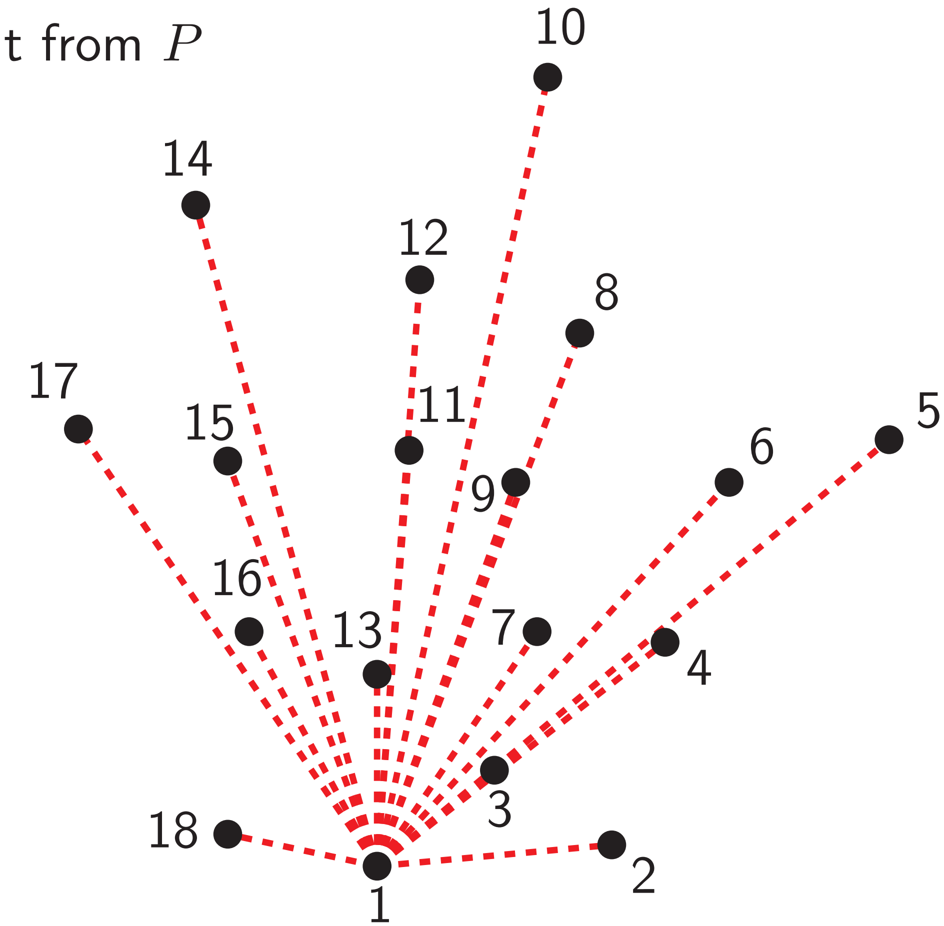
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s$(\text{top}(l))$

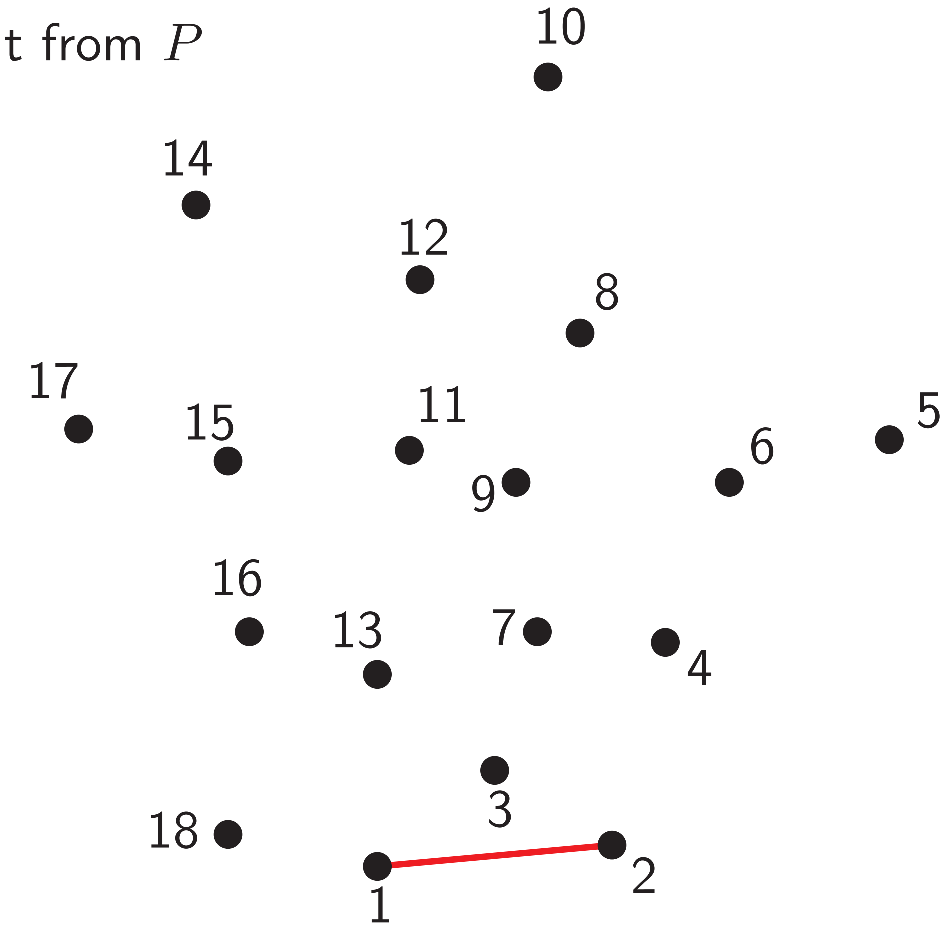
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

$s(\text{top}(l))$

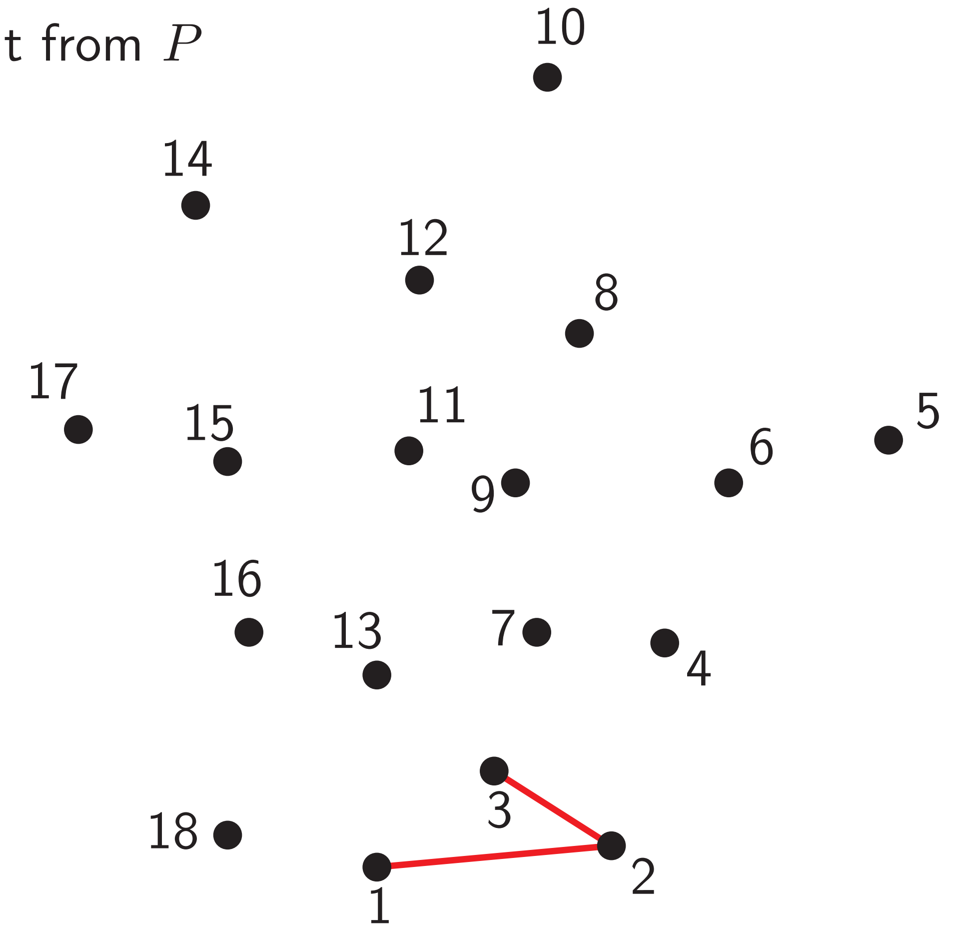
left turn:

in l

e i

from l

Retu



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
- ... point in l and delete if from P

Adva...

W... t points $p_i \in P$ to be explored, do:

...$s(\text{top}(l)))$

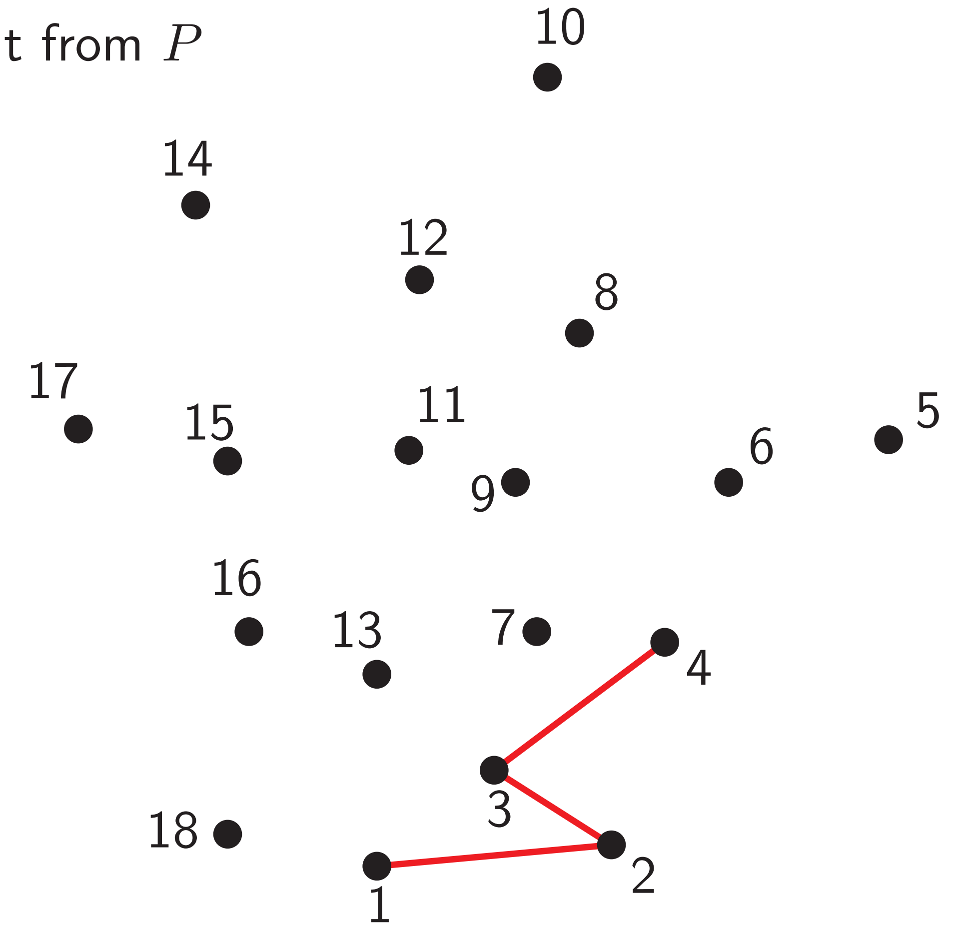
... left turn:

... in l

...e i

...from l

Retu...



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
- … point in l and delete if from P

Adv…

W… t points $p_i \in P$ to be explored, do:

…$s(\text{top}(l))$

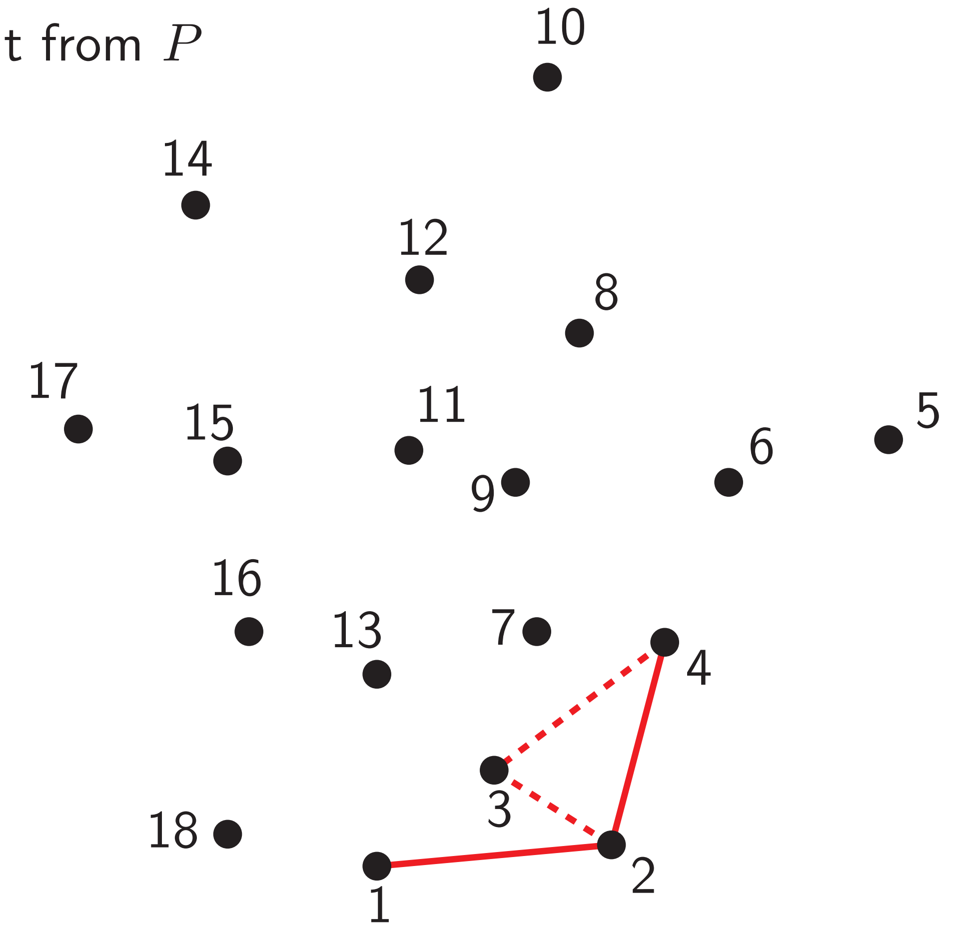
… left turn:

…$_i$ in l

…e i

…from l

Retu…



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

$s(\text{top}(l))$

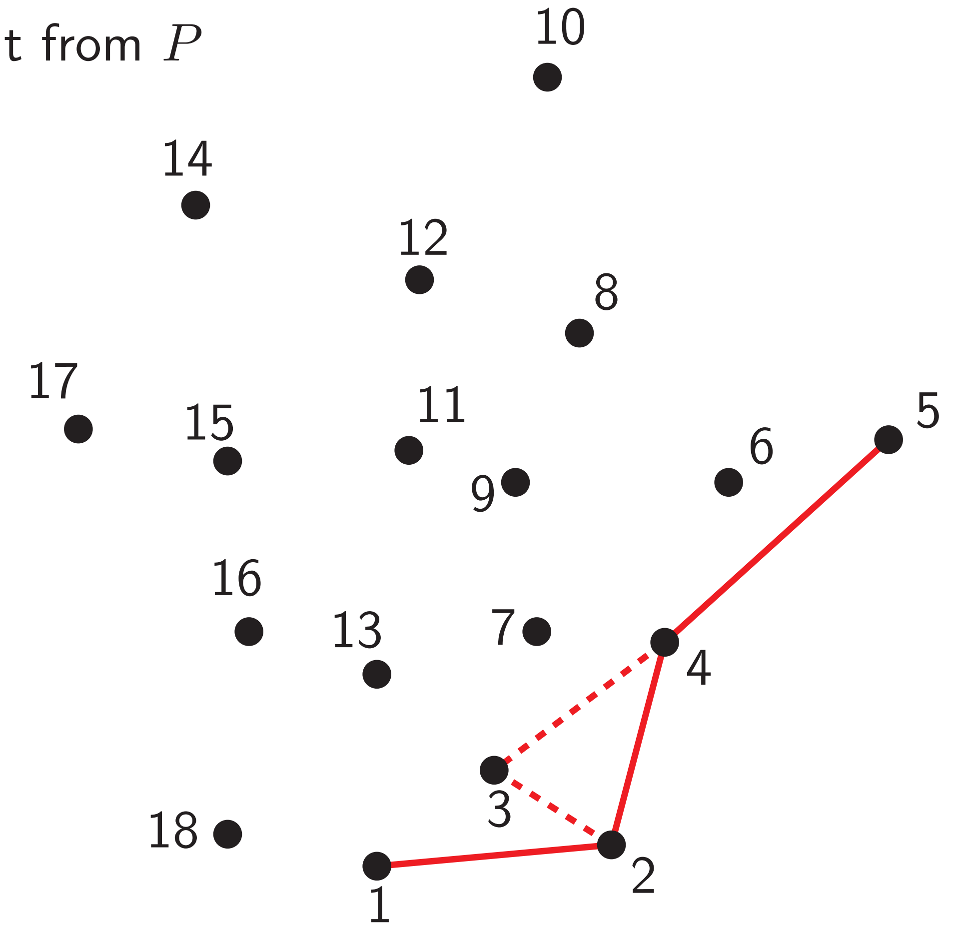
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

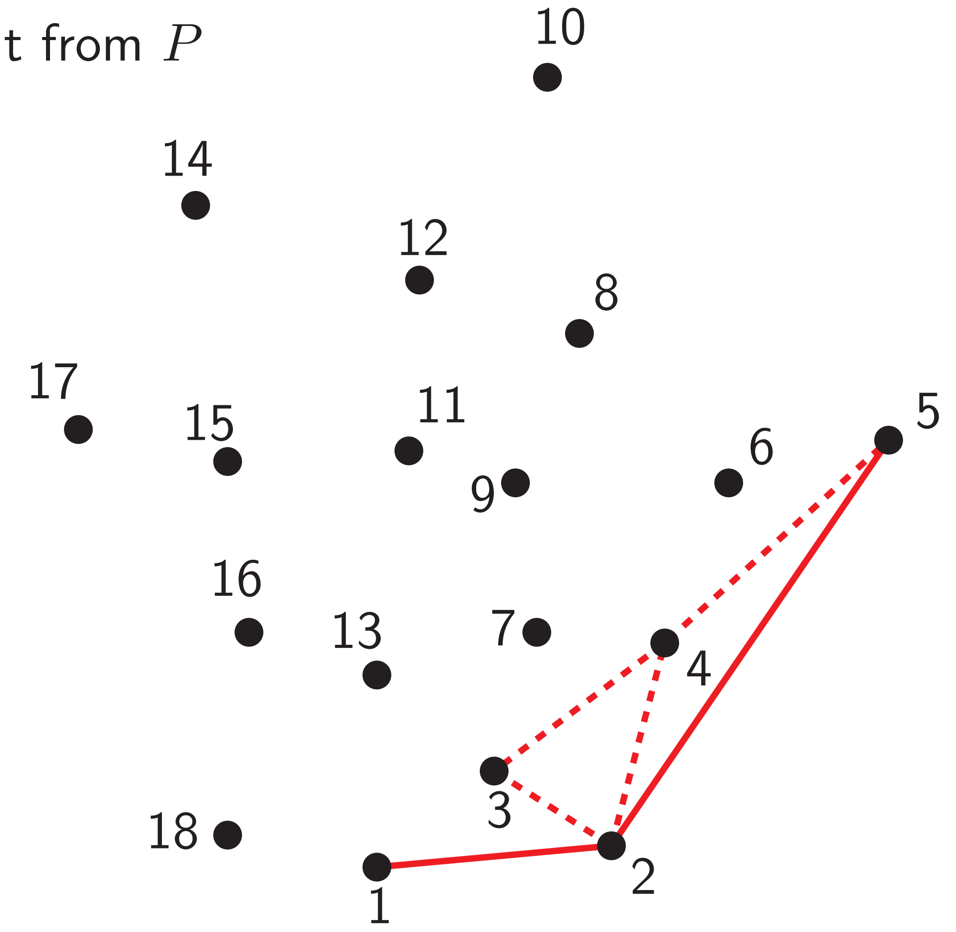
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

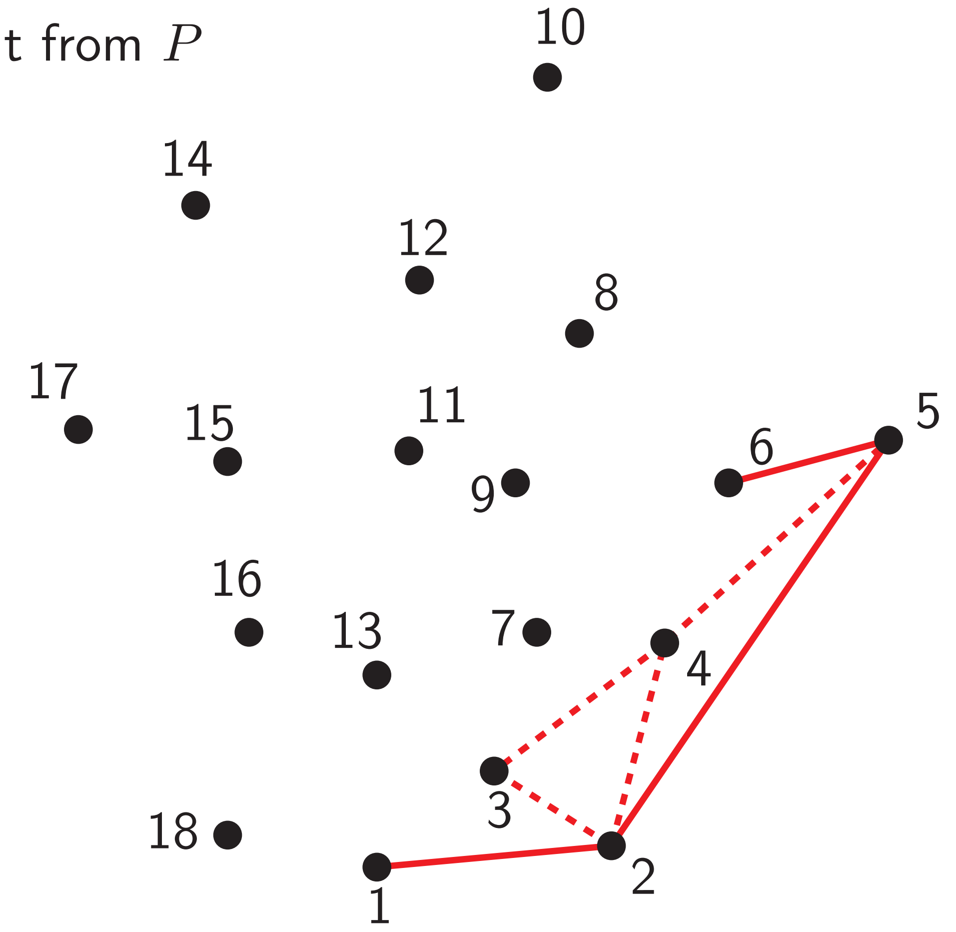
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s$(\text{top}(l))$

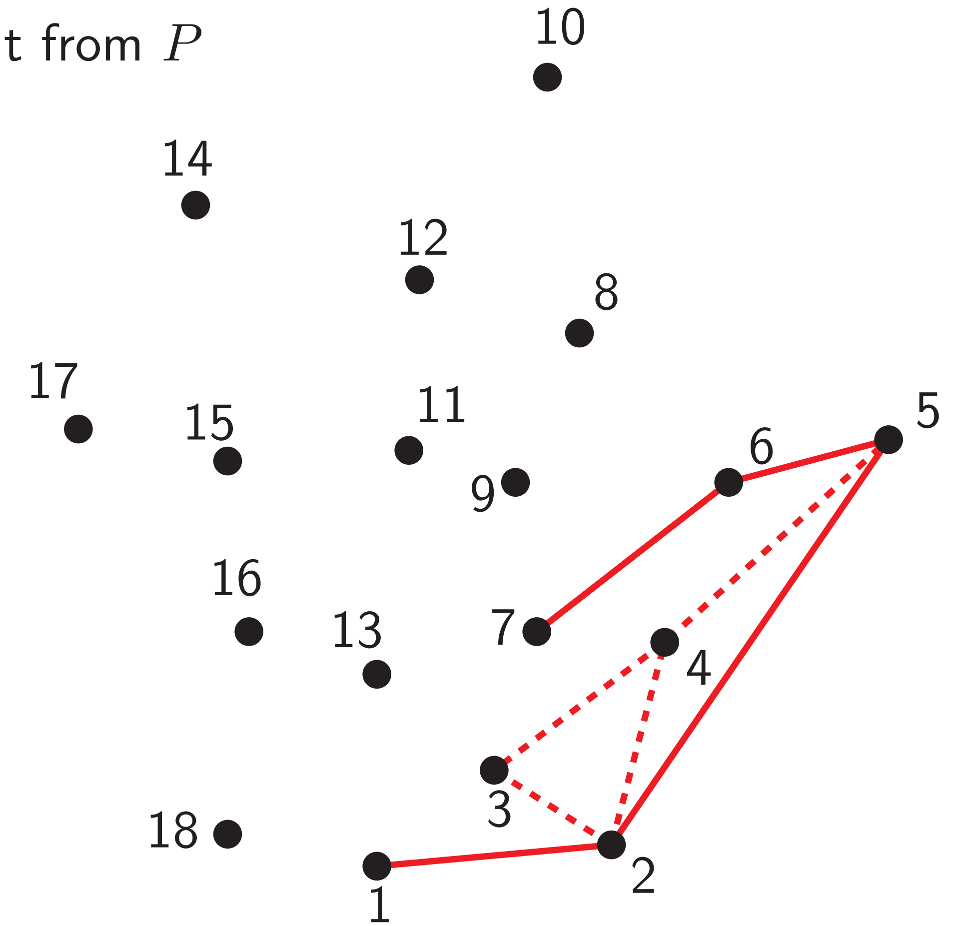
left turn:

in l

e i

from l

Retu



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
- ... point in l and delete if from P

Adva...

W... t points $p_i \in P$ to be explored, do:

...s(top(l))

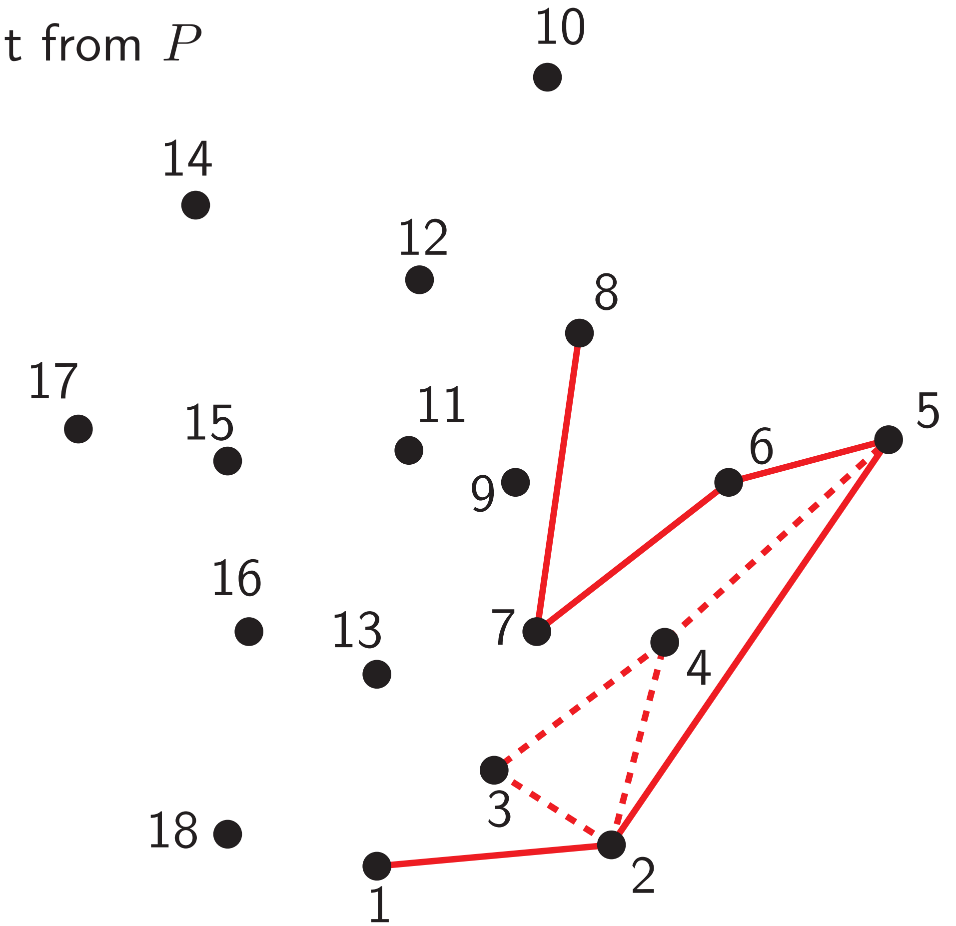
... left turn:

... in l

...e i

...from l

Retu...



