Convex hull

CPSC490 2014/15WT2

Nasa Rouf

Best Sum

Authored by Khongor on *Dec 05 2013*

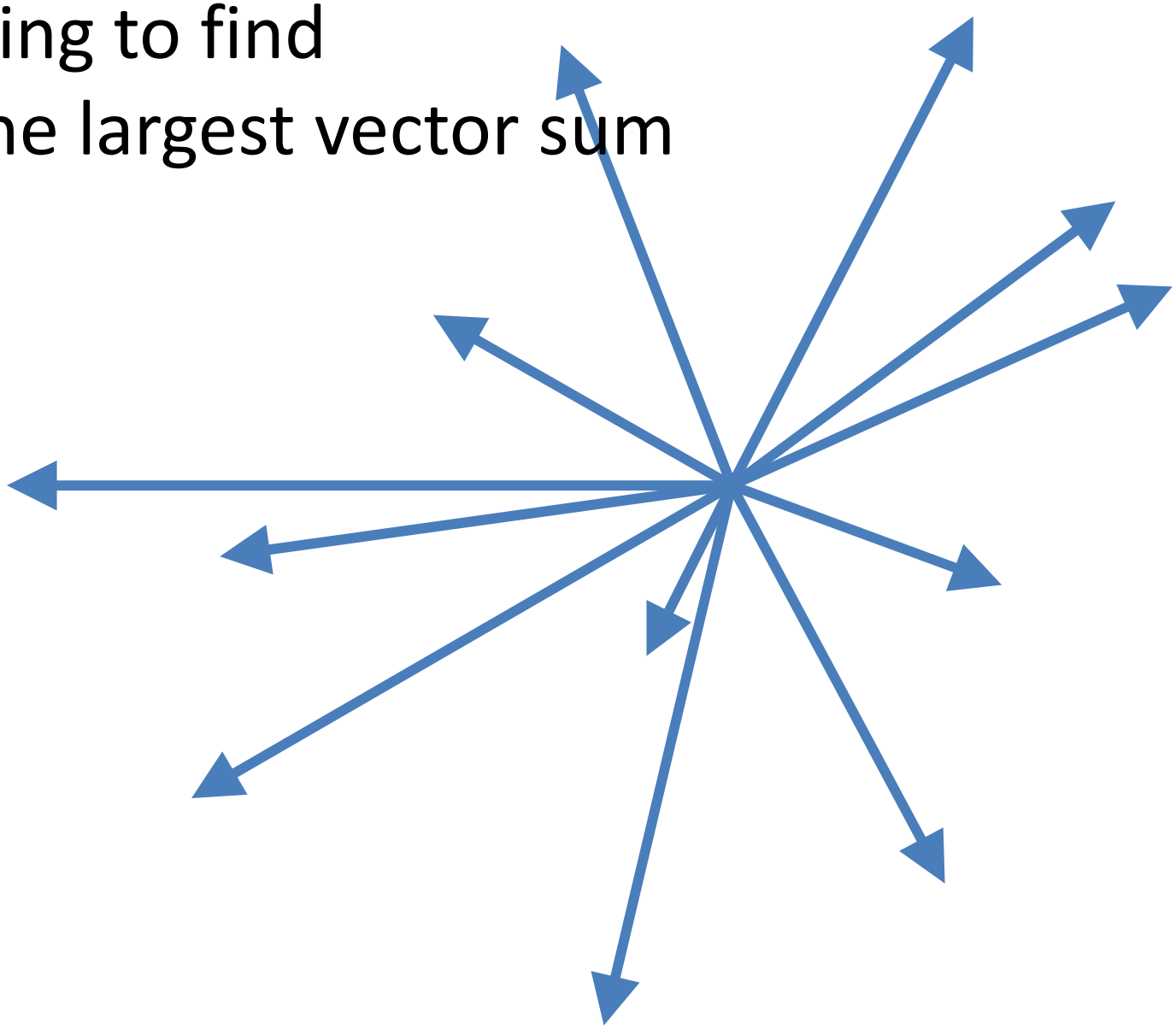
Problem Statement

You are given two arrays *A* and *B* of length **N**. Let S be the set of integers from 1 to **N**. Can you find the maximum possible value of $(A_{i1}+A_{i2}+...+A_{ik})^2+(B_{i1}+B_{i2}+...+B_{ik})^2$ where {i1,i2...ik} is a non-empty subset of S?

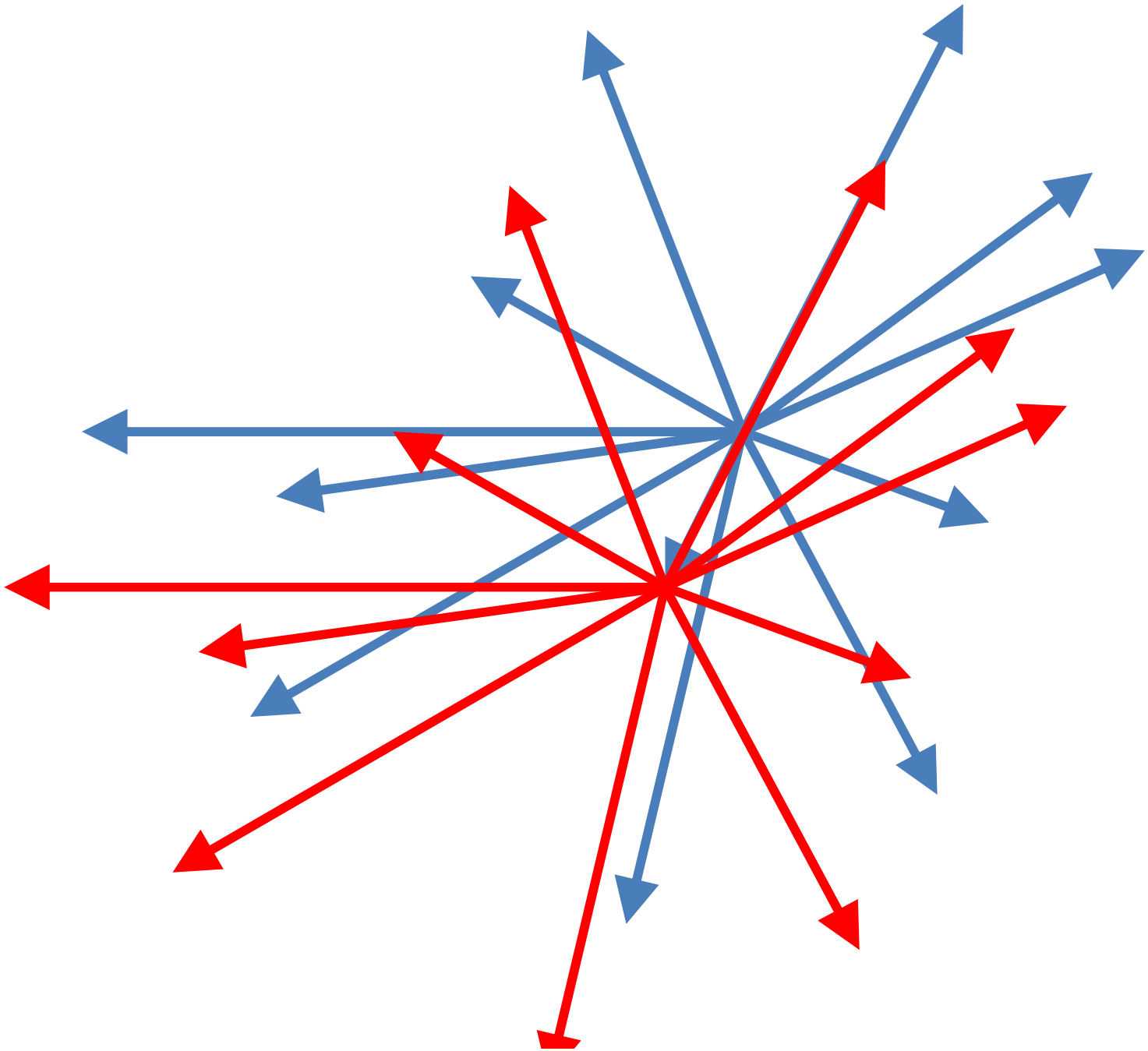
[https://www.hackerrank.com/challenges/best-sum]

Best sum

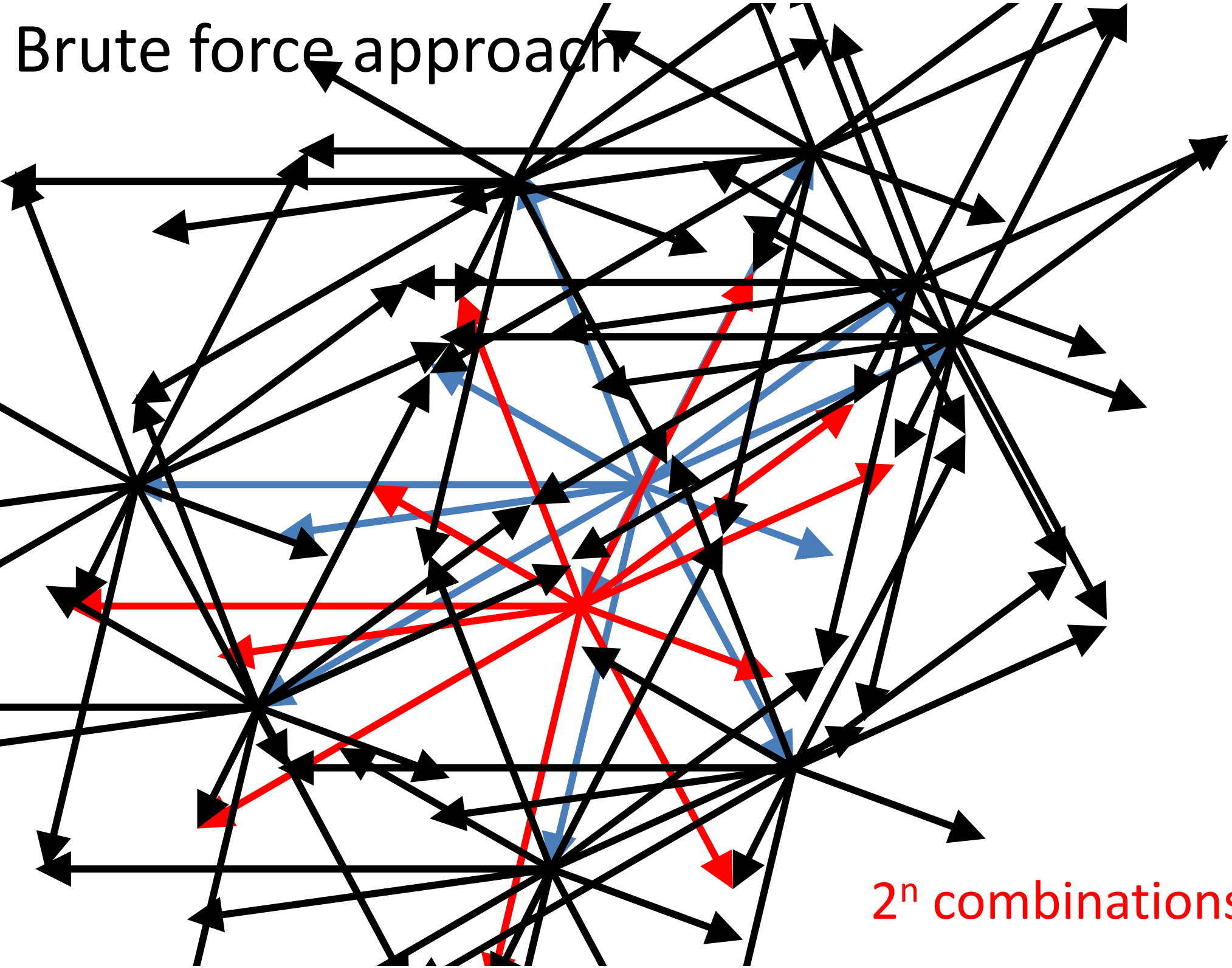
Asking to find
the largest vector sum



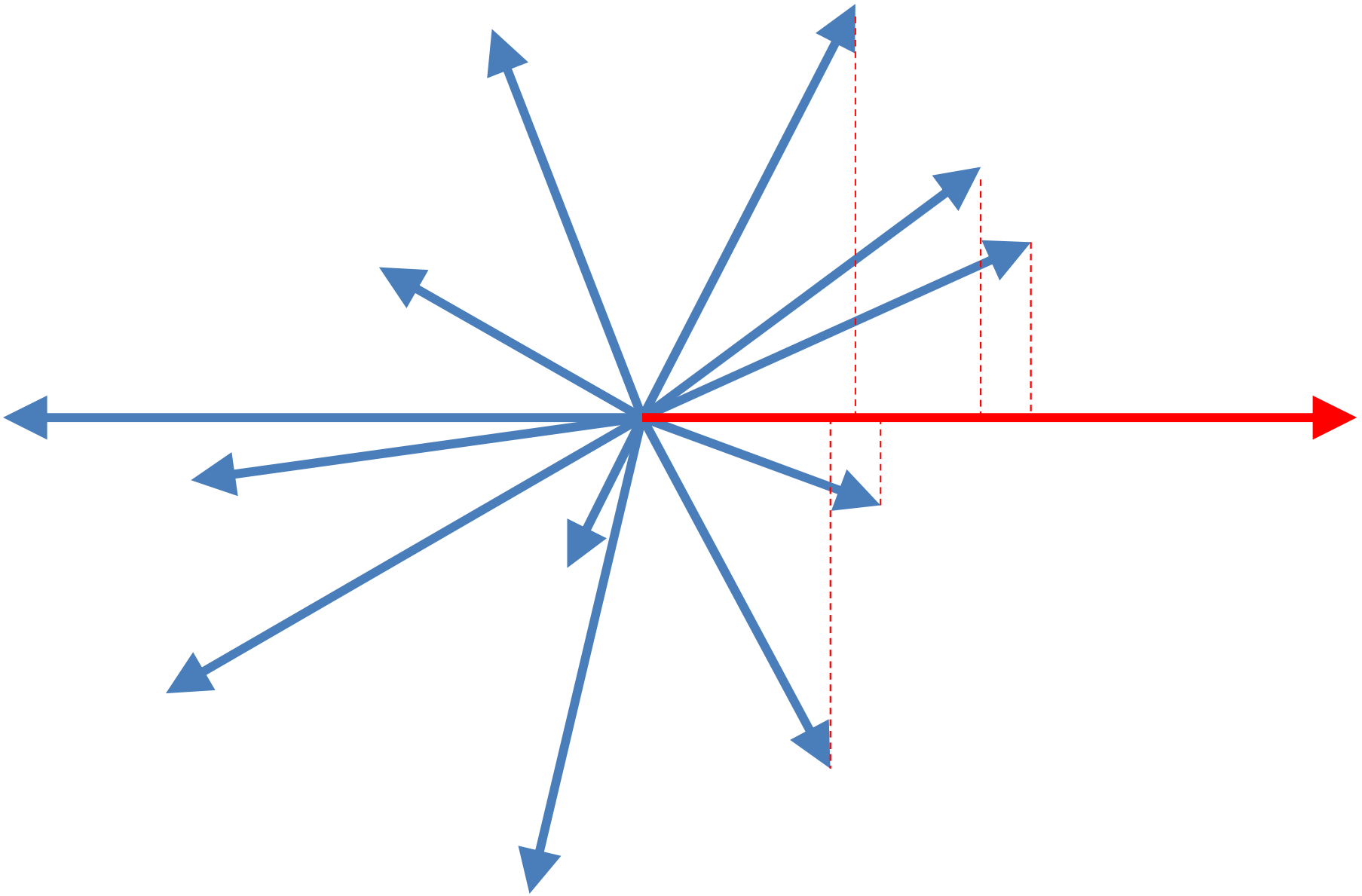
Brute force approach



Brute force approach

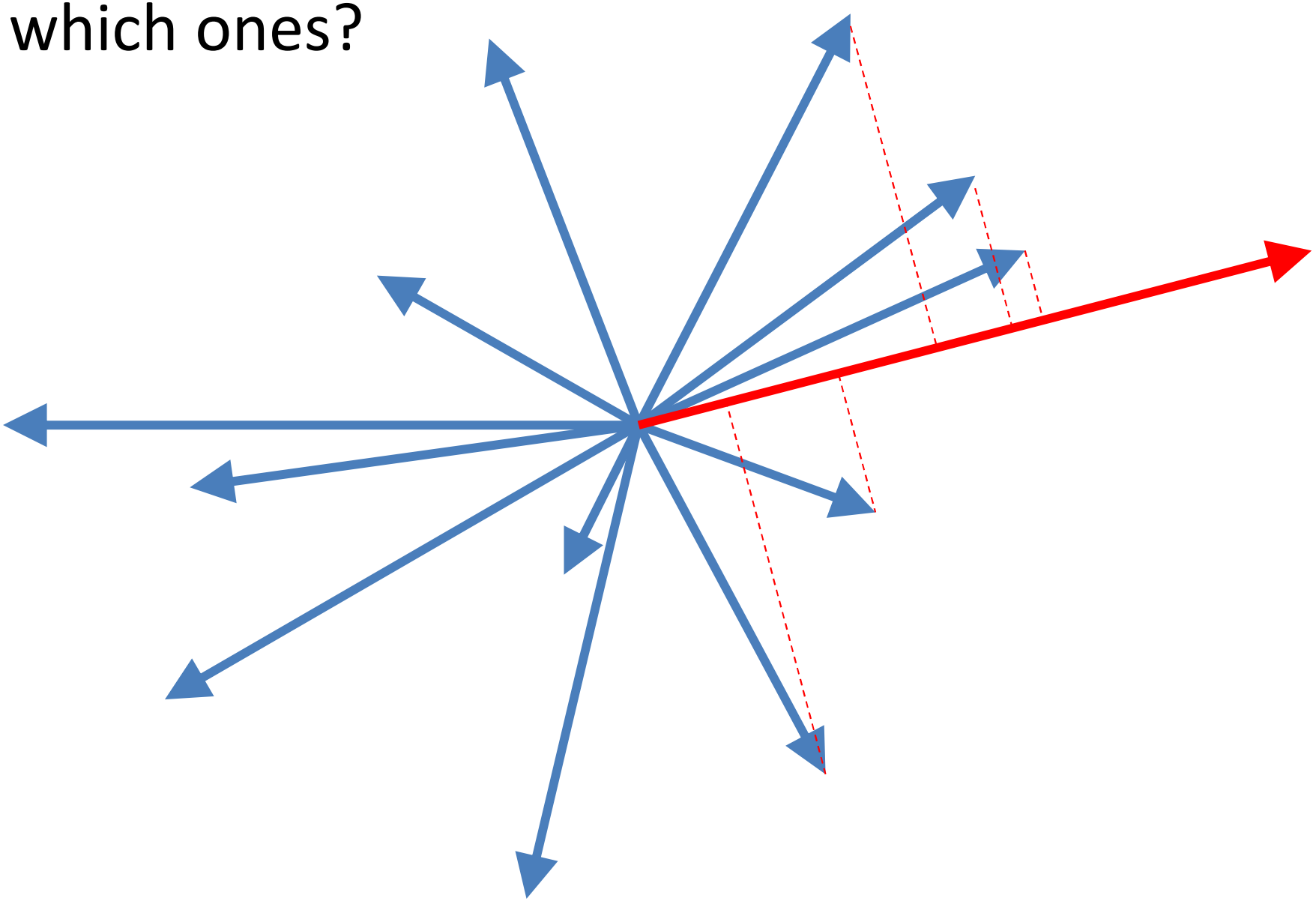


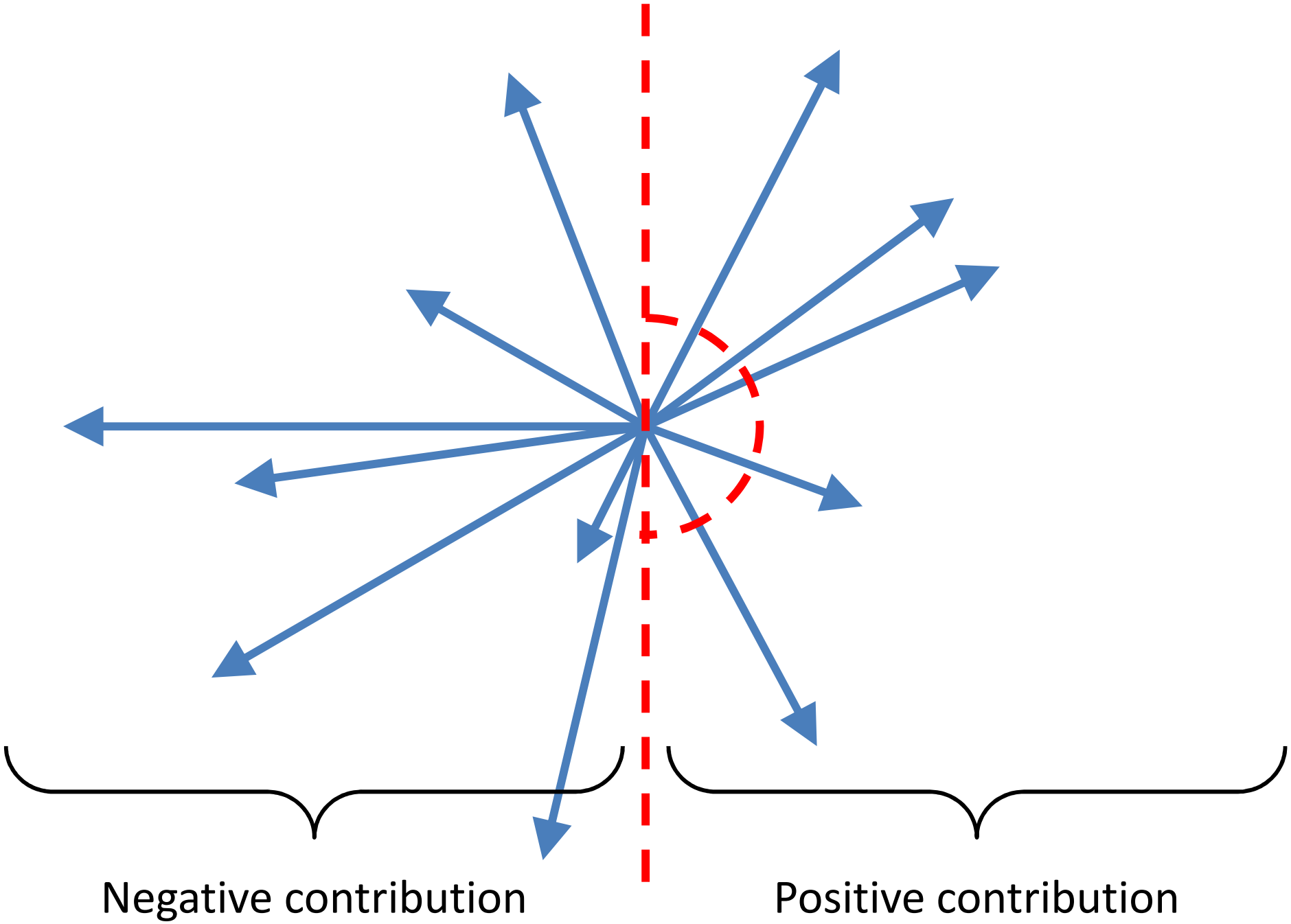
What if we knew the direction

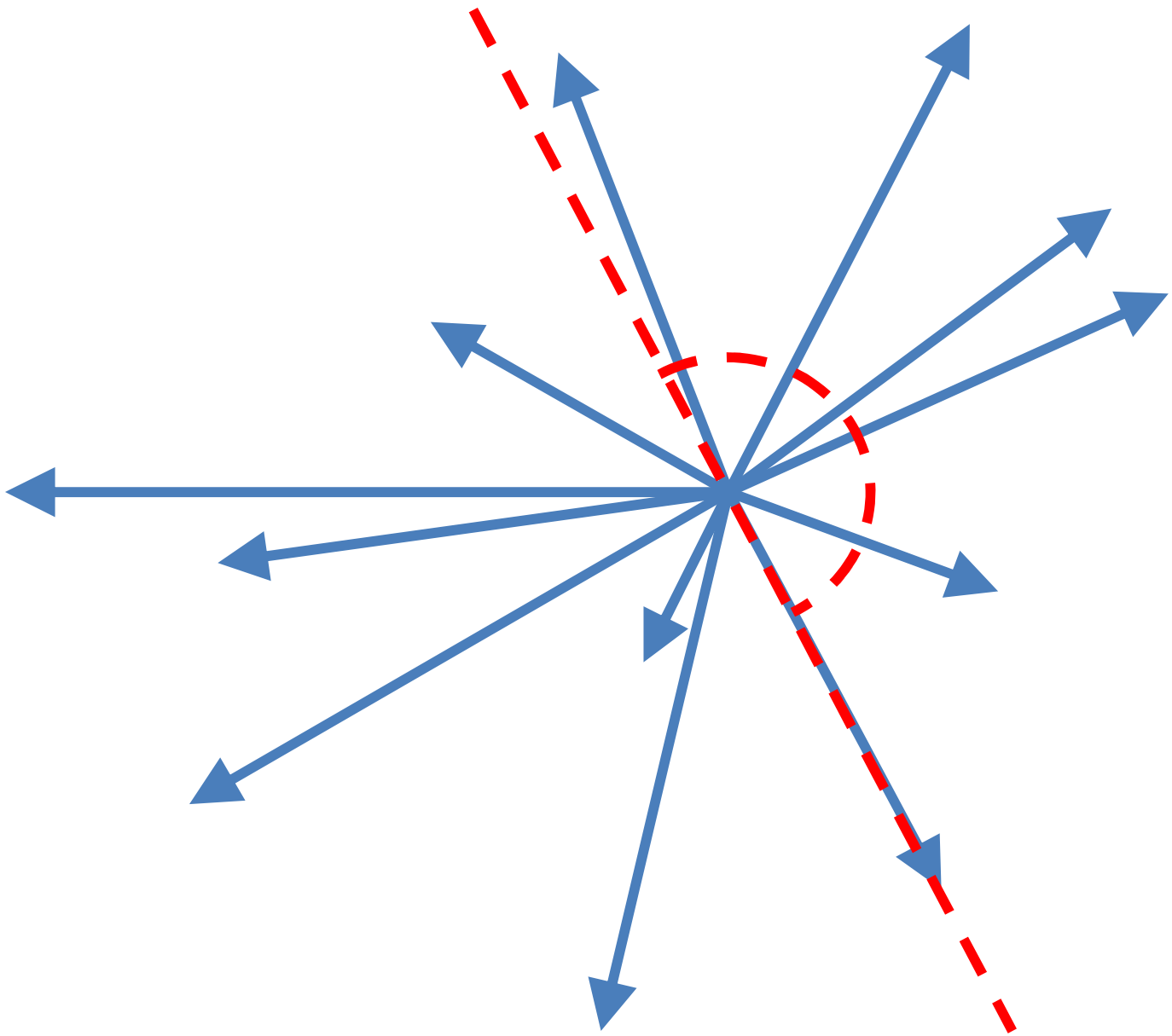


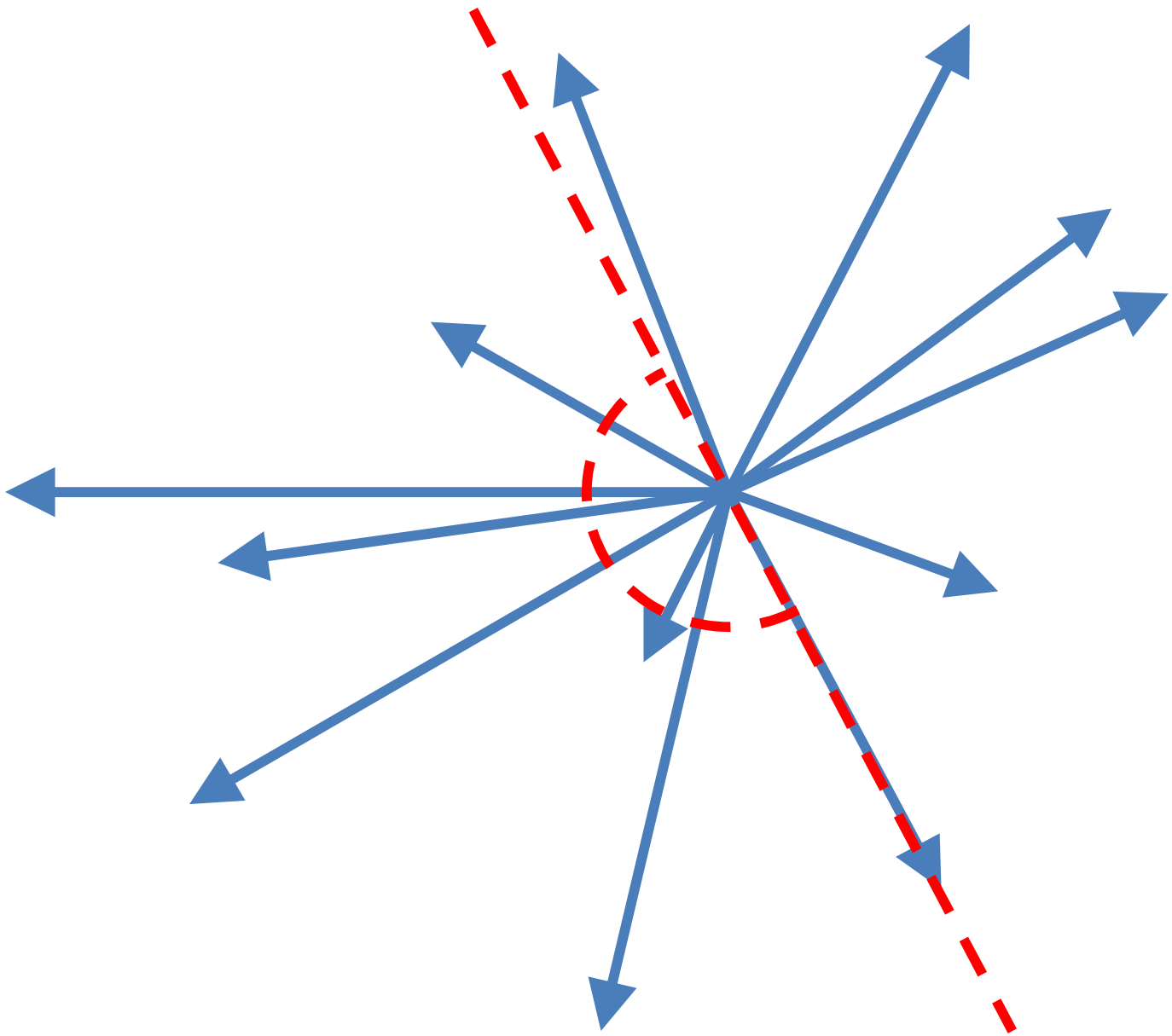
Try all directions

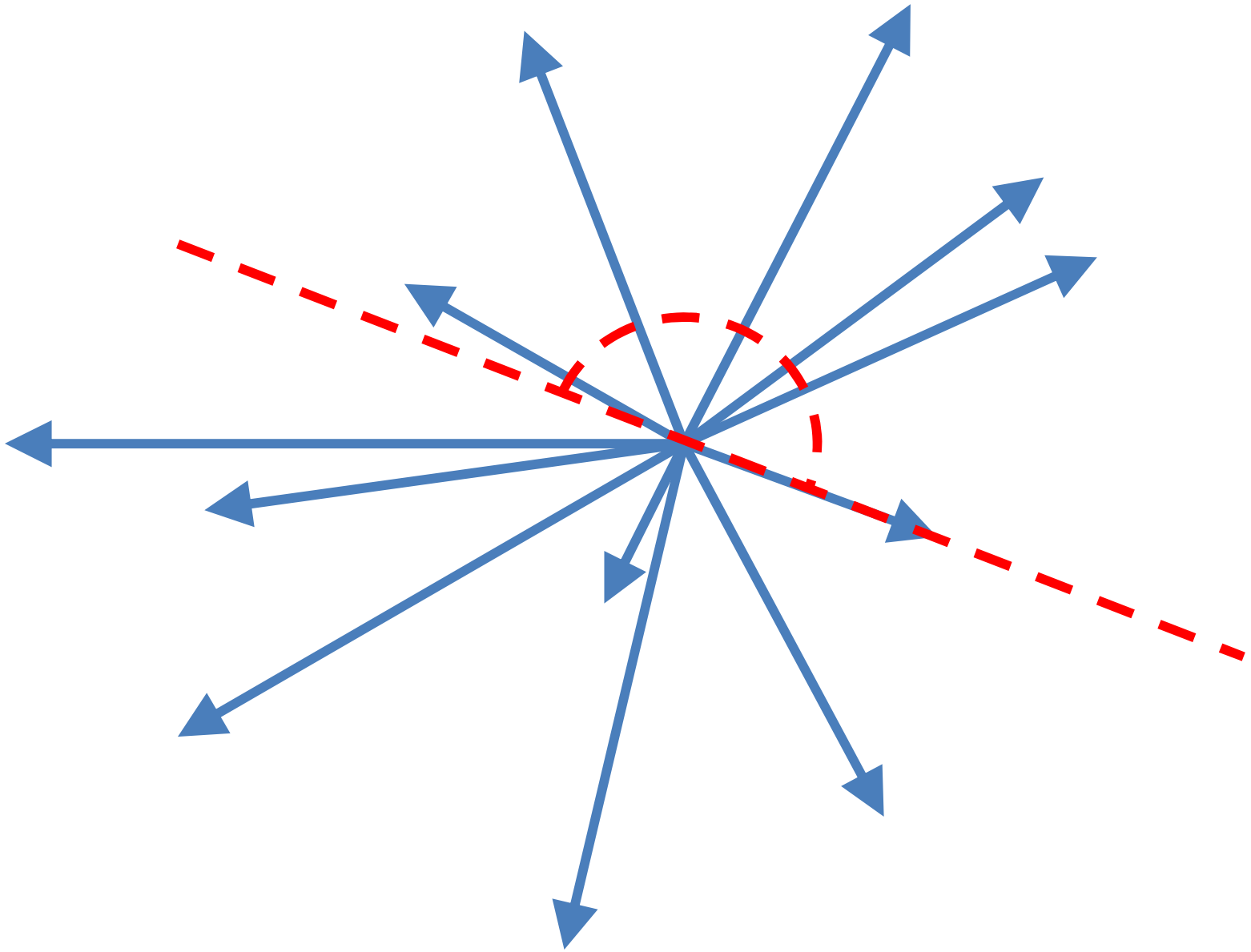
But which ones?

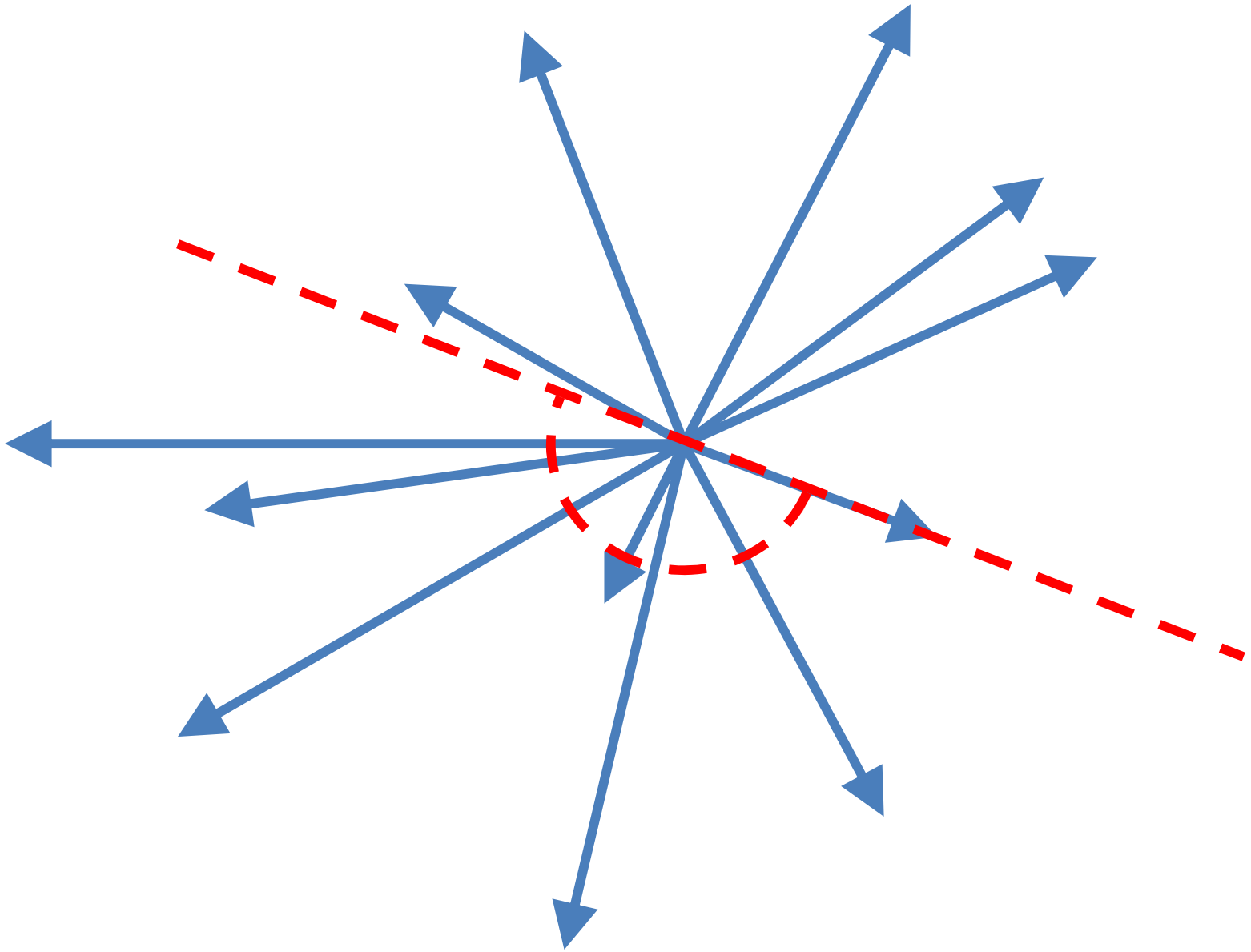



Negative contribution
Positive contribution









Sorting vectors by angle

$\theta = \arctan(y/x)$

C++/Java:

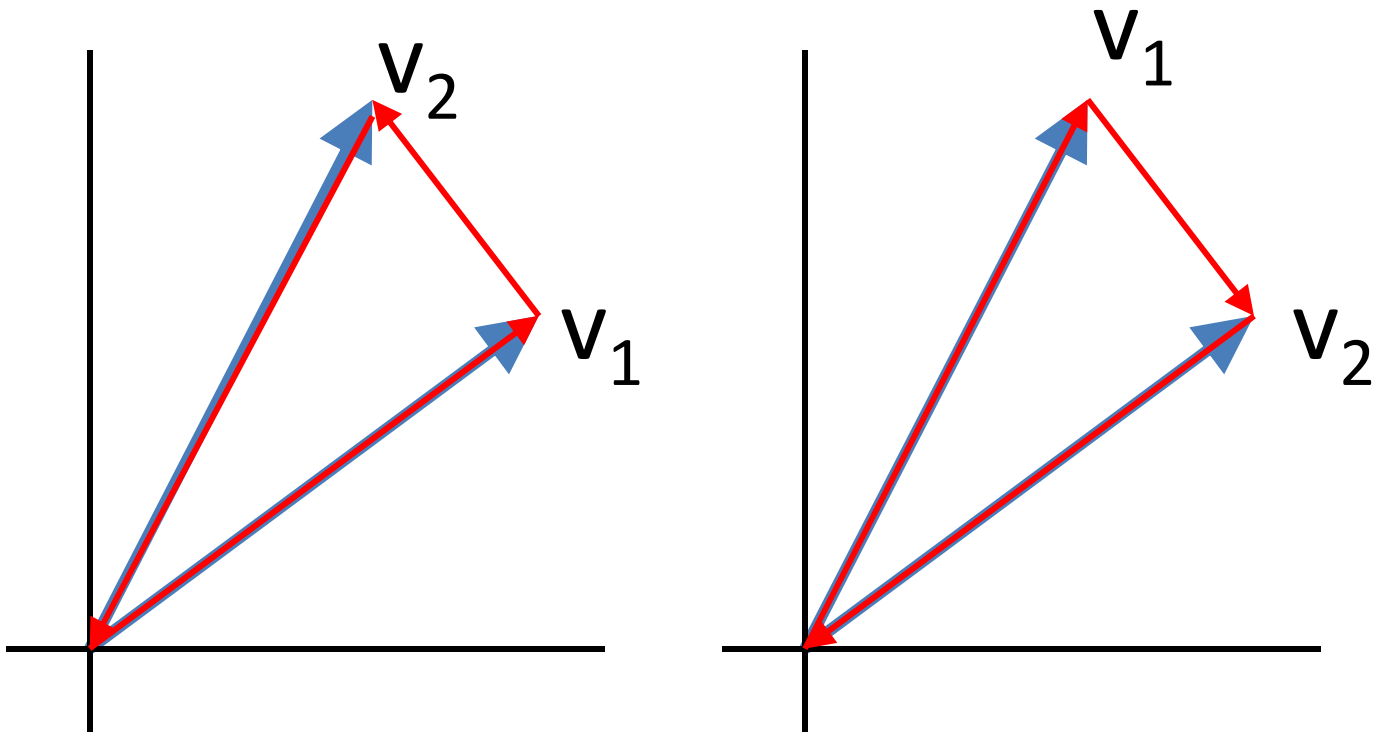
atan(y/x) returns $-\pi/2$ ... $+\pi/2$

atan2(y, x) returns $-\pi$ ... $+\pi$

Sorting vectors by angle

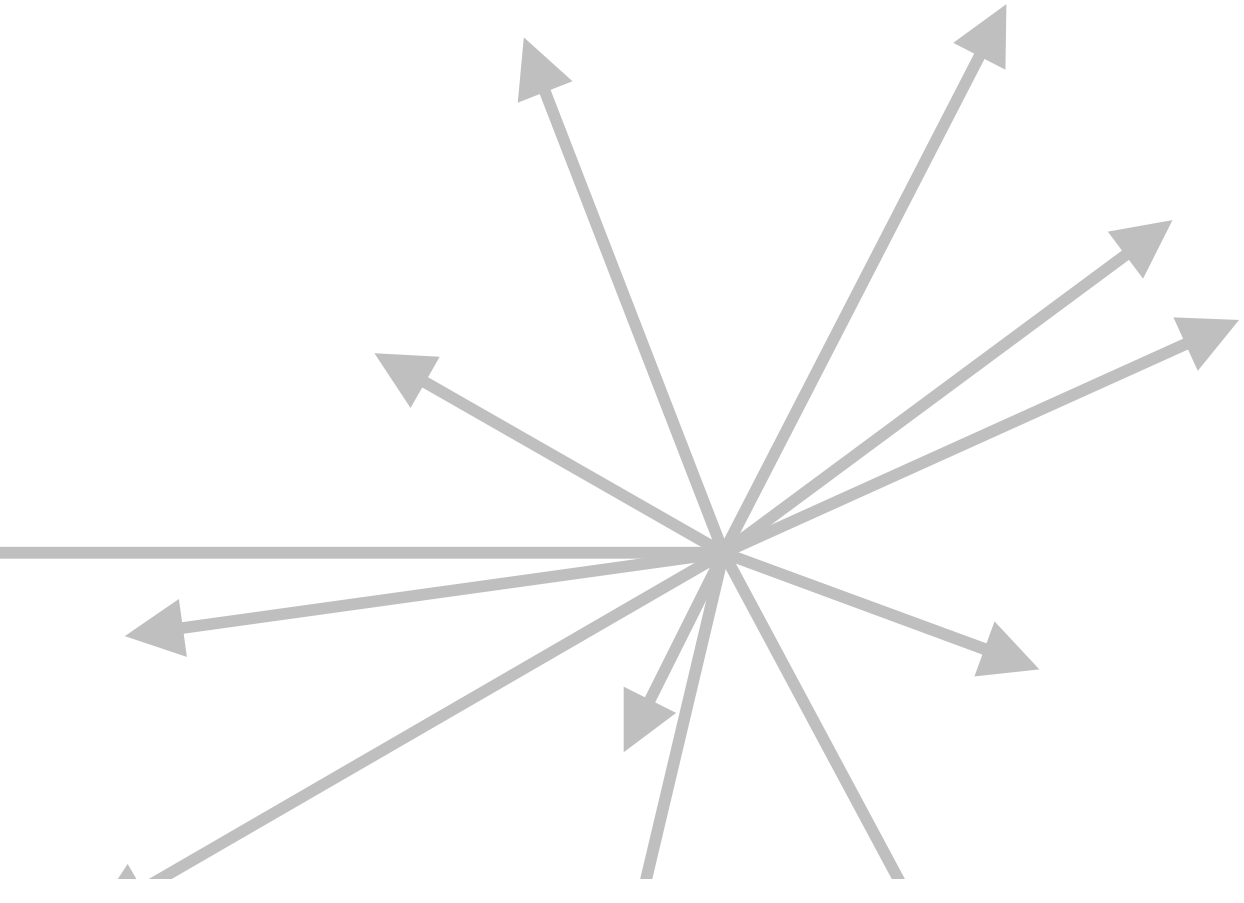
comparing v_1 and v_2 :

- Different quadrants: simple
- For same quadrants: compare turns

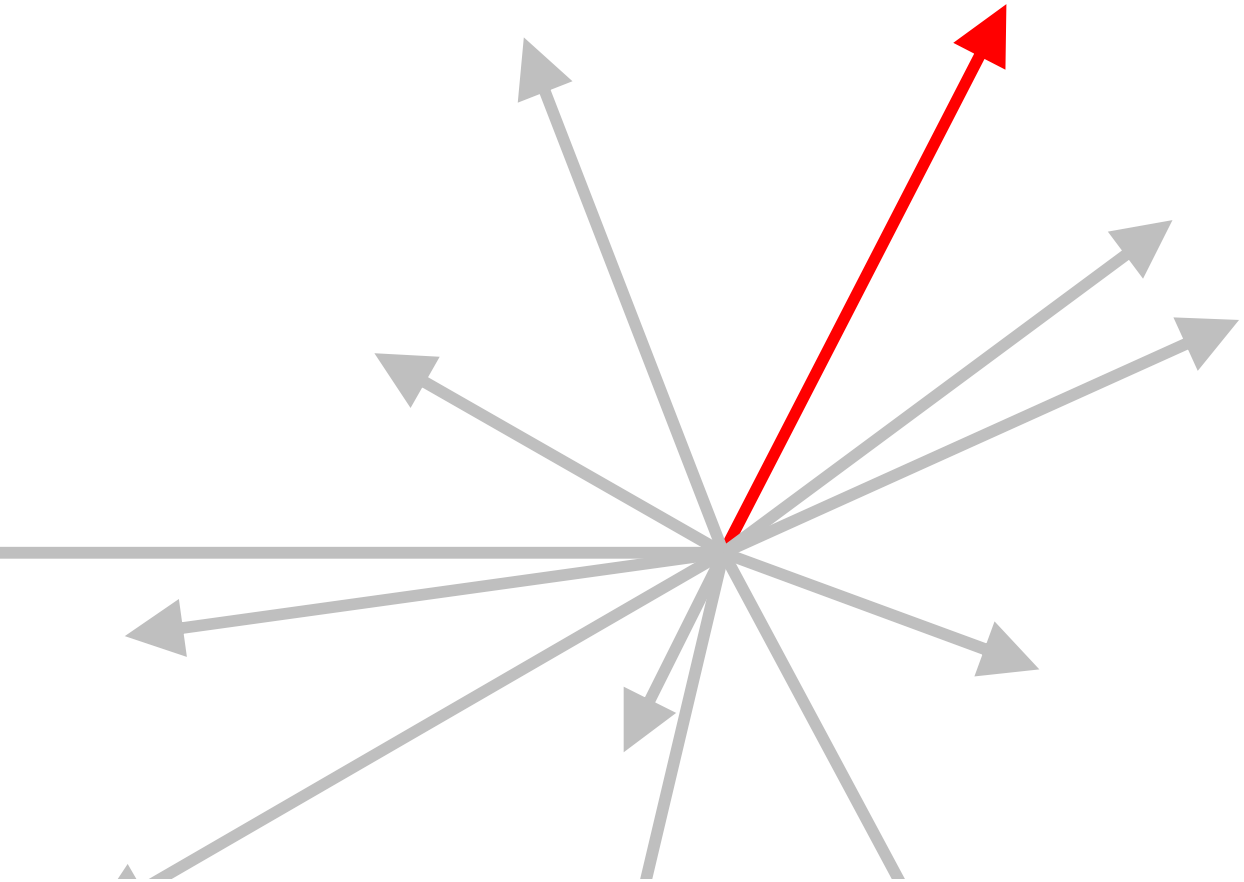


All integer calculations!

