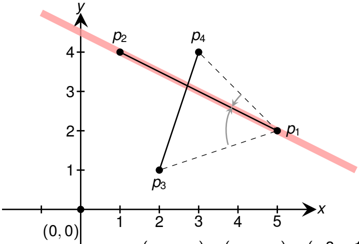


$$(p_3 - p_1) \times (p_2 - p_1) = (-3, -1) \times (-4, 2) = -10$$

$$(p_4 - p_1) \times (p_2 - p_1) = (-2, 2) \times (-4, 2) = 4$$

7: Geometric Algorithms

Frank Stajano Thomas Sauerwald

Lent 2016

UNIVERSITY OF CAMBRIDGE

Outline

Introduction and Line Intersection

Convex Hull

Glimpse at (More) Advanced Algorithms

Introduction

Computational Geometry

- Branch that studies algorithms for geometric problems

Introduction

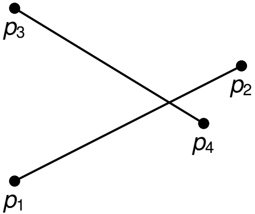
Computational Geometry

- Branch that studies algorithms for geometric problems
- typically, input is a set of points, line segments etc.

Introduction

Computational Geometry

- Branch that studies algorithms for geometric problems
- typically, input is a set of points, line segments etc.



Do these lines intersect?

Introduction

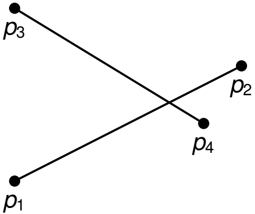
Computational Geometry

- Branch that studies algorithms for geometric problems
- typically, input is a set of points, line segments etc.

Applications

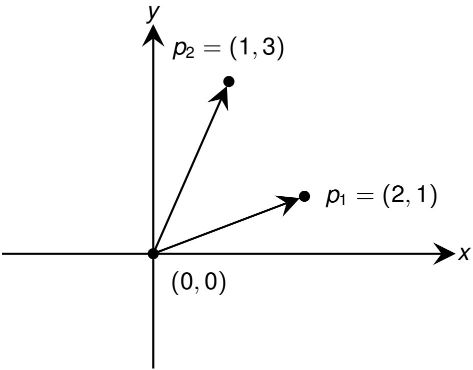
- computer graphics
- computer vision
- textile layout
- VLSI design

$\vdots$

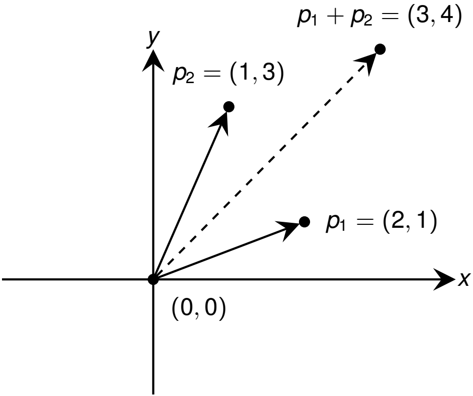


Do these lines intersect?

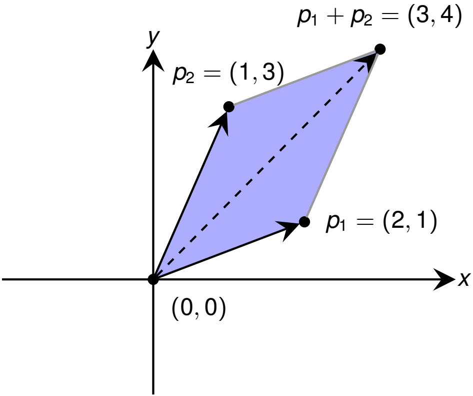
Cross Product (Area)



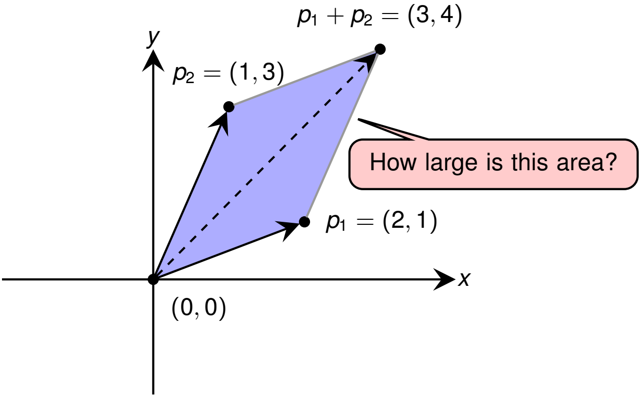
Cross Product (Area)



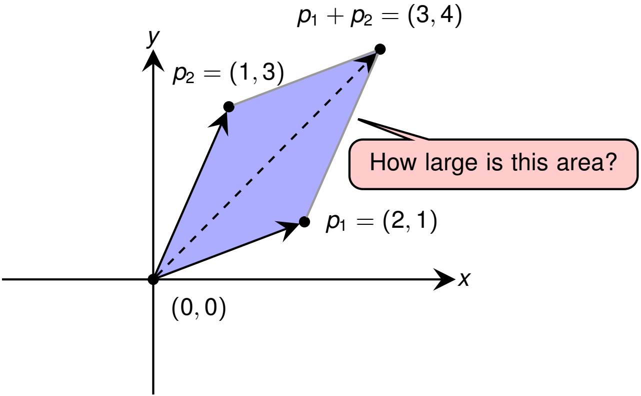
Cross Product (Area)



Cross Product (Area)