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

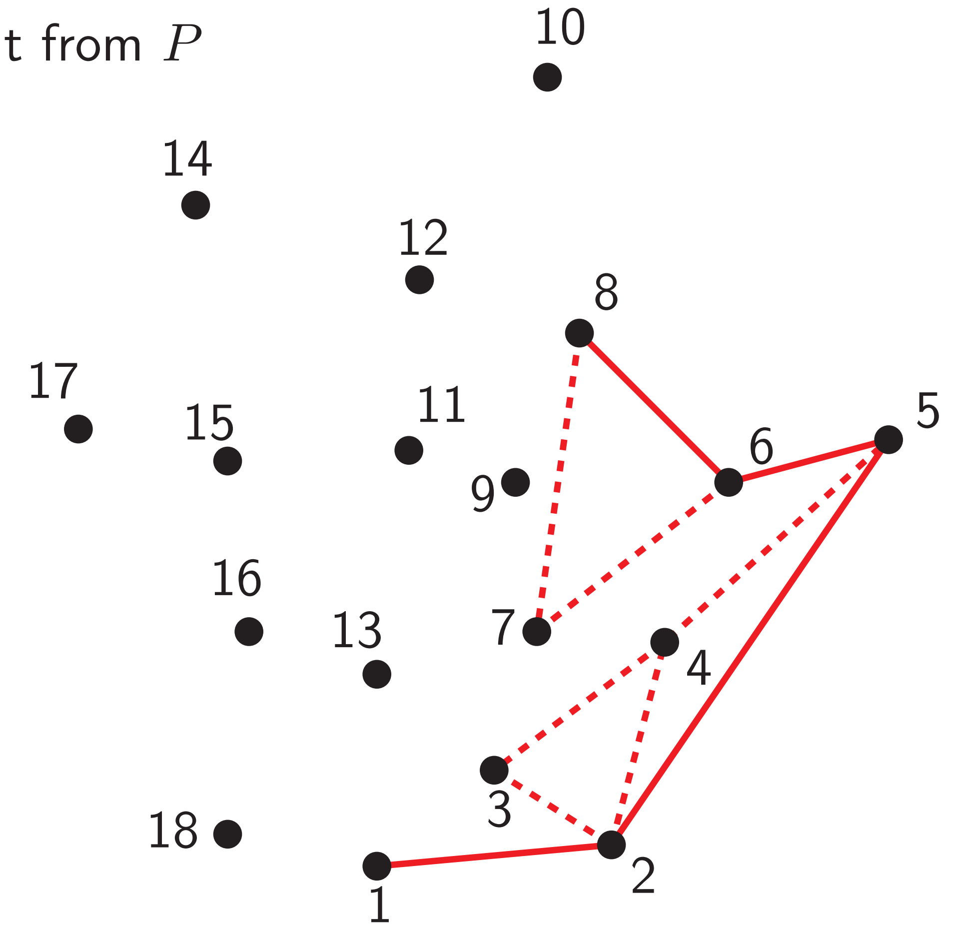
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adv

W t points $p_i \in P$ to be explored, do:

s$(\mathrm{top}(l))$

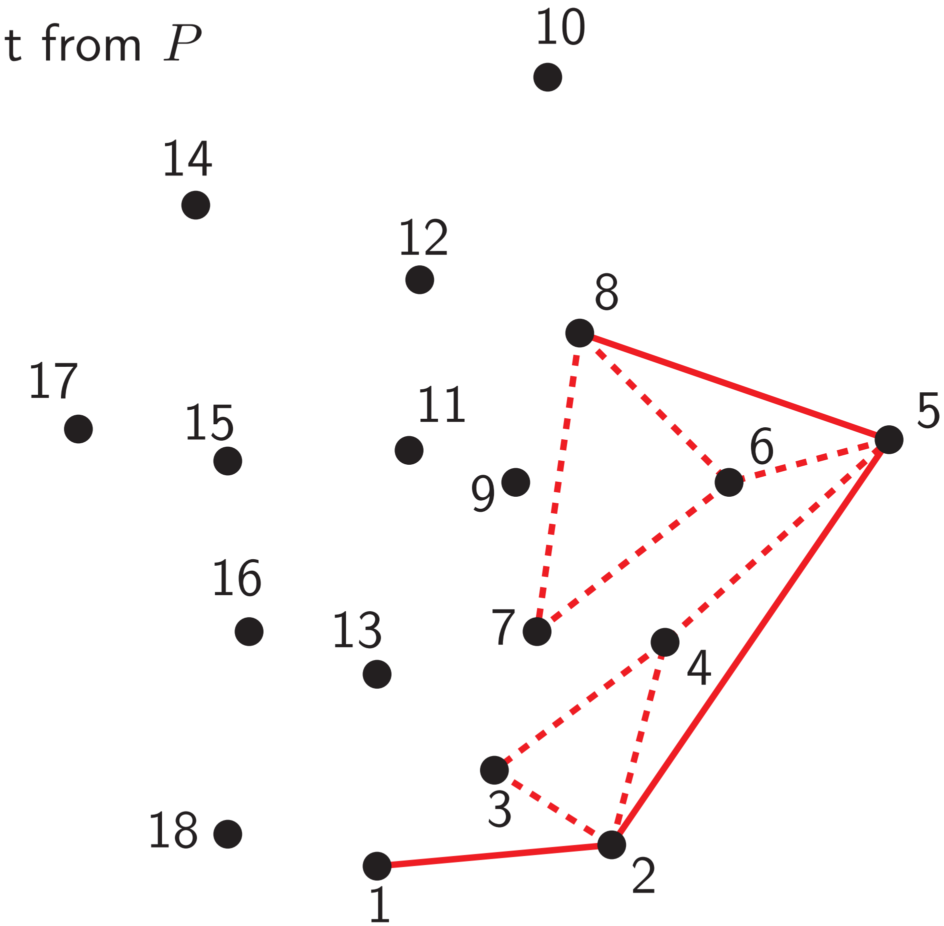
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

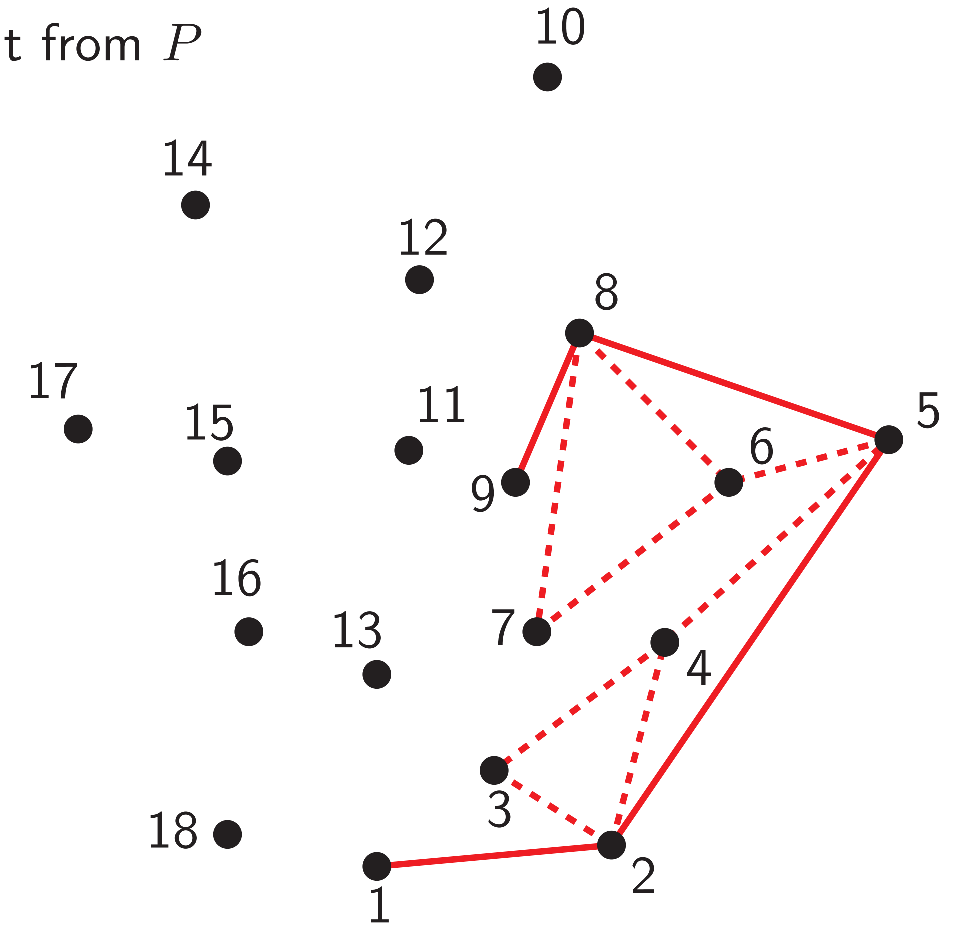
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

$s(\mathrm{top}(l))$

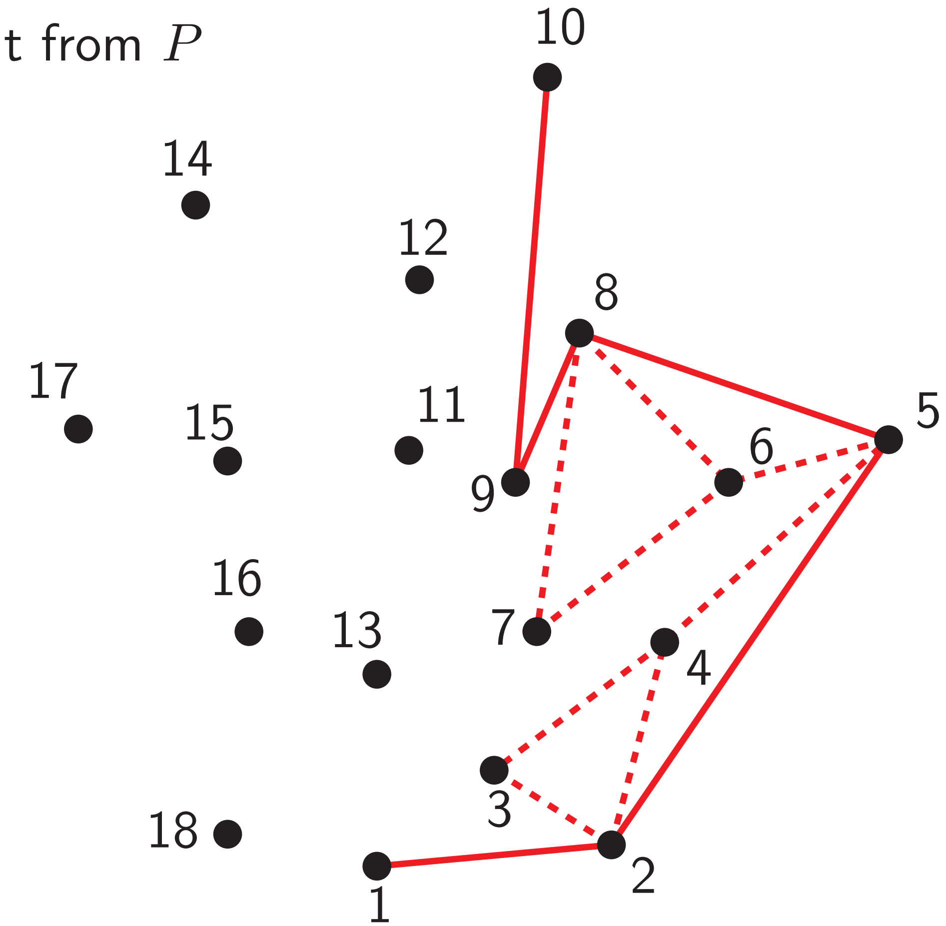
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adv

W t points $p_i \in P$ to be explored, do:

s$(\text{top}(l))$

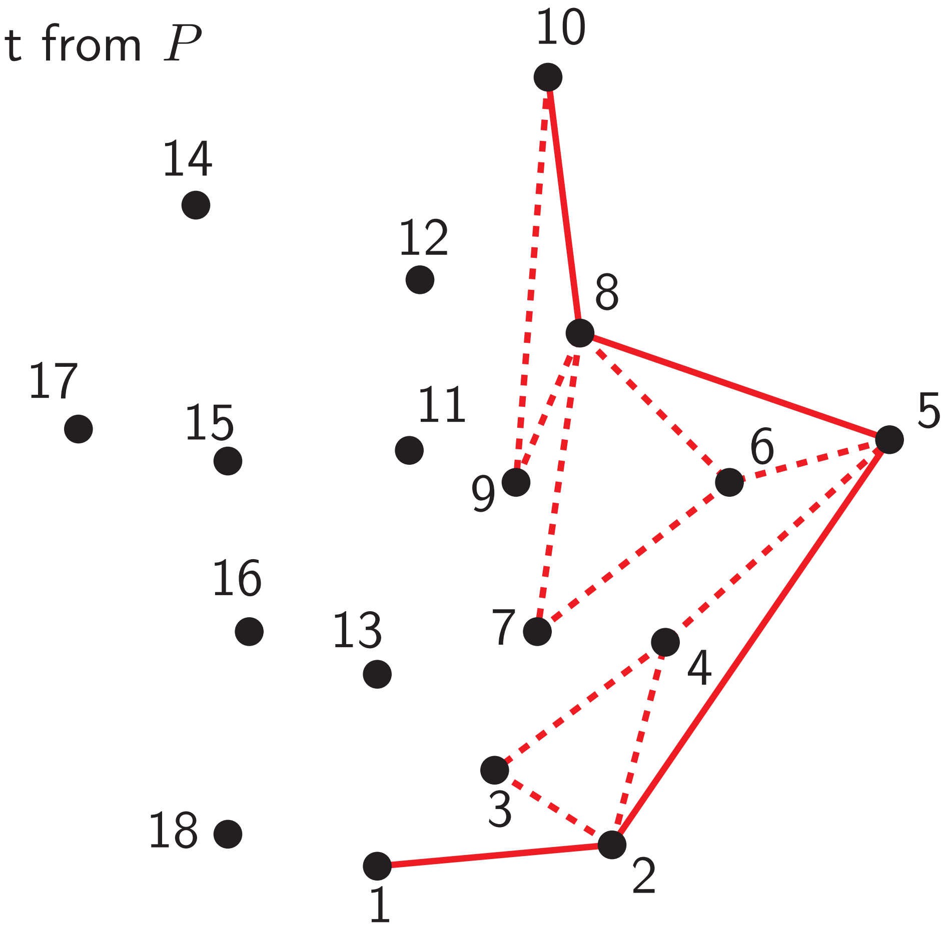
left turn:

in l

e i

from l

Retu



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
- … point in l and delete if from P

Adv…

W… t points $p_i \in P$ to be explored, do:

… $s(\text{top}(l))$

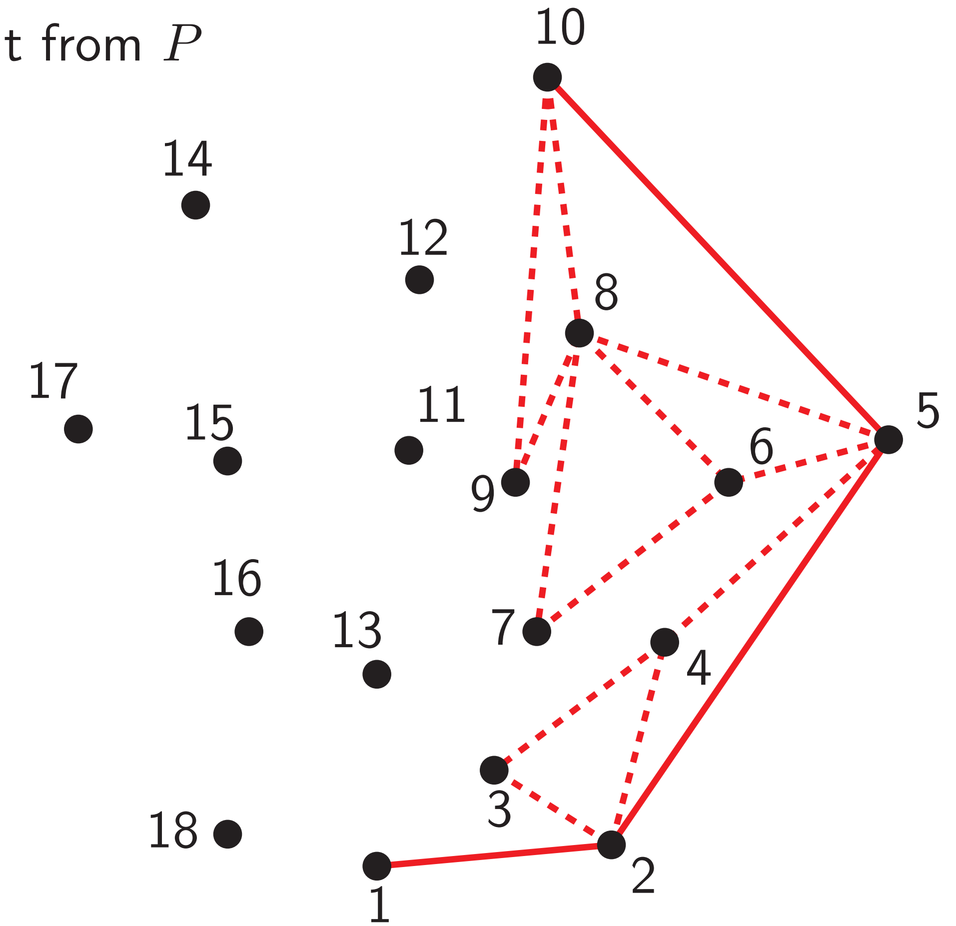
… left turn:

… in l

… e i

… from l

Retu…



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
- … point in l and delete if from P

Adv…

W… t points $p_i \in P$ to be explored, do:

…s(top(l))

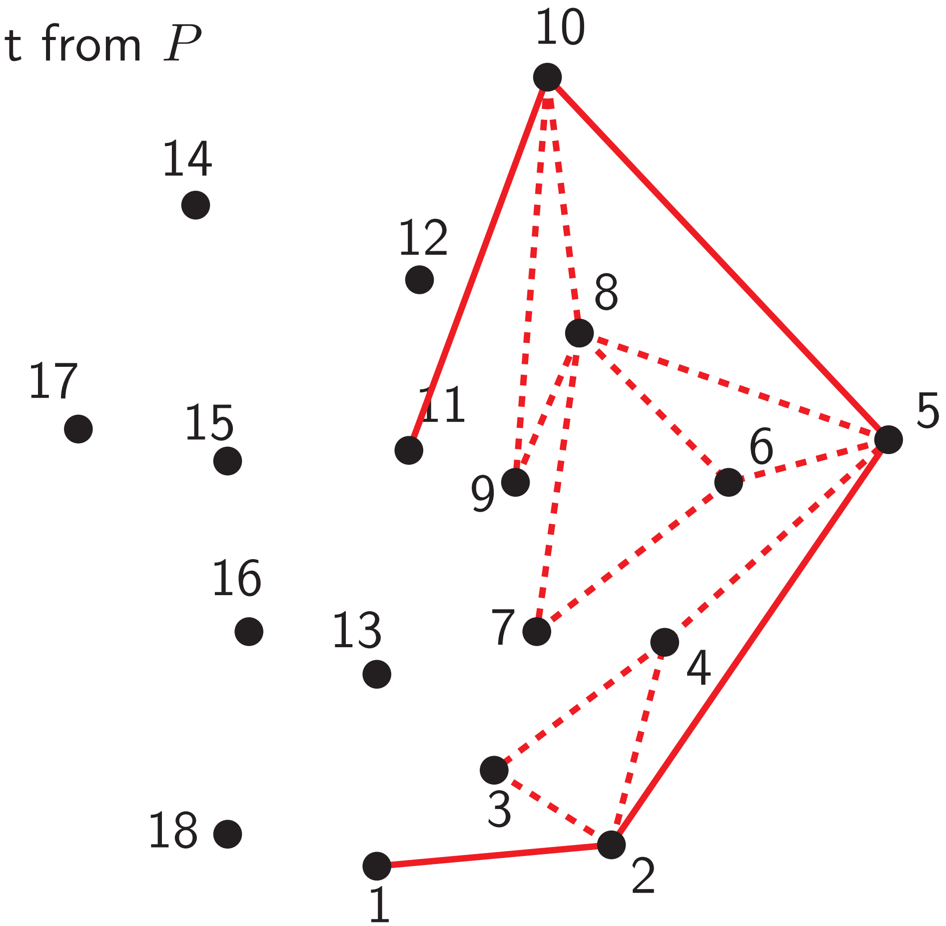
… left turn:

… in l

…e i

…from l

Retu…



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
- ... point in l and delete if from P

Adva...

W... t points $p_i \in P$ to be explored, do:

... $s(\mathrm{top}(l))$

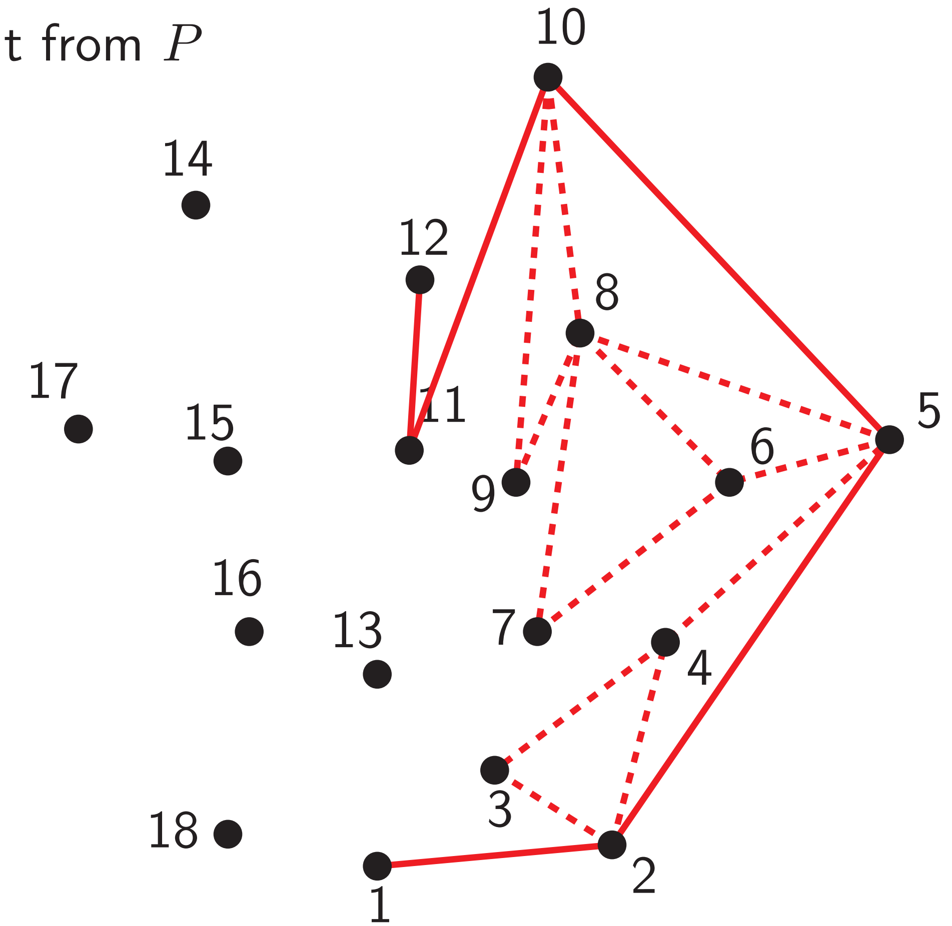
... left turn:

... $_i$ in l

... e i

... from l

Retu...



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
- ... point in l and delete if from P

Adva...

W... t points $p_i \in P$ to be explored, do:

...s(top(l)))

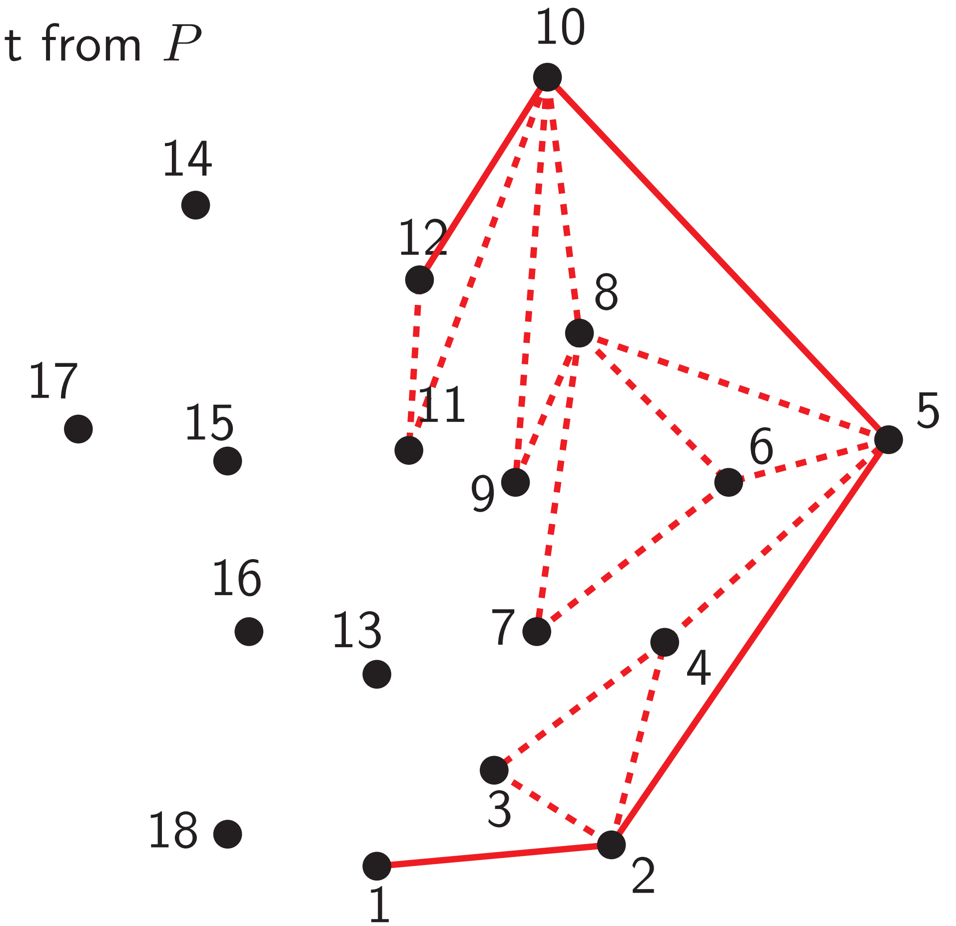
... left turn:

... in l

...e i

...from l

Retu...