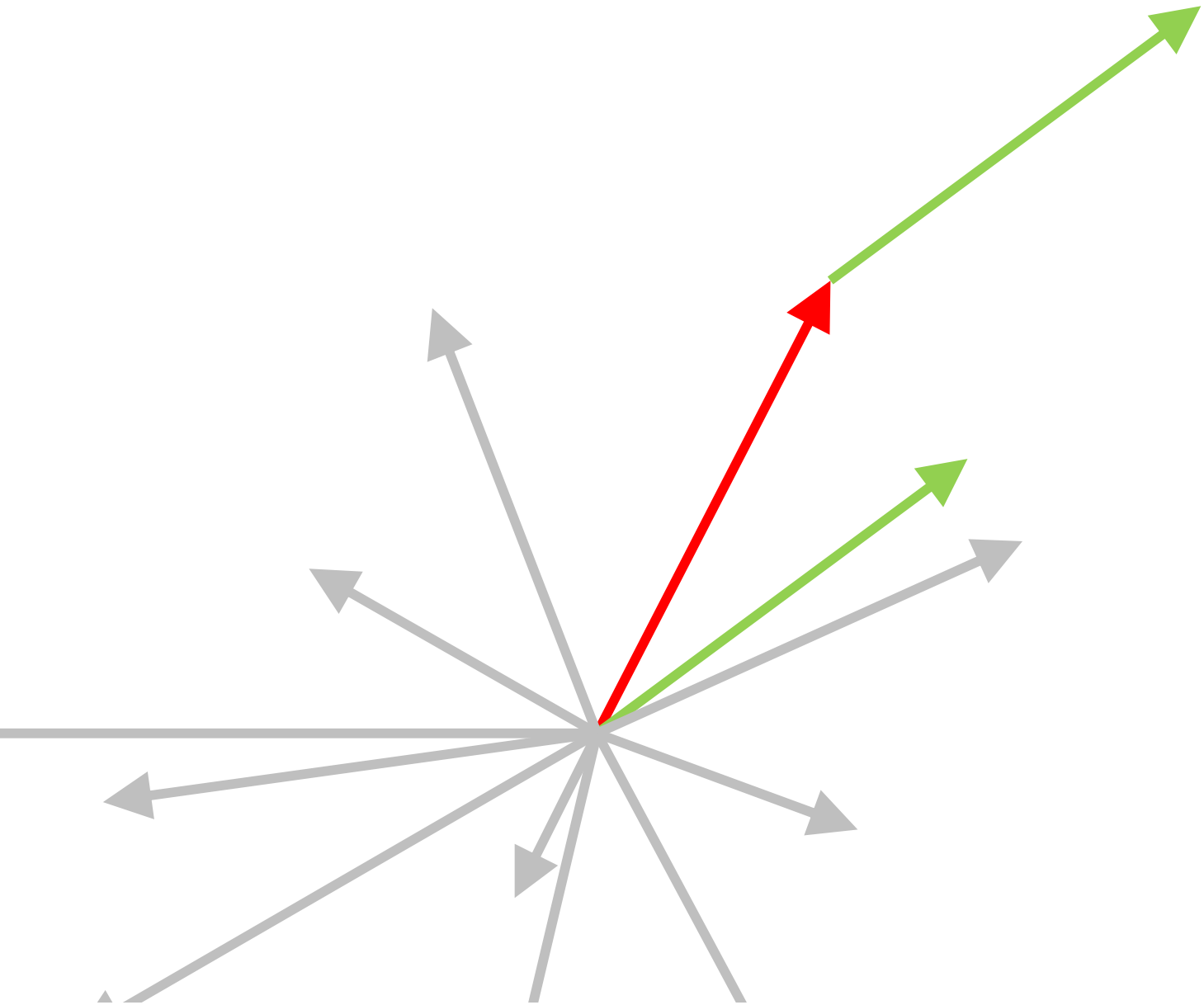
Best sum



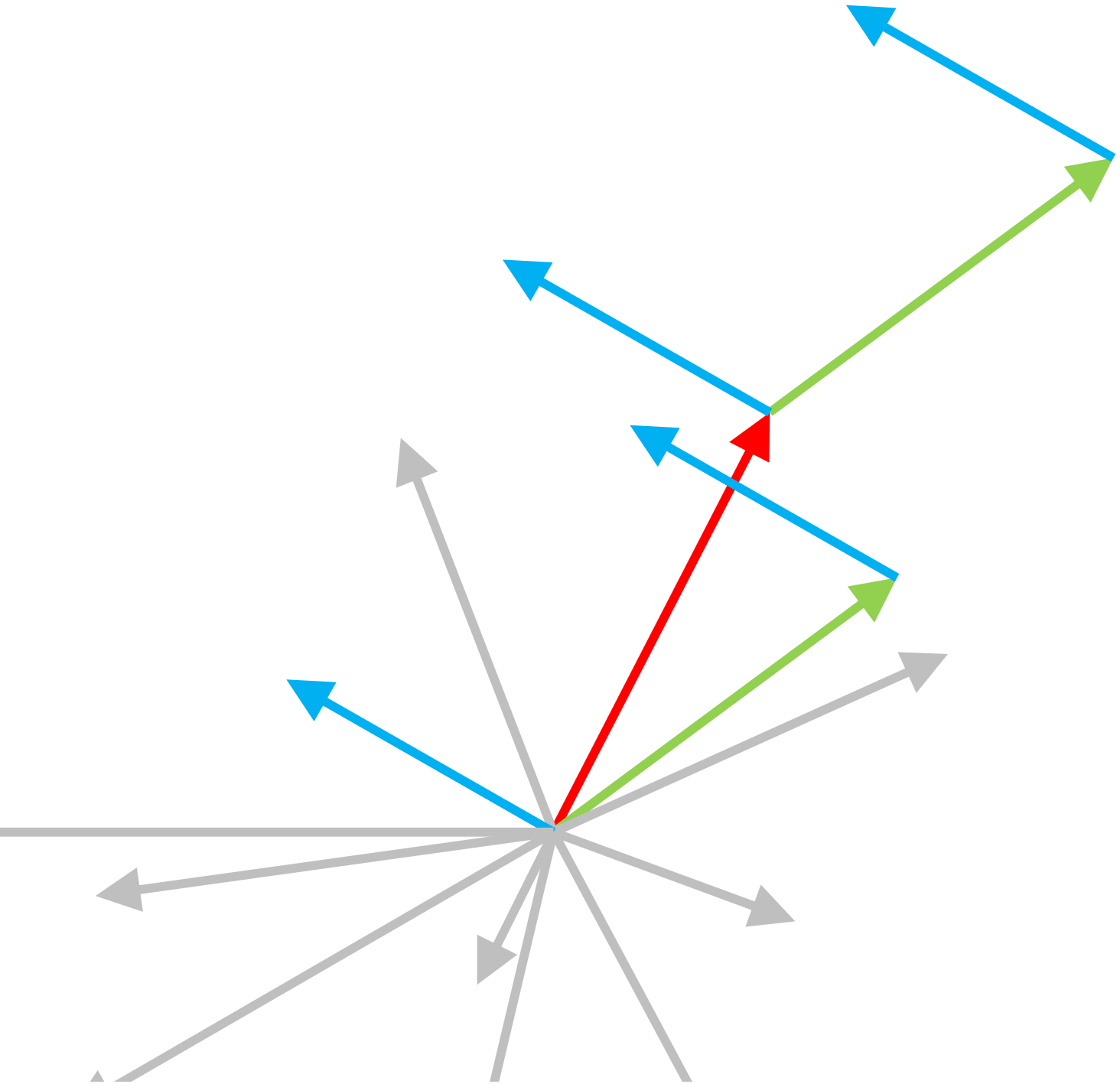
Best sum



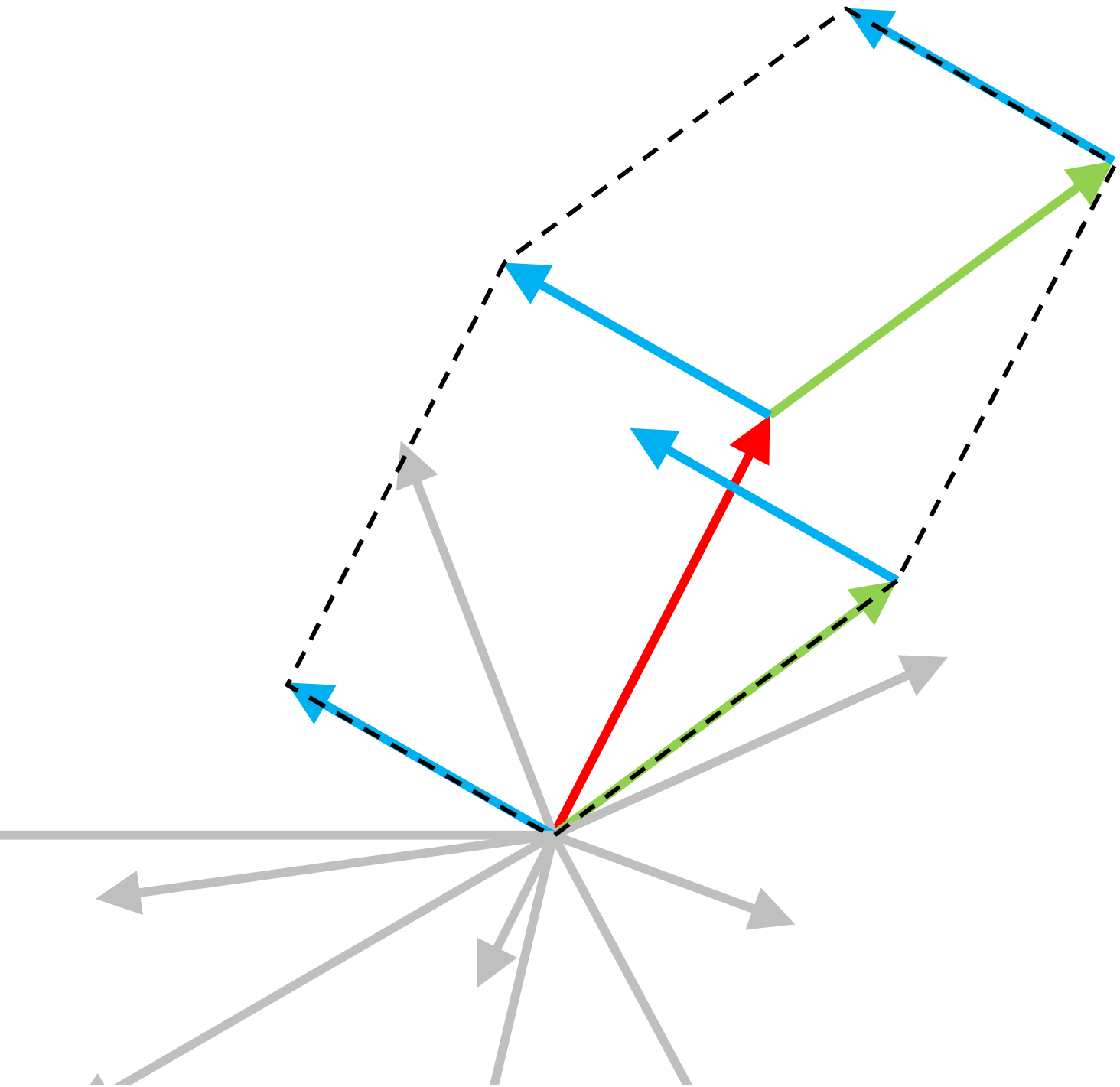
Best sum



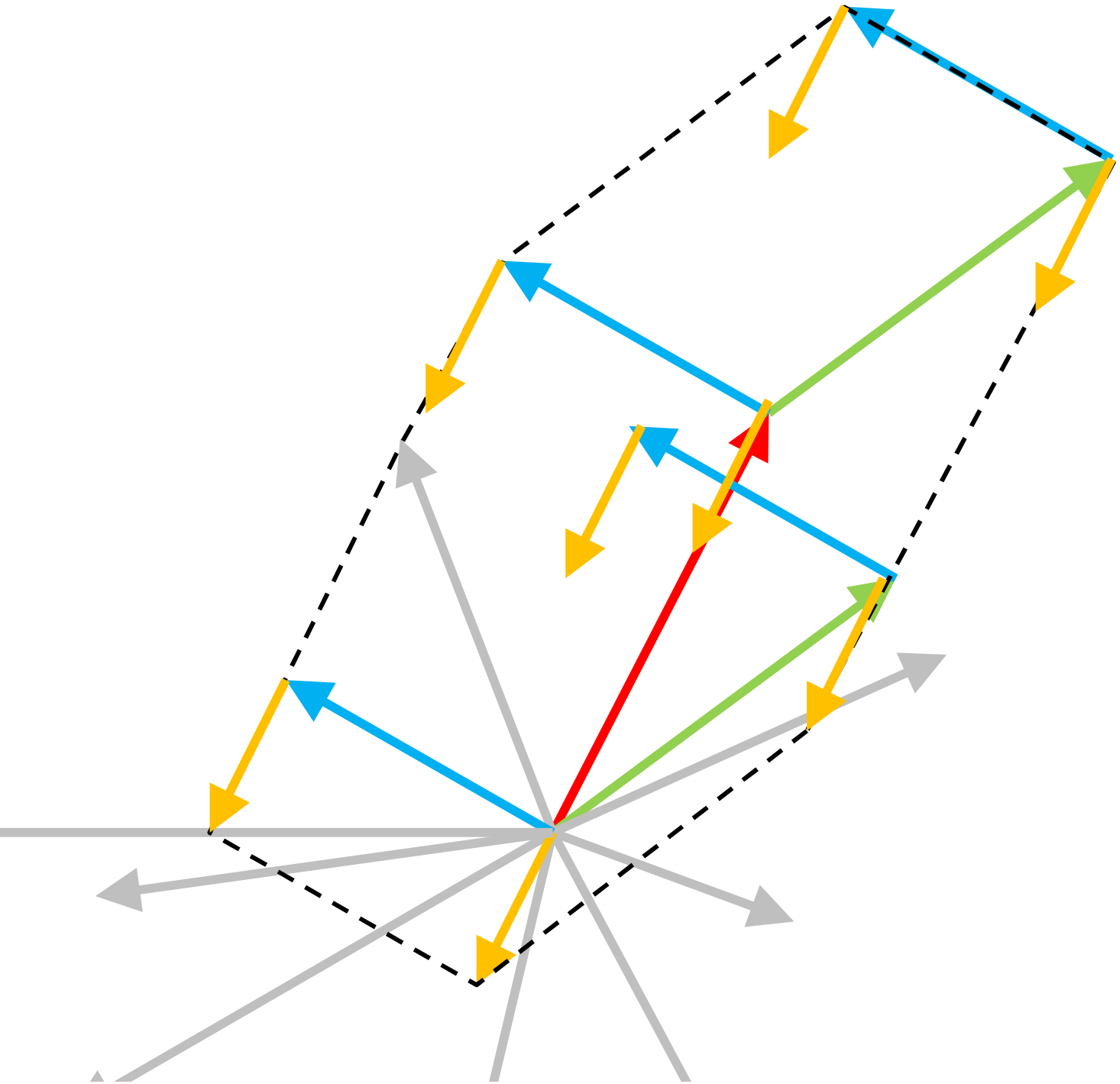
Best sum



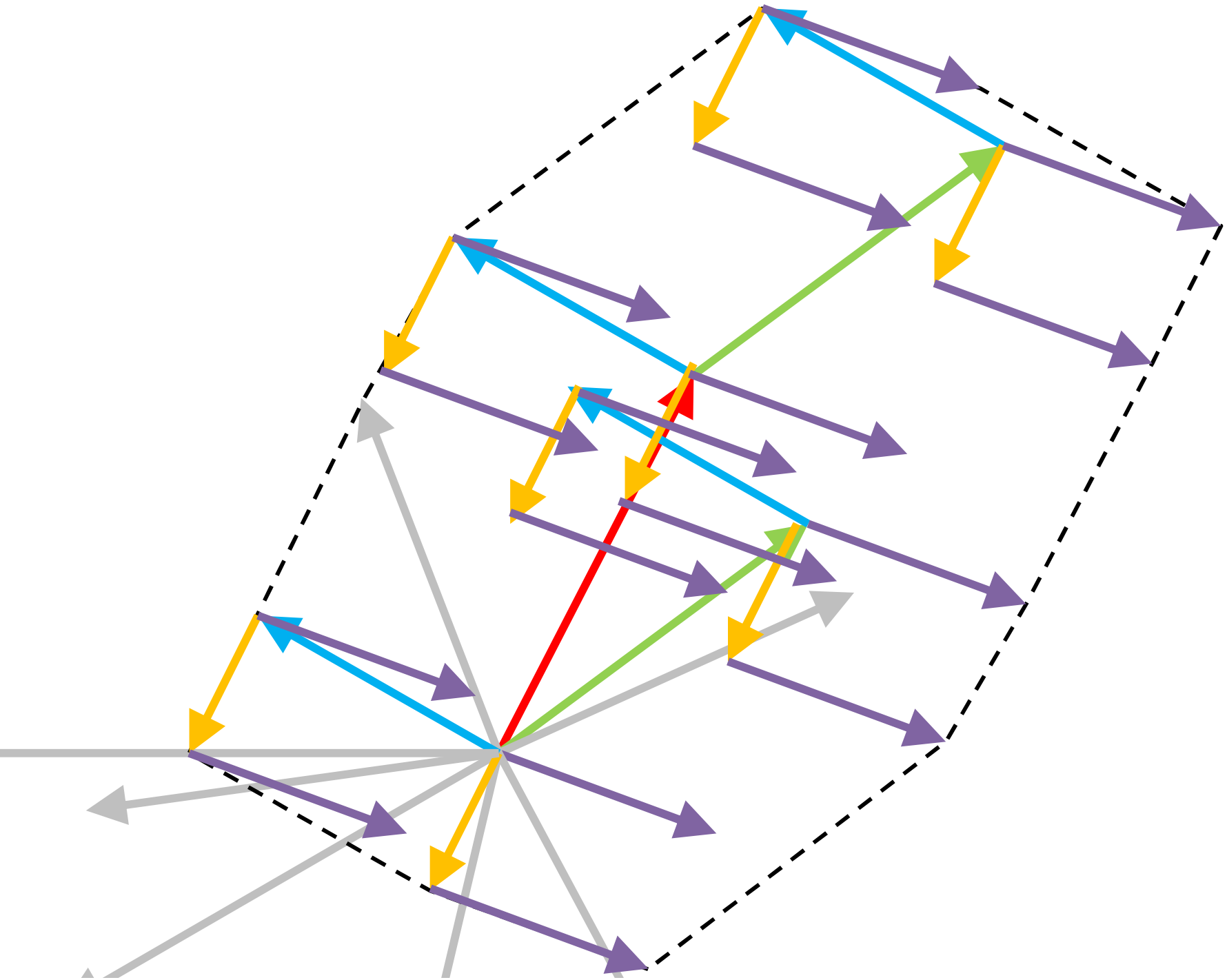
Best sum



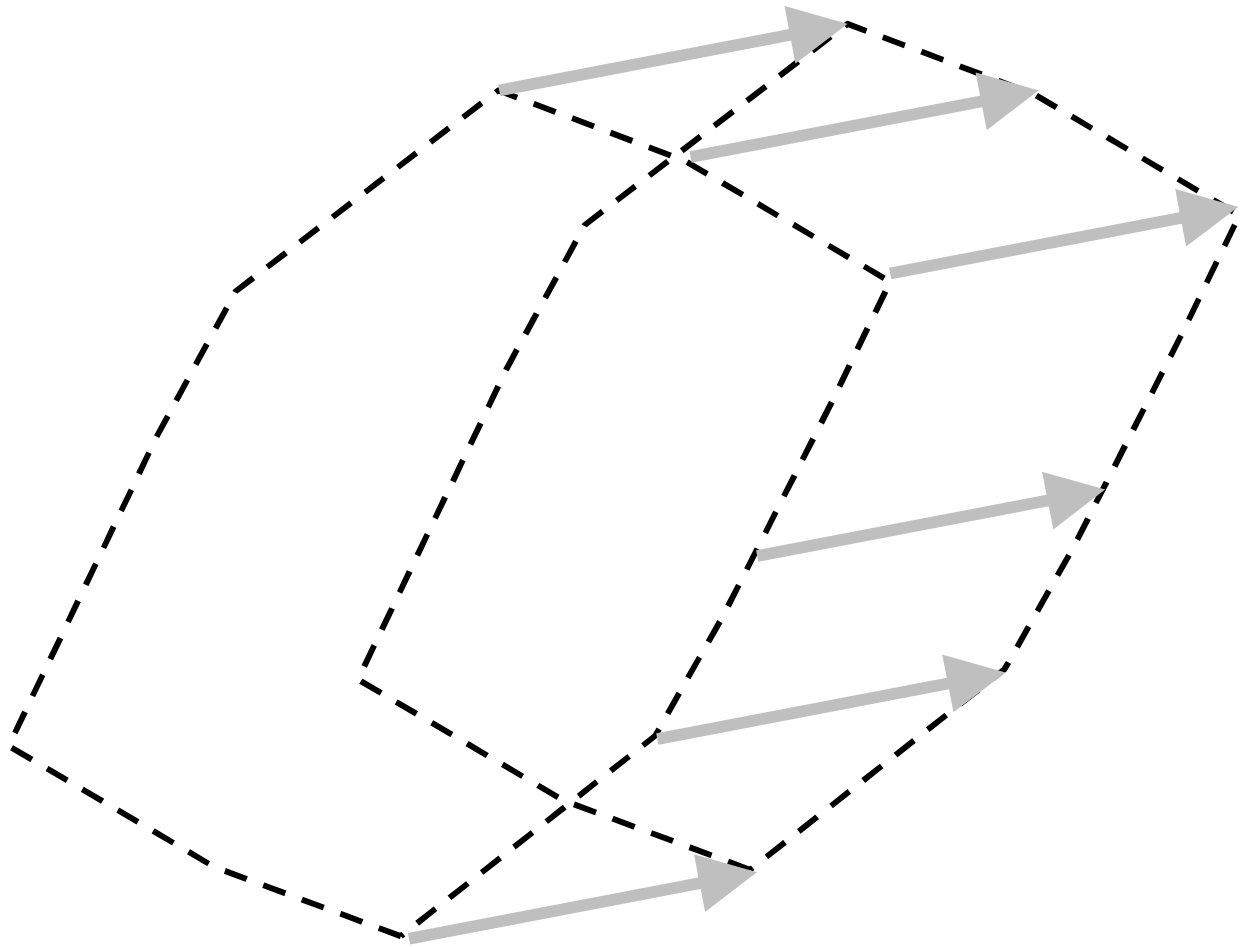
Best sum



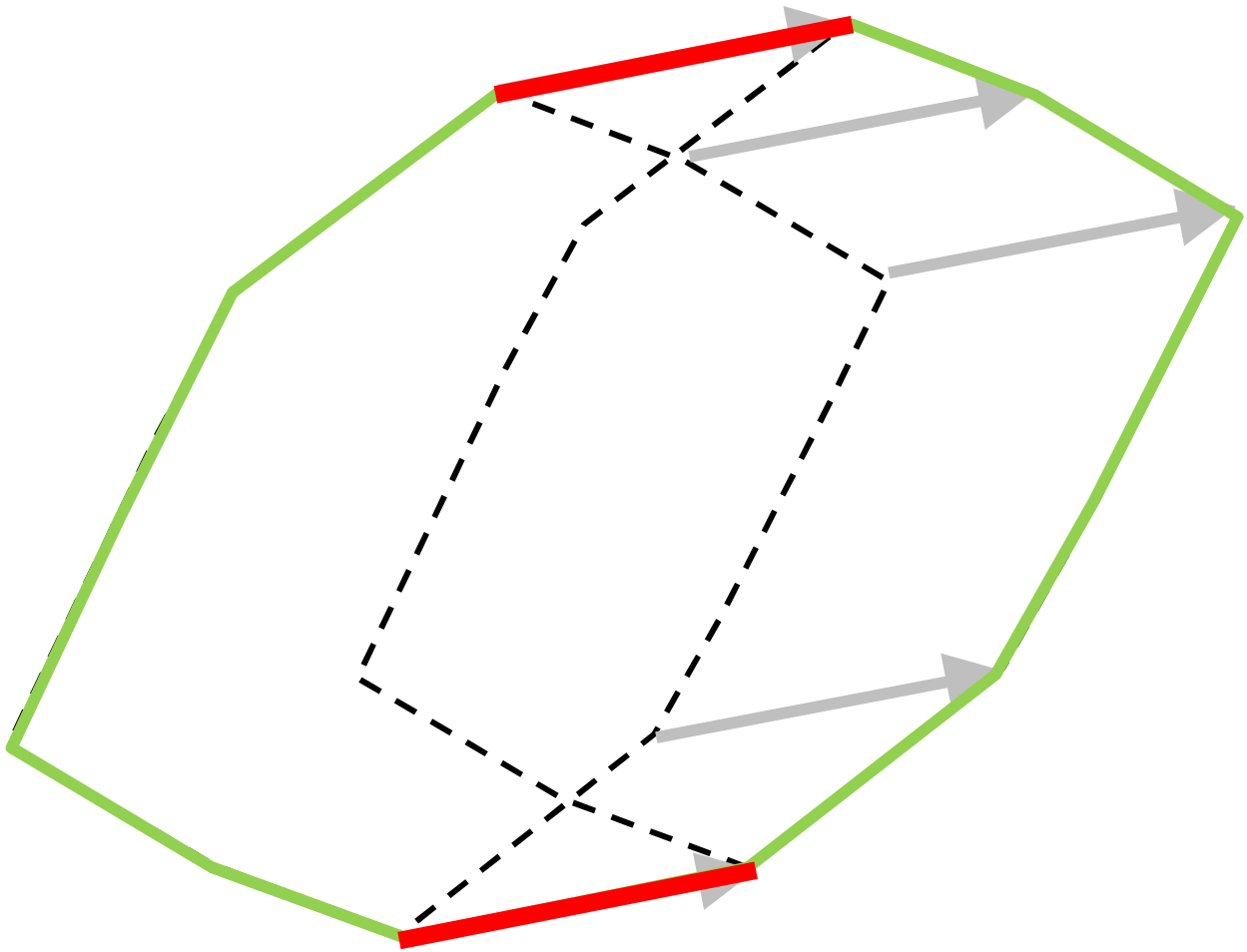
Best sum



Max points after each iteration: 2*i

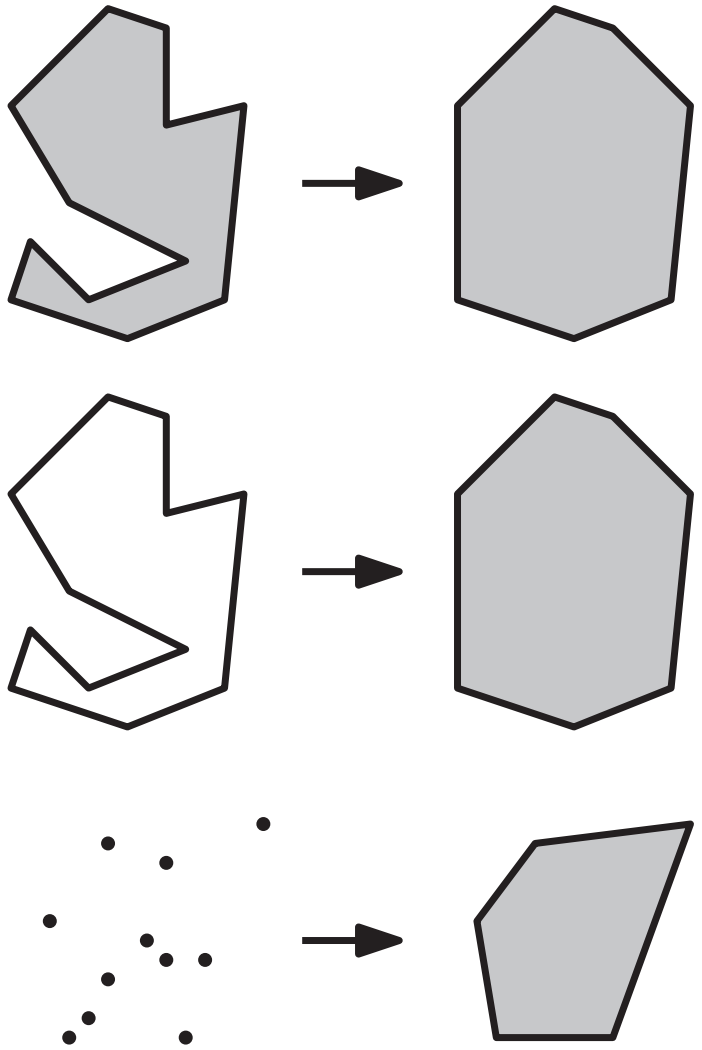


What is the total time complexity?



Convex hull

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

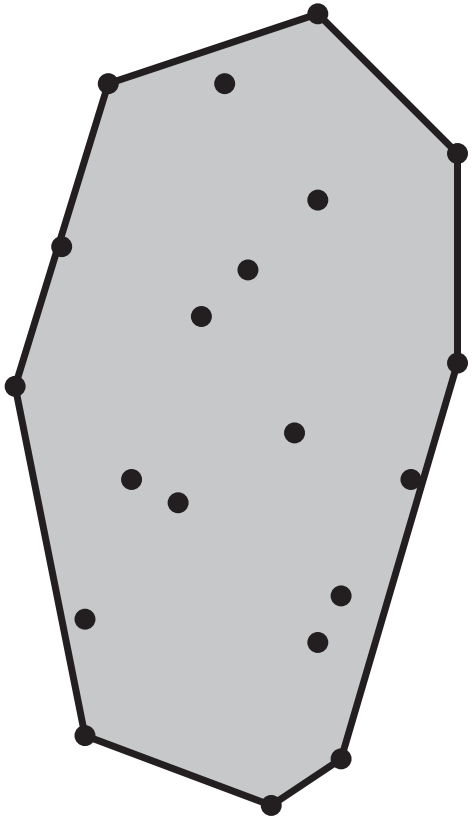


Convex hull problem

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

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

Question: Why can't we expect to do any better than $O(n)$ time?



Convex hull problem

Assume the n points are distinct

The output has at least 4 and at most $2n$ coordinates, so it has size between $O(1)$ and $O(n)$

The output is a convex polygon so it should be returned as a sorted sequence of the points, clockwise (CW) along the boundary

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

Naive approach

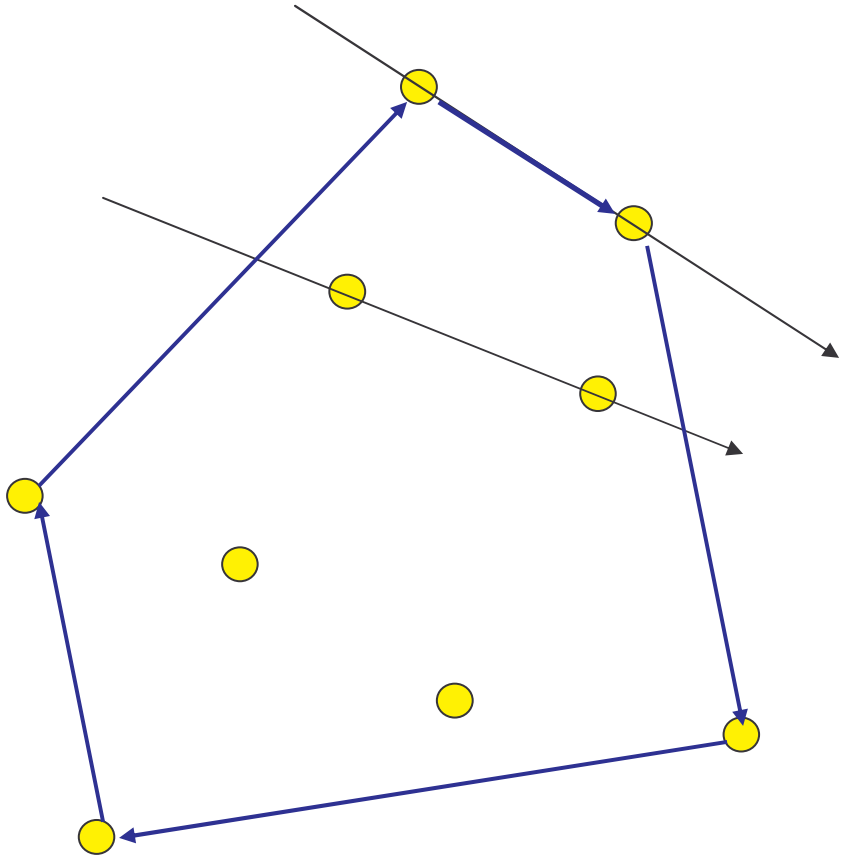
1. For all pairs (p,q) of points in P

/* Check if $p \rightarrow q$ forms a boundary edge

   A. For all points $r \in P-\{p,q\}$:
      - If r lies to the left of directed line $p \rightarrow q$, then go to Step 2
   B. Add (p,q) to the set of edges E

2. Endfor
3. Order the edges in E to form the boundary of $CH(P)$

Example



Details

- How to test if r lies to the left of a directed line $p \rightarrow q$?
  - Basic geometric operation
  - Reduces to checking the sign of a certain determinant
  - Constant time operation
- How to order the edges in E ?
  - Sort

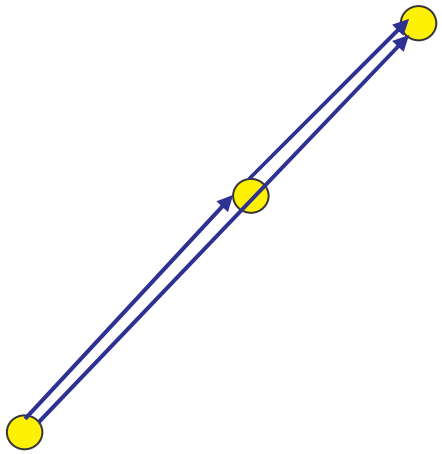
Analysis

- Outer loop: $O(n^2)$ repetitions
- Inner loop: $O(n)$ repetitions
- Total time: $O(n^3)$

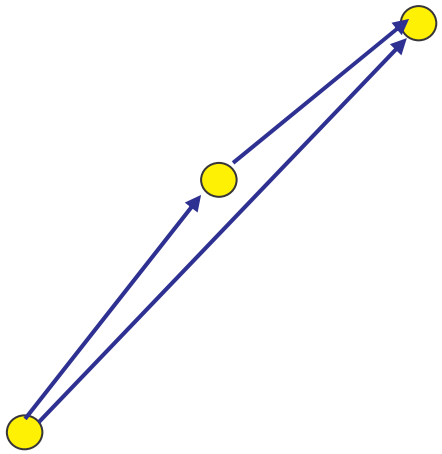
Problems

- Running time pretty high
- Algorithm does weird things:
  - What 3 points are collinear ? (degeneracy)
  - What 3 points are near-collinear ? (robustness)
- The last issue highly non-trivial
- Many ways of dealing with it:
  - Higher precision
  - Arbitrary precision
- Our approach: sweep it under the carpet

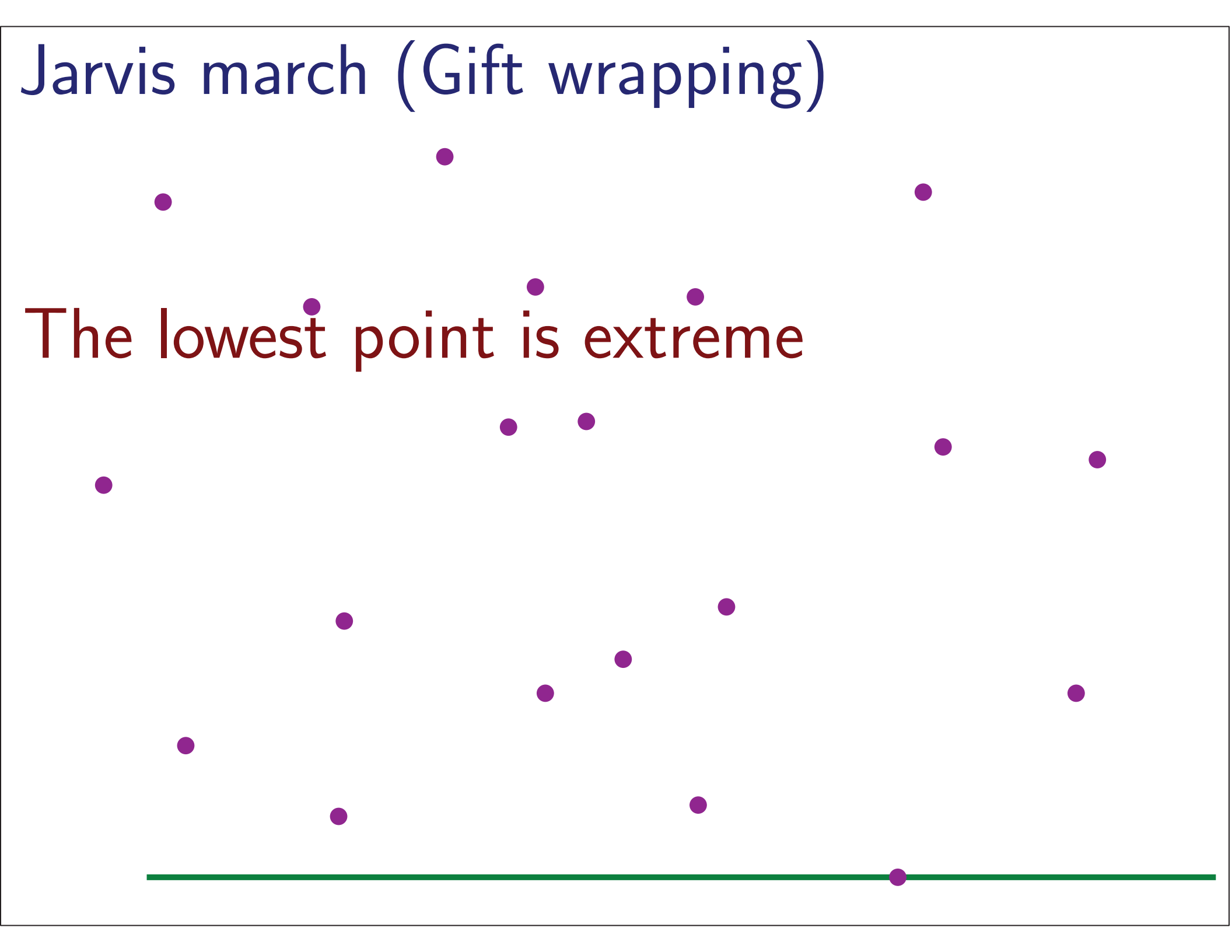
Collinear points



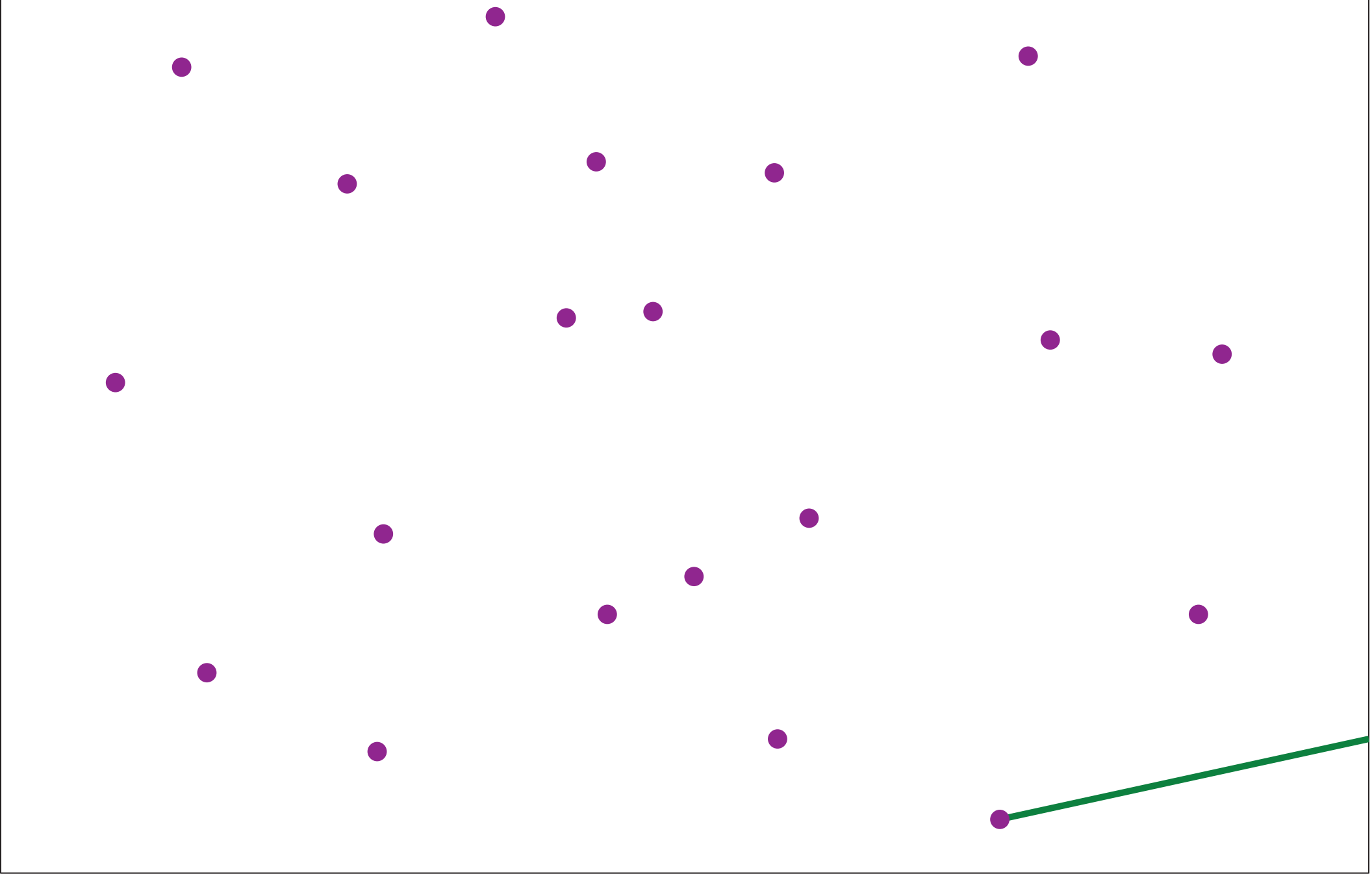
Nearly collinear points



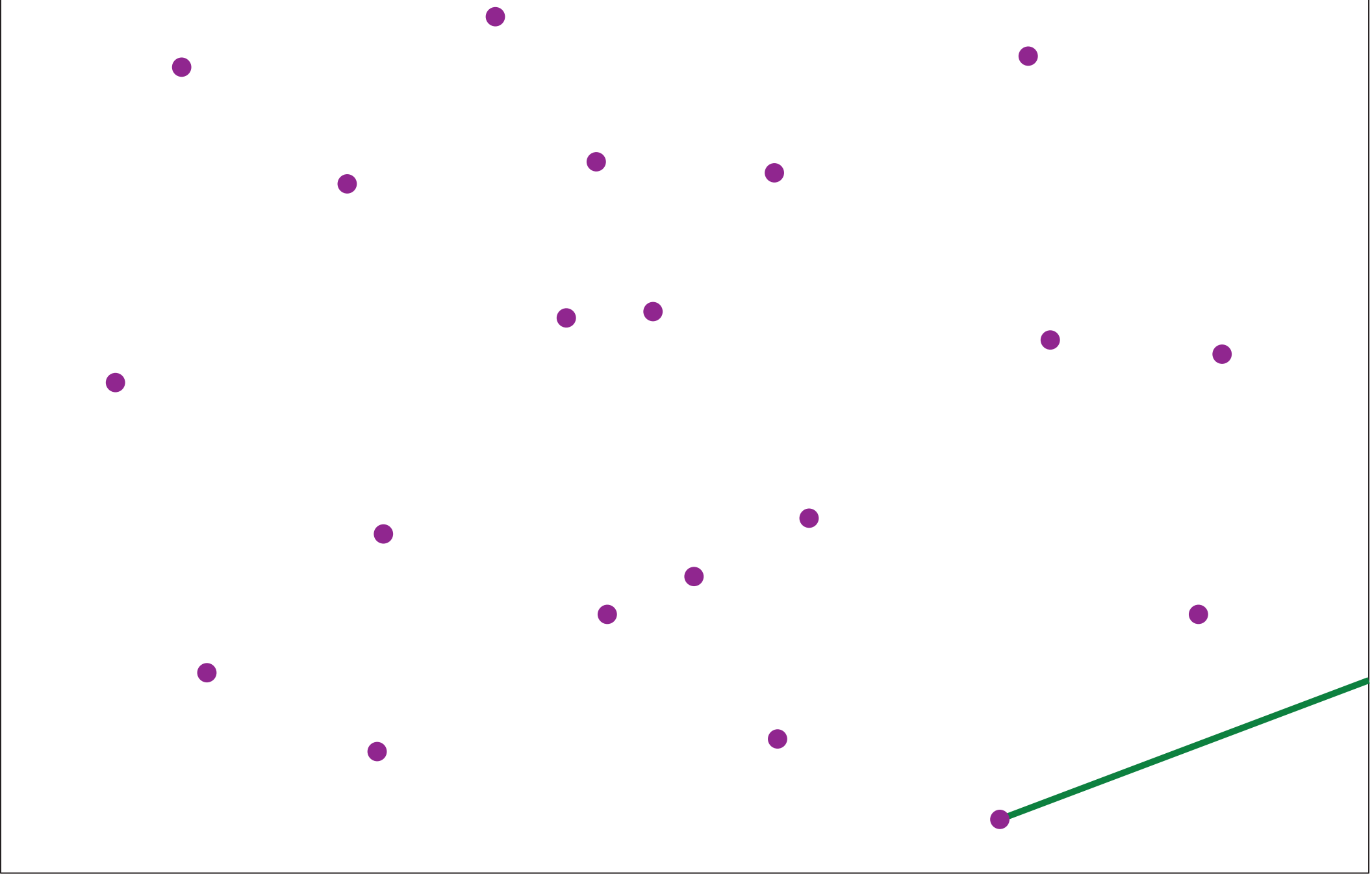
Jarvis march (Gift wrapping)



