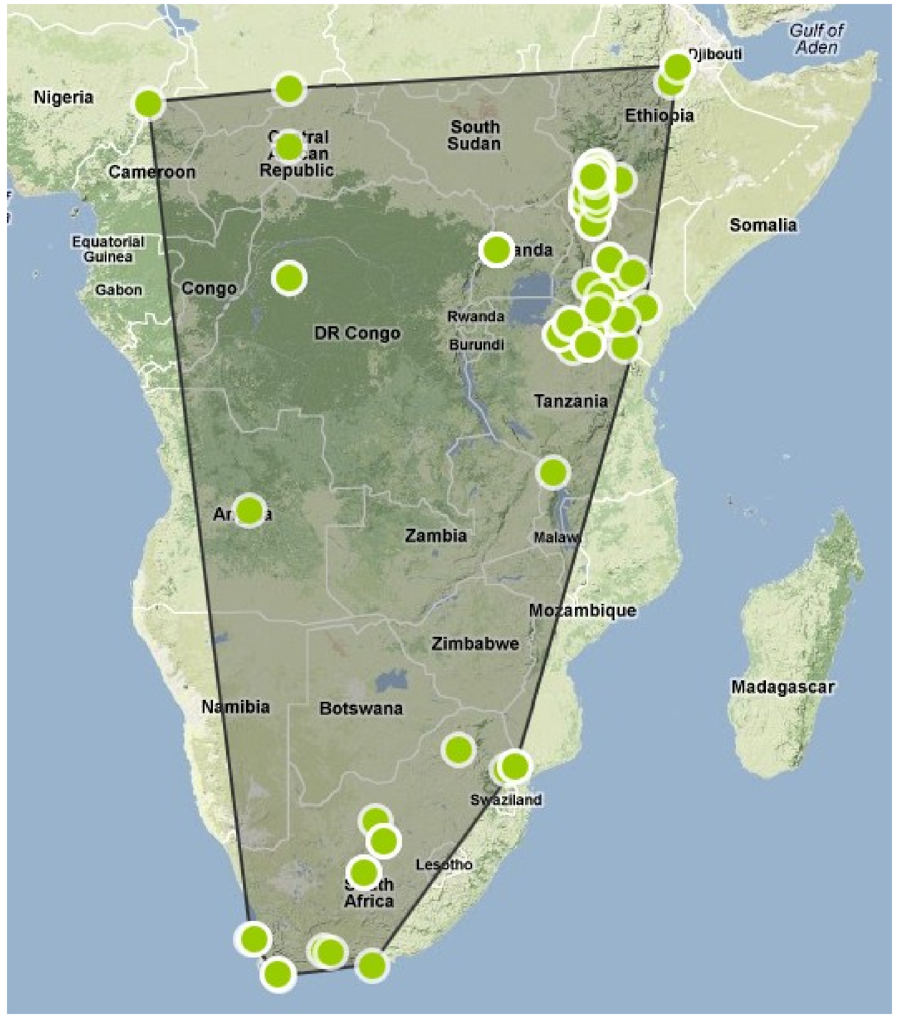
Convex Hulls

Geometric Algorithms

Wildlife analysis: extent of occurrence



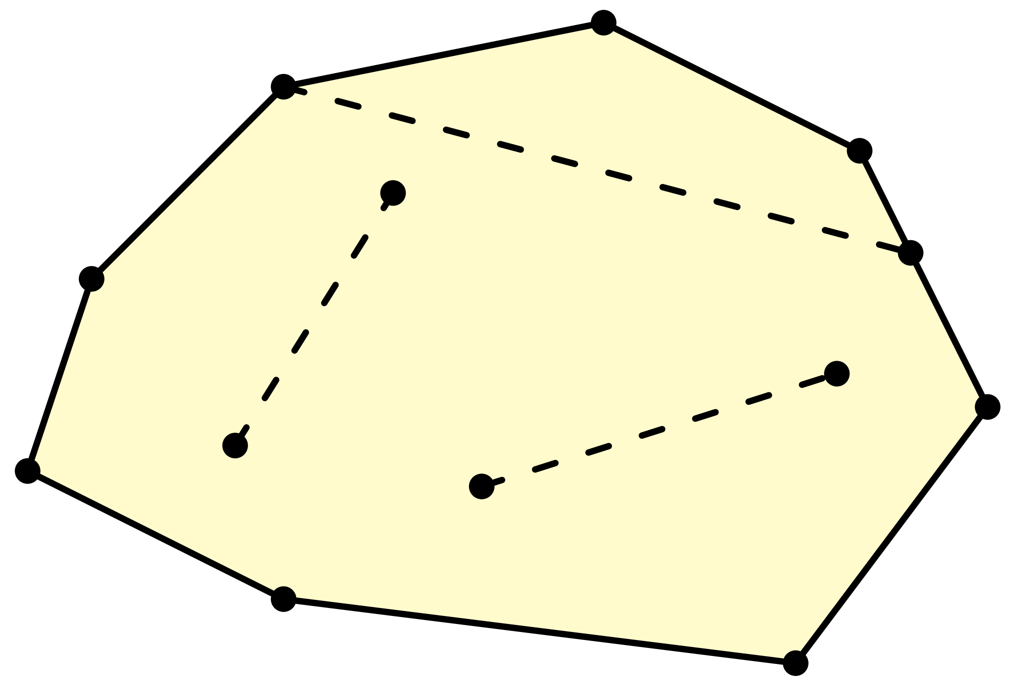
Black Rhino (Diceros bicornis)

http://geocat.kew.org/

http://www.iucnredlist.org/

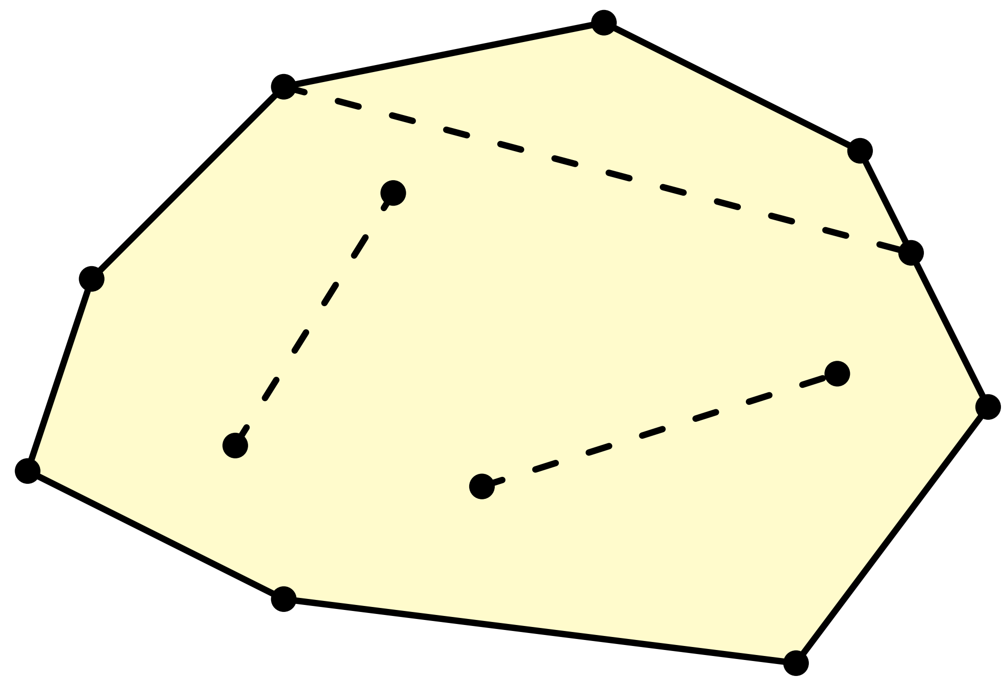
Convexity

Definition: A shape or set is convex if for any two points that are part of the shape, the whole connecting line segment is also part of the shape



Convexity

Definition: A shape or set is convex if for any two points that are part of the shape, the whole connecting line segment is also part of the shape

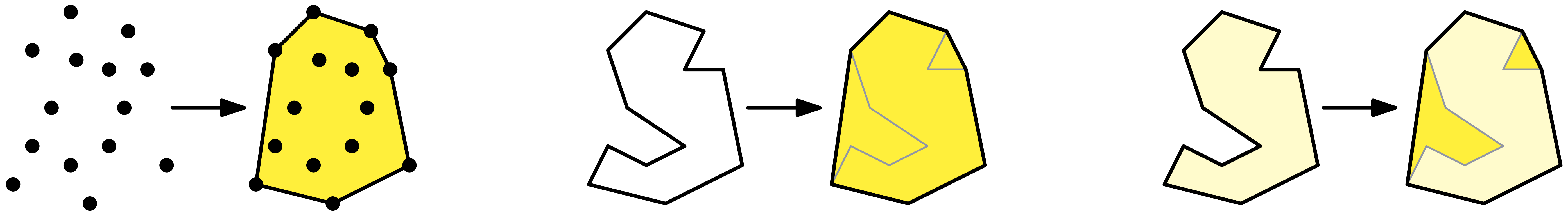


Question: Which of the following shapes are convex?

- point
- line segment
- line
- circle
- disk
- quadrant

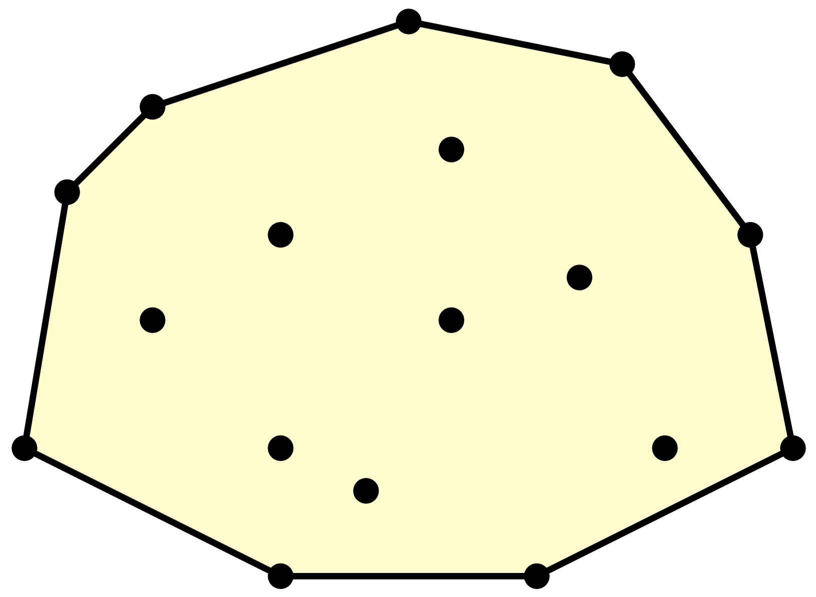
Convex hull

Definition: For any subset of the plane (set of points, polygonal chain, simple polygon), its convex hull is the smallest convex set that contains that subset



Convex hull problem

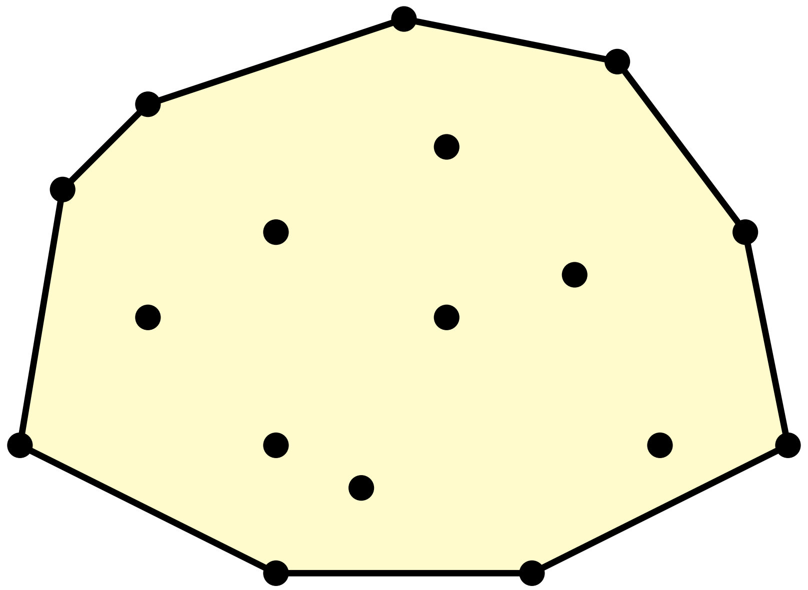
Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently



Convex hull problem

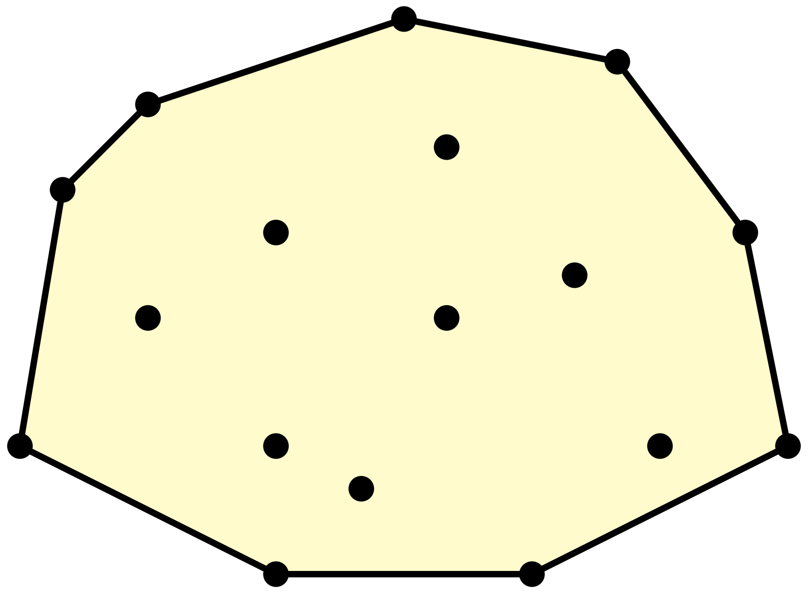
Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently

- input has $2n$ coordinates, so $O(n)$ size



Convex hull problem

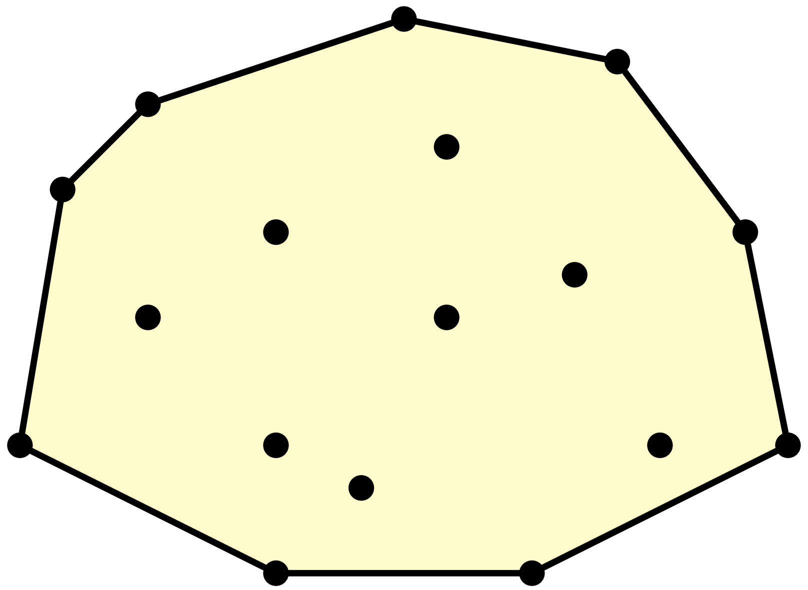
Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently



- input has $2n$ coordinates, so $O(n)$ size
- output?

Convex hull problem

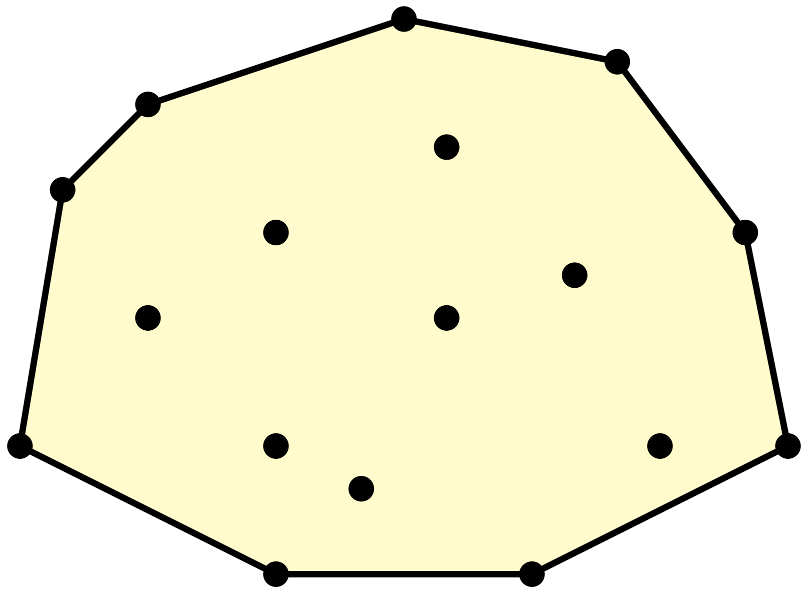
Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently



- input has $2n$ coordinates, so $O(n)$ size
- output?
  - assume the n points are distinct

Convex hull problem

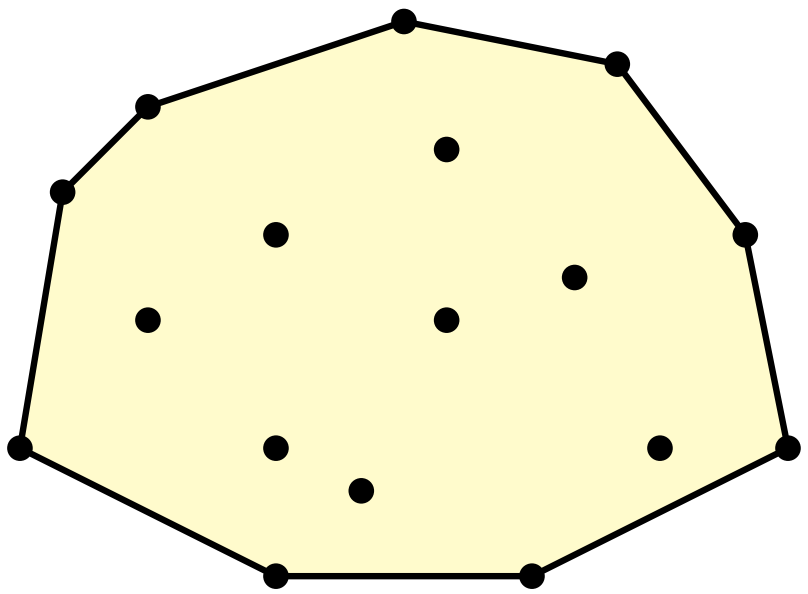
Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently



- input has $2n$ coordinates, so $O(n)$ size
- output?
  - assume the n points are distinct
  - output has at least 2 and at most n points

Convex hull problem

Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently

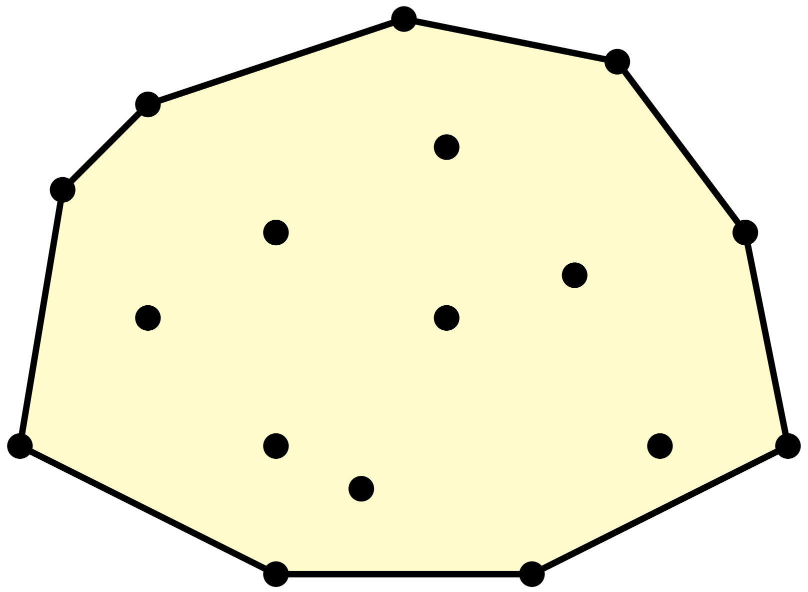


- input has $2n$ coordinates, so $O(n)$ size
- output?
  - assume the n points are distinct
  - output has at least 2 and at most n points
  - output size is between $O(1)$ and $O(n)$

Convex hull problem: questions

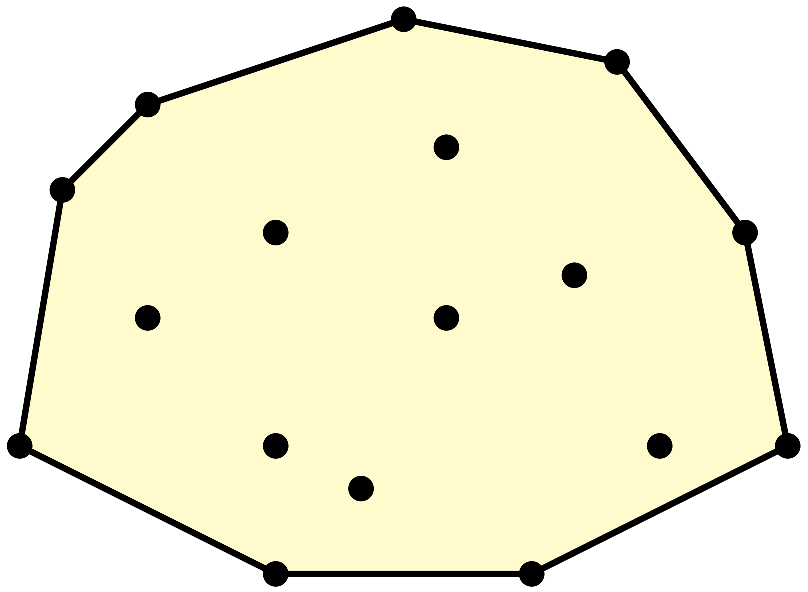
Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently

Question: Is there an algorithm computing a convex hull faster than $O(n)$ time in the worst case?



Convex hull problem: questions

Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently

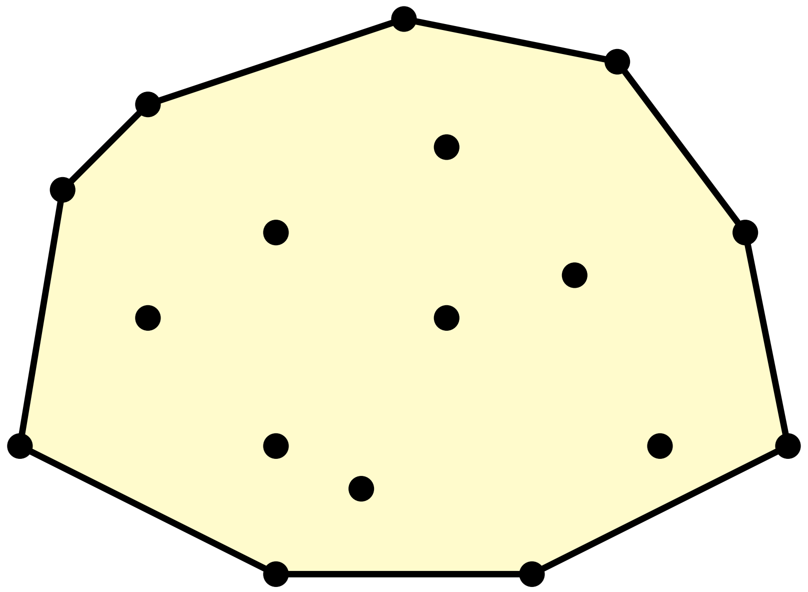


Question: Is there an algorithm computing a convex hull faster than $O(n)$ time in the worst case?

- the output is a convex polygon so it should be returned as a sorted sequence of points, counter-clockwise along the boundary

Convex hull problem: questions

Problem: Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently



Question: Is there an algorithm computing a convex hull faster than $O(n)$ time in the worst case?

- the output is a convex polygon so it should be returned as a sorted sequence of points, counter-clockwise along the boundary

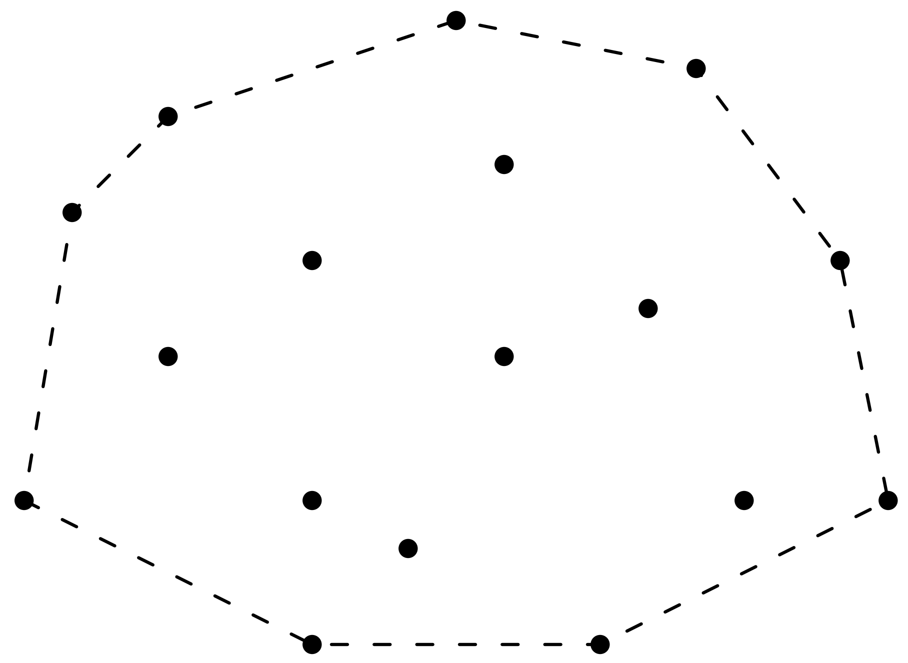
Question: Is there any hope of finding an $O(n)$ time algorithm?

Developing an algorithm

To develop an algorithm, draw many sketches to gain insight, make various observations, find useful properties

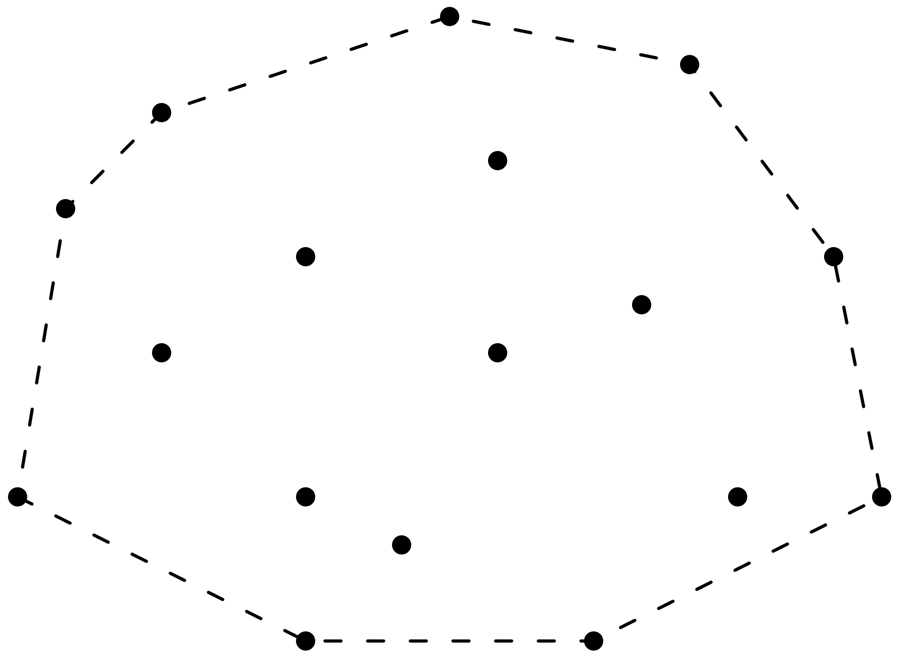
Convex hull algorithm

To develop an algorithm, draw many sketches to gain insight, make various observations, find useful properties



Convex hull algorithm

To develop an algorithm, draw many sketches to gain insight, make various observations, find useful properties



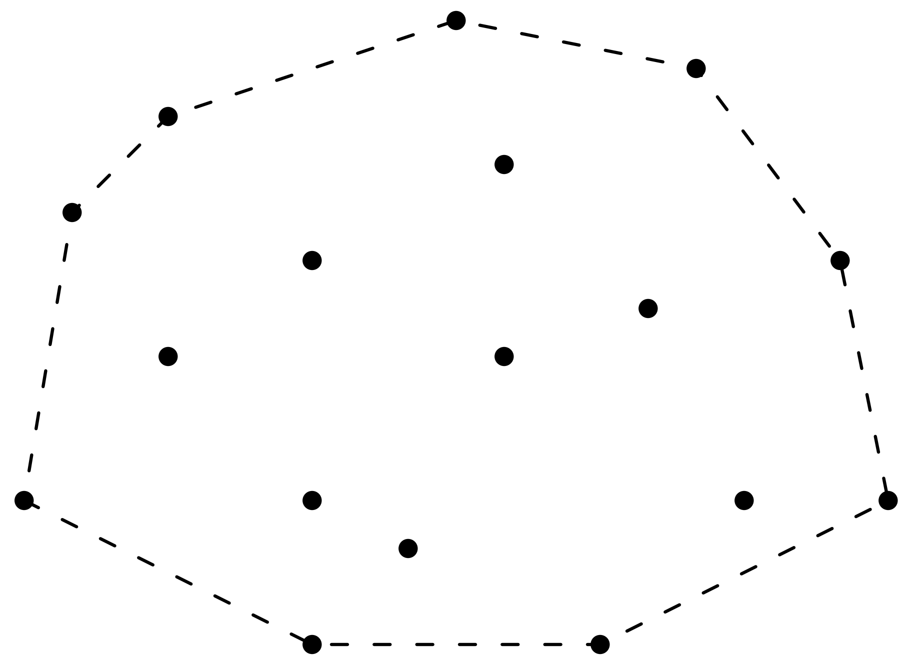
Property: The vertices of the convex hull are always points from the input

Convex hull algorithm

To develop an algorithm, draw many sketches to gain insight, make various observations, find useful properties

Property: The vertices of the convex hull are always points from the input

Observation: The edges of the convex hull connect two points of the input

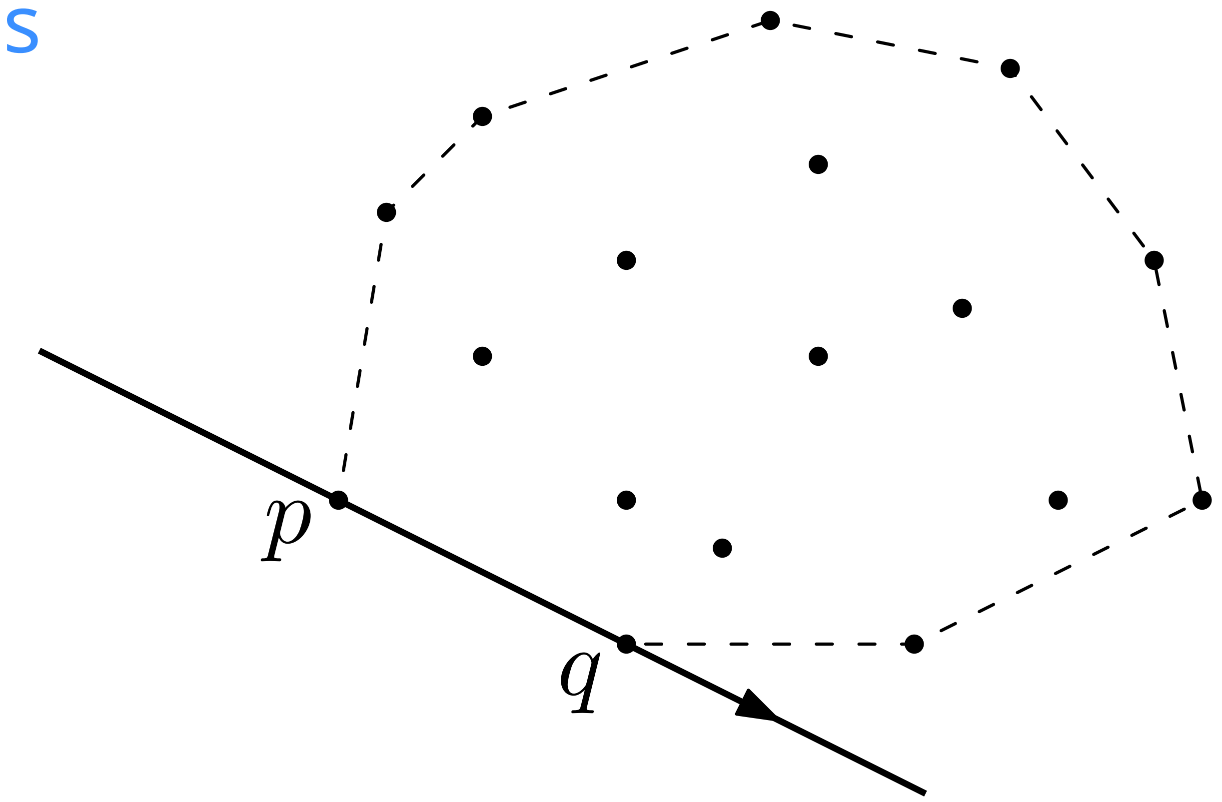


Convex hull algorithm

To develop an algorithm, draw many sketches to gain insight, make various observations, find useful properties

Property: The vertices of the convex hull are always points from the input

Observation: The edges of the convex hull connect two points of the input

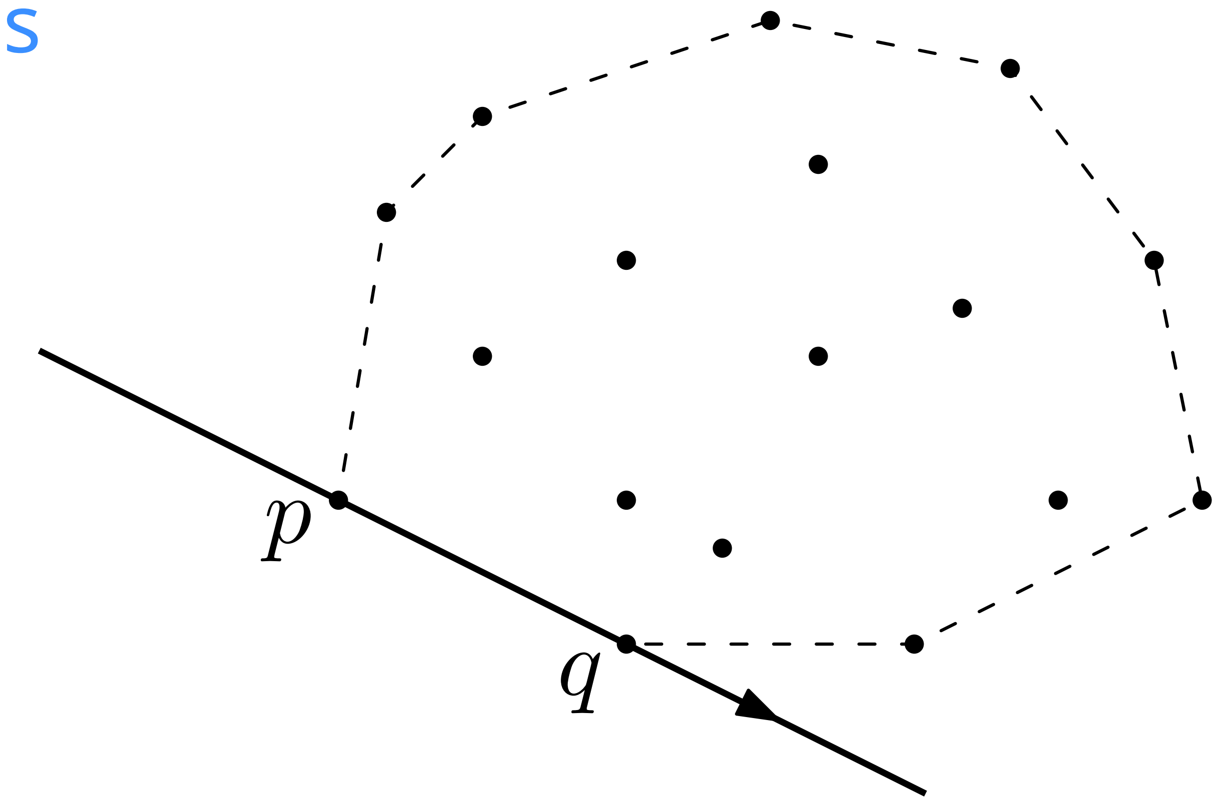


Convex hull algorithm

To develop an algorithm, draw many sketches to gain insight, make various observations, find useful properties

Property: The vertices of the convex hull are always points from the input

Observation: The edges of the convex hull connect two points of the input



all points lie left of the directed line from p to q (if the edge from p to q is a CCW convex hull edge)

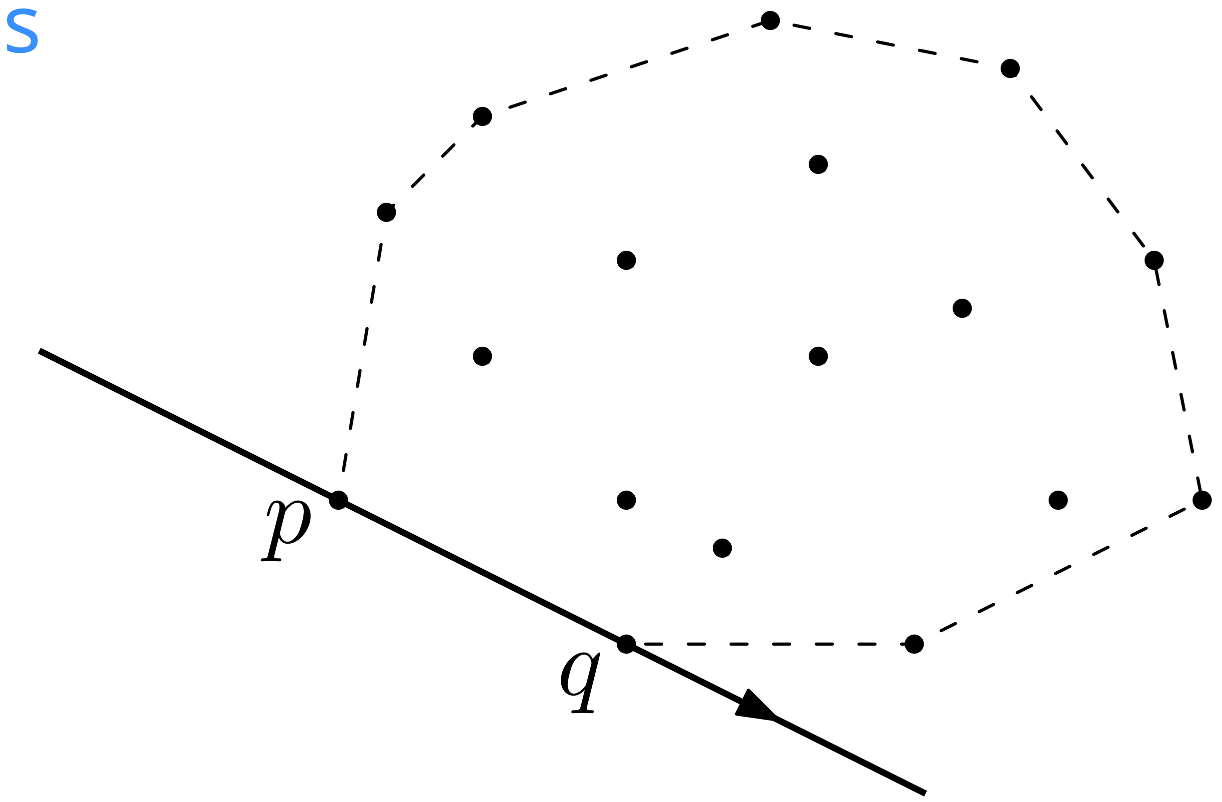
Convex hull algorithm

To develop an algorithm, draw many sketches to gain insight, make various observations, find useful properties

Property: The vertices of the convex hull are always points from the input

Observation: The edges of the convex hull connect two points of the input

Property: The supporting line of any convex hull edge has all input points to one side



all points lie left of the directed line from p to q (if the edge from p to q is a CCW convex hull edge)

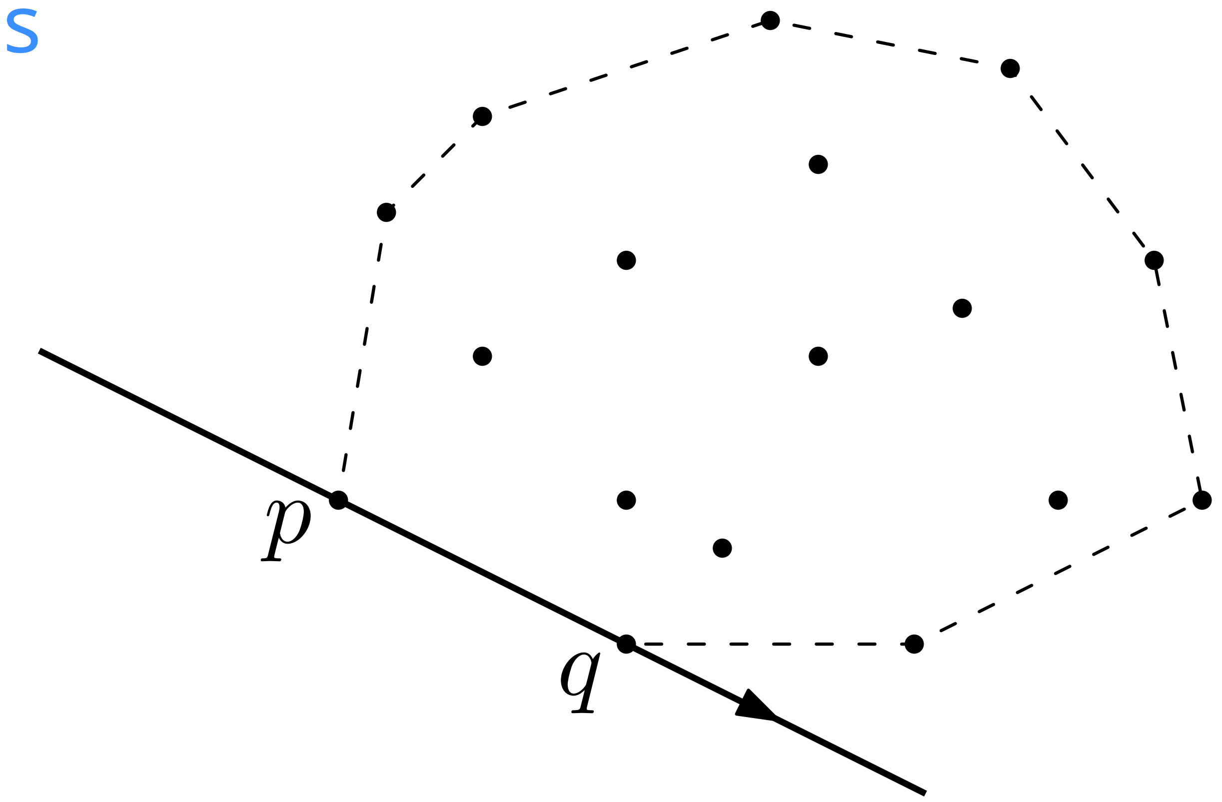
Convex hull algorithm

To develop an algorithm, draw many sketches to gain insight, make various observations, find useful properties

Property: The vertices of the convex hull are always points from the input

Observation: The edges of the convex hull connect two points of the input

Property: The supporting line of any convex hull edge has all input points to one side



all points lie left of the directed line from p to q (if the edge from p to q is a CCW convex hull edge)

Simple algorithm?