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
- point in l and delete if from P

Adv

W t points $p_i \in P$ to be explored, do:

$s(\text{top}(l))$

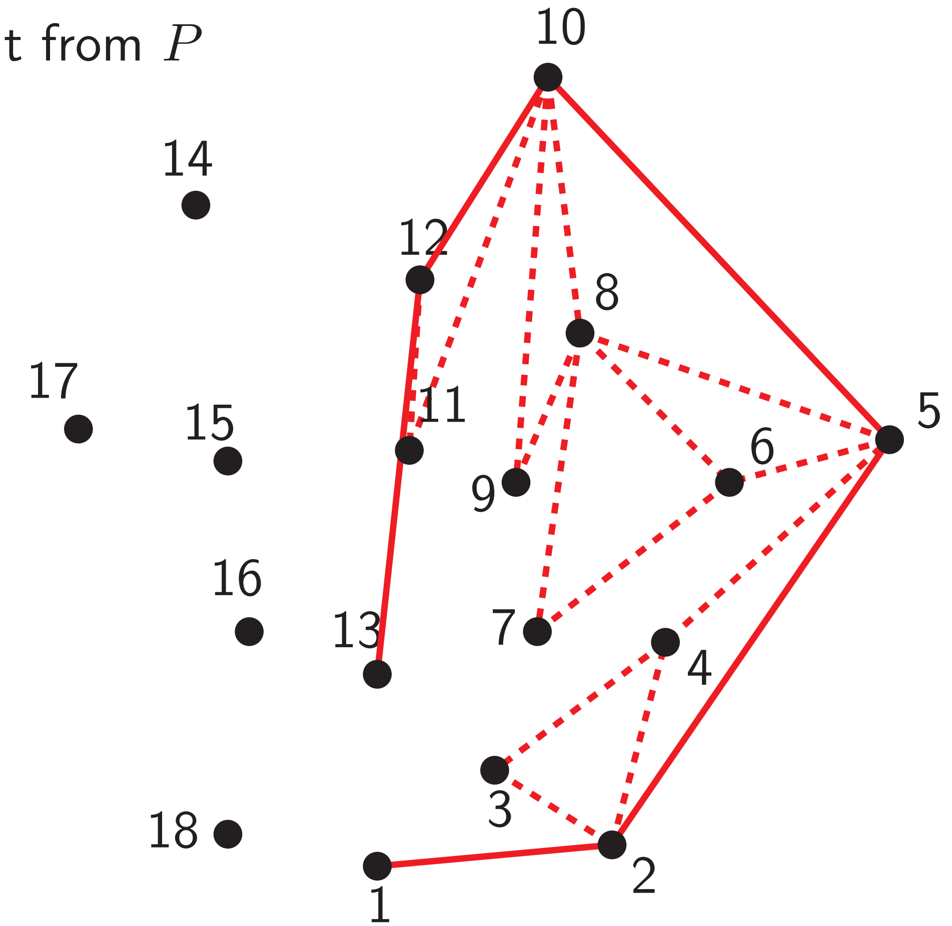
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

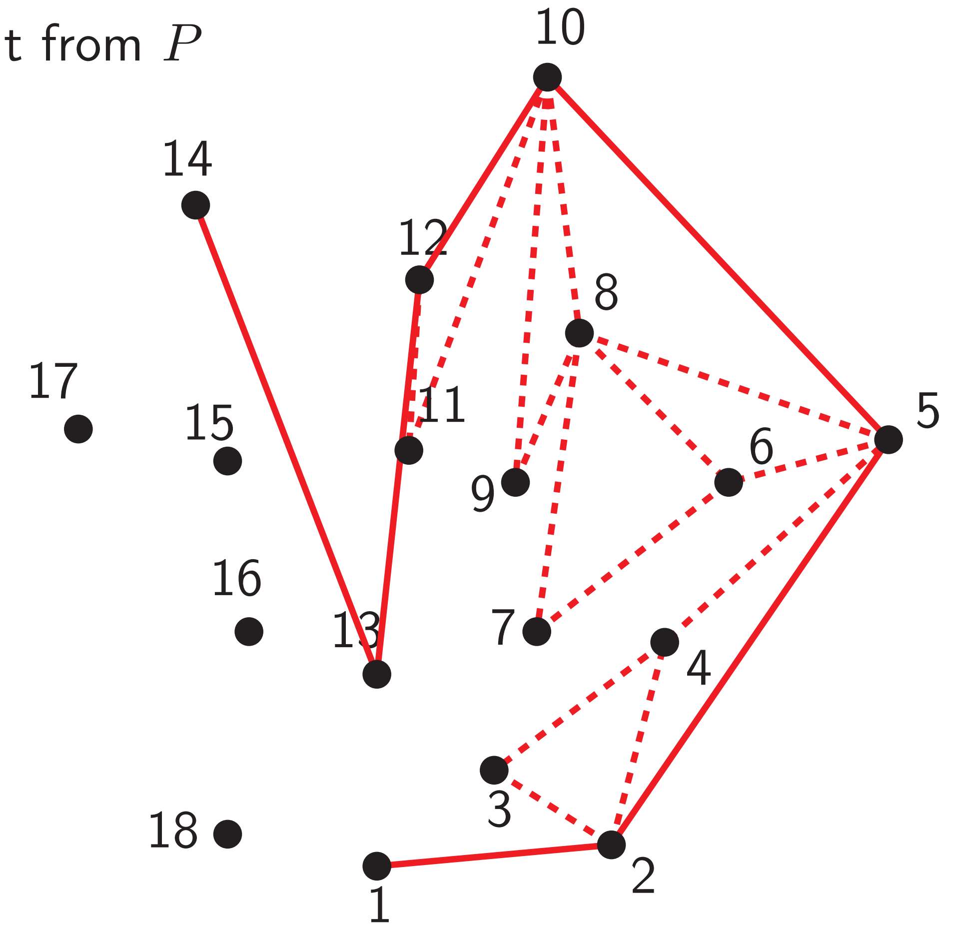
left turn:

; in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

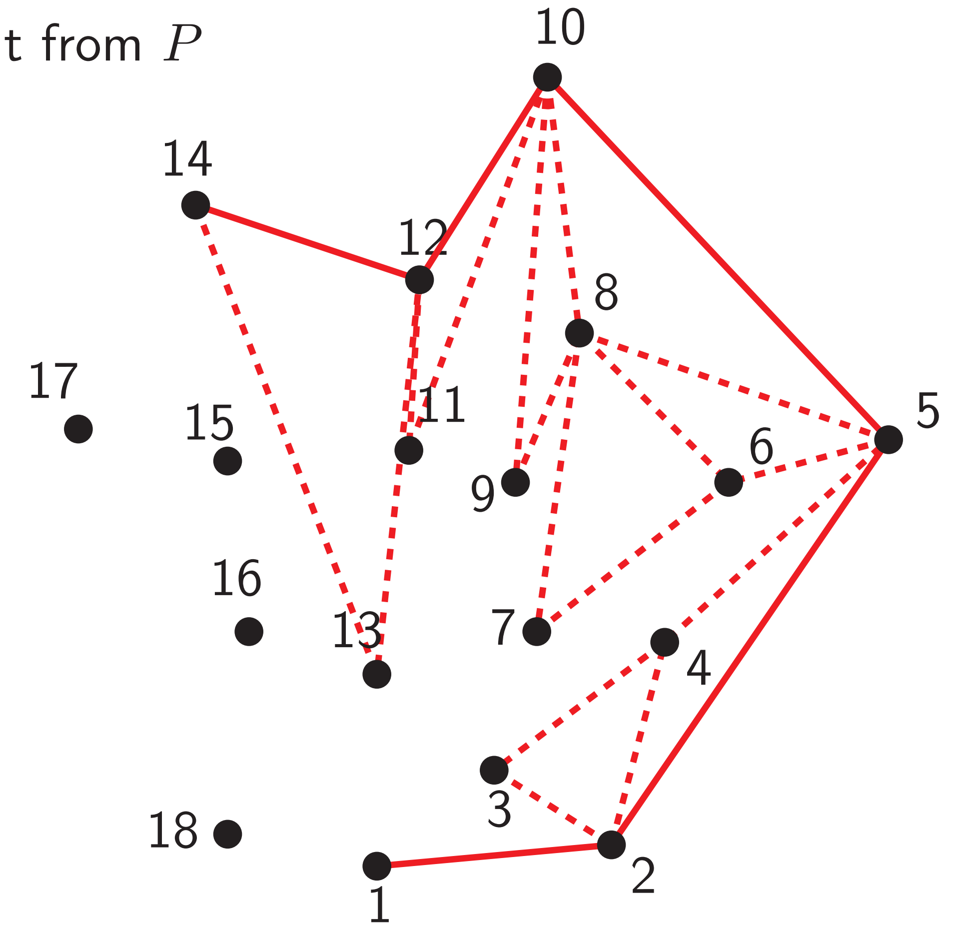
left turn:

in l

e i

from l

Retu



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

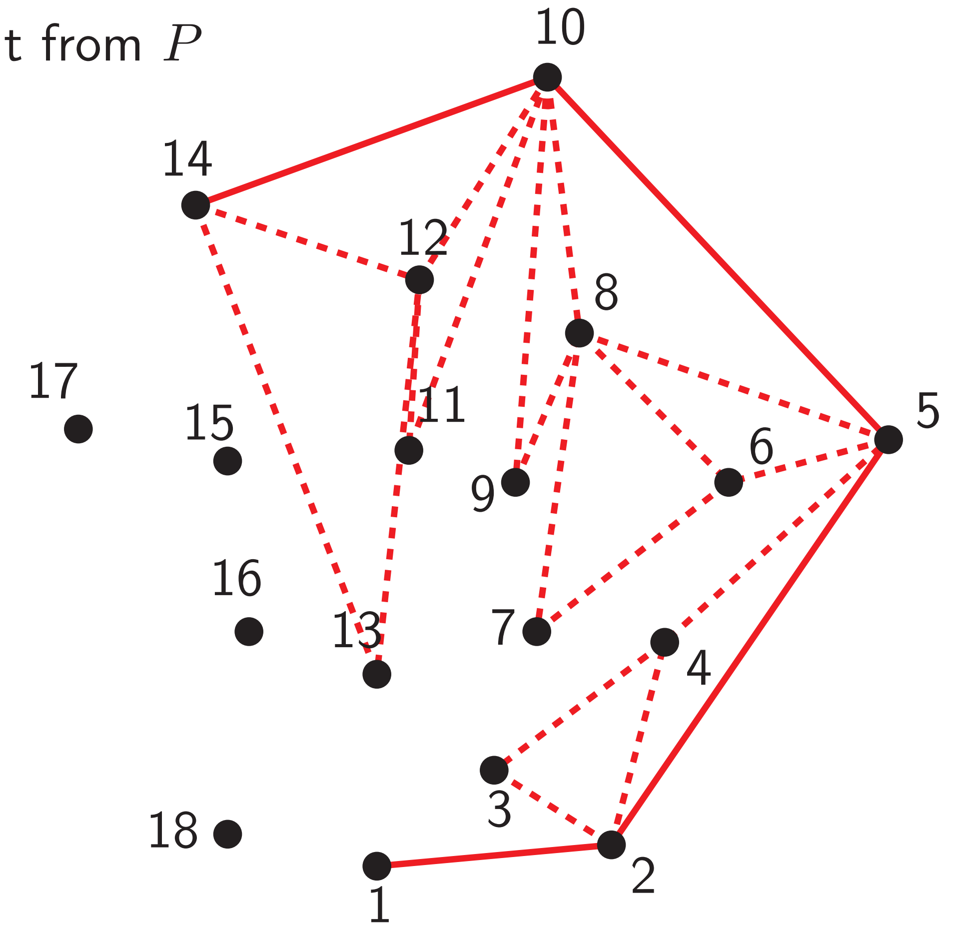
left turn:

in l

e i

from l

Retu



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
- … point in l and delete if from P

Adva…

W… t points $p_i \in P$ to be explored, do:

…s(top(l))

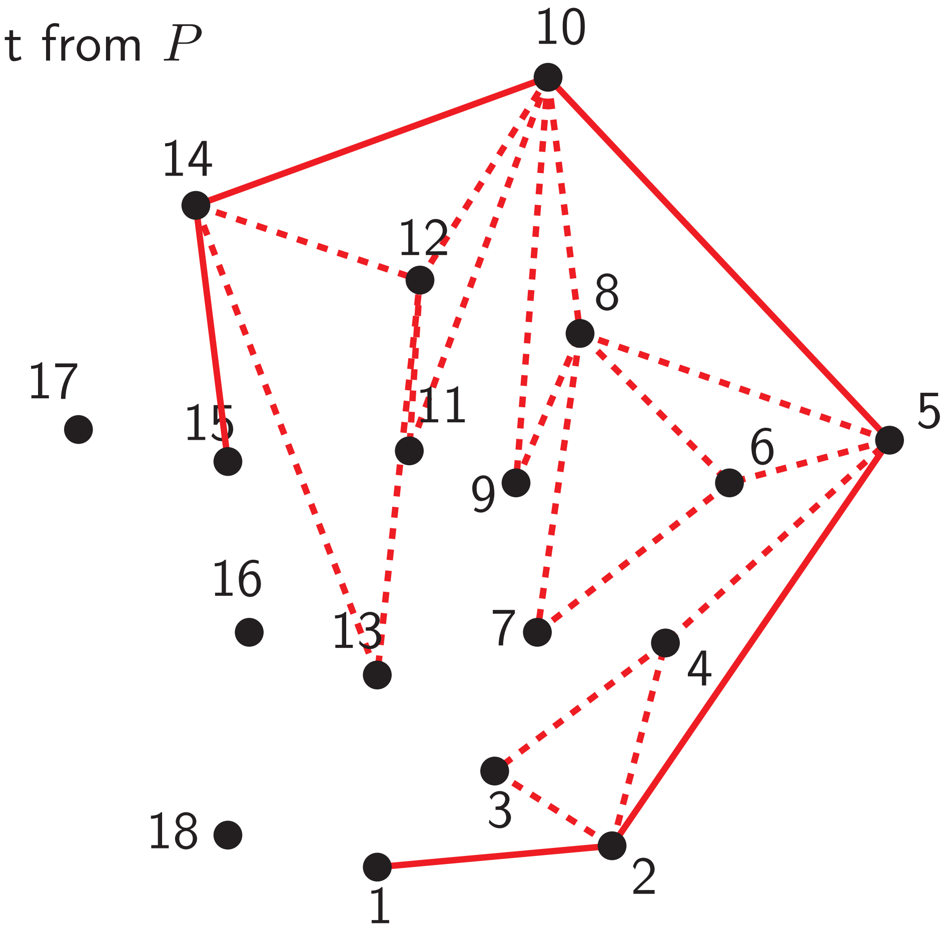
… left turn:

… in l

…e i

…from l

Retu…



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

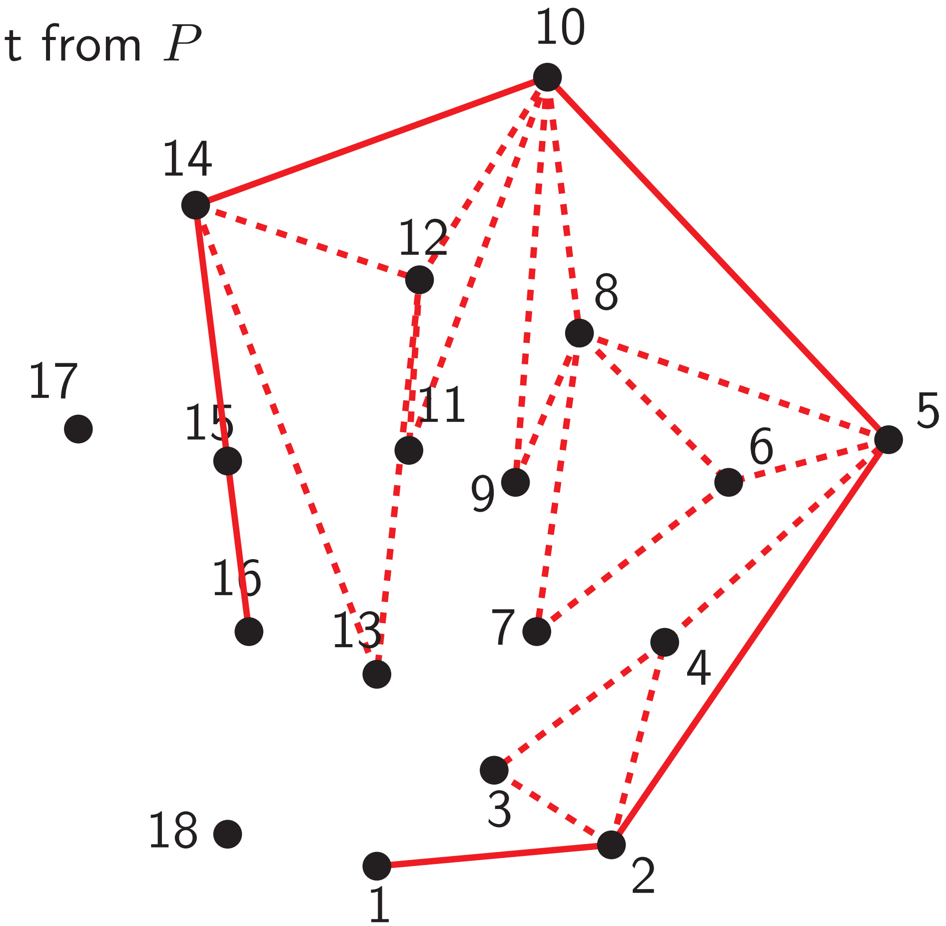
left turn:

in l

e i

from l

Retu



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
- ... point in l and delete if from P

Adv...

W... t points $p_i \in P$ to be explored, do:

... $\text{s}(\text{top}(l))$

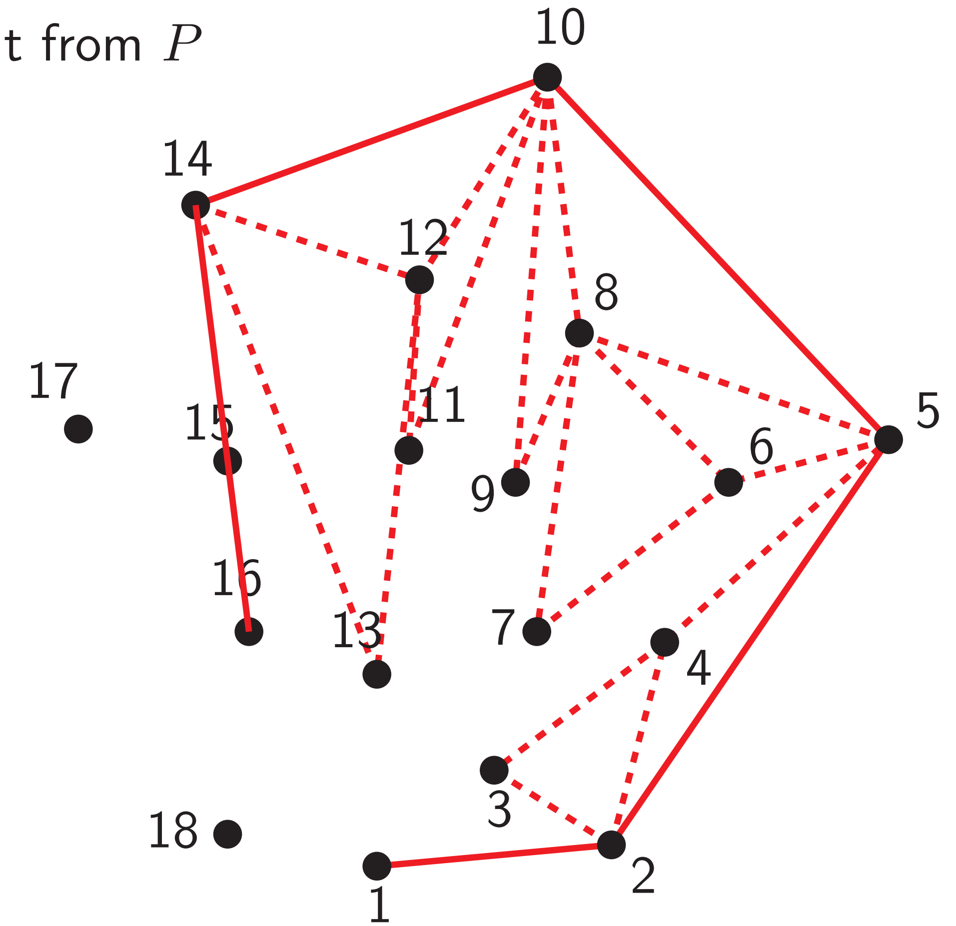
... left turn:

... in l

... e i

... from l

Retu...



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
- … point in l and delete if from P

Adva…

W… t points $p_i \in P$ to be explored, do:

…s(top(l))

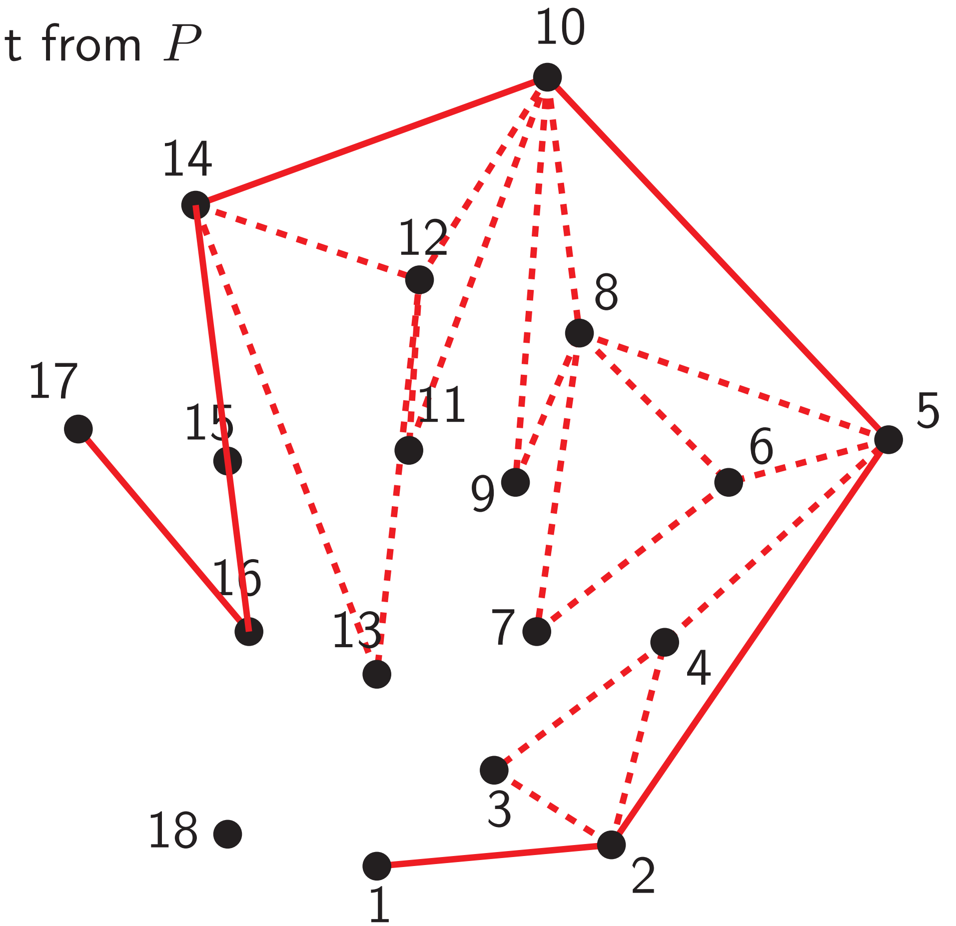
… left turn:

… in l

…e i

…from l

Retu…



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
- ... point in l and delete if from P

Adva...

W... t points $p_i \in P$ to be explored, do:

...$s(\text{top}(l))$

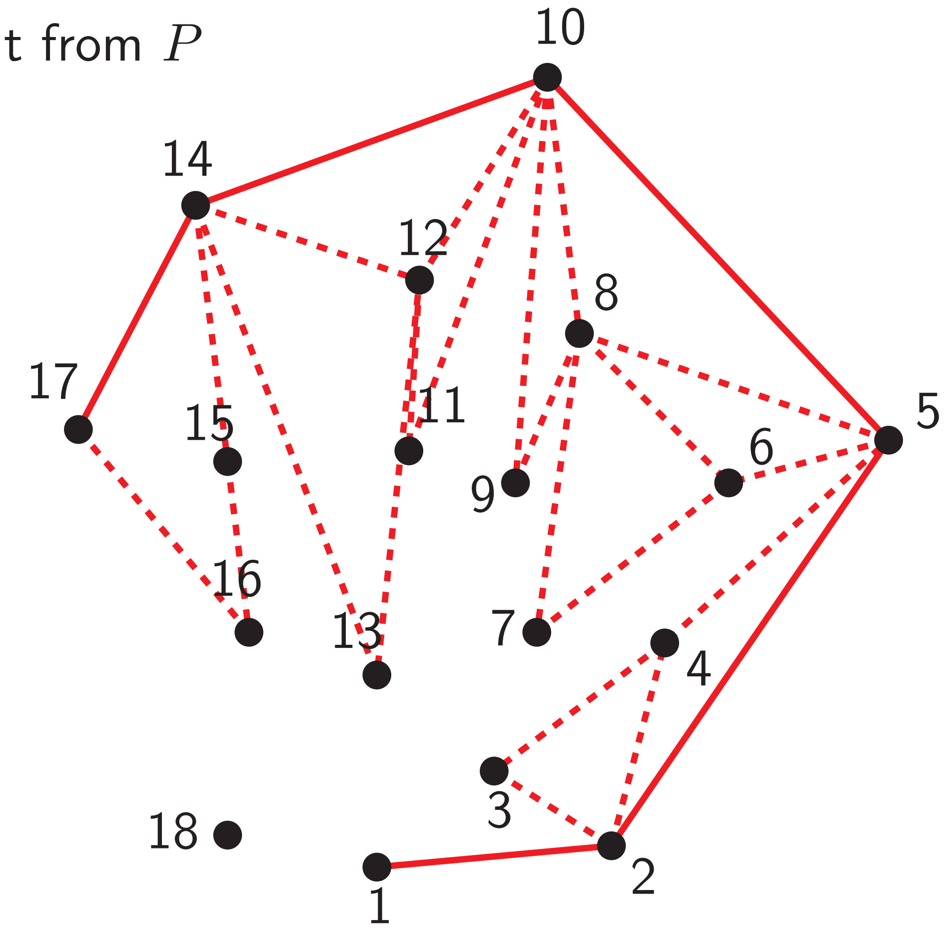
... left turn:

... in l

...e i

...from l

Retu...