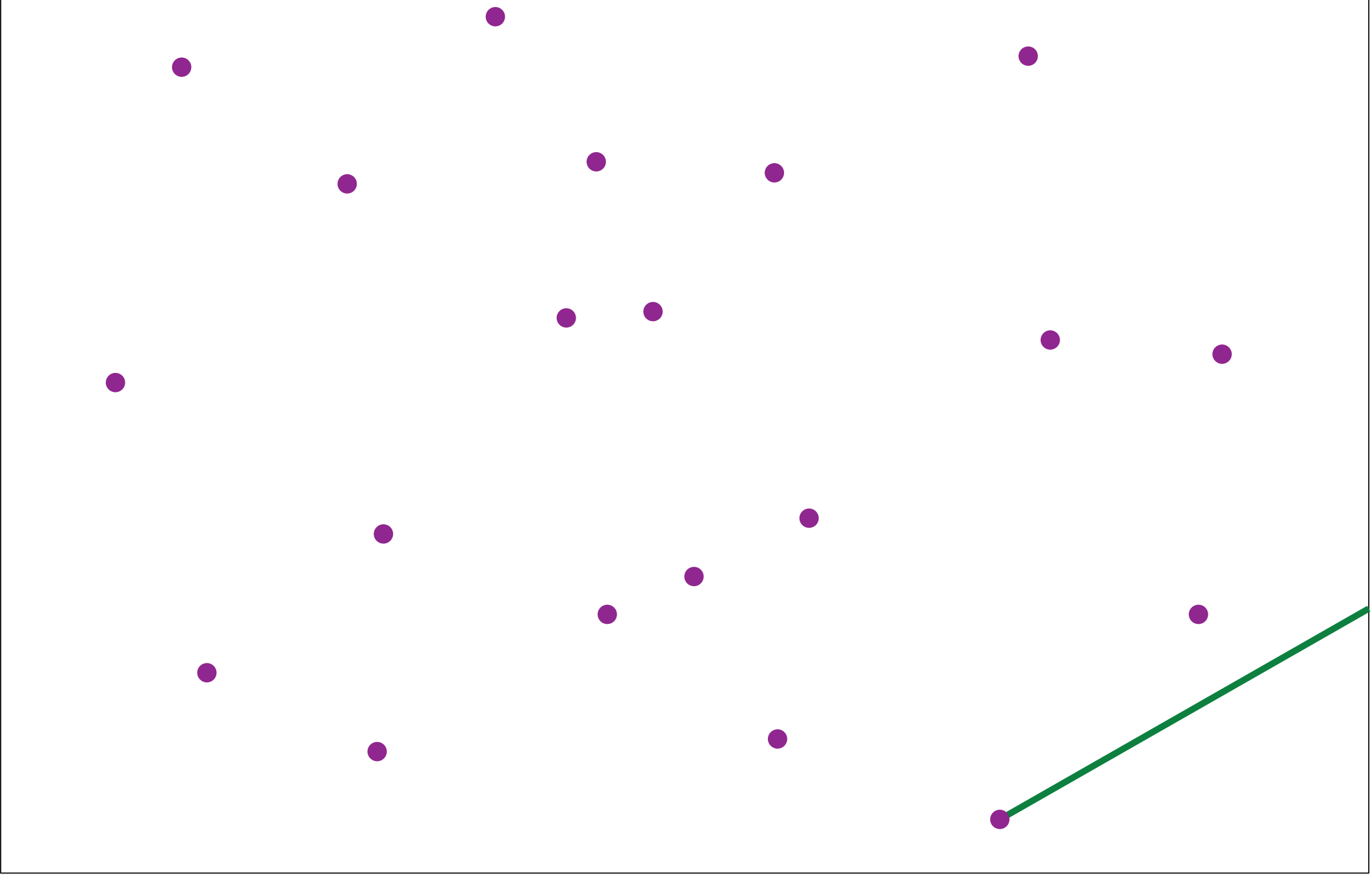
Jarvis march (Gift wrapping)



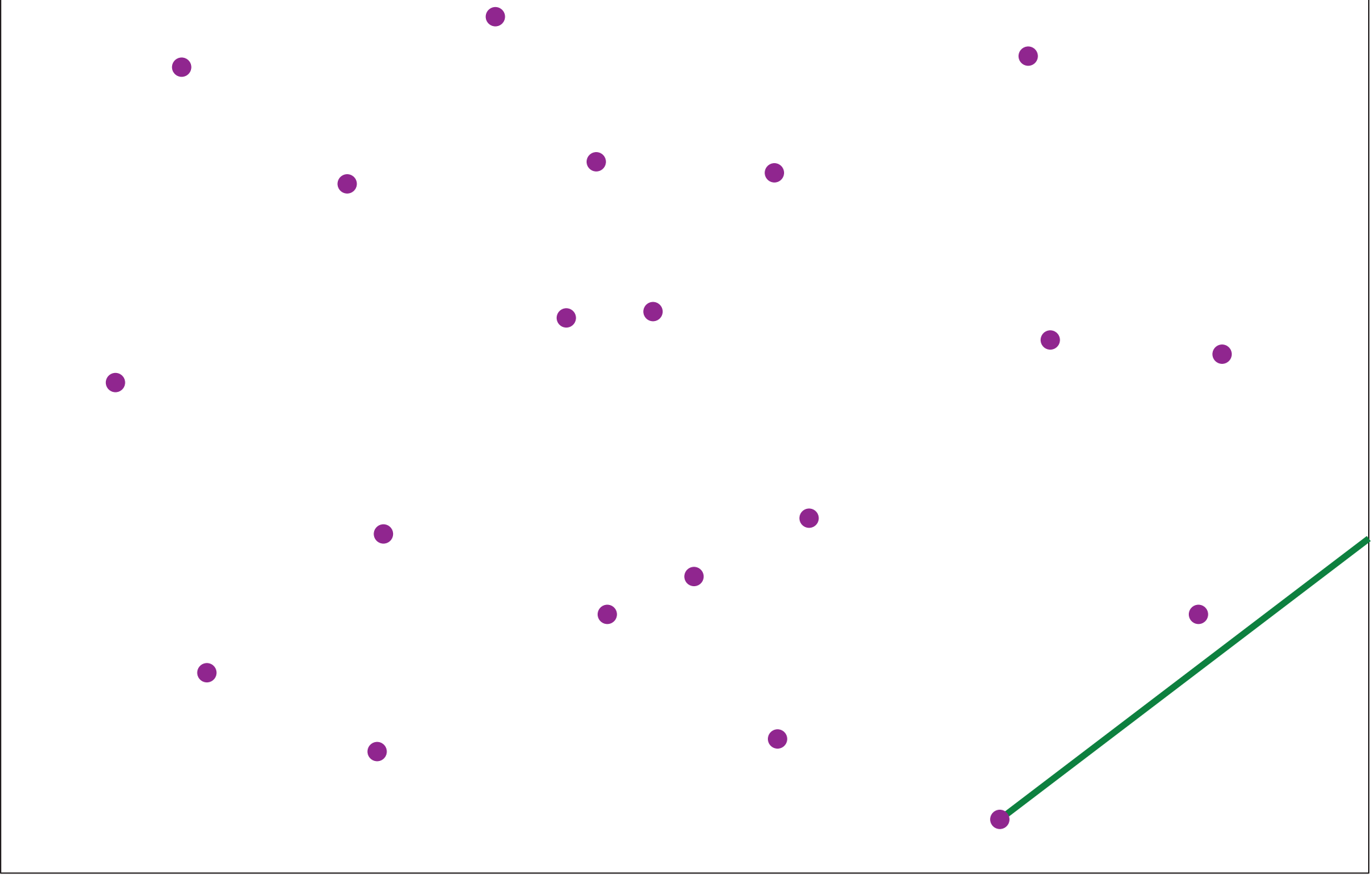
Jarvis march (Gift wrapping)



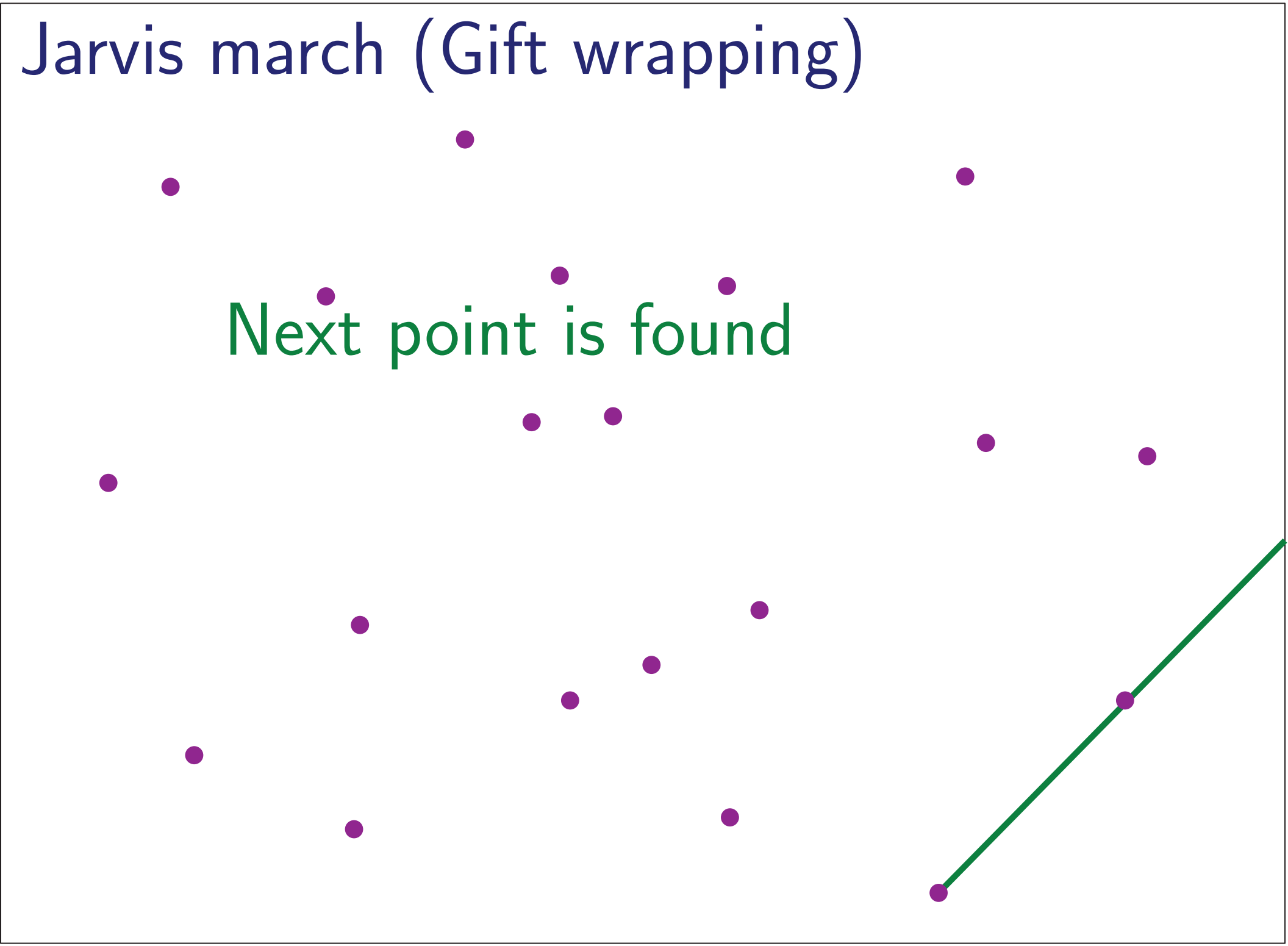
Jarvis march (Gift wrapping)



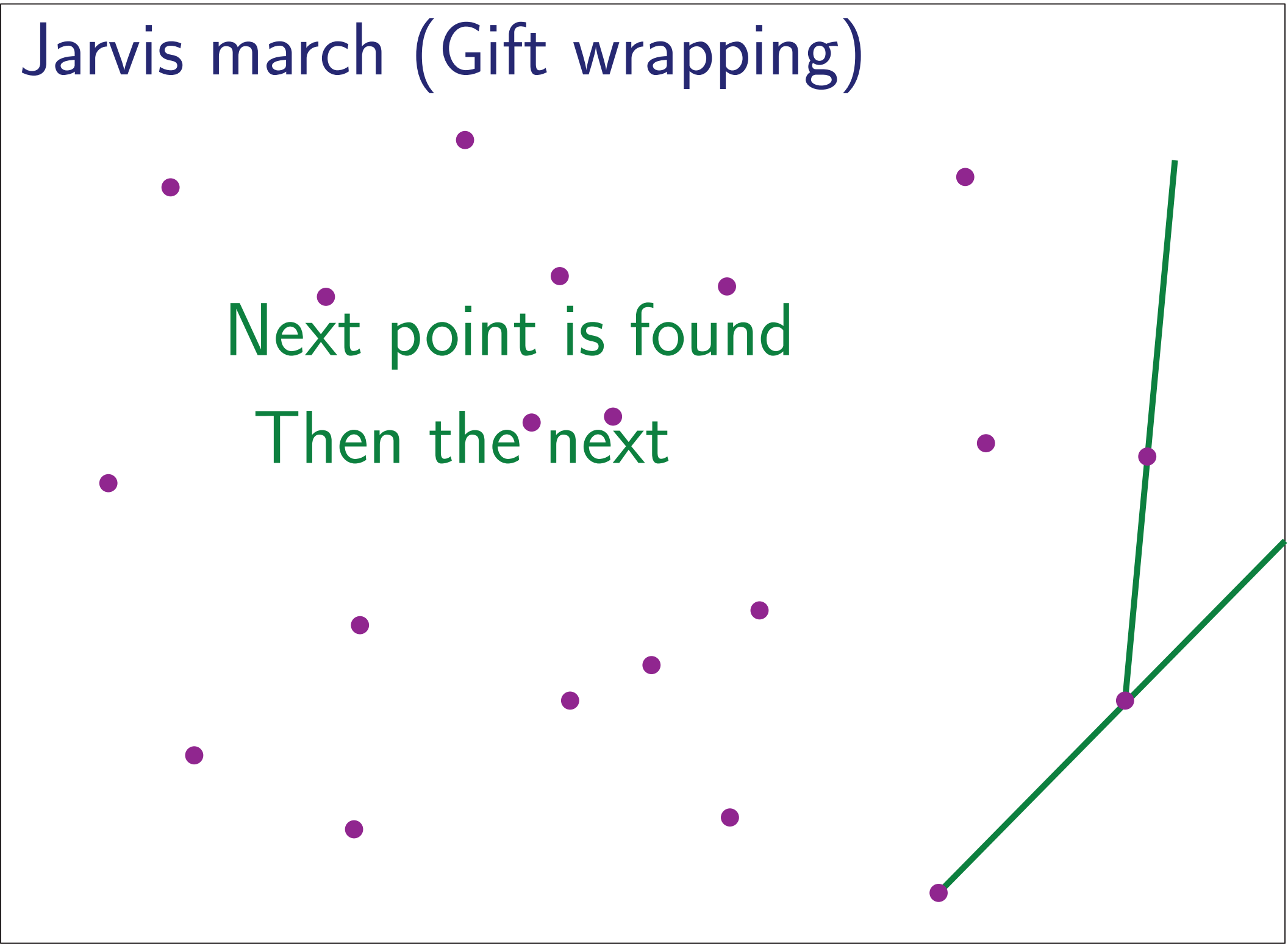
Jarvis march (Gift wrapping)



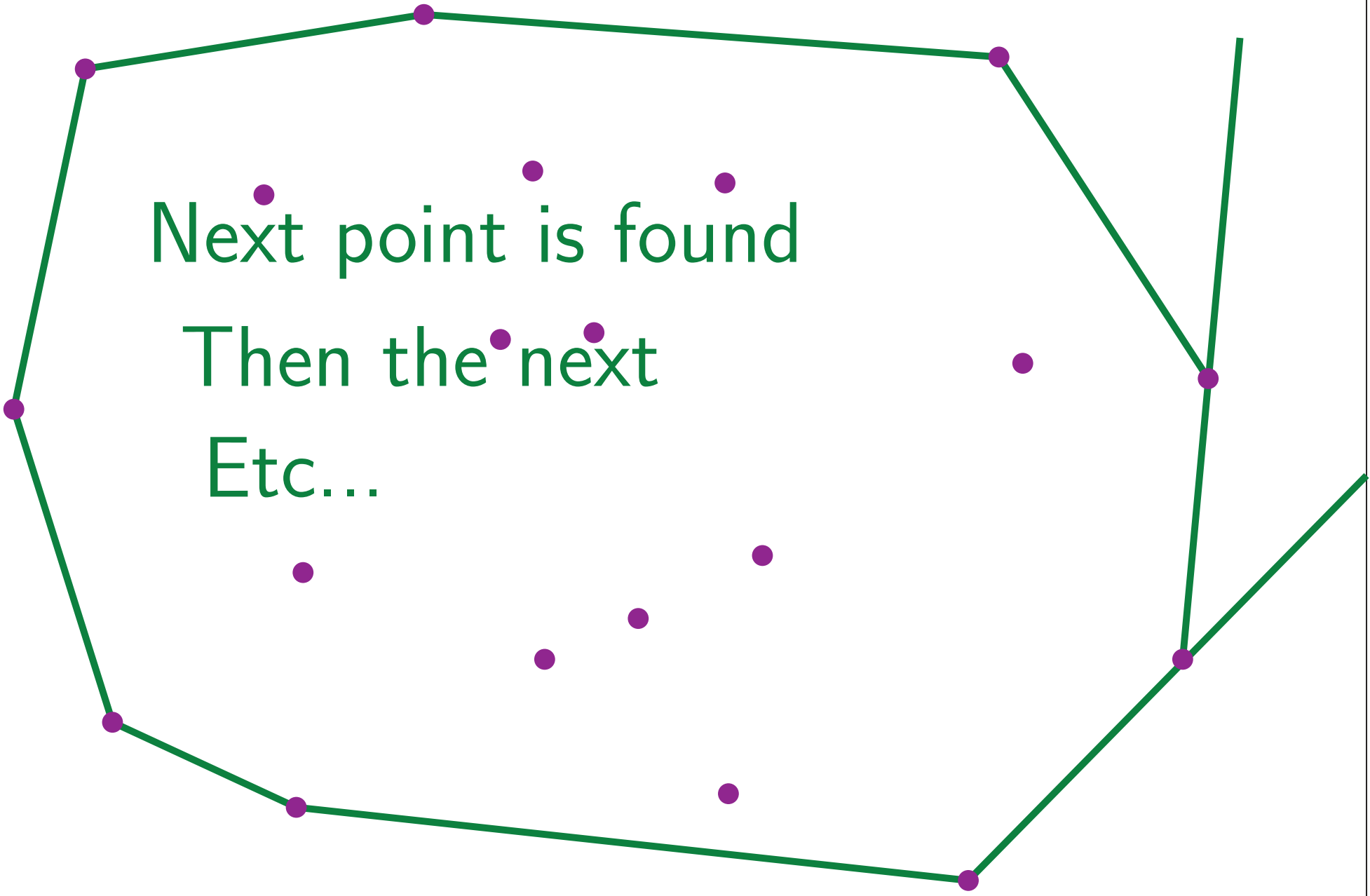
Jarvis march (Gift wrapping)



Jarvis march (Gift wrapping)



Jarvis march (Gift wrapping)



2D Gift wrapping algorithm

Input: polygon $p = \{p_0 \ldots p_n\}$

h_0 = lowest_leftmost_point in p;

for (i = 0; h not closed; i++)

h_{i+1} = rightmost point from h_i in $p - h$;

Return h

2D Gift wrapping complexity

Input: polygon $p = \{p_0 \ldots p_n\}$

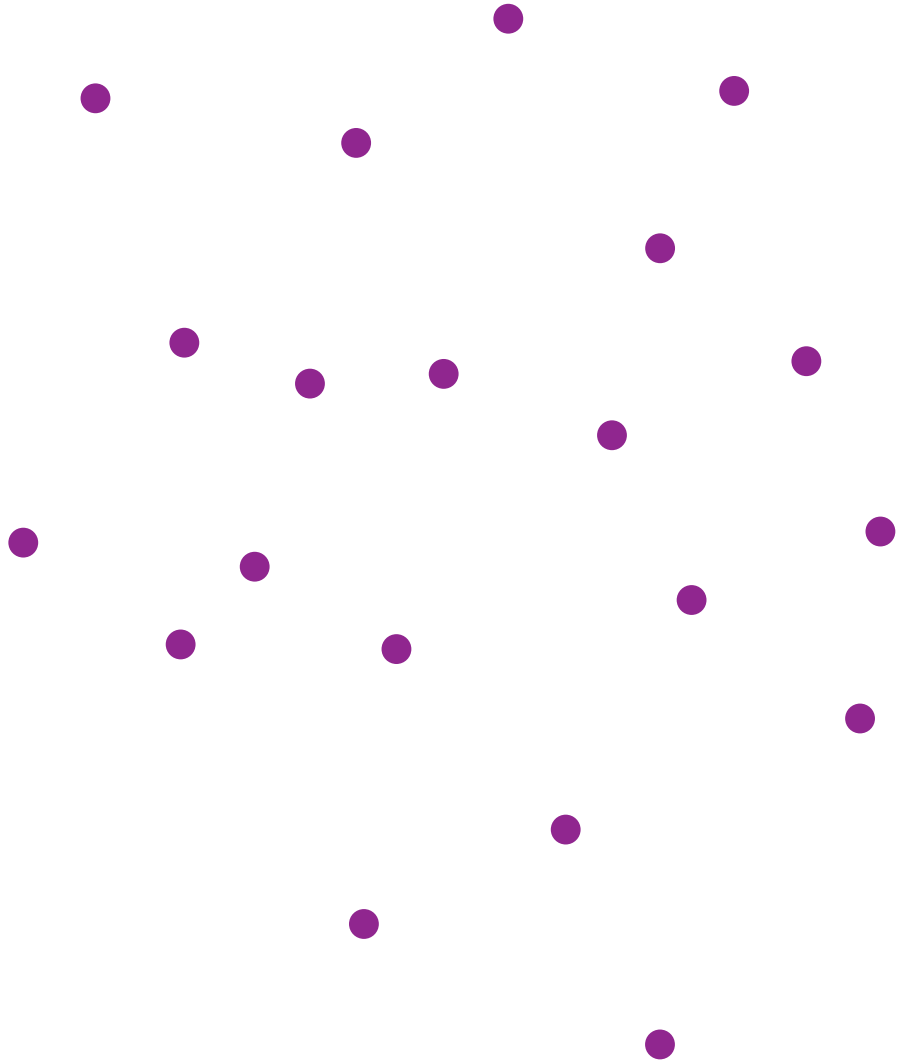
h_0 = lowest_leftmost_point in p;

for (i = 0; h not closed; i++) **O(H)**

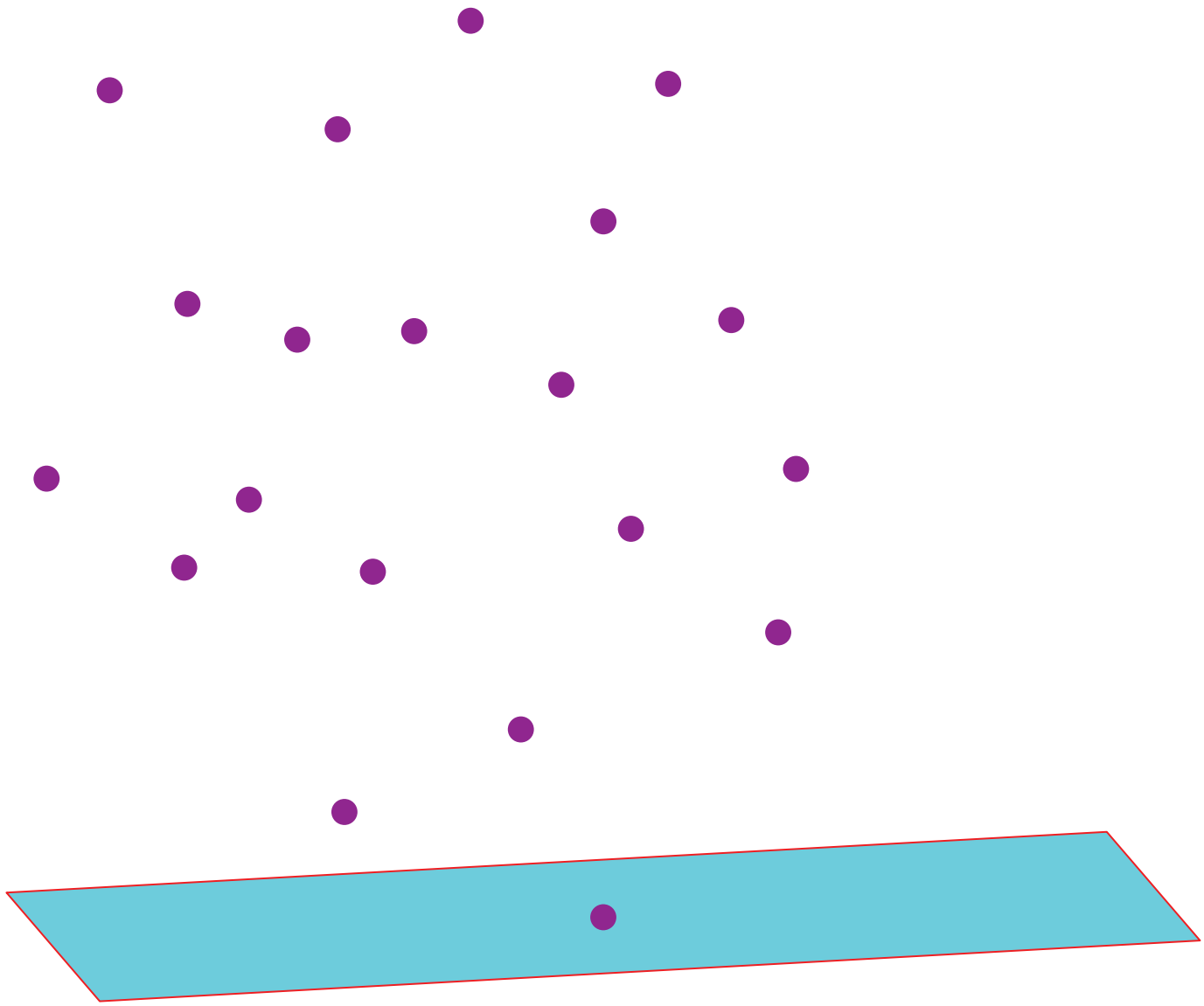
h_{i+1} = rightmost point from h_i in $p - h$; **O(N)**