Convex hull algorithm

Algorithm SlowConvexHull(P)

Input: set P of distinct points in the plane

Output: list L with vertices of $CH(P)$ in counter-clockwise order

$E \leftarrow \varnothing$

for all ordered pairs $(p, q) \in P \times P$ with $p \neq q$ **do**

    $valid \leftarrow true$

    for all points $r \in P$ not equal to p or q **do**

        if r lies right of the directed line from p to q **then**

            $valid \leftarrow false$

    if $valid$ **then**

        add directed edge $\overrightarrow{pq}$ to E

from the set E of edges construct a list L of vertices of $CH(P)$, sorted in counter-clockwise order

Questions to keep in mind

Question: How must line 5 (if r lies right of $\overline{pq} \ldots$) be interpreted to make the algorithm correct, i.e., how do we handle degeneracies?

Question: How efficient is the algorithm?

Question: Is the algorithm robust against rounding errors?

Note: Robustness is a huge issue when implementing geometric algorithms

Convex hull algorithm 2

Another approach: incremental, from left to right

Convex hull algorithm 2

Another approach: incremental, from left to right

Let's first compute the upper boundary of the convex hull this way (property: on the upper hull, points appear in x-order)

Convex hull algorithm 2

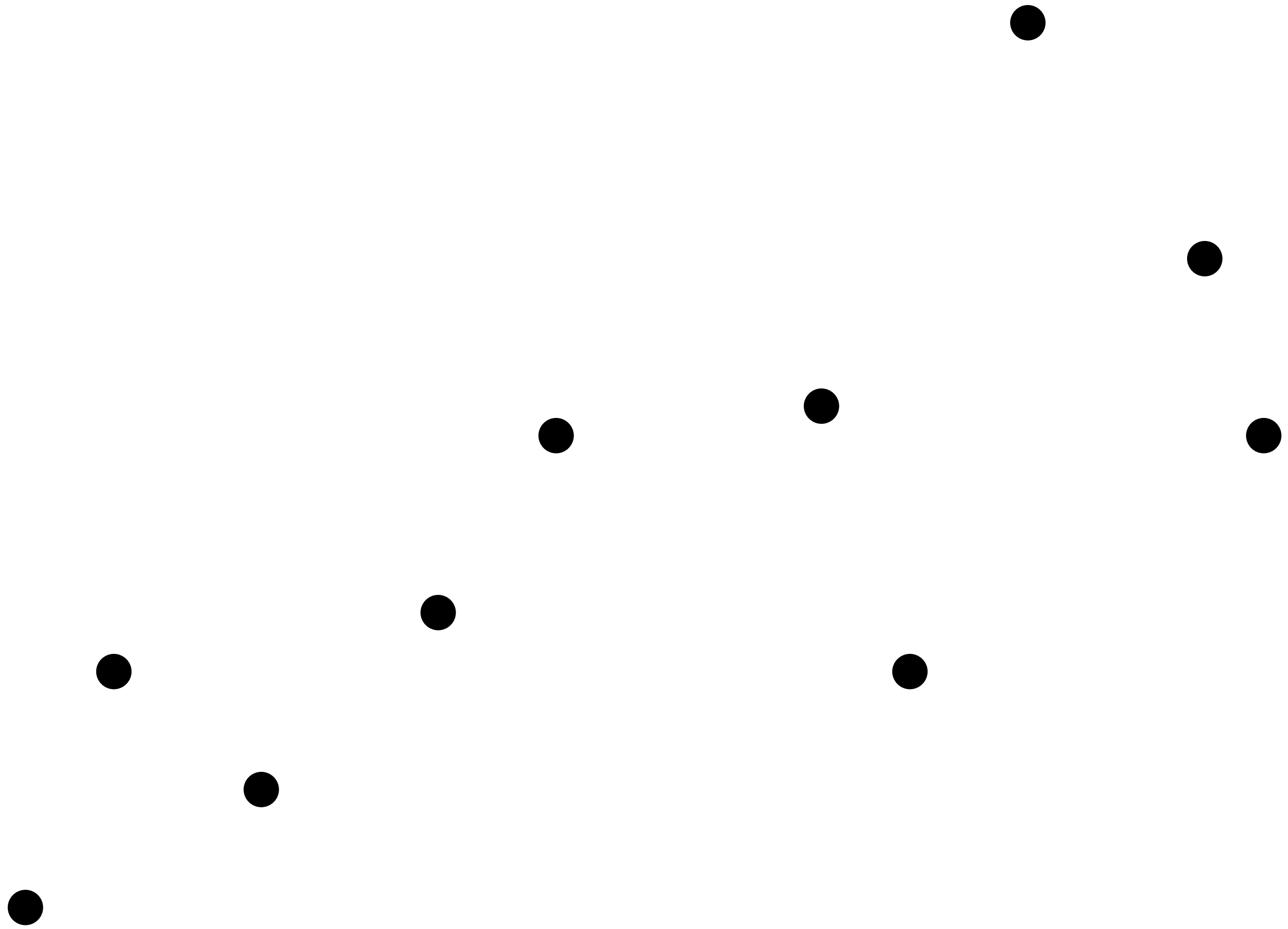
Another approach: incremental, from left to right

Let's first compute the upper boundary of the convex hull this way (property: on the upper hull, points appear in x-order)

Main idea: Sort the points from left to right (= by x-coordinate). Then insert the points in this order, and maintain the upper hull so far

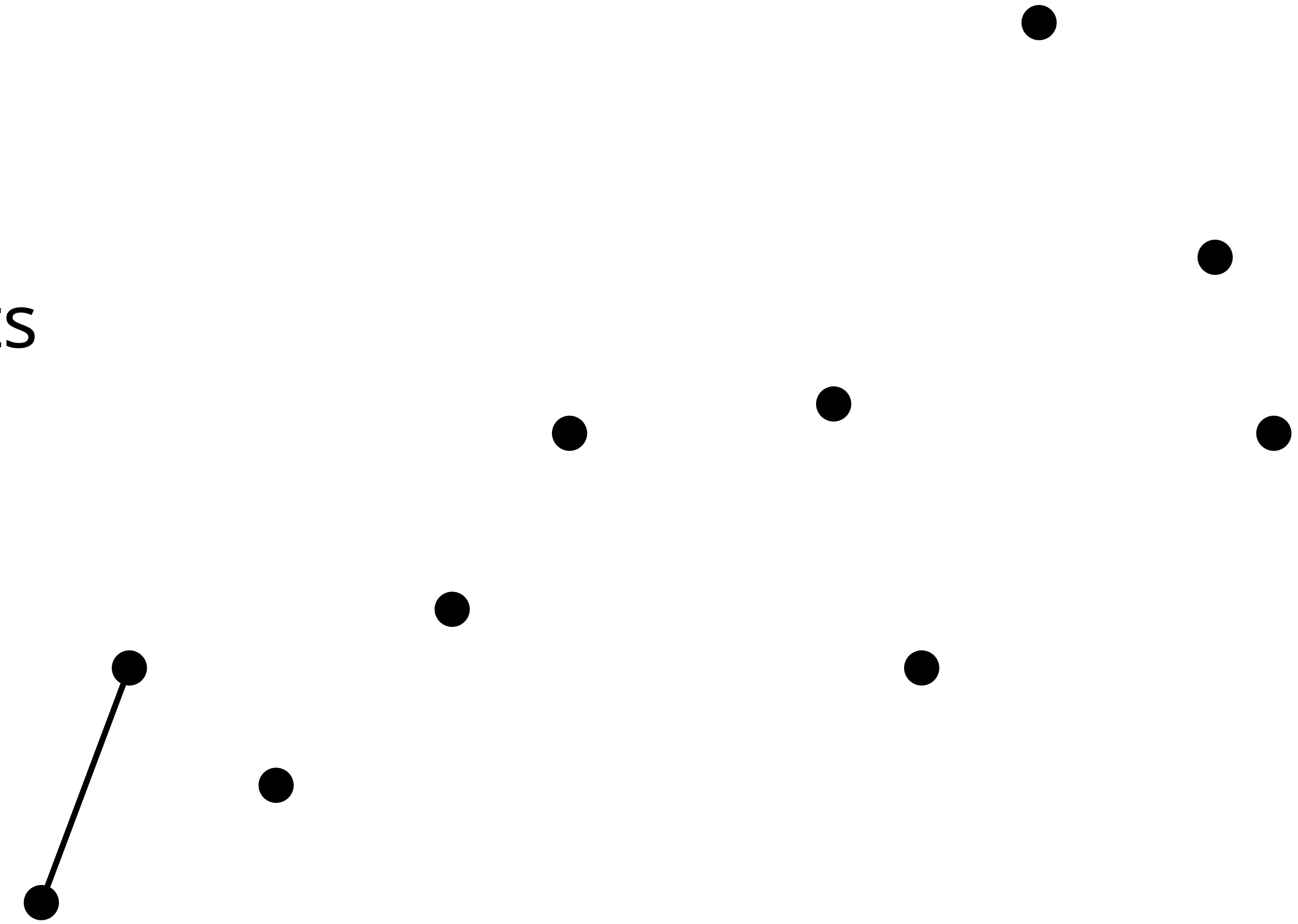
Convex hull algorithm 2

Observation: from left to right, there are only right turns on the upper hull



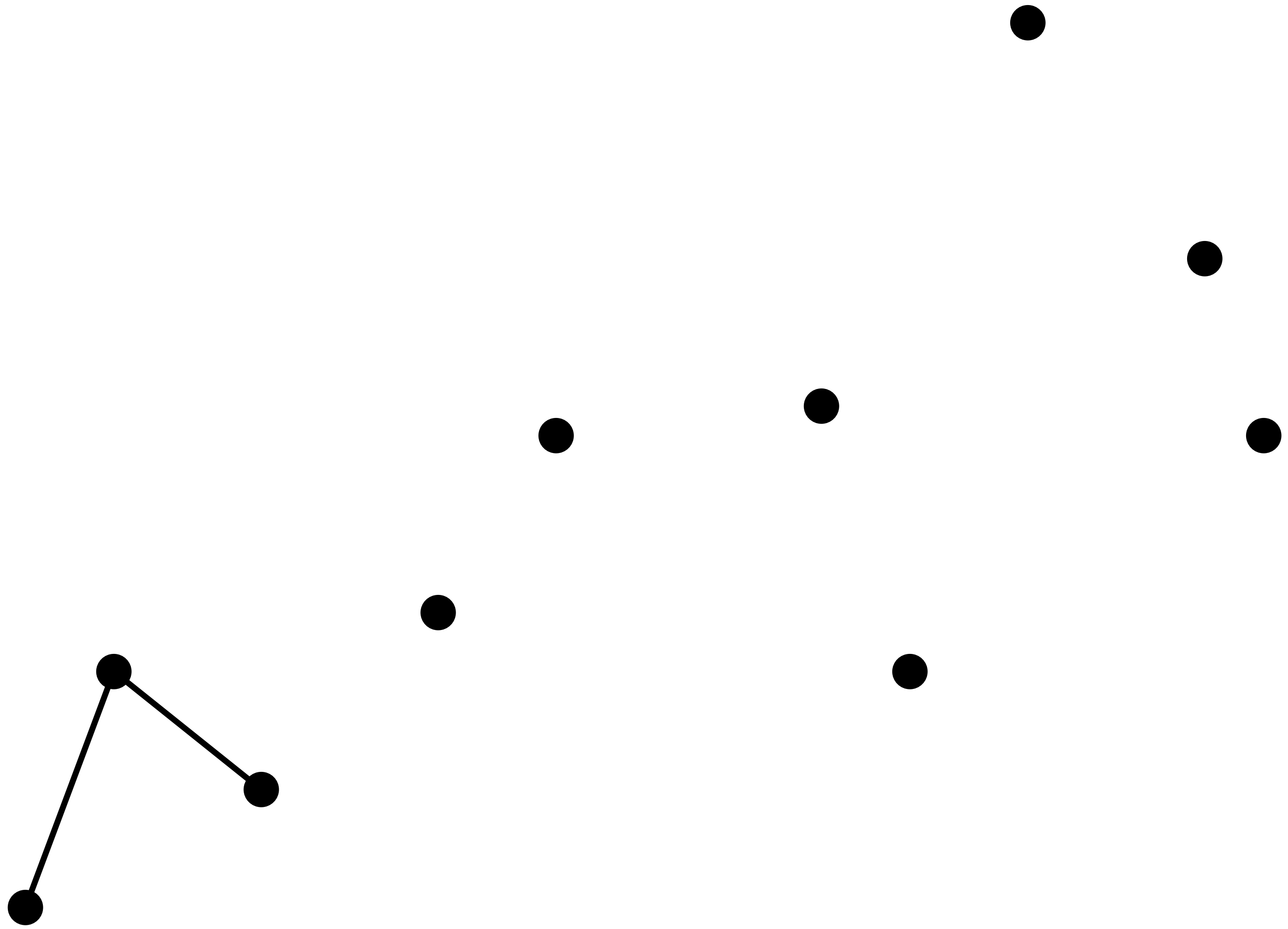
Convex hull algorithm 2

Initialize by inserting the leftmost two points



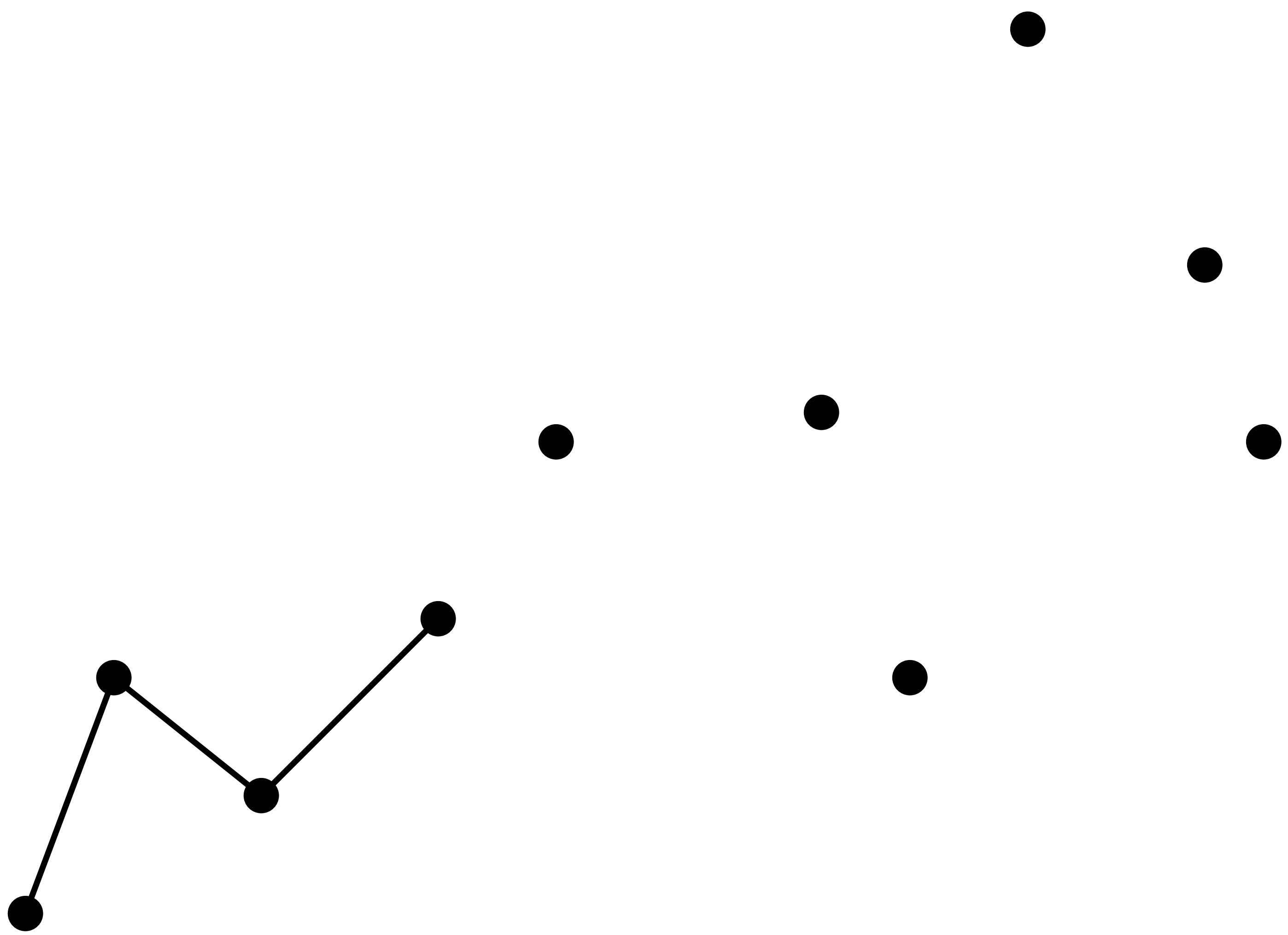
Convex hull algorithm 2

When we consider the third point there will be a right turn at the previous point, so we add it



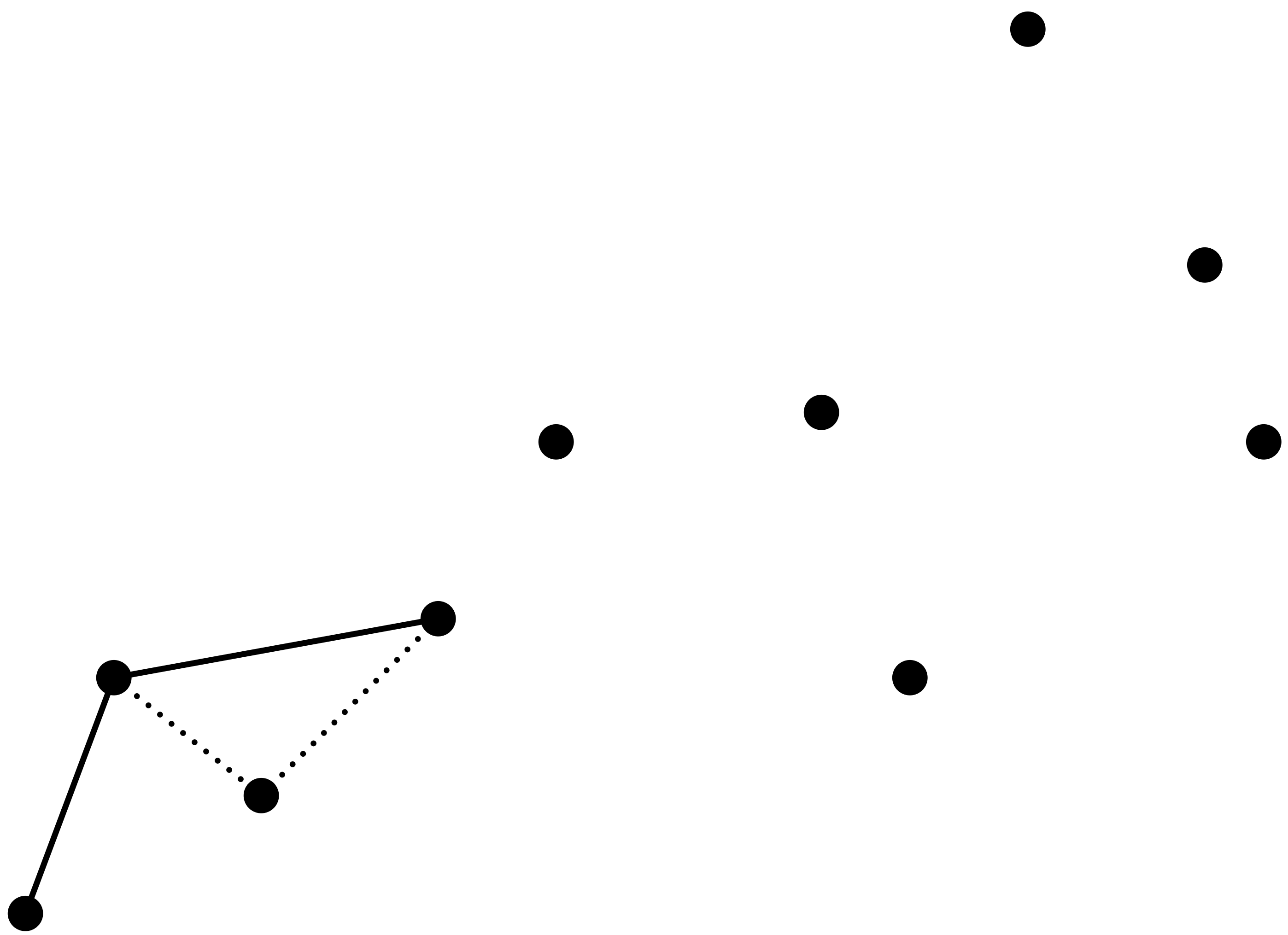
Convex hull algorithm 2

When we consider the fourth point we get a left turn at the third point



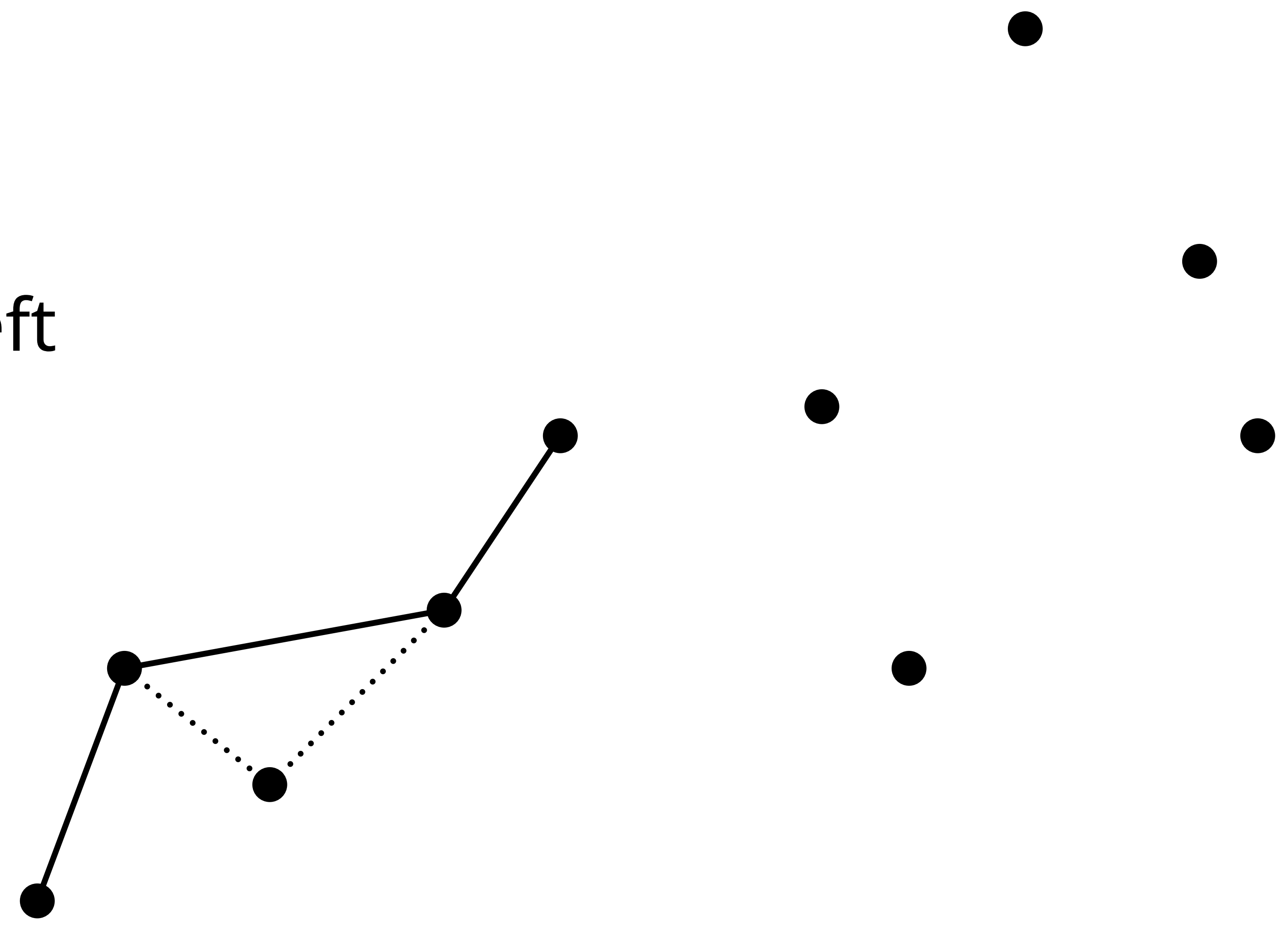
Convex hull algorithm 2

. . . so we remove the third point from the upper hull when we add the fourth



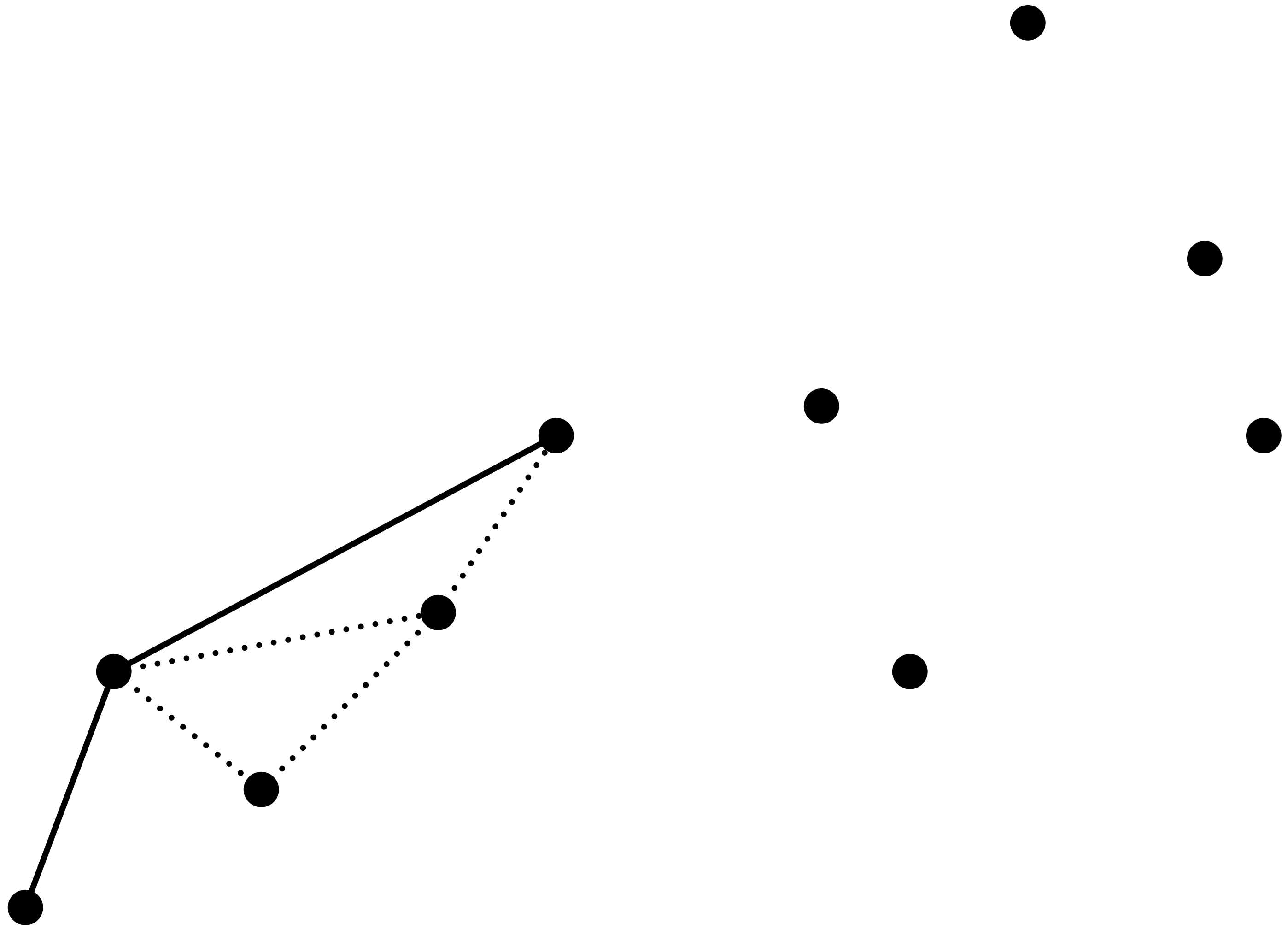
Convex hull algorithm 2

When we consider the fifth point we get a left turn at the fourth point



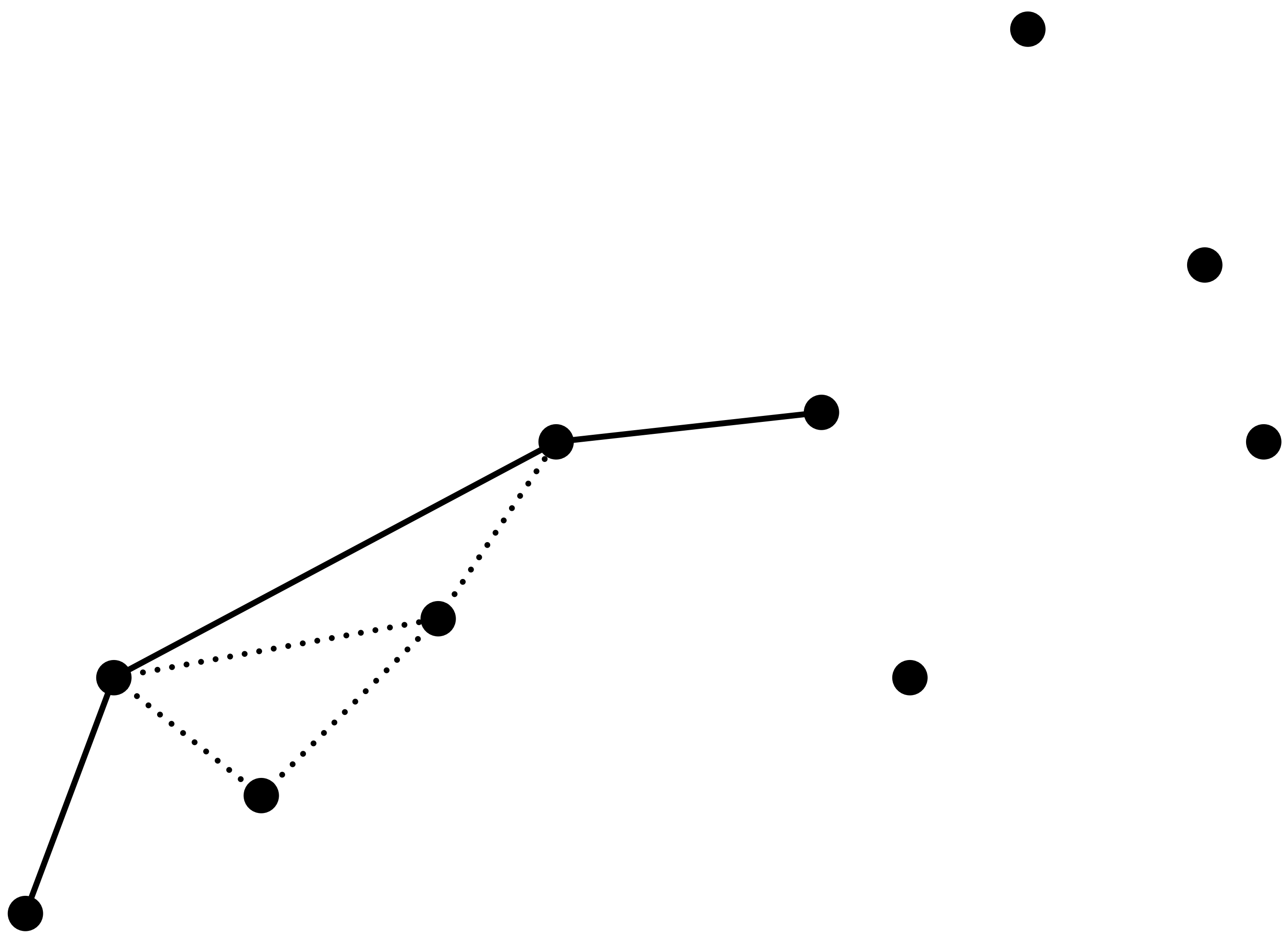
Convex hull algorithm 2

. . . so we remove the fourth point when we add the fifth



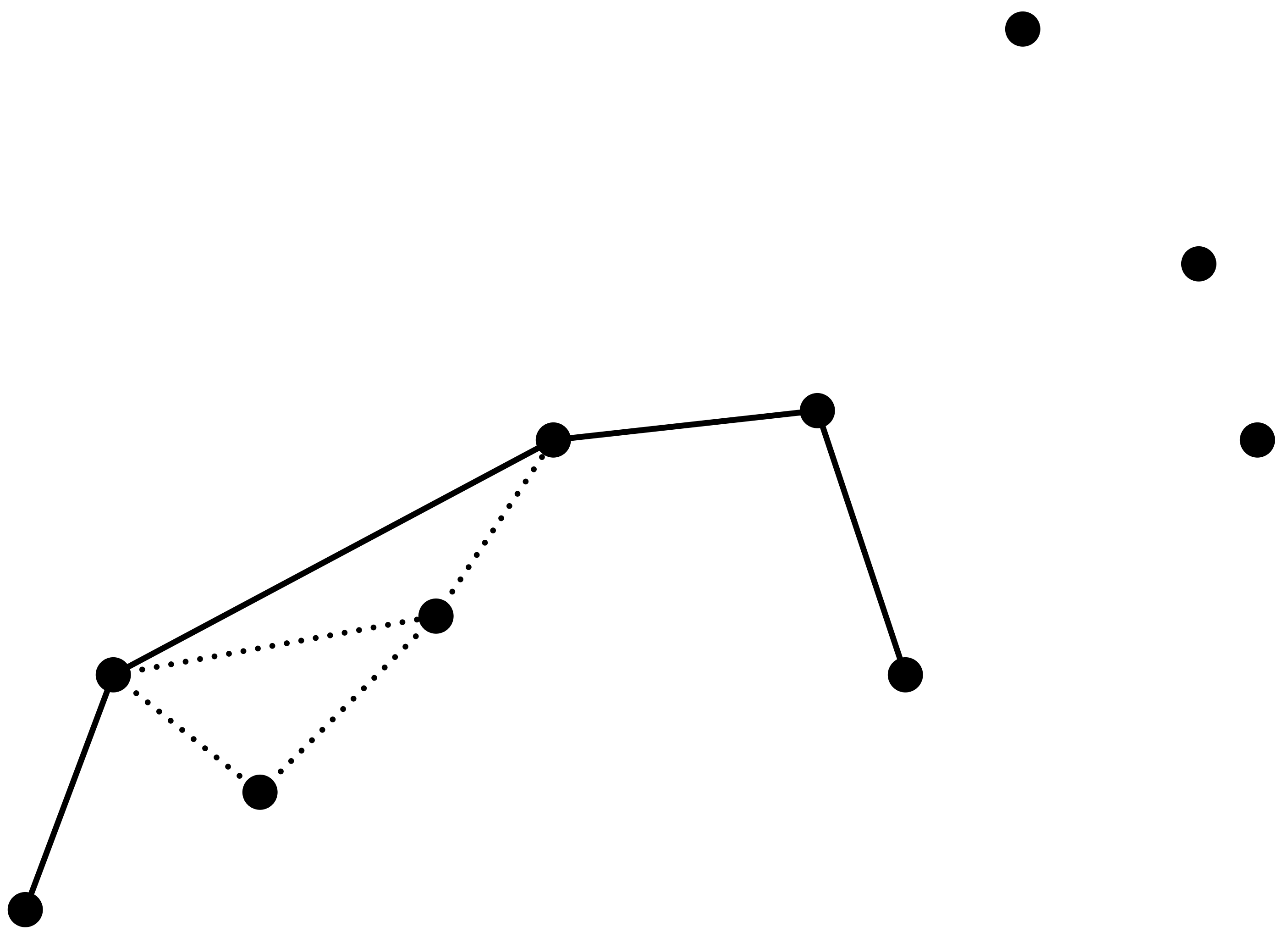
Convex hull algorithm 2

When we consider the sixth point we get a right turn at the fifth point, so we just add it



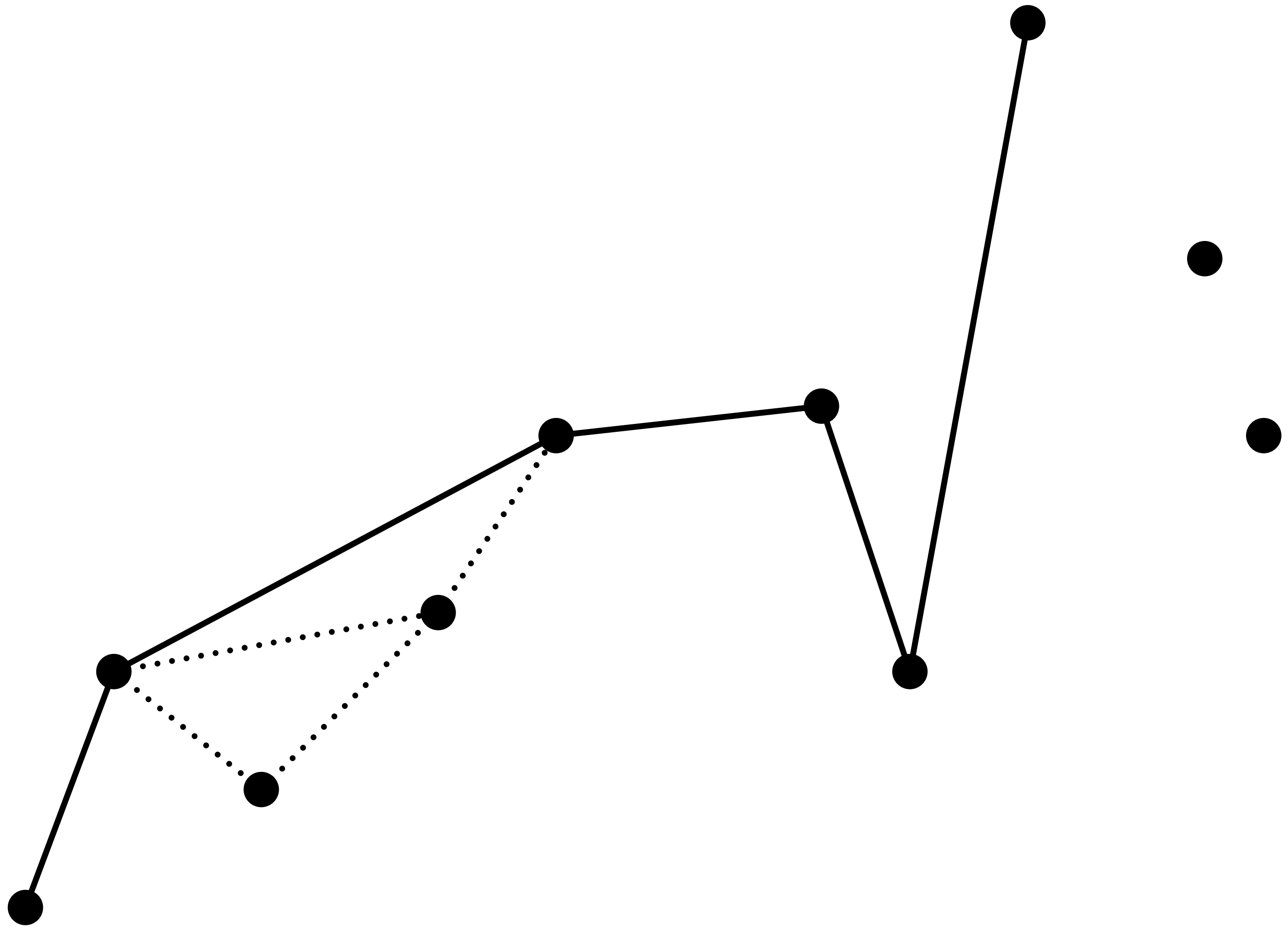
Convex hull algorithm 2

We also just add the seventh point



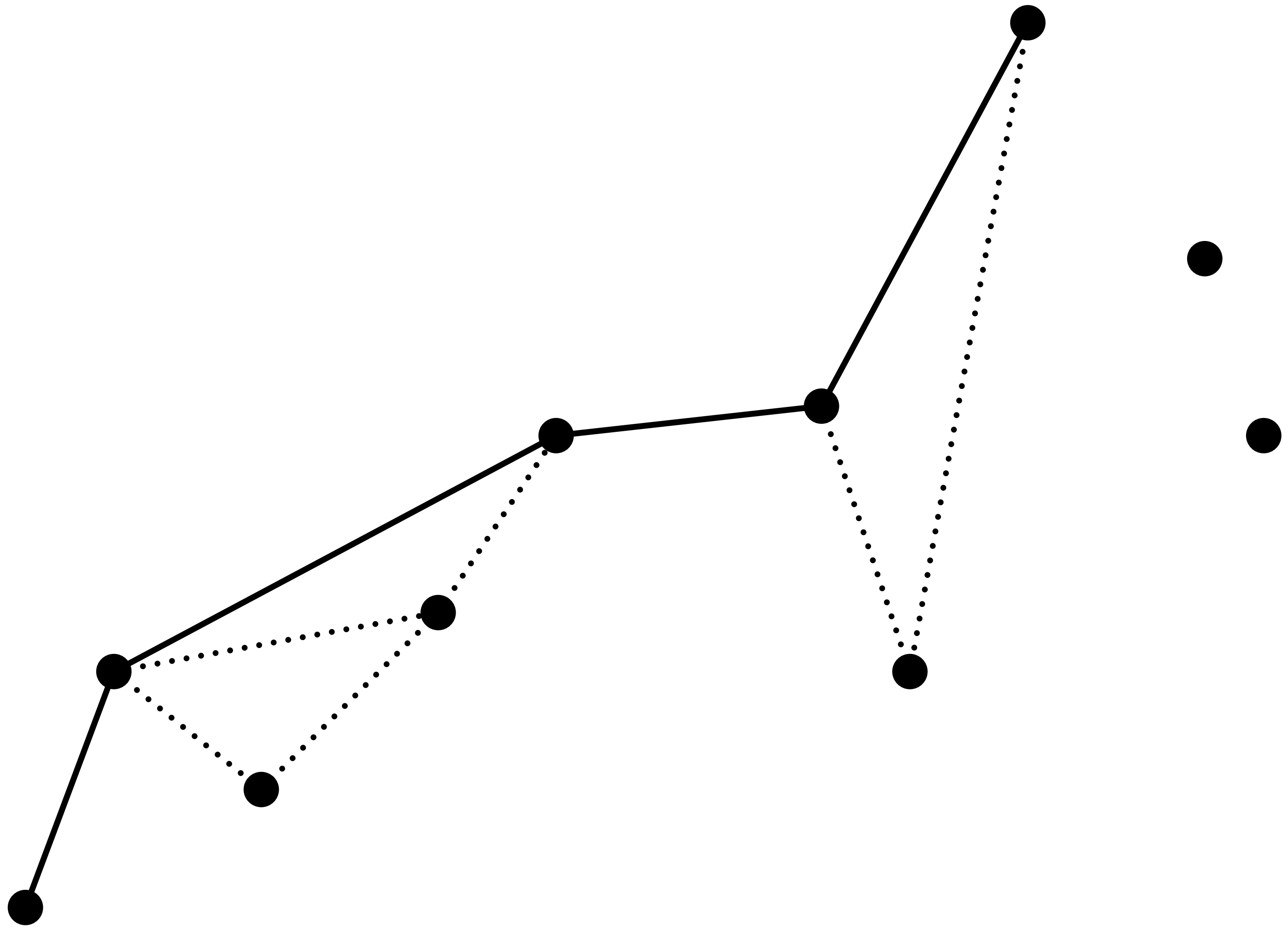
Convex hull algorithm 2

When considering the eighth point. . .