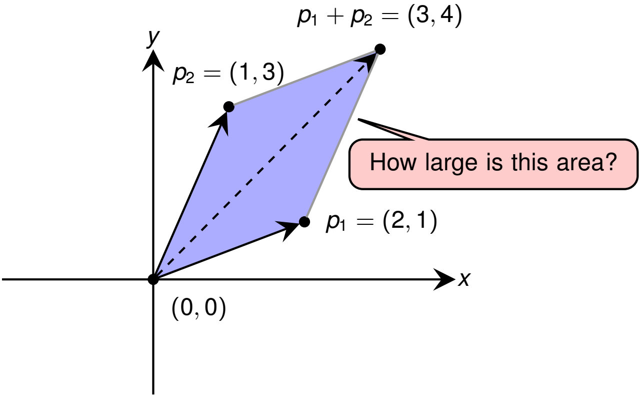


Cross Product (Area)



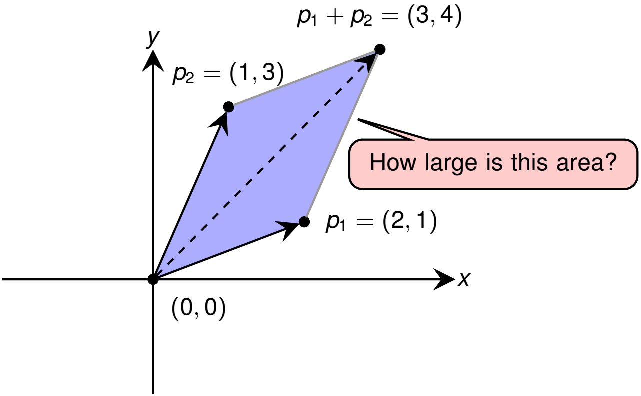
$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix}$$

Cross Product (Area)



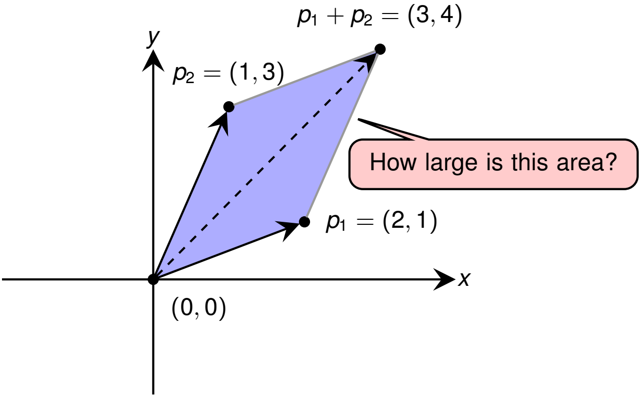
$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1$$

Cross Product (Area)



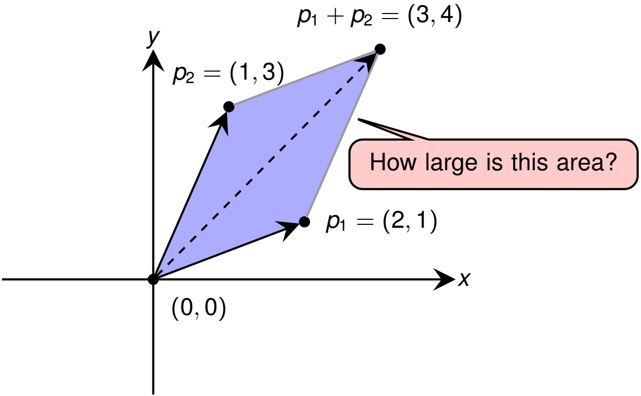
$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = 2 \cdot 3 - 1 \cdot 1$$

Cross Product (Area)



$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$$

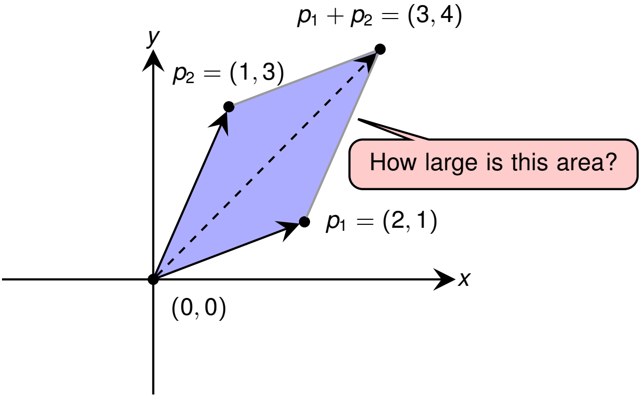
Cross Product (Area)



$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$$

$p_2 \times p_1$

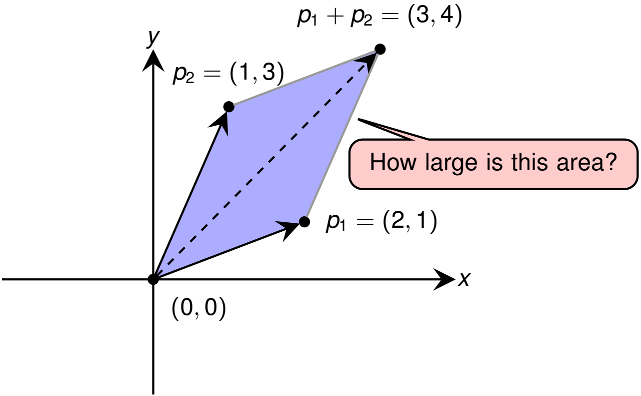
Cross Product (Area)



$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$$

$$p_2 \times p_1 = y_1 x_2 - y_2 x_1$$

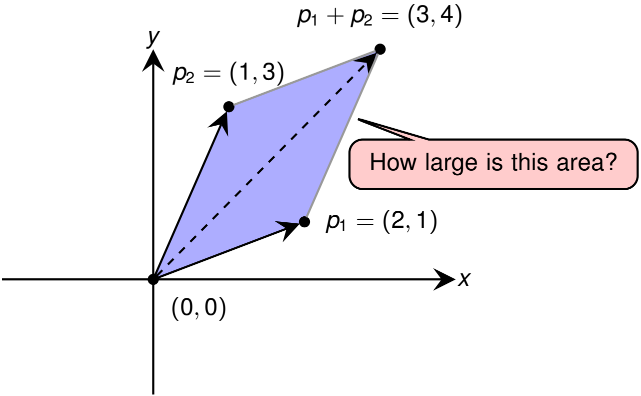
Cross Product (Area)