Return h

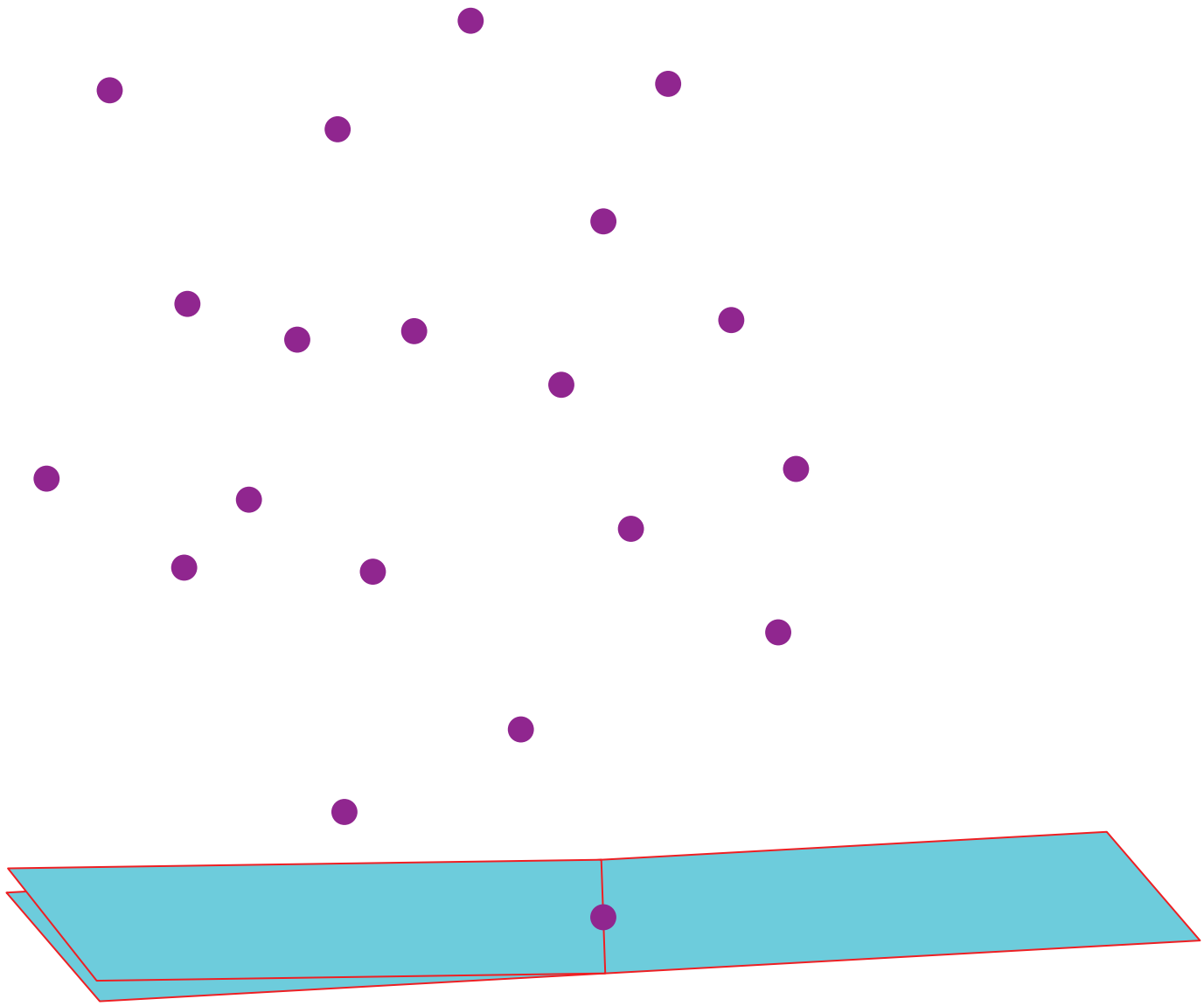
3D: Gift wrapping



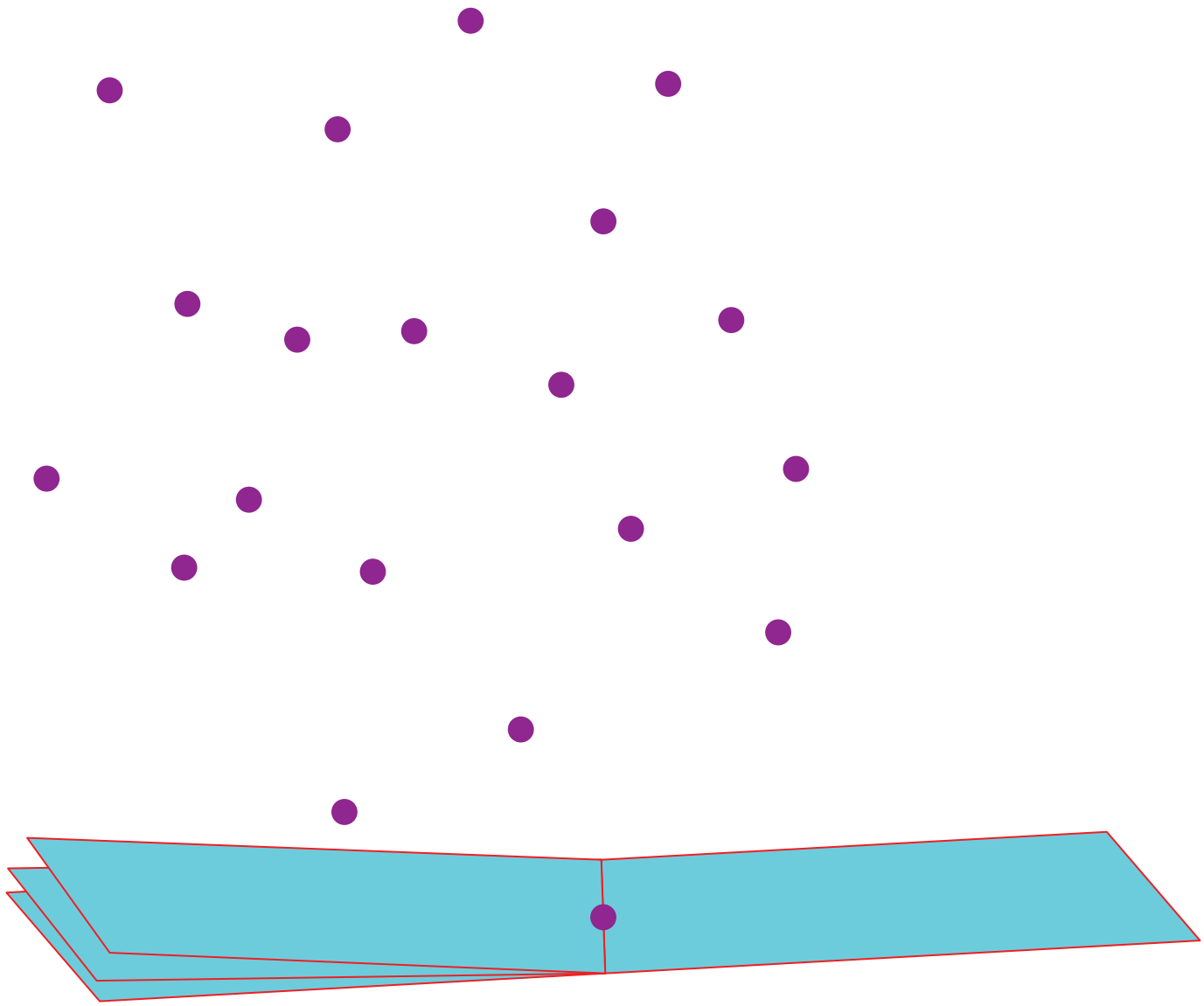
3D: Gift wrapping



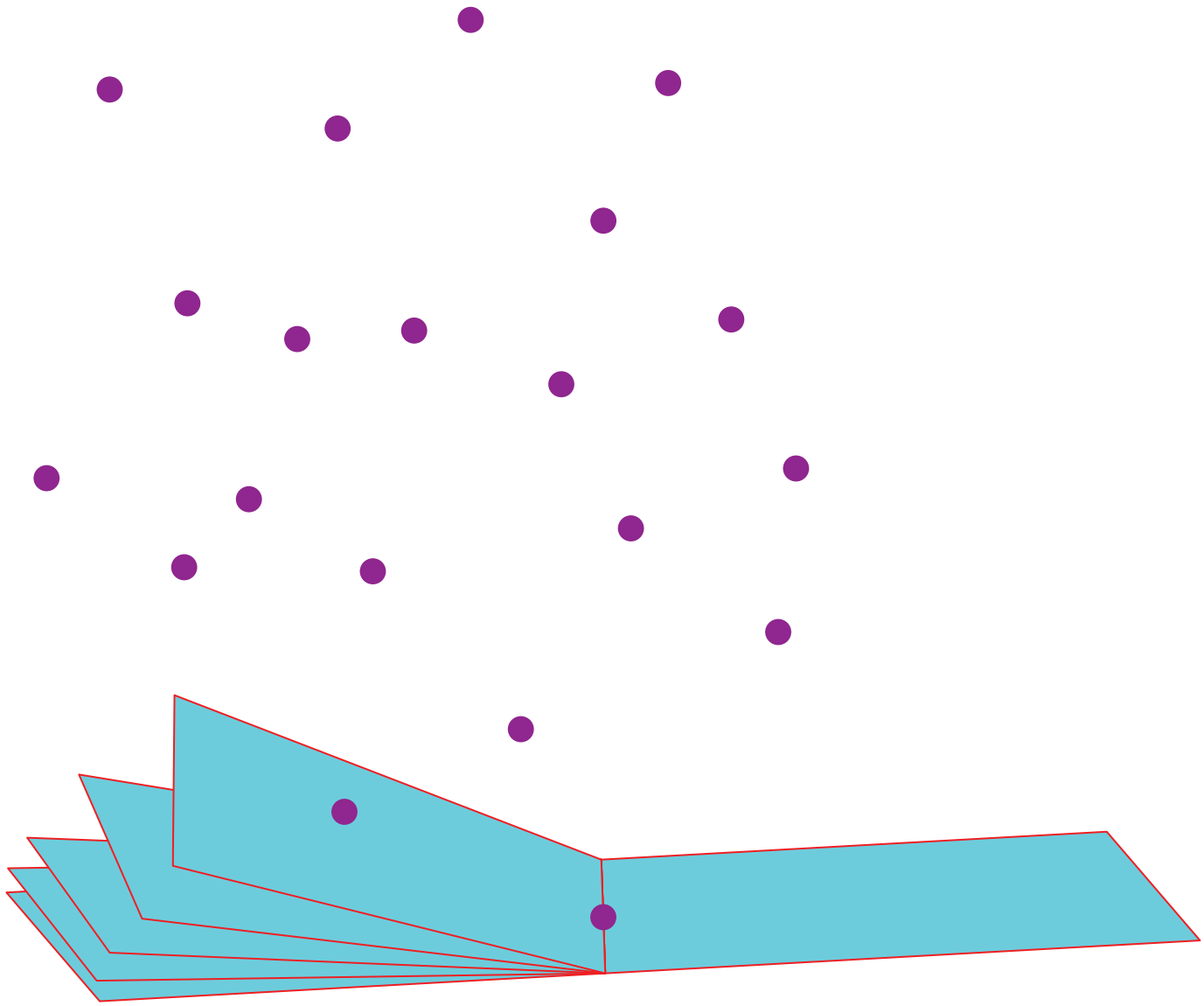
3D: Gift wrapping



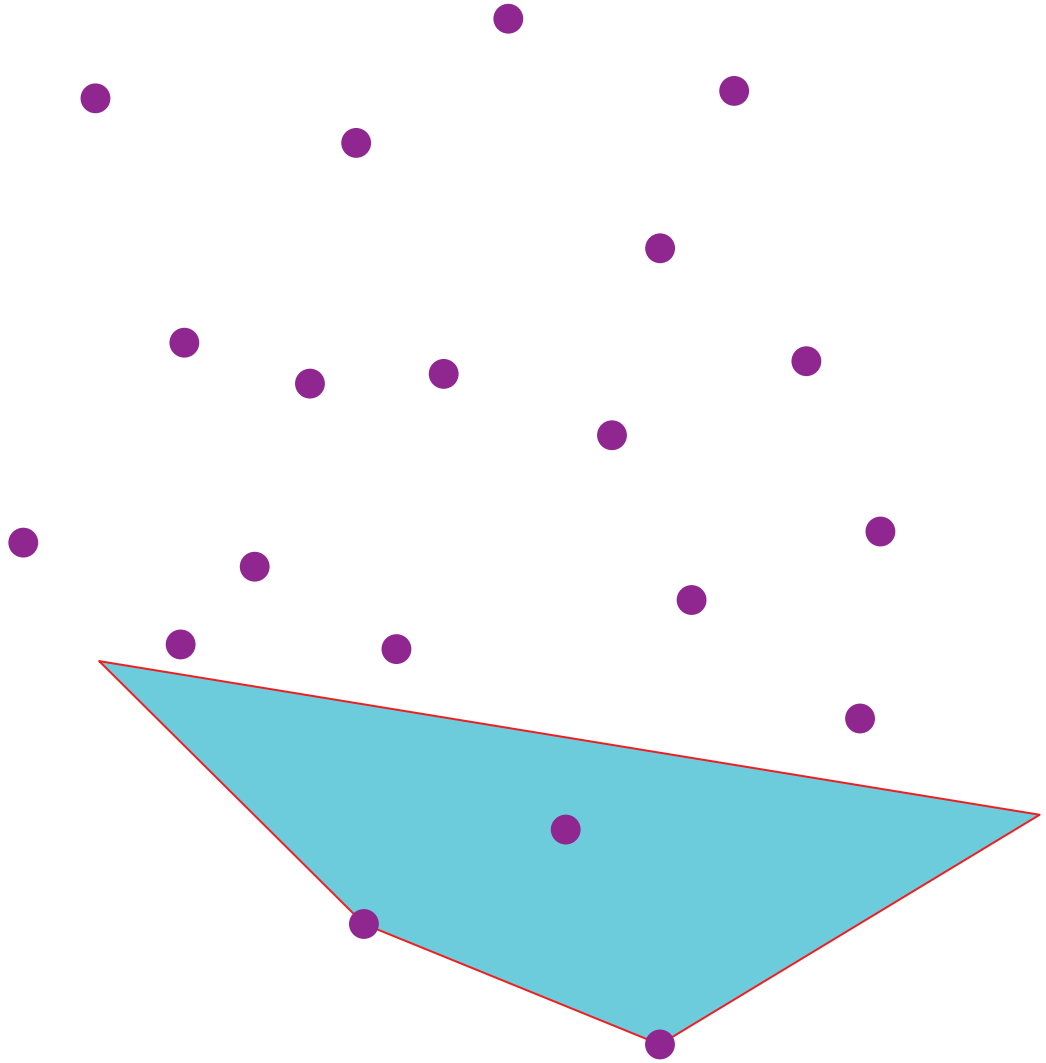
3D: Gift wrapping



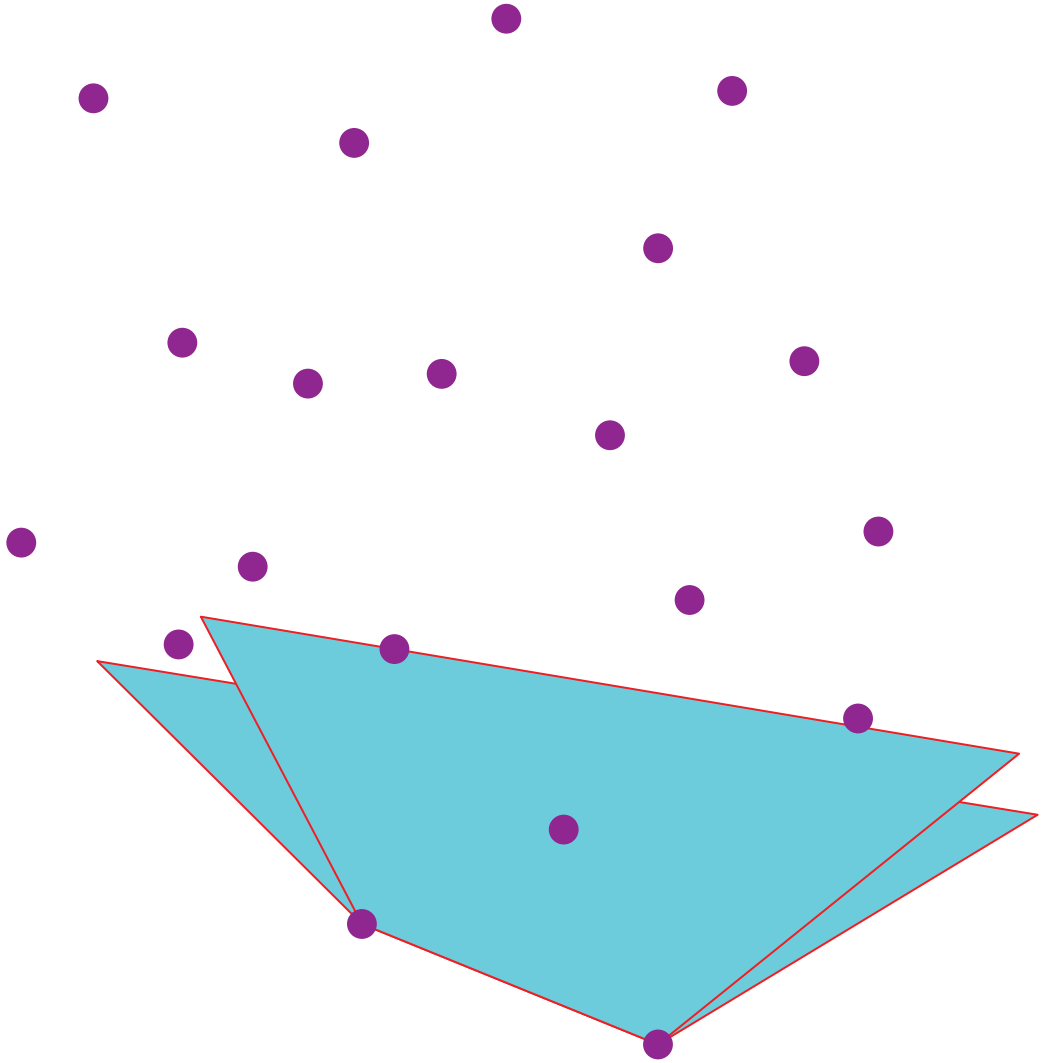
3D: Gift wrapping



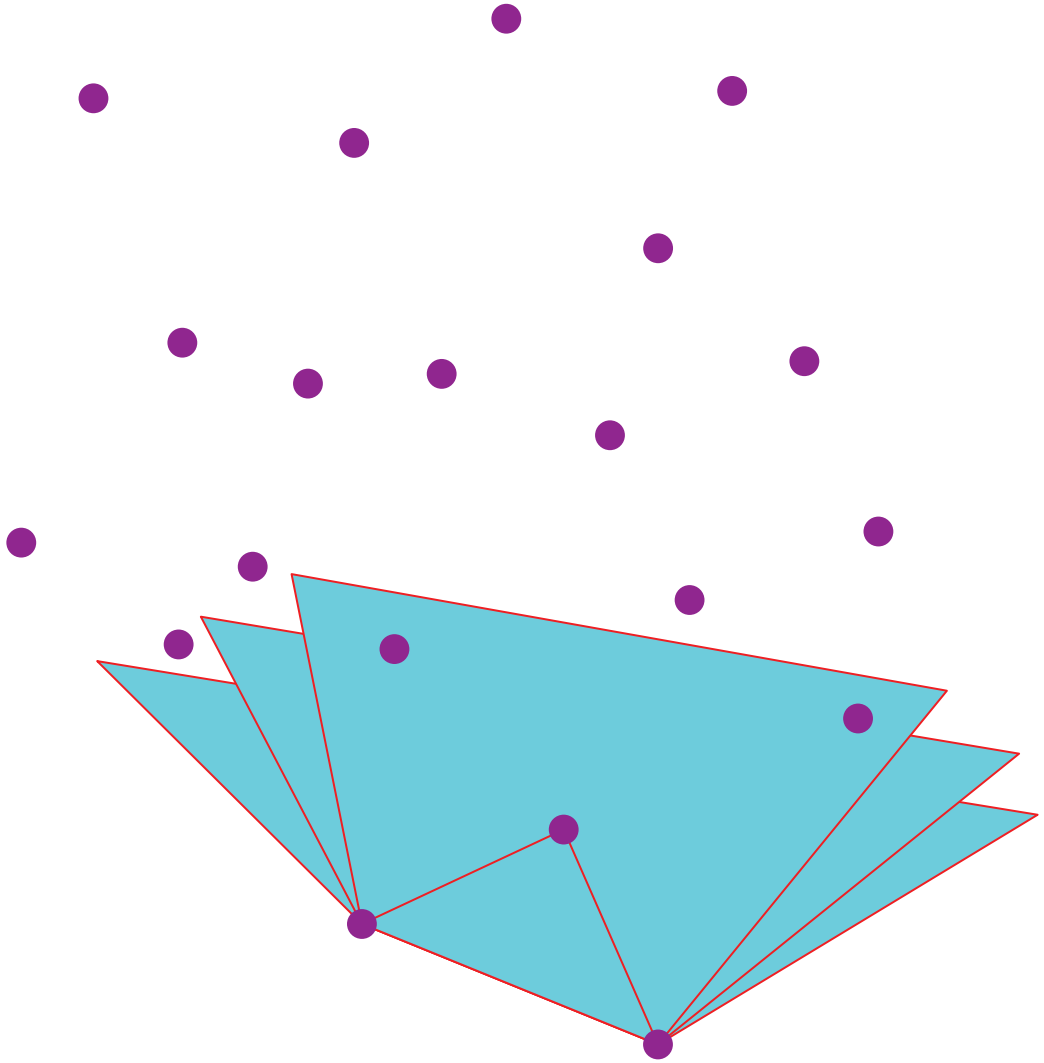
3D: Gift wrapping



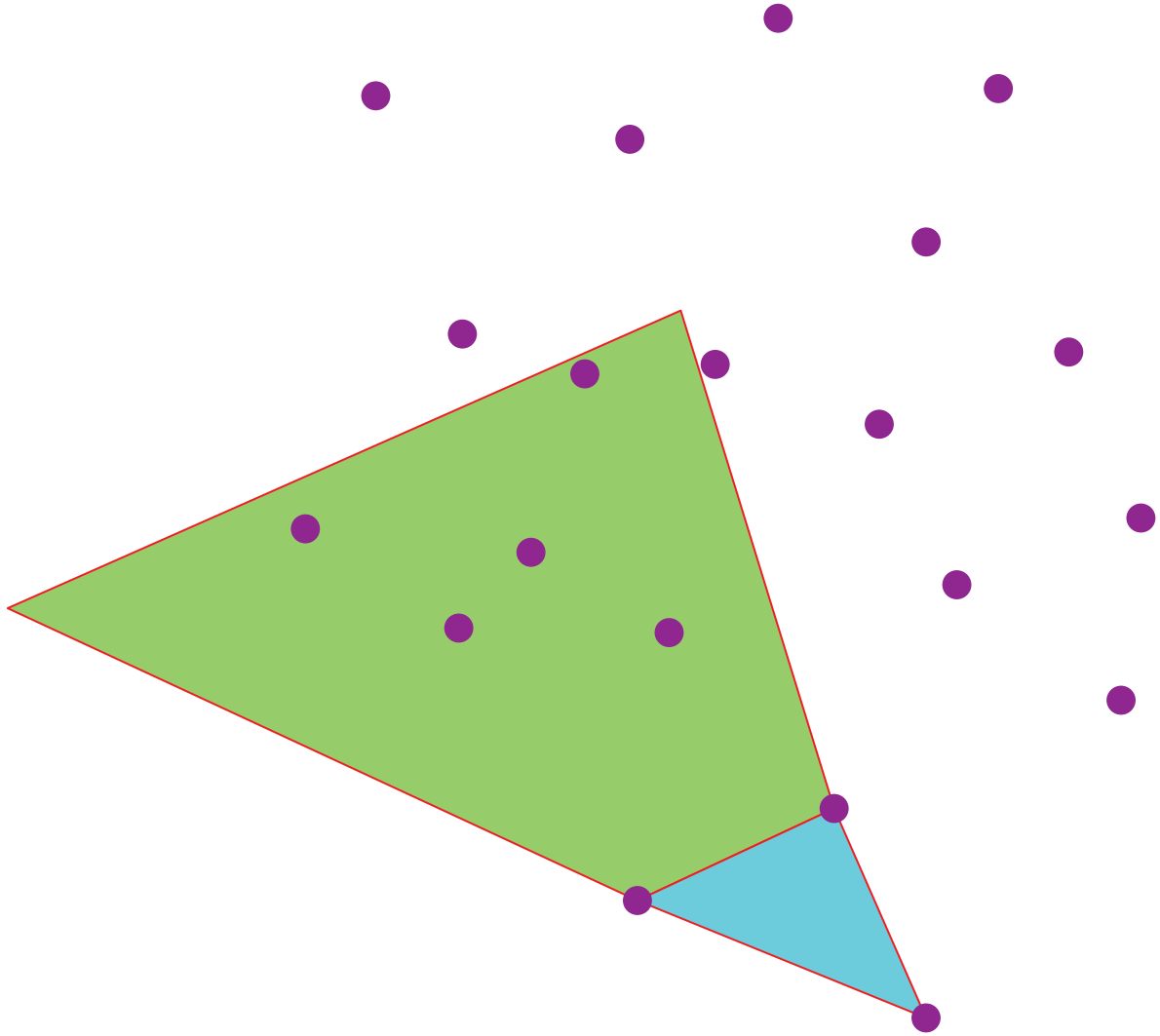
3D: Gift wrapping



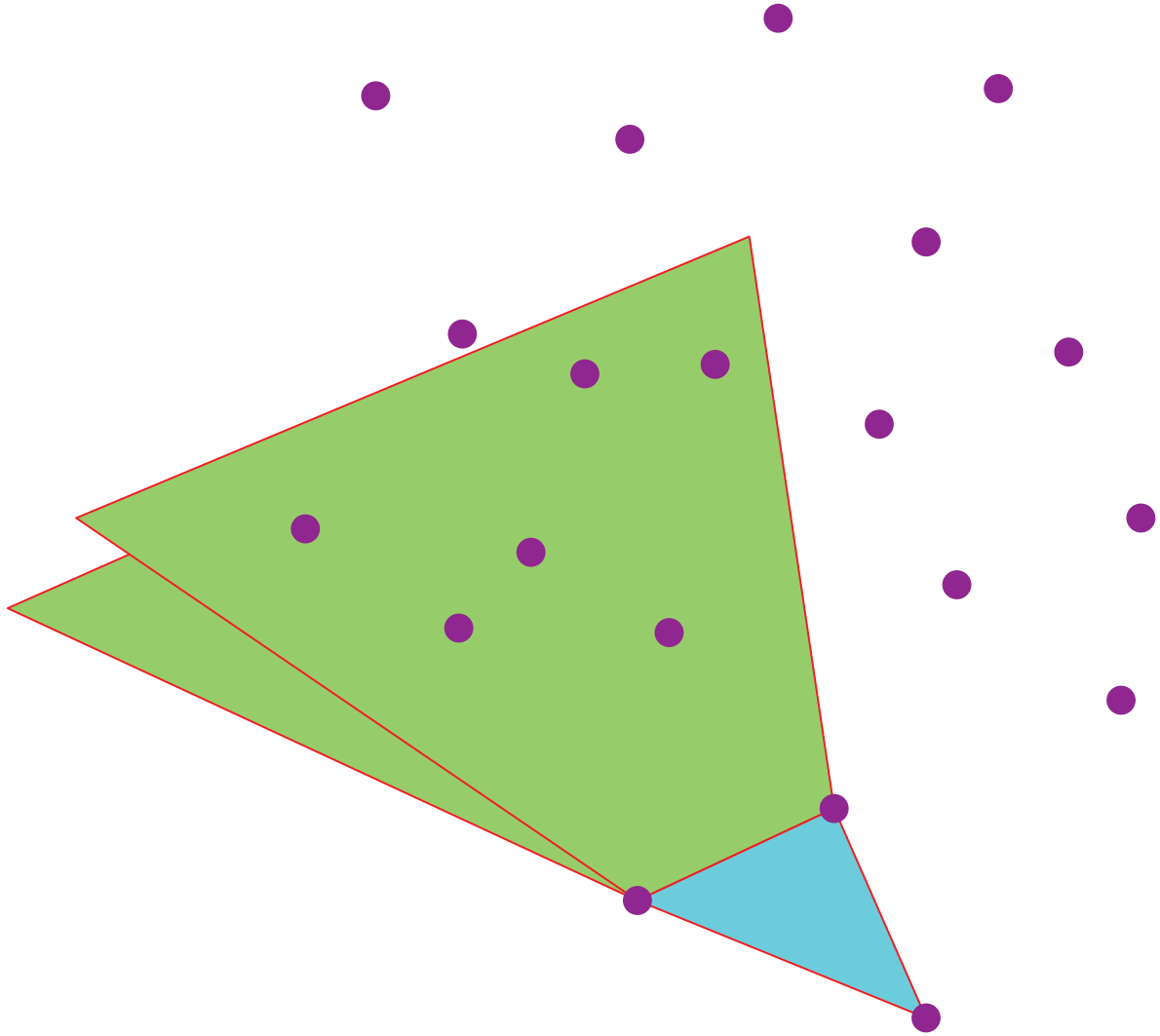
3D: Gift wrapping



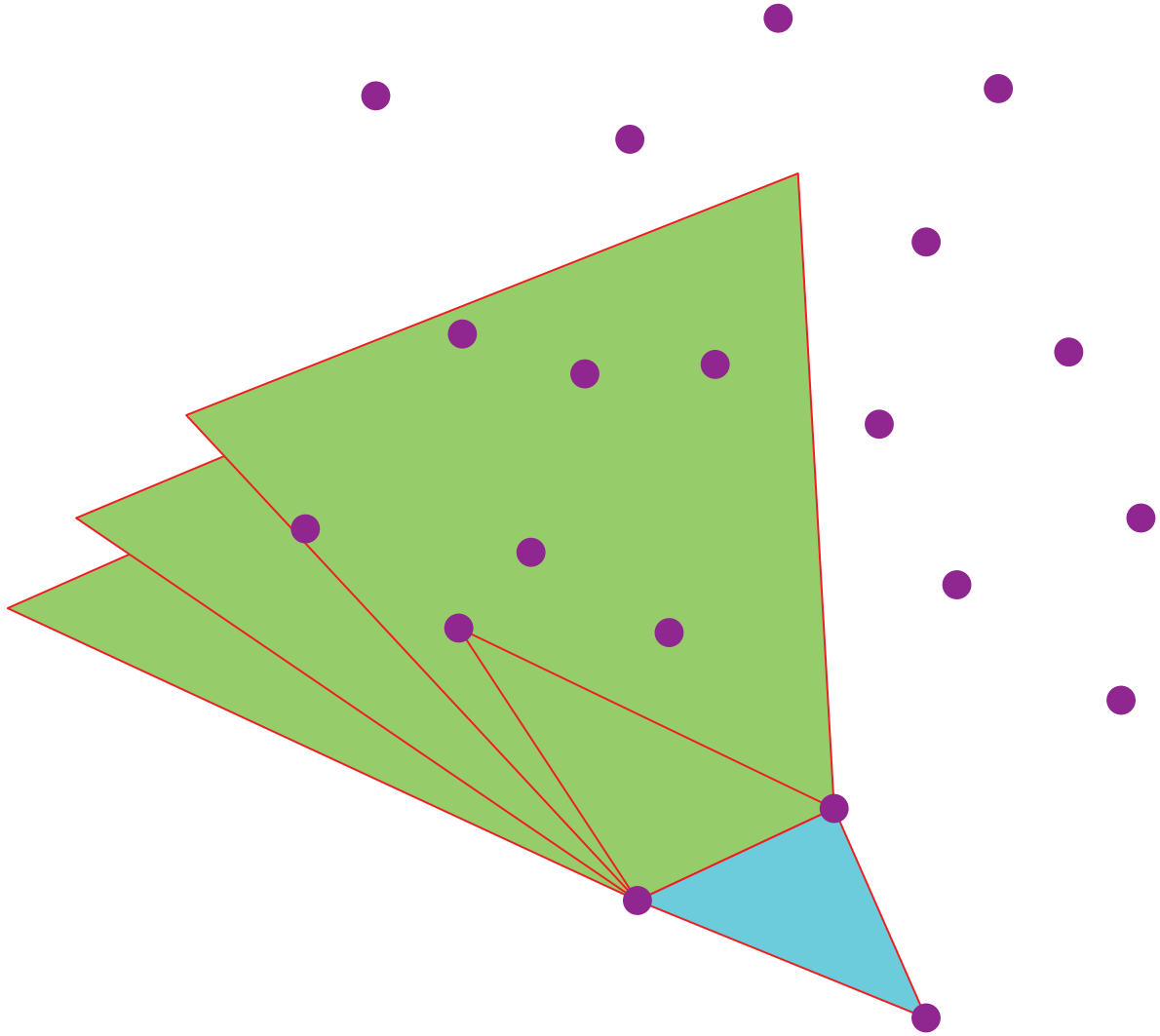
3D: Gift wrapping



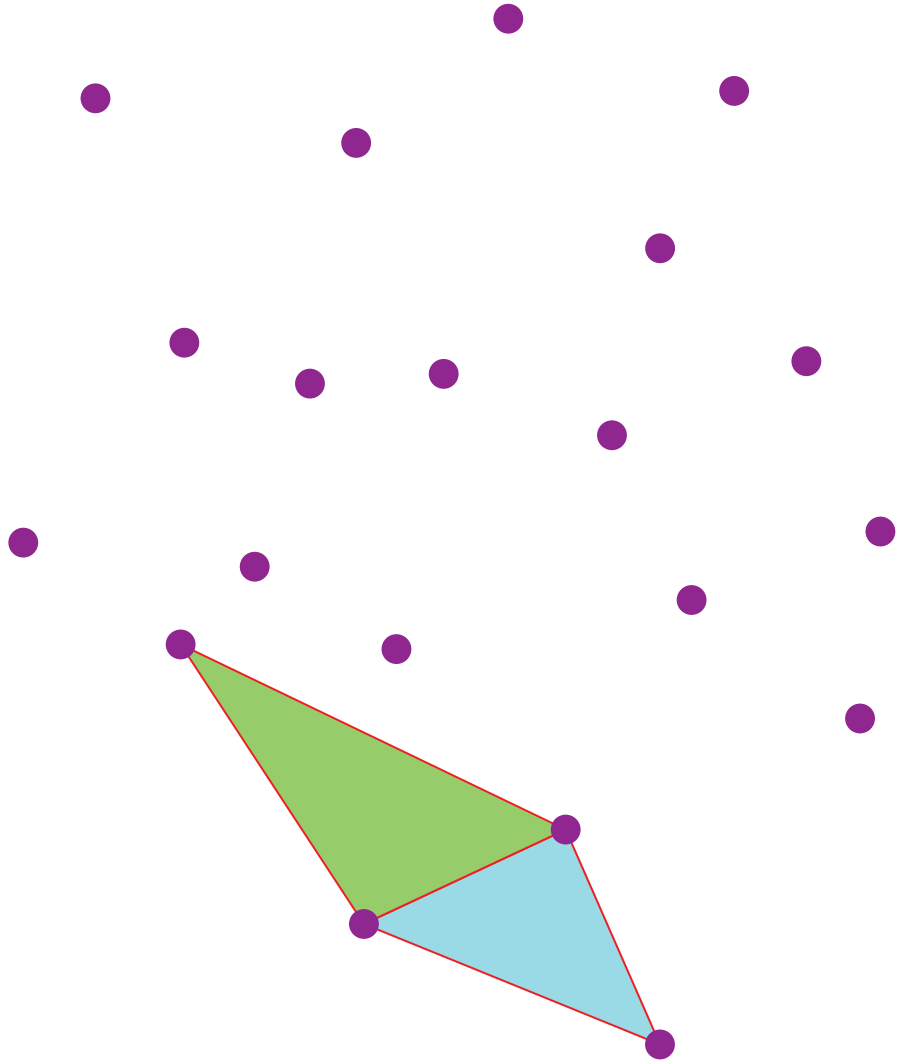
3D: Gift wrapping



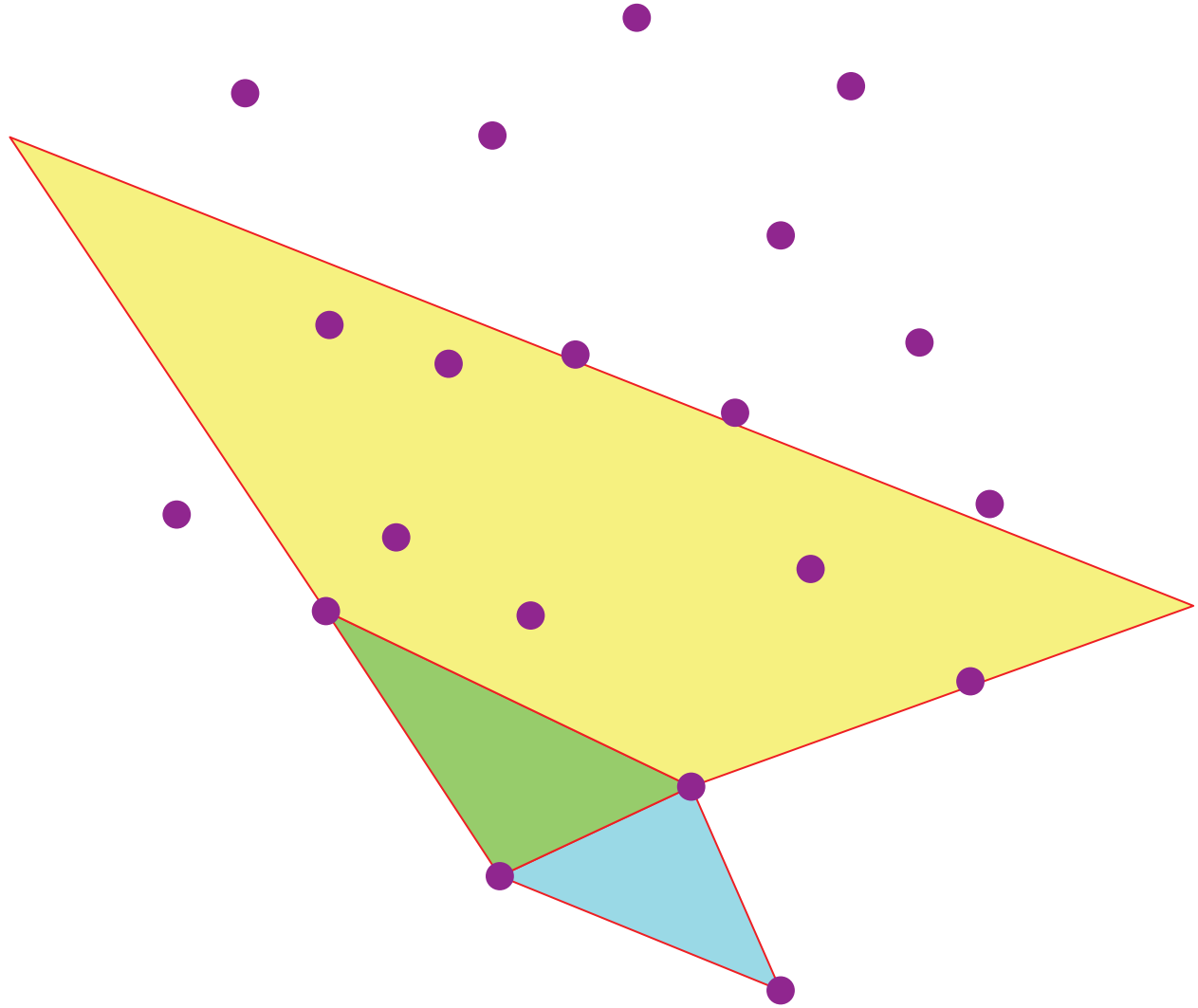
3D: Gift wrapping



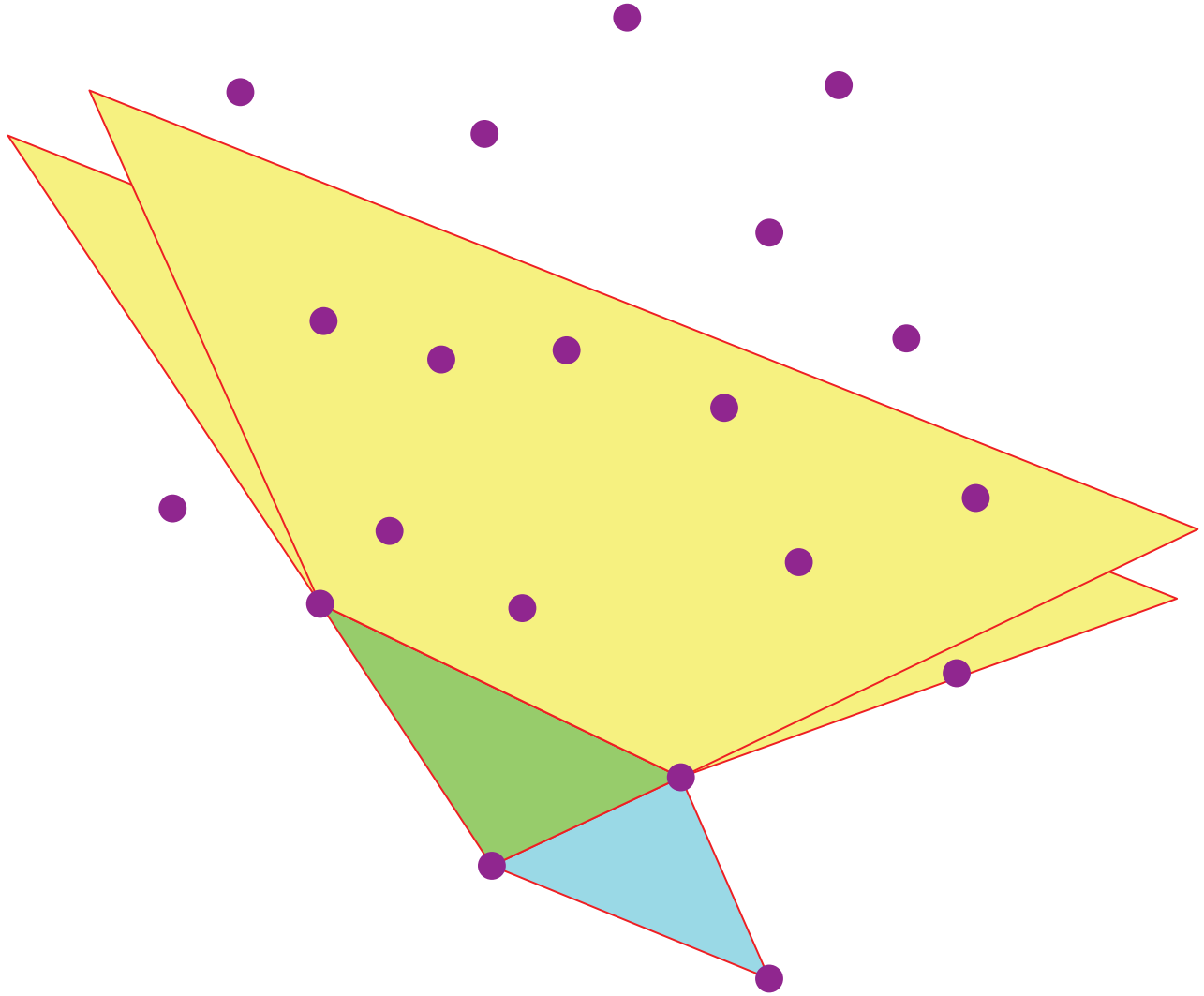
3D: Gift wrapping



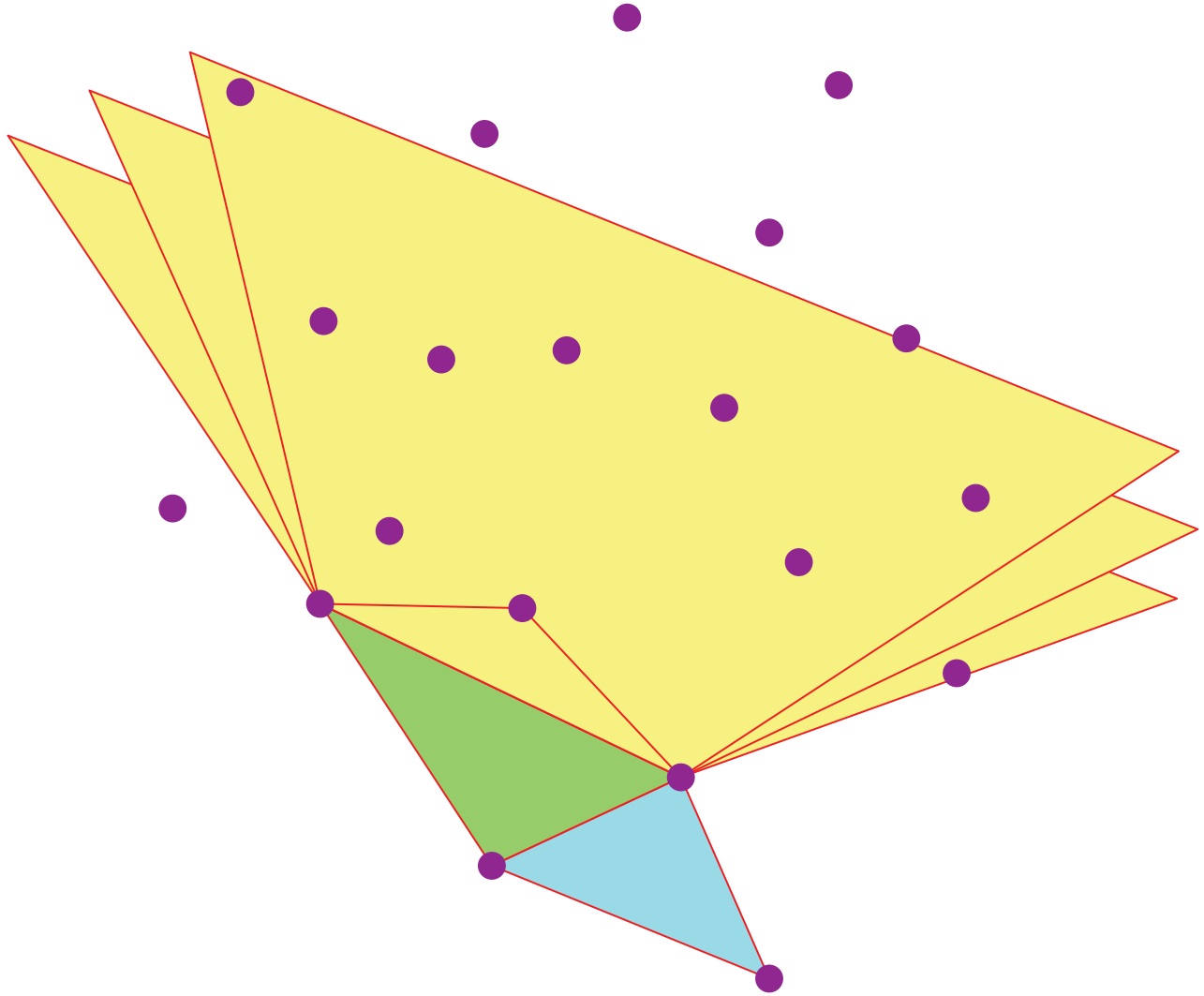
3D: Gift wrapping



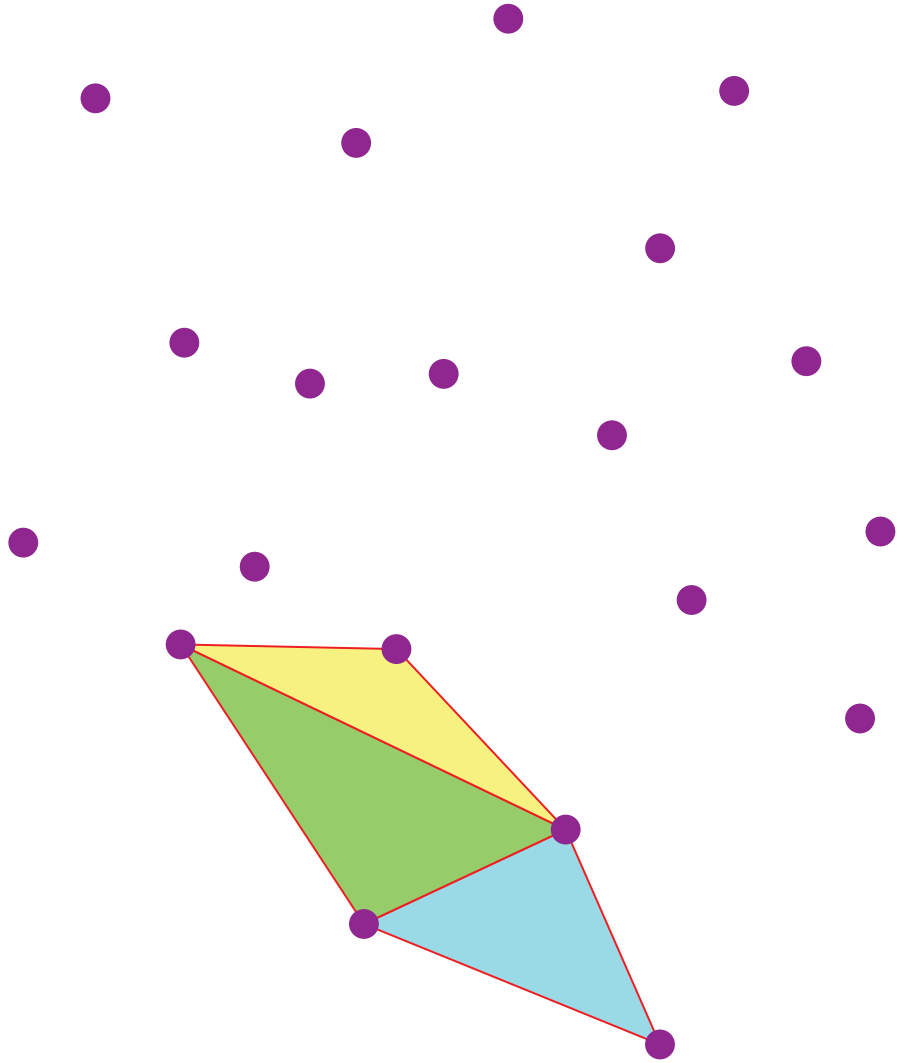
3D: Gift wrapping



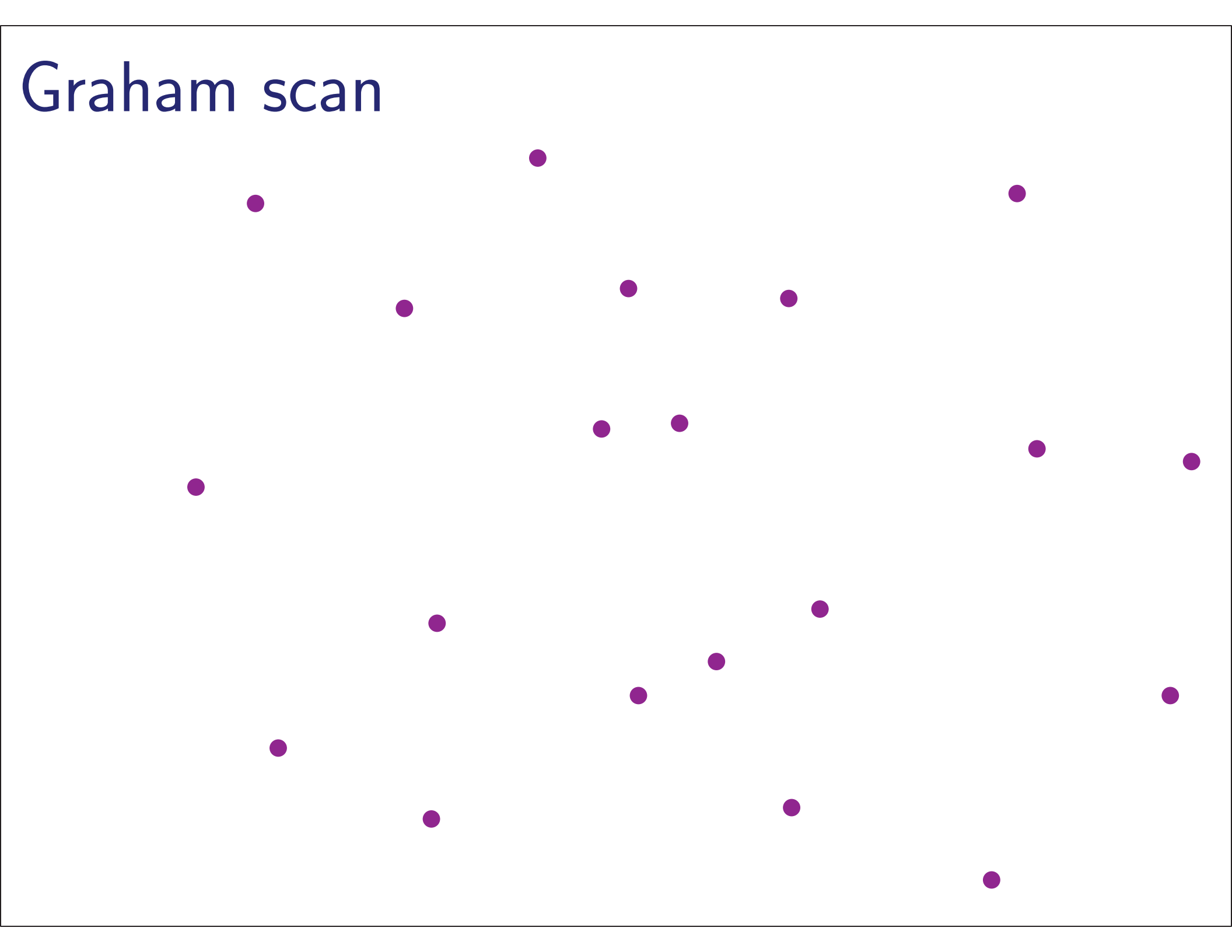
3D: Gift wrapping



3D: Gift wrapping

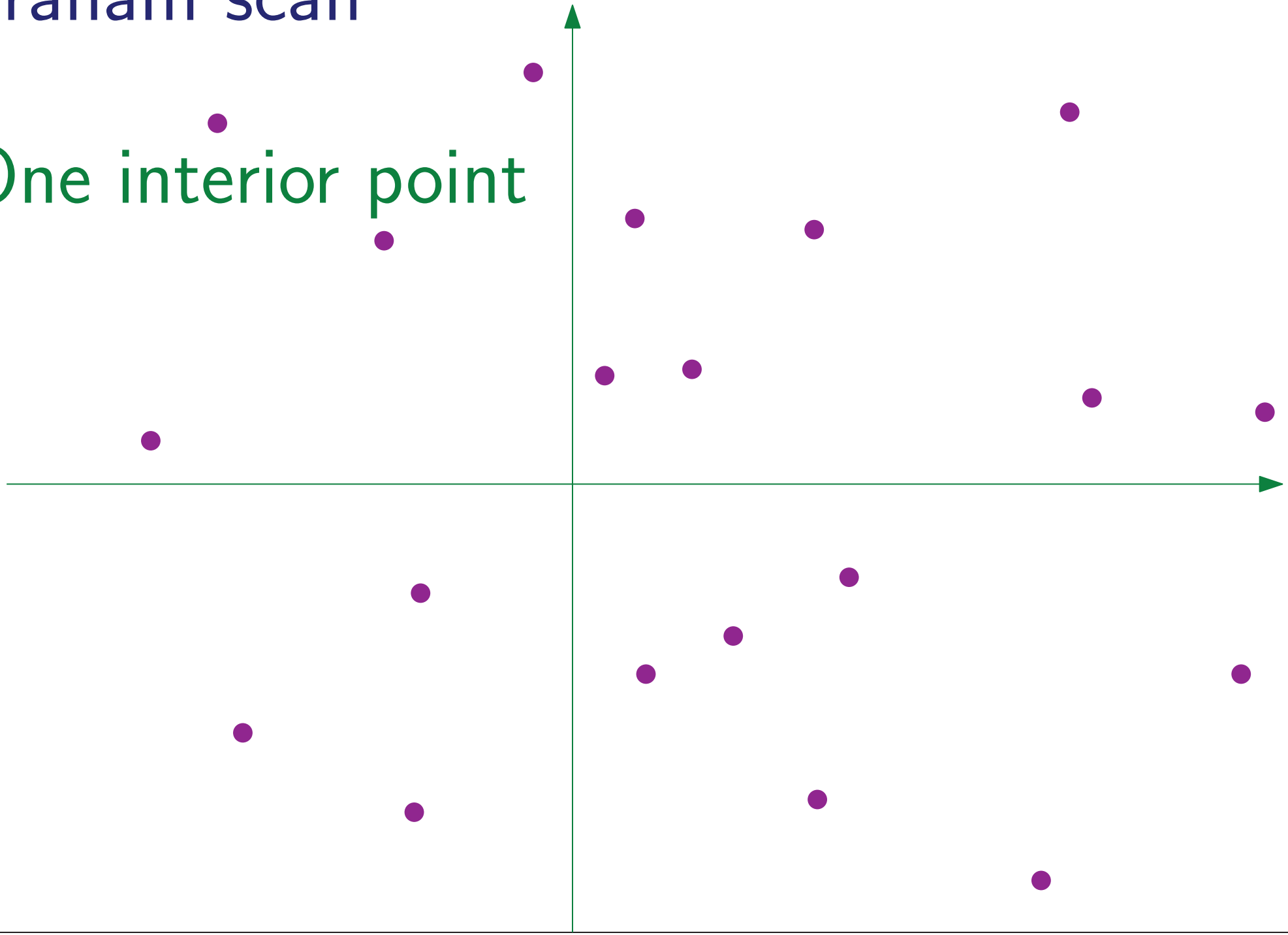


Graham scan



Graham scan

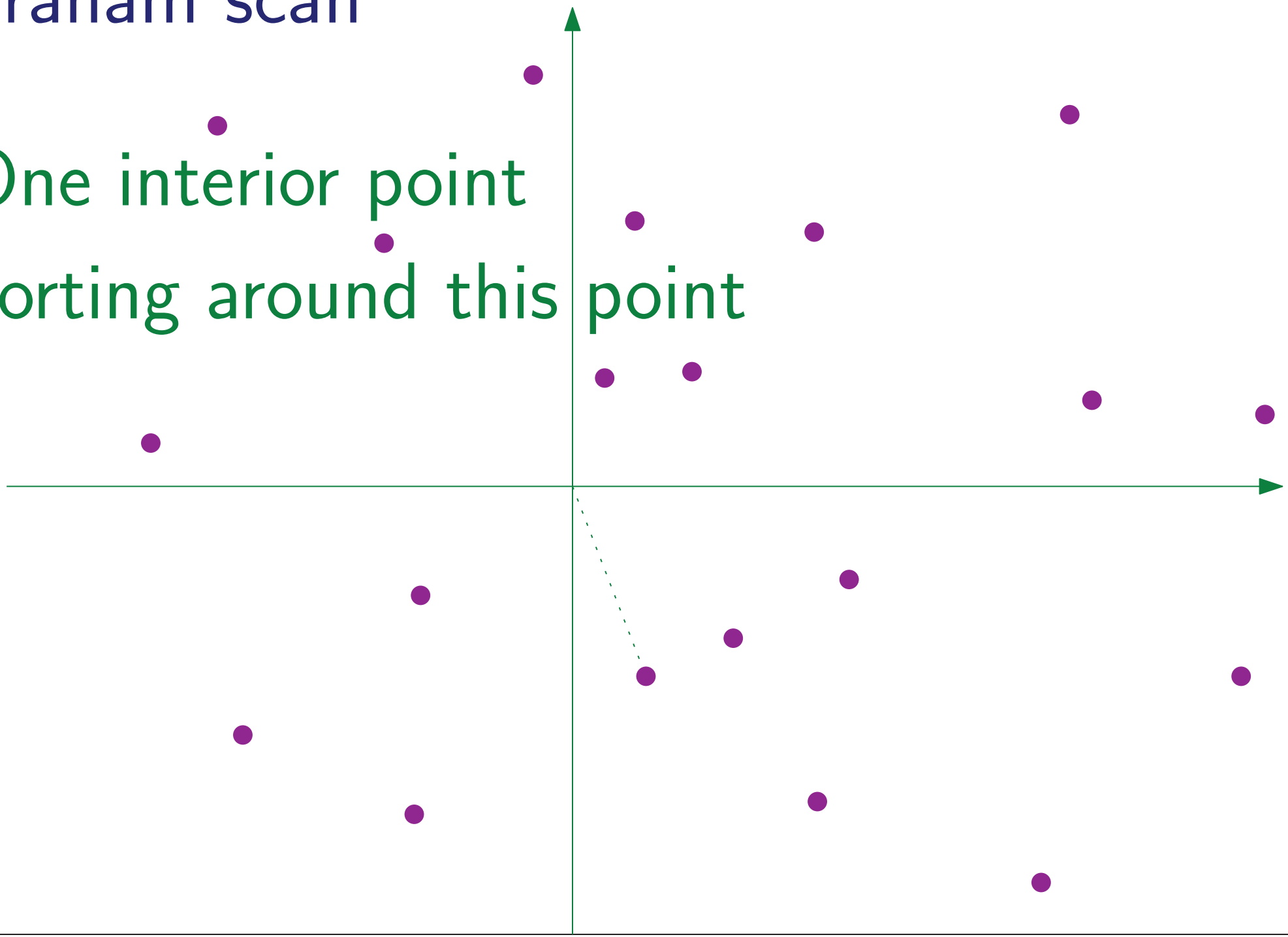
One interior point



Graham scan

One interior point

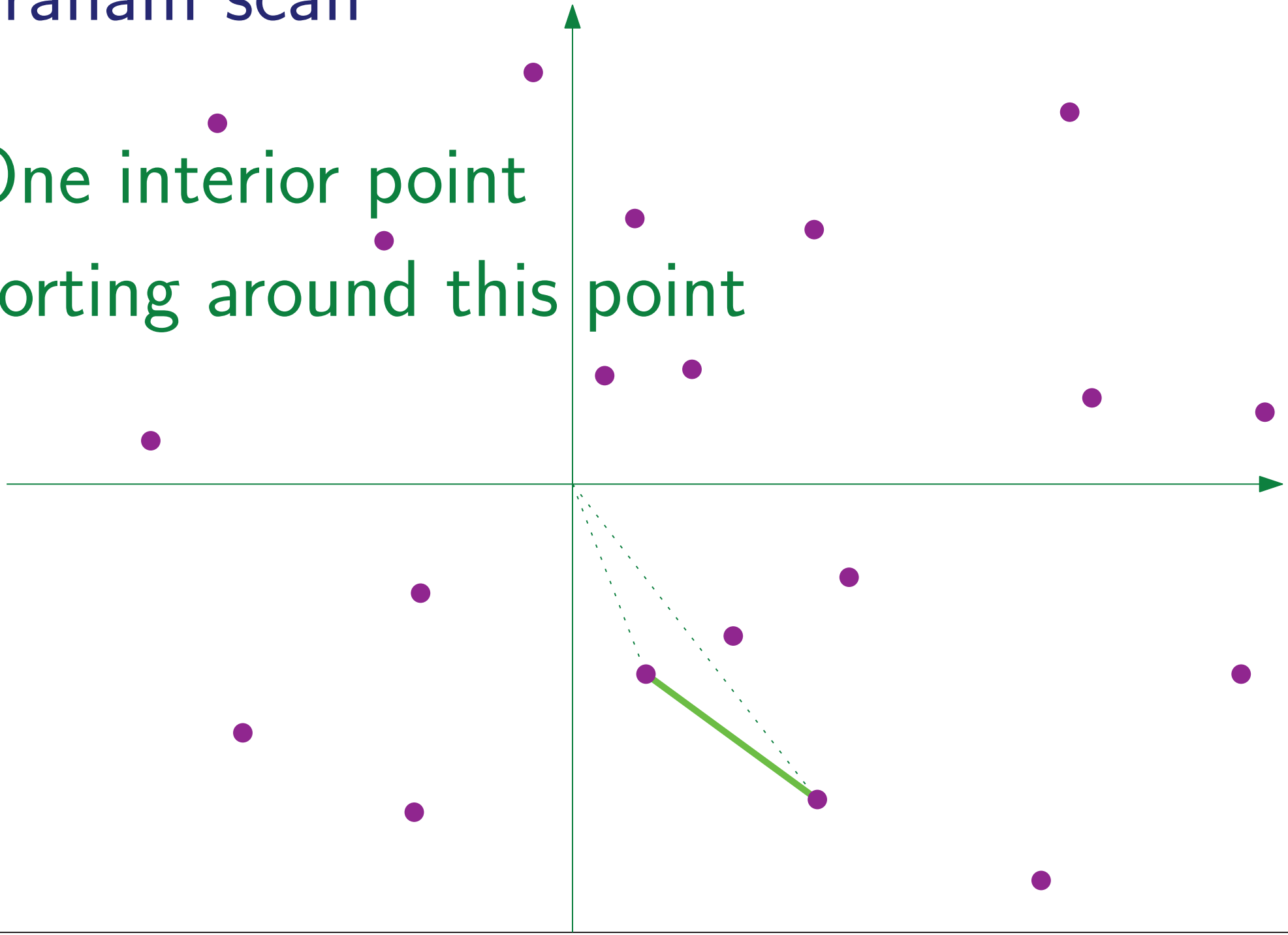
Sorting around this point



Graham scan

One interior point

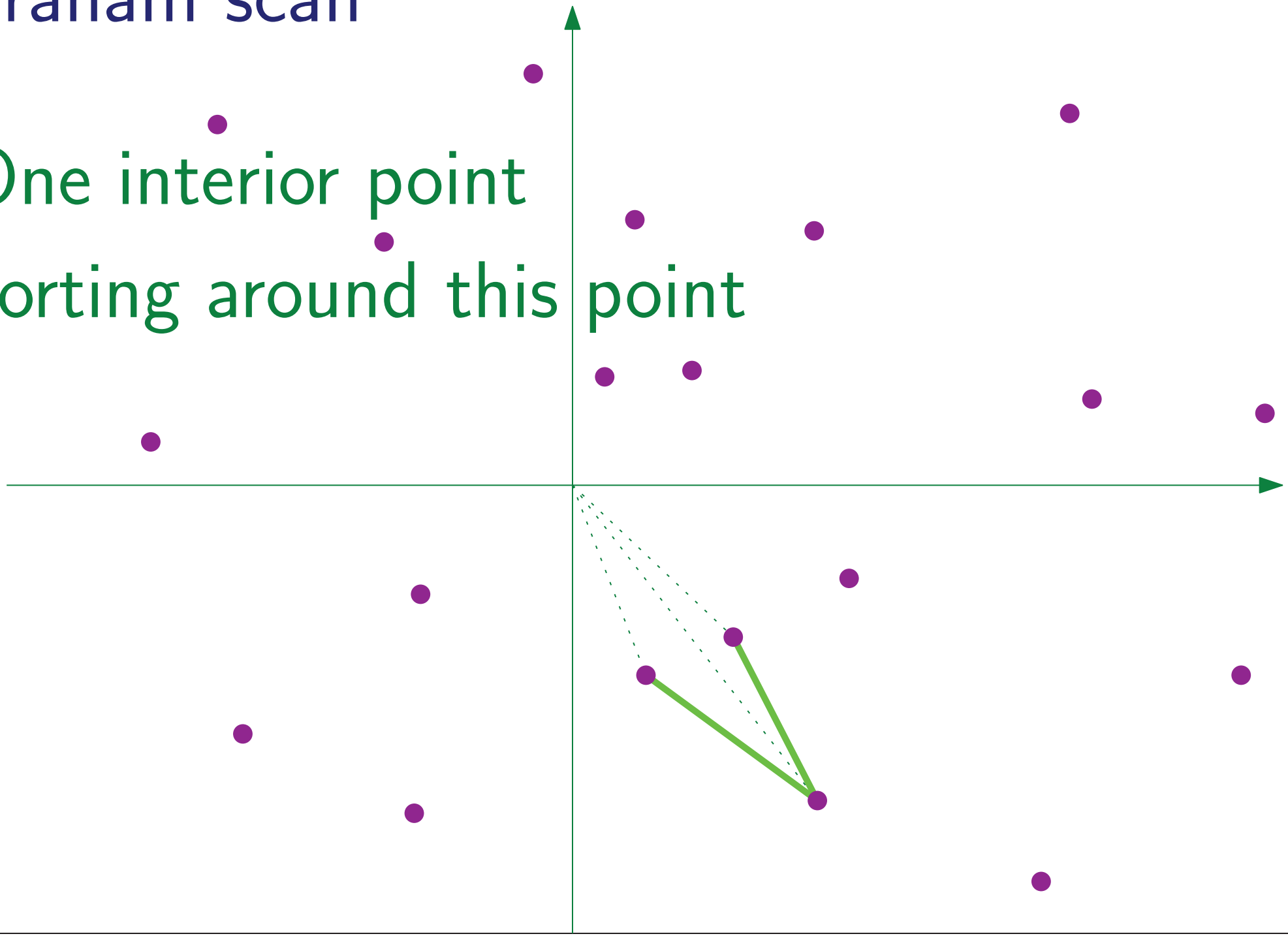
