

# CONVEX HULLS

**Vera Sacristán**

Computational Geometry  
Facultat d'Informàtica de Barcelona  
Universitat Politècnica de Catalunya

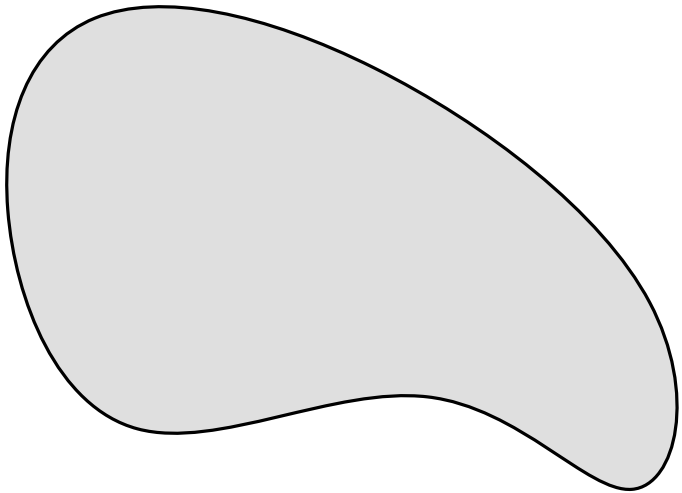
# CONVEX HULL

# CONVEX HULL

A set  $C$  is said to be **convex** if  $\forall p, q \in C$  the segment  $\overline{pq}$  is enclosed in  $C$ .

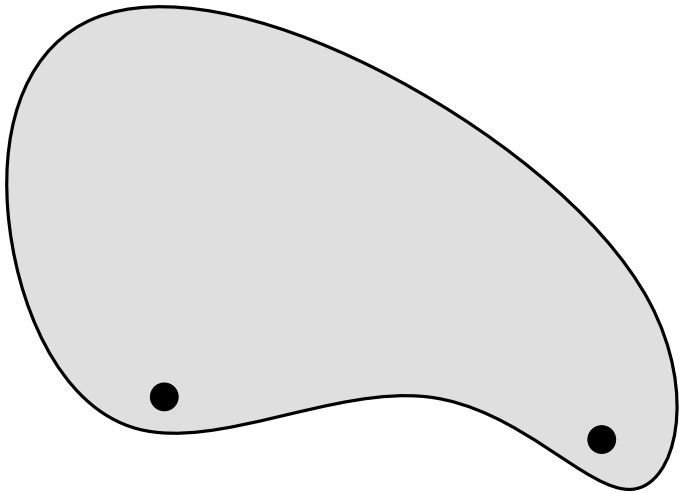
# CONVEX HULL

A set  $C$  is said to be **convex** if  $\forall p, q \in C$  the segment  $\overline{pq}$  is enclosed in  $C$ .



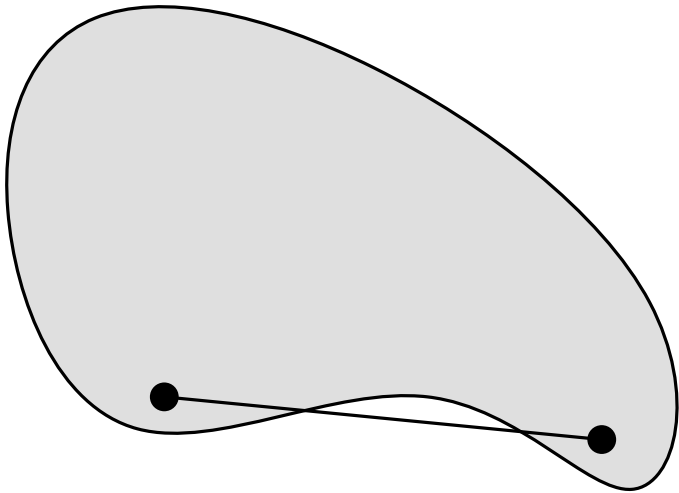
# CONVEX HULL

A set  $C$  is said to be **convex** if  $\forall p, q \in C$  the segment  $\overline{pq}$  is enclosed in  $C$ .



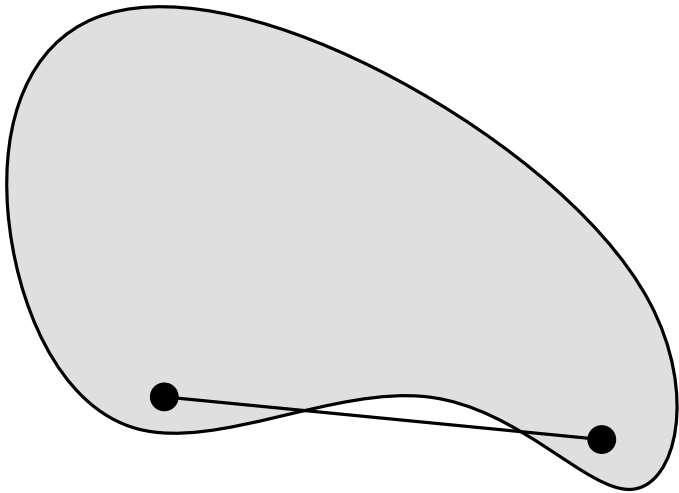
# CONVEX HULL

A set  $C$  is said to be **convex** if  $\forall p, q \in C$  the segment  $\overline{pq}$  is enclosed in  $C$ .



# CONVEX HULL

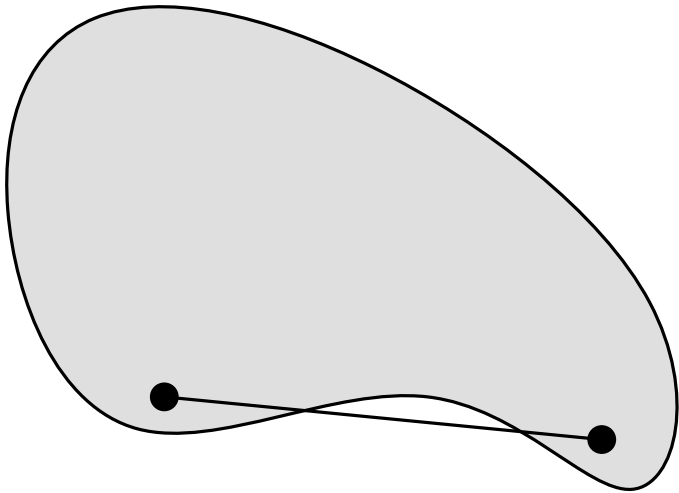
A set  $C$  is said to be **convex** if  $\forall p, q \in C$  the segment  $\overline{pq}$  is enclosed in  $C$ .



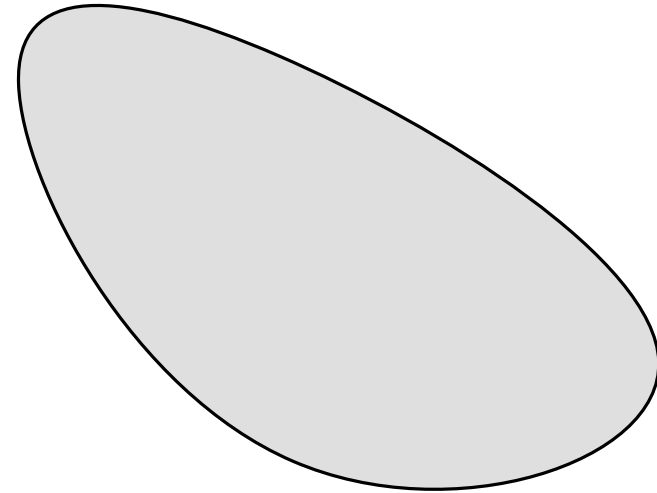
not convex

# CONVEX HULL

A set  $C$  is said to be **convex** if  $\forall p, q \in C$  the segment  $\overline{pq}$  is enclosed in  $C$ .



not convex



convex

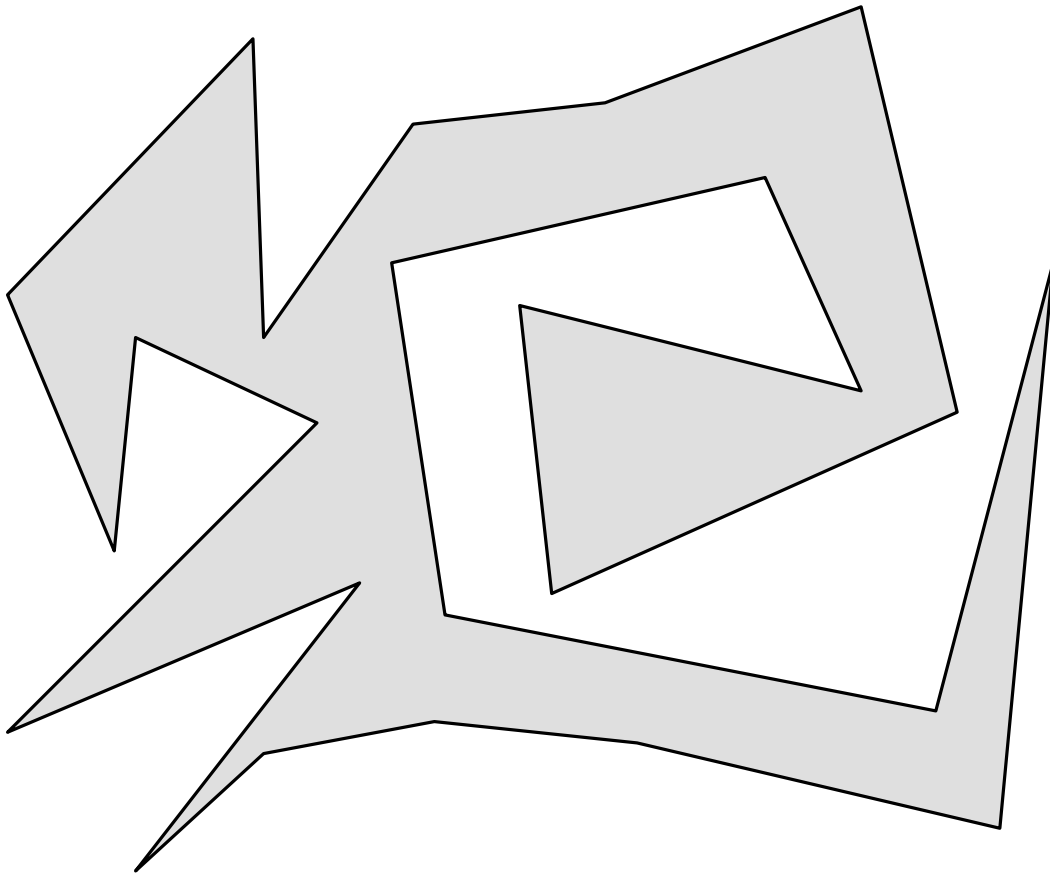


# CONVEX HULL

The **convex hull** of a set  $X$  is the smallest convex set  $C$  enclosing  $X$ .

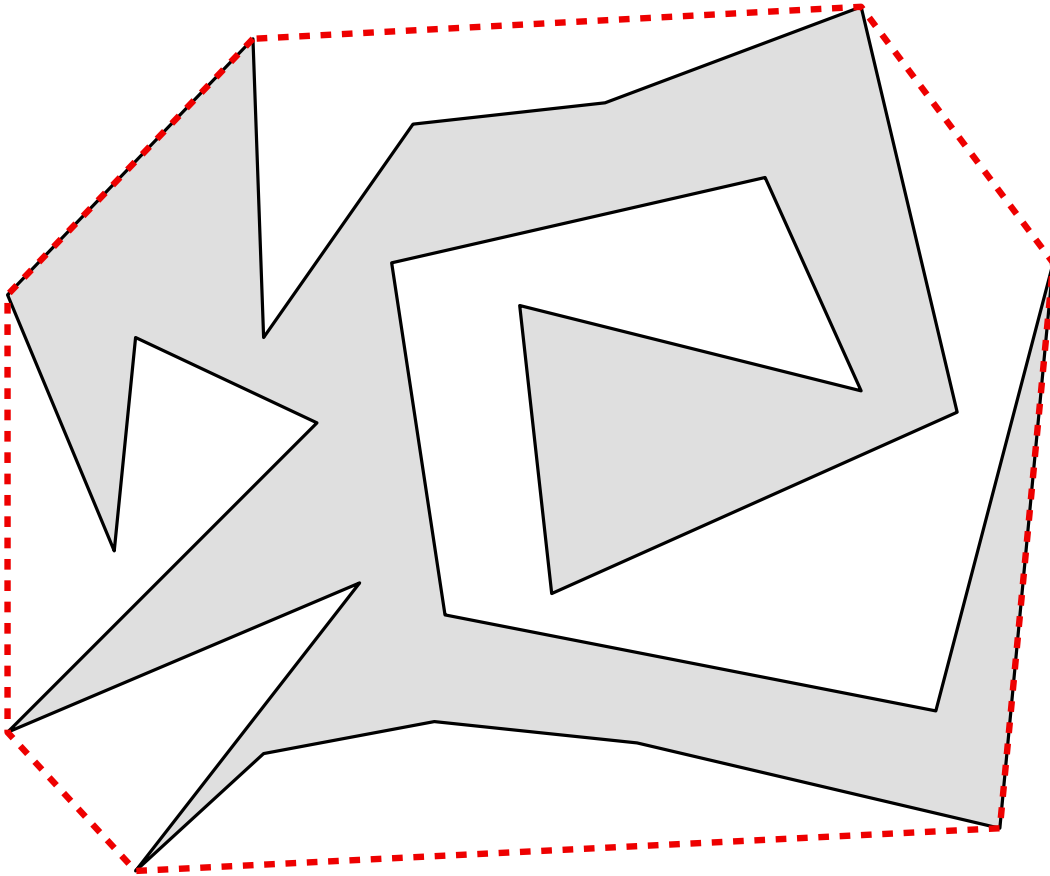
# CONVEX HULL

The **convex hull** of a set  $X$  is the smallest convex set  $C$  enclosing  $X$ .



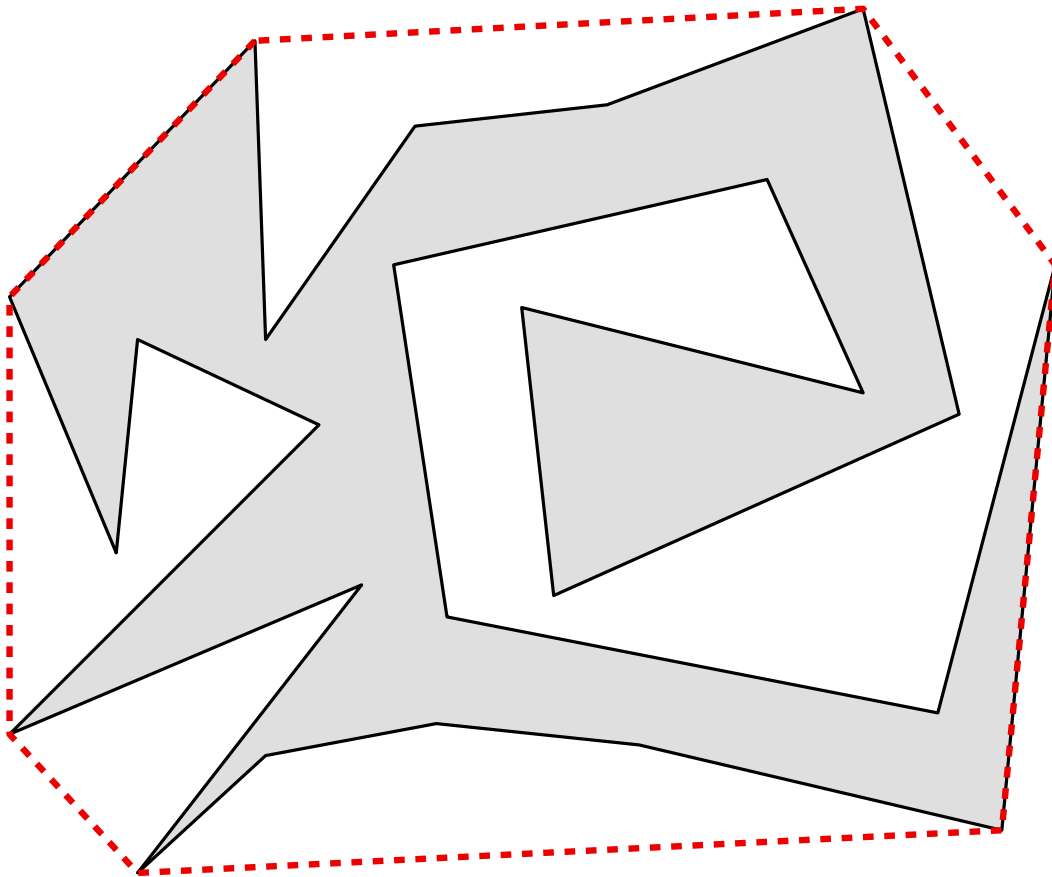
# CONVEX HULL

The **convex hull** of a set  $X$  is the smallest convex set  $C$  enclosing  $X$ .



# CONVEX HULL

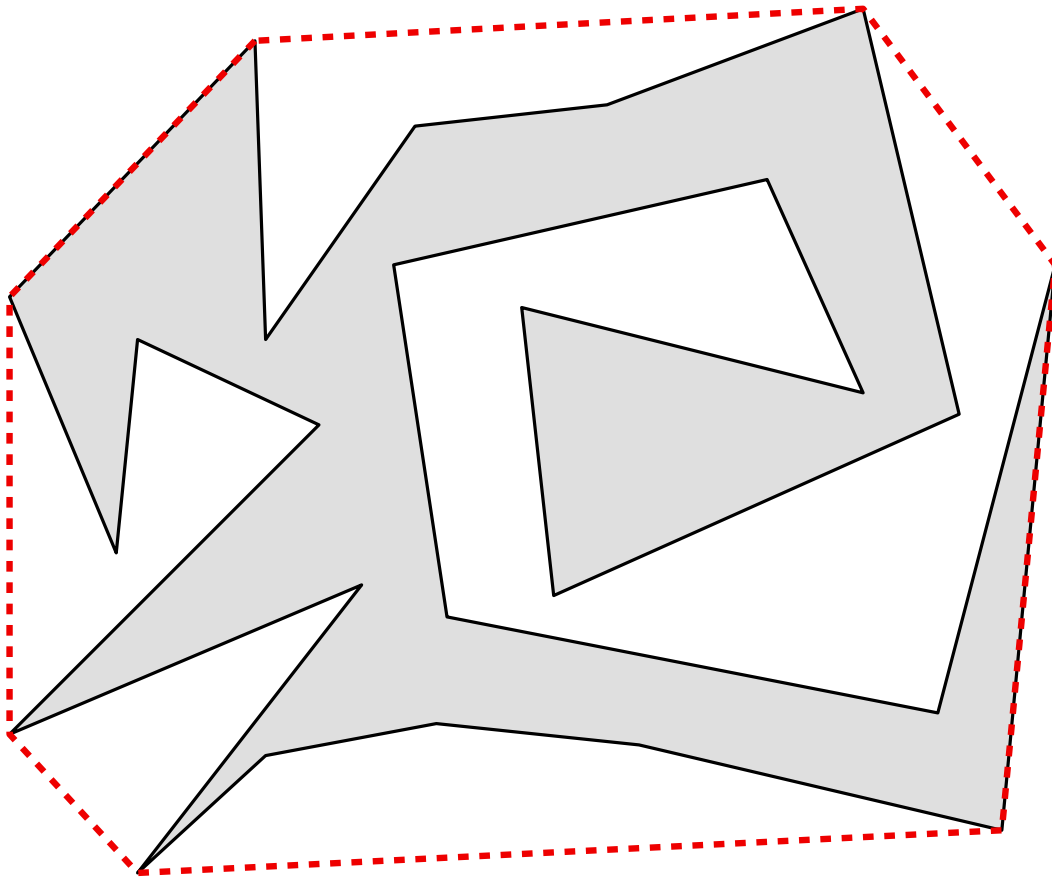
The **convex hull** of a set  $X$  is the smallest convex set  $C$  enclosing  $X$ .



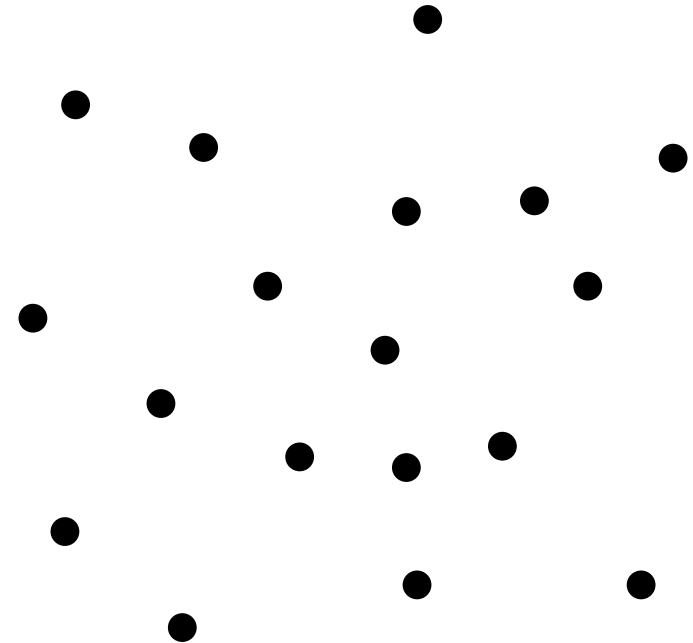
The convex hull of a simple polygon is a convex polygon.

# CONVEX HULL

The **convex hull** of a set  $X$  is the smallest convex set  $C$  enclosing  $X$ .

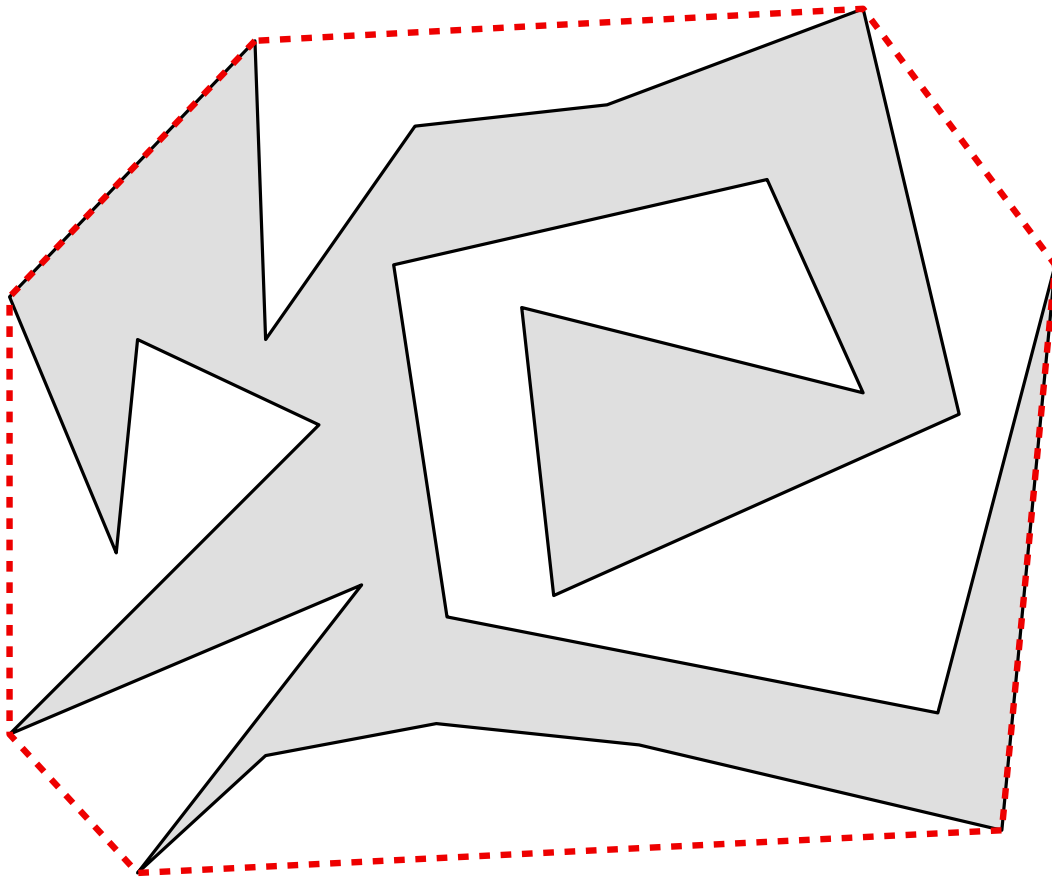


The convex hull of a simple polygon is a convex polygon.

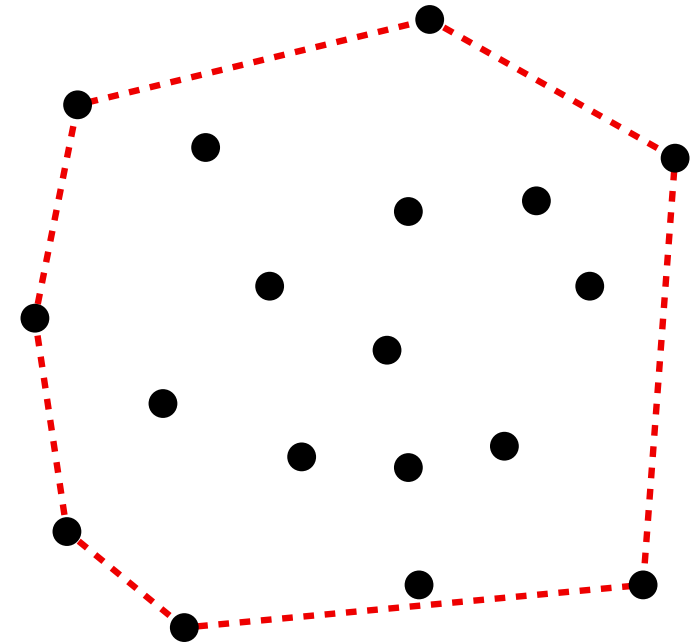


# CONVEX HULL

The **convex hull** of a set  $X$  is the smallest convex set  $C$  enclosing  $X$ .

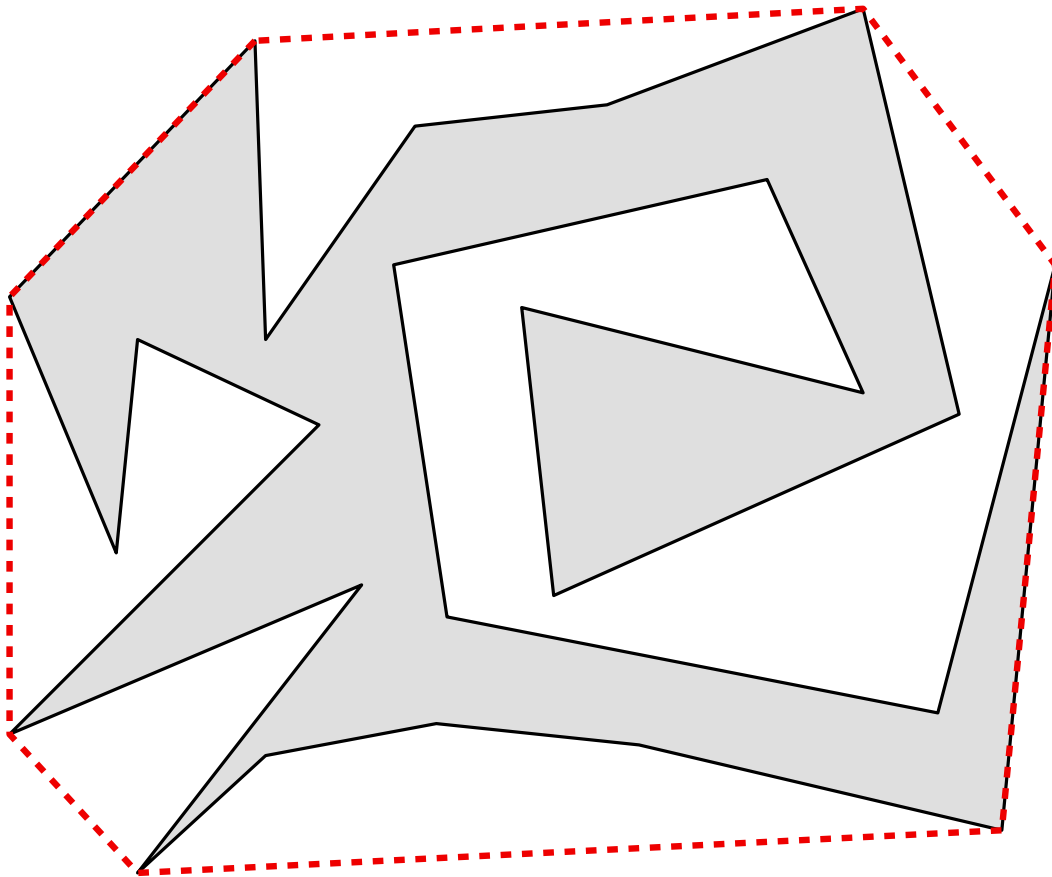


The convex hull of a simple polygon is a convex polygon.

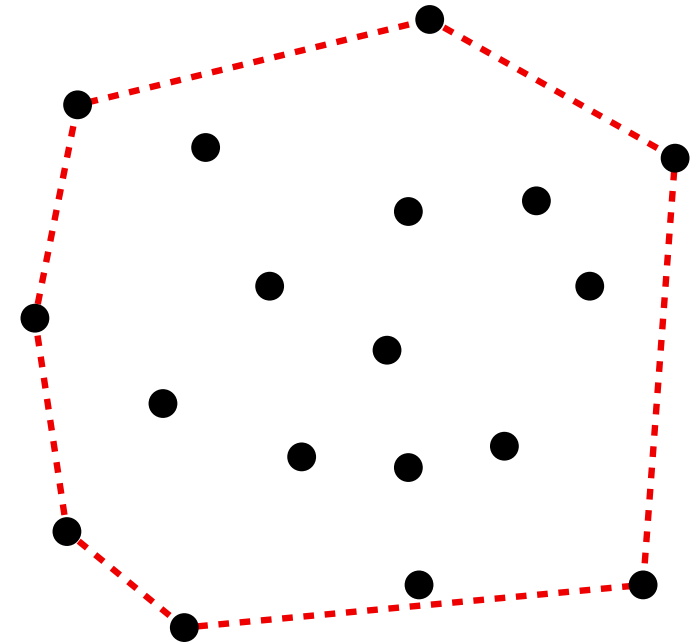


# CONVEX HULL

The **convex hull** of a set  $X$  is the smallest convex set  $C$  enclosing  $X$ .



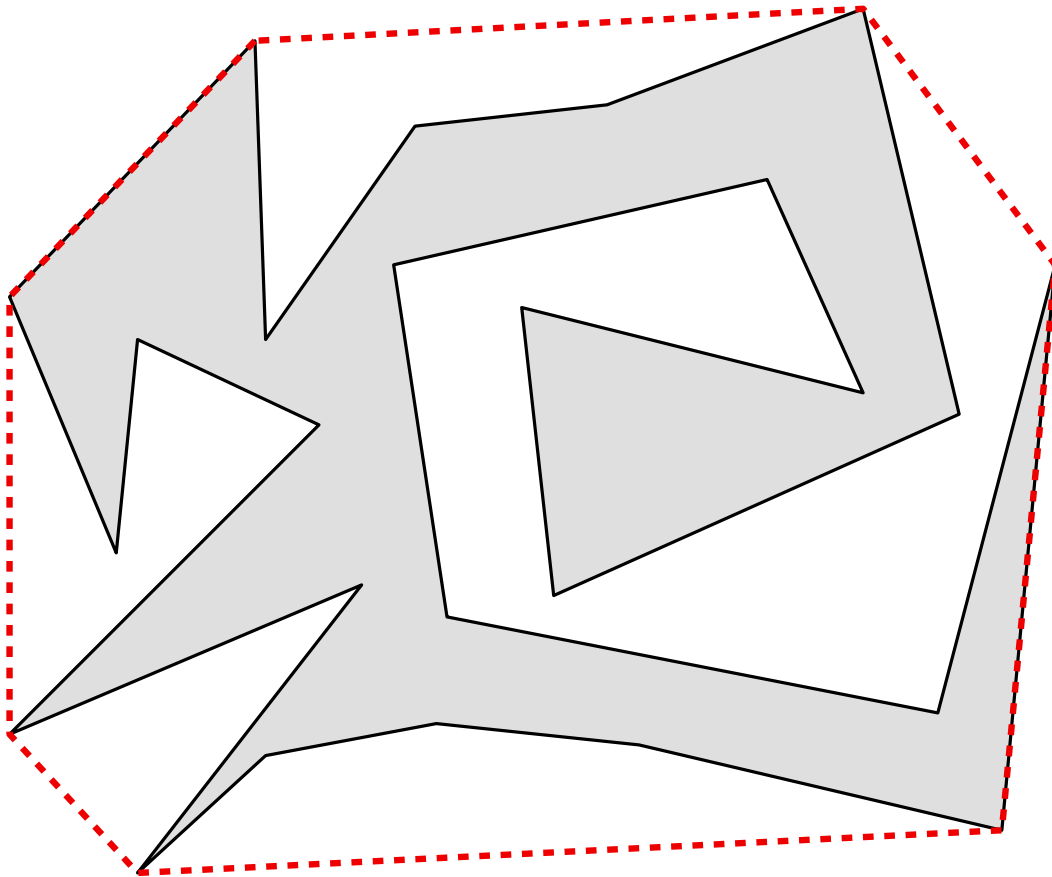
The convex hull of a simple polygon is a convex polygon.



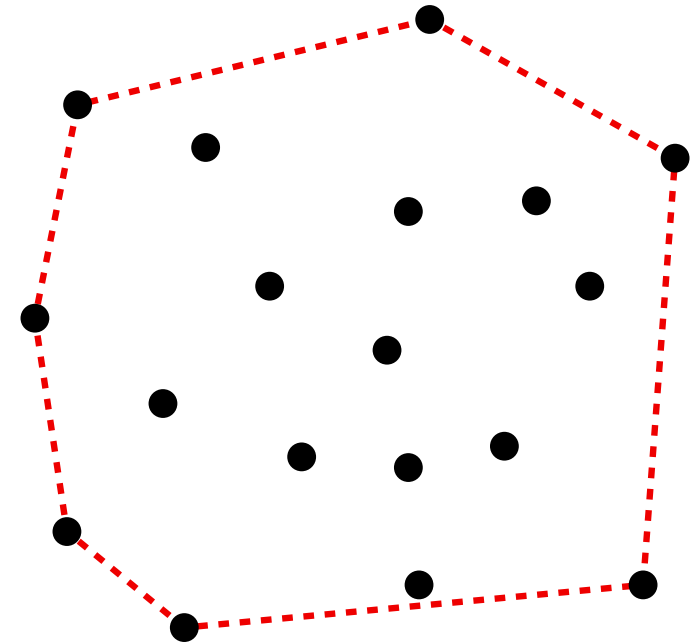
The convex hull of a finite set of points in the plane is a convex polygon.

# CONVEX HULL

The **convex hull** of a set  $X$  is the smallest convex set  $C$  enclosing  $X$ .



The convex hull of a simple polygon is a convex polygon.



The convex hull of a finite set of points in the plane is a convex polygon.

In both cases, the vertices of  $ch(X)$  are points of  $X$ .



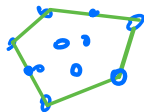
# CONVEX HULL

## Computing the extreme points

Deseamos dar un algoritmo para calcular el cierre convexo de un conjunto  $S$  de  $n$  puntos en el plano.

Podríamos dar como salida:

1. Todos los puntos de la frontera  $ch(S)$ , en orden arbitrario.



2. los puntos extremos, es decir los vértices del cierre convexo, en orden arbitrario

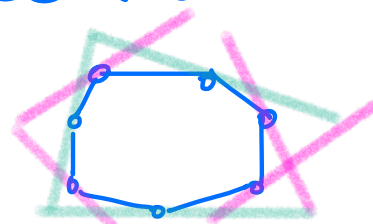


3. Todos los puntos de  $\partial ch(S)$  en el orden de recorrido

4. los puntos extremos, es decir los vértices del cierre convexo, en orden el orden de recorrido.



Puntos extremos: más alto, más bajo, más a la izq, más a la derecha...  
i.e. si existe una recta que pasa por este y únicamente intersecta a  $ch(S)$  en ese punto.



Si no están en pos  
gral. podrían ser  
los fin. tas.

# CONVEX HULL

## Computing the extreme points

### Characterization

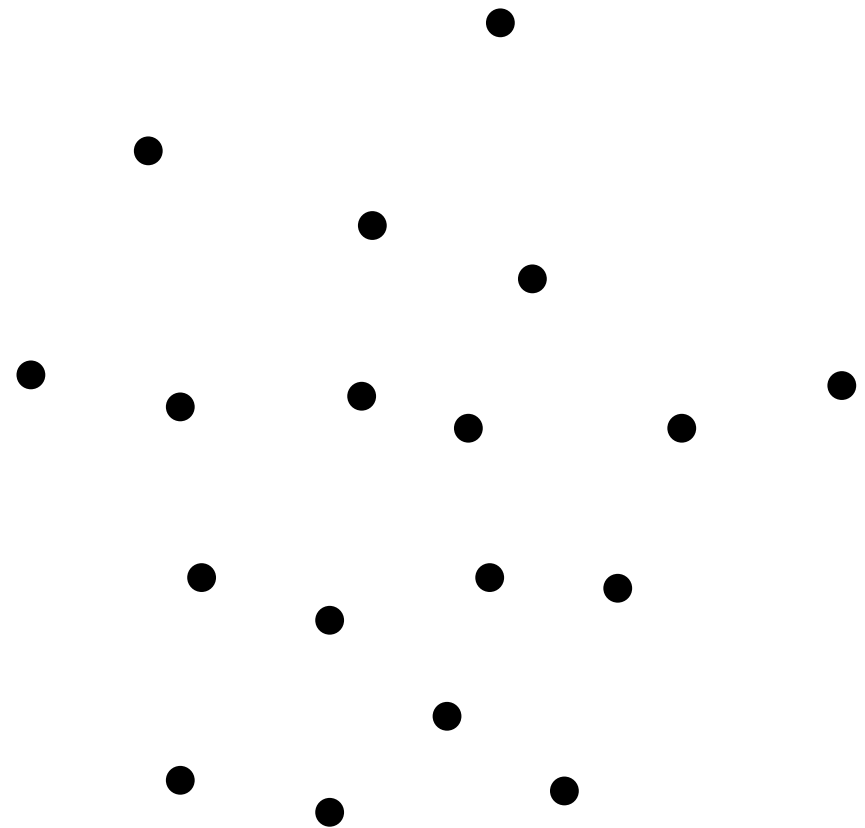
Given  $X = \{p_1, \dots, p_n\}$ , the point  $p_i$  belongs to the boundary of the convex hull of  $X$  if and only if  $p_i$  does not lie in any of the triangles  $p_j p_k p_l$ .

# CONVEX HULL

## 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  $p_j p_k p_l$ .

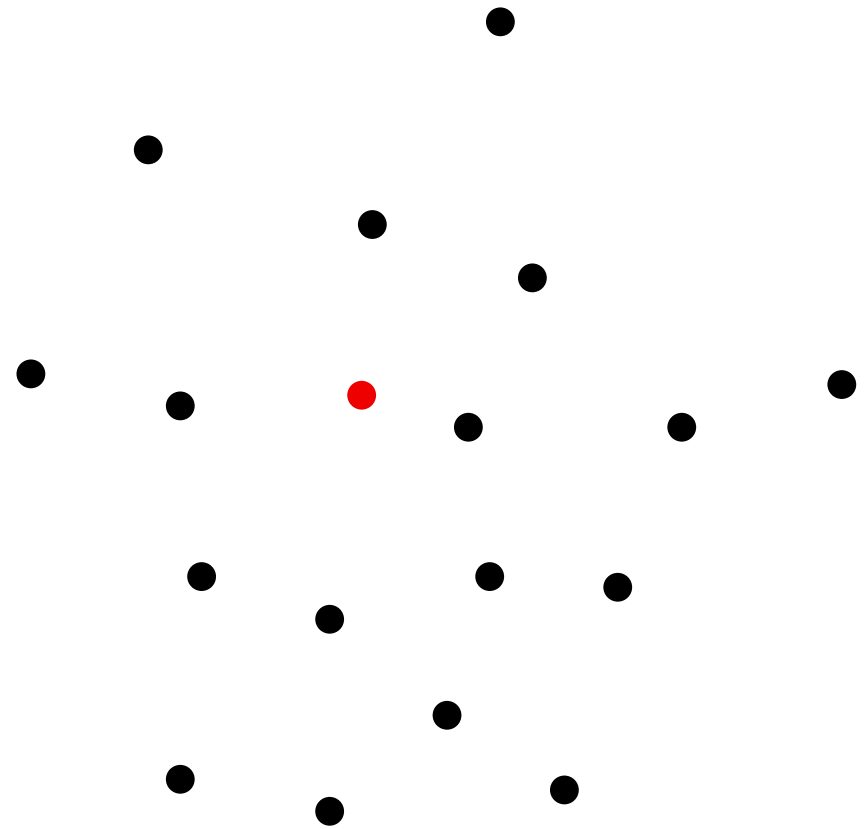


# CONVEX HULL

## 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  $p_j p_k p_l$ .



# CONVEX HULL

## 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  $p_j p_k p_l$ .

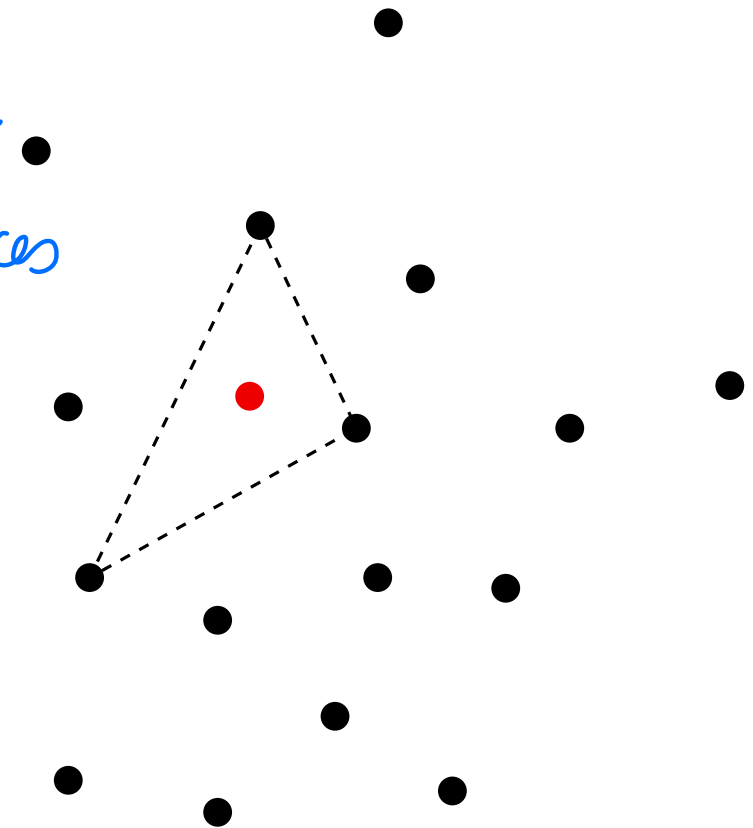
### Demostración.

Sabemos que  $ch(S)$  es la combinación convexa de todos los subconjuntos de  $S$  de tamaño 3, i.e. Es la unión de todas las  $\Delta$ 's con vértices en  $S$ .

Sea  $p \in S$ , tq  $p$  está en  $ch(S)$ , si  $p$  está en el interior de algún triángulo, entonces no es extremo.

Si no existe ningún  $\Delta$  que contenga a  $p$ , entonces  $p$  es extremo.

Es fácil ver.



# CONVEX HULL

## 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:  $p_1, \dots, p_n$  ← todos los puntos distintos.

Output: set of the extreme points

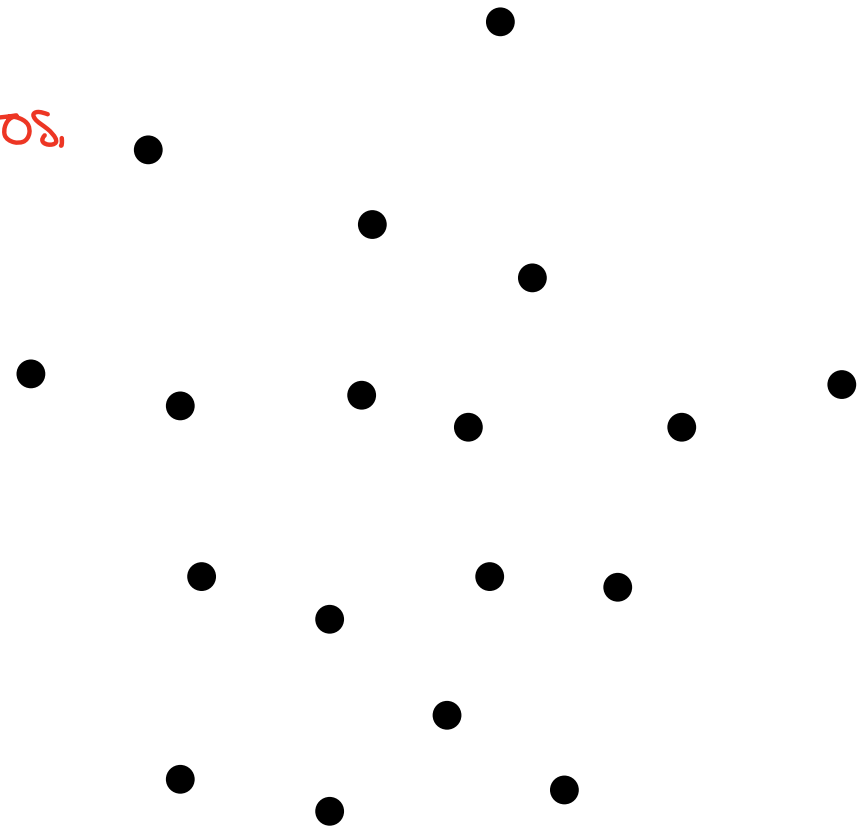
Procedure:

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

## 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.

**Algorithm:** INTERIOR POINTS

for each  $i$  do

for each  $j \neq i$  do

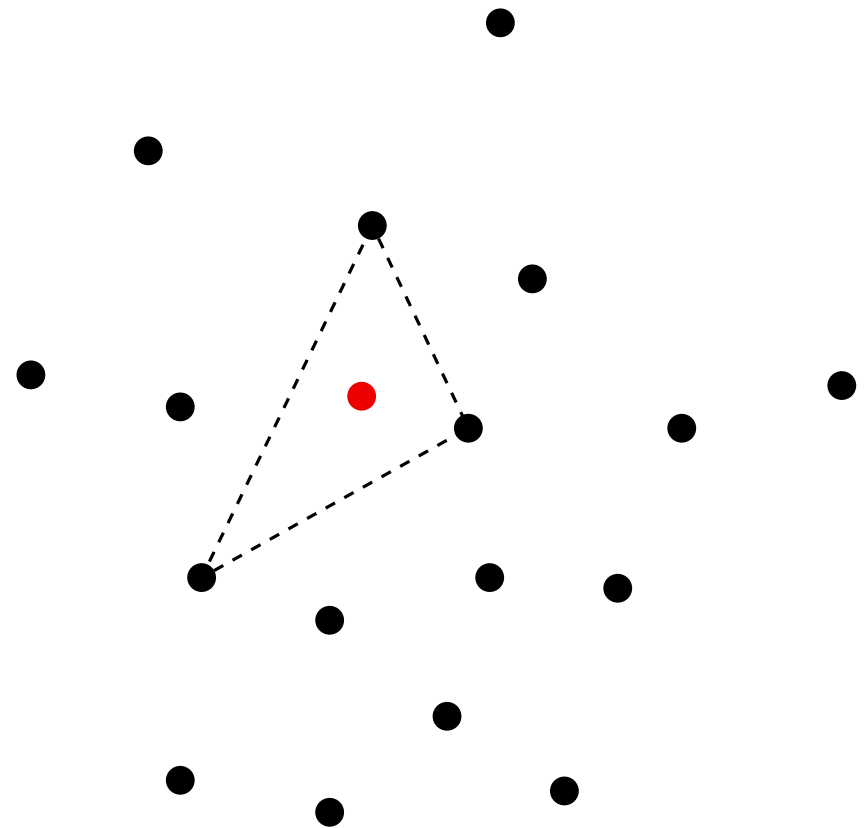
for each  $k \neq i \neq j$  do

for each  $l \neq i \neq j \neq k$  do

if  $p_i \in \Delta(p_j, p_k, p_l)$

then  $p_i$  is nonextreme

O'Rourke



# CONVEX HULL

## 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

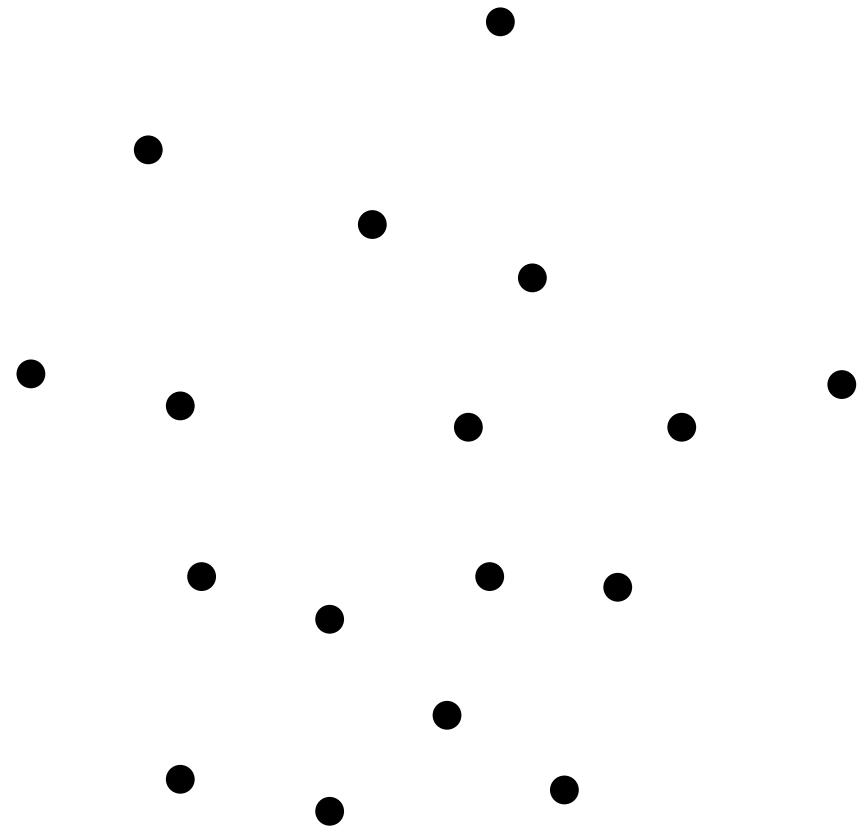
Procedure:

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

## 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

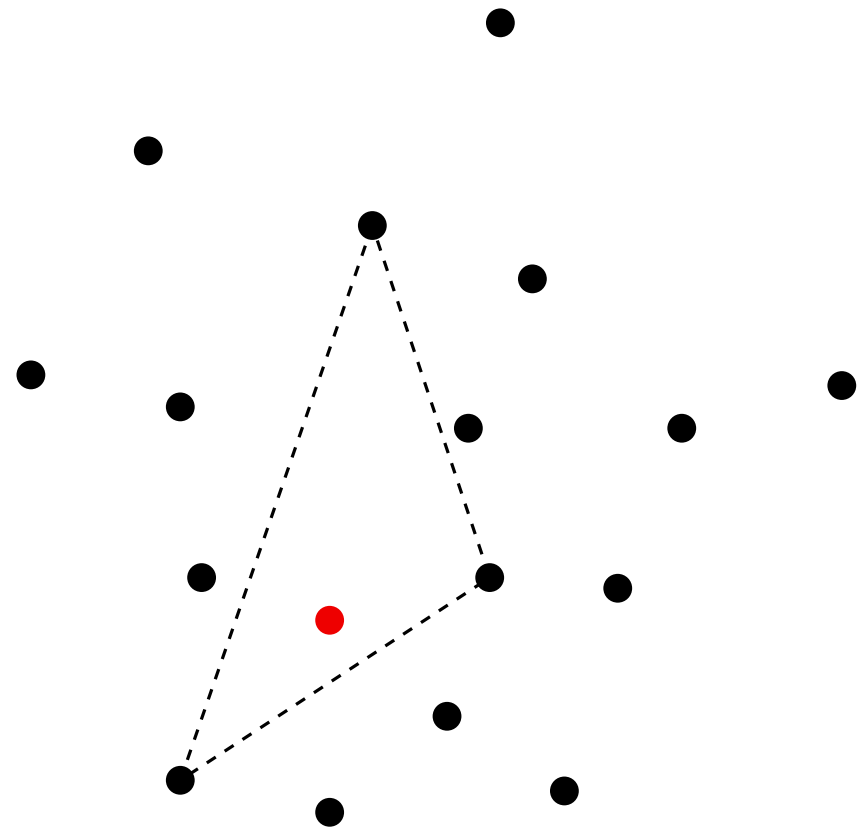
Procedure:

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

## 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

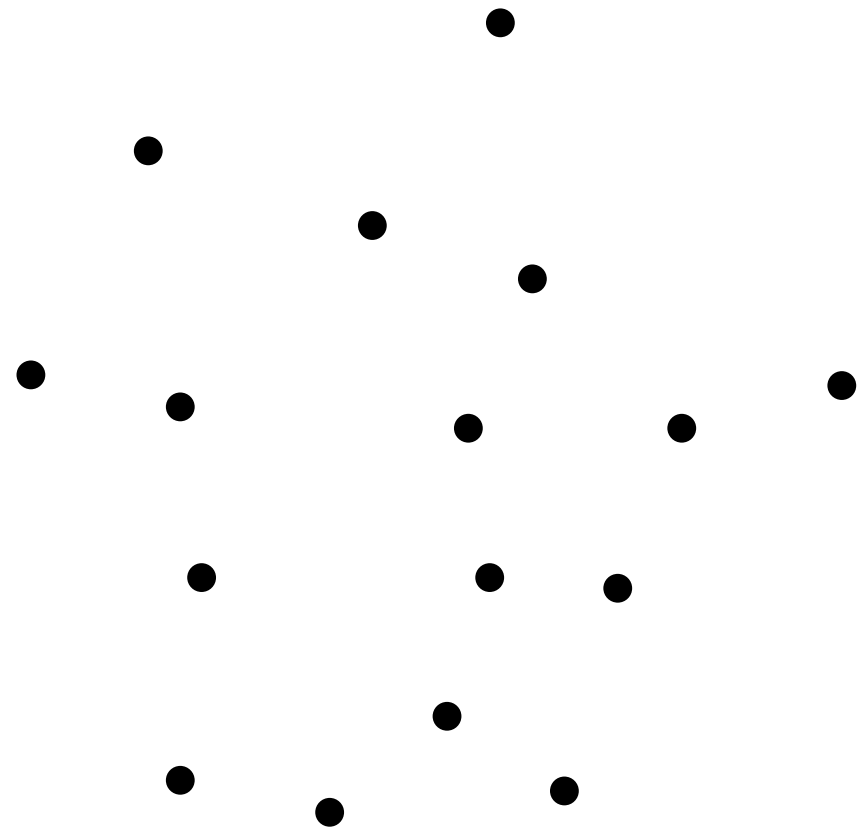
Procedure:

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

## 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

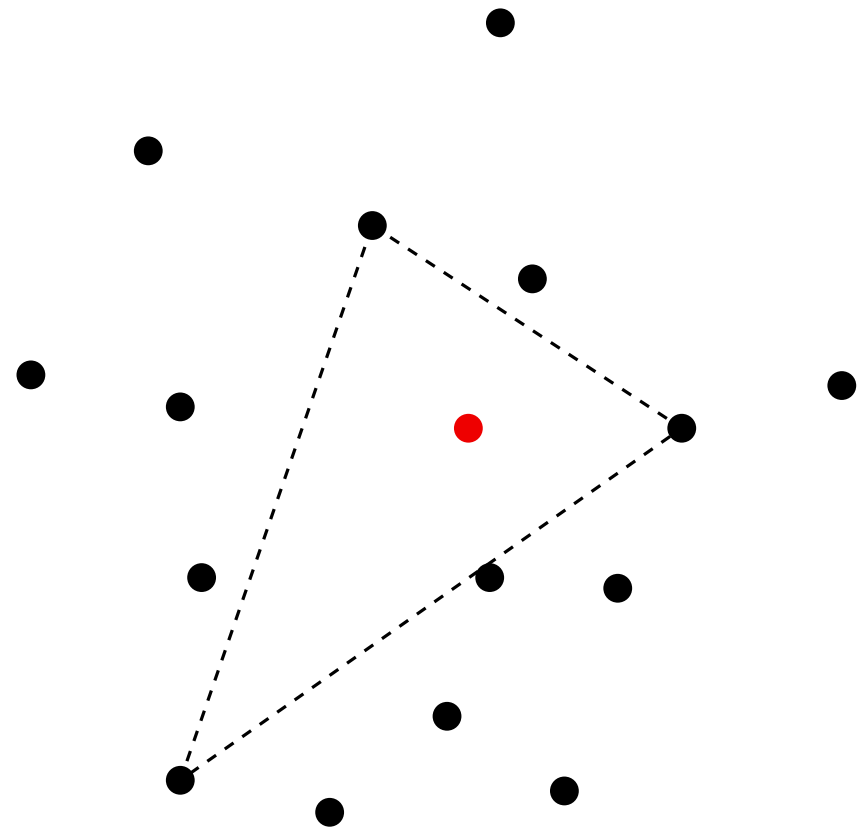
Procedure:

