

Übung Algorithmische Geometrie

Einführung & Konvexe Hülle

LEHRSTUHL FÜR ALGORITHMIK · INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Andreas Gemsa
26.04.2012



Dozent



- Martin Nöllenburg
- noellenburg@kit.edu
- Raum 319
- Sprechzeiten: Termin per Mail vereinbaren

Übungsleiter



- Andreas Gemsa
- gemsa@kit.edu
- Raum 322
- Sprechzeiten: Termin per Mail vereinbaren

Termine

- Vorlesung: Di 9:45 – 11:15 Uhr, Raum 301
- Übung: Do 10:15 – 11:00 Uhr, Raum 131

Ausgabe Blatt 1

for Woche $i = 2 \dots 14$ **do**

if Tag == Dienstag **then**

 Ausgabe Blatt i

 Abgabe Blatt $(i - 1)$

if Tag == Donnerstag & !isFeiertag(Tag) **then**

 Besprechung Blatt $(i - 1)$

- Bearbeitung der Aufgaben und Abgabe der Lösungen in Zweiergruppen erwünscht
- Übung wöchentlich 45 Minuten
- abweichende Regelung an Feiertagen

Übungsblatt 1

Optimaler Algorithmus

Übungsblatt 1 - Aufgabe 1 - a)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

Übungsblatt 1 - Aufgabe 1 - a)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Übungsblatt 1 - Aufgabe 1 - a)

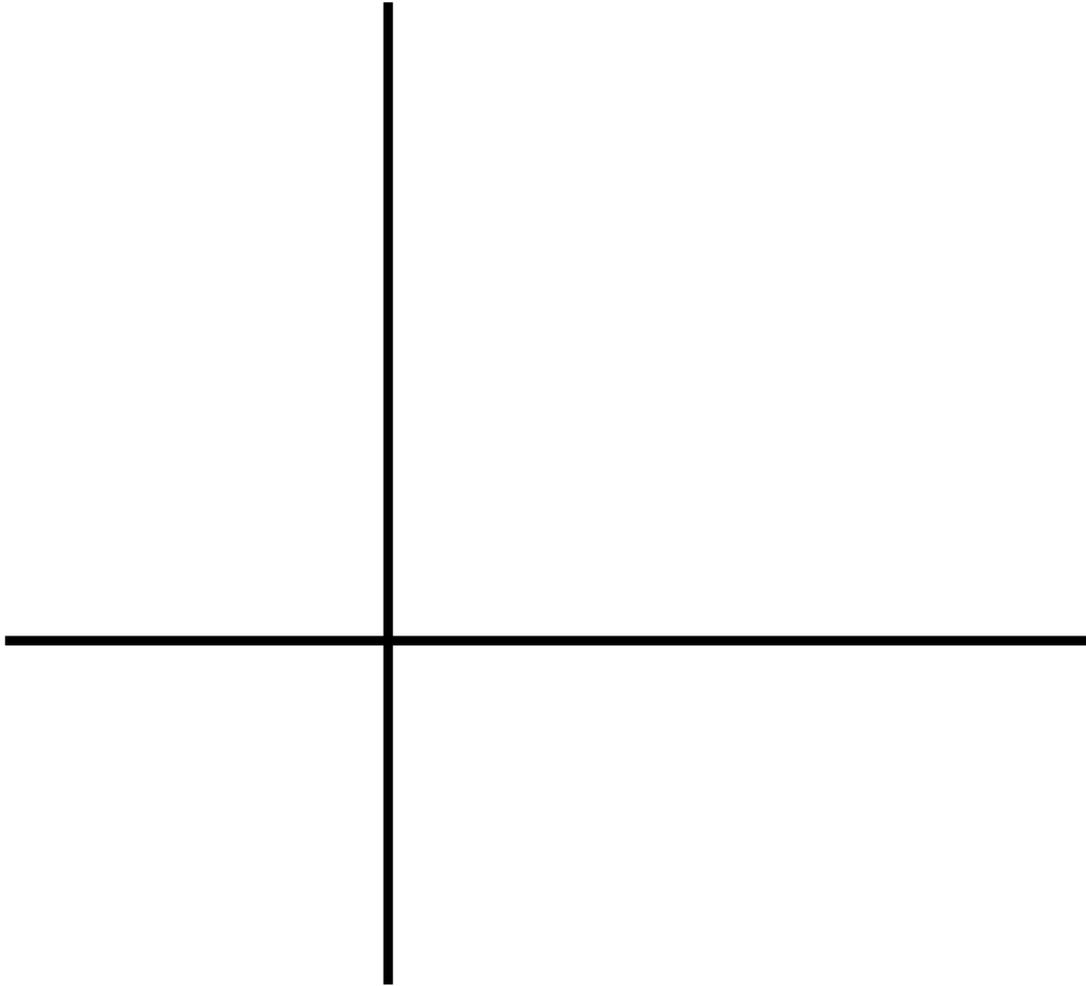
Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

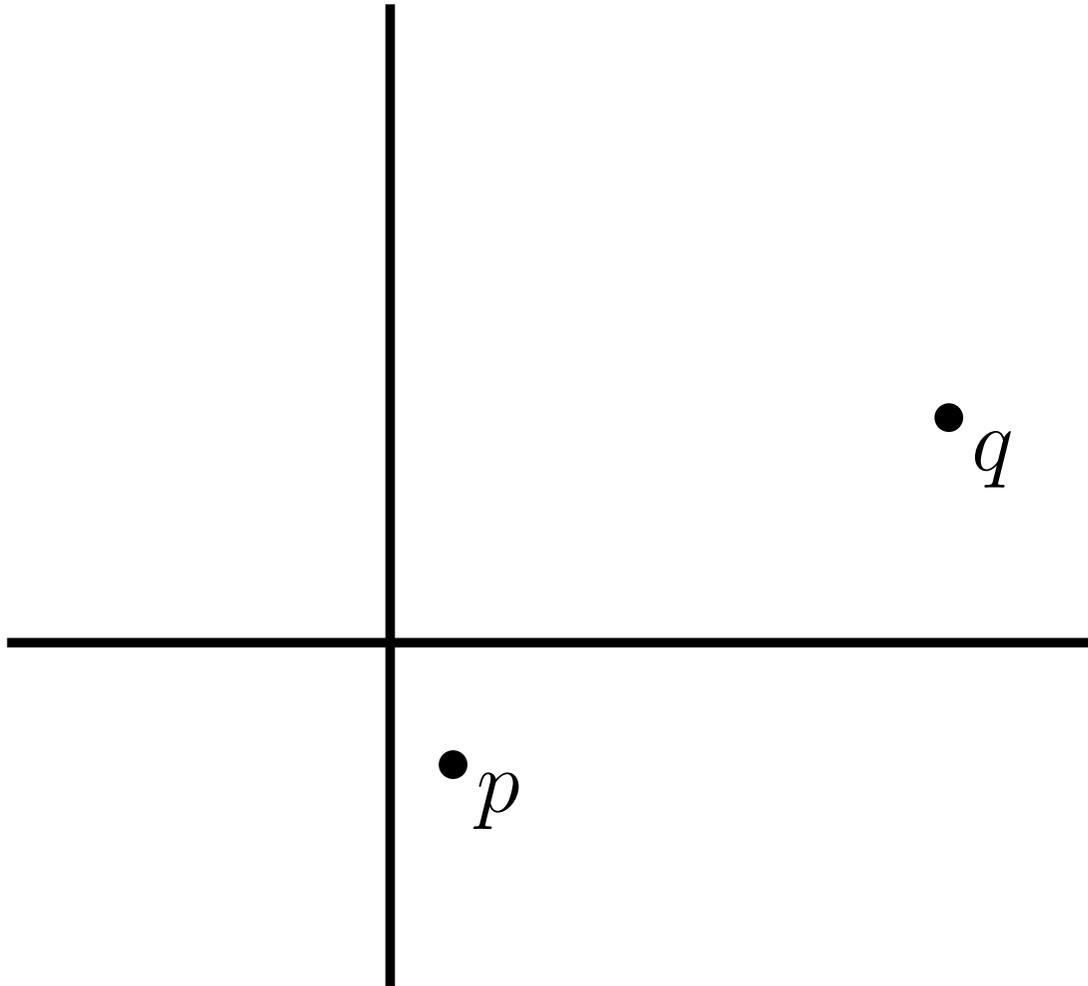
$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

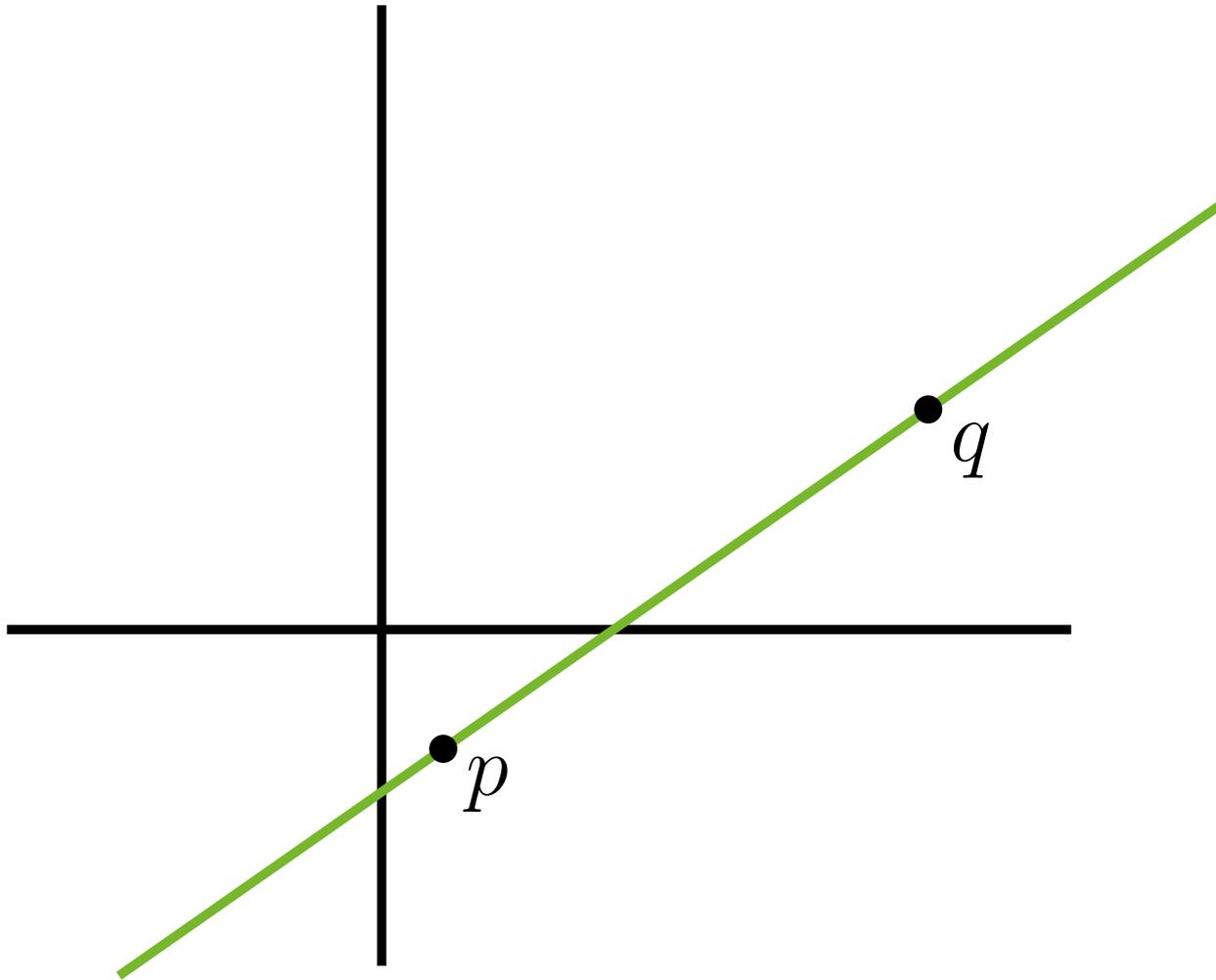
Aufgabe 1 - a)



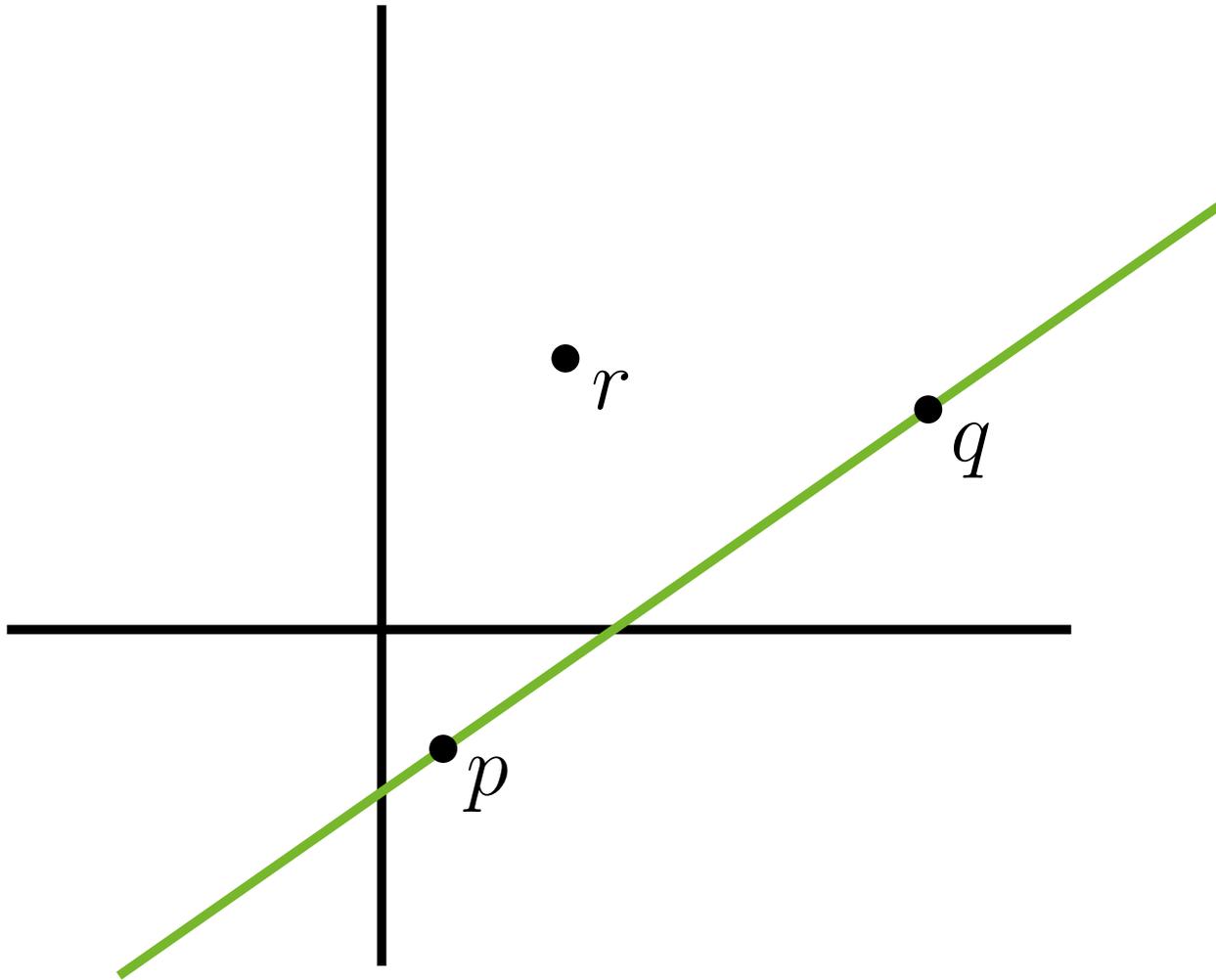
Aufgabe 1 - a)



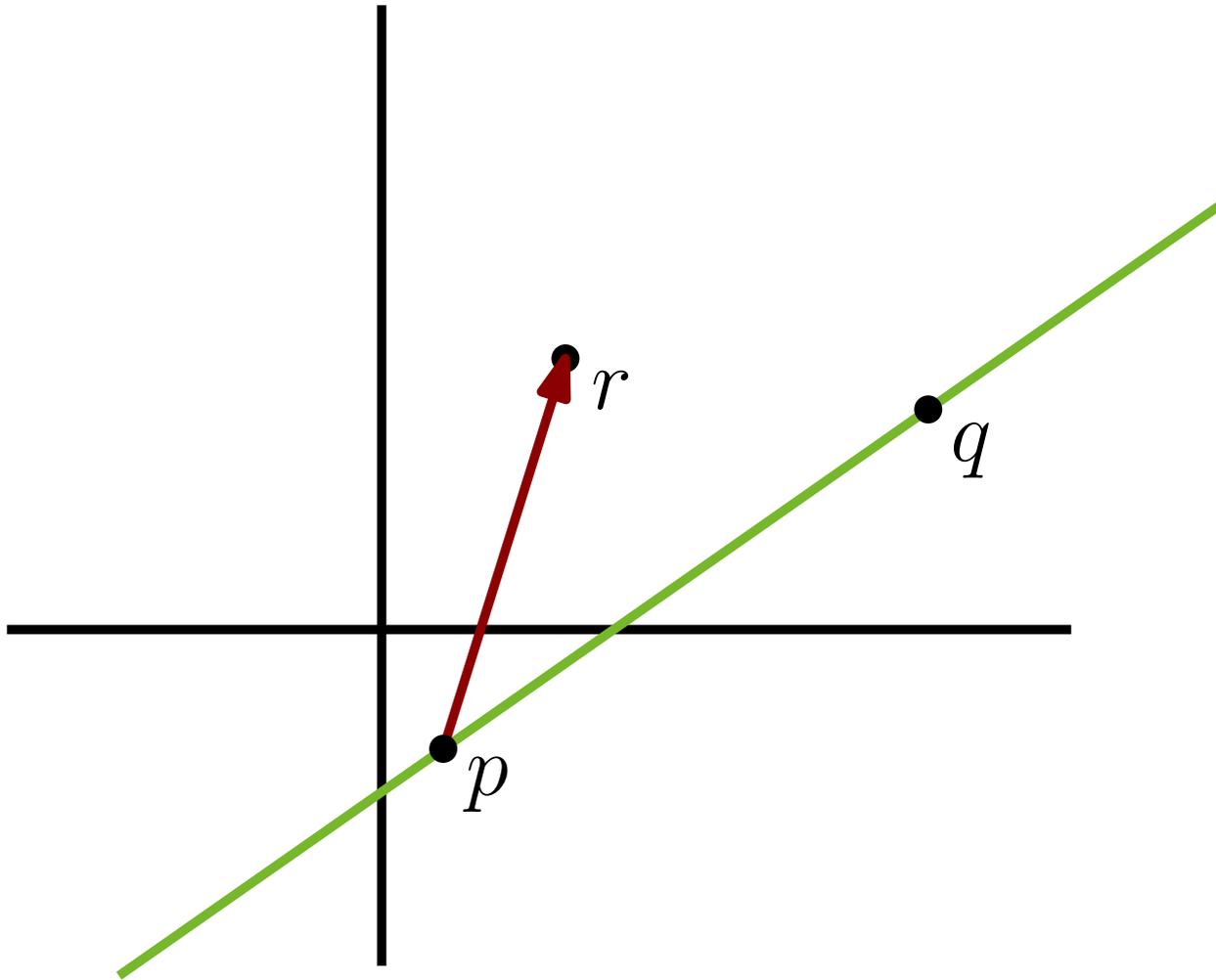
Aufgabe 1 - a)



Aufgabe 1 - a)

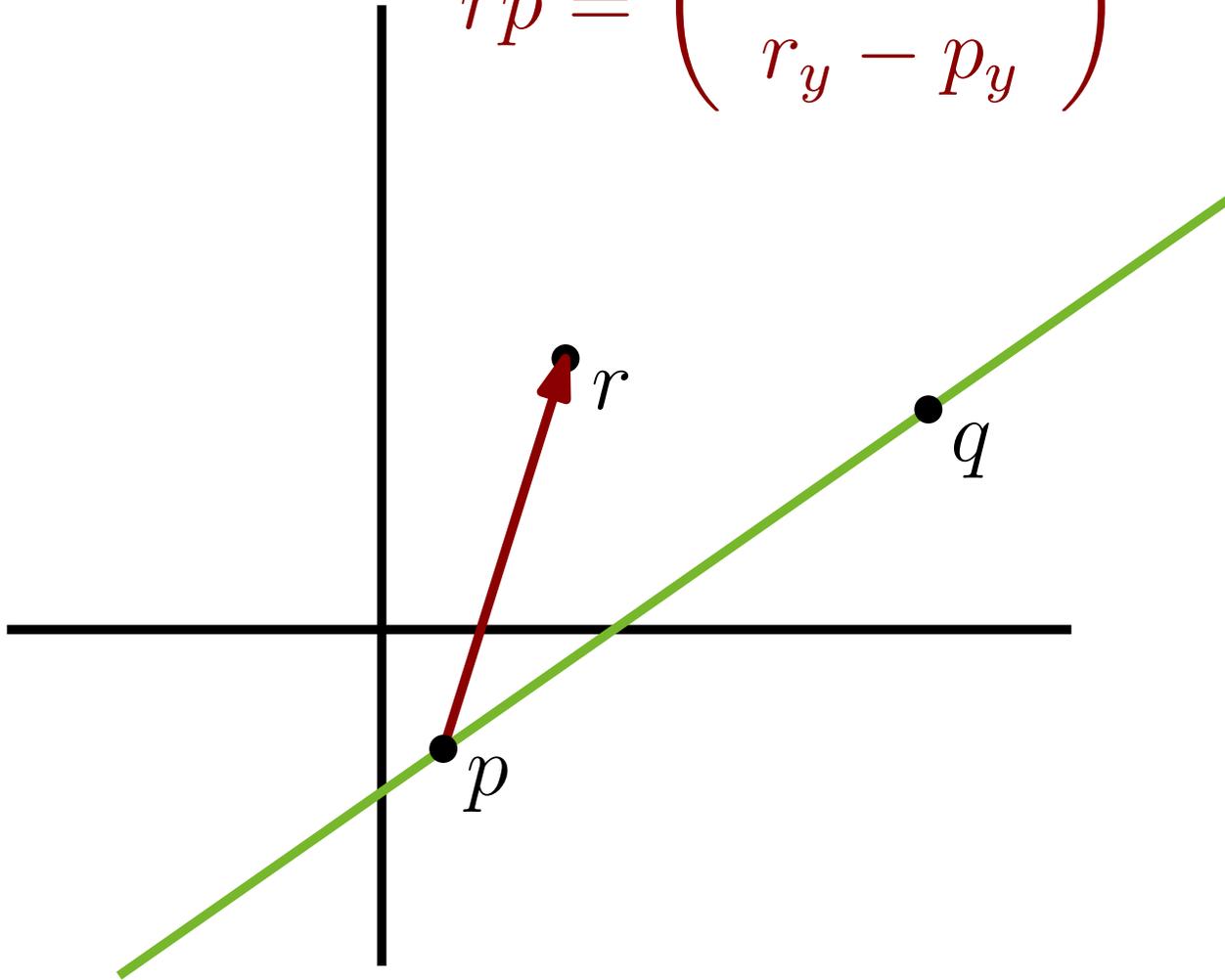


Aufgabe 1 - a)