Sorting around this point



Graham scan

One interior point

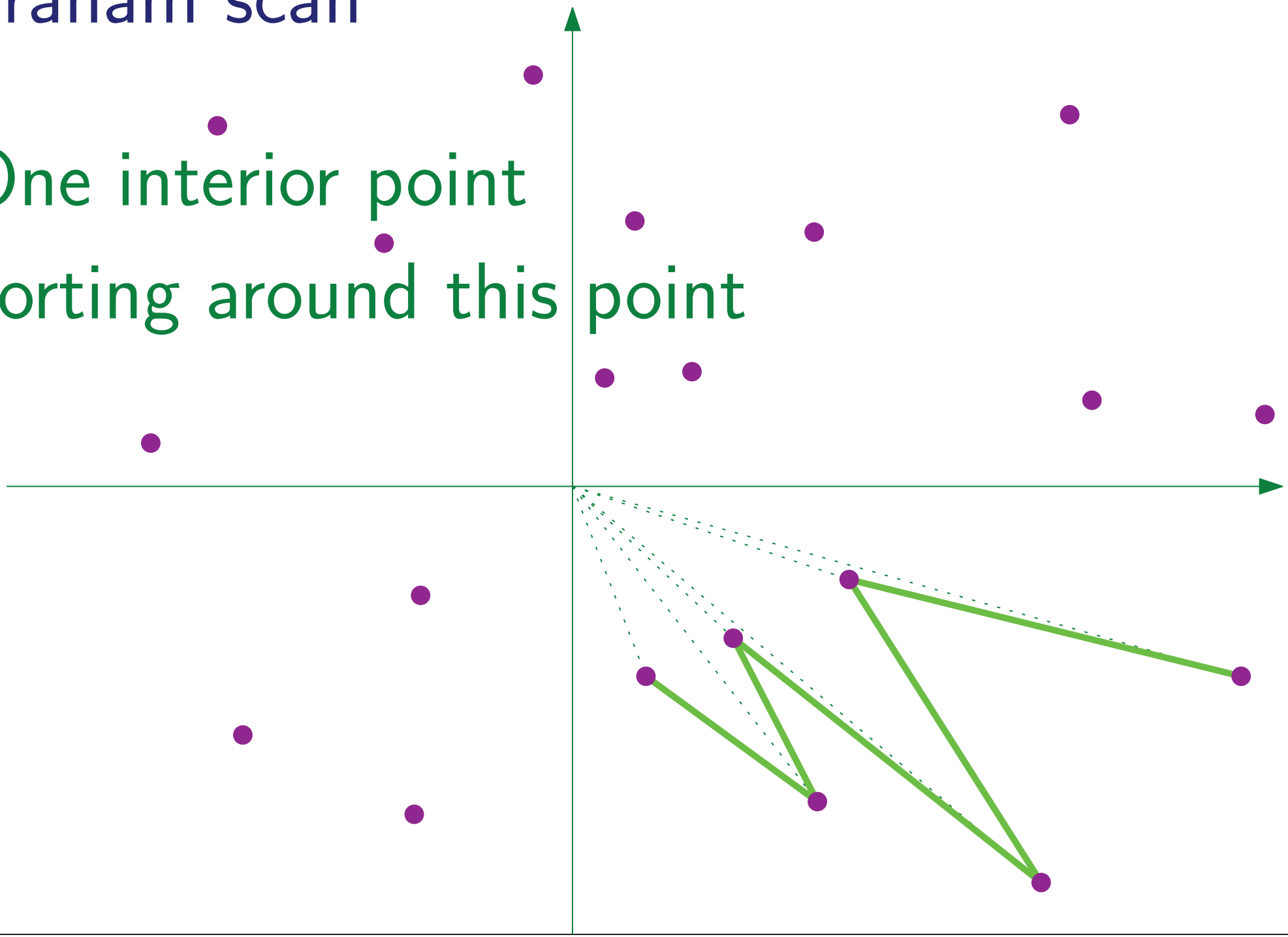
Sorting around this point



Graham scan

One interior point

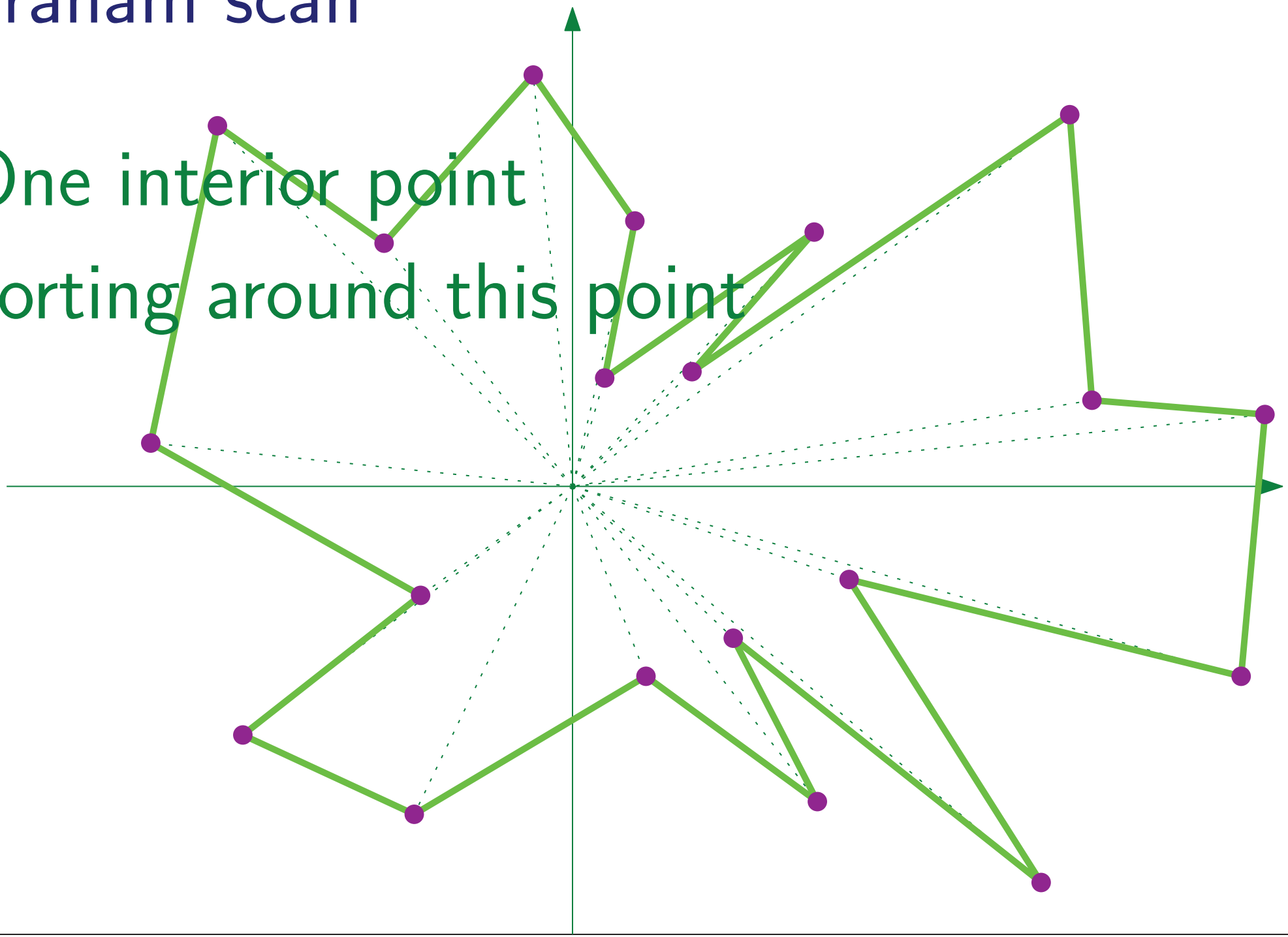
Sorting around this point



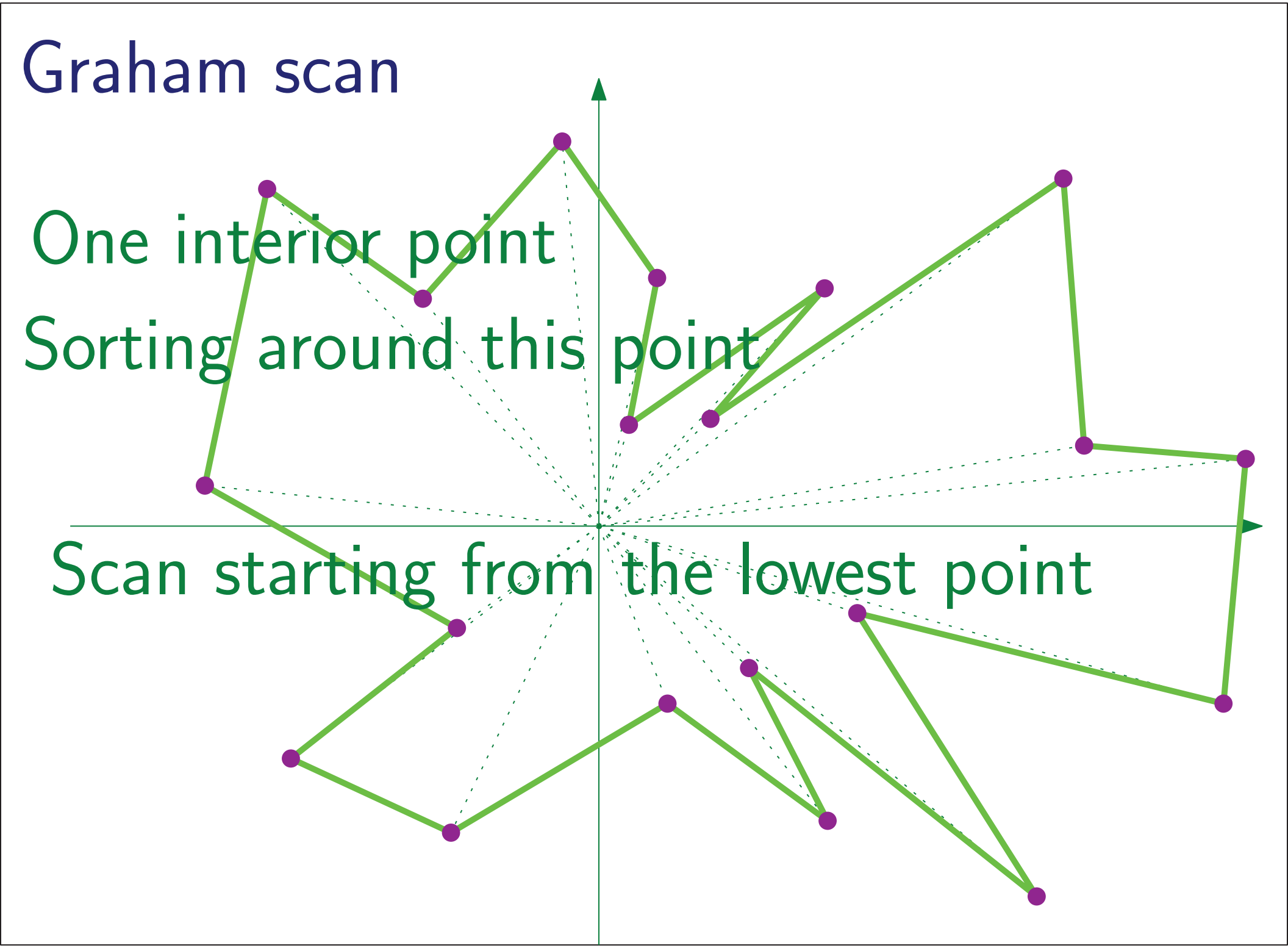
Graham scan

One interior point

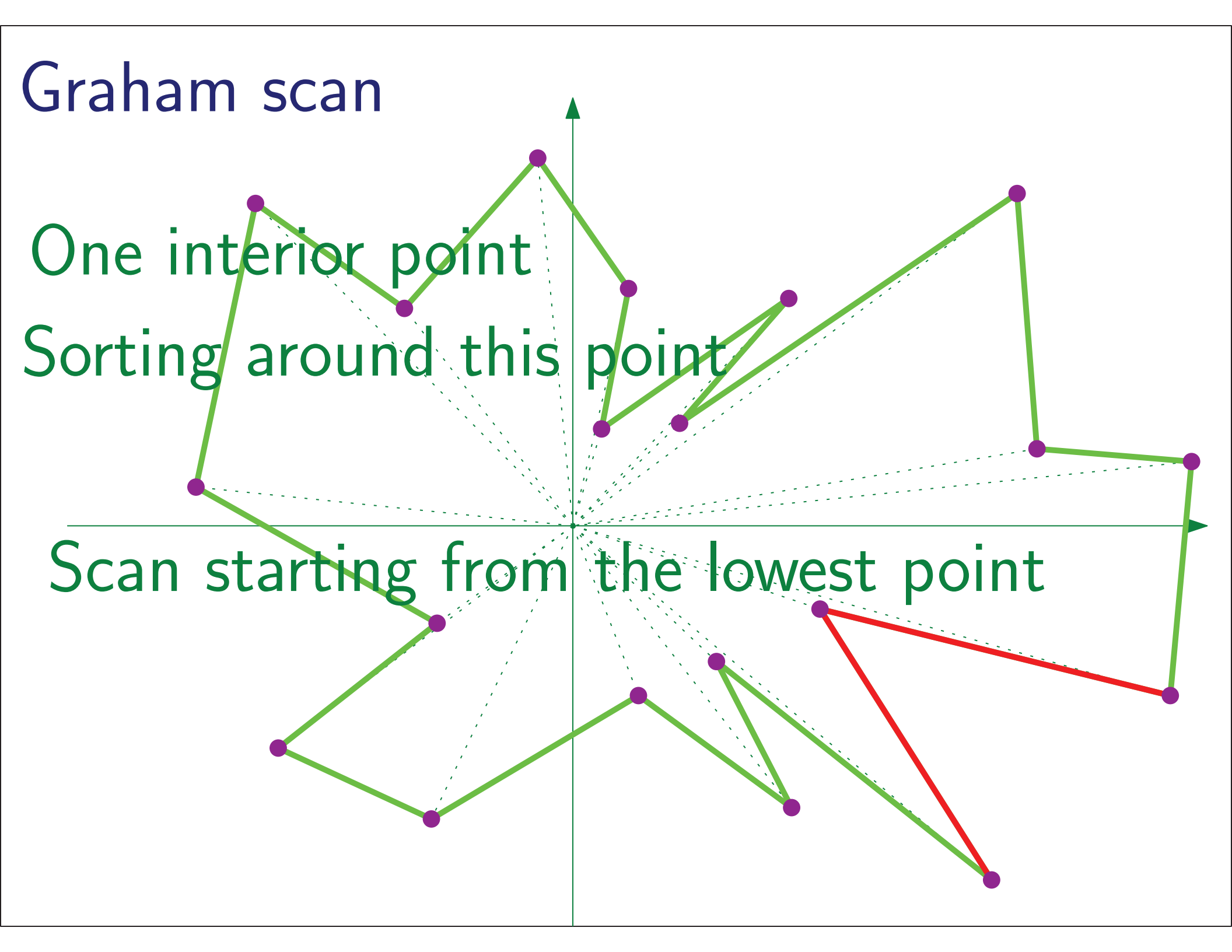
Sorting around this point



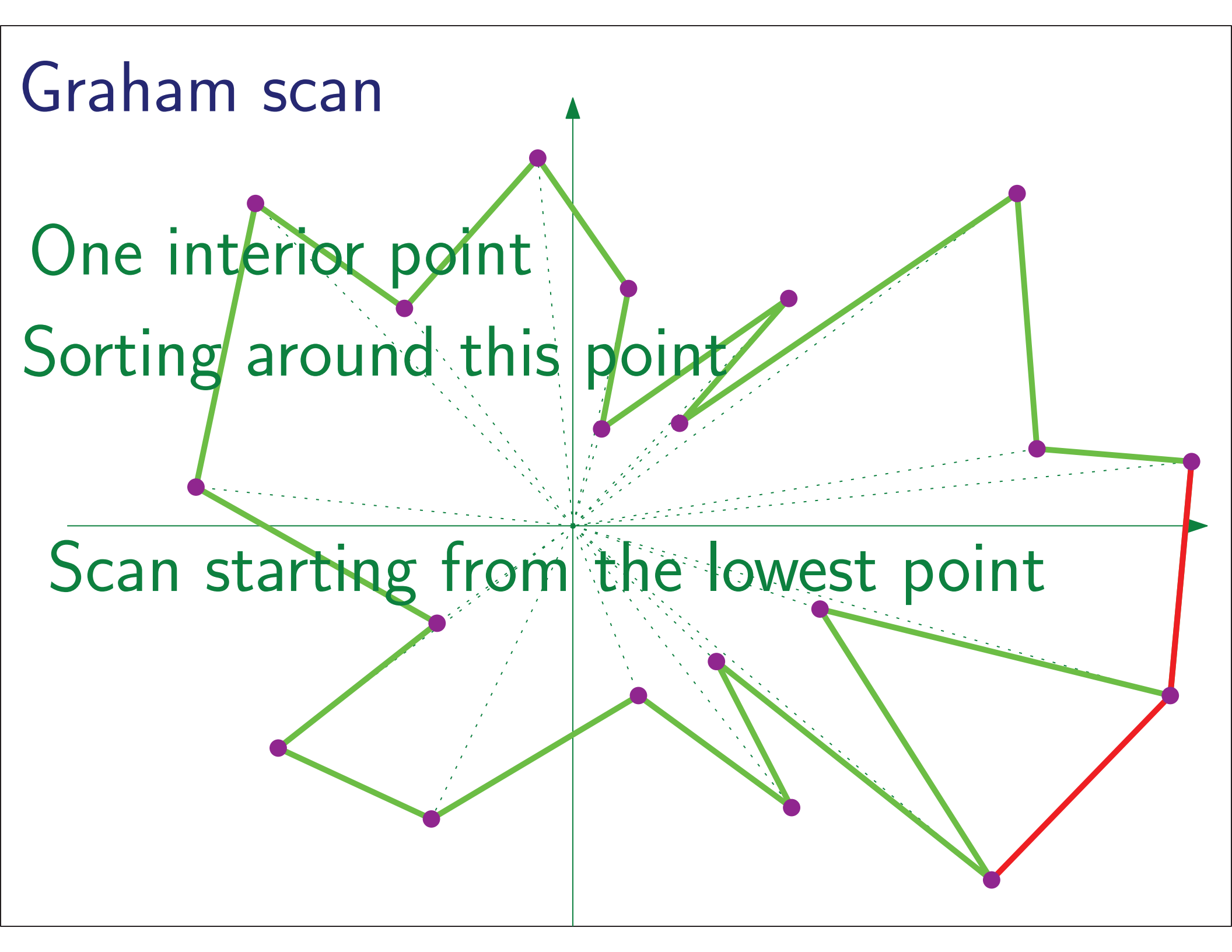
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

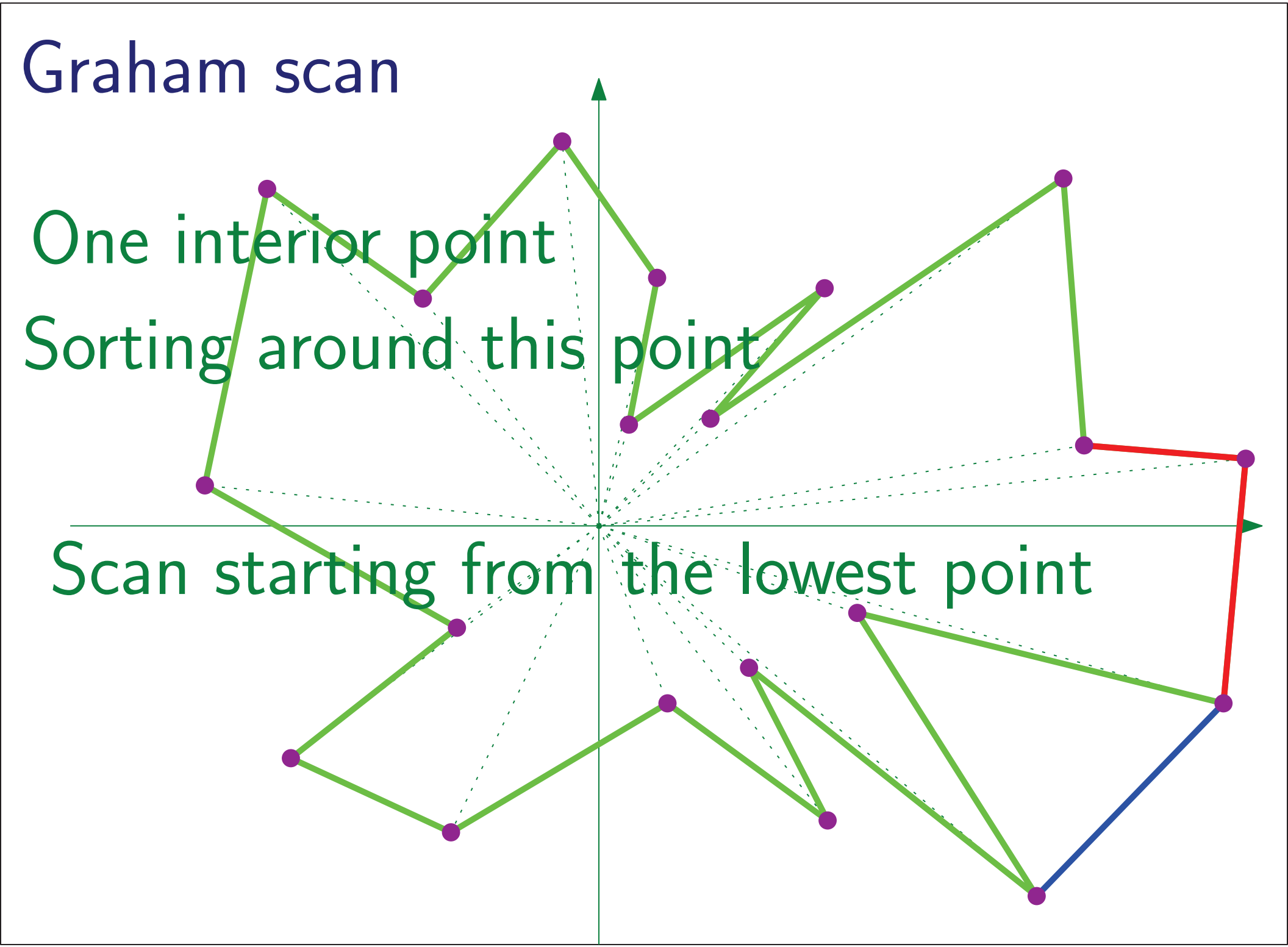
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

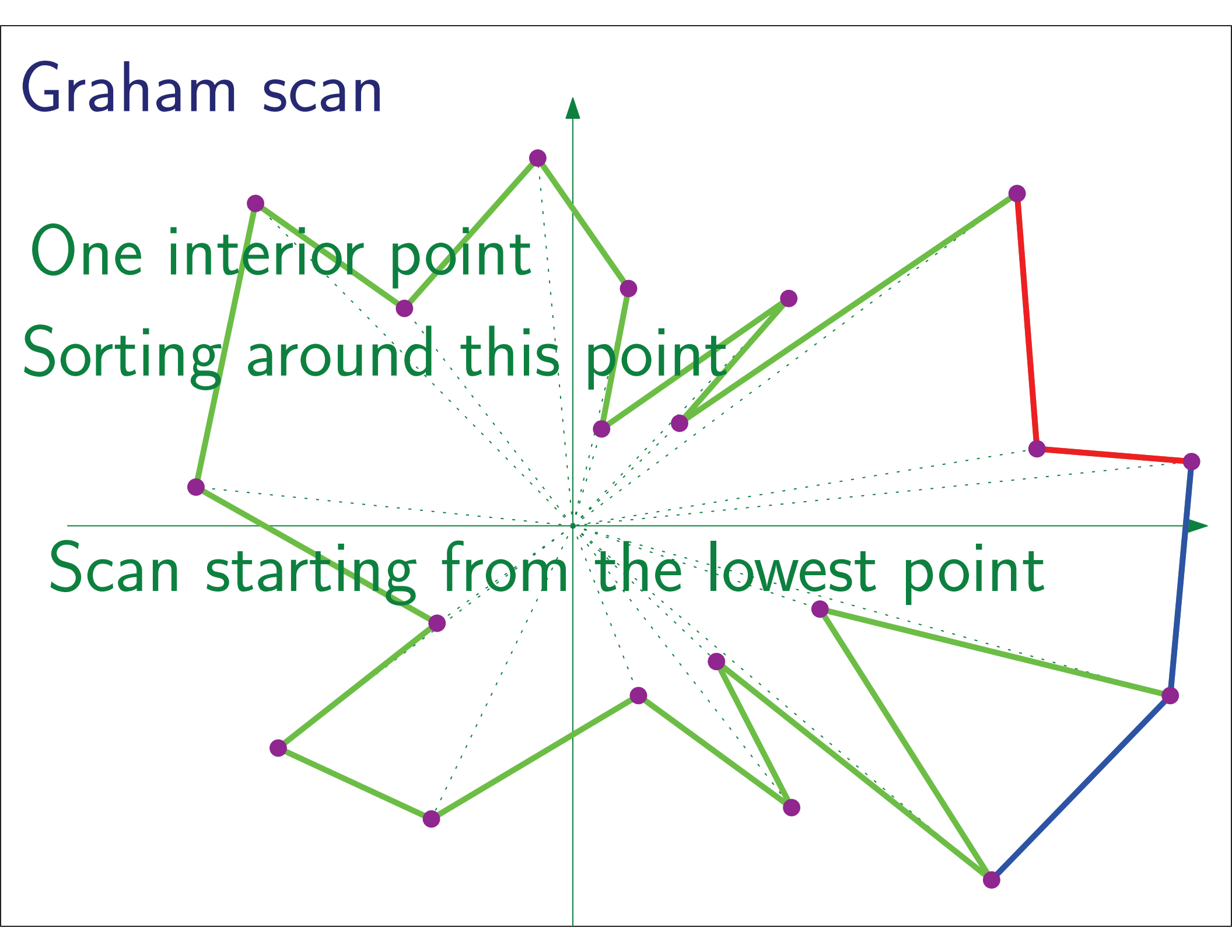
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

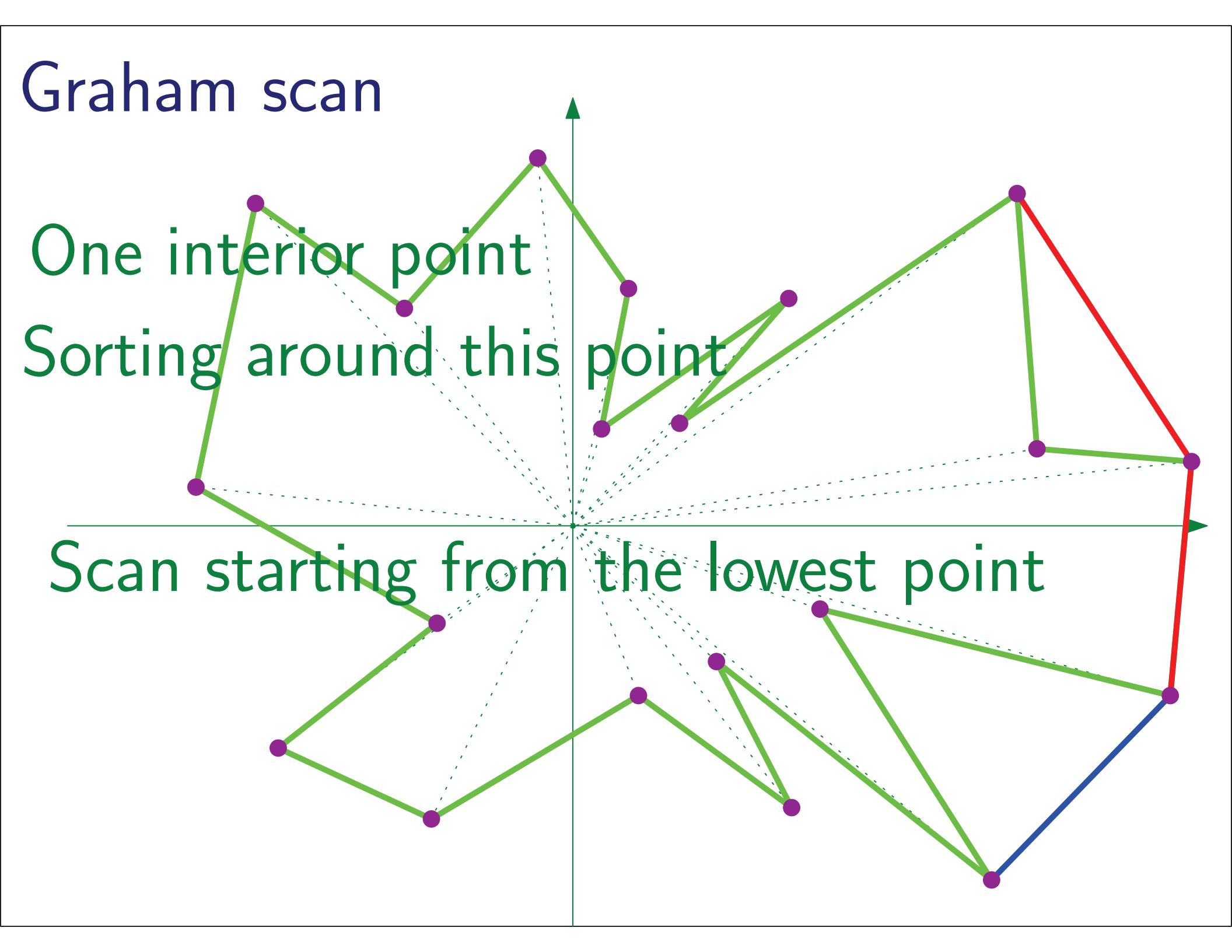
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

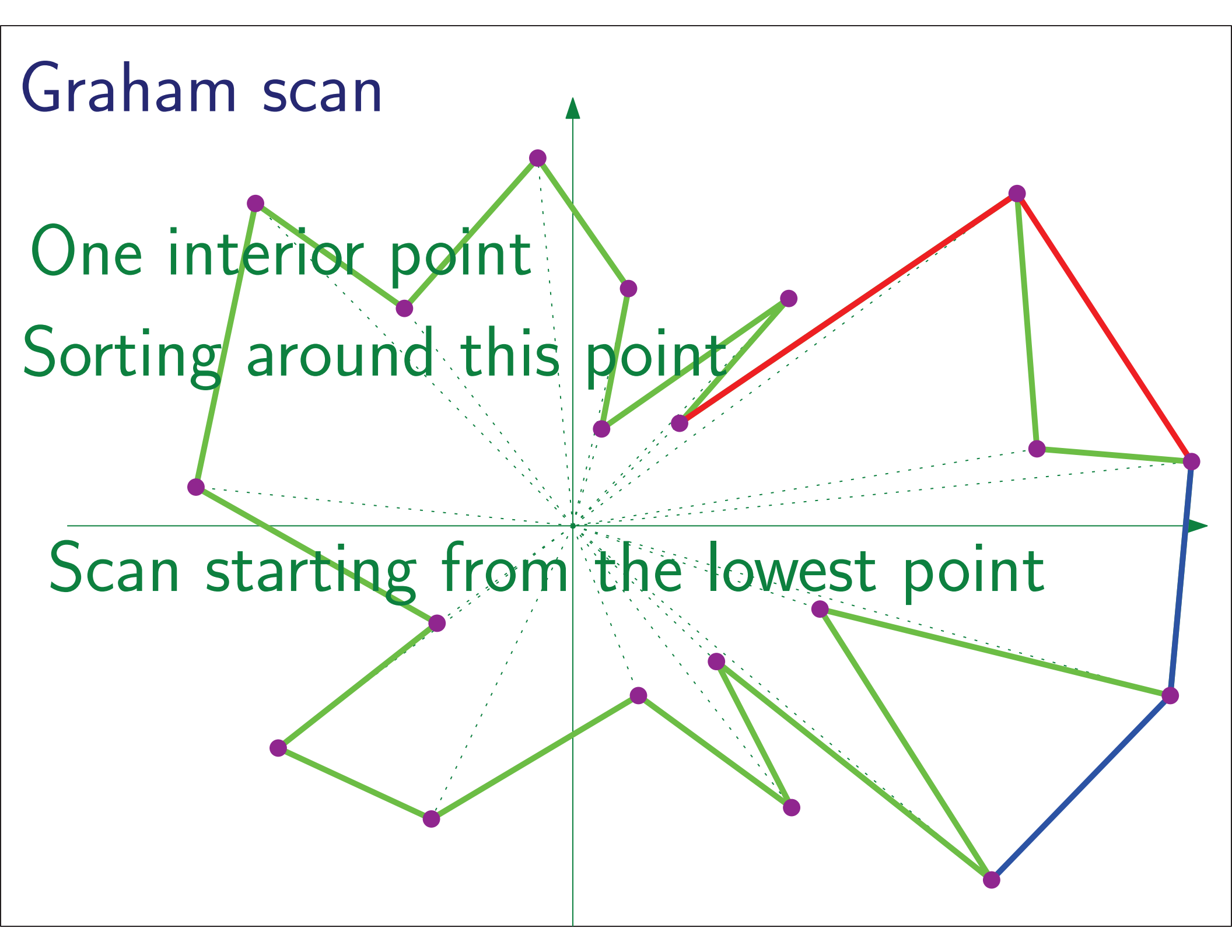
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

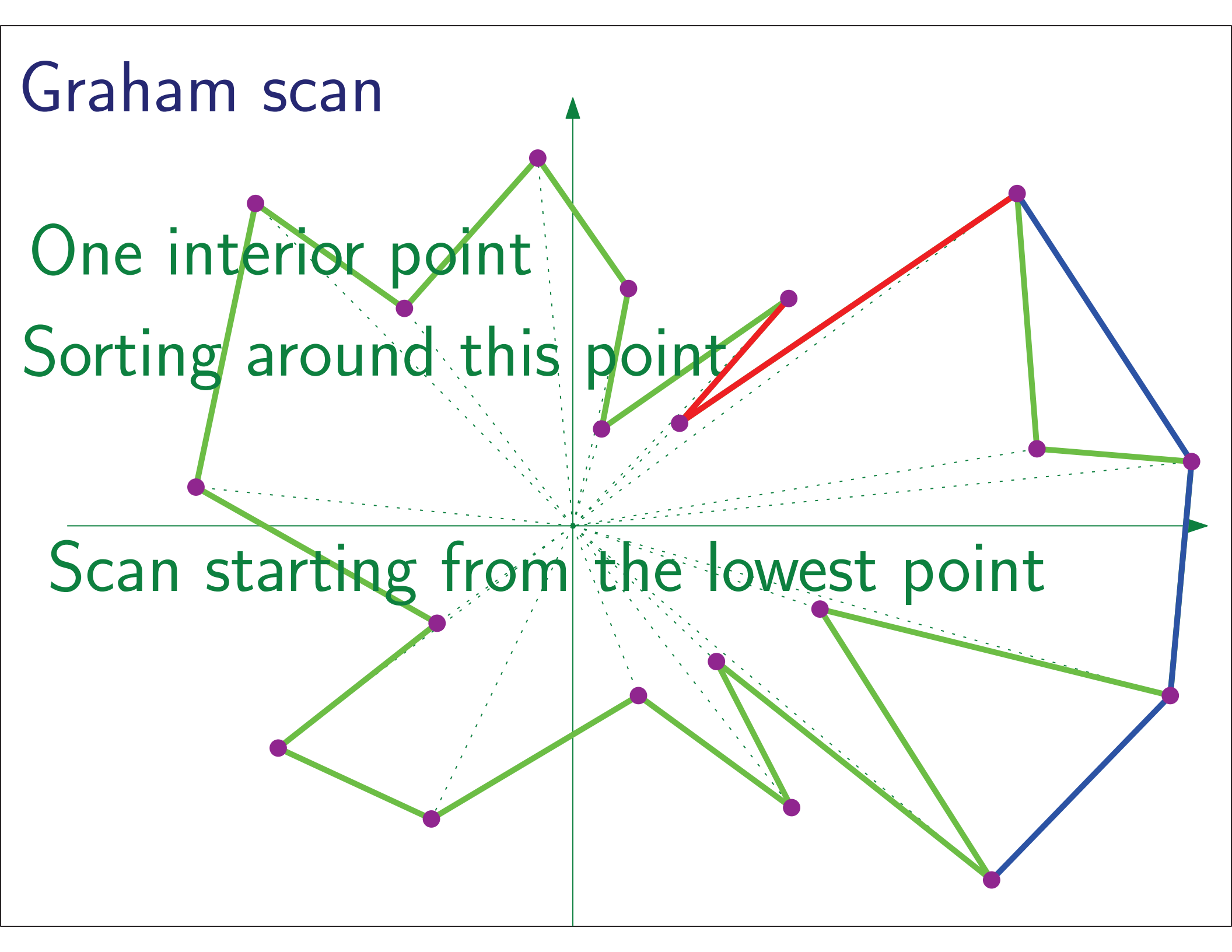
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

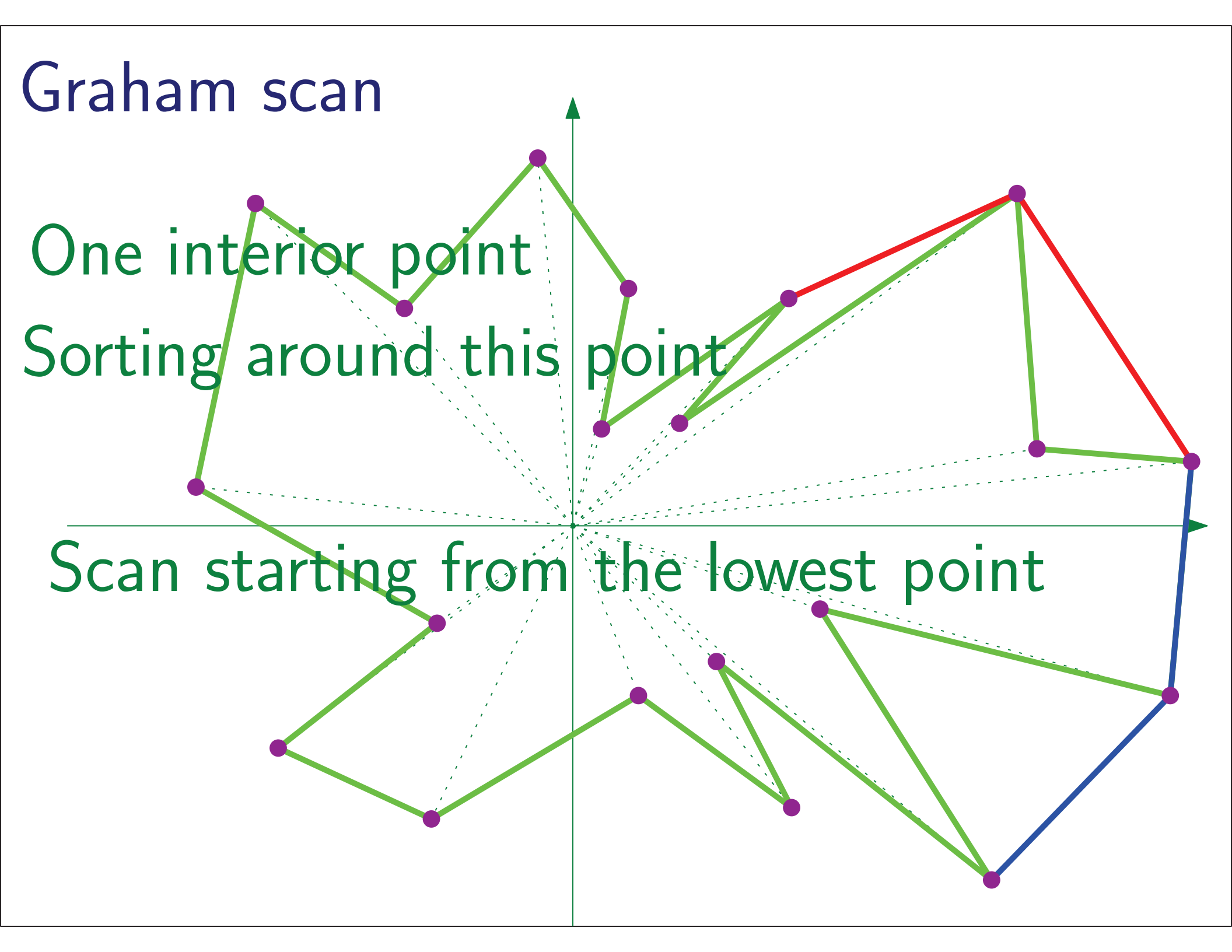
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

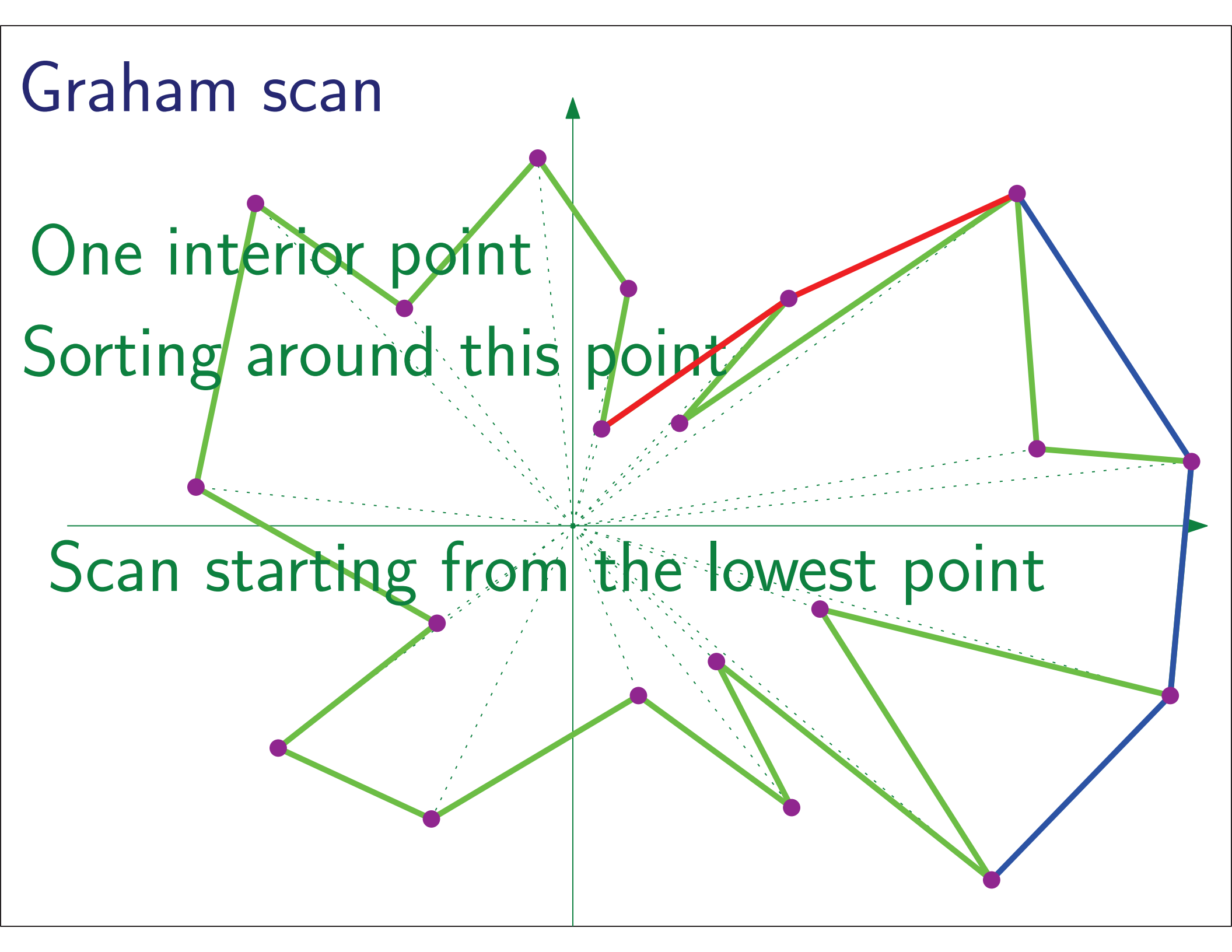
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

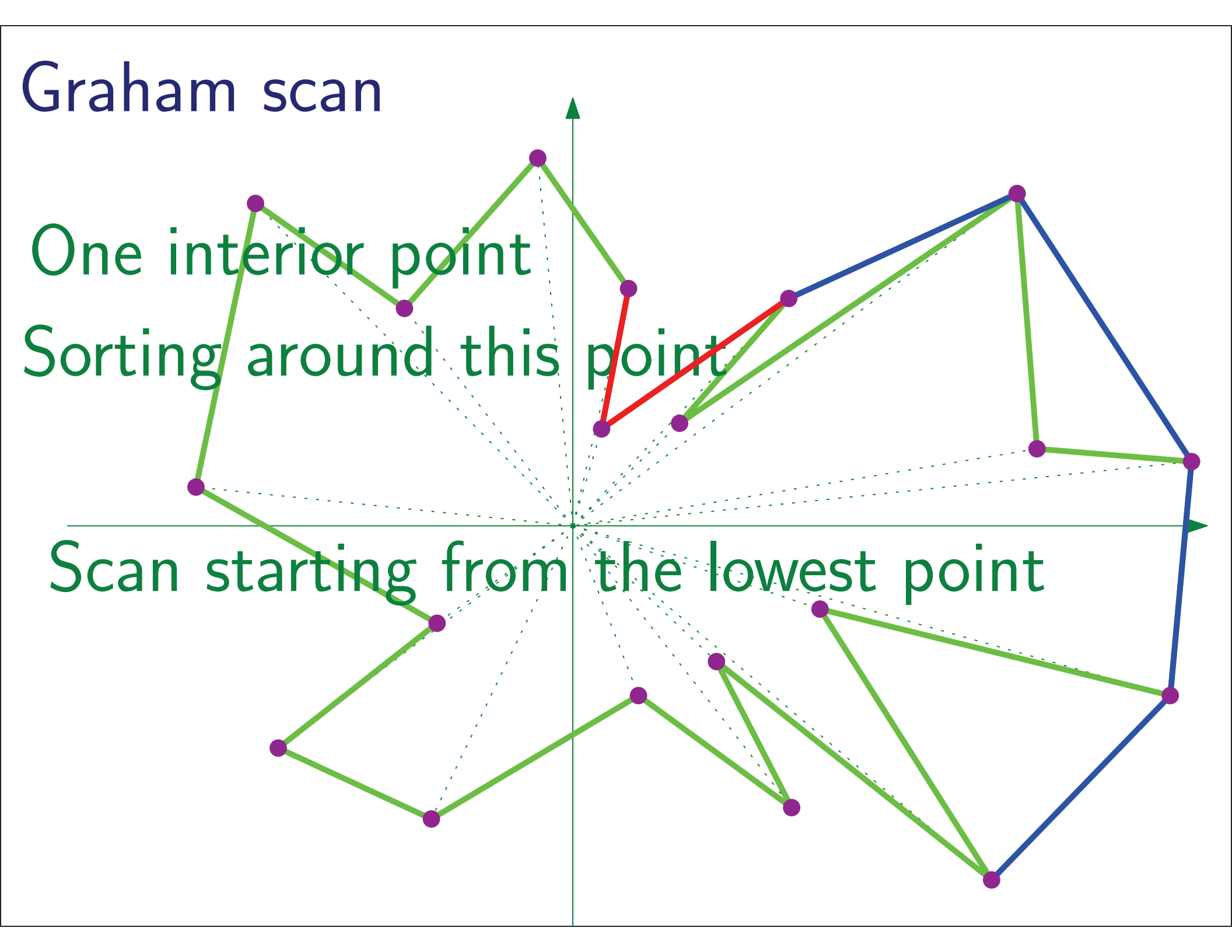
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