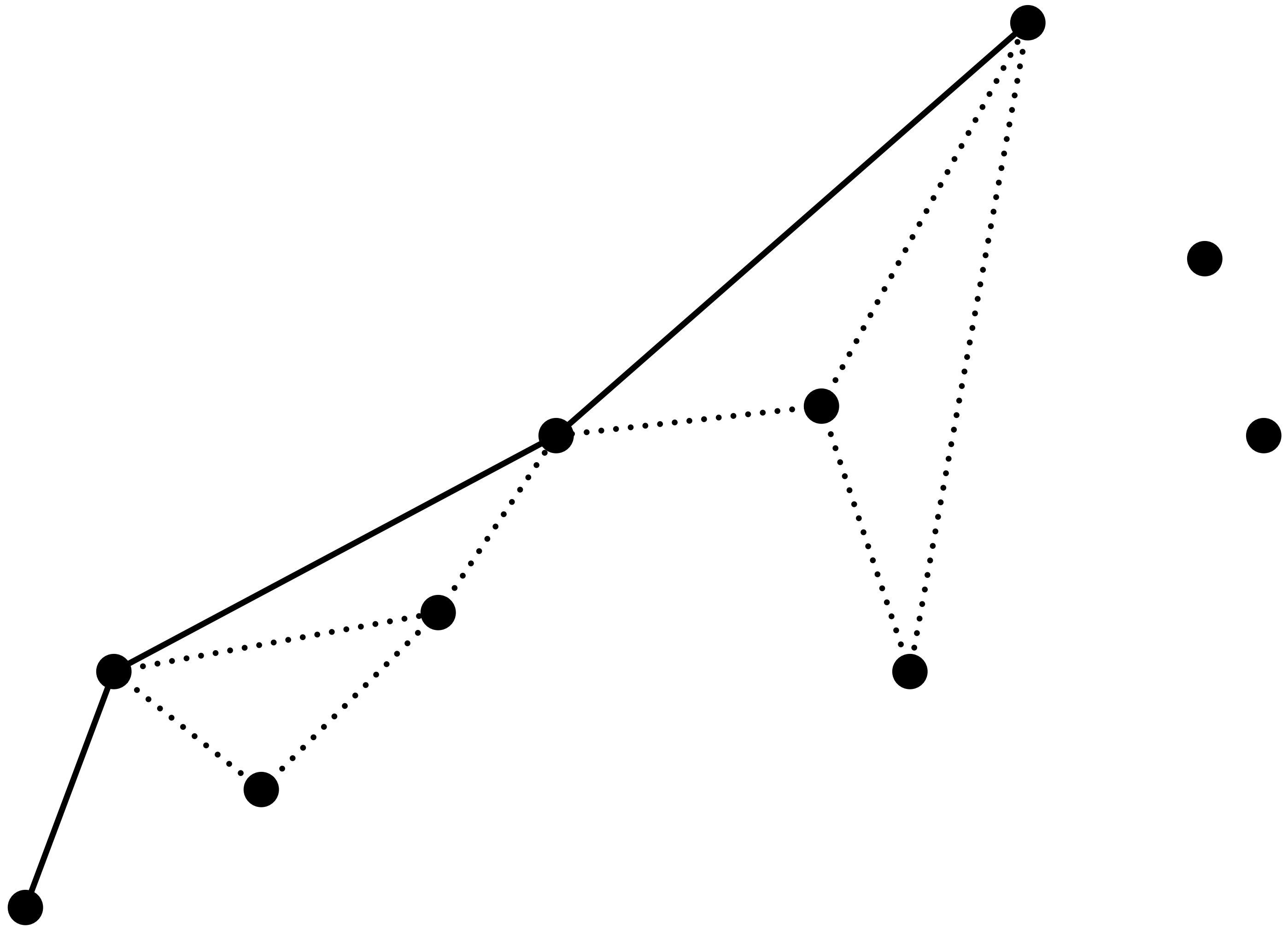
Convex hull algorithm 2

. . . we remove the seventh point



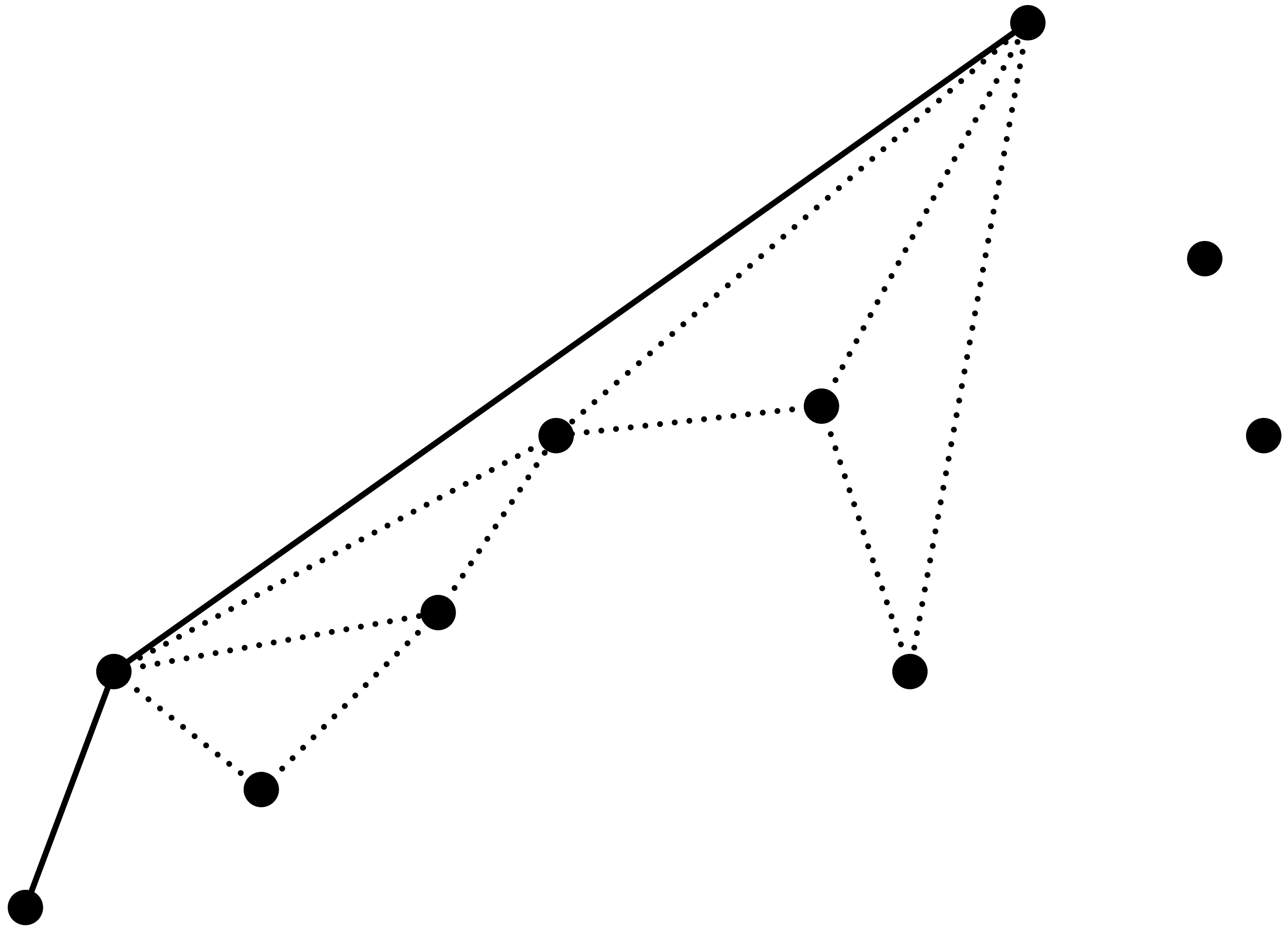
Convex hull algorithm 2

. . . and also the sixth point



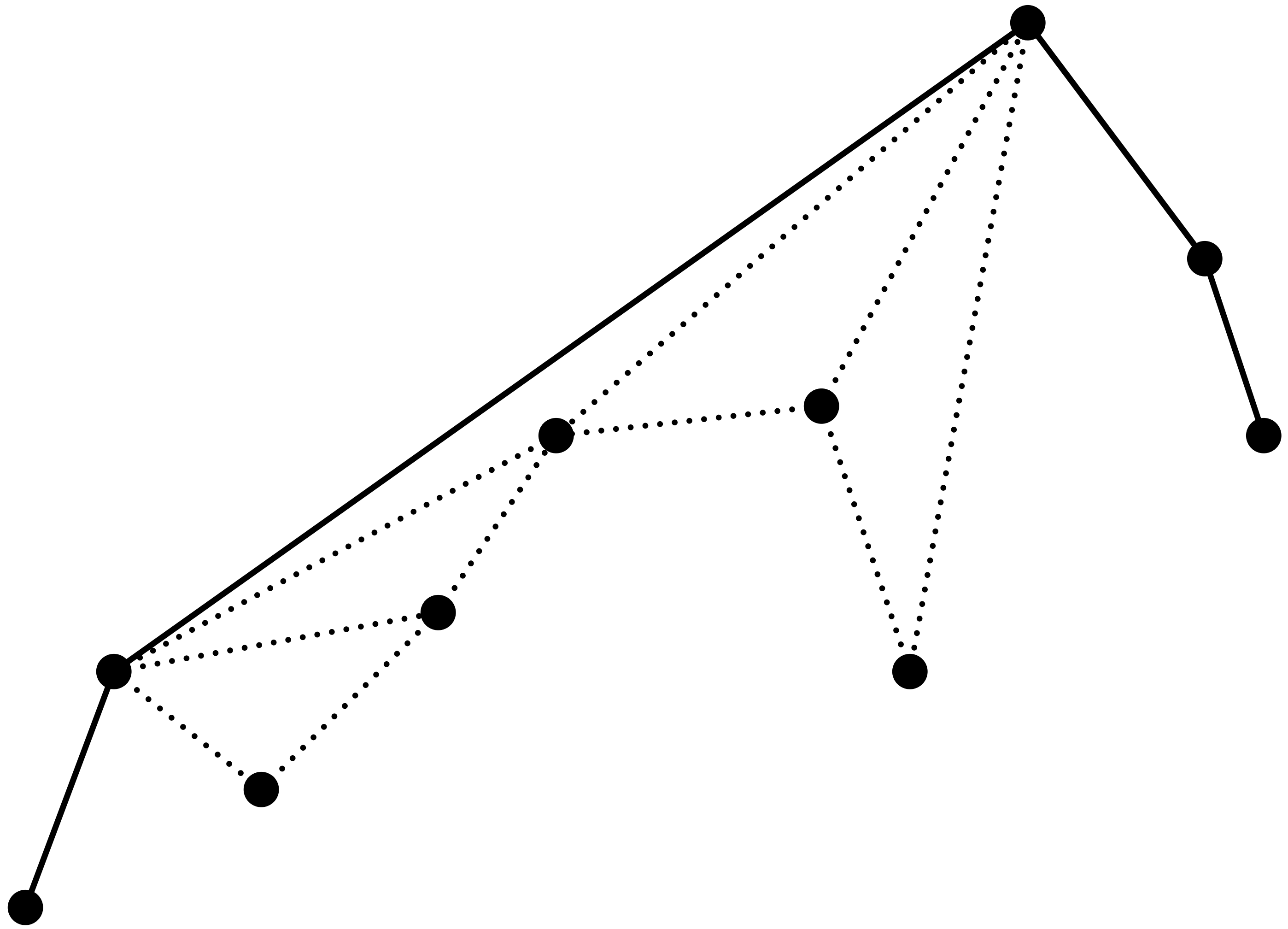
Convex hull algorithm 2

. . . and also the fifth point



Convex hull algorithm 2

after two more steps we get the upper hull



Convex hull algorithm 2: Graham Scan

Algorithm GRAHAMSCAN(P)

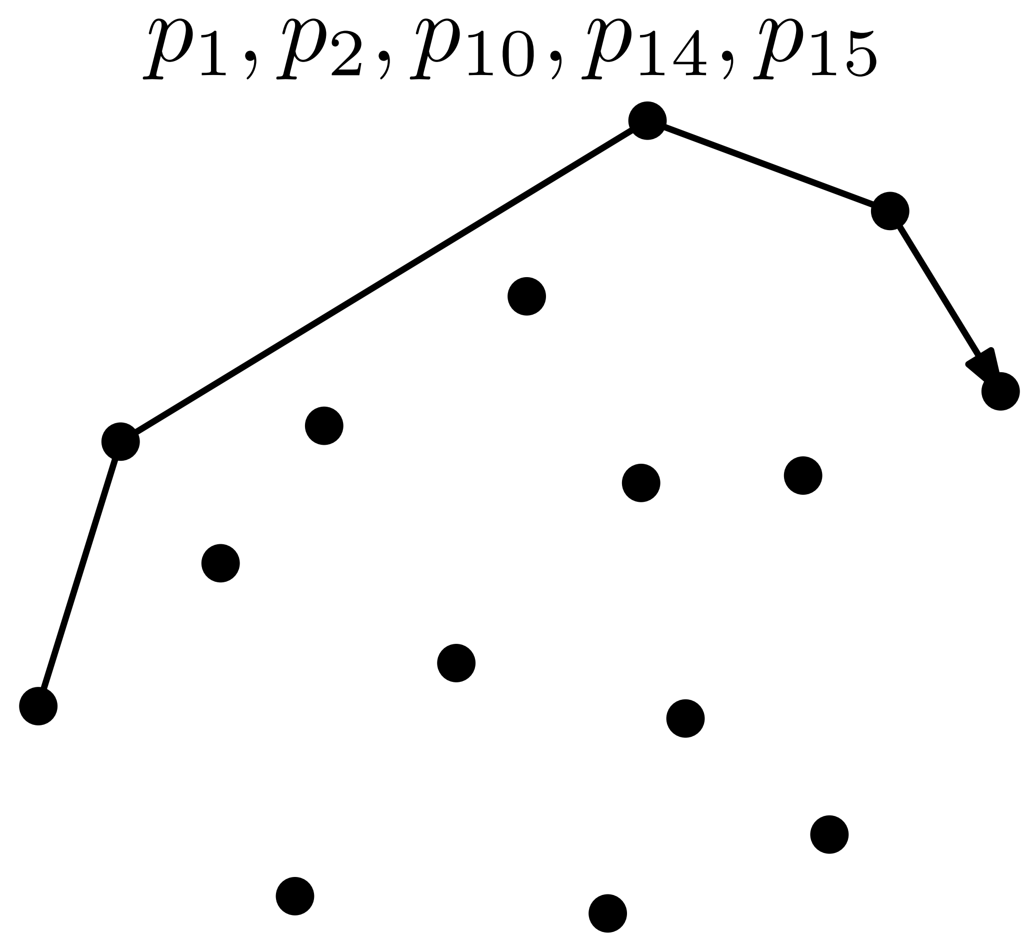
Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

```
sort the points by x-coordinate, resulting in a sequence p_1, ..., p_n
put the points p_1 and p_2 in a list L_upper, with p_1 as the first point
for i ← 3 to n do
    append p_i to L_upper
    while L_upper contains more than two points and the last three points do
    not make a right turn do
        delete the middle of the last three points from L_upper
```

Convex hull algorithm 2: Graham Scan

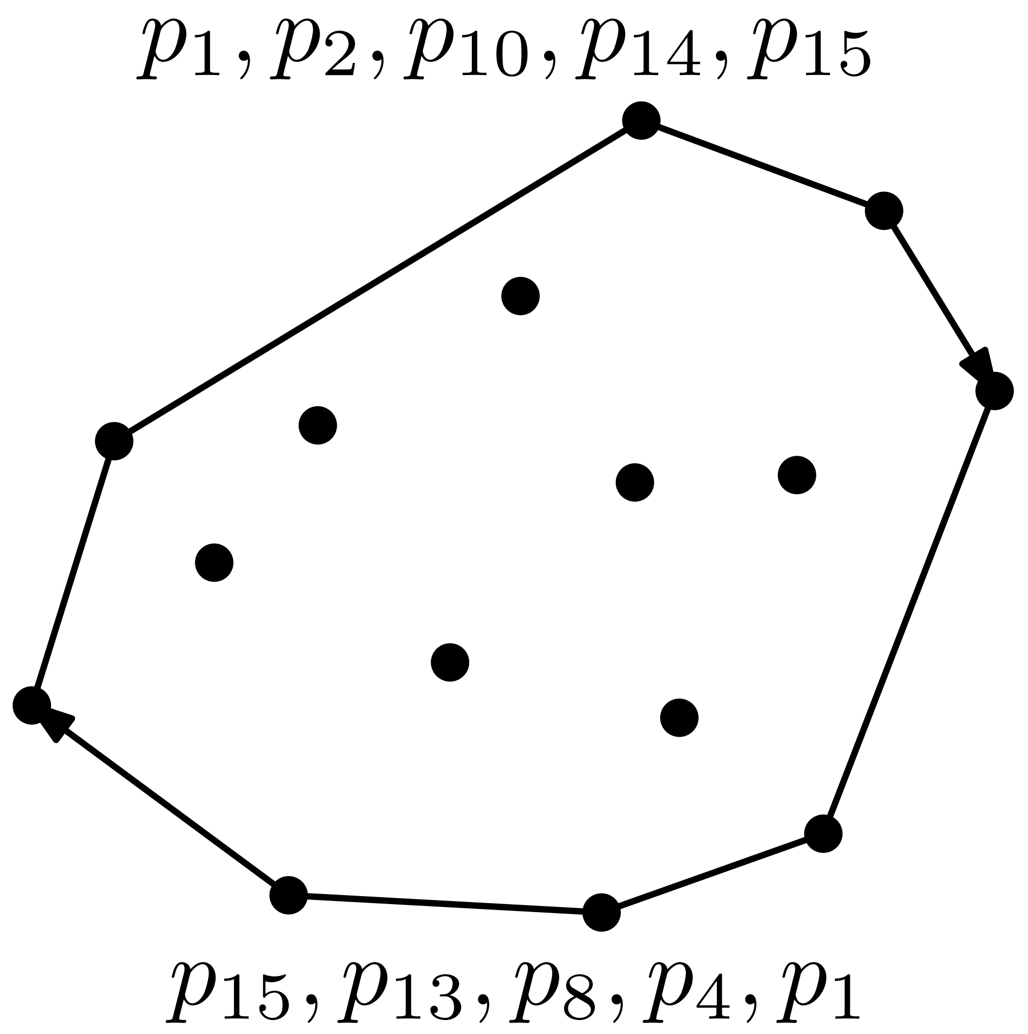
We have computed the upper convex hull



Convex hull algorithm 2: Graham Scan

We have computed the upper convex hull

Then we do the same for the lower convex hull, from right to left

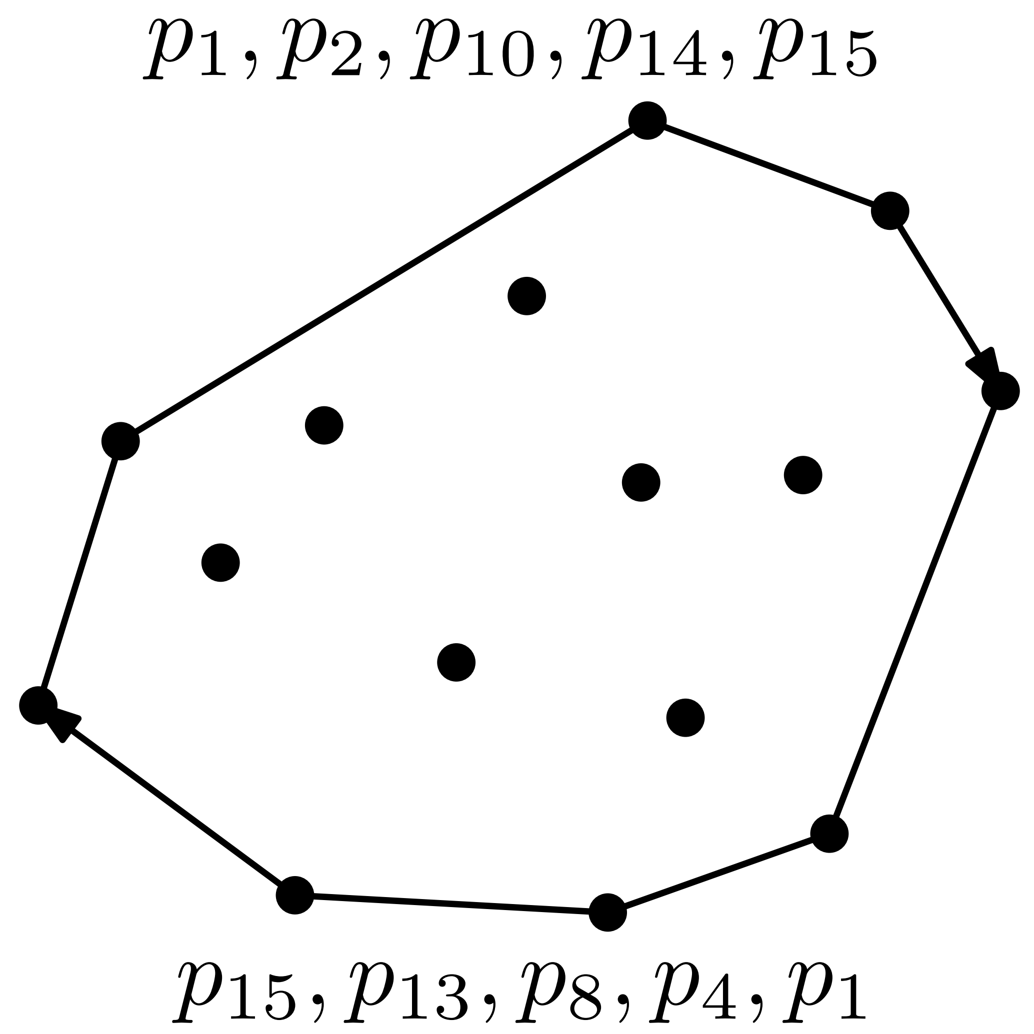


Convex hull algorithm 2: Graham Scan

We have computed the upper convex hull

Then we do the same for the lower convex hull, from right to left

We remove the first and last points of the lower convex hull

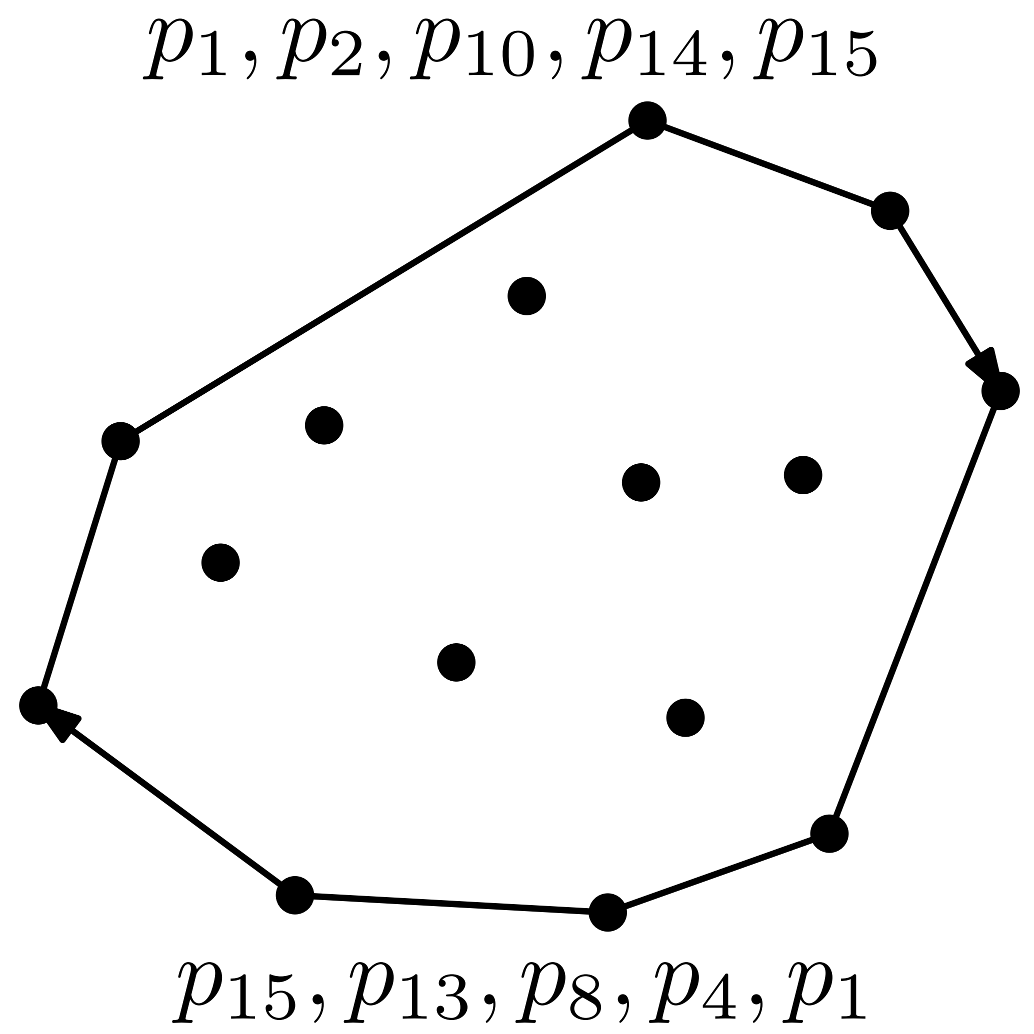


Convex hull algorithm 2: Graham Scan

We have computed the upper convex hull

Then we do the same for the lower convex hull, from right to left

We remove the first and last points of the lower convex hull . . . and concatenate the two lists in one



Algorithm analysis

Algorithm analysis generally has two components

- proof of correctness
- efficiency analysis, proof of running time

Correctness

Are the general observations on which the algorithm is based correct?

Does the algorithm handle degenerate cases correctly?

Correctness

Are the general observations on which the algorithm is based correct?

Does the algorithm handle degenerate cases correctly?

Here:

- Does the sorted order matter if two or more points have the same x-coordinate?
- What happens if there are three or more collinear points, in particular on the convex hull?

Efficiency

For each line of pseudocode identify

- how much time it takes
- how many times it is executed once

Efficiency

For each line of pseudocode identify
- how much time it takes
- how many times it is executed once

Operations on a constant number of constant-size objects take constant time

Efficiency

For each line of pseudocode identify

- how much time it takes
- how many times it is executed once

Operations on a constant number of constant-size objects take constant time

Consider the loop-structure and examine how often each line of pseudocode is executed

Efficiency

For each line of pseudocode identify

- how much time it takes
- how many times it is executed once

Operations on a constant number of constant-size objects take constant time

Consider the loop-structure and examine how often each line of pseudocode is executed

Sometimes there are global arguments why an algorithm is more efficient than it seems at first

Efficiency

Algorithm GRAHAMSCAN(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

1: sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$
2: put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point
3: **for** $i \leftarrow 3$ to n **do**
4: append p_i to L_{upper}
5: **while** L_{upper} contains more than two points **and** the last three points do not make a right turn **do**
6: delete the middle of the last three points from L_{upper}

Efficiency

Algorithm GRAHAMSCAN(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

1: sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$ $\quad O(n \log n)$
2: put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point
3: **for** $i \leftarrow 3$ to n **do**
4: append p_i to L_{upper}
5: **while** L_{upper} contains more than two points **and** the last three points do not make a right turn **do**
6: delete the middle of the last three points from L_{upper}

Efficiency

Algorithm GRAHAMSCAN(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

1: sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$ $O(n \log n)$

2: put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point $O(1)$

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

4: append p_i to L_{upper}

5: **while** L_{upper} contains more than two points **and** the last three points do not make a right turn **do**

6: delete the middle of the last three points from L_{upper}

Efficiency

Algorithm GRAHAMSCAN(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$ $O(n \log n)$

put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point $O(1)$

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

  append p_i to L_{upper} $O(1)$

  while L_{upper} contains more than two points **and** the last three points do not make a right turn **do**

    delete the middle of the last three points from L_{upper}

Efficiency

Algorithm GrahamScan(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$ $O(n \log n)$

put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point $O(1)$

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

    append p_i to L_{upper} $O(1)$

    while L_{upper} contains more than two points **and** the last three points do not make a right turn **do**

        delete the middle of the last three points from L_{upper} $O(1)$

Efficiency

Algorithm GRAHAMSCAN(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

1: sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$ $O(n \log n)$

2: put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point $O(1)$

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

4: append p_i to L_{upper} $O(1)$

5: **while** L_{upper} contains more than two points **and** the last three points do not make a right turn **do**

6: delete the middle of the last three points from L_{upper} $O(1)$

$\times n-2$

Efficiency

Algorithm GRAHAMSCAN(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$ $\quad O(n \log n)$

put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point $\quad O(1)$

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

  append p_i to L_{upper} $\quad O(1)$

  while L_{upper} contains more than two points **and** the last three points do not make a right turn **do**

    delete the middle of the last three points from L_{upper} $\quad O(1)$ $\}\times k_i$

$\}\times n-2$

Efficiency

Algorithm GRAHAMSCAN(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

1: sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$ — $O(n \log n)$
2: put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point — $O(1)$
3: **for** $i \leftarrow 3$ to n **do** — $\times n-2$
4: append p_i to L_{upper} — $O(1)$
5: **while** L_{upper} contains more than two points **and** the last three points do not make a right turn **do**
6: delete the middle of the last three points from L_{upper} — $O(1)$ $\}\times k_i$

Total time:

$$O(n \log n) + O(1) + \sum_{i=3}^{n} (O(1) + k_i \cdot O(1)) =$$

Efficiency

Algorithm GRAHAMSCAN(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in clockwise order

1: sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$ — $O(n \log n)$

2: put the points p_1 and p_2 in a list L_{upper}, with p_1 as the first point — $O(1)$

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

4: append p_i to L_{upper} — $O(1)$

5: **while** L_{upper} contains more than two points **and** the last three points do not make a right turn **do**

6: delete the middle of the last three points from L_{upper} — $O(1)$ $\}\times k_i$

$\}\times n-2$ (lines 4–6)

Total time:

$$O(n \log n) + O(1) + \sum_{i=3}^{n} (O(1) + k_i \cdot O(1)) = O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

Efficiency: attempt 1

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

Efficiency: attempt 1

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

Since $k_i = O(n)$, we get

Efficiency: attempt 1

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

Since $k_i = O(n)$, we get

$$O(n \log n) + \sum_{i=3}^{n} O(n) = O(n^2)$$

Efficiency: attempt 1

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

Since $k_i = O(n)$, we get

$$O(n \log n) + \sum_{i=3}^{n} O(n) = O(n^2)$$

Question: Is this analysis tight?

Efficiency: attempt 1

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

Since $k_i = O(n)$, we get

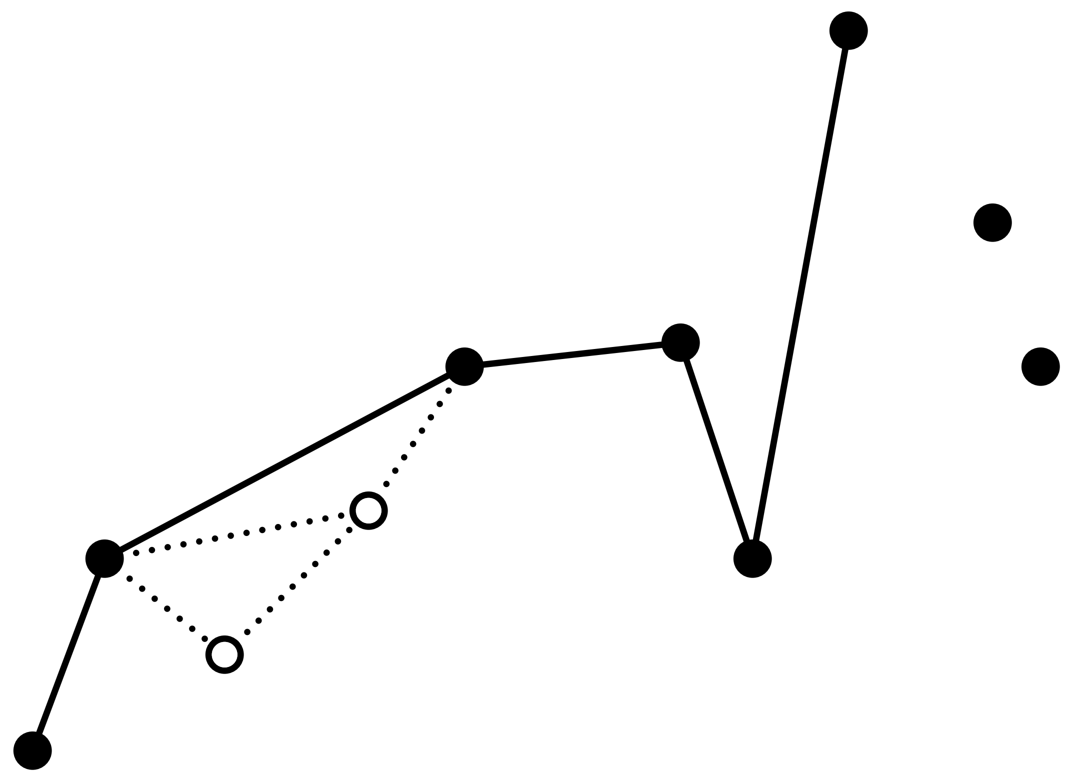
$$O(n \log n) + \sum_{i=3}^{n} O(n) = O(n^2)$$

Question: Is this analysis tight?

Sometimes there are global arguments why an algorithm is more efficient than it seems at first

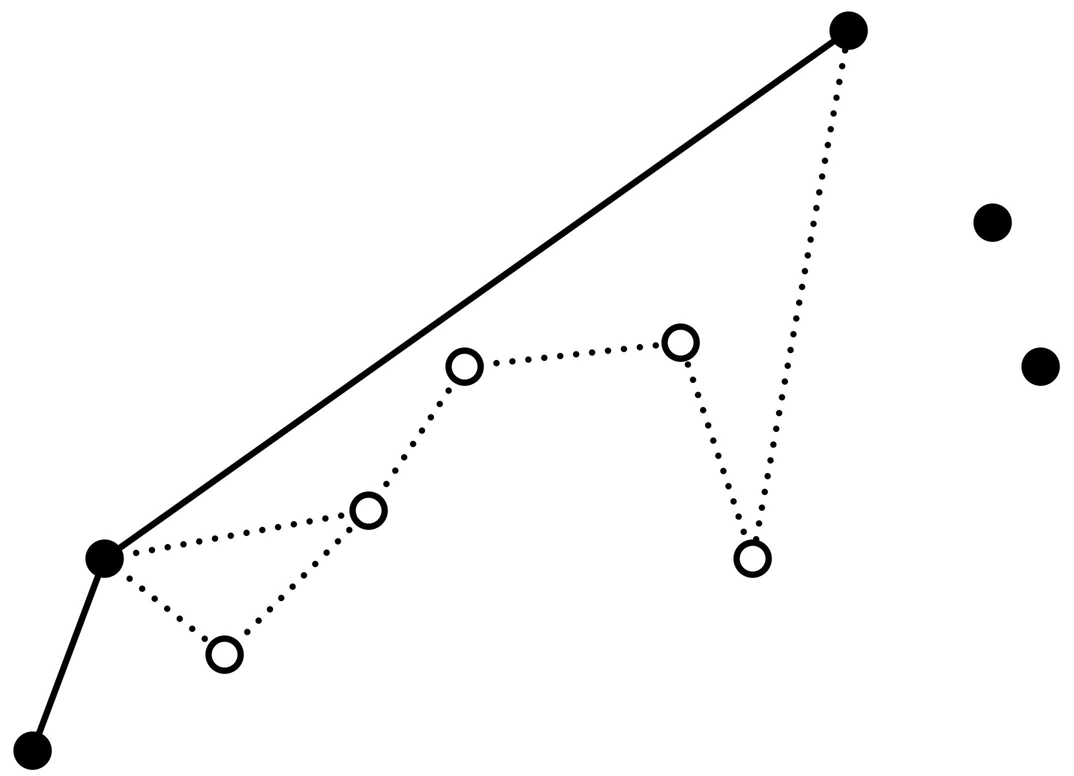
Efficiency: attempt 2

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$



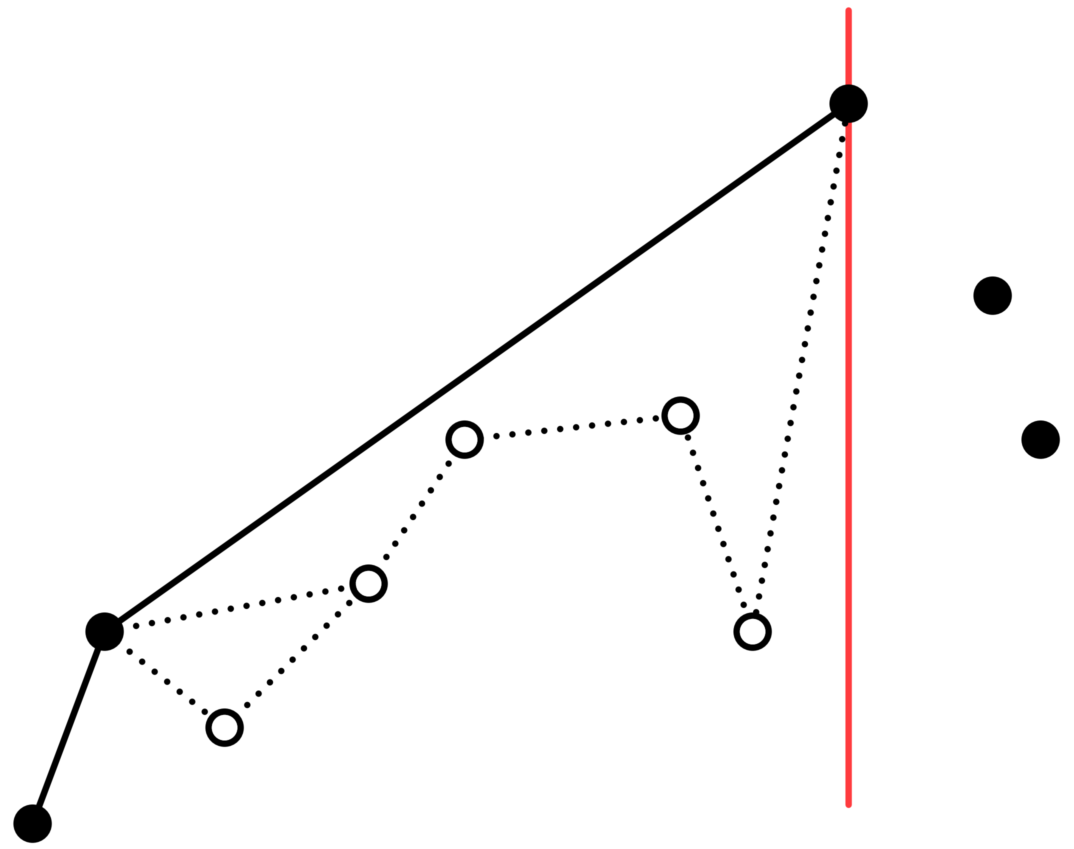
Efficiency: attempt 2

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$



Efficiency: attempt 2

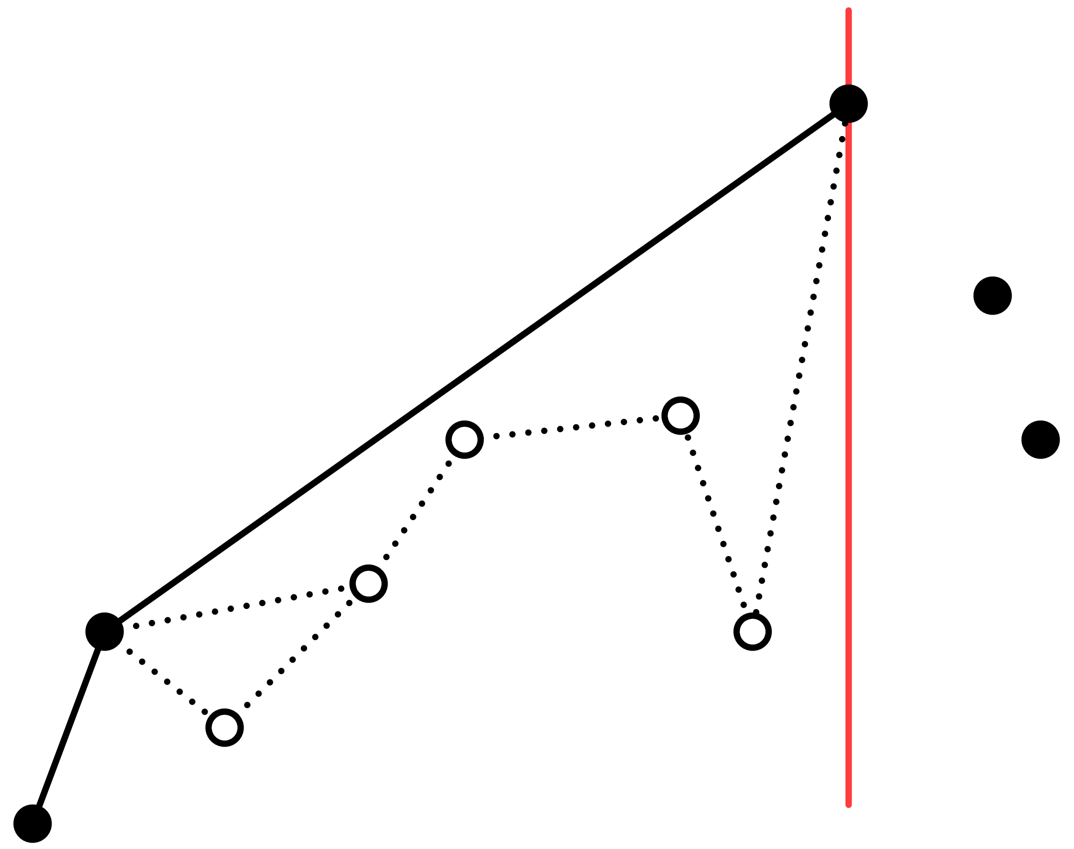
Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$



Efficiency: attempt 2

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

Global argument: each point can be removed only once from the upper hull

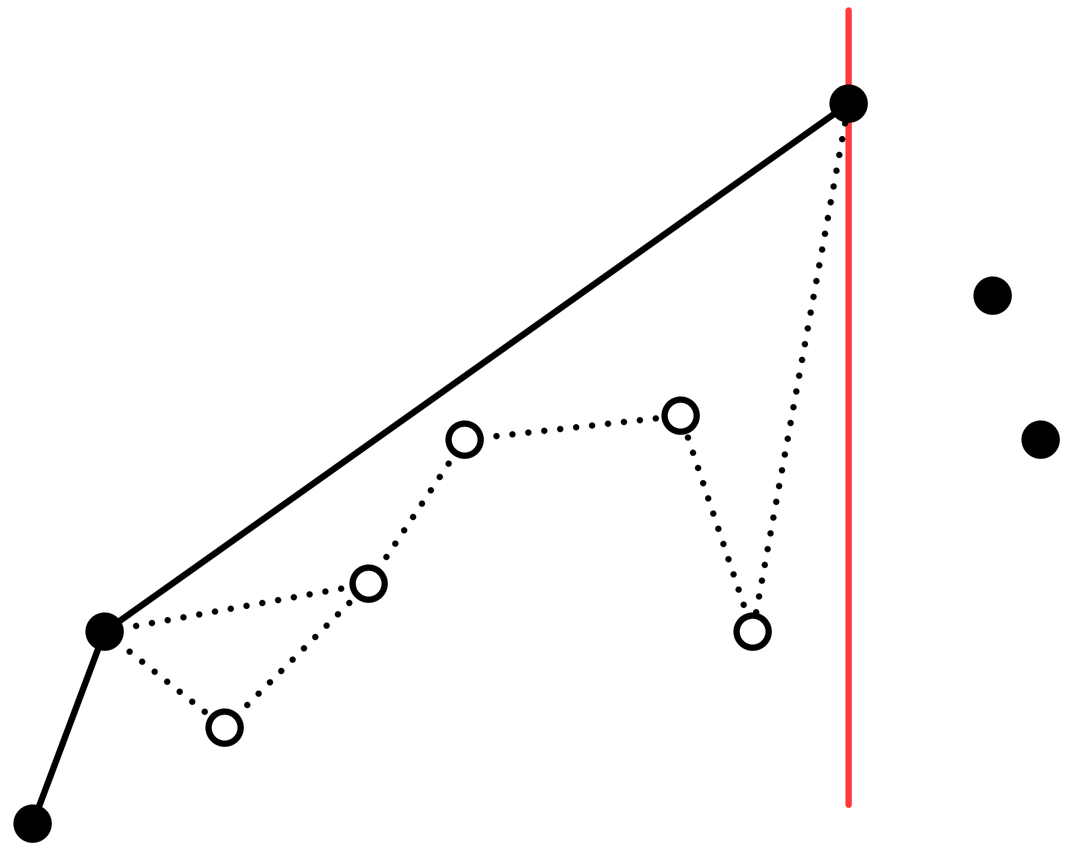


Efficiency: attempt 2

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

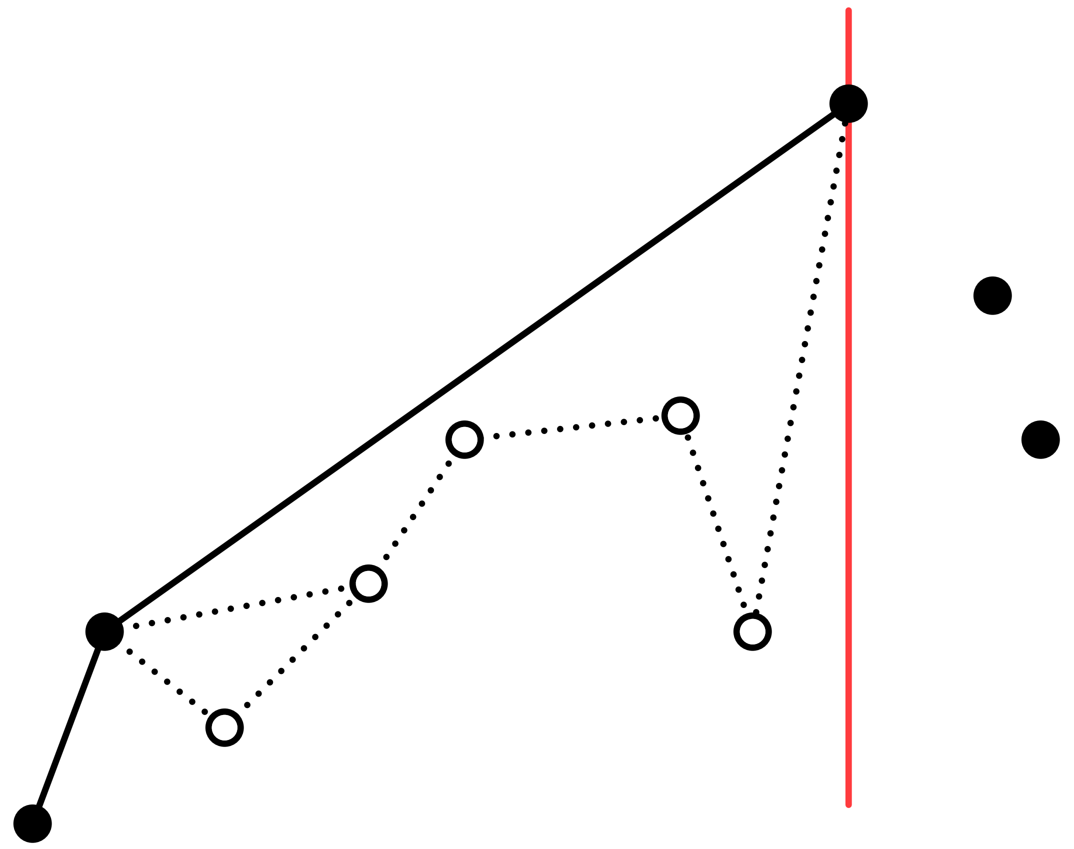
Global argument: each point can be removed only once from the upper hull

This gives us: $$\sum_{i=3}^{n} k_i \leq n$$



Efficiency: attempt 2

Total time: $$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$



Global argument: each point can be removed only once from the upper hull

This gives us: $$\sum_{i=3}^{n} k_i \leq n$$

Hence,

$$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i) = O(n \log n) + O(n) = O(n \log n)$$

Result

The convex hull of a set of n points in the plane can be computed in $O(n \log n)$ time, and this is worst-case optimal.