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

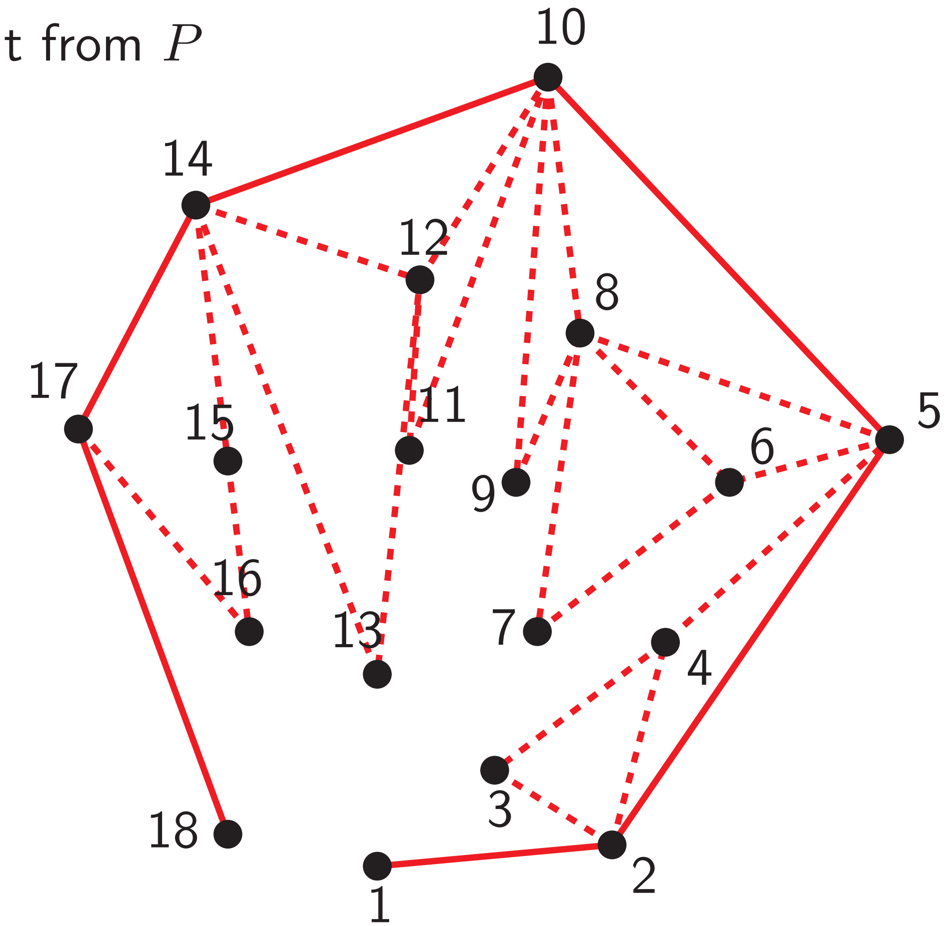
left turn:

in l

e i

from l

Retu



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
- … point in l and delete if from P

Adva…

W… t points $p_i \in P$ to be explored, do:

… $s(\mathrm{top}(l))$

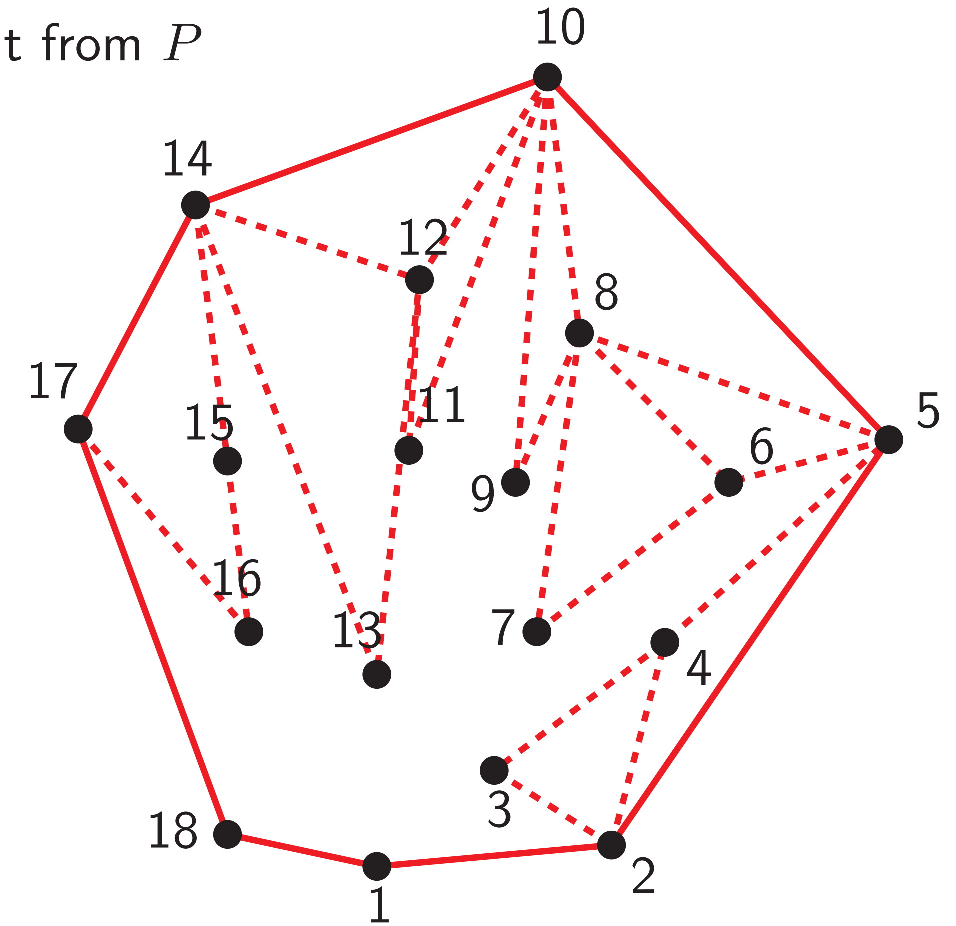
… left turn:

… in l

… e i

… from l

Retu…



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
- point in l and delete if from P

Adva

W t points $p_i \in P$ to be explored, do:

s(top(l))

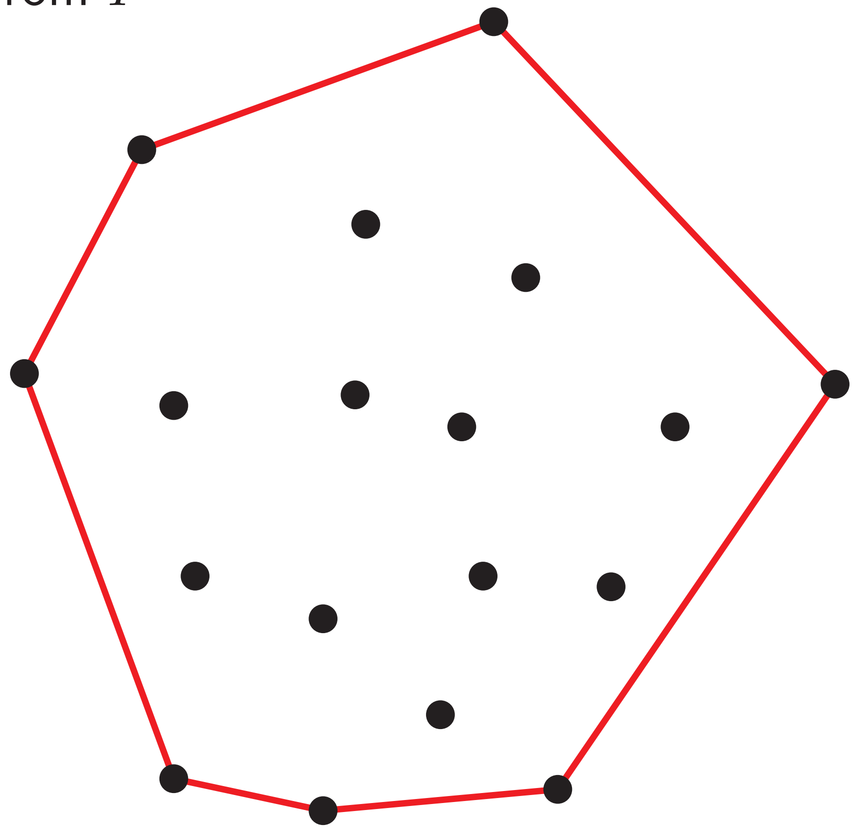
left turn:

in l

e i

from l

Retu



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
- … point in l and delete if from P

Adva…

W… t points $p_i \in P$ to be explored, do:

…s(top(l))

… left turn:

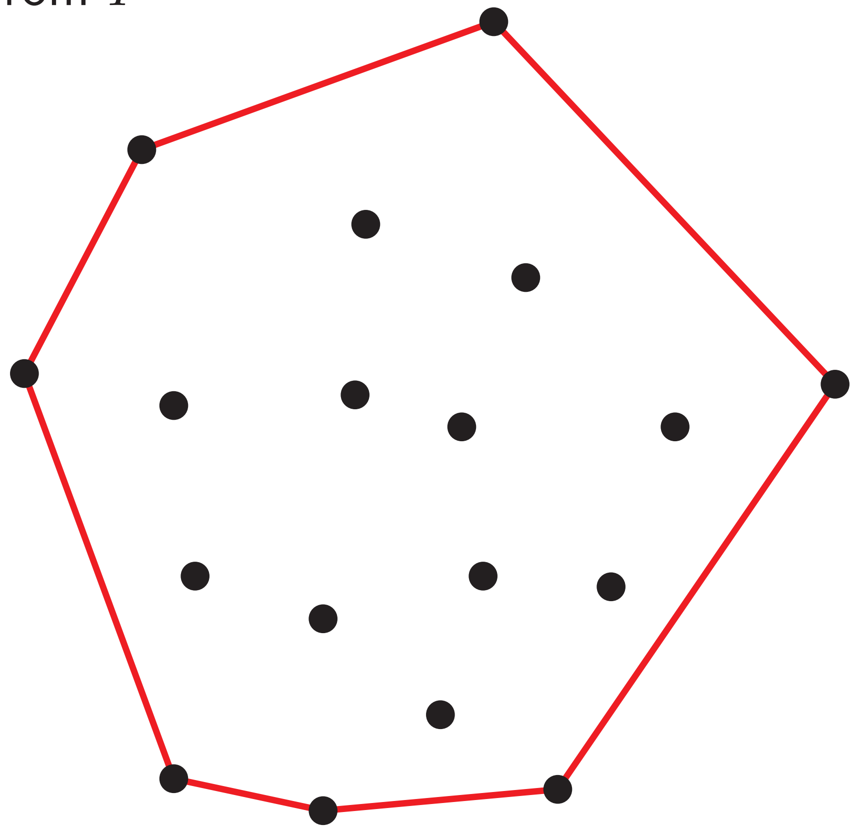
… in l

…e i

…from l

Retu…

Run… …$(n \log n)$



Incremental algorithm

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adva

Fr, do:

e exterior of the polygon defined by l:

te the points p_l and p_r

g the supporting lines

p_i to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu

CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

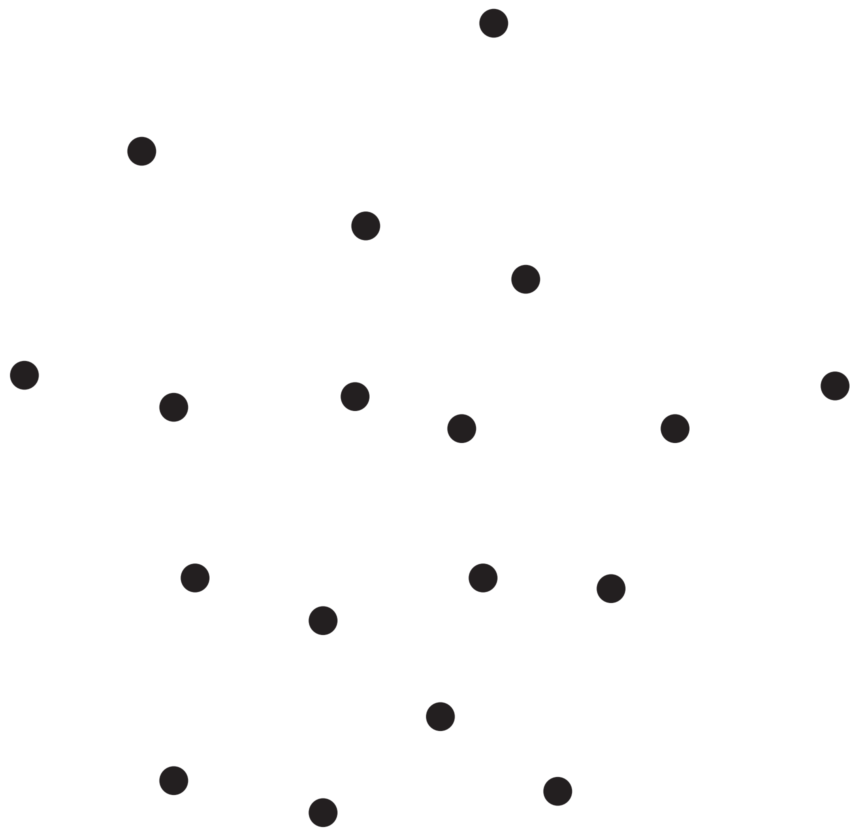
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \dots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr, do:

e exterior of the polygon defined by l:

te the points p_l and p_r

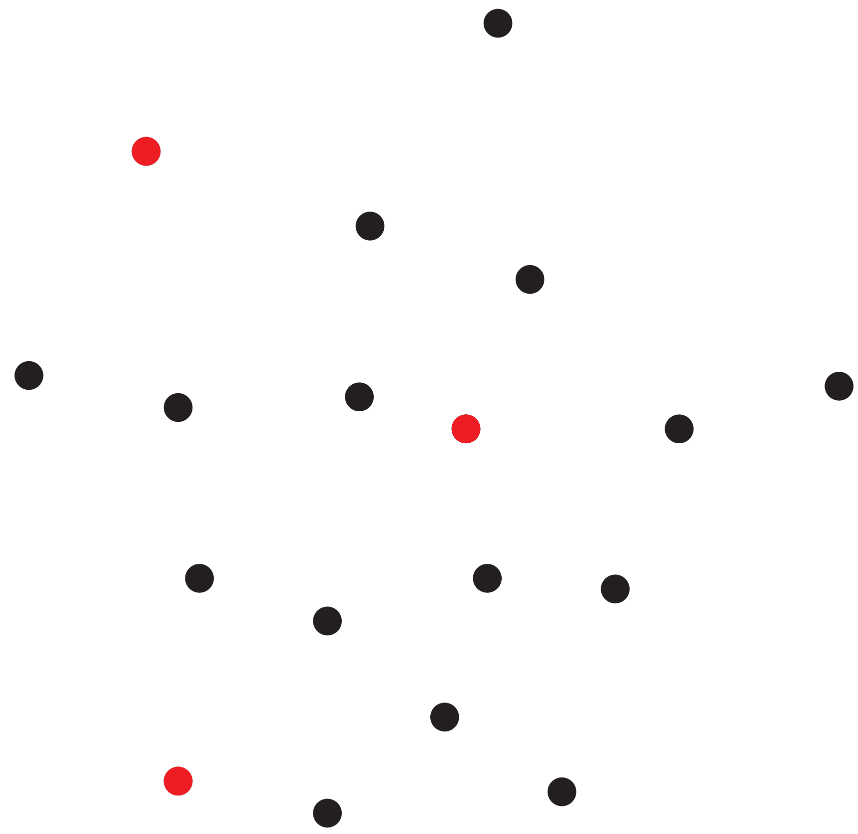
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv…

Fr…, do:

…e exterior of the polygon defined by l:

…te the points p_l and p_r

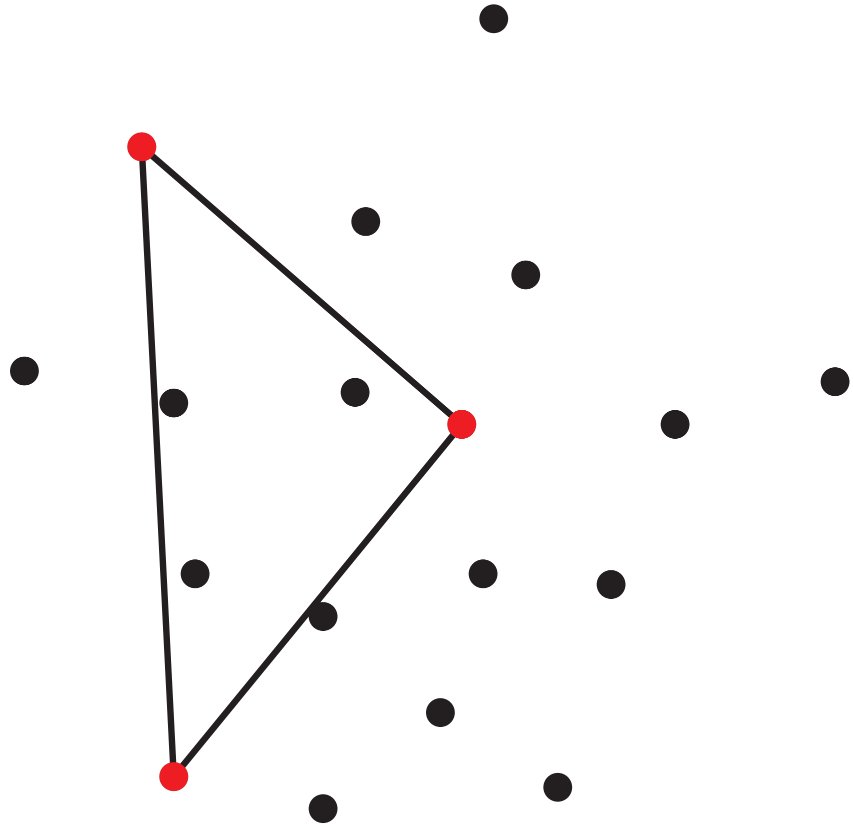
…g the supporting lines

…$_i$ to the polygon

…e the chain $p_l, \dots, p_r$ in l

…he chain p_l, p_i, p_r

Retu…



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

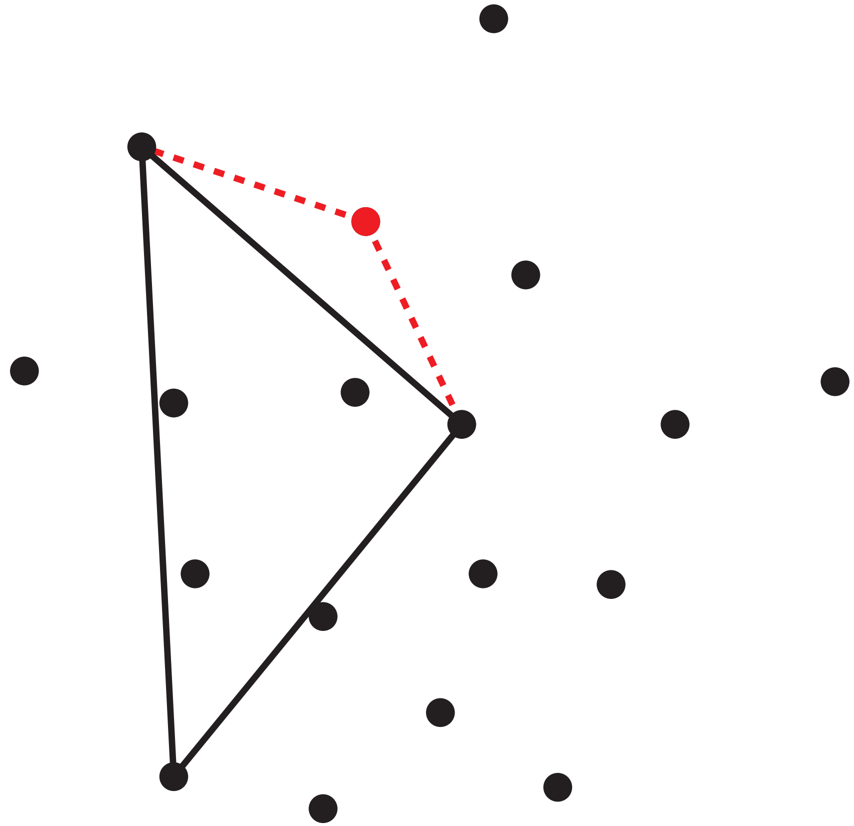
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

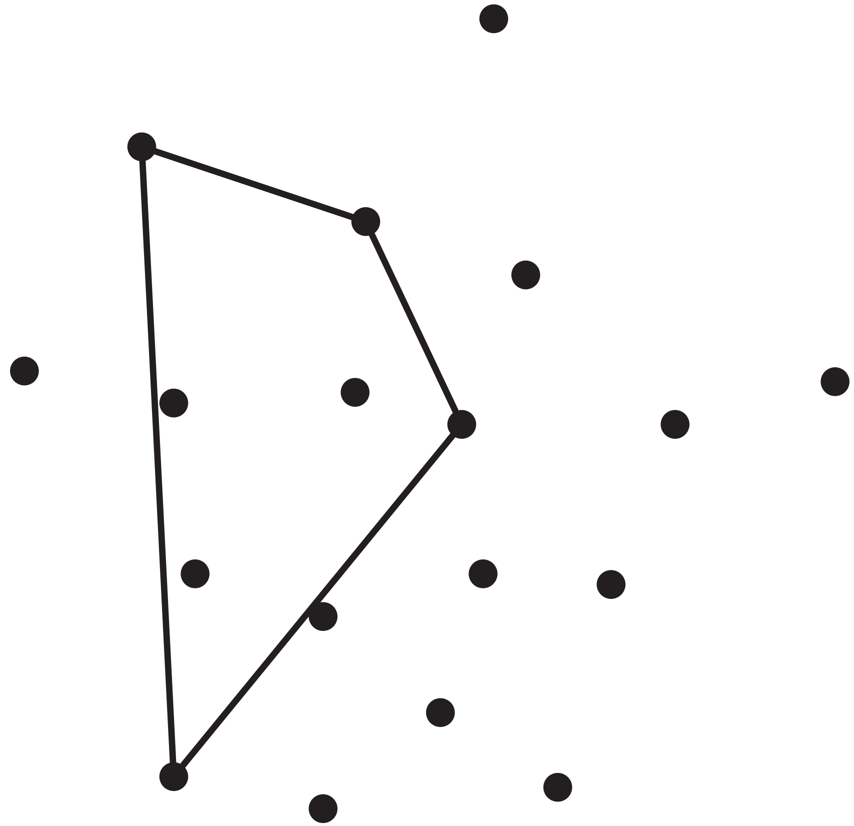
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

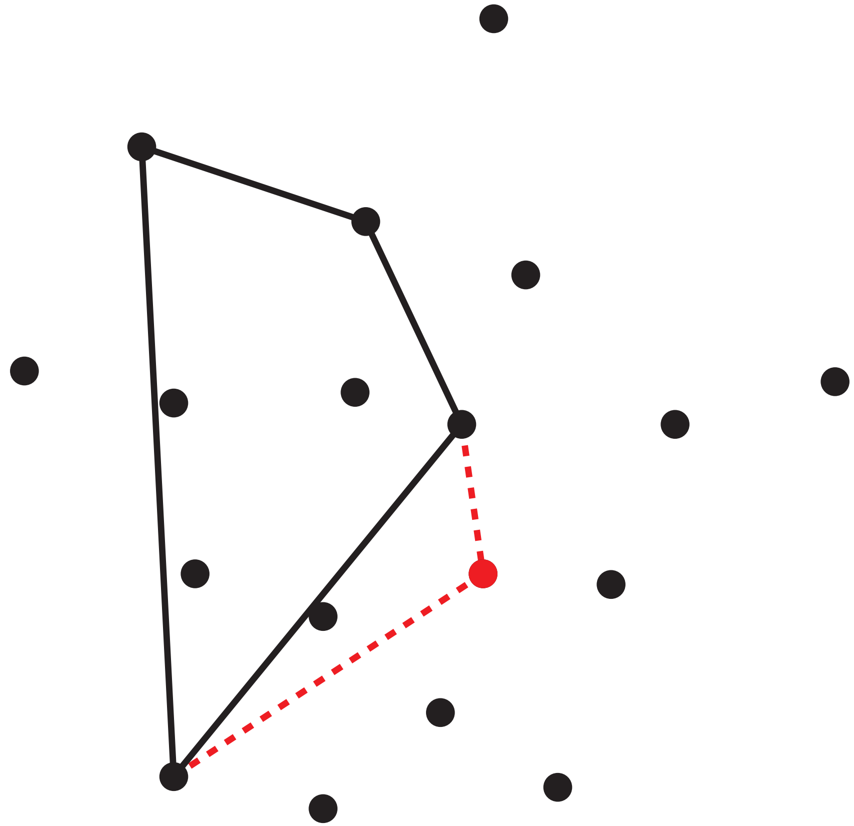
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

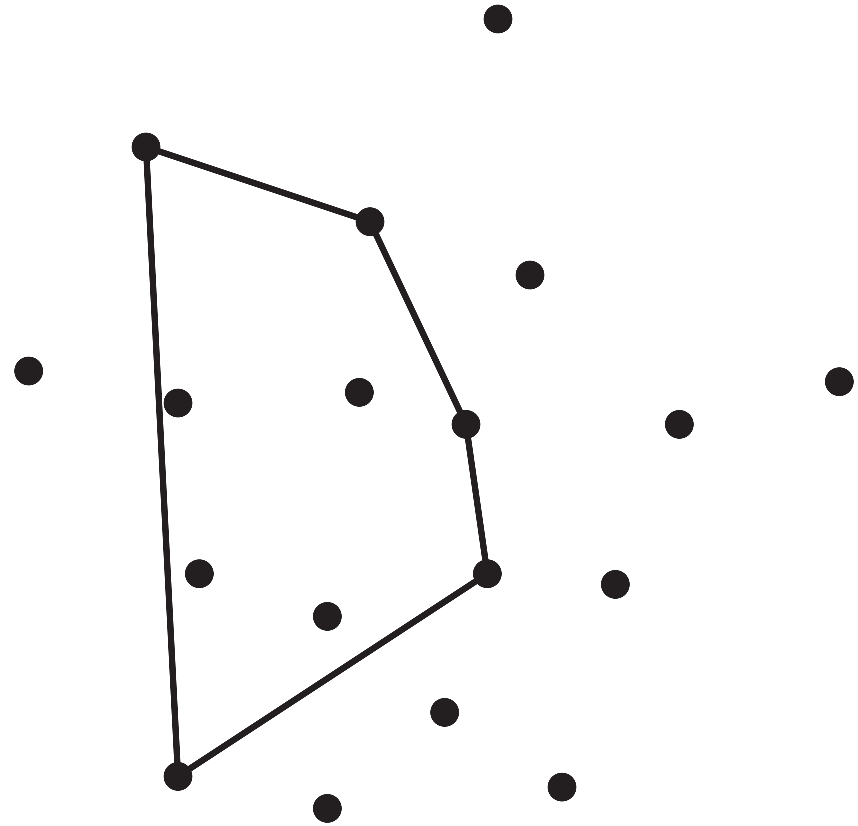
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

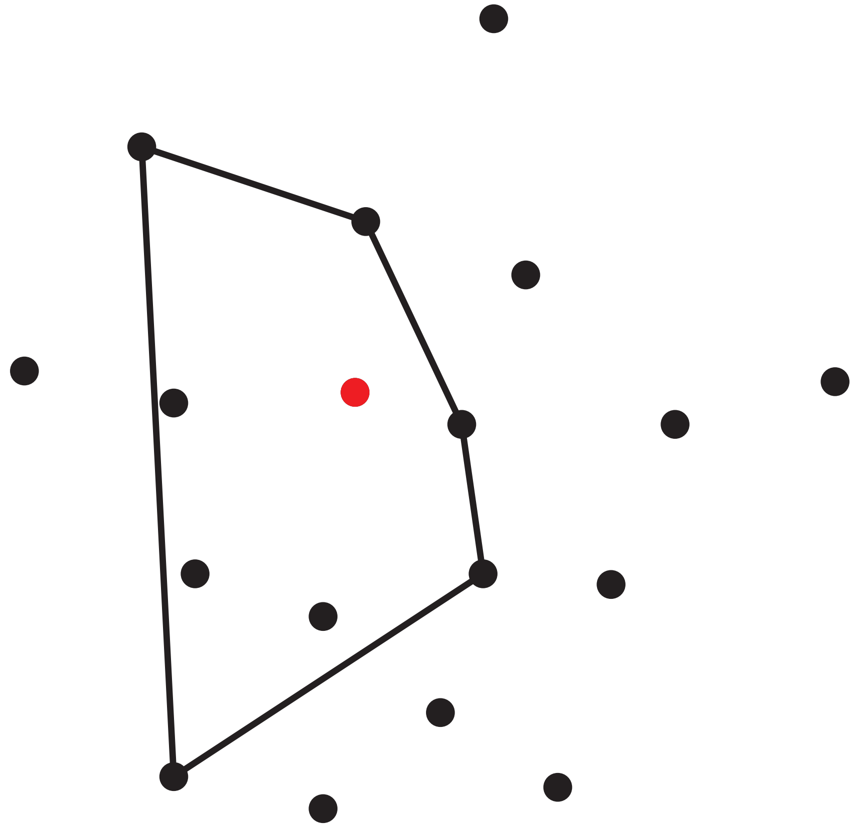
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

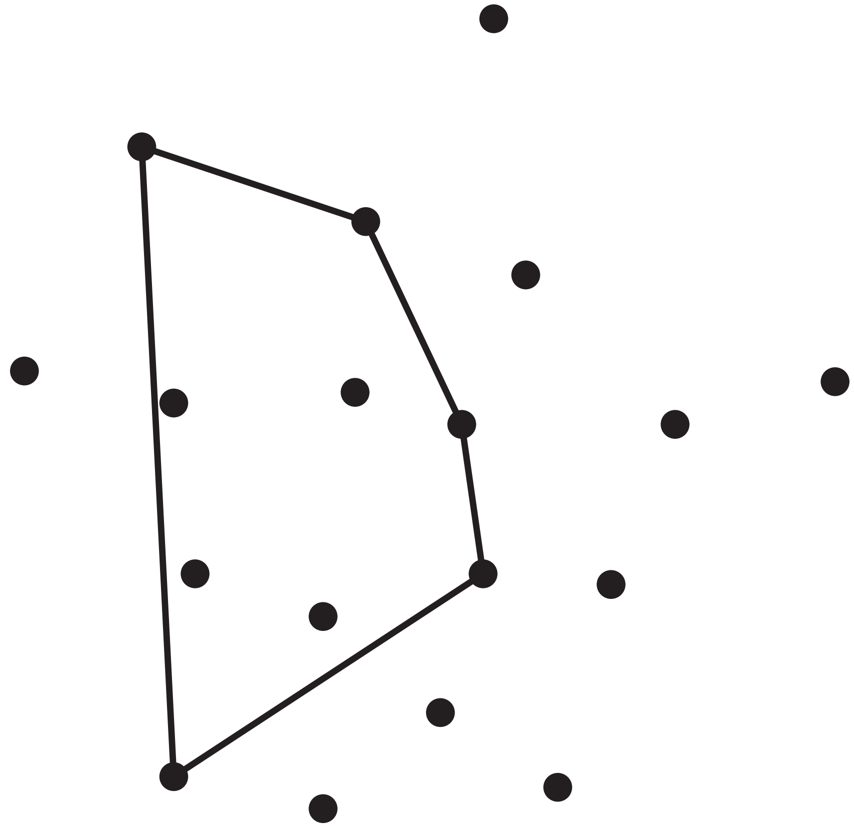
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

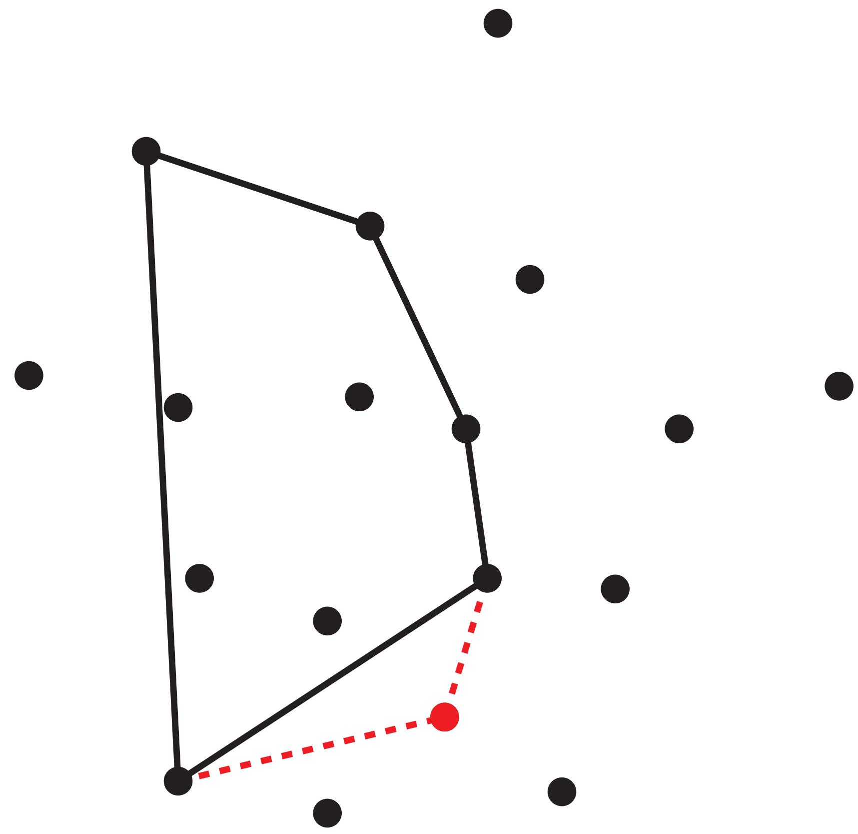
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adva…