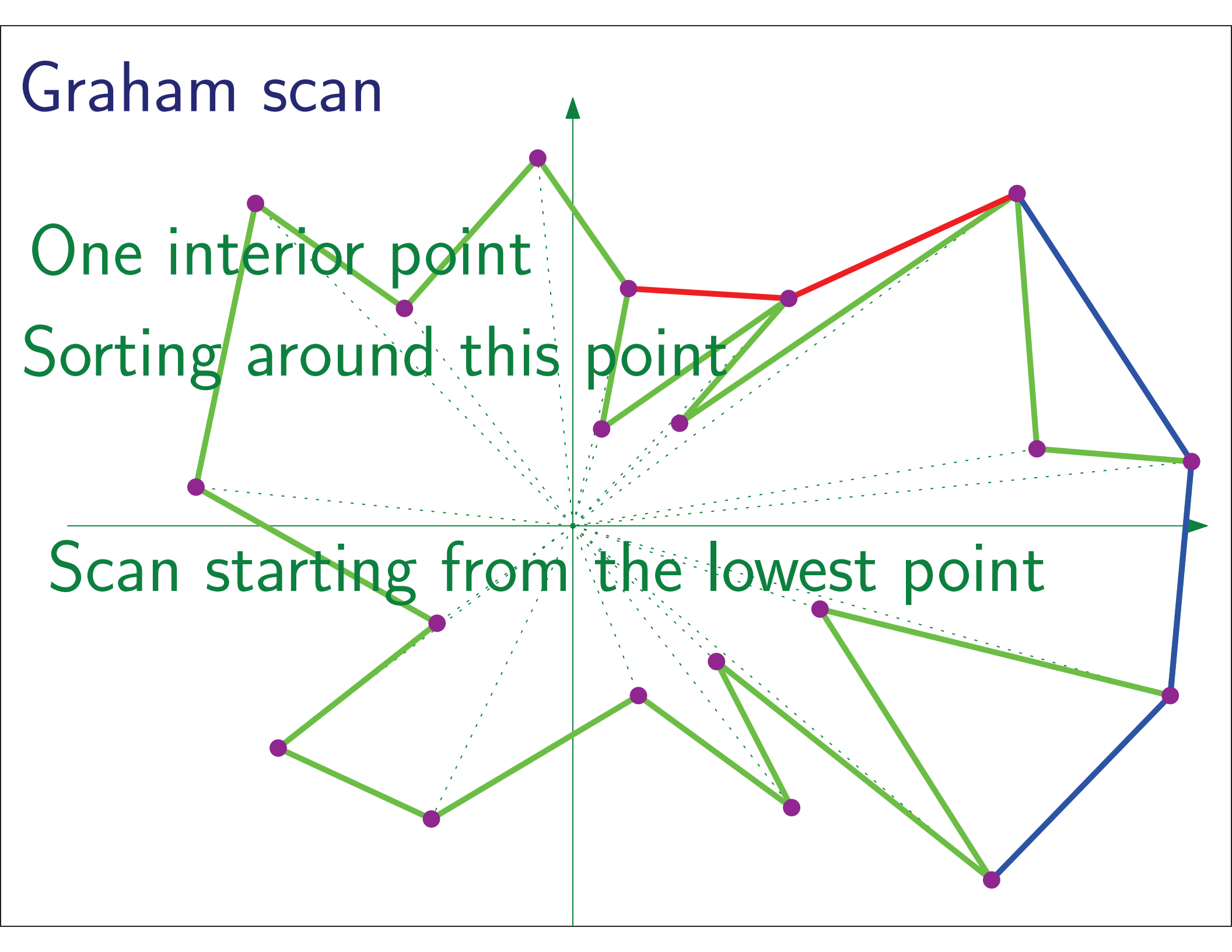
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

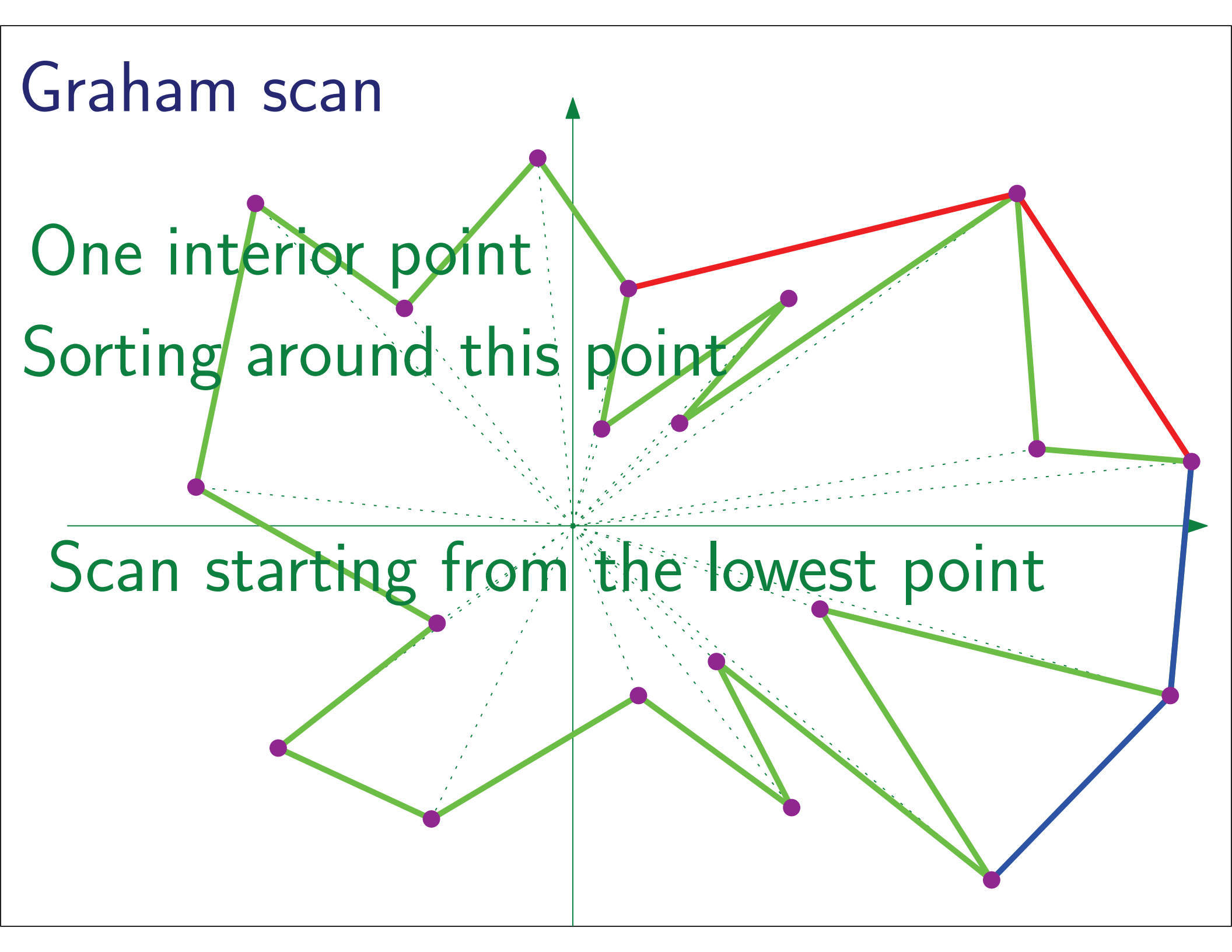
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

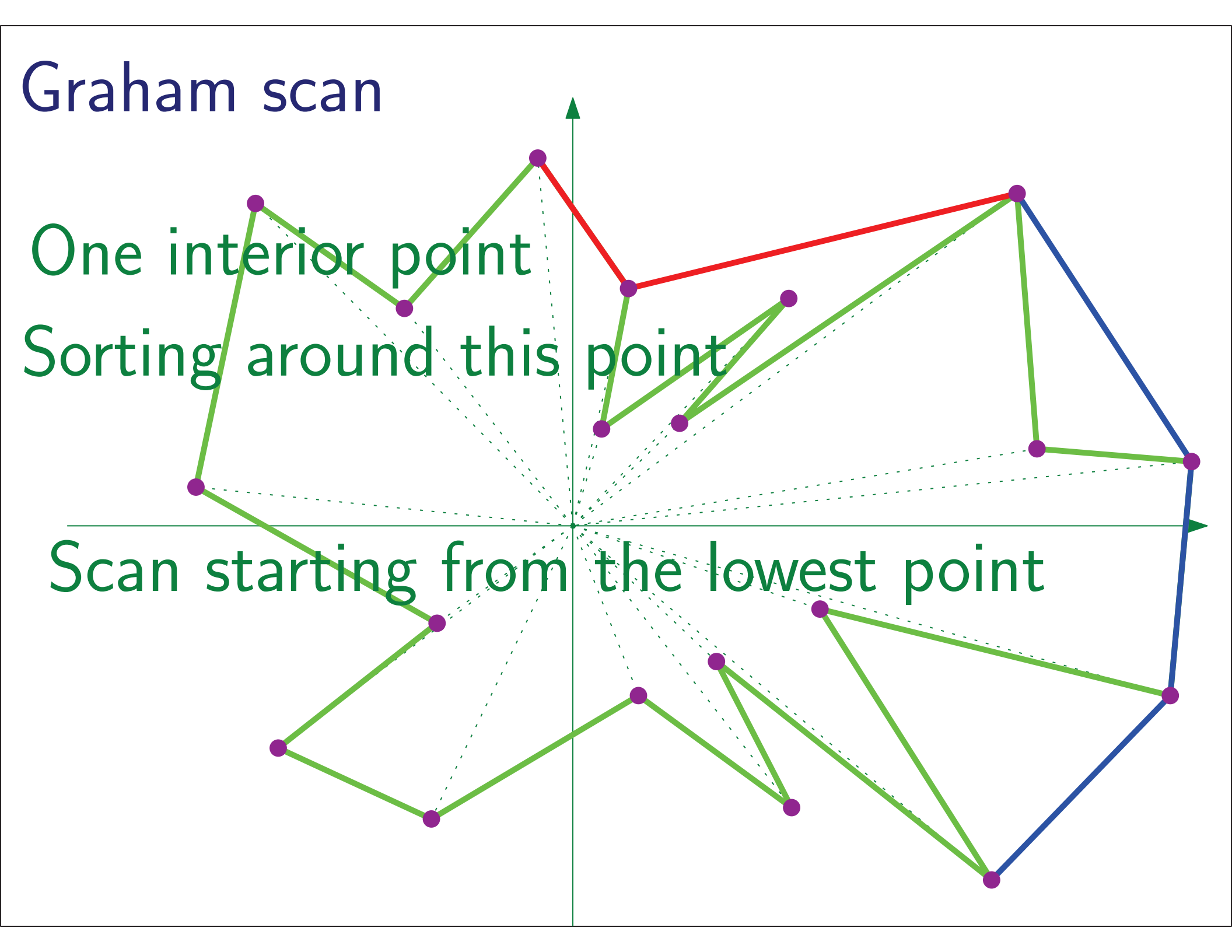
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

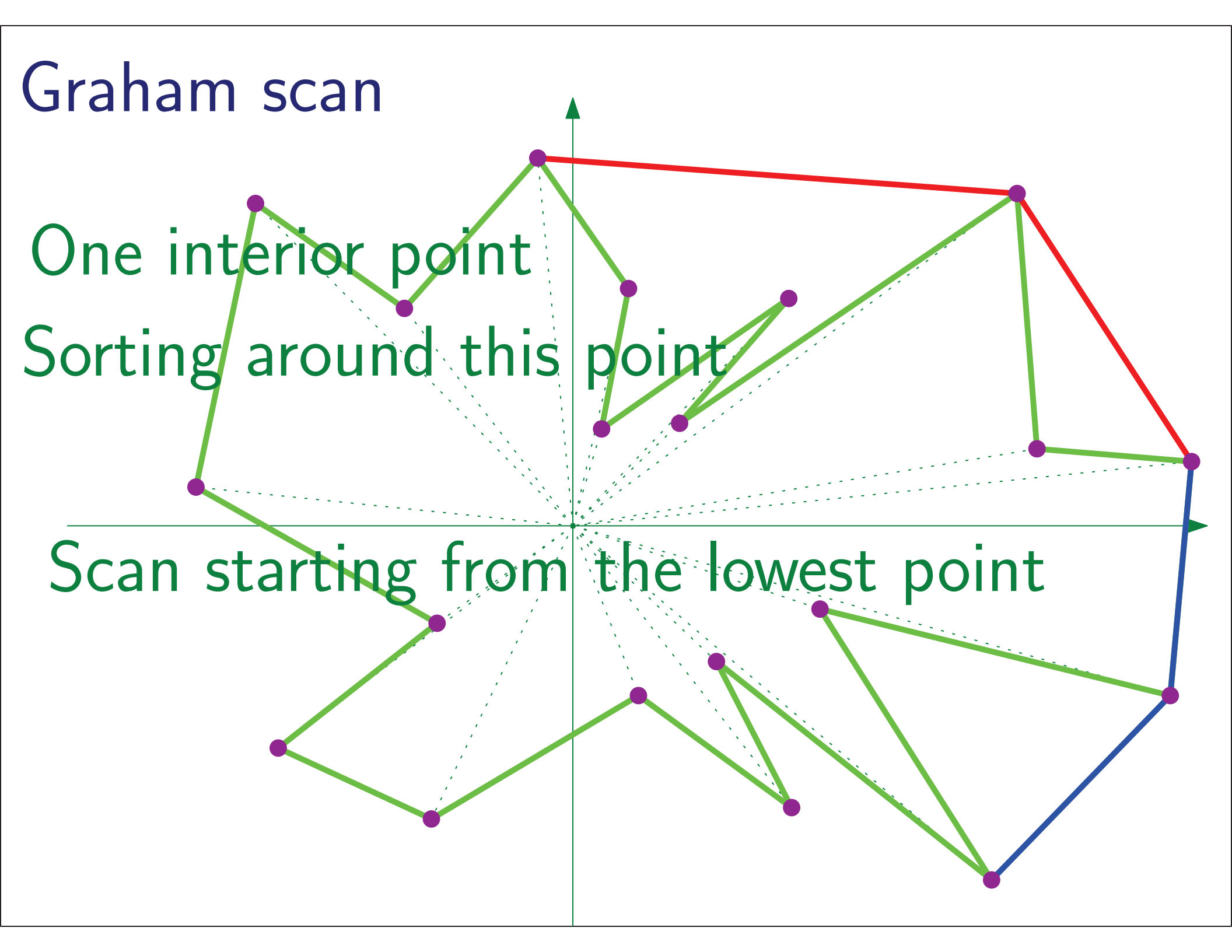
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

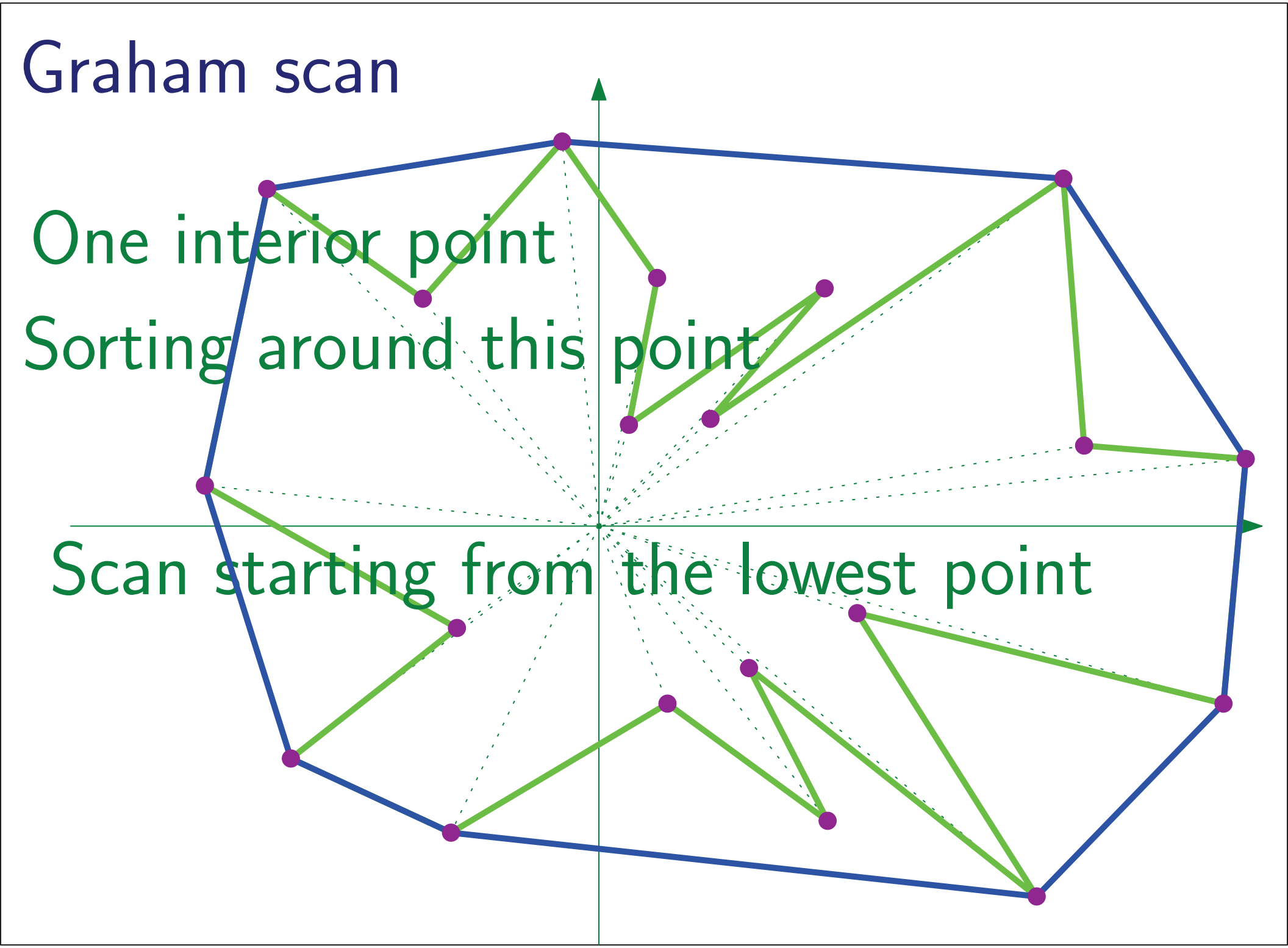
Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

Graham scan


One interior point
Sorting around this point
Scan starting from the lowest point

Graham Scan

Input : S a set of n points.
origin $=$ barycenter of 3 points of S;
sort S around the origin;
$u =$ the lowest point of S;
$v = u$;
while $v.next \neq u$
    if $(v, v.next, v.next.next)$ turn left
        $v = v.next$;
    else
        $v.next = v.next.next$;
        if $v \neq u$ $v = v.previous$;

Graham Scan
Complexity

Input : S a set of n points.
origin $=$ barycenter of 3 points of S;
sort S around the origin;
$u =$ the lowest point of S;
$v = u$;
while $v.next \neq u$
    if $(v, v.next, v.next.next)$ turn left
        $v = v.next$;
    else
        $v.next = v.next.next$;
        if $v \neq u$ $v = v.previous$;

Graham Scan
Complexity

Input : S a set of n points.
origin $=$ barycenter of 3 points of S;
sort S around the origin;
$u =$ the lowest point of S;
$v = u$;
while $v.next \neq u$
    if $(v, v.next, v.next.next)$ turn left
        $v = v.next$;
    else
        $v.next = v.next.next$;
        if $v \neq u$ $v = v.previous$;

Graham Scan
Complexity

Input : S a set of n points.
origin $=$ barycenter of 3 points of S;
sort S around the origin;
$u =$ the lowest point of S;
$v = u$;
while $v.next \neq u$
    if $(v, v.next, v.next.next)$ turn left
        $v = v.next$;
    else
        $v.next = v.next.next$;
        if $v \neq u$ $v = v.previous$;

$O(n \log n)$

Graham Scan
Complexity

Input : S a set of n points.

origin $=$ barycenter of 3 points of S; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S;

$v = u$;

while $v.next \neq u$

    if $(v, v.next, v.next.next)$ turn left

        $v = v.next$;

    else

        $v.next = v.next.next$;

        if $v \neq u$ $v = v.previous$;

Graham Scan
Complexity

Input : S a set of n points.

origin $=$ barycenter of 3 points of S; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S; $O(n)$

$v = u$;

while $v.next \neq u$

    if $(v, v.next, v.next.next)$ turn left

        $v = v.next$;

    else

        $v.next = v.next.next$;

        if $v \neq u$ $v = v.previous$;

Graham Scan
Complexity

Input : S a set of n points.

origin $=$ barycenter of 3 points of S; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S; $O(n)$

$v = u$;

while $v.next \neq u$

    if $(v, v.next, v.next.next)$ turn left

        $v = v.next$;

    else

        $v.next = v.next.next$;

        if $v \neq u$ $v = v.previous$;

Graham Scan
Complexity

Input : S a set of n points.

origin $=$ barycenter of 3 points of S; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S; $O(n)$

$v = u$;

while $v.next \neq u$

if $(v, v.next, v.next.next)$ turn left

$v = v.next$;

else

$v.next = v.next.next$;

if $v \neq u$ $v = v.previous$;

Graham Scan
Complexity

Input : S a set of n points.

origin $=$ barycenter of 3 points of S; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S; $O(n)$

$v = u$;

while $v.next \neq u$

if $(v, v.next, v.next.next)$ turn left

$v = v.next$;

else

$v.next = v.next.next$;

if $v \neq u$ $v = v.previous$;

at most n deletions

Graham Scan
Complexity

Input : S a set of n points.

origin $=$ barycenter of 3 points of S; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S; $O(n)$

$v = u$;

while $v.next \neq u$

if $(v, v.next, v.next.next)$ turn left

$v = v.next$;

else

$v.next = v.next.next$;

if $v \neq u$ $v = v.previous$;

at most n deletions

Graham Scan
Complexity

Input : S a set of n points.

origin $=$ barycenter of 3 points of S; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S; $O(n)$

$v = u$;

while $v.next \neq u$

  if $(v, v.next, v.next.next)$ turn left

    $v = v.next$;

  else

    $v.next = v.next.next$;

    if $v \neq u$ $v = v.previous$;

at most n times

at most n deletions

Graham Scan
Complexity

Input : S a set of n points.

origin $=$ barycenter of 3 points of S; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S; $O(n)$

$v = u$;

while $v.next \neq u$

  if $(v, v.next, v.next.next)$ turn left

    $v = v.next$;

  else

    $v.next = v.next.next$;

    if $v \neq u$ $v = v.previous$;

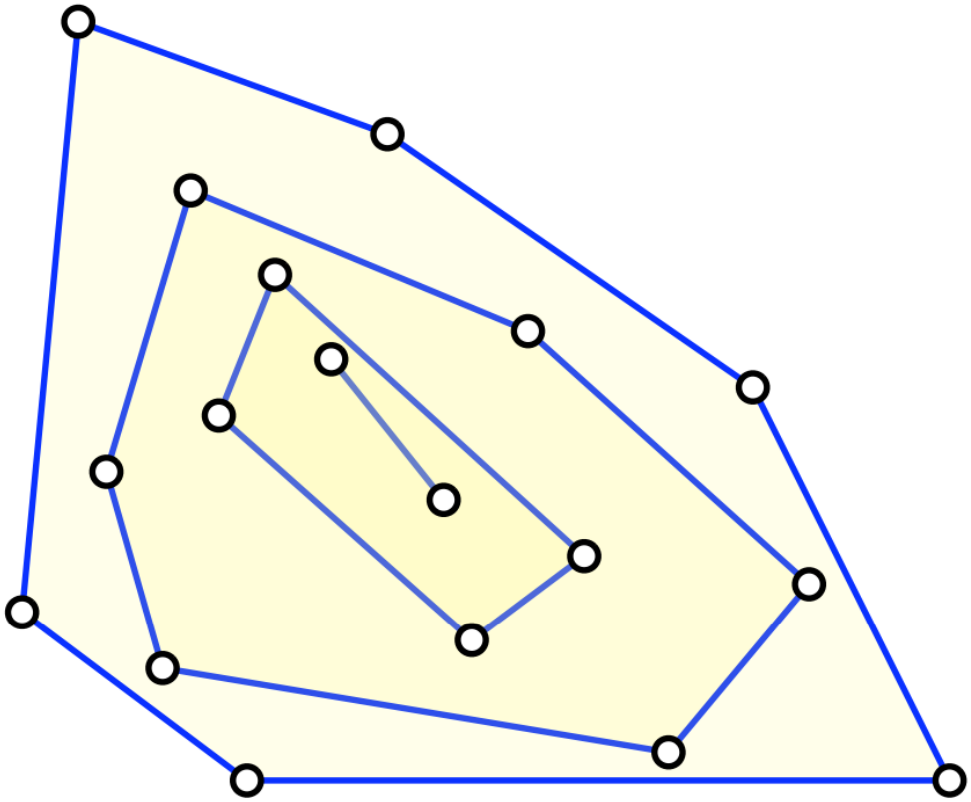
$O(n)$

Graham Scan
Complexity

Input : S a set of n points.
origin $=$ barycenter of 3 points of S;
sort S around the origin;
$u =$ the lowest point of S;
$v = u$;
while $v.next \neq u$
    if $(v, v.next, v.next.next)$ turn left
        $v = v.next$;
    else
        $v.next = v.next.next$;
        if $v \neq u$ $v = v.previous$;

$O(n \log n)$

“Onion peeling”



The *convex layers* of a point set X are defined by repeatedly computing the convex hull of X and removing its vertices from X, until X is empty.

(a) Describe an algorithm to compute the convex layers of a given set of n points in the plane in $O(n^2)$ time.

*(b) Describe an algorithm to compute the convex layers of a given set of n points in the plane in $O(n \log n)$ time.

[http://web.engr.illinois.edu/~jeffe/teaching/compgeom/notes/01-convexhull.pdf]

Developing an algorithm

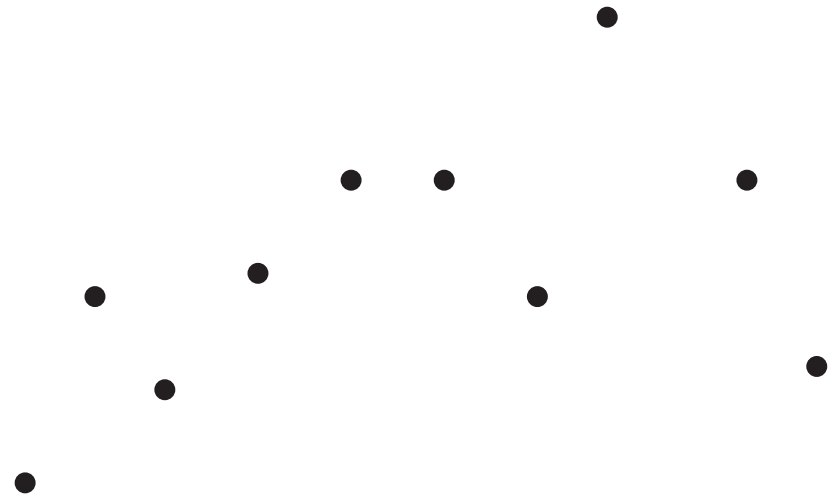
Another approach: incremental, from left to right

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

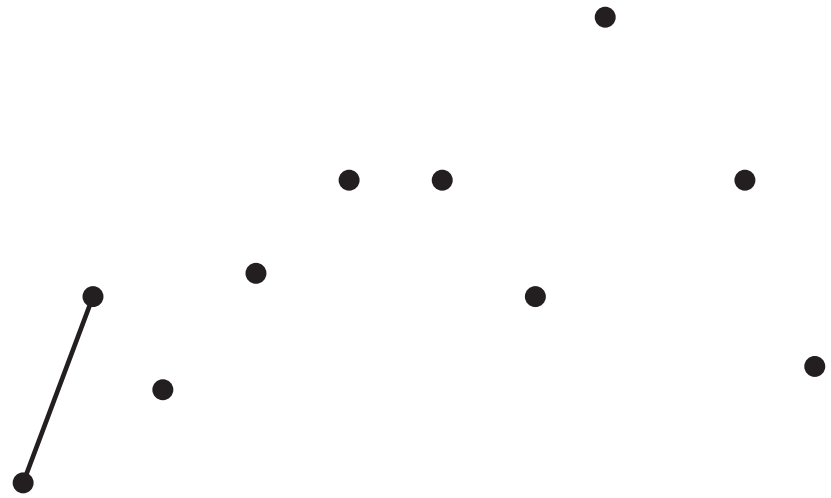
Developing an algorithm

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



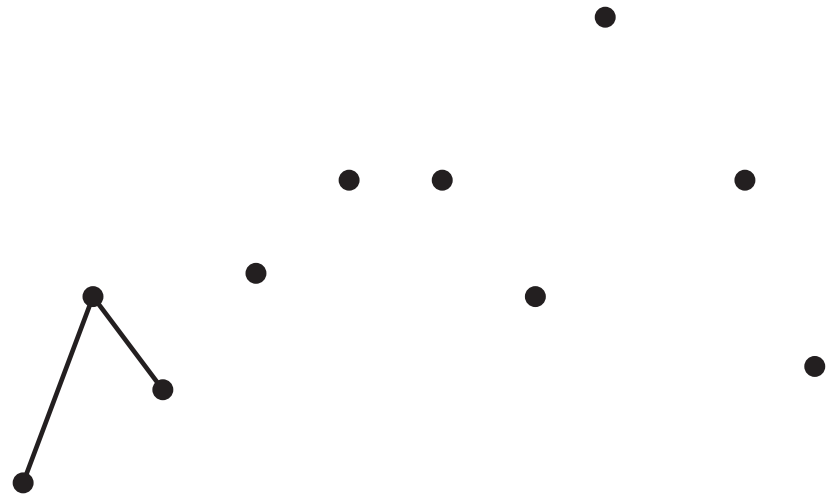
Developing an algorithm

Initialize by inserting the leftmost two points



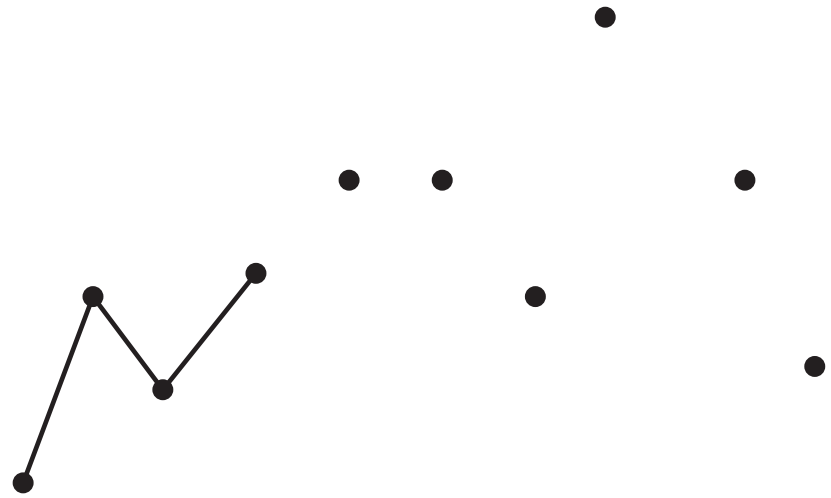
Developing an algorithm

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



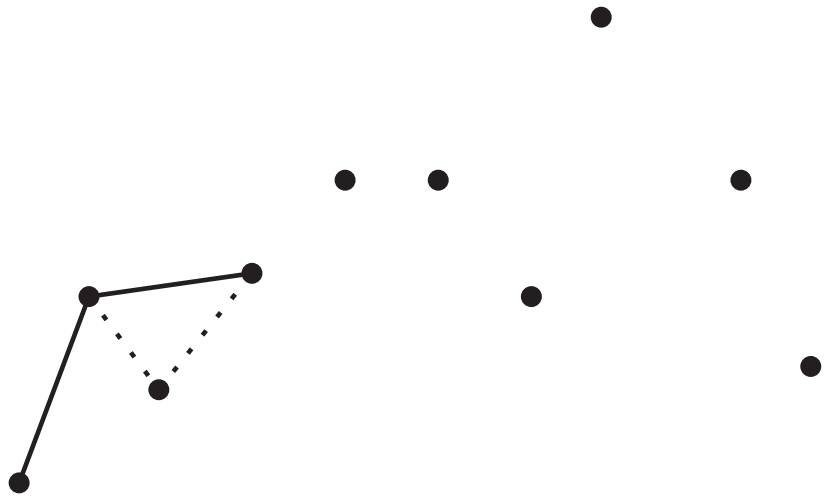
Developing an algorithm

If we add the fourth point we get a left turn at the third point



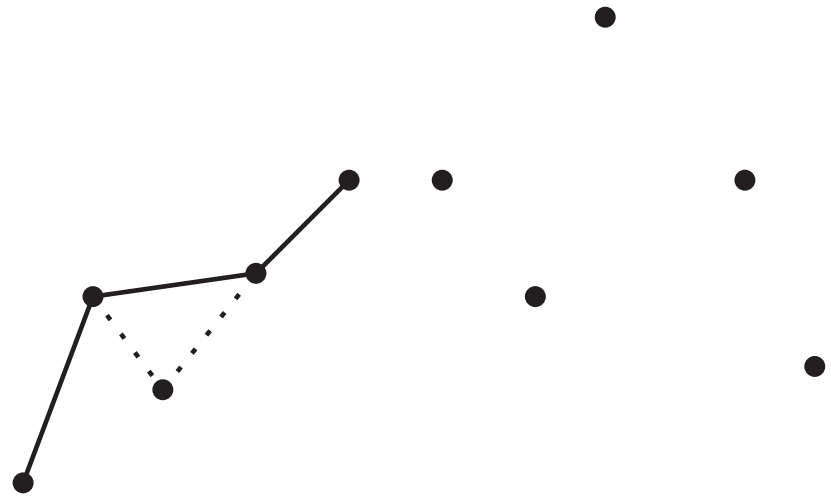
Developing an algorithm

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



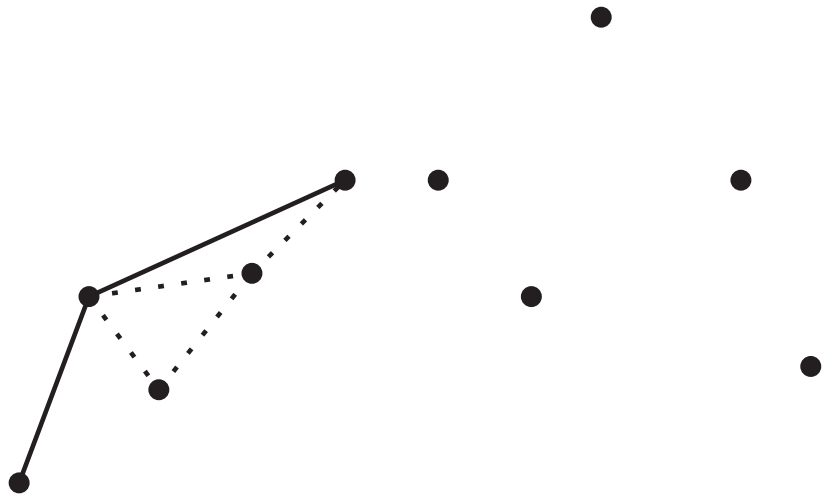
Developing an algorithm

If we add the fifth point we get a left turn at the fourth point



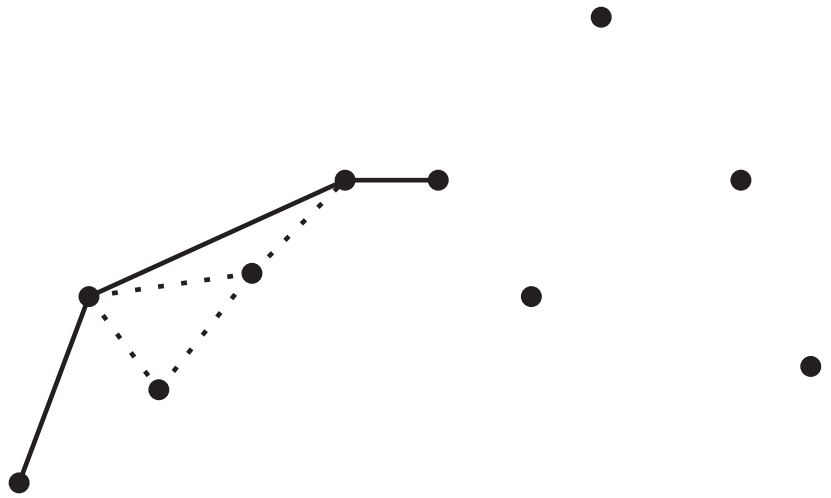
Developing an algorithm

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



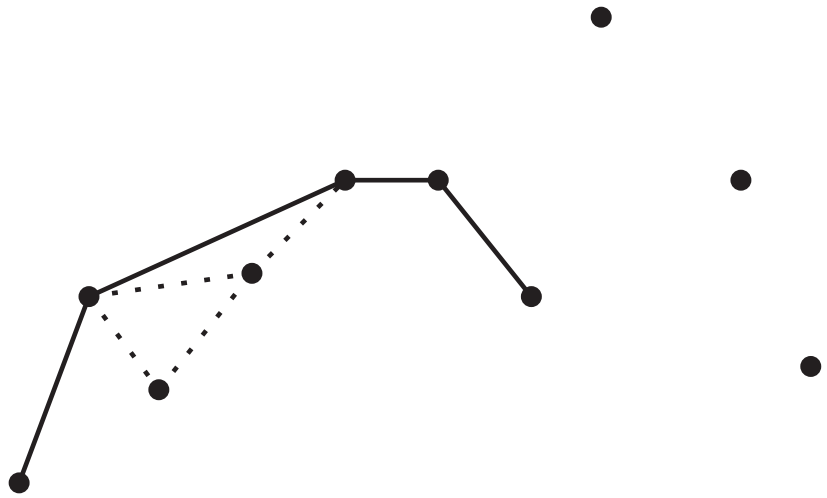
Developing an algorithm

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



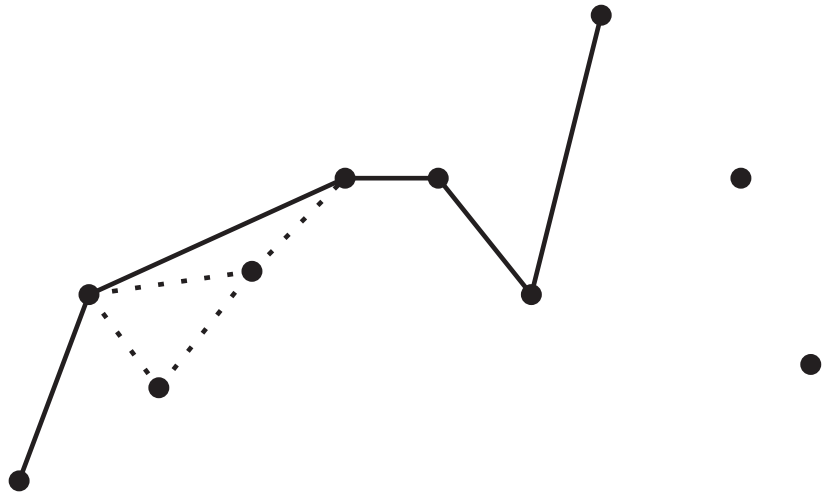
Developing an algorithm

We also just add the seventh point



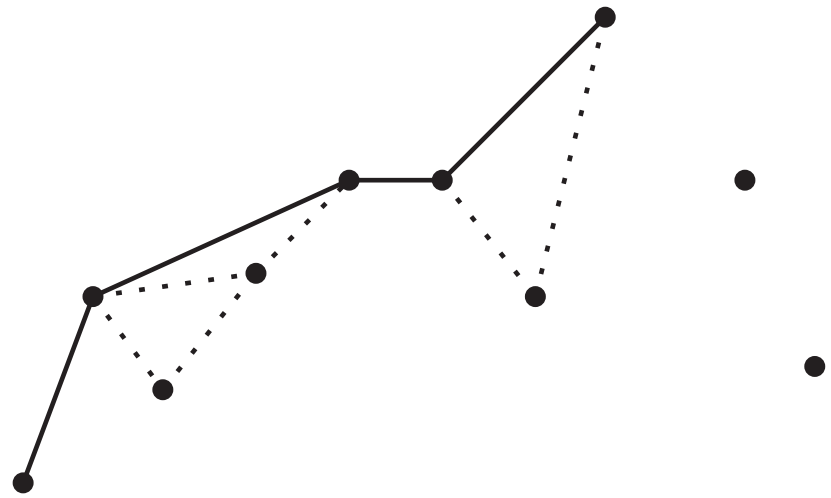
Developing an algorithm

When adding the eight point ... we must remove the seventh point



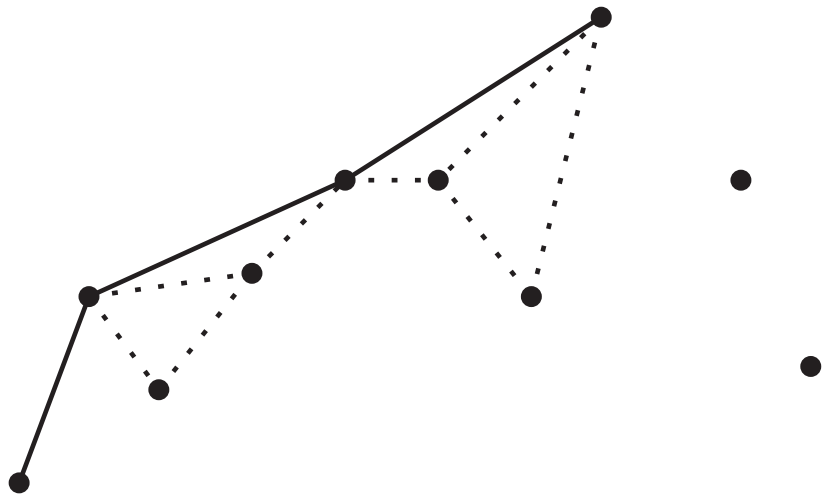
Developing an algorithm

. . . we must remove the seventh point



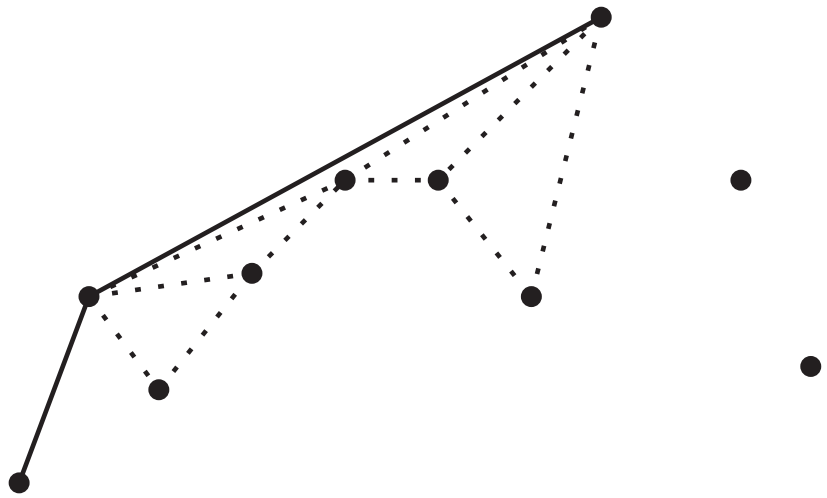
Developing an algorithm

. . . and also the sixth point



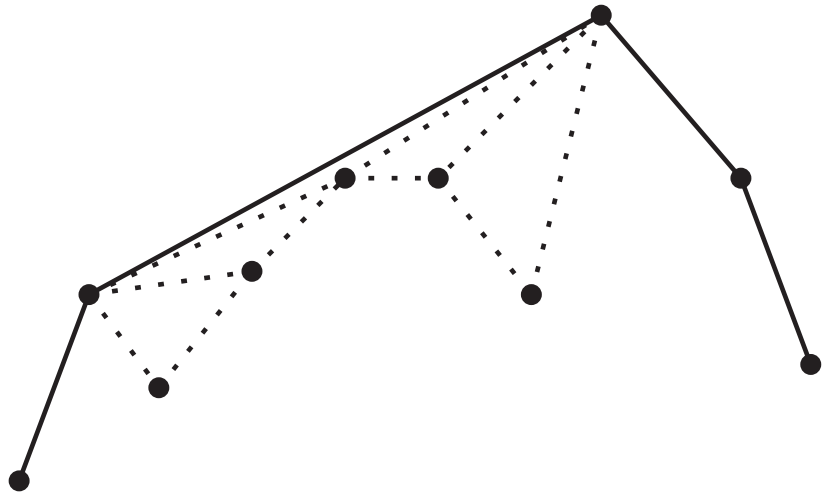
Developing an algorithm

. . . and also the fifth point

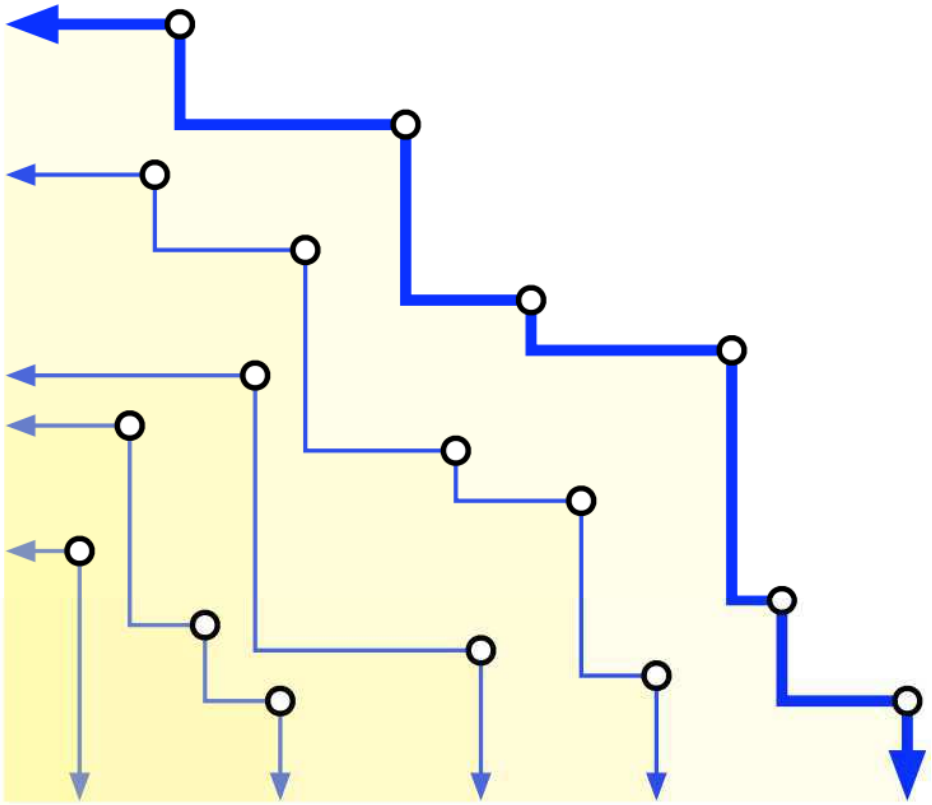


Developing an algorithm

After two more steps we get:



“Pareto optimality”



Let X be a set of points in the plane. A point p in X is *Pareto-optimal* if no other point in X is both above and to the right of p. The Pareto-optimal points can be connected by horizontal and vertical lines into the *staircase* of X, with a Pareto-optimal point at the top right corner of every step. See the figure above.