Result

The convex hull of a set of n points in the plane can be computed in $O(n \log n)$ time, and this is worst-case optimal.

Question: Can we do better?

Result

The convex hull of a set of n points in the plane can be computed in $O(n \log n)$ time, and this is worst-case optimal.

Question: Can we do better?

$O(nh)$ if the convex hull has h vertices?
$O(n \log h)$?

Result

The convex hull of a set of n points in the plane can be computed in $O(n \log n)$ time, and this is worst-case optimal.

Question: Can we do better?

$O(nh)$ if the convex hull has h vertices?
$O(n \log h)$? Yes we can!

An output-sensitive algorithm

Idea: start on the convex hull and wrap around the convex hull

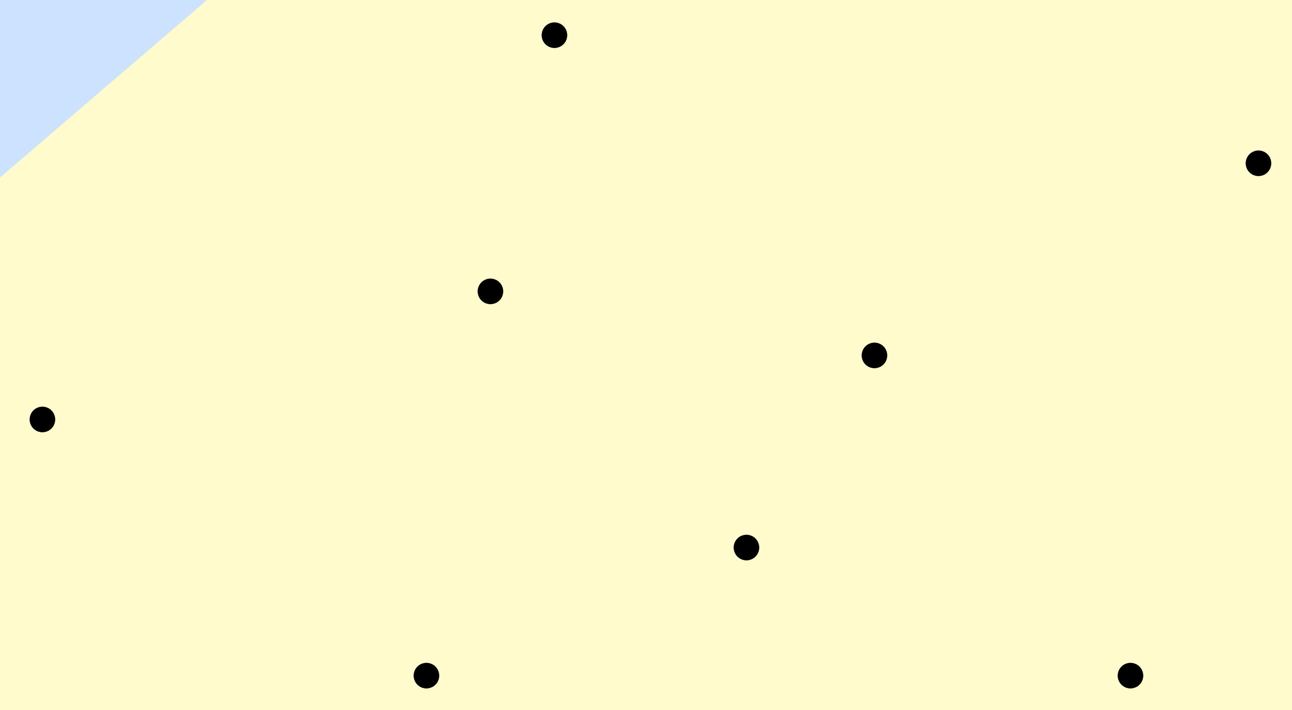
An output-sensitive algorithm

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

```
p_0 ← (∞, ∞), p_1 ← right-most vertex in P, insert p_1 into L
while true do
    choose p_{i+1} ∈ P maximizing ∠p_{i-1} p_i p_{i+1}
    if p_{i+1} = p_1 then
        break
    insert p_{i+1} into L
```



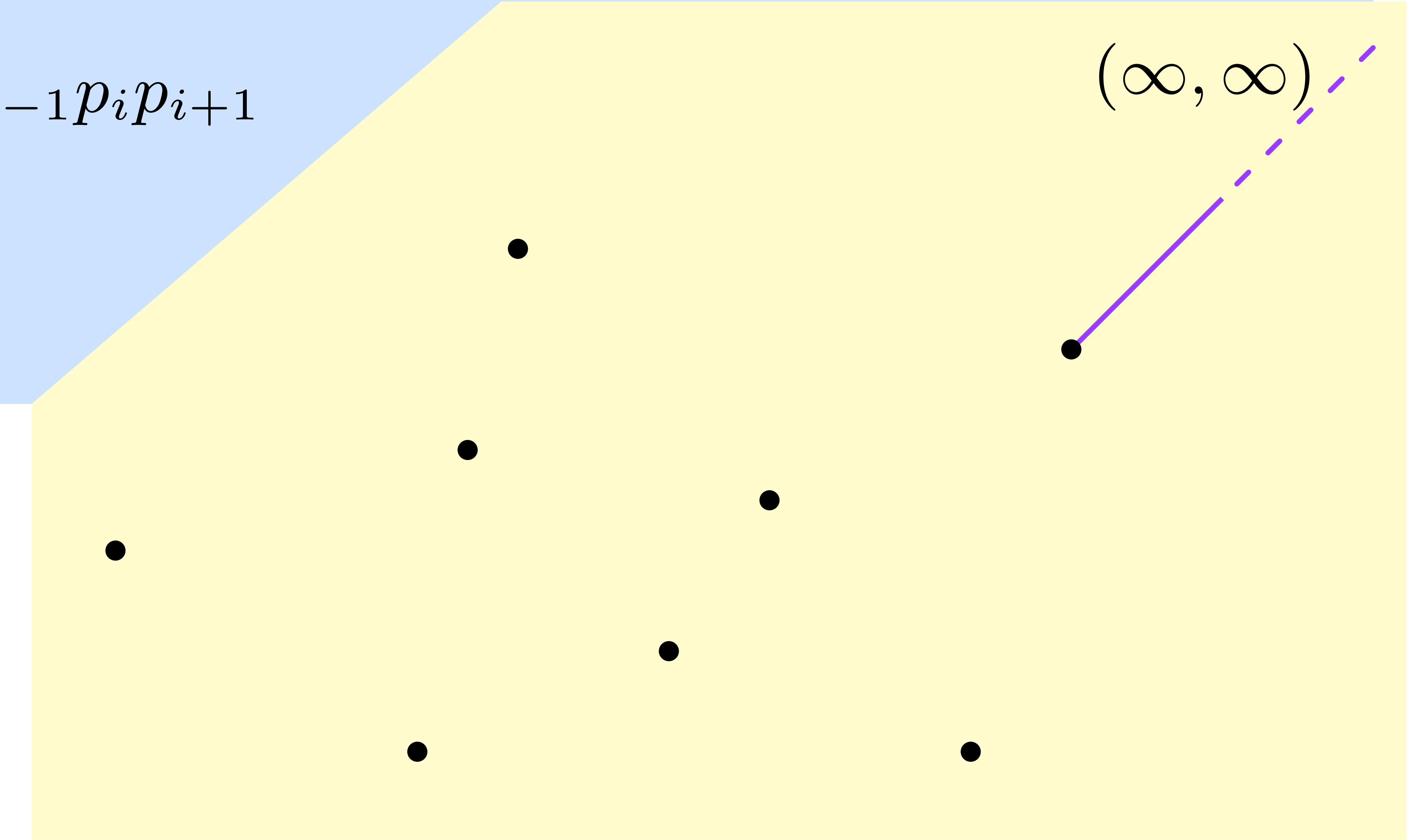
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

```
p_0 ← (∞, ∞), p_1 ← right-most vertex in P, insert p_1 into L
while true do
    choose p_{i+1} ∈ P maximizing ∠p_{i−1} p_i p_{i+1}
    if p_{i+1} = p_1 then
        break
    insert p_{i+1} into L
```



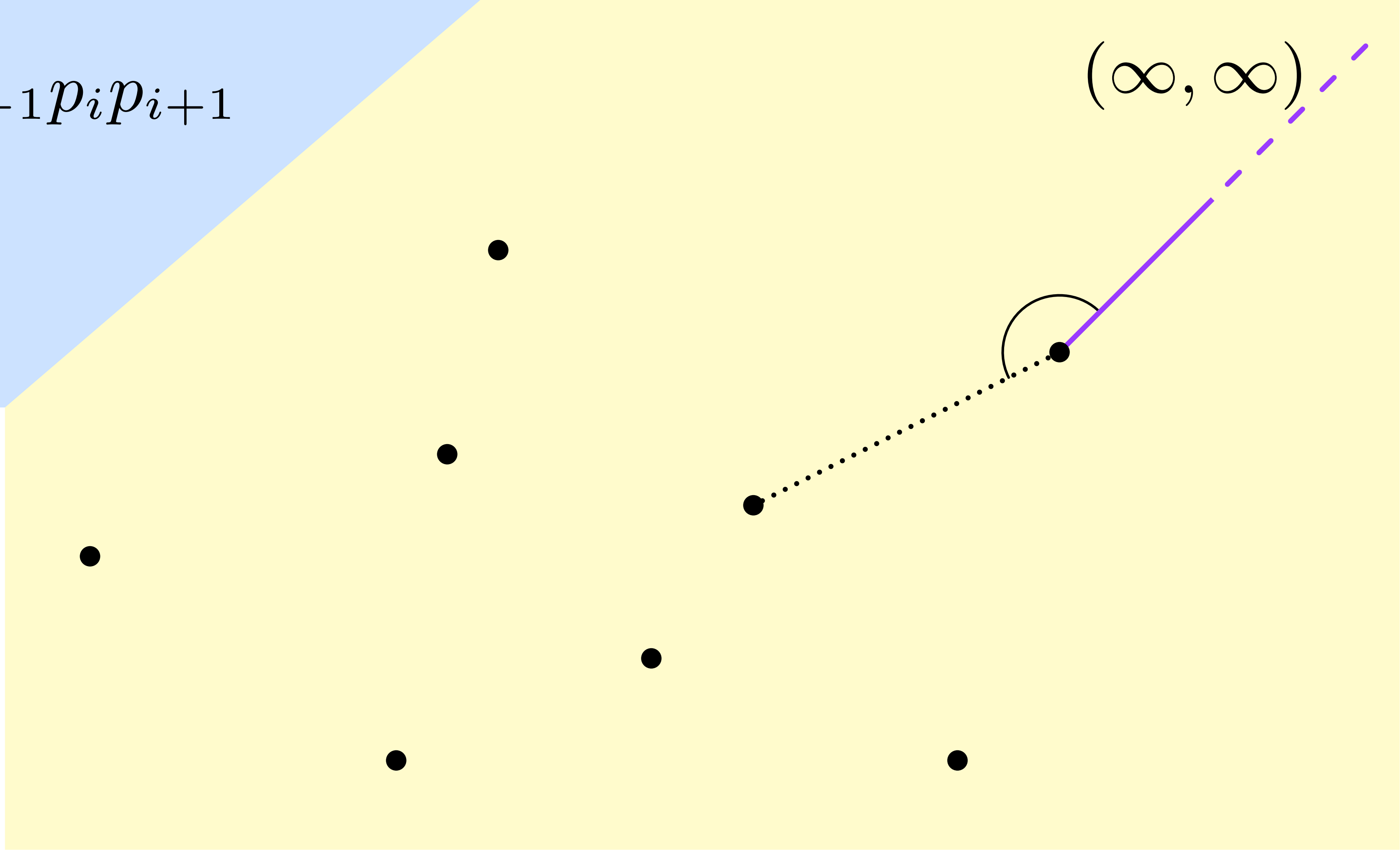
An output-sensitive algorithm

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



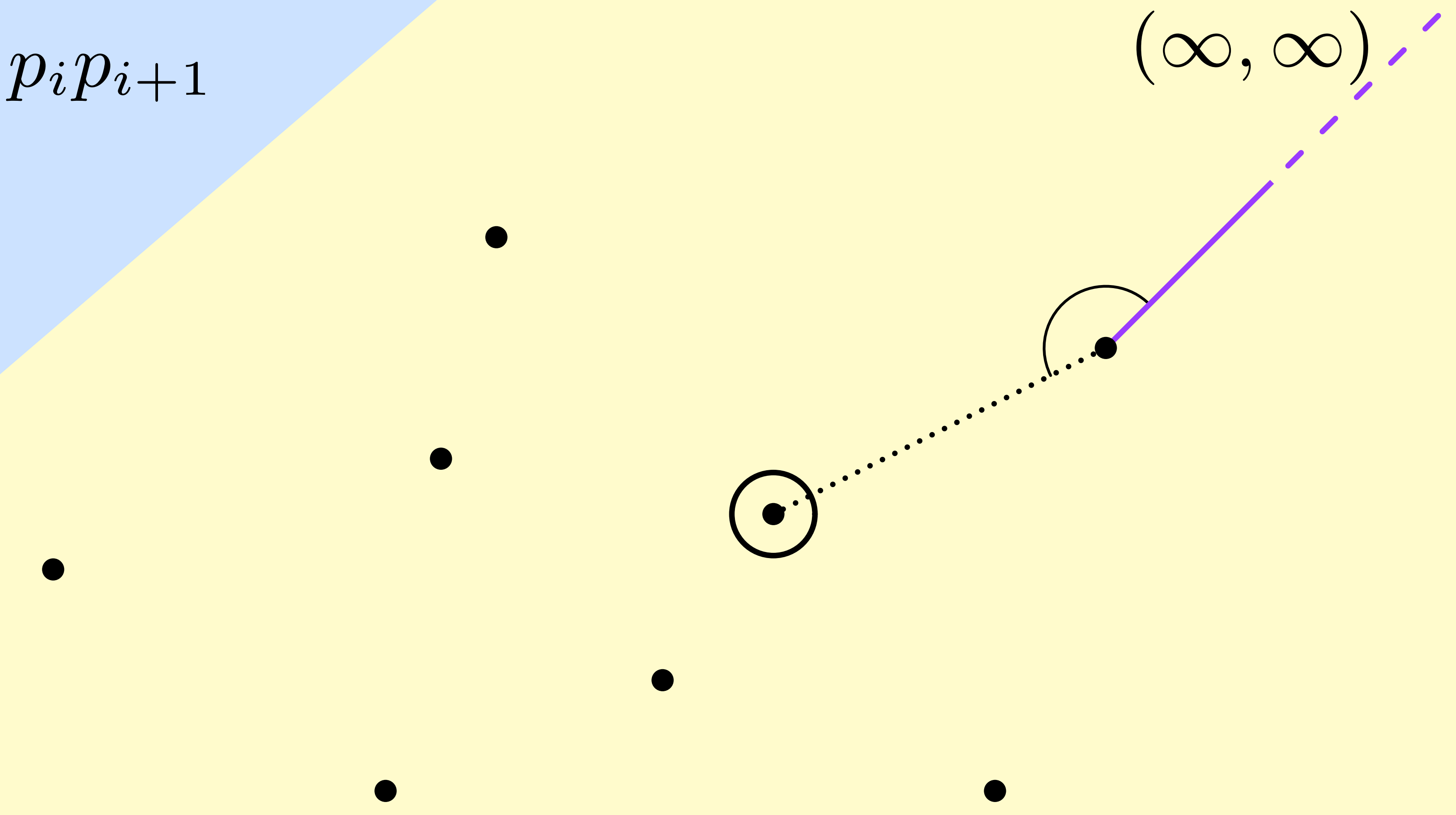
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

```
p_0 ← (∞, ∞), p_1 ← right-most vertex in P, insert p_1 into L
while true do
    choose p_{i+1} ∈ P maximizing ∠p_{i−1} p_i p_{i+1}
    if p_{i+1} = p_1 then
        break
    insert p_{i+1} into L
```



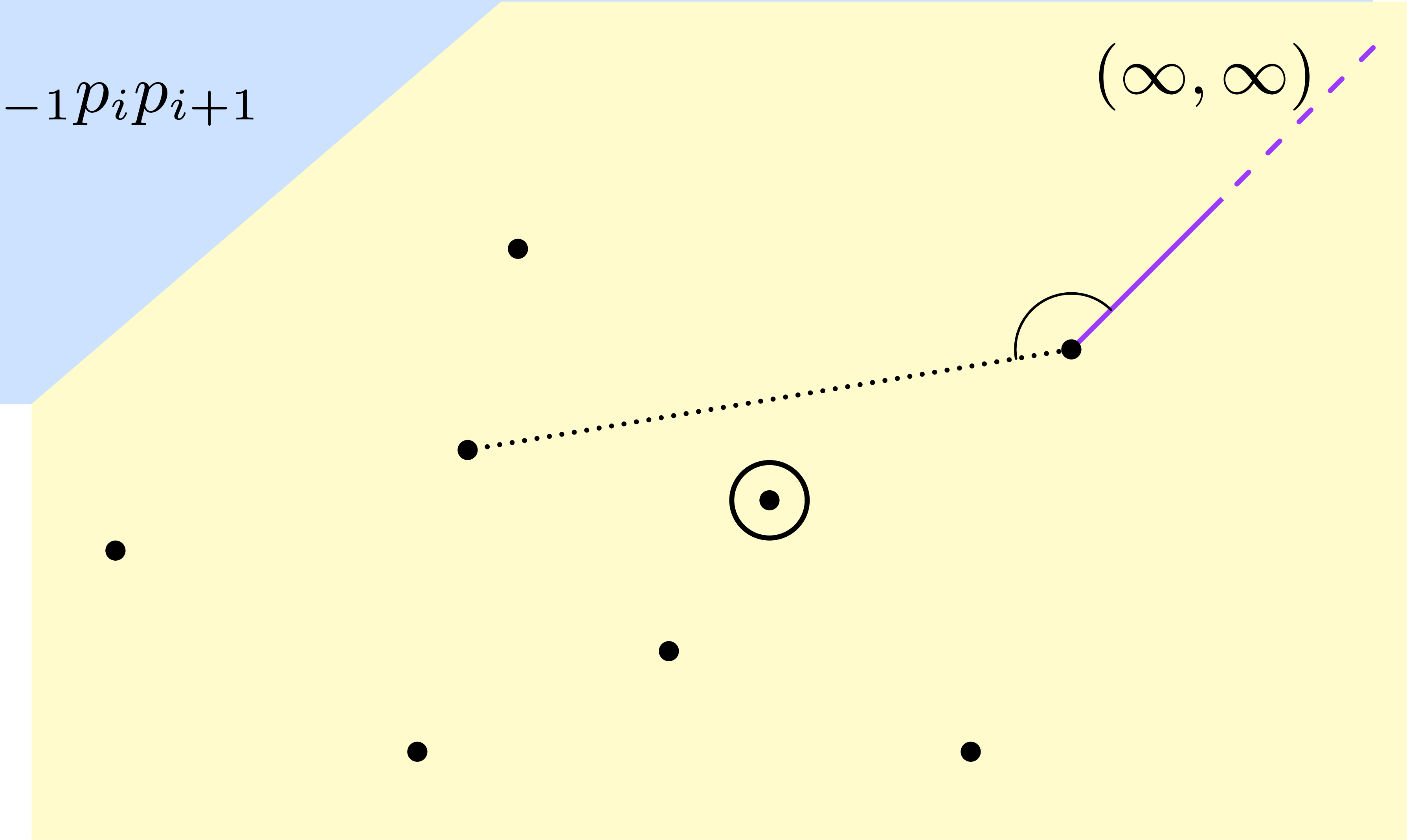
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

```
p_0 ← (∞, ∞), p_1 ← right-most vertex in P, insert p_1 into L
while true do
    choose p_{i+1} ∈ P maximizing ∠p_{i−1} p_i p_{i+1}
    if p_{i+1} = p_1 then
        break
    insert p_{i+1} into L
```



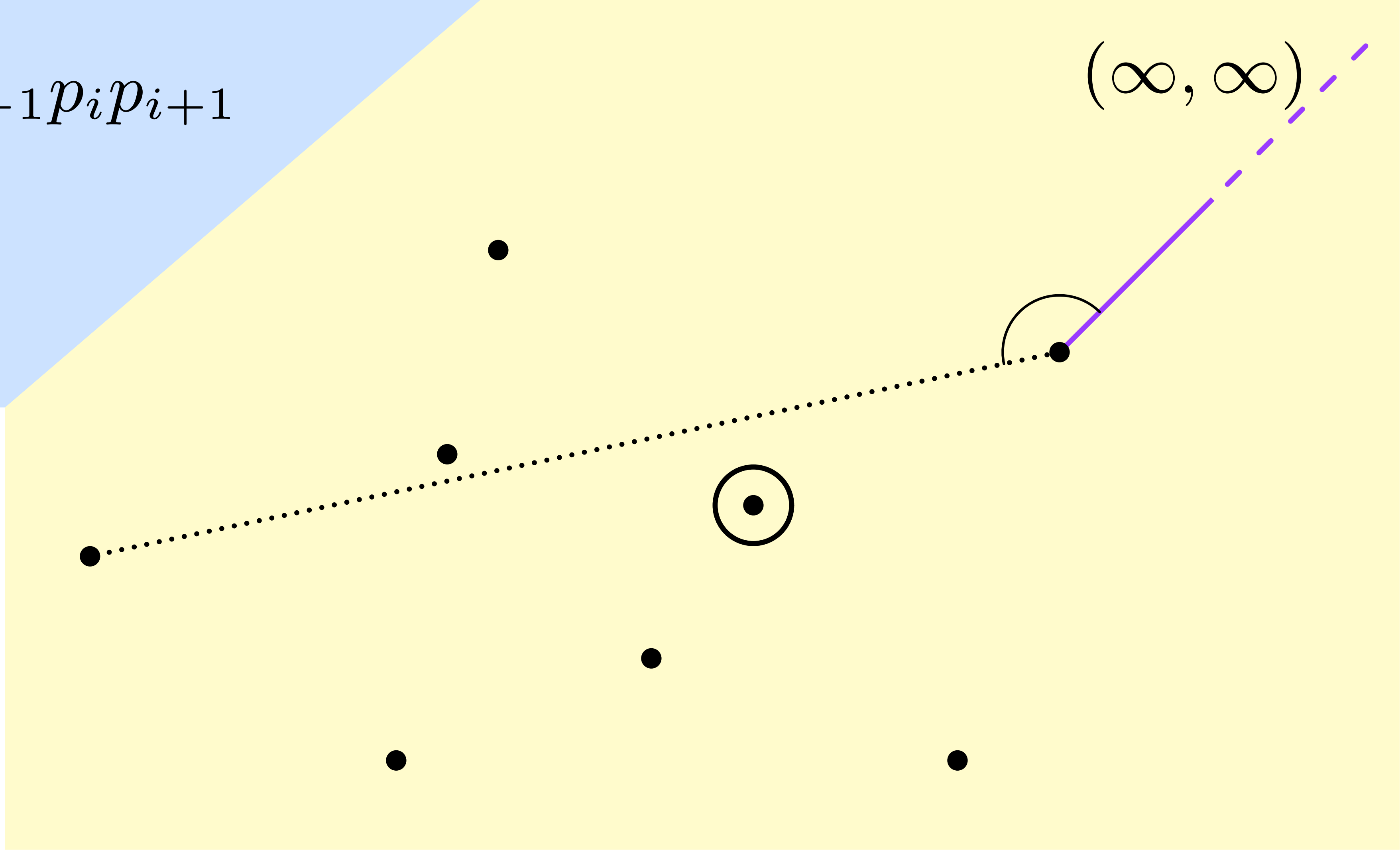
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



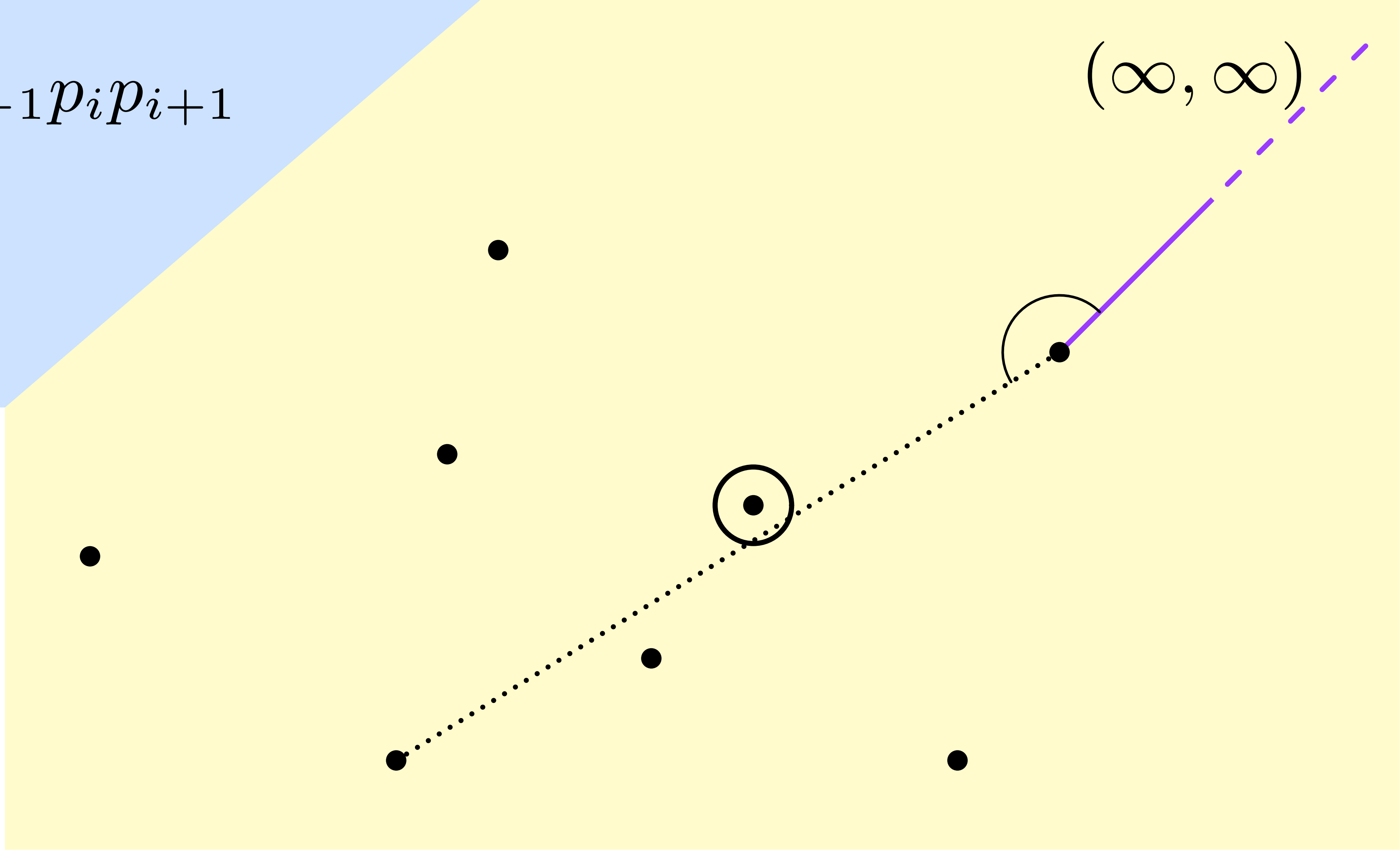
An output-sensitive algorithm

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



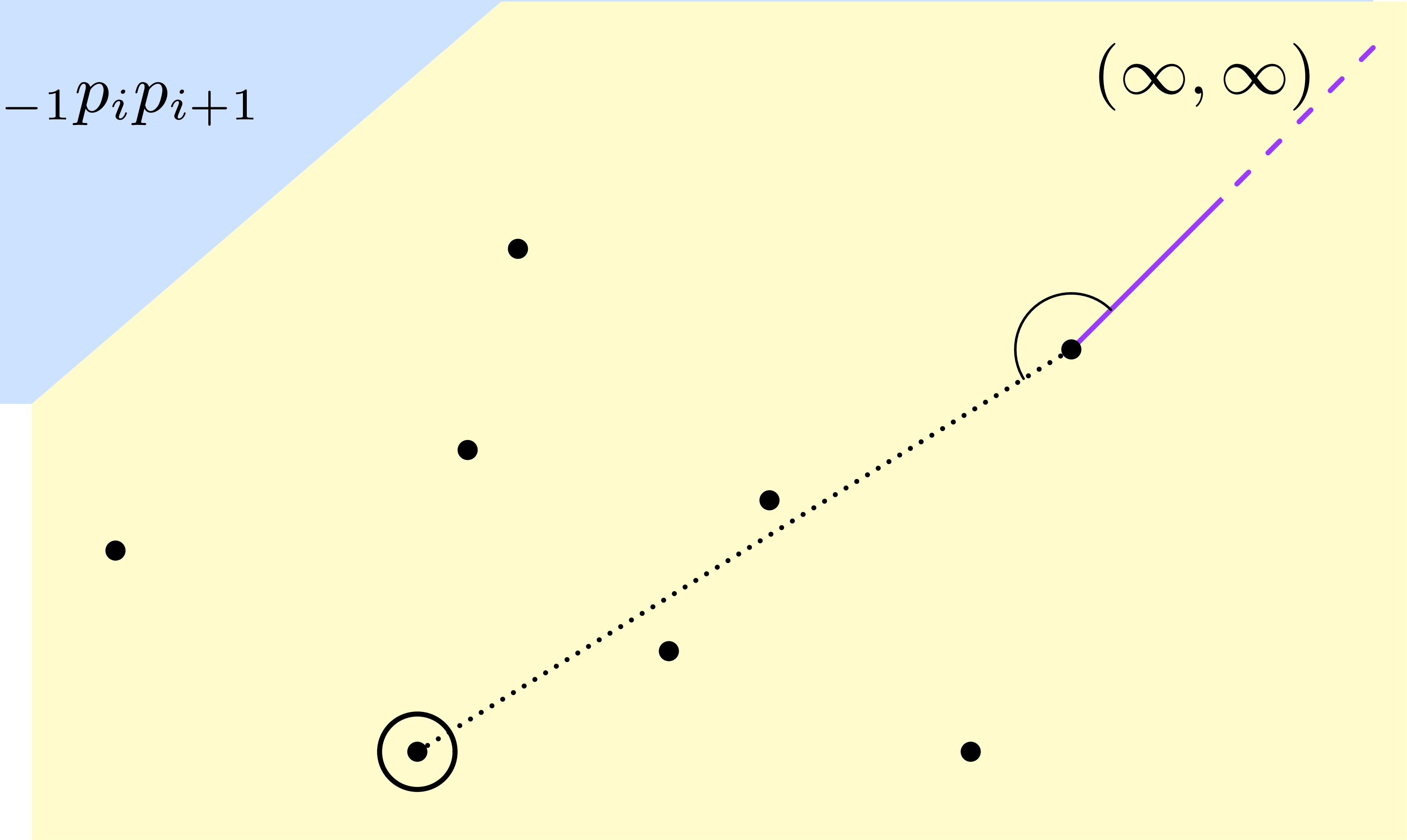
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

```
p_0 ← (∞, ∞), p_1 ← right-most vertex in P, insert p_1 into L
while true do
    choose p_{i+1} ∈ P maximizing ∠p_{i-1} p_i p_{i+1}
    if p_{i+1} = p_1 then
        break
    insert p_{i+1} into L
```



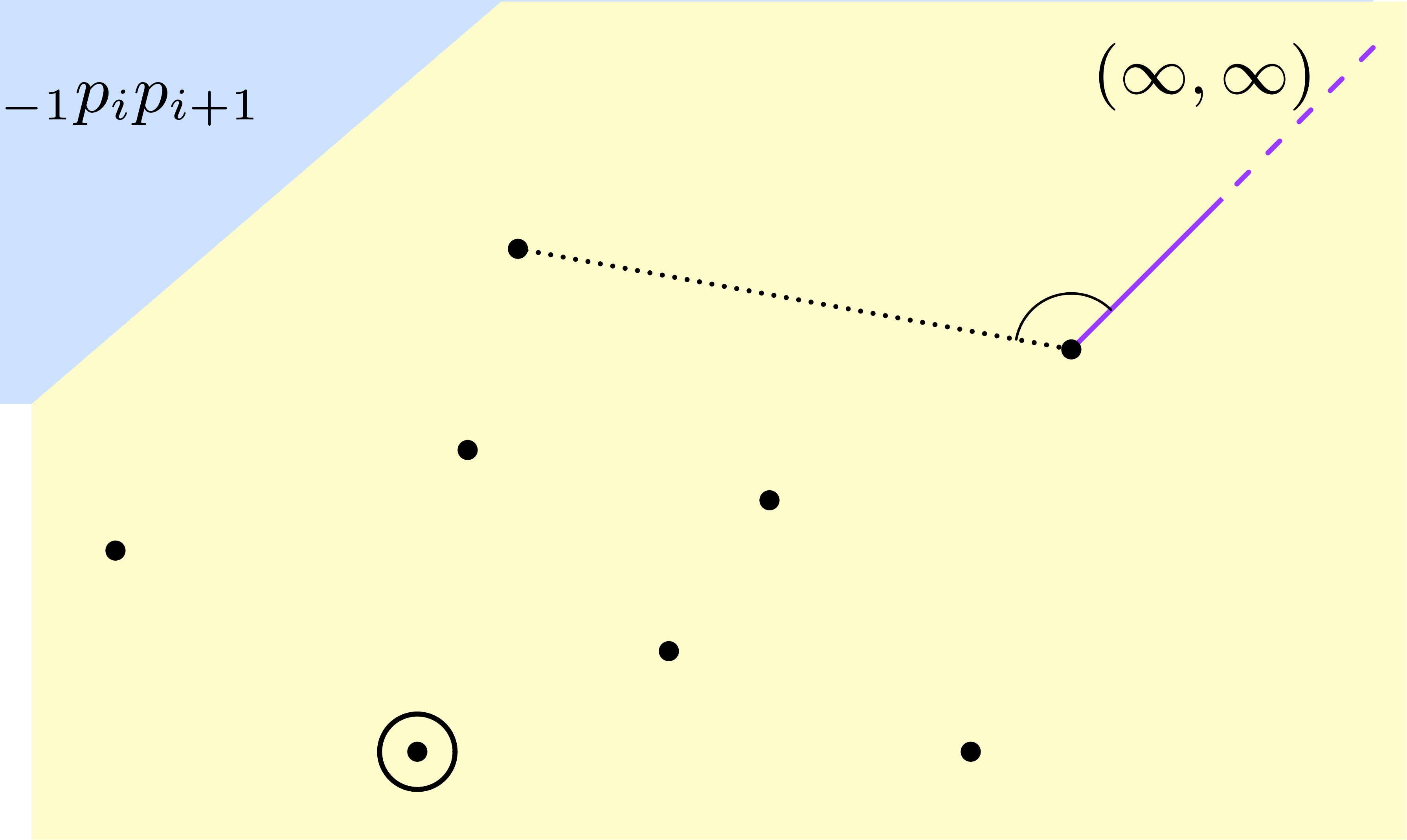
An output-sensitive algorithm

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



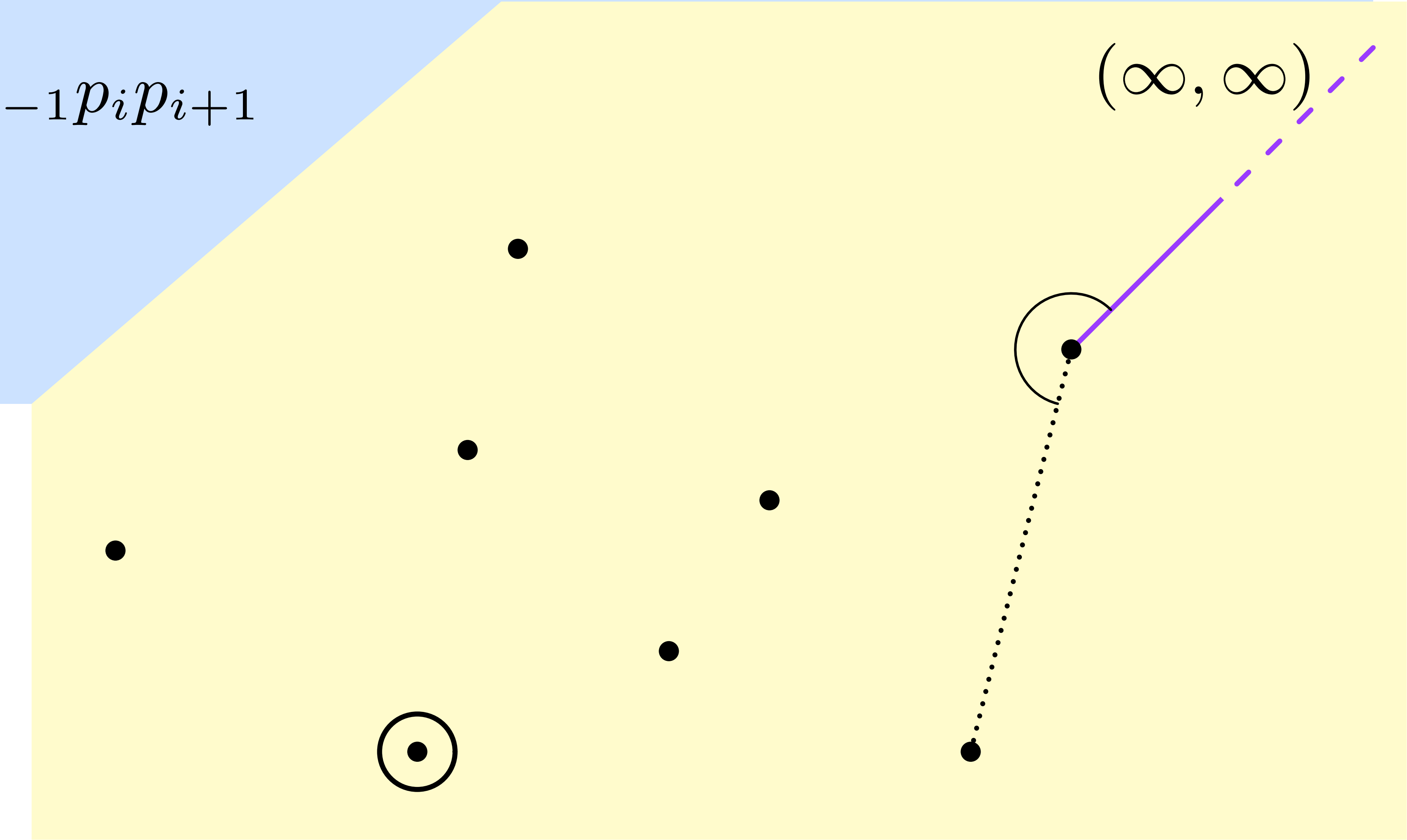
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