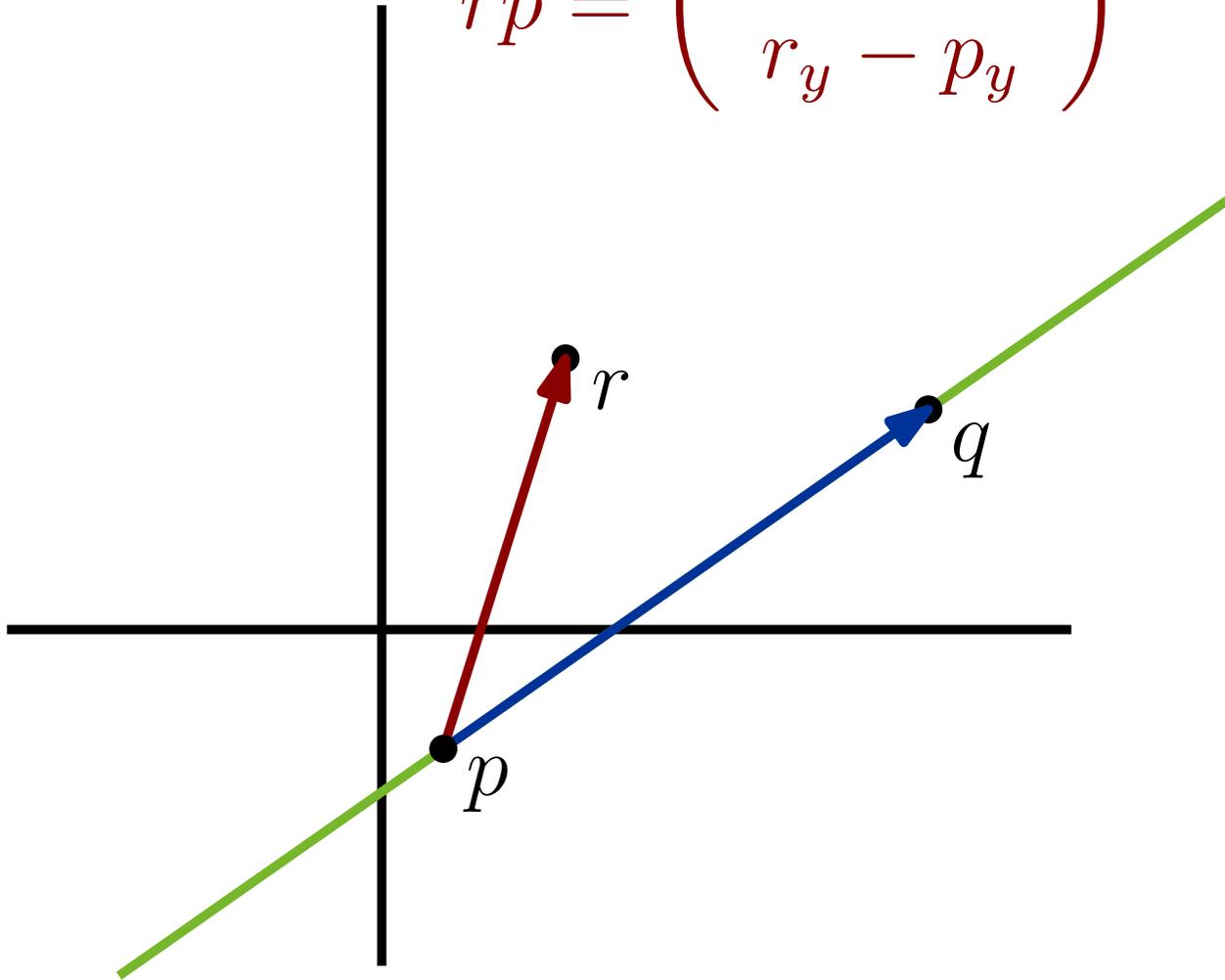
Aufgabe 1 - a)

$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix}$$



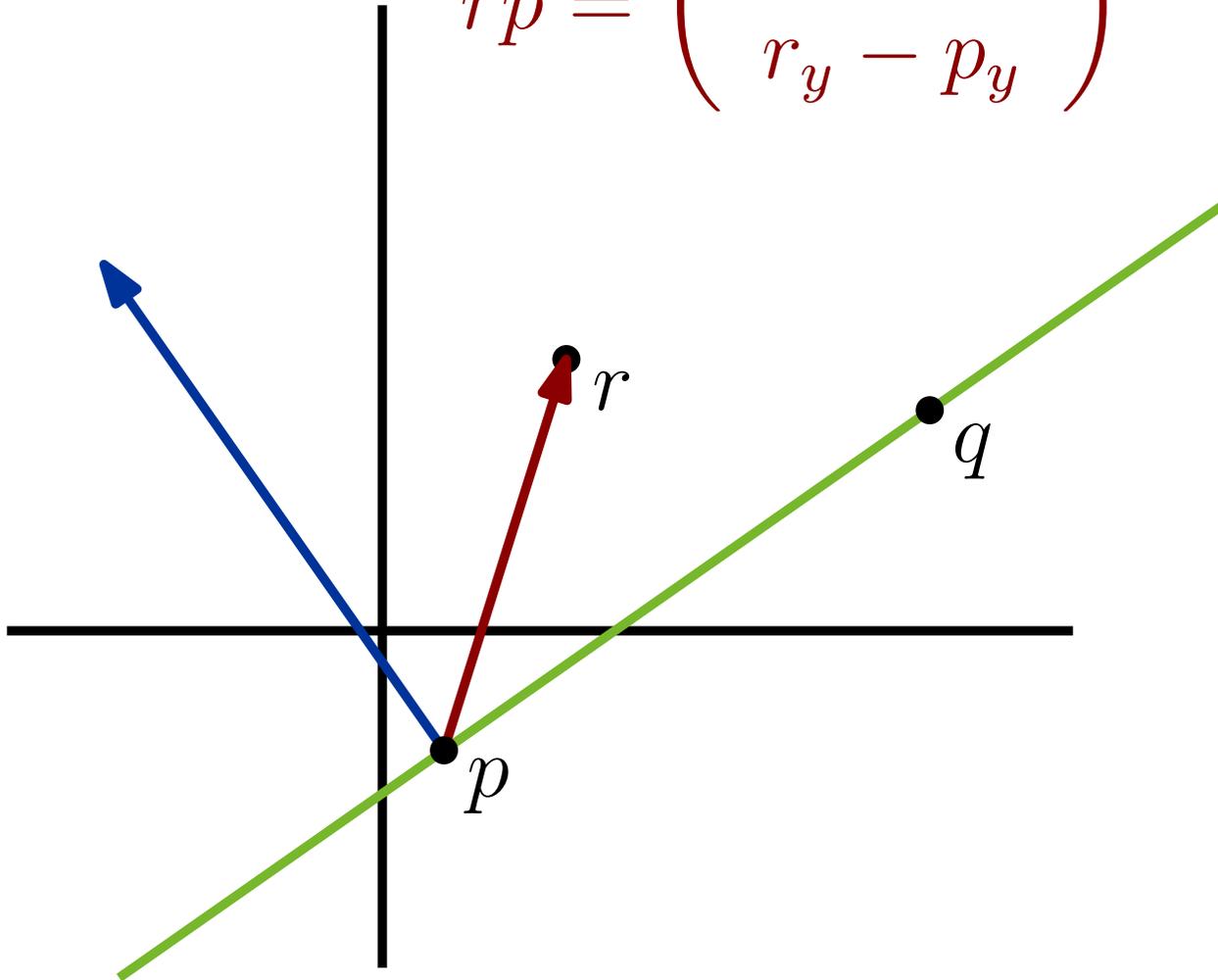
Aufgabe 1 - a)

$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix}$$



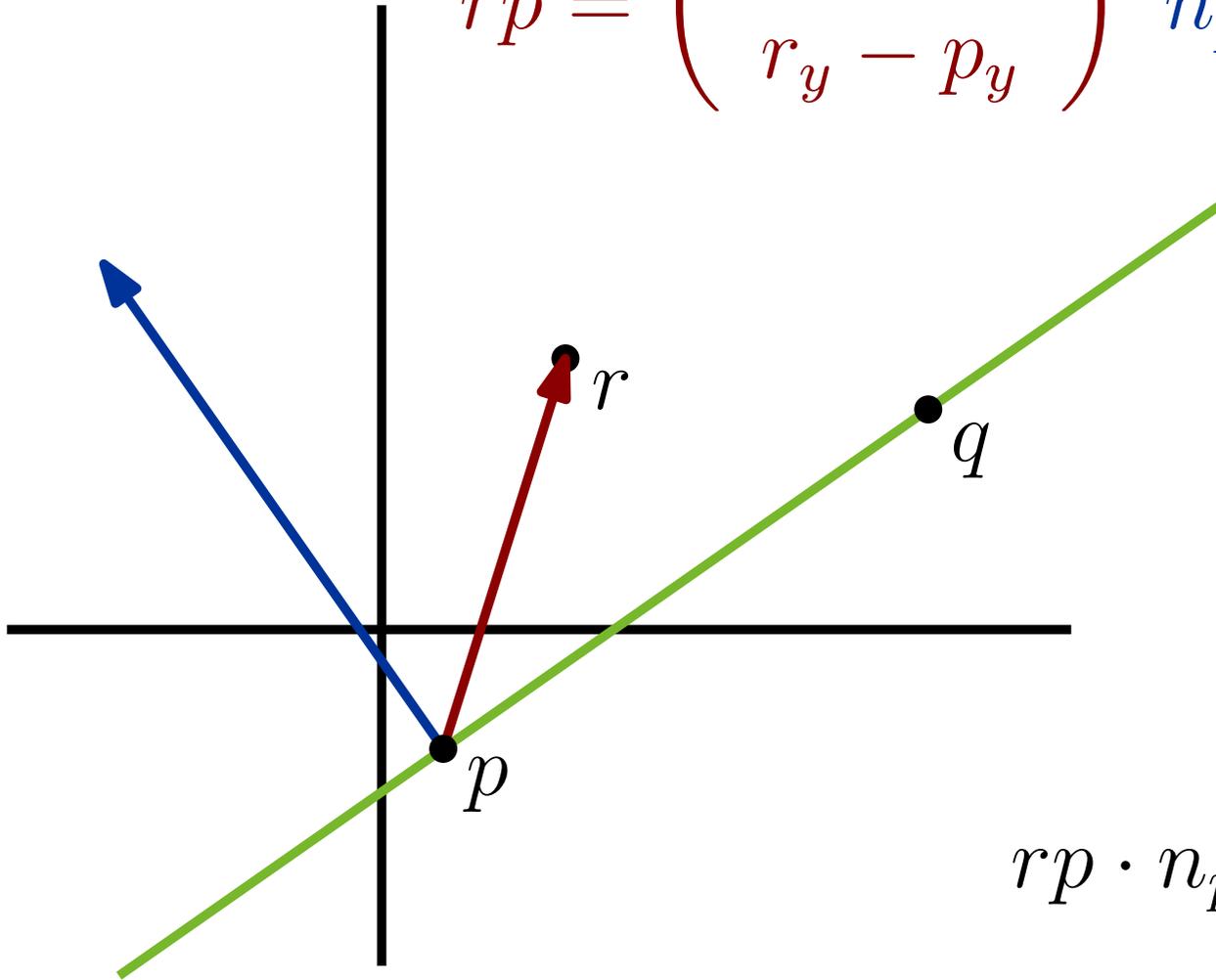
Aufgabe 1 - a)

$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix}$$



Aufgabe 1 - a)

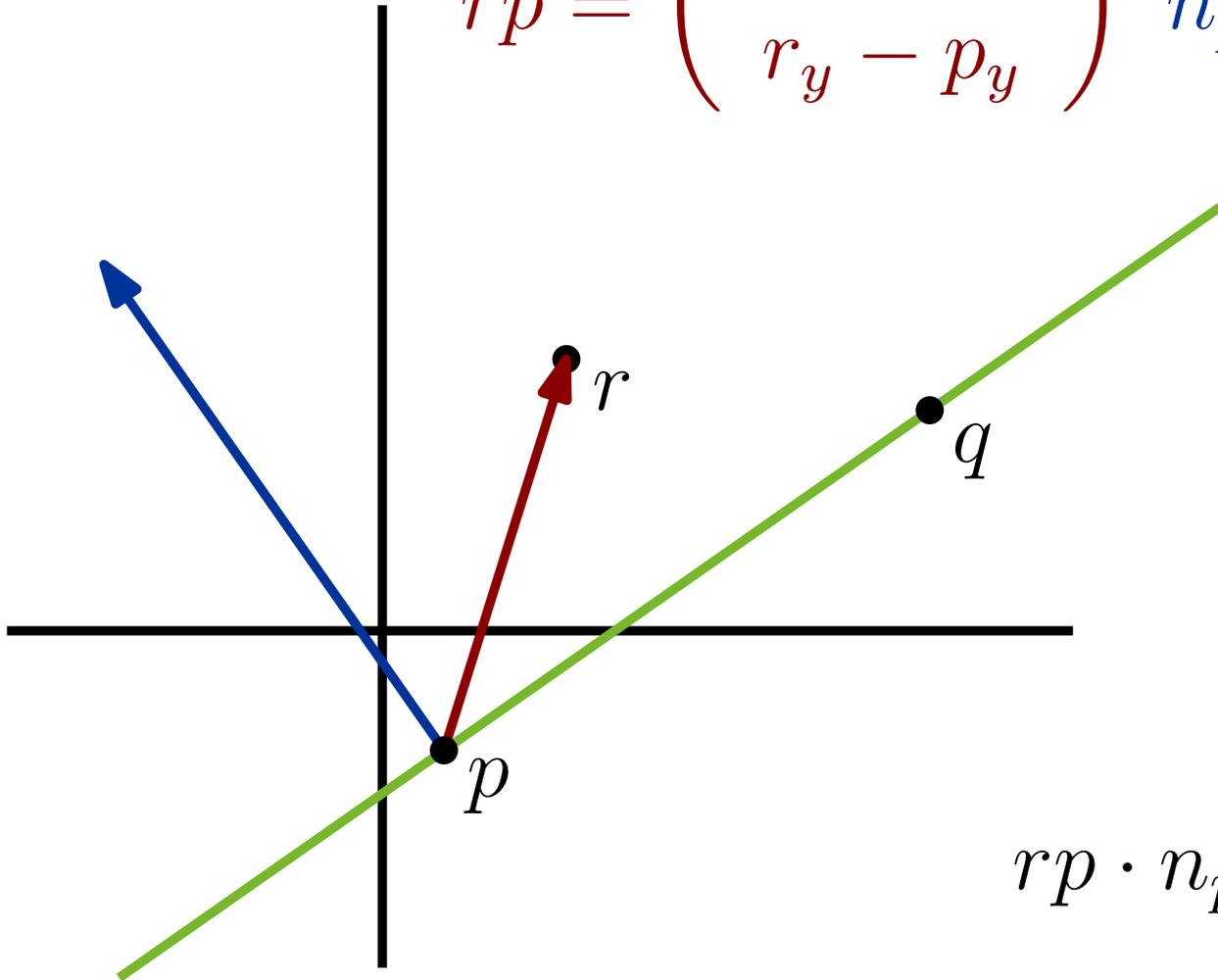
$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix} \quad n_{pq} = \begin{pmatrix} p_y - q_y \\ q_x - p_x \end{pmatrix}$$



$$rp \cdot n_{pq} = \det(A)$$

Aufgabe 1 - a)

$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix} \quad n_{pq} = \begin{pmatrix} p_y - q_y \\ q_x - p_x \end{pmatrix}$$



$$rp \cdot n_{pq} = \det(A)$$

Aufgabe 1 - a)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

Aufgabe 1 - a)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.



Aufgabe 1 - a)

+ b)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

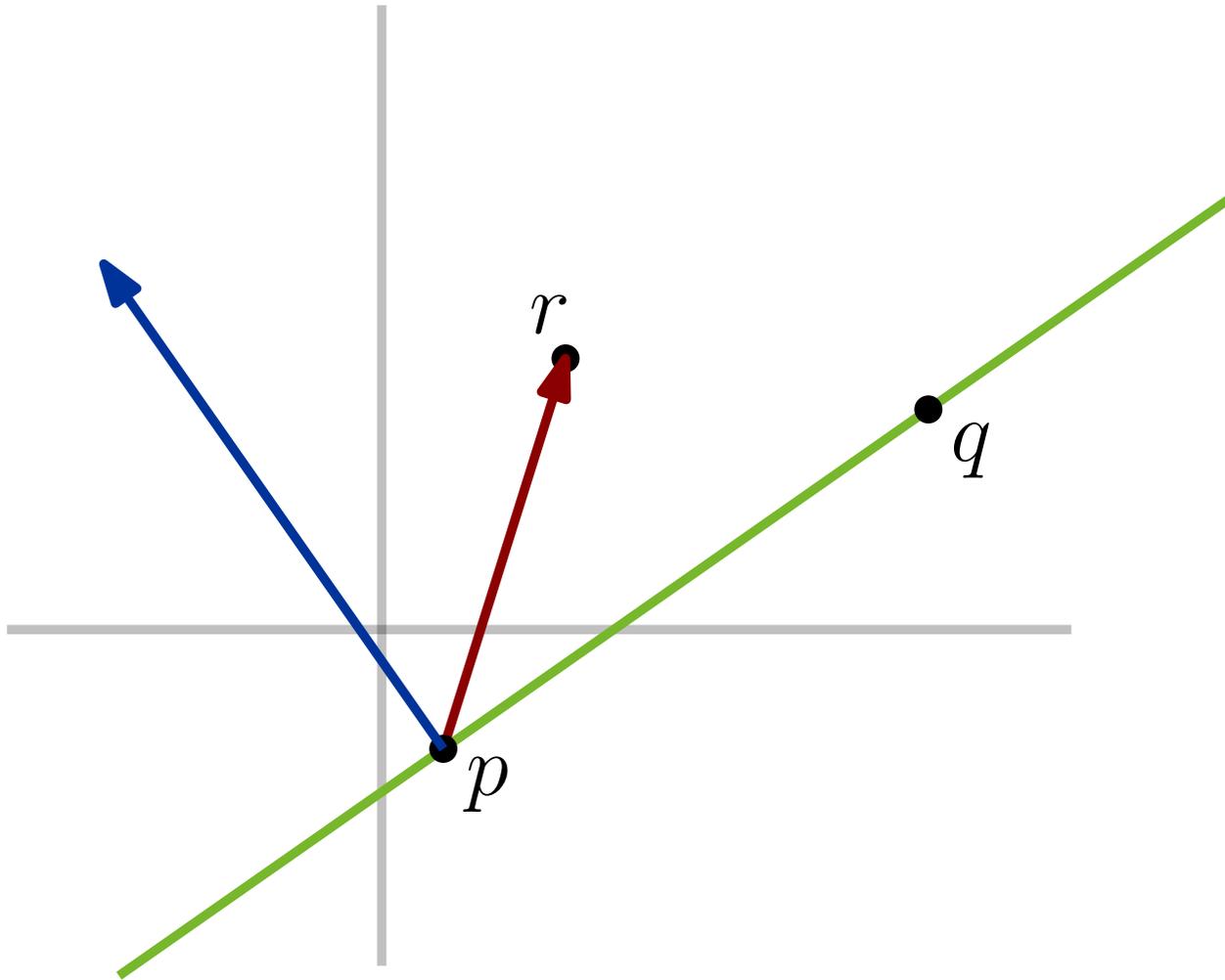
$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

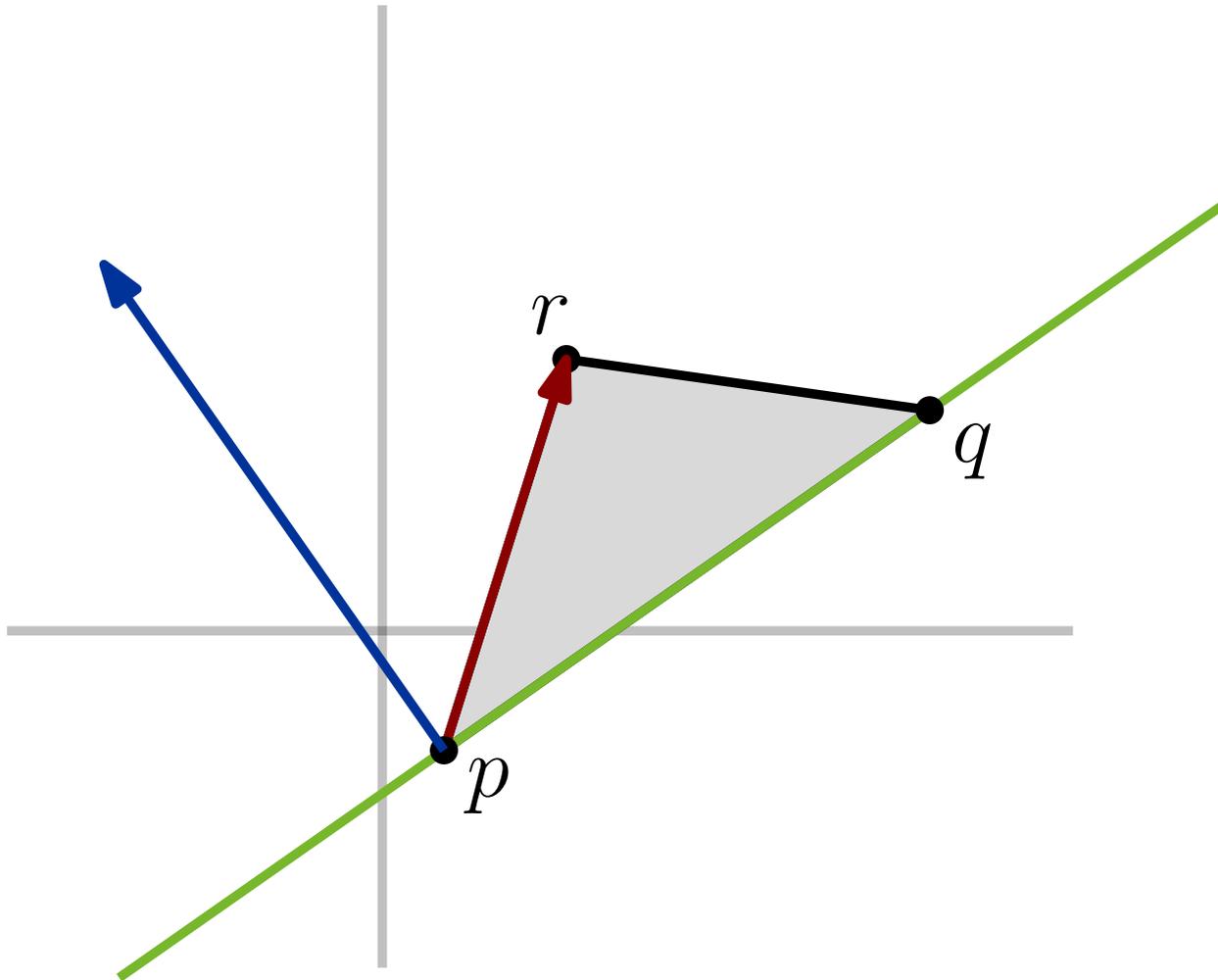


Beweise: $|\det(A)| = \text{doppelter Flächeninhalt des Dreiecks } pqr$

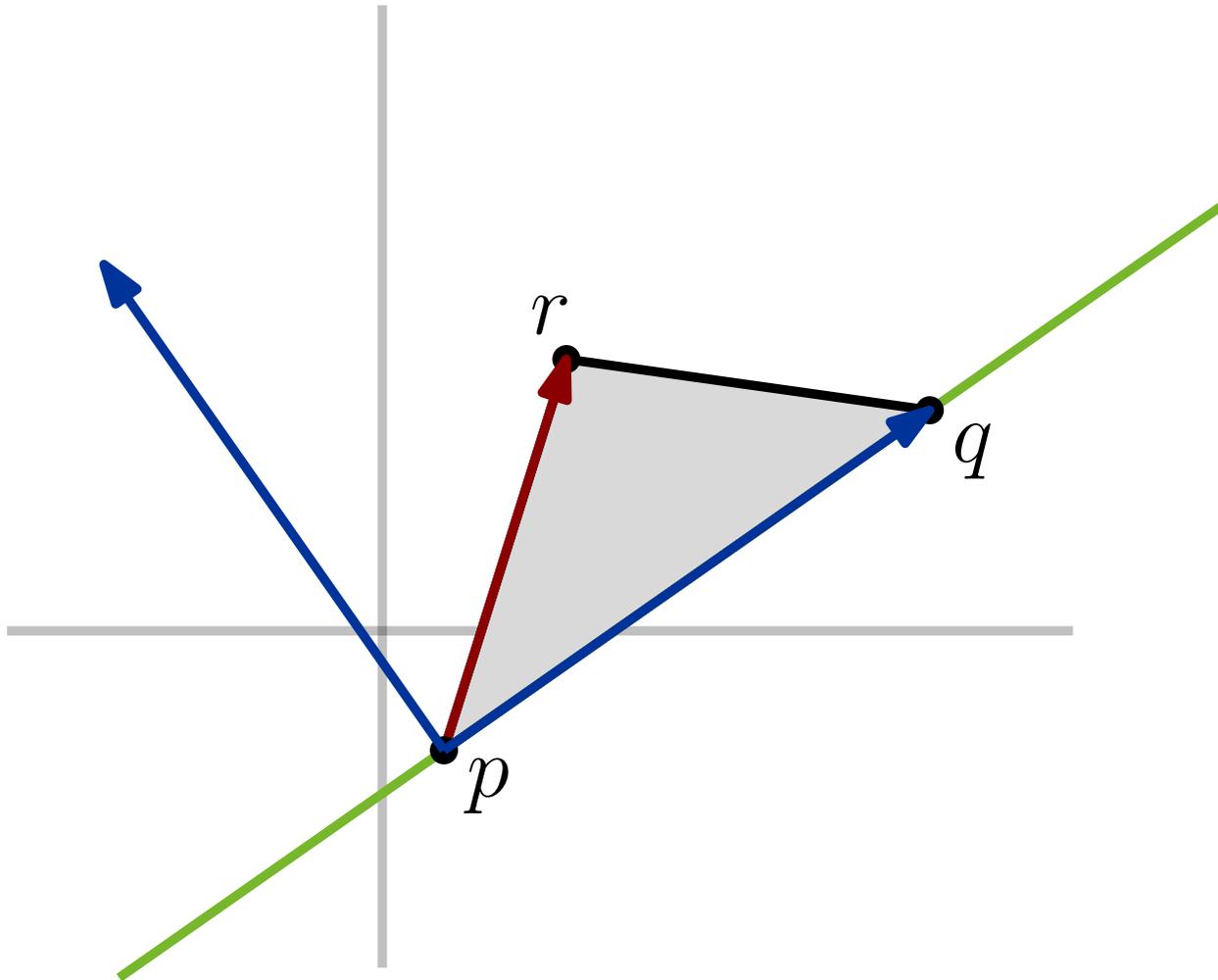
Aufgabe 1 - b)