[http://web.engr.illinois.edu/~jeffe/teaching/compgeom/notes/01-convexhull.pdf]

Deterministic incremental algorithm

Complexity

$O(n \log n)$

Input : S set of n points.

sort S in x;

initialize a circular list with the 3 leftmost points

such that u is on the right $u, u.next, u.next.next$ turn left;

For v the next point in x

w=u

while $(v, u, u.next)$ turn right

$u = u.next$;

$v.next = u$; $u.previous = v$;

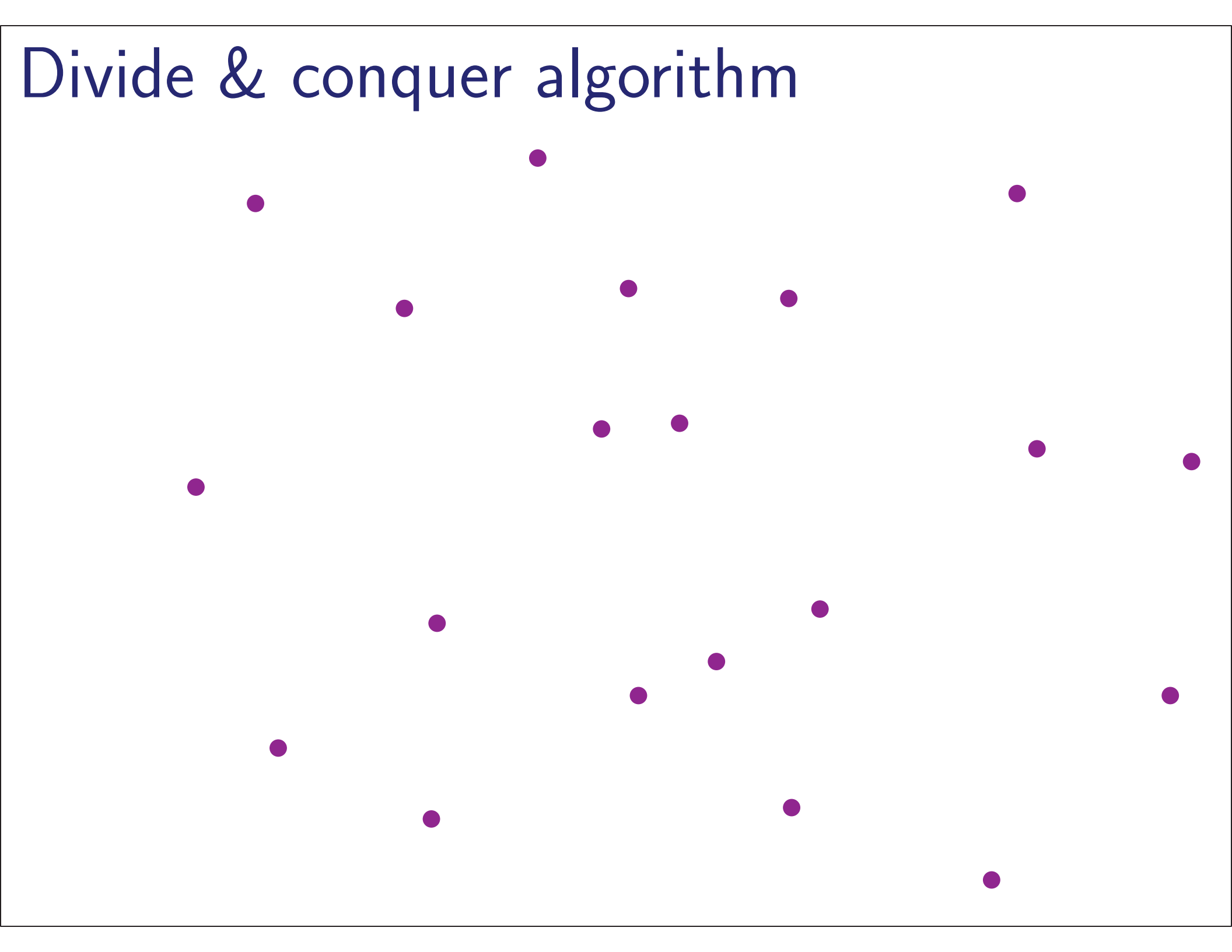
while $(v, w, w.previous)$ turn left

$w = w.previous$;

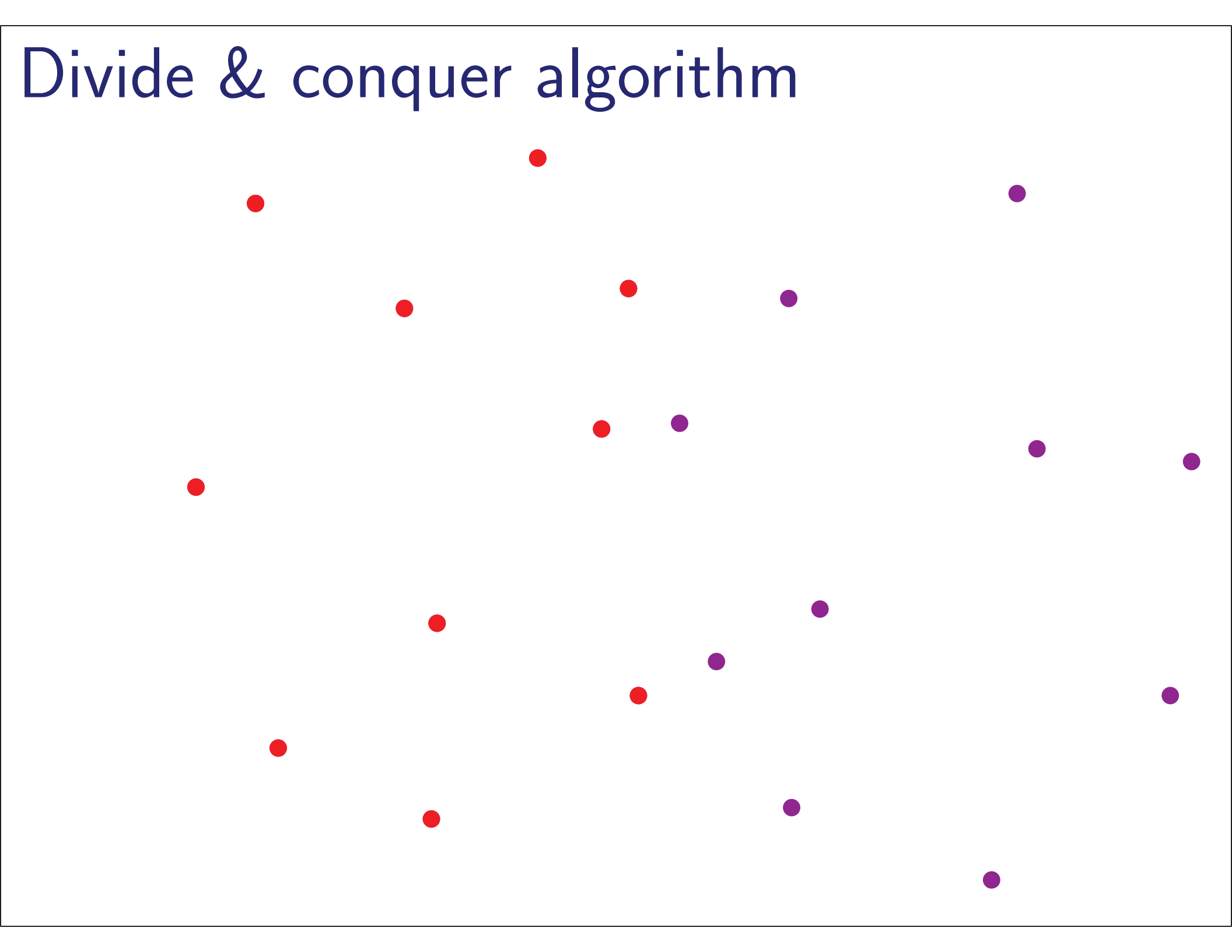
$v.previous = w$; $w.next = v$;

$u = v$;

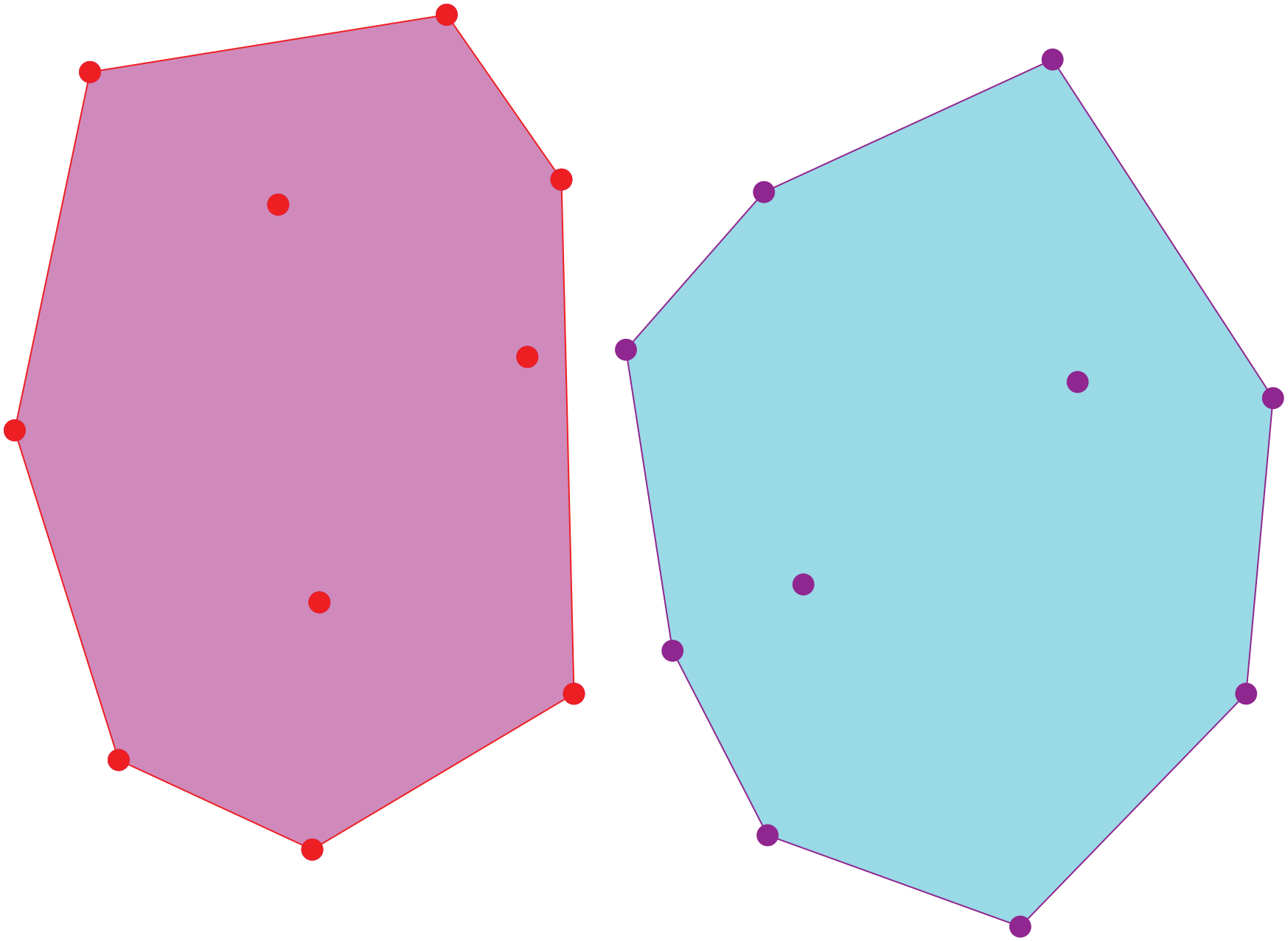
Divide & conquer algorithm



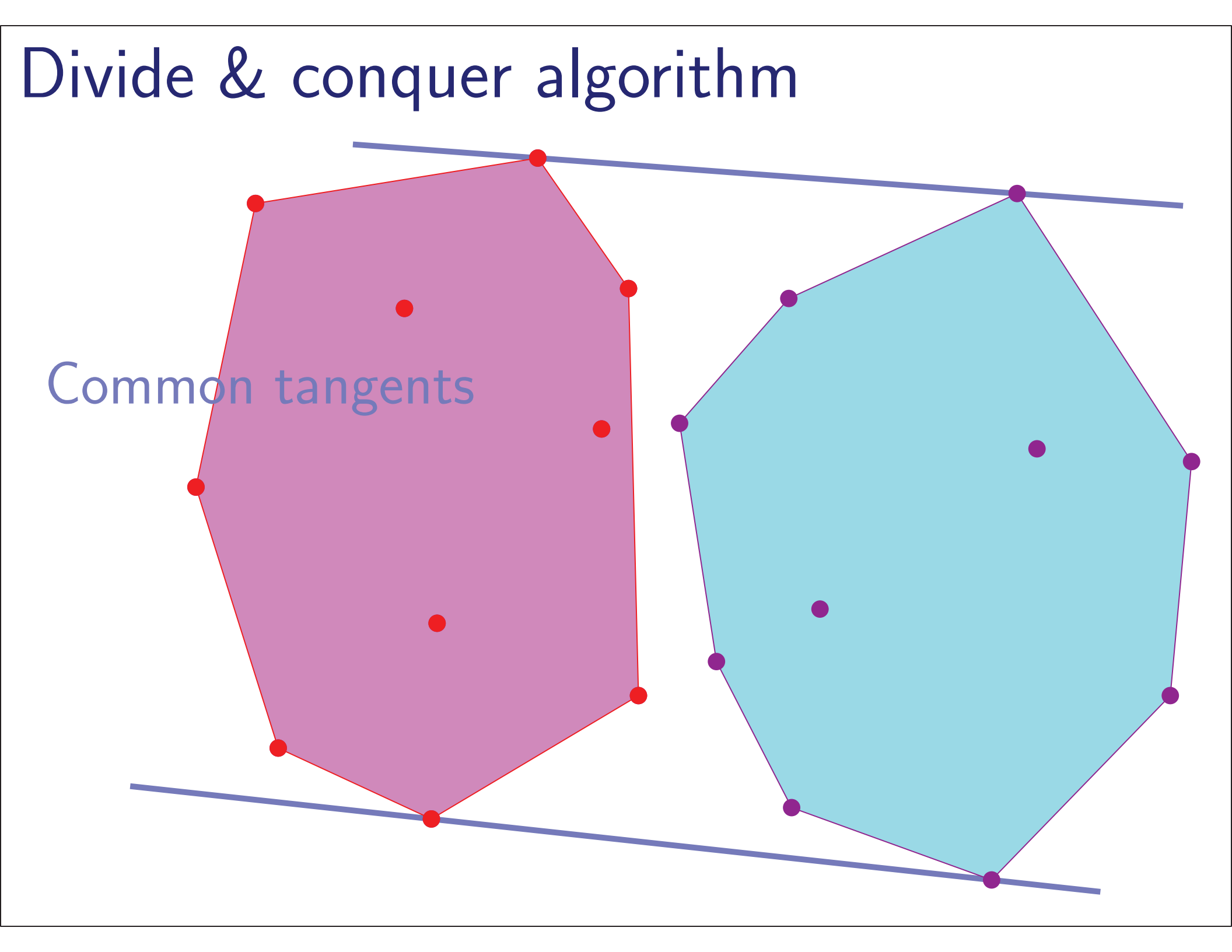
Divide & conquer algorithm



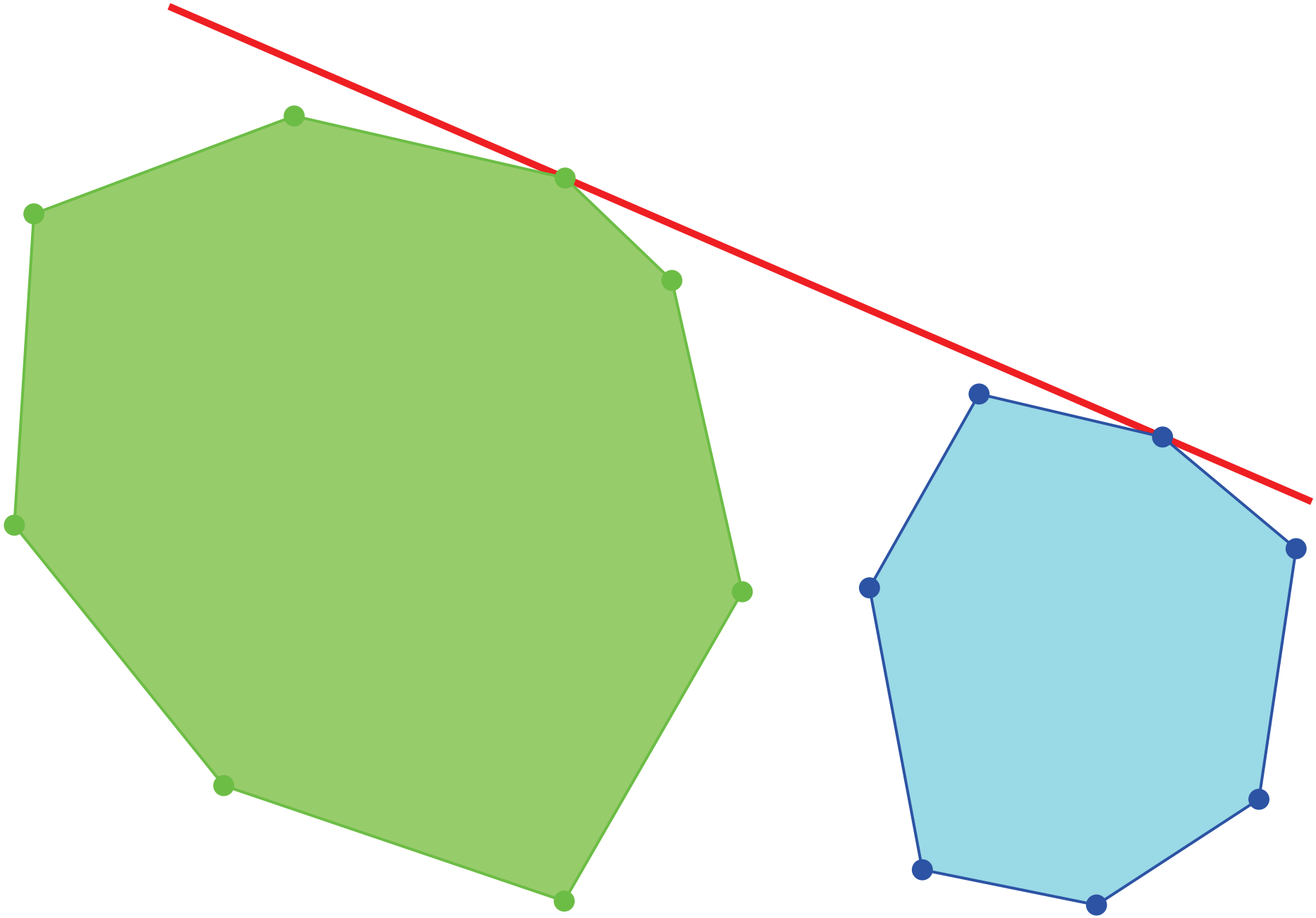
Divide & conquer algorithm



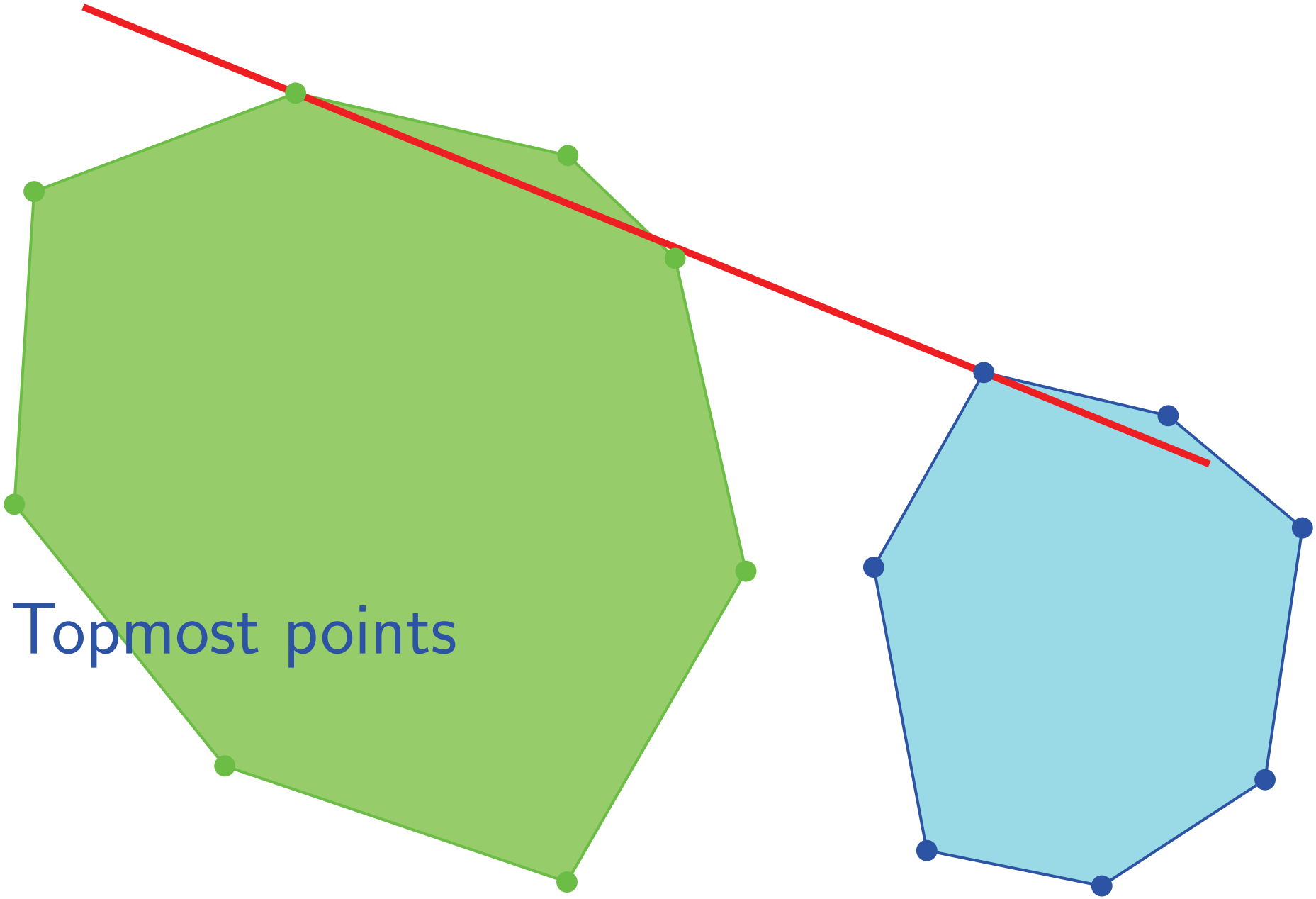
Divide & conquer algorithm



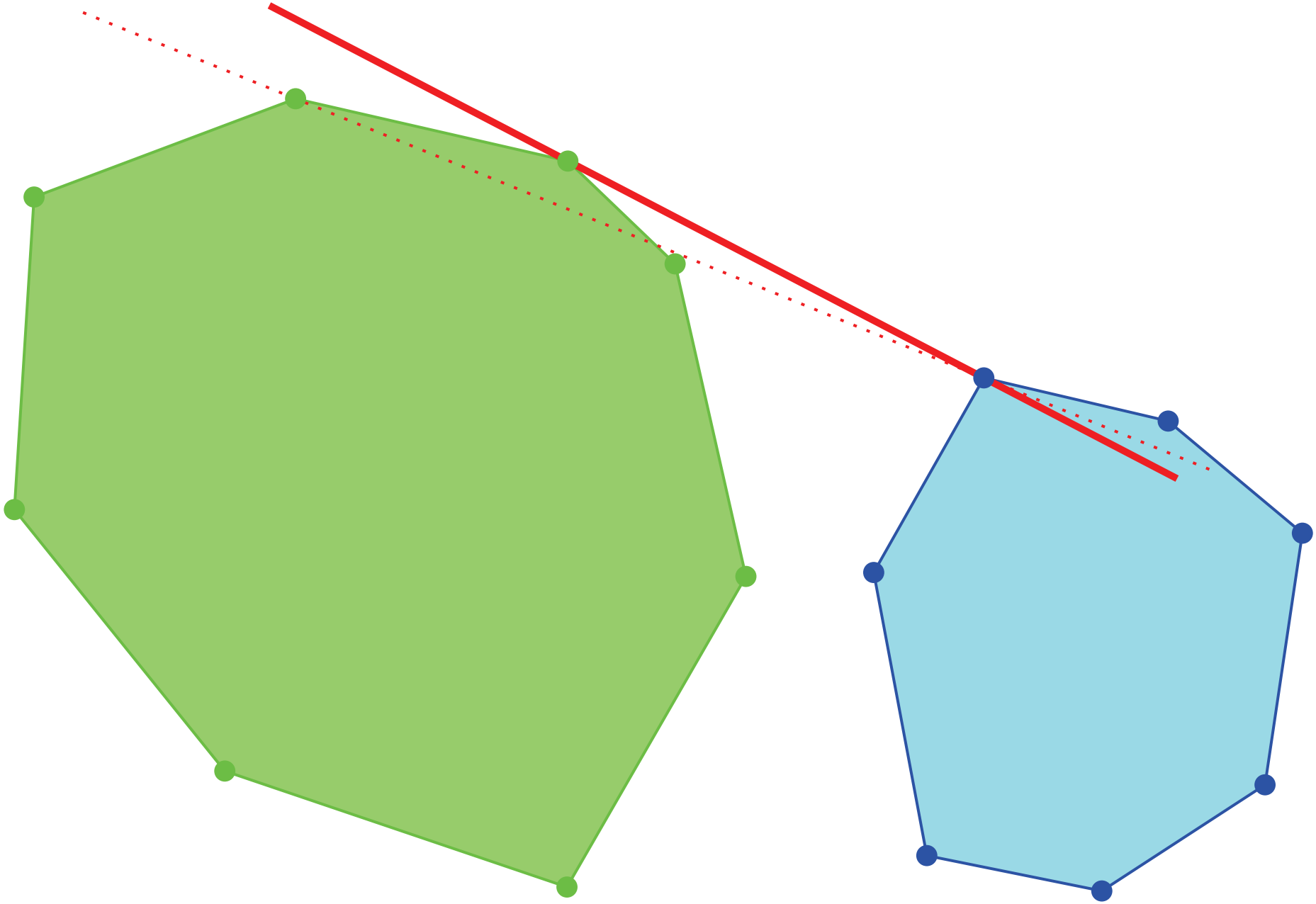
Upper tangent



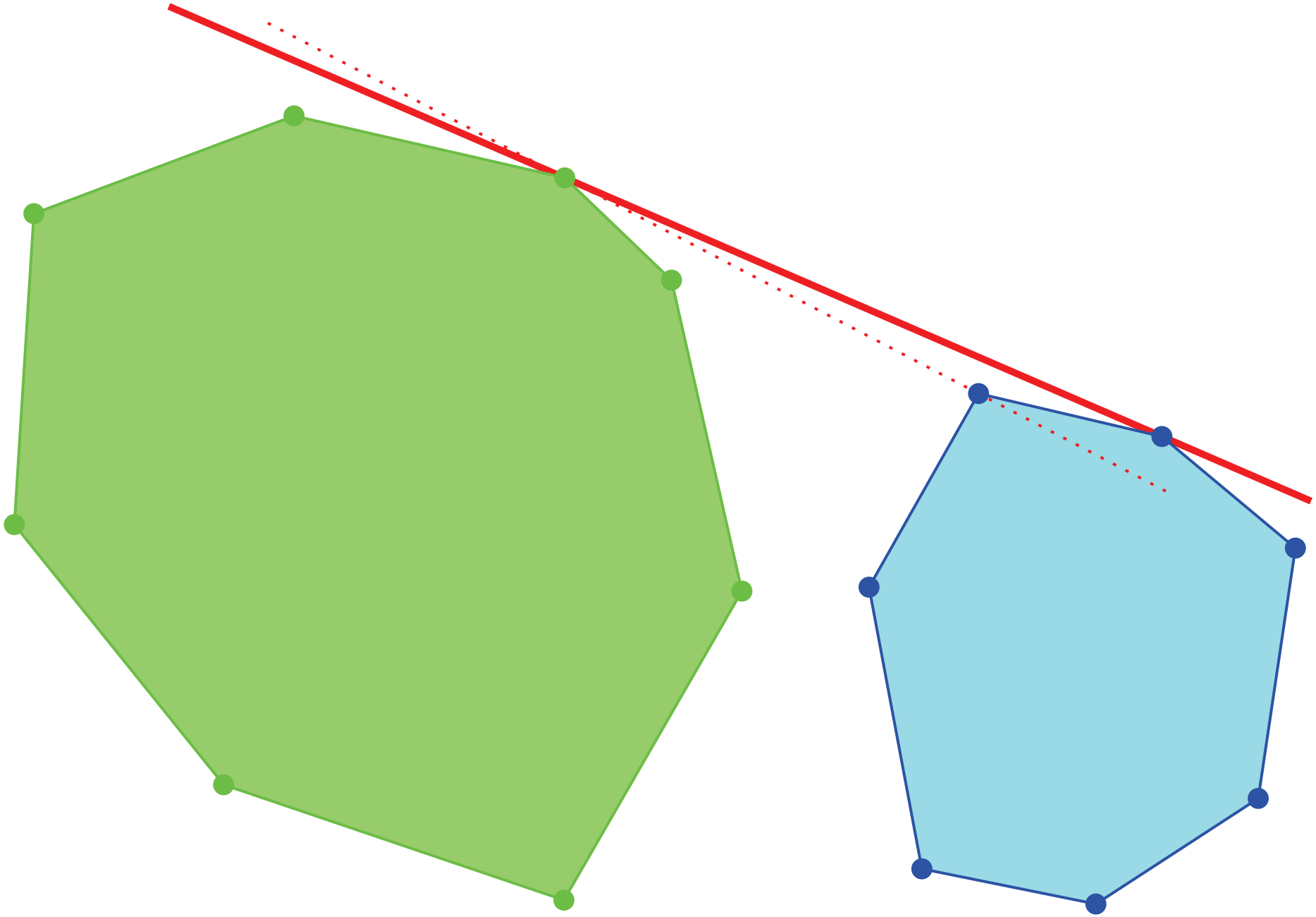
Upper tangent



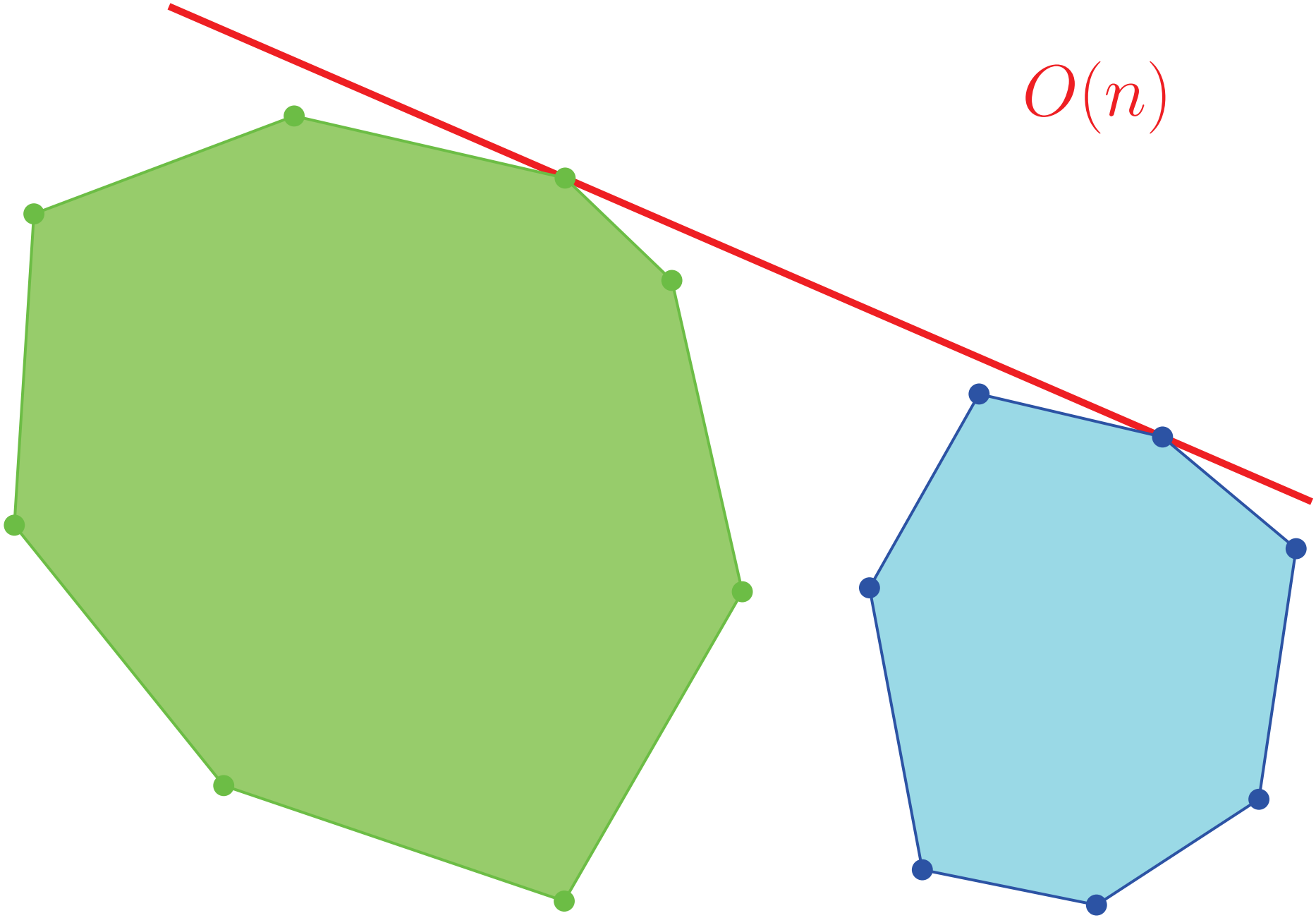
Upper tangent



Upper tangent



Upper tangent



Divide & conquer algorithm

Complexity

$$f(n) =$$

Divide & conquer algorithm

Complexity

$$f(n) = A \cdot n + f(\tfrac{n}{2}) + f(\tfrac{n}{2})$$

Divide & conquer algorithm

Complexity

$$f(n) = A \cdot n + f(\tfrac{n}{2}) + f(\tfrac{n}{2})$$

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

Divide & conquer algorithm

Complexity

$$f(n) = A \cdot n + f(\tfrac{n}{2}) + f(\tfrac{n}{2})$$

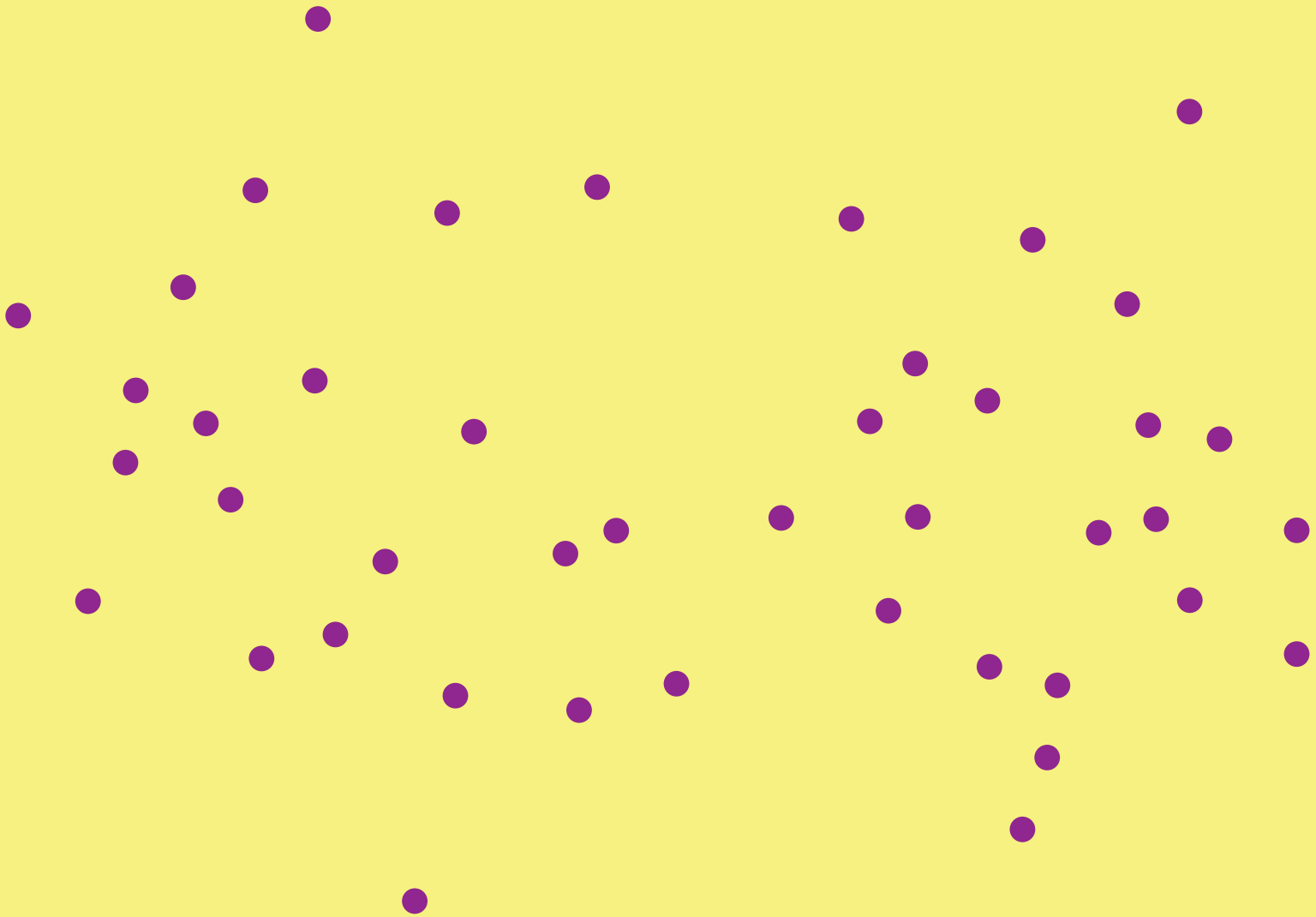
$$= O(n \log n)$$

Divide and merge in $O(n)$

Balanced partition

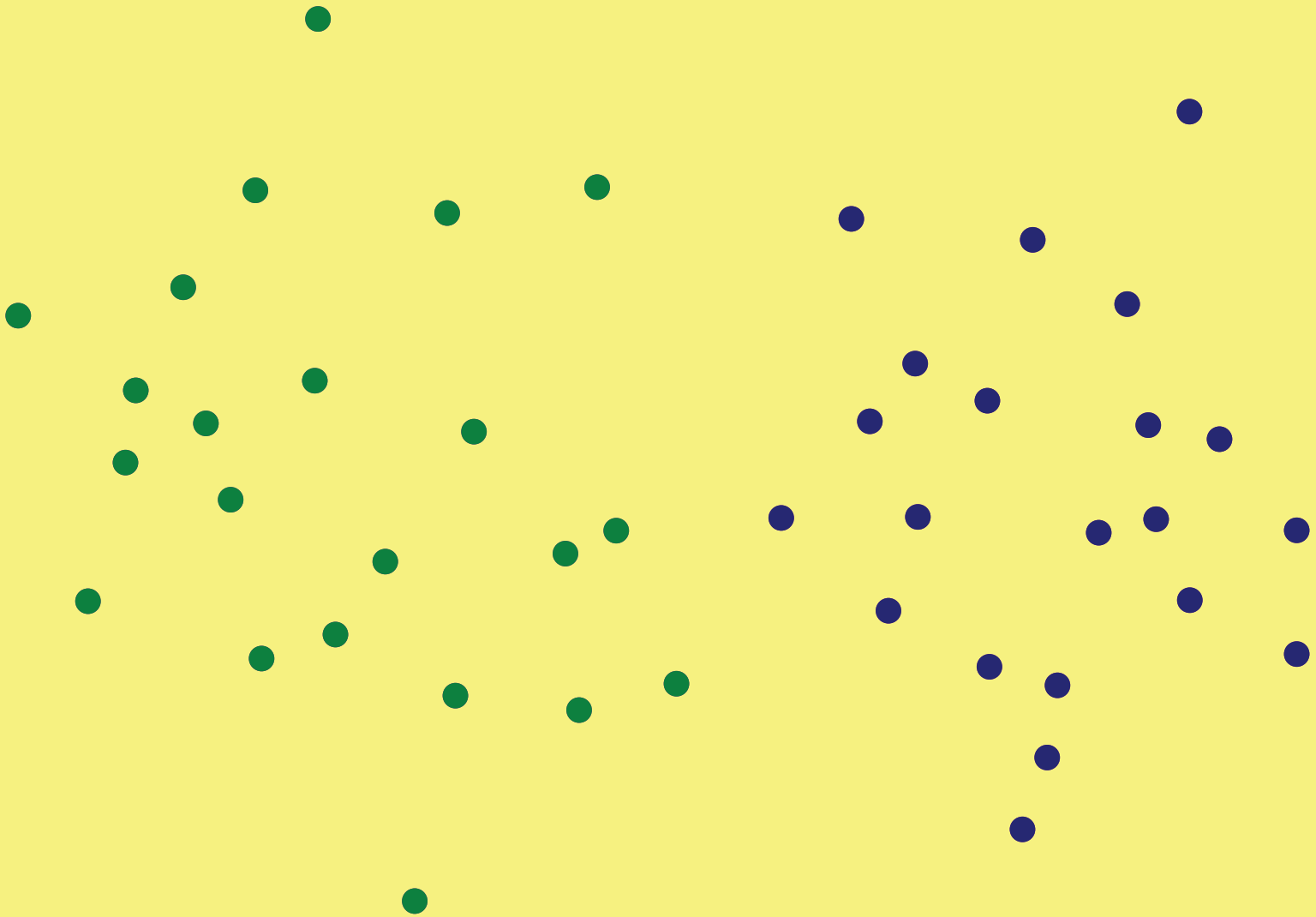
(preprocessing in $O(n \log n)$

3D Divide & conquer algorithm

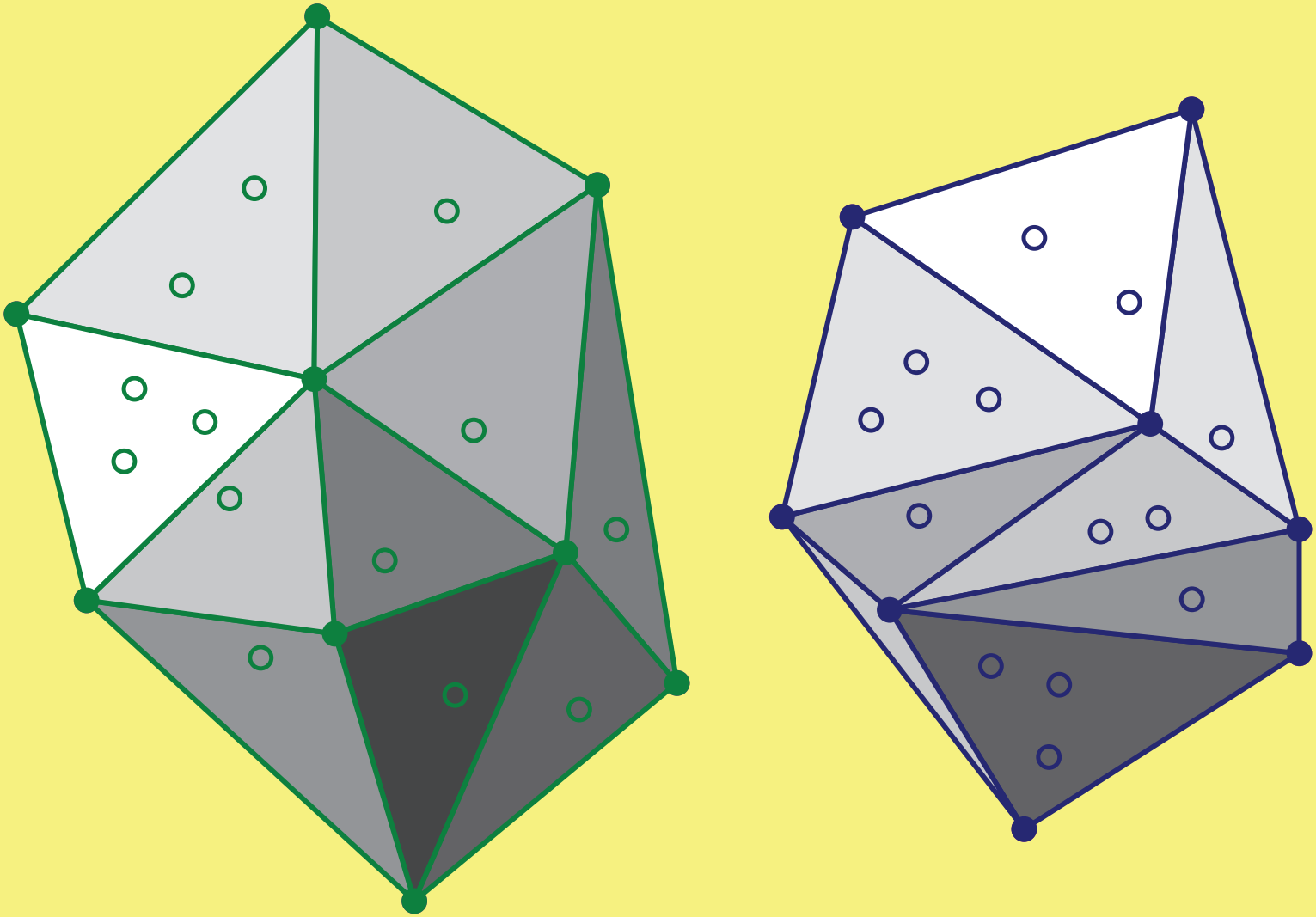


3D Divide & conquer algorithm

Sort in x and divide

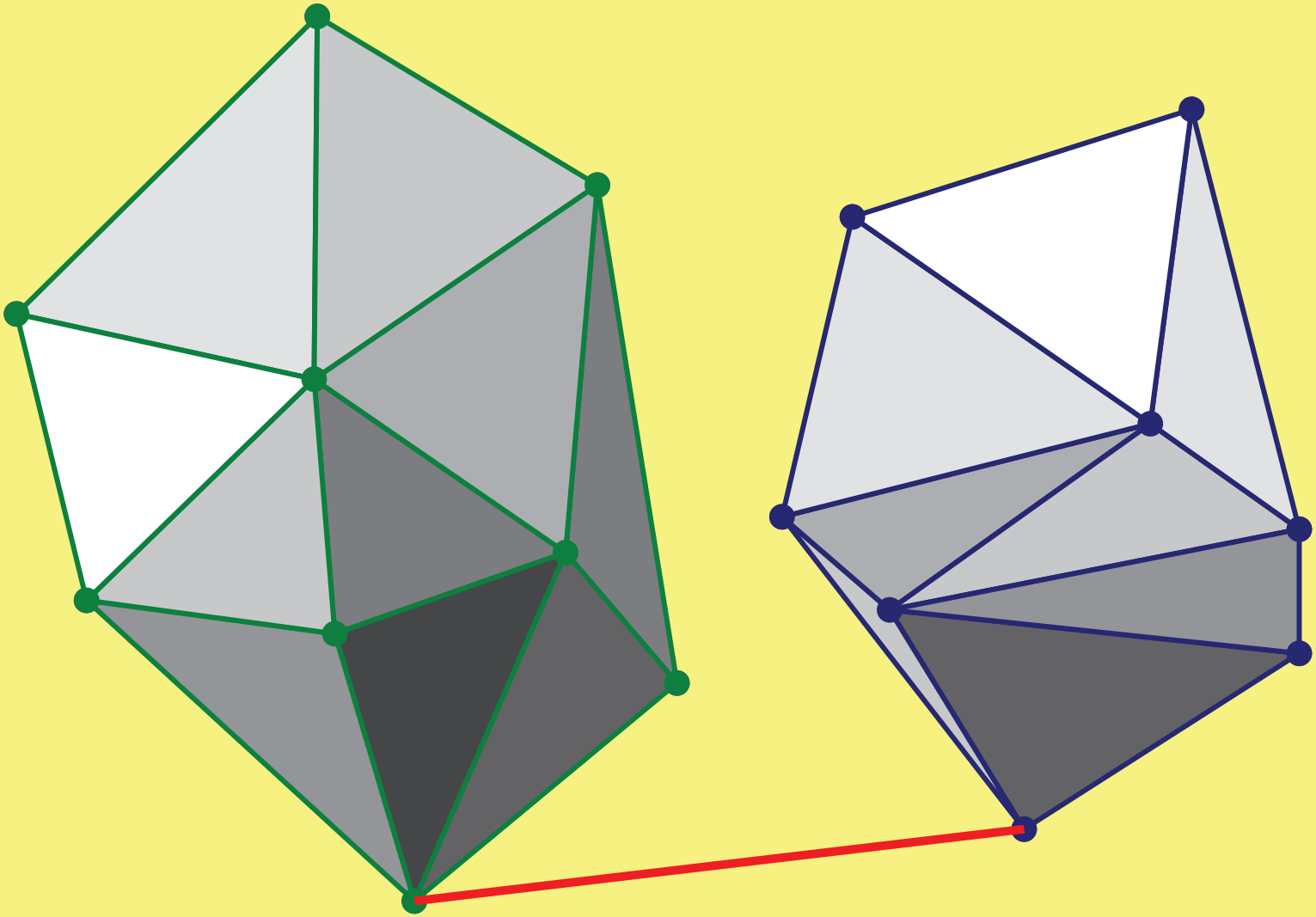


3D Divide & conquer algorithm



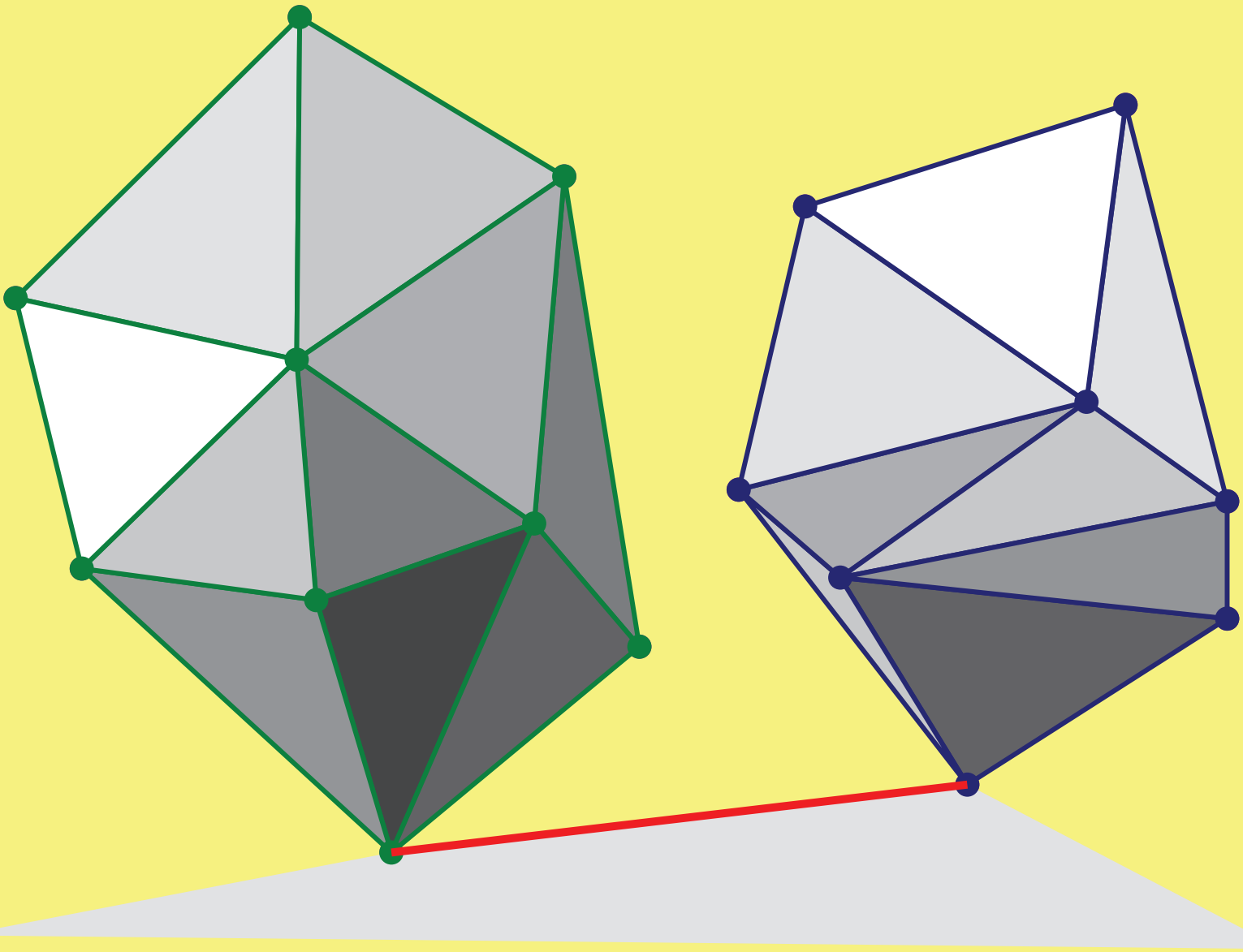
3D Divide & conquer algorithm

Find new edge: construct the CH of the projections,
use the 2d algo for bitangent