$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$$

$$p_2 \times p_1 = y_1 x_2 - y_2 x_1 = -(p_1 \times p_2)$$

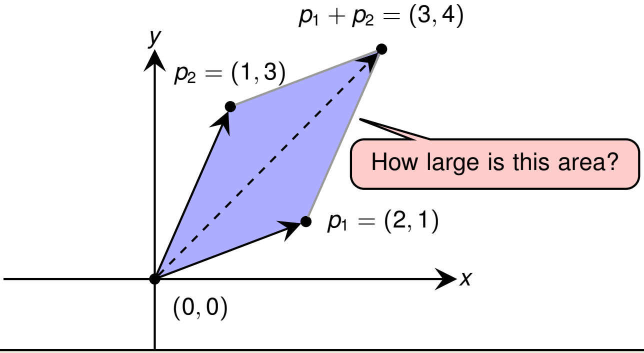
Cross Product (Area)



$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$$

$$p_2 \times p_1 = y_1 x_2 - y_2 x_1 = -(p_1 \times p_2) = -5$$

Cross Product (Area)

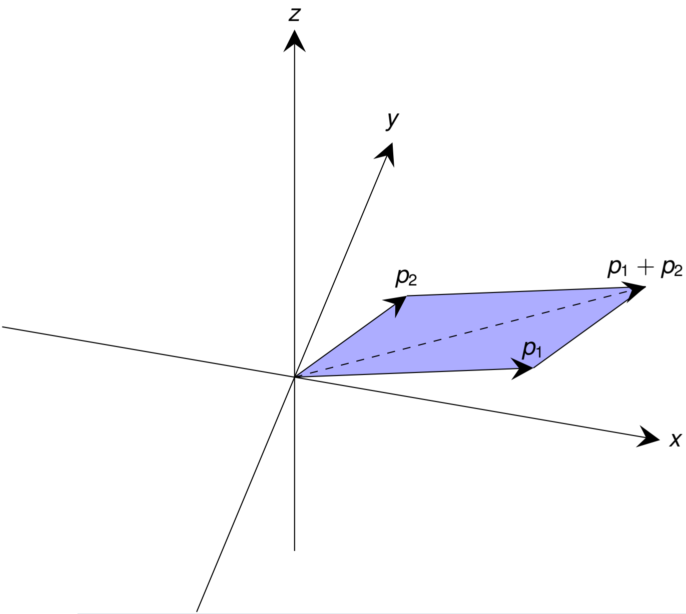


Alternatively, one could take the dot-product (but not used here):
$$p_1 \cdot p_2 = \|p_1\| \cdot \|p_2\| \cdot \cos(\phi).$$

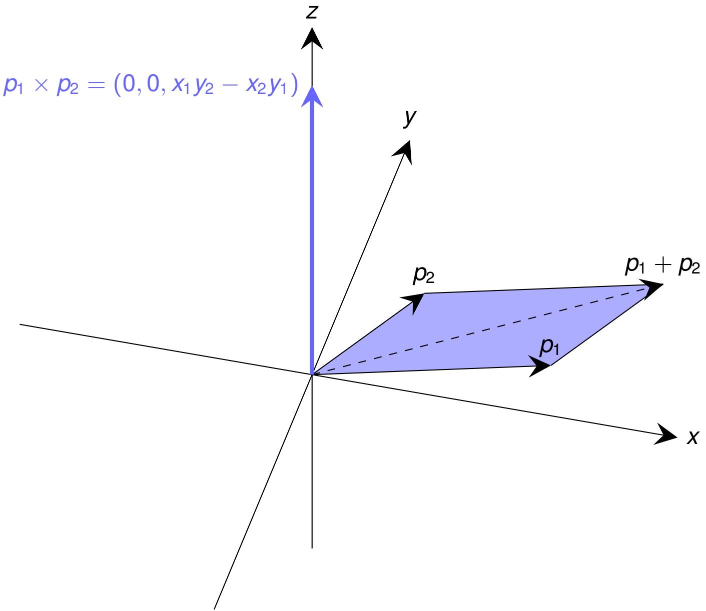
$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$$