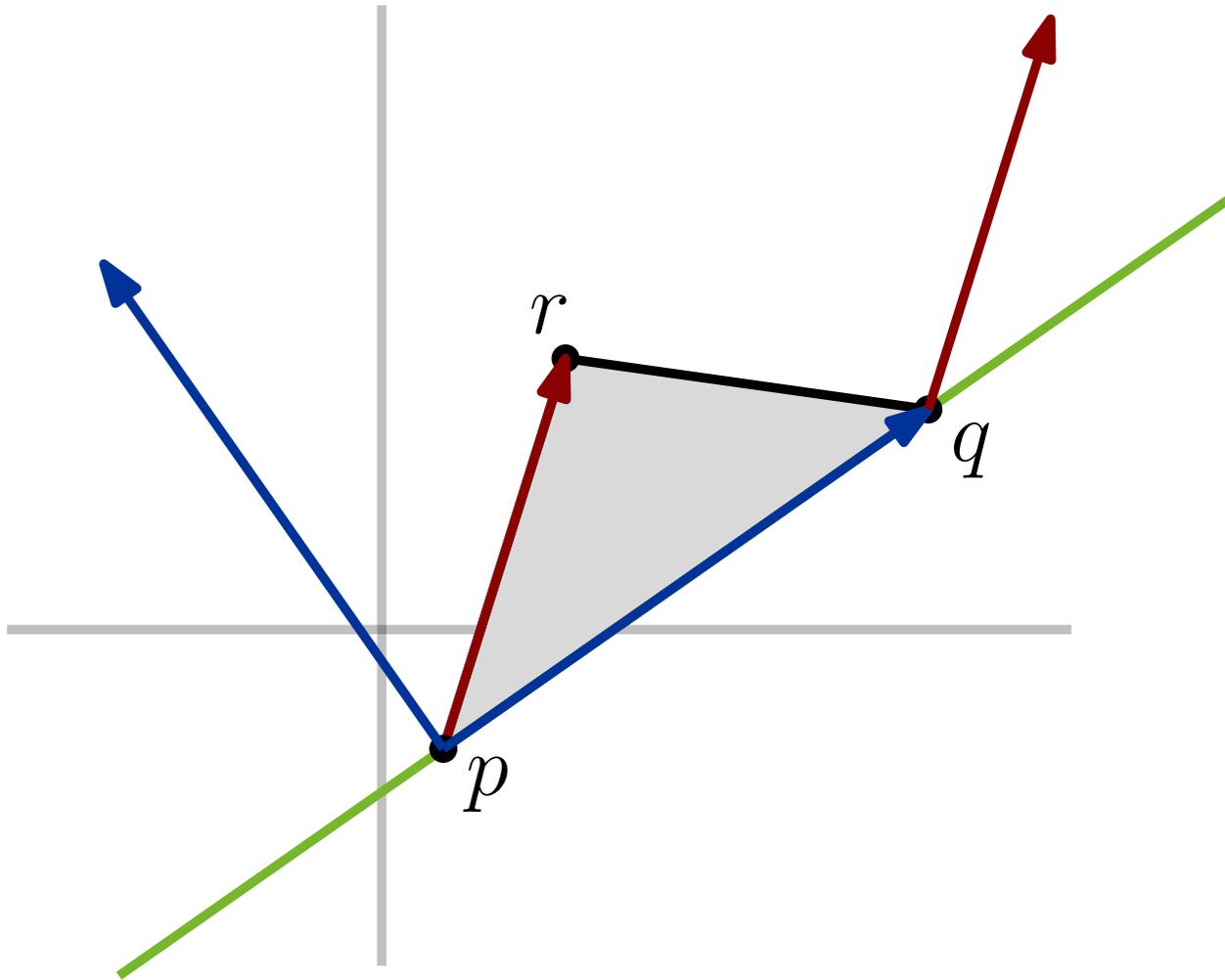
Aufgabe 1 - b)



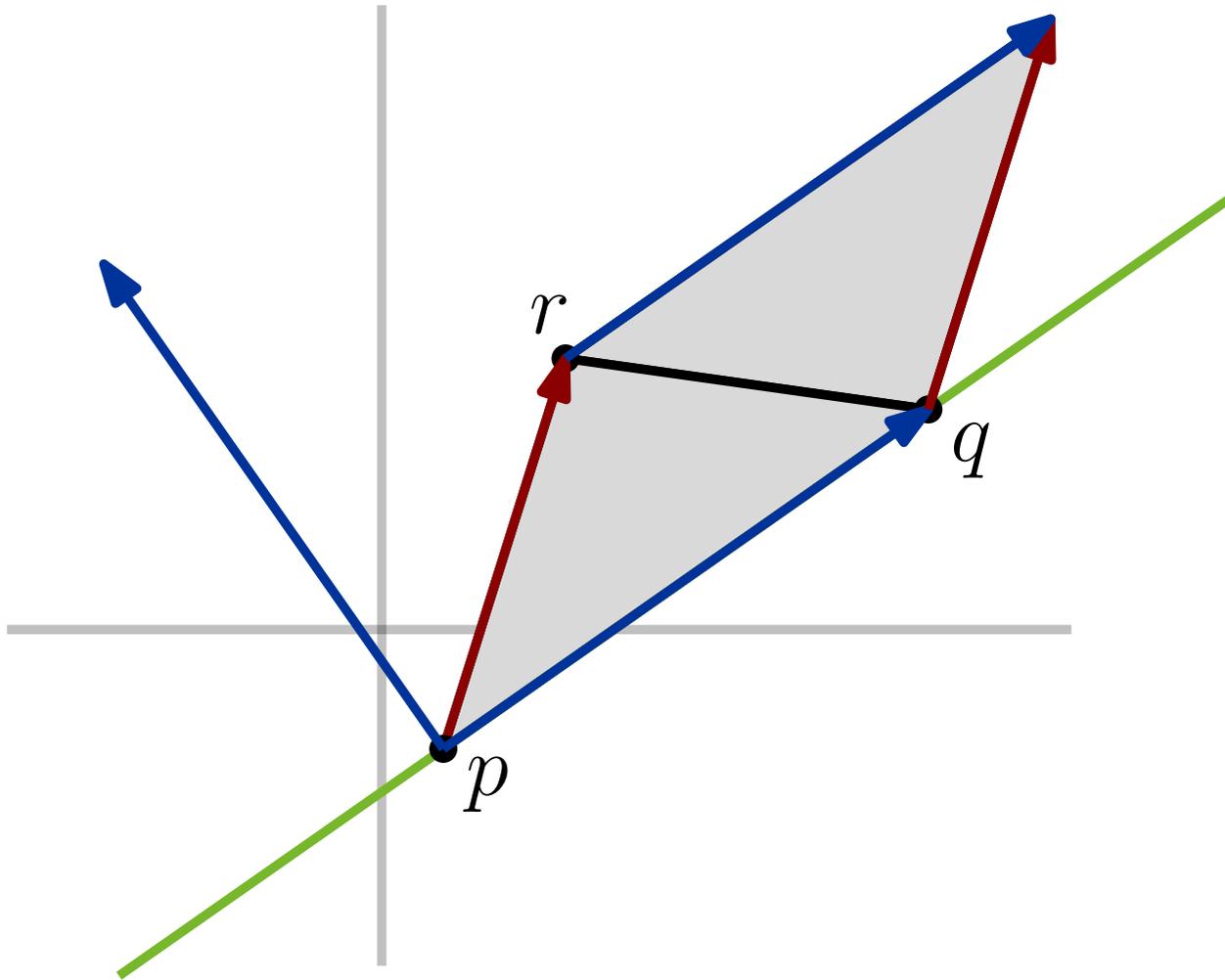
Aufgabe 1 - b)



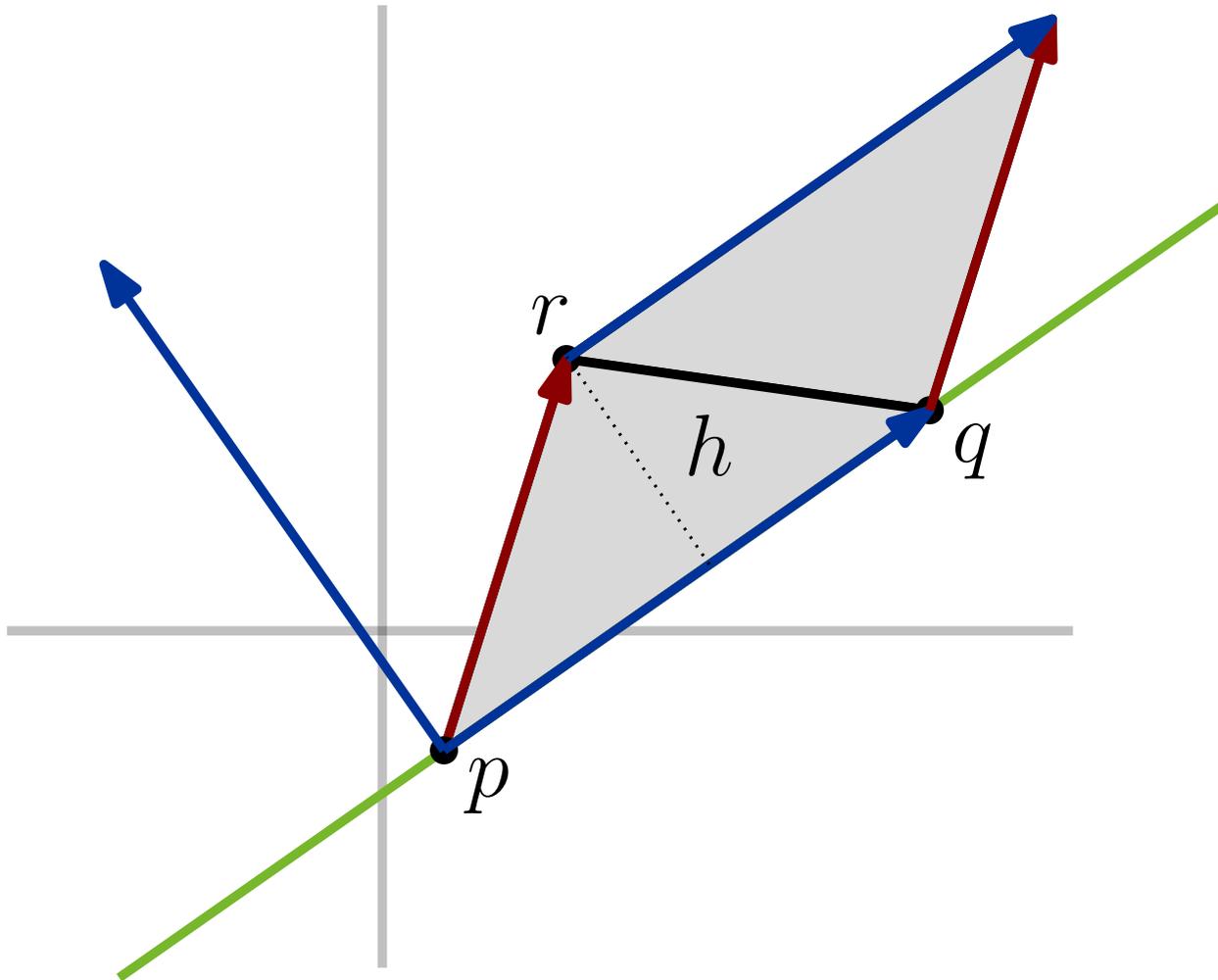
Aufgabe 1 - b)



Aufgabe 1 - b)

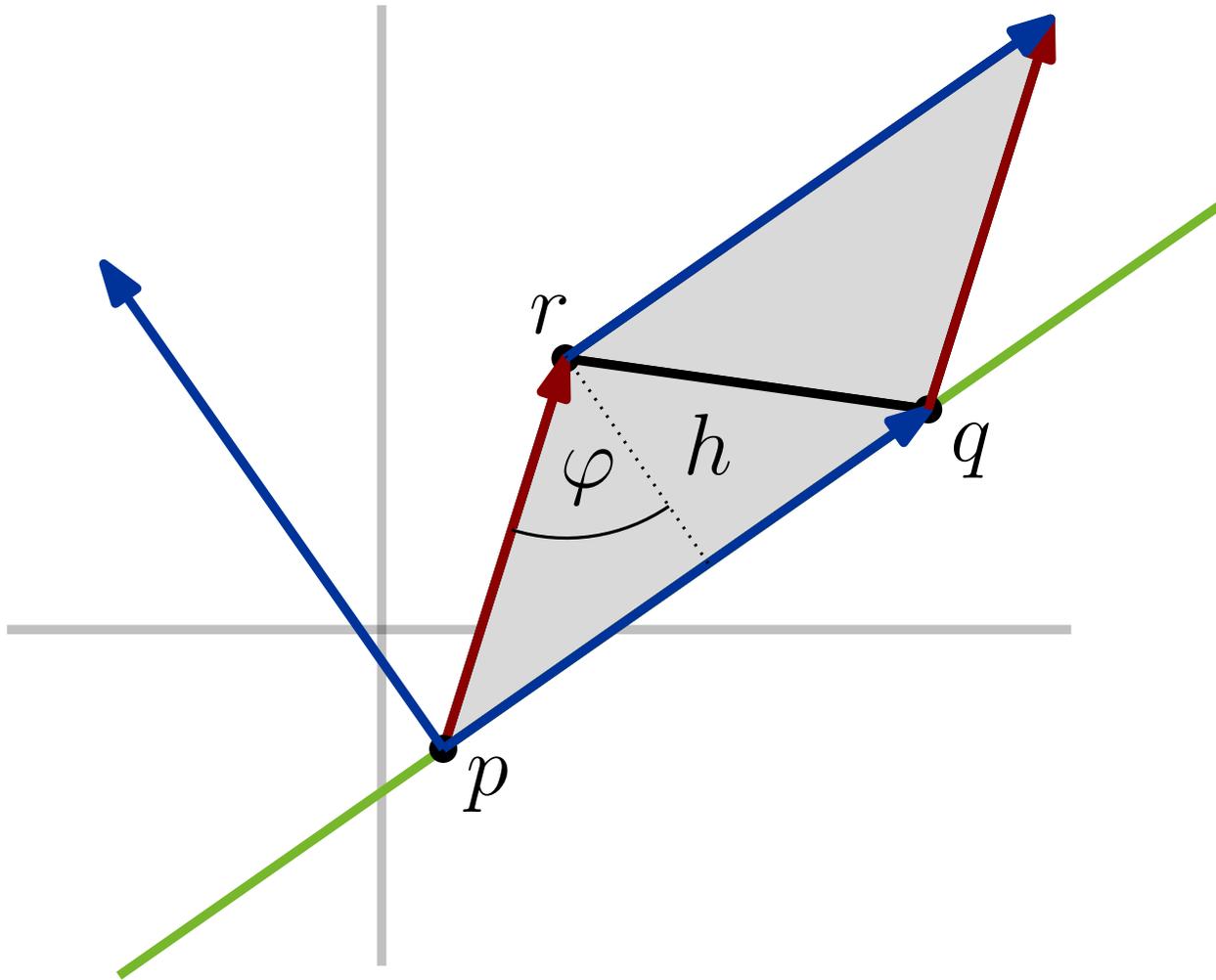


Aufgabe 1 - b)



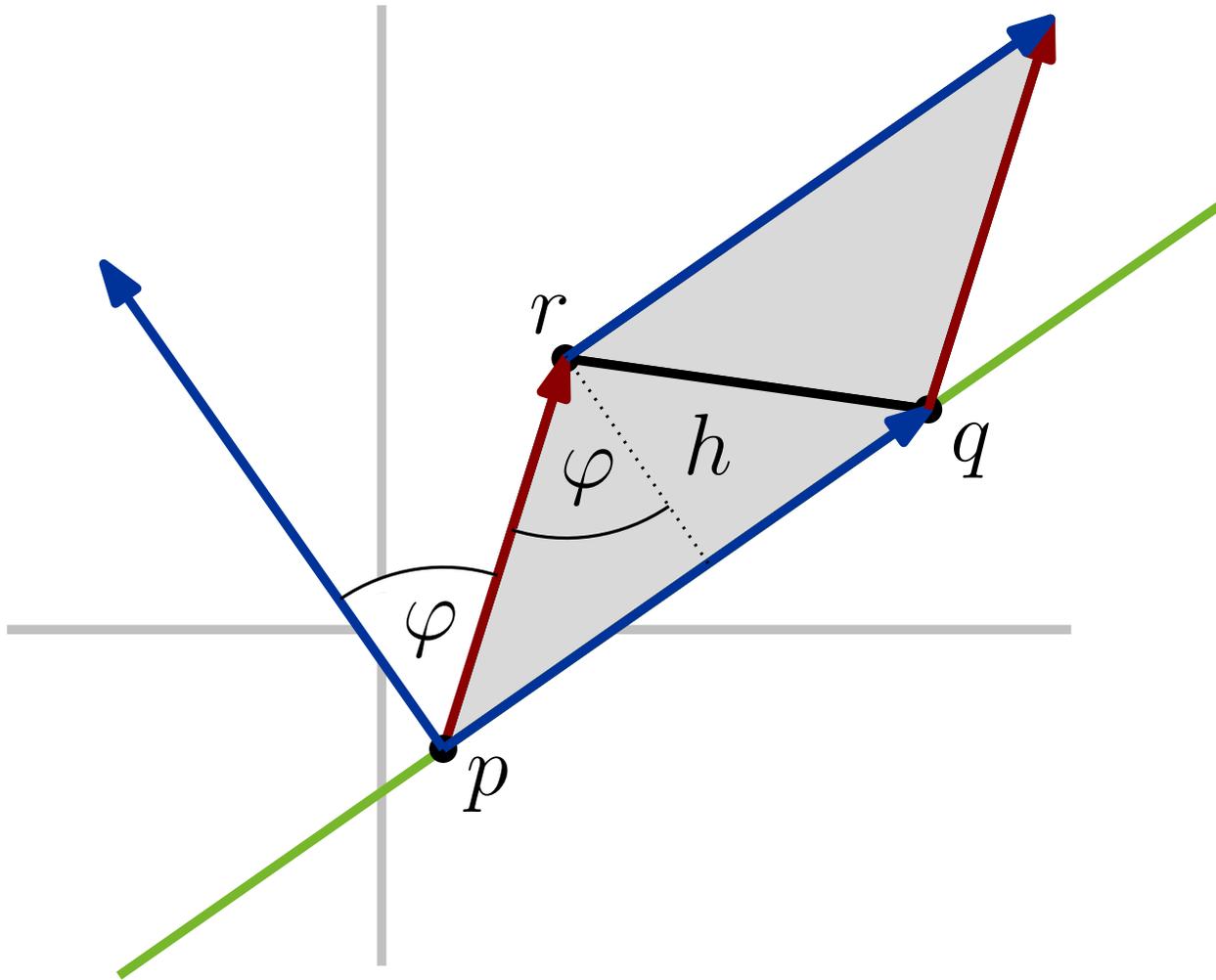
$$A_{pqr} = |pq| \cdot h$$

Aufgabe 1 - b)



$$A_{pqr} = |pq| \cdot h$$

Aufgabe 1 - b)



$$A_{pqr} = |pq| \cdot h$$

Aufgabe 1 - a)

+ b)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.



Beweise: $|\det(A)| = \text{doppelter Flächeninhalt des Dreiecks } pqr$

Aufgabe 1 - a)

+ b)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.



Beweise: $|\det(A)| = \text{doppelter Flächeninhalt des Dreiecks } pqr$



Aufgabe 2 (a + d)

Aufgabe 2 (a + d)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

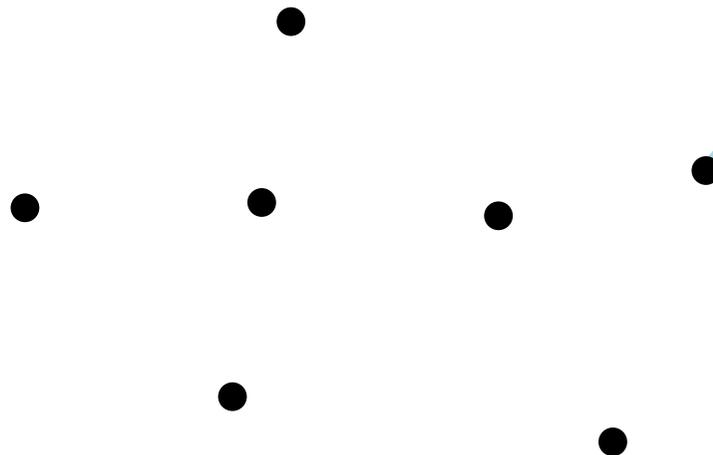
while true do

 Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

if $p_{j+1} == p_1$ **then**

break

return (p_1, \dots, p_h)



Aufgabe 2 (a + d)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

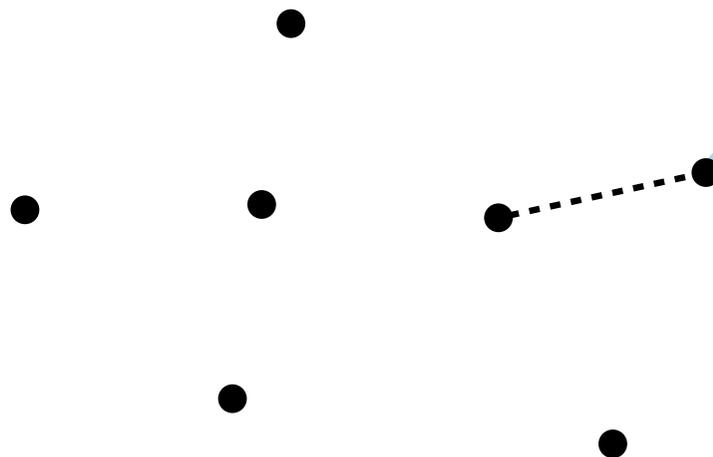
while true do

 Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

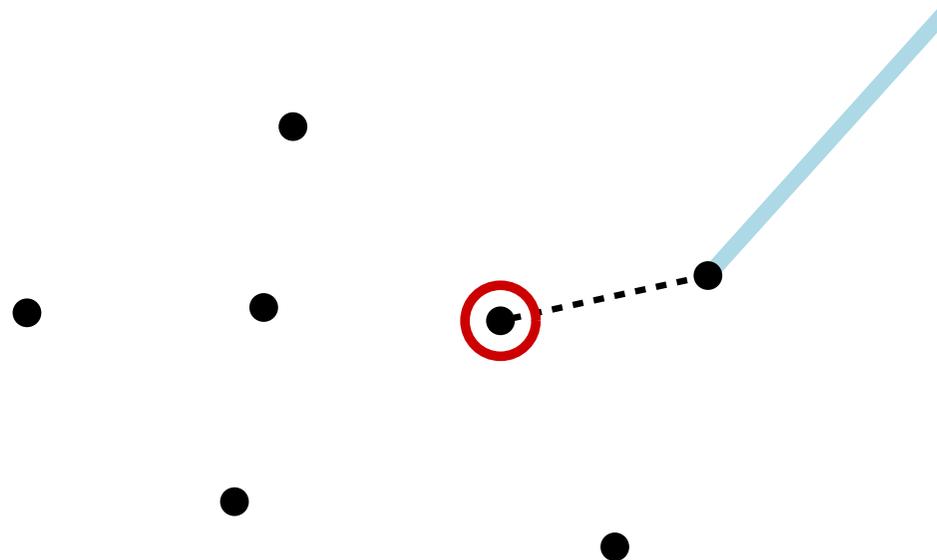
if $p_{j+1} == p_1$ **then**

break

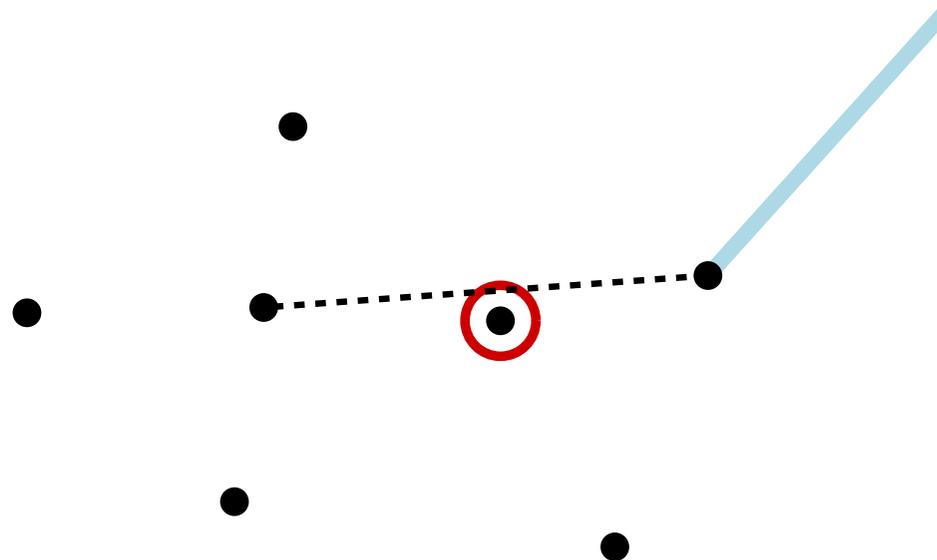
return (p_1, \dots, p_h)