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

## 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

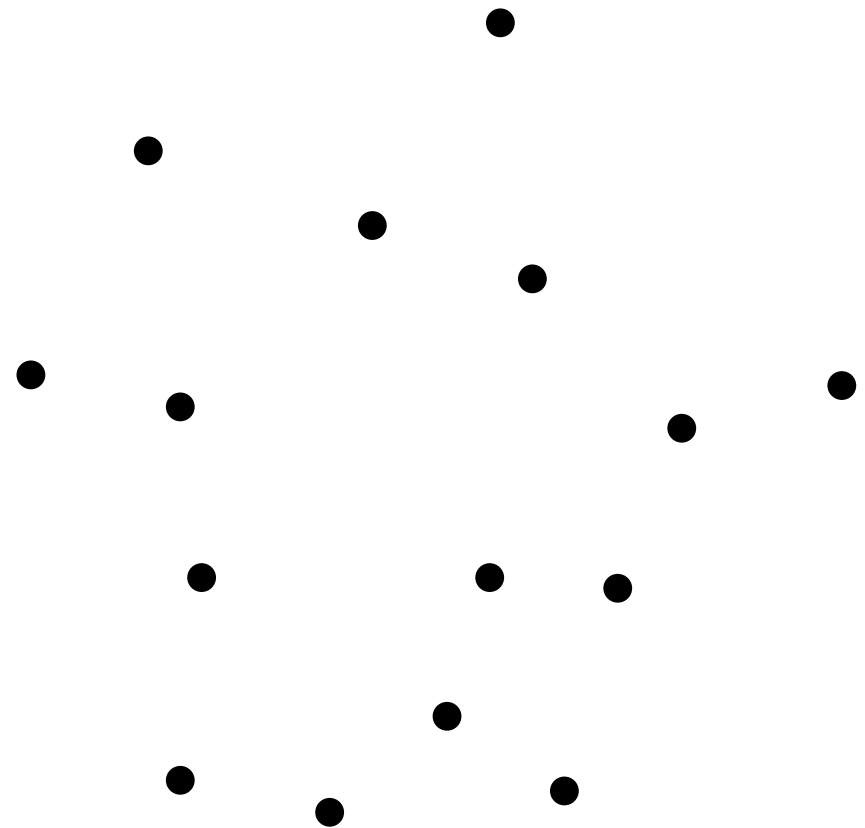
Procedure:

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

## 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

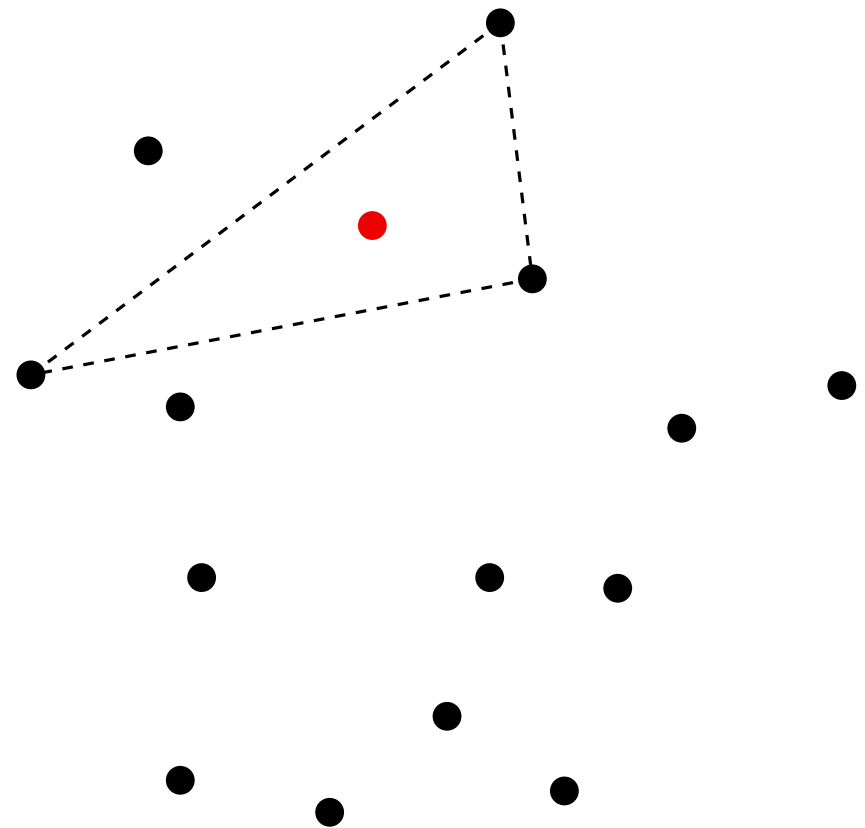
Procedure:

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

## 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

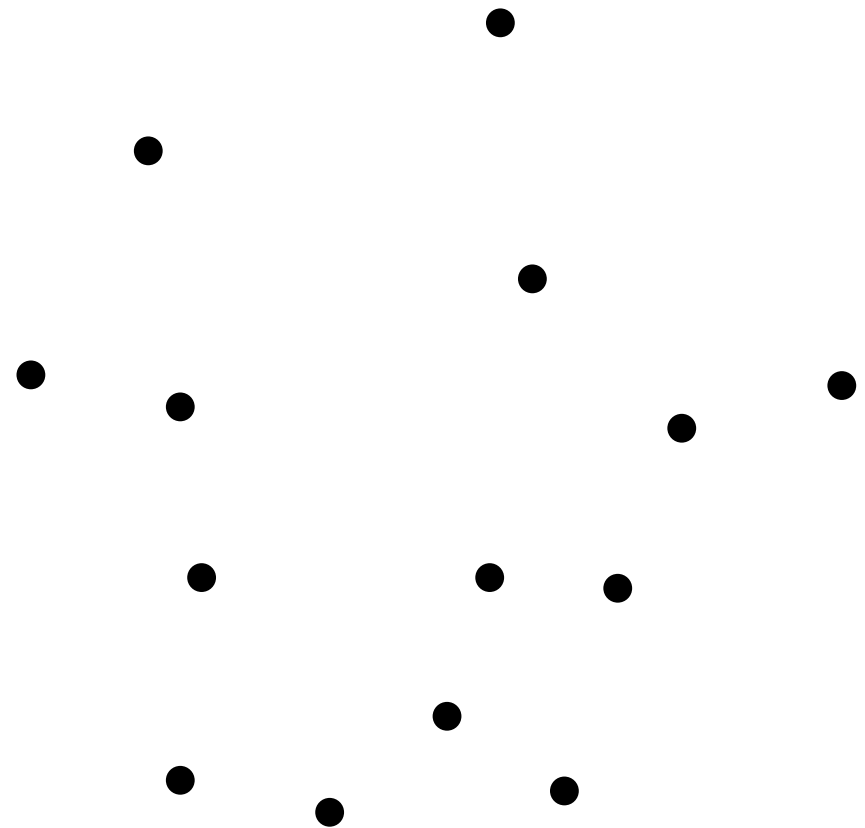
Procedure:

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

## 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

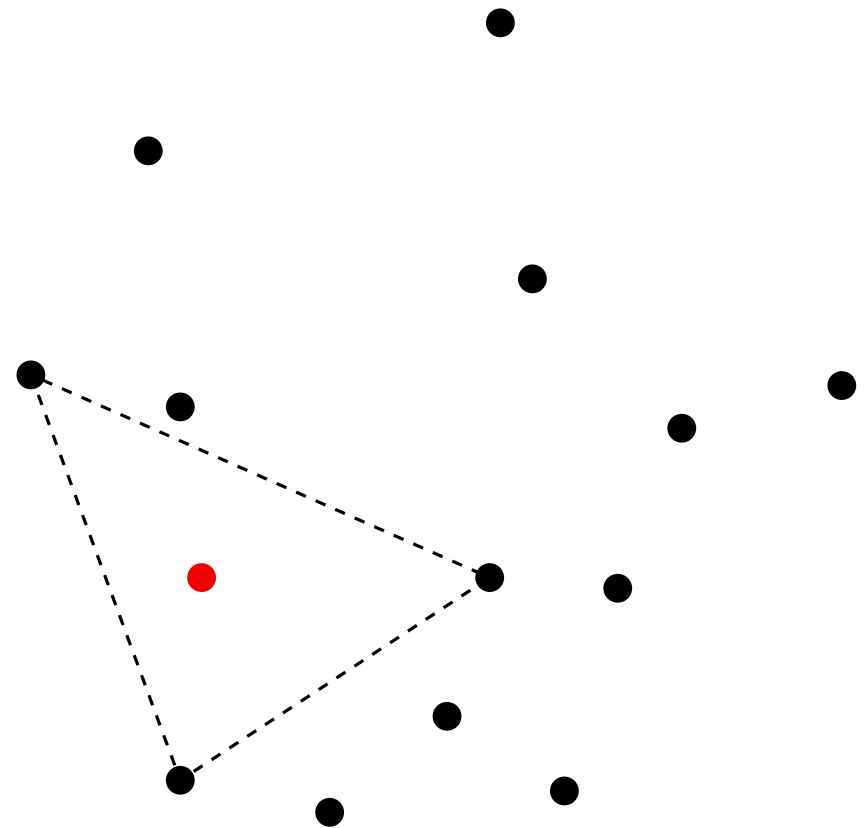
Procedure:

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

## 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

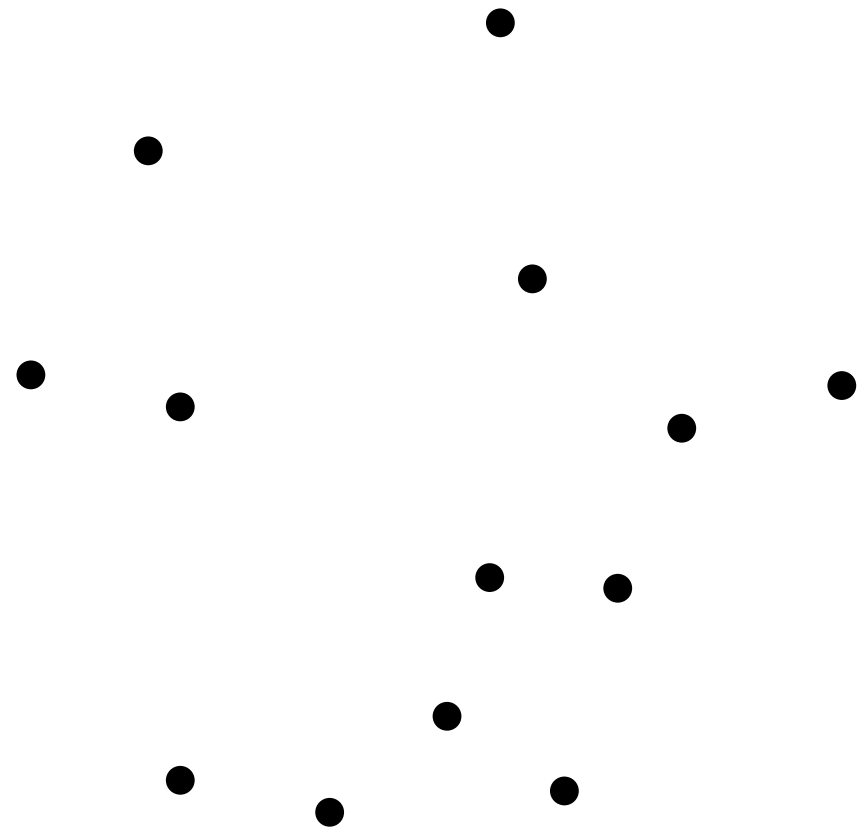
Procedure:

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

## 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

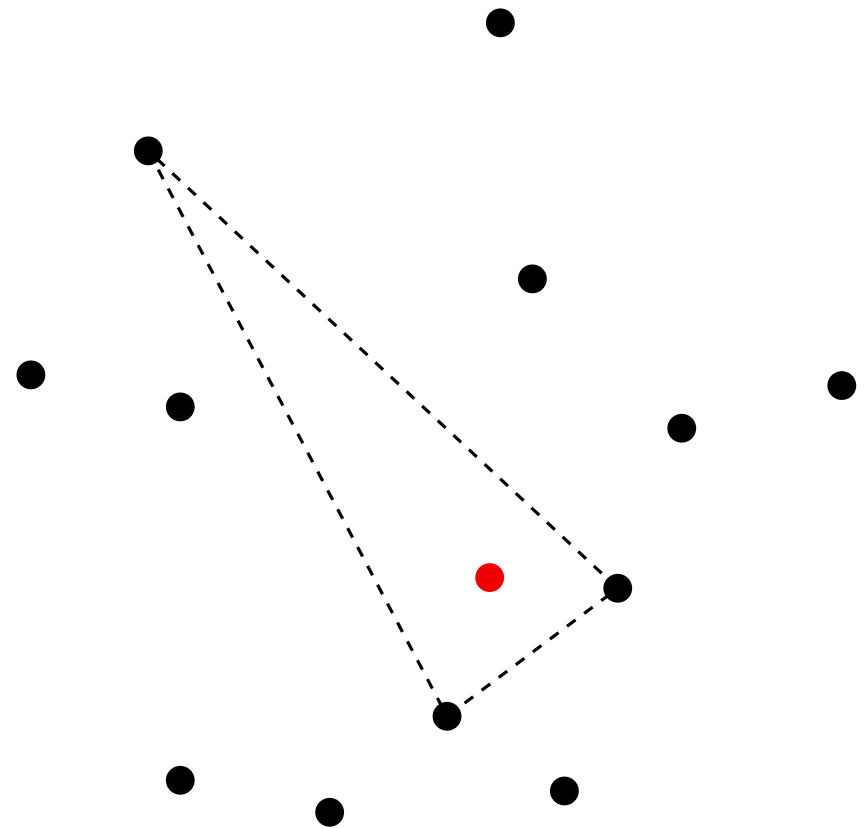
Procedure:

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

## 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

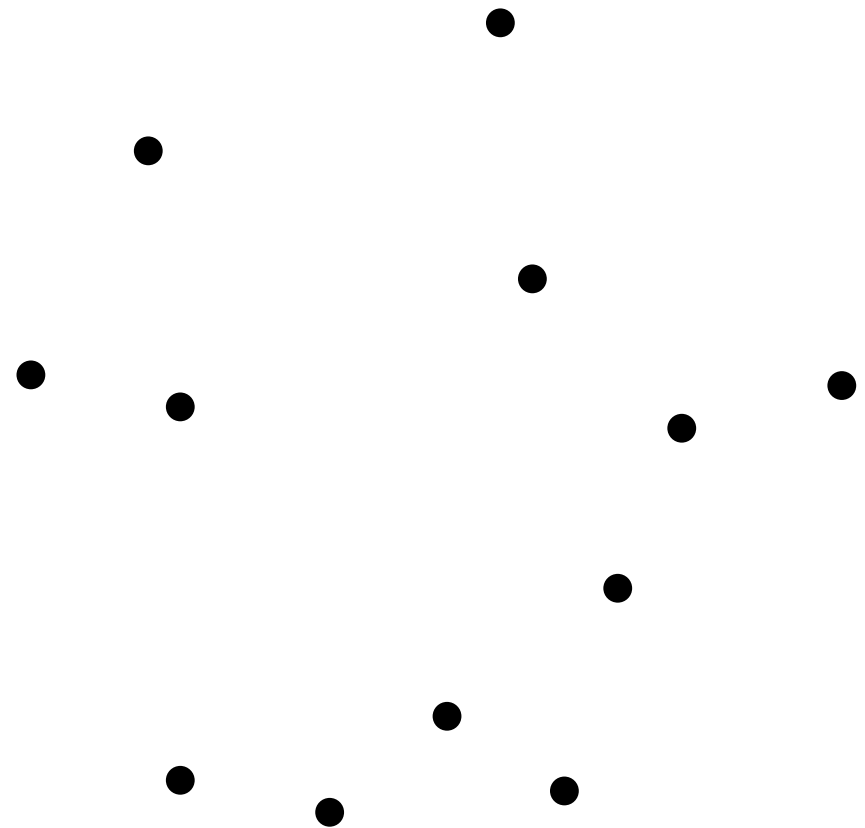
Procedure:

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

## 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

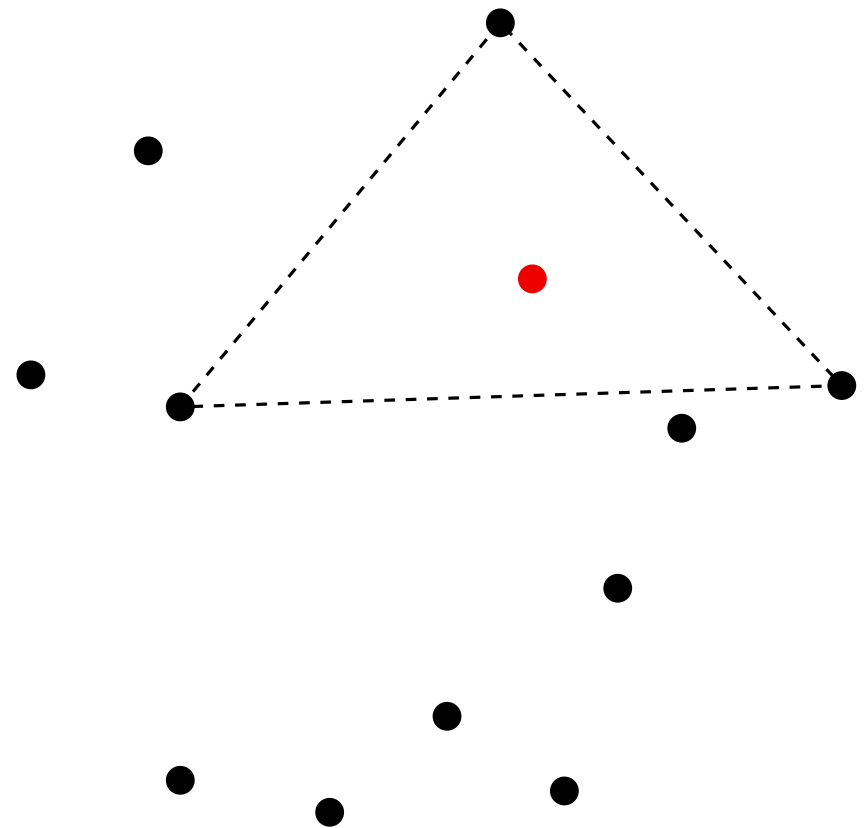
Procedure:

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

## 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

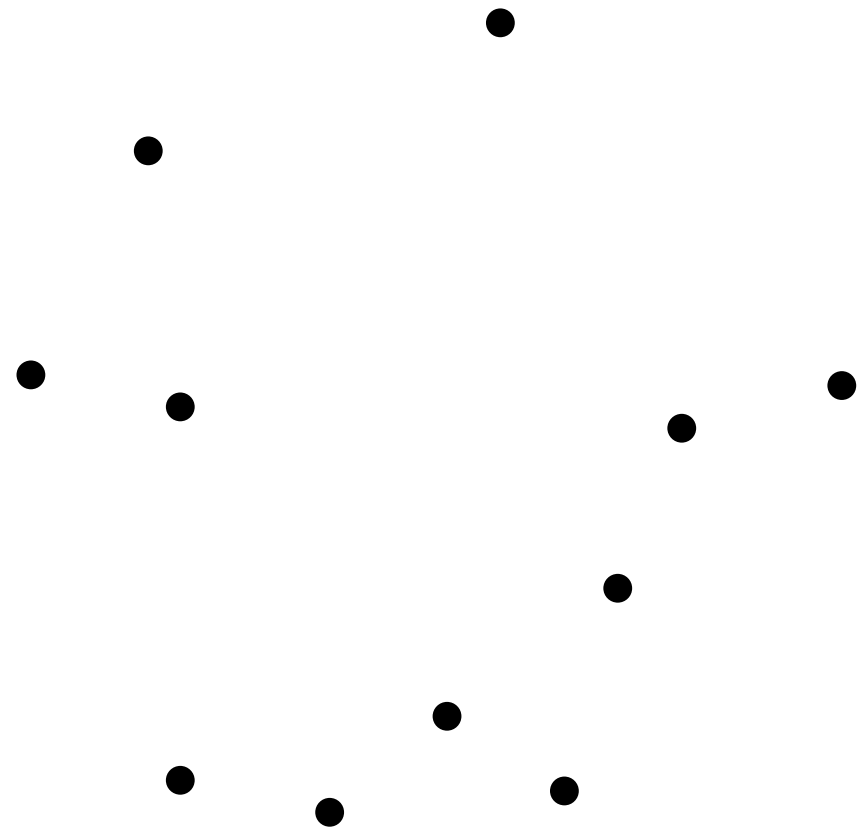
Procedure:

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

## 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

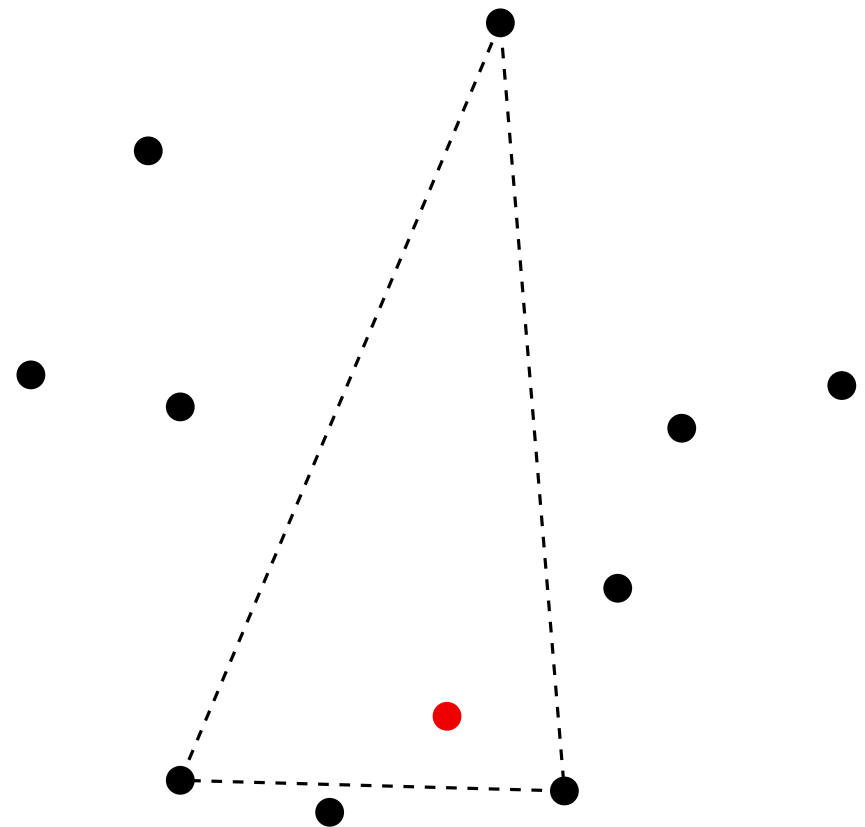
Procedure:

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

## 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

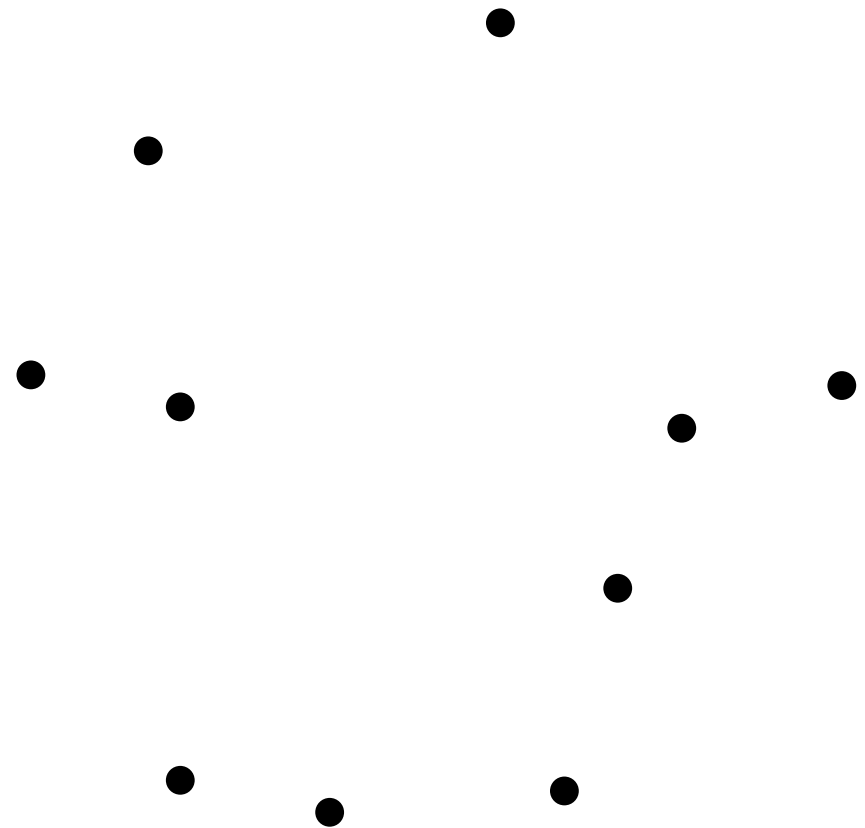
Procedure:

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

## 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

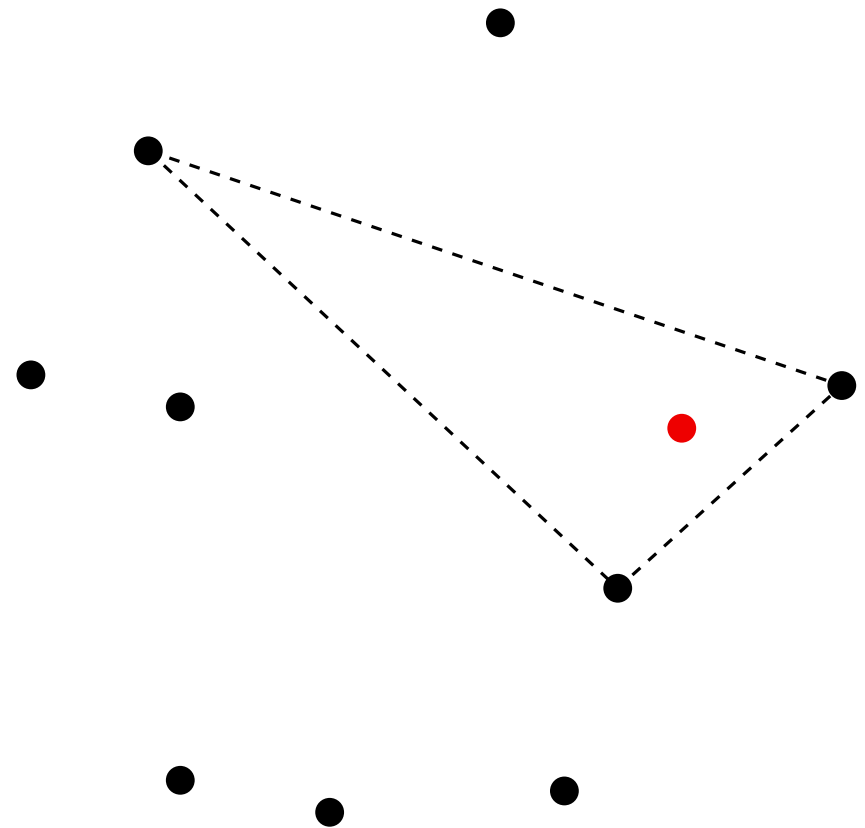
Procedure:

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

## 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

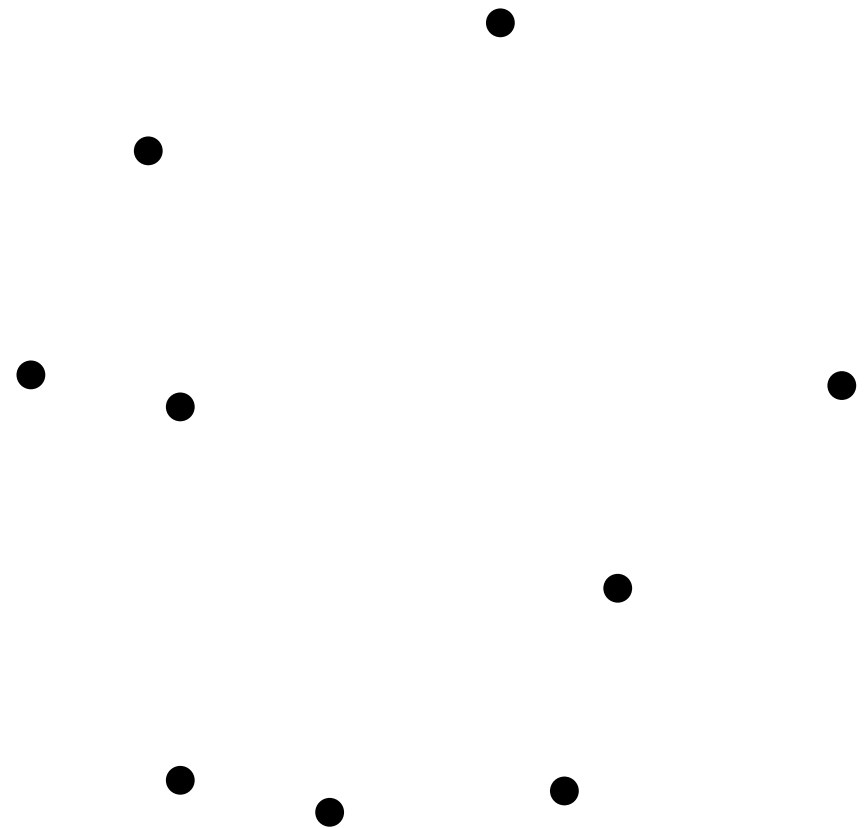
Procedure:

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

## 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

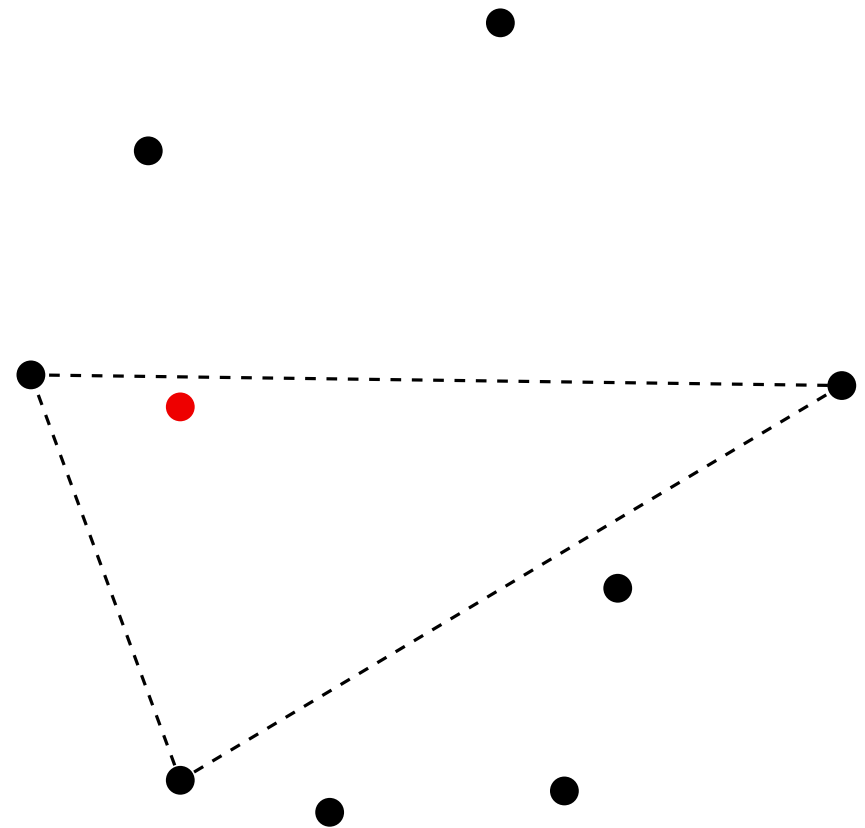
Procedure:

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

## 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

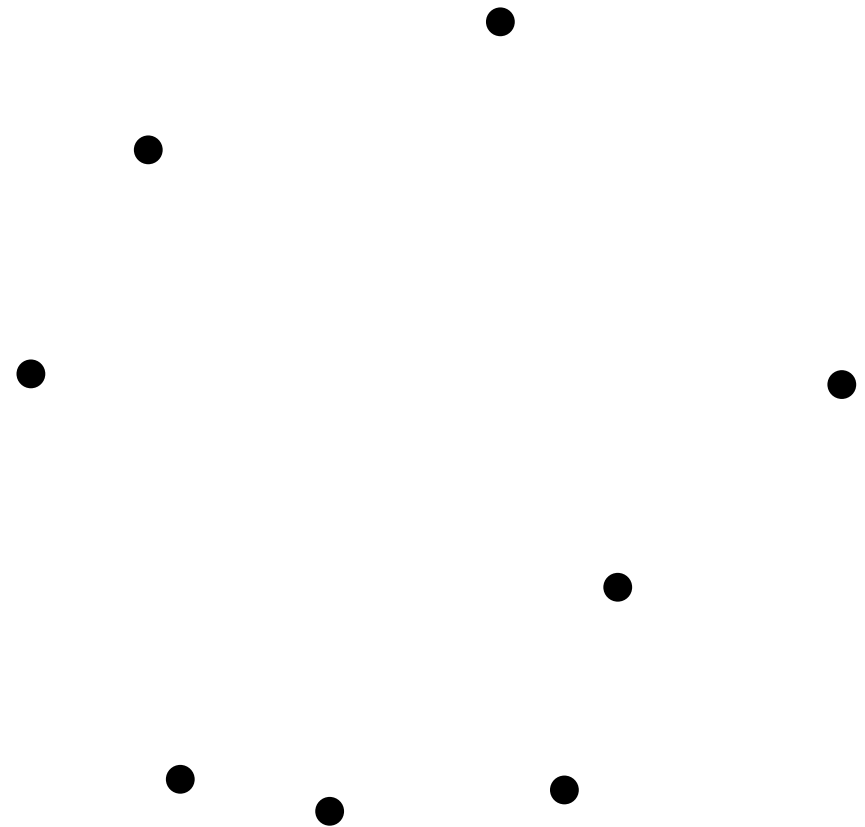
Procedure:

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

## 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

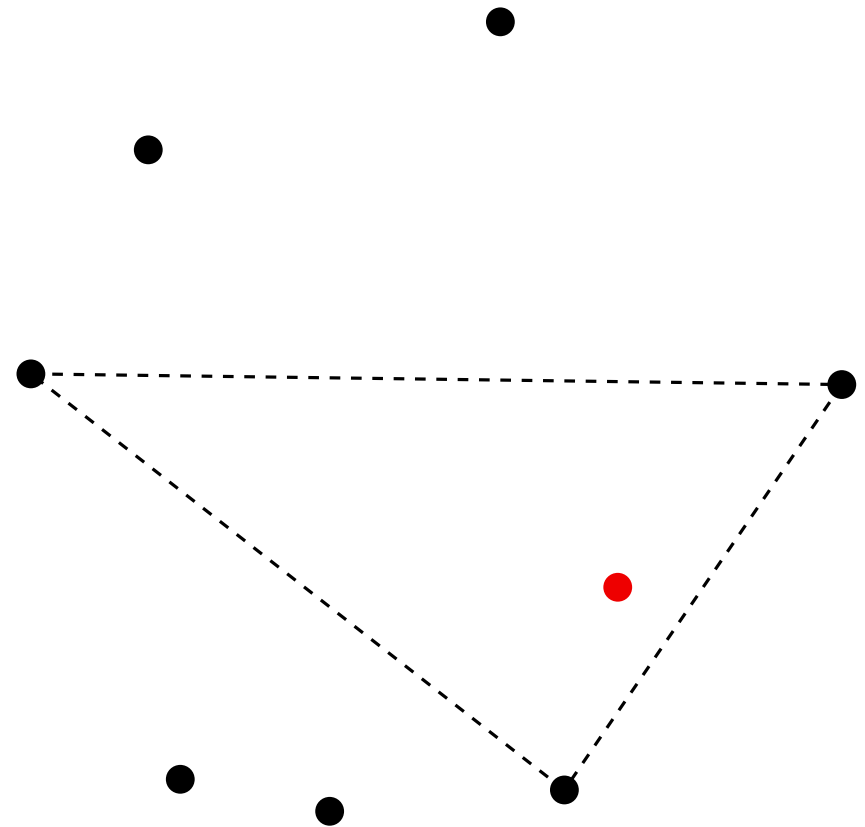
Procedure:

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

## 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

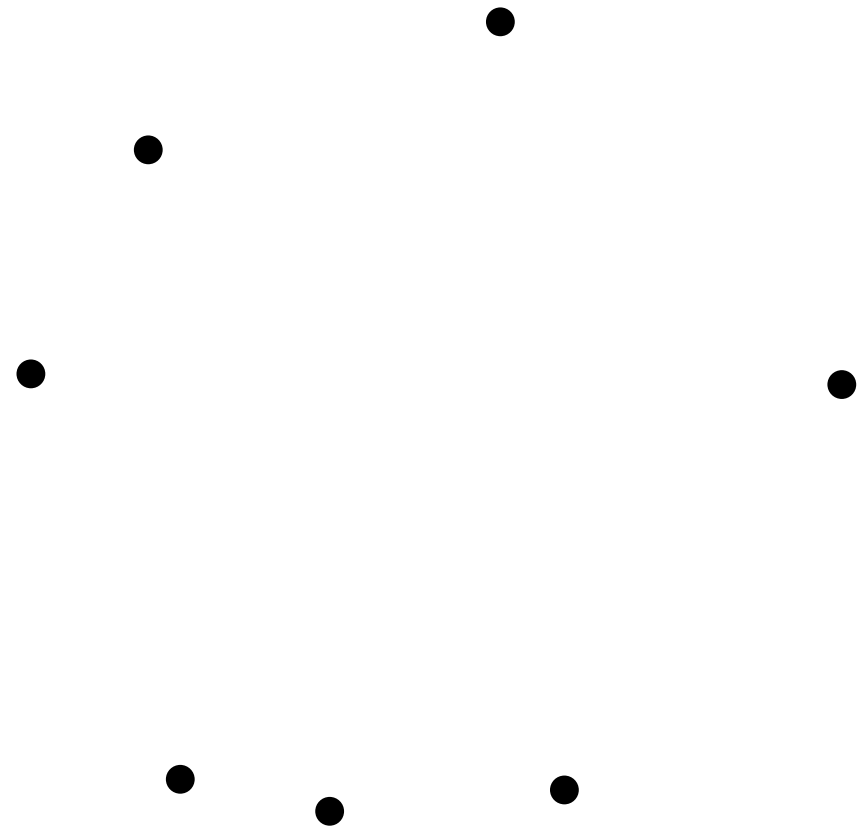
Procedure:

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

## 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:

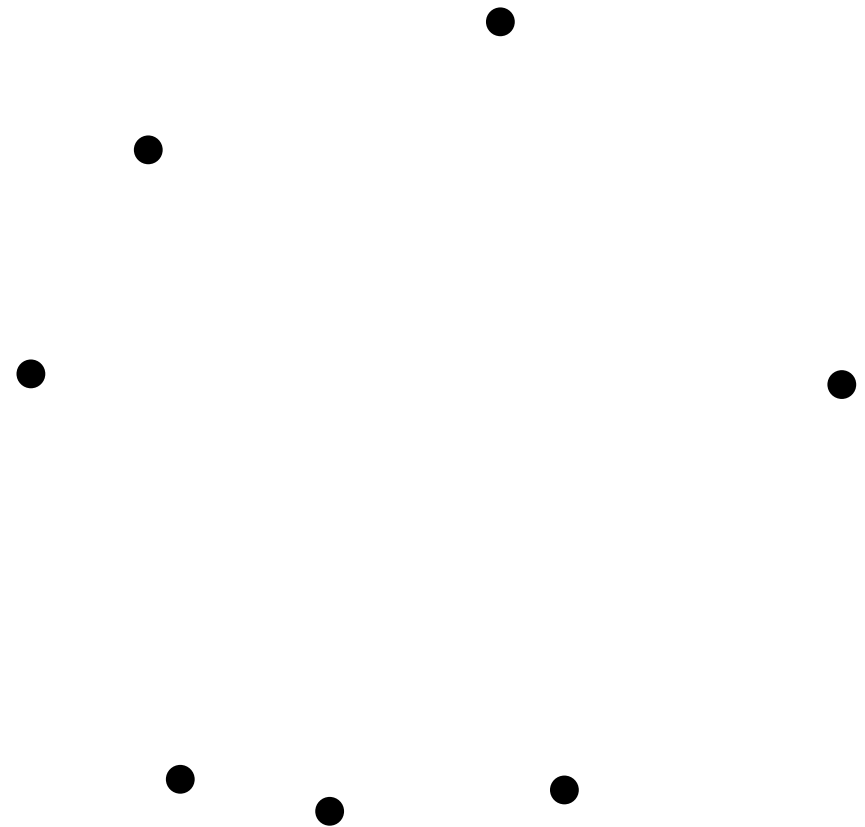
For each  $i$ ,

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.

Running time:  $O(n^4)$



# CONVEX HULL

Computing the extreme segments

# CONVEX HULL

## Computing the extreme segments

### Characterization

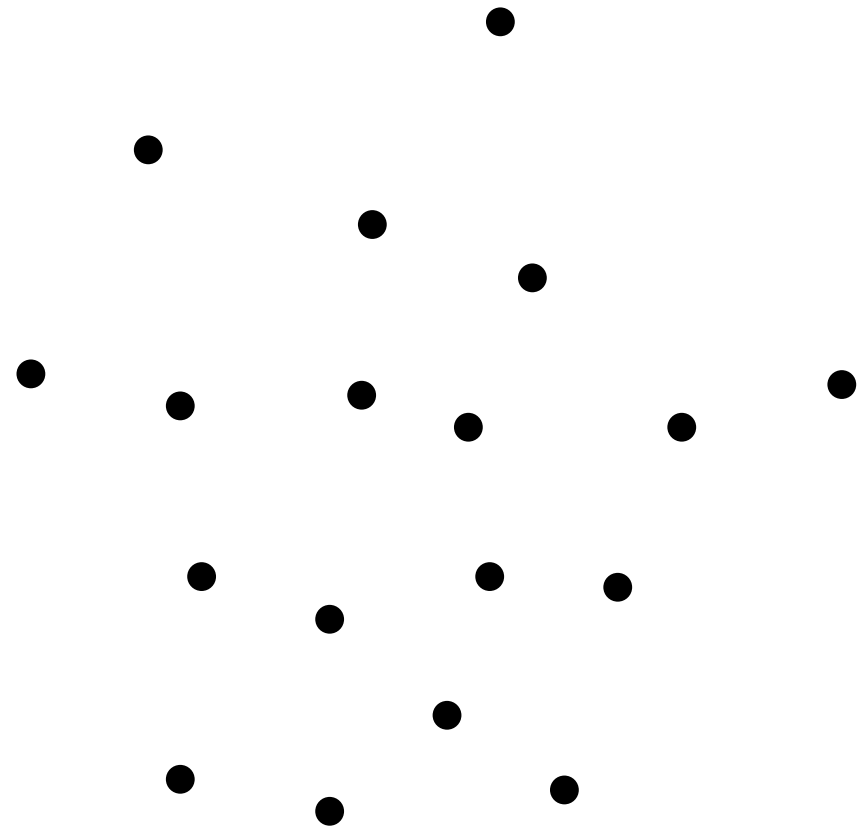
Given  $X = \{p_1, \dots, p_n\}$ , the segment  $p_i p_j$  is an extreme segment if and only if all the points  $p_k$  with  $k \neq i, j$  lie in the same halfplane defined by the line  $p_i p_j$ .

# CONVEX HULL

## 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  $p_i p_j$ .



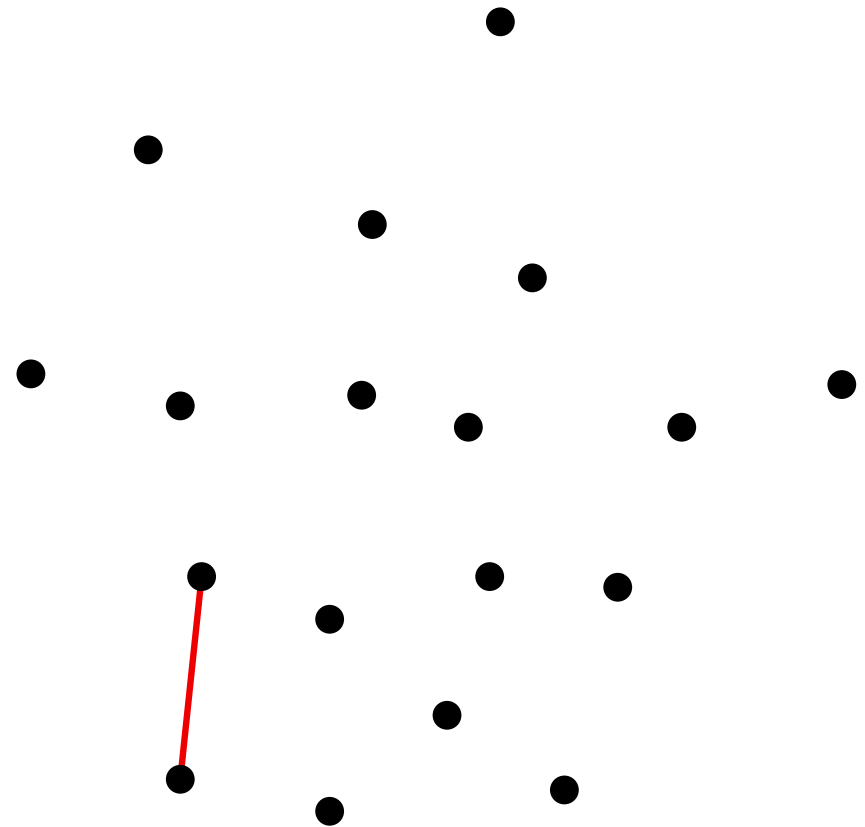


# CONVEX HULL

## 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  $p_i p_j$ .

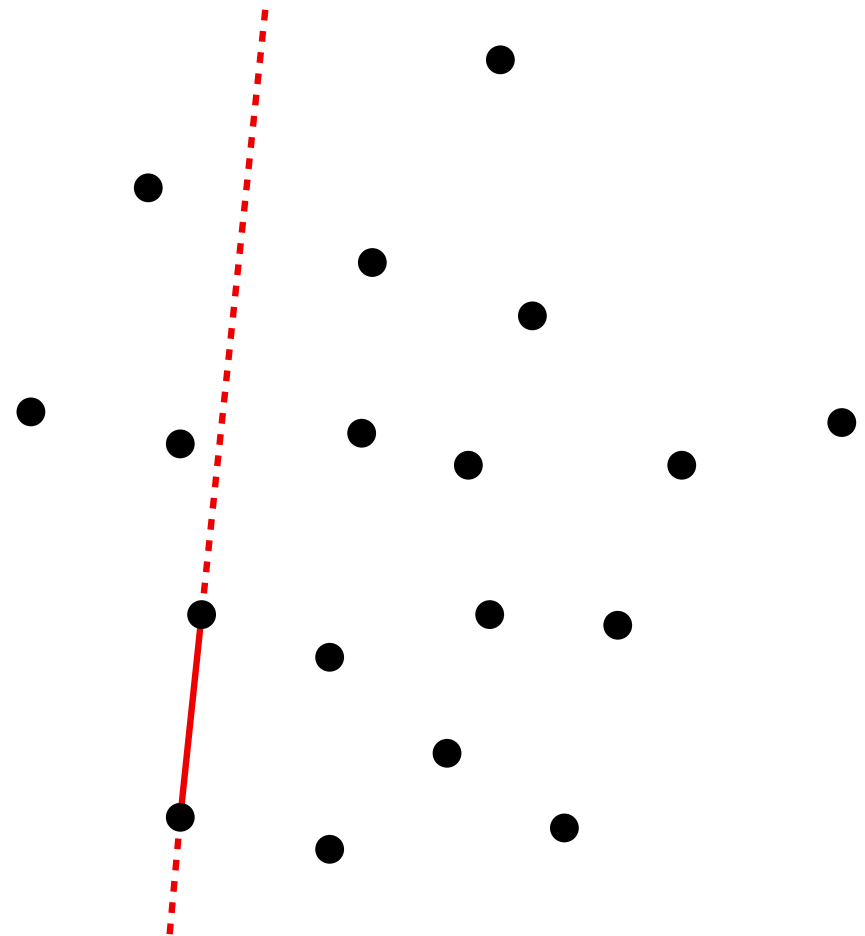


# CONVEX HULL

## 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  $p_i p_j$ .

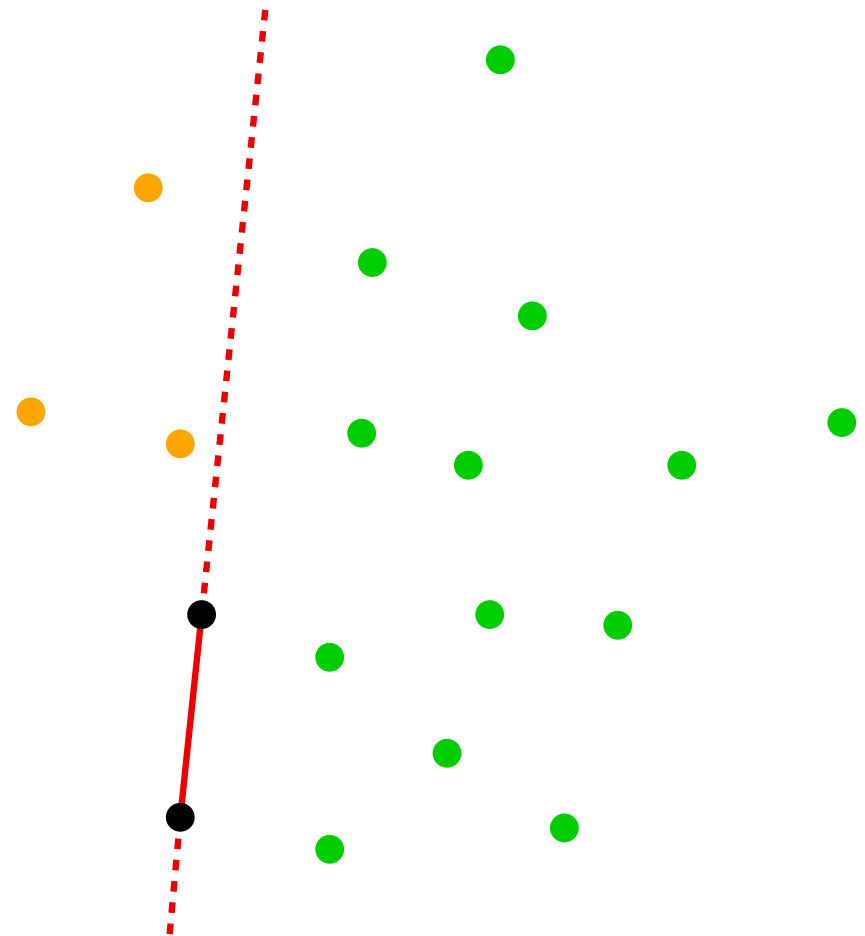


# CONVEX HULL

## 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  $p_i p_j$ .

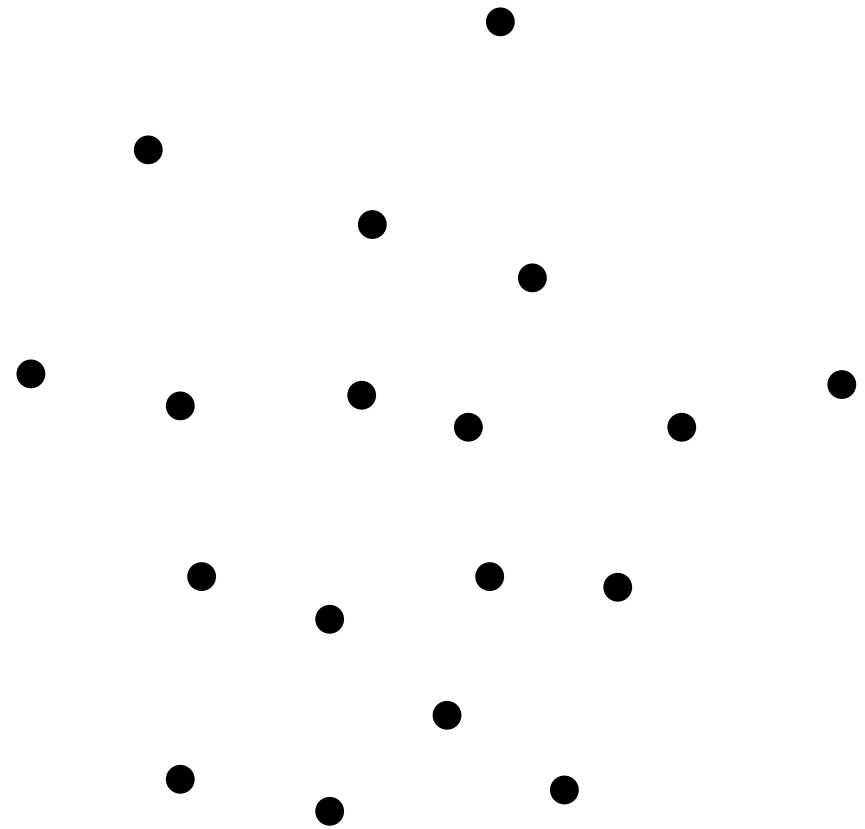


# CONVEX HULL

## 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  $p_i p_j$ .

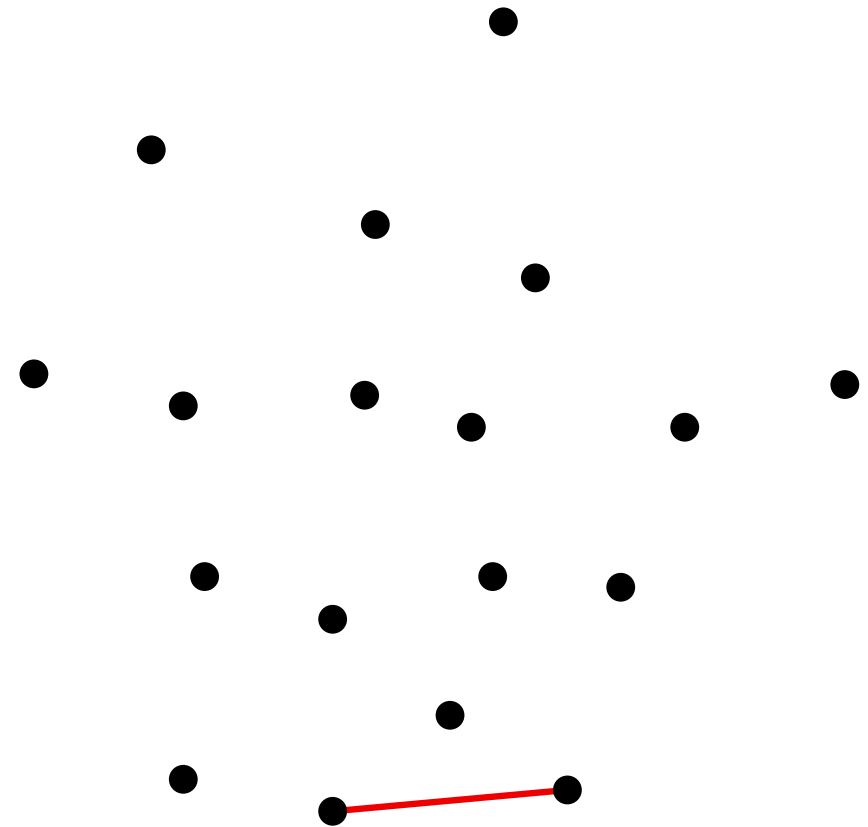


# CONVEX HULL

## 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  $p_i p_j$ .

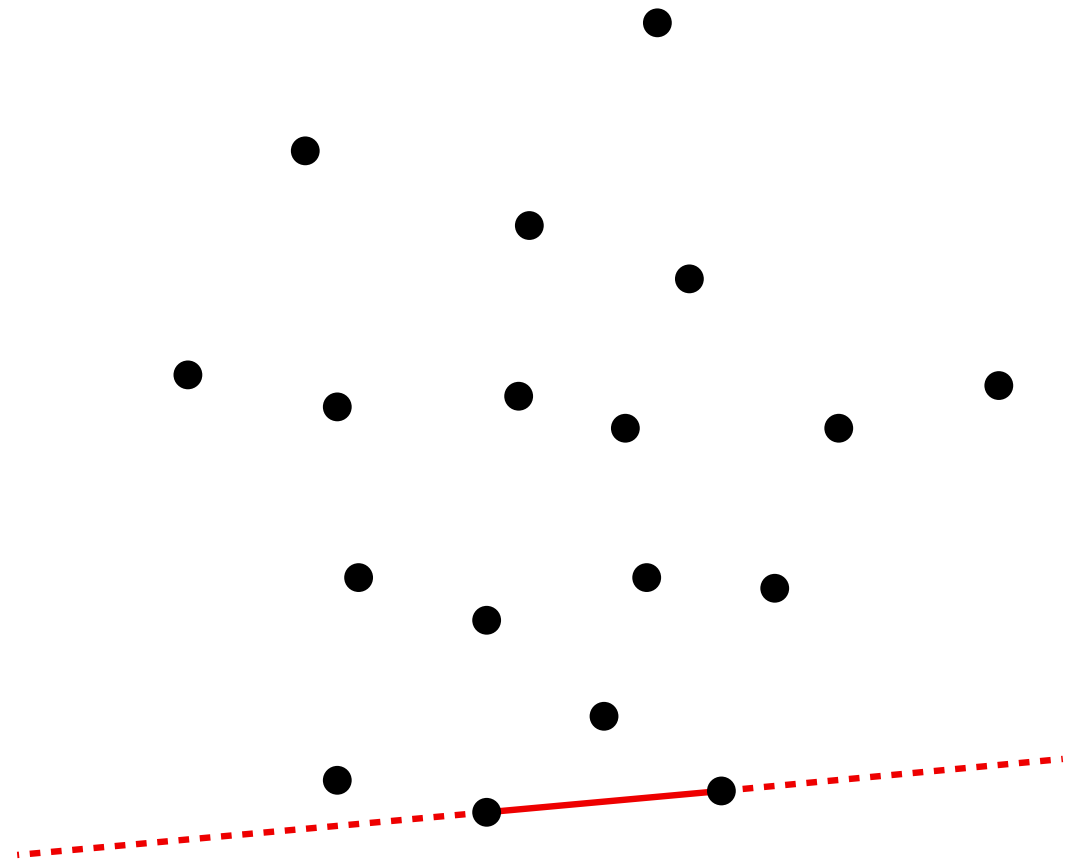


# CONVEX HULL

## 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  $p_i p_j$ .

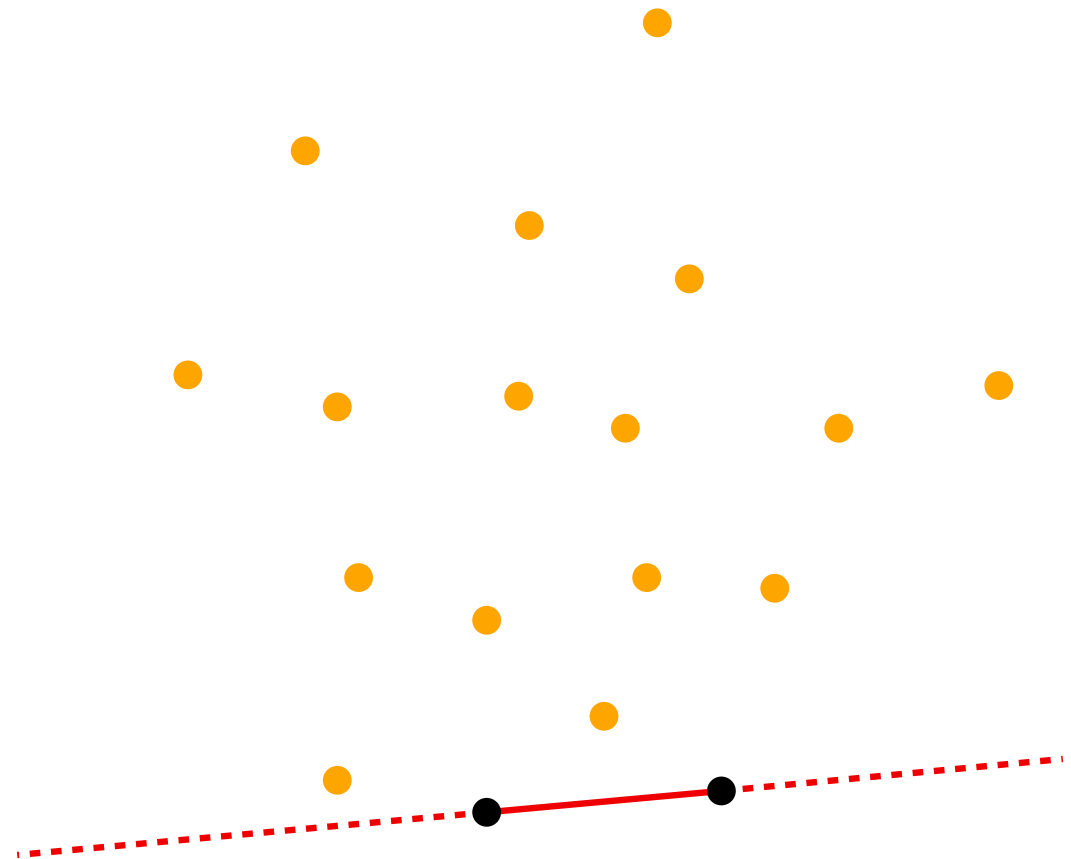


# CONVEX HULL

## 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  $p_i p_j$ .



# CONVEX HULL

## 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  $p_i p_j$ .

### Algorithm

Input:  $p_1, \dots, p_n$

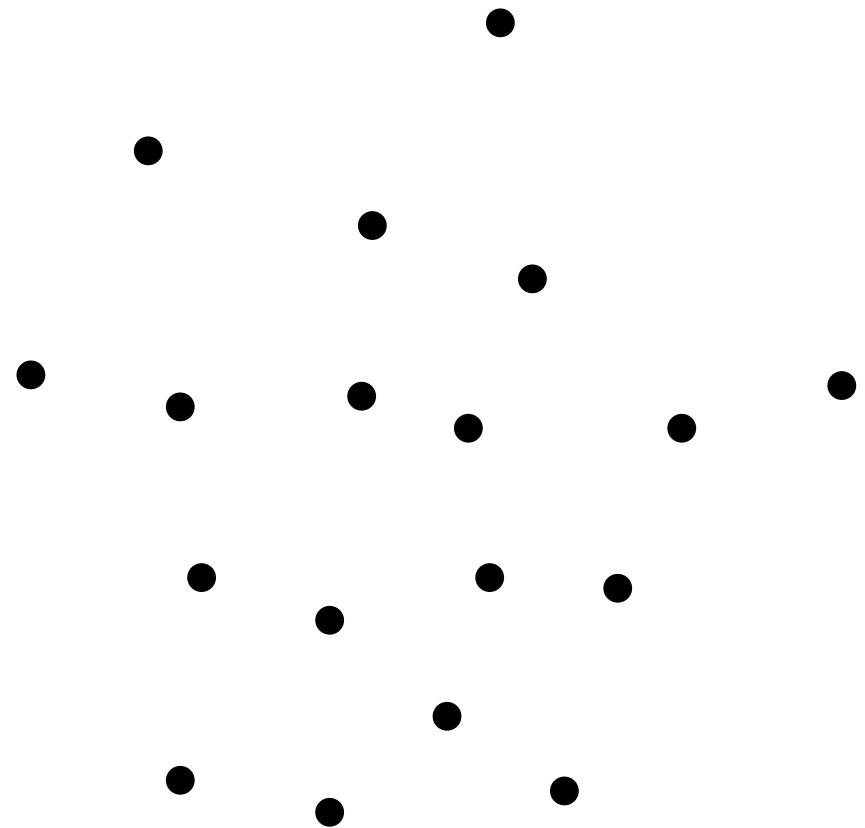
Output: set of the extreme segments

Procedure:

For each  $i, j$ ,

Check whether all  $p_k$  with  $k \neq i, j$   
lie in the same halfplane defined by  $p_i p_j$ .

