

# Envolvente Convexo

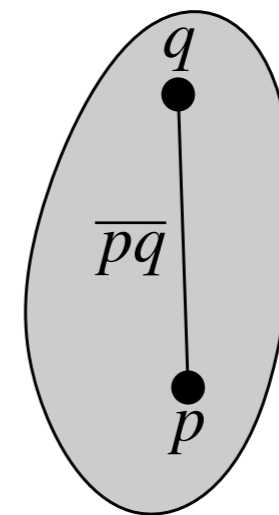
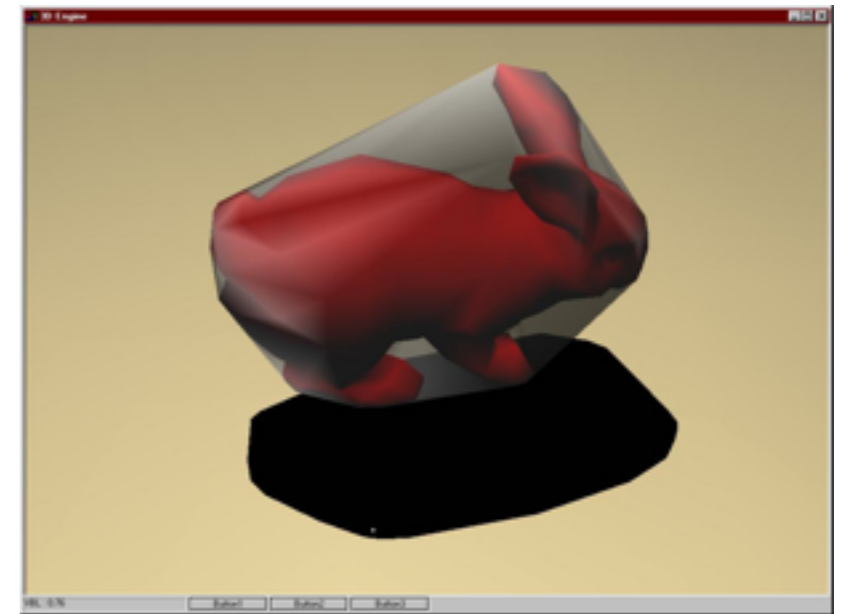
comp-420

# Algunas referencias

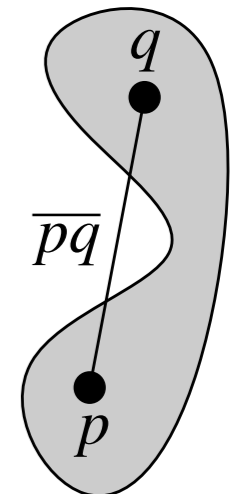
- M. de Berg, O. Cheong, M. van Kreveld, M. Overmars. **Computational Geometry Algorithms and Applications**. 3rd. edition. Springer 2008.
- S.L. Devadoss, J.O'Rourke. **Discrete and Computational Geometry**. Princeton University Press. 2011.
- <http://www.cs.uu.nl/docs/vakken/ga/slides1.pdf>
- <http://jeffe.cs.illinois.edu/teaching/compgeom/notes/O1-convexhull.pdf>
- [http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg\\_ss\\_07w/Webprojects/Chen\\_hull/applications.htm](http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_ss_07w/Webprojects/Chen_hull/applications.htm)

# Envolvente convexo

- ▶ Uno de los primeros problemas estudiados en la geometría computacional.
- ▶ Un conjunto  $S$  del plano, se llama convexo si y solo si, para cualquier par de puntos  $p, q \in S$ , el segmento  $pq$  está contenido completamente en  $S$ .



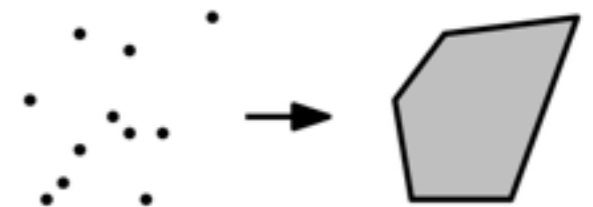
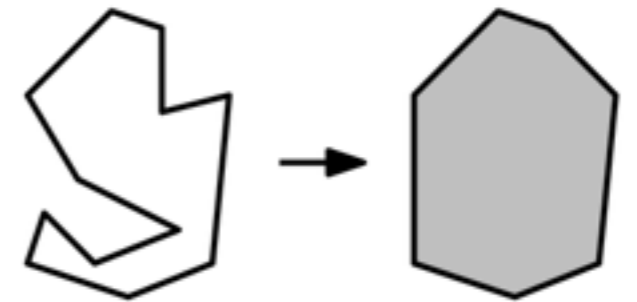
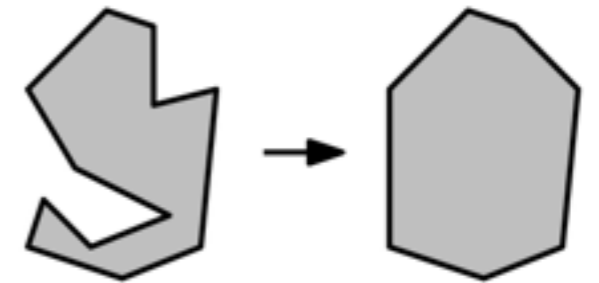
convexo