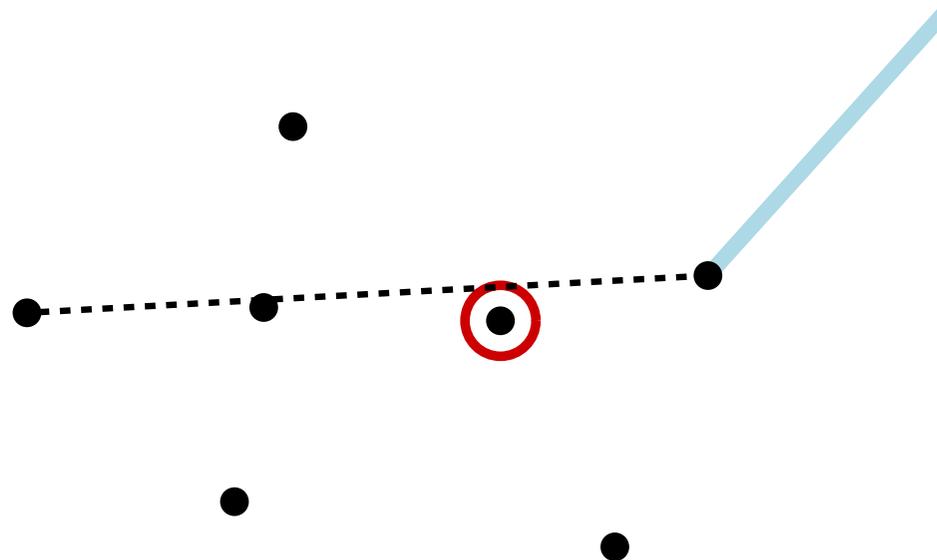
Aufgabe 2 (a + d)



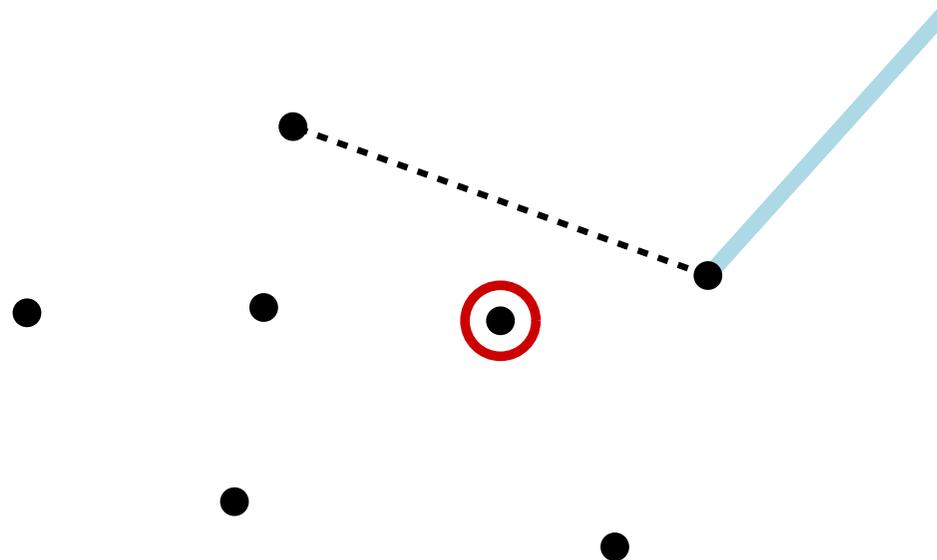
Aufgabe 2 (a + d)



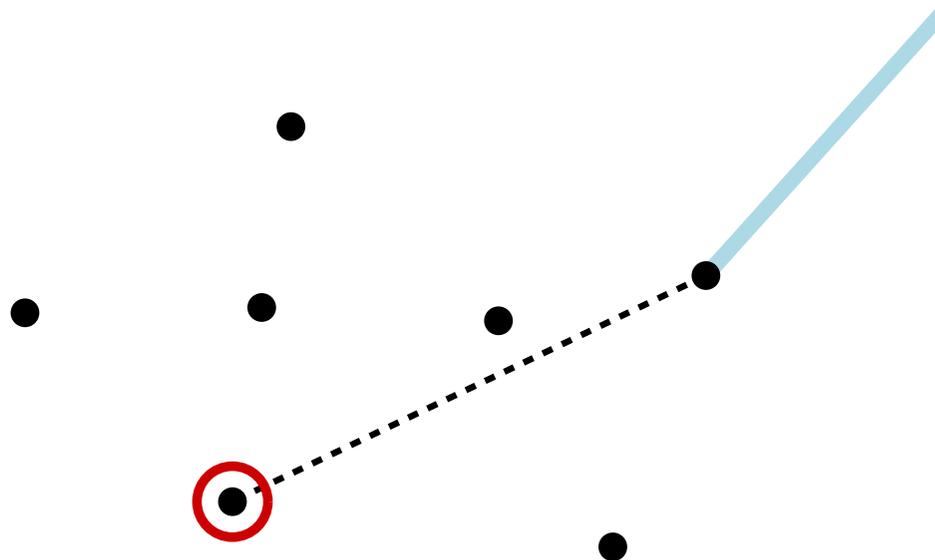
Aufgabe 2 (a + d)



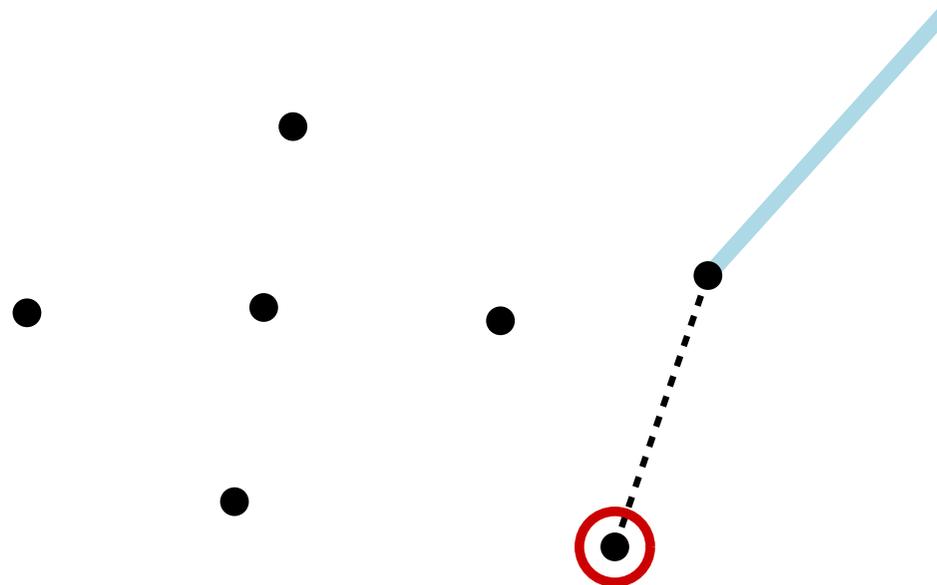
Aufgabe 2 (a + d)



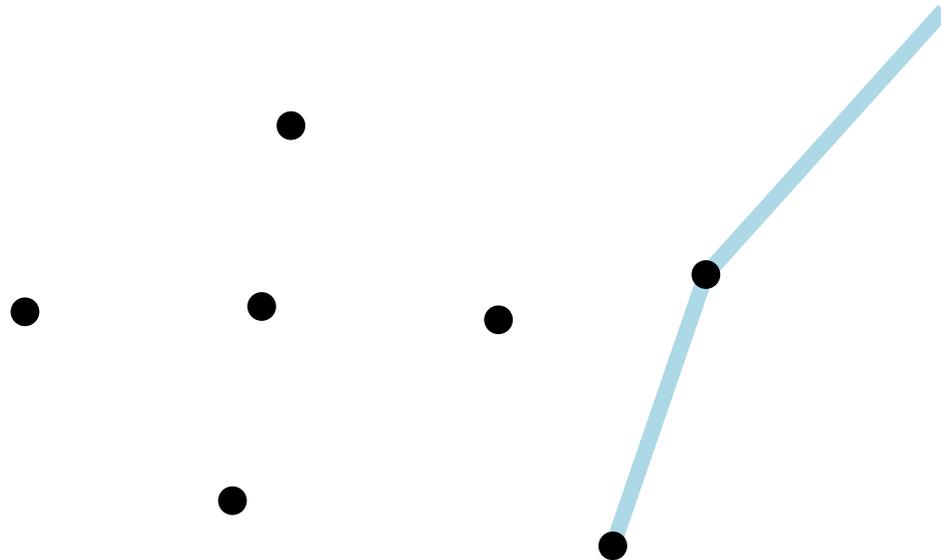
Aufgabe 2 (a + d)



Aufgabe 2 (a + d)



Aufgabe 2 (a + d)



Aufgabe 2 (a + d)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

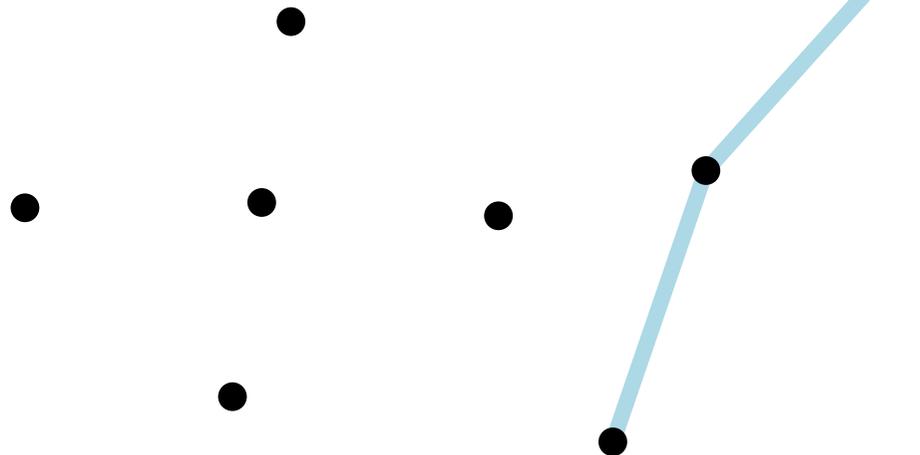
while true do

 Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

if $p_{j+1} == p_1$ **then**

break

return (p_1, \dots, p_h)



Aufgabe 2 (a + d)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

while true do

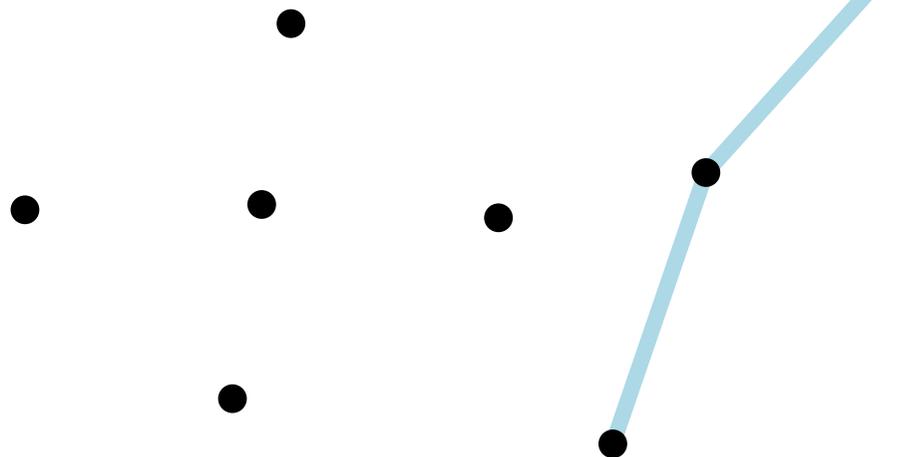
$\mathcal{O}(n)$

Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

if $p_{j+1} == p_1$ **then**

break

return (p_1, \dots, p_h)

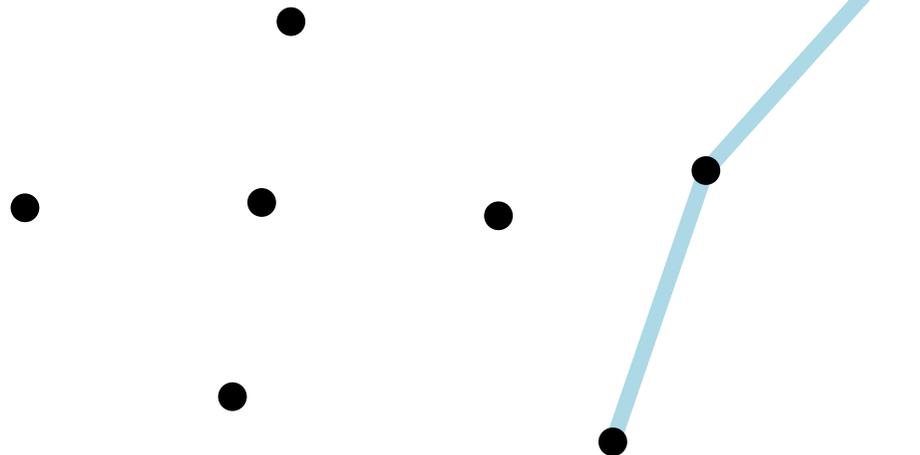


Aufgabe 2 (a + d)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

```
while true do  $\mathcal{O}(h)$   $\mathcal{O}(n)$   
    Wähle  $p_{j+1} \in P$  der den Winkel  $p_{j-1}p_jp_{j+1}$  maximiert.  
    if  $p_{j+1} == p_1$  then  
        break  
return  $(p_1, \dots, p_h)$ 
```



Aufgabe 2 (a + d)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

while true do $\mathcal{O}(h)$ $\mathcal{O}(n)$

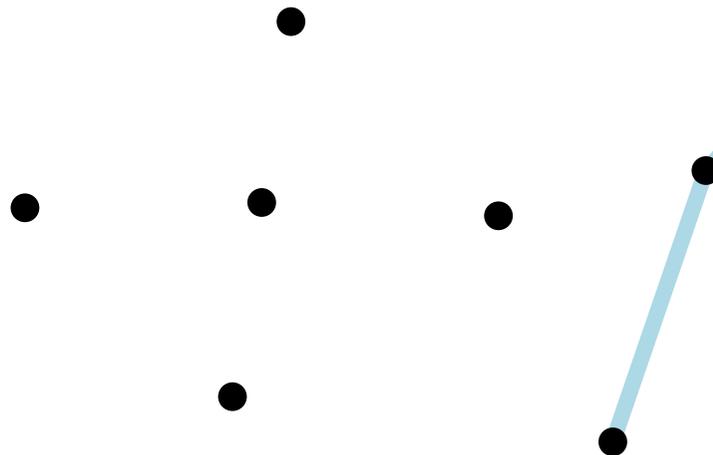
 Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

if $p_{j+1} == p_1$ **then**

 break

return (p_1, \dots, p_h)

Laufzeit: $\mathcal{O}(nh)$



Aufgabe 2 (b)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

while true do $\mathcal{O}(h)$ $\mathcal{O}(n)$

 Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

if $p_{j+1} == p_1$ **then**

 break

return (p_1, \dots, p_h)

Laufzeit: $\mathcal{O}(nh)$

Sonderfall:

Aufgabe 2 (b)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

while true do $\mathcal{O}(h)$ $\mathcal{O}(n)$

 Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

if $p_{j+1} == p_1$ **then**

 break

return (p_1, \dots, p_h)

Laufzeit: $\mathcal{O}(nh)$

Sonderfall:



Aufgabe 2 (c)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

```
while true do  $\mathcal{O}(h)$   $\mathcal{O}(n)$   
    Wähle  $p_{j+1} \in P$  der den Winkel  $p_{j-1}p_jp_{j+1}$  maximiert.  
    if  $p_{j+1} == p_1$  then  
        break  
return  $(p_1, \dots, p_h)$  Laufzeit:  $\mathcal{O}(nh)$ 
```

Korrektheit (Beweisidee):

- IA: p_1 liegt auf der konvexen Hülle

Aufgabe 2 (c)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

```
while true do  $\mathcal{O}(h)$   $\mathcal{O}(n)$   
    Wähle  $p_{j+1} \in P$  der den Winkel  $p_{j-1}p_jp_{j+1}$  maximiert.  
    if  $p_{j+1} == p_1$  then  
        break  
return  $(p_1, \dots, p_h)$  Laufzeit:  $\mathcal{O}(nh)$ 
```

Korrektheit (Beweisidee):

- IA: p_1 liegt auf der konvexen Hülle
- IV: Ersten i Punkte sind die ersten i Punkte der $CH(P)$

Aufgabe 2 (c)

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

```
while true do  $\mathcal{O}(h)$   $\mathcal{O}(n)$   
    Wähle  $p_{j+1} \in P$  der den Winkel  $p_{j-1}p_jp_{j+1}$  maximiert.  
    if  $p_{j+1} == p_1$  then  
        break  
return  $(p_1, \dots, p_h)$  Laufzeit:  $\mathcal{O}(nh)$ 
```

Korrektheit (Beweisidee):

- IA: p_1 liegt auf der konvexen Hülle
- IV: Ersten i Punkte sind die ersten i Punkte der $CH(P)$
- IS: Durch Konstruktion liegt p_{i+1} rechts von der Geraden durch p_i und $p_{i-1} \Rightarrow$ 'Rechtsknick'

Aufgabe 3

a) Berechnung konvexer Hülle benötigt $\Omega(n \log n)$.

Aufgabe 3

- a) Berechnung konvexer Hülle benötigt $\Omega(n \log n)$.
- b) Gegeben: einfaches [nicht notwendigerweise konvexes] Polygon. Konstruiere daraus in $\mathcal{O}(n)$ seine konvexe Hülle. Warum ist das kein Widerspruch zu a) ?

Aufgabe 3

- a) Berechnung konvexer Hülle benötigt $\Omega(n \log n)$.
- b) Gegeben: einfaches [nicht notwendigerweise kovexes] Polygon. Konstruiere daraus in $\mathcal{O}(n)$ seine konvexe Hülle. Warum ist das kein Widerspruch zu a) ?

UpperConvexHull(P)

$\langle p_1, p_2, \dots, p_n \rangle \leftarrow$ sortiere P von links nach rechts

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

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

$L.append(p_i)$

while $|L| > 2$ **and** letzte 3 Punkte in L kein Rechtsknick **do**

 entferne vorletzten Punkt aus L

return L

Übungsblatt 1

Optimaler Algorithmus

Optimalität?

Wann ist welcher Algorithmus besser?

Optimalität?

Wann ist welcher Algorithmus besser?

Viele Punkte auf $CH(P)$: $\mathcal{O}(n \log n)$ Graham Scan

Wenige Punkte auf $CH(P)$: $\mathcal{O}(nh)$ Gift Wrapping

Optimalität?

Wann ist welcher Algorithmus besser?

Viele Punkte auf $CH(P)$: $\mathcal{O}(n \log n)$ Graham Scan

Wenige Punkte auf $CH(P)$: $\mathcal{O}(nh)$ Gift Wrapping

Geht es vielleicht noch besser?

Optimalität?

Wann ist welcher Algorithmus besser?

Viele Punkte auf $CH(P)$: $\mathcal{O}(n \log n)$ Graham Scan

Wenige Punkte auf $CH(P)$: $\mathcal{O}(nh)$ Gift Wrapping

Geht es vielleicht noch besser?

ja

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

└ Berechne per GrahamScan $CH(P_i)$

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

for j von 1 bis h **do**

└ **for** i von 1 bis n/h **do**

└└ Berechne Punkt $q_i \in P_i$ der den Winkel

└└└ $p_{j-1}p_jq_i$ maximiert.

└└ $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$

return (p_1, \dots, p_h)

Chans Algorithmus

Angenommen, wir kennen h :

$\text{ChanHull}(P, h)$

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

└ Berechne per GrahamScan $CH(P_i)$

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

for j von 1 bis h **do**

└ **for** i von 1 bis n/h **do**

└└ Berechne Punkt $q_i \in P_i$ der den Winkel

└└ $p_{j-1}p_jq_i$ maximiert.

└ $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$

return (p_1, \dots, p_h)

Chans Algorithmus

Angenommen, wir kennen h :

$\text{ChanHull}(P, h)$

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

└ Berechne per GrahamScan $CH(P_i)$

GrahamScan

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

for j von 1 bis h **do**

Gift Wrapping

└ **for** i von 1 bis n/h **do**

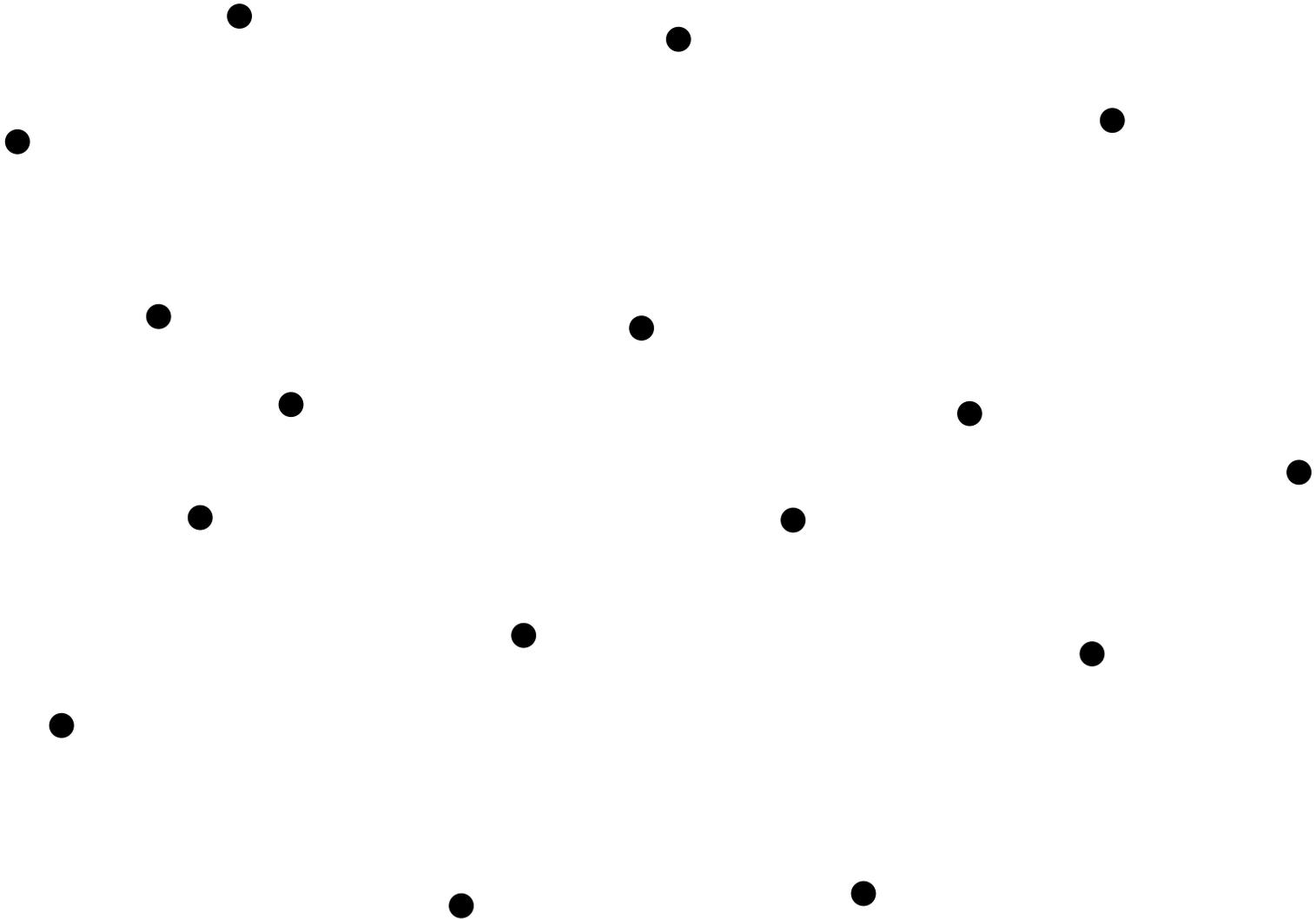
└ Berechne Punkt $q_i \in P_i$ der den Winkel
└ $p_{j-1}p_jq_i$ maximiert.