In the affirmative, return the segment  $p_i p_j$ .





# CONVEX HULL

## 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  $p_i p_j$ .

### Algorithm

Input:  $p_1, \dots, p_n$

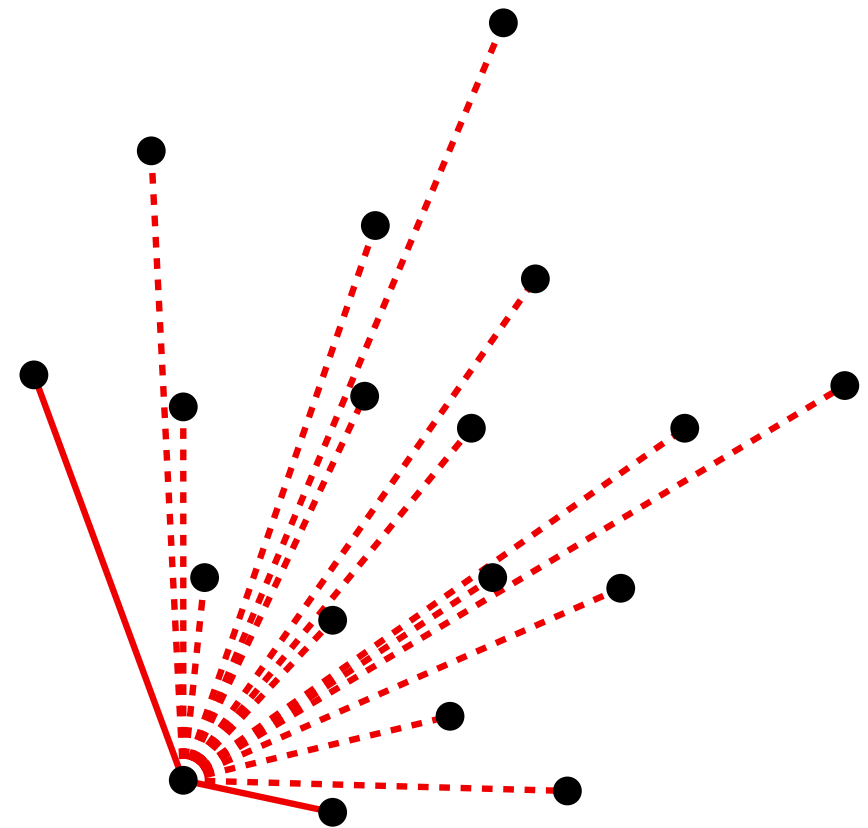
Output: set of the extreme segments

Procedure:

For each  $i, j$ ,

Check whether all  $p_k$  with  $k \neq i, j$   
lie in the same halfplane defined by  $p_i p_j$ .

In the affirmative, return the segment  $p_i p_j$ .



# CONVEX HULL

## 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  $p_i p_j$ .

### Algorithm

Input:  $p_1, \dots, p_n$

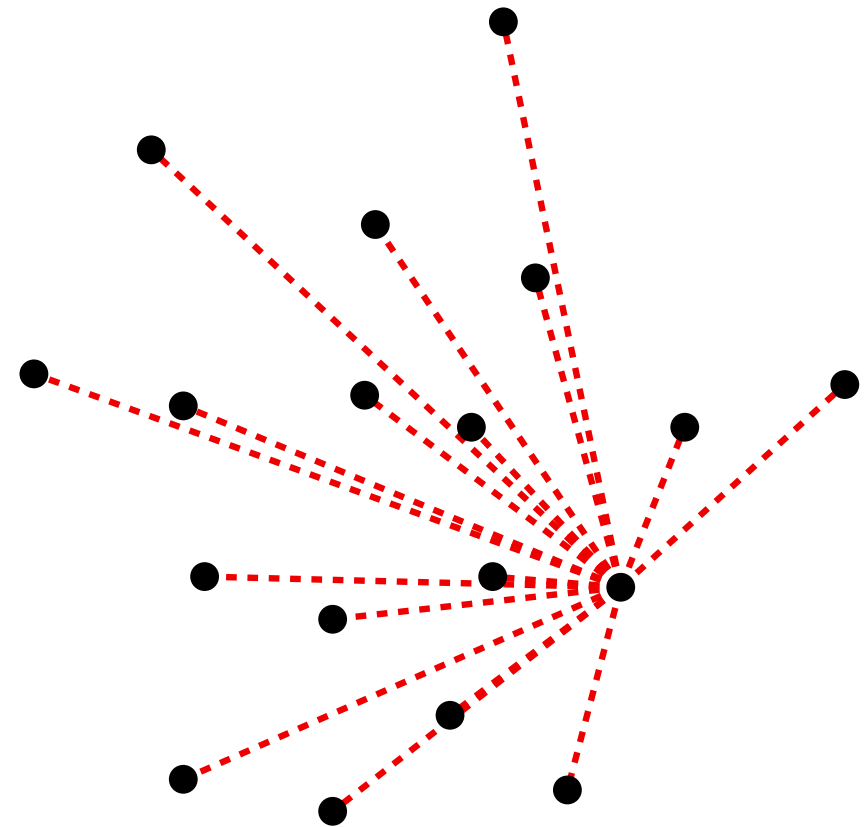
Output: set of the extreme segments

Procedure:

For each  $i, j$ ,

Check whether all  $p_k$  with  $k \neq i, j$   
lie in the same halfplane defined by  $p_i p_j$ .

In the affirmative, return the segment  $p_i p_j$ .



# CONVEX HULL

## 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  $p_i p_j$ .

### Algorithm

Input:  $p_1, \dots, p_n$

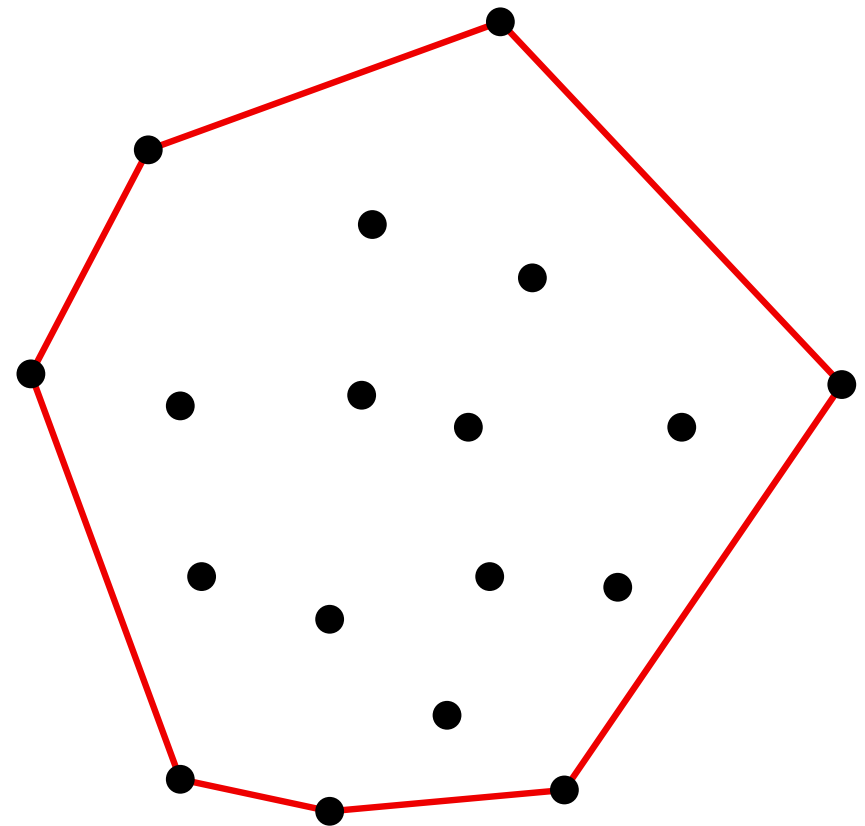
Output: set of the extreme segments

Procedure:

For each  $i, j$ ,

Check whether all  $p_k$  with  $k \neq i, j$   
lie in the same halfplane defined by  $p_i p_j$ .

In the affirmative, return the segment  $p_i p_j$ .



# CONVEX HULL

## 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  $p_i p_j$ .

### Algorithm

Input:  $p_1, \dots, p_n$

Output: set of the extreme segments

Procedure:

For each  $i, j$ ,

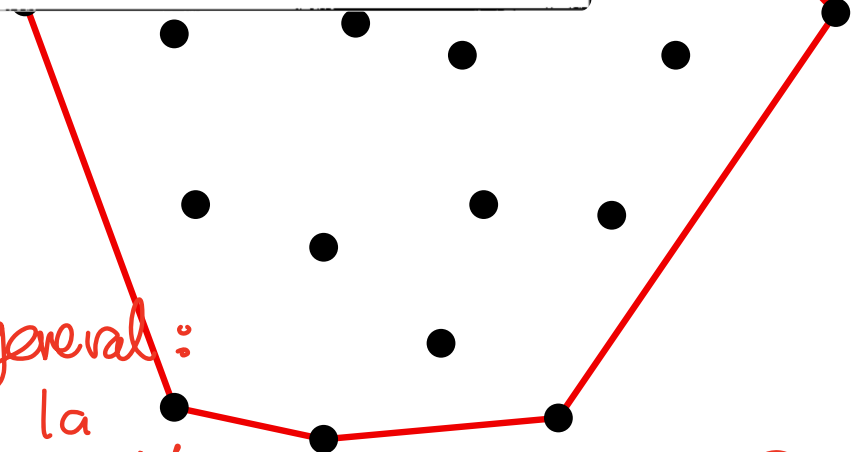
Check whether all  $p_k$  with  $k \neq i, j$  lie in the same halfplane defined by  $p_i p_j$ .  
In the affirmative, return the segment  $p_i p_j$ .

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

Problema si los puntos no están en posición general: el algoritmo produce la arista  $p_i p_j$ ,  $p_i p_k$  y  $p_k p_j$   $\forall$   $\rightarrow$  Produce salida  $\textcircled{2}$

O'Rourke.

```
Algorithm: EXTREME EDGES
for each i do
  for each j ≠ i do
    for each k ≠ i ≠ j do
      if pk is not left or on (pi, pj)
        then (pi, pj) is not extreme
```



# CONVEX HULL

Computing the convex hull

# CONVEX HULL

Computing the convex hull

(sorted list of its vertices)

# CONVEX HULL

## Computing the convex hull

### Input:

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

### Output:

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

↗ en el algoritmo original se genera como salida la lista de aristas, esta es una pequeña variante: generar los vértices como salida.

\* consultar original en O'Rourke

# CONVEX HULL

## Computing the convex hull

### Input:

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

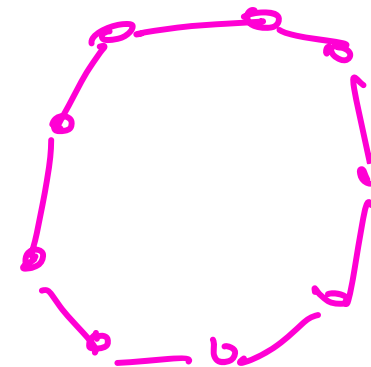
### Output:

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

## Characterization

Given  $X = \{p_1, \dots, p_n\}$ , 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  $p_i p_j$ .

Podemos mejorar el algoritmo anterior  
Hay exactamente  $n$  aristas.






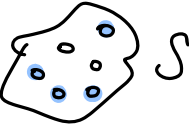
# CONVEX HULL

Jarvis march, 1973. Raymond Austin Jarvis, Ing. Electrico Australiano  
murió en 2013,

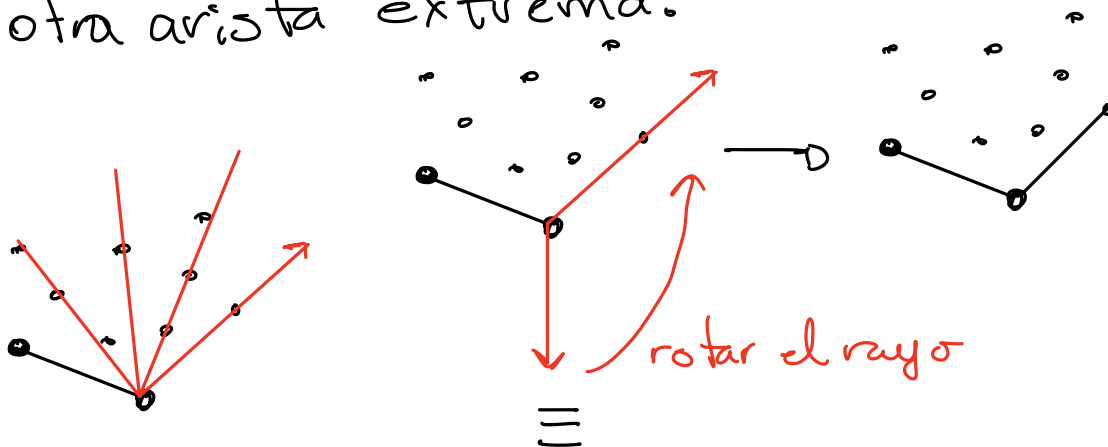
Gift wrapping.

Suponemos posición general. 

Idea: Usar la información ya encontrada para encontrar nueva información. Esta es una técnica muy común.

Podemos replantear el problema: 

Dada una arista extrema de  $ch(S)$ , encontrar otra arista extrema.



# CONVEX HULL

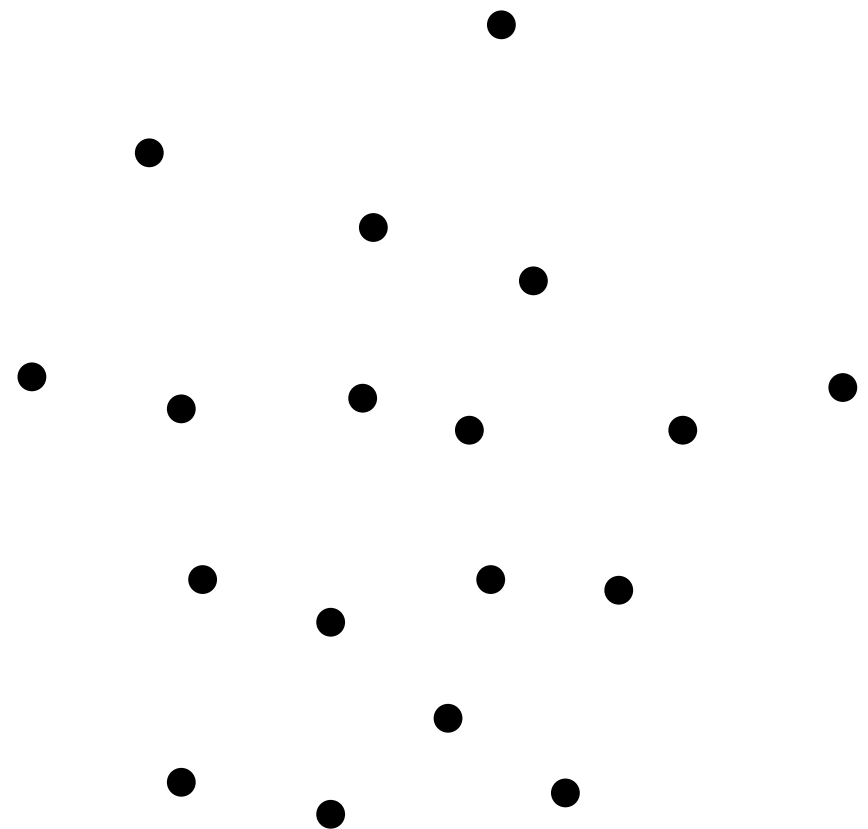
## Jarvis march

- $O(n)$  *El de menor coord y, por ejemplo.*
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

## Jarvis march

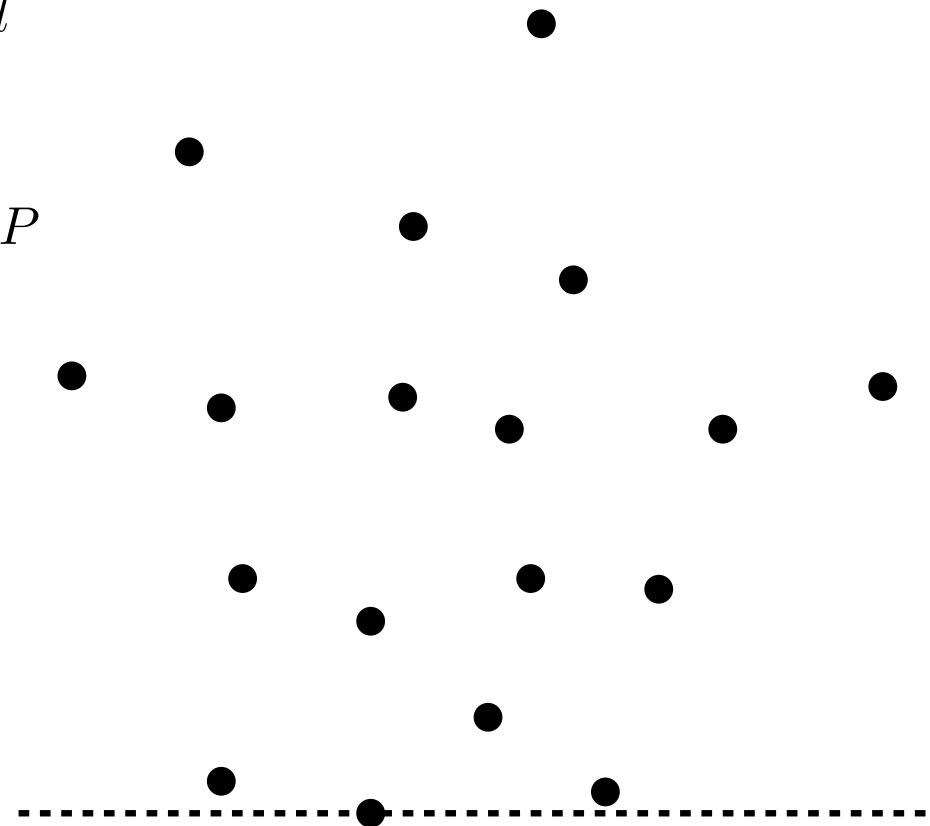
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## 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  $l$

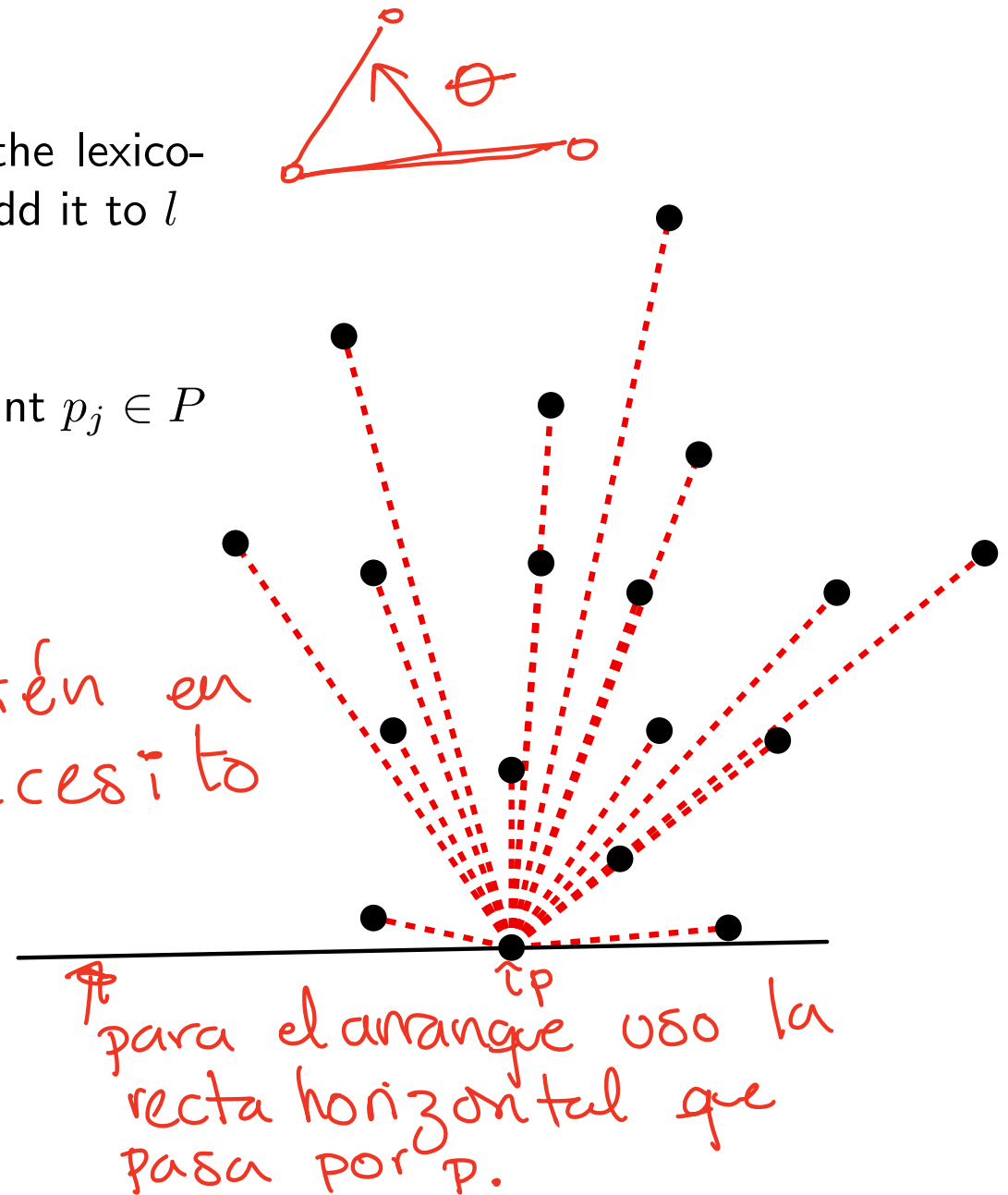


# CONVEX HULL

## 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  $l$

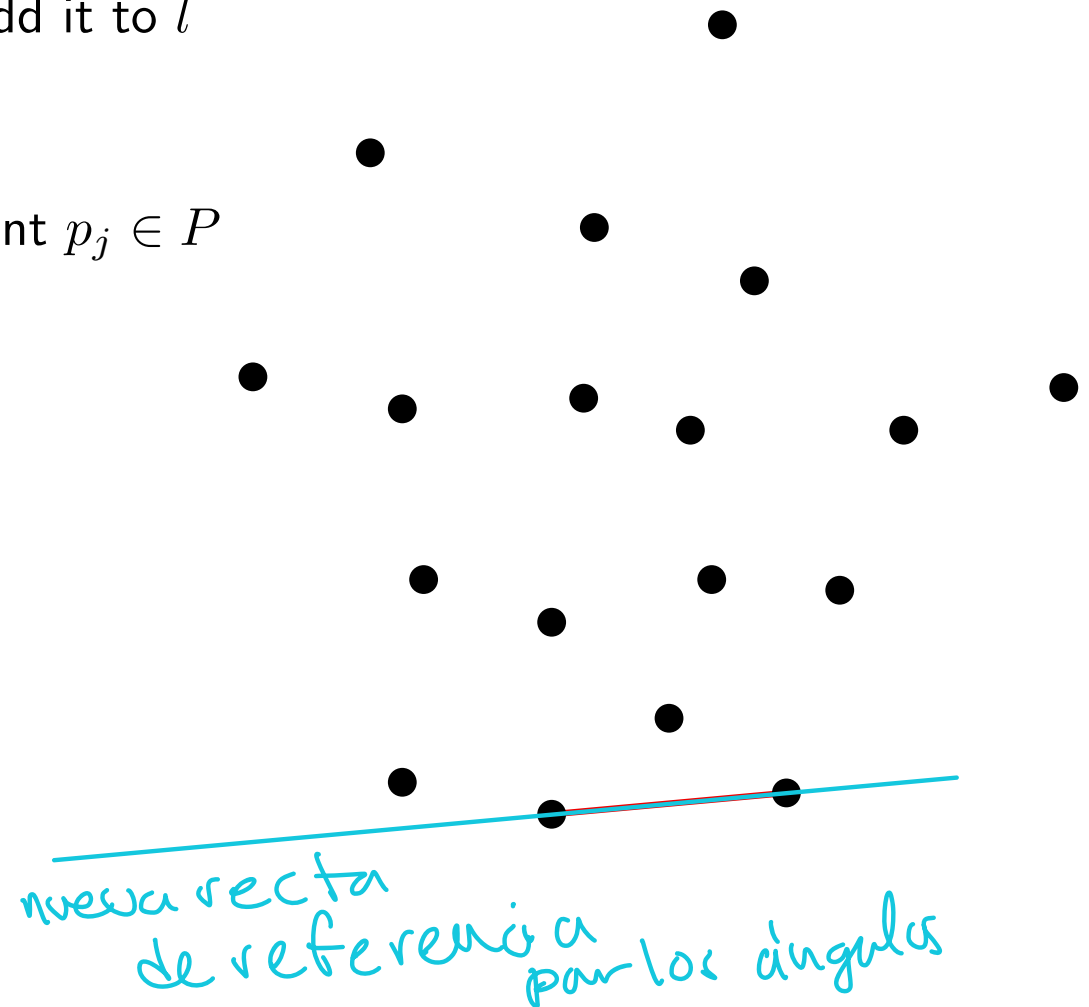
no necesario que estén en orden, únicamente necesario el mínimo



# CONVEX HULL

## Jarvis march

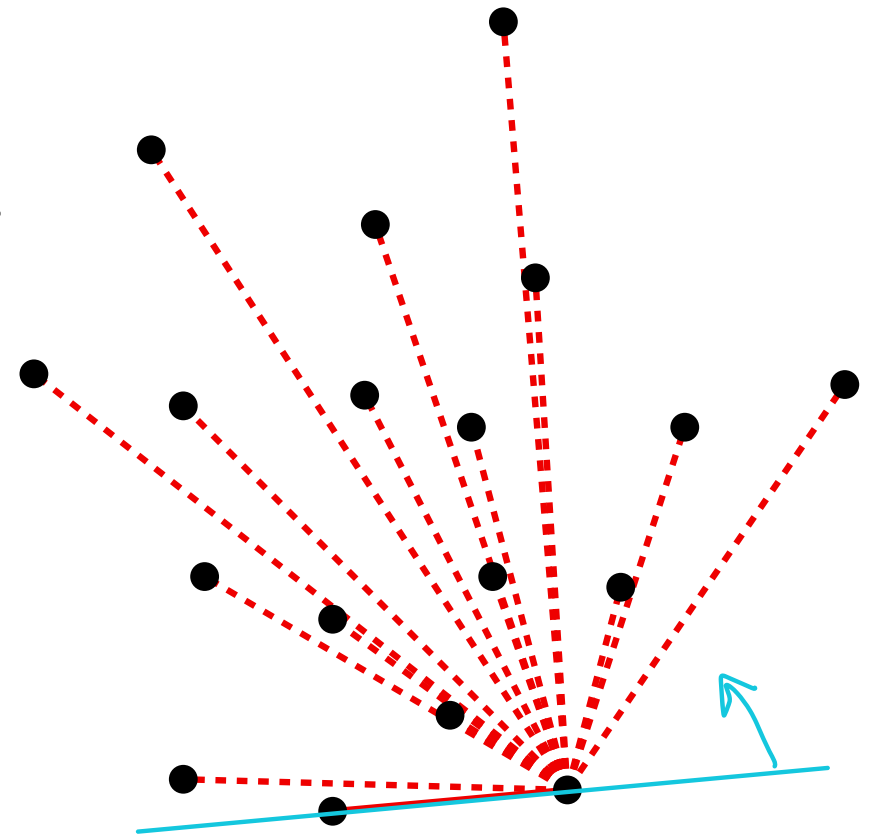
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## Jarvis march

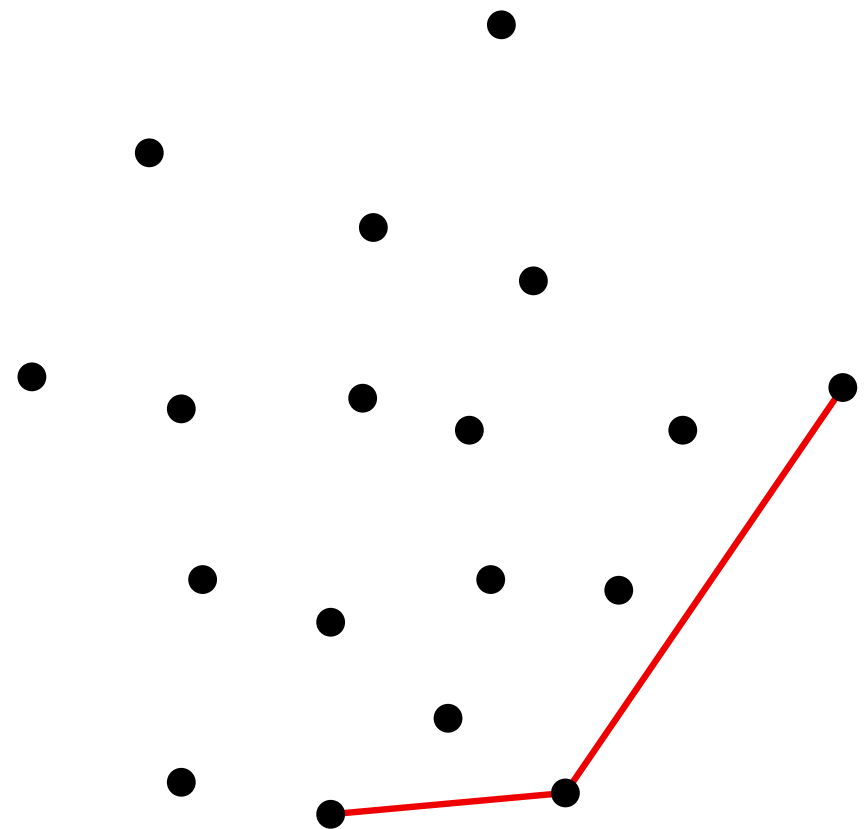
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## 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  $l$

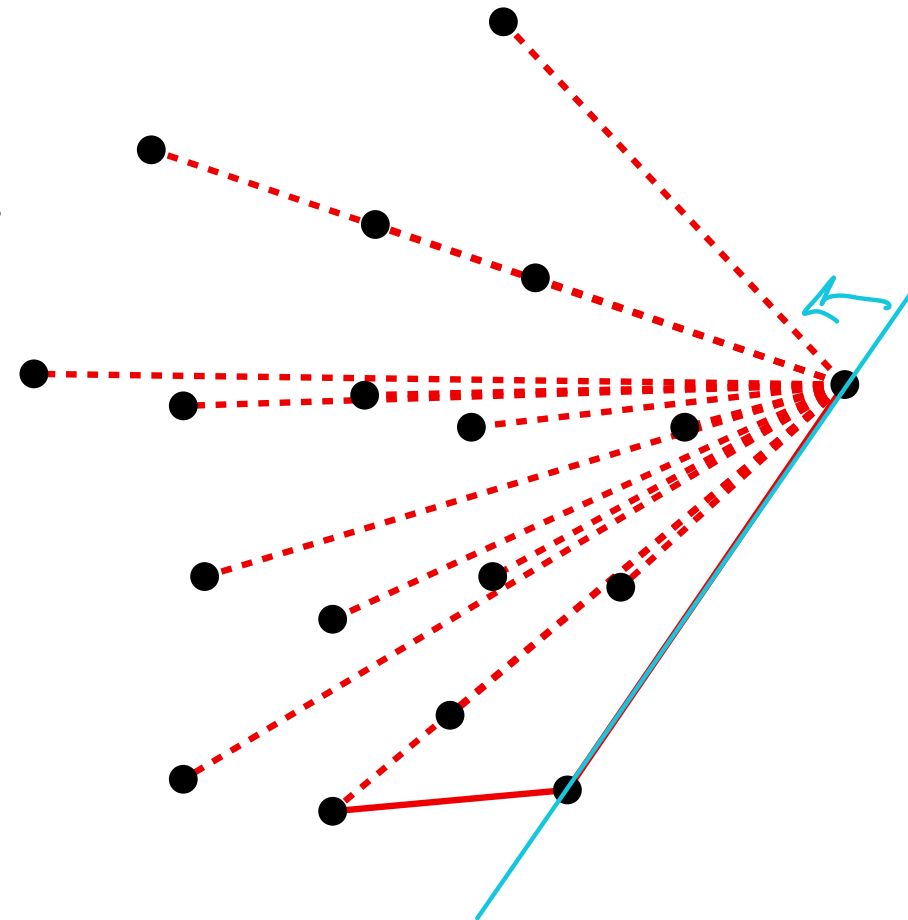




# CONVEX HULL

## Jarvis march

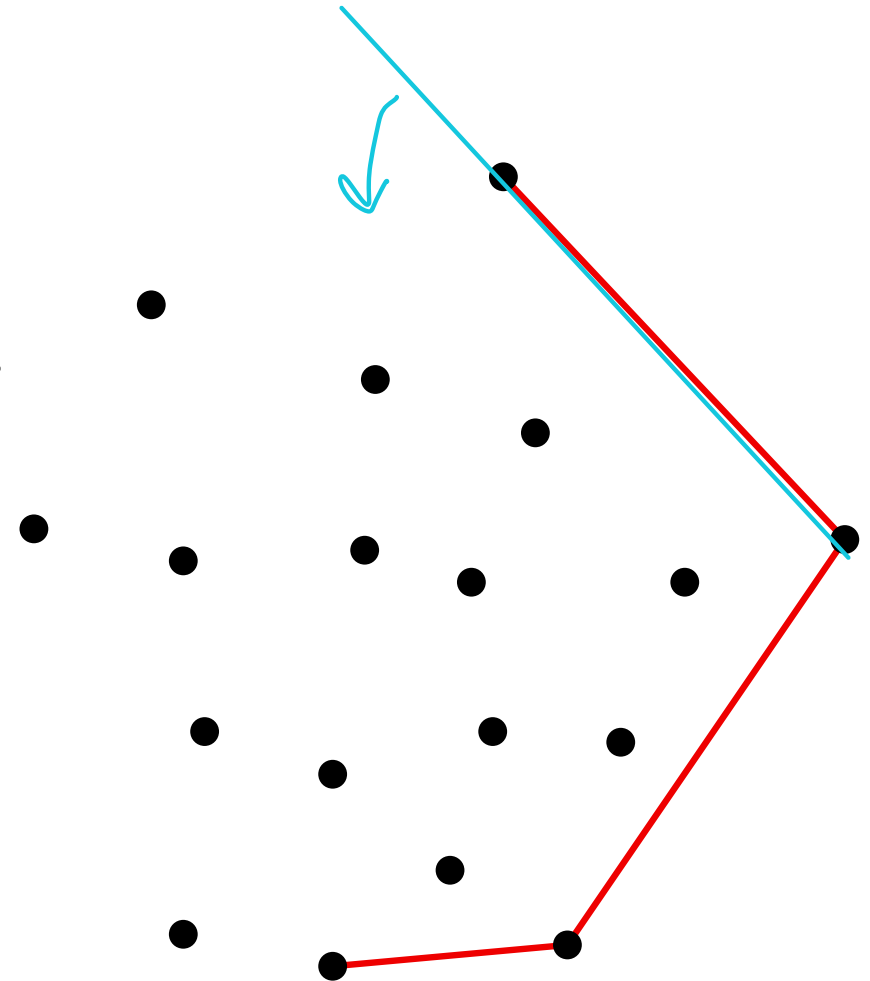
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## Jarvis march

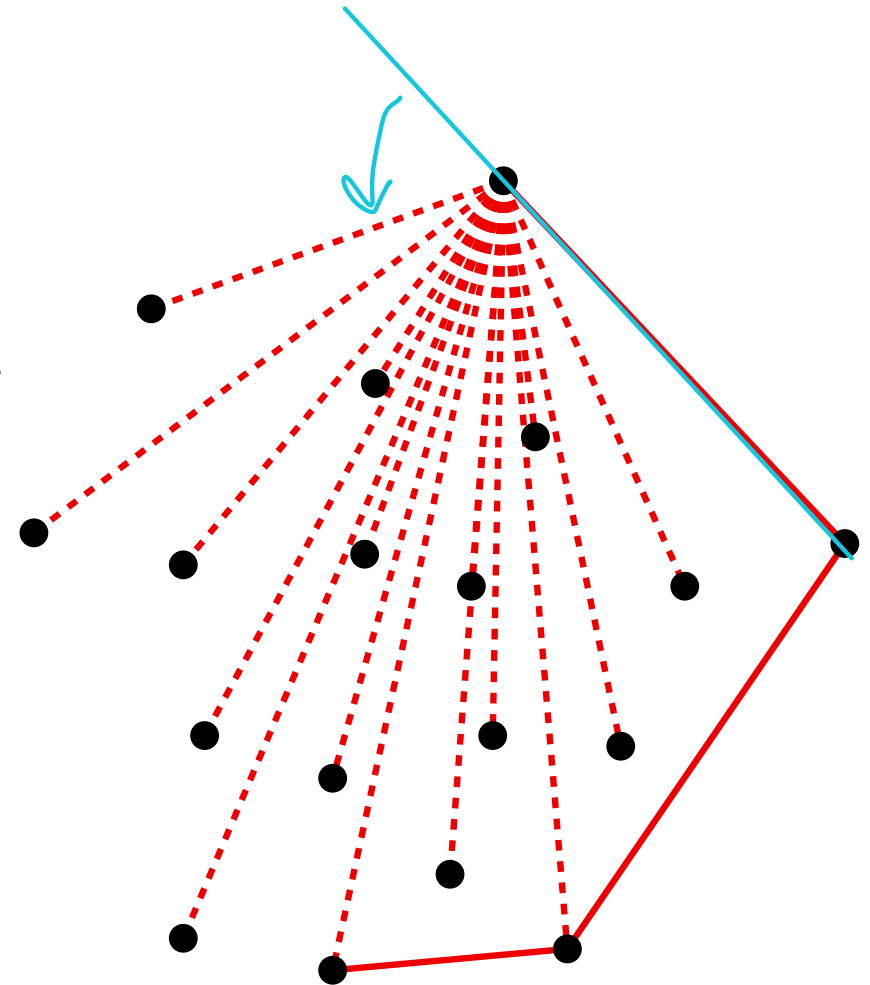
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## Jarvis march

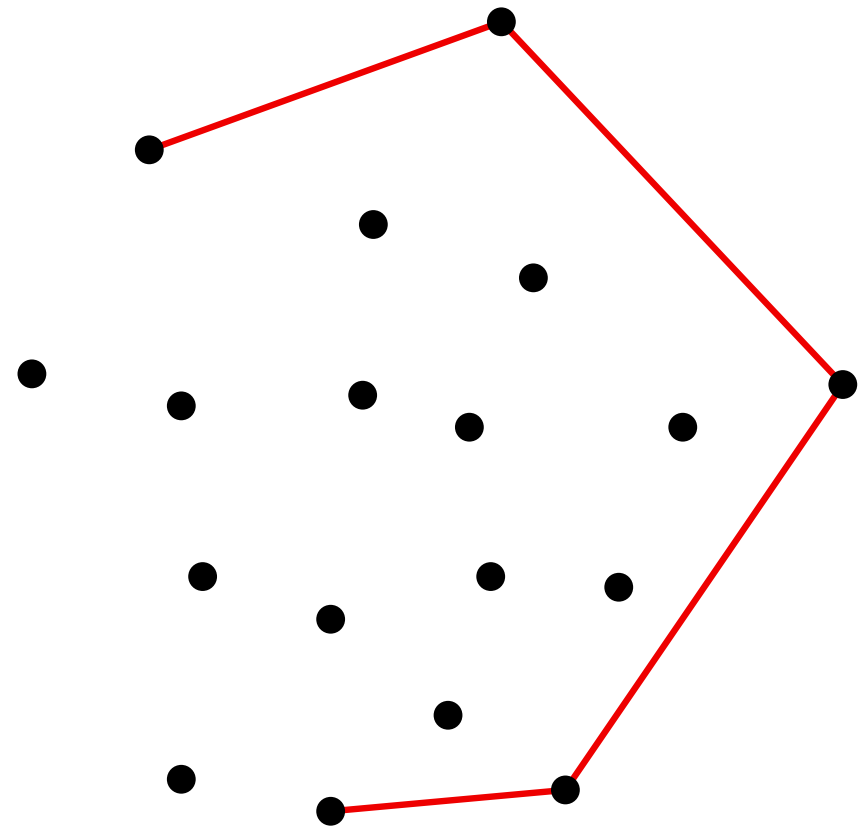
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## Jarvis march

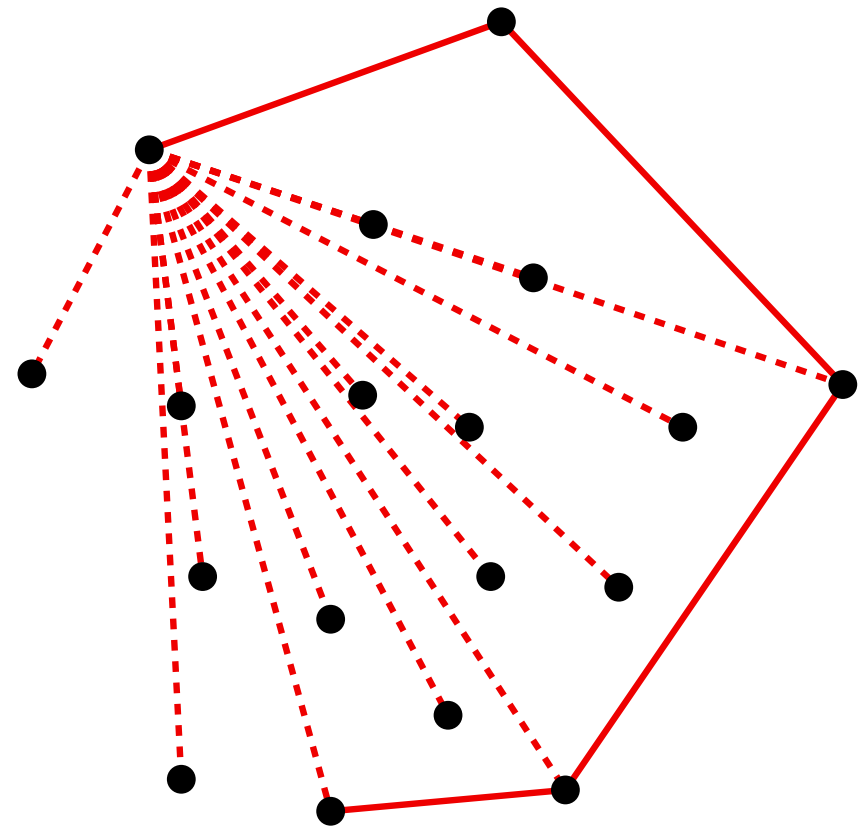
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## Jarvis march

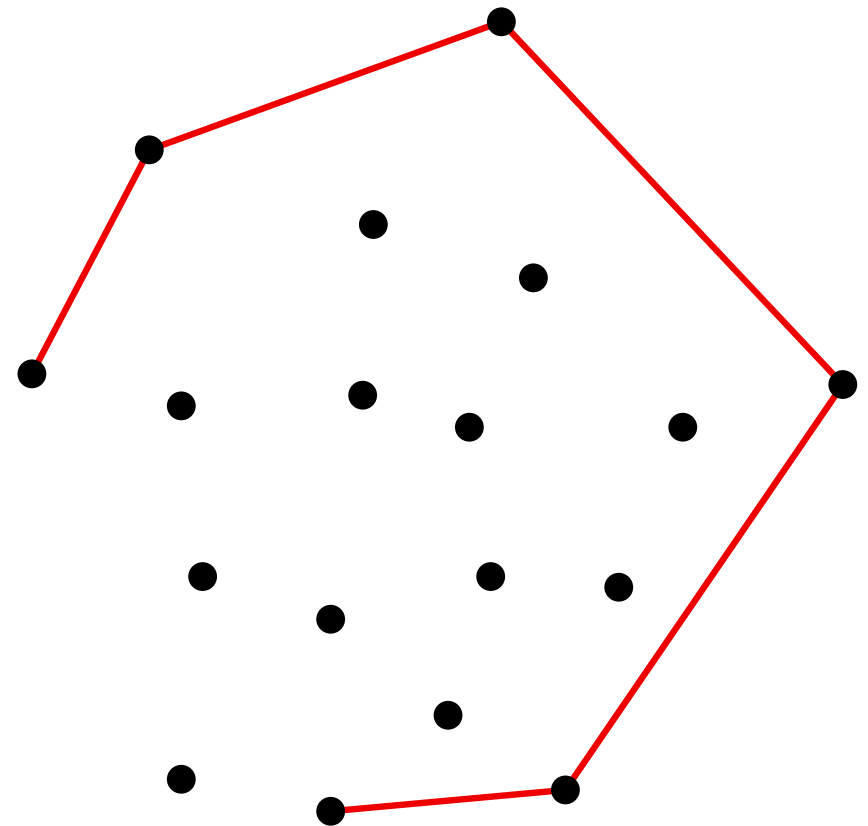
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## Jarvis march

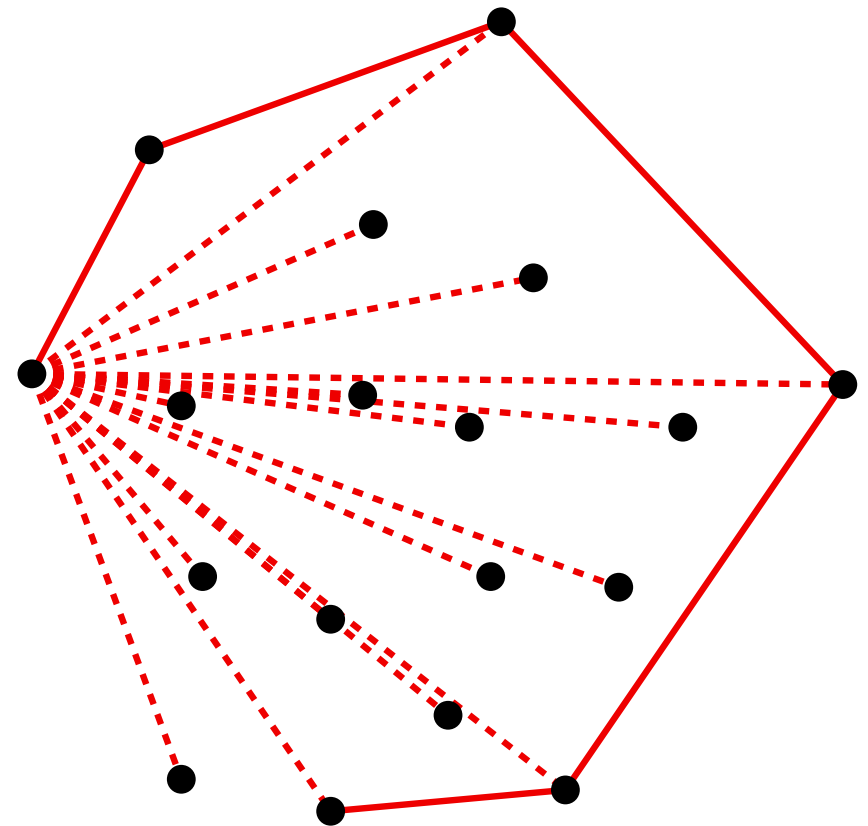
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## Jarvis march

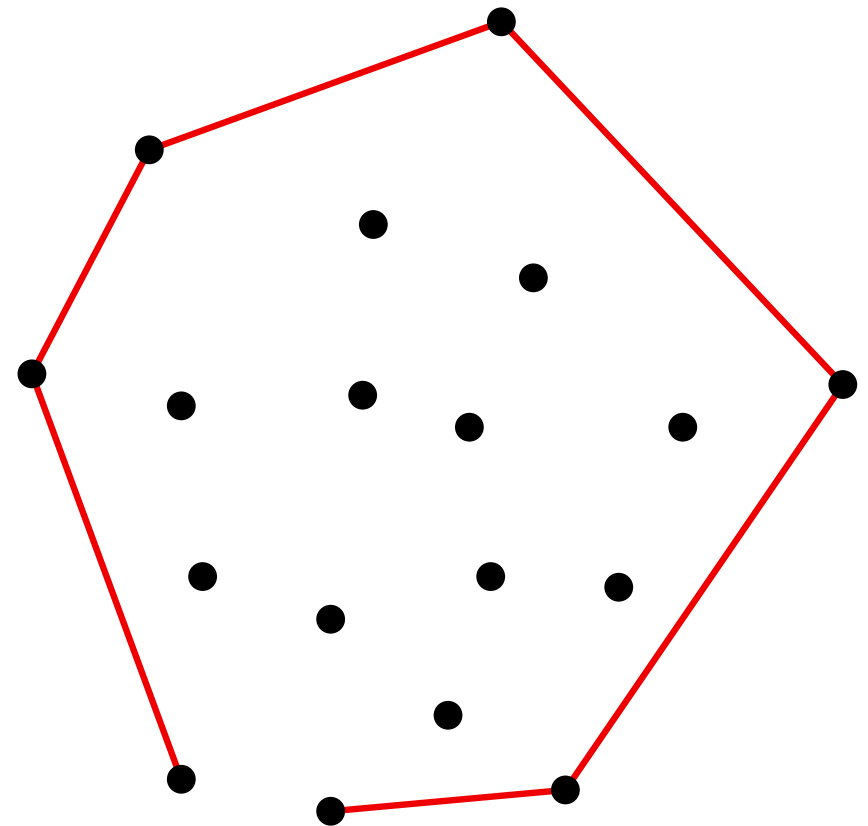
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## 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  $l$

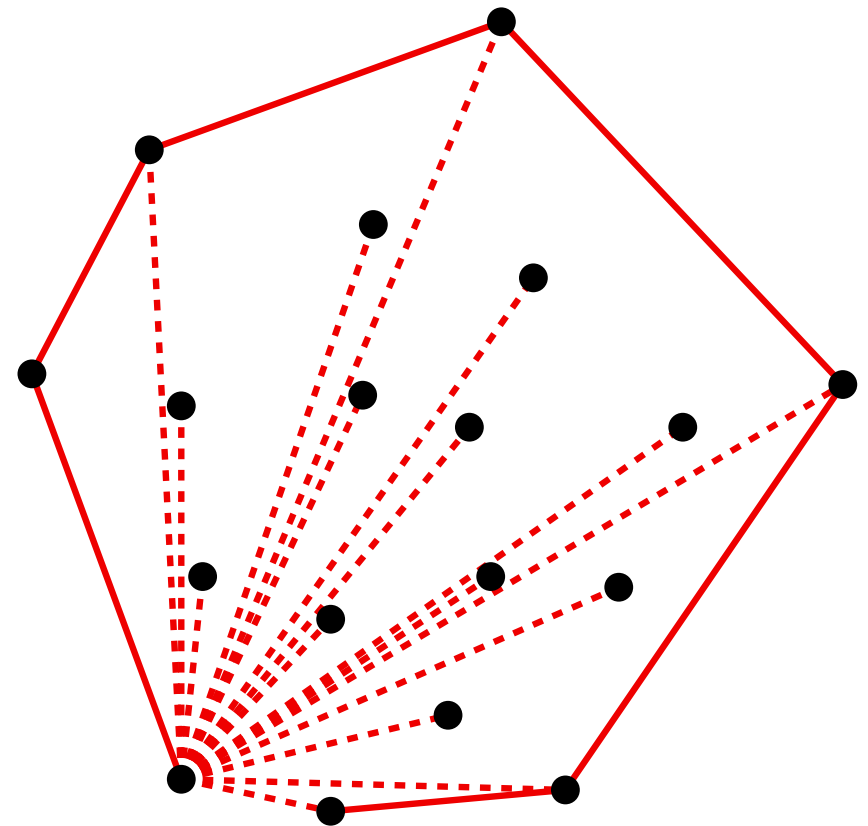




# CONVEX HULL

## Jarvis march

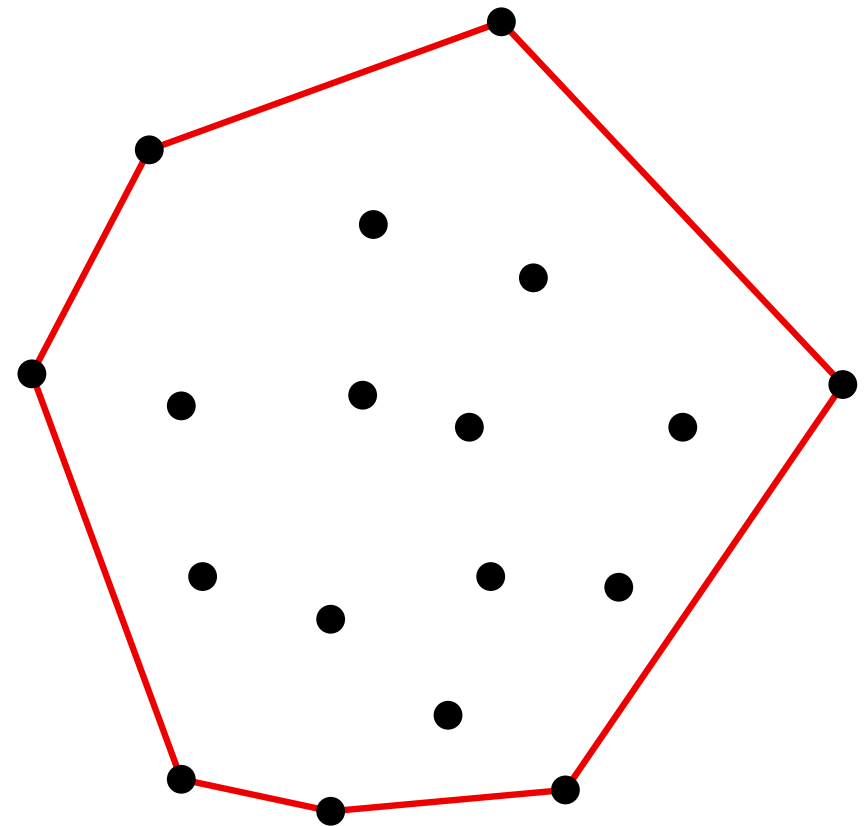
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

## Jarvis march

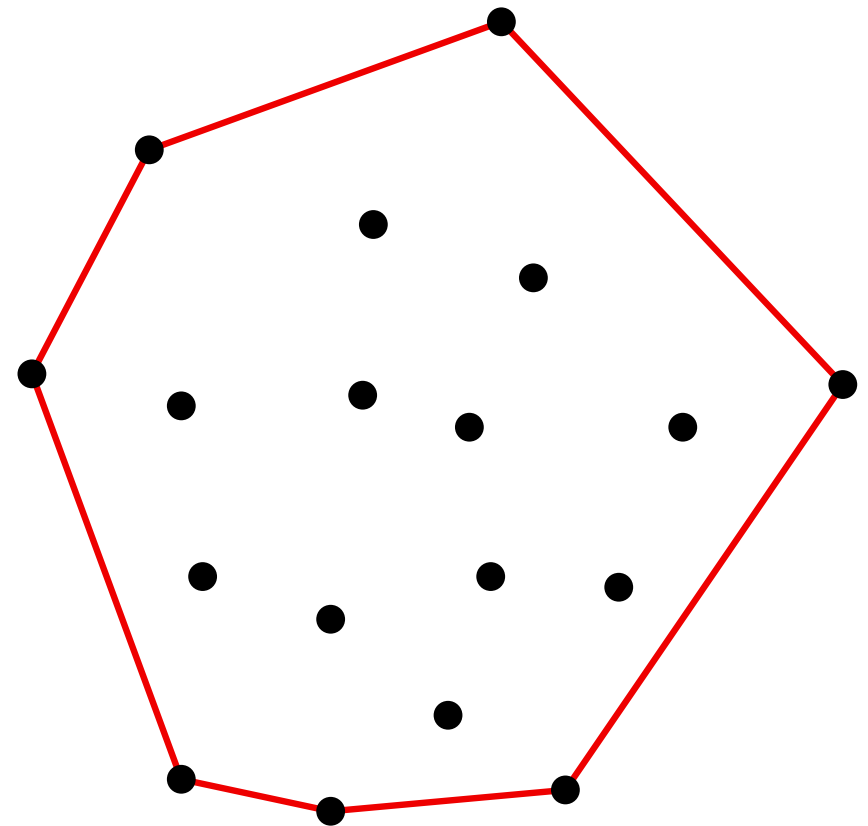
1. Find a vertex of  $ch(P)$  (for example, the lexicographically smaller point  $p_i \in P$ ) and add it to  $l$
2. While  $v = \text{Last}(l) \neq \text{First}(l)$ , do:
  - (a) Detect the angularly rightmost point  $p_j \in P$  with respect to  $v$ .
  - (b) Add  $p_j$  to  $l$
3. Return  $l$



# CONVEX HULL

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

tamaño de la salida  $\rightarrow$   $h$   
tamaño de la entrada  $\rightarrow$   $n$

output sensitive

$h = \#$  de vértices del cierre convexo.