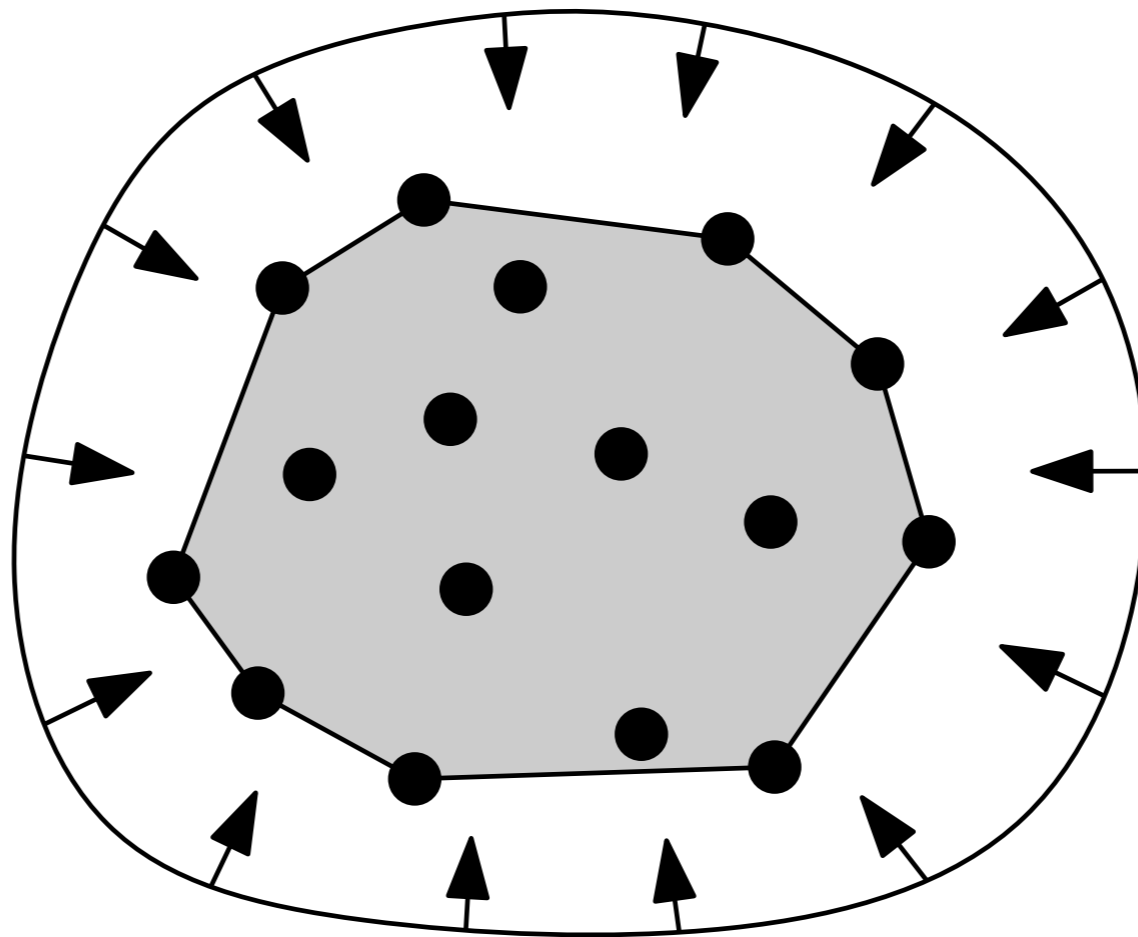
no convexo

# Envolvente convexo

- ▶ Para cualquier subconjunto del plano  $S$ , su envolvente convexo  $CH(S)$  es el conjunto convexo más pequeño que pueda contener a  $S$ .

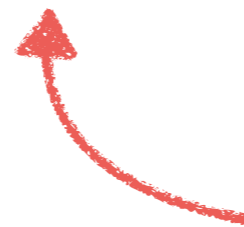


# Envolvente convexo



## DEFINICIÓN:

- El envolvente convexo de  $S$ , denotado como  $CH(S)$ , es la intersección de todas las regiones convexas que contienen a  $S$ .



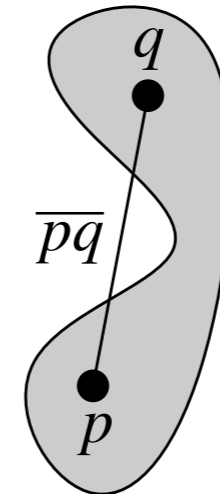
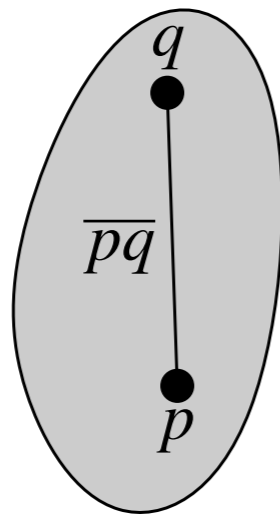
no es útil computacionalmente

# Otra forma de ver $CH(S)$

- Podemos también definir como el polígono convexo más grande cuyos vértices son puntos en  $S$ .

# Región convexa

- Región convexa ssi cualquiera dos puntos  $p$  y  $q$  en la región  $R$  son mutuamente visibles.



- El segmento de recta entre  $p$  y  $q$  está también en  $R$ .
- Tratamos de capturar la “visibilidad” de todos los puntos del conjunto  $S$ .

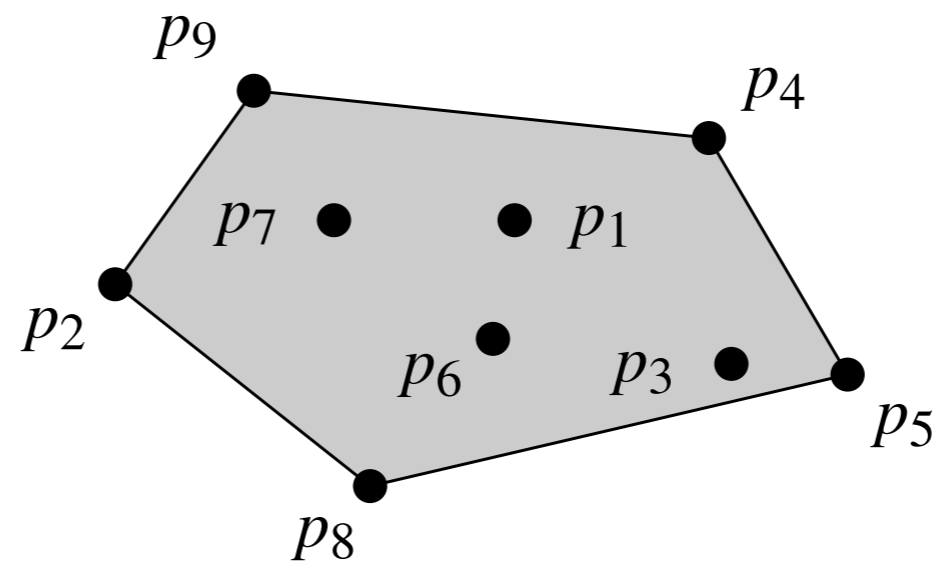


## TEOREMA:

- Para un conjunto de puntos  $S = \{p_1, \dots, p_n\}$ , el envolvente convexo de  $S$  es el conjunto de todas las combinaciones convexas de  $S$ .

# Envolvente convexo

- Dado un conjunto de puntos  $P = \{p_1, p_2, \dots, p_n\}$  en el plano, calcular una lista que contenga aquellos puntos de  $P$  que sean vértices de  $\mathcal{CH}(P)$ , en orden de las manecillas del reloj.
- **entrada:** conjunto de puntos  $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$ .
- **salida:** representación del envolvente convexo  $p_4, p_5, p_8, p_2, p_9$ .



# Concretamente ...

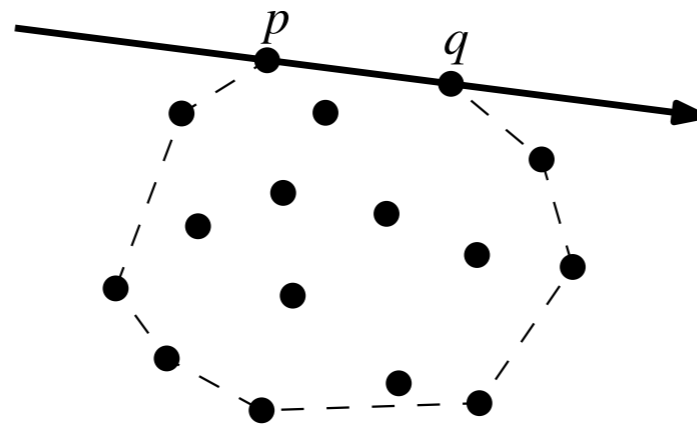
- ▶ Representamos a los puntos de  $S$  con sus coordenadas cartesianas, en dos arreglos  $X[1..n]$  y  $Y[1..n]$ .
- ▶  $CH(S)$  se puede representar como una lista circular de vértices en orden anti-horario (arbitrariamente).
- ▶ Si el punto  $i$  es un vértice de  $CH(S)$ ,  $next[i]$  es el índice del punto que sigue en  $CH(S)$  en sentido antihorario y  $pred[i]$ , es el índice del punto siguiente en sentido horario.
- ▶ No importa qué vértice tomemos como cabeza de la lista.
- ▶ Para simplificar supondremos que no hay 3 vértices co-lineales.