└ $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$

return (p_1, \dots, p_h)

Beispiel

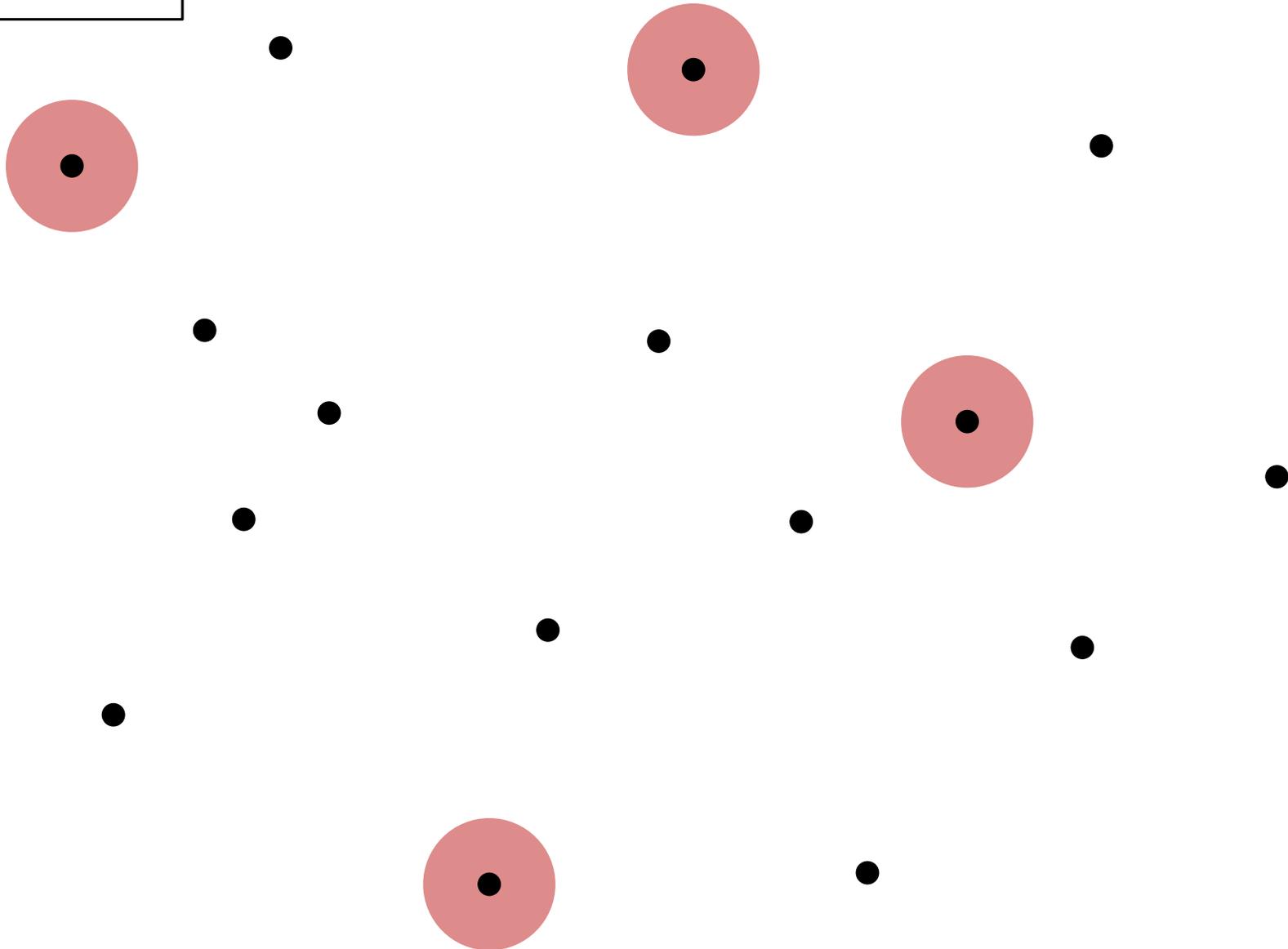
GrahamScan



$n = 16$

Beispiel

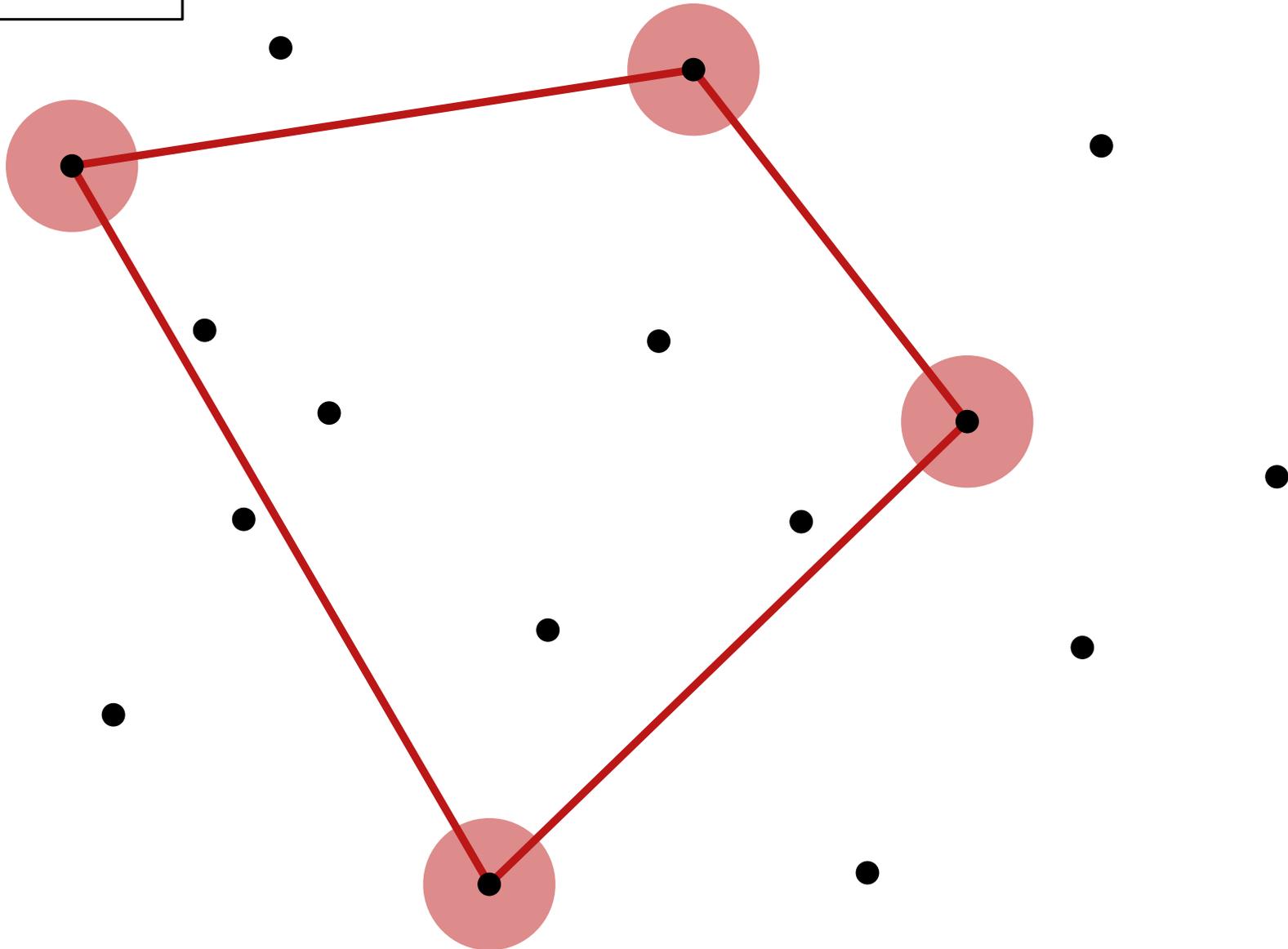
GrahamScan



$n = 16$

Beispiel

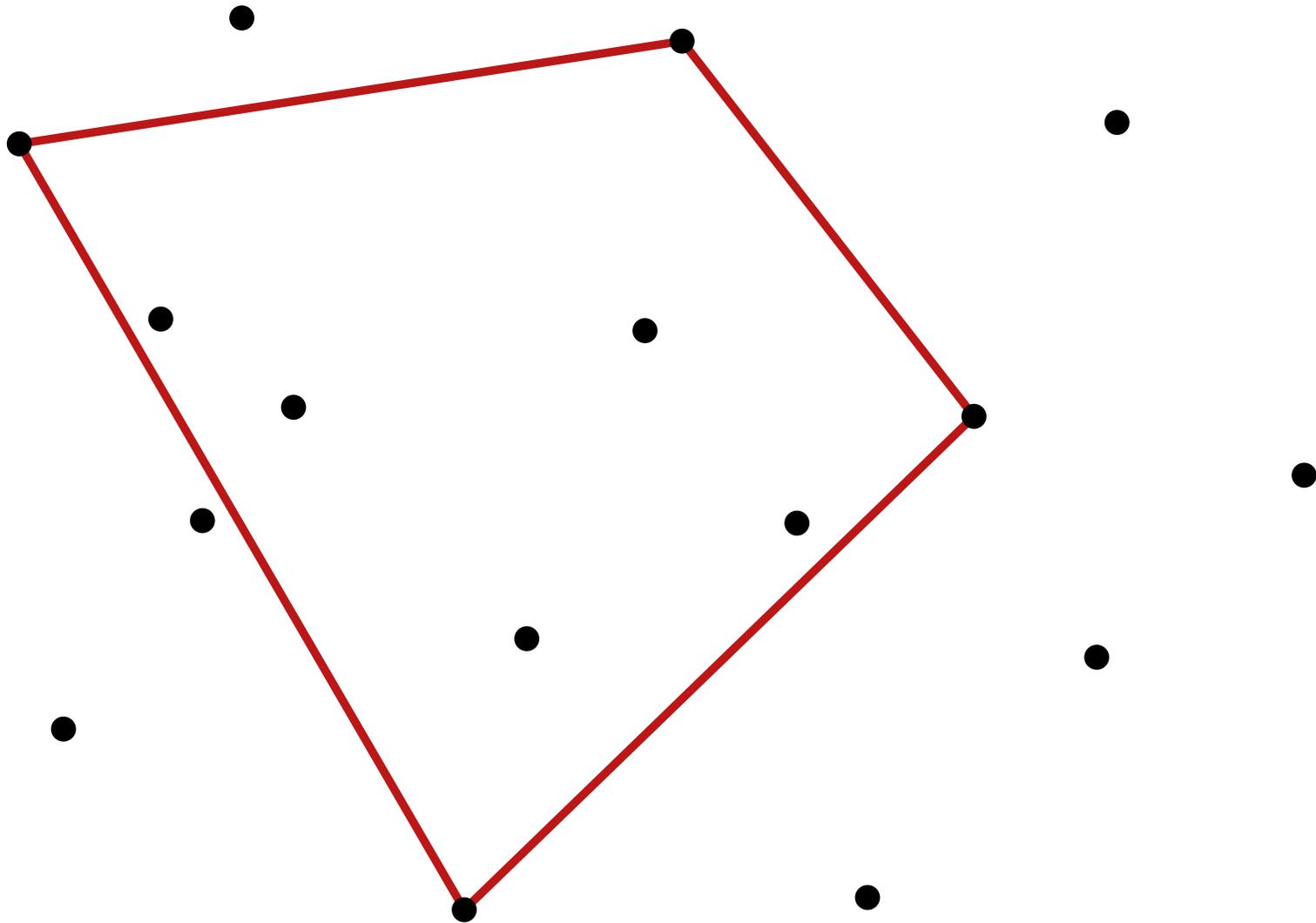
GrahamScan



$n = 16$

Beispiel

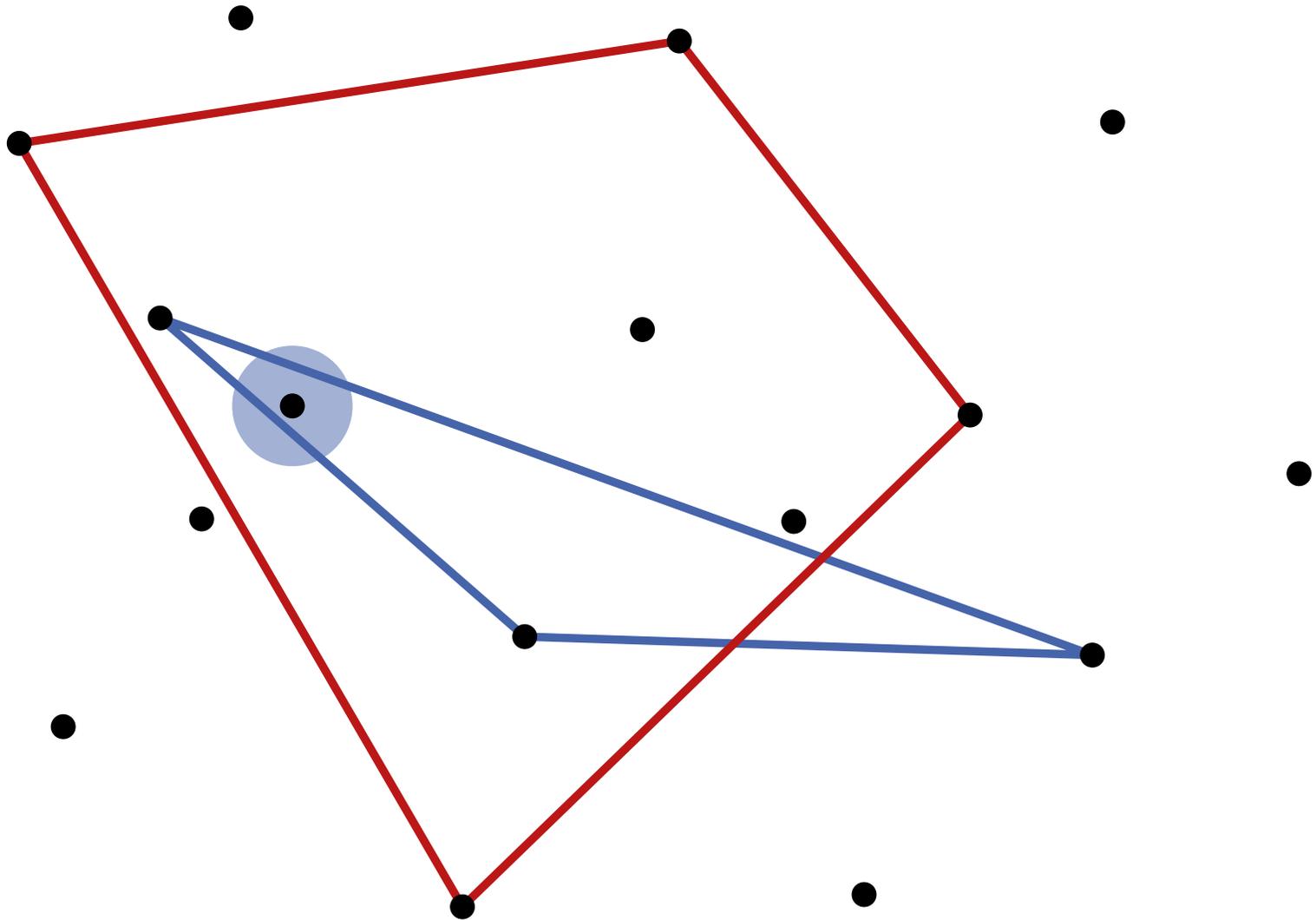
GrahamScan



$n = 16$

Beispiel

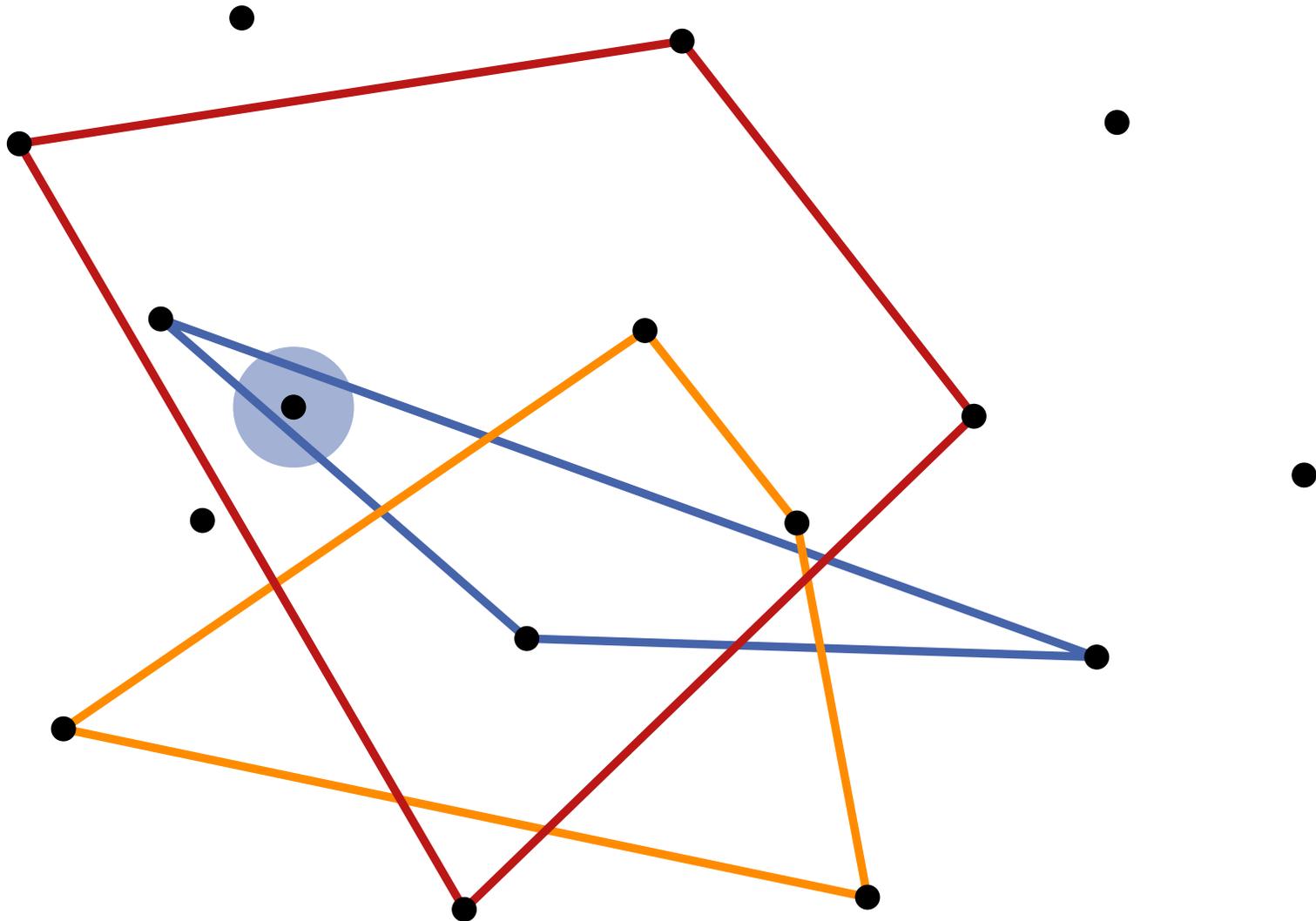
GrahamScan



$n = 16$

Beispiel

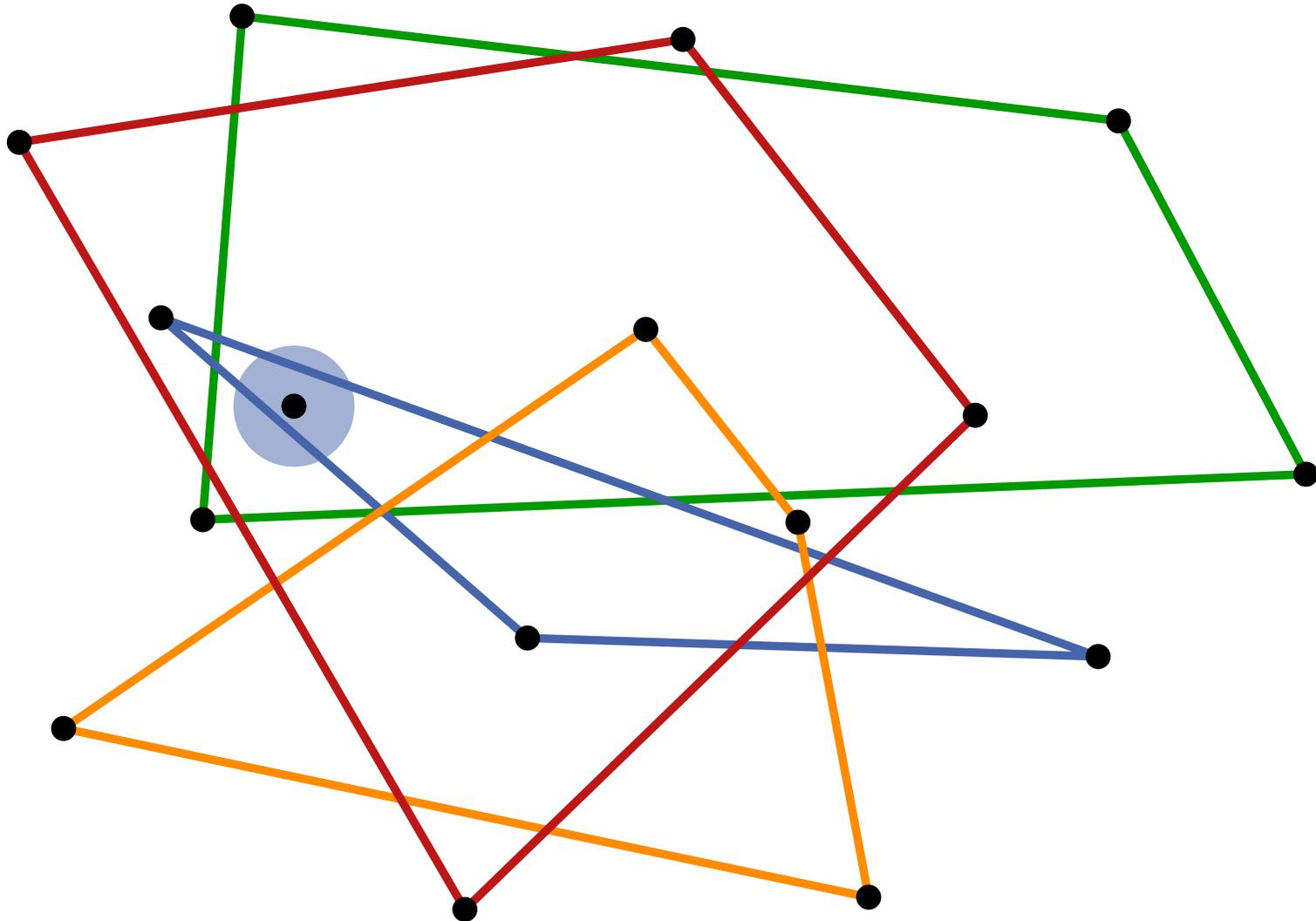
GrahamScan



$n = 16$

Beispiel

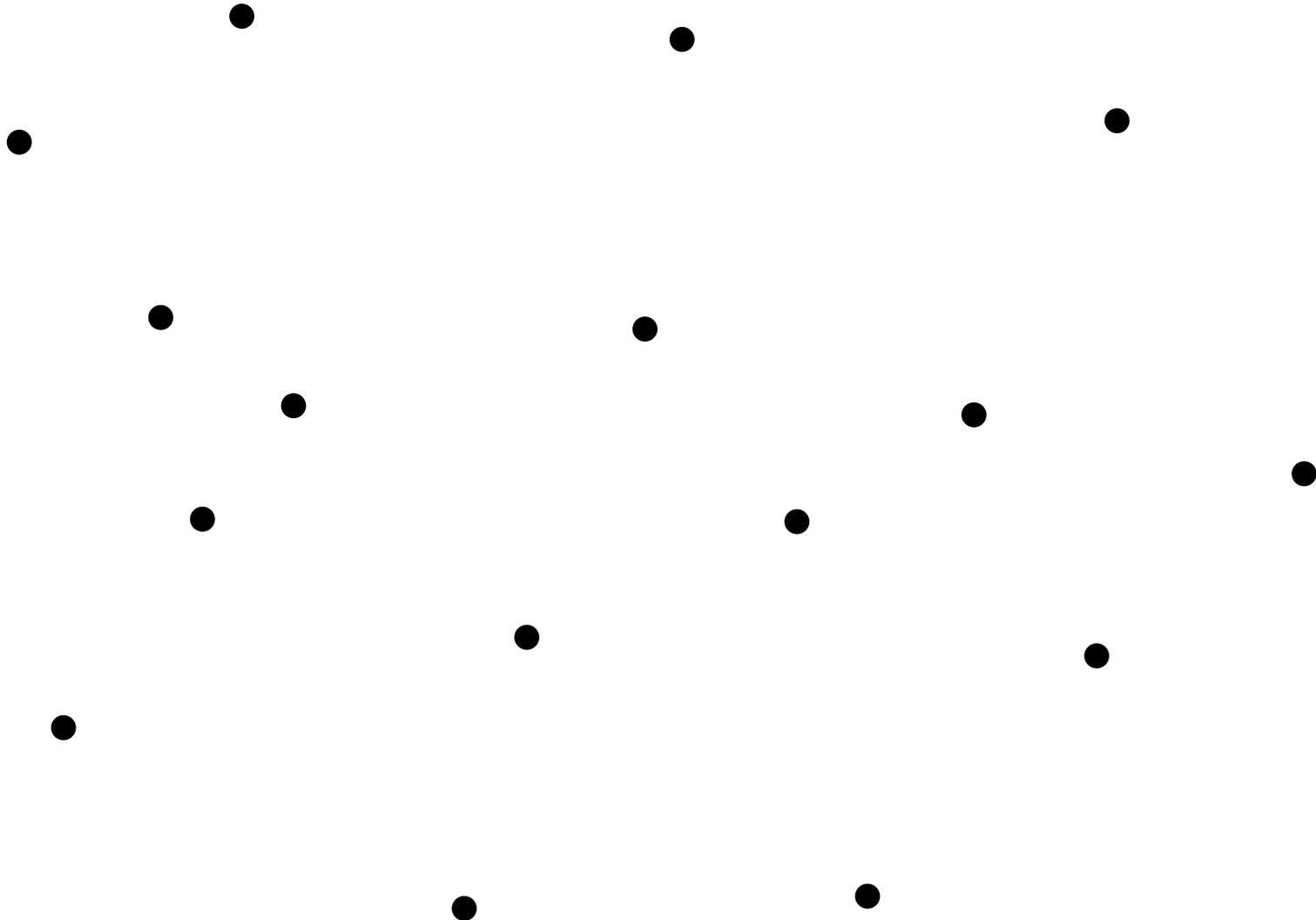
GrahamScan



$n = 16$

Beispiel

GrahamScan

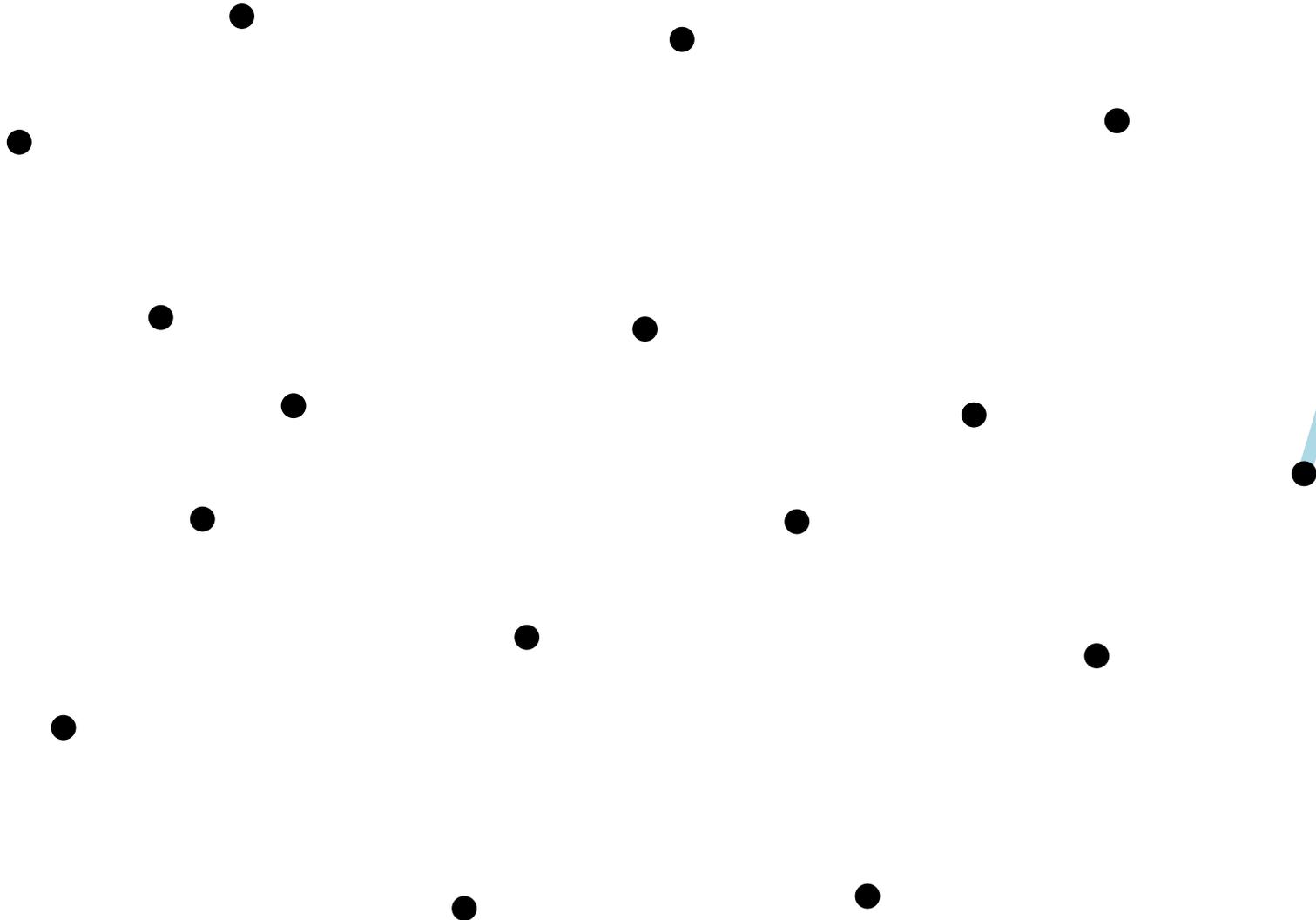


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

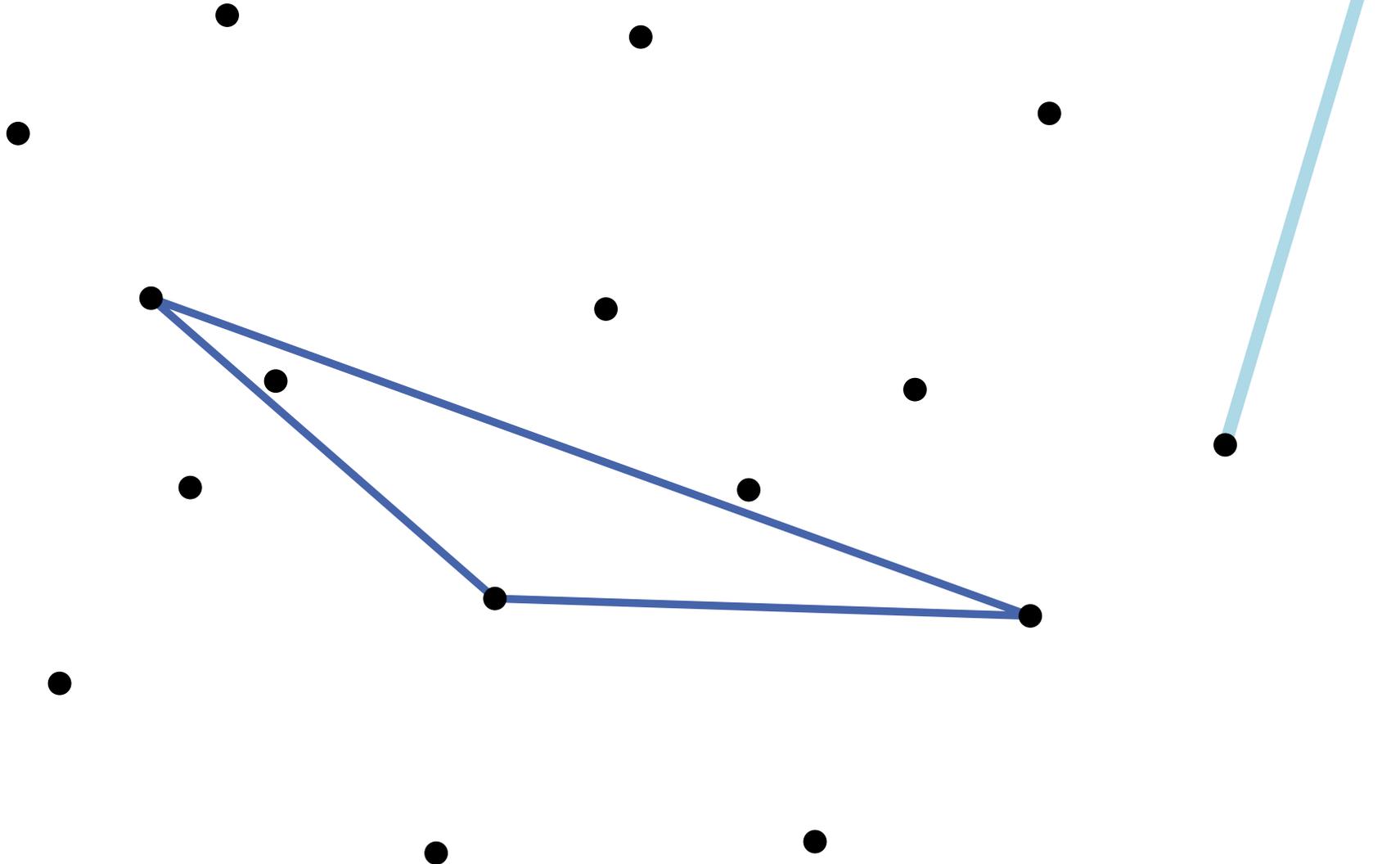


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

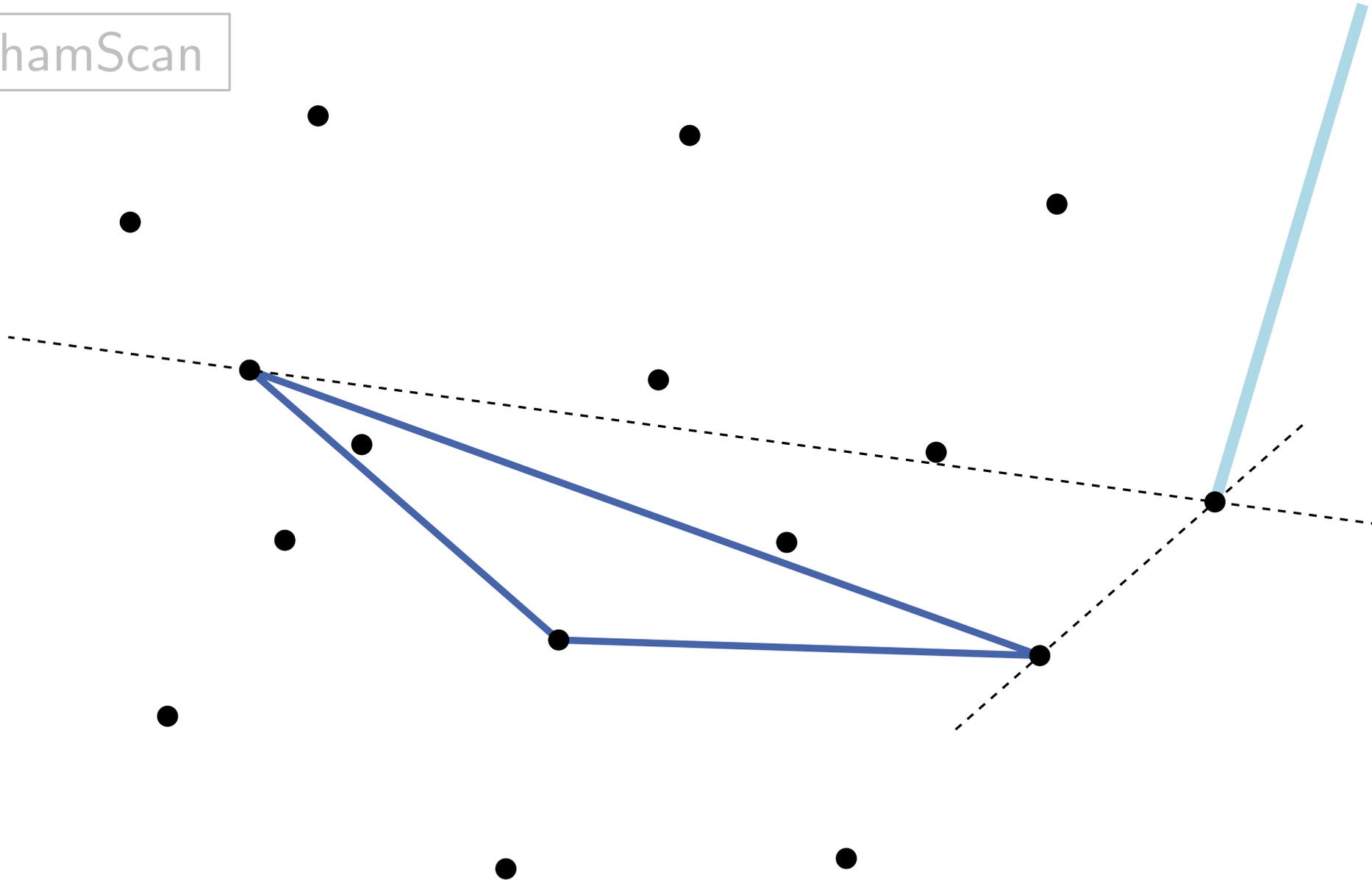


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

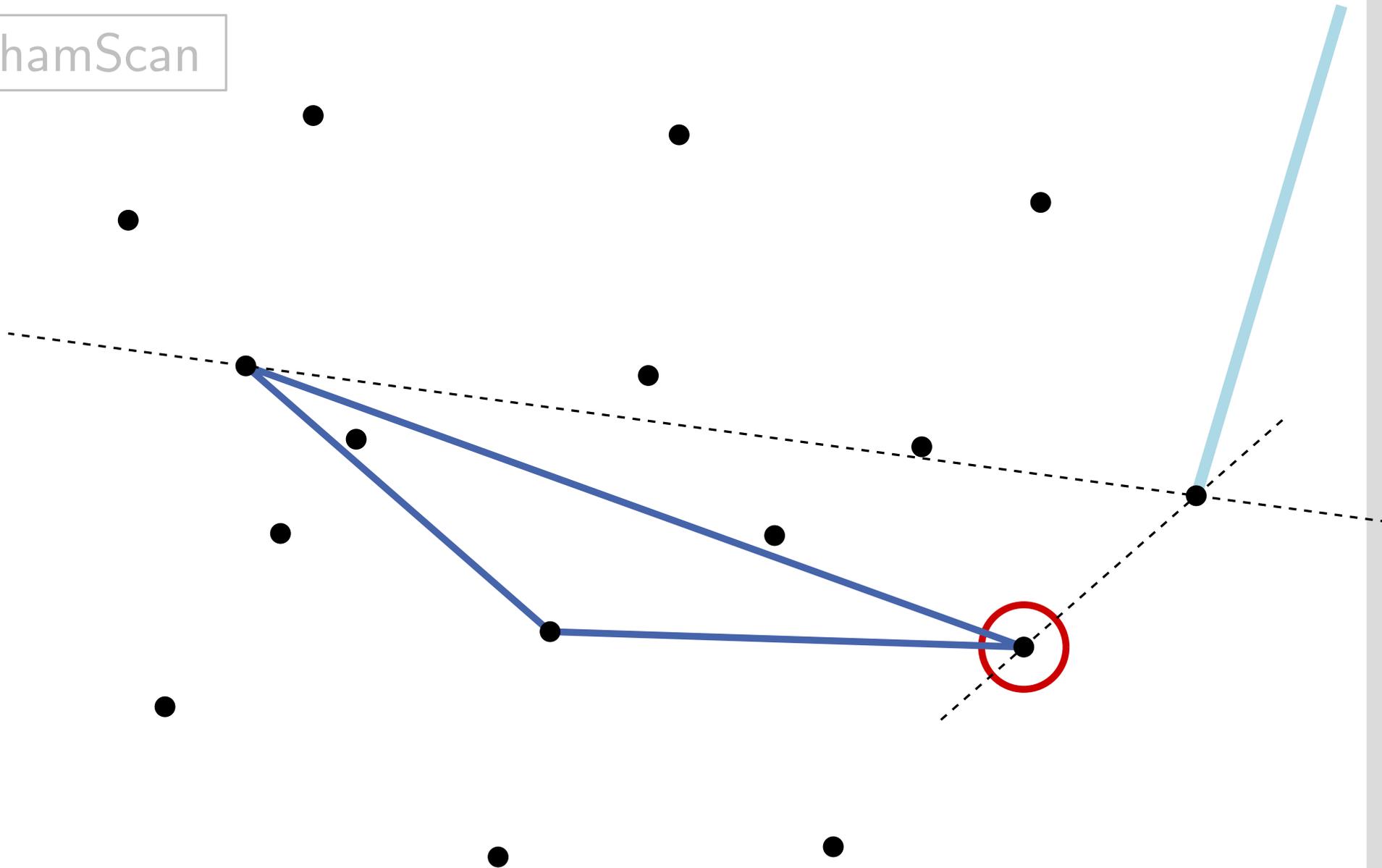


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

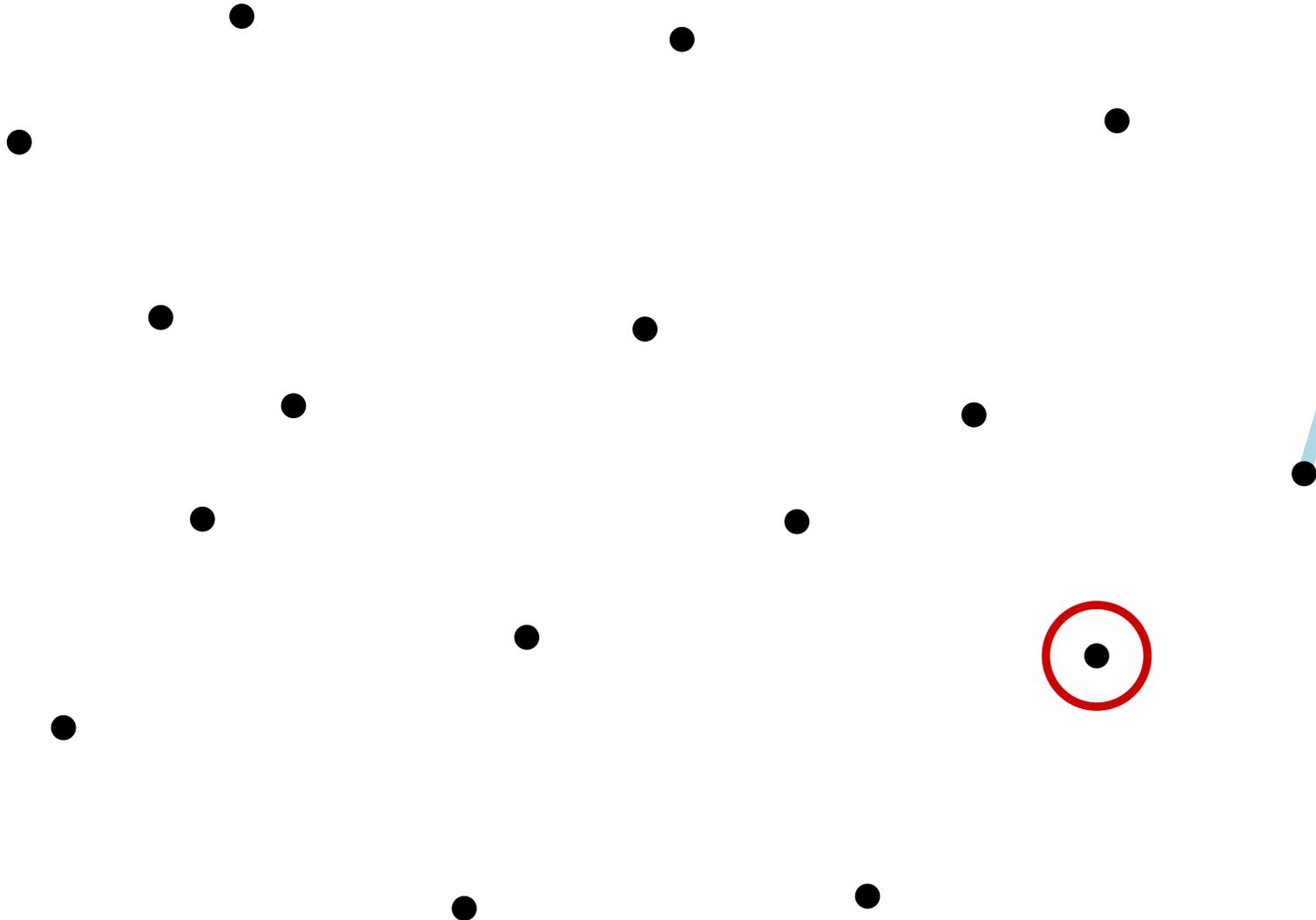


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

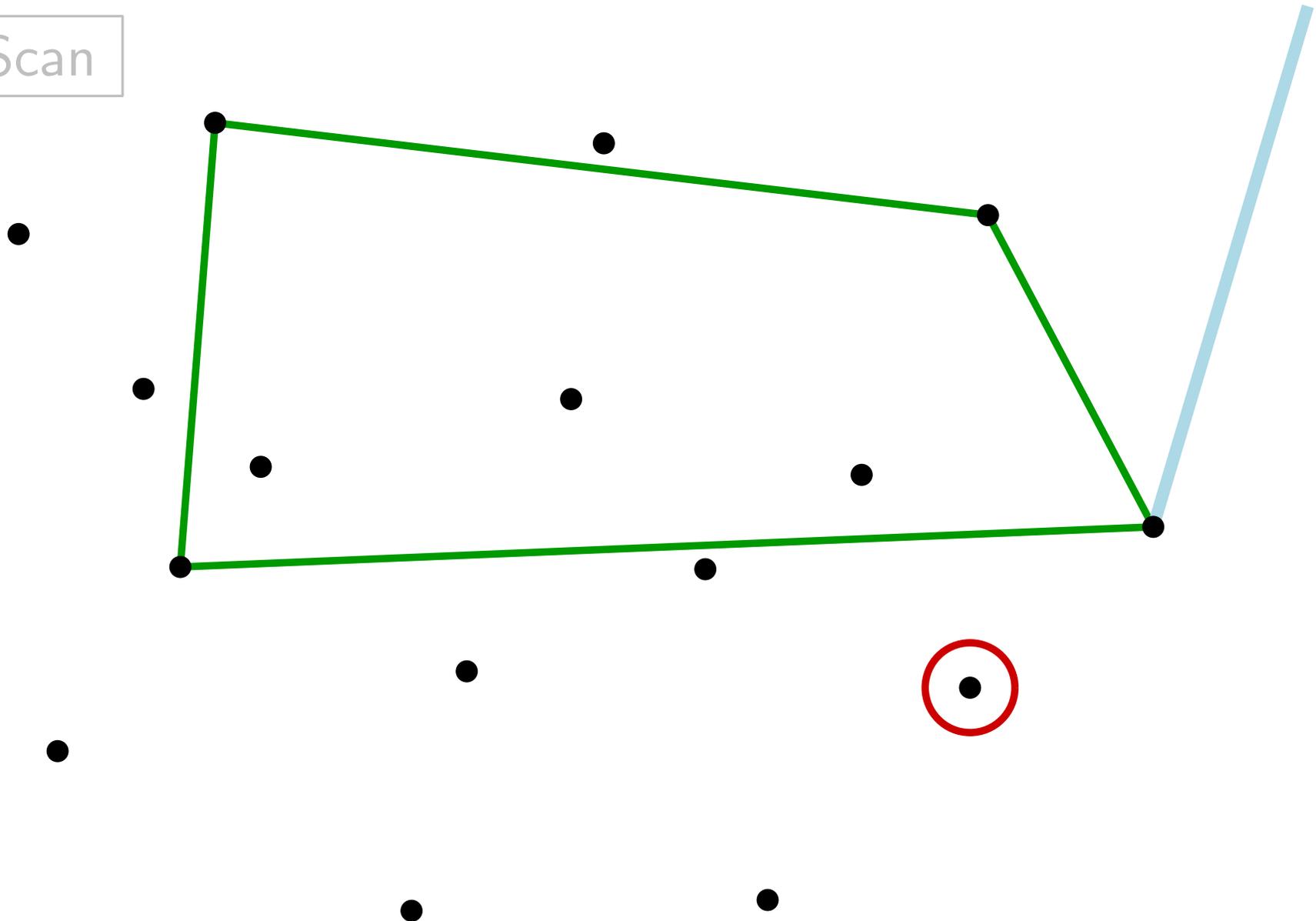


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

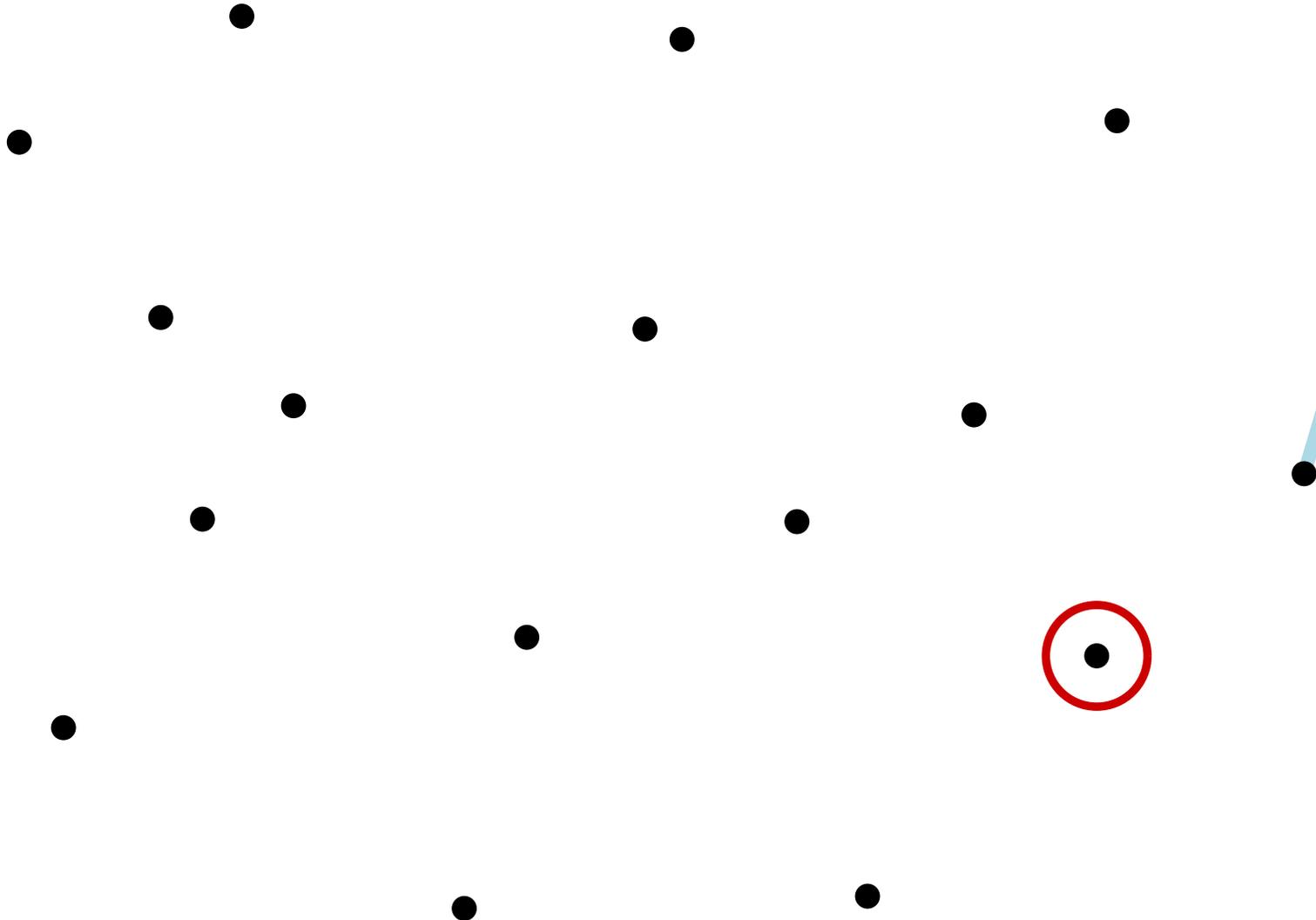


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

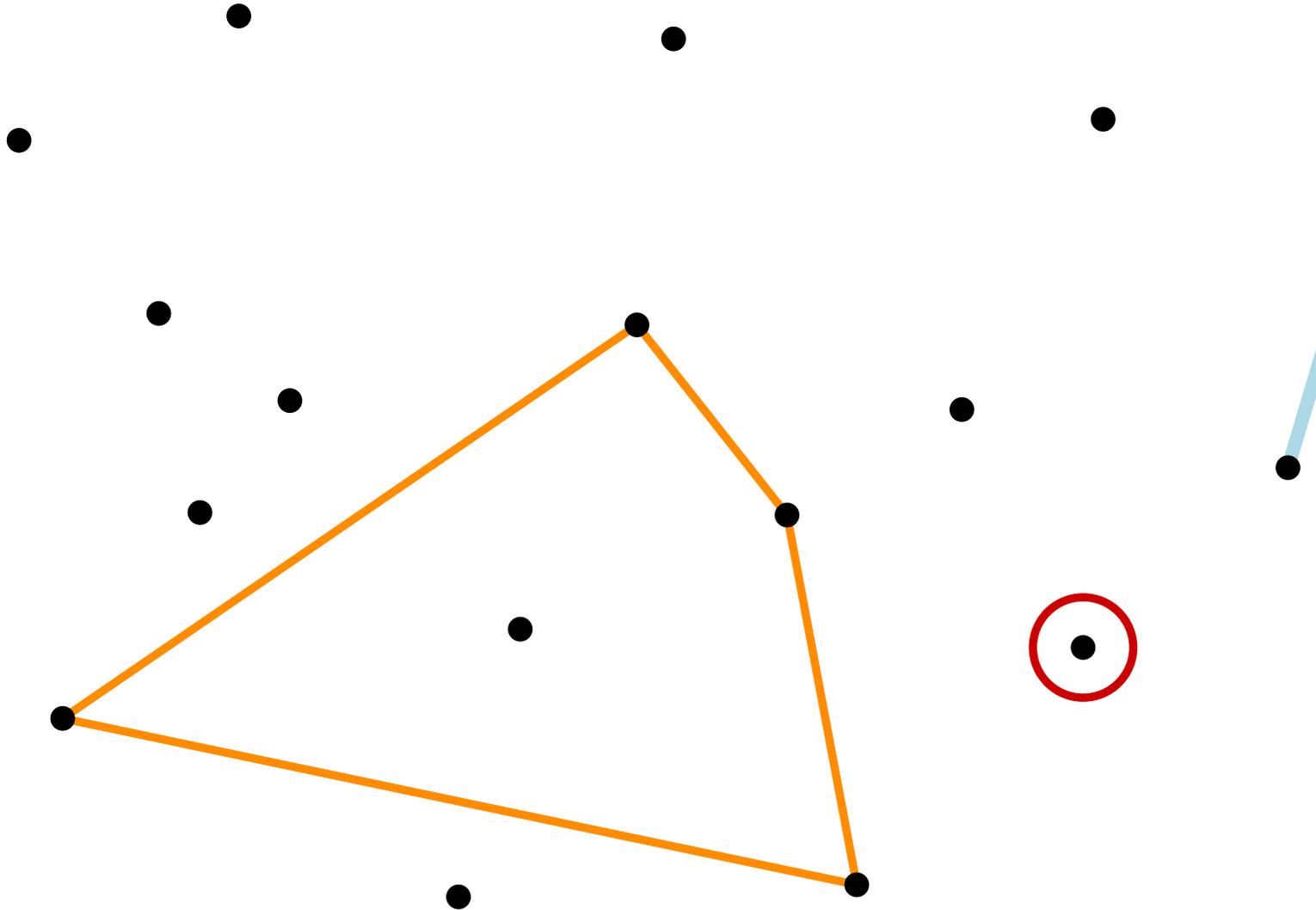


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

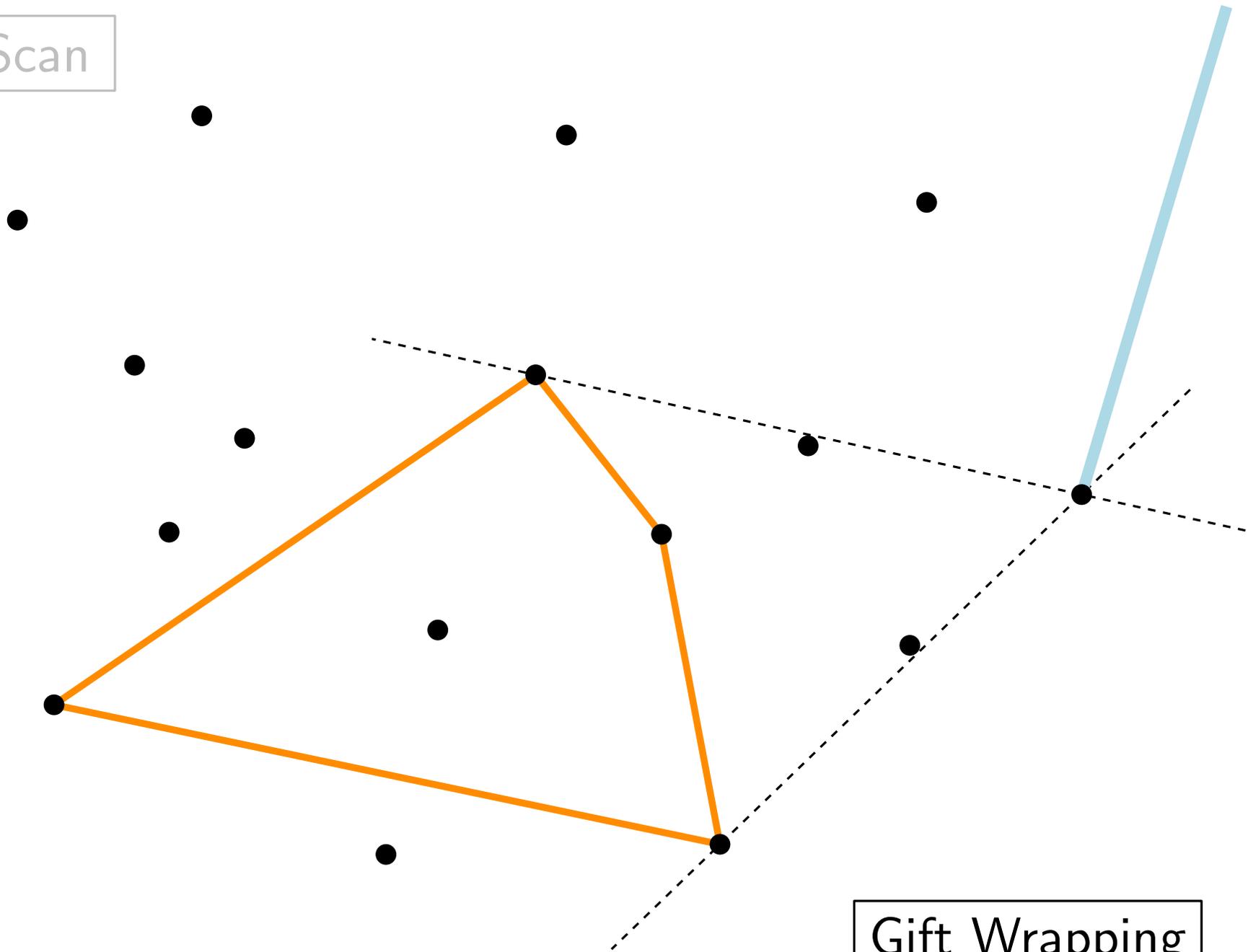


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

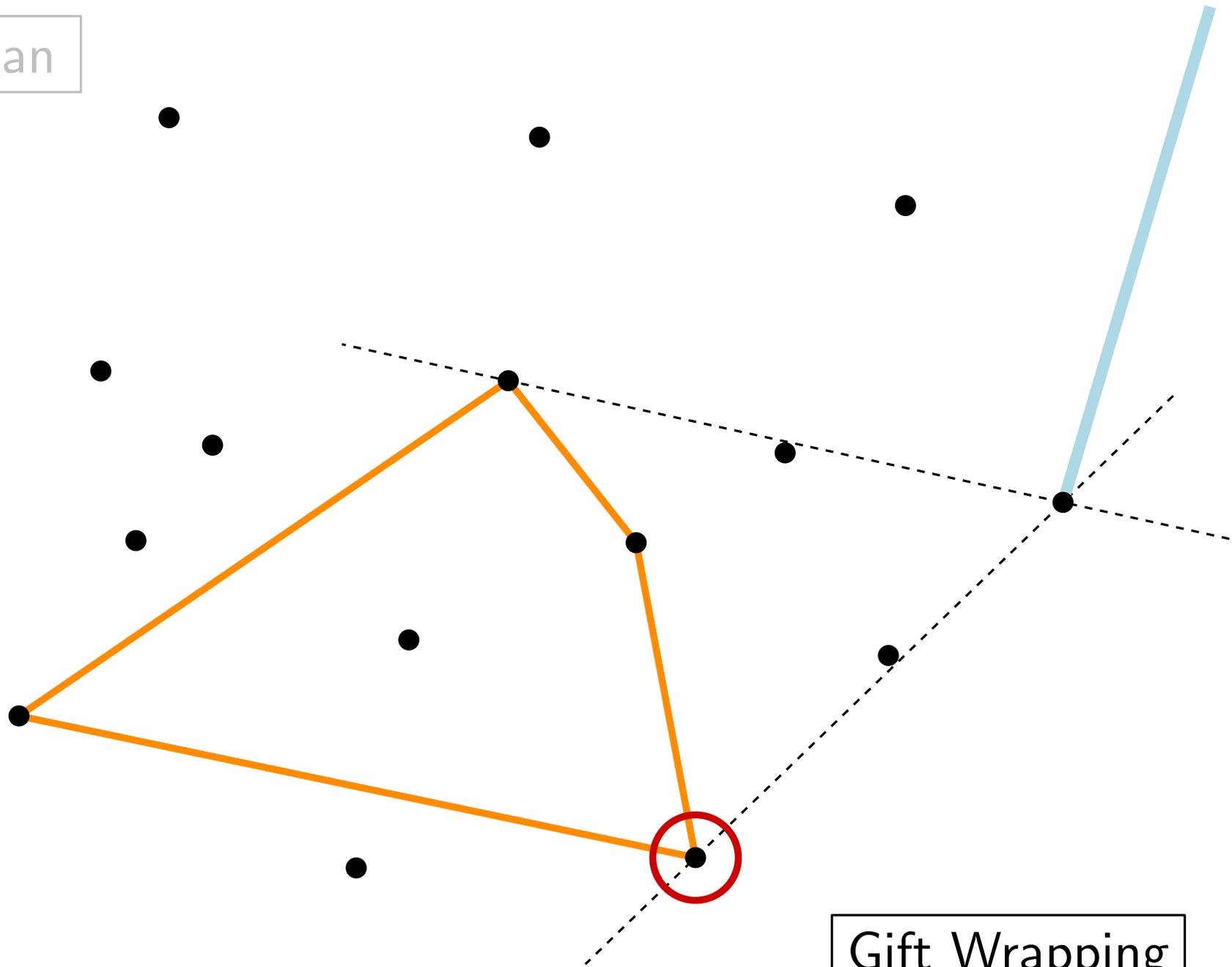


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

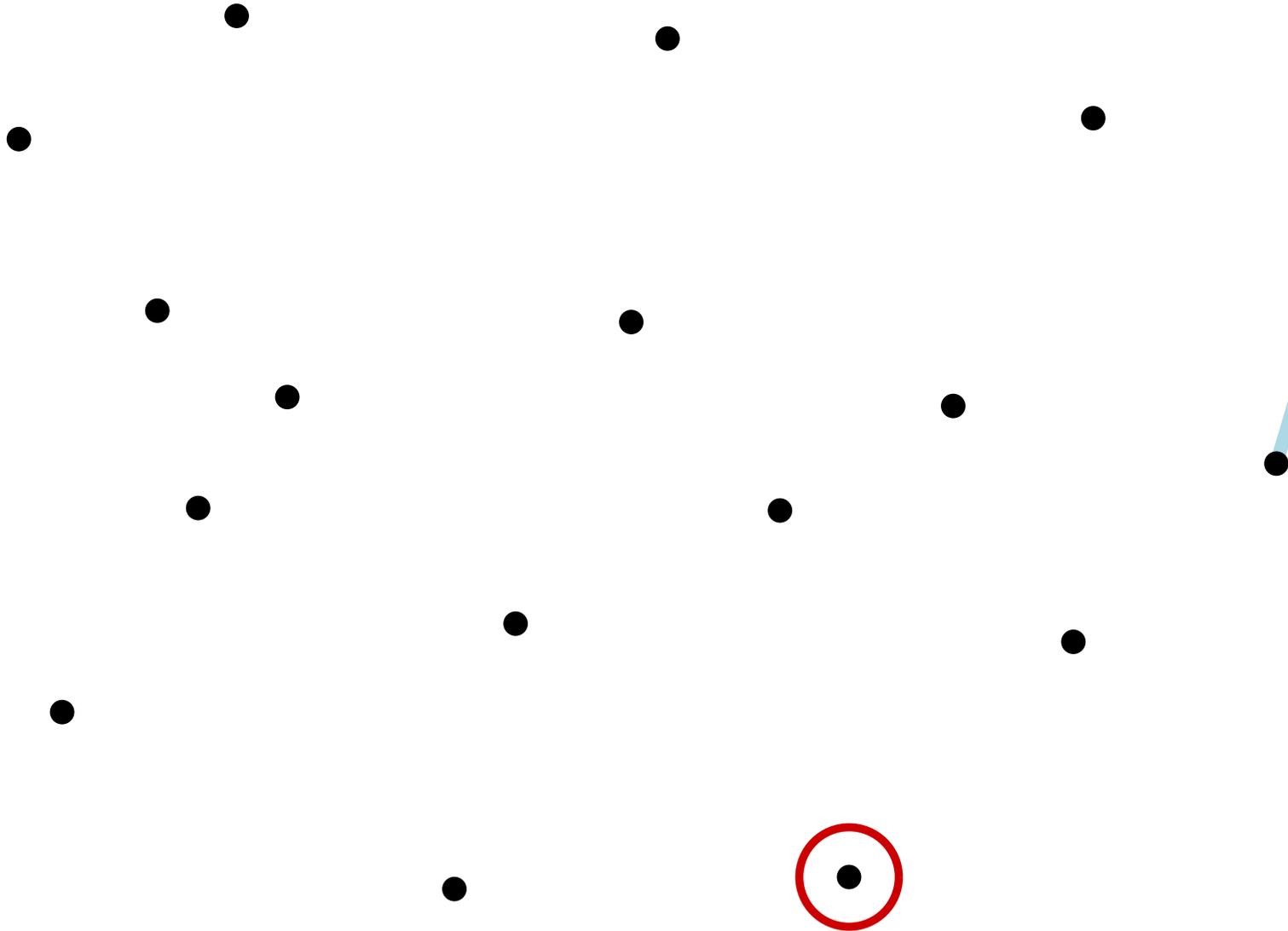


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

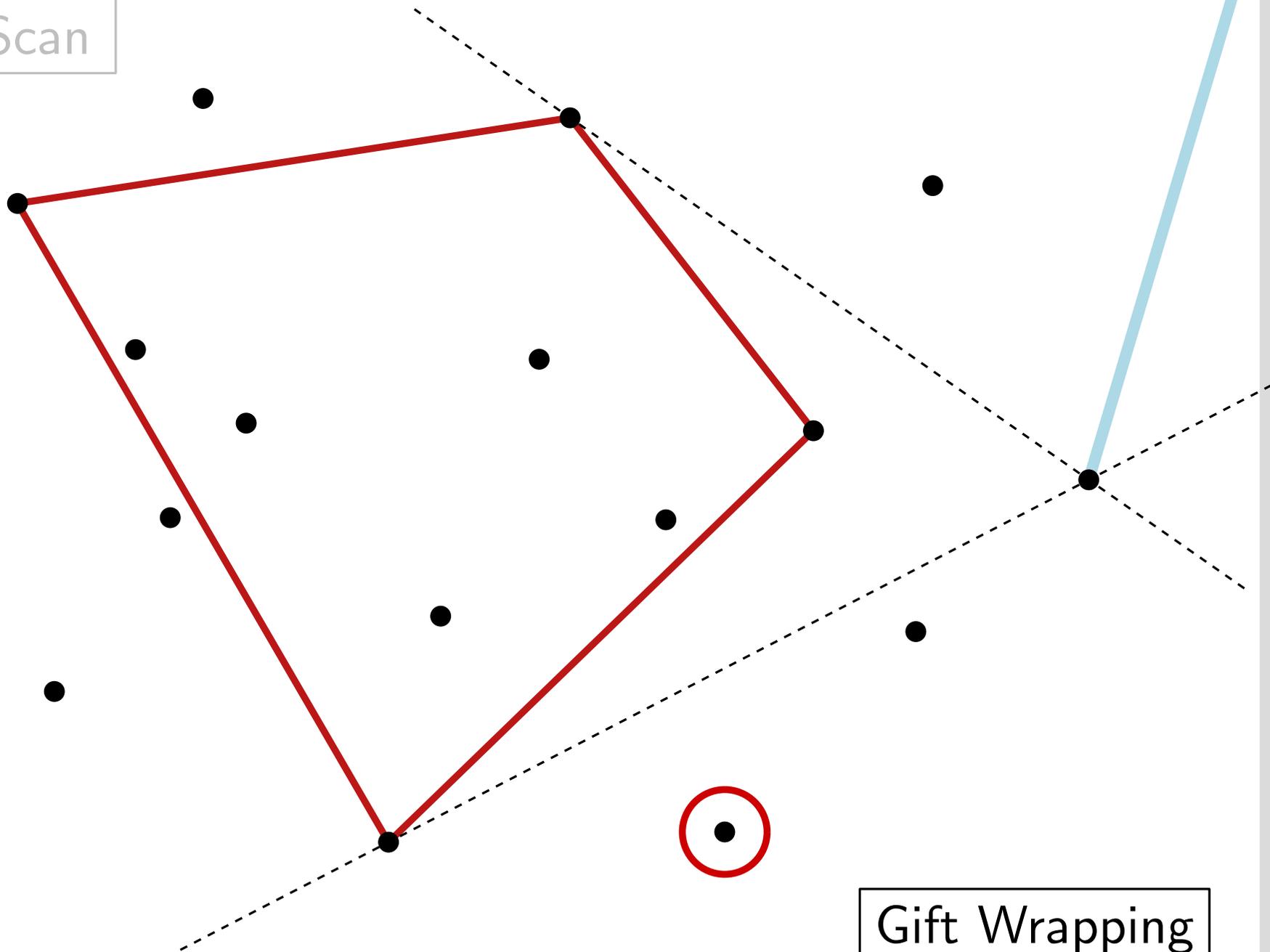


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

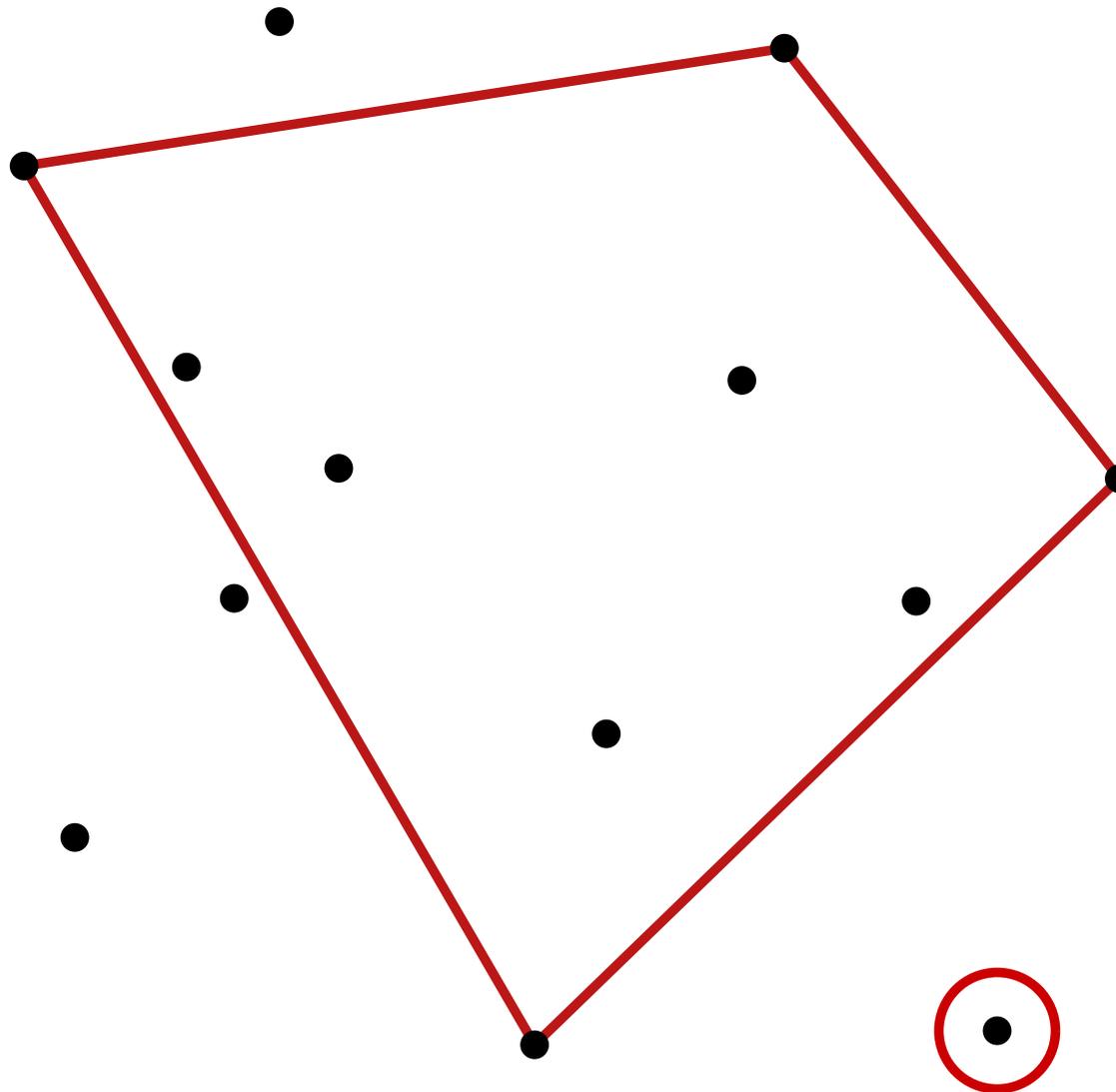


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

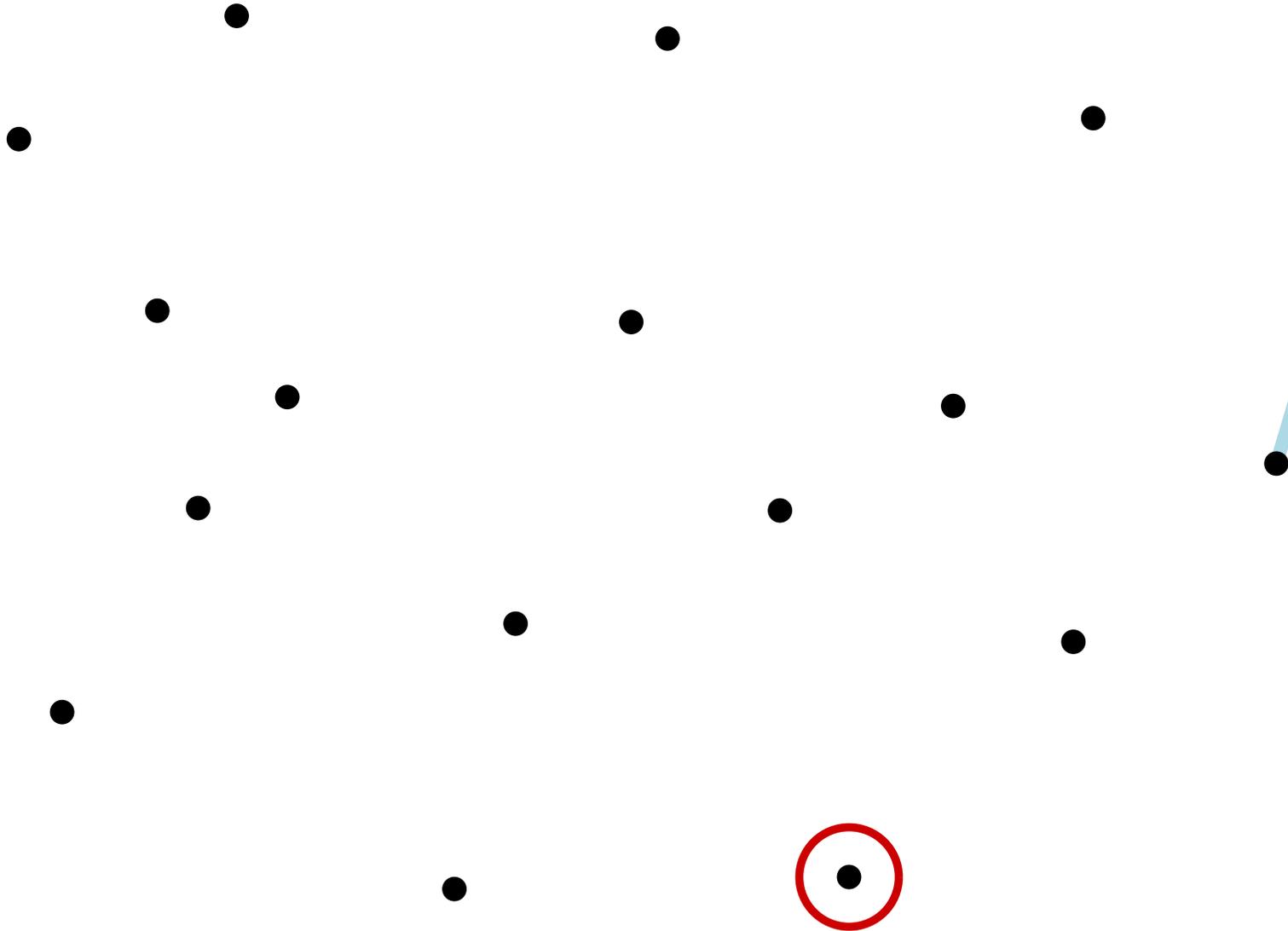


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

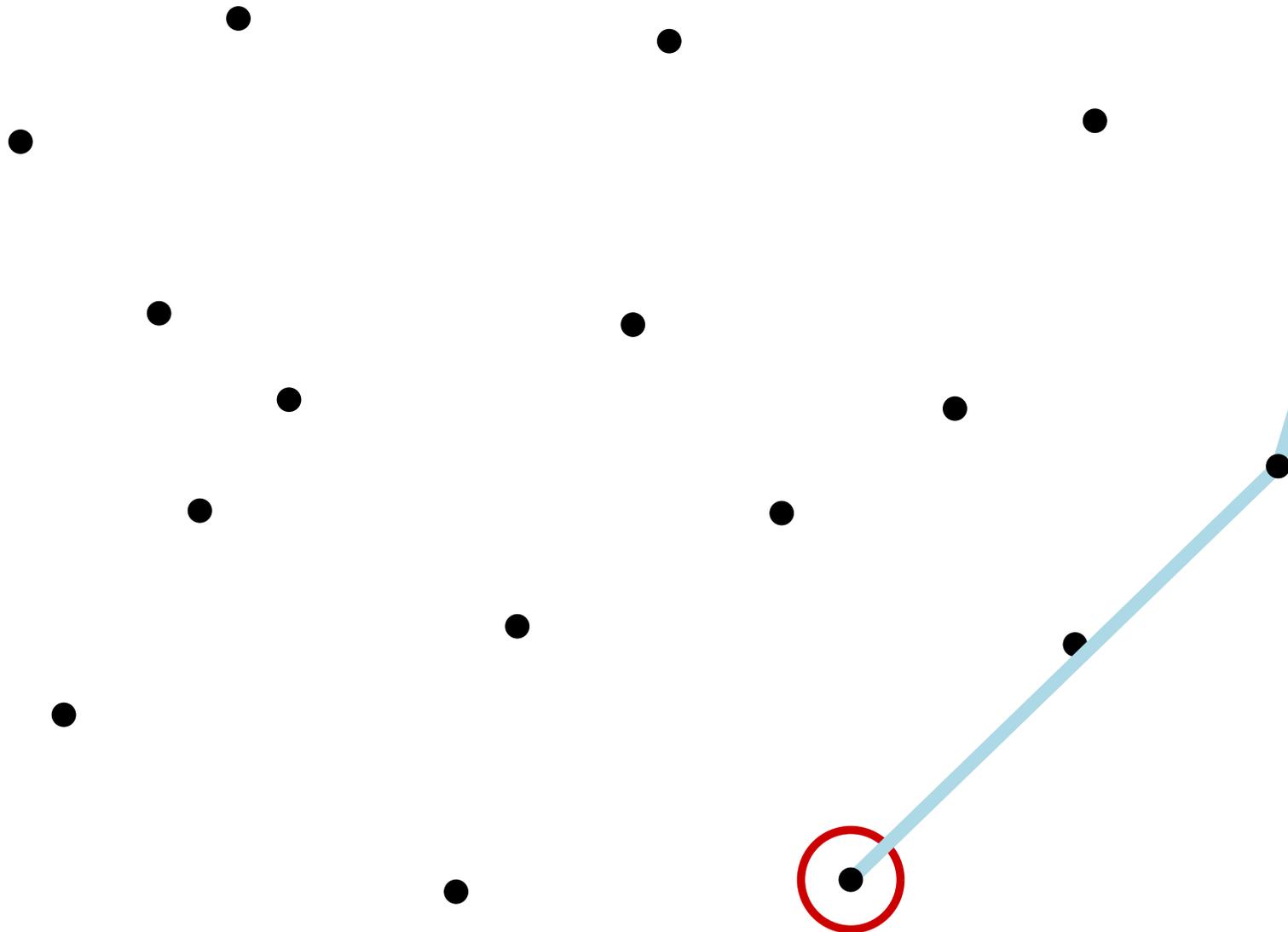


$n = 16$

Gift Wrapping

Beispiel

GrahamScan

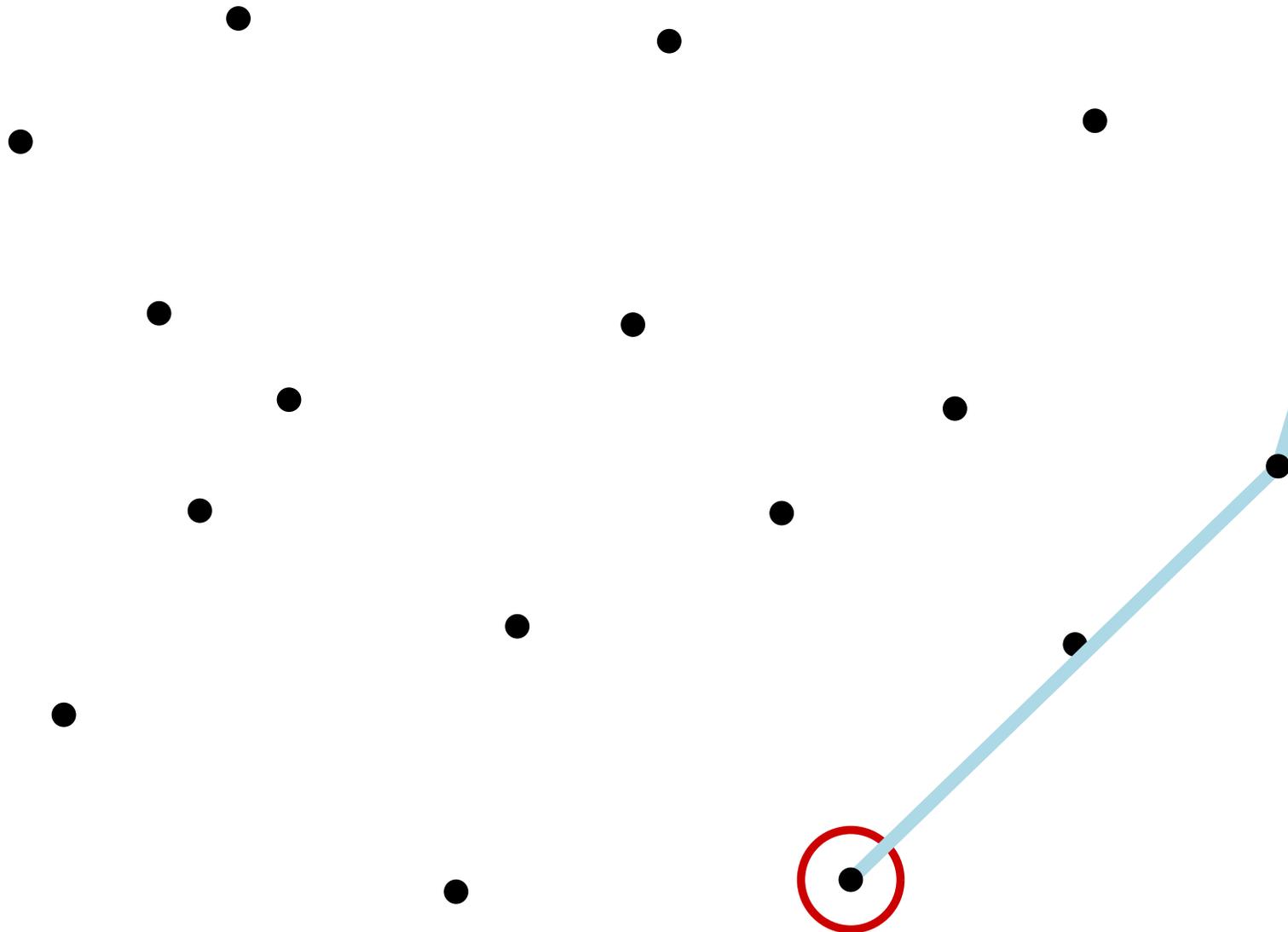


$n = 16$

Gift Wrapping

Beispiel

GrahamScan



$n = 16$

Gift Wrapping

Chans Algorithmus

Angenommen, wir kennen h :

$\text{ChanHull}(P, h)$

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

└ Berechne per GrahamScan $CH(P_i)$

GrahamScan

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

for j von 1 bis h **do**

Gift Wrapping

└ **for** i von 1 bis n/h **do**

└└ Berechne Punkt $q_i \in P_i$ der den Winkel

└└ $p_{j-1}p_jq_i$ maximiert.

└ $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$

return (p_1, \dots, p_h)

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

└ Berechne per GrahamScan $CH(P_i)$

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

for j von 1 bis h **do**

└ **for** i von 1 bis n/h **do**

└└ Berechne Punkt $q_i \in P_i$ der den Winkel
└└ $p_{j-1}p_jq_i$ maximiert.

└ $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$

return (p_1, \dots, p_h)

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

└ Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

for j von 1 bis h **do**

└ **for** i von 1 bis n/h **do**

└└ Berechne Punkt $q_i \in P_i$ der den Winkel
└└ $p_{j-1}p_jq_i$ maximiert.

└ $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$

return (p_1, \dots, p_h)

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

```
for  $i$  von 1 bis  $n/h$  do  $\mathcal{O}(n/h)$   
└ Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(h \log h)$ 