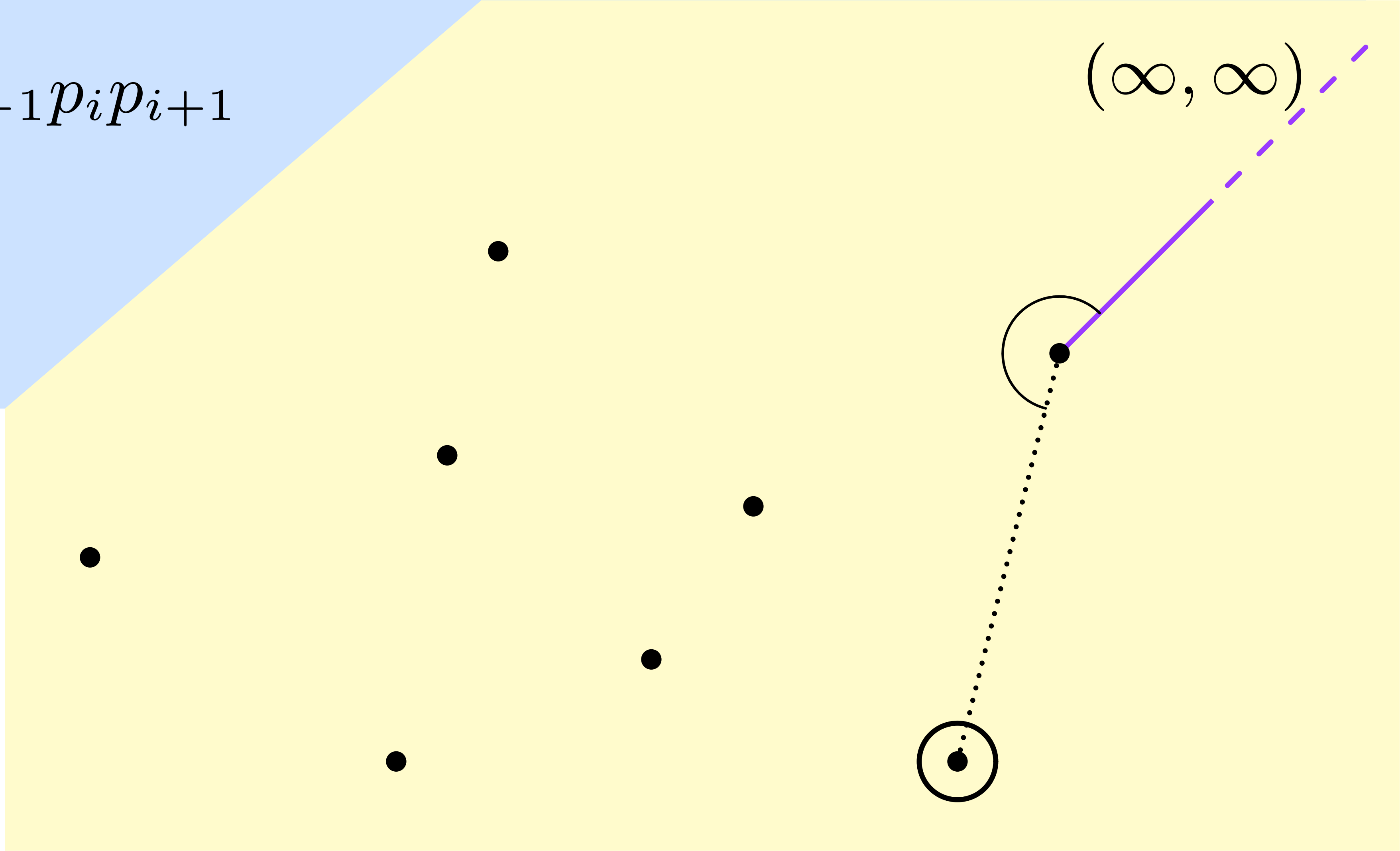
An output-sensitive algorithm

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



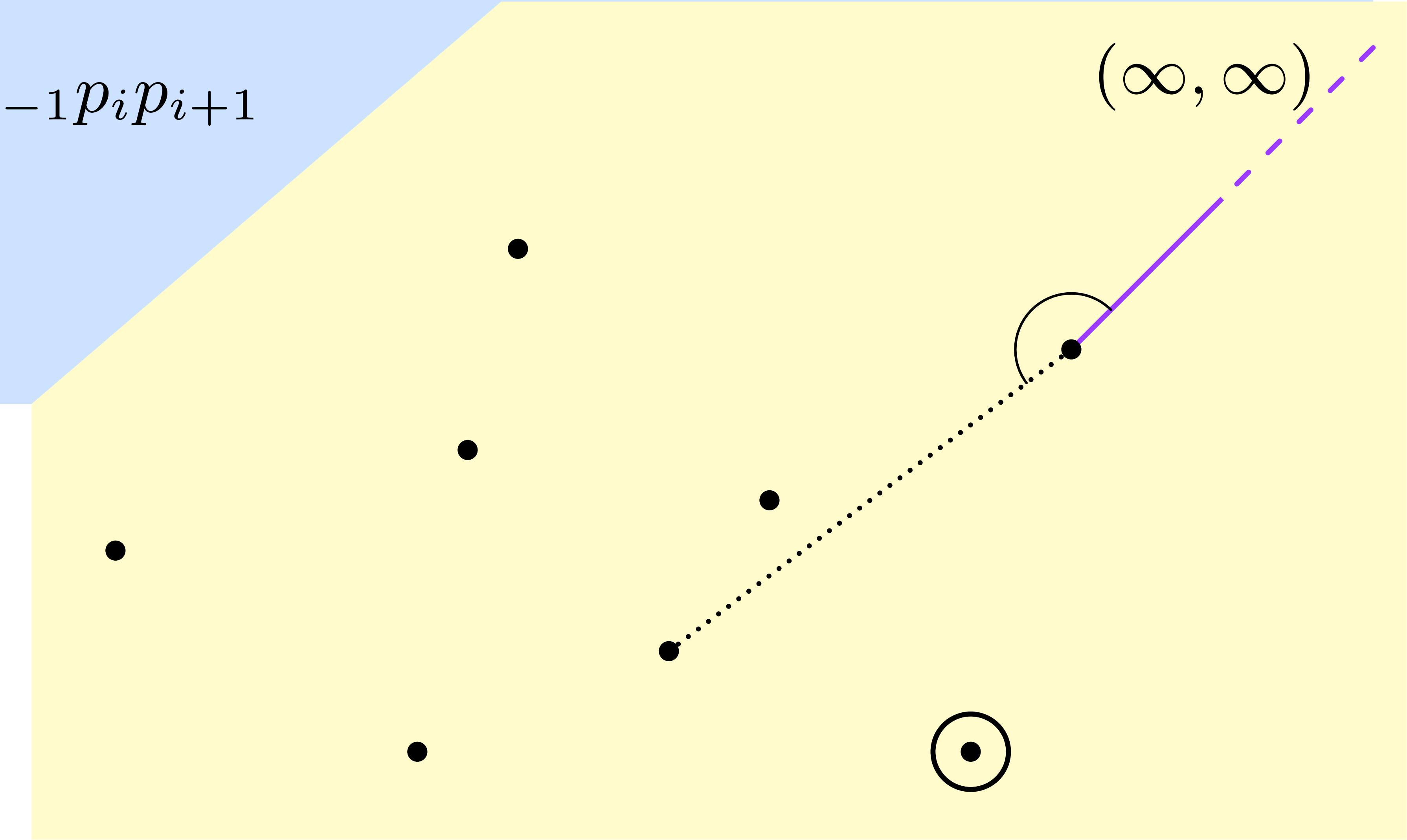
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

```
p_0 ← (∞, ∞), p_1 ← right-most vertex in P, insert p_1 into L
while true do
    choose p_{i+1} ∈ P maximizing ∠p_{i−1} p_i p_{i+1}
    if p_{i+1} = p_1 then
        break
    insert p_{i+1} into L
```



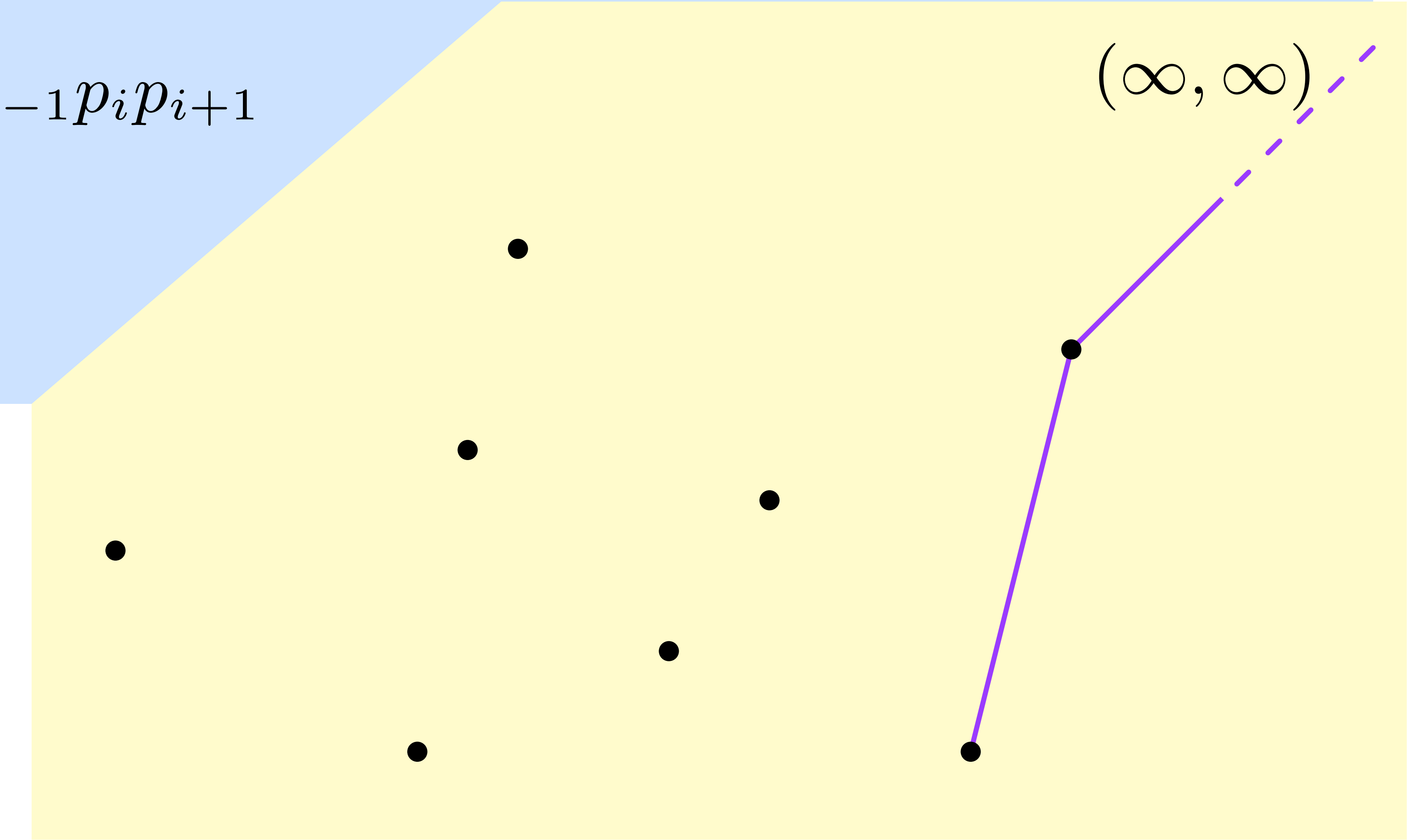
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



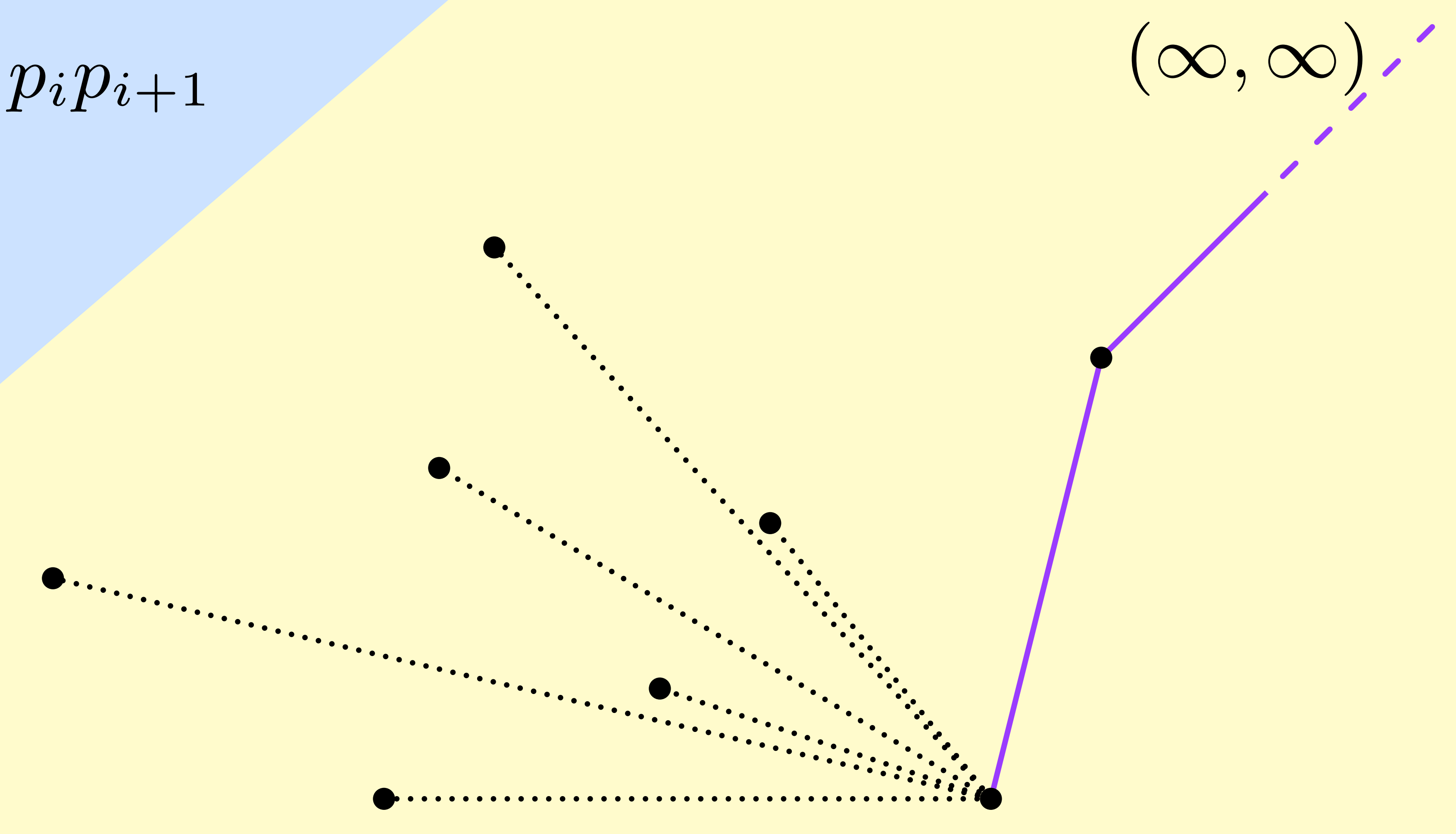
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



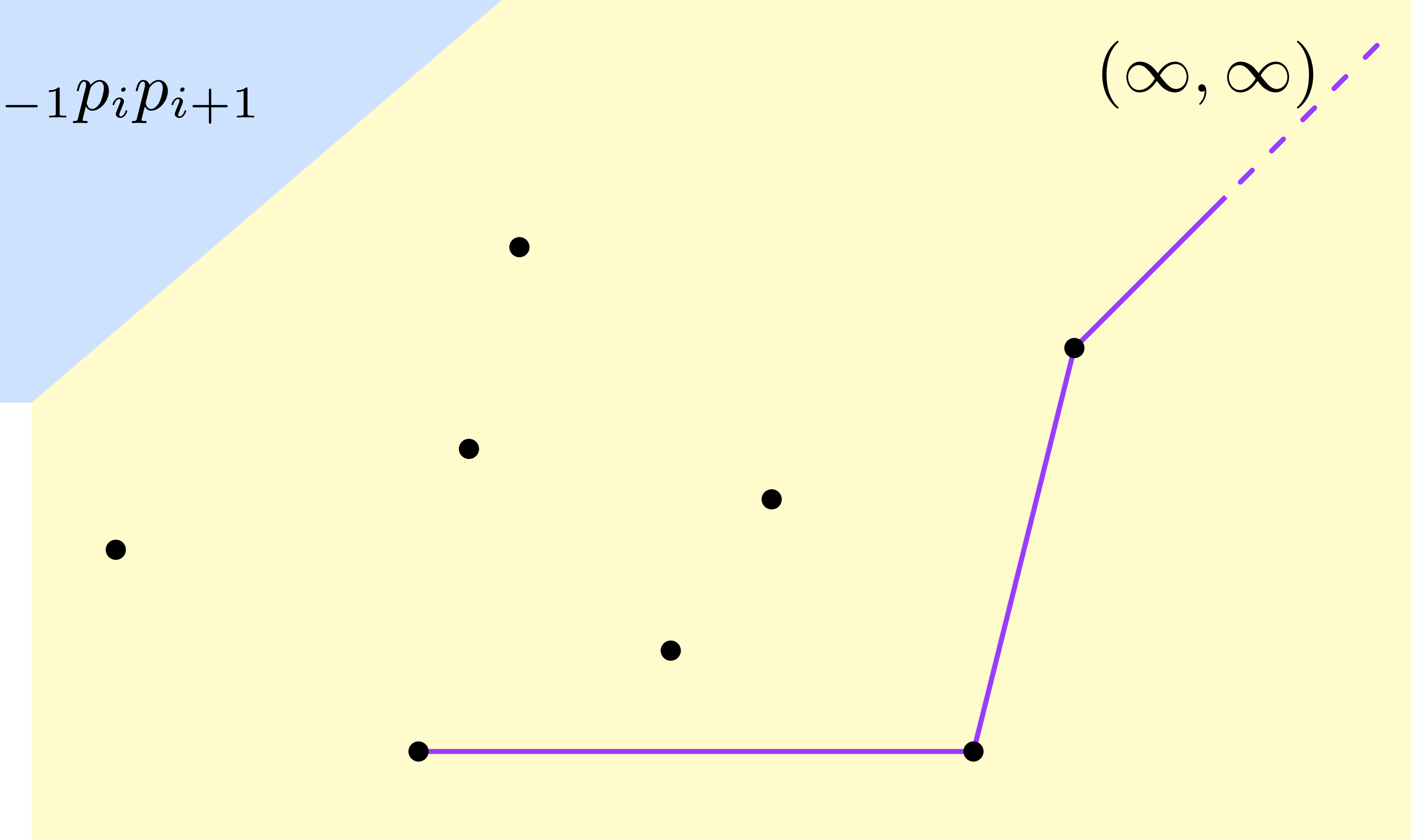
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1}p_ip_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



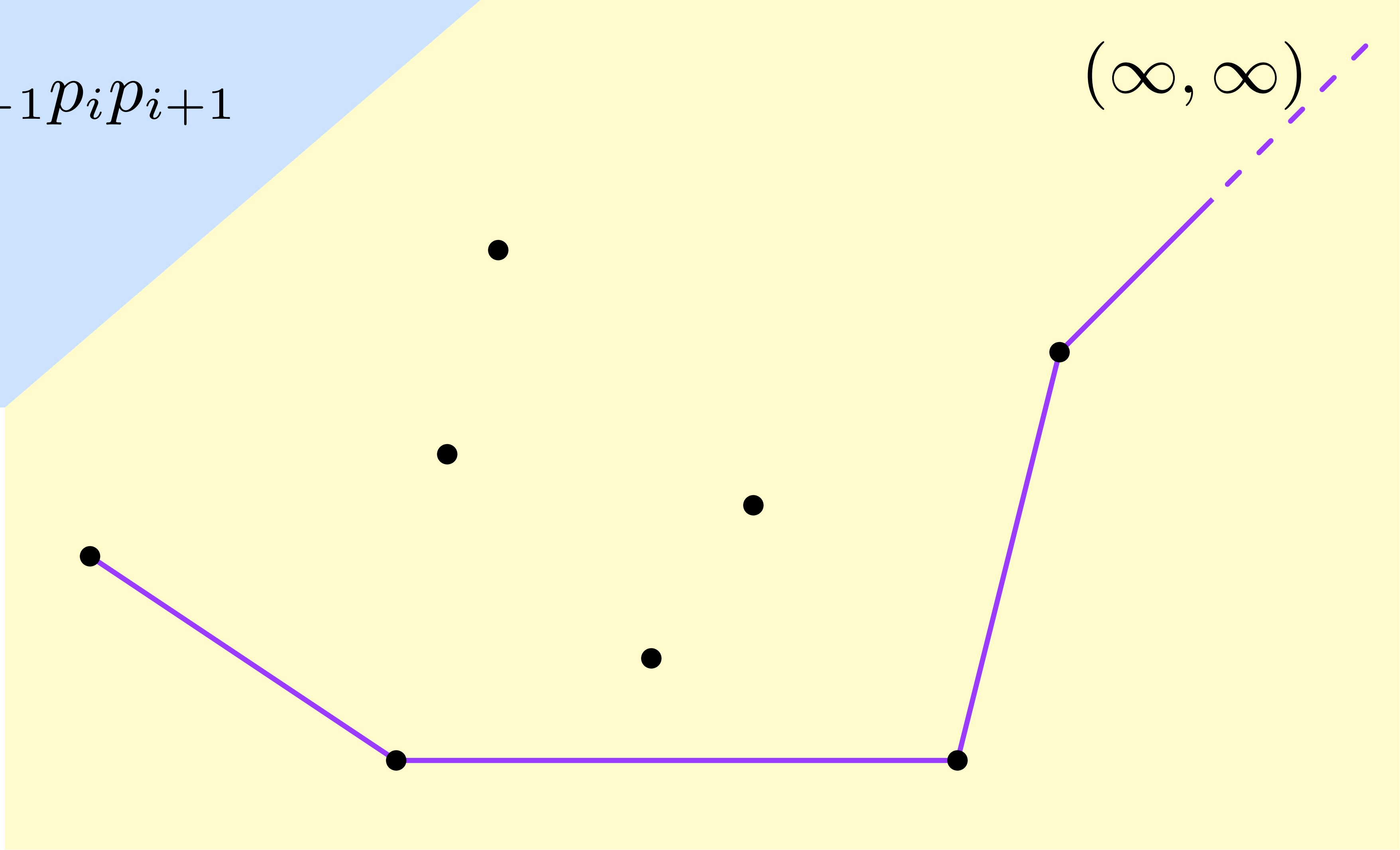
An output-sensitive algorithm

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



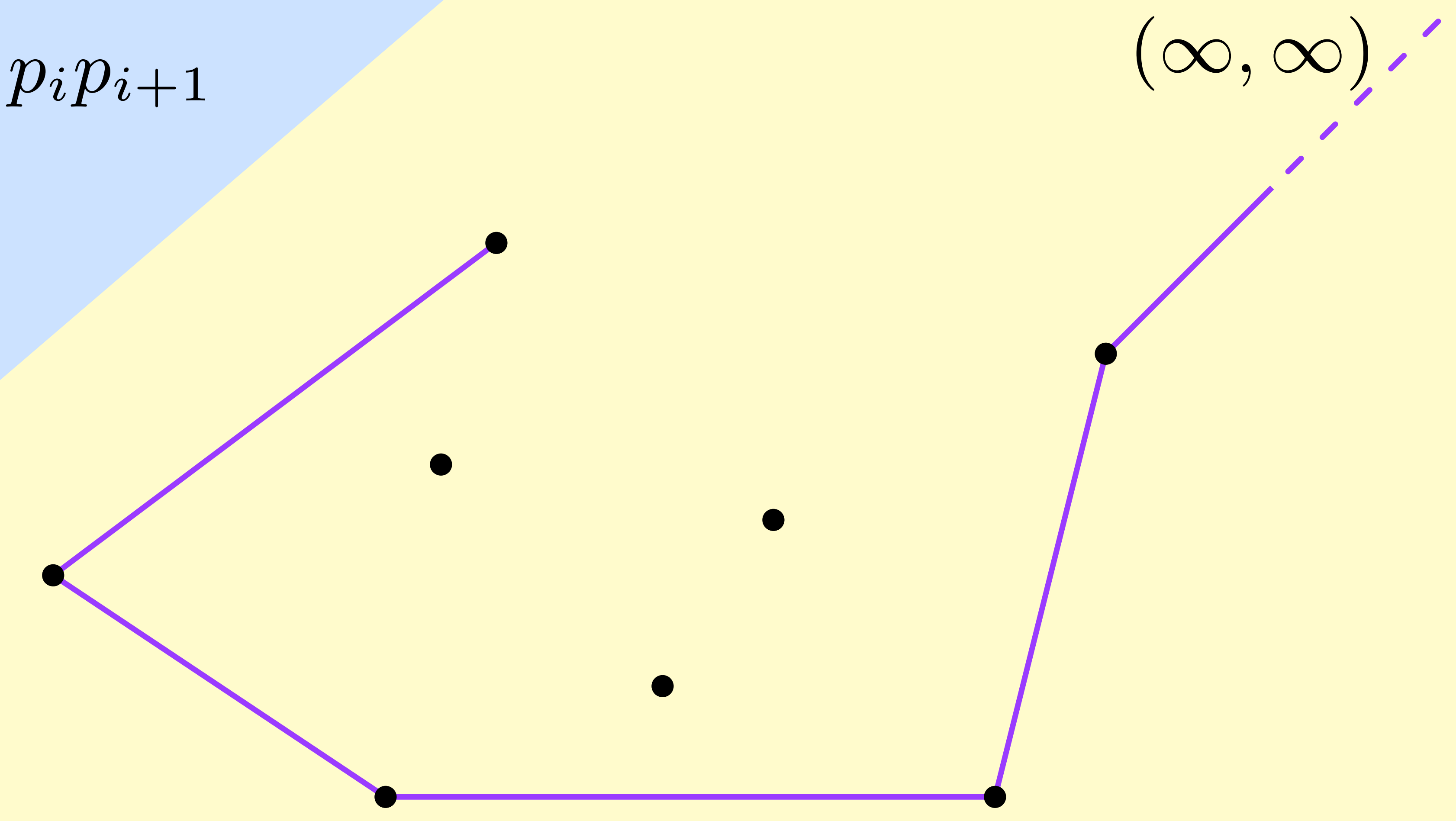
An output-sensitive algorithm

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

```
p_0 ← (∞, ∞), p_1 ← right-most vertex in P, insert p_1 into L
while true do
    choose p_{i+1} ∈ P maximizing ∠p_{i-1} p_i p_{i+1}
    if p_{i+1} = p_1 then
        break
    insert p_{i+1} into L
```



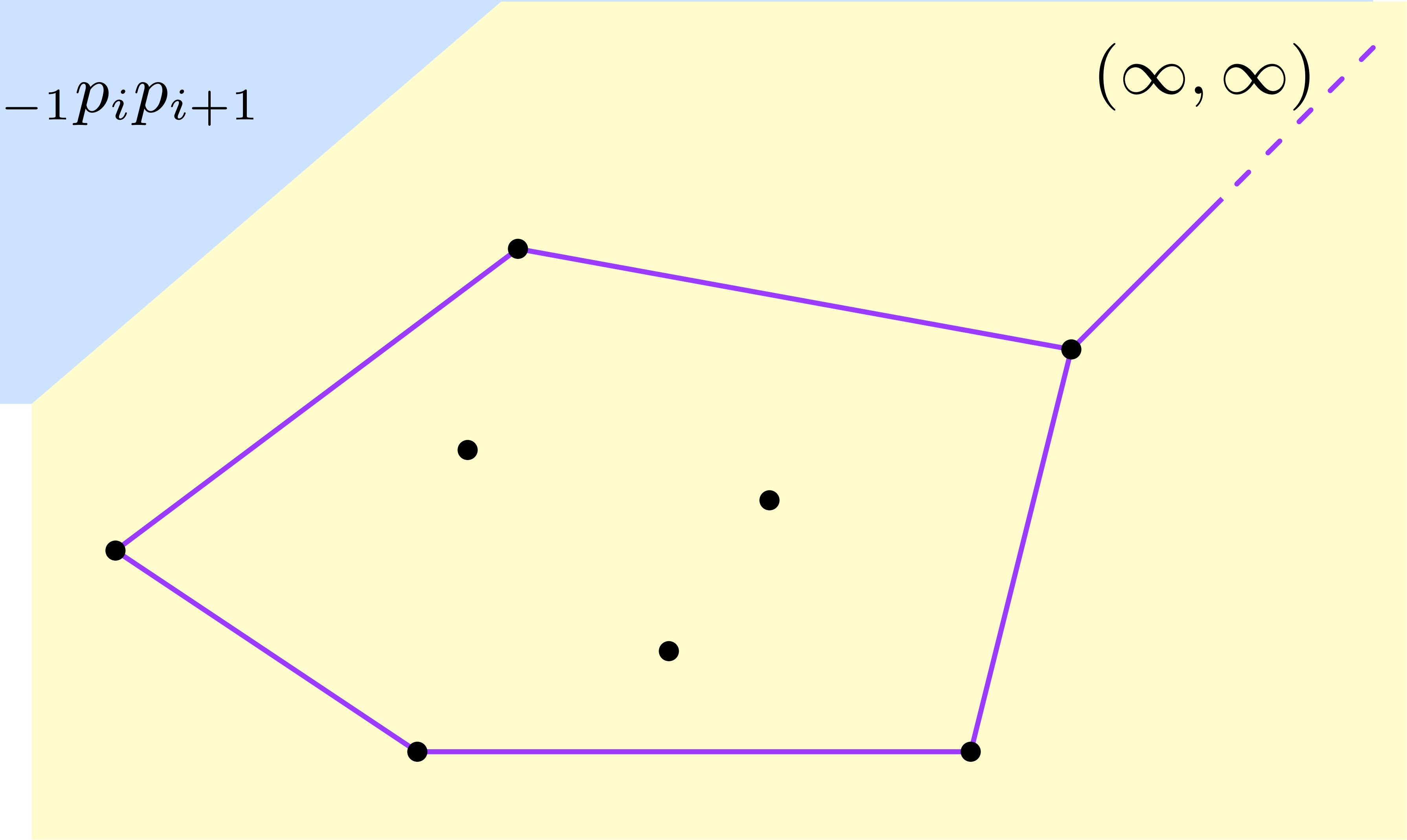
An output-sensitive algorithm

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3:     choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4:     **if** $p_{i+1} = p_1$ **then**
5:         **break**
6:     insert p_{i+1} into L



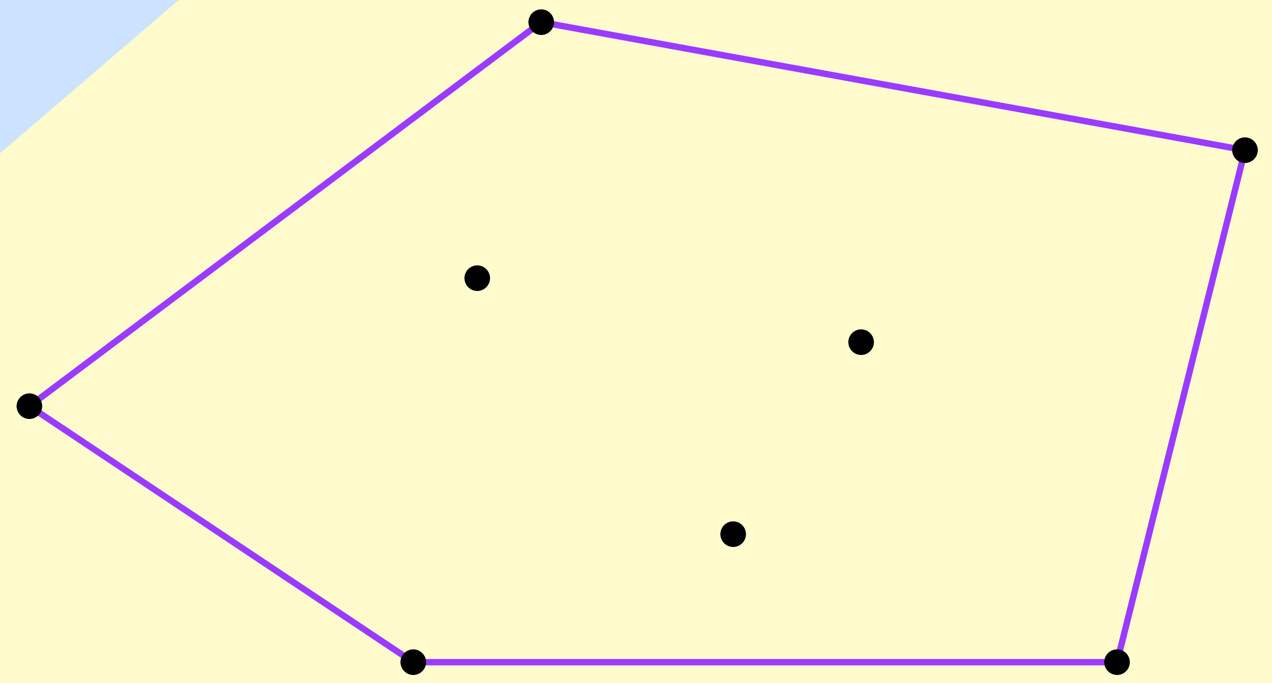
An output-sensitive algorithm

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L



Gift wrapping algorithm: correctness

Algorithm GiftWrapping(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L

Gift wrapping algorithm: correctness

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L

Loop invariant: $p_1, \ldots, p_i$ are the first i points of $CH(P)$

Gift wrapping algorithm: correctness

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
while $true$ **do**
    choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
    if $p_{i+1} = p_1$ **then**
        break
    insert p_{i+1} into L

Loop invariant: $p_1, \ldots, p_i$ are the first i points of $CH(P)$

- initialization: p_1 is on the convex hull

Gift wrapping algorithm: correctness

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1}p_ip_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L

Loop invariant: $p_1, \ldots, p_i$ are the first i points of $CH(P)$

- initialization: p_1 is on the convex hull
- maintenance: by construction, all points lie to the right of the line p_i and p_{i+1}

Gift wrapping algorithm: efficiency

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L
2: **while** $true$ **do**
3:     choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4:     **if** $p_{i+1} = p_1$ **then**
5:         **break**
6:     insert p_{i+1} into L

Gift wrapping algorithm: efficiency

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

1: $p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L $O(n)$
2: **while** $true$ **do**
3: choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$
4: **if** $p_{i+1} = p_1$ **then**
5: **break**
6: insert p_{i+1} into L

Gift wrapping algorithm: efficiency

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

```
p_0 ← (∞, ∞), p_1 ← right-most vertex in P, insert p_1 into L        O(n)
while true do
    choose p_{i+1} ∈ P maximizing ∠p_{i-1} p_i p_{i+1}              O(n)
    if p_{i+1} = p_1 then
        break
    insert p_{i+1} into L
```

Gift wrapping algorithm: efficiency

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L	$O(n)$
while $true$ **do**	
choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$	$O(n)$
if $p_{i+1} = p_1$ **then**	$O(1)$
break	
insert p_{i+1} into L	$O(1)$

Gift wrapping algorithm: efficiency

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L — $O(n)$

while $true$ **do**

  choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$ — $O(n)$

  if $p_{i+1} = p_1$ **then** — $O(1)$

    break

  insert p_{i+1} into L — $O(1)$

$\times h$

Gift wrapping algorithm: efficiency

Algorithm GIFTWRAPPING(P)

Input: set P of points in the plane

Output: list L containing vertices of $CH(P)$ in counterclockwise order

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most vertex in P, insert p_1 into L — $O(n)$

while $true$ **do**

  choose $p_{i+1} \in P$ maximizing $\angle p_{i-1} p_i p_{i+1}$ — $O(n)$

  if $p_{i+1} = p_1$ **then** — $O(1)$

    break

  insert p_{i+1} into L — $O(1)$

$\times h$

Total time:

$$O(n) + h \cdot (O(n) + O(1) + O(1)) = O(hn)$$

Optimality?

When should we choose which algorithm?

Optimality?

When should we choose which algorithm?

- many points on $CH(P)$: $O(n \log n)$ Graham Scan
- few points on $CH(P)$: $O(nh)$ Gift Wrapping

Optimality?

When should we choose which algorithm?

- many points on $CH(P)$: $O(n \log n)$ Graham Scan
- few points on $CH(P)$: $O(nh)$ Gift Wrapping

Can we do better?

Optimality?

When should we choose which algorithm?