# Casos simples

- ▶ Cuando  $n=1$ , ¿Cómo es  $CH(s)$ ?
  - ▶ ¿y cuando  $n=2$ ?
  - ▶ ¿y cuando  $n=3$ ?
- 
- ▶ El test  $CCW$  o  $CW$  juega el mismo papel en el algoritmo de  $CH$  que una comparación en los algoritmos de ordenamiento (sorting).

# Envolvente convexo (convex hull)

- El algoritmo para calcular  $\mathcal{CH}(P)$  se basa en su estructura de aristas:
- Los puntos extremos  $p$  y  $q$  de cada arista son puntos de  $P$ .
- Si esta arista está dirigida de tal forma que  $\mathcal{CH}(P)$  esté a la derecha, entonces todos los puntos de  $P$  deben estar a la derecha de esta línea.
- El argumento inverso es también cierto, si todos los puntos de  $P \setminus \{p, q\}$  está a la derecha de la línea dirigida entre  $p$  y  $q$ , entonces  $\overline{pq}$  es una arista de  $\mathcal{CH}(P)$ .



# Algoritmo

**Algorithm** SLOWCONVEXHULL( $P$ )

*Input.* A set  $P$  of points in the plane.

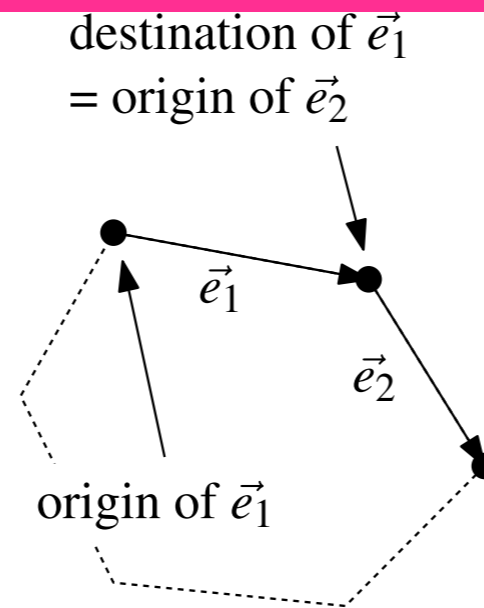
*Output.* A list  $\mathcal{L}$  containing the vertices of  $\mathcal{CH}(P)$  in clockwise order.

1.  $E \leftarrow \emptyset$ .
2. **for** all ordered pairs  $(p, q) \in P \times P$  with  $p$  not equal to  $q$
3.     **do**  $valid \leftarrow \mathbf{true}$
4.         **for** all points  $r \in P$  not equal to  $p$  or  $q$
5.             **do if**  $r$  lies to the left of the directed line from  $p$  to  $q$
6.                 **then**  $valid \leftarrow \mathbf{false}$ .
7.             **if**  $valid$  **then** Add the directed edge  $\vec{pq}$  to  $E$ .
8. From the set  $E$  of edges construct a list  $\mathcal{L}$  of vertices of  $\mathcal{CH}(P)$ , sorted in clockwise order.

# Envolvente convexo (convex hull)

- ¿Cómo se puede verificar que los puntos estén a la derecha o a la izquierda de una línea dirigida?
- en los algoritmos de geometría computacional suponemos esta operación como una de **tiempo de ejecución constante**.
- ¿Cómo se puede construir una lista, a partir de las aristas, ordenada en el orden de las manecillas del reloj?
- Las aristas en  $E$  son dirigidas, por lo que tenemos un **origen y un destino**.
- Como las aristas están dirigidas de tal forma que los otros puntos estén a su derecha, el vértice destino viene después del origen cuando están ordenados en **sentido de las manecillas del reloj**.

# Envolvente convexo (convex hull)

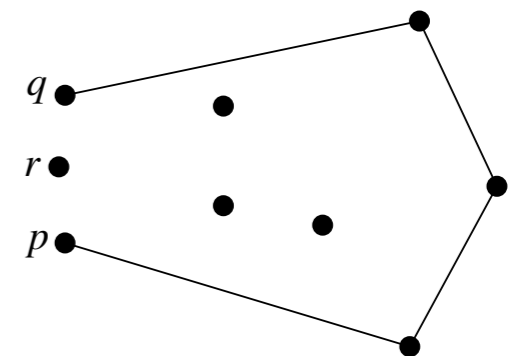
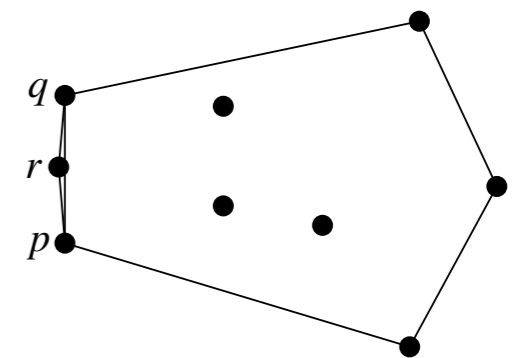
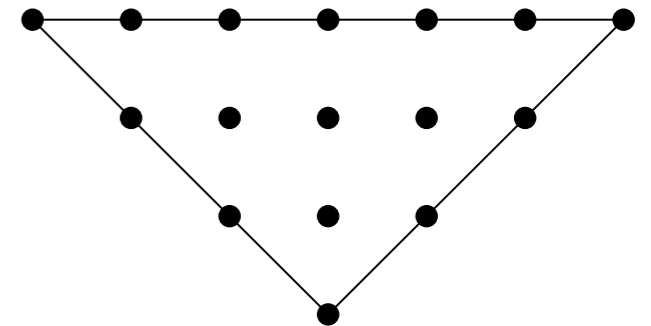


- Una implementación simple del procedimiento de la línea 8 toma  $O(n^2)$  pero se puede mejorar a  $O(n \lg n)$ .
- El resto del algoritmo domina el tiempo total de ejecución.
- Verificamos  $n^2 - n$  pares de puntos. Para cada par miramos  $n - 2$  puntos para encontrar los que están del lado derecho. Esto hace en total  $O(n^3)$ .
- La operación de la línea 8 toma  $O(n^2)$  por lo que la complejidad total del algoritmo es  $O(n^3)$ .



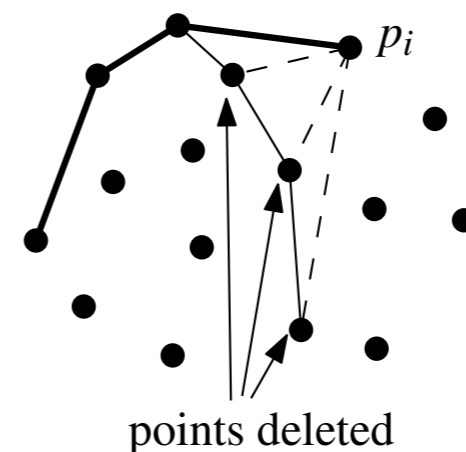
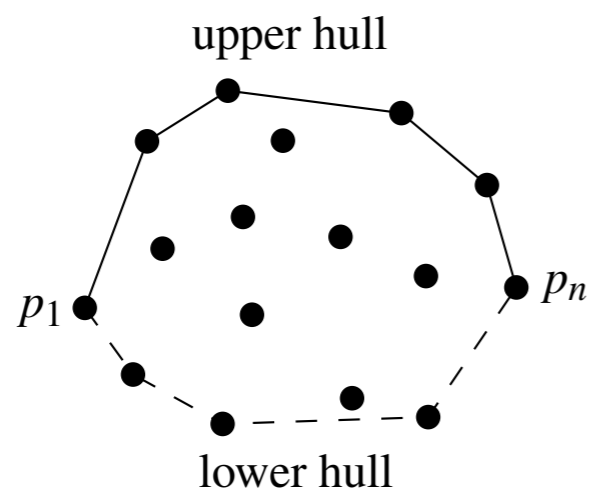
# Envolvente convexo (convex hull)

- Algunos detalles sobre este algoritmo:
- verificar co-linearidad en los puntos que no están a la derecha.
- errores de punto flotante que no permitan determinar de manera correcta la configuración de los puntos en el plano.
- en este caso habrá problemas al tratar de reconstruir la representación del convex hull.