```

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

```
for  $j$  von 1 bis  $h$  do
```

```
└ for  $i$  von 1 bis  $n/h$  do
```

```
└└ Berechne Punkt  $q_i \in P_i$  der den Winkel
```

```
└└└  $p_{j-1}p_jq_i$  maximiert.
```

```
└└  $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$ 
```

```
return  $(p_1, \dots, p_h)$ 
```

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

```
for  $i$  von 1 bis  $n/h$  do  $\mathcal{O}(n/h)$   
└ Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(h \log h)$ 
```

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

```
for  $j$  von 1 bis  $h$  do
```

```
└ for  $i$  von 1 bis  $n/h$  do
```

```
└└ Berechne Punkt  $q_i \in P_i$  der den Winkel  
└└  $p_{j-1}p_jq_i$  maximiert.
```

```
└  $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$ 
```

```
return  $(p_1, \dots, p_h)$ 
```

$\mathcal{O}(n \log h)$

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

```
for  $i$  von 1 bis  $n/h$  do  $\mathcal{O}(n/h)$   
└ Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(h \log h)$ 
```

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

```
for  $j$  von 1 bis  $h$  do
```

```
└ for  $i$  von 1 bis  $n/h$  do
```

```
└└ Berechne Punkt  $q_i \in P_i$  der den Winkel  $\mathcal{O}(\log h)$   
└└  $p_{j-1}p_jq_i$  maximiert.
```

```
└  $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$ 
```

```
return  $(p_1, \dots, p_h)$ 
```

$\mathcal{O}(n \log h)$

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

```
for  $i$  von 1 bis  $n/h$  do  $\mathcal{O}(n/h)$   
└ Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(h \log h)$ 
```

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

```
for  $j$  von 1 bis  $h$  do  $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$ 
```

```
└ for  $i$  von 1 bis  $n/h$  do
```

```
└└ Berechne Punkt  $q_i \in P_i$  der den Winkel  $\mathcal{O}(\log h)$   
└└  $p_{j-1}p_jq_i$  maximiert.
```

```
└  $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$ 
```

```
return  $(p_1, \dots, p_h)$ 
```

$\mathcal{O}(n \log h)$

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

```
for  $i$  von 1 bis  $n/h$  do  $\mathcal{O}(n/h)$   
  └ Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(h \log h)$ 
```

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

```
for  $j$  von 1 bis  $h$  do  $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$ 
```

```
  for  $i$  von 1 bis  $n/h$  do
```

```
    └ Berechne Punkt  $q_i \in P_i$  der den Winkel  $\mathcal{O}(\log h)$   
       $p_{j-1}p_jq_i$  maximiert.
```

```
     $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$ 
```

```
return  $(p_1, \dots, p_h)$ 
```

$\mathcal{O}(n \log h)$
 $\mathcal{O}(n \log h)$

Chans Algorithmus

Angenommen, wir kennen h :

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

```
for  $i$  von 1 bis  $n/h$  do  $\mathcal{O}(n/h)$   
└ Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(h \log h)$ 
```

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

```
for  $j$  von 1 bis  $h$  do  $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$ 
```

```
└ for  $i$  von 1 bis  $n/h$  do
```

```
└└ Berechne Punkt  $q_i \in P_i$  der den Winkel  $\mathcal{O}(\log h)$   
└└  $p_{j-1}p_jq_i$  maximiert.
```

```
└  $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$ 
```

```
return  $(p_1, \dots, p_h)$ 
```

$\mathcal{O}(n \log h)$

$\mathcal{O}(n \log h)$
 $\mathcal{O}(n \log h)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P, h)

Unterteile P in Mengen P_i mit je h Knoten