- many points on $CH(P)$: $O(n \log n)$ Graham Scan
- few points on $CH(P)$: $O(nh)$ Gift Wrapping

Can we do better? yes, by combining both algorithms

Chan’s Algorithms

Chan's Algorithms

Chan's Algorithms

Suppose we know h:

Chan's Algorithms

Suppose we know h:

Algorithm ChanHull(P,h)

```
partition P into sets P_i with h points each
for i ← 1 to n/h do
    L_i ← GrahamScan(P_i)
p_0 ← (∞, ∞), p_1 ← right-most point in P, insert p_1 into L
for j ← 1 to h − 1 do
    for i ← 1 to n/h do
        find point q_i ∈ L_i maximizing ∠p_{j−1} p_j q_i
    p_{j+1} ← max_∠{q_1, ..., q_{n/h}}
    insert p_{j+1} into L
return L
```

Chan's Algorithms

Suppose we know h:

Algorithm CHANHULL(P,h)

partition P into sets P_i with h points each
for $i \leftarrow 1$ to n/h **do**
    $L_i \leftarrow$ GRAHAMSCAN(P_i)

Graham Scan

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L
for $j \leftarrow 1$ to $h - 1$ **do**
    for $i \leftarrow 1$ to n/h **do**
        find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$
    $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/h}\}$
    insert p_{j+1} into L

Gift Wrapping

return L

Chan's Algorithms

Suppose we know h:

Algorithm ChanHull(P,h)

```
partition P into sets P_i with h points each
for i ← 1 to n/h do
    L_i ← GrahamScan(P_i)                         O(h log h)
p_0 ← (∞, ∞), p_1 ← right-most point in P, insert p_1 into L
for j ← 1 to h − 1 do
    for i ← 1 to n/h do
        find point q_i ∈ L_i maximizing ∠p_{j−1} p_j q_i
    p_{j+1} ← max_∠{q_1, ..., q_{n/h}}
    insert p_{j+1} into L
return L
```

Chan's Algorithms

Suppose we know h:

Algorithm CHANHULL(P,h)

partition P into sets P_i with h points each

for $i \leftarrow 1$ to n/h **do**

  $L_i \leftarrow$ GRAHAMSCAN(P_i) $\quad O(h \log h) \} \times n/h$

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L

for $j \leftarrow 1$ to $h - 1$ **do**

  for $i \leftarrow 1$ to n/h **do**

    find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$

  $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/h}\}$

  insert p_{j+1} into L

return L

Chan's Algorithms

Suppose we know h:

Algorithm ChanHull(P,h)

partition P into sets P_i with h points each
for $i \leftarrow 1$ to n/h **do**
    $L_i \leftarrow$ GrahamScan(P_i) $O(h \log h)$ $\}\times n/h$ $\} O(n \log h)$
$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L
for $j \leftarrow 1$ to $h - 1$ **do**
    for $i \leftarrow 1$ to n/h **do**
        find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$
    $p_{j+1} \leftarrow \max_\angle \{q_1, \ldots, q_{n/h}\}$
    insert p_{j+1} into L
return L

Chan's Algorithms

Suppose we know h:

Algorithm ChanHull(P,h)

```
partition P into sets P_i with h points each
for i ← 1 to n/h do
    L_i ← GrahamScan(P_i)
p_0 ← (∞, ∞), p_1 ← right-most point in P, insert p_1 into L
for j ← 1 to h − 1 do
    for i ← 1 to n/h do
        find point q_i ∈ L_i maximizing ∠p_{j−1} p_j q_i
    p_{j+1} ← max_∠{q_1, ..., q_{n/h}}
    insert p_{j+1} into L
return L
```

$O(h \log h)$ $\}\times n/h$ $\}\ O(n \log h)$

$O(\log h)$

Chan's Algorithms

Suppose we know h:

Algorithm $\textsc{ChanHull}(P, h)$

partition P into sets P_i with h points each

for $i \leftarrow 1$ to n/h **do**
  $L_i \leftarrow \textsc{GrahamScan}(P_i)$ $O(h \log h)$ $\}\times n/h$ $\}\ O(n \log h)$

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L

for $j \leftarrow 1$ to $h - 1$ **do**
  for $i \leftarrow 1$ to n/h **do**
    find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$ $O(\log h)$
  $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/h}\}$ $O(n/h)$
  insert p_{j+1} into L $O(1)$

return L

Chan's Algorithms

Suppose we know h:

Algorithm CHANHULL(P,h)

partition P into sets P_i with h points each

for $i \leftarrow 1$ to n/h **do**

  $L_i \leftarrow$ GRAHAMSCAN(P_i) $O(h \log h)$ $\}\times n/h$ $\}\ O(n \log h)$

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L

for $j \leftarrow 1$ to $h - 1$ **do**

  for $i \leftarrow 1$ to n/h **do**

    find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$ $O(\log h)$ $\}\times O(n)$

  $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/h}\}$ $O(n/h)$

  insert p_{j+1} into L $O(1)$ $\}\times O(h)$

return L

Chan's Algorithms

Suppose we know h:

Algorithm CHANHULL(P,h)

partition P into sets P_i with h points each

for $i \leftarrow 1$ to n/h **do**

  $L_i \leftarrow$ GRAHAMSCAN(P_i) $O(h \log h)$ $\}\times n/h$ $\}$ $O(n \log h)$

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L

for $j \leftarrow 1$ to $h-1$ **do**

  for $i \leftarrow 1$ to n/h **do**

    find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$ $O(\log h)$ $\}\times O(n)$

  $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/h}\}$ $O(n/h)$

  insert p_{j+1} into L $O(1)$ $\}\times O(h)$ $\}$ $O(n \log h)$

return L

Chan’s Algorithms

Suppose we know h:

Algorithm CHANHULL(P,h)

partition P into sets P_i with h points each

for $i \leftarrow 1$ to n/h **do**

  $L_i \leftarrow$ GRAHAMSCAN(P_i) $O(h \log h)$ $\}\times n/h$ $\Big\}\, O(n \log h)$

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L

for $j \leftarrow 1$ to $h - 1$ **do**

  for $i \leftarrow 1$ to n/h **do**

    find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$ $O(\log h)$ $\}\times O(n)$

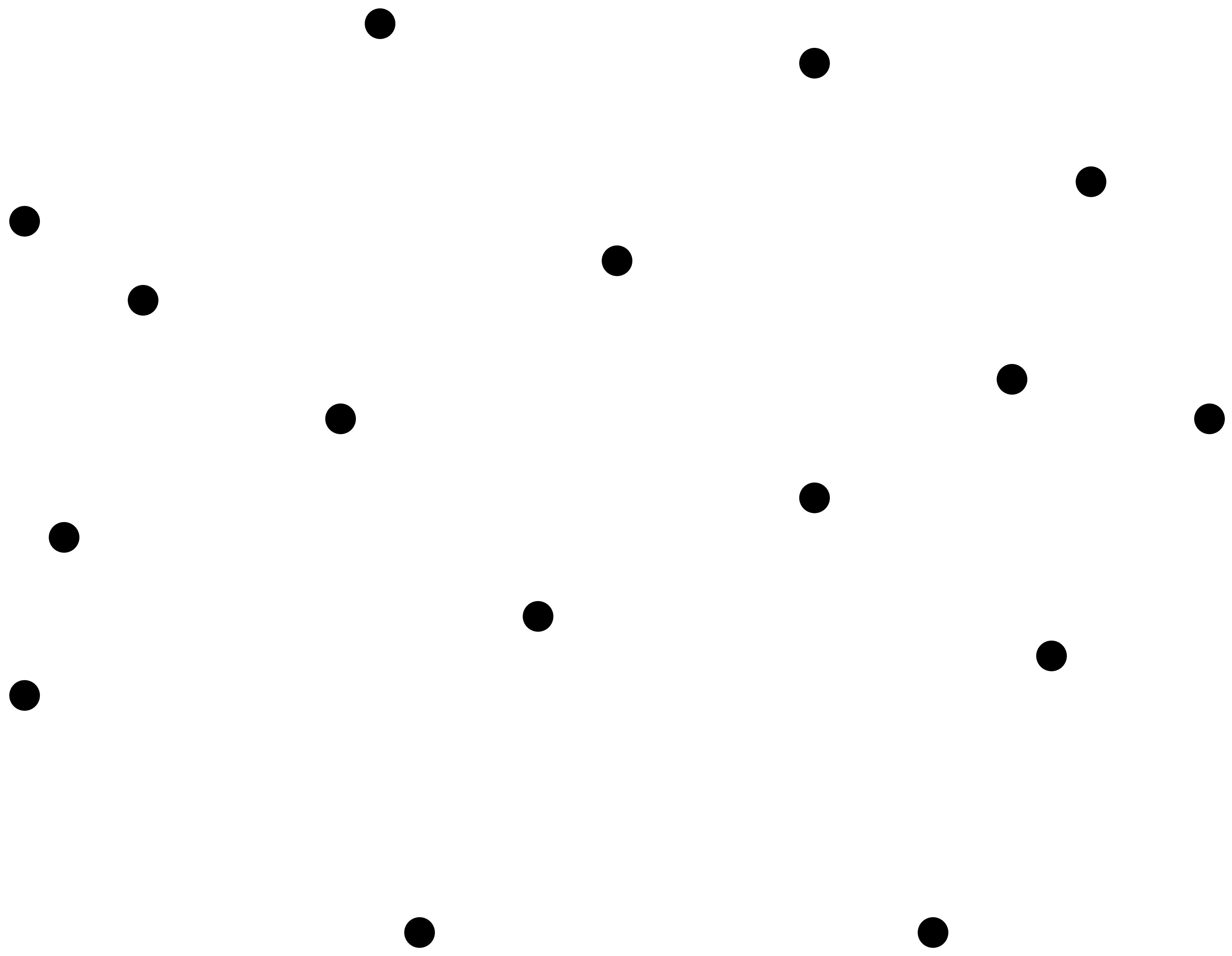
  $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/h}\}$ $O(n/h)$