# Envolvente convexo (convex hull)

- Agregamos los puntos uno a uno, actualizando la solución después de cada adición.
- Los puntos se agregan **de izquierda a derecha** por lo que primero se ordenan de acuerdo a su coordenada  $x$ , obteniendo una secuencia ordenada  $p_1, p_2, \dots, p_n$  que establece el orden en que son añadidos.
- Al estar trabajando de izquierda a derecha sería también conveniente tener los puntos de la envolvente convexa ordenados de izquierda a derecha. Como esto no es el caso **encontramos primero la parte superior de la envolvente** (upper hull - que va de izquierda a derecha) y luego el lower hull.



# Algoritmo iterativo

- Observación: de izquierda a derecha, solo hay vueltas a la derecha en el envolvente superior (upper hull)



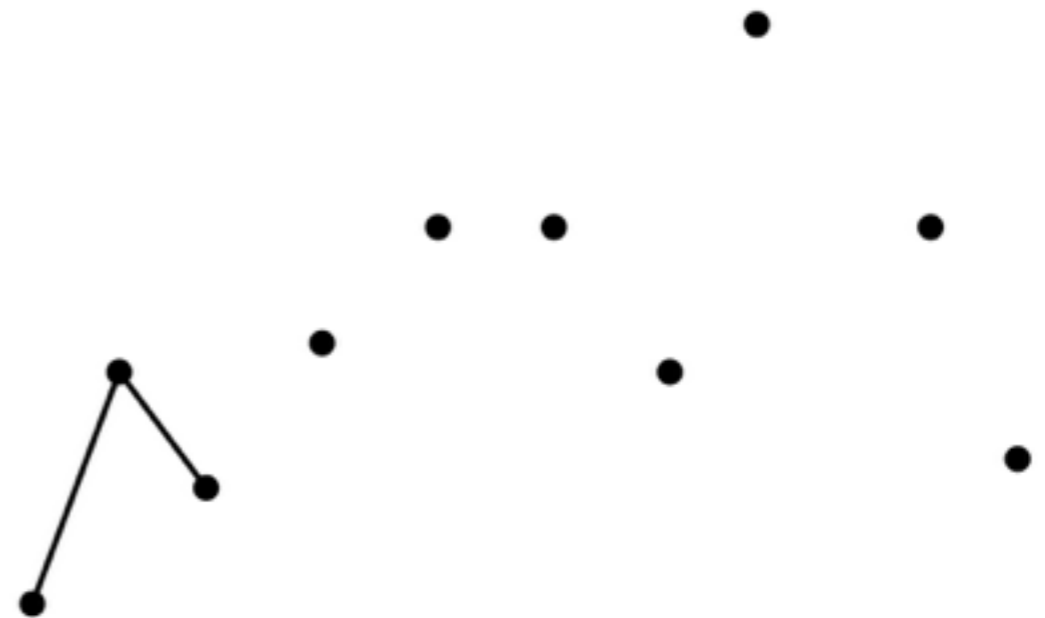
# Algoritmo iterativo

- ▶ Inicializar insertando los puntos más a la izquierda.



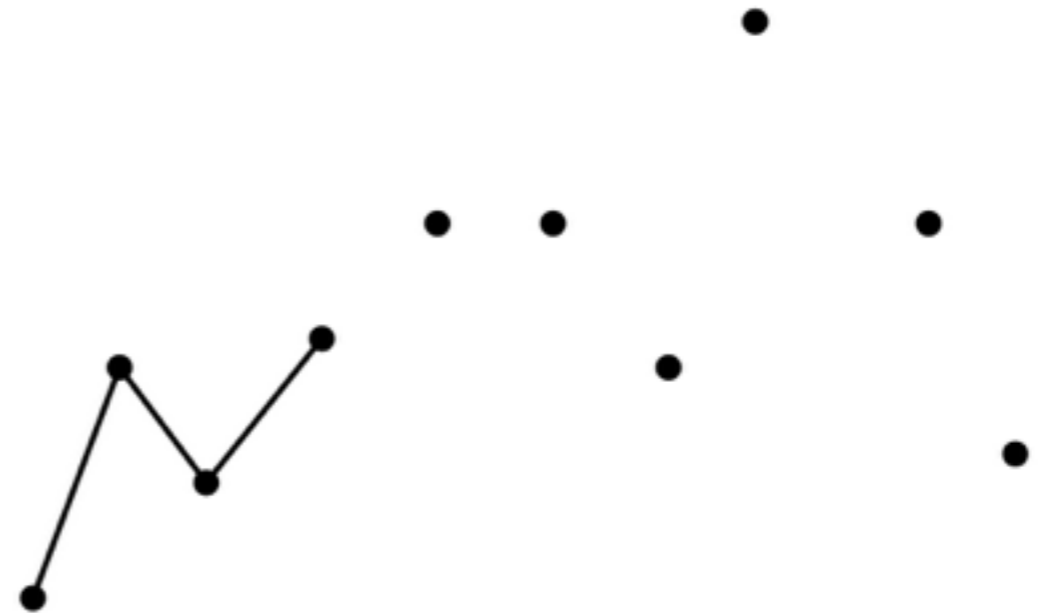
# Algoritmo iterativo

- Si agregamos un tercer punto, habrá una vuelta a la derecha con el punto anterior. Agregarlo.



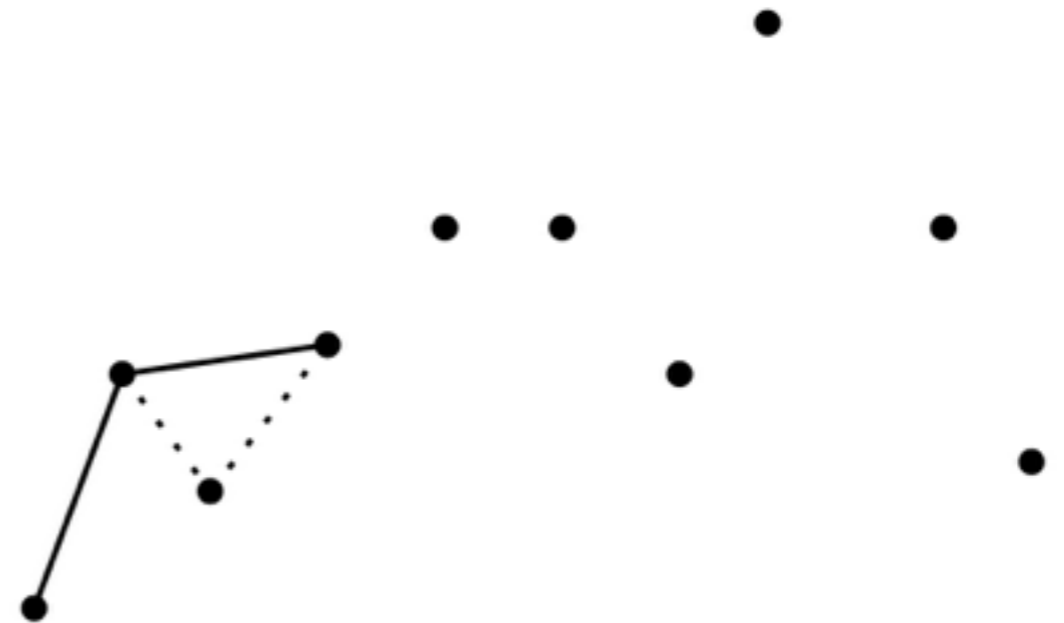
# Algoritmo iterativo

- Si agregamos un cuarto punto, tenemos una vuelta a la izquierda en el tercer punto.