$$p_2 \times p_1 = y_1 x_2 - y_2 x_1 = -(p_1 \times p_2) = -5$$

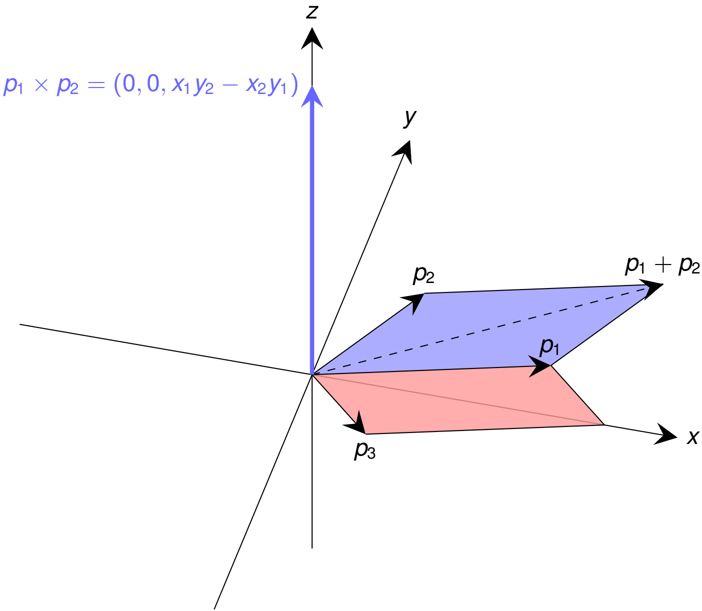
Cross Product in 3D



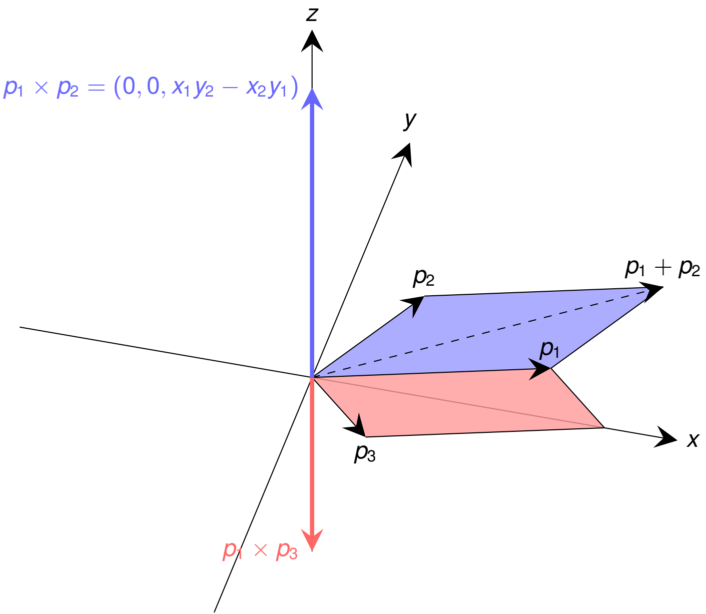
Cross Product in 3D



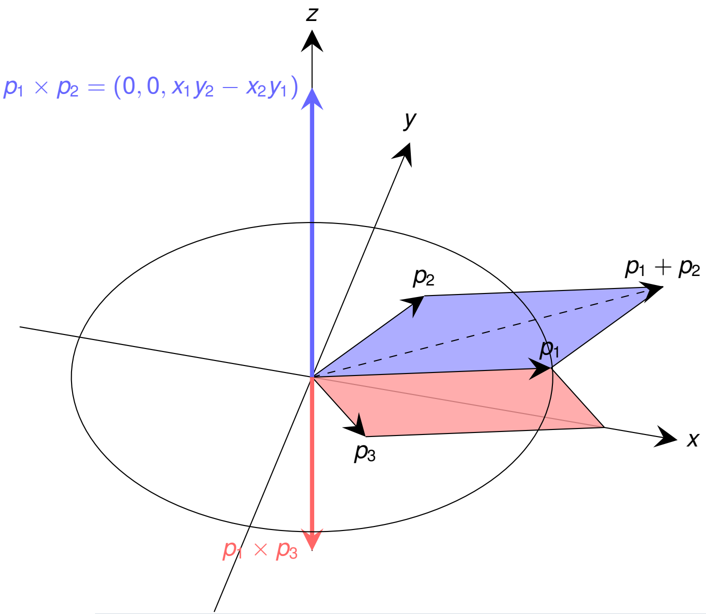
Cross Product in 3D



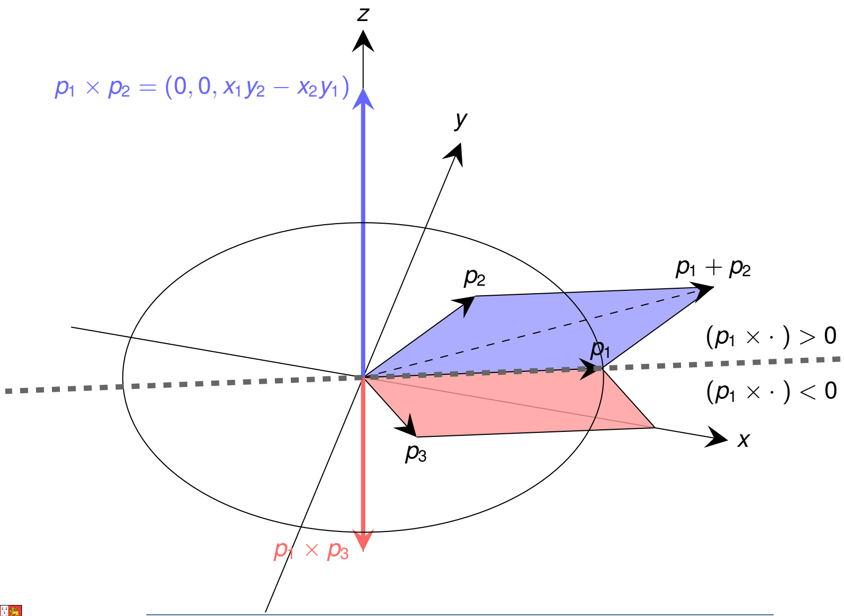
Cross Product in 3D



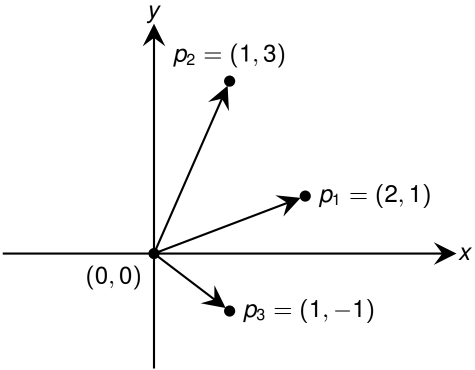
Cross Product in 3D



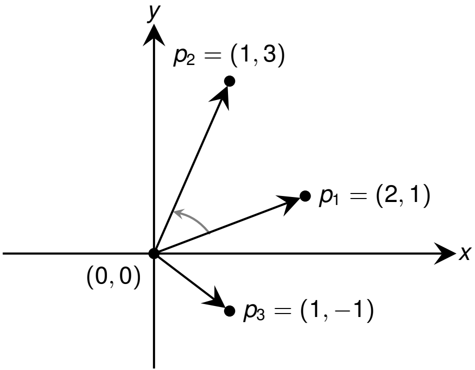
Cross Product in 3D



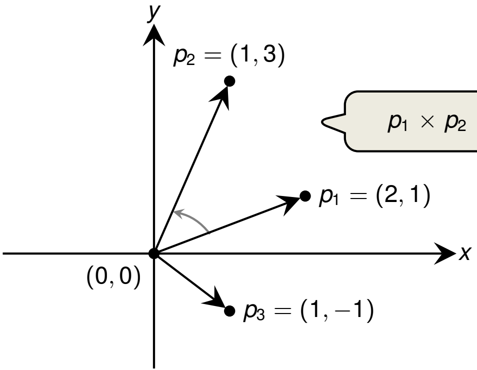
Using Cross product to determine Turns



Using Cross product to determine Turns