# CONVEX HULL

QuickHull algorithm (by prune-and-search) *Varios investigadores al final de la década de los 70. lo bautizaron QuickHull por su similitud con QuickSort.*   
 *Preparafa & Shamos, 1985.*

## QuickSort

```
QUICKSORT(A[1..n]):  
  if (n > 1)  
    Choose a pivot element A[p]  
    r ← PARTITION(A, p)  
    QUICKSORT(A[1..r-1])  «Recurse!»  
    QUICKSORT(A[r+1..n]) «Recurse!»
```

```
PARTITION(A[1..n], p):  
  swap A[p] ↔ A[n]  
  ℓ ← 0  «#items < pivot»  
  for i ← 1 to n-1  
    if A[i] < A[n]  
      ℓ ← ℓ + 1  
      swap A[ℓ] ↔ A[i]  
  swap A[n] ↔ A[ℓ + 1]  
  return ℓ + 1
```

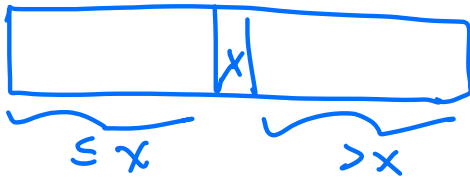


Figure 1.8. Quicksort

# CONVEX HULL

## QuickHull algorithm (by prune-and-search)

### Initialization

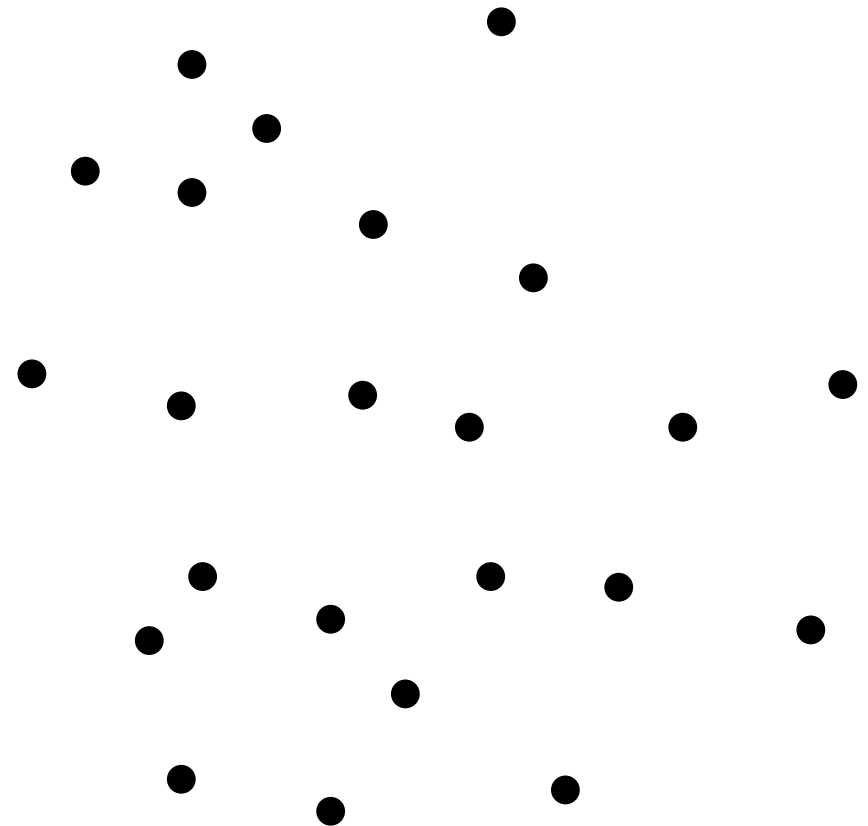
1. Find the extreme points in the horizontal and vertical directions.
2. Compute the convex hull of these (between 2 and 8) points.
3. Test all the remaining points, and classify them according to their position (NE, SE, SW, NW) or eliminate them if they lie in the interior.

# CONVEX HULL

## 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.

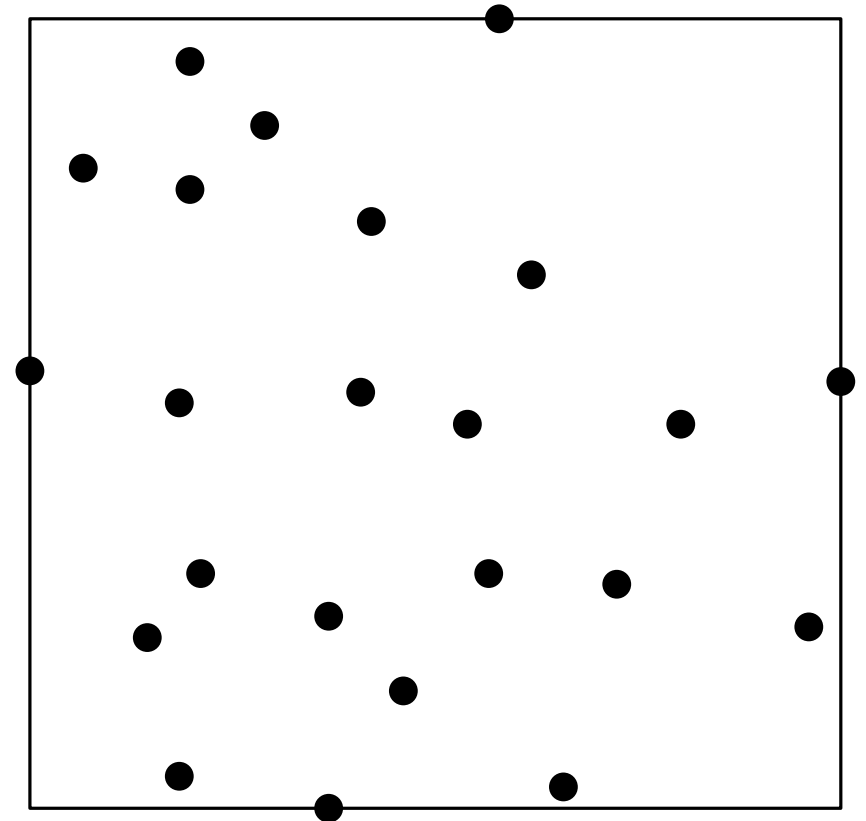


# CONVEX HULL

## 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.

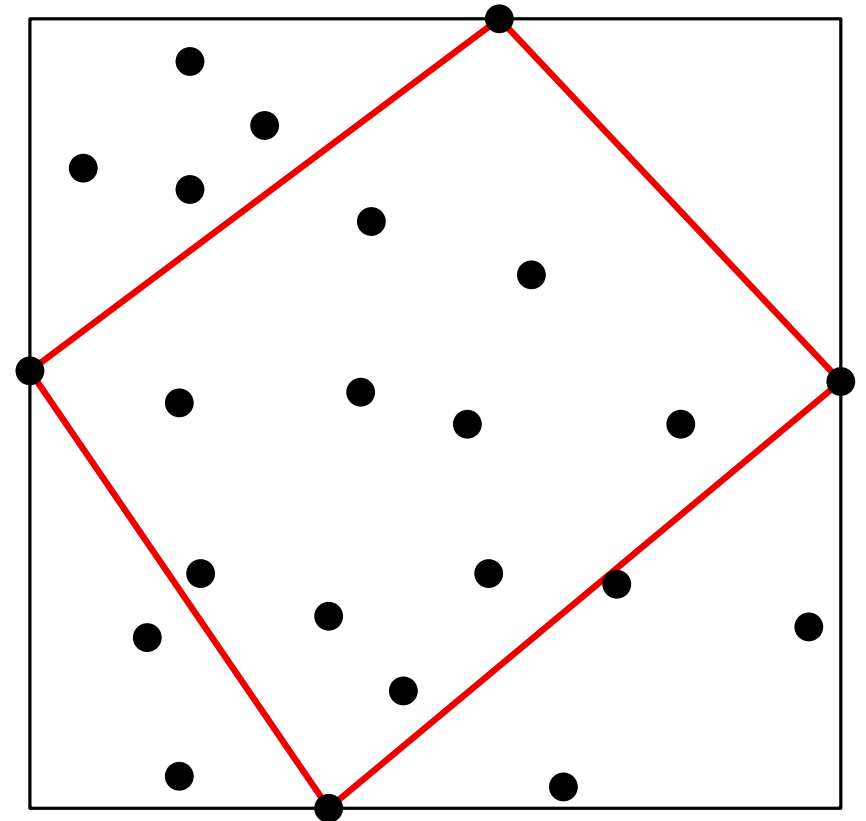


# CONVEX HULL

## 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.



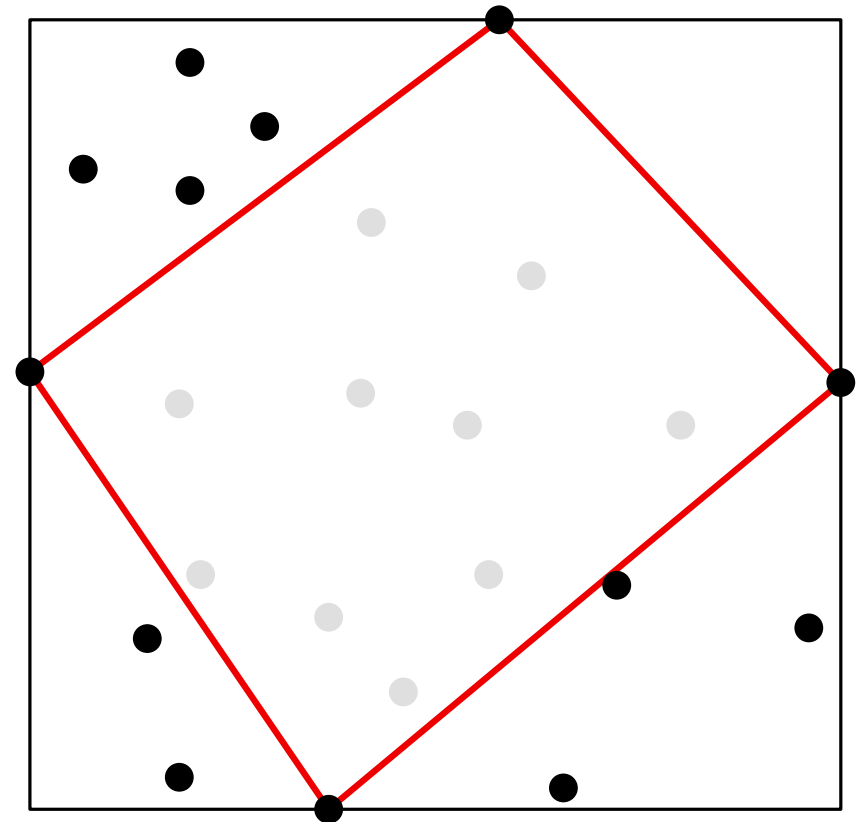


# CONVEX HULL

## 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.



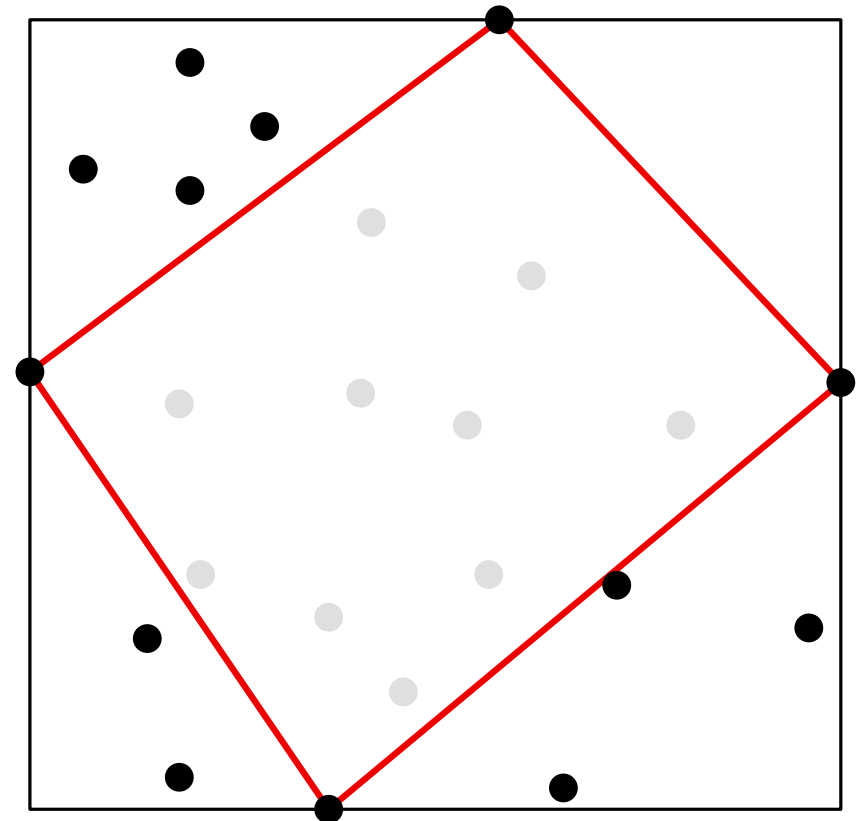
# CONVEX HULL

## 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:**  $O(n)$



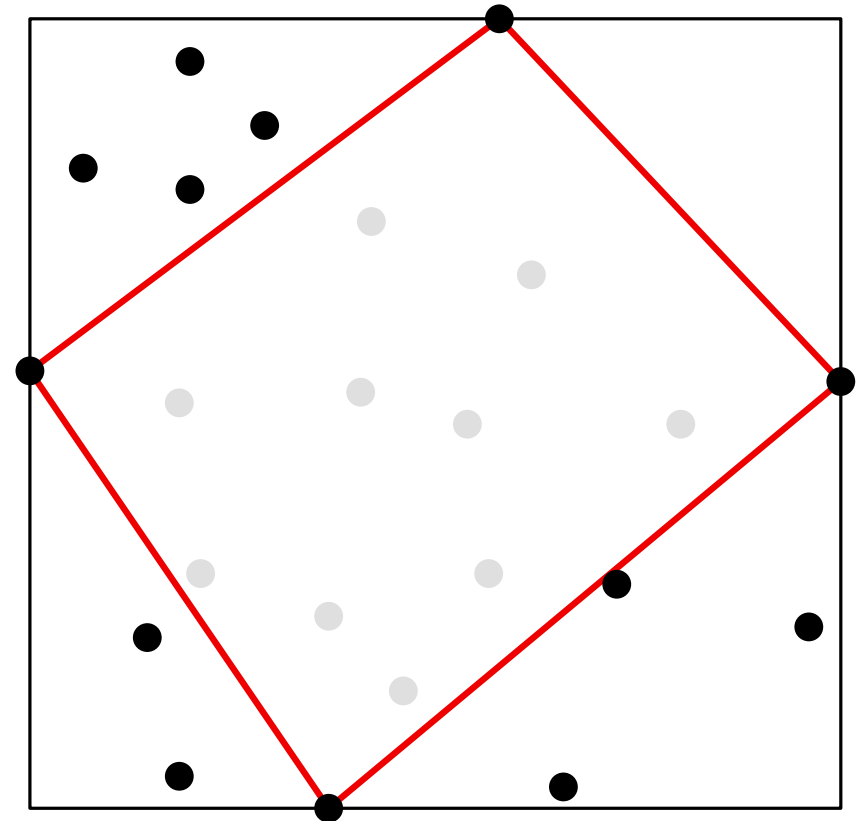
# CONVEX HULL

## 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 the 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.



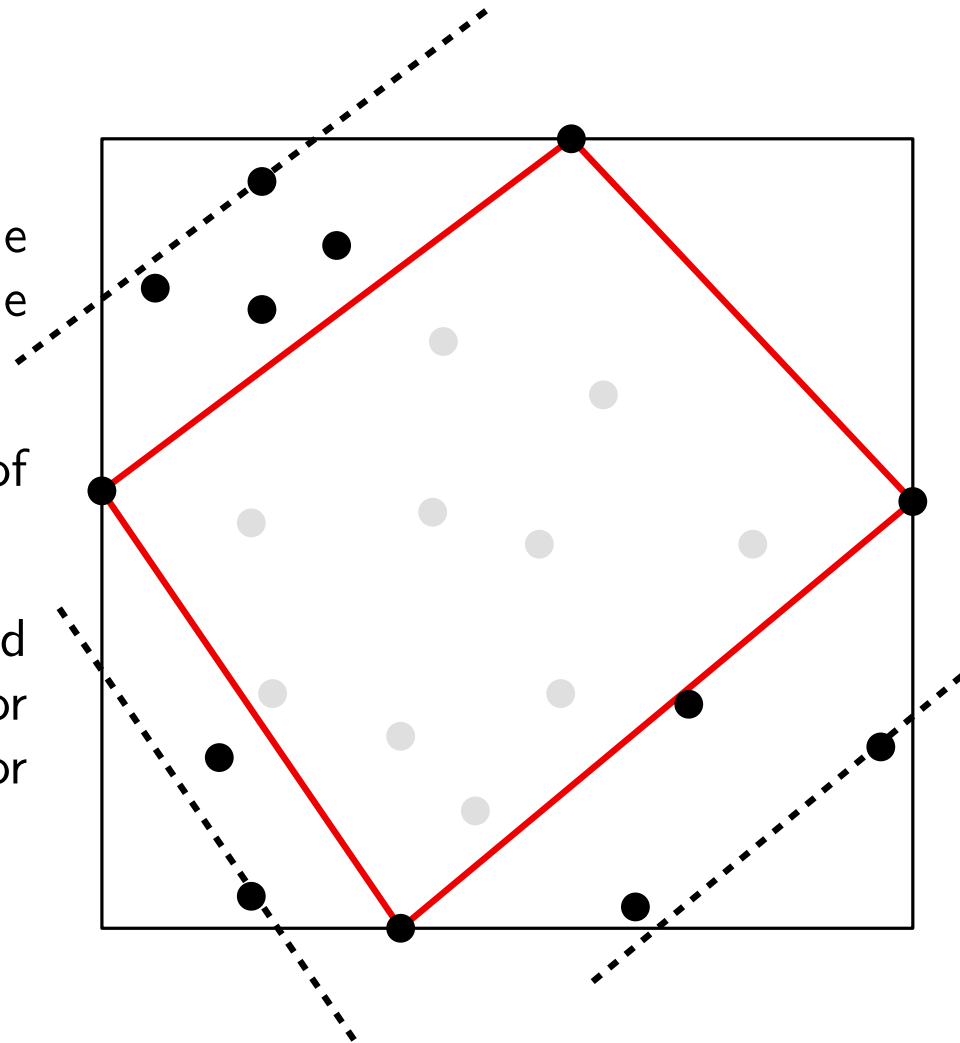
# CONVEX HULL

## 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 the 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.



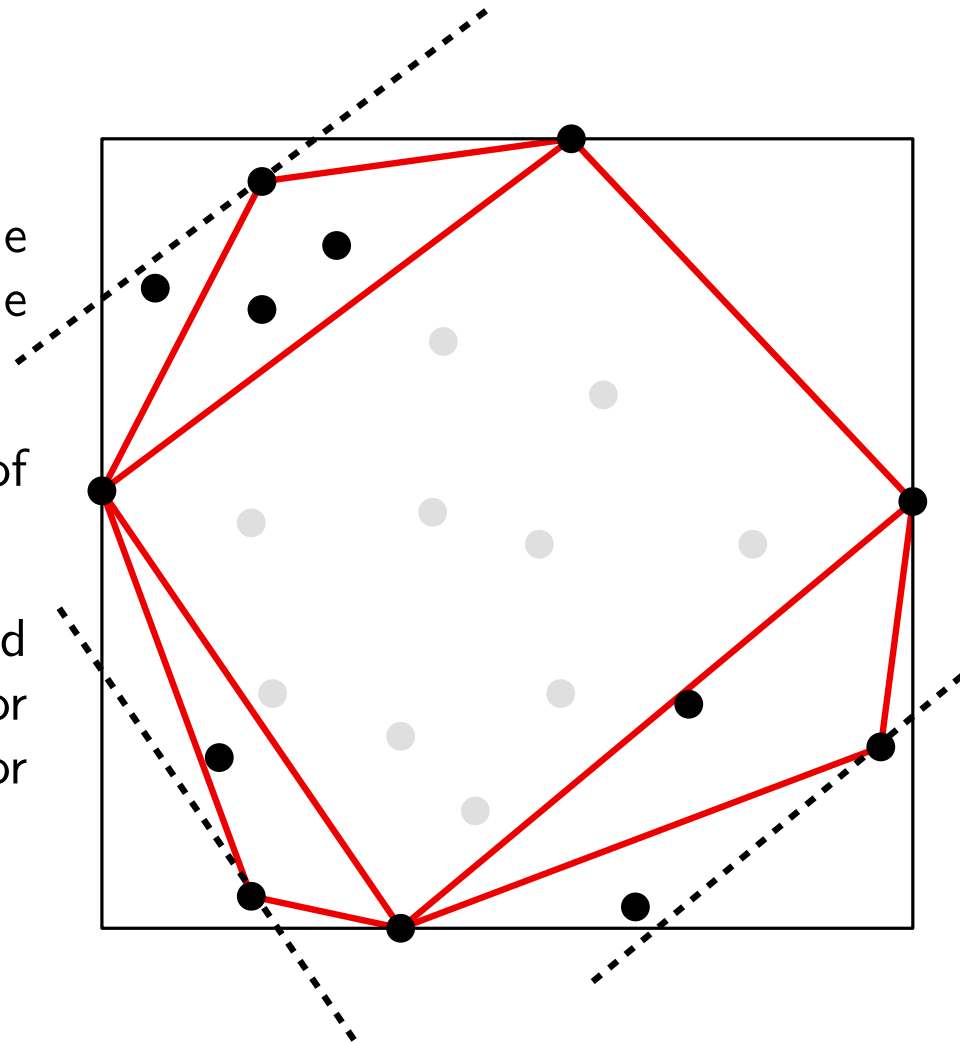
# CONVEX HULL

## 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 the 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.



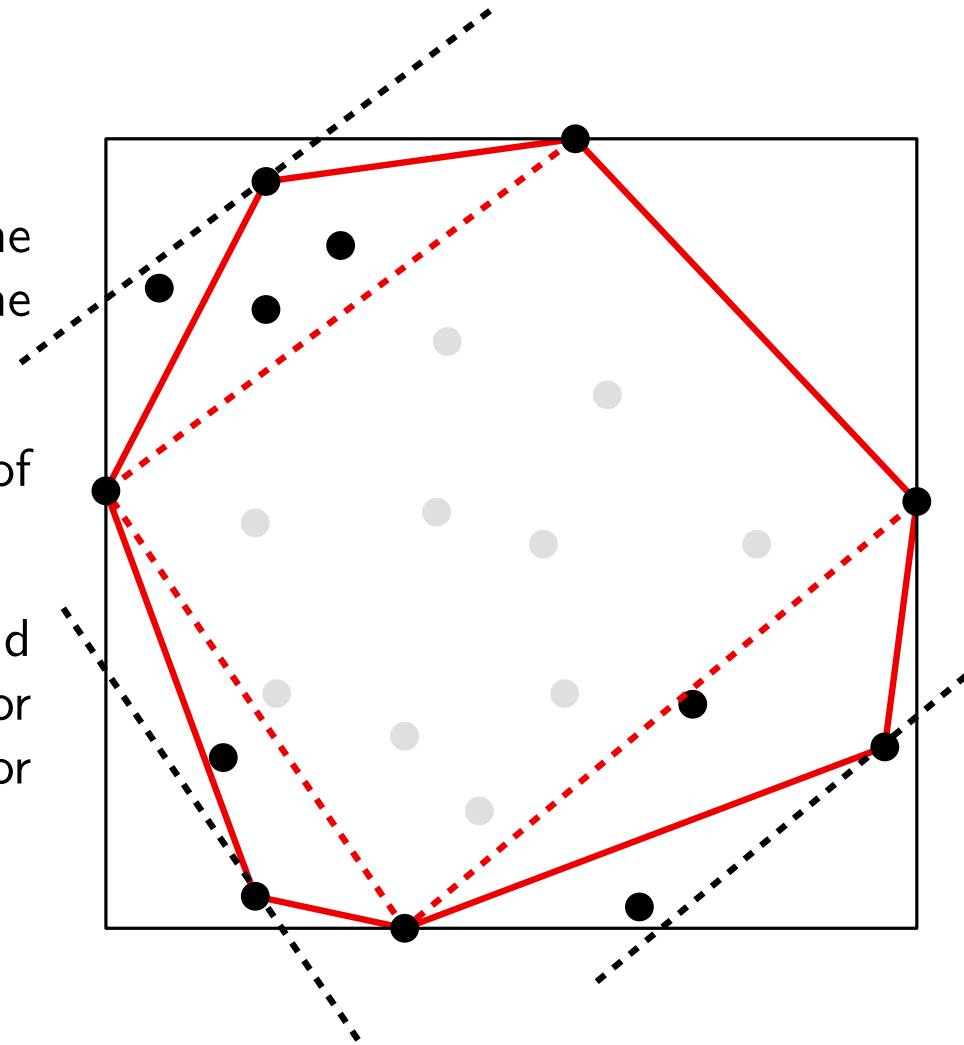
# CONVEX HULL

## 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 the 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.



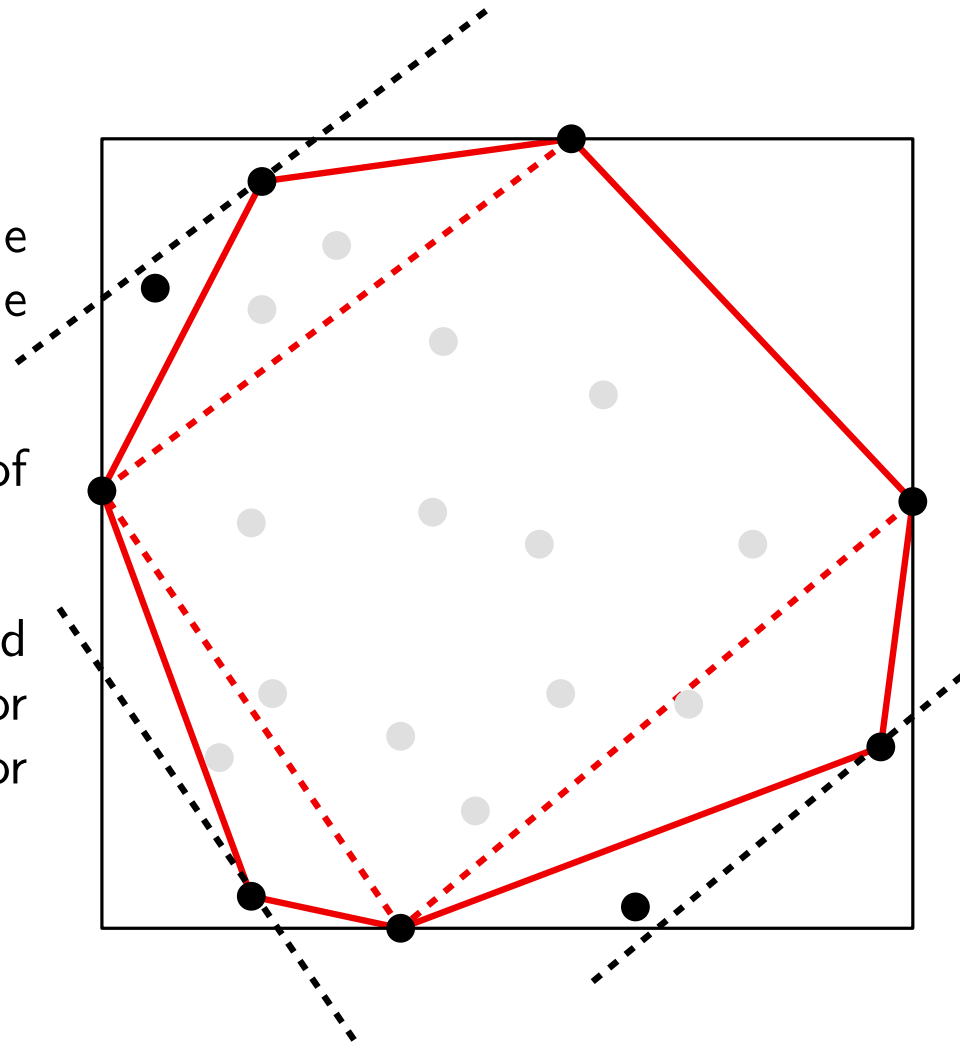
# CONVEX HULL

## 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 the 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.



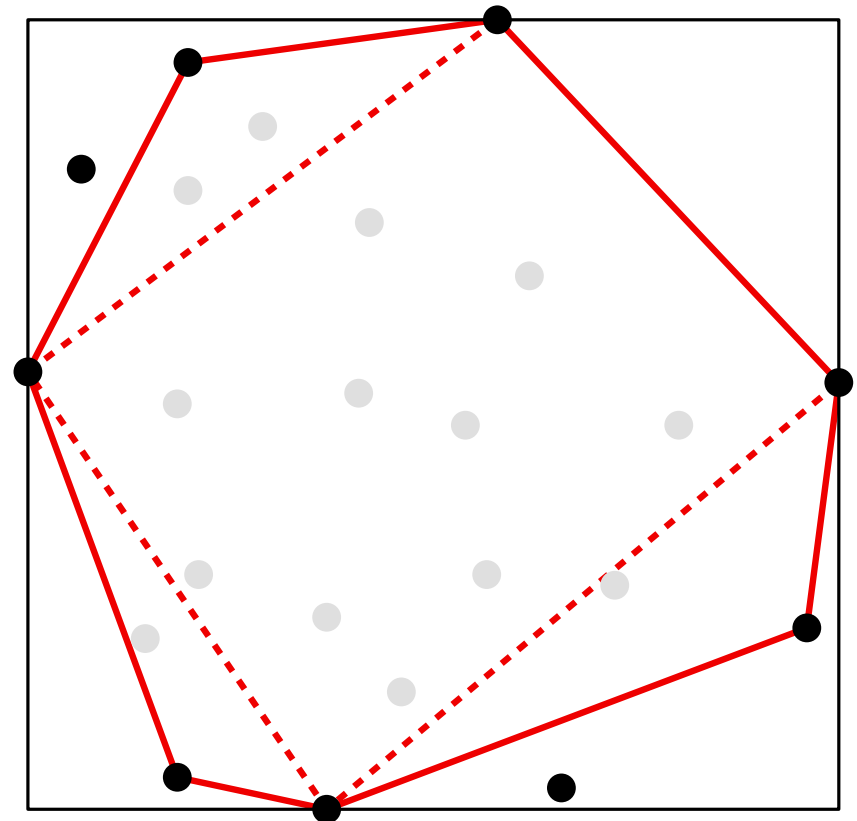
# CONVEX HULL

## 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 the 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.





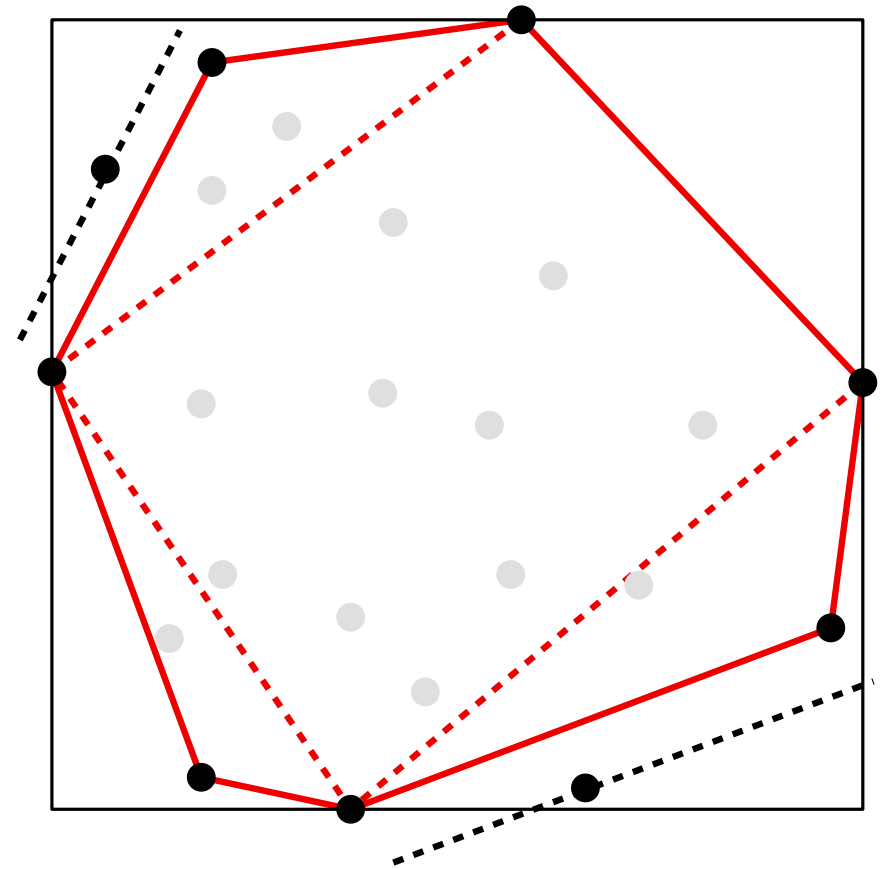
# CONVEX HULL

## 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 the 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.



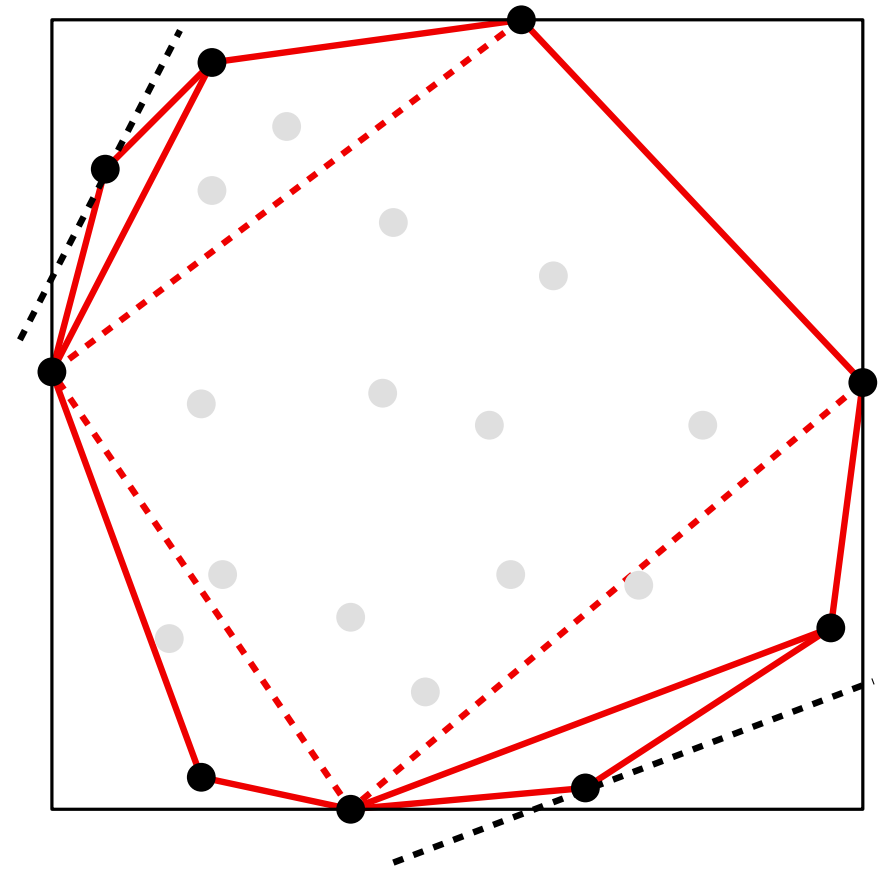
# CONVEX HULL

## 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 the 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.



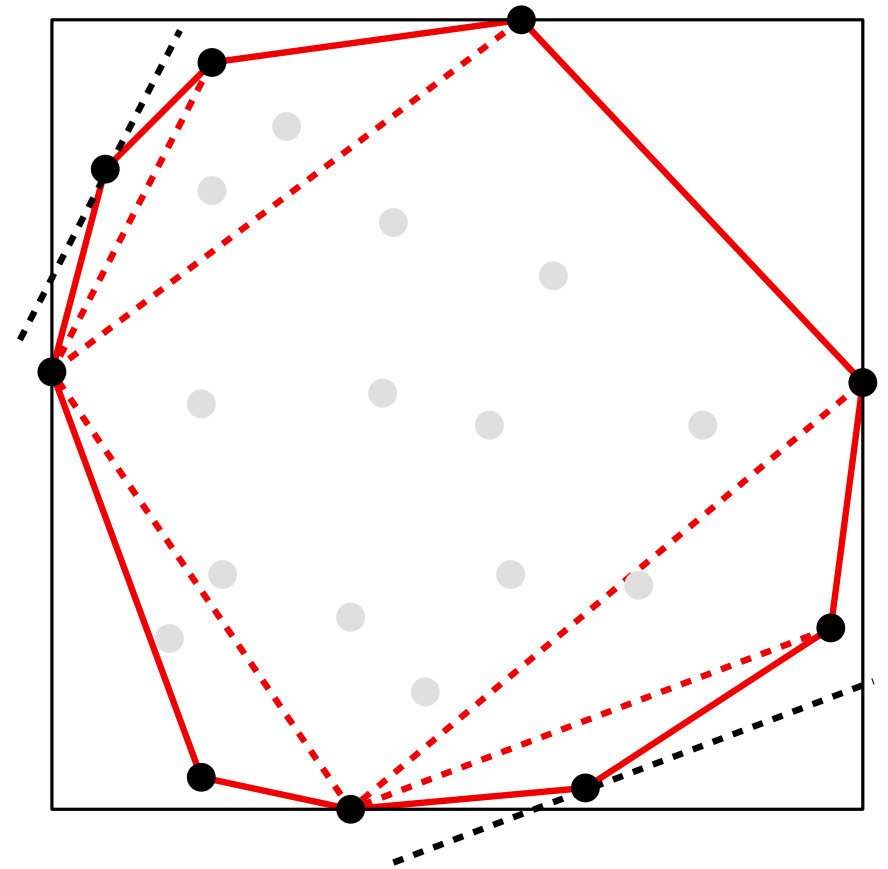
# CONVEX HULL

## 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 the 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.



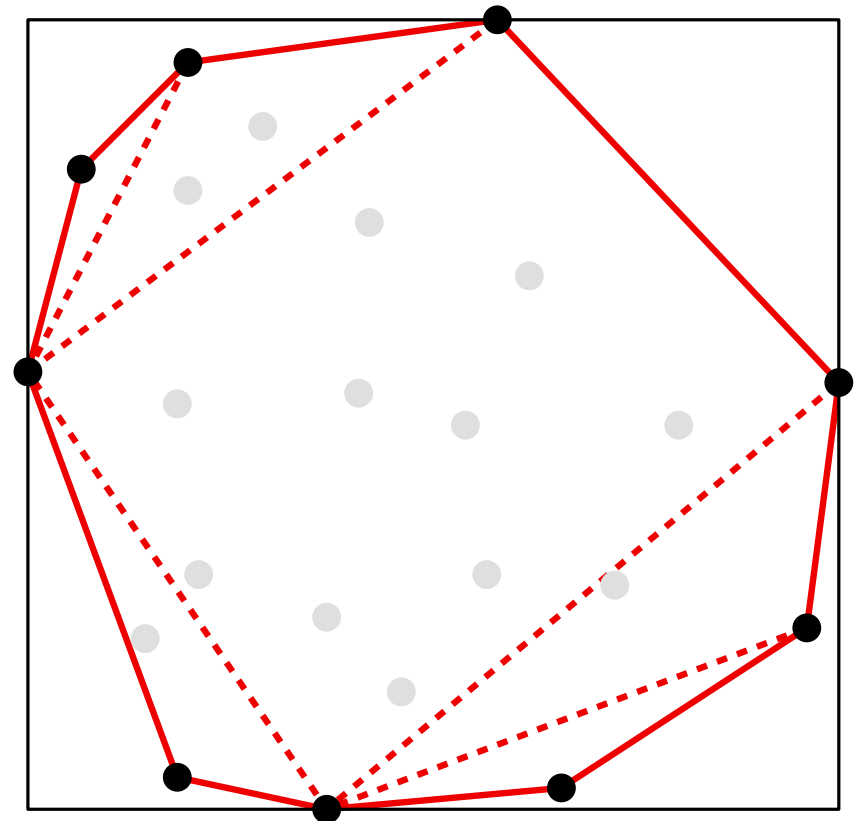
# CONVEX HULL

## 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 the 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.



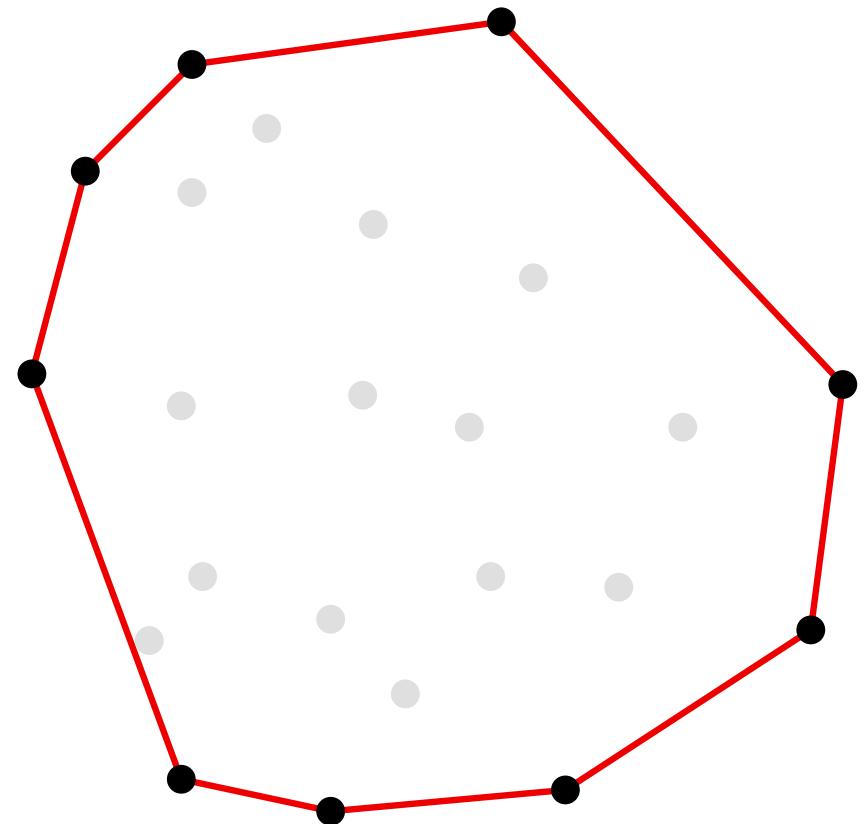
# CONVEX HULL

## 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 the 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

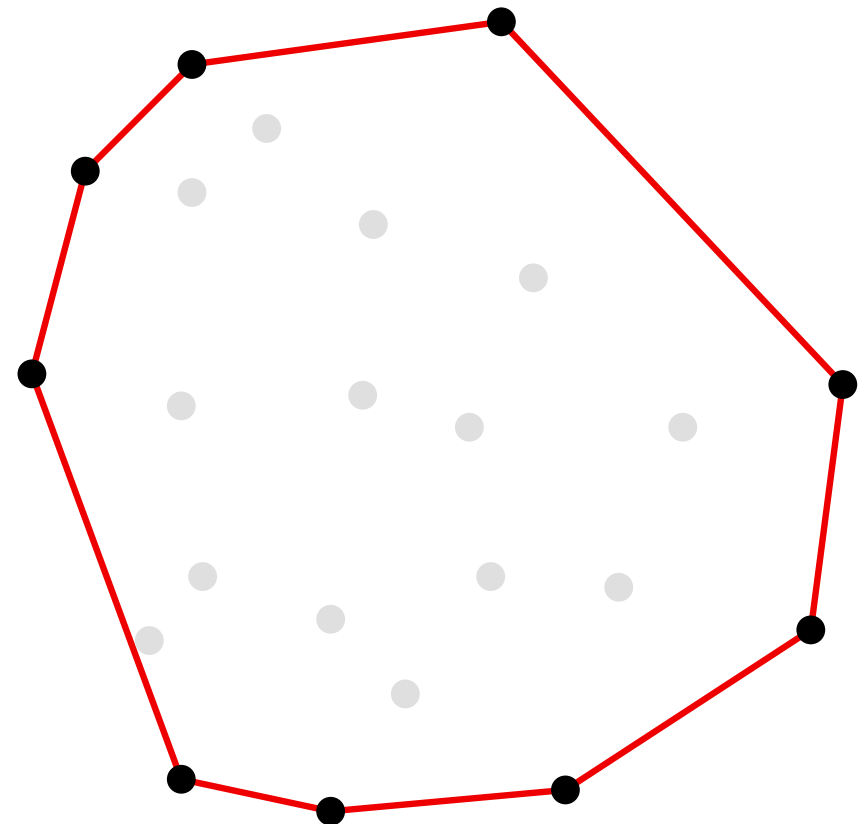
## QuickHull algorithm (by prune-and-search)

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 the 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.

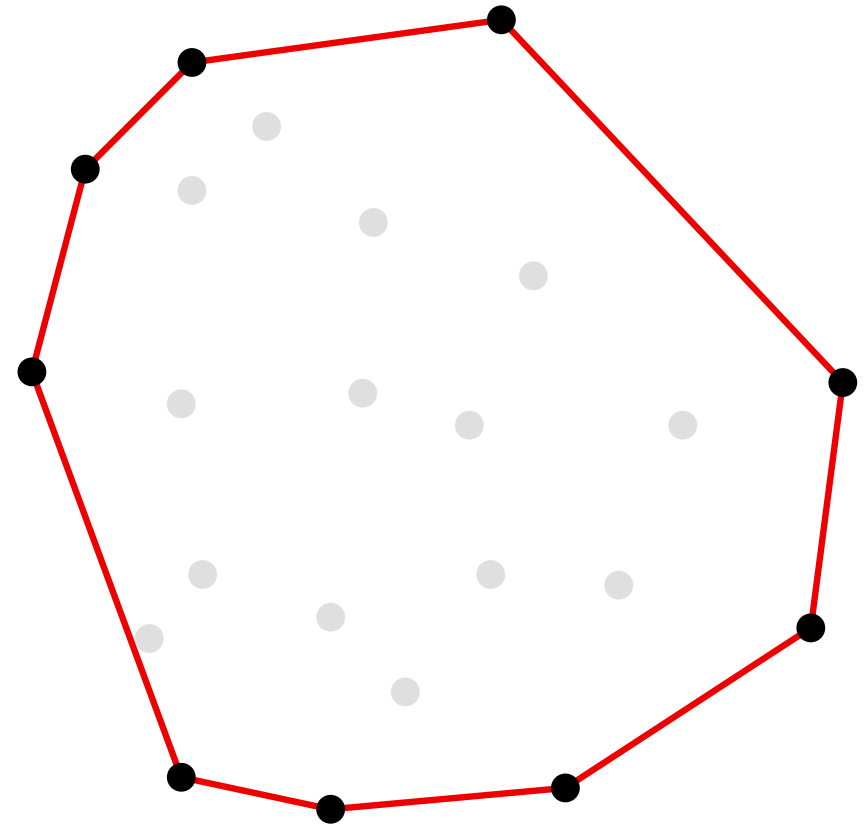
**Running time of this step:**  $O(n^2)$



# CONVEX HULL

QuickHull algorithm (by prune-and-search)

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



# CONVEX HULL

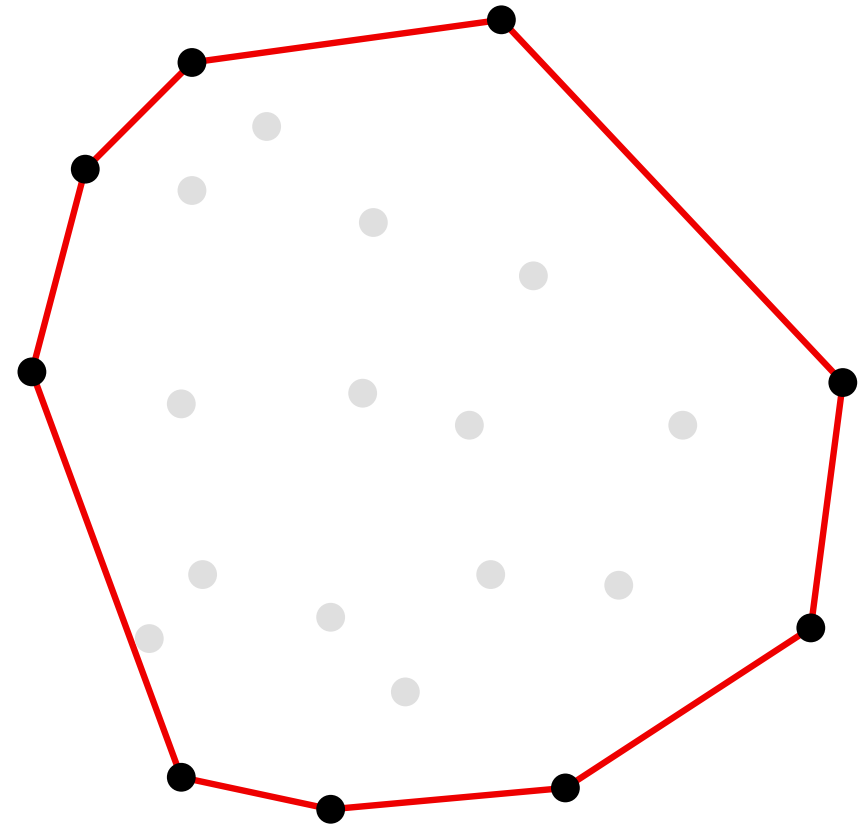
## QuickHull algorithm (by prune-and-search)

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.