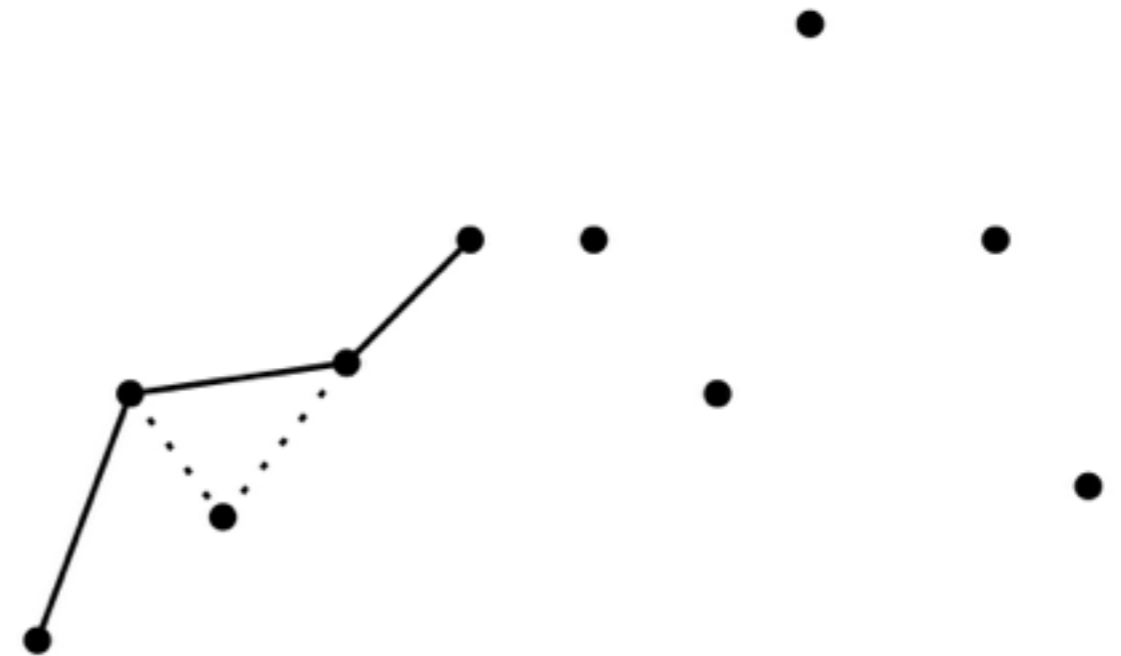
# Algoritmo iterativo

- Entonces removemos el tercer punto del envolvente superior y agregamos el cuarto punto.



# Algoritmo iterativo

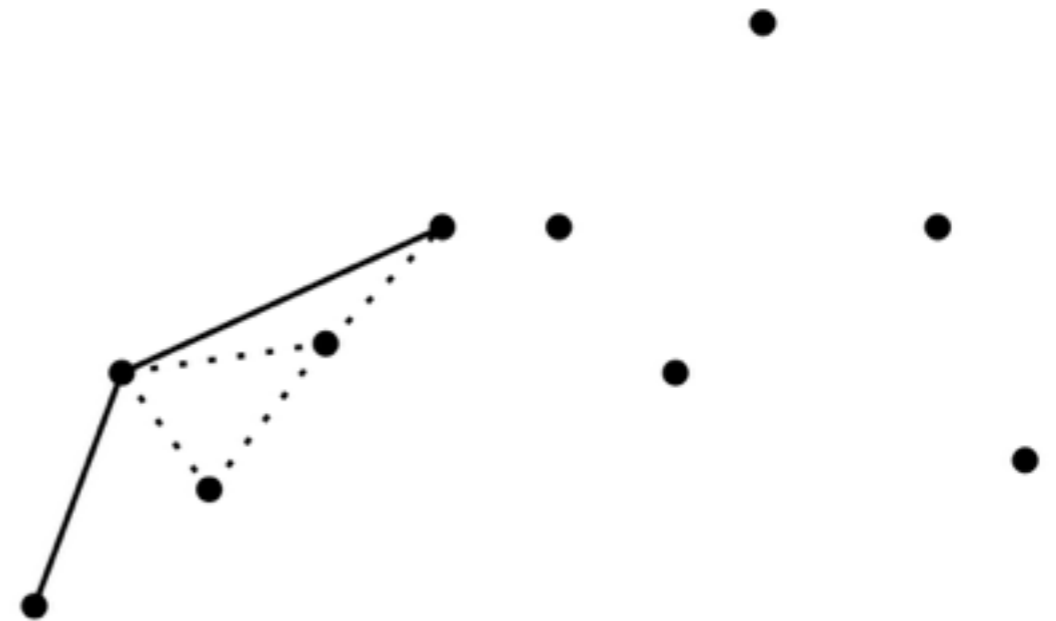
- Si agregamos el quinto punto, tenemos una vuelta a la izquierda en el cuarto punto.





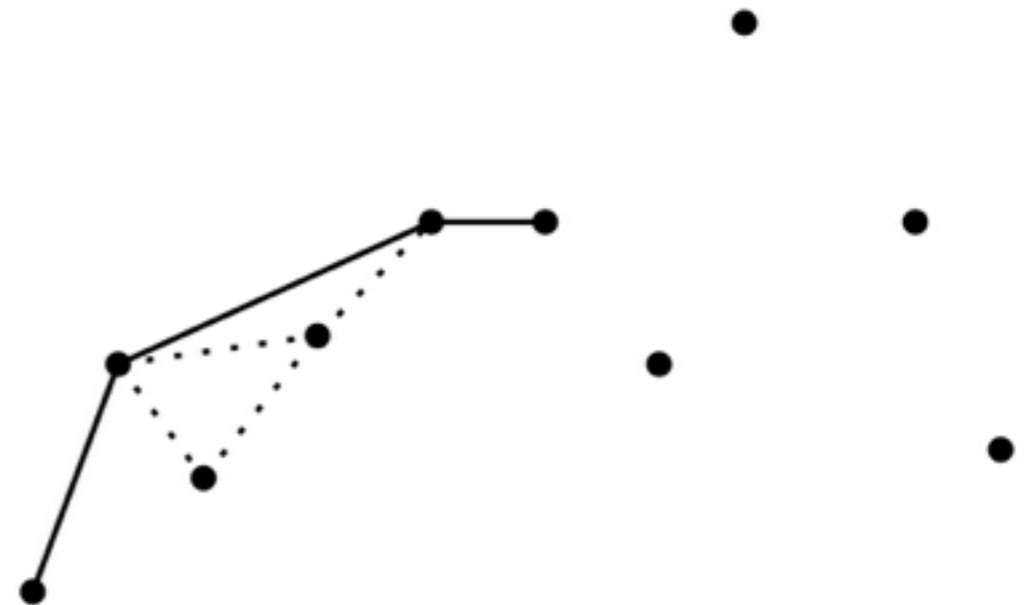
# Algoritmo iterativo

- Entonces quitamos el cuarto punto y agregamos el quinto.