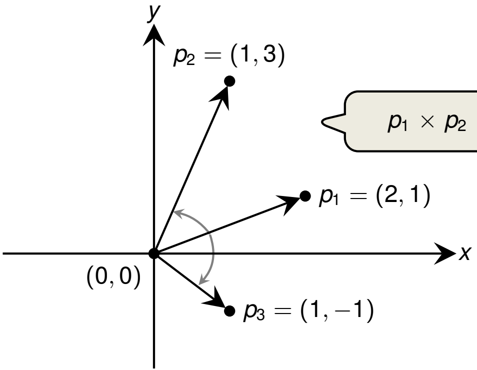


Using Cross product to determine Turns



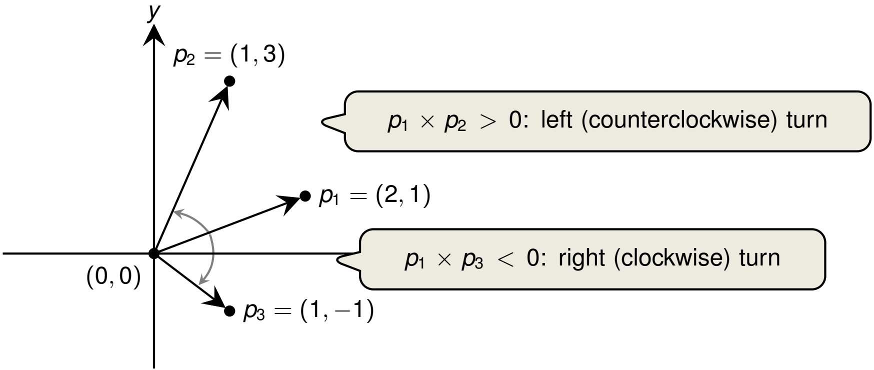
$p_1 \times p_2 > 0$: left (counterclockwise) turn

Using Cross product to determine Turns

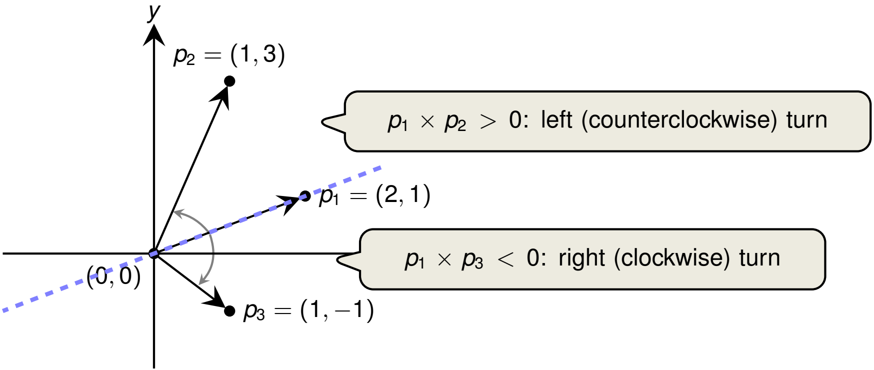


$p_1 \times p_2 > 0$: left (counterclockwise) turn

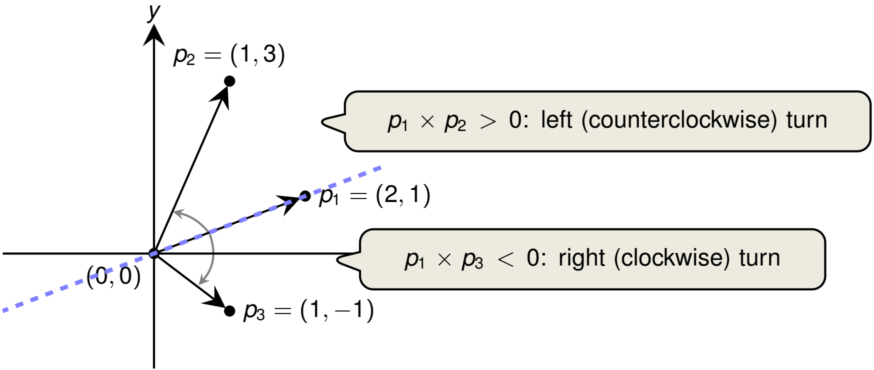
Using Cross product to determine Turns



Using Cross product to determine Turns

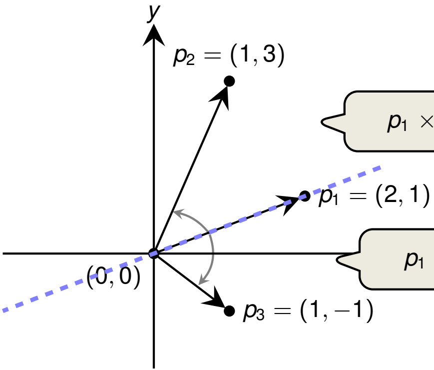


Using Cross product to determine Turns



Sign of cross product determines turn!