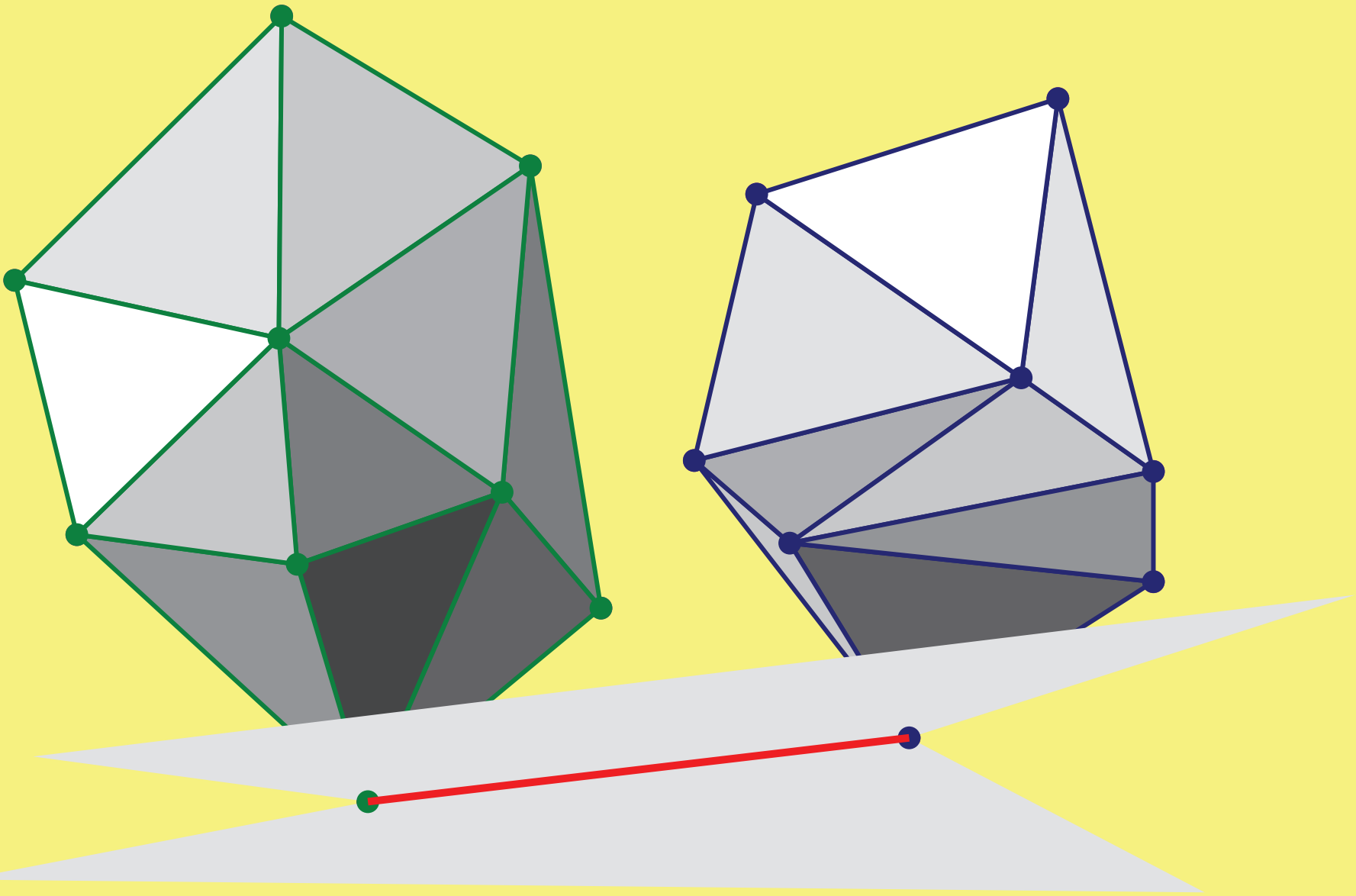
3D Divide & conquer algorithm

Use a wrapping algorithm around the new edges



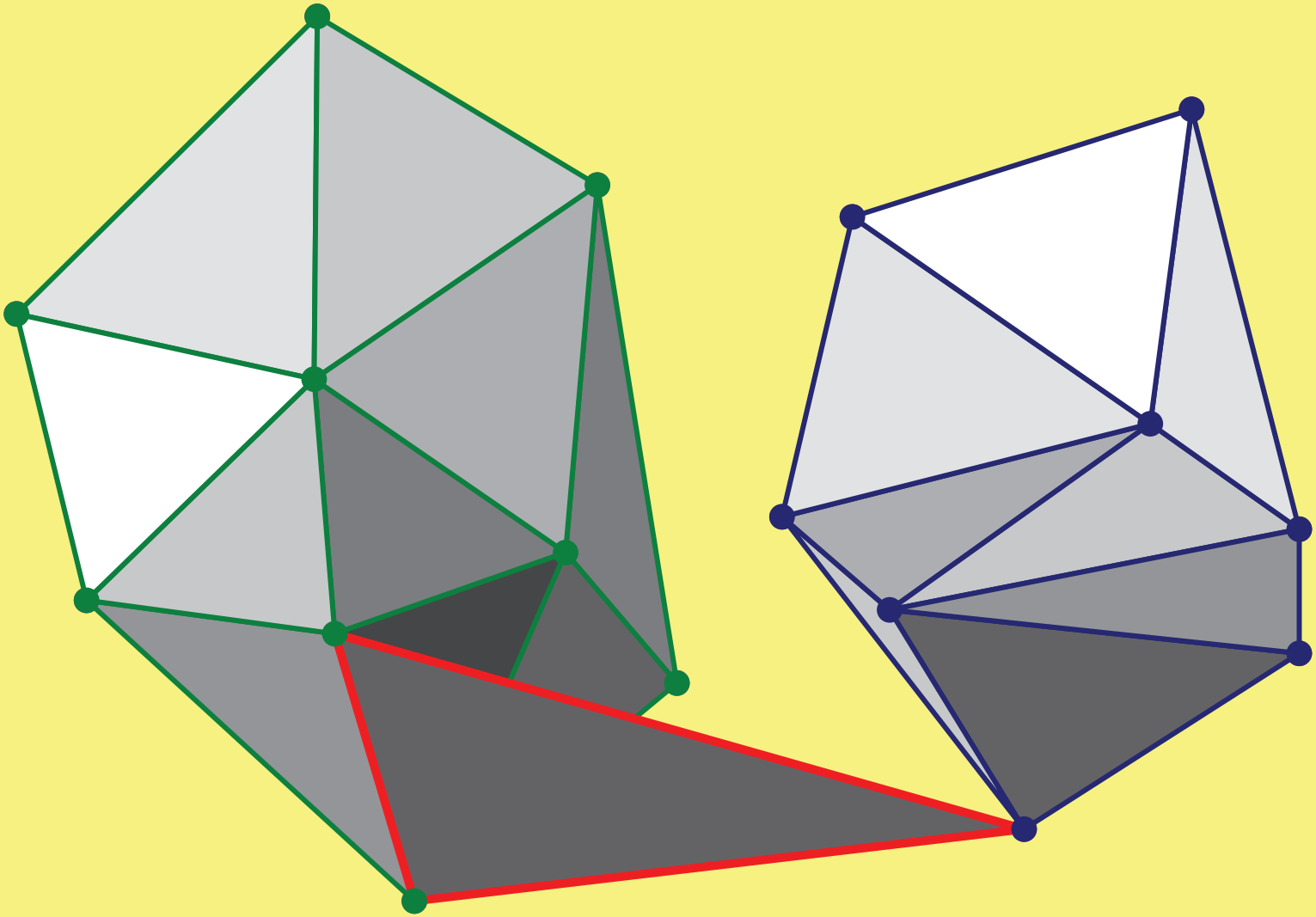
3D Divide & conquer algorithm

Use a wrapping algorithm around the new edges



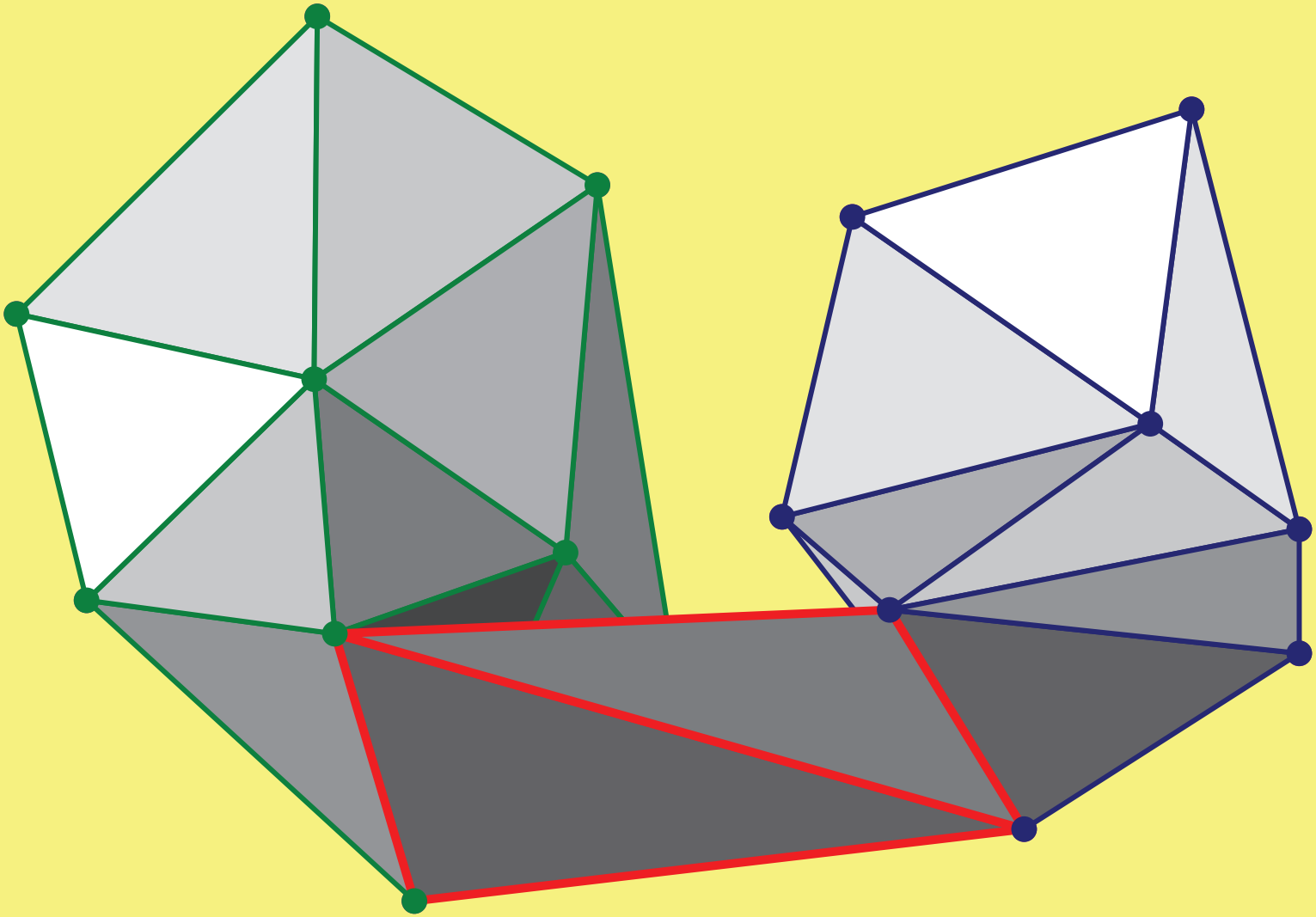
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



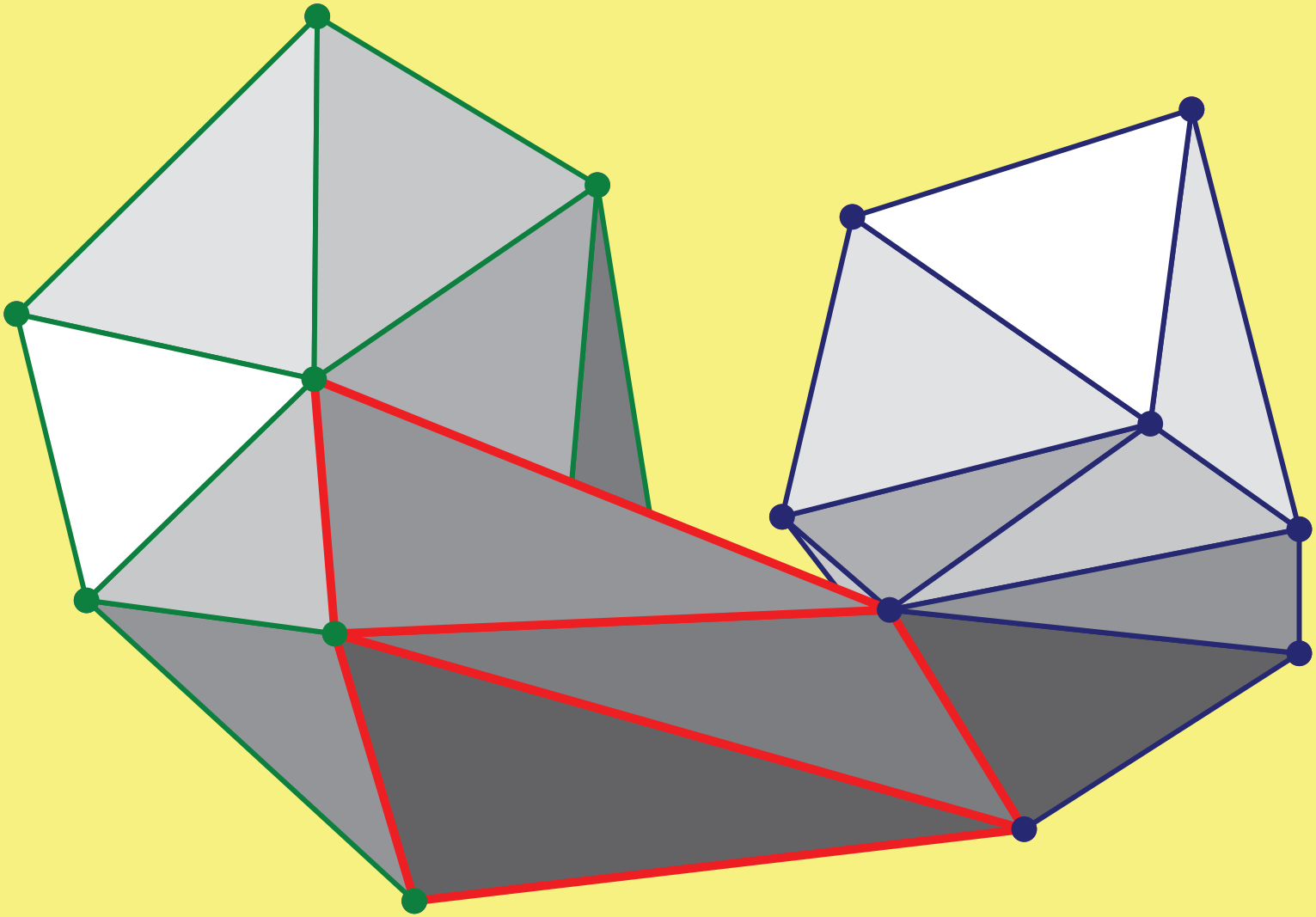
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



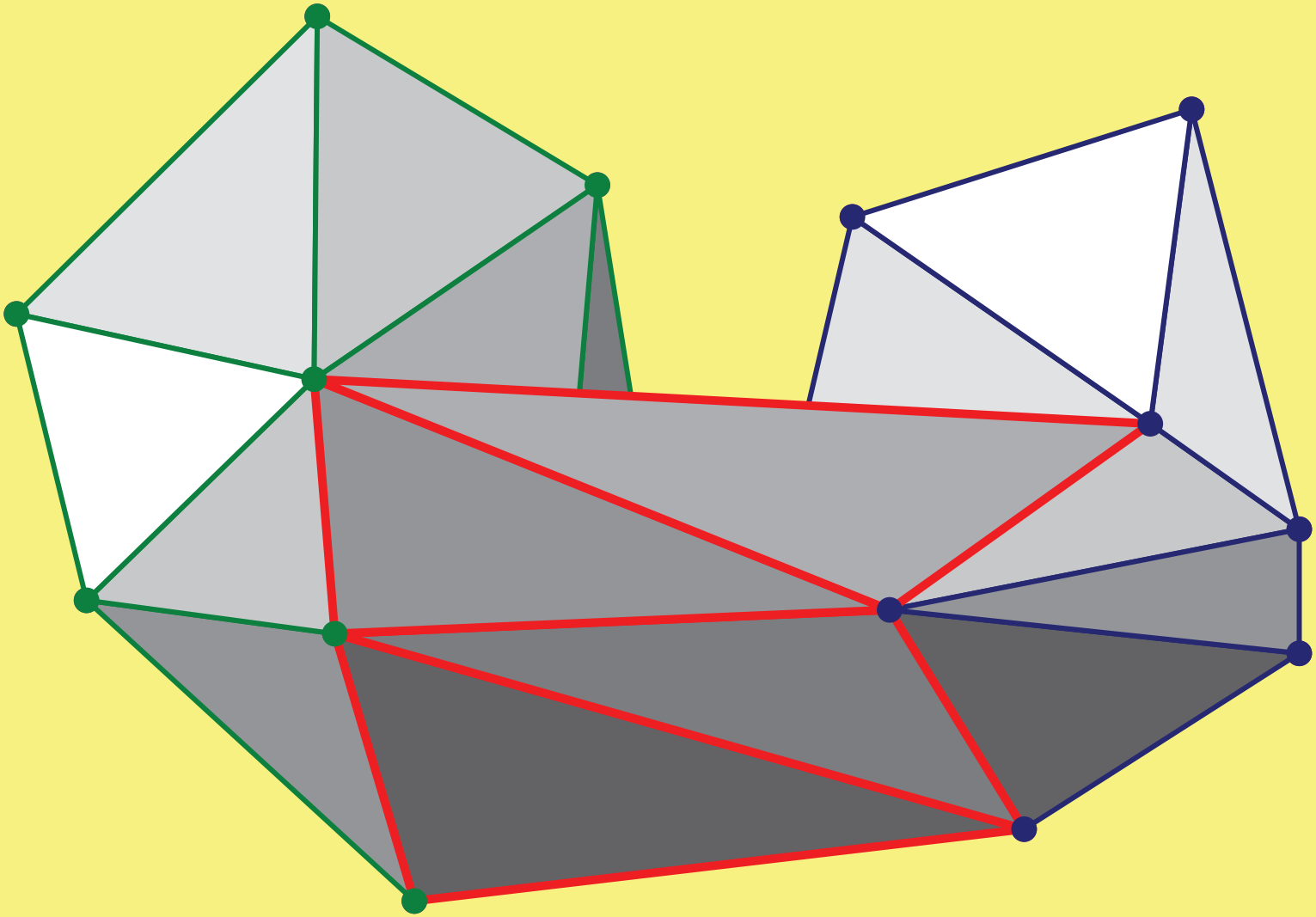
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



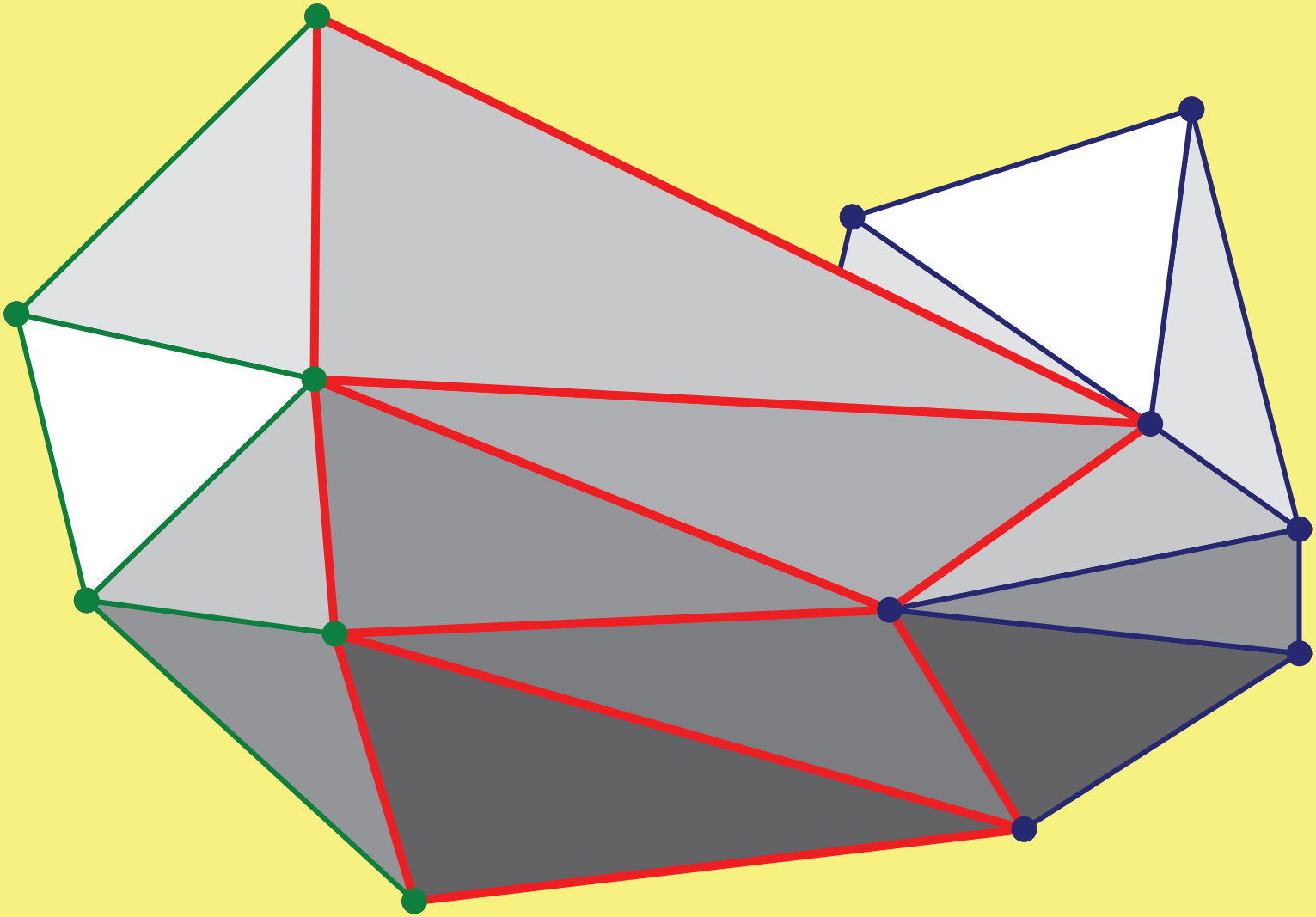
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



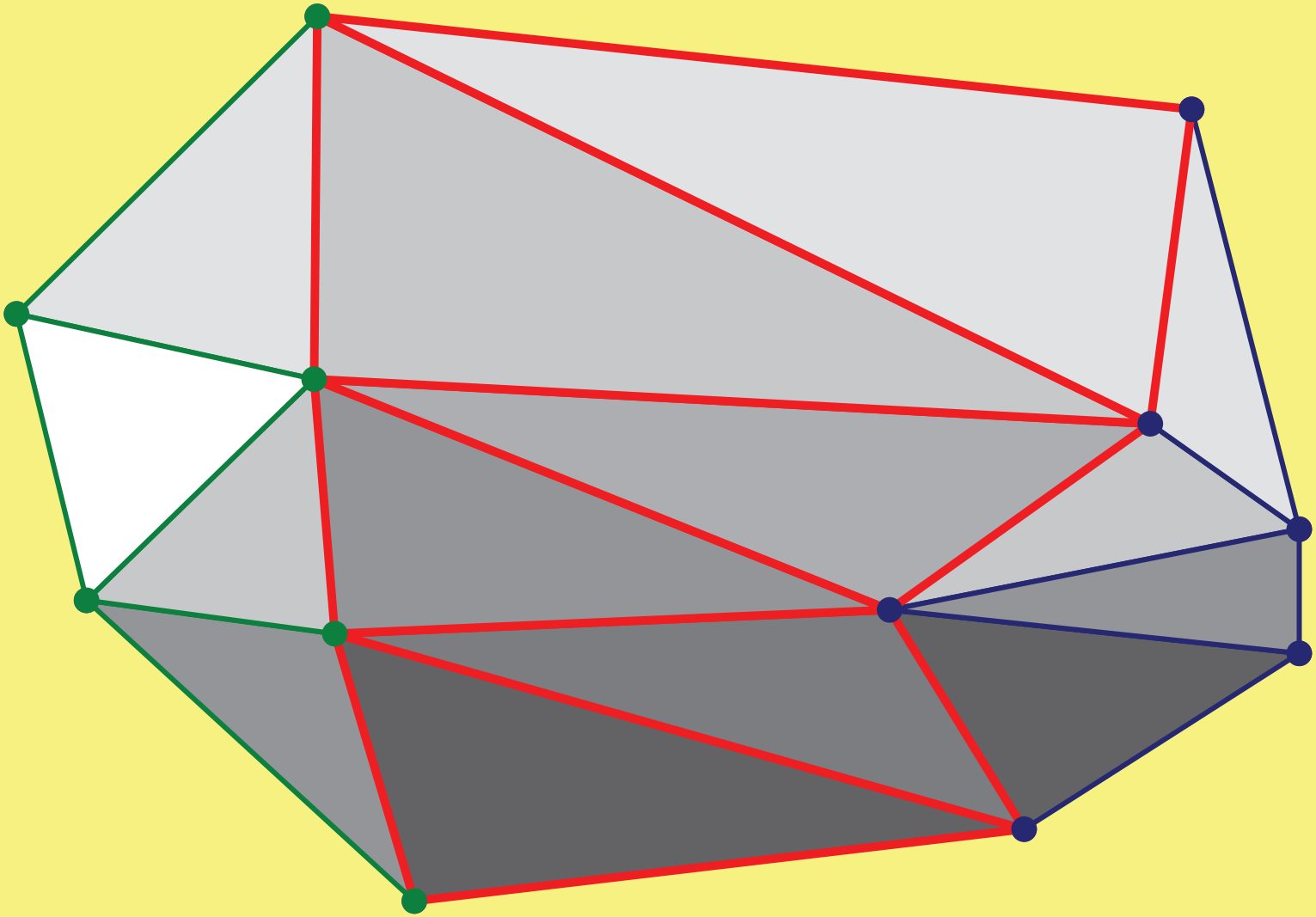
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



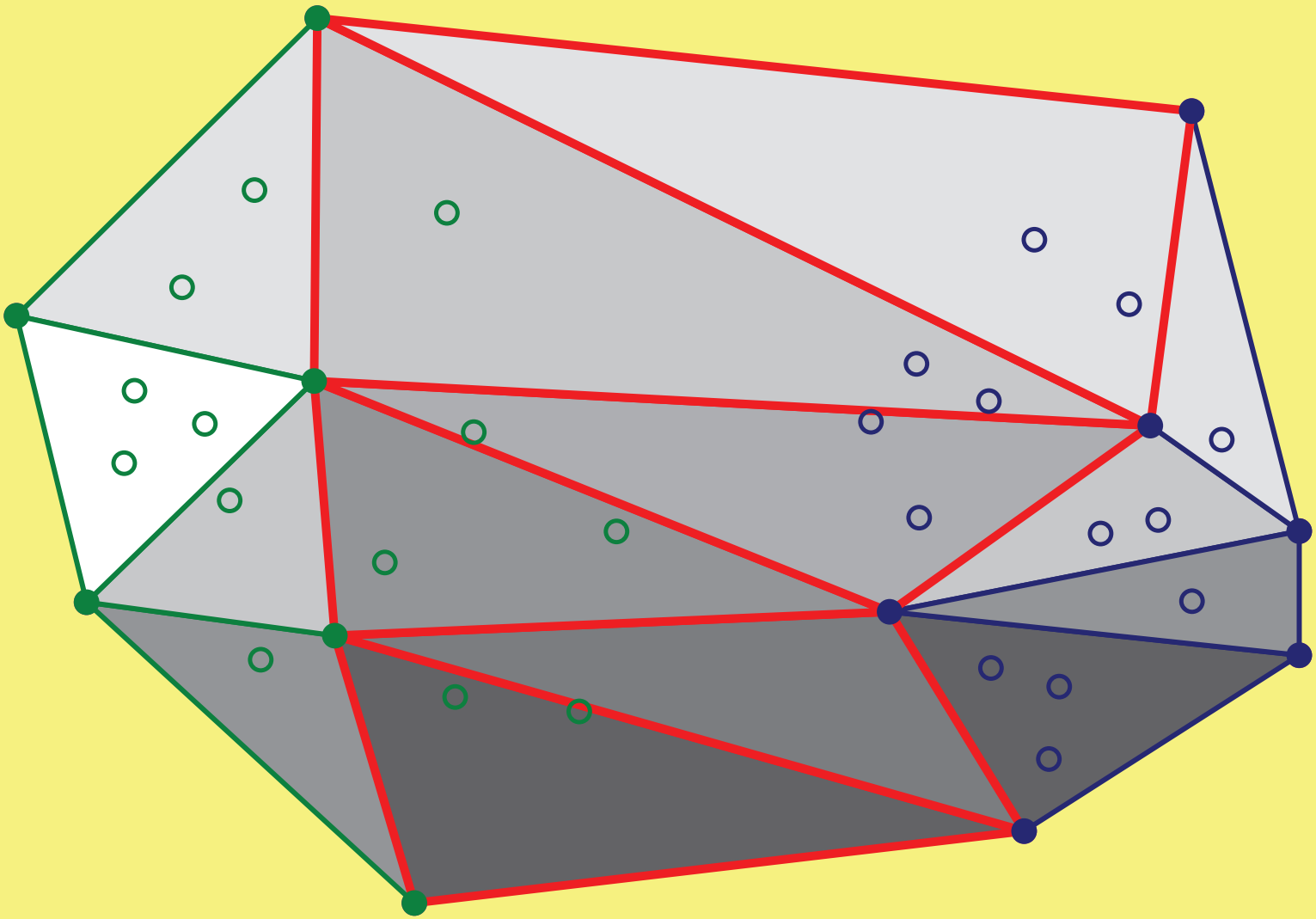
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



3D Divide & conquer algorithm

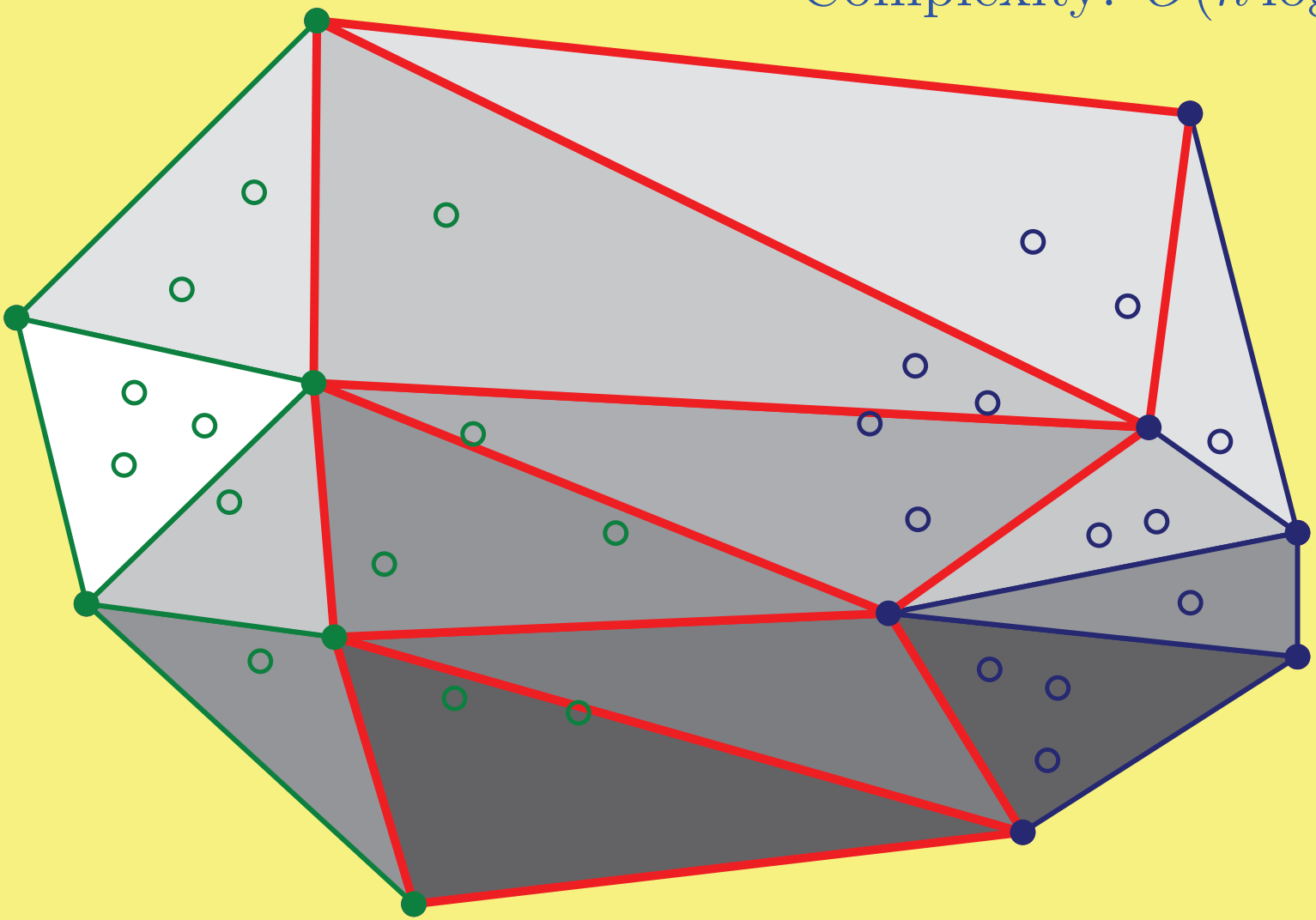
Search only in the star of the new edge vertices Merge in $O(n)$

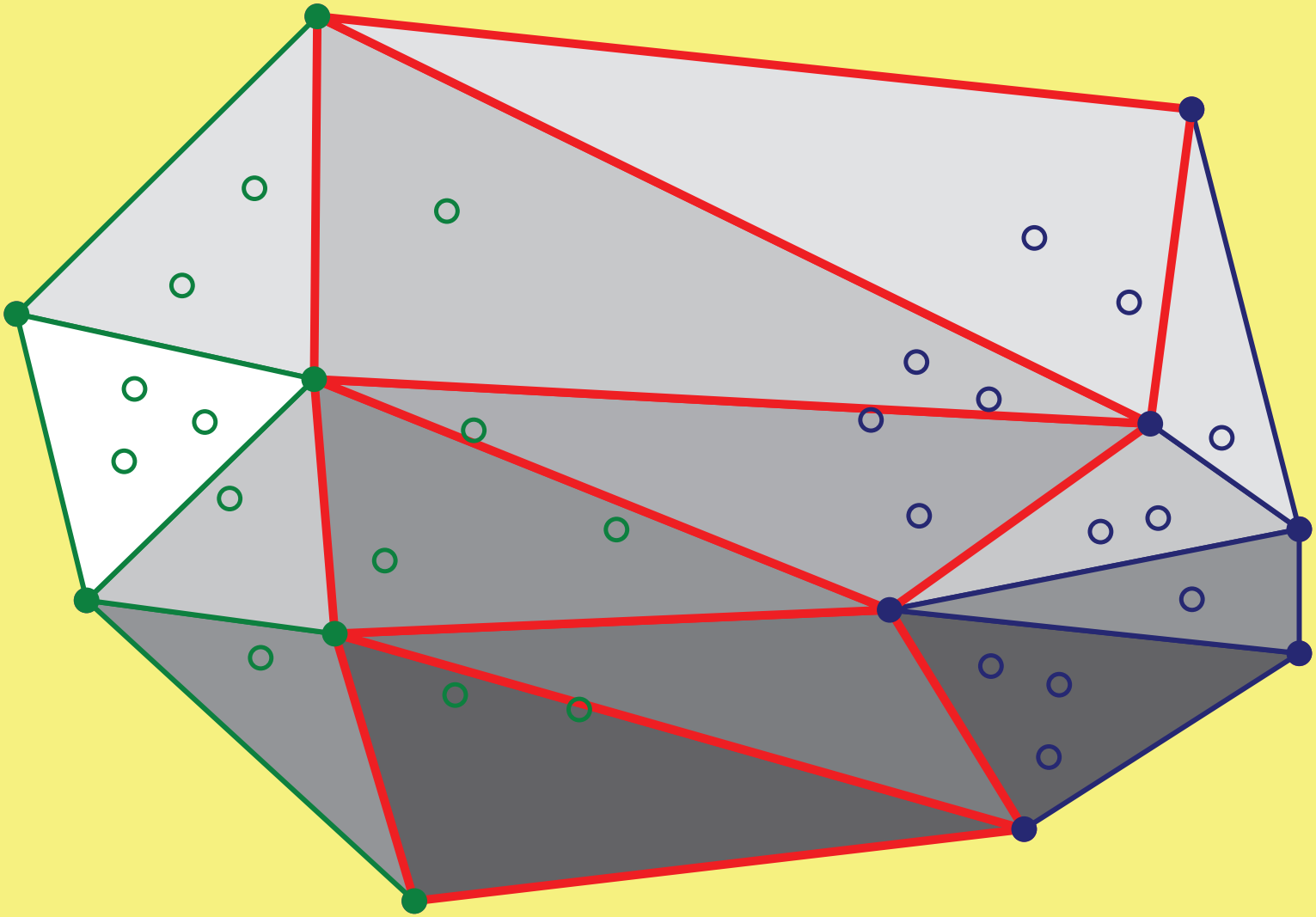


3D Divide & conquer algorithm

Search only in the star of the new edge vertices Merge in $O(n)$

Complexity: $O(n \log n)$

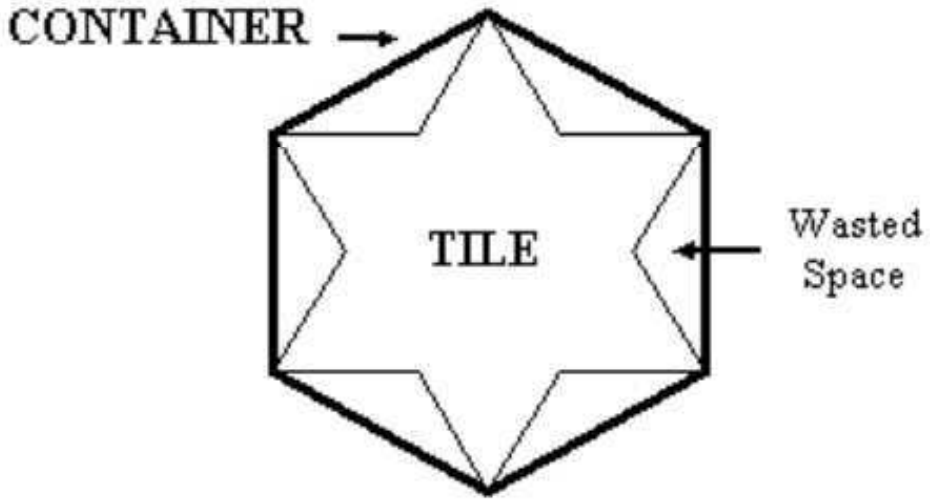




UVa Online Judge

10065 Useless Tile Packers

Yes, as you have apprehended the *Useless Tile Packers* (UTP) pack tiles. The tiles are of uniform thickness and have simple polygonal shape. For each tile a container is custom-built. The floor of the container is a convex polygon and under this constraint it has the minimum possible space inside to hold the tile it is built for. But this strategy leads to wasted space inside the container.



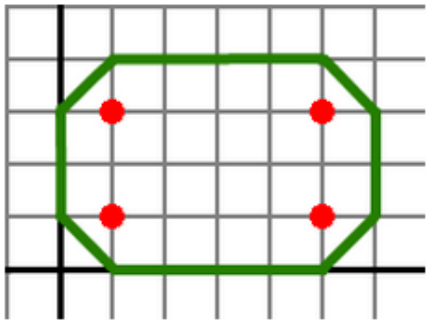
The UTP authorities are interested to know the percentage of wasted space for a given tile.

[http://uva.onlinejudge.org/external/100/10065.pdf]

C. Happy Farm 5

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

modeled by the turns. During every turn the shepherd can either stay where he stands or step in one of eight directions: horizontally, vertically, or diagonally. As the coordinates should always remain integer, then the length of a horizontal and vertical step is equal to 1, and the length of a diagonal step is equal to $\sqrt{2}$. The cows do not move. You have to minimize the number of moves the shepherd needs to run round the whole herd.



[http://codeforces.com/problemset/problem/50/C]

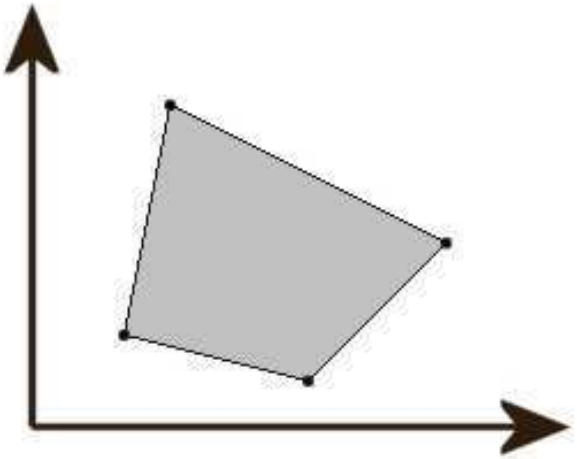
Problem Statement for ElectronicScarecrows

The black dots represents electronic scarecrows and the area shaded gray is the part of the field that is inaccessible to the birds.

This sounds great, but there are two drawbacks. First, the scarecrows are of course very expensive, so a farmer can't afford very many of them. Second, they are quite heavy and need firm soil to stand on, and must also be in range of a power outlet. This severely limits the number of locations the farmer can place such scarecrows.

Given the coordinates of possible locations for the scarecrows and the maximum number of scarecrows the farmer can afford to buy, calculate the largest area that can be guarded by these scarecrows. The farmer's field is a rectangular area, and all locations given will be inside this area.

- **x** will contain between 3 and 40 elements, inclusive.
- **y** will contain between 3 and 40 elements, inclusive.
- **x** will contain the same number of elements as **y**.
- Each element in **x** will be between 0 and 1000, inclusive.
- Each element in **y** will be between 0 and 1000, inclusive.
- No location will appear more than once.
- **n** will be between 3 and 40, inclusive.



[http://community.topcoder.com/stat?c=problem_statement&pm=1960&rd=4670]

UVa Online Judge

10089 Repackaging

Coffee cups of three different sizes (identified as size 1, size 2 and size 3 cups) are produced in factories under *ACM* (Association of Cup Makers) and are sold in various packages. Each type of package is identified by three positive integers (S_1, S_2, S_3) where S_i $(1 \leq i \leq 3)$ denotes the number of size i cups included in the package. There is no package with $S_1 = S_2 = S_3$.

But recently it has been discovered that there is a great demand for packages containing equal number cups of all three sizes. So, as an emergency step to meet the demand *ACM* has decided to unpack the cups from some of the packages stored in its (unlimited) stock of unsold products and repack them in packages having equal number of cups of all three sizes. For example, suppose *ACM* has the following four types of packages in its stock: (1, 2, 3), (1, 11, 5), (9, 4, 3) and (2, 3, 2). So, one can unpack three (1, 2, 3) packages, one (9, 4, 3) package and two (2, 3, 2) packages and repack the cups to produce sixteen (1, 1, 1) packages. One can even produce eight (2, 2, 2) packages or four (4, 4, 4) packages or two (8, 8, 8) packages or one (16, 16, 16) package or even different combination of packages each containing equal number of size 1, size 2 and size 3 cups. Note that all the unpacked cups are used to produce the new packages, i.e., no unpacked cup is wasted.

ACM has hired you to write a program that will decide whether it is possible to produce packages containing equal number of all three types of cups using all the cups that can be found by unpacking any combination of existing packages in the stock.

[http://uva.onlinejudge.org/external/100/10089.pdf]

Sources

http://www.cs.uu.nl/docs/vakken/ga/slides1.pdf

http://www.loria.fr/~pougetma/enseignement/webimpa/2-3D-enveloppe-convexe-od.pdf

http://www3.jouy.inra.fr/miaj/public/vigneron/talks/diam.pdf