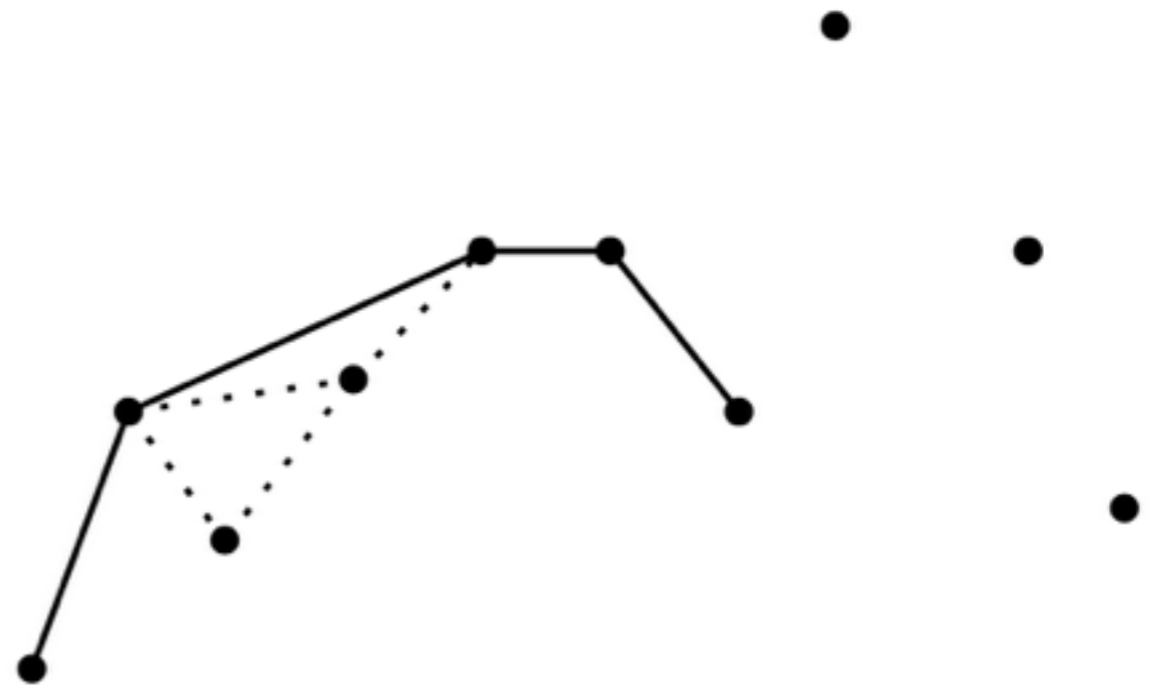
# Algoritmo iterativo

- Si agregamos el sexto punto, tenemos una vuelta a la derecha en el quinto punto así que agregamos al sexto punto.



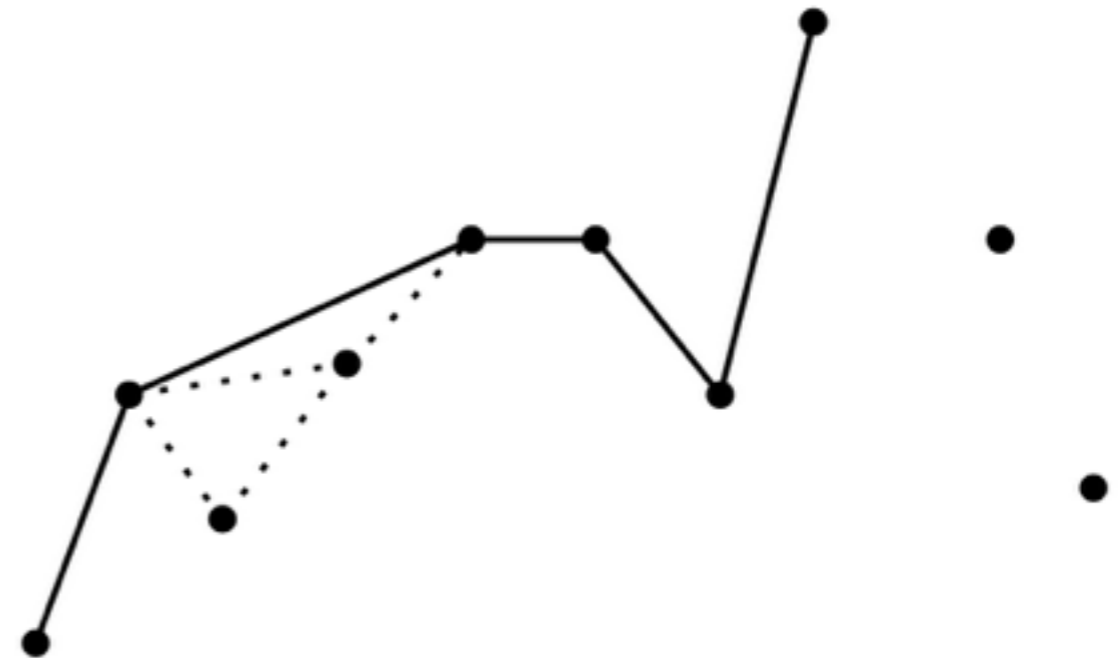
# Algoritmo iterativo

► Agregamos el séptimo punto...