# CONVEX HULL

## Graham's algorithm

# CONVEX HULL

## 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 it 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

## 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 it 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:

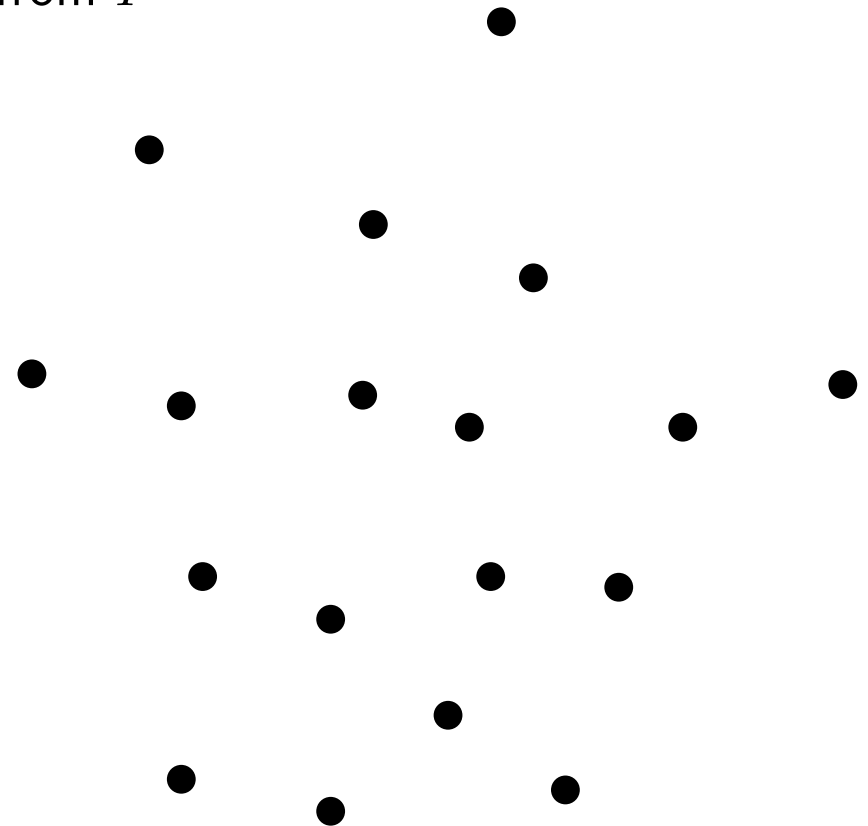
- Push  $p_i$  in  $l$

- Advance  $i$

- Else:

- Pop  $p$  from  $l$

Return  $l$



# CONVEX HULL

## 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 it 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:

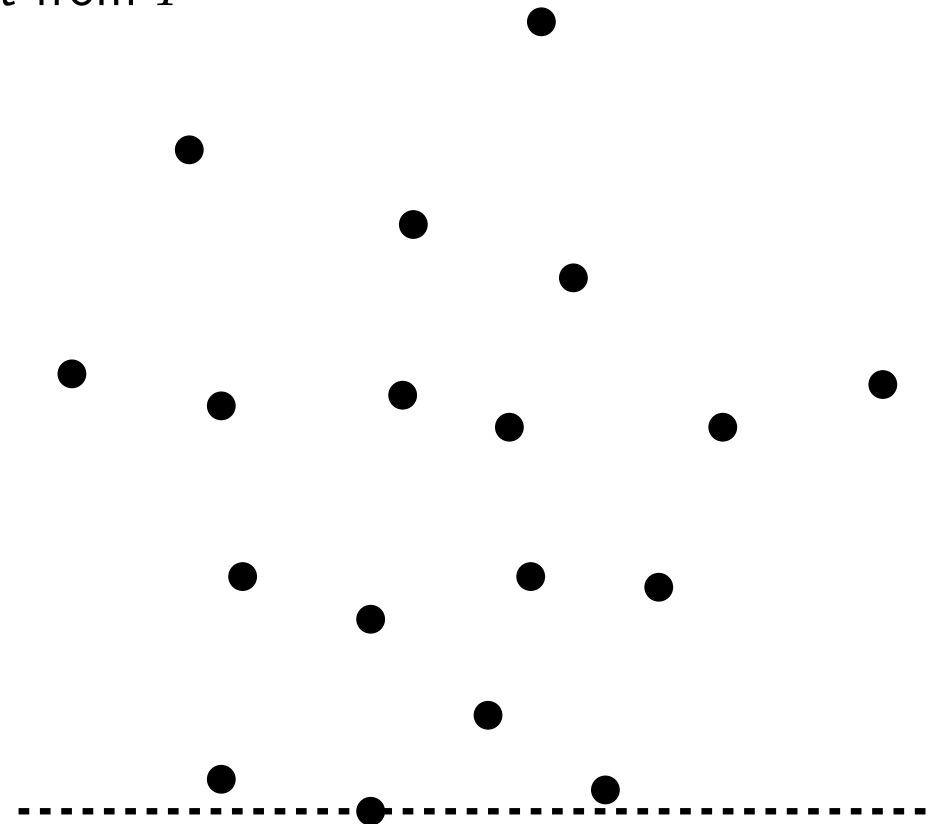
- Push  $p_i$  in  $l$

- Advance  $i$

- Else:

- Pop  $p$  from  $l$

Return  $l$



# CONVEX HULL

## 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 it from  $P$

### Advance

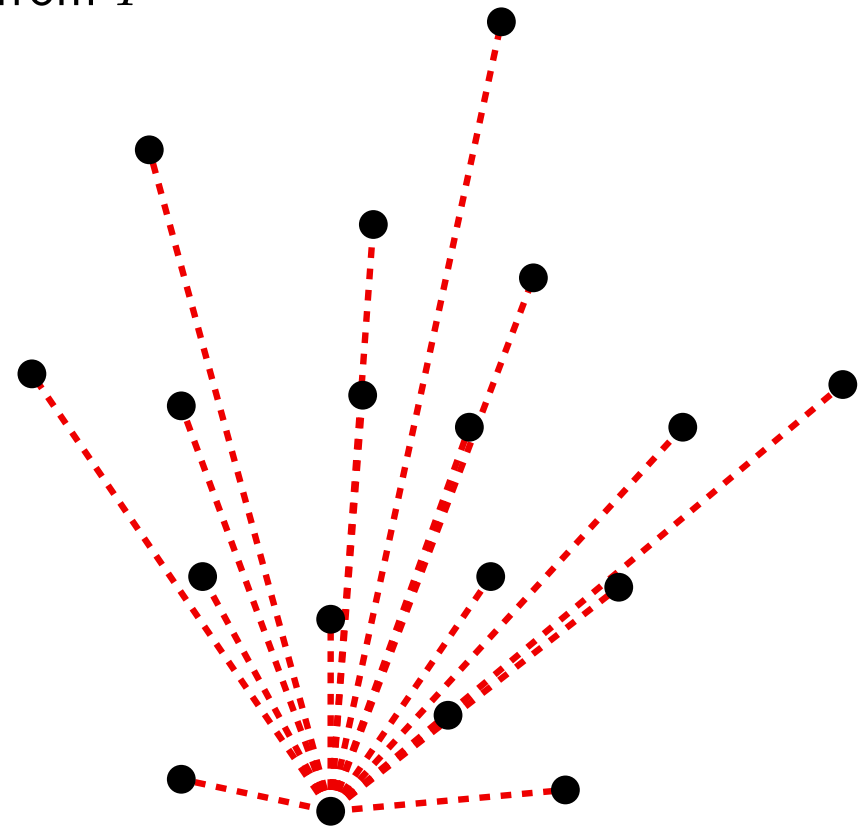
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

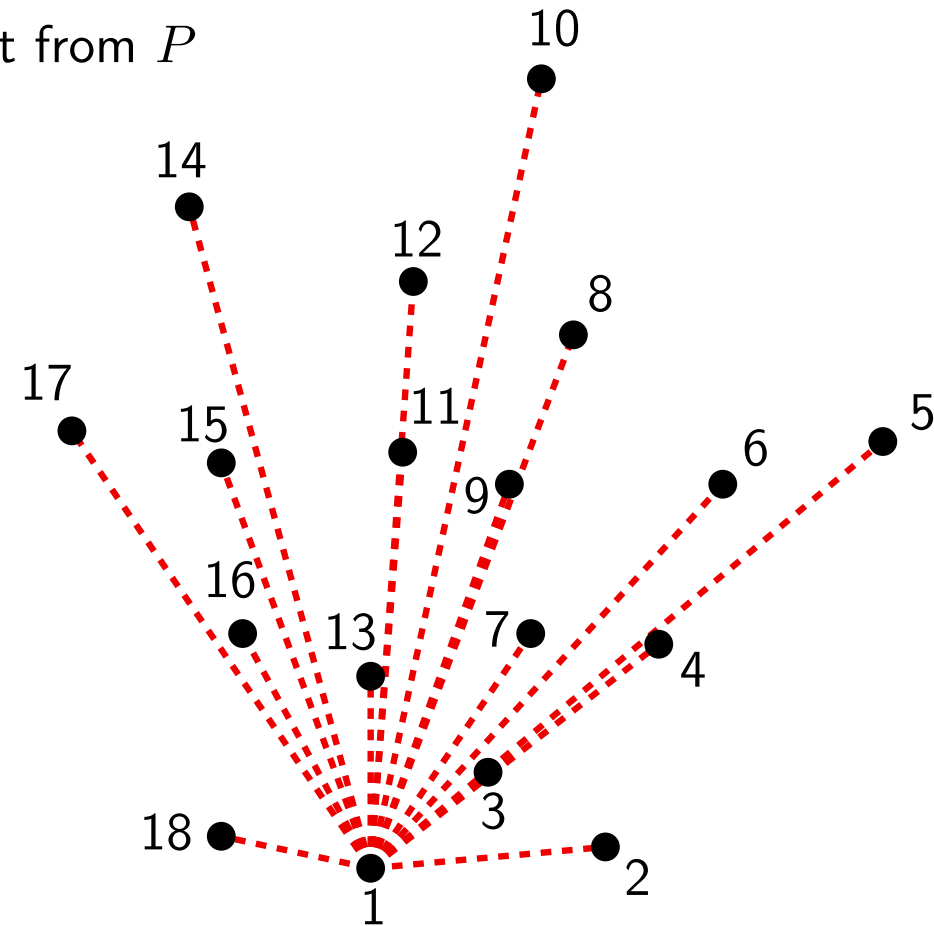
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

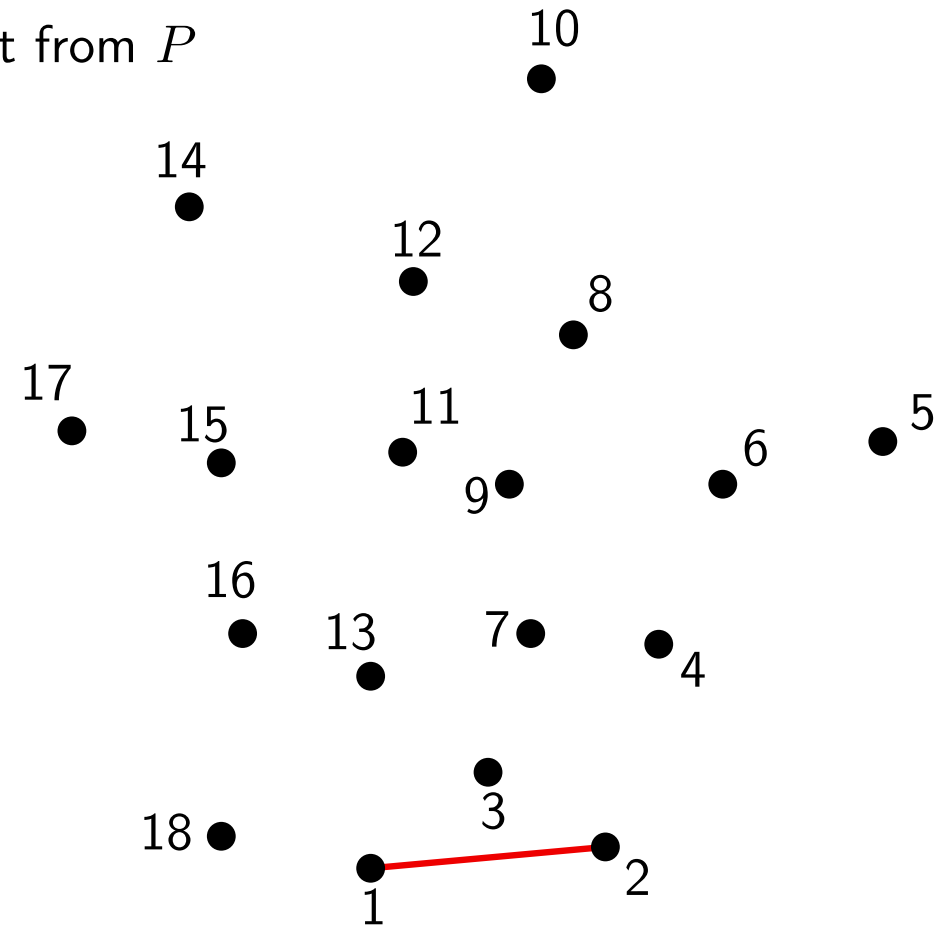
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

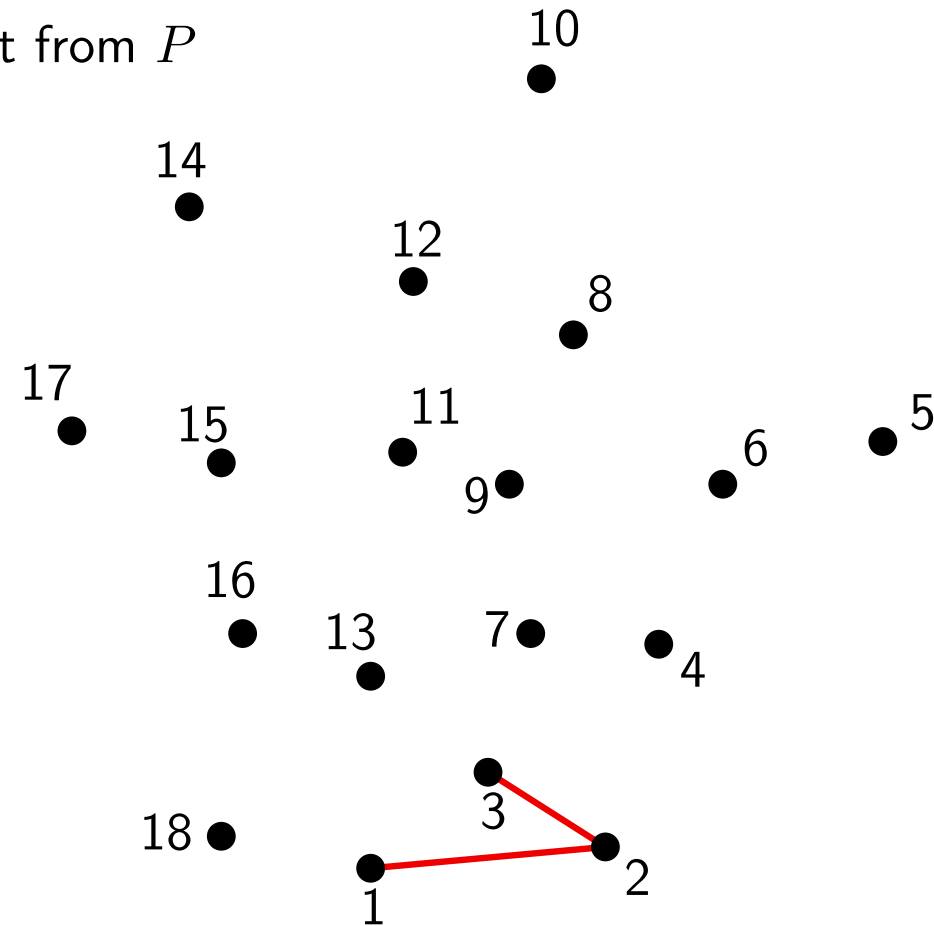
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

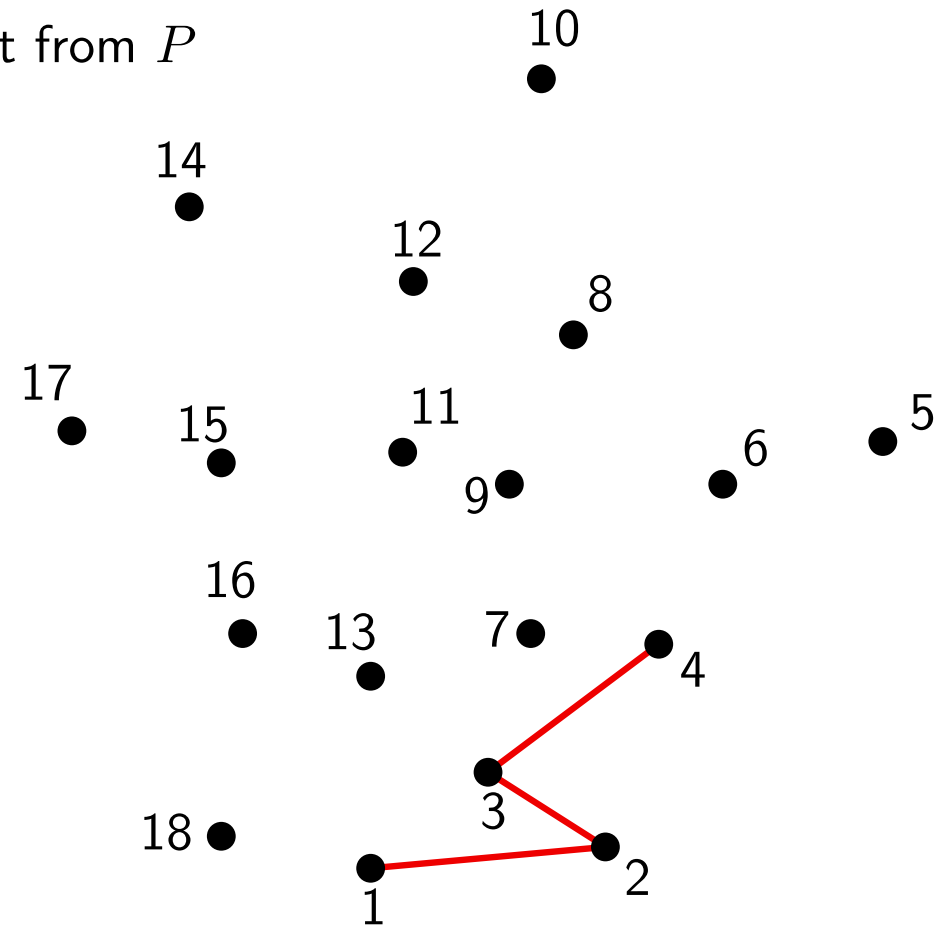
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

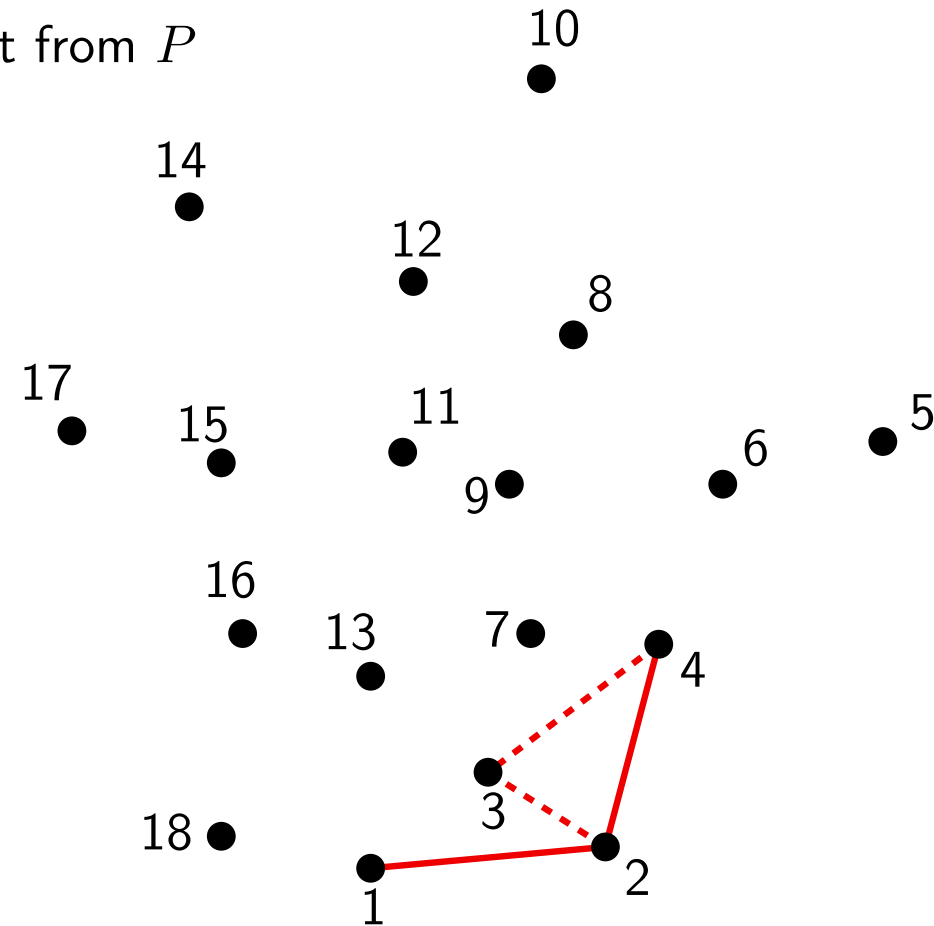
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

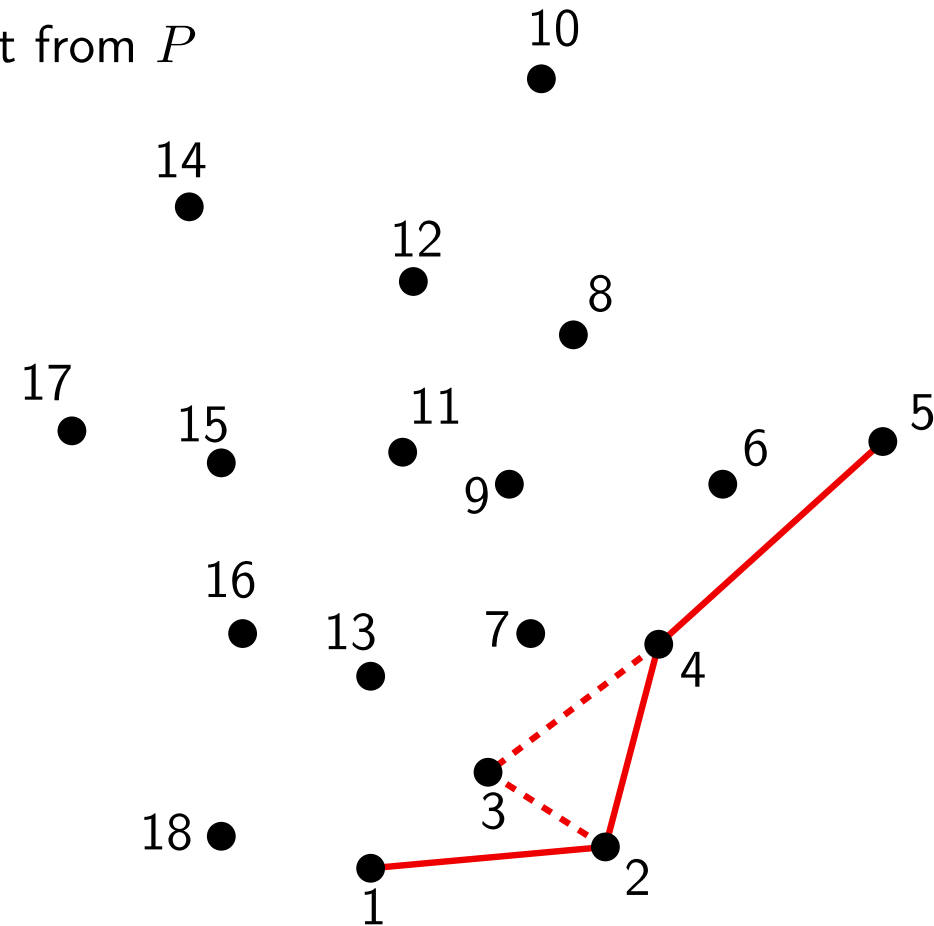
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

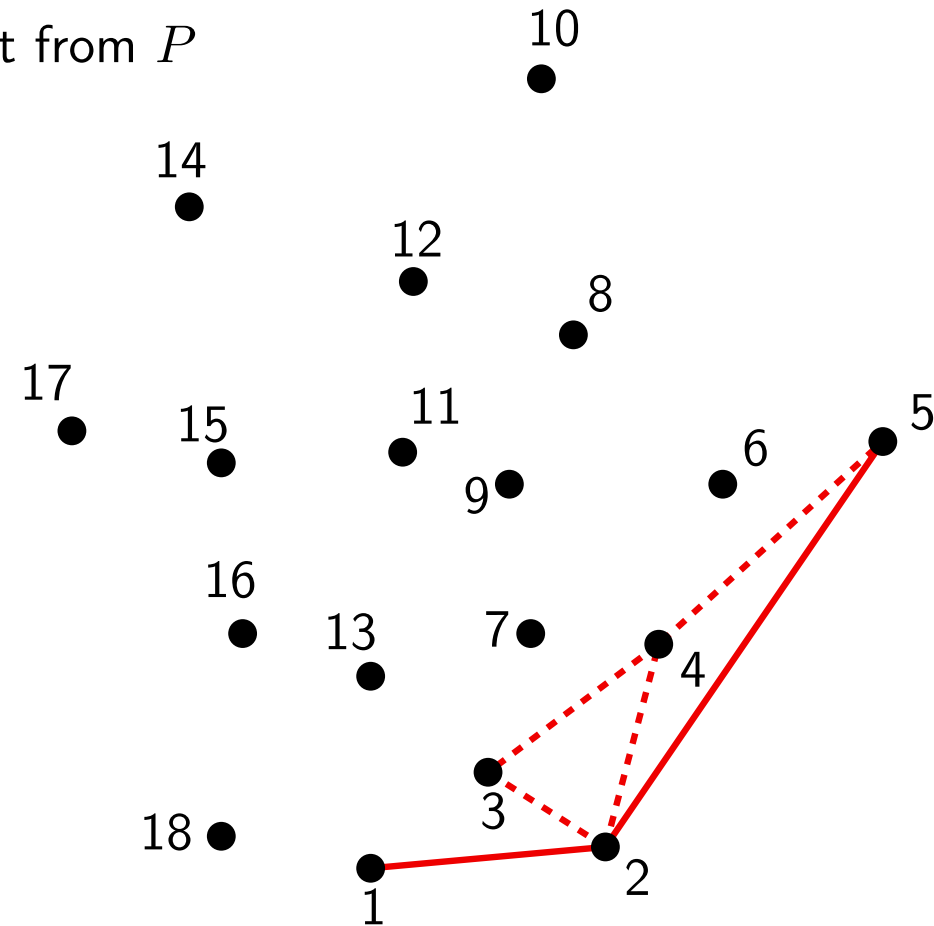
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

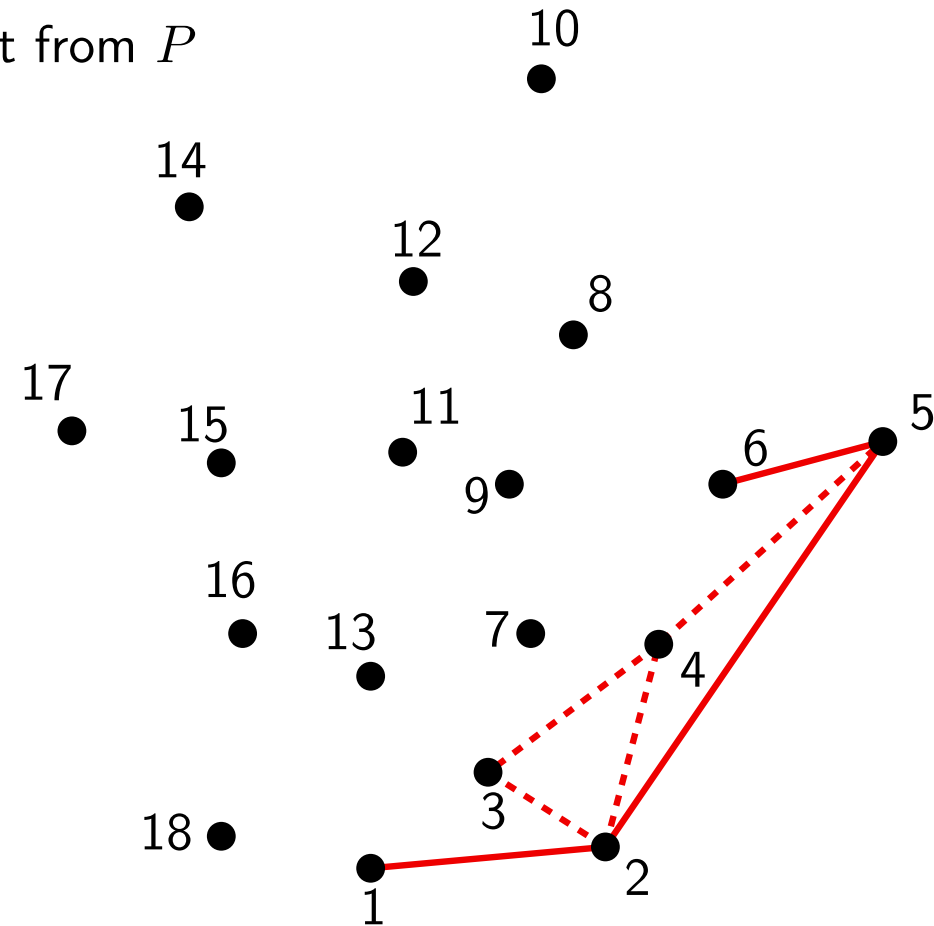
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

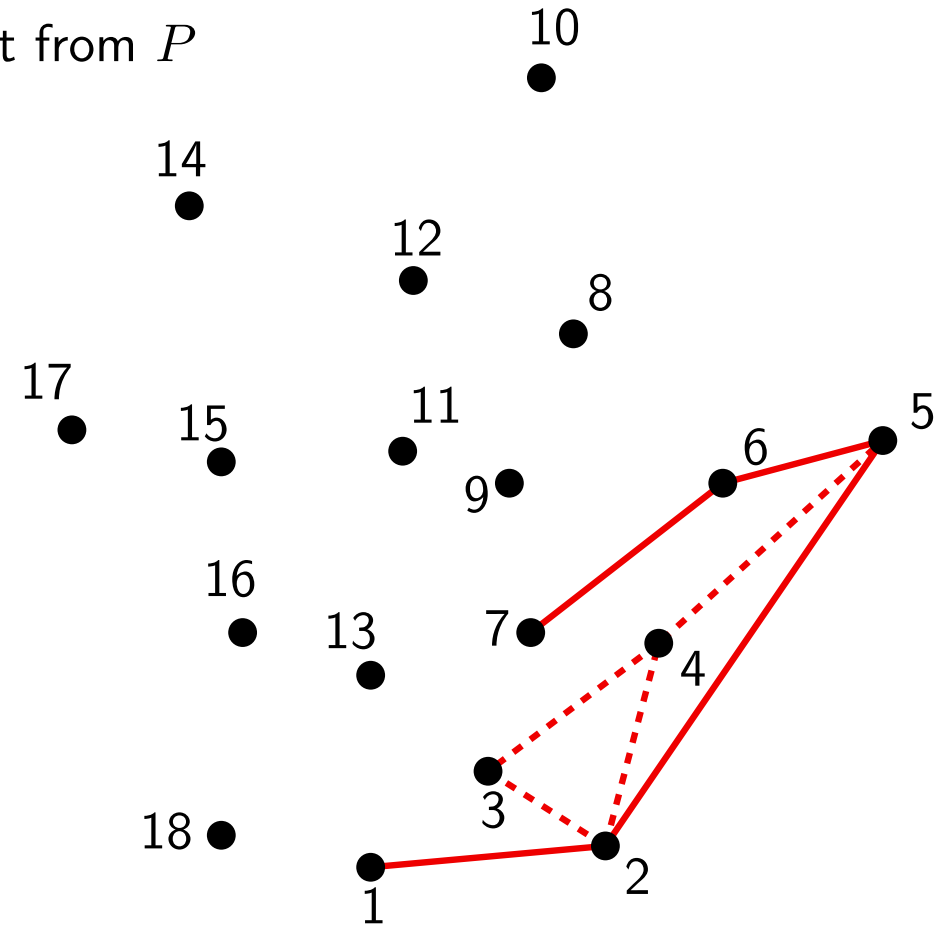
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

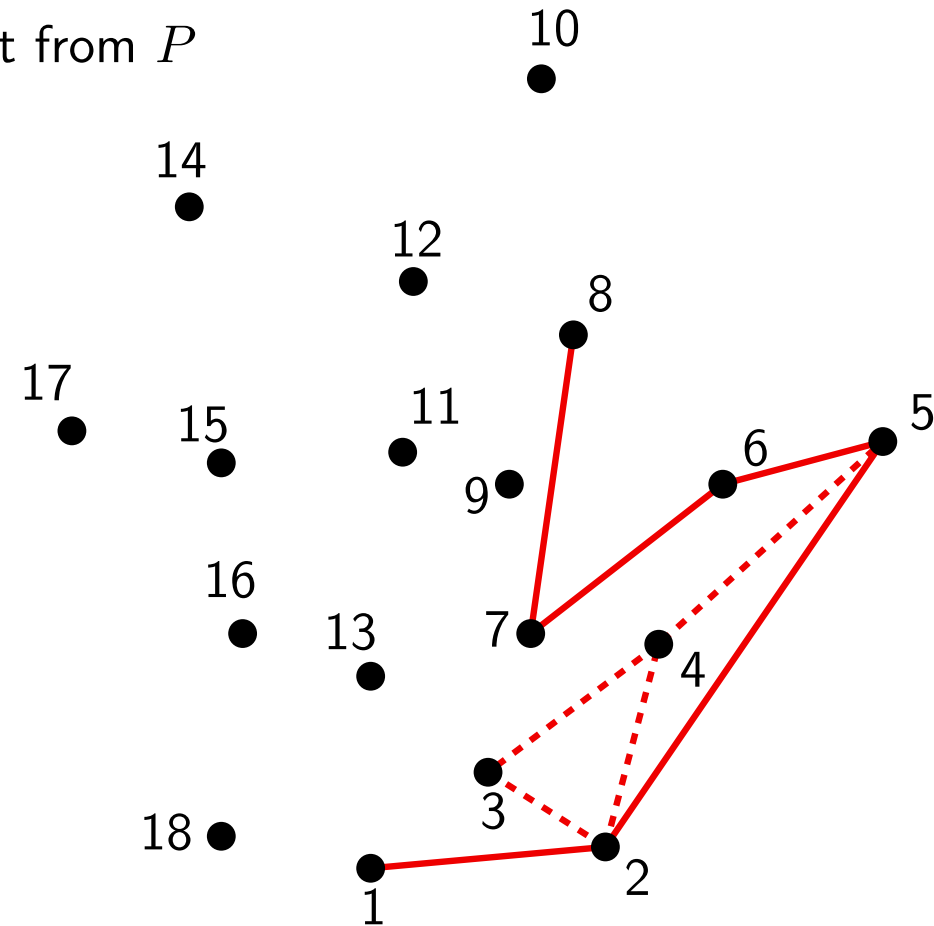
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

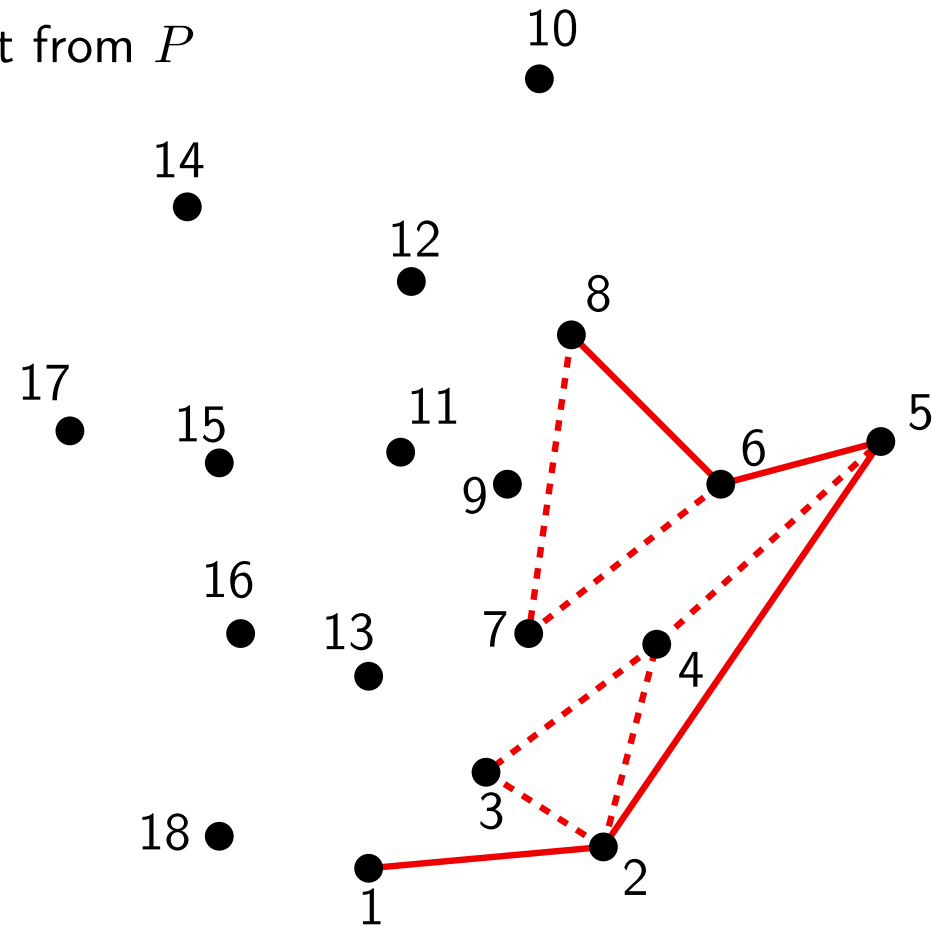
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

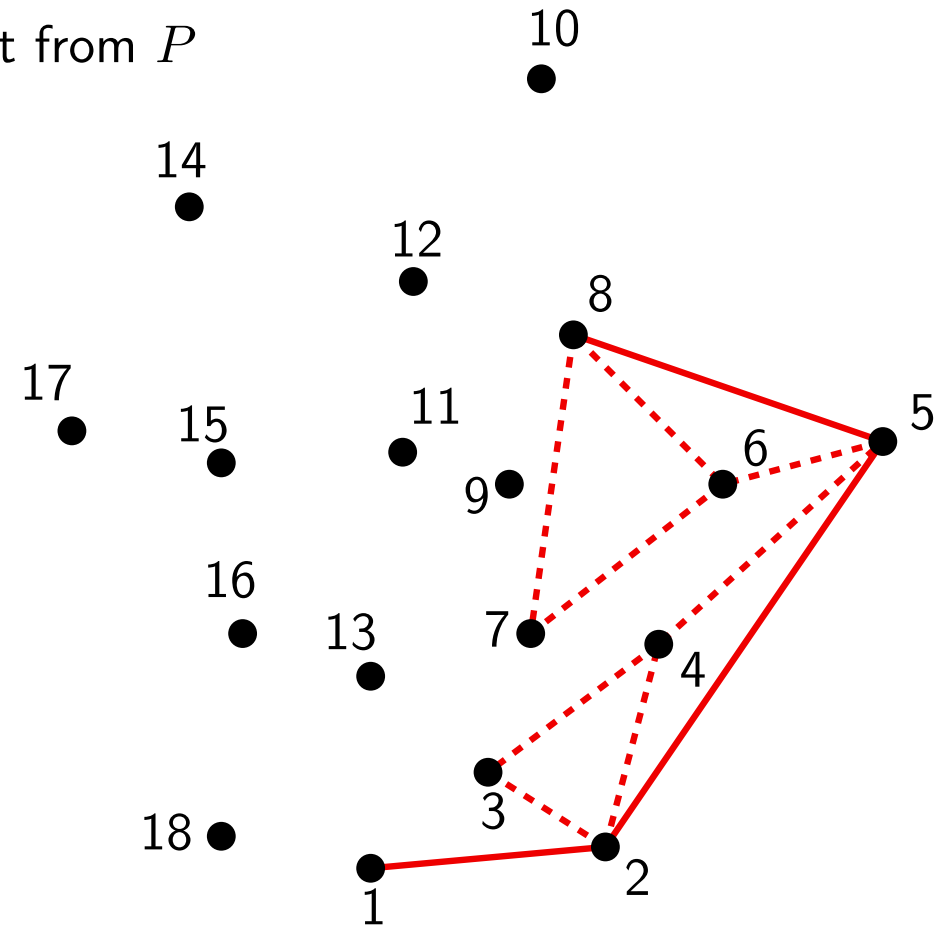
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

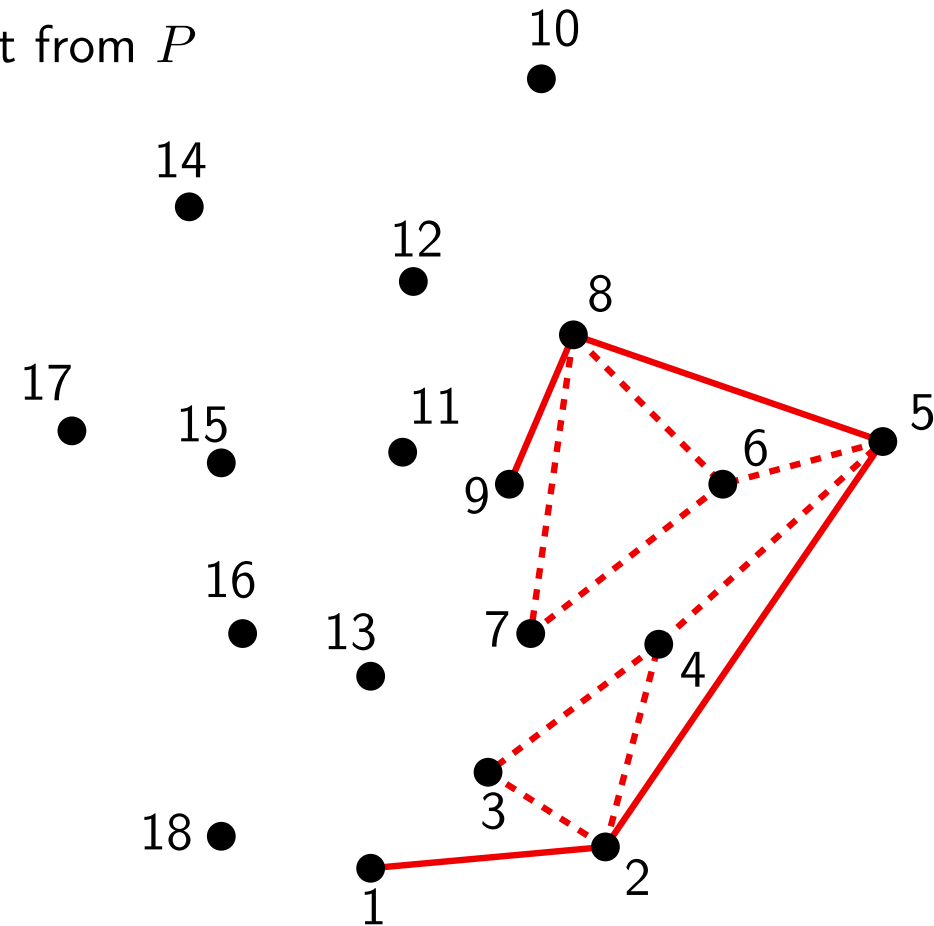
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

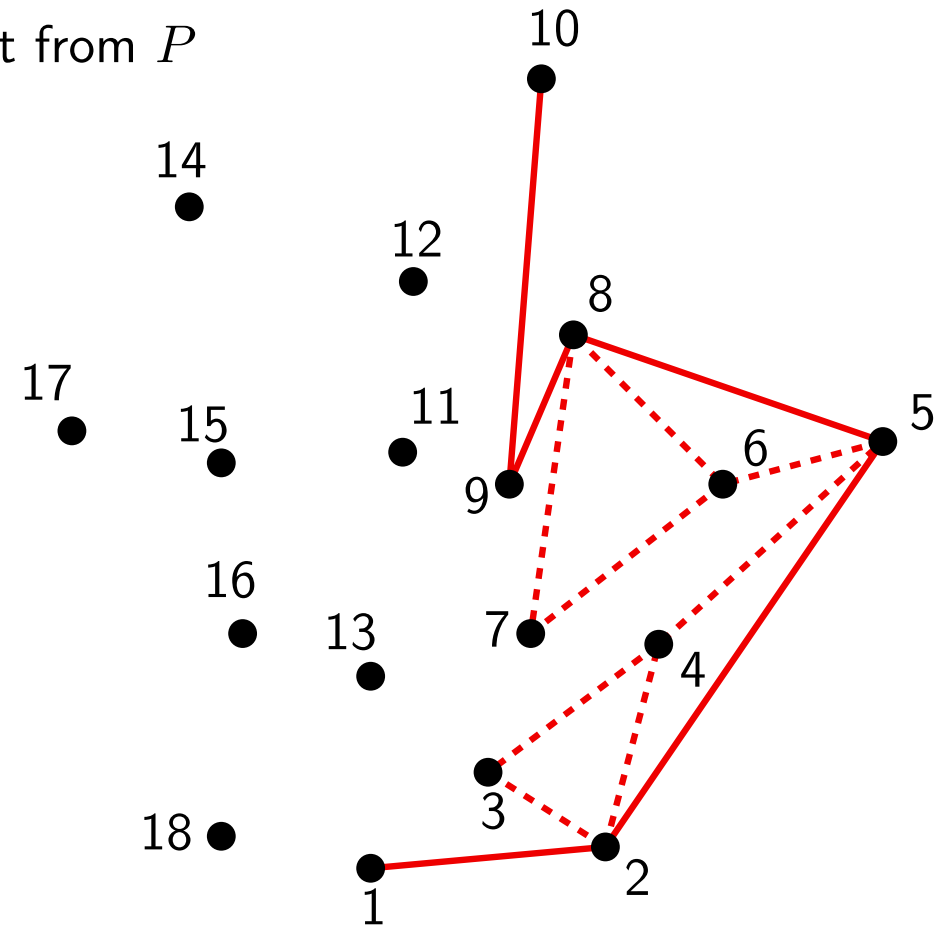
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

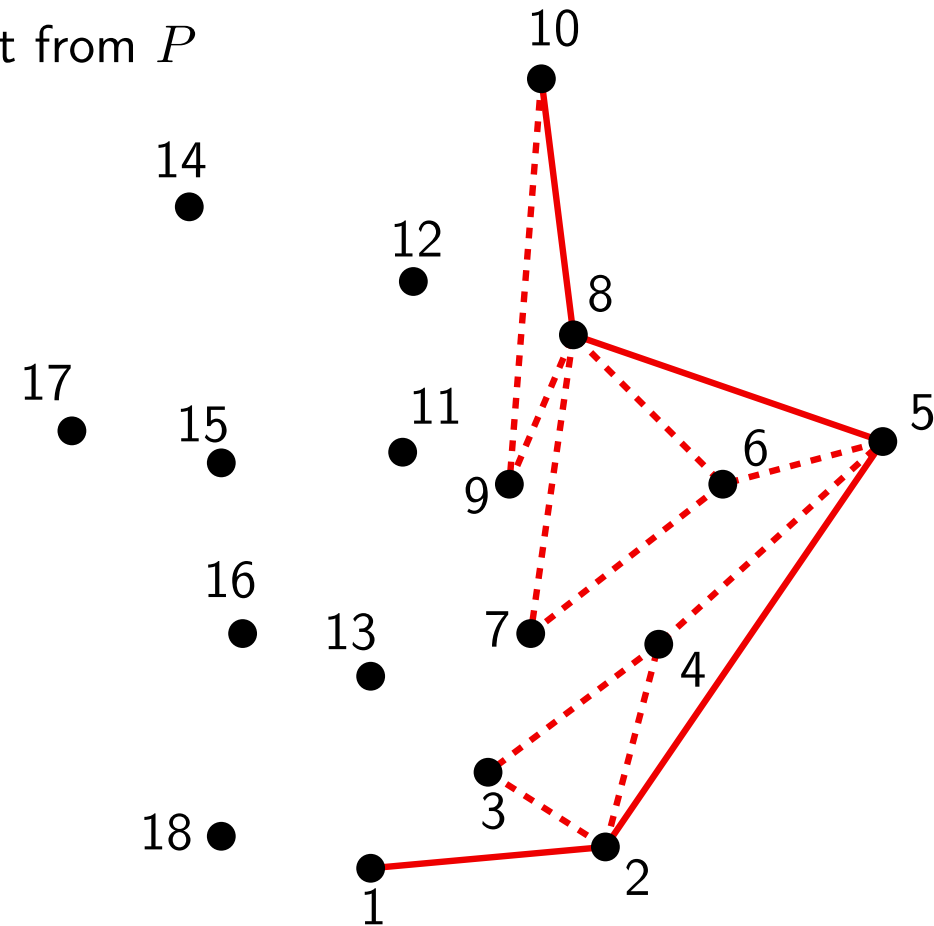
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

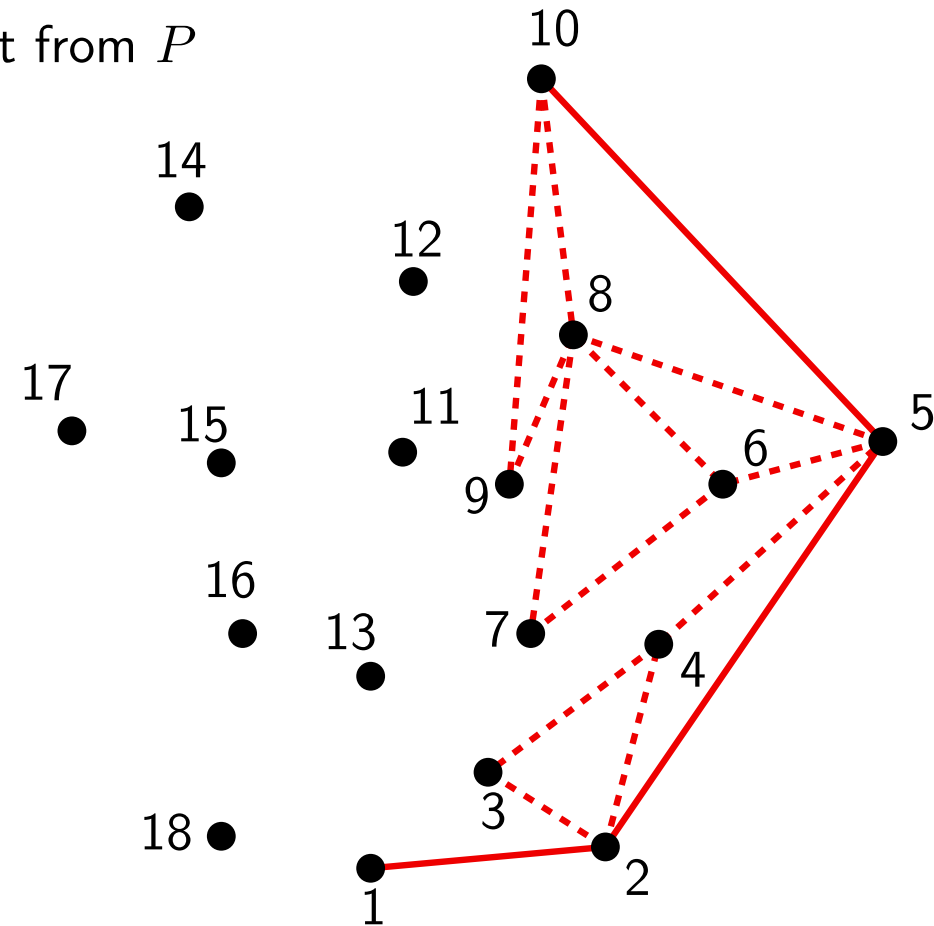
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

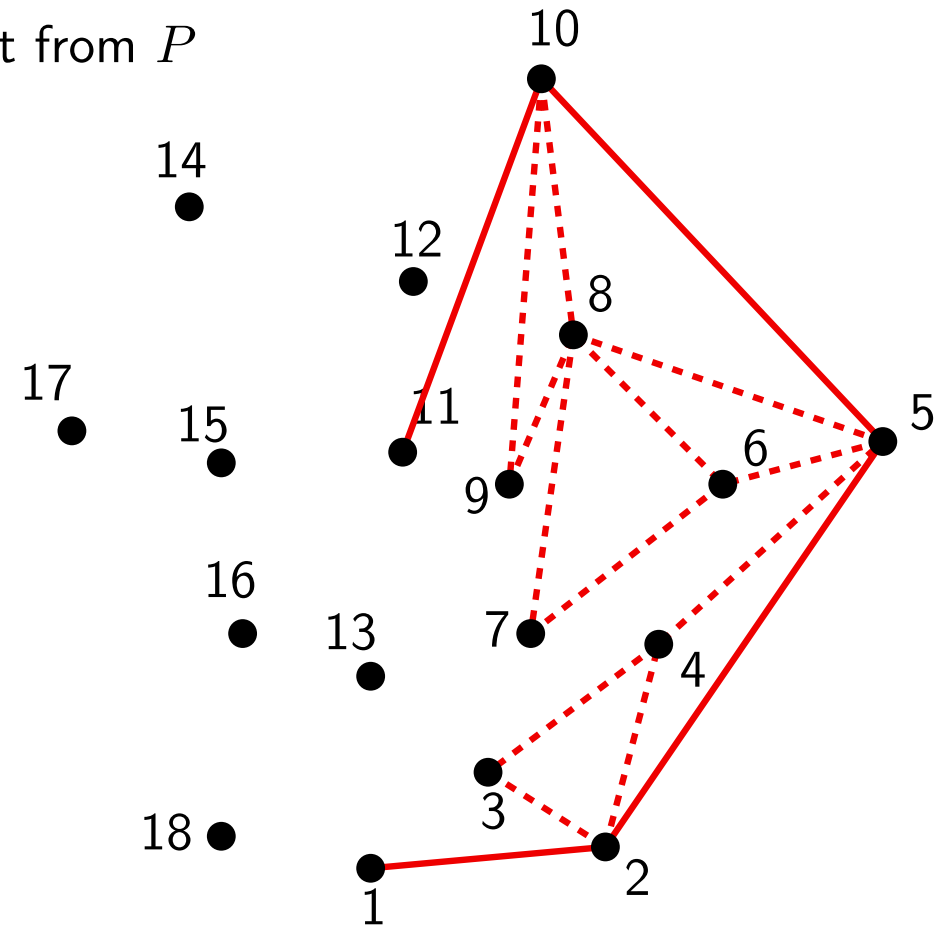
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

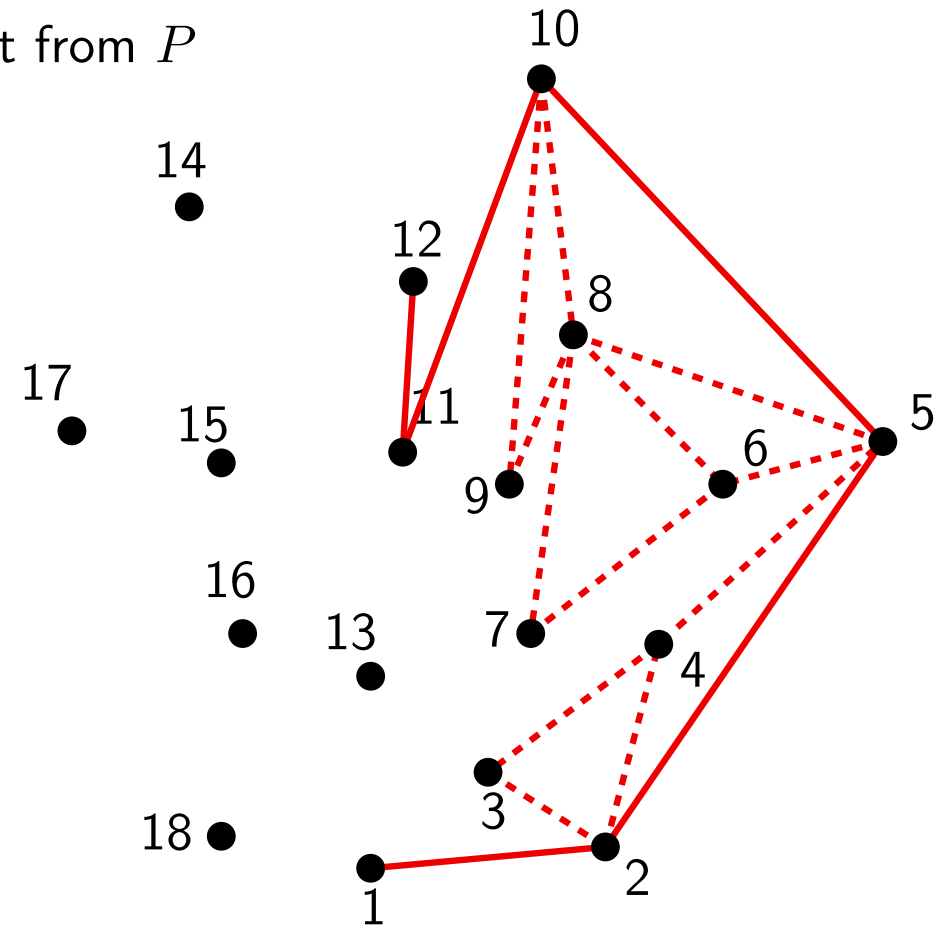
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

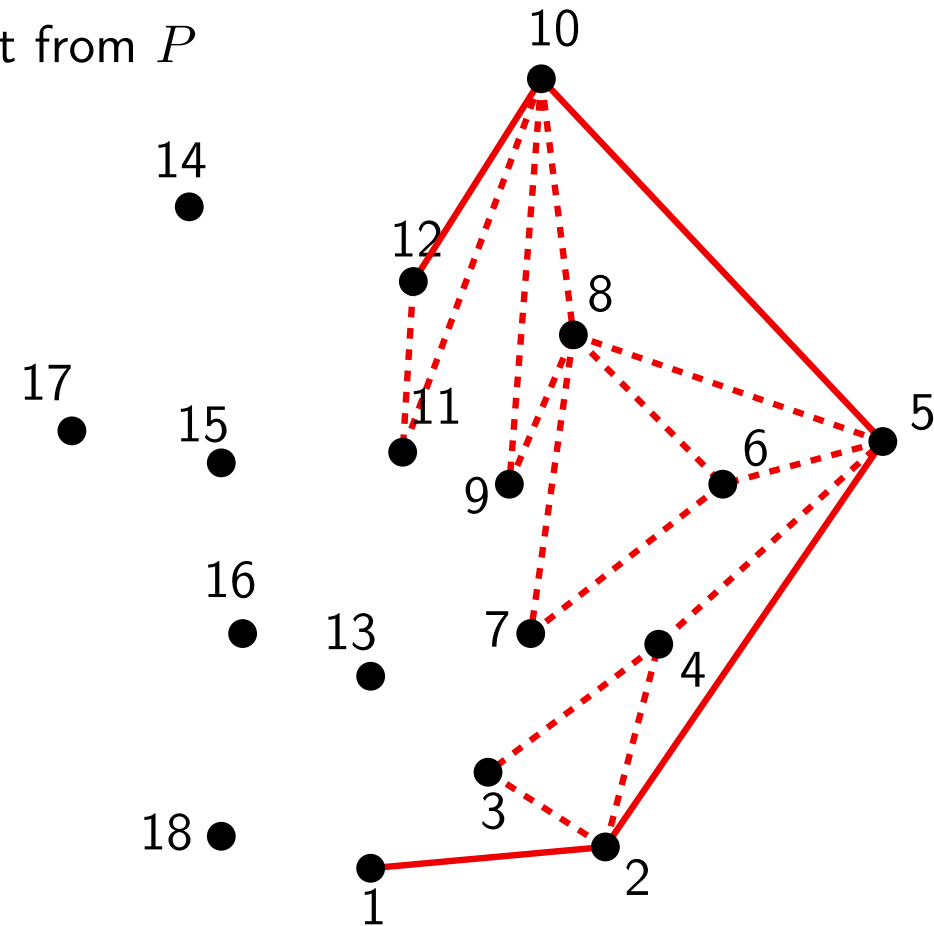
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

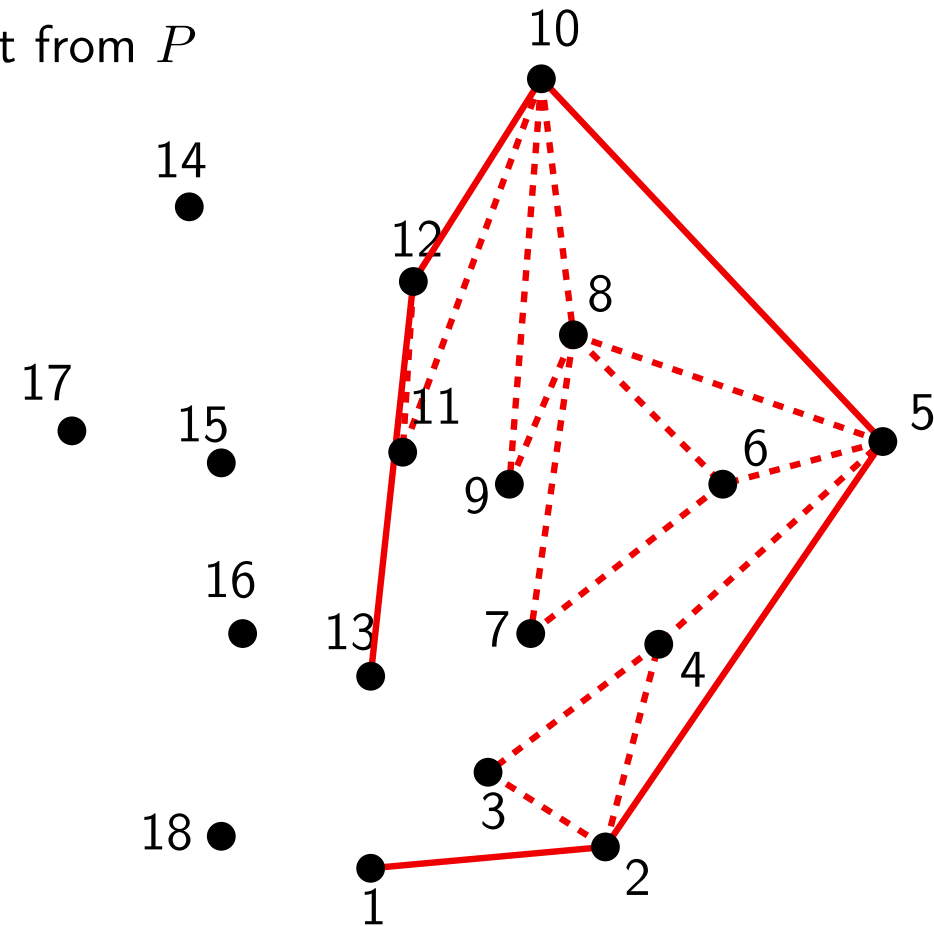
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