  insert p_{j+1} into L $O(1)$ $\Big\}\times O(h)$ $\Big\}\, O(n \log h)$

return L

Total running time: $O(n \log h)$

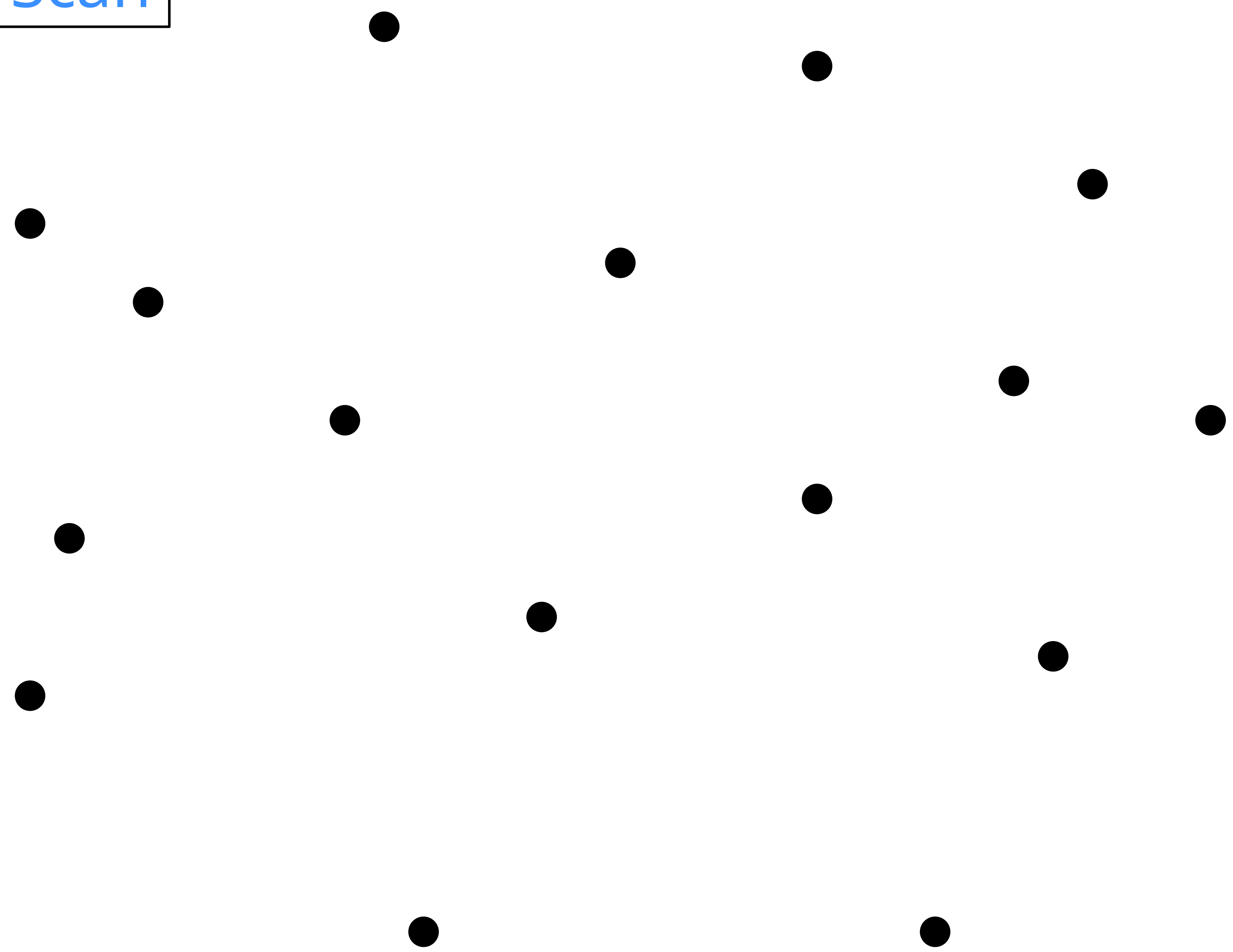
Example



$n = 16$

Example

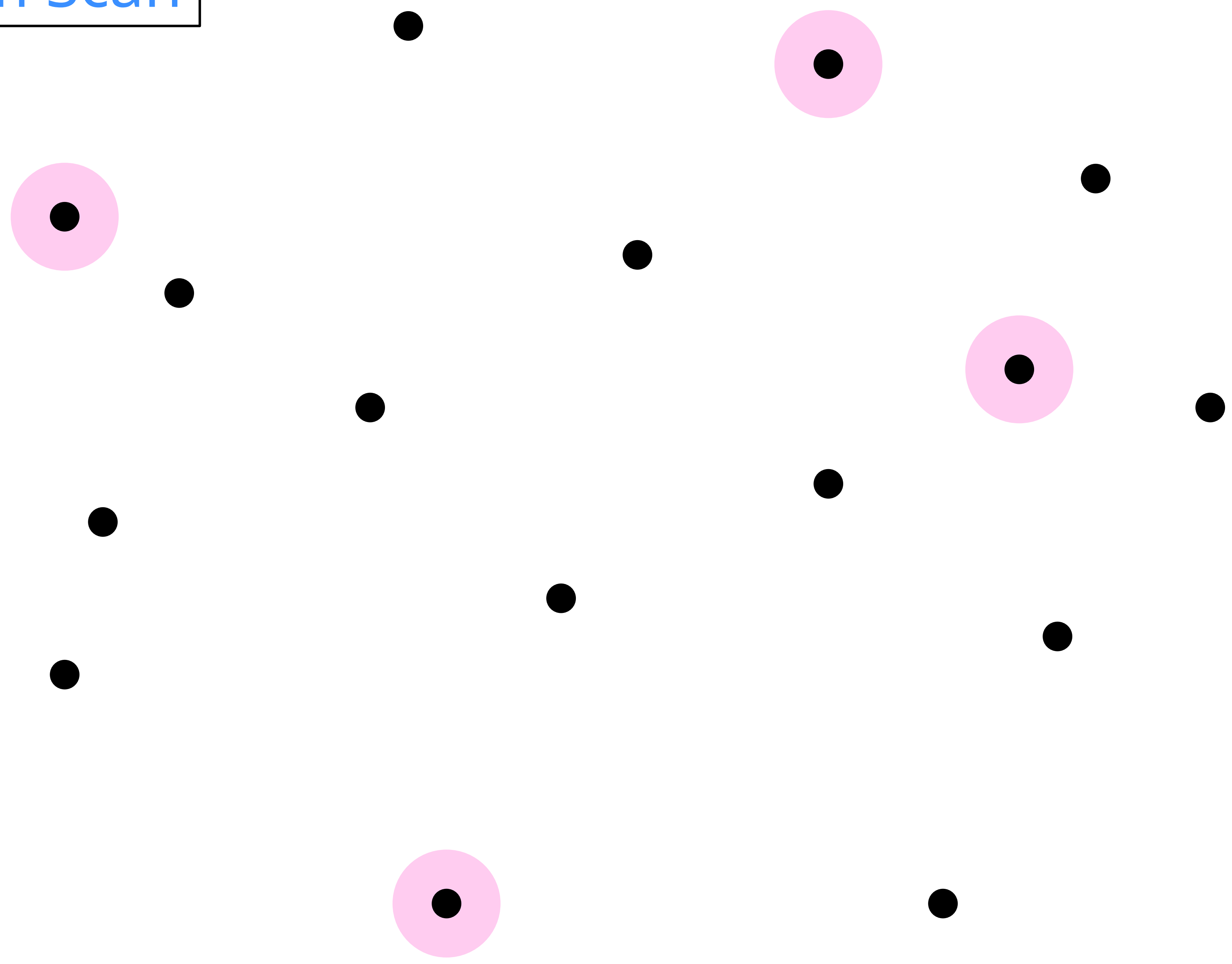
Graham Scan



$n = 16$

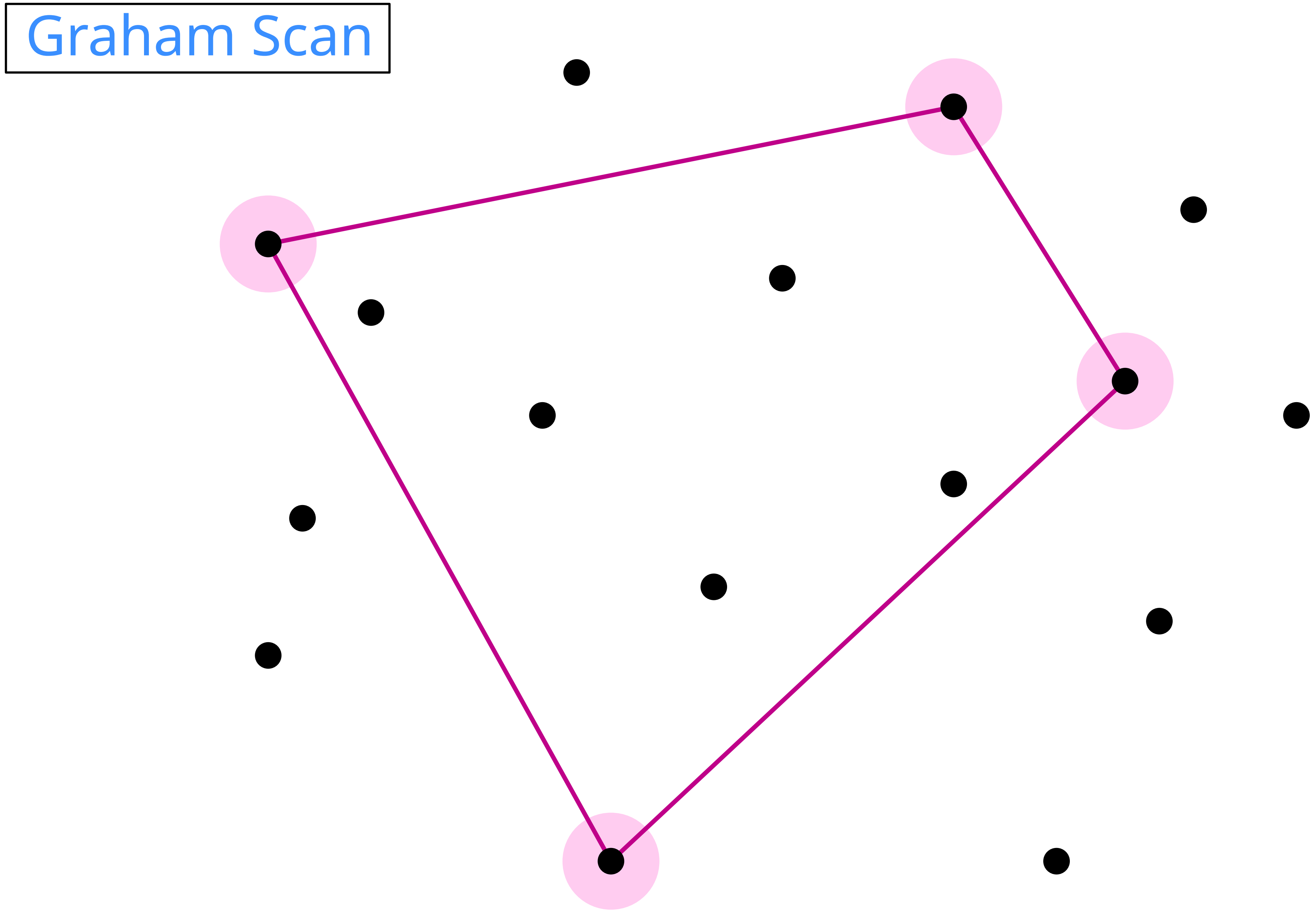
Example

Graham Scan



$n = 16$

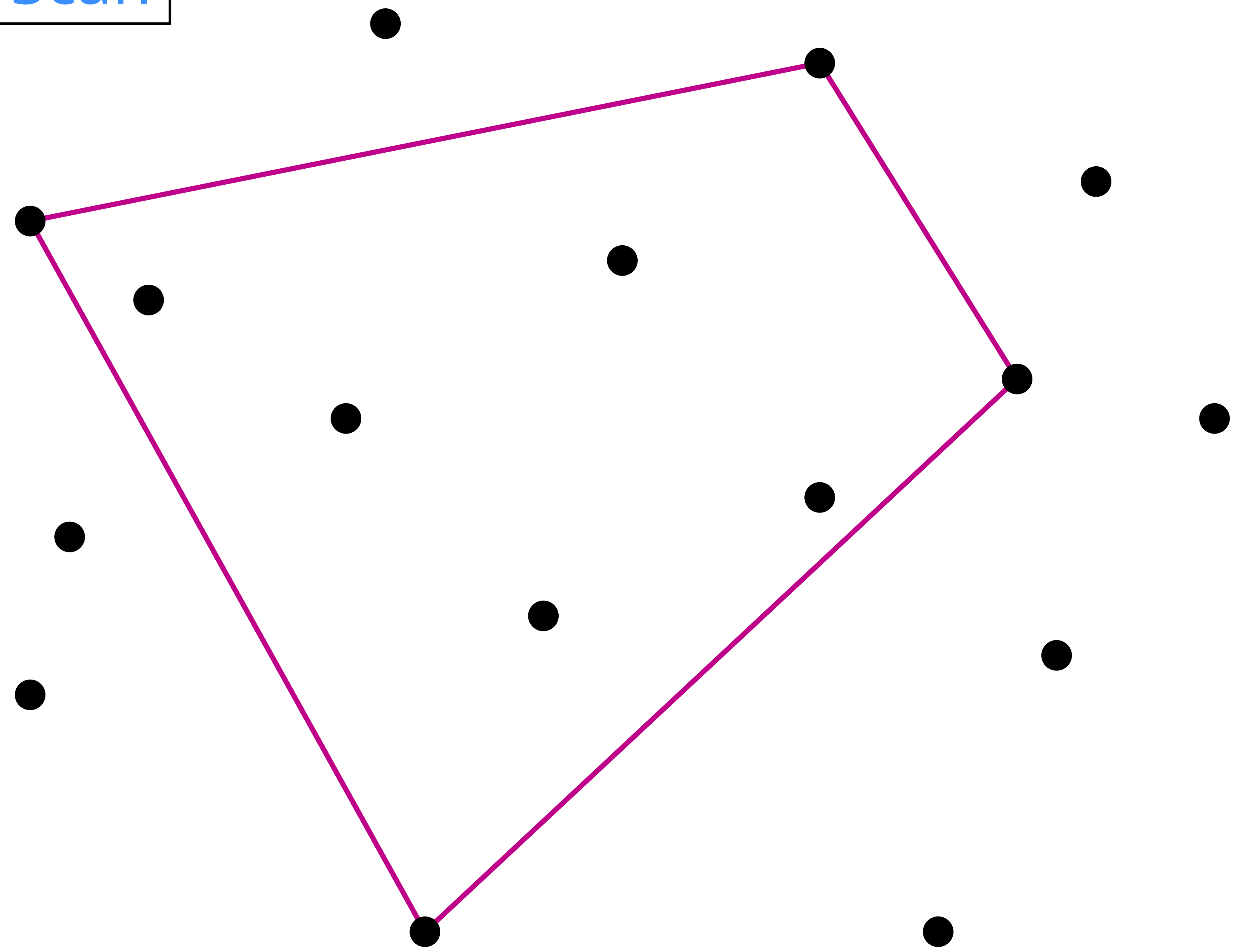
Example



$n = 16$

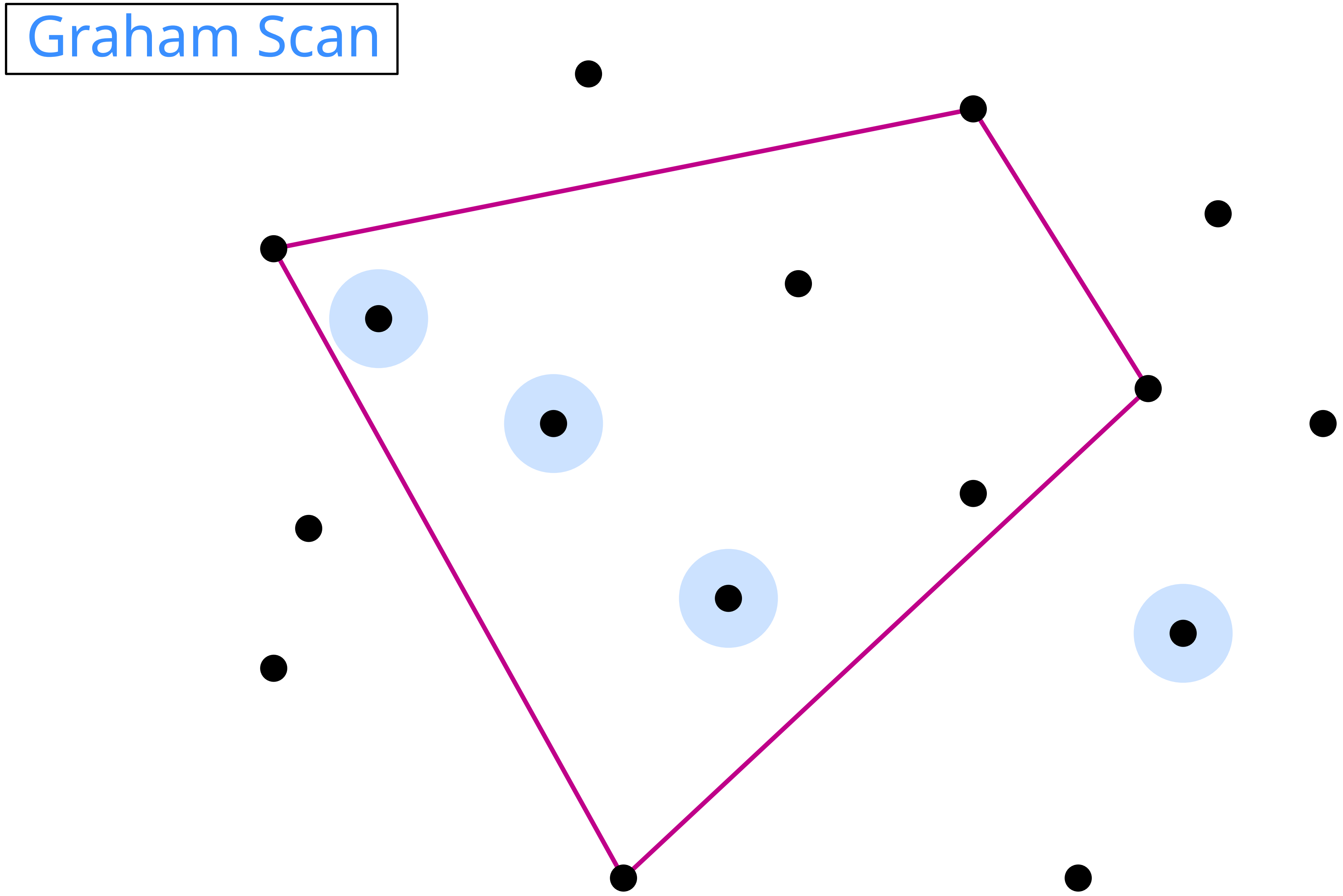
Example

Graham Scan



$n = 16$

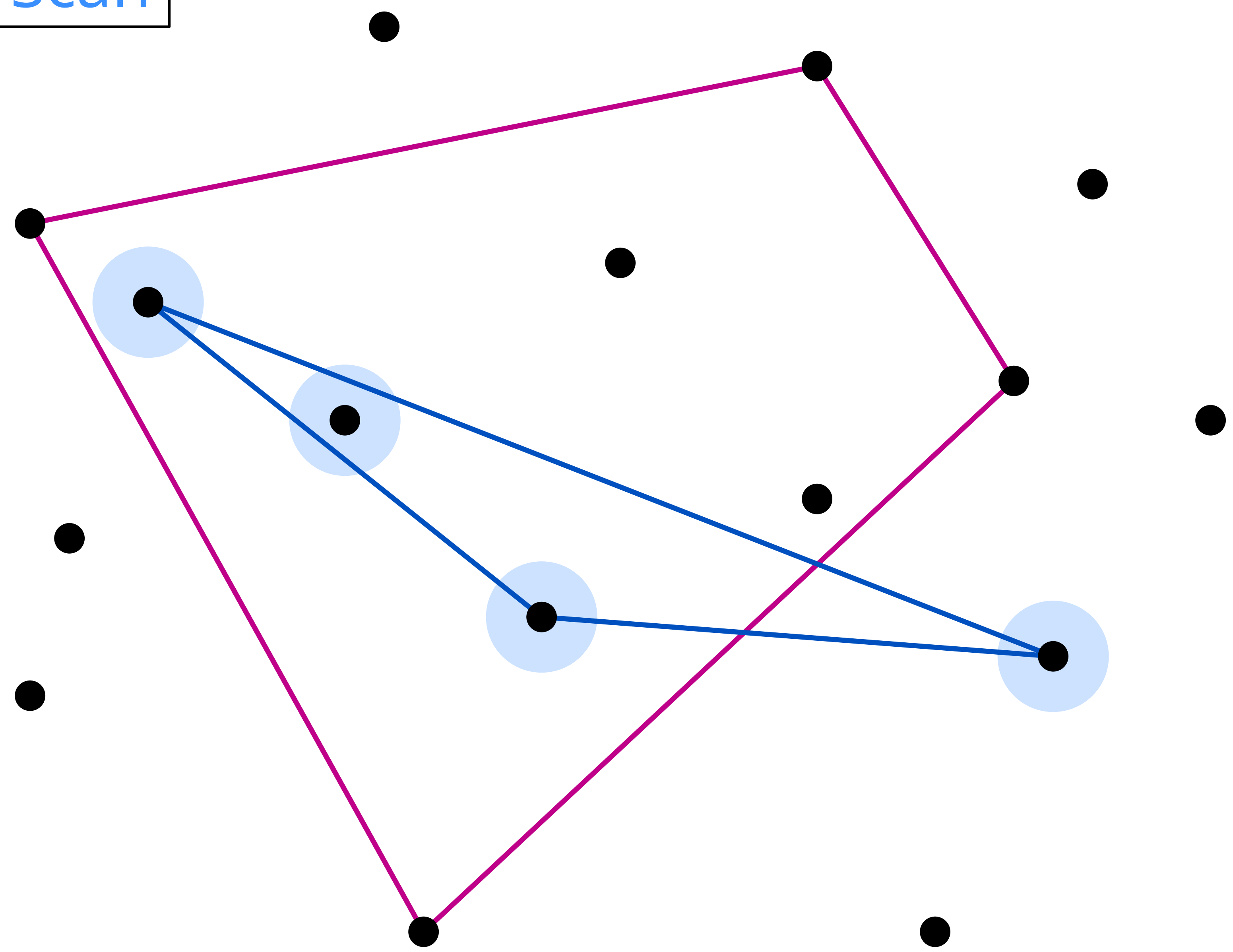
Example



$n = 16$

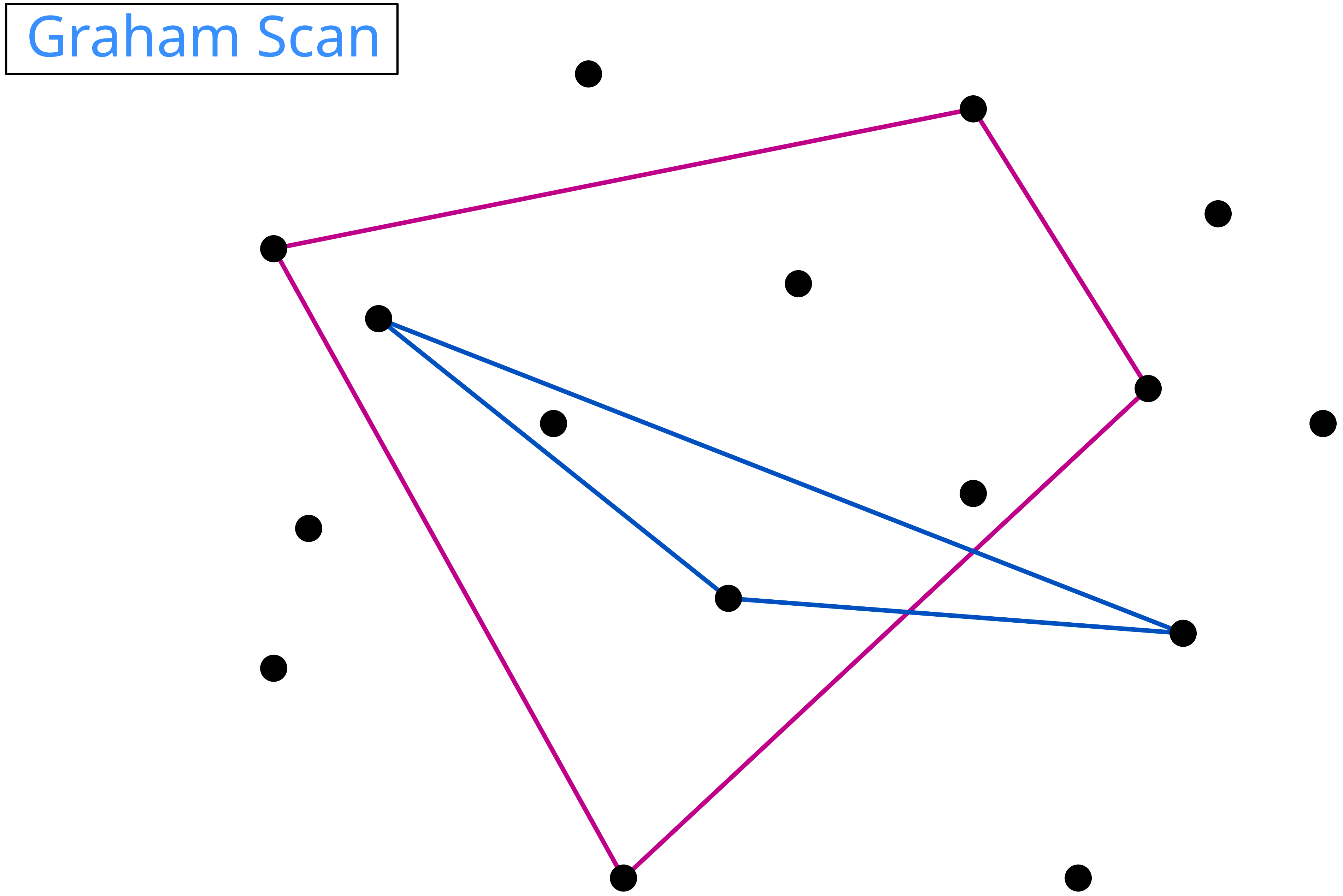
Example

Graham Scan



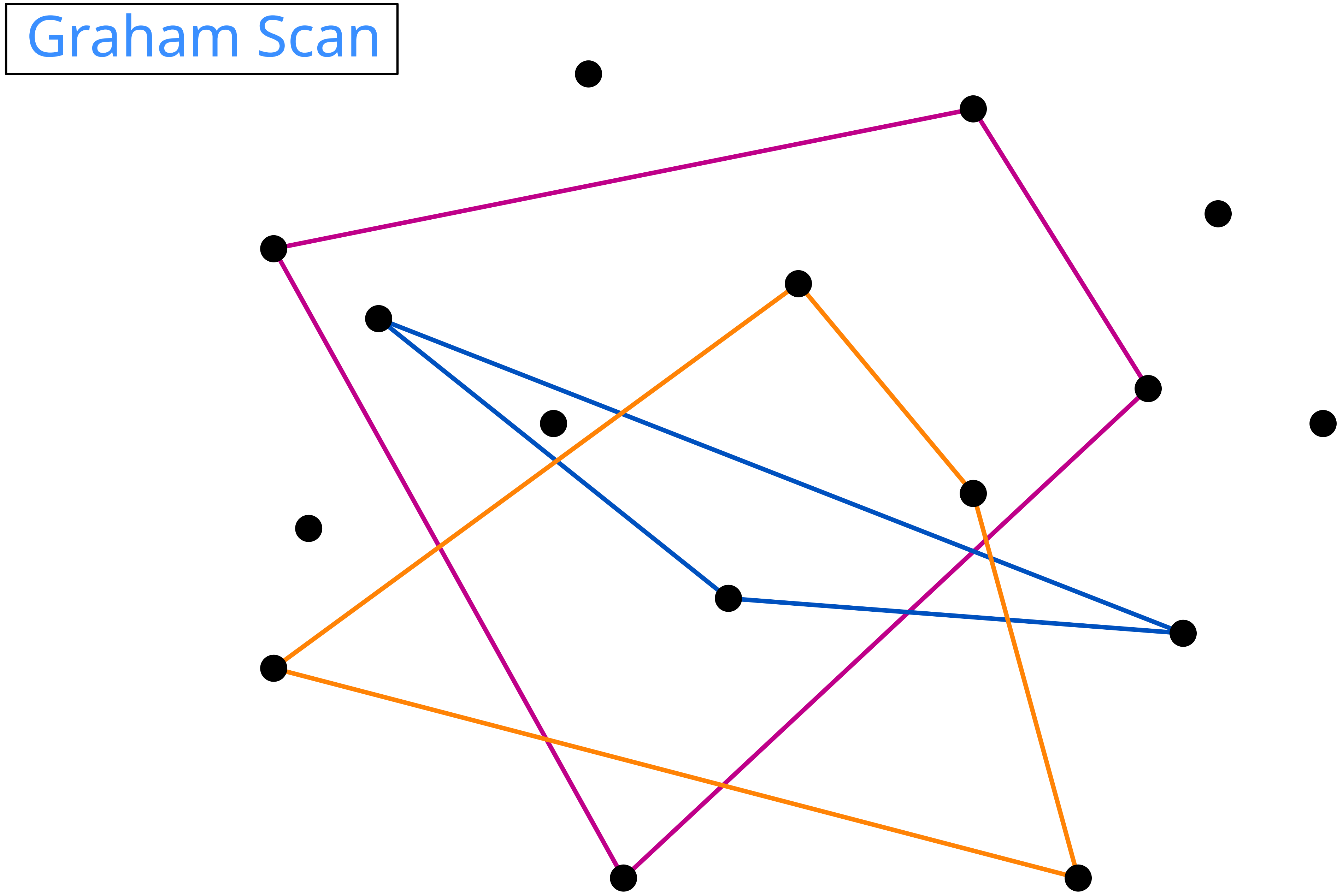
$n = 16$

Example



$n = 16$

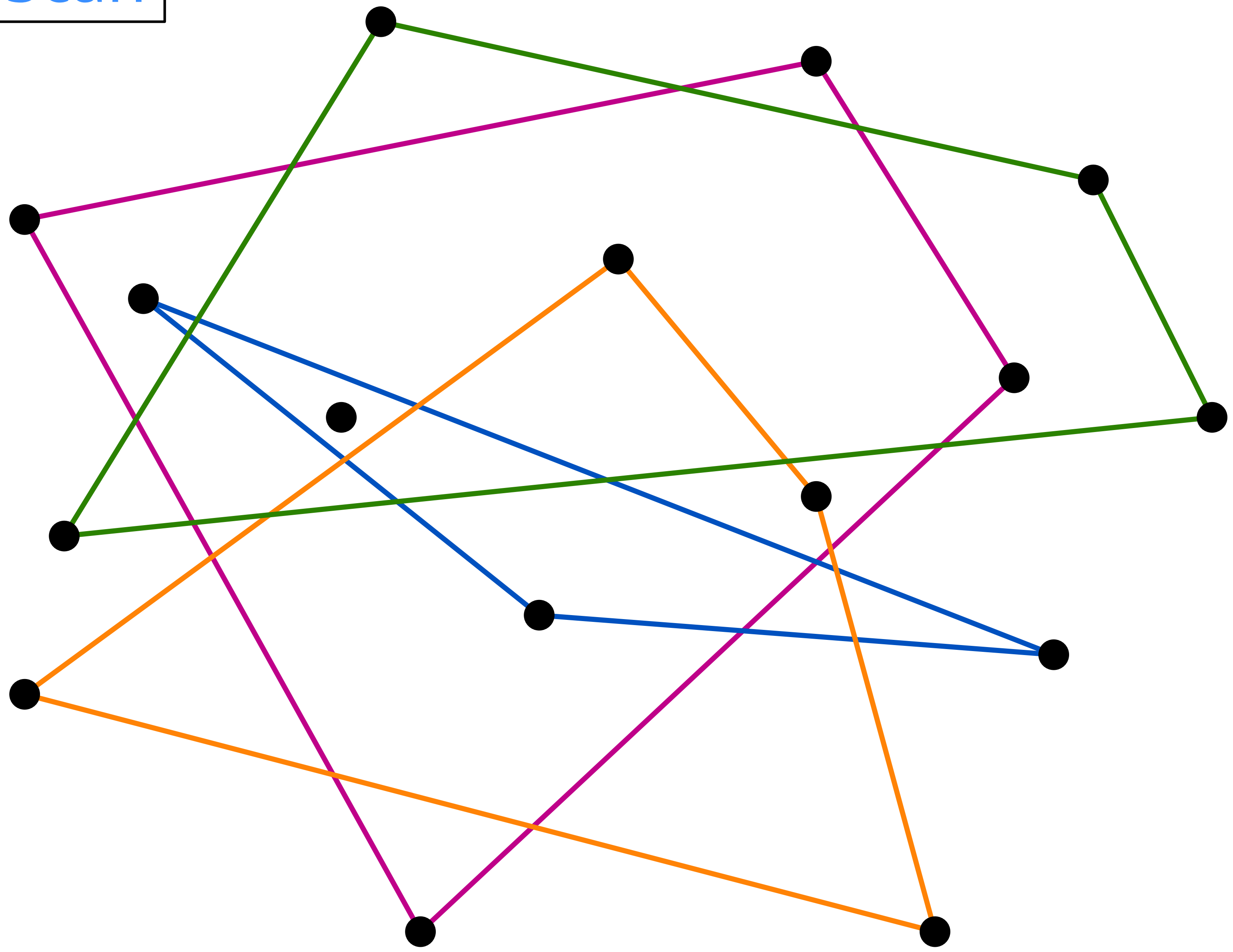
Example



$n = 16$

Example

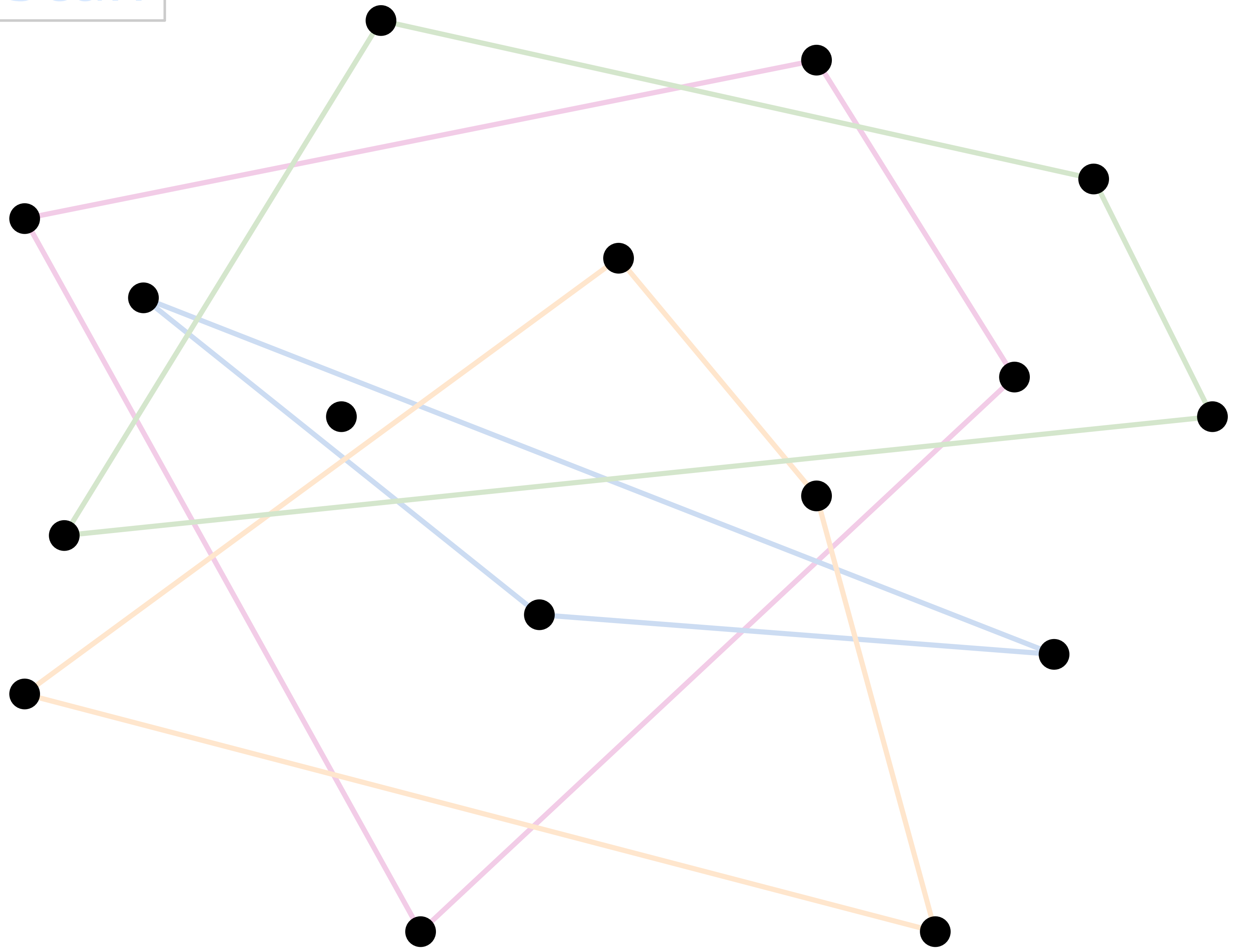
Graham Scan



$n = 16$

Example

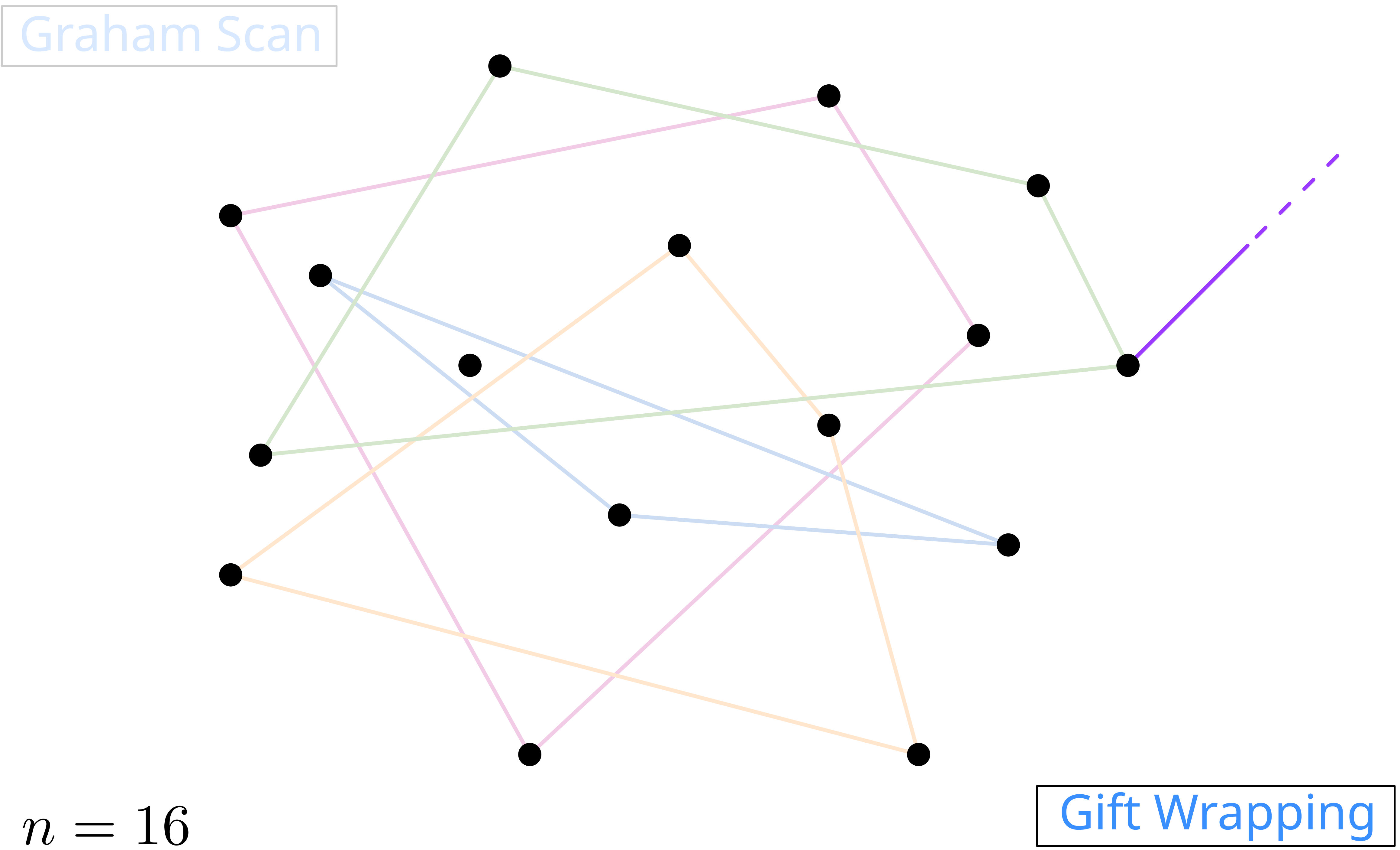
Graham Scan



$n = 16$

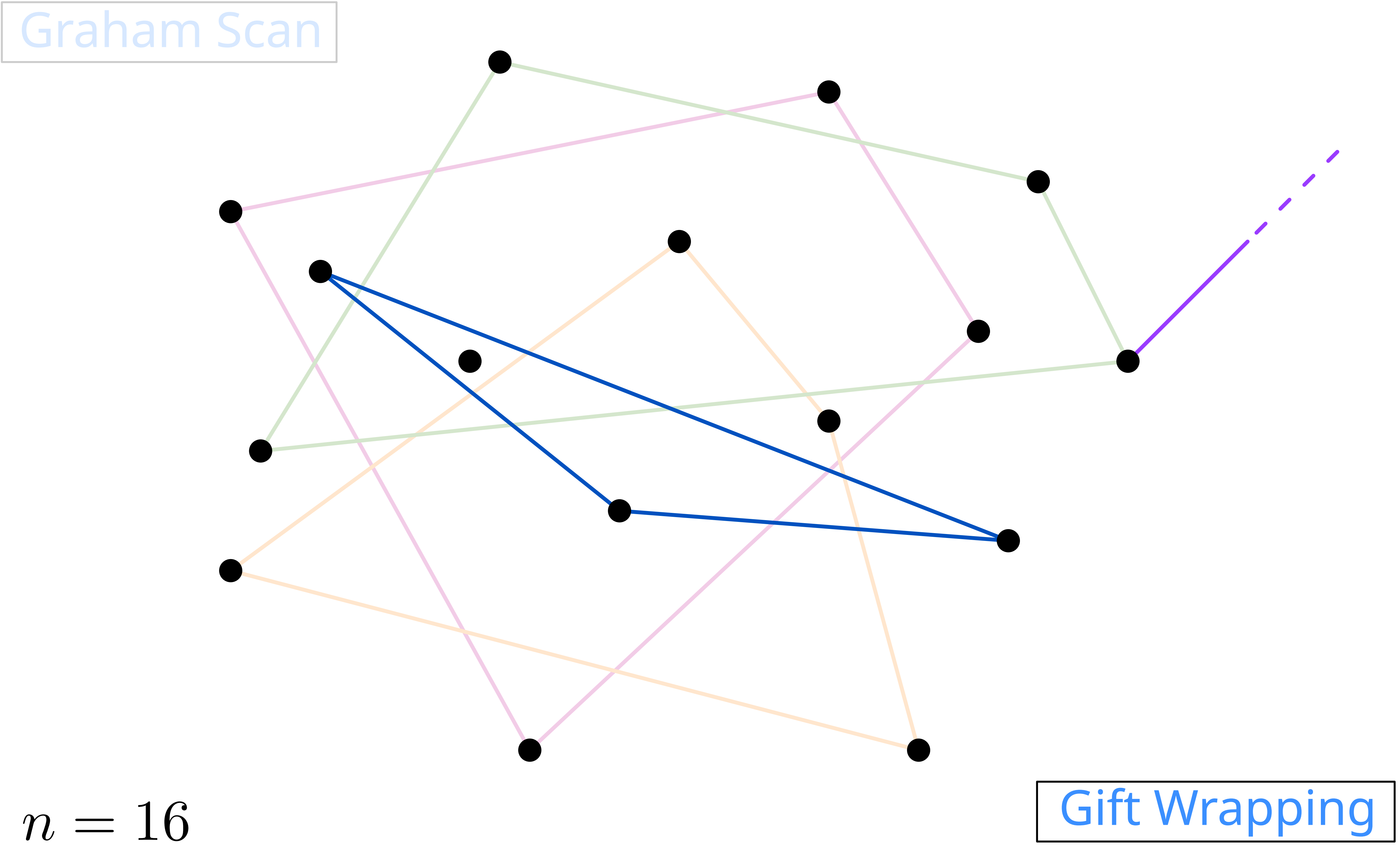
Gift Wrapping

Example



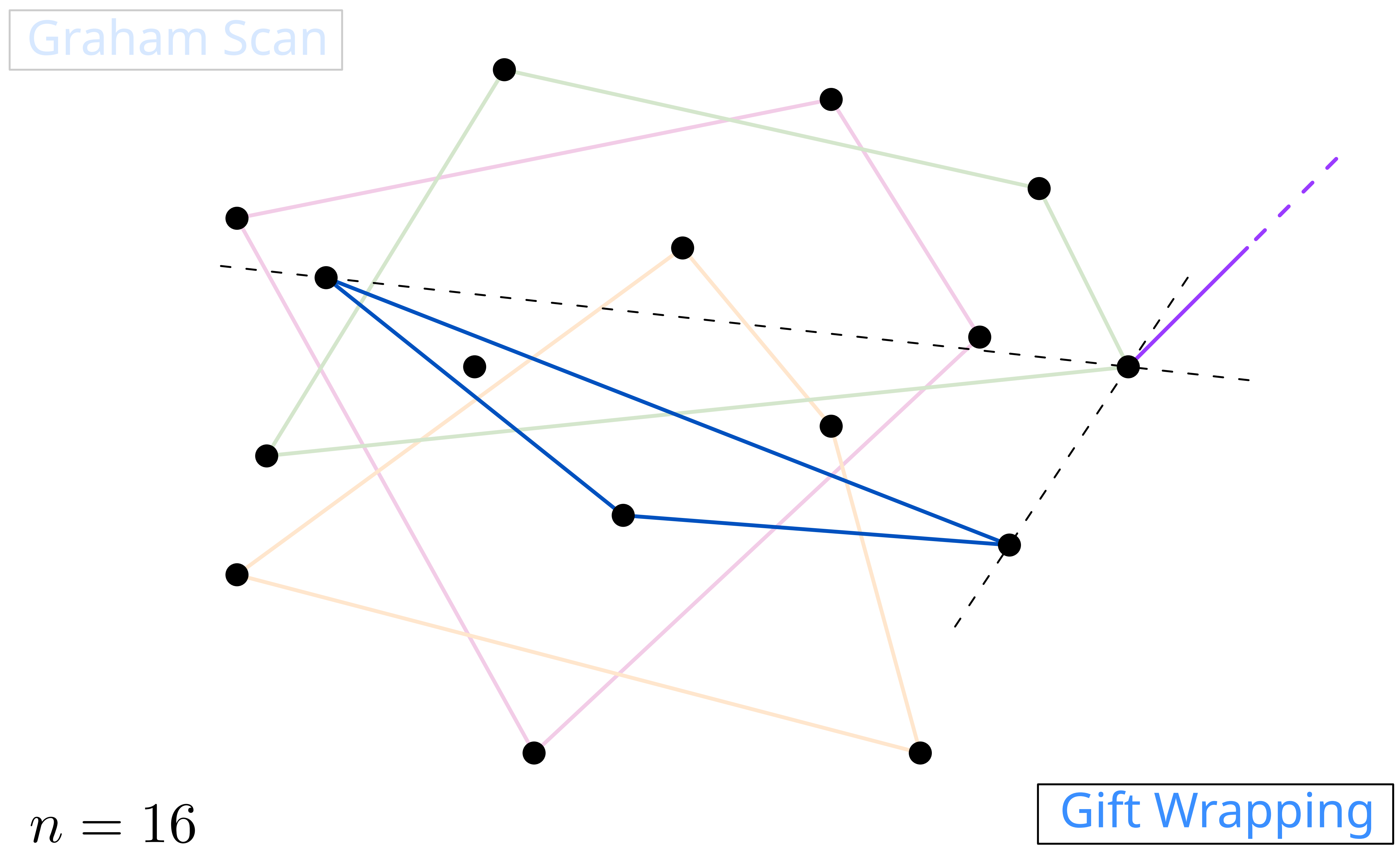
$n = 16$

Example



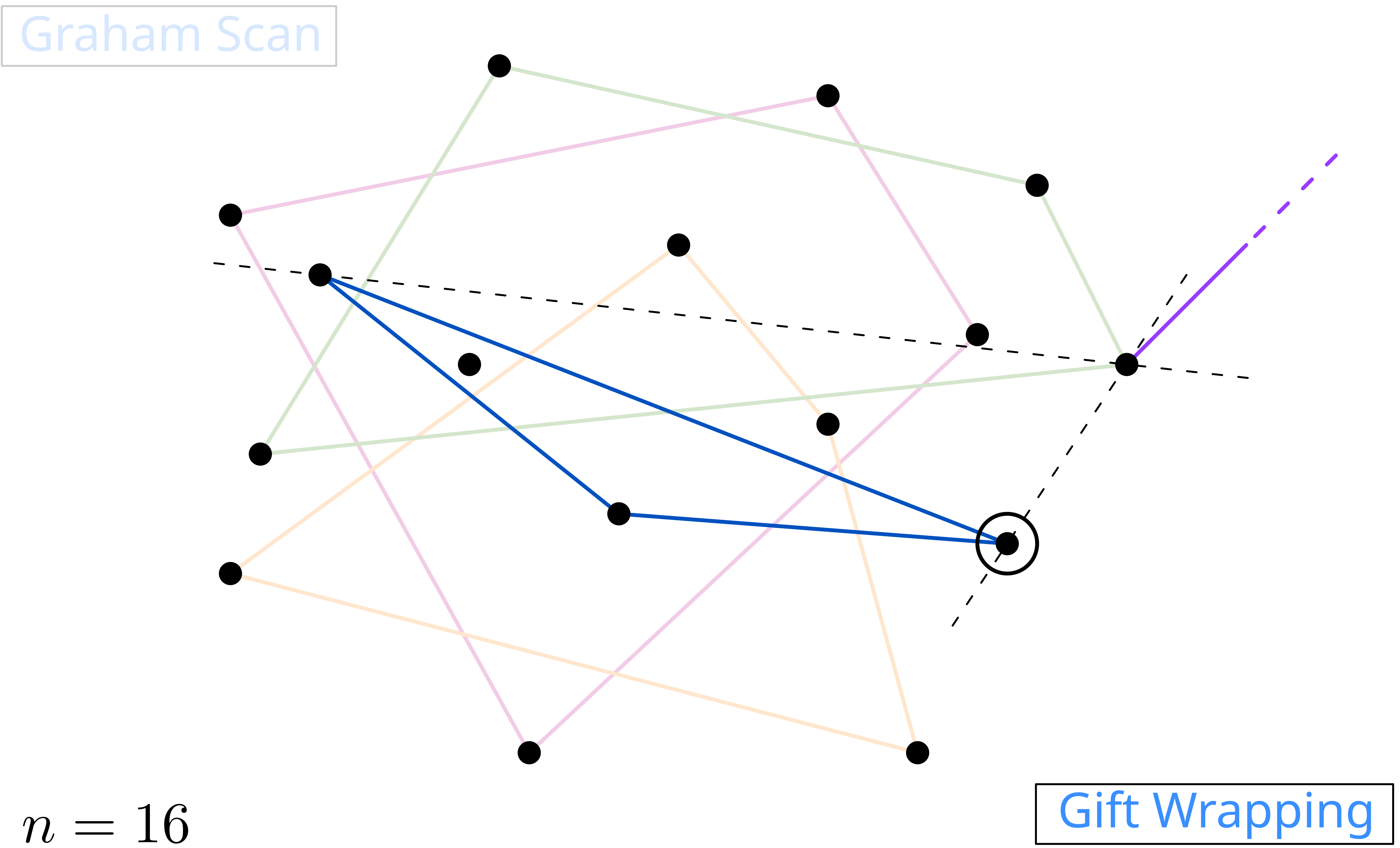
$n = 16$

Example



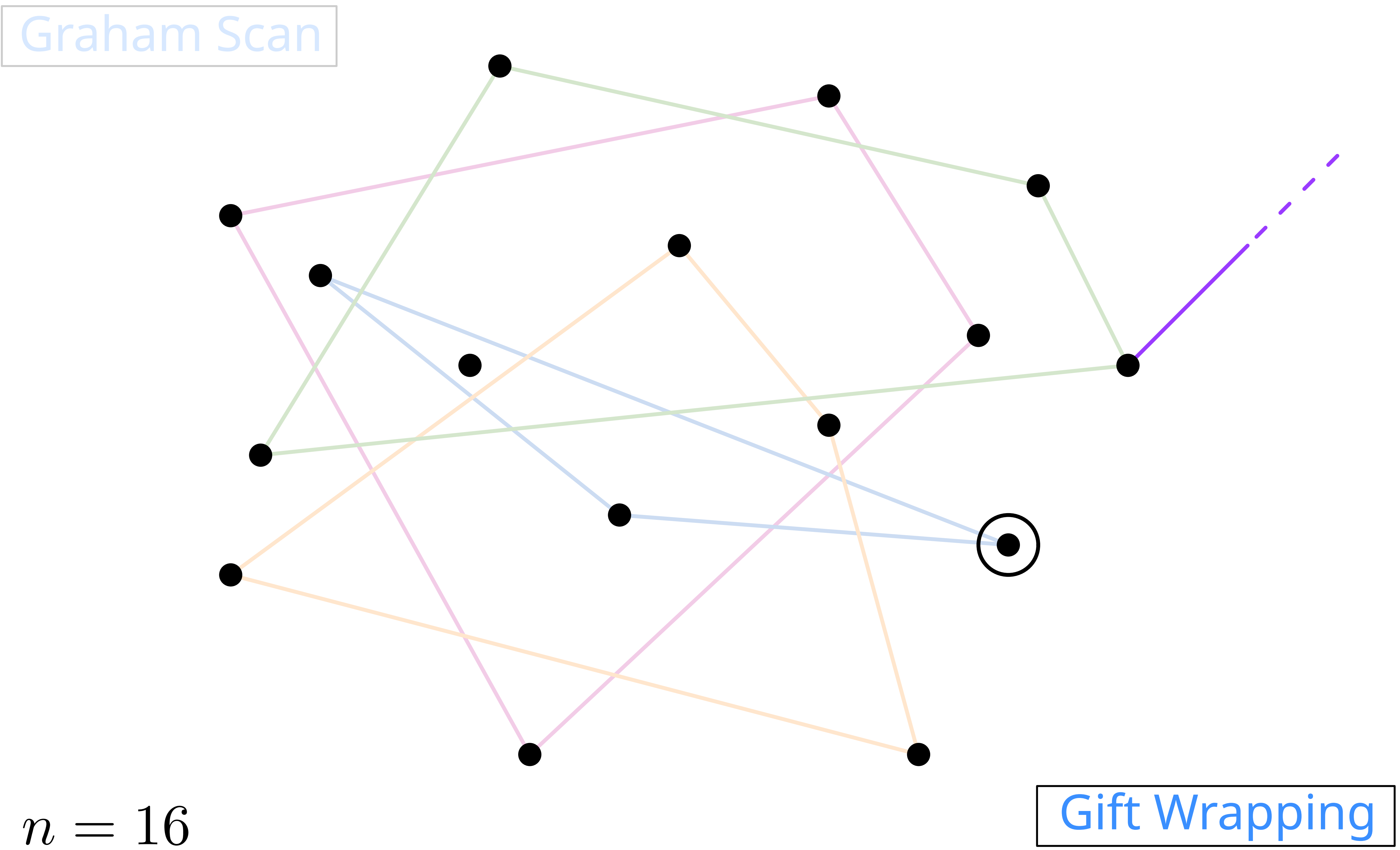
$n = 16$

Example



$n = 16$

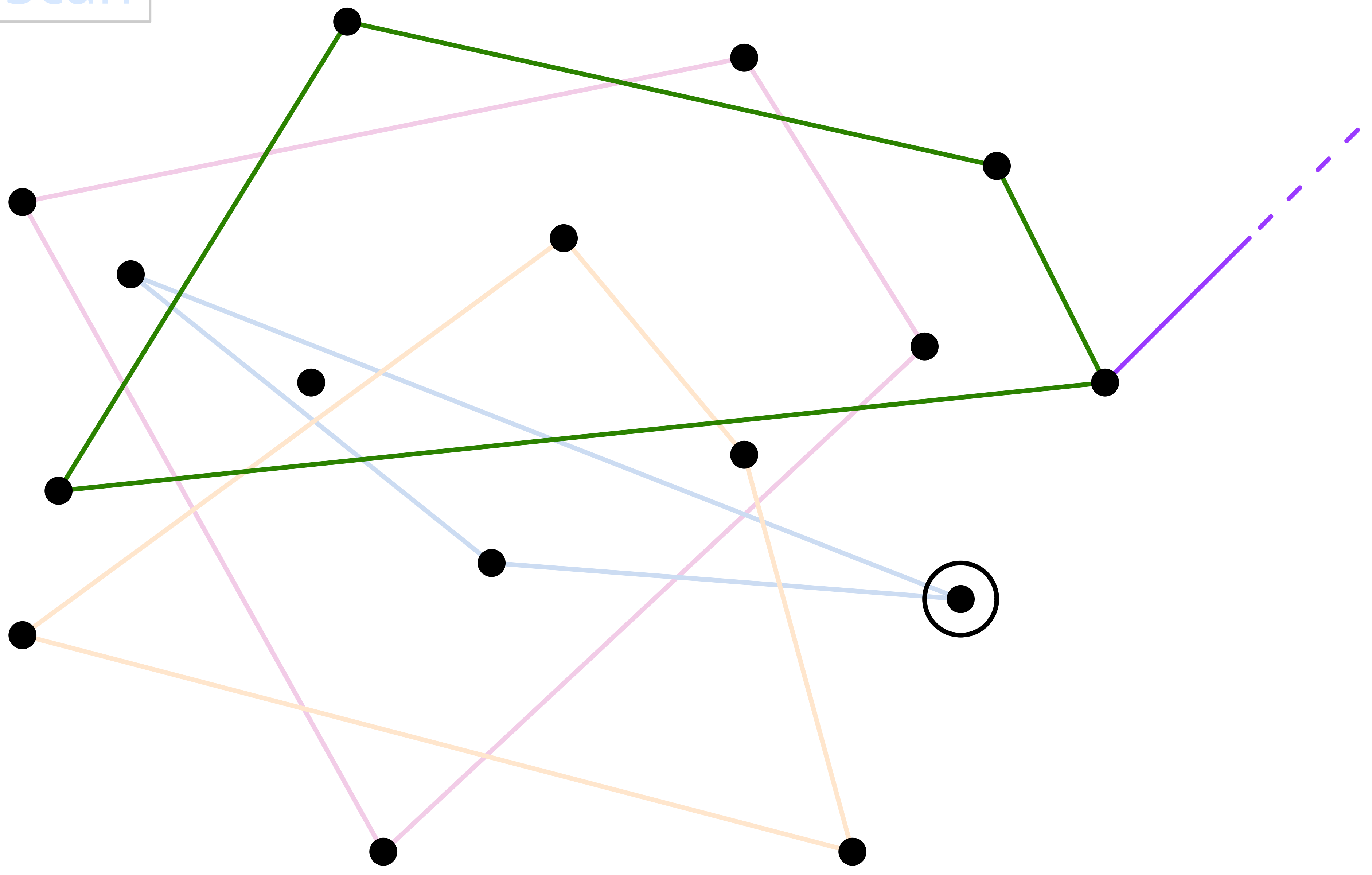
Example



$n = 16$

Example

Graham Scan

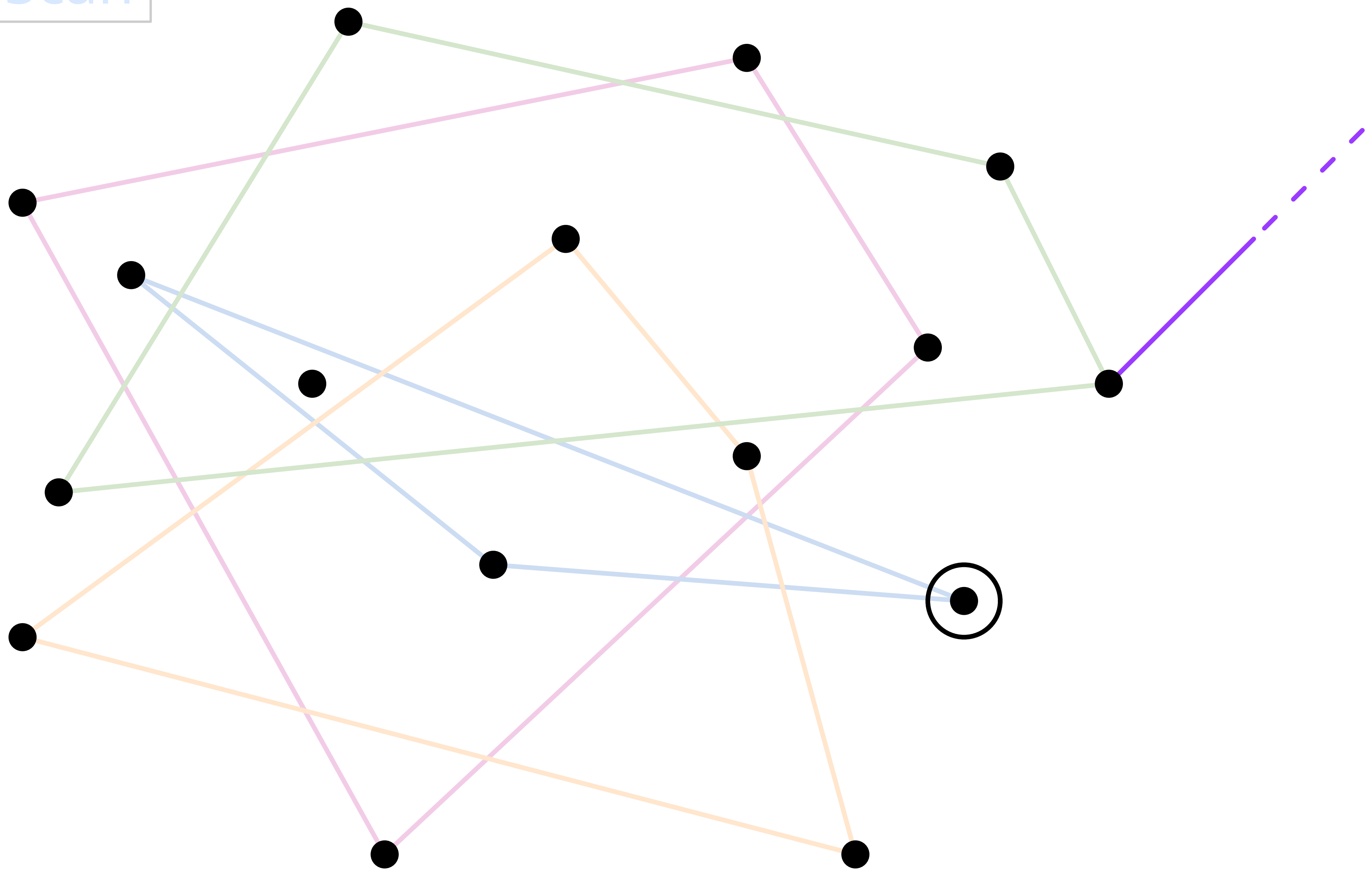


$n = 16$

Gift Wrapping

Example

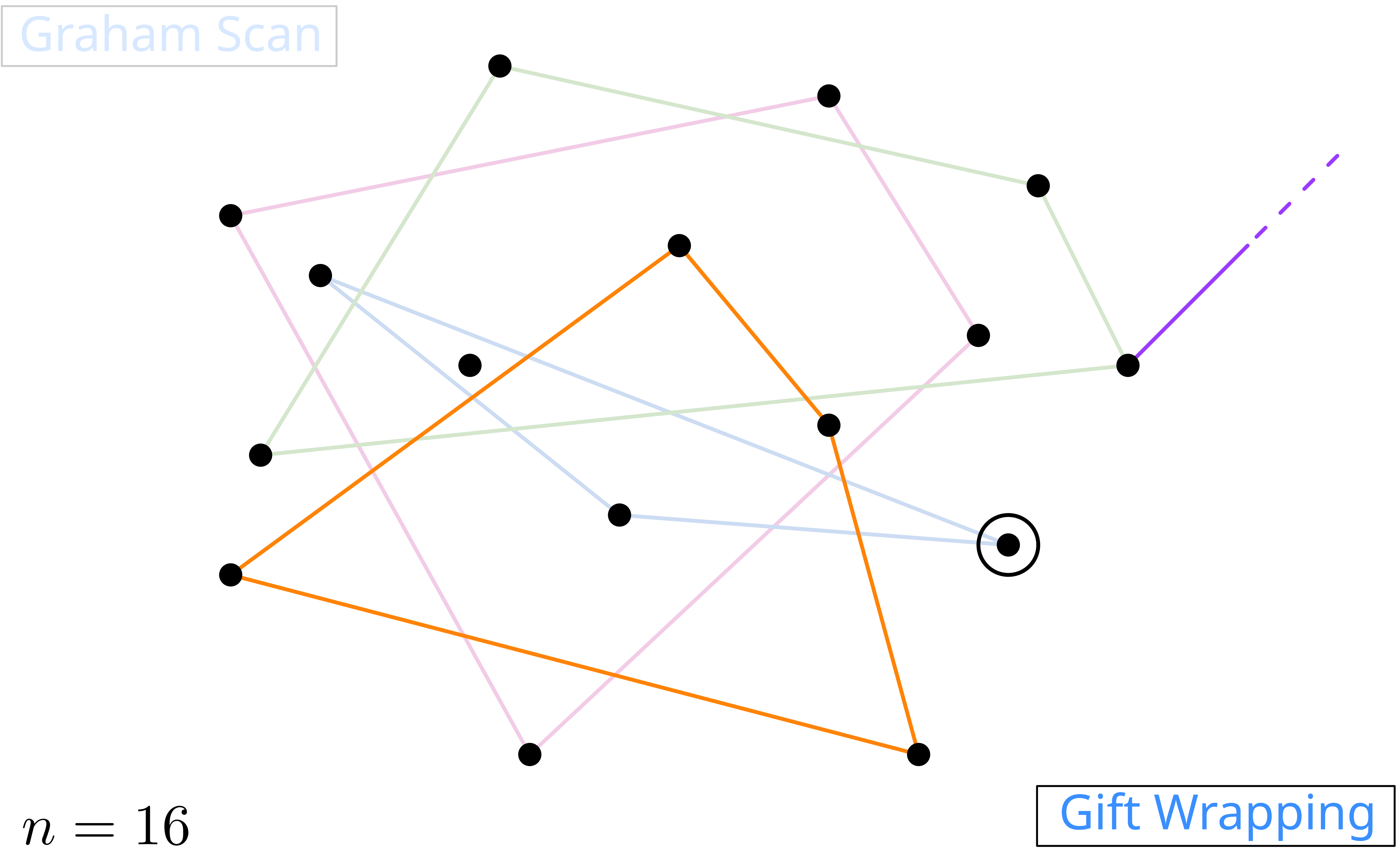
Graham Scan



Gift Wrapping

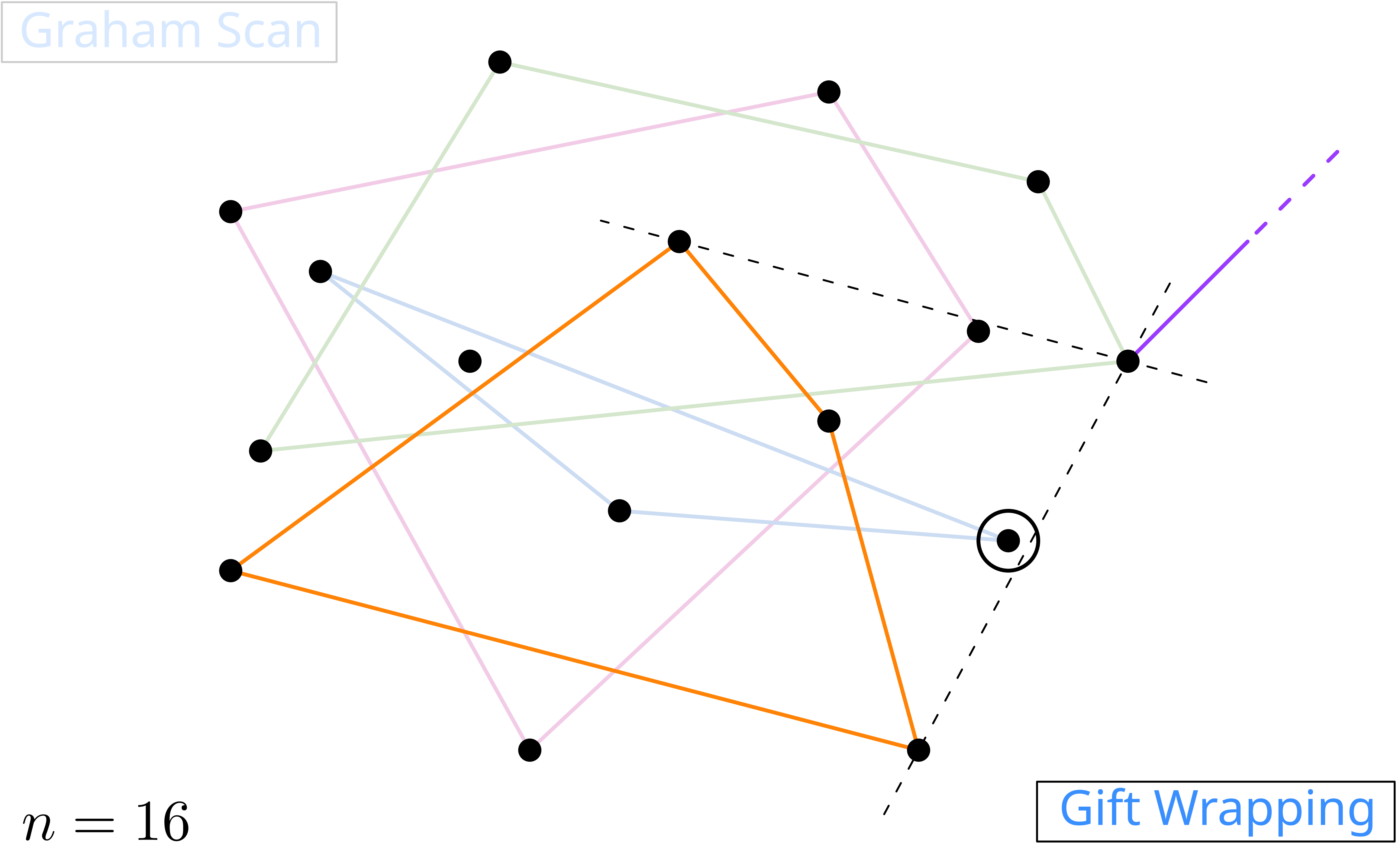
$n = 16$

Example



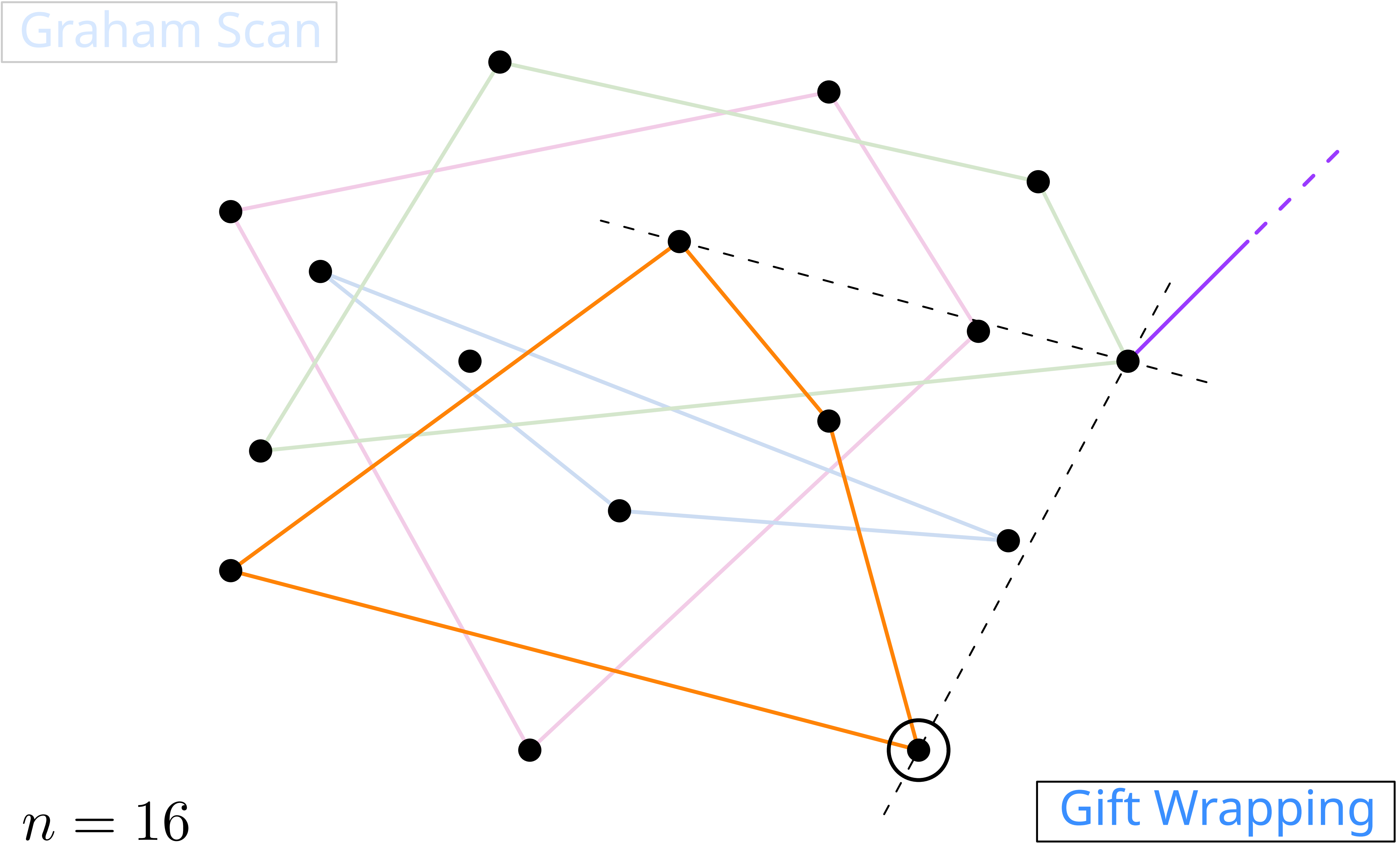
$n = 16$

Example



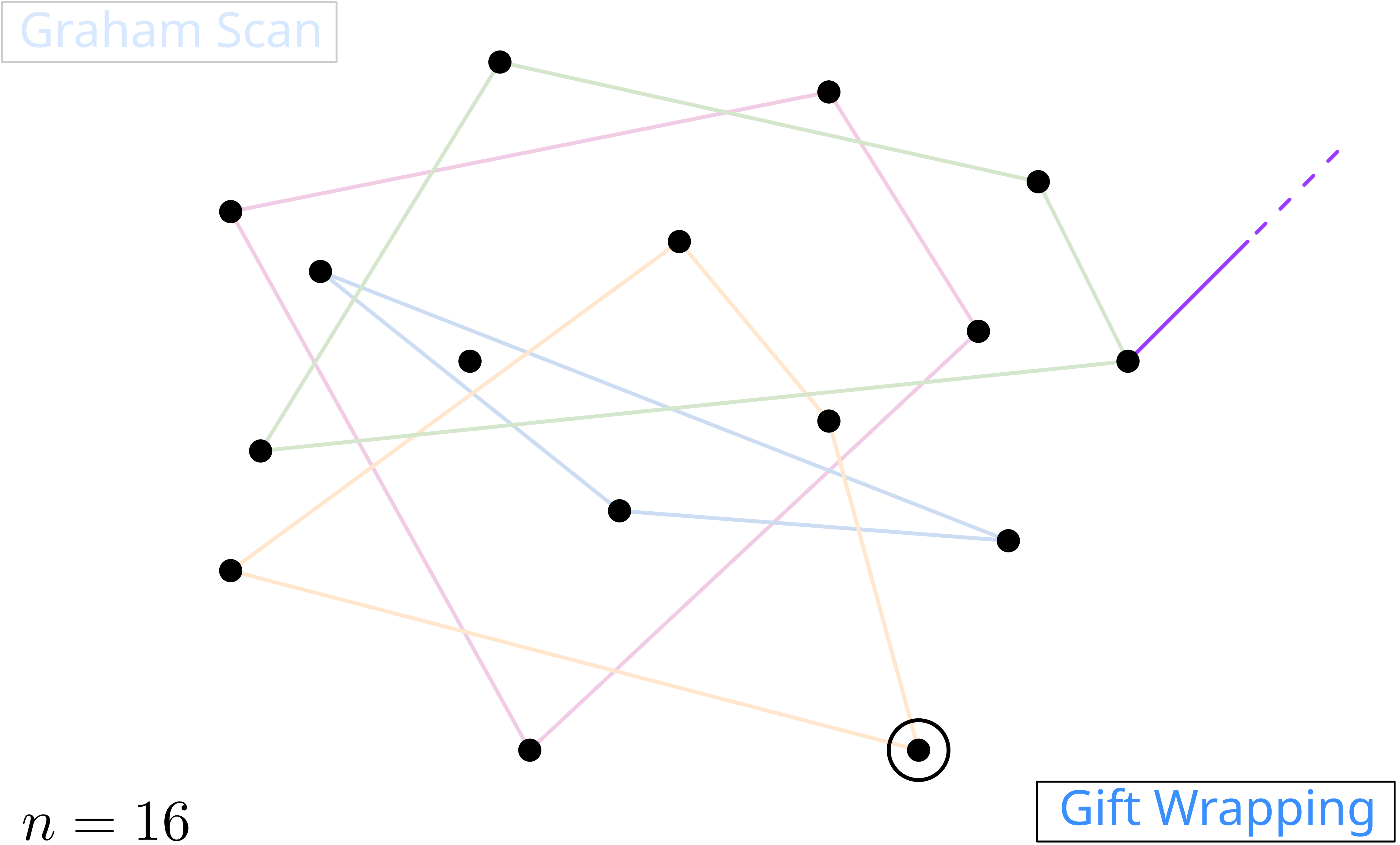
$n = 16$

Example



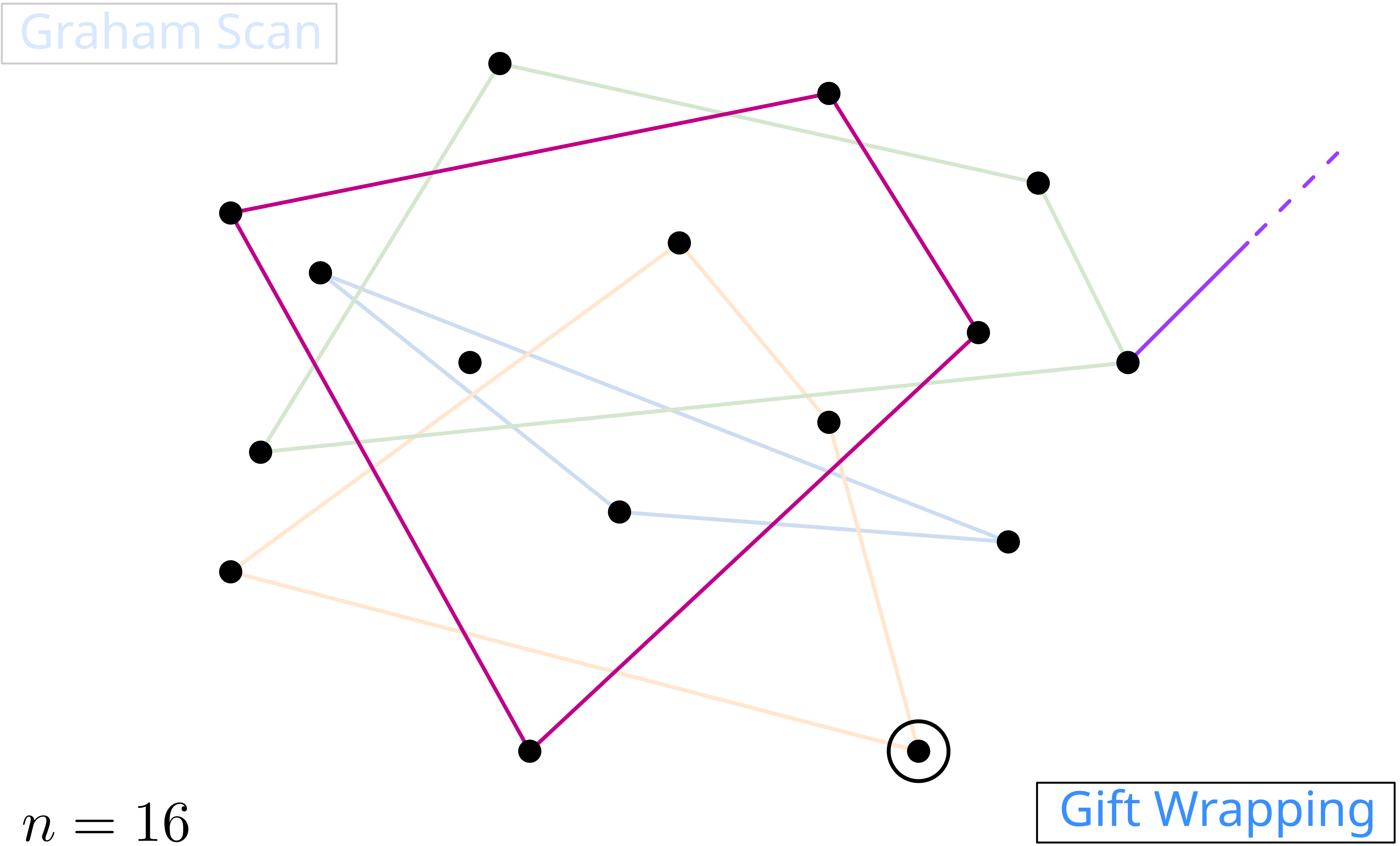
$n = 16$

Example



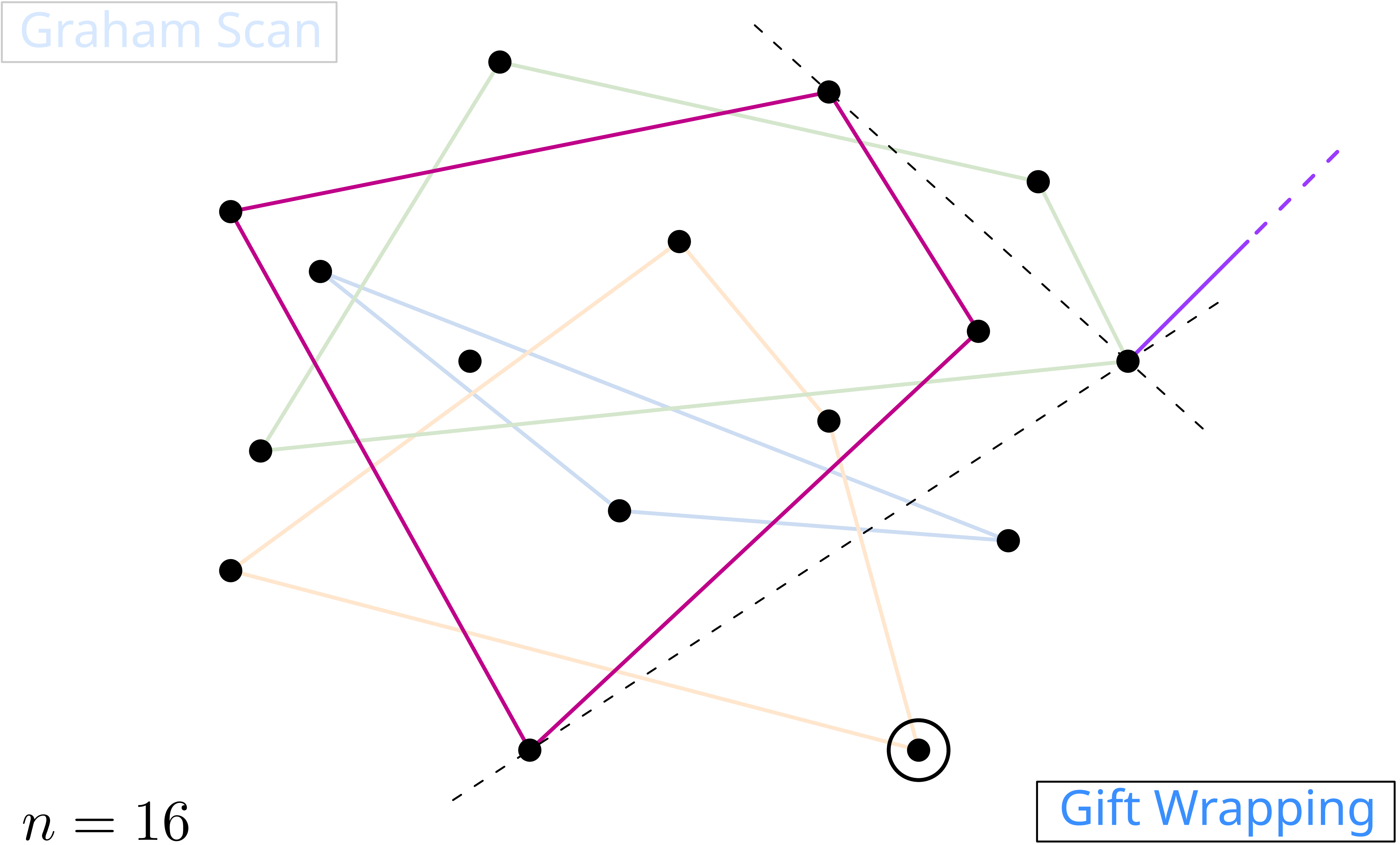
$n = 16$

Example



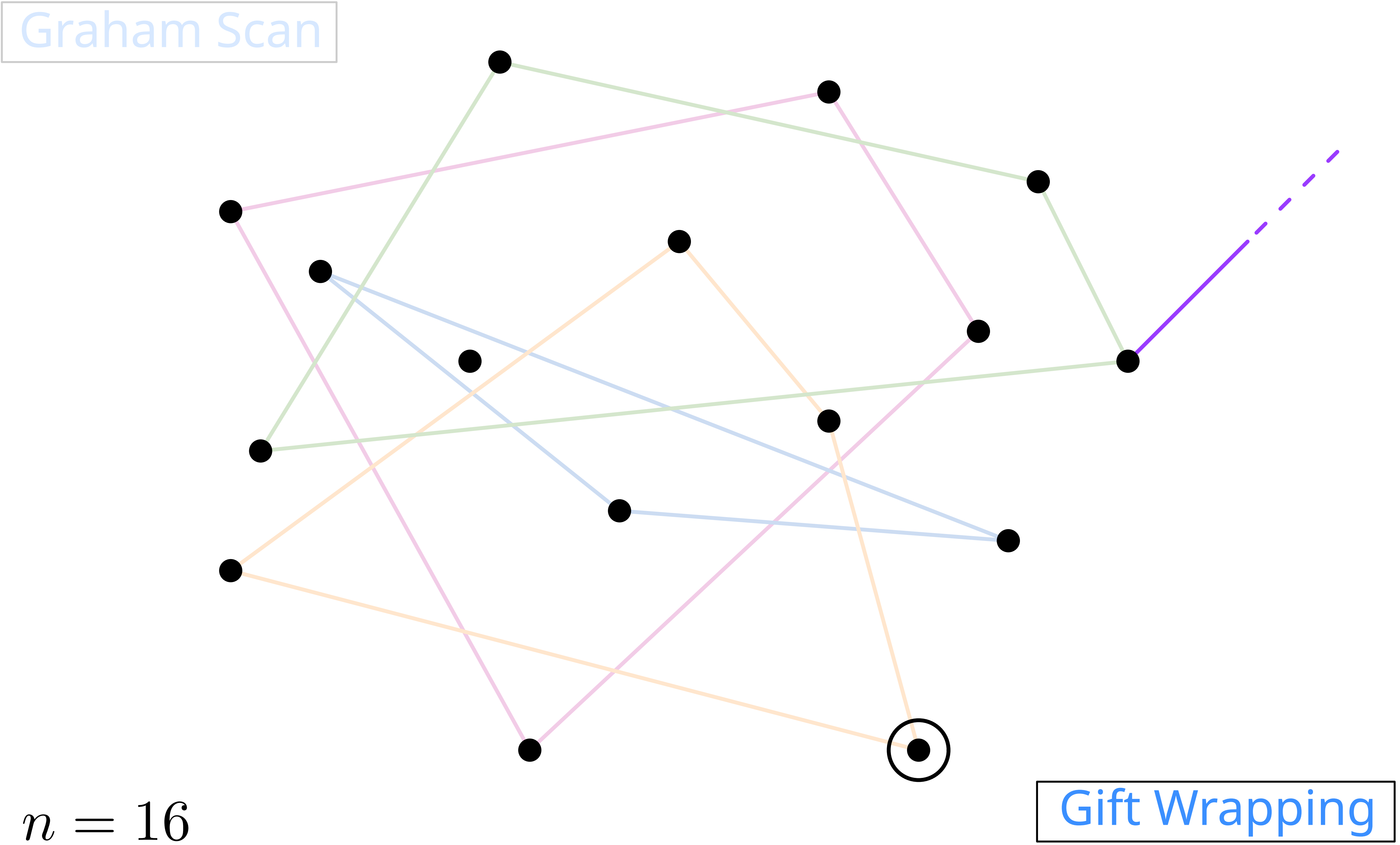
$n = 16$

Example



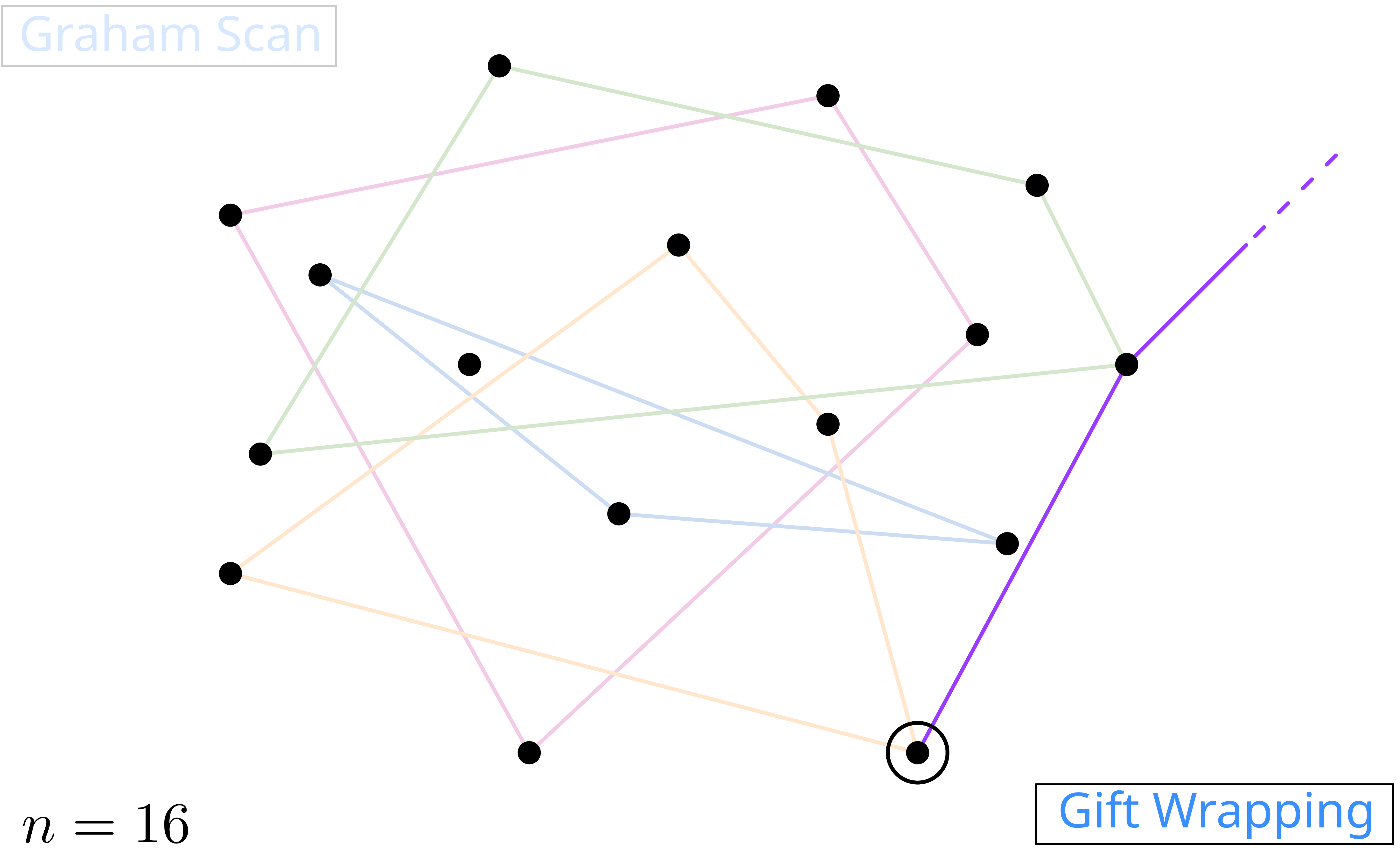
$n = 16$

Example



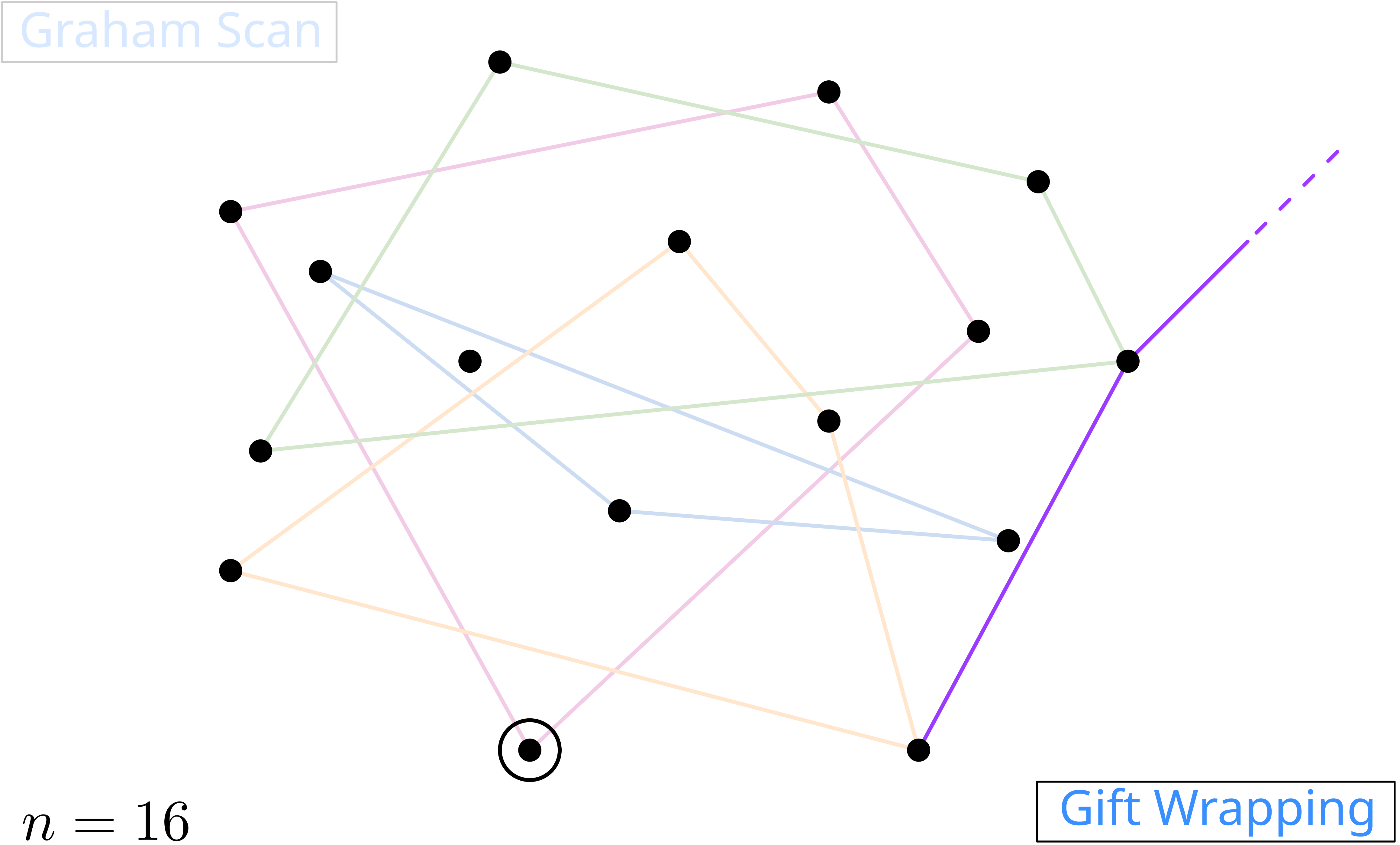
$n = 16$

Example



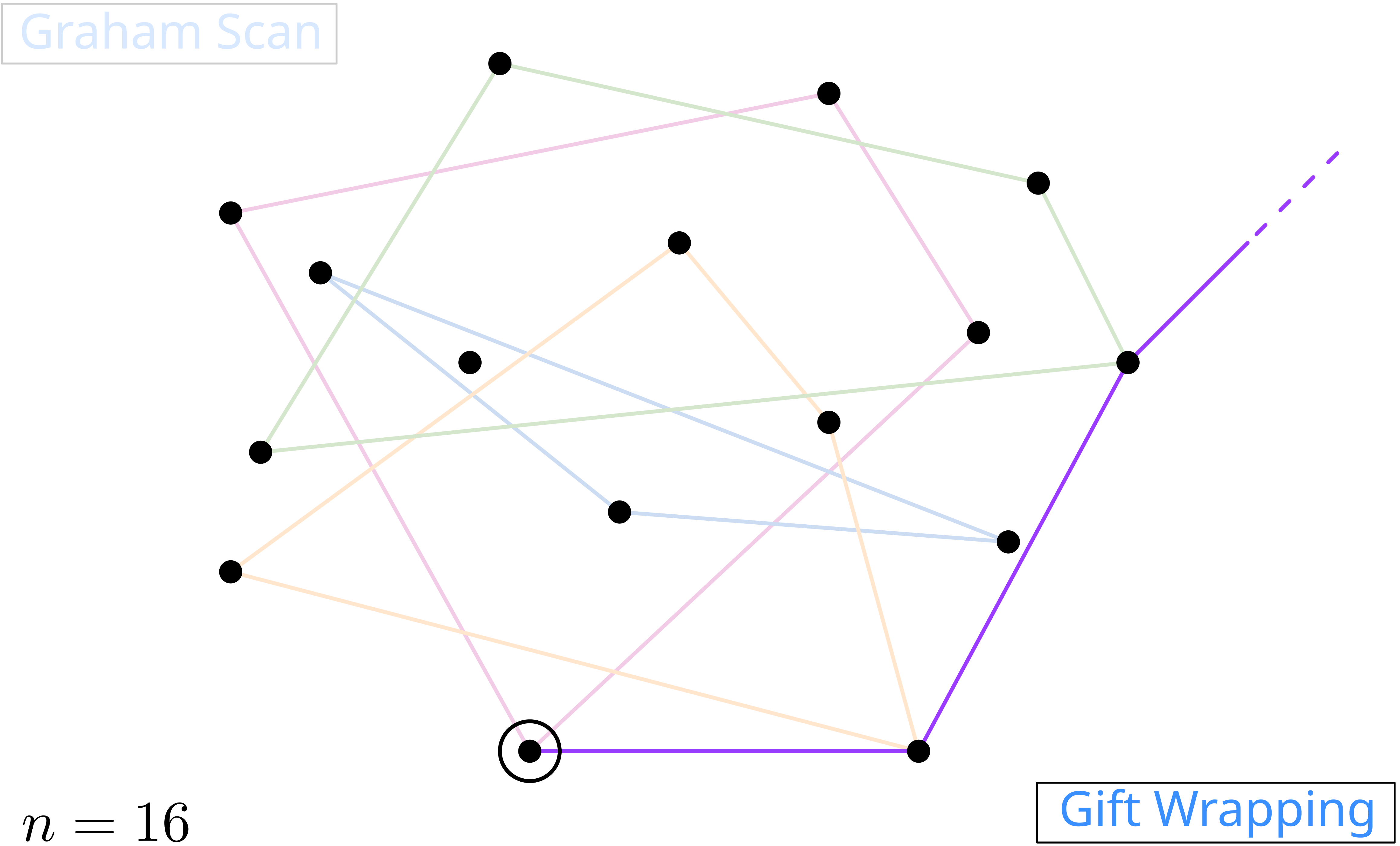
$n = 16$

Example



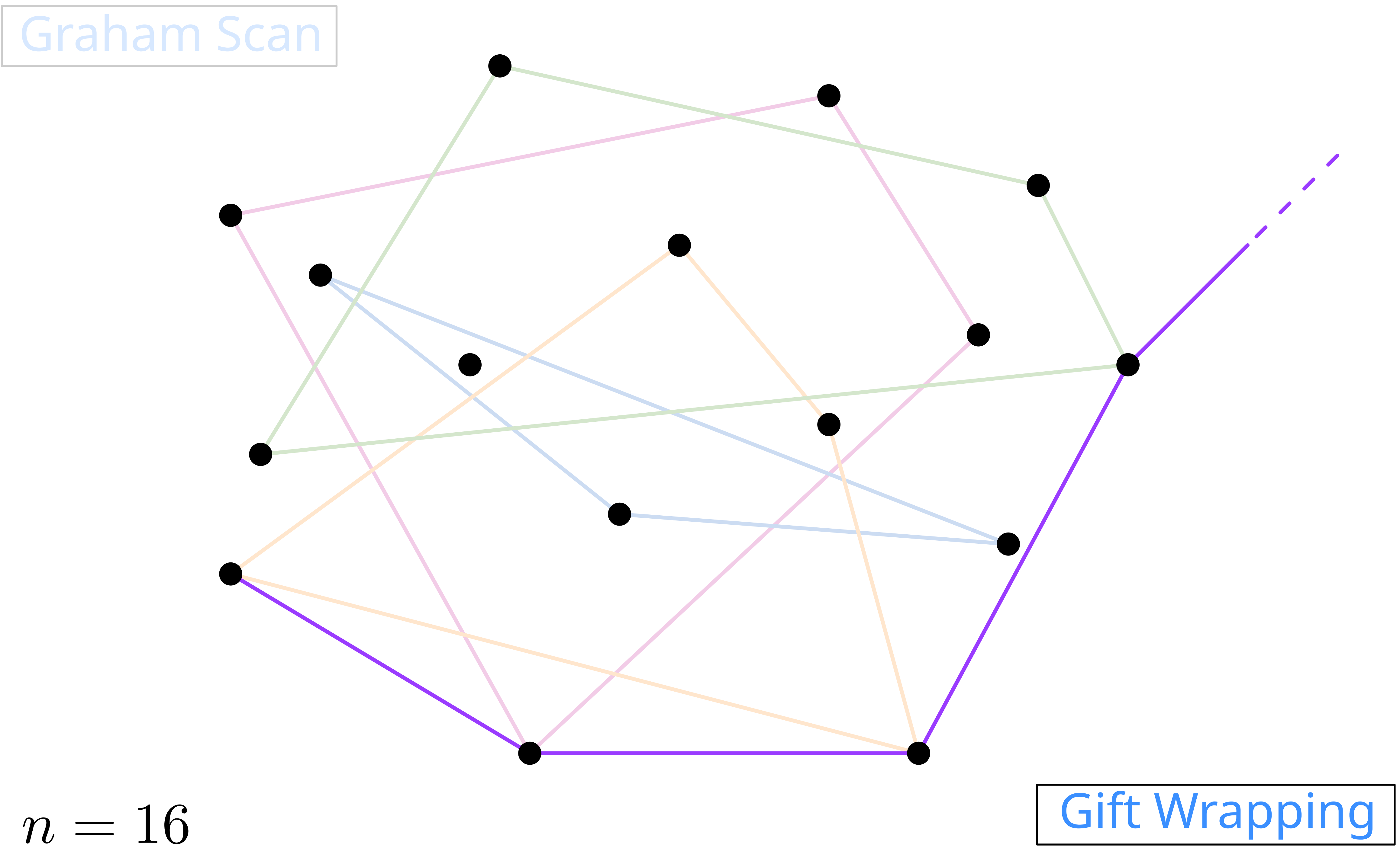
$n = 16$

Example



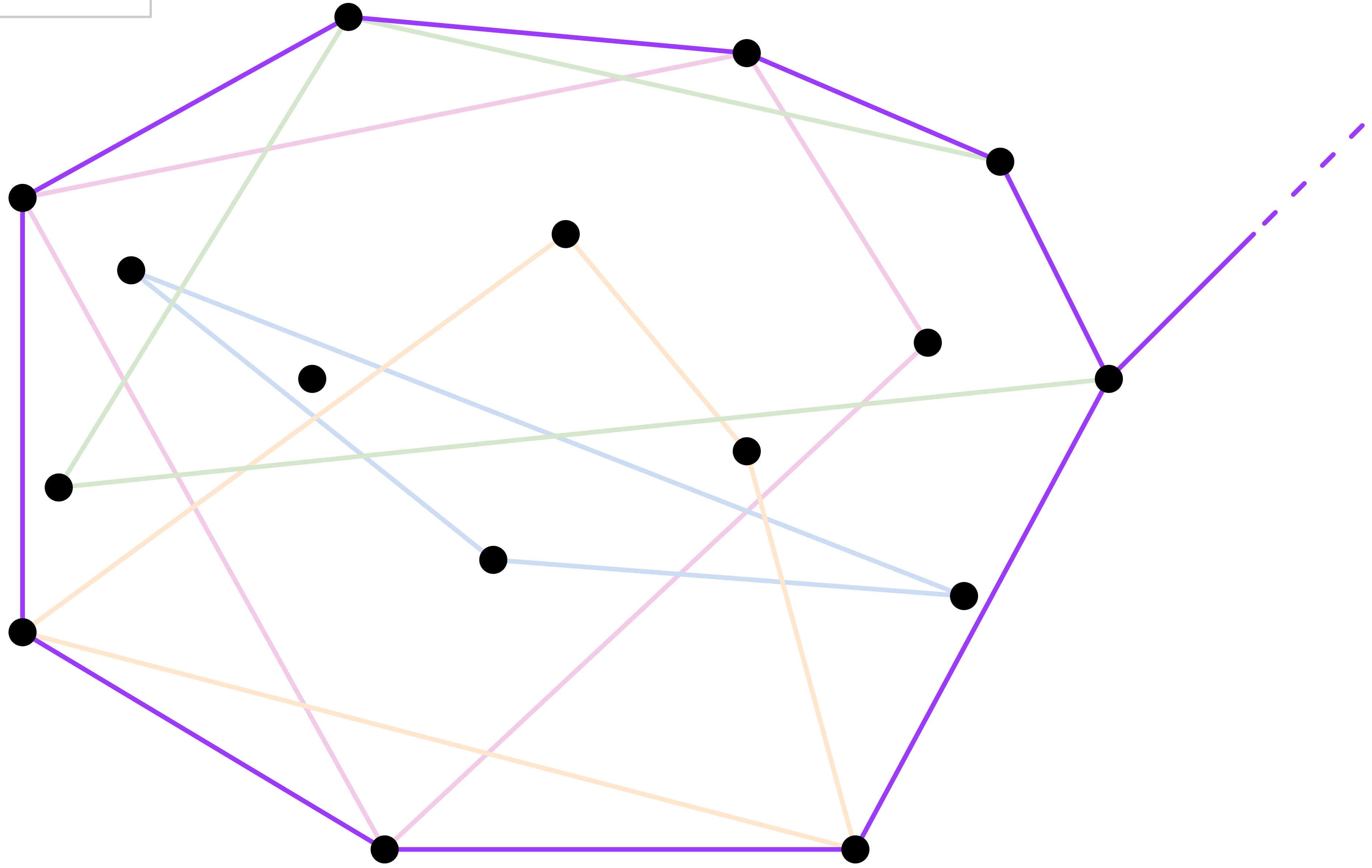
$n = 16$

Example



$n = 16$

Example

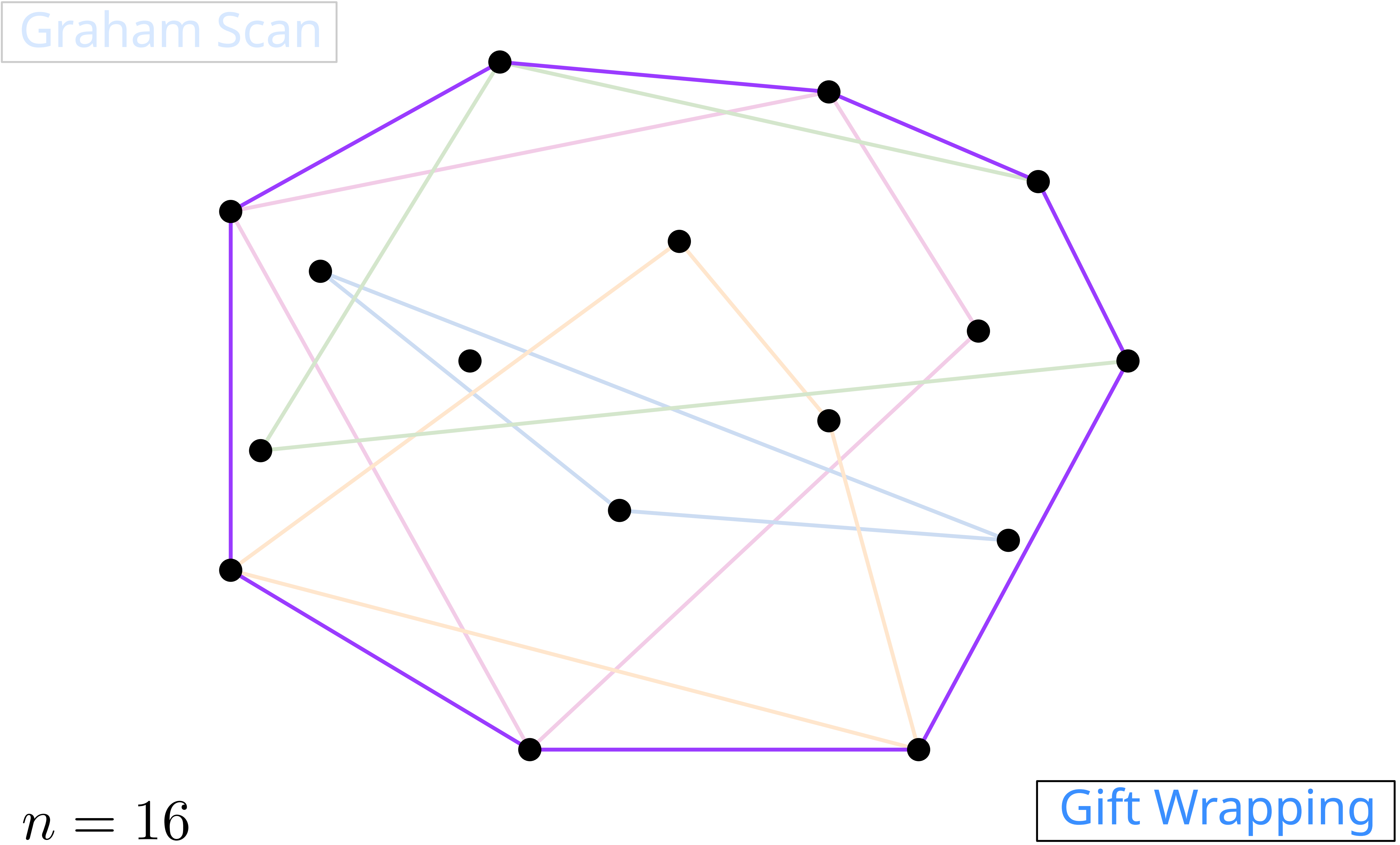


$n = 16$

Graham Scan

Gift Wrapping

Example



$n = 16$

Chan’s Algorithms

But in general we do not know h

Chan's Algorithms

But in general we do not know h

Algorithm CHANHULL(P, m)

partition P into sets P_i with m points each

for $i \leftarrow 1$ to n/m **do**

  $L_i \leftarrow$ GRAHAMSCAN(P_i) $O(m \log m)$ $\}\times n/m$

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

$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L

for $j \leftarrow 1$ to $m - 1$ **do**

  for $i \leftarrow 1$ to n/m **do**

    find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$ $O(\log m)$ $\}\times O(n)$

  $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/m}\}$ $O(n/m)$

  insert p_{j+1} into L $O(1)$

$\}\times O(m)$

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

return L

Chan's Algorithms

But in general we do not know h

Algorithm ChanHull(P, m)

partition P into sets P_i with m points each
for $i \leftarrow 1$ to n/m **do**
    $L_i \leftarrow$ GrahamScan(P_i)
$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L
for $j \leftarrow 1$ to $m - 1$ **do**
    for $i \leftarrow 1$ to n/m **do**
        find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$
    $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/m}\}$
    insert p_{j+1} into L
return L

Chan's Algorithms

But in general we do not know h

Algorithm ChanHull(P, m)

partition P into sets P_i with m points each
for $i \leftarrow 1$ to n/m **do**
    $L_i \leftarrow$ GrahamScan(P_i)
$p_0 \leftarrow (\infty, \infty)$, $p_1 \leftarrow$ right-most point in P, insert p_1 into L
for $j \leftarrow 1$ to $m - 1$ **do**
    for $i \leftarrow 1$ to n/m **do**
        find point $q_i \in L_i$ maximizing $\angle p_{j-1} p_j q_i$
    $p_{j+1} \leftarrow \max_{\angle}\{q_1, \ldots, q_{n/m}\}$
    insert p_{j+1} into L
    if $p_{j+1} = p_1$ **then**
        return L
return failure

What's up with m?

Suggestions?

What's up with m?

Algorithm FULLCHANHULL(P)

for $t \leftarrow 0, 1, 2, \ldots$ **do**
    $m = \ldots$