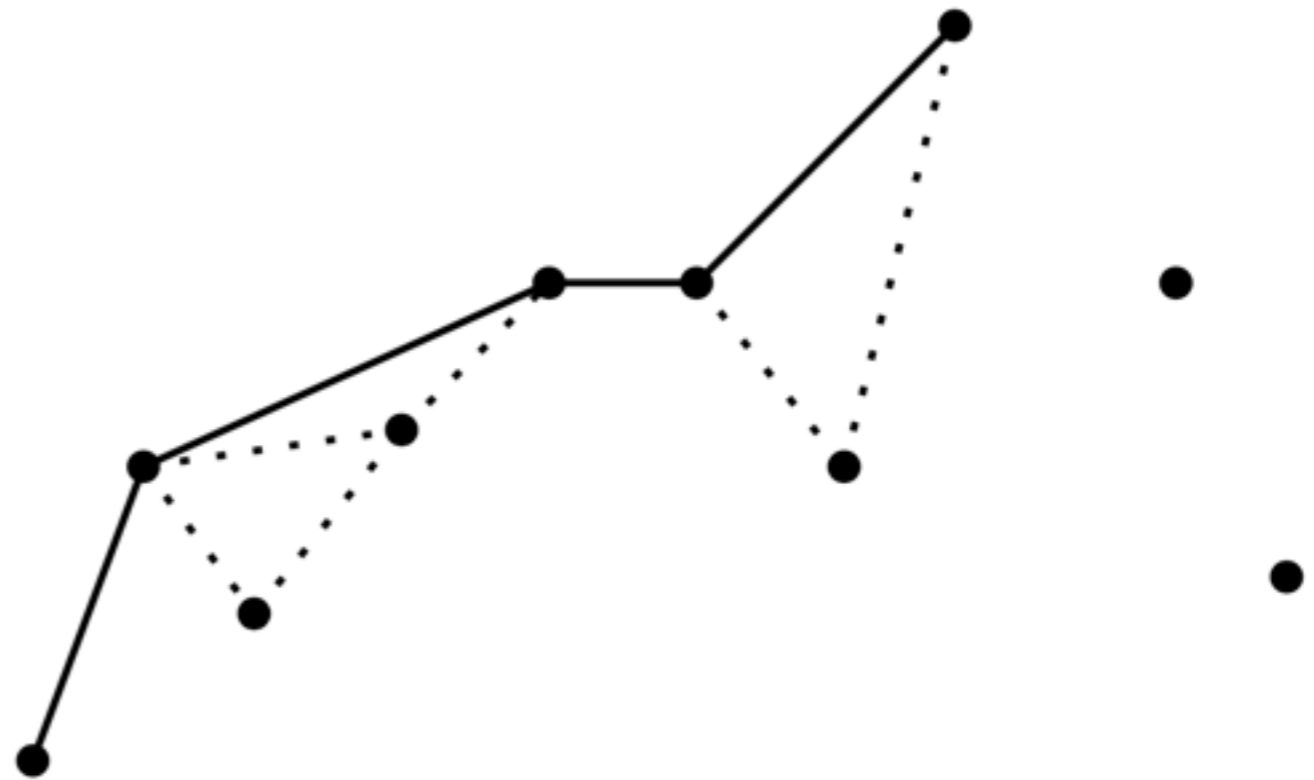
# Algoritmo iterativo

- ▶ Cuando agregamos el octavo punto ...
  - ▶ tenemos que quitar el séptimo



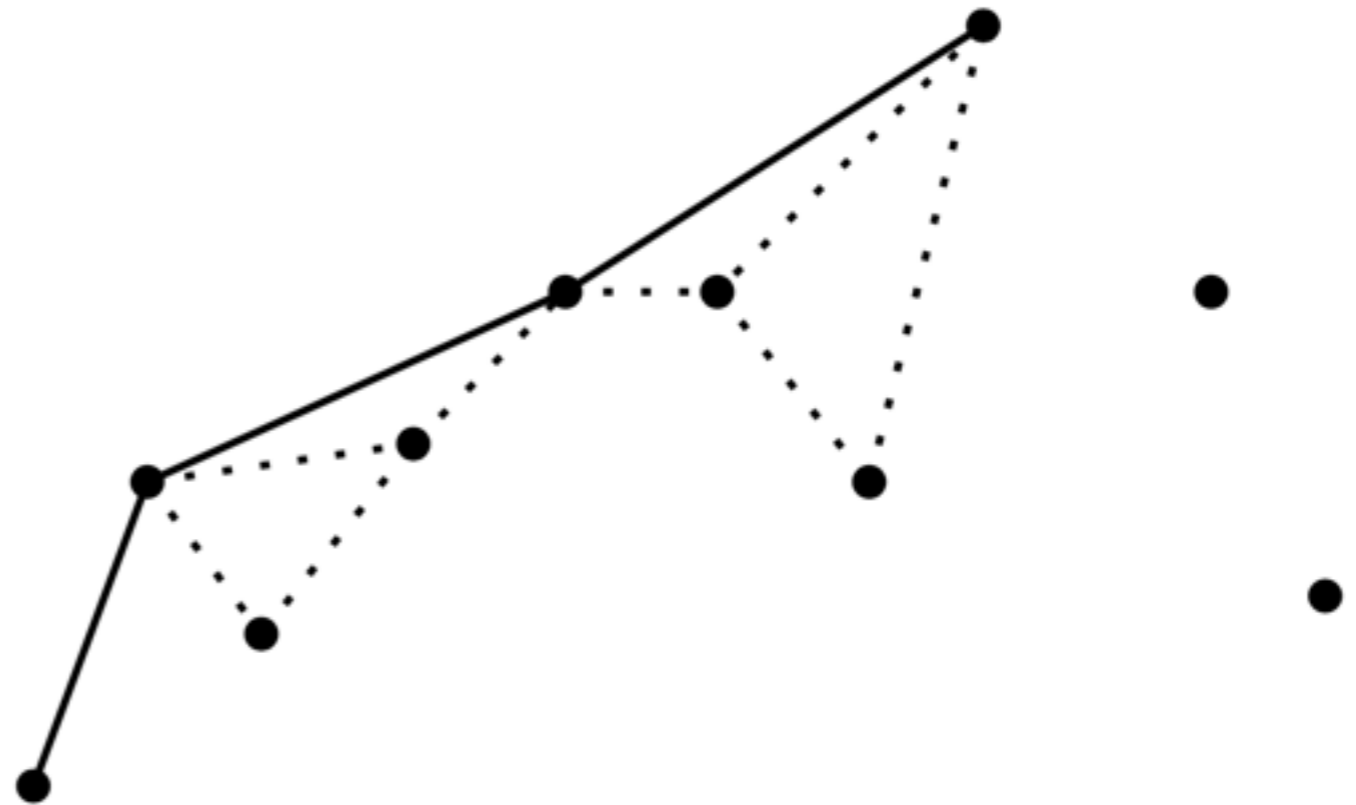
# Algoritmo iterativo

► y el sexto ...



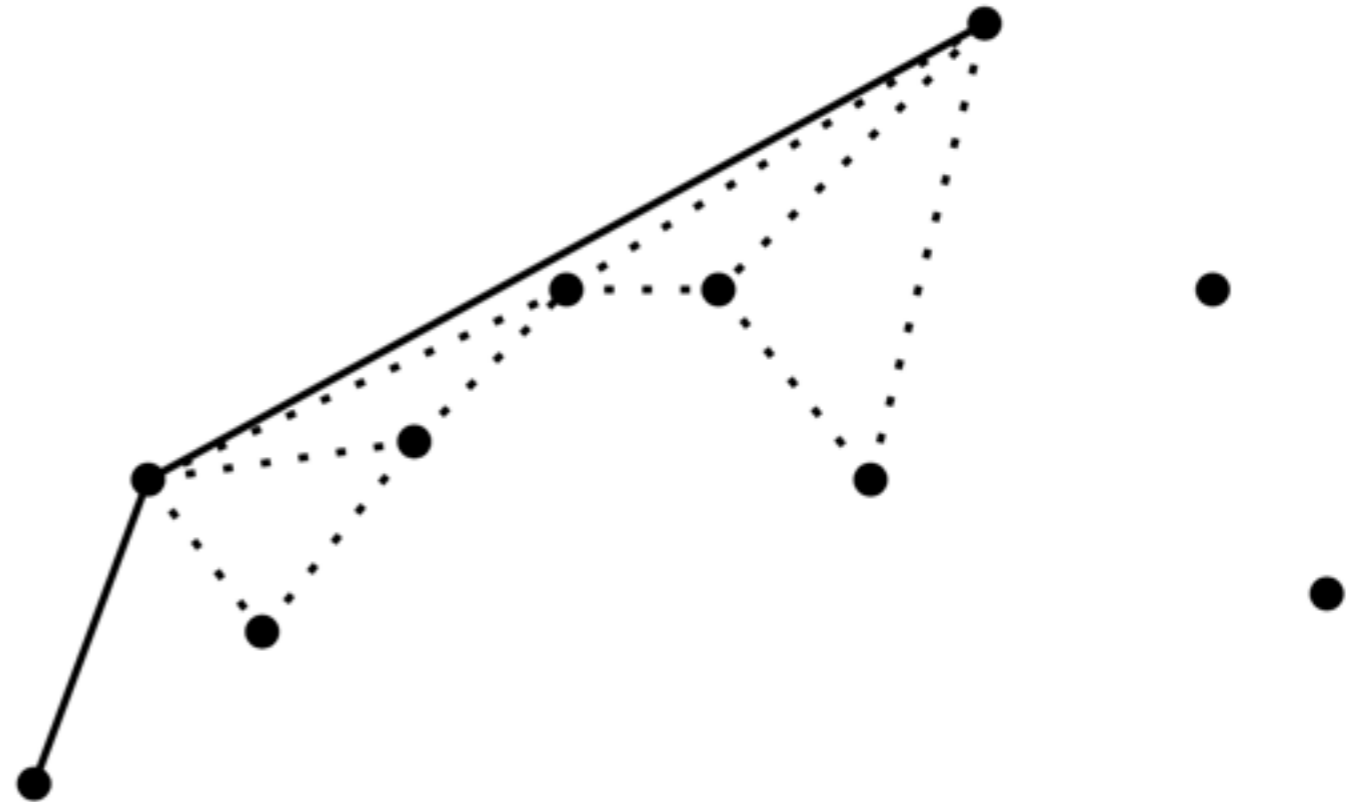
# Algoritmo iterativo

► y el sexto ...