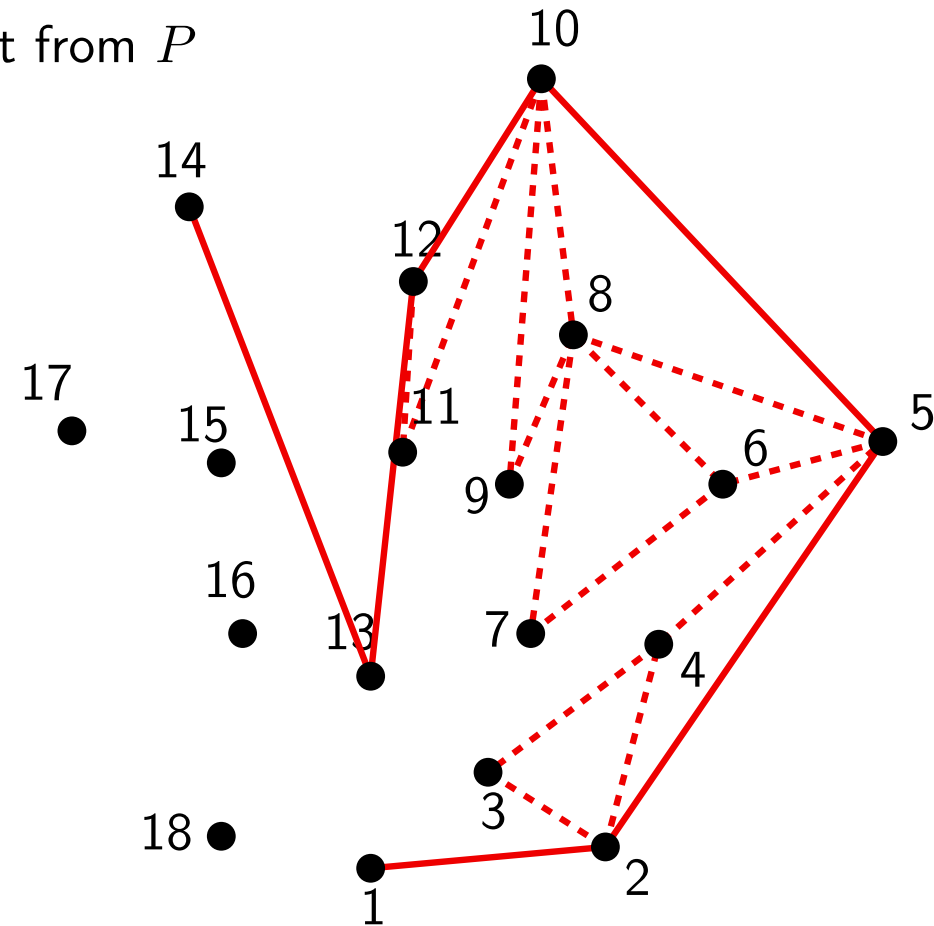
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

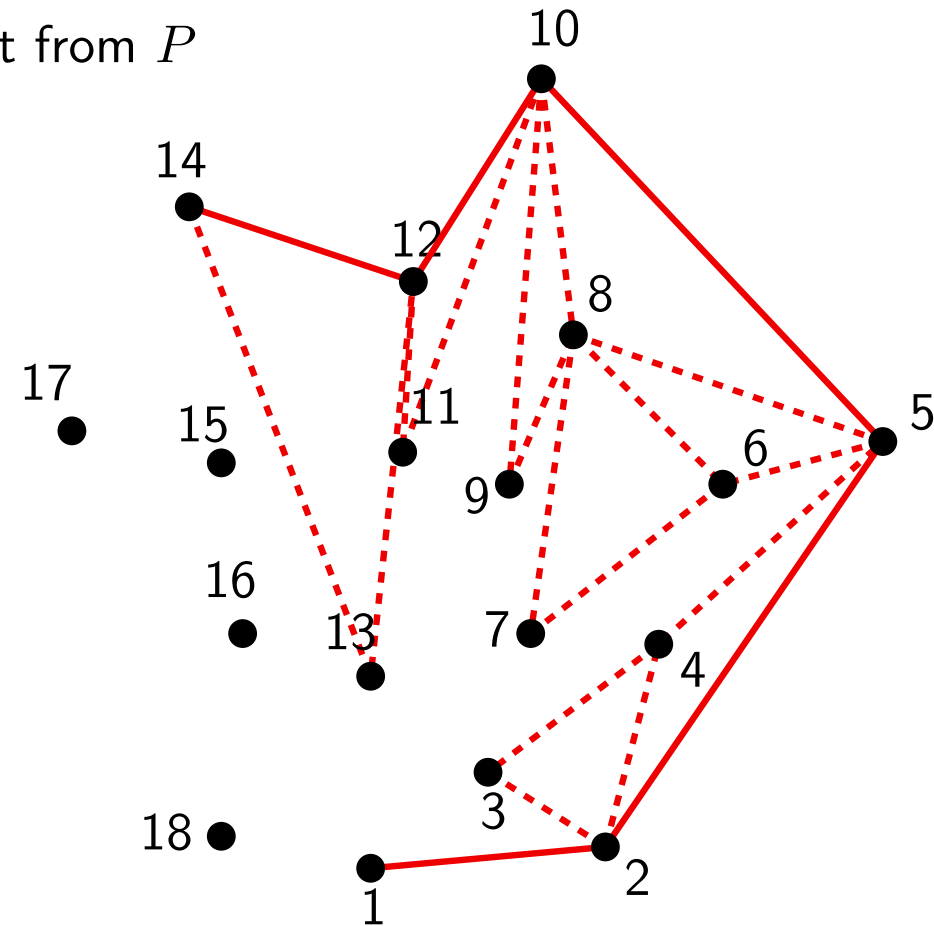
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

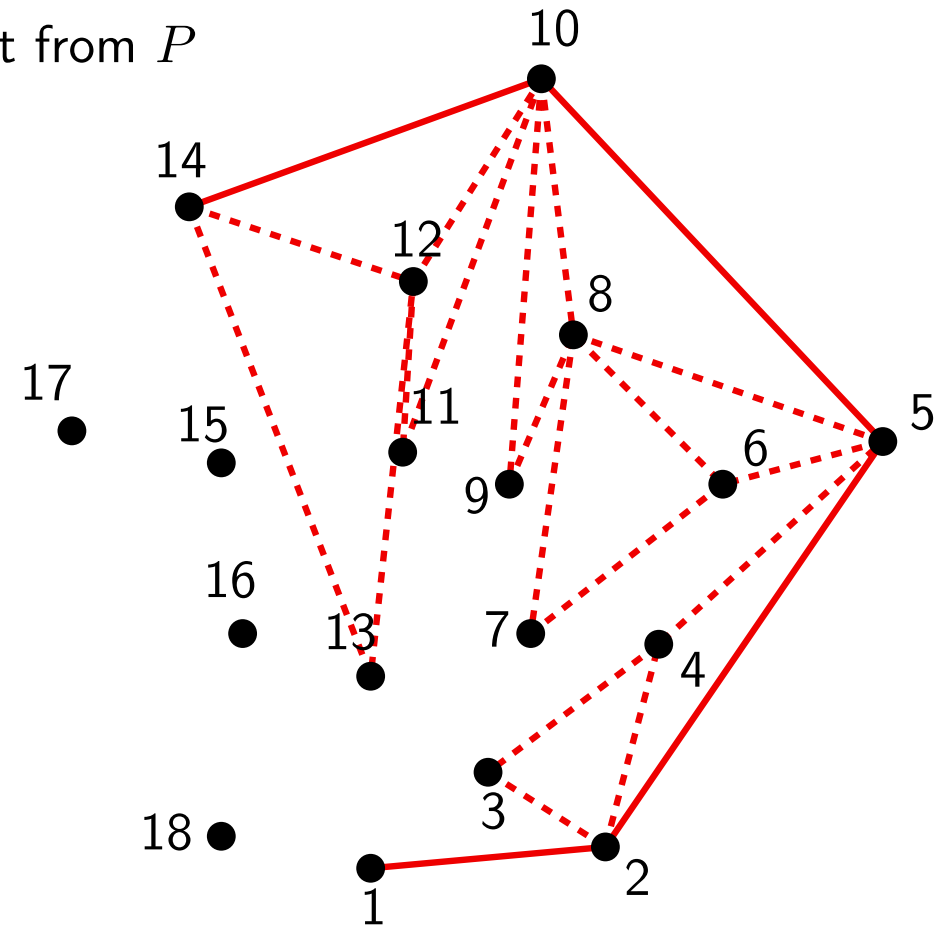
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

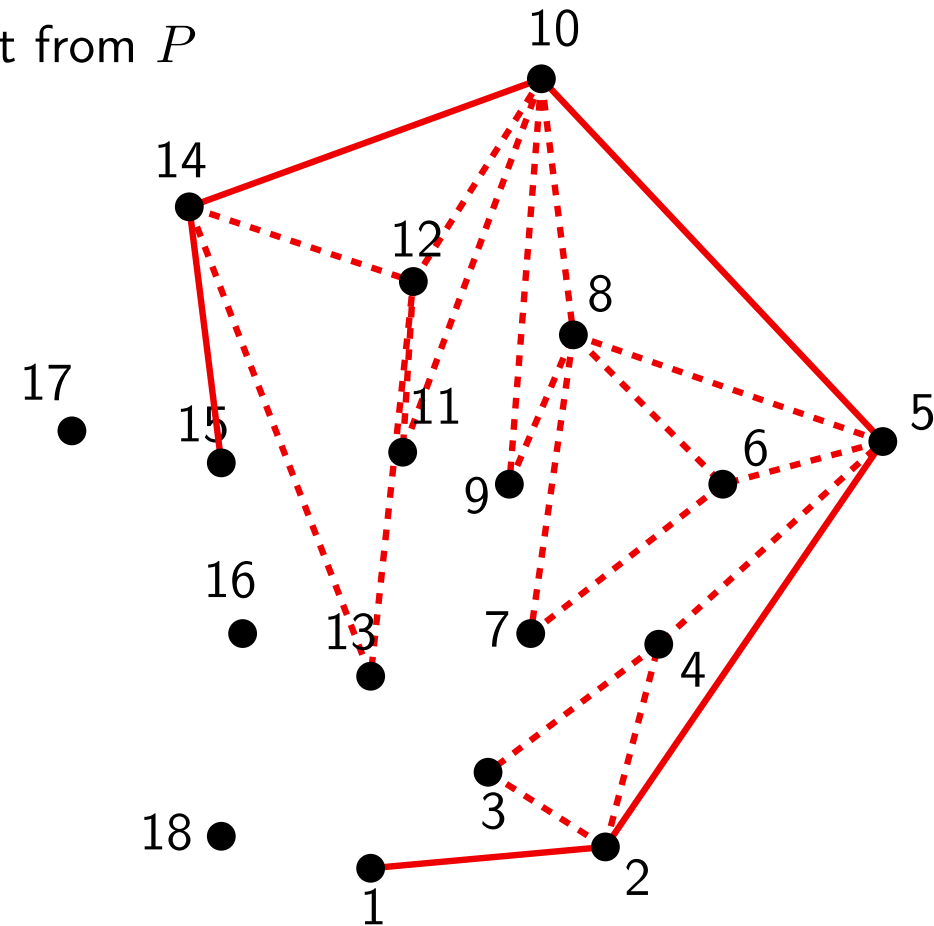
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

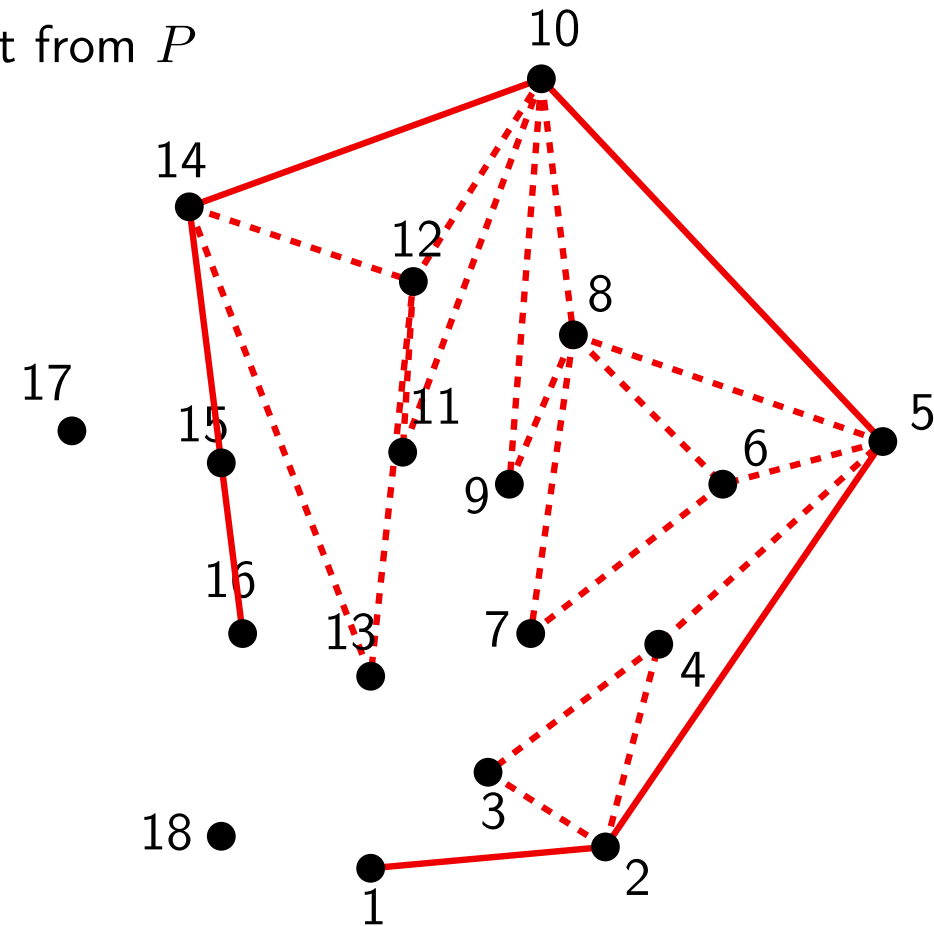
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

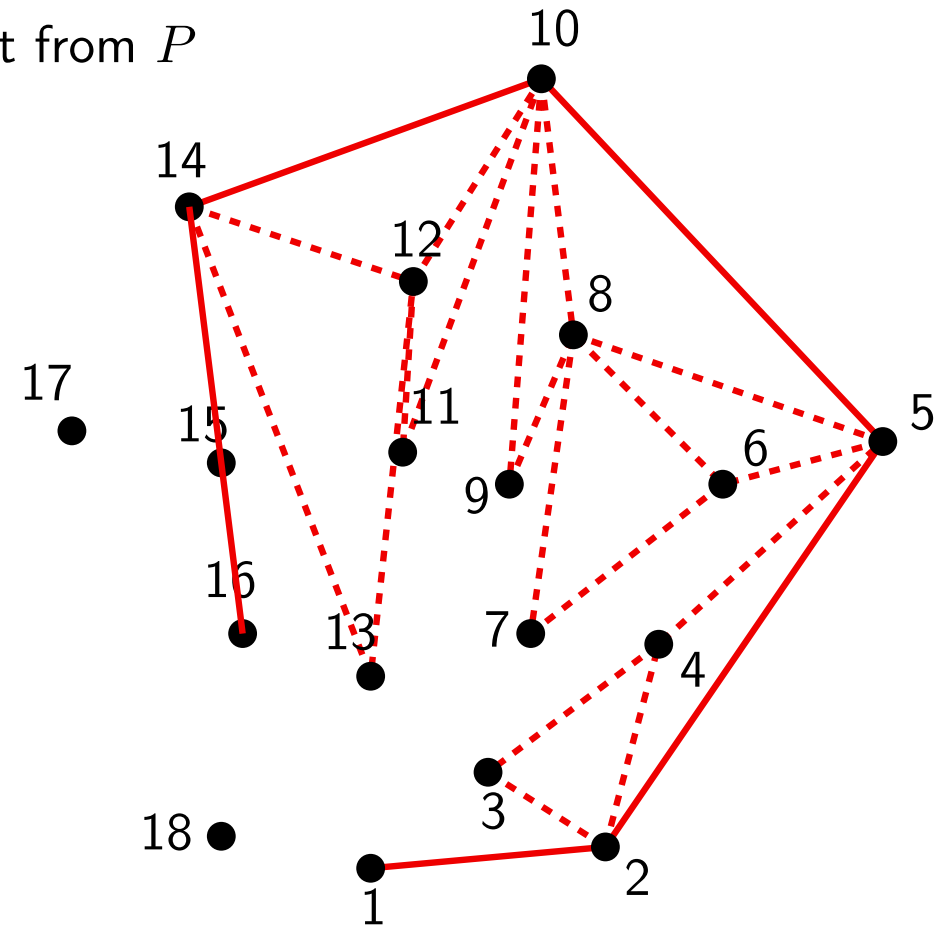
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

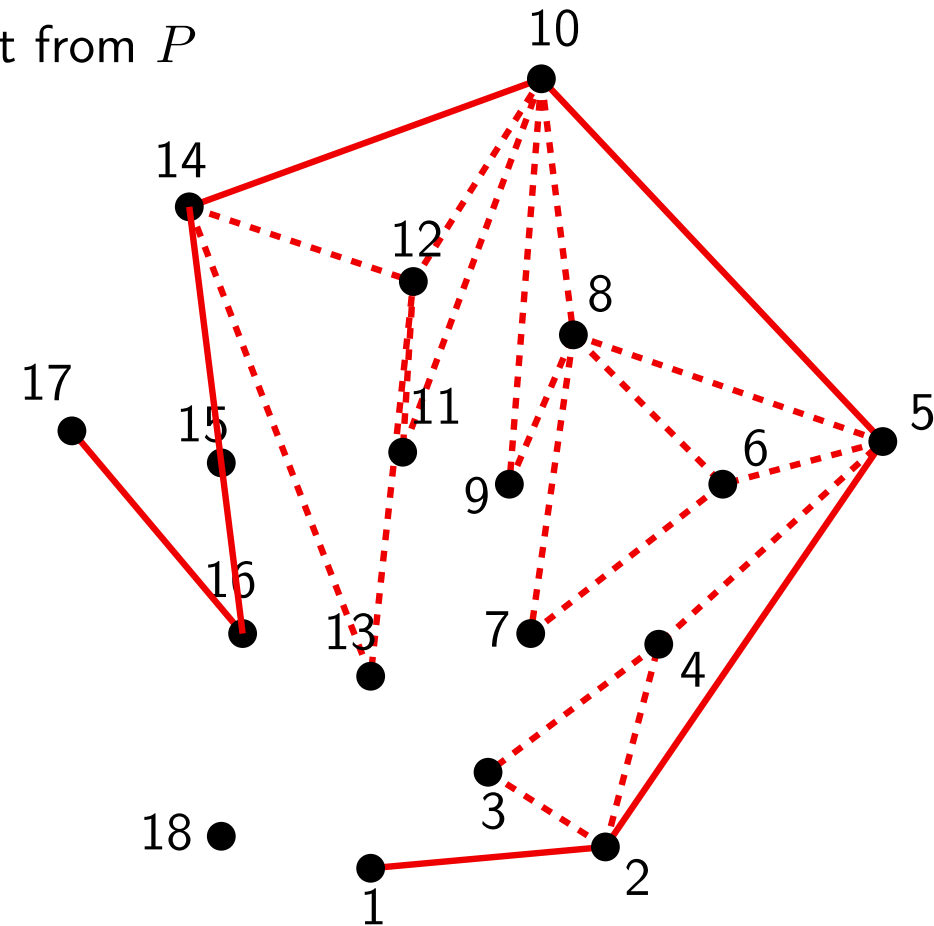
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

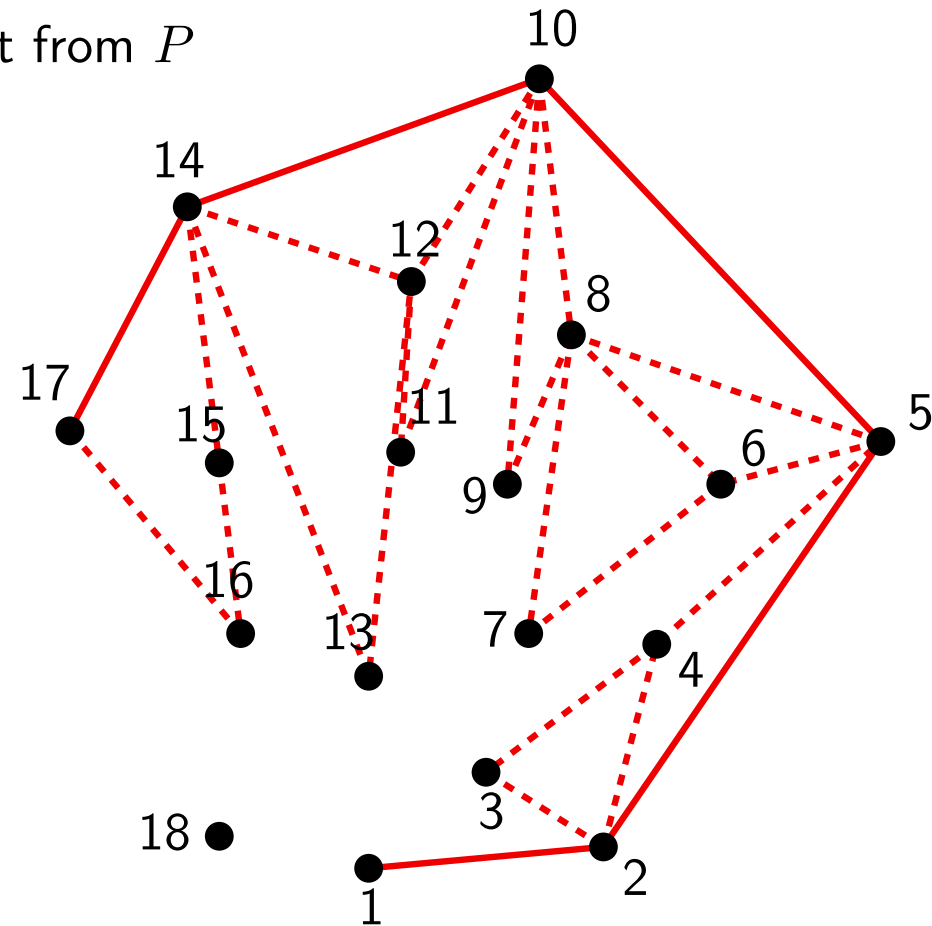
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

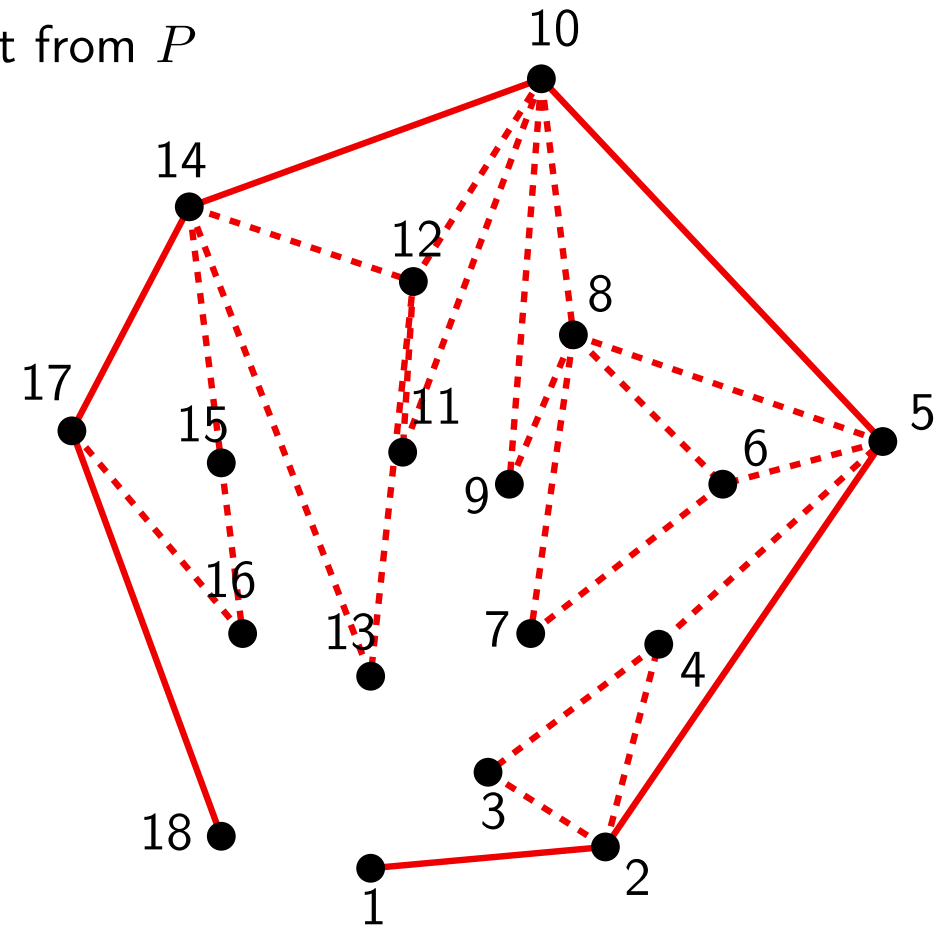
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it from  $P$

### Advance

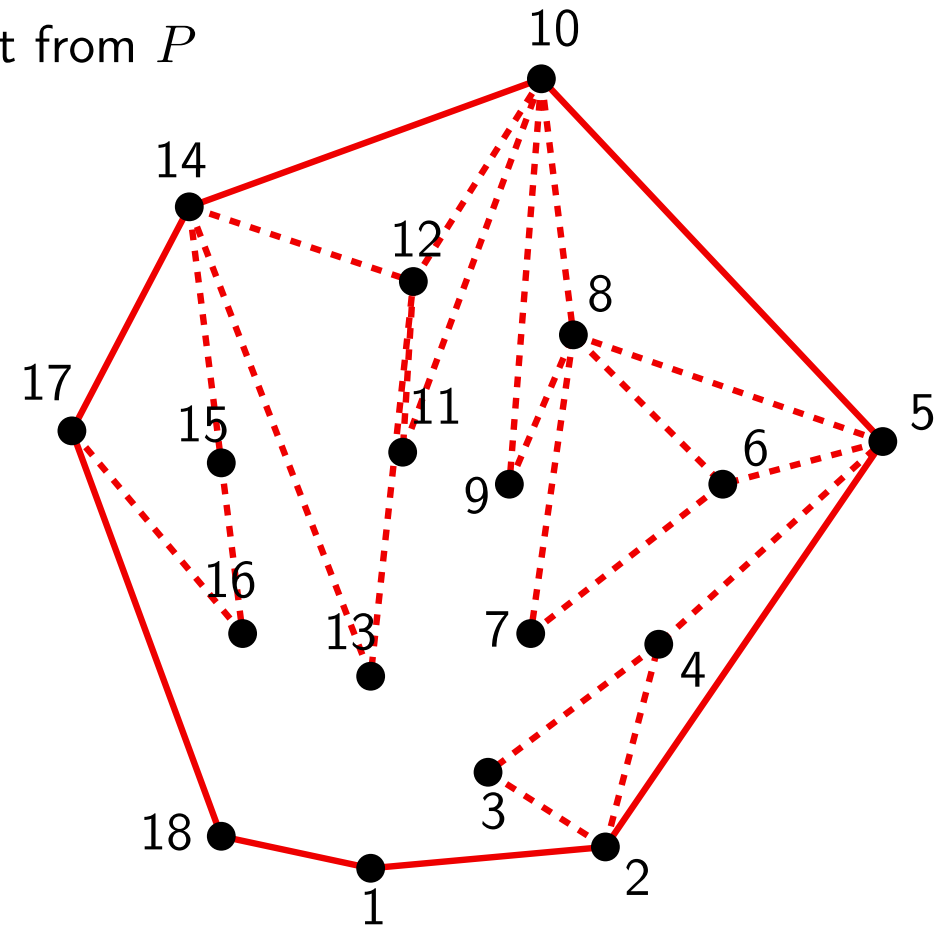
While there exist points  $p_i \in P$  to be explored, do:

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

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

- 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

## 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 it 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:

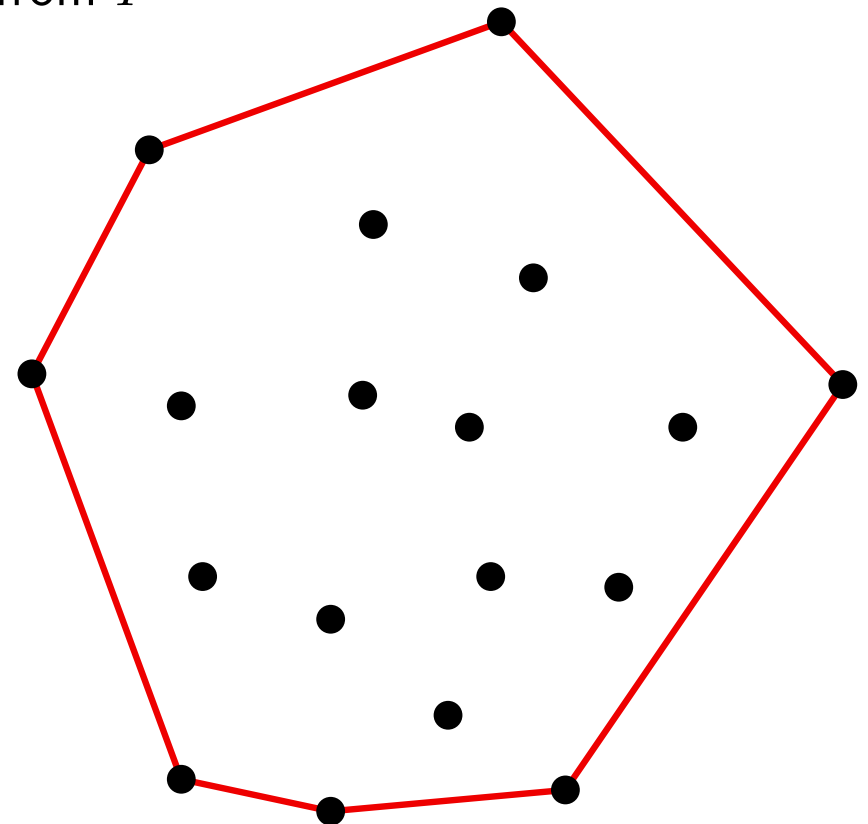
- Push  $p_i$  in  $l$

- Advance  $i$

- Else:

- Pop  $p$  from  $l$

Return  $l$



# CONVEX HULL

## 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 it 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  $l$

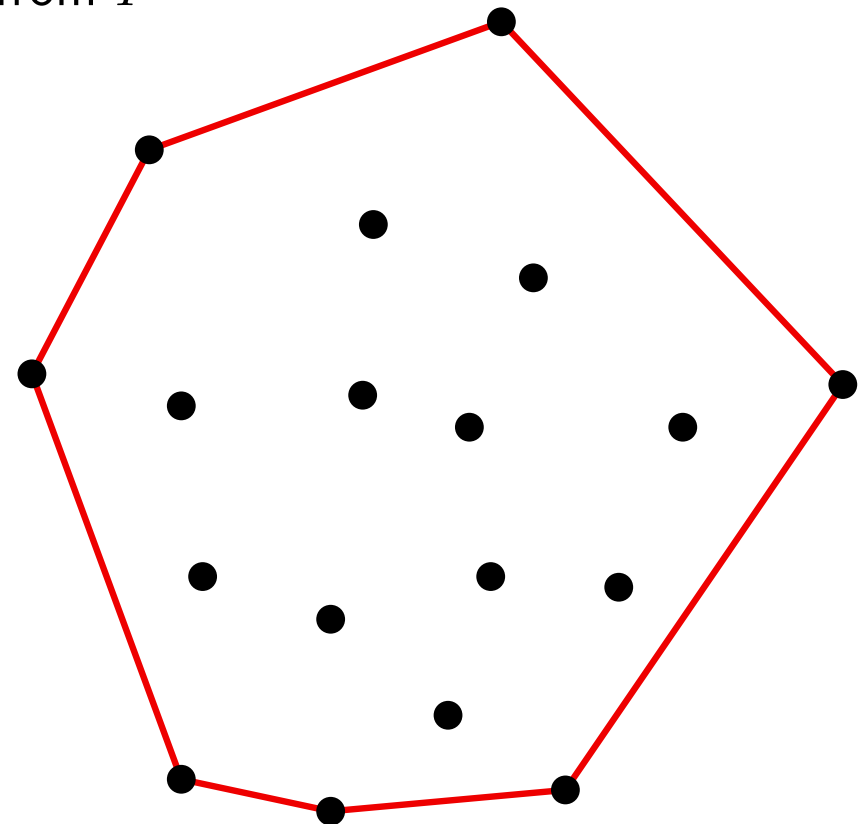
- Advance  $i$

- Else:

- Pop  $p$  from  $l$

Return  $l$

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



# CONVEX HULL

**Incremental algorithm**

# CONVEX HULL

## 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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

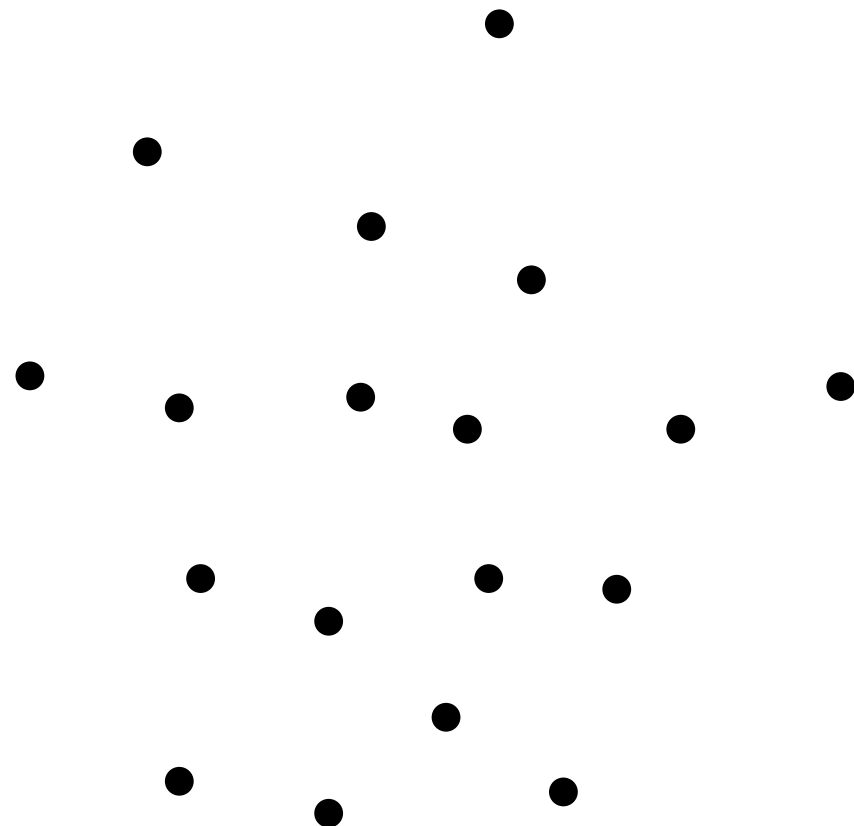
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

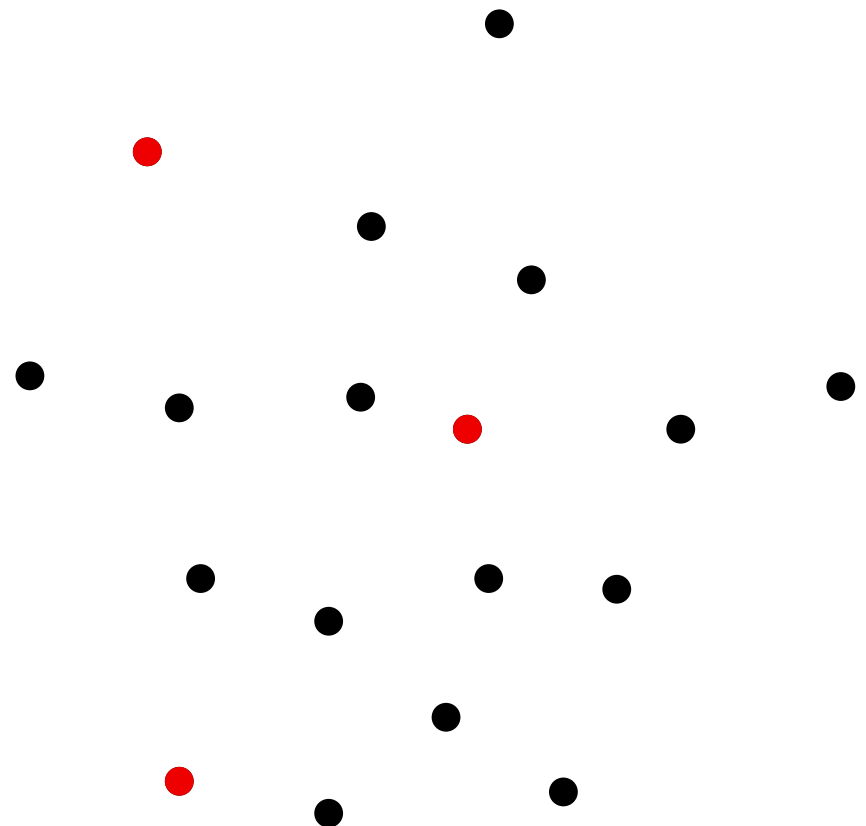
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

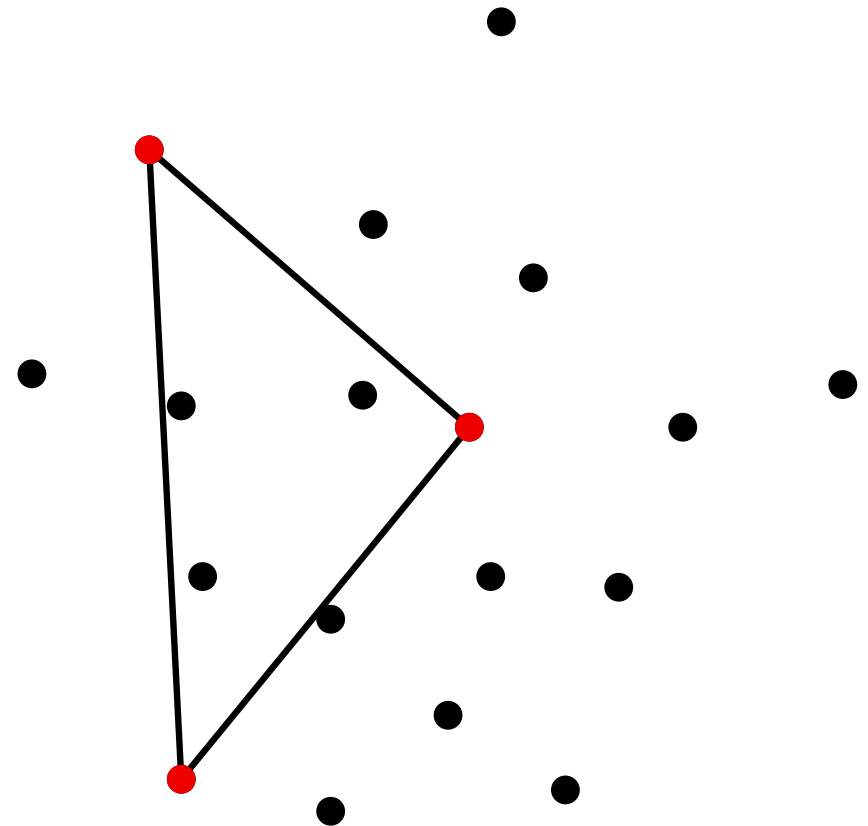
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

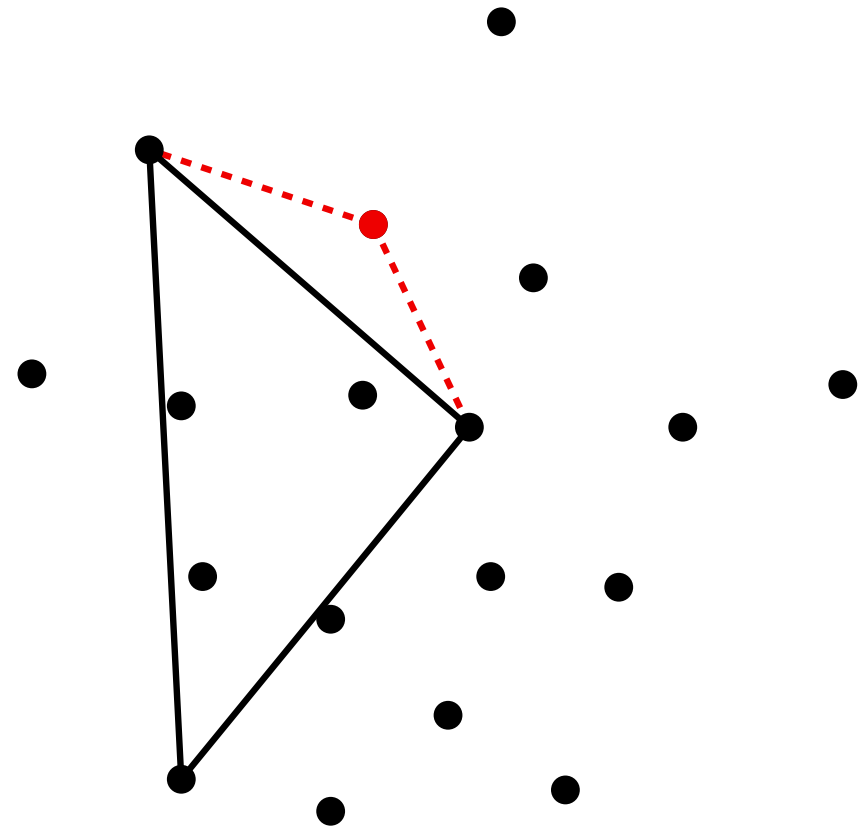
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

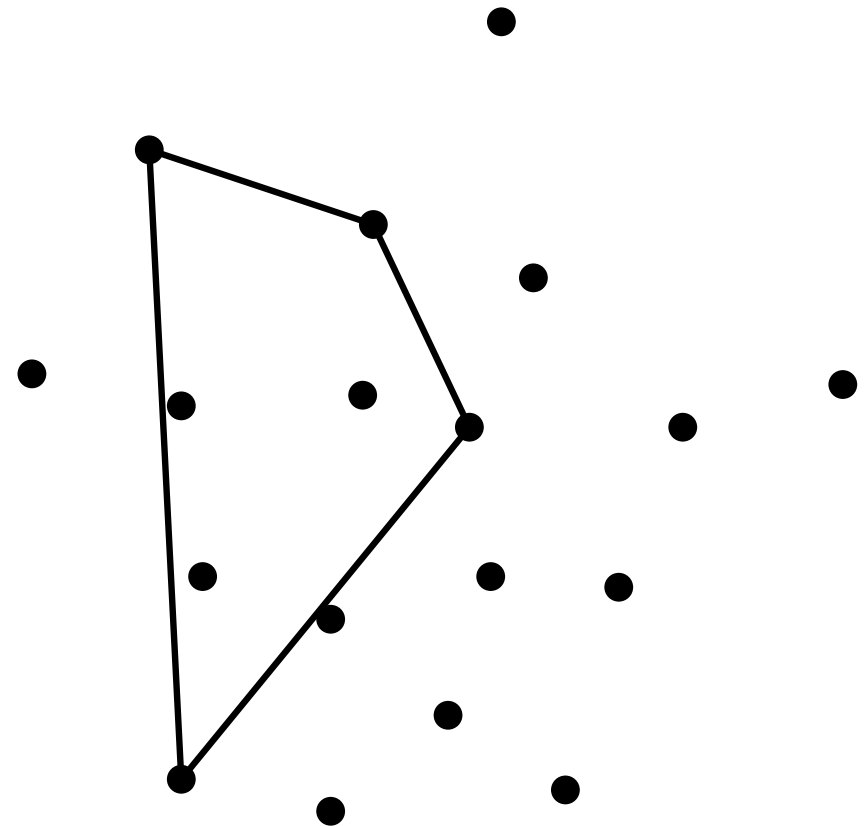
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

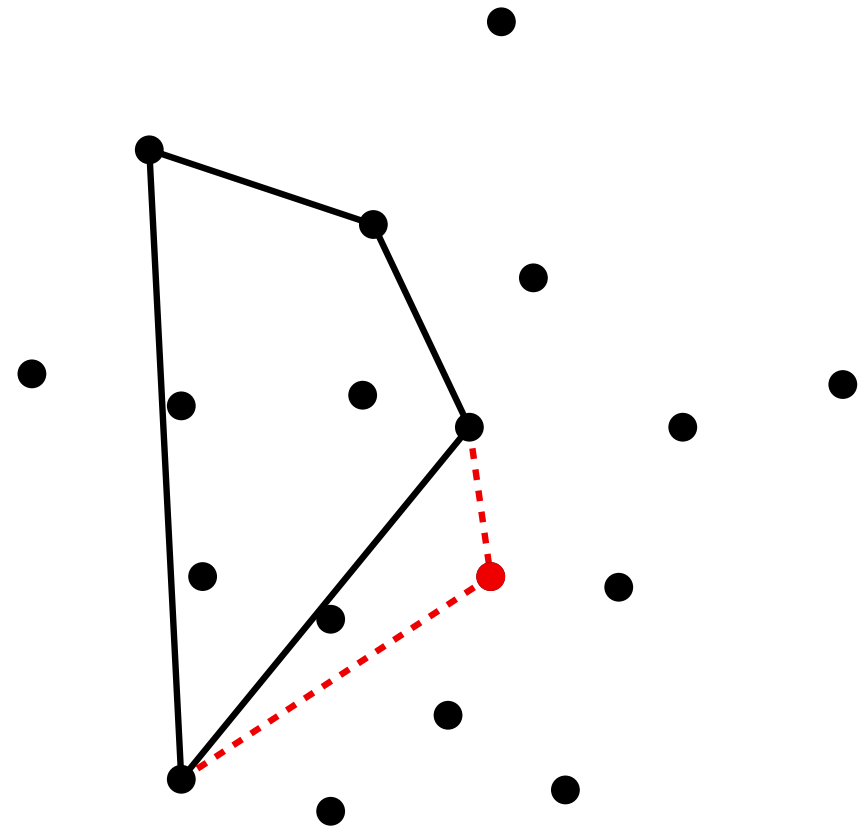
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

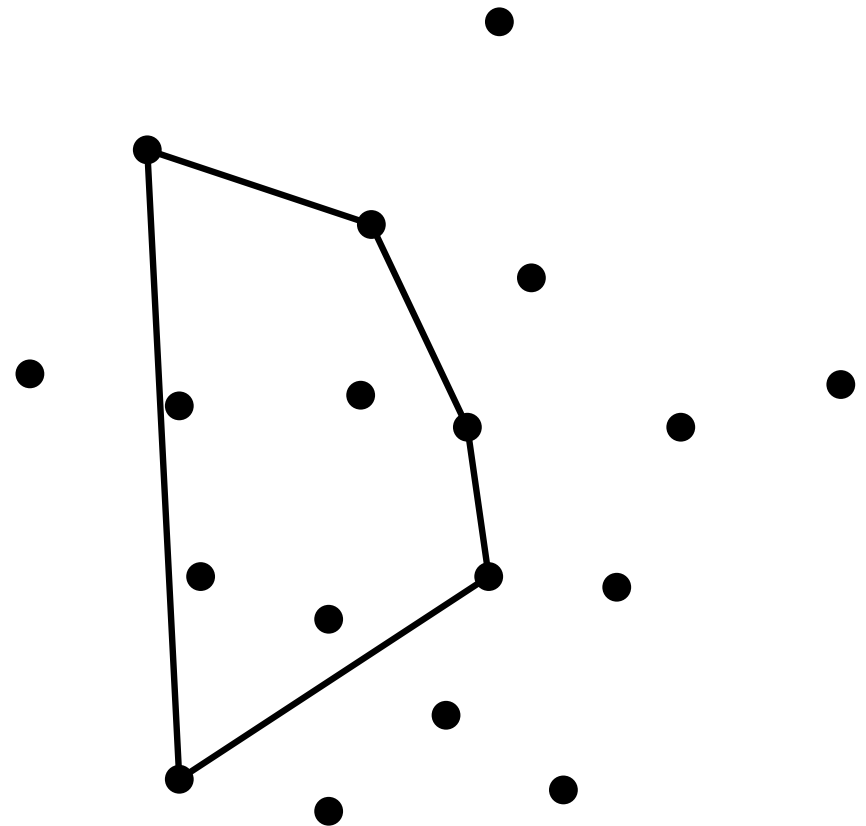
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

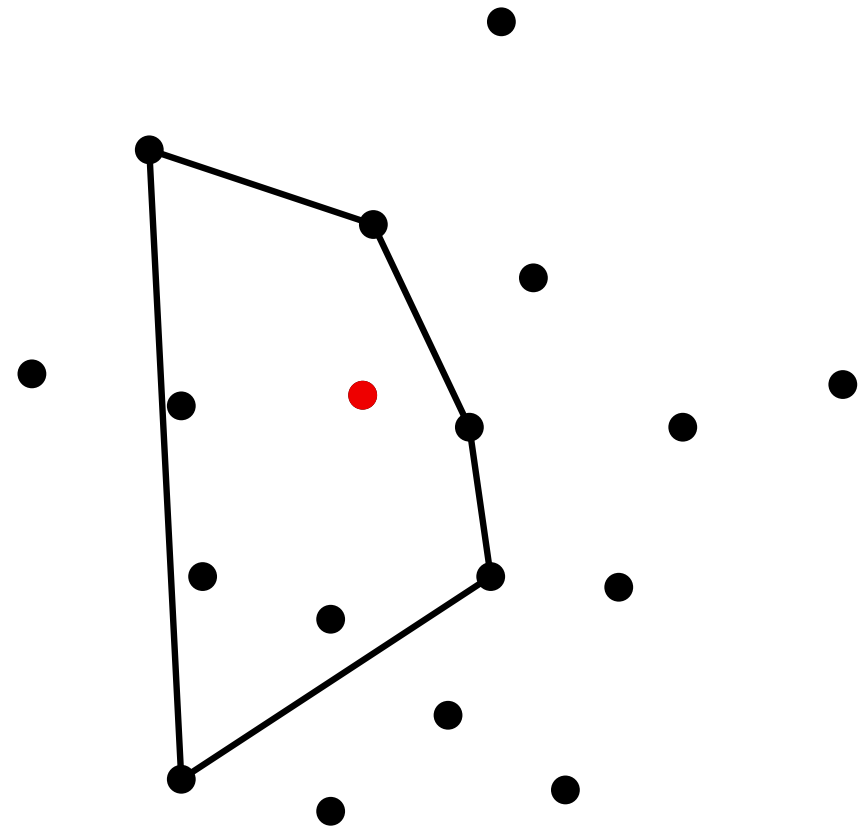
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

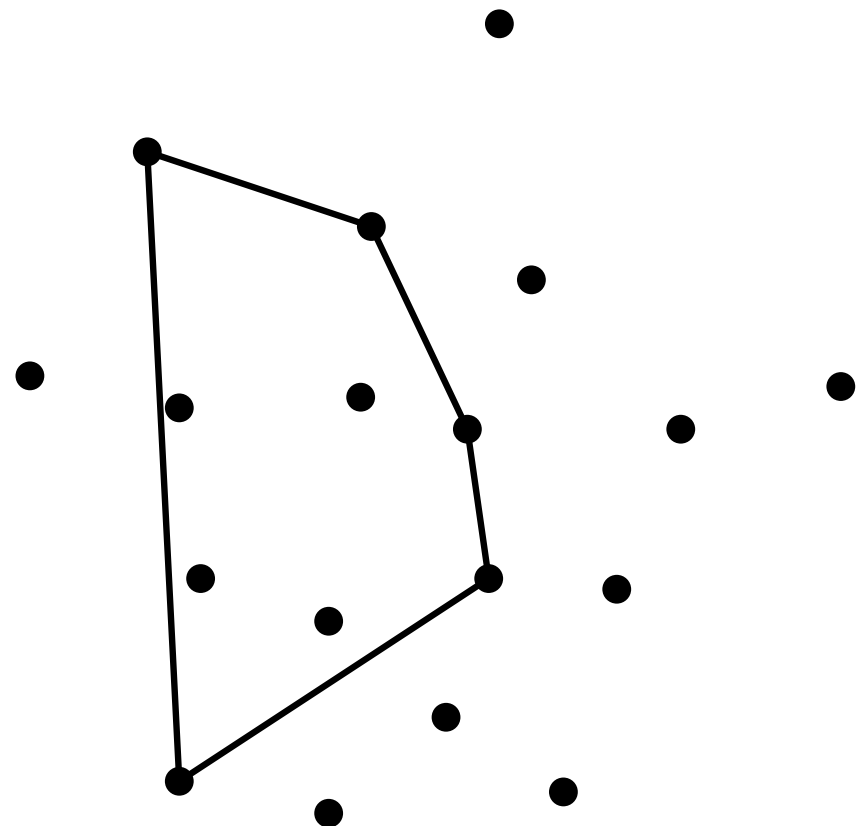
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

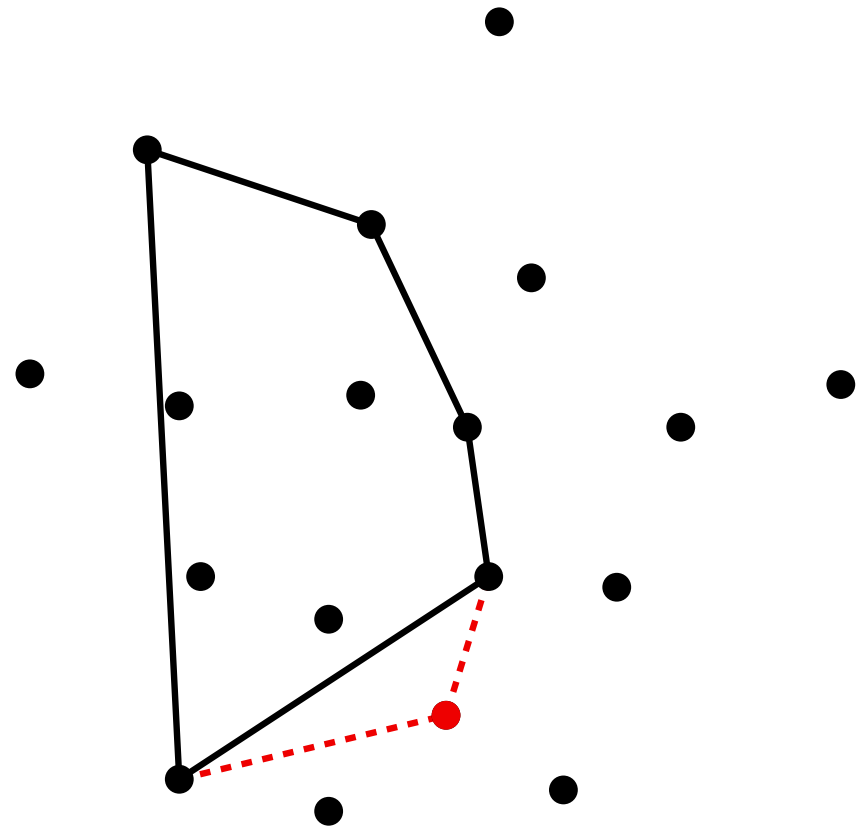
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

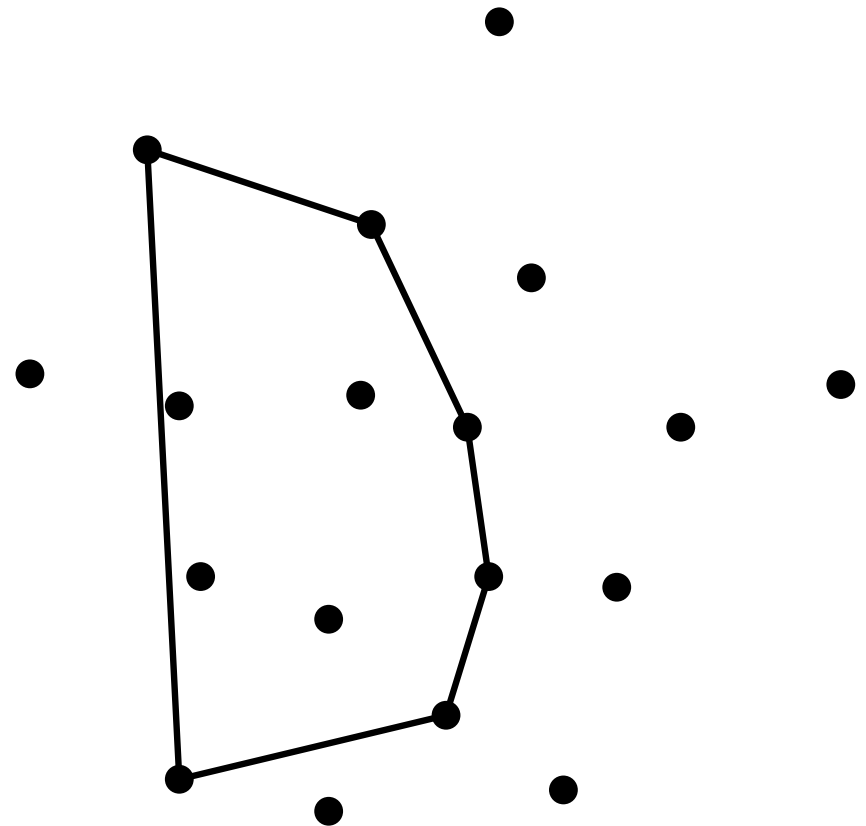
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