Fr… , do:

…e exterior of the polygon defined by l:

…te the points p_l and p_r

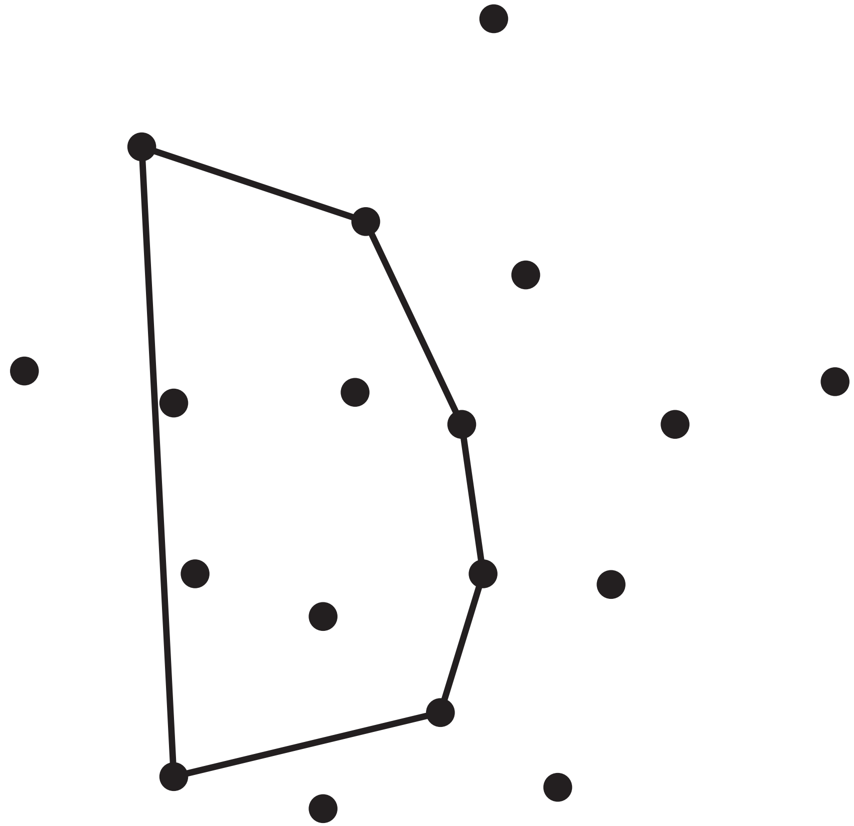
…g the supporting lines

…$_i$ to the polygon

…e the chain $p_l, \ldots, p_r$ in l

…he chain p_l, p_i, p_r

Retu…



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adva

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

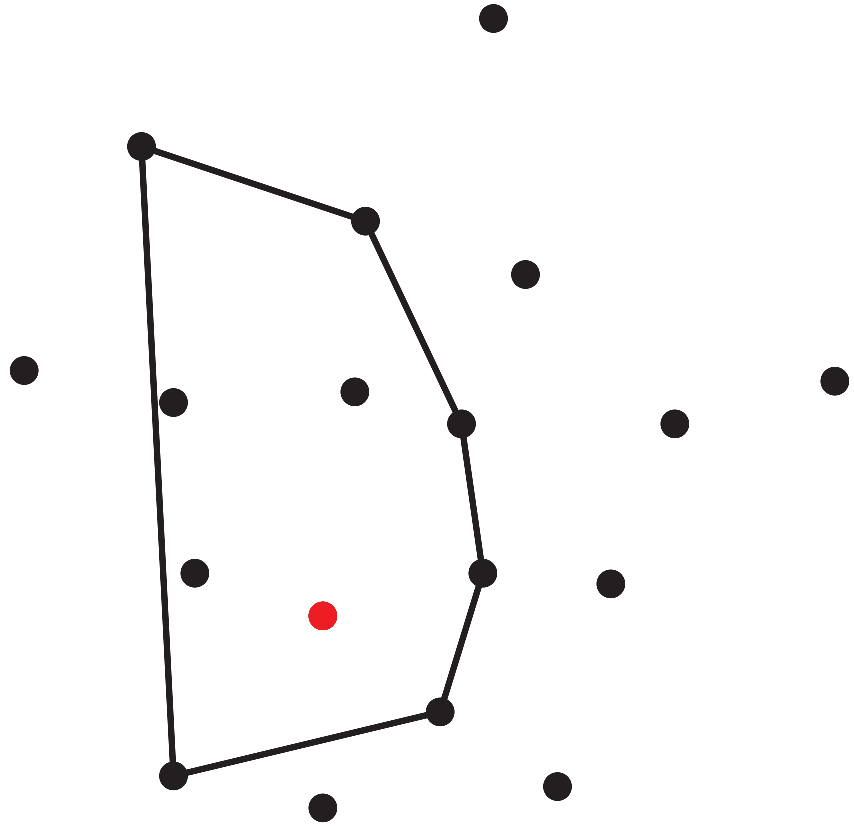
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

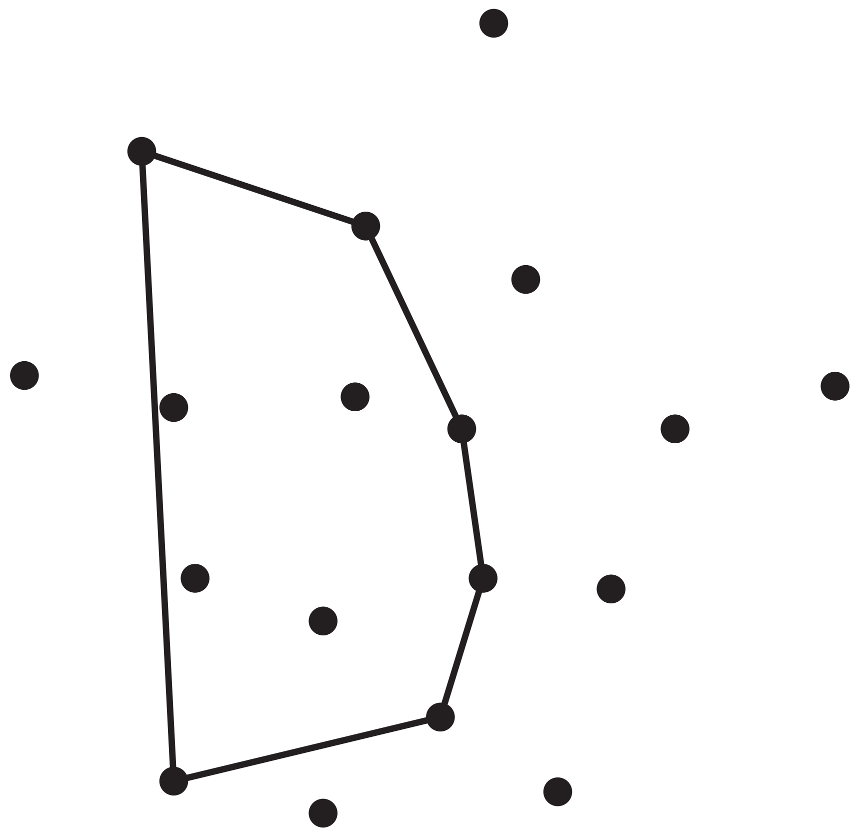
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adva

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

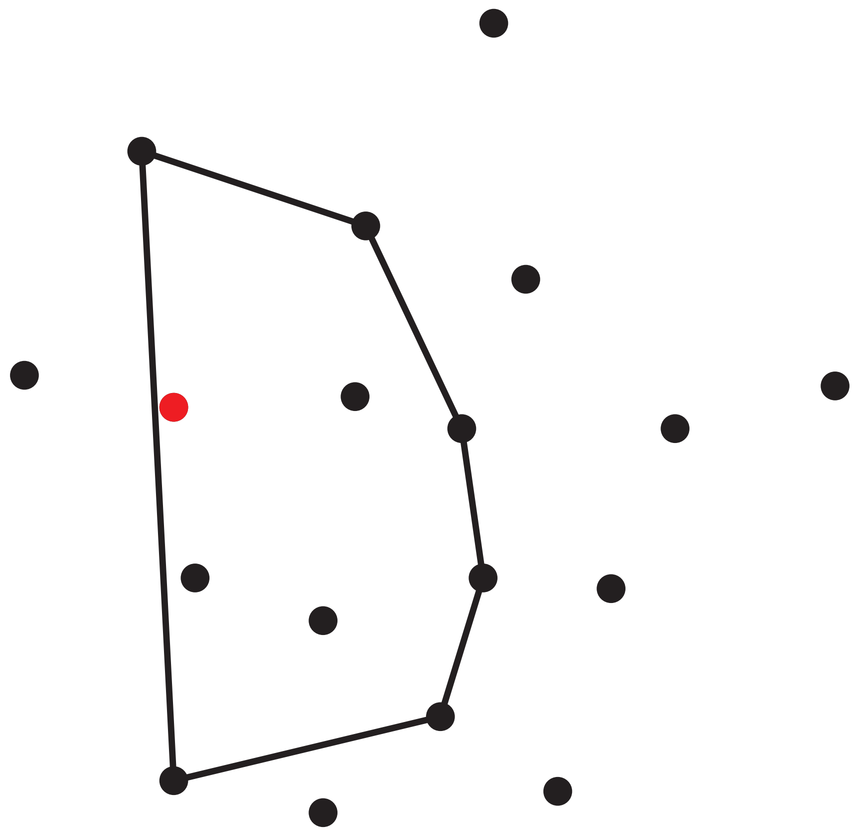
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv…

Fr… , do:

…e exterior of the polygon defined by l:

…te the points p_l and p_r

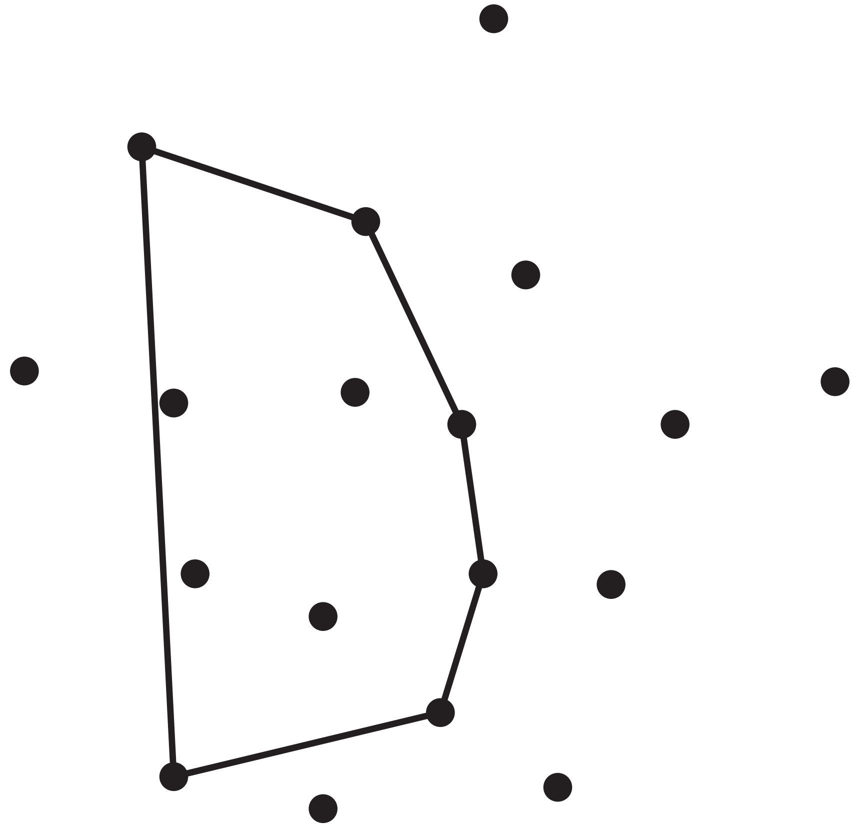
…g the supporting lines

…$_i$ to the polygon

…e the chain $p_l, \ldots, p_r$ in l

…he chain p_l, p_i, p_r

Retu…



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

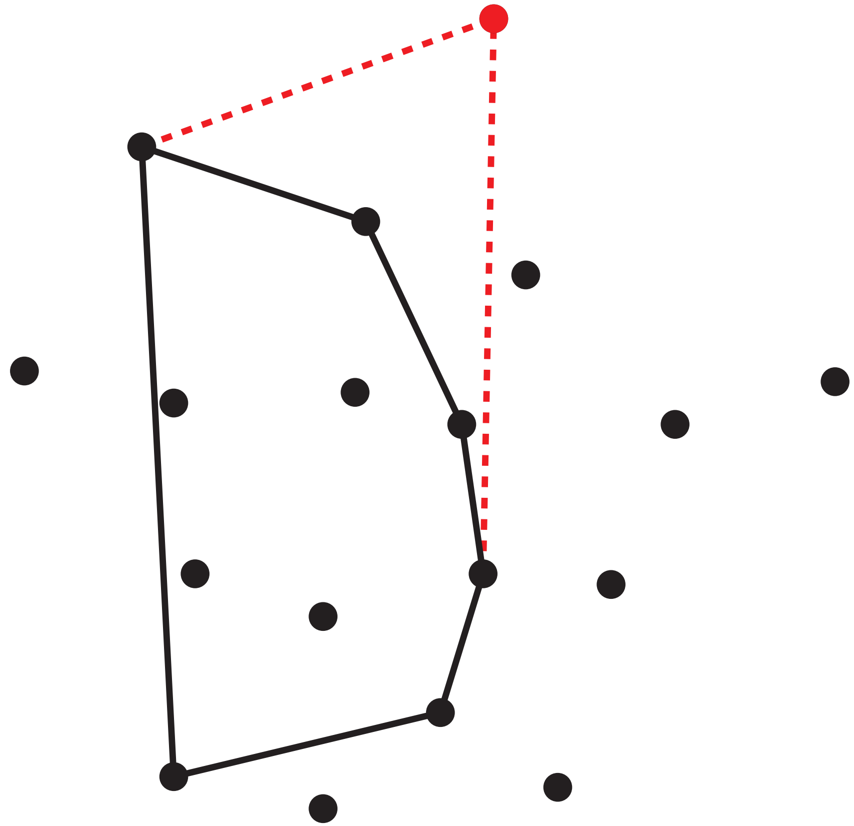
g the supporting lines

p_i to the polygon

e the chain $p_l, \dots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

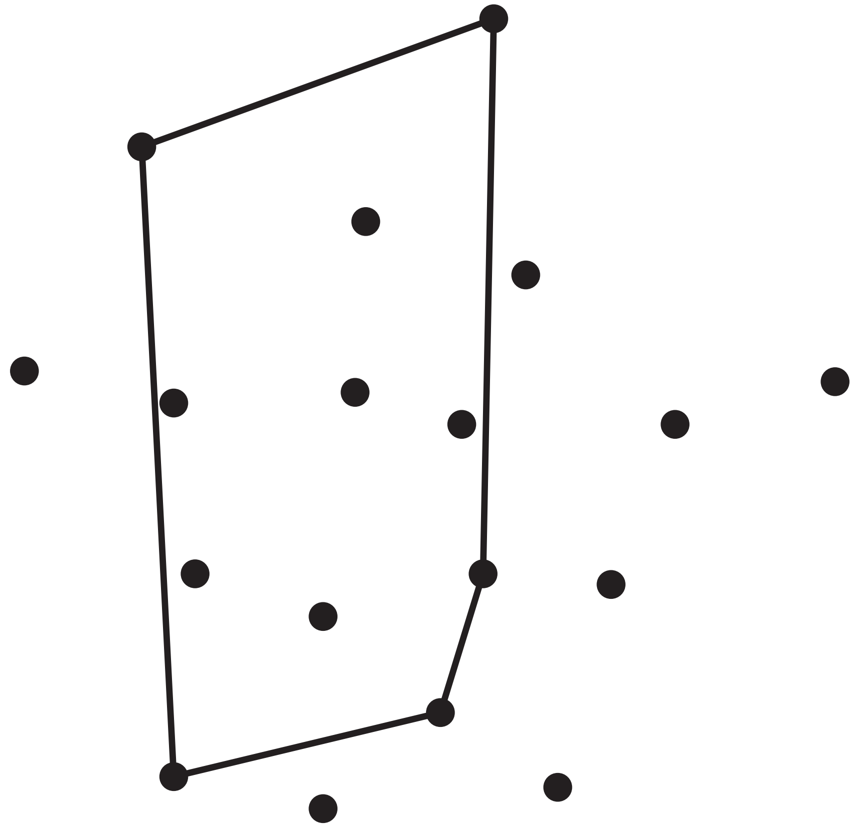
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

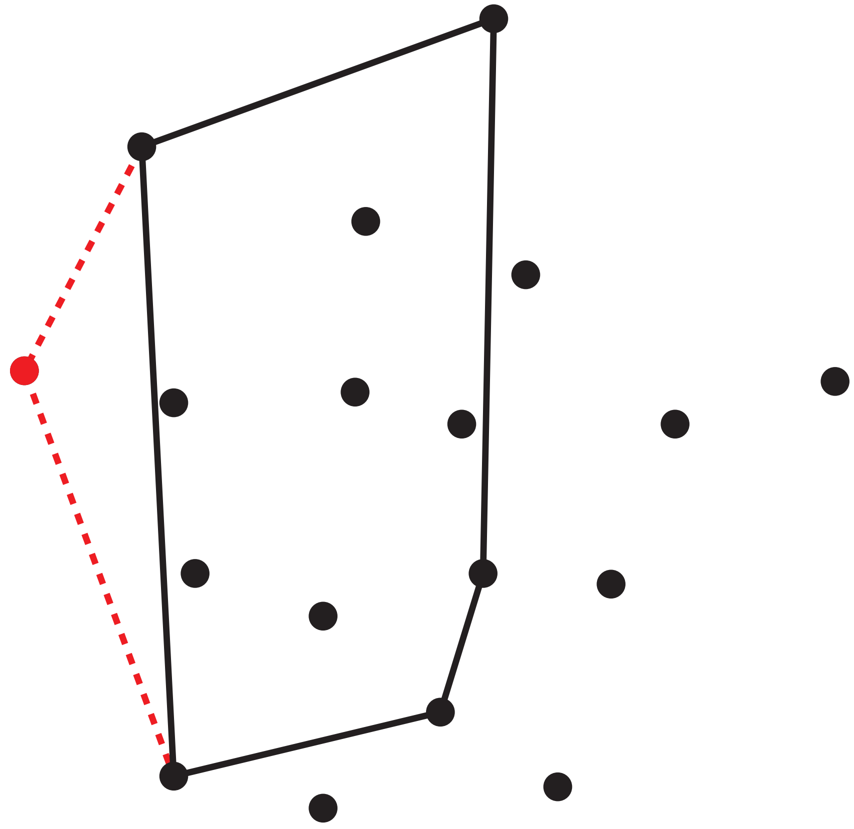
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr..., do:

e exterior of the polygon defined by l:

te the points p_l and p_r

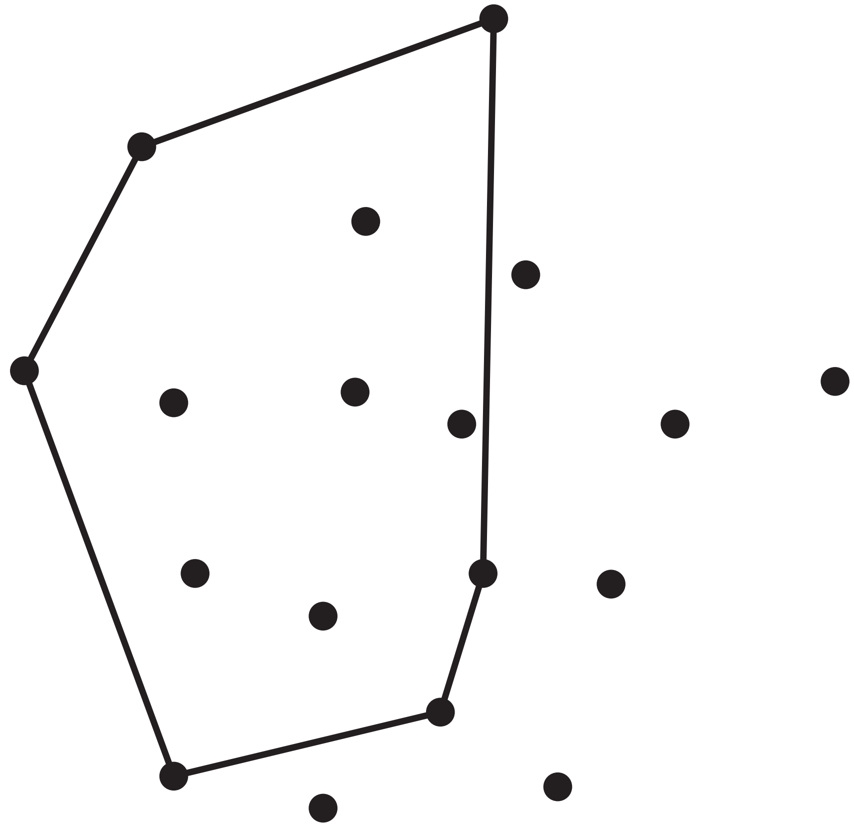
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



CONVEX HULL IN 2D

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

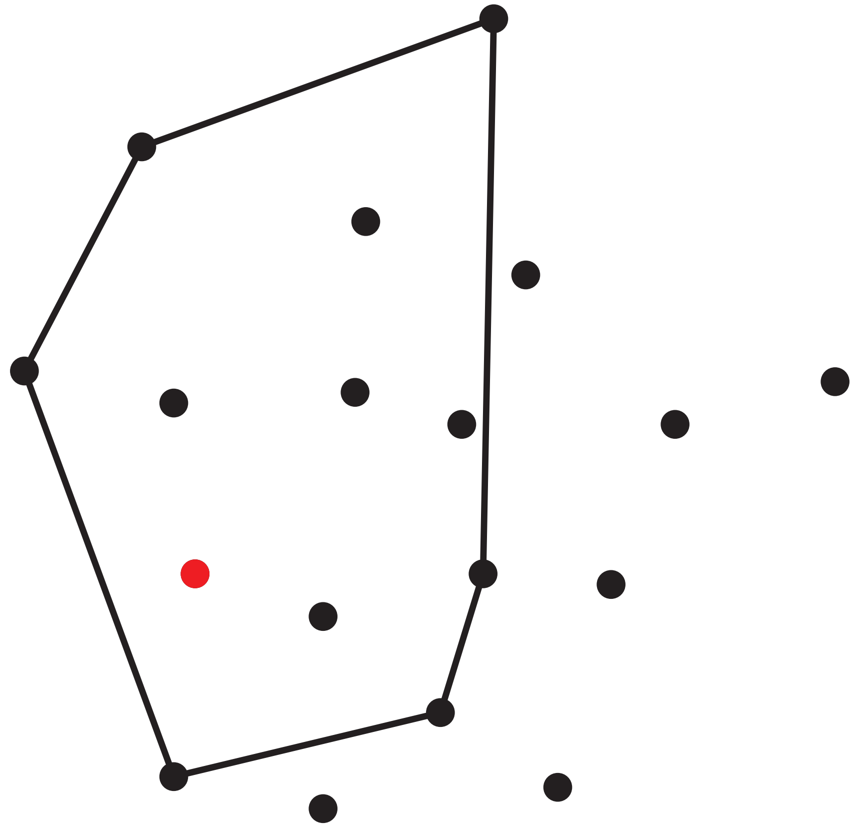
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \dots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr..., do:

e exterior of the polygon defined by l:

te the points p_l and p_r

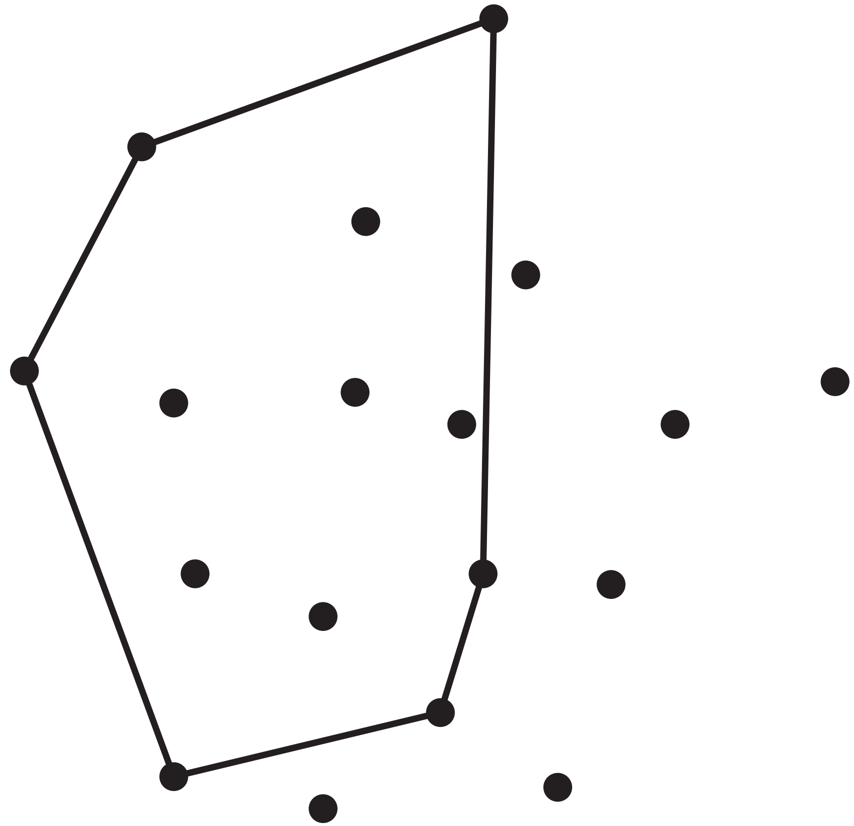
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adva

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

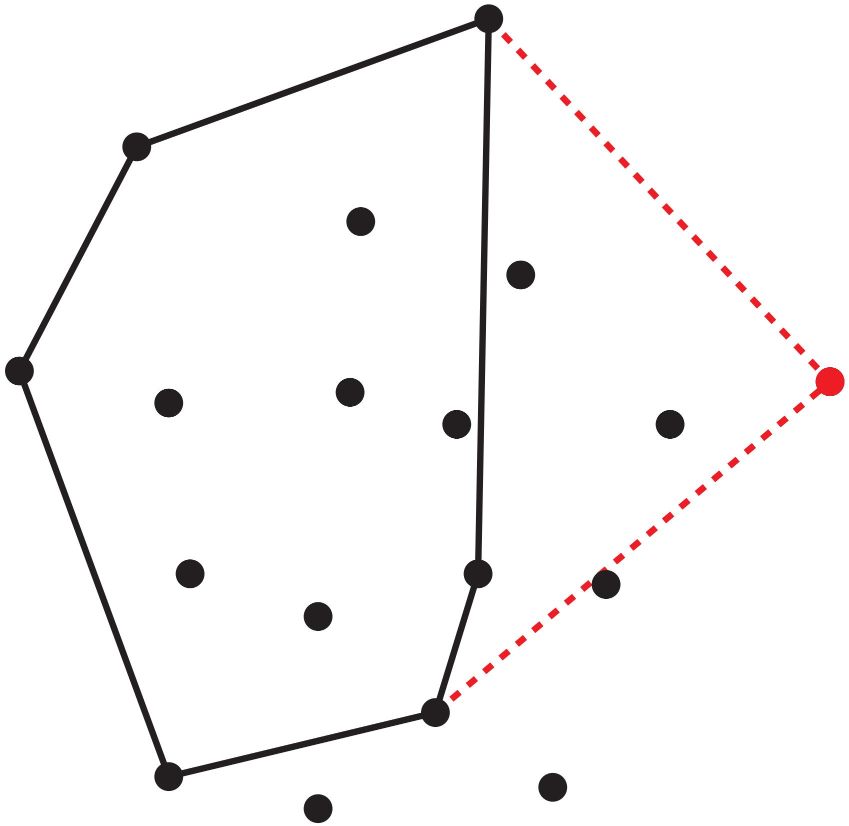
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adva