```
for  $i$  von 1 bis  $n/h$  do  $\mathcal{O}(n/h)$   
└ Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(h \log h)$ 
```

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

```
for  $j$  von 1 bis  $h$  do  $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$ 
```

```
└ for  $i$  von 1 bis  $n/h$  do
```

```
└└ Berechne Punkt  $q_i \in P_i$  der den Winkel  $\mathcal{O}(\log h)$   
└└  $p_{j-1}p_jq_i$  maximiert.
```

```
└  $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$ 
```

```
return  $(p_1, \dots, p_h)$ 
```

$\mathcal{O}(n \log h)$
 $\mathcal{O}(n \log h)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull($P, \overset{m}{\times}$) ←

Unterteile P in Mengen P_i mit je h Knoten

```
for  $i$  von 1 bis  $n/h$  do  $\mathcal{O}(n/h)$   
  Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(h \log h)$ 
```

$p_0 = (\infty, \infty)$

$p_1 =$ rechtester Knoten in P

```
for  $j$  von 1 bis  $h$  do  $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$ 
```

```
  for  $i$  von 1 bis  $n/h$  do
```

```
    Berechne Punkt  $q_i \in P_i$  der den Winkel  $\mathcal{O}(\log h)$   
     $p_{j-1}p_jq_i$  maximiert.
```

```
   $p_{j+1} = \max\{q_1, \dots, q_{n/h}\}$ 
```

```
return  $(p_1, \dots, p_h)$ 
```

$\mathcal{O}(n \log h)$
 $\mathcal{O}(n \log h)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P, m)

Unterteile P in Mengen P_i mit je m Knoten

```
for  $i$  von 1 bis  $n/m$  do  $\mathcal{O}(n/m)$   
   $\lfloor$  Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(m \log m)$ 
```

$p_0 = (0, -\infty)$

$p_1 =$ linkester Knoten au P

$\mathcal{O}(m) \cdot \mathcal{O}(n/m) = \mathcal{O}(n)$

```
for  $j$  von 1 bis  $m$  do
```

```
  for  $i$  von 1 bis  $n/m$  do
```

```
     $\lfloor$  Berechne Punkt  $q_i \in P_i$  der den Winkel  $\mathcal{O}(\log m)$   
     $\lfloor$   $p_{j-1}p_jq_i$  maximiert.
```

```
   $p_{j+1} = \max\{q_1, \dots, q_{n/m}\}$ 
```

```
return  $(p_1, \dots, p_m)$   $\mathcal{O}(n \log m)$ 
```

$\mathcal{O}(n \log m)$
 $\mathcal{O}(n \log m)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P, m)

Unterteile P in Mengen P_i mit je m Knoten

```
for  $i$  von 1 bis  $n/m$  do  $\mathcal{O}(n/m)$   
└ Berechne per GrahamScan  $CH(P_i)$   $\mathcal{O}(m \log m)$ 
```

$p_0 = (0, -\infty)$

$p_1 =$ linkester Knoten au P

$\mathcal{O}(m) \cdot \mathcal{O}(n/m) = \mathcal{O}(n)$

for j von 1 bis m **do**

```
for  $i$  von 1 bis  $n/m$  do
```

```
└ Berechne Punkt  $q_i \in P_i$  der den Winkel  $\mathcal{O}(\log m)$   
└  $p_{j-1}p_jq_i$  maximiert.
```

```
 $p_{j+1} = \max\{q_1, \dots, q_{n/m}\}$ 
```

```
return  $(p_1, \dots, p_m)$   $\mathcal{O}(n \log m)$ 
```

$\mathcal{O}(n \log m)$
 $\mathcal{O}(n \log m)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P, m)

Unterteile P in Mengen P_i mit je m Knoten

...

$p_0 = (0, -\infty)$

$p_1 =$ linkerster Knoten auf P

for j von 1 bis m **do**

for i von 1 bis n/m **do**

 Berechne Punkt $q_i \in P_i$ der den Winkel $\angle p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log m)$

$p_{j+1} = \max\{q_1, \dots, q_{n/m}\}$

if $p_{j+1} == p_1$ **then**

return (p_1, \dots, p_j)

return failure

$$\mathcal{O}(m) \cdot \mathcal{O}(n/m) = \mathcal{O}(n)$$

$\mathcal{O}(n \log m)$
 $\mathcal{O}(n \log m)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P, m)

Unterteile P in Mengen P_i mit je m Knoten

...

$p_0 = (0, -\infty)$

$p_1 =$ linkerster Knoten au P

for j von 1 bis m **do**

for i von 1 bis n/m **do**

 Berechne Punkt $q_i \in P_i$ der den Winkel $\mathcal{O}(\log m)$