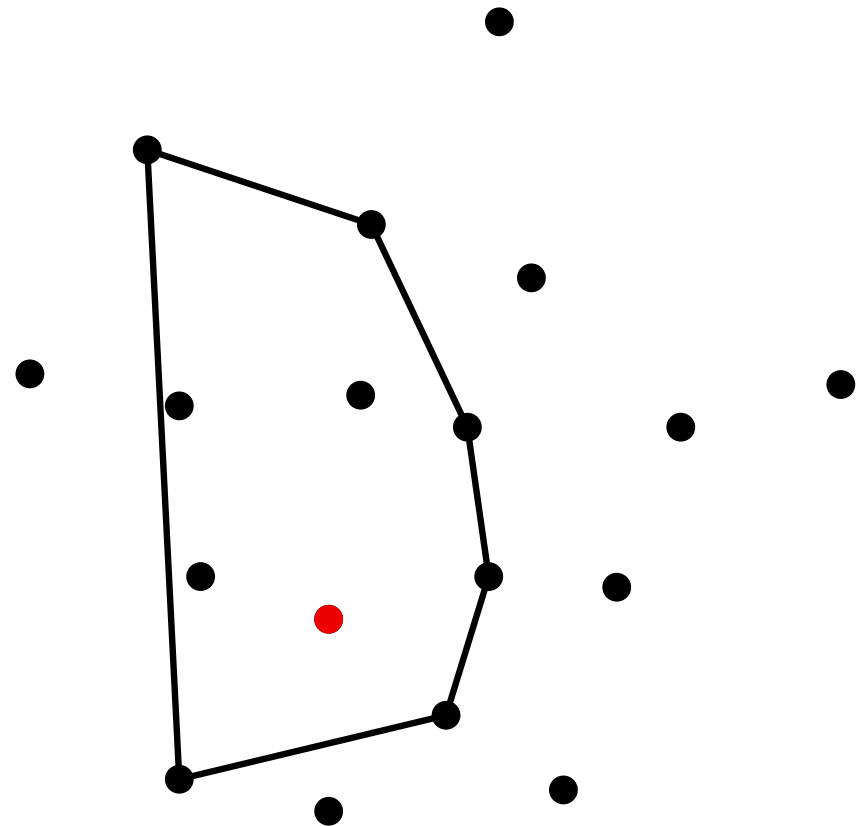
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

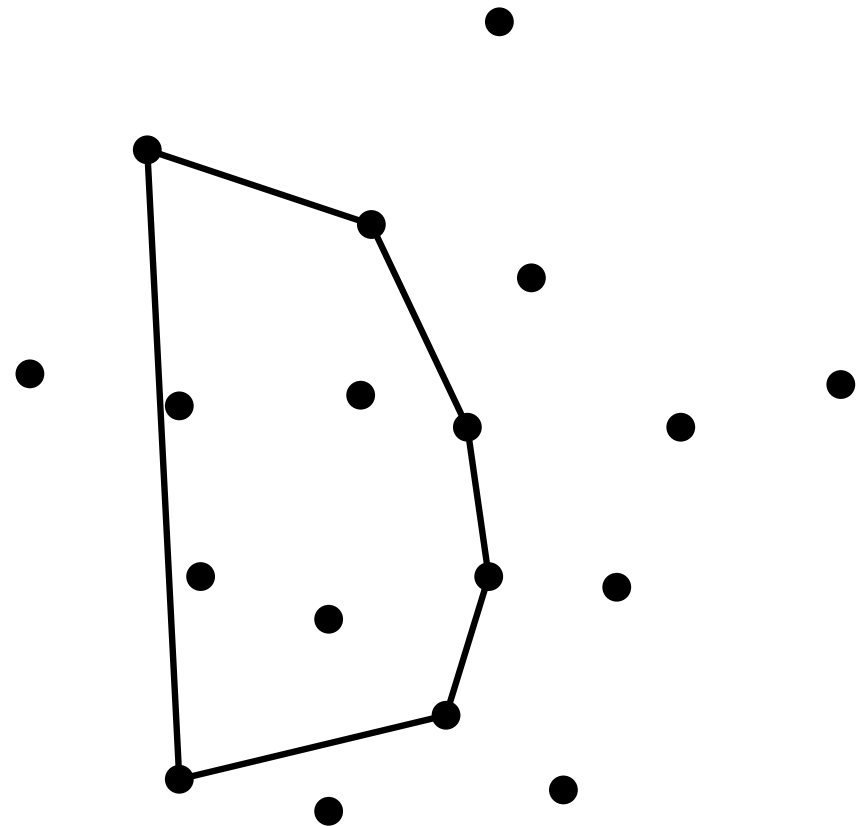
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

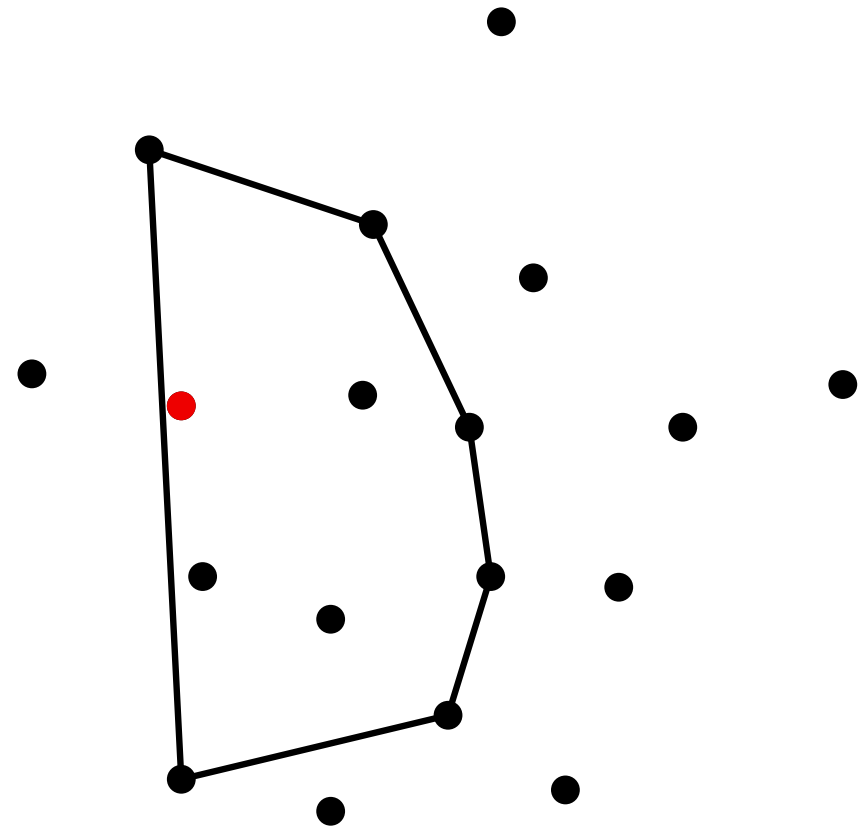
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

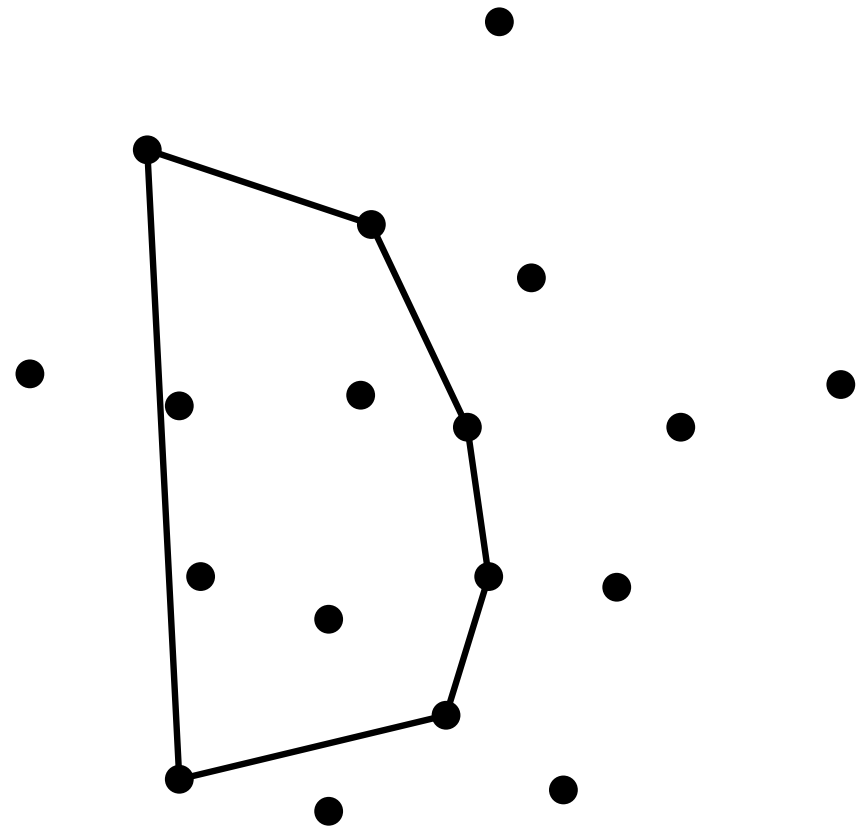
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

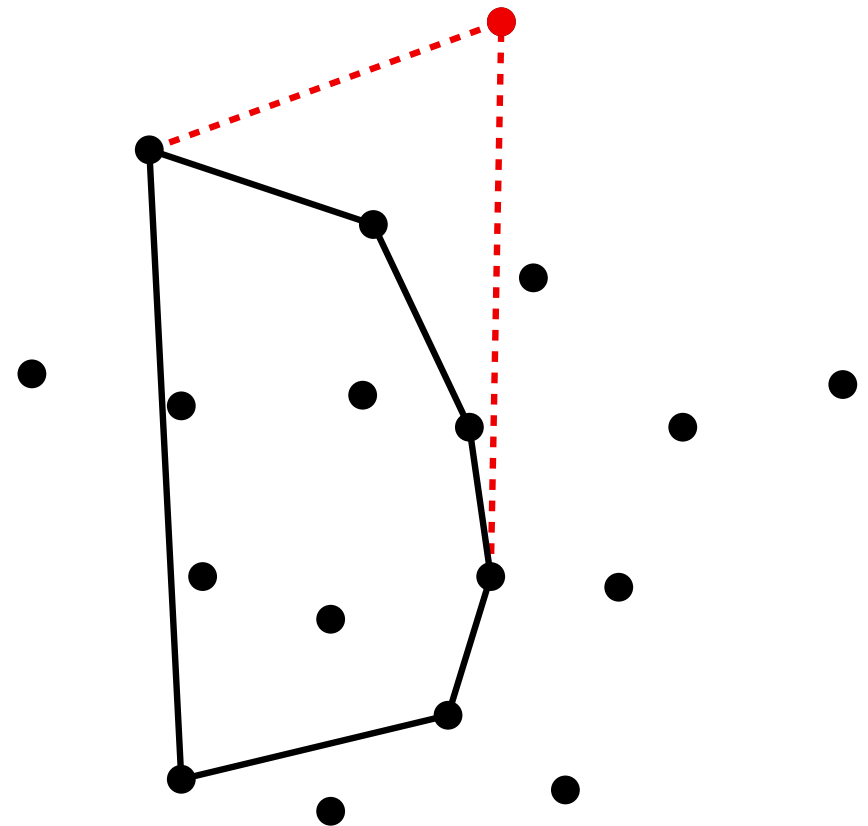
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

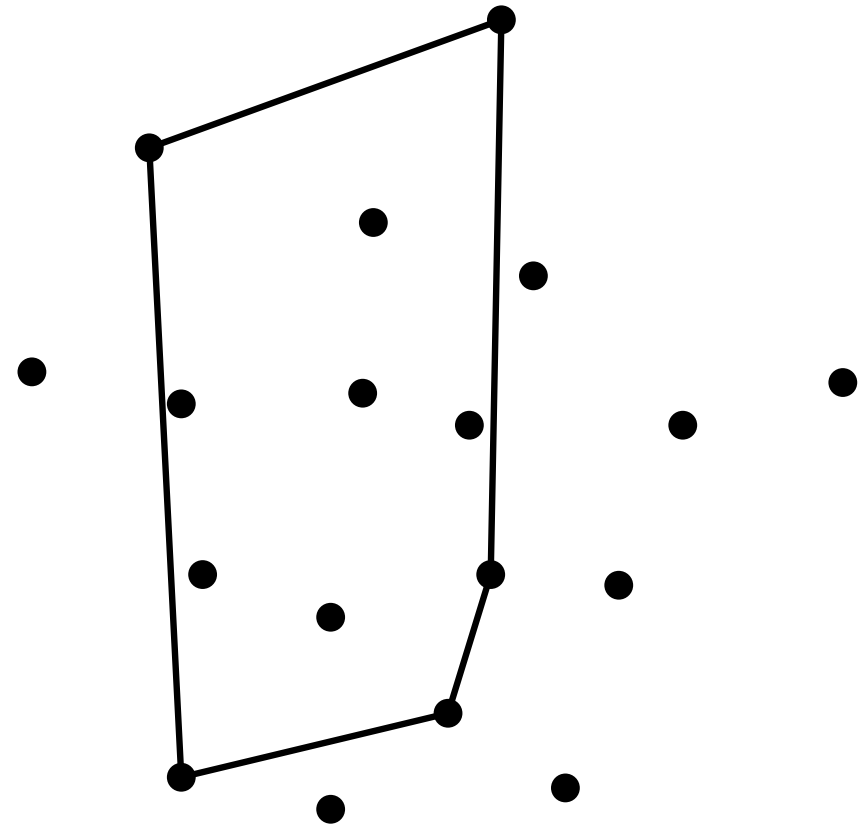
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

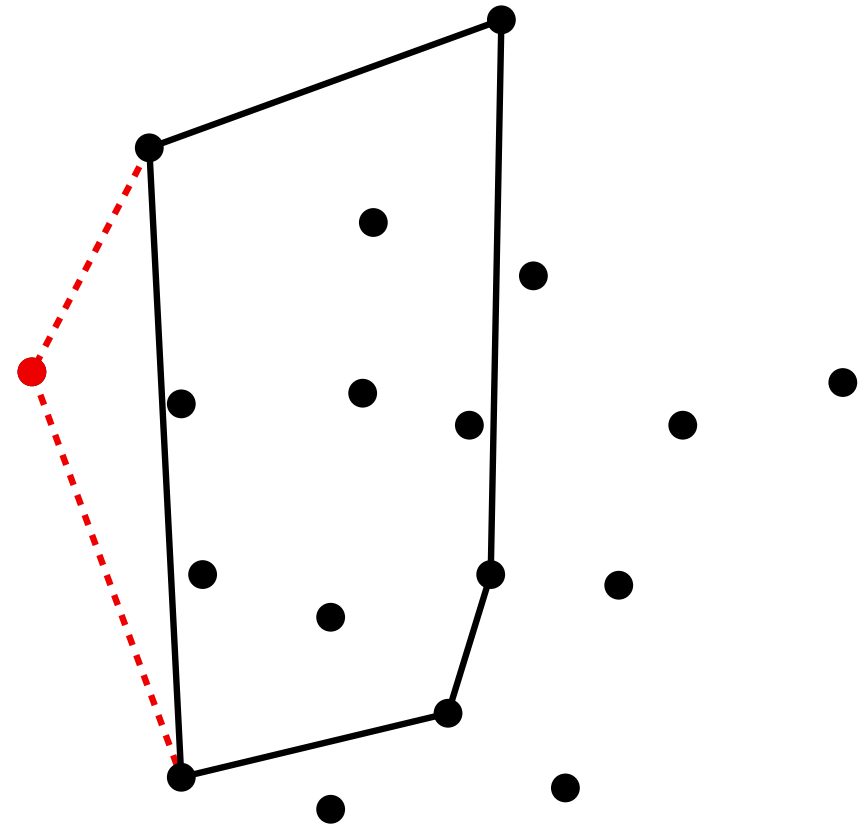
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

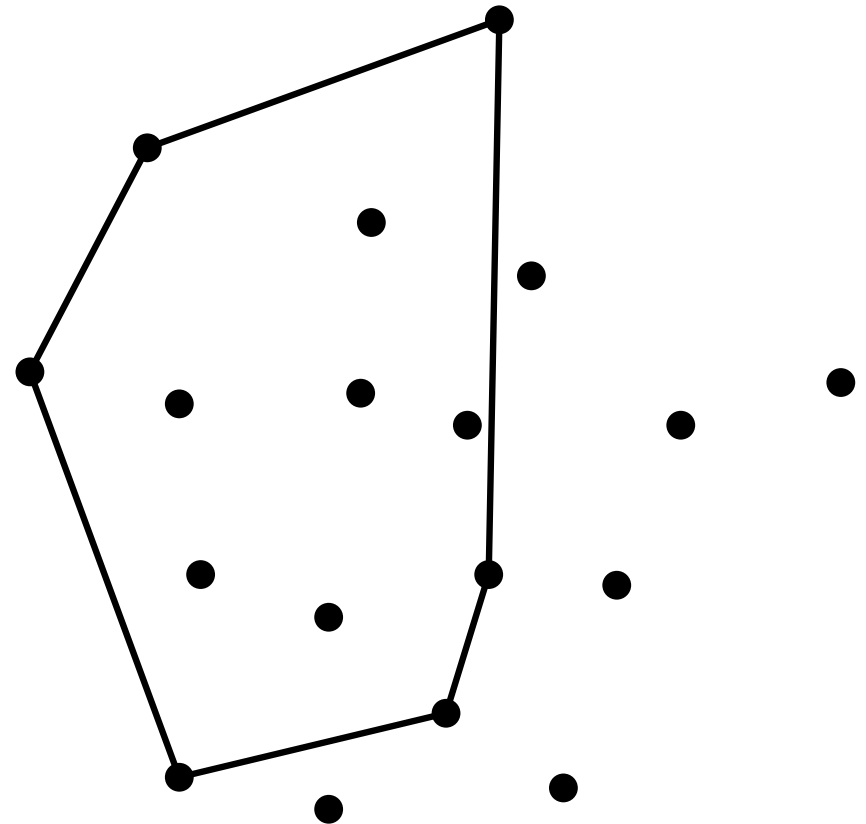
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

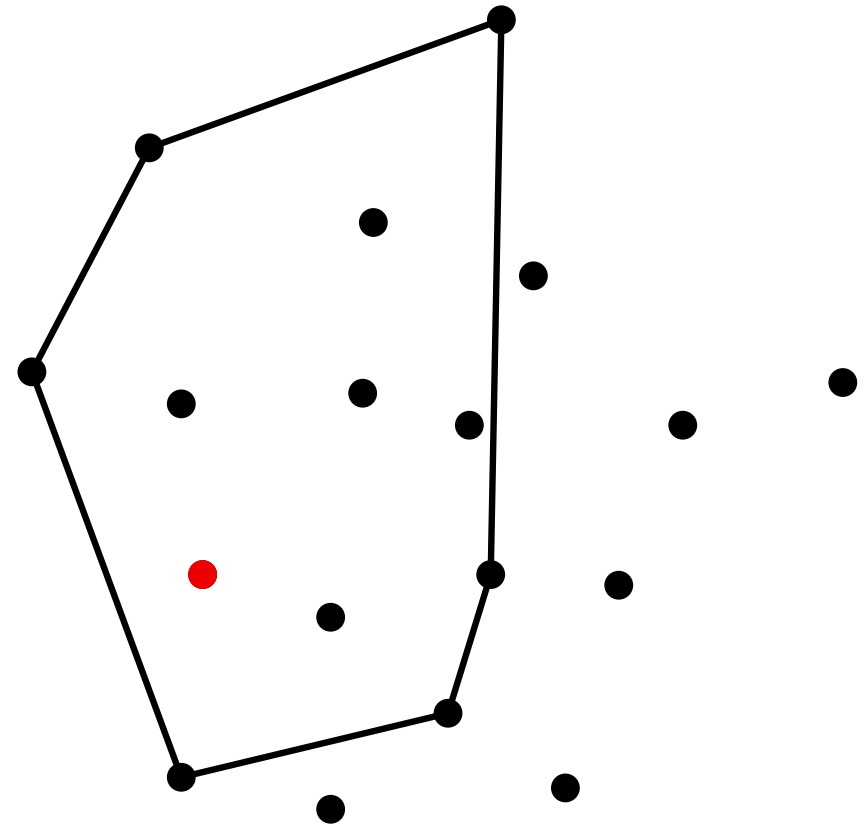
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

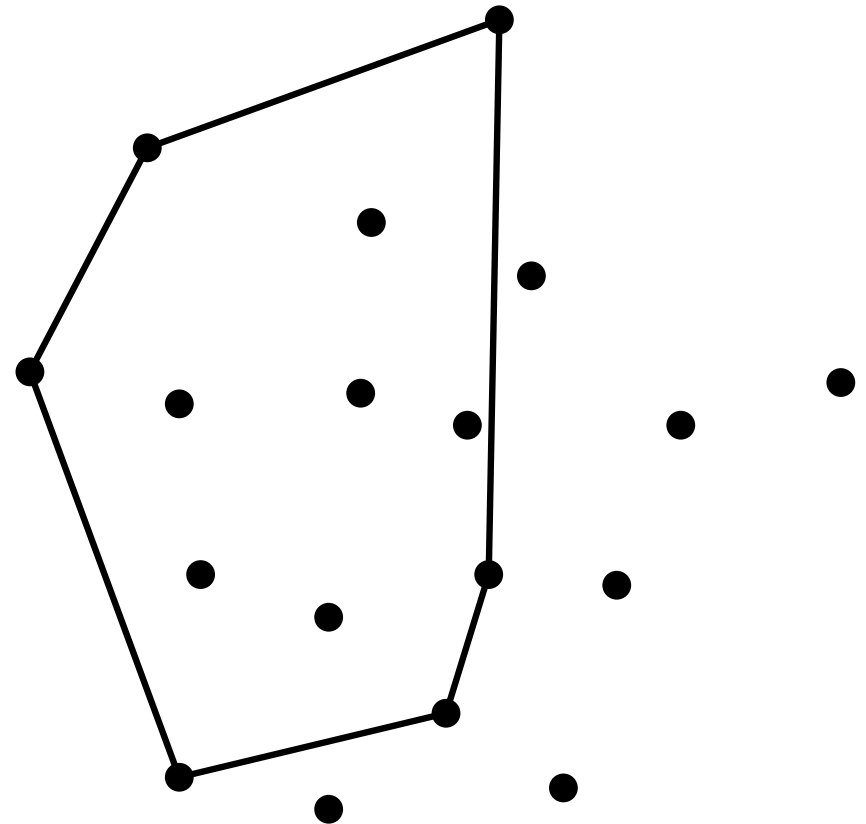
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

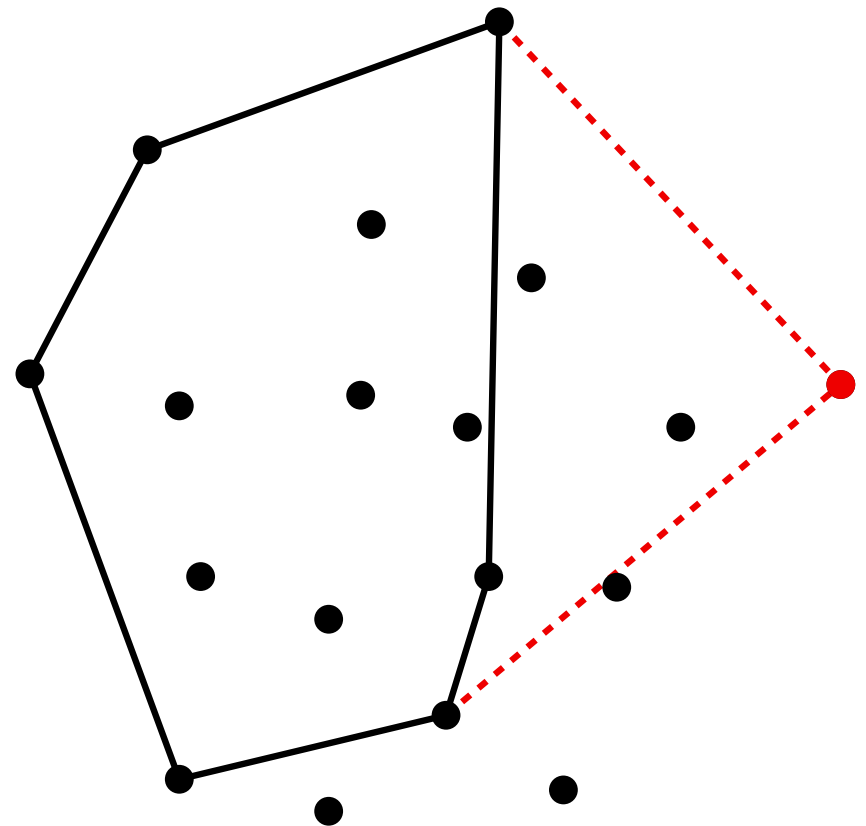
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

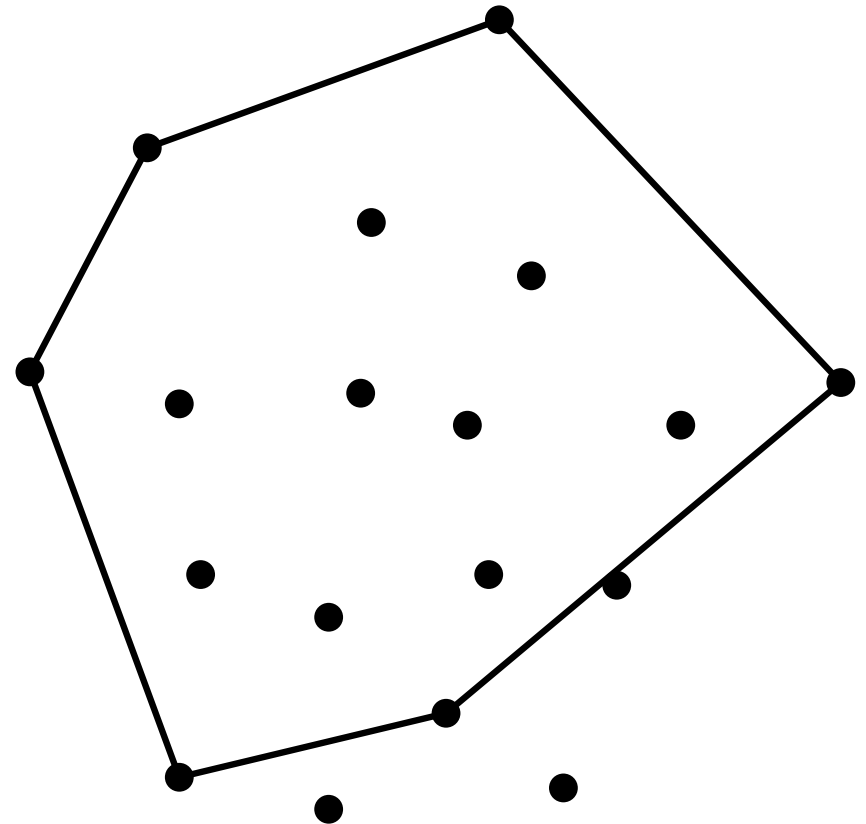
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

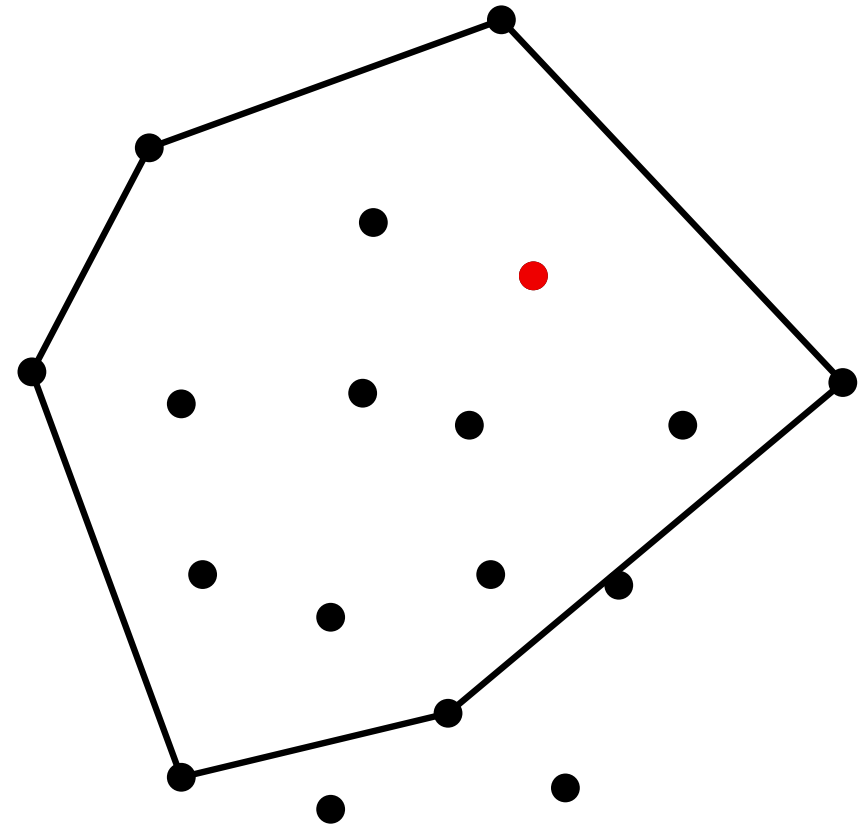
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

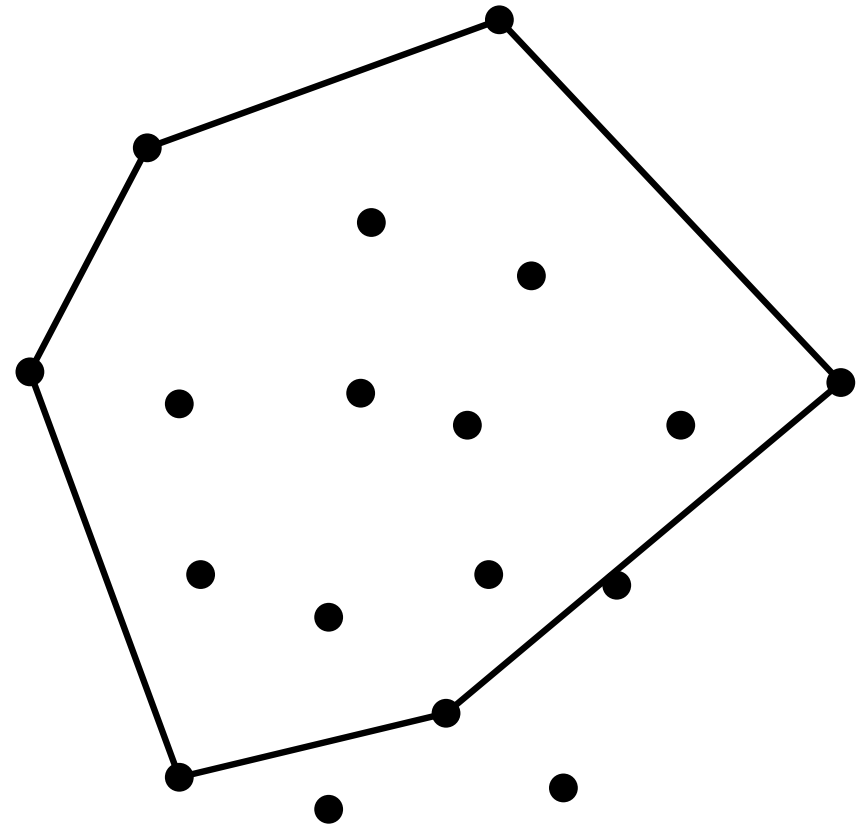
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

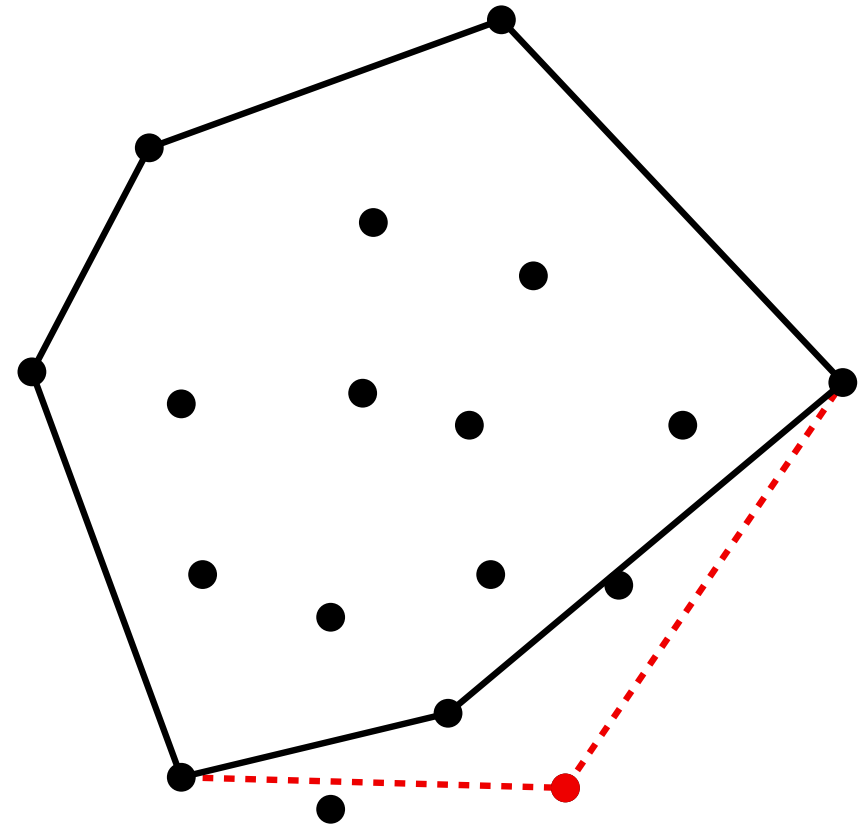
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

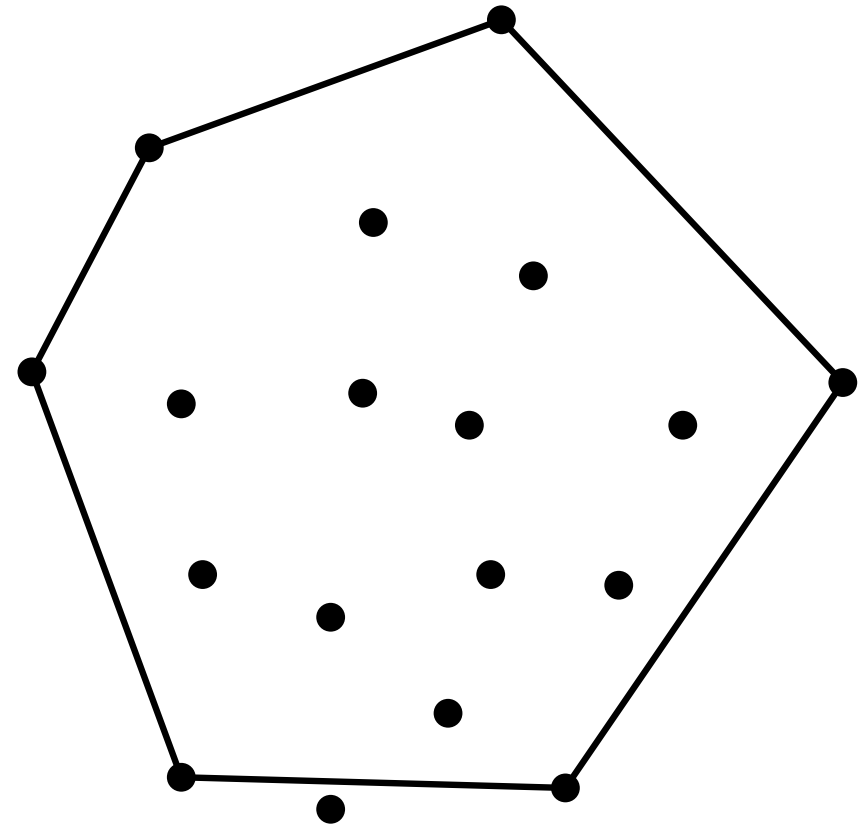
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

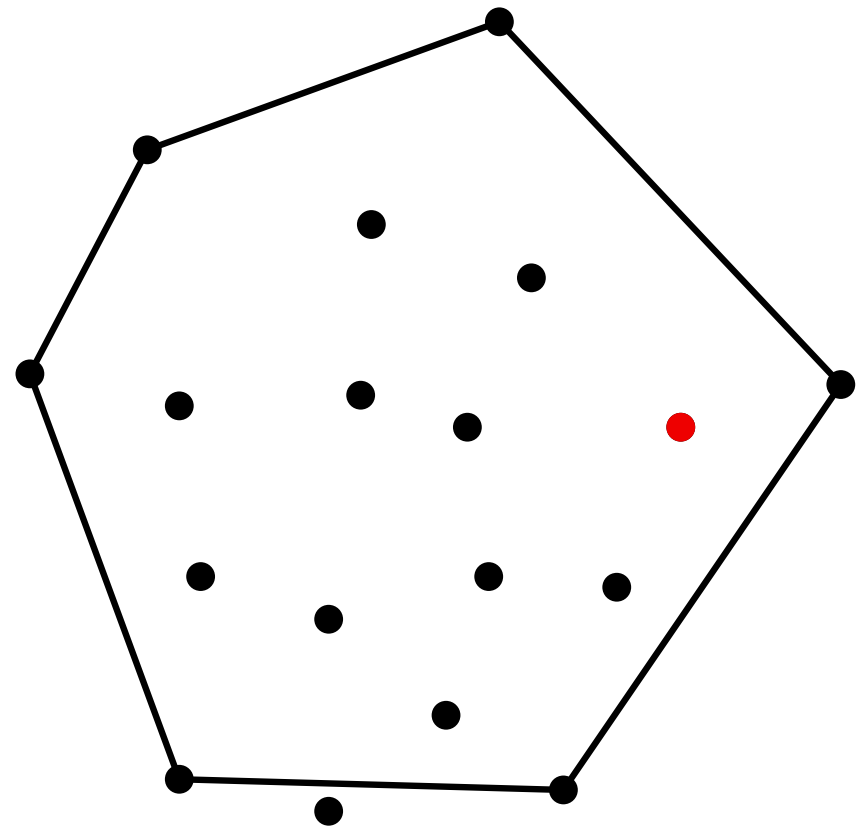
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

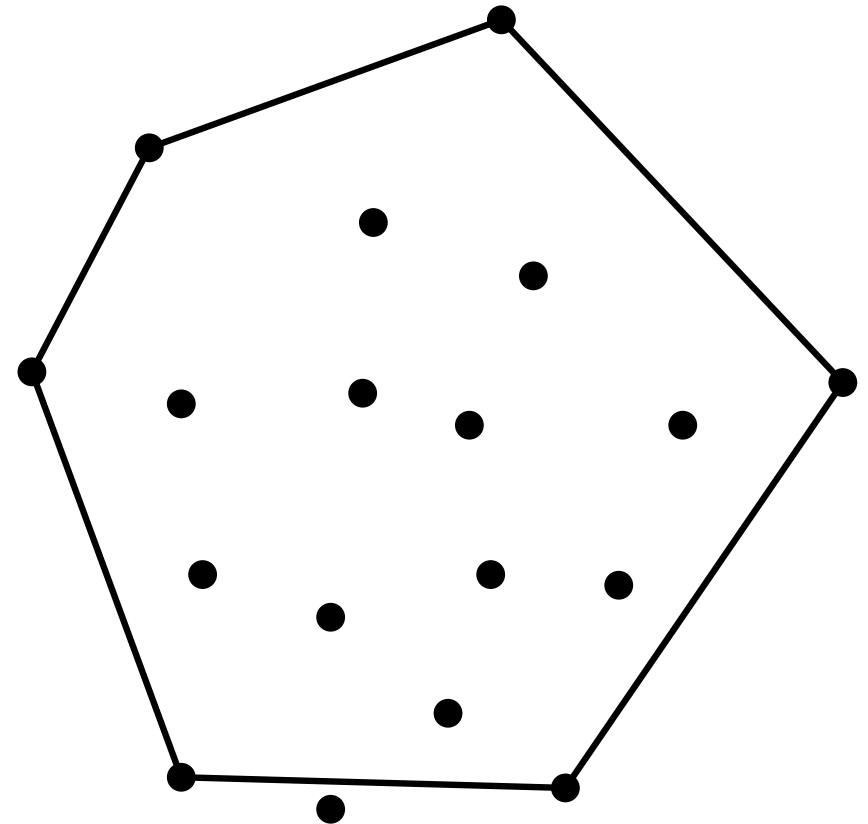
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

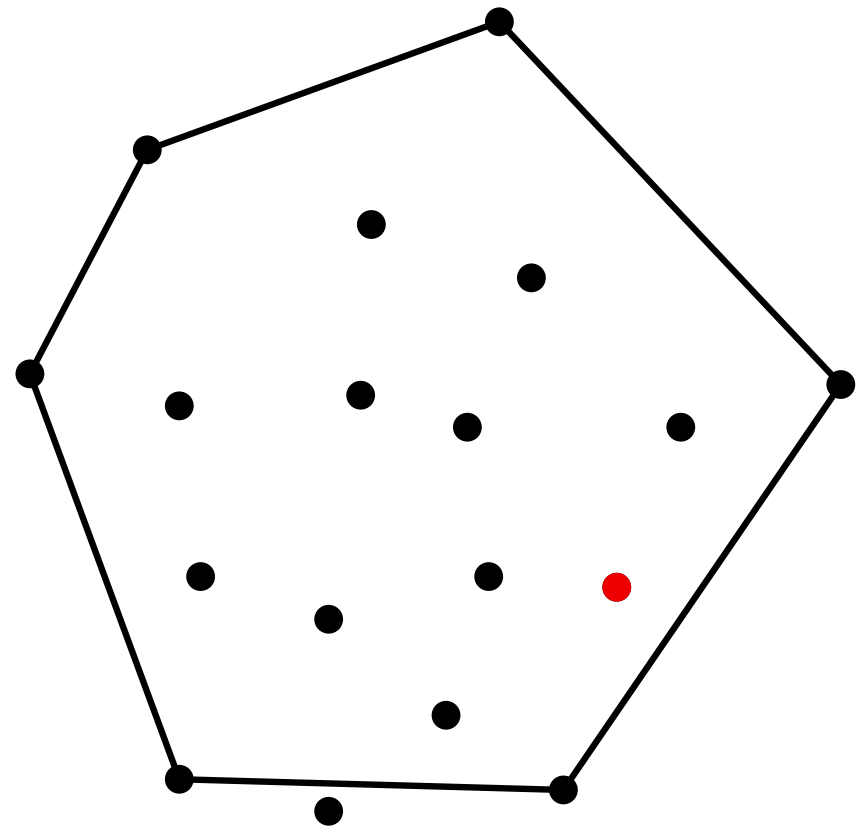
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

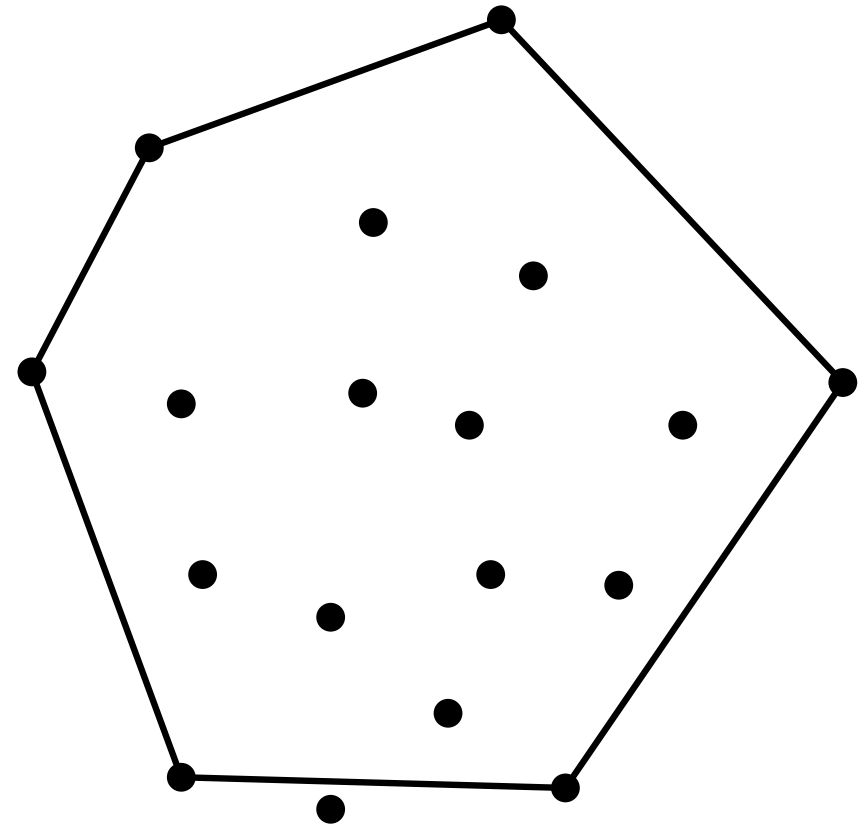
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

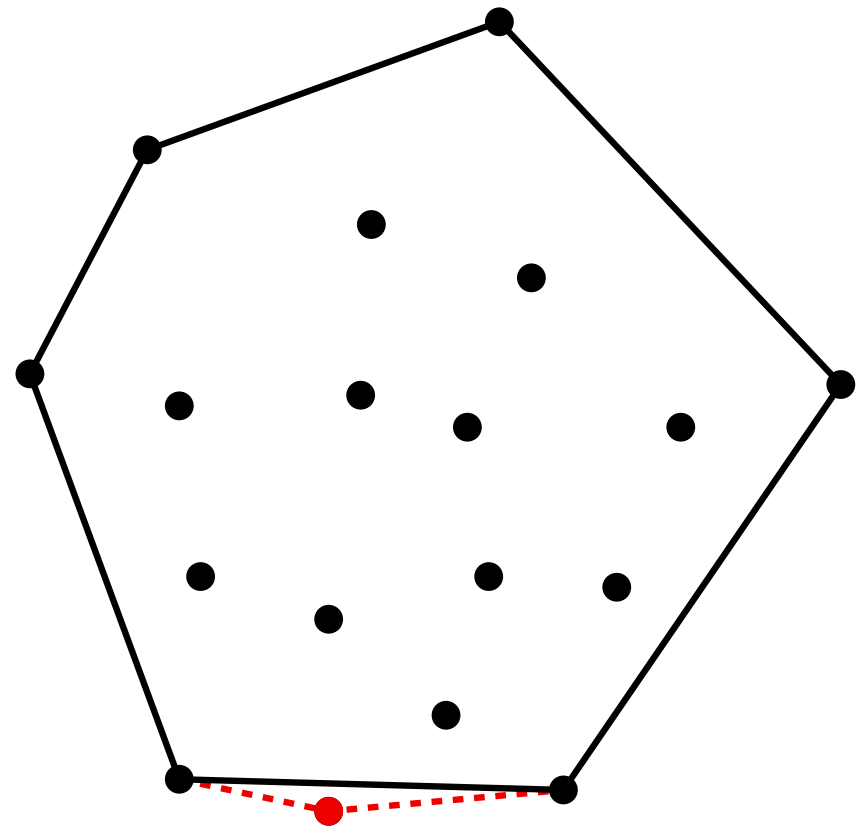
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

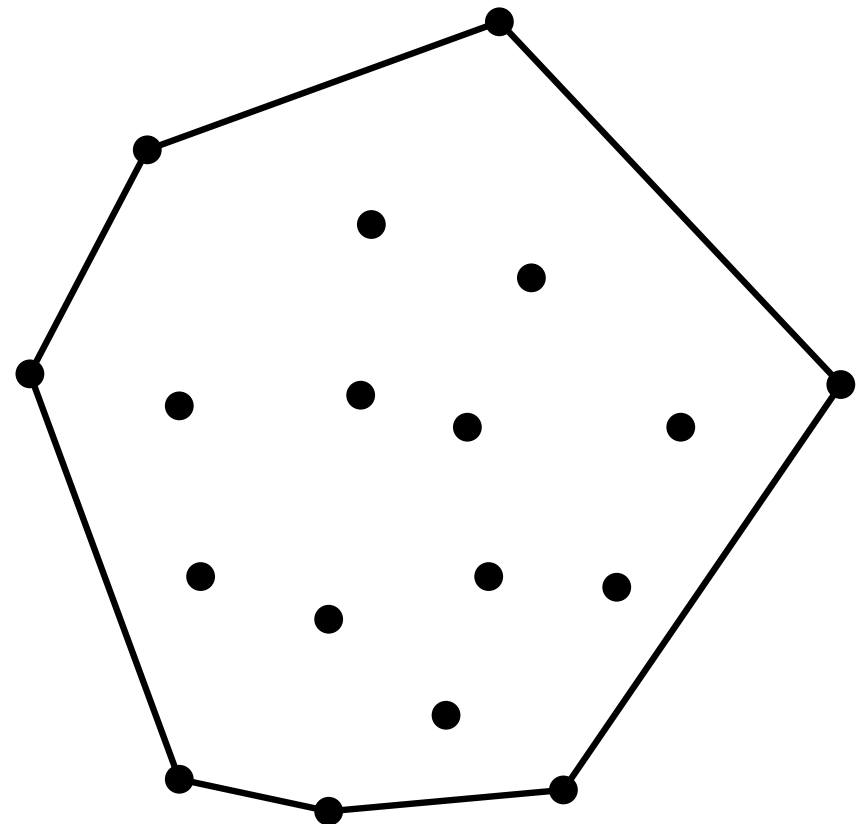
Advance

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

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

Advance

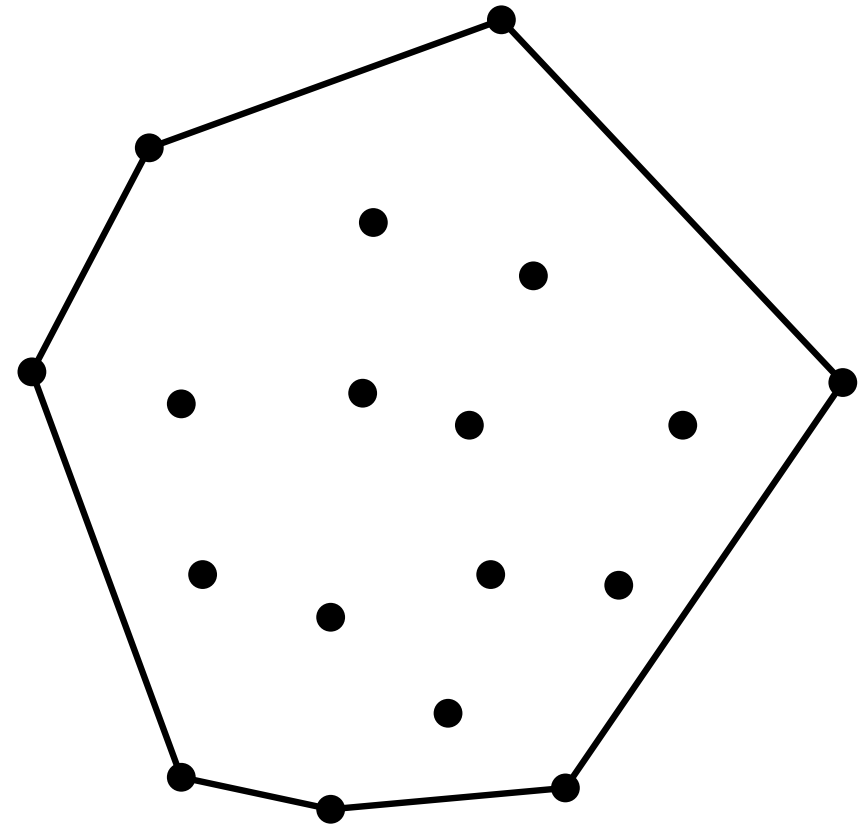
From  $i = 4$  to  $n$ , do:

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  $l$

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



# CONVEX HULL

## Incremental algorithm

Initialization

$$l = p_1, p_2, p_3$$

Advance

From  $i = 4$  to  $n$ , do:

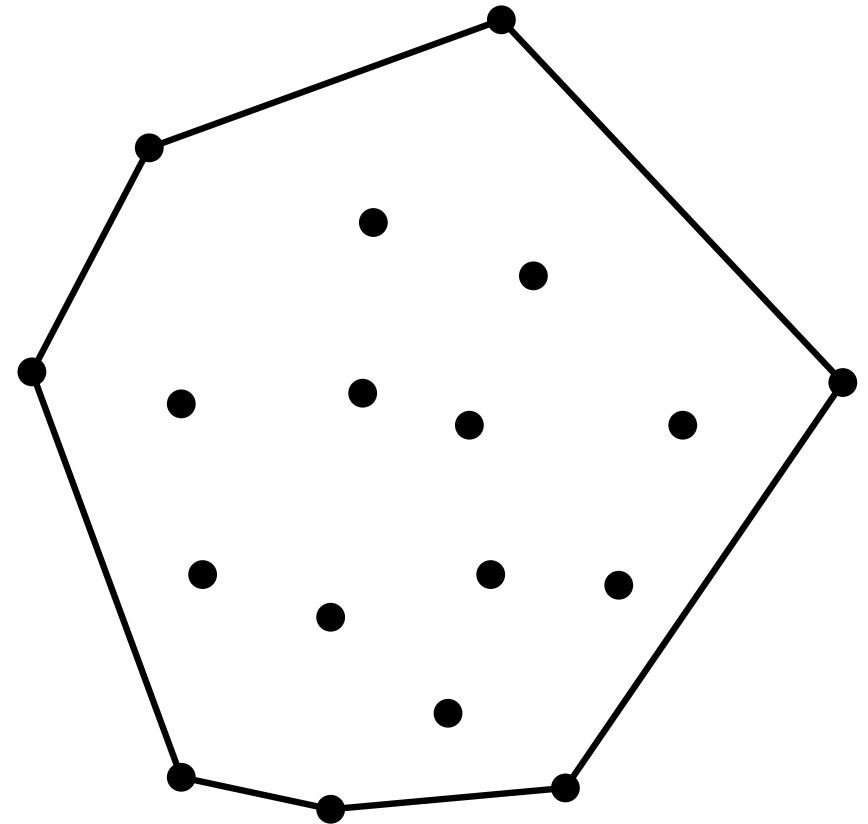
If  $p_i$  lies in the exterior of the polygon defined by  $l$ :