Using Cross product to determine Turns



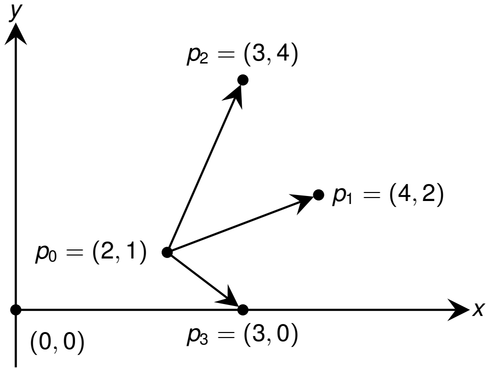
$p_1 \times p_2 > 0$: left (counterclockwise) turn

$p_1 \times p_3 < 0$: right (clockwise) turn

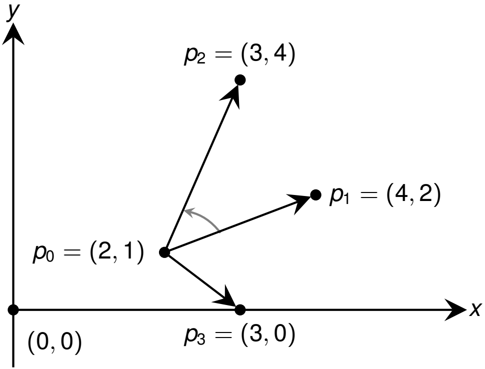
Sign of cross product determines turn!

Cross product equals zero iff vectors are colinear

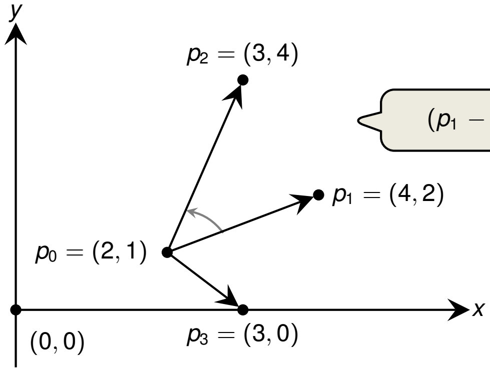
Using Cross product to determine Turns (origin shifted)



Using Cross product to determine Turns (origin shifted)

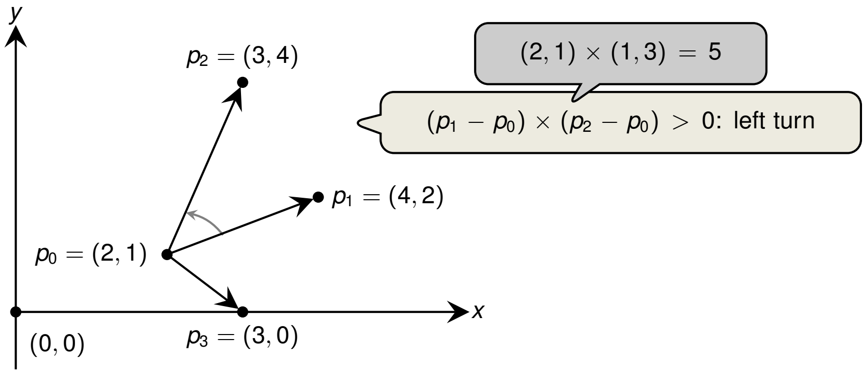


Using Cross product to determine Turns (origin shifted)

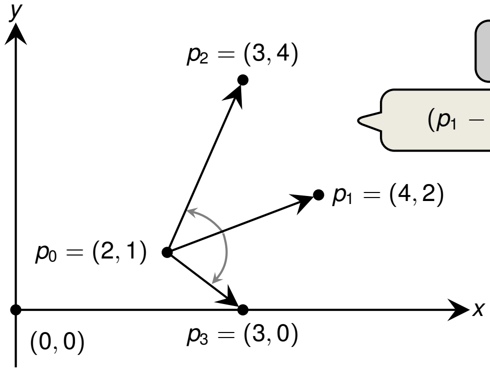


$(p_1 - p_0) \times (p_2 - p_0) > 0$: left turn

Using Cross product to determine Turns (origin shifted)



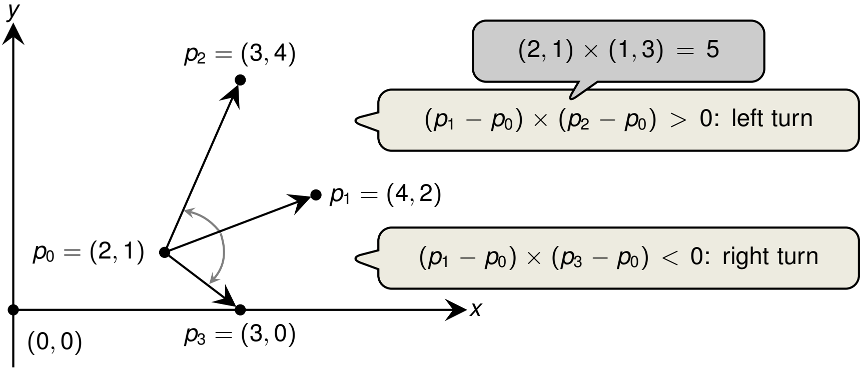
Using Cross product to determine Turns (origin shifted)



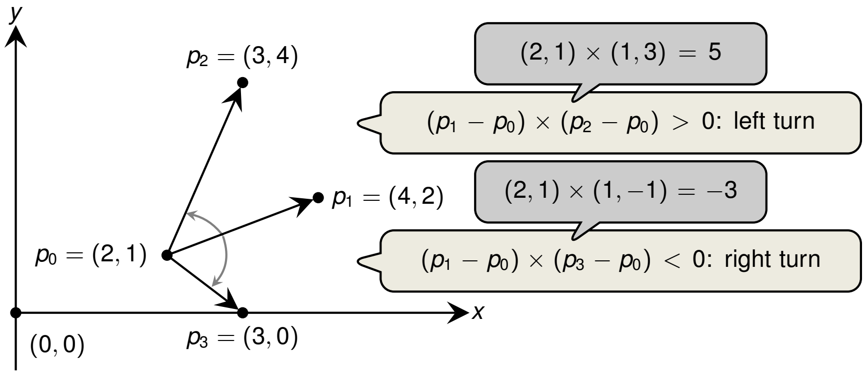
$(2,1) \times (1,3) = 5$

$(p_1 - p_0) \times (p_2 - p_0) > 0$: left turn

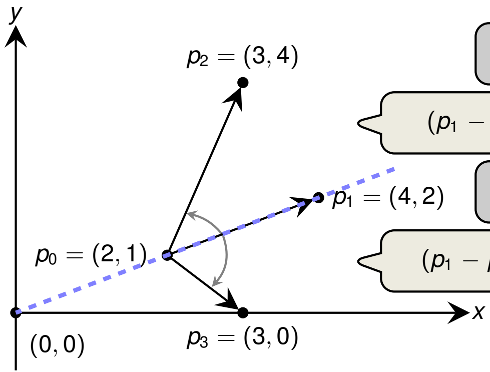
Using Cross product to determine Turns (origin shifted)