$p_{j-1}p_jq_i$ maximiert.

$p_{j+1} = \max\{q_1, \dots, q_{n/m}\}$

if $p_{j+1} == p_1$ **then**

return (p_1, \dots, p_j)

return failure

$\mathcal{O}(n \log m)$

$\mathcal{O}(n \log m)$
 $\mathcal{O}(n \log m)$
 $\mathcal{O}(n \log m)$

Was ist mit m ?

Vorschläge?

Was ist mit m ?

FullChanHull(P)

```
for  $t = 0, 1, 2, \dots$  do  
   $m = \min\{n, 2^{2^t}\}$   
  result = ChanHull( $P, m$ )  
  if result  $\neq$  failure then  
    break  
return result
```

Was ist mit m ?

FullChanHull(P)

```
for  $t = 0, 1, 2, \dots$  do
```

```
   $m = \min\{n, 2^{2^t}\}$ 
```

```
  result = ChanHull( $P, m$ )
```

```
  if result  $\neq$  failure then
```

```
    break
```

```
return result
```

$$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$$

Laufzeit:

Was ist mit m ?

FullChanHull(P)

for $t = 0, 1, 2, \dots$ **do**

$m = \min\{n, 2^{2^t}\}$

result = ChanHull(P, m) $\mathcal{O}(n \log m) =$

if result \neq failure **then**

$\mathcal{O}(n \log 2^{2^t})$

break

return result

Laufzeit:

$$\mathcal{O}(\log \log h) \sum_{t=0} \mathcal{O}(n \log 2^{2^t})$$

Was ist mit m ?

FullChanHull(P)

```
for  $t = 0, 1, 2, \dots$  do
```

```
   $m = \min\{n, 2^{2^t}\}$ 
```

```
  result = ChanHull( $P, m$ )
```

```
  if result  $\neq$  failure then
```

```
    break
```

```
return result
```

$$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$$

Laufzeit:

$$\mathcal{O}(\log \log h) \sum_{t=0} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(2^t)$$

Was ist mit m ?

FullChanHull(P)

```
for  $t = 0, 1, 2, \dots$  do
```

```
   $m = \min\{n, 2^{2^t}\}$ 
```

```
  result = ChanHull( $P, m$ )
```

```
  if result  $\neq$  failure then
```

```
    break
```

```
return result
```

$\mathcal{O}(n \log m) =$

$\mathcal{O}(n \log 2^{2^t})$

Laufzeit:

$$\mathcal{O}(\log \log h) \sum_{t=0} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(2^t)$$

$$\leq \mathcal{O}(n) \cdot \mathcal{O}(2^{\log \log h + 1})$$

Was ist mit m ?

FullChanHull(P)

```
for  $t = 0, 1, 2, \dots$  do
```

```
   $m = \min\{n, 2^{2^t}\}$ 
```

```
  result = ChanHull( $P, m$ )
```

```
  if result  $\neq$  failure then
```

```
    break
```

```
return result
```

$$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(2^t)$$

$$\leq \mathcal{O}(n) \cdot \mathcal{O}(2^{\log \log h + 1}) = \mathcal{O}(n) \cdot \mathcal{O}(2^{\log \log h})$$

Was ist mit m ?

FullChanHull(P)

```
for  $t = 0, 1, 2, \dots$  do
```

```
   $m = \min\{n, 2^{2^t}\}$ 
```

```
  result = ChanHull( $P, m$ )
```

```
  if result  $\neq$  failure then
```

```
    break
```

```
return result
```

$$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(2^t)$$

$$\leq \mathcal{O}(n) \cdot \mathcal{O}(2^{\log \log h + 1}) = \mathcal{O}(n) \cdot \mathcal{O}(2^{\log \log h})$$

$$= \mathcal{O}(n) \cdot \mathcal{O}(\log h)$$

Was ist mit m ?

FullChanHull(P)

```
for  $t = 0, 1, 2, \dots$  do
```

```
   $m = \min\{n, 2^{2^t}\}$ 
```

```
  result = ChanHull( $P, m$ )
```

```
  if result  $\neq$  failure then
```

```
    break
```

```
return result
```

$$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(2^t)$$

$$\leq \mathcal{O}(n) \cdot \mathcal{O}(2^{\log \log h + 1}) = \mathcal{O}(n) \cdot \mathcal{O}(2^{\log \log h})$$

$$= \mathcal{O}(n) \cdot \mathcal{O}(\log h) = \mathcal{O}(n \log h)$$

Das war's!

Achtung!

Vorlesung:

Donnerstag, 10.05, 9:45 Uhr

Polygontriangulierung

Raum 131, Gebäude 50.34

Nächster Übung:

Donnerstag, 10.05, 9:45 Uhr

Segmentschnitte & Polygontriangulierung

Raum 131, Gebäude 50.34