# Algoritmo iterativo

► y el quinto ...



# Algoritmo

## **Algorithm** CONVEXHULL( $P$ )

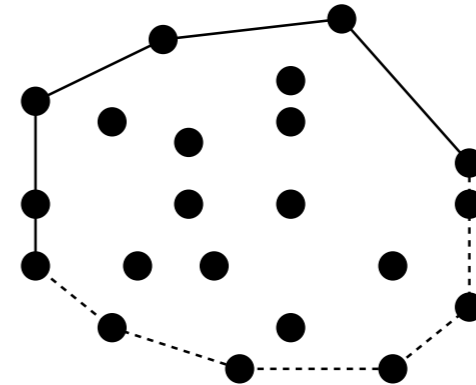
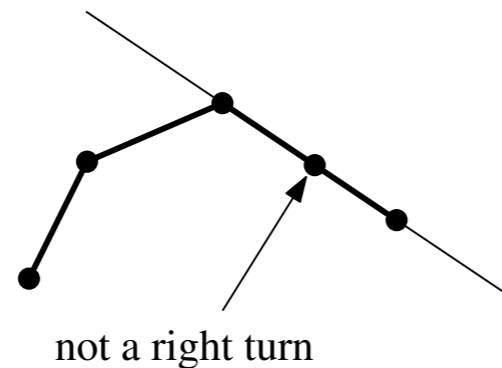
*Input.* A set  $P$  of points in the plane.

*Output.* A list containing the vertices of  $\mathcal{CH}(P)$  in clockwise order.

1. Sort the points by  $x$ -coordinate, resulting in a sequence  $p_1, \dots, p_n$ .
2. Put the points  $p_1$  and  $p_2$  in a list  $\mathcal{L}_{\text{upper}}$ , with  $p_1$  as the first point.
3. **for**  $i \leftarrow 3$  **to**  $n$
4.     **do** Append  $p_i$  to  $\mathcal{L}_{\text{upper}}$ .
5.     **while**  $\mathcal{L}_{\text{upper}}$  contains more than two points **and** the last three points in  $\mathcal{L}_{\text{upper}}$  do not make a right turn
6.     **do** Delete the middle of the last three points from  $\mathcal{L}_{\text{upper}}$ .
7. Put the points  $p_n$  and  $p_{n-1}$  in a list  $\mathcal{L}_{\text{lower}}$ , with  $p_n$  as the first point.
8. **for**  $i \leftarrow n - 2$  **downto** 1
9.     **do** Append  $p_i$  to  $\mathcal{L}_{\text{lower}}$ .
10.     **while**  $\mathcal{L}_{\text{lower}}$  contains more than 2 points **and** the last three points in  $\mathcal{L}_{\text{lower}}$  do not make a right turn
11.     **do** Delete the middle of the last three points from  $\mathcal{L}_{\text{lower}}$ .
12. Remove the first and the last point from  $\mathcal{L}_{\text{lower}}$  to avoid duplication of the points where the upper and lower hull meet.
13. Append  $\mathcal{L}_{\text{lower}}$  to  $\mathcal{L}_{\text{upper}}$ , and call the resulting list  $\mathcal{L}$ .
14. **return**  $\mathcal{L}$



# Algoritmo incremental: casos degenerados



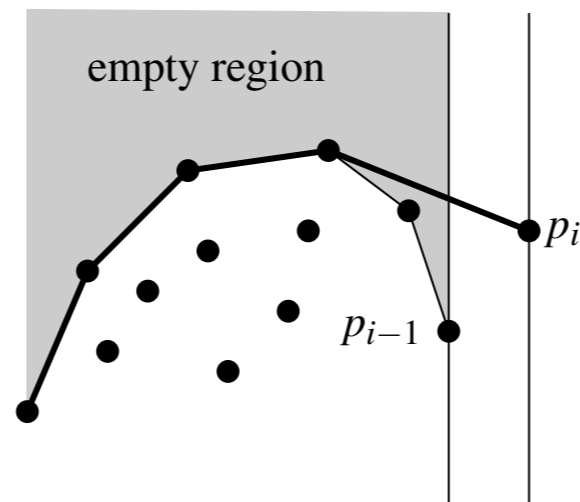
- Un punto tiene la *misma coordenada  $x$*  que otro punto. En este caso podemos ordenarles primero respecto a  $x$  y luego respecto a  $y$ .
- Los tres últimos puntos pueden ser co-lineares. En este caso, el punto intermedio no debería ocurrir en el envolvente convexo. Estos casos se tratan como si fueran vuelta a la izquierda.
- Errores de redondeo o de punto flotante resultarán en un polígono convexo pero no necesariamente el correcto.

# Envolvente convexo (convex hull)

- El envolvente convexo de un conjunto de puntos  $n$  en el plano se puede calcular en tiempo  $O(n \lg n)$ .
- Prueba por inducción: antes del ciclo **for**, la lista  $\mathcal{L}_{\text{upper}}$  contiene a los puntos  $p_1$  y  $p_2$ , que trivialmente forma el envolvente superior de  $\{p_1, p_2\}$ .
- Supongamos que  $\mathcal{L}_{\text{upper}}$  contiene los vértices del envolvente superior de  $\{p_1, \dots, p_{i-1}\}$  y vamos a agregar  $p_i$ . Después de ejecutar el ciclo **while**, sabemos que tendremos una cadena con solamente vueltas a la derecha.
- Además sabemos que esta cadena empieza en el punto extremo izquierdo y termina en el punto extremo derecho de  $\{p_1, \dots, p_i\}$ .
- Si probamos que todos los puntos de  $\{p_1, \dots, p_i\}$  que no estén en  $\mathcal{L}_{\text{upper}}$  están debajo de la cadena, entonces  $\mathcal{L}_{\text{upper}}$  es correcta.

# Envolvente convexo (convex hull)

- Por inducción sabemos que no había punto más arriba de la cadena antes que  $p_i$  fuera añadido.



- Ordenamiento:  $O(n \lg n)$
- Ciclo for:  $O(n)$
- Ciclo while:  $O(n)$

# Algoritmo incremental: casos degenerados

- **Primera etapa:** Entender la geometría del problema, ignorar los casos degenerados.
- **Segunda etapa:** Ajustar el algoritmo a los casos degenerados. Tratar de generalizar y no resolver caso por caso.
- **Tercera etapa:** Implementación, especial atención a redondeos y puntos flotantes.