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

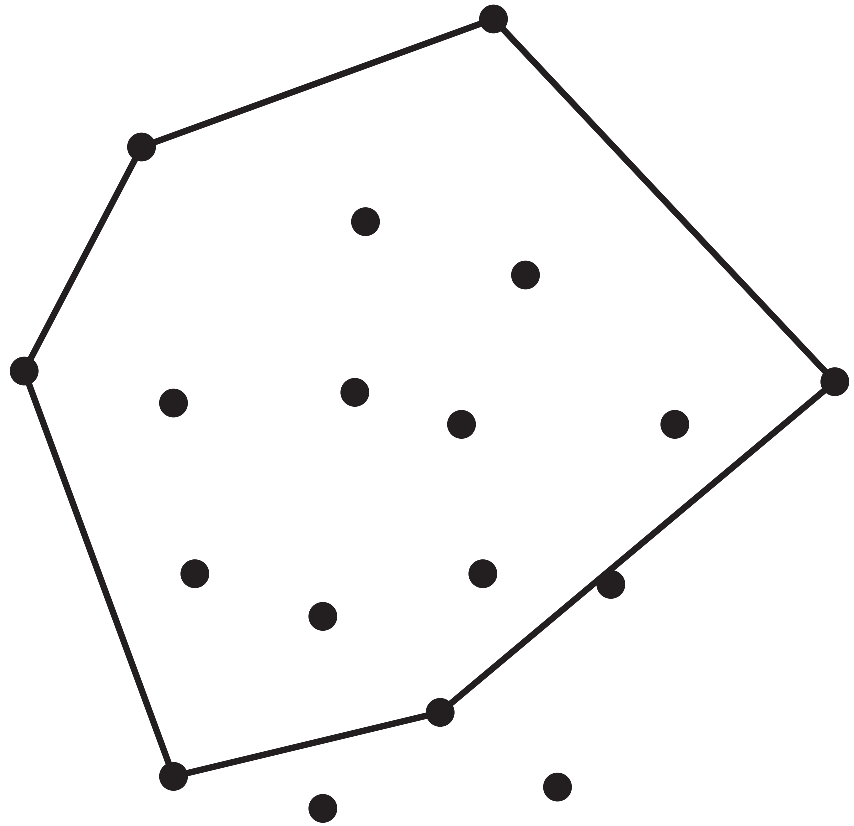
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

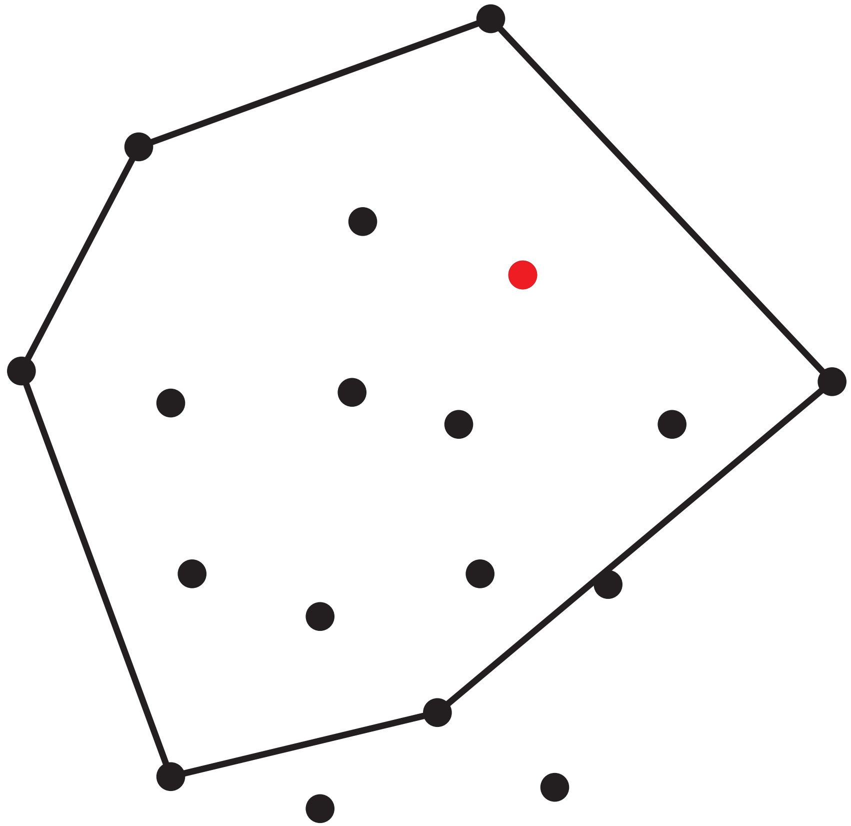
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

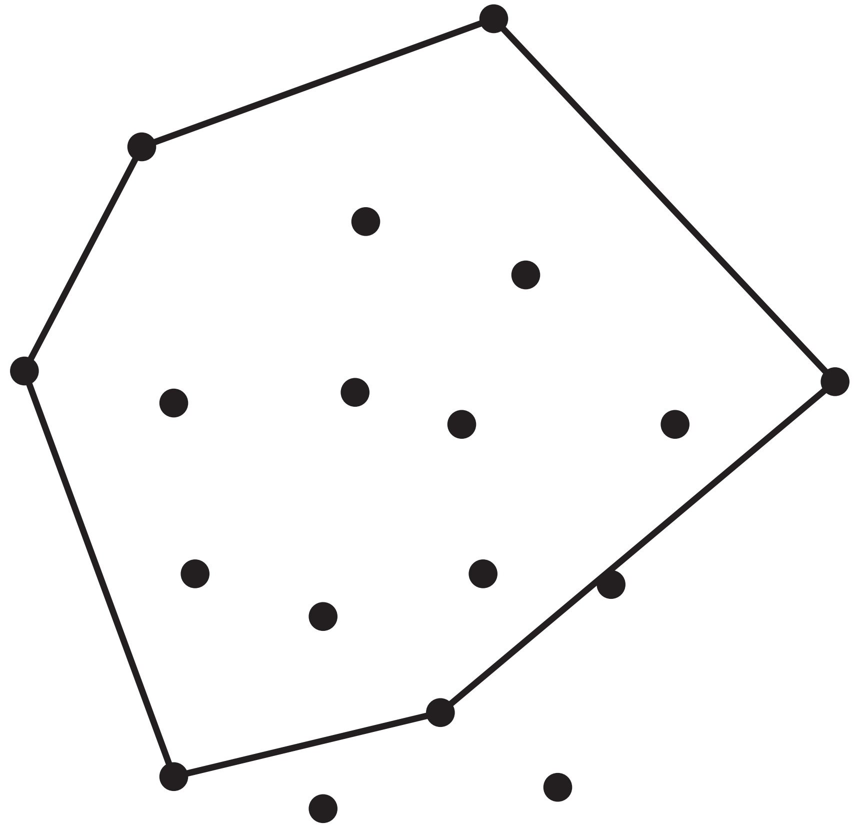
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

Adv

Fr , do:

e exterior of the polygon defined by l:

te the points p_l and p_r

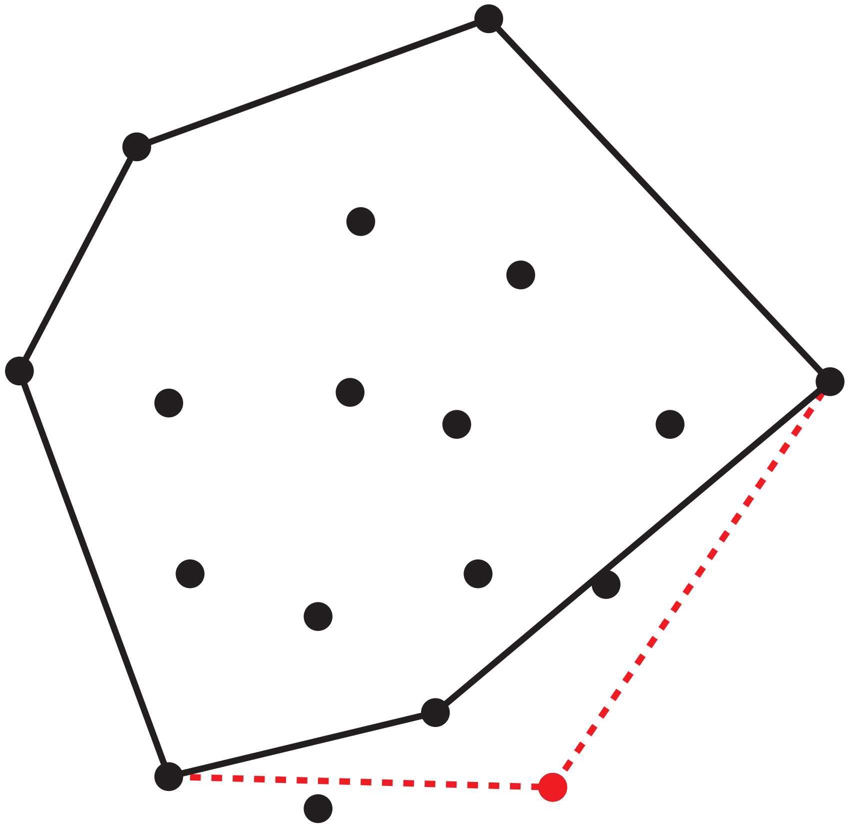
g the supporting lines

$_i$ to the polygon

e the chain $p_l, \ldots, p_r$ in l

he chain p_l, p_i, p_r

Retu



Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

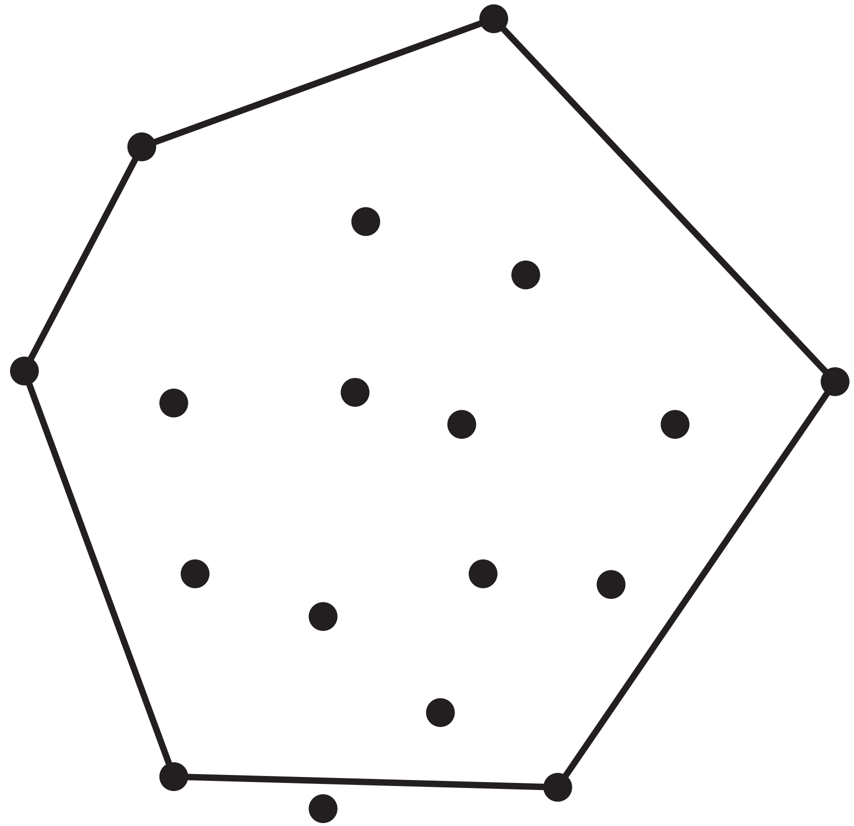
Advance

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

$l = p_1, p_2, p_3$

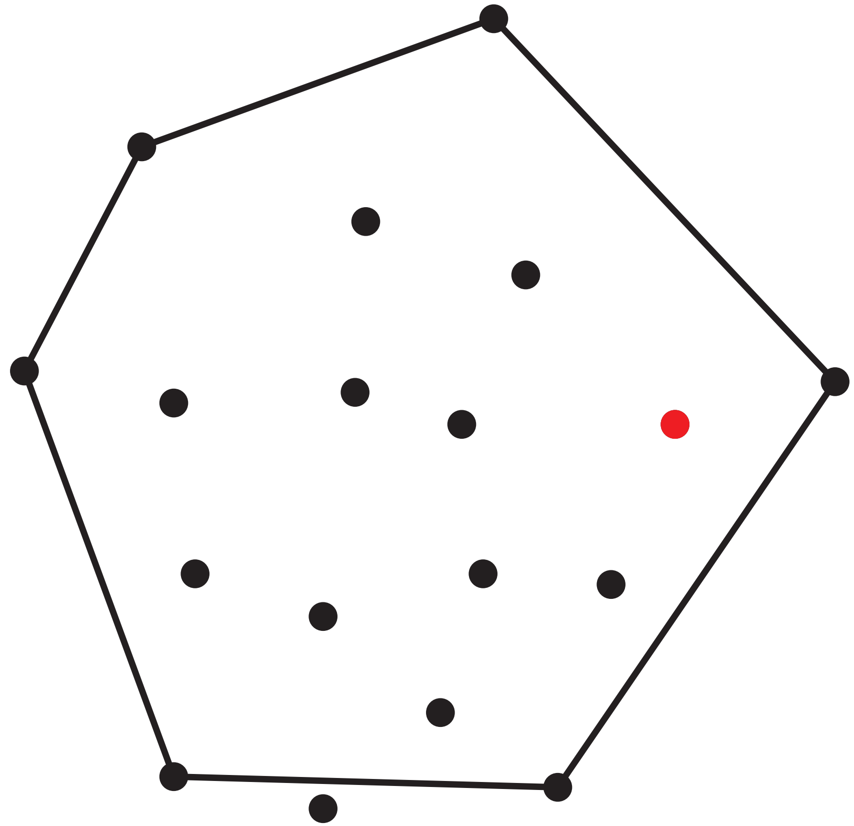
Advance

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

$l = p_1, p_2, p_3$

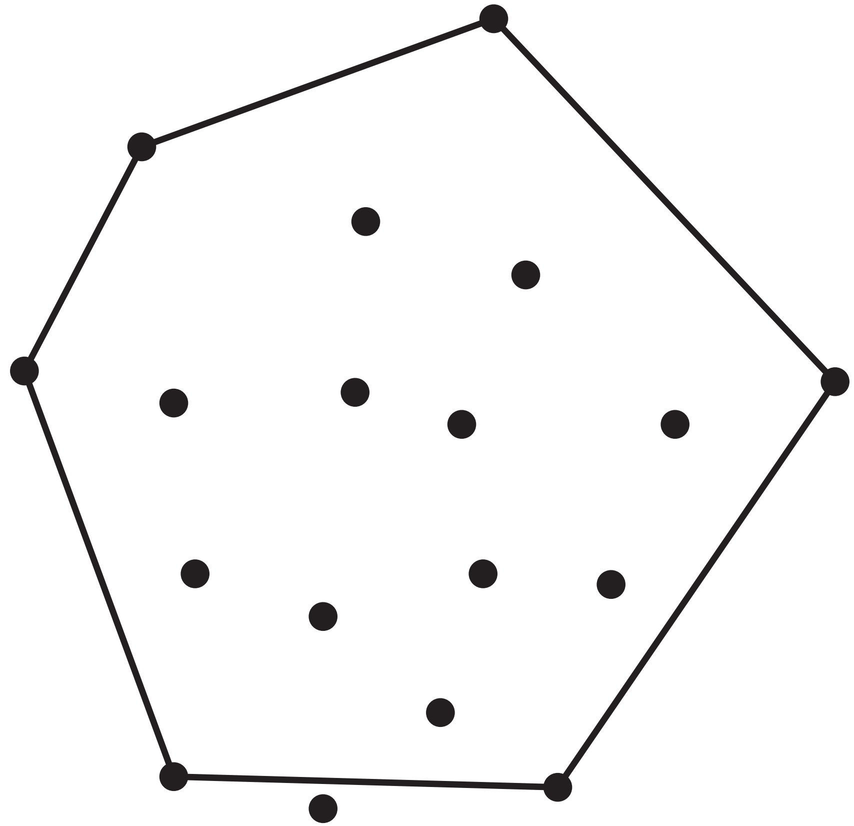
Advance

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

$l = p_1, p_2, p_3$

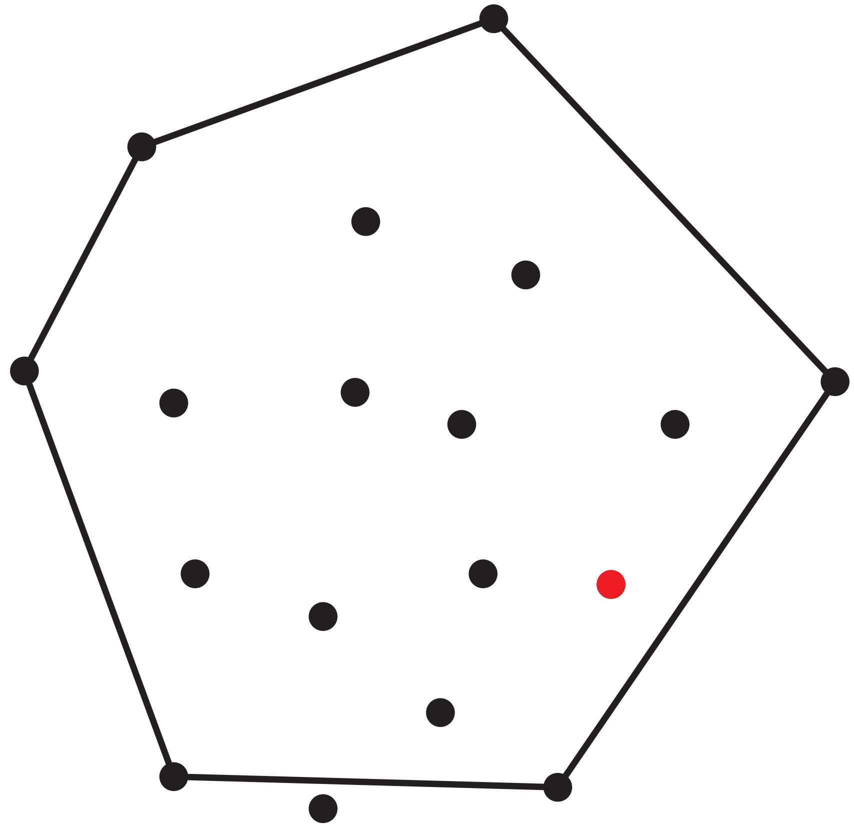
Advance

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

$l = p_1, p_2, p_3$

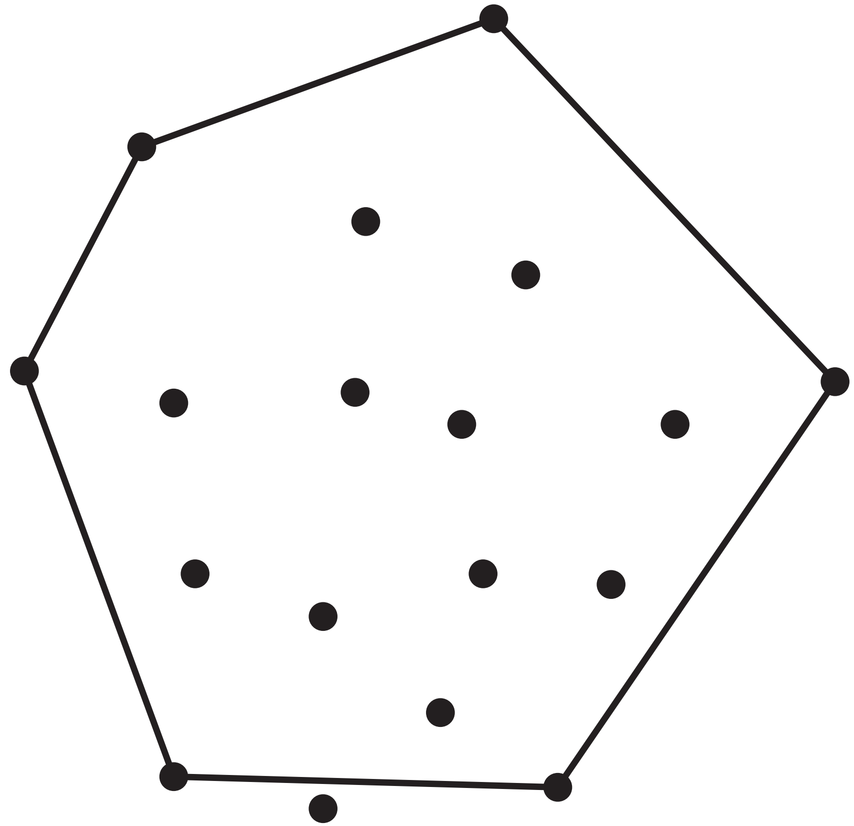
Advance

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

$l = p_1, p_2, p_3$

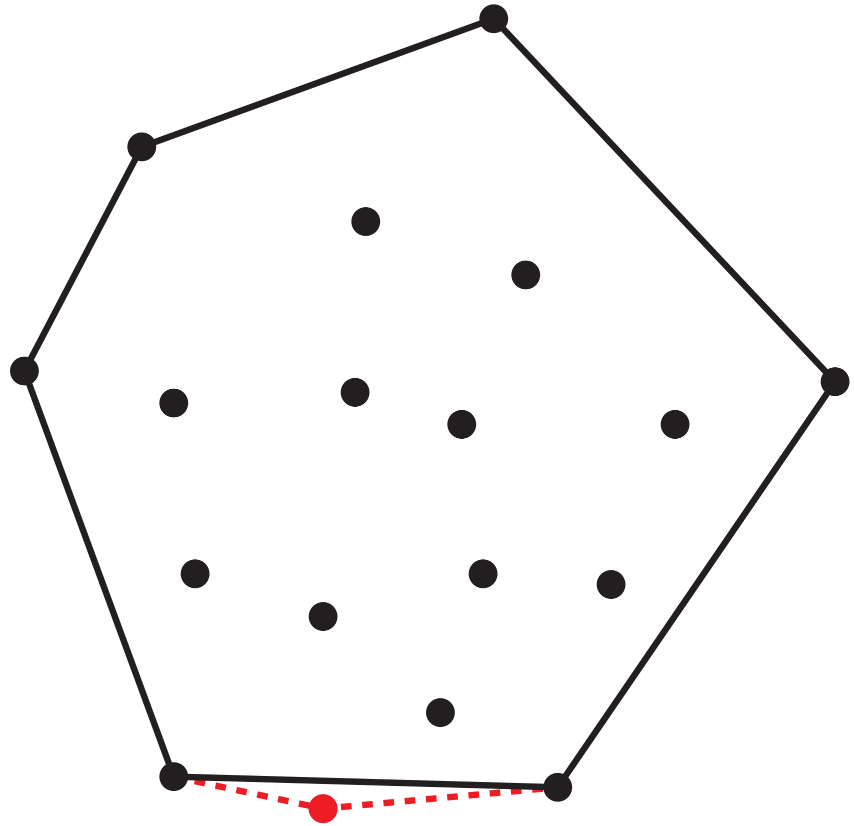
Advance

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

$l = p_1, p_2, p_3$

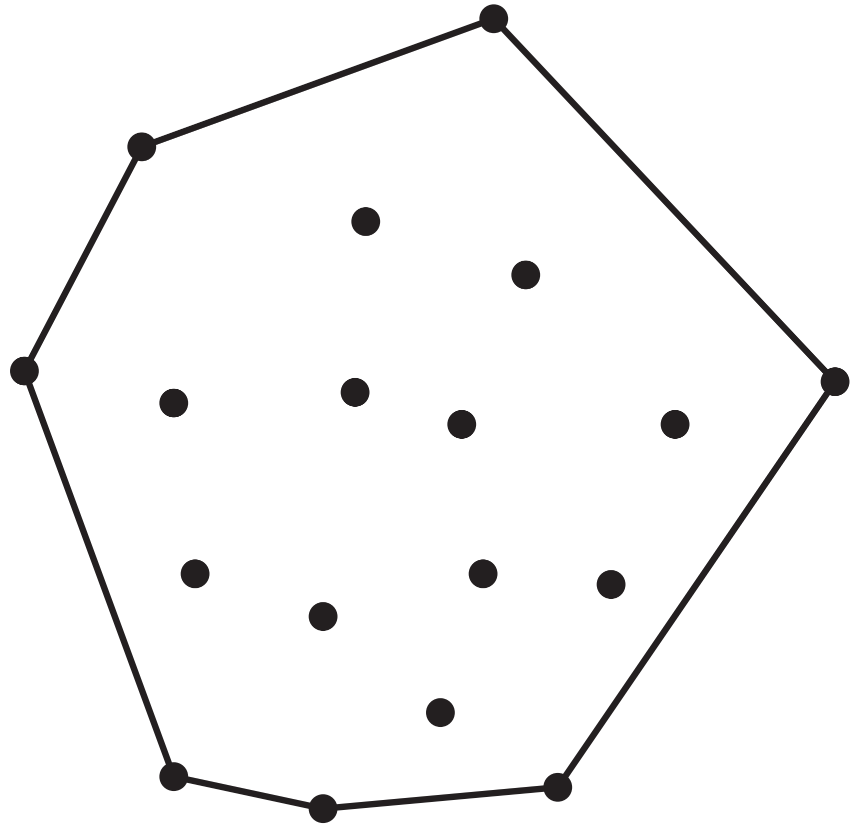
Advance

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

$l = p_1, p_2, p_3$

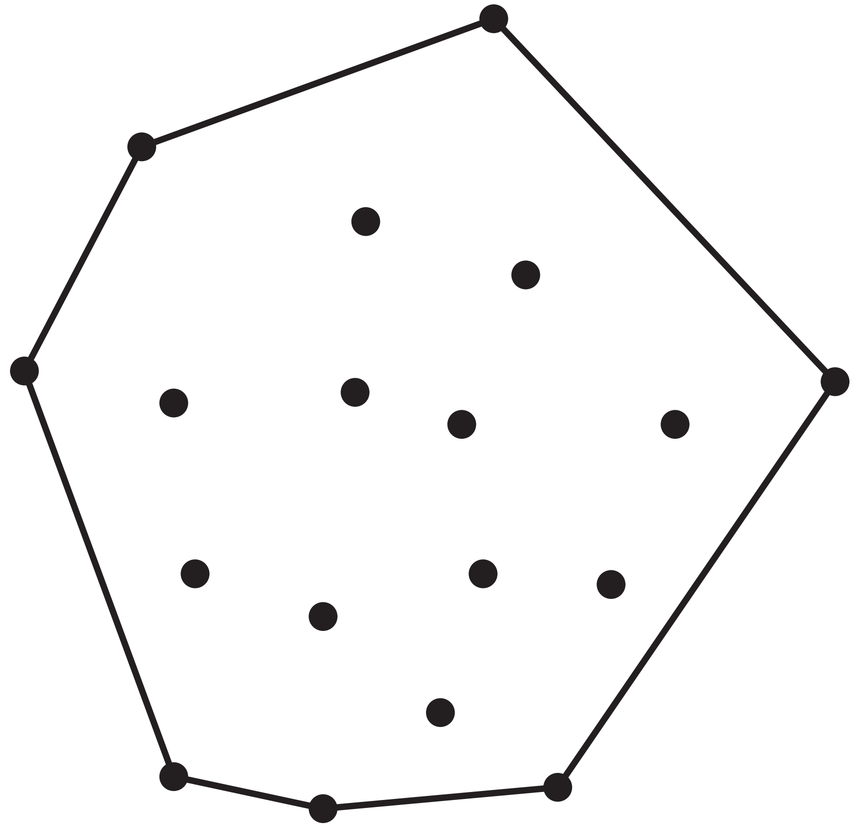
Advance

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



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

Incremental algorithm

Initialization

$l = p_1, p_2, p_3$

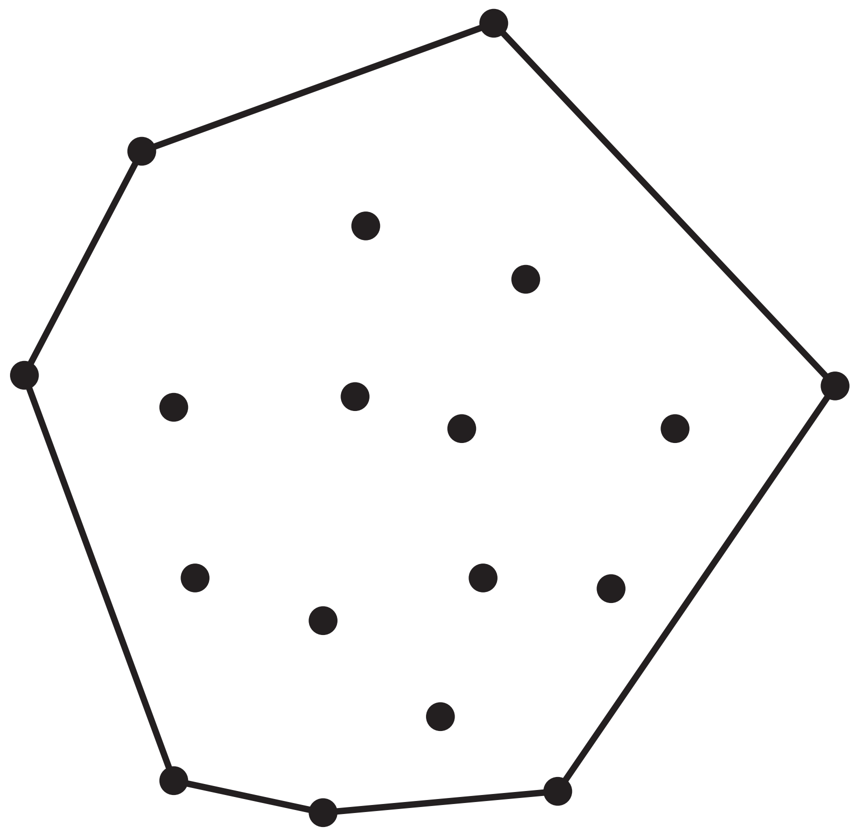
Advance

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

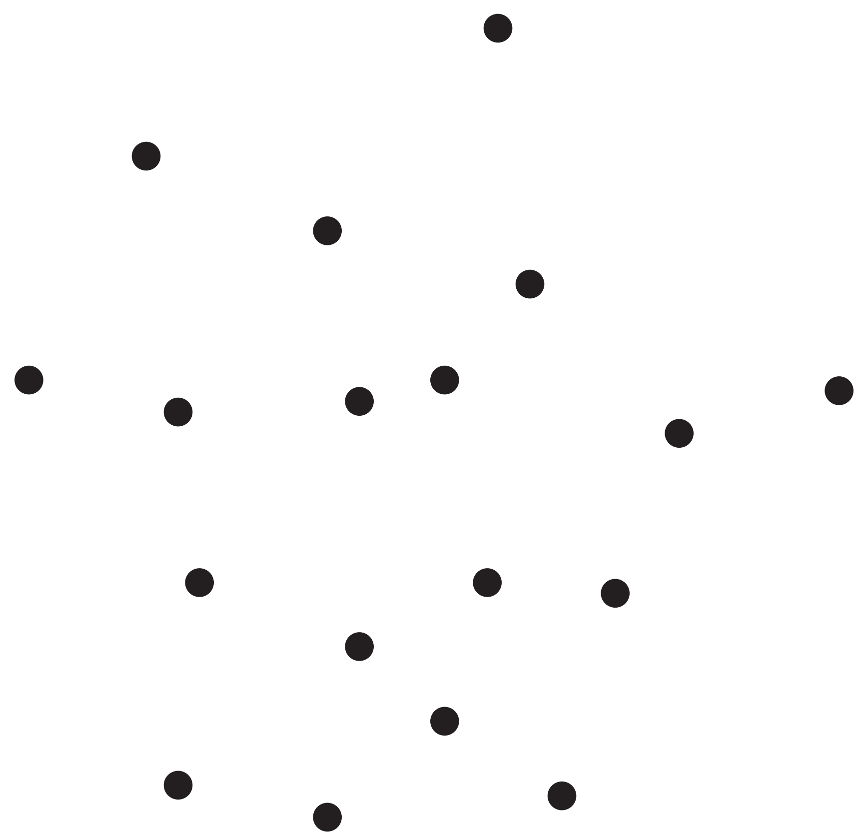
Divide-and-conquer algorithm

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



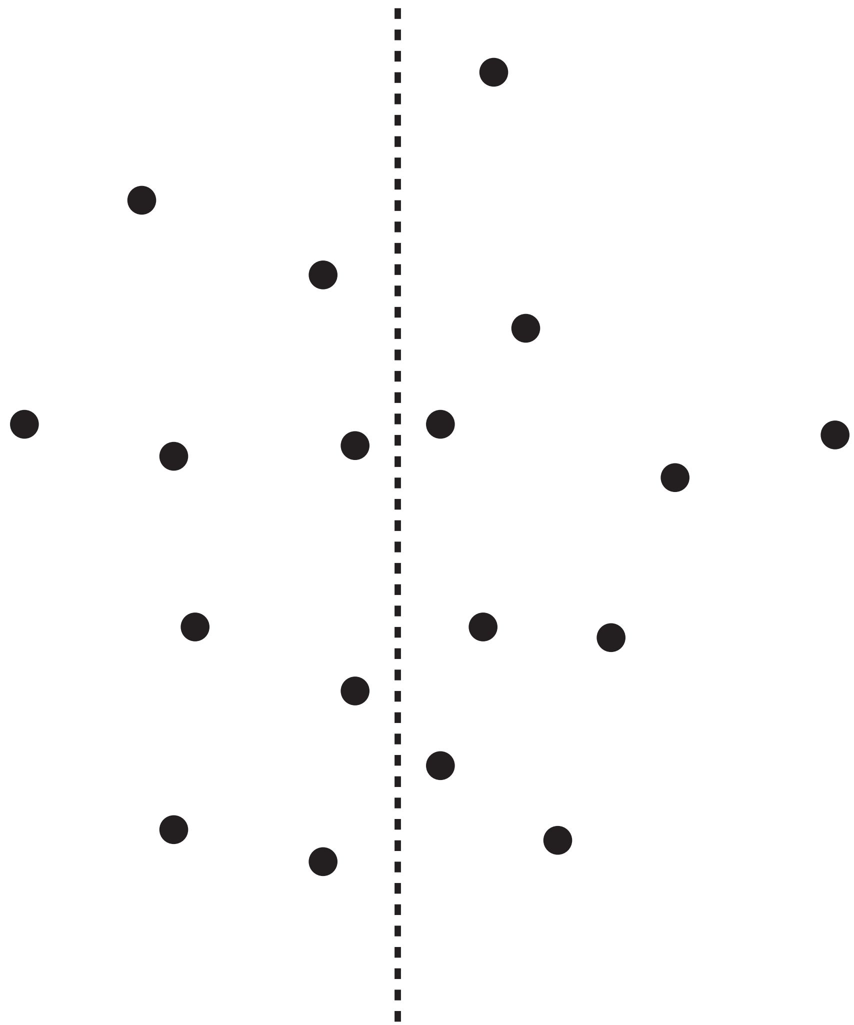
Divide-and-conquer algorithm

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



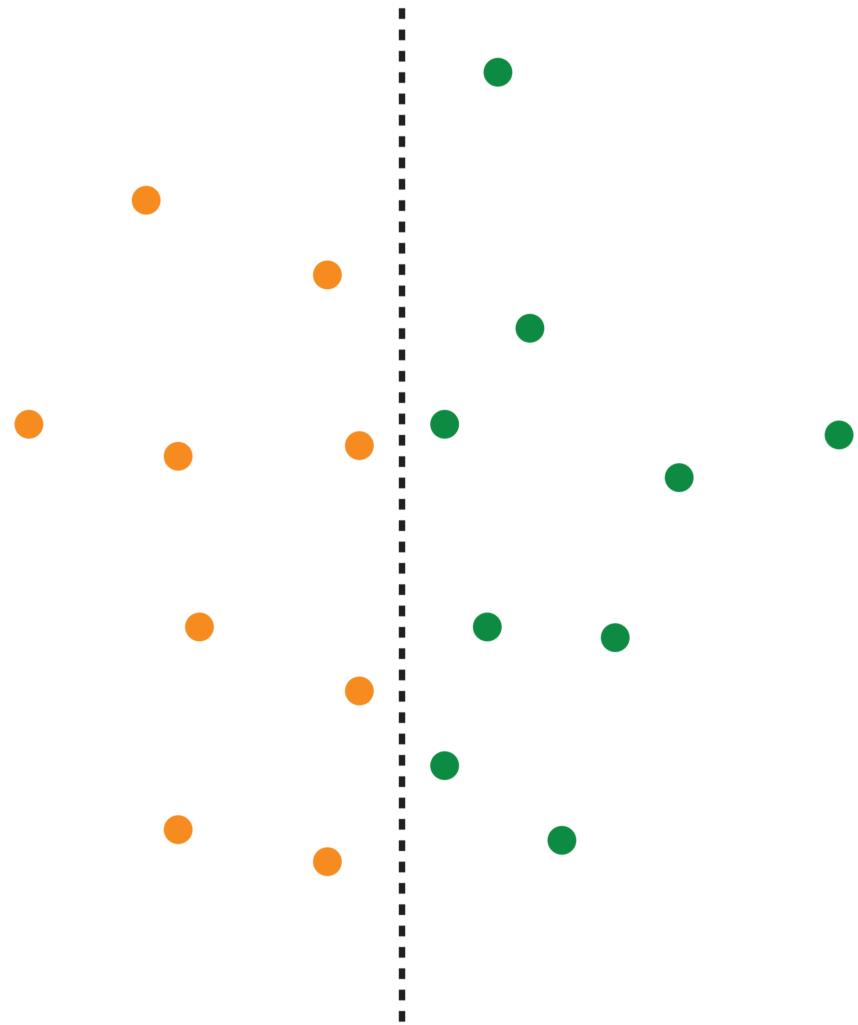
Divide-and-conquer algorithm

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



CONVEX HULL IN 2D

Divide-and-conquer algorithm

Initialization

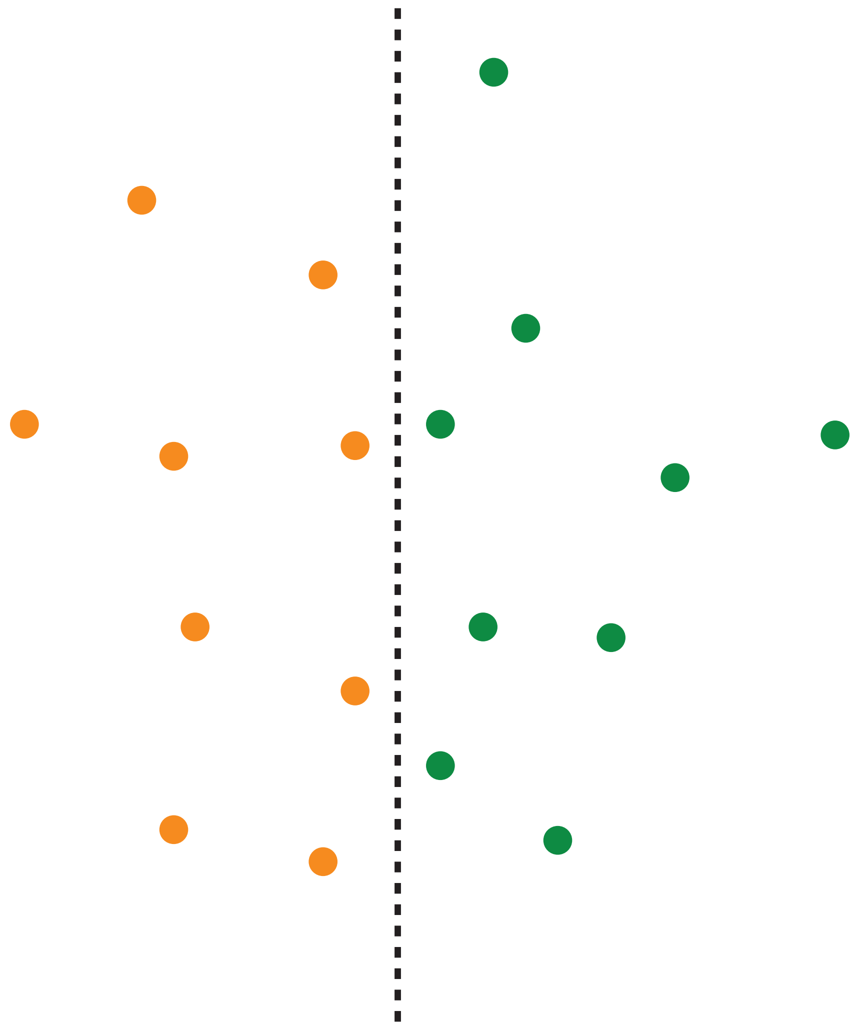
1. Sort the points by abscissae

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



Divide-and-conquer algorithm

Initialization

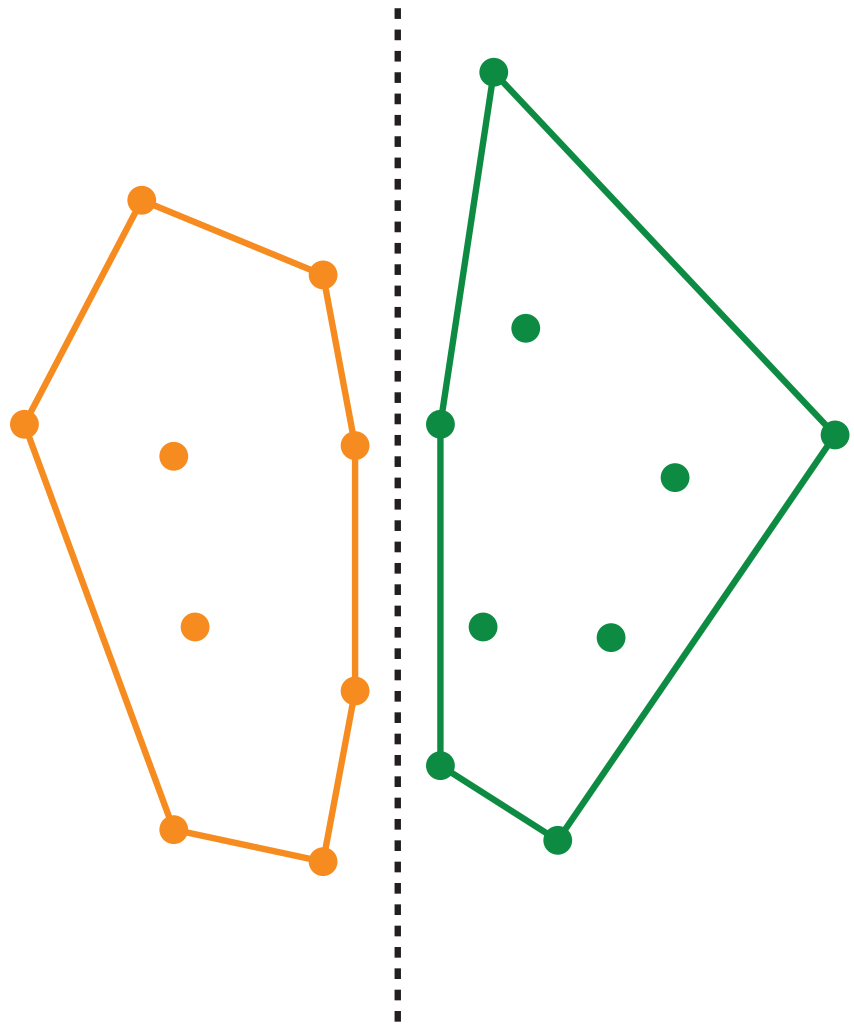
1. Sort the points by abscissae

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

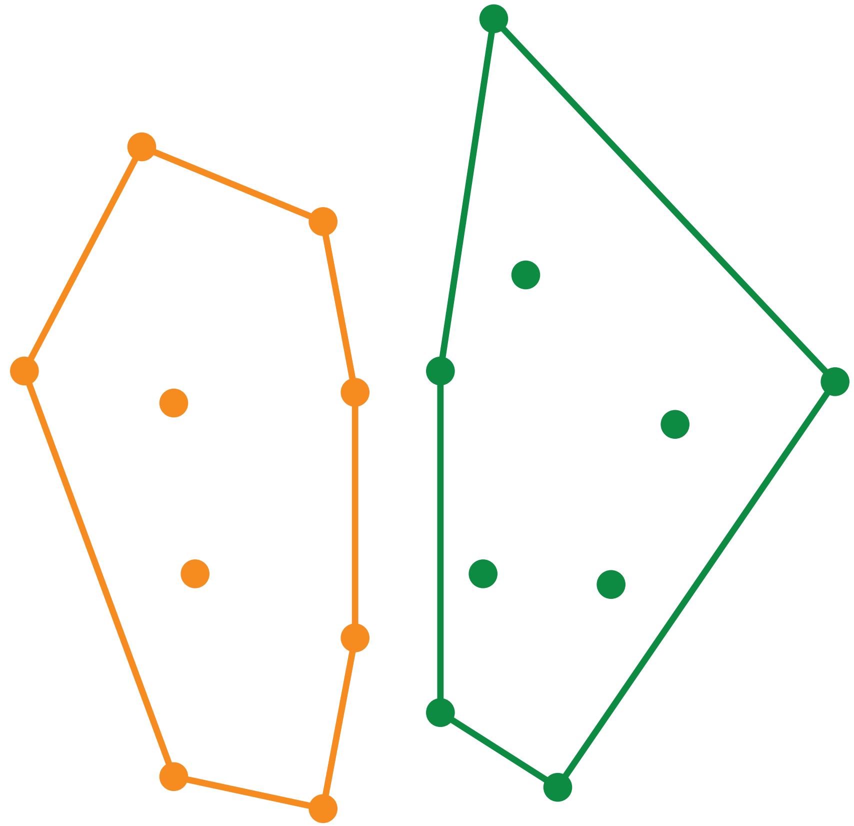
1. Sort the points by abscissae

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

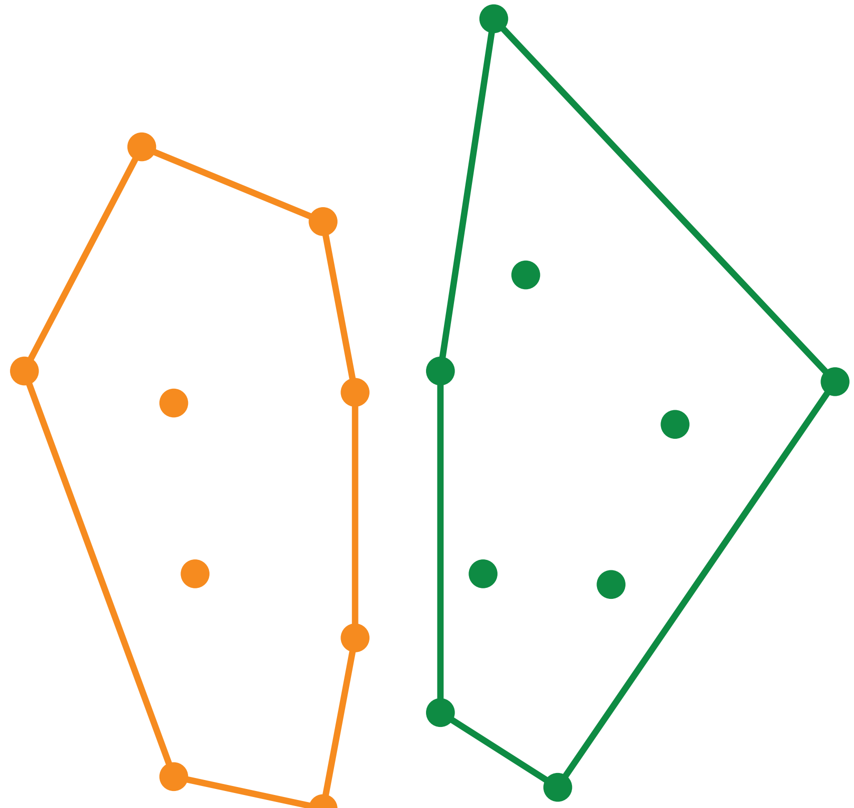
Recursion

1. Recursively compute the convex hull of the two subsets

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

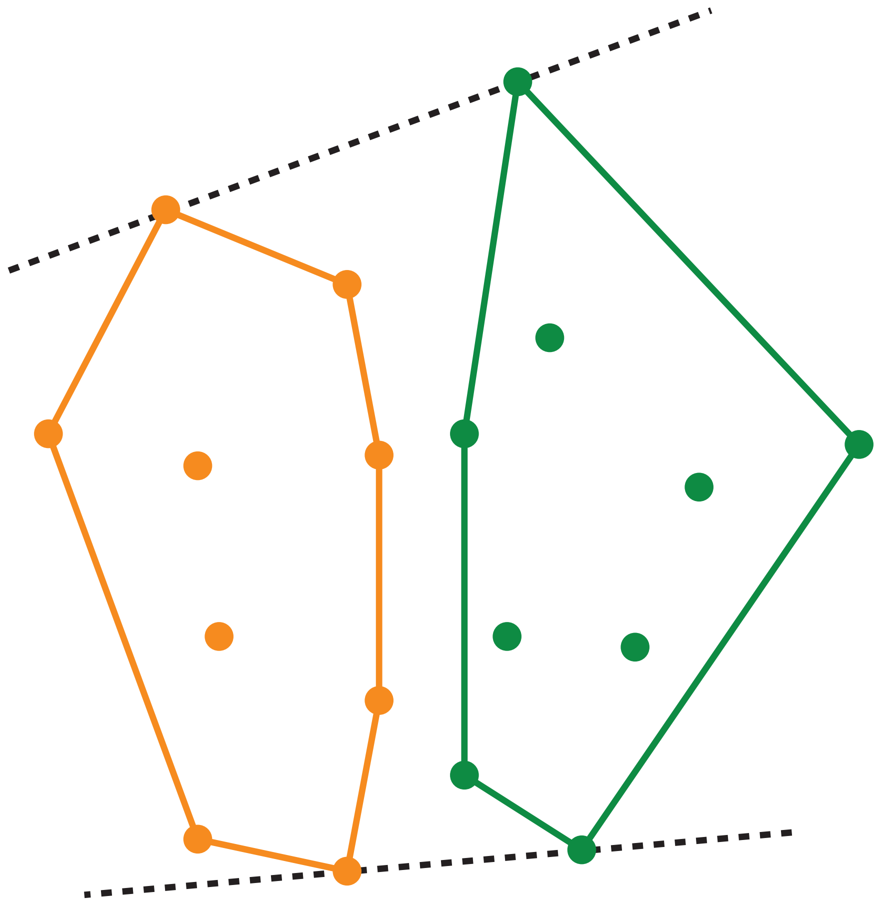
Recursion

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



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

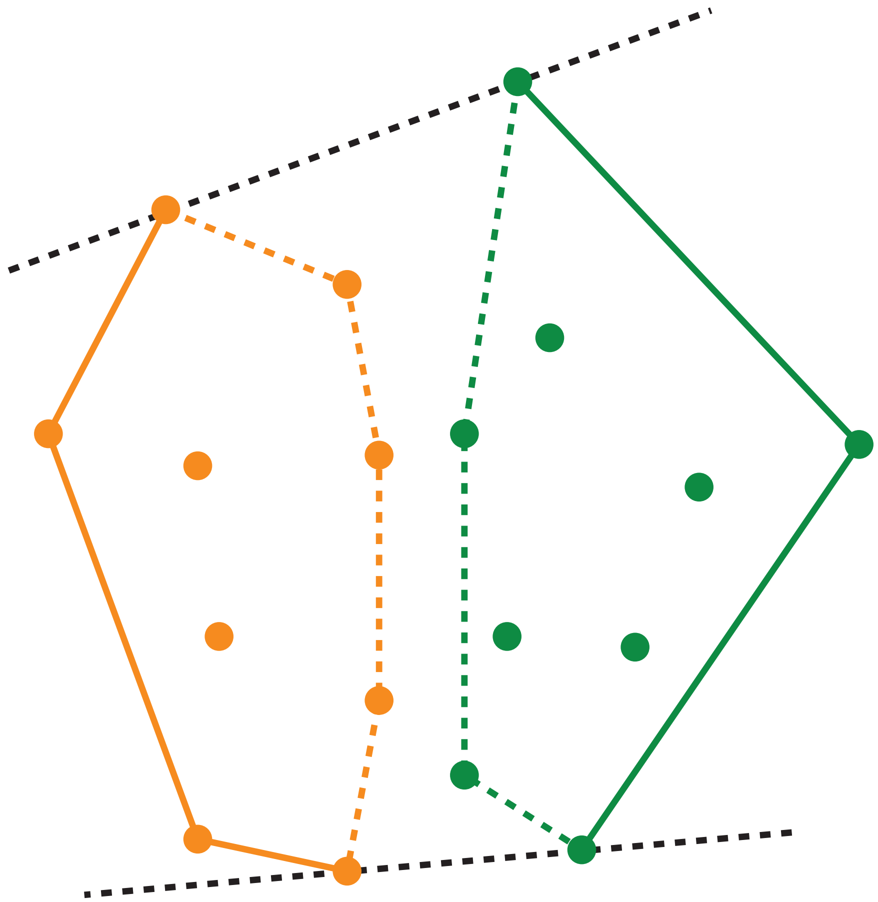
Recursion

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

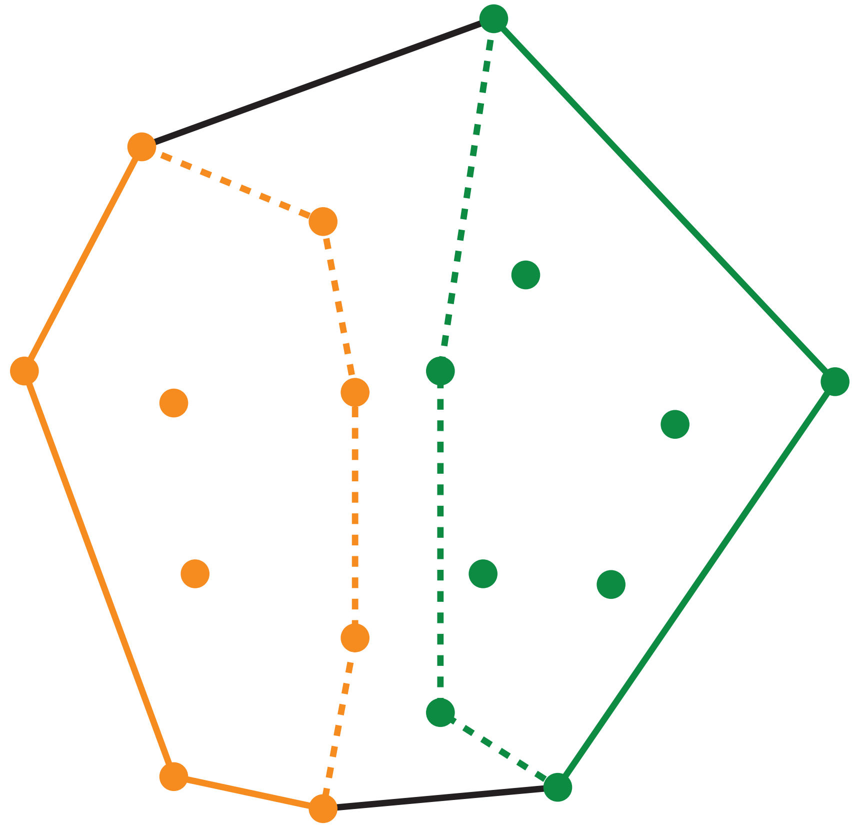
1. Divide the points (x_i, y_i) into two subsets, wrt the median value of the abscissae

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



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

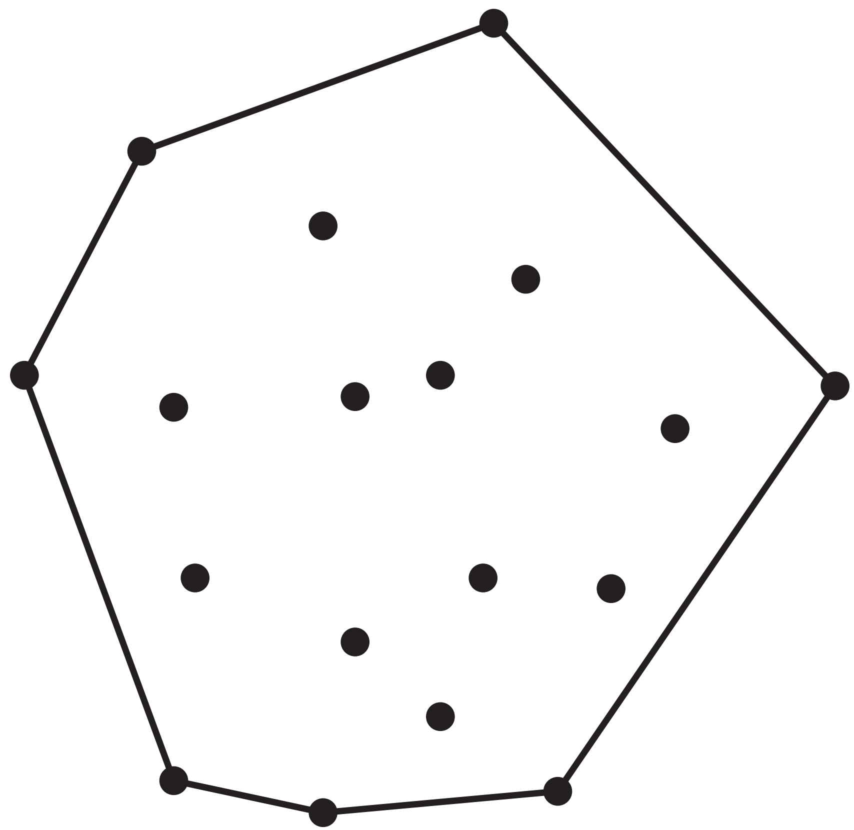
Recursion

1. Recursively compute the convex hull of the two subsets

Merge

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

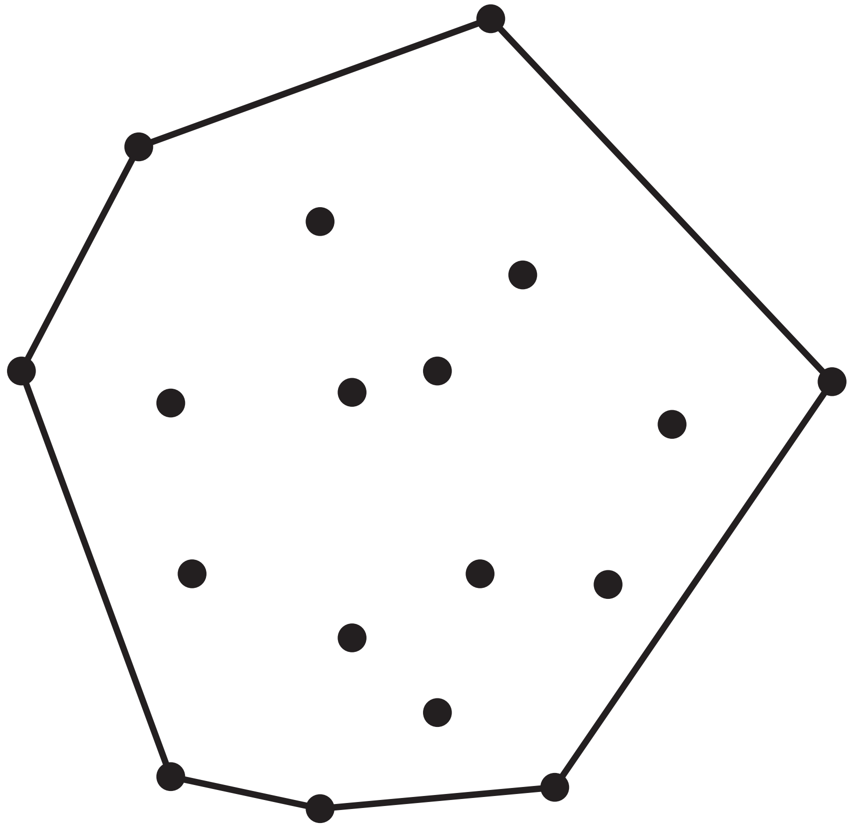
2. Delete the interior chains of the two polygons and join the external chains through the supporting segments