Using Cross product to determine Turns (origin shifted)



Using Cross product to determine Turns (origin shifted)



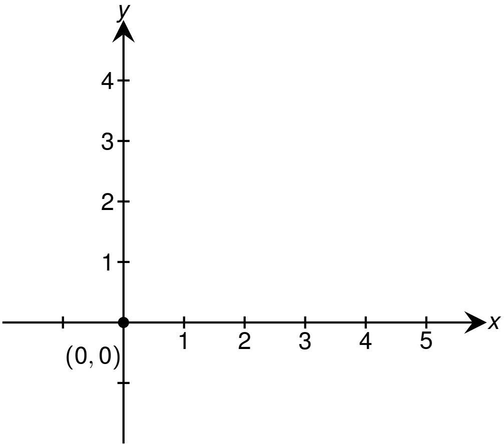
$(2,1) \times (1,3) = 5$

$(p_1 - p_0) \times (p_2 - p_0) > 0$: left turn

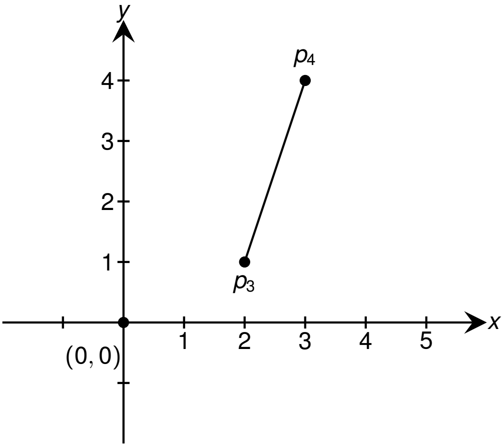
$(2,1) \times (1,-1) = -3$

$(p_1 - p_0) \times (p_3 - p_0) < 0$: right turn

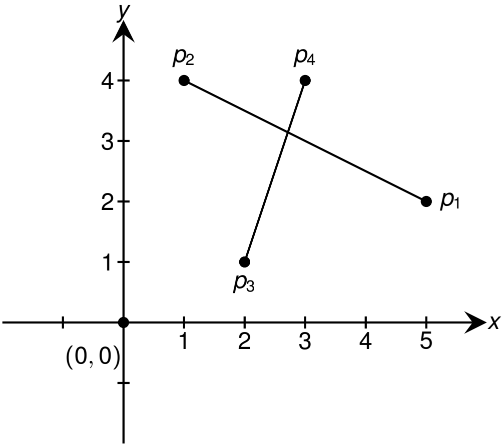
Solving Line Intersection



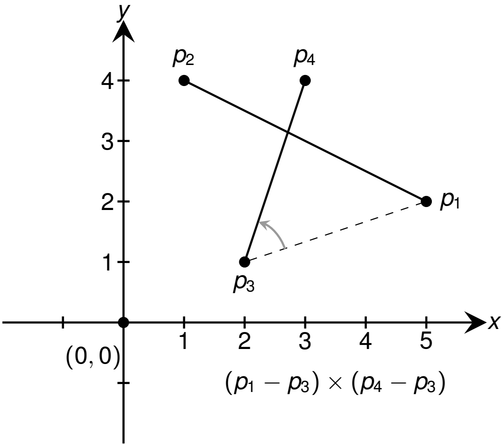
Solving Line Intersection



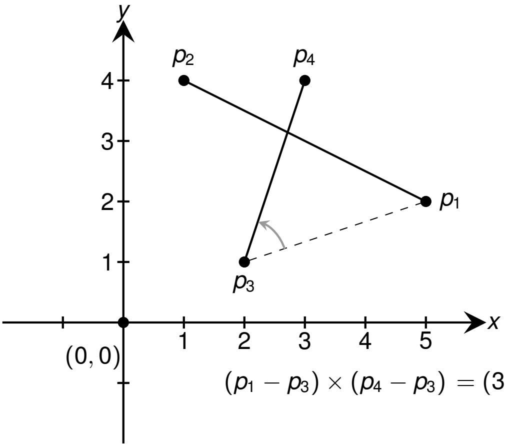
Solving Line Intersection



Solving Line Intersection

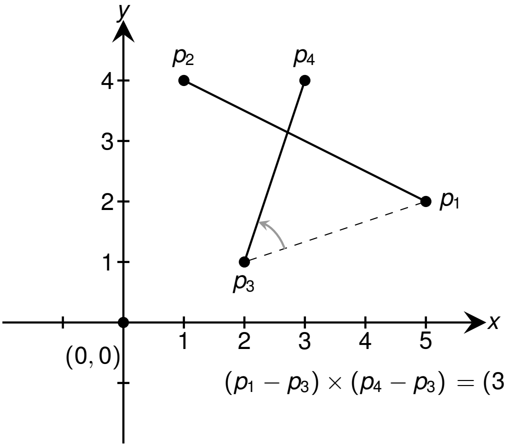


Solving Line Intersection



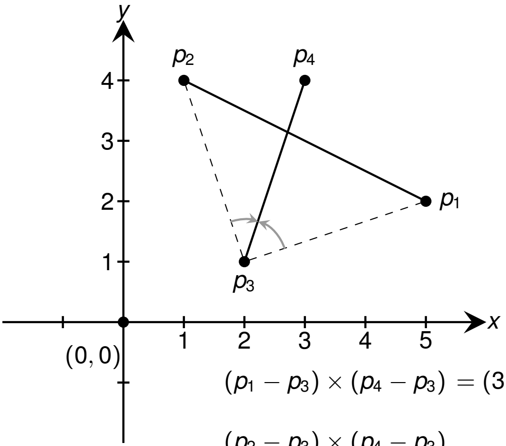
$$(p_1 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3)$$