    $result \leftarrow$ CHANHULL(P, m)
    if $result \neq$ failure **then**
        break
return $result$

What's up with m?

Algorithm FULLCHANHULL(P)

for $t \leftarrow 0, 1, 2, \ldots$ **do**
    $m = \min\{n, 2^{2^t}\}$
    $result \leftarrow$ CHANHULL(P, m)
    if $result \neq$ failure **then**
        break
return $result$

What's up with m?

Algorithm FULLCHANHULL(P)

for $t \leftarrow 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  $result \leftarrow$ CHANHULL(P, m) $O(n \log m) = O(n \log 2^{2^t})$
  if $result \neq$ failure **then**
    break
return $result$

What's up with m?

Algorithm FULLCHANHULL(P)

1: **for** $t \leftarrow 0, 1, 2, \ldots$ **do**
2: $\quad m = \min\{n, 2^{2^t}\}$
3: $\quad result \leftarrow$ CHANHULL(P, m) $\quad O(n \log m) = O(n \log 2^{2^t})$
4: $\quad$ **if** $result \neq$ failure **then**
5: $\quad\quad$ **break**
6: **return** $result$

$\times O(\log \log h)$

What's up with m?

Algorithm FullChanHull(P)

for $t \leftarrow 0, 1, 2, \ldots$ **do**
    $m = \min\{n, 2^{2^t}\}$
    $result \leftarrow$ ChanHull(P, m) $\quad O(n \log m) = O(n \log 2^{2^t})$
    if $result \neq$ failure **then**
        break

$\times O(\log \log h)$

return $result$

Running time:

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

What's up with m?

Algorithm FullChanHull(P)

for $t \leftarrow 0, 1, 2, \ldots$ **do**
- $m = \min\{n, 2^{2^t}\}$
- $result \leftarrow$ ChanHull(P, m) $\quad O(n \log m) = O(n \log 2^{2^t})$
- **if** $result \neq$ failure **then**
  - **break**

$\times O(\log \log h)$

return $result$

Running time:

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

What's up with m?

Algorithm FULLCHANHULL(P)

for $t \leftarrow 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  $\mathit{result} \leftarrow$ CHANHULL(P, m) $O(n \log m) = O(n \log 2^{2^t})$
  if $\mathit{result} \neq$ failure **then**
    break
return result

$\times O(\log \log h)$

Running time:

$$\sum_{t=0}^{\log\log h} O(n \log 2^{2^t}) = O(n) \sum_{t=0}^{\log\log h} O(2^t) \leq O(n) \cdot O(2^{\log\log h + 1})$$

What's up with m?

Algorithm FULLCHANHULL(P)

for $t \leftarrow 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  $result \leftarrow$ CHANHULL(P, m) $O(n \log m) = O(n \log 2^{2^t})$
  if $result \neq$ failure **then**
    break
return $result$

$\times O(\log \log h)$

Running time:

$$\sum_{t=0}^{\log \log h} O(n \log 2^{2^t}) = O(n) \sum_{t=0}^{\log \log h} O(2^t) \leq O(n) \cdot O(2^{\log \log h + 1})$$

$$= O(n) \cdot O(\log h)$$

What's up with m?

Algorithm FullChanHull(P)

for $t \leftarrow 0, 1, 2, \ldots$ **do**

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

  $result \leftarrow$ ChanHull(P, m) $O(n \log m) = O(n \log 2^{2^t})$

  if $result \neq$ failure **then**

    break

$\times O(\log \log h)$

return $result$

Running time:

$$\sum_{t=0}^{\log \log h} O(n \log 2^{2^t}) = O(n) \sum_{t=0}^{\log \log h} O(2^t) \leq O(n) \cdot O(2^{\log \log h + 1})$$

$$= O(n) \cdot O(\log h) = O(n \log h)$$

Summary

Slow Convex Hull $O(n^3)$

Graham Scan $O(n \log n)$

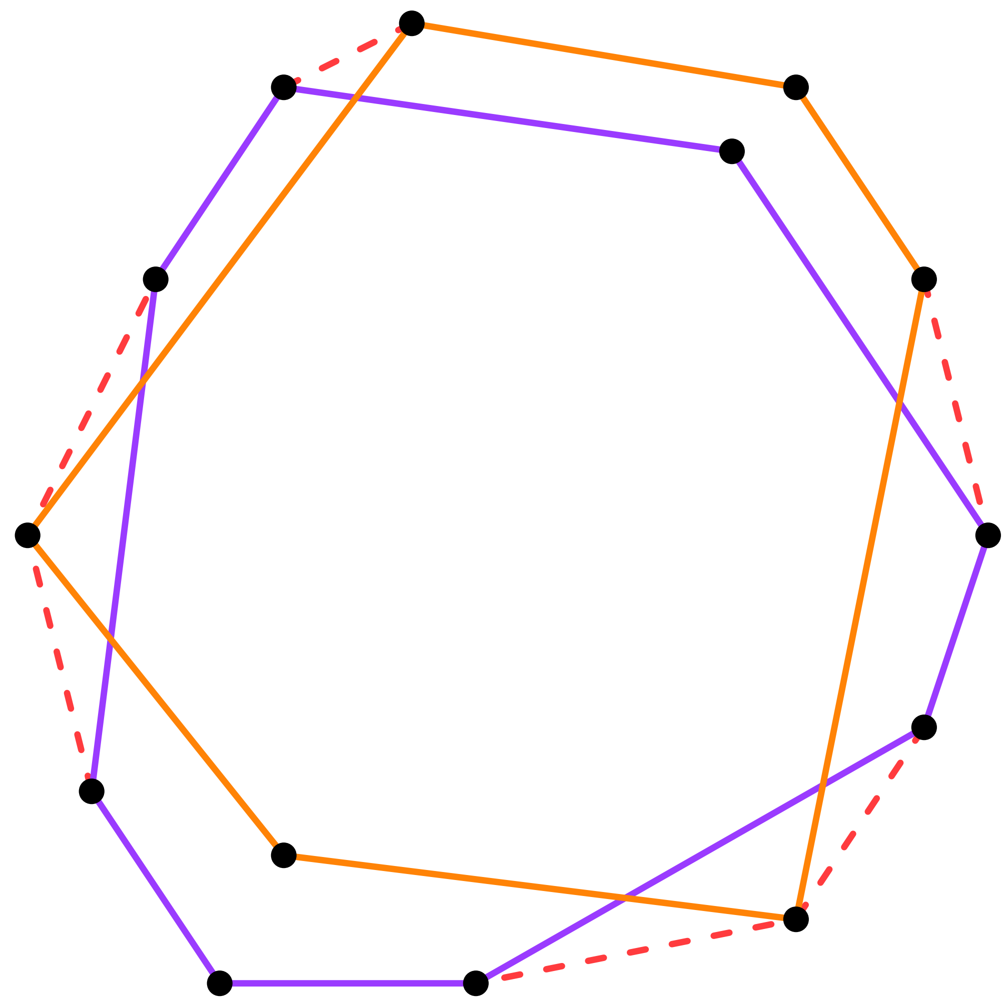
Gift Wrapping $O(nh)$

Chan's Hull $O(n \log h)$

Other approaches: divide and conquer

Split the point set in two halves, compute the convex hulls recursively, and merge

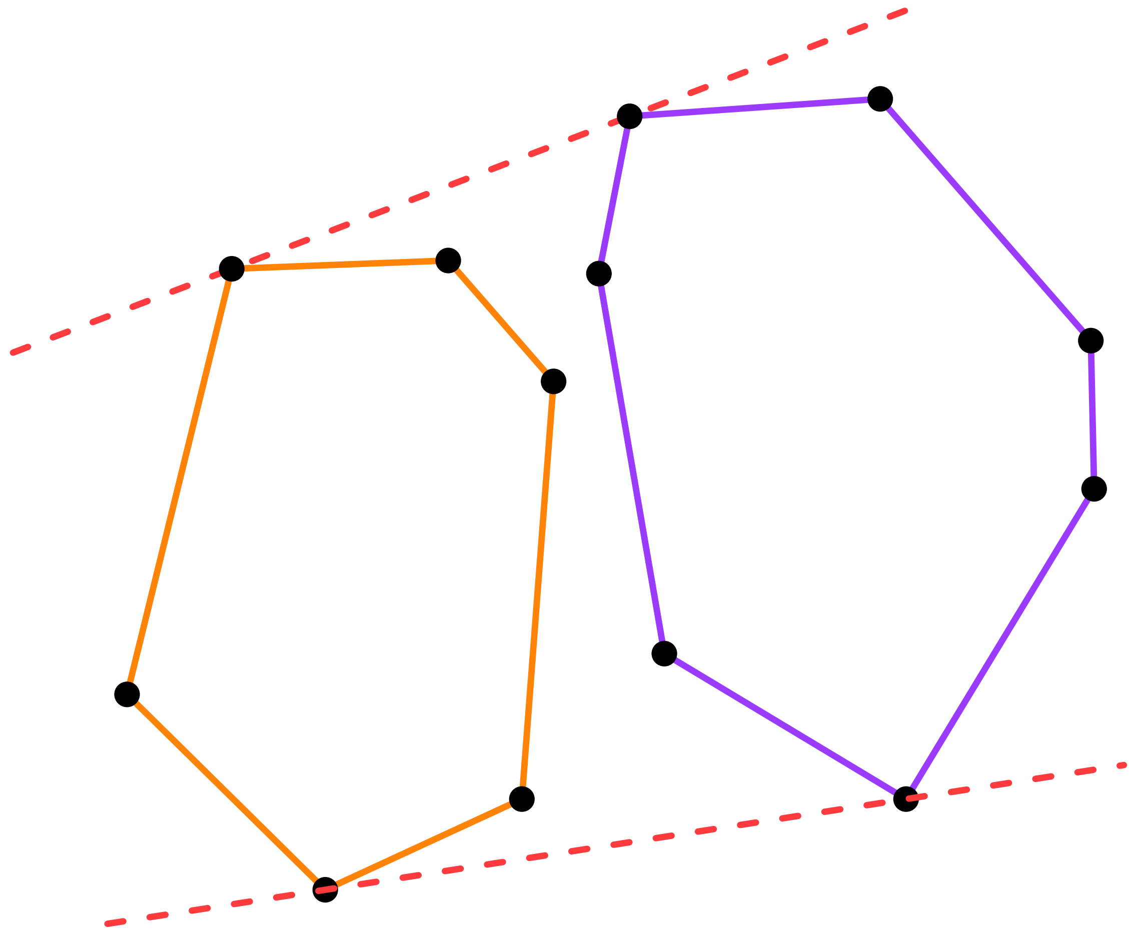
The merge step involves finding “extreme vertices” in every direction



Other approaches: divide and conquer

Alternatively: split the point set in two halves on x-coordinate, compute the convex hulls recursively, and merge

The merge step now comes down to finding two common tangent lines



Convex hulls in 3D

For a 3-dimensional point set, the convex hull is a convex polyhedron

It has vertices (0-dim.), edges (1-dim.), and facets (2-dim.) on its boundary, and a 3-dimensional interior

The boundary is a planar graph, so it has $O(n)$ vertices, edges, and facets

Convex hulls in 4D

For a 4-dimensional point set, the convex hull is a convex polyhedron

It has vertices (0-dim.), edges (1-dim.), 2-facets (2-dim.), and 3-facets (3-dim.) on its boundary, and a 4-dimensional interior

Its boundary can have $\Theta(n^2)$ facets in the worst case!