- 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  $l$

**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.



# CONVEX HULL

**Divide-and-conquer algorithm**

# CONVEX HULL

## Divide-and-conquer algorithm

### Initialization

1. Sort the points by abscissae

# CONVEX HULL

## 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

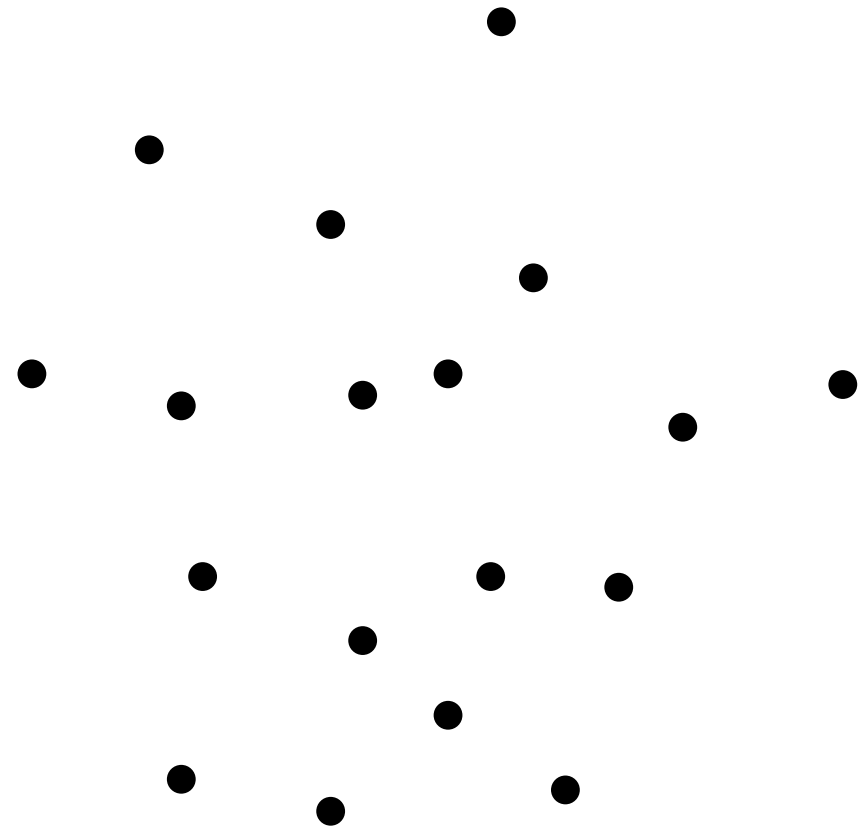
## Divide-and-conquer algorithm

### 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

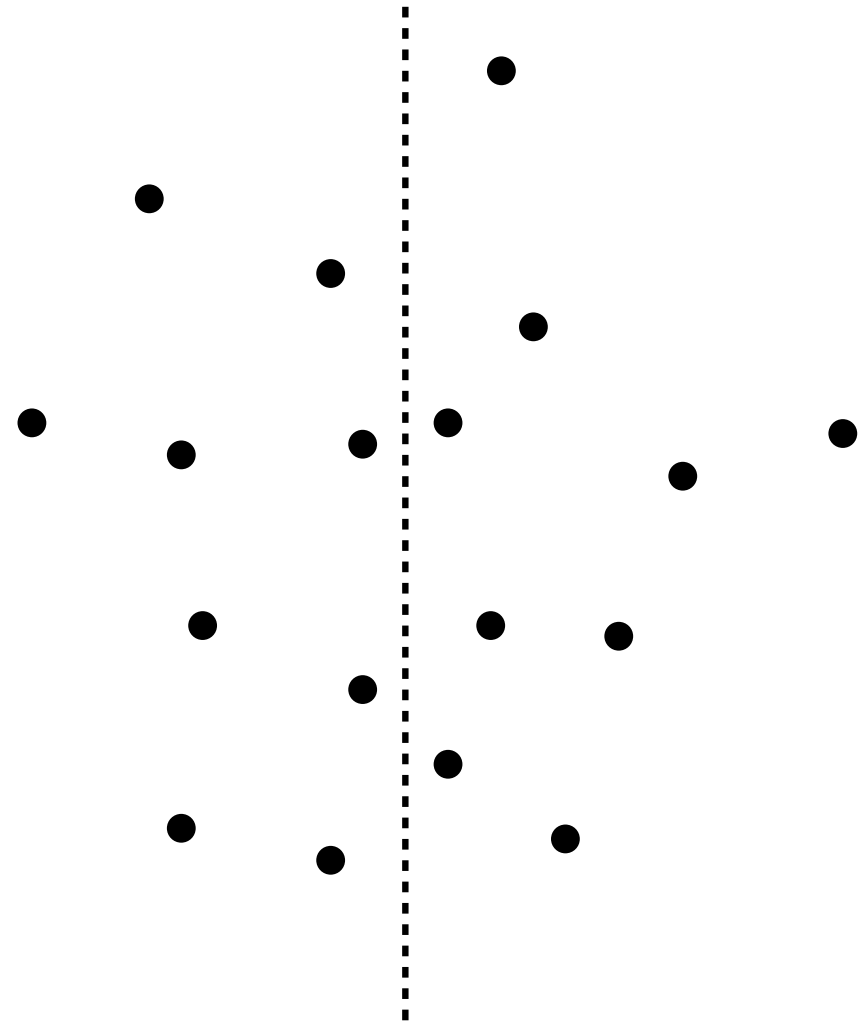
## Divide-and-conquer algorithm

### 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

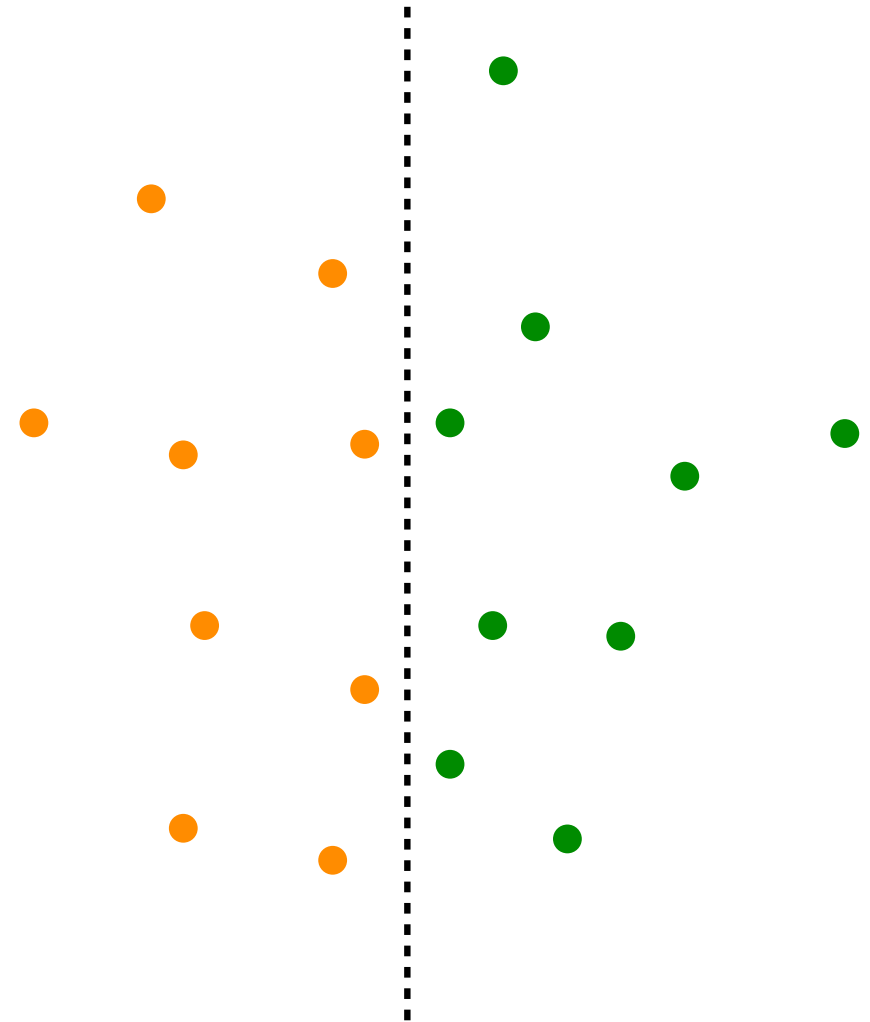
## Divide-and-conquer algorithm

### 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

## Divide-and-conquer algorithm

### Initialization

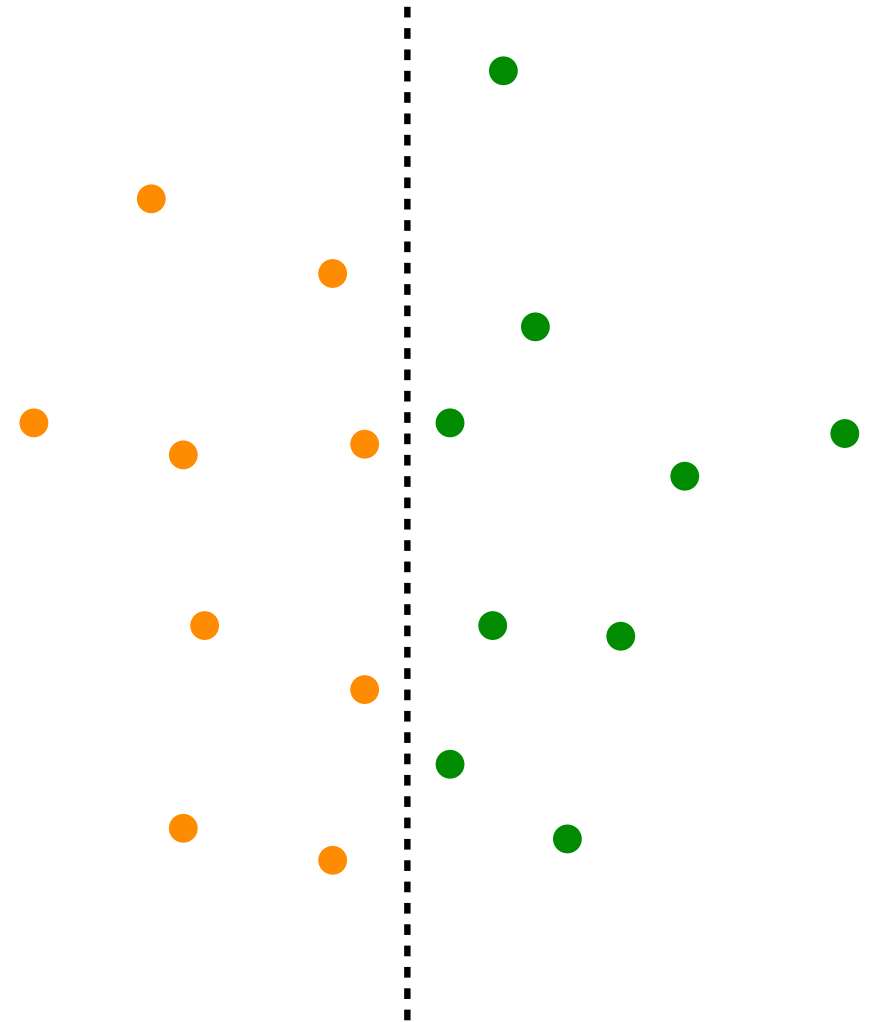
1. Sort the points by abscissae

### 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

## Divide-and-conquer algorithm

### Initialization

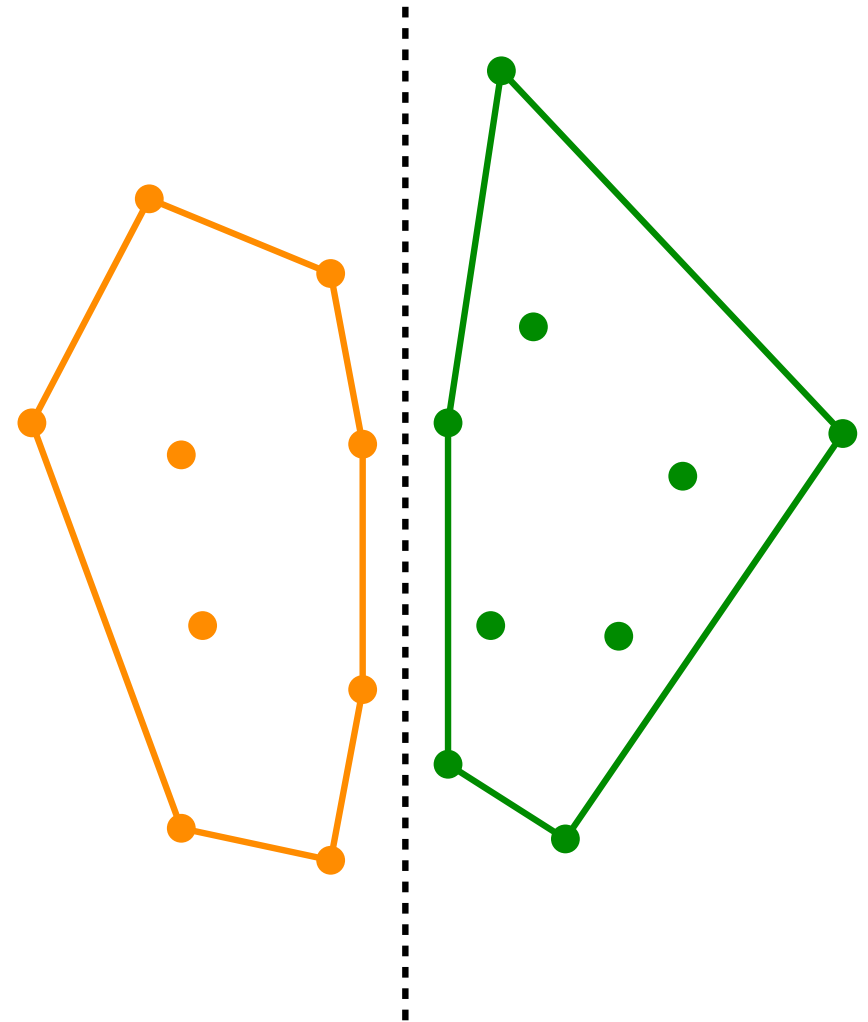
1. Sort the points by abscissae

### 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

## Divide-and-conquer algorithm

### Initialization

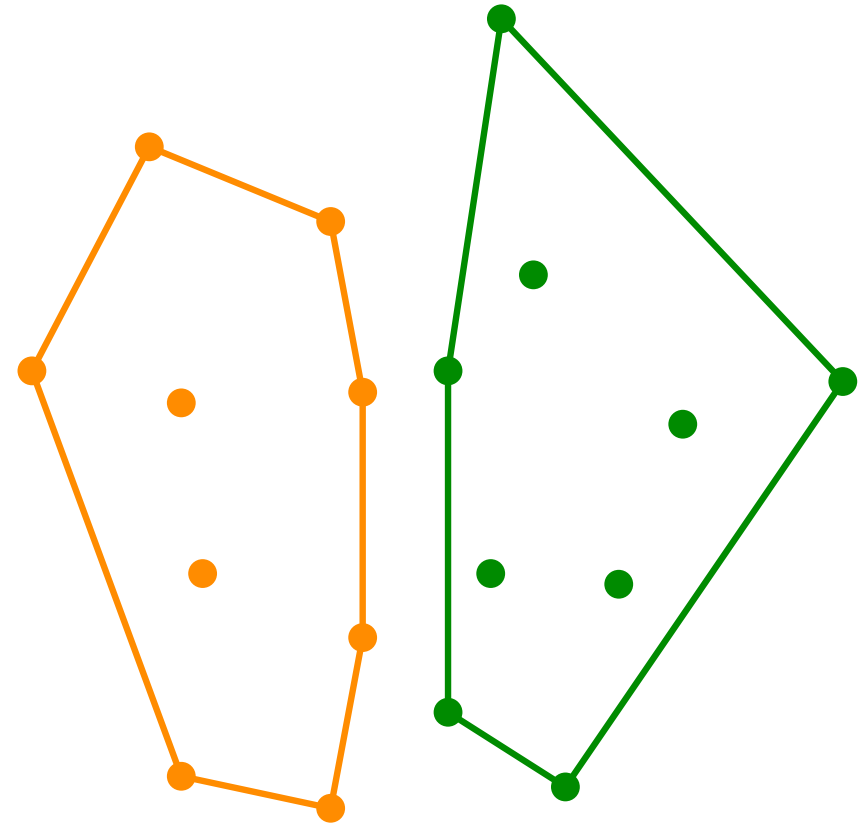
1. Sort the points by abscissae

### 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

## Divide-and-conquer algorithm

### Initialization

1. Sort the points by abscissae

### Division

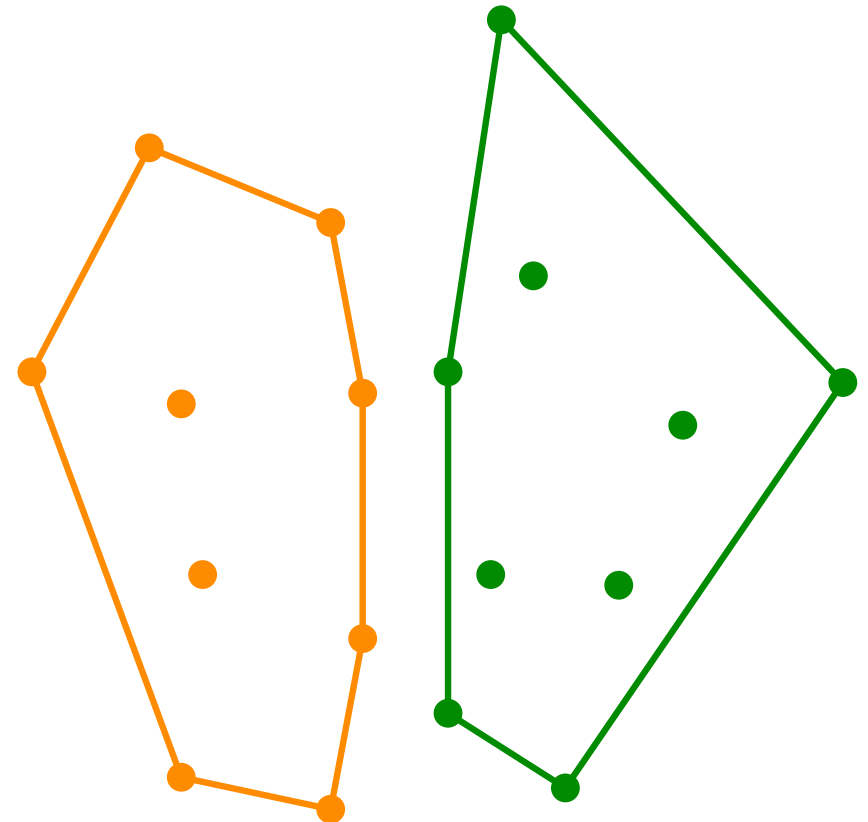
1. Divide the points  $(x_i, y_i)$  into two subsets, wrt the median value of the abscissae

### 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

## Divide-and-conquer algorithm

### Initialization

1. Sort the points by abscissae

### Division

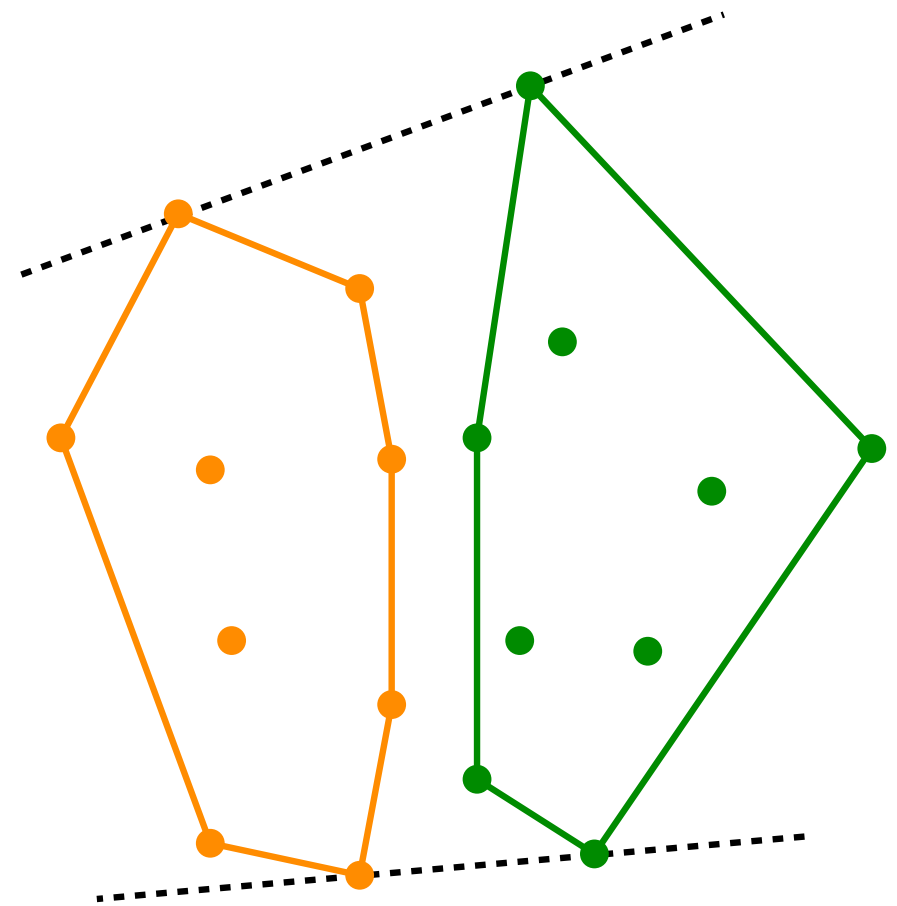
1. Divide the points  $(x_i, y_i)$  into two subsets, wrt the median value of the abscissae

### 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

## Divide-and-conquer algorithm

### Initialization

1. Sort the points by abscissae

### Division

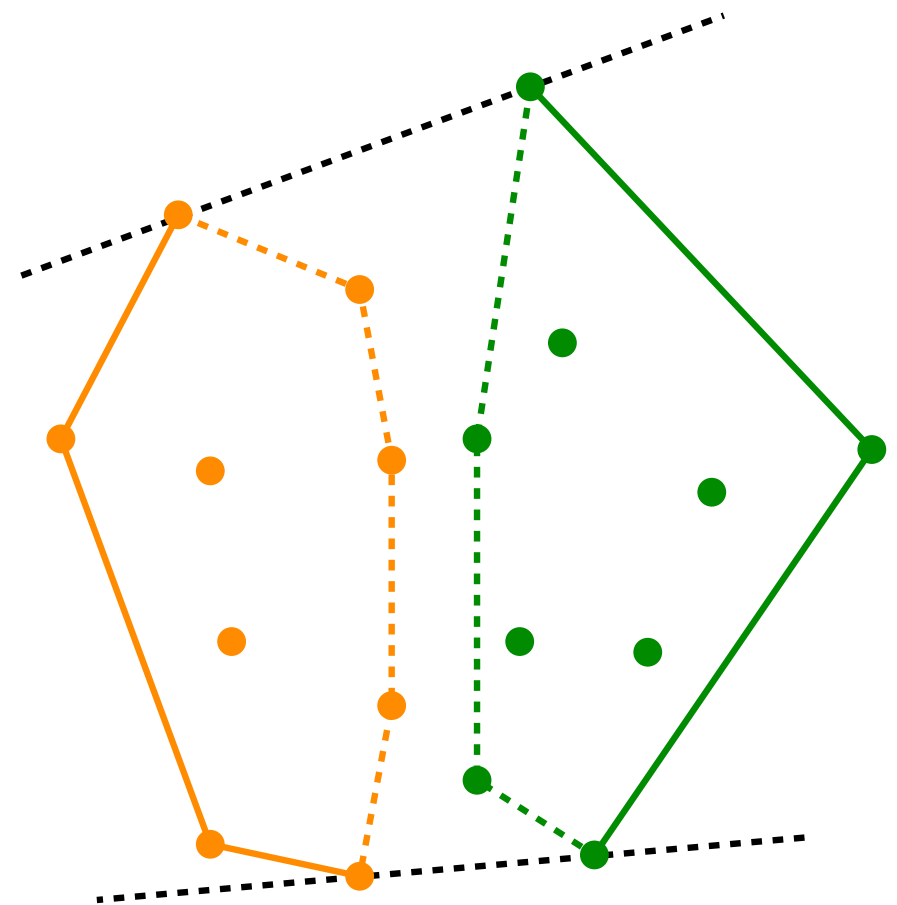
1. Divide the points  $(x_i, y_i)$  into two subsets, wrt the median value of the abscissae

### 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

## Divide-and-conquer algorithm

### Initialization

1. Sort the points by abscissae

### Division

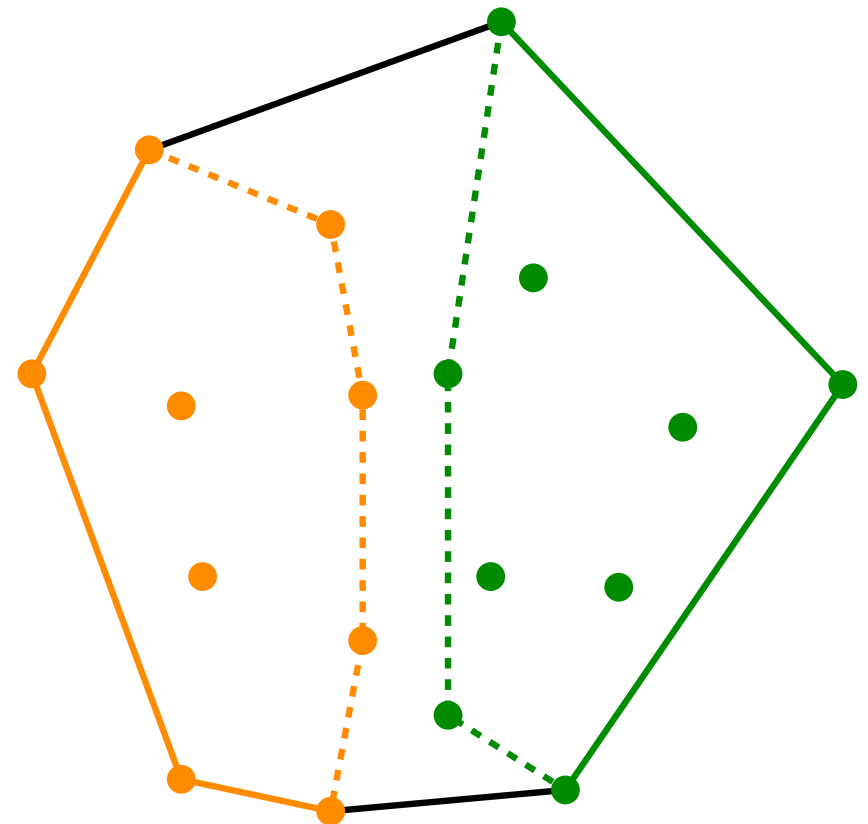
1. Divide the points  $(x_i, y_i)$  into two subsets, wrt the median value of the abscissae

### 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

## Divide-and-conquer algorithm

### Initialization

1. Sort the points by abscissae

### Division

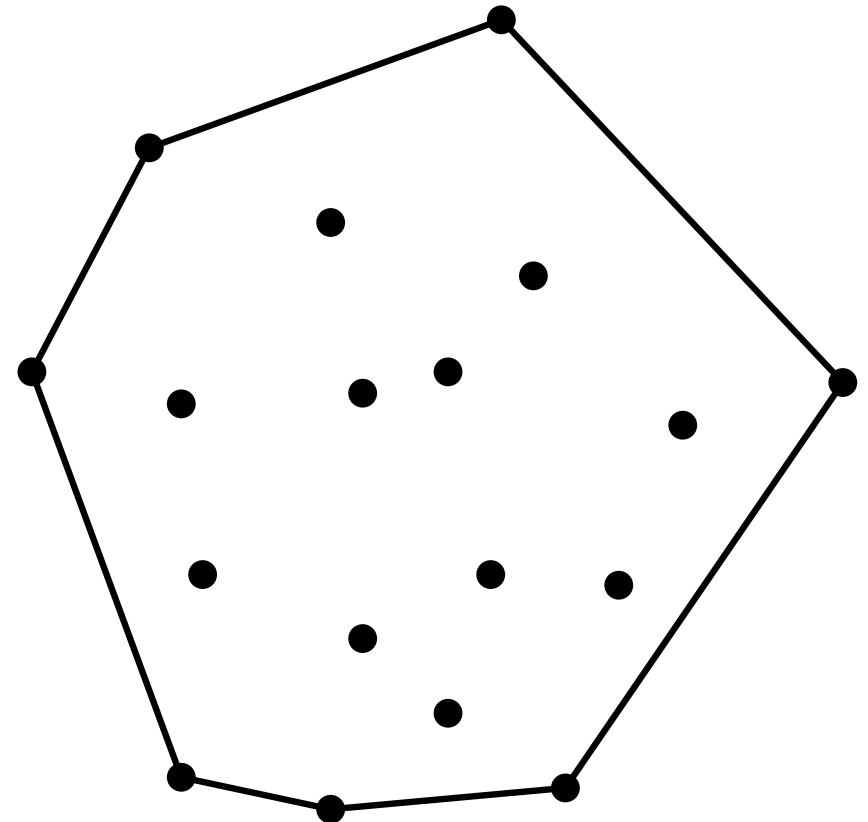
1. Divide the points  $(x_i, y_i)$  into two subsets, wrt the median value of the abscissae

### 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



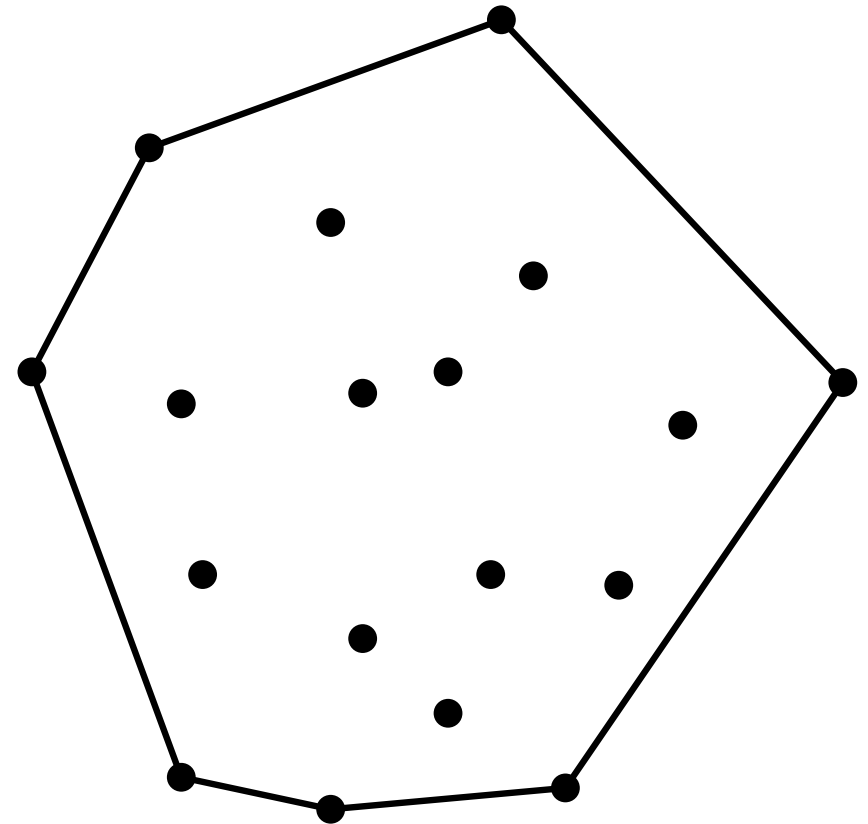


# CONVEX HULL

**Divide-and-conquer algorithm**

**Running time**

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



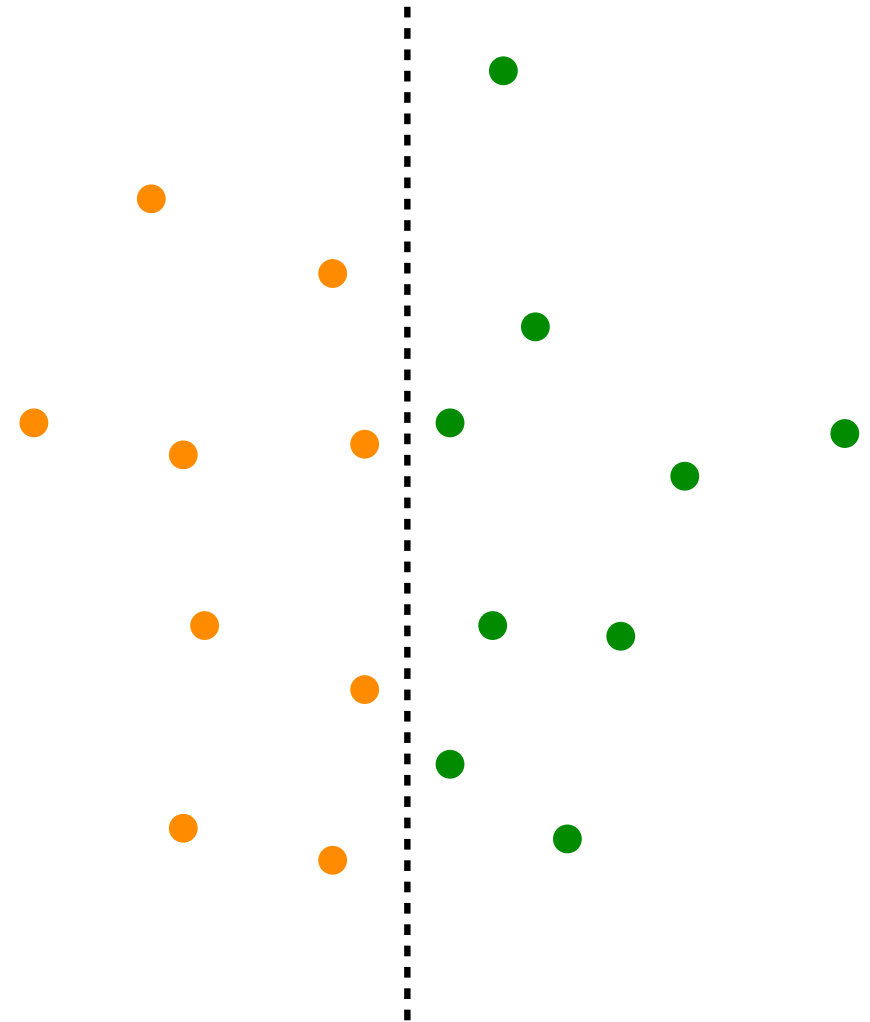
# CONVEX HULL

## Divide-and-conquer algorithm

### Running time

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

Division:  $O(n)$



# CONVEX HULL

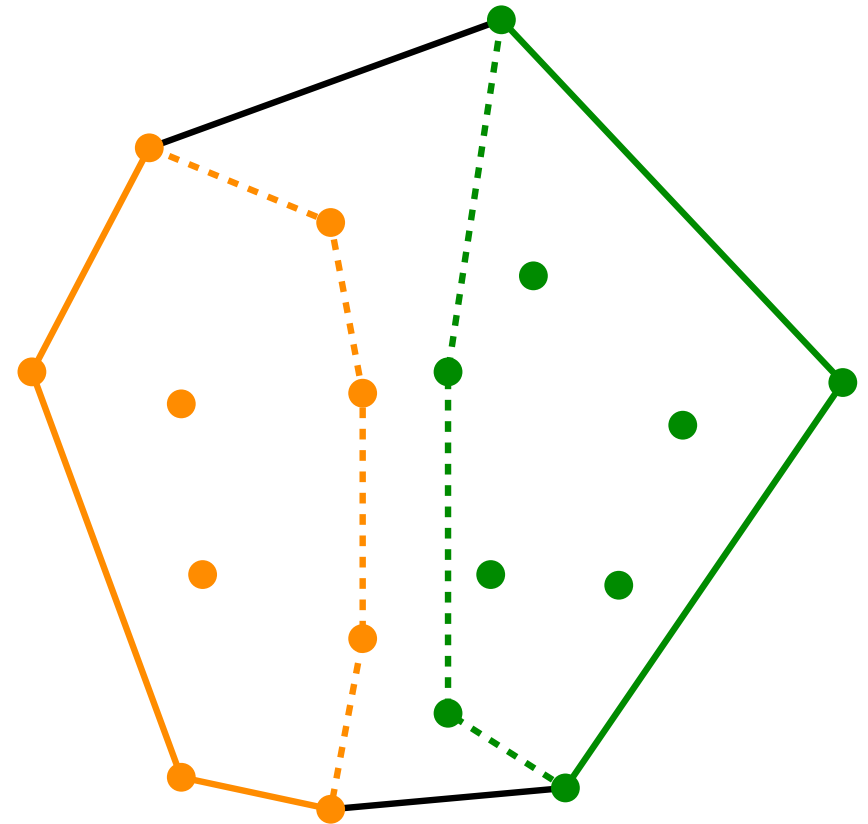
## Divide-and-conquer algorithm

### Running time

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

Division:  $O(n)$

Merge:  $O(n)$



# CONVEX HULL

## Divide-and-conquer algorithm

### Running time

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

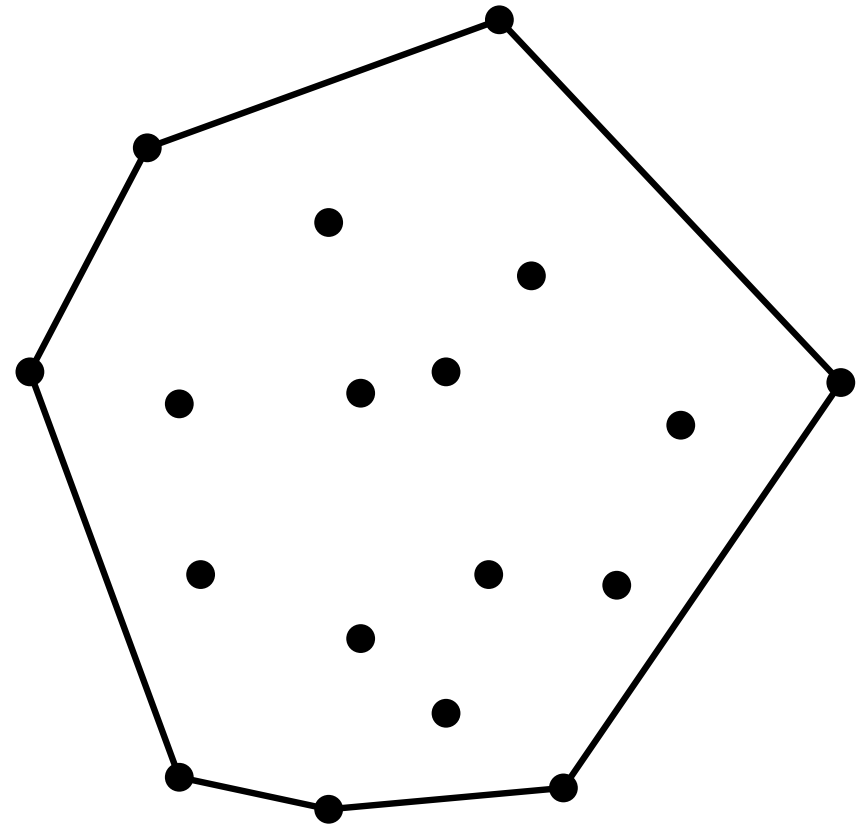
Division:  $O(n)$

Merge:  $O(n)$

Advance:

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

Overall:  $O(n \log n)$



# CONVEX HULL

Lower bound

# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers

# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



**Input:**  $n$  points

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





# CONVEX HULL

## Lower bound

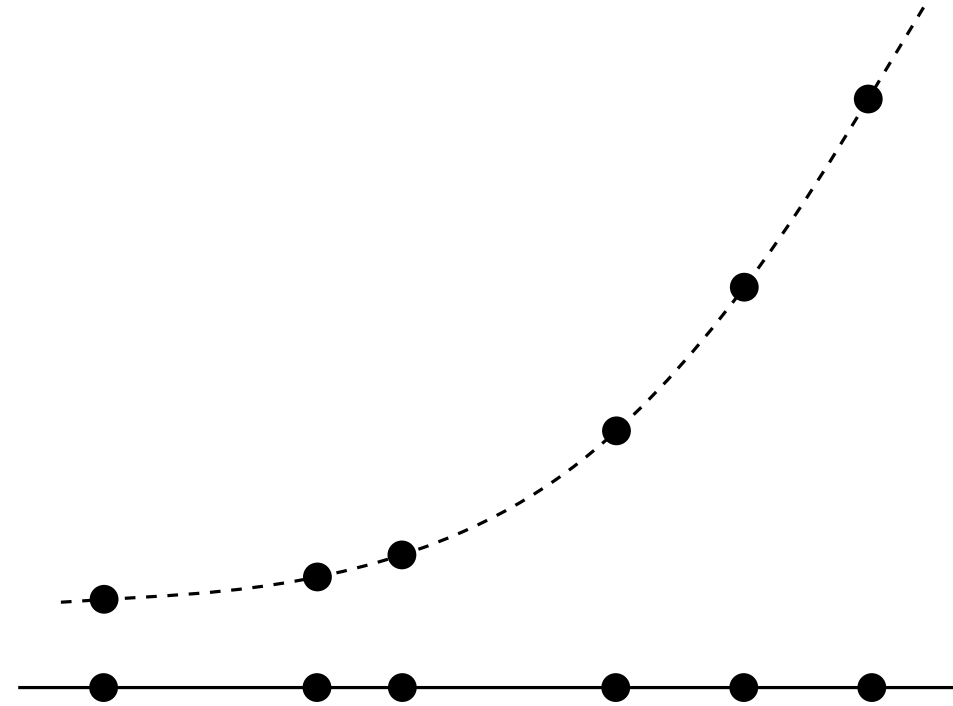
**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



**Input:**  $n$  points

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



# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



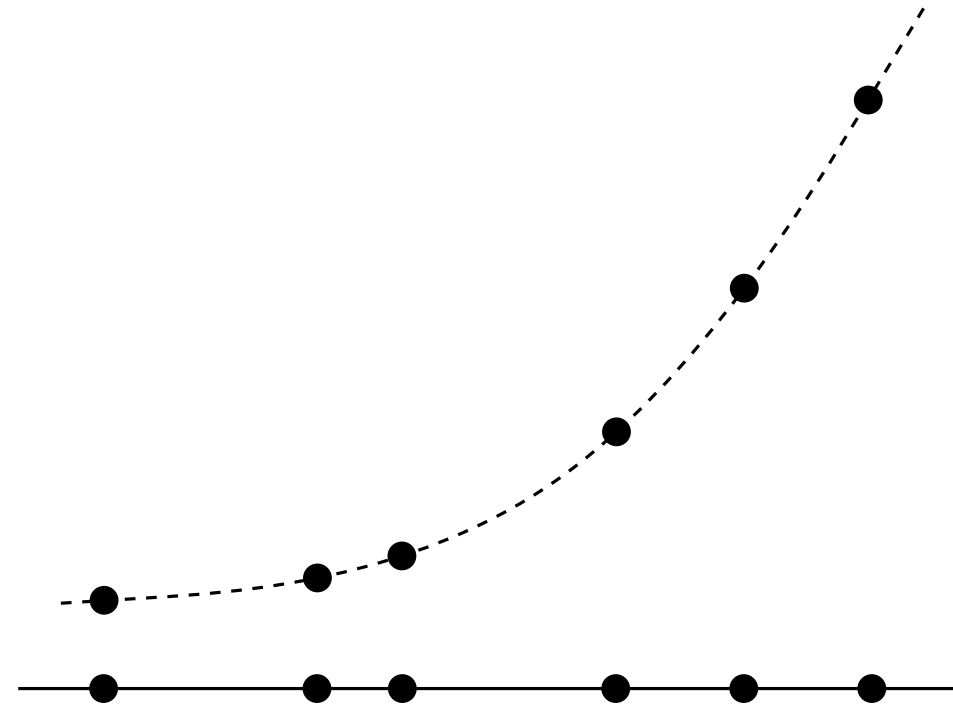
**Input:**  $n$  points

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



**Output:** convex hull of the points

Sorted list of the vertices of the convex hull



# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



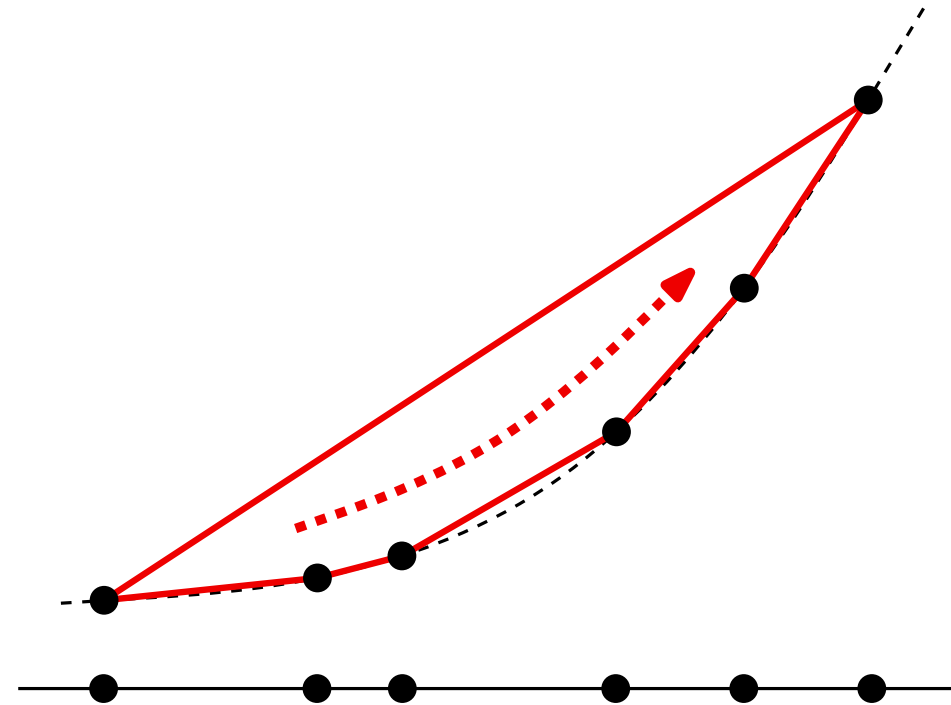
**Input:**  $n$  points

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



**Output:** convex hull of the points

Sorted list of the vertices of the convex hull



# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



**Input:**  $n$  points

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



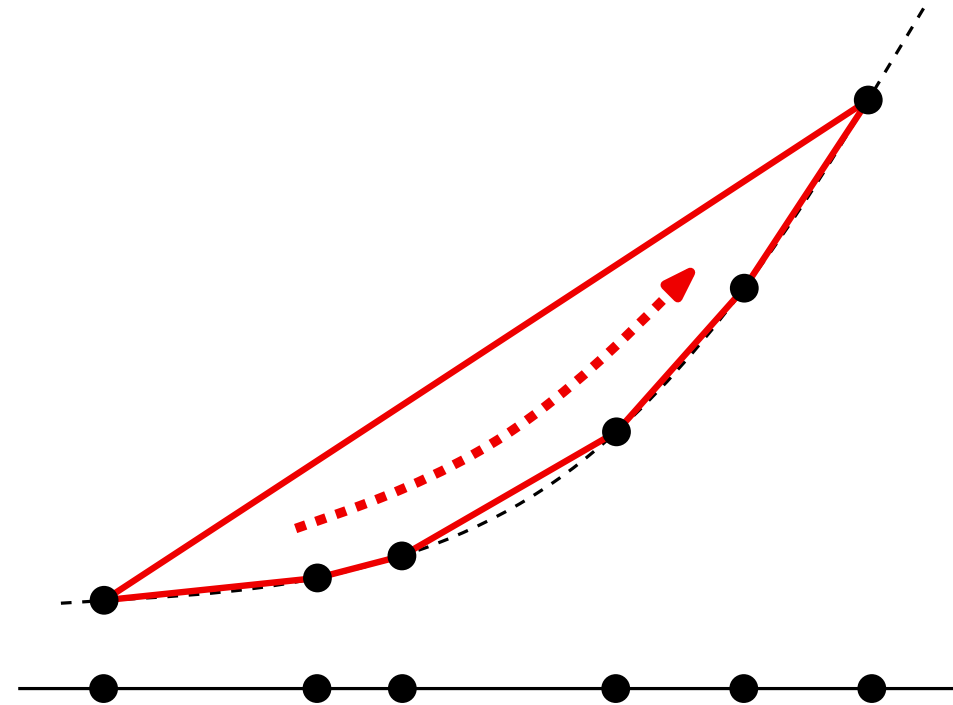
**Output:** convex hull of the points

Sorted list of the vertices of the convex hull



**Output:** sorting the numbers

Sorted list of the numbers  $x_1, \dots, x_n$



# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



**Input:**  $n$  points

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



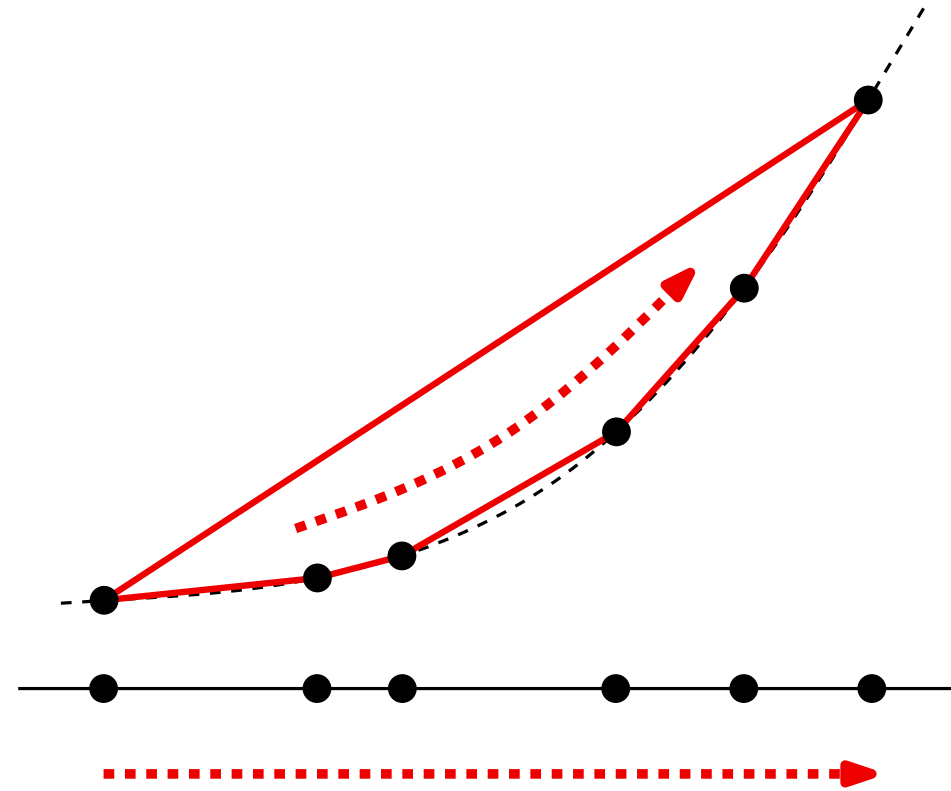
**Output:** convex hull of the points

Sorted list of the vertices of the convex hull



**Output:** sorting the numbers

Sorted list of the numbers  $x_1, \dots, x_n$

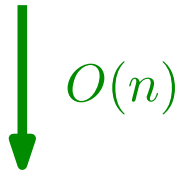


# CONVEX HULL

## Lower bound

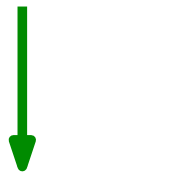
**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



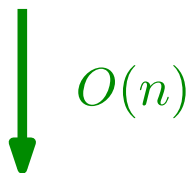
**Input:**  $n$  points

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



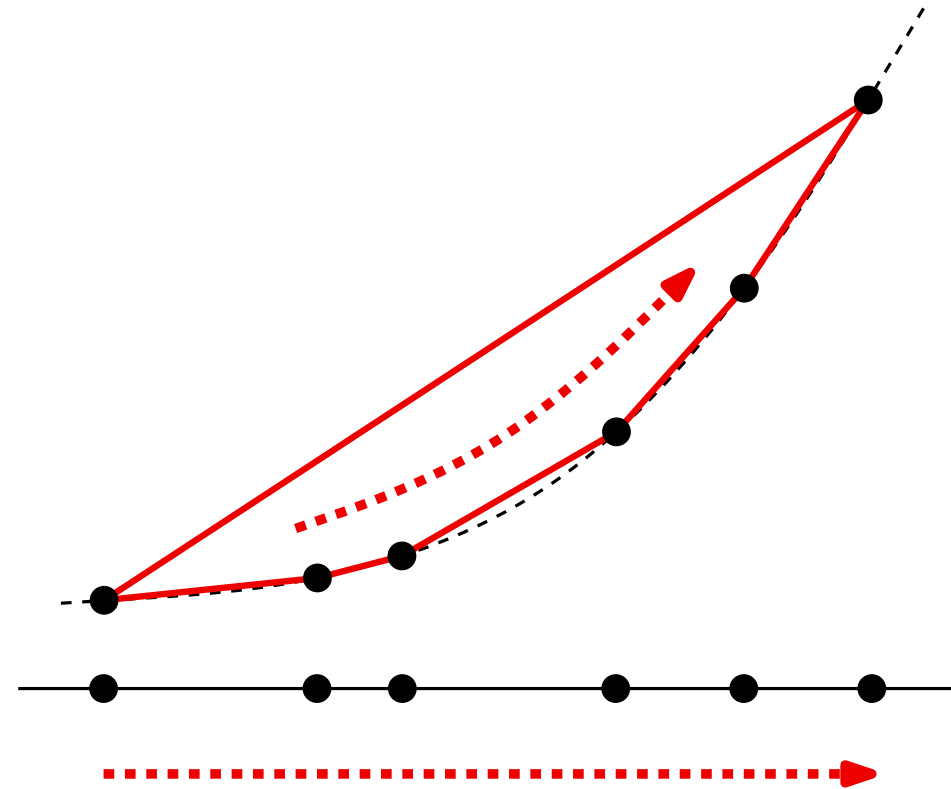
**Output:** convex hull of the points

Sorted list of the vertices of the convex hull



**Output:** sorting the numbers

Sorted list of the numbers  $x_1, \dots, x_n$

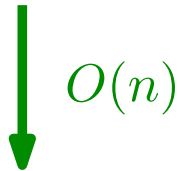


# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers



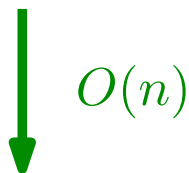
**Input:**  $n$  points

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



**Output:** convex hull of the points

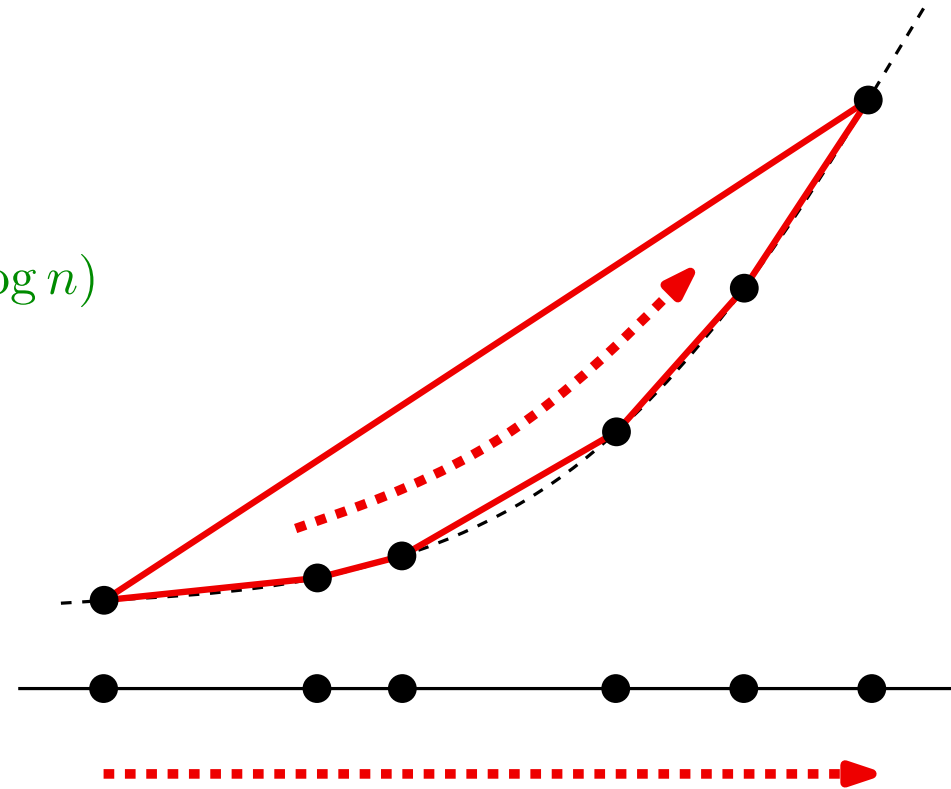
Sorted list of the vertices of the convex hull



**Output:** sorting the numbers

Sorted list of the numbers  $x_1, \dots, x_n$

$\Omega(n \log n)$



# CONVEX HULL

## Lower bound

**Input:**  $n$  real numbers

$x_1, \dots, x_n$  real numbers

$O(n)$

**Input:**  $n$  points

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

$\Omega(n \log n)$

**Output:** convex hull of the points

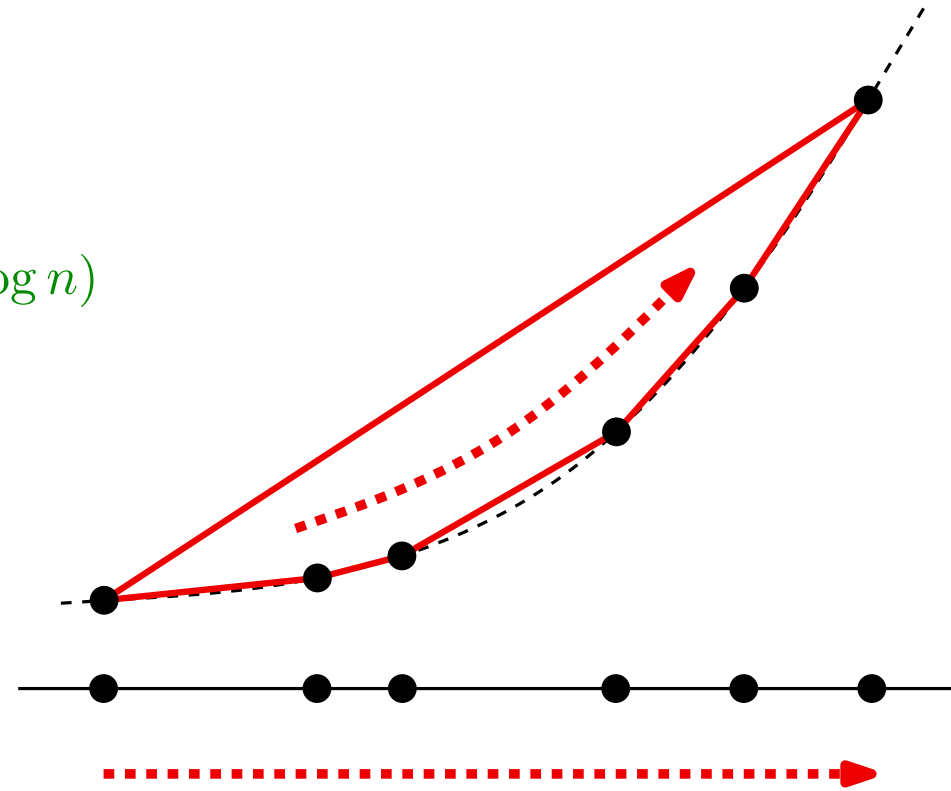
Sorted list of the vertices of the convex hull

$O(n)$

**Output:** sorting the numbers

Sorted list of the numbers  $x_1, \dots, x_n$

$\Omega(n \log n)$





# CONVEX HULL

## Extensions

# CONVEX HULL

## Extensions

- Convex hull of a set of  $n$  points in 3D  
(proposed for a theory presentation)
  - Gift wrapping
  - Divide-and-conquer
  - Incremental
- Convex hull of a simple polygon  
(proposed for a theory presentation)
  - 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?

# CONVEX HULL

## SOME LINKS TO PLAY WITH THE CONSTRUCTION OF CONVEX HULLS

In 2D:

[http://www.dma.fi.upm.es/recursos/aplicaciones/geometria\\_computacional\\_y\\_grafos/](http://www.dma.fi.upm.es/recursos/aplicaciones/geometria_computacional_y_grafos/)

In 3D:

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

# CONVEX HULL

## SOME LINKS TO PLAY WITH THE CONSTRUCTION OF CONVEX HULLS

In 2D:

[http://www.dma.fi.upm.es/recursos/aplicaciones/geometria\\_computacional\\_y\\_grafos/](http://www.dma.fi.upm.es/recursos/aplicaciones/geometria_computacional_y_grafos/)

In 3D:

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

## TO LEARN MORE

- 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.