Solving Line Intersection



$$(p_1 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3) = 8$$

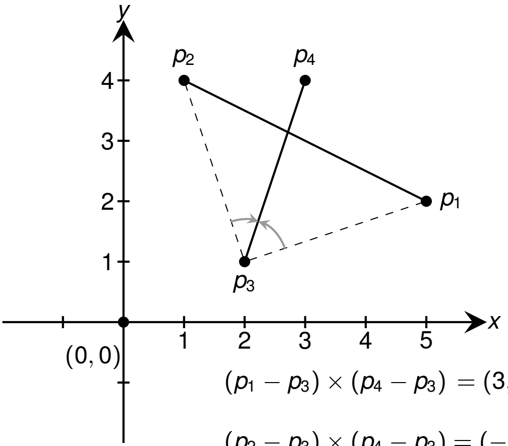
Solving Line Intersection



$$(p_1 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3) = 8$$

$$(p_2 - p_3) \times (p_4 - p_3)$$

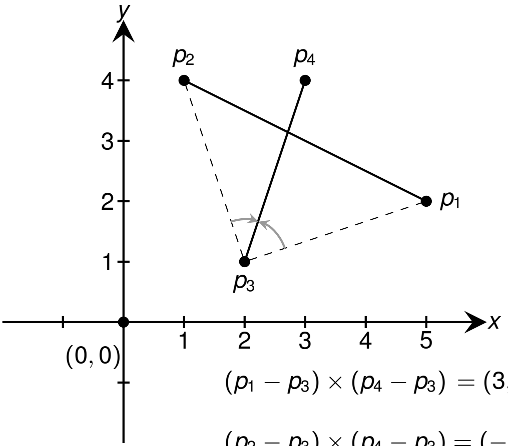
Solving Line Intersection



$$(p_1 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3) = 8$$

$$(p_2 - p_3) \times (p_4 - p_3) = (-1, 3) \times (1, 3)$$

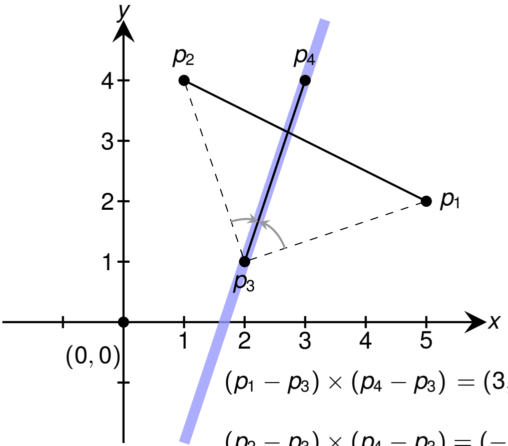
Solving Line Intersection



$$(p_1 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3) = 8$$

$$(p_2 - p_3) \times (p_4 - p_3) = (-1, 3) \times (1, 3) = -6$$

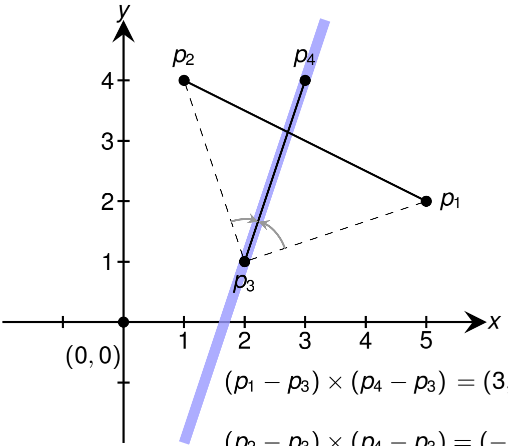
Solving Line Intersection



$$(p_1 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3) = 8$$

$$(p_2 - p_3) \times (p_4 - p_3) = (-1, 3) \times (1, 3) = -6$$

Solving Line Intersection

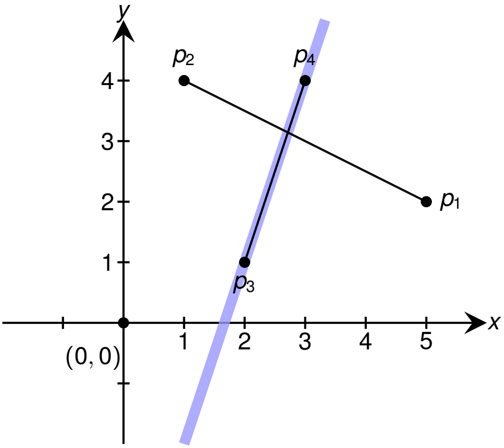


$$(p_1 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3) = 8$$

$$(p_2 - p_3) \times (p_4 - p_3) = (-1, 3) \times (1, 3) = -6$$

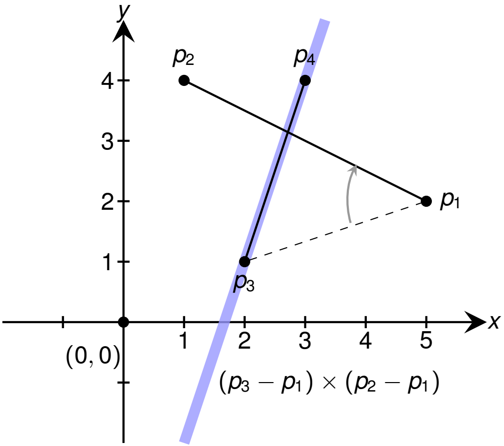
Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Solving Line Intersection