Divide-and-conquer algorithm

Running time

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

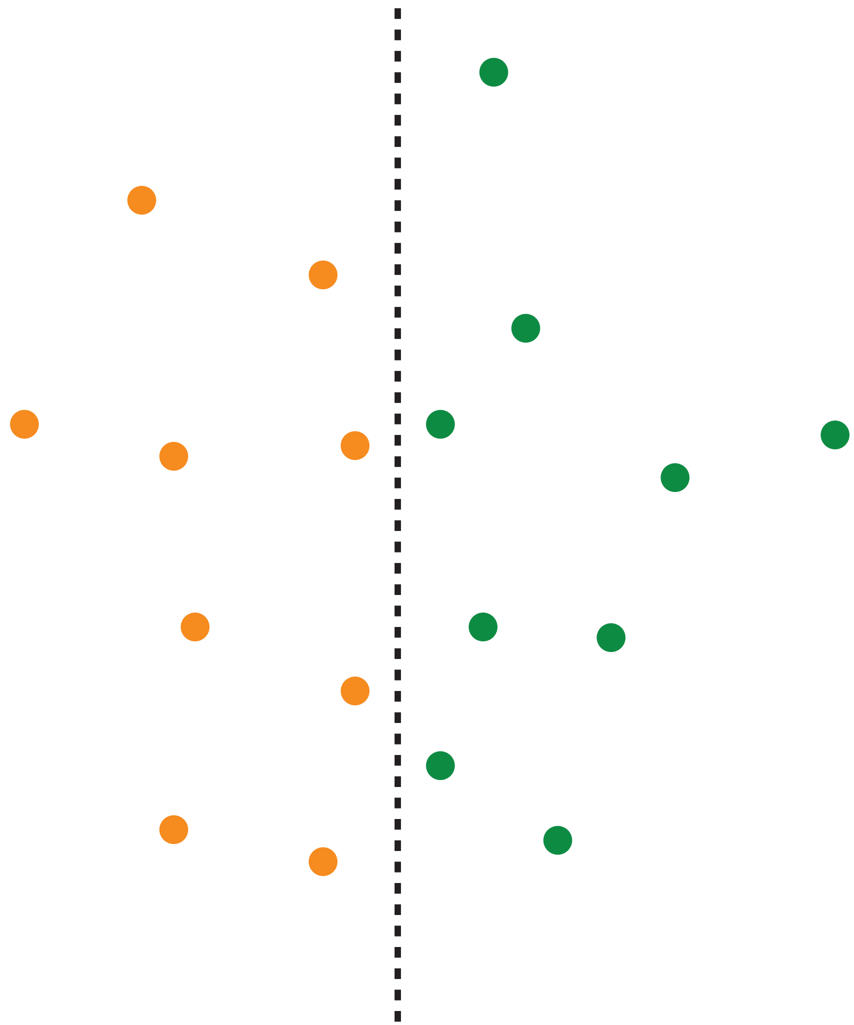


Divide-and-conquer algorithm

Running time

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

Division: $O(n)$



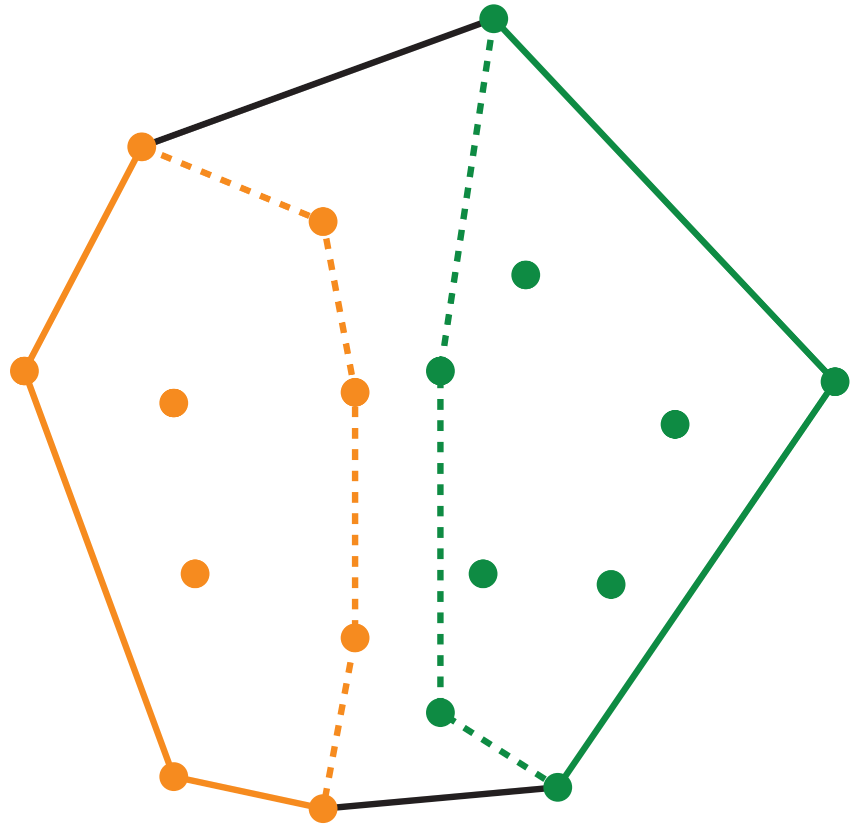
Divide-and-conquer algorithm

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)

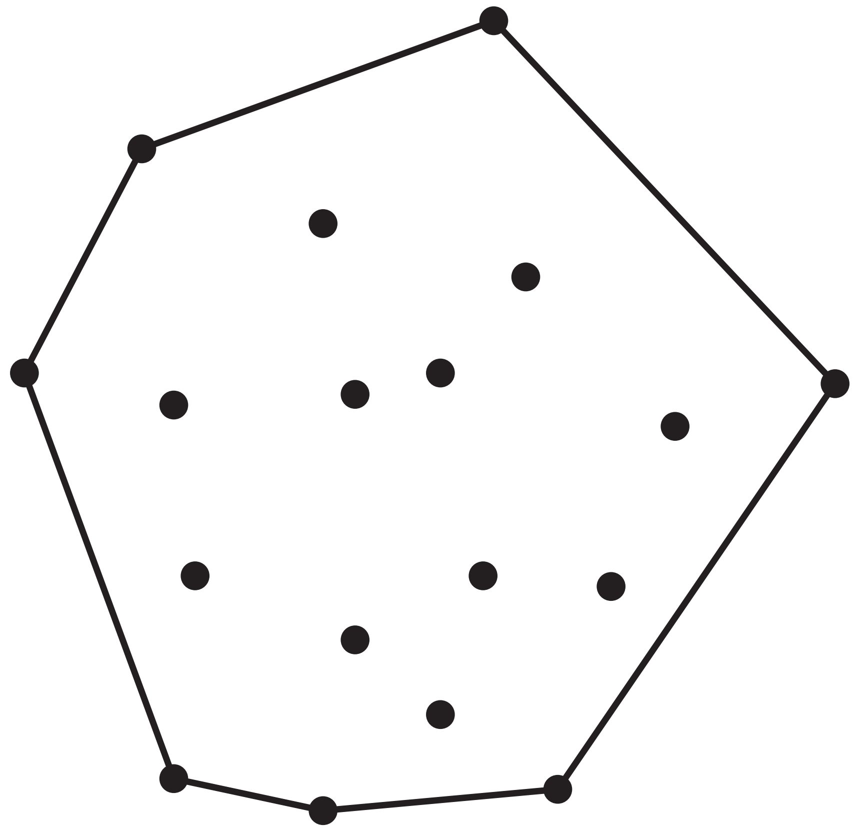
Division: $O(n)$

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 $p_i = (x_i, x_i^2)$

CONVEX HULL IN 2D

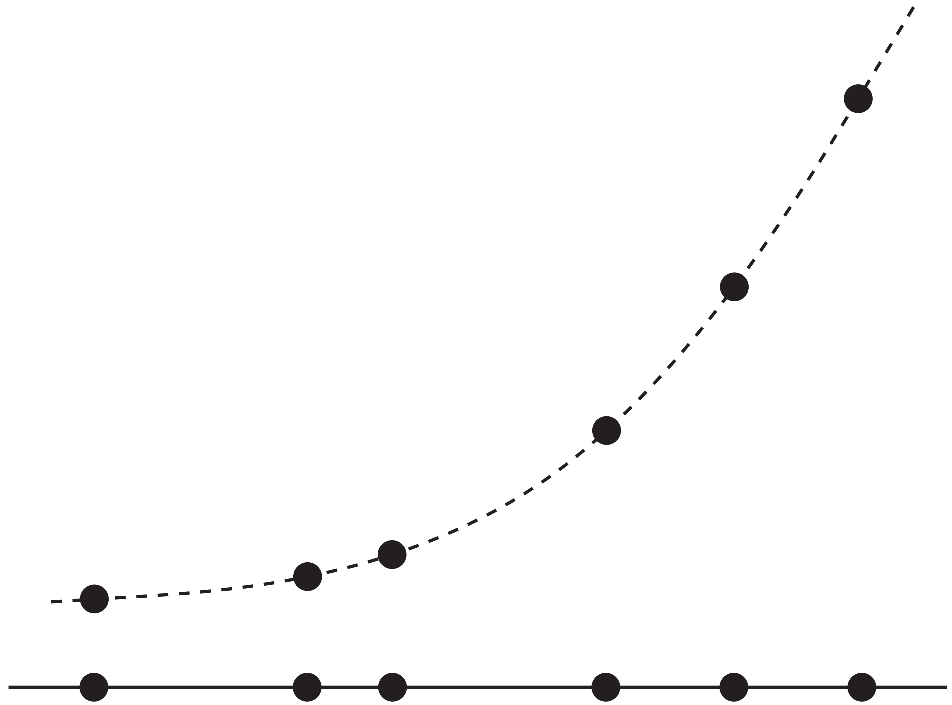
Lower bound

Input: n **real numbers**

$x_1, \ldots, x_n$ real numbers

Input: n **points**

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

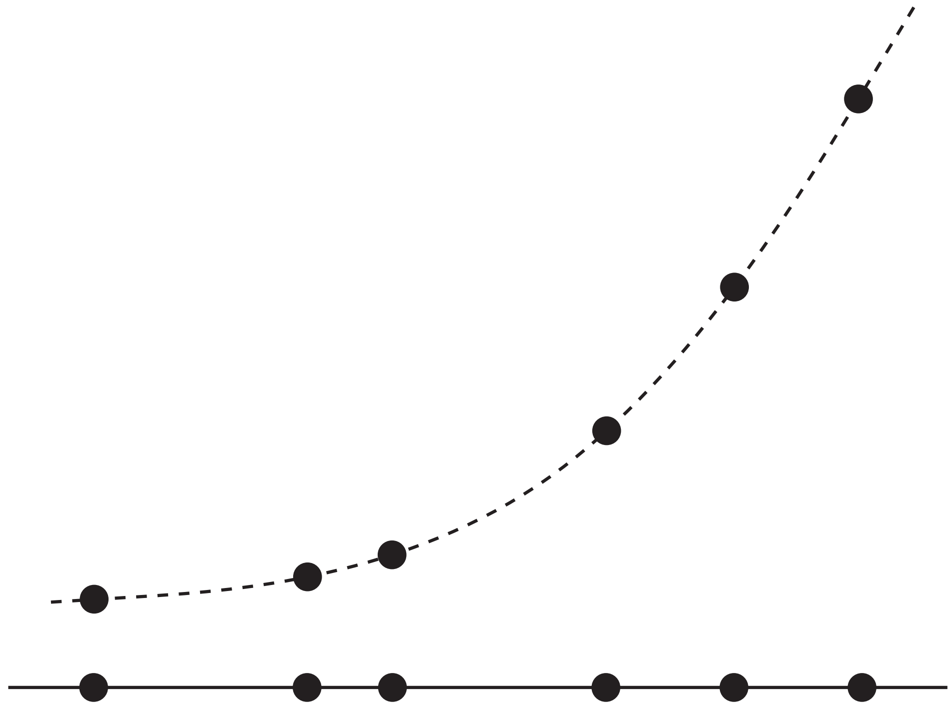


Lower bound

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

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

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



Lower bound

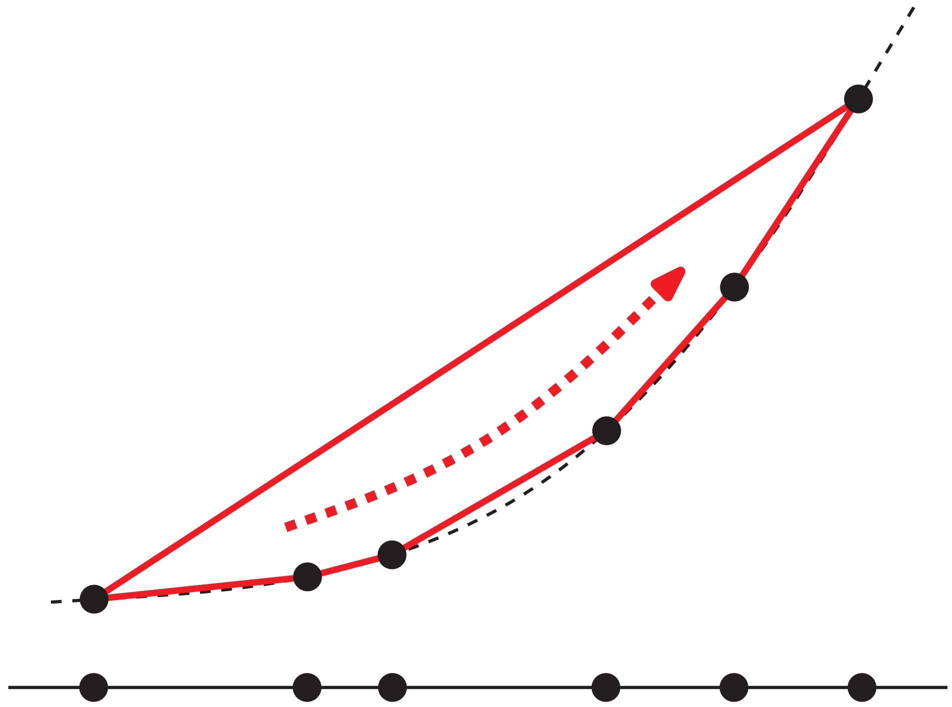
Input: n real numbers
$x_1, \ldots, x_n$ real numbers

↓

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

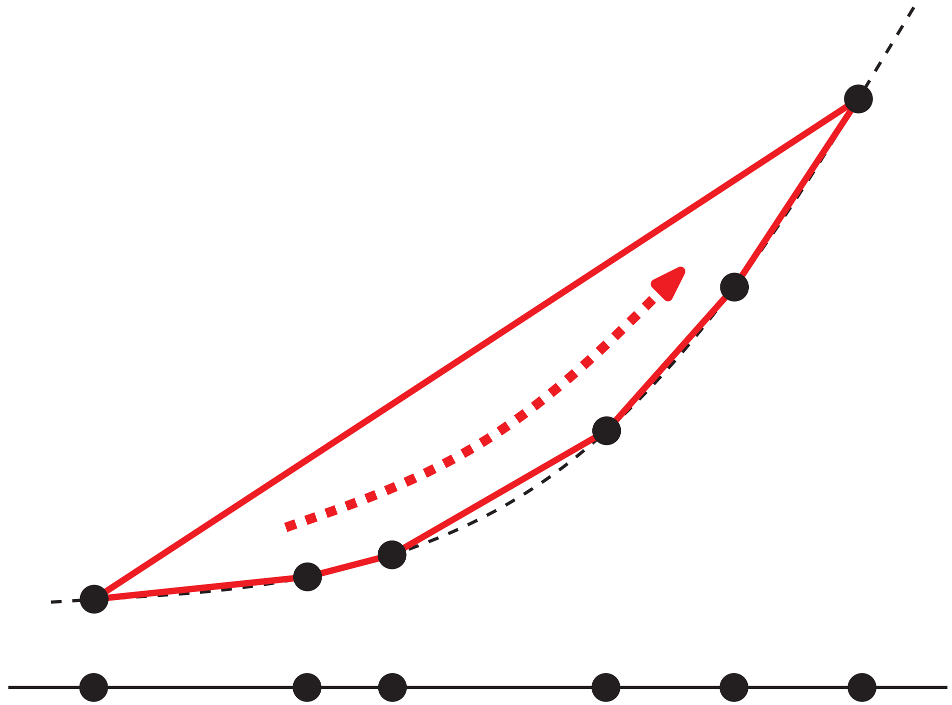
Lower bound

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 $x_1, \ldots, x_n$



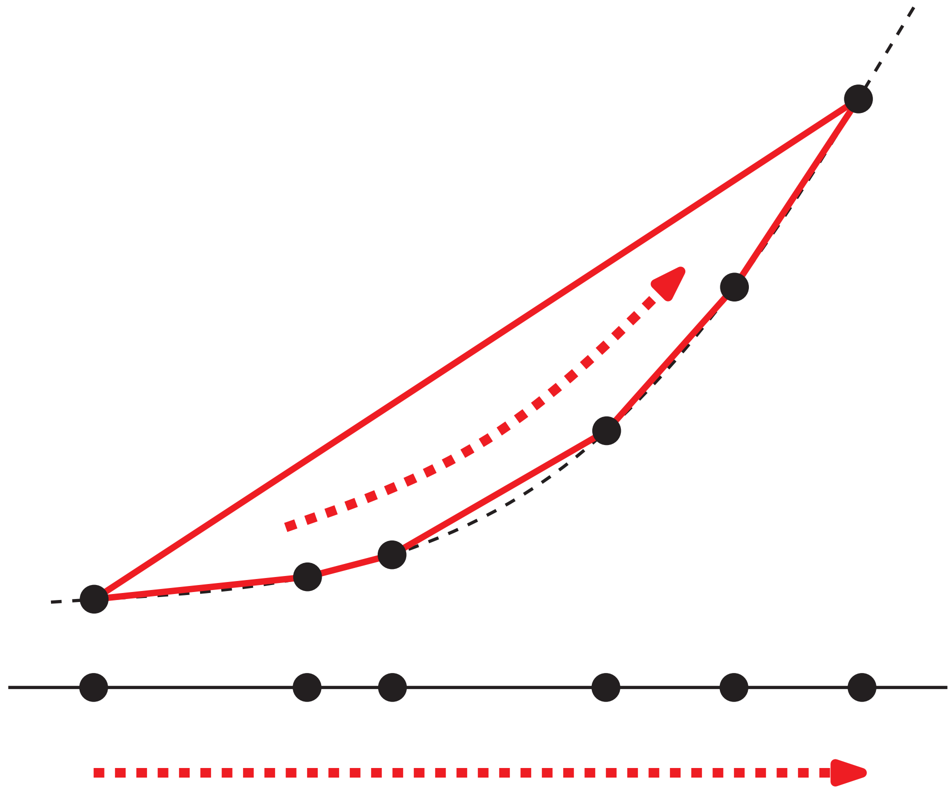
Lower bound

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 $x_1, \ldots, x_n$



Lower bound

Input: n real numbers

$x_1, \ldots, x_n$ real numbers

$O(n)$

Input: n points

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

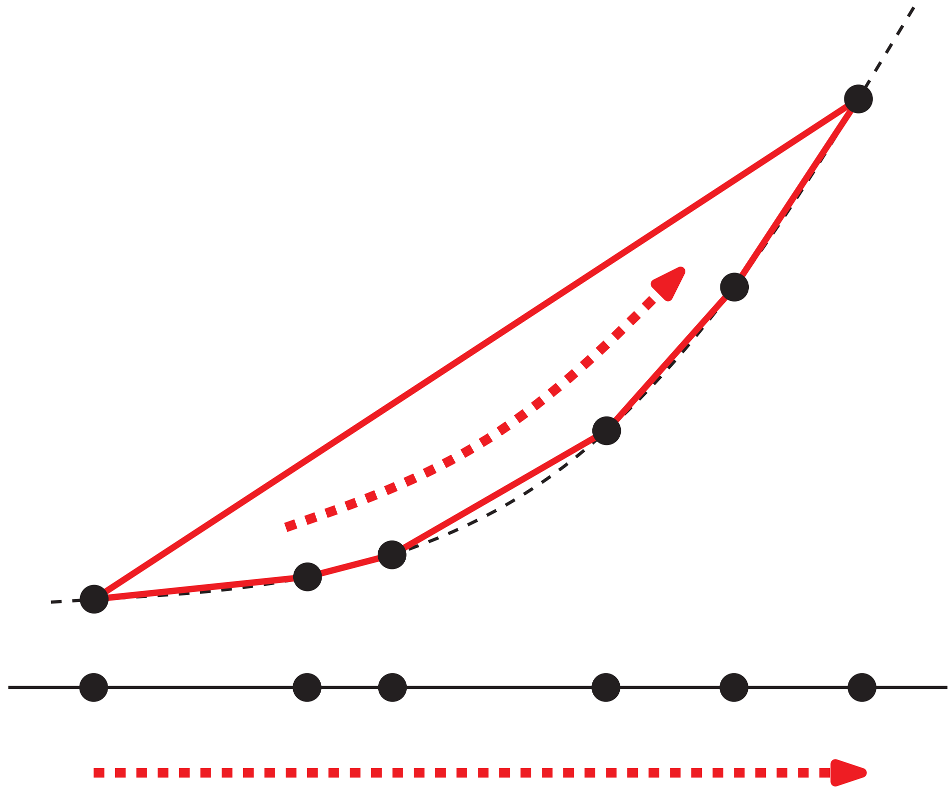
Output: convex hull of the points

Sorted list of the vertices of the convex hull

$O(n)$

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

$O(n)$

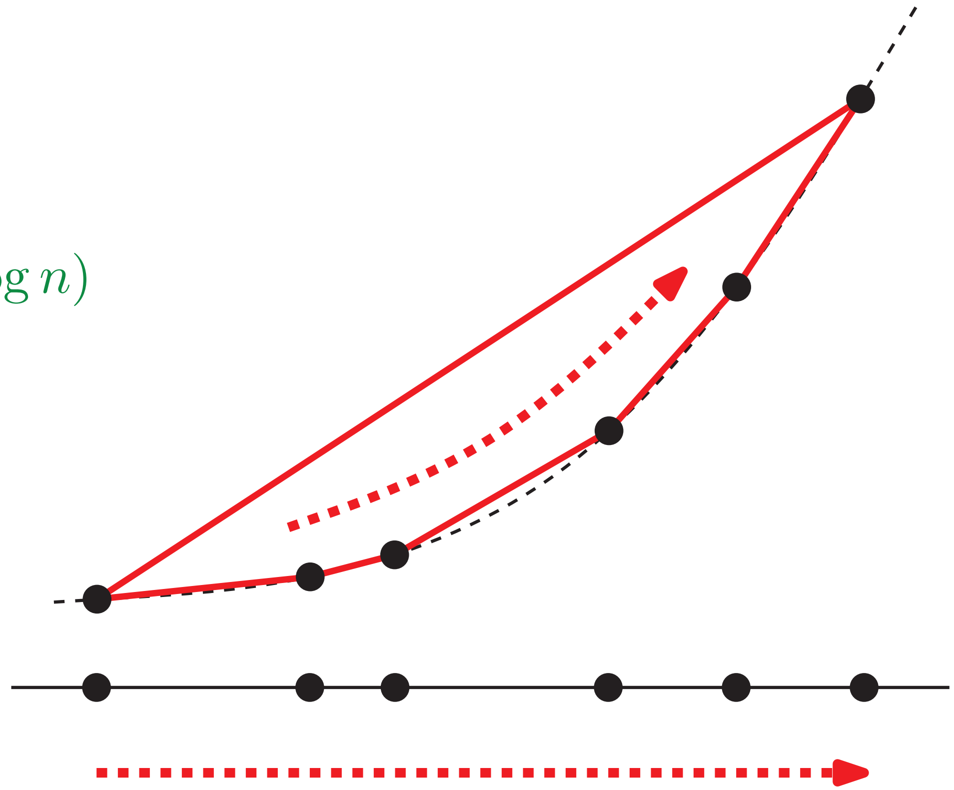
Input: n points
$p_1, \ldots, p_n$, with $p_i = (x_i, x_i^2)$

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

$O(n)$

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

↓ $O(n)$

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

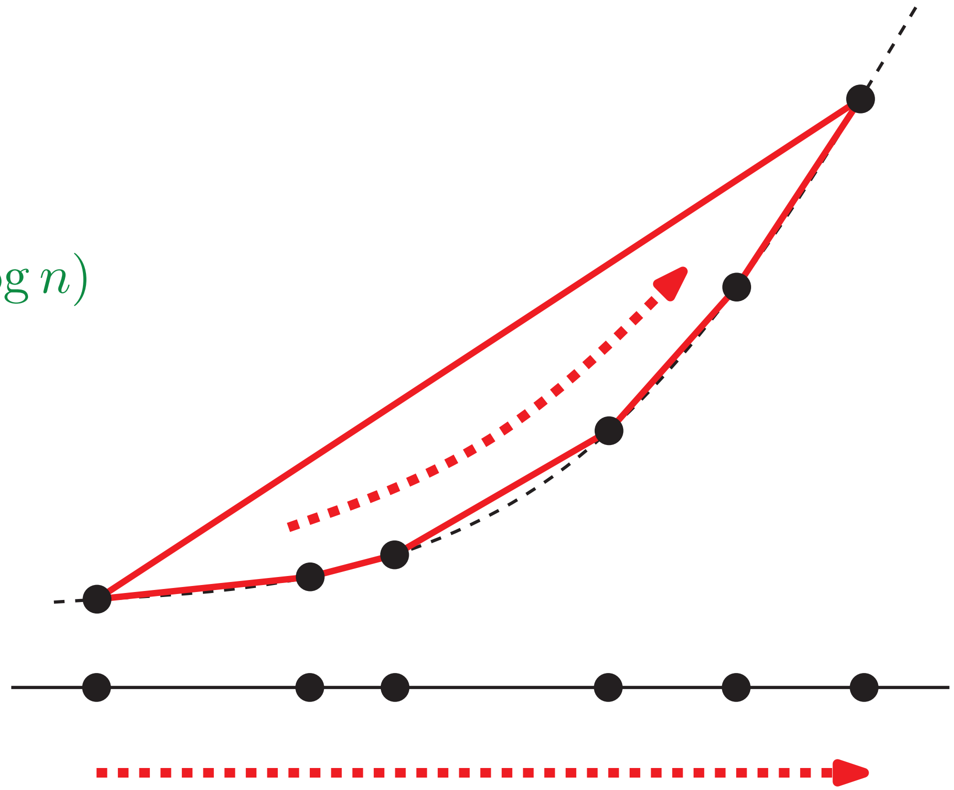
↓ $\Omega(n \log n)$

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 $x_1, \ldots, x_n$

(Overall, from the n real numbers to sorting the numbers: $\Omega(n \log n)$)



CONVEX HULL IN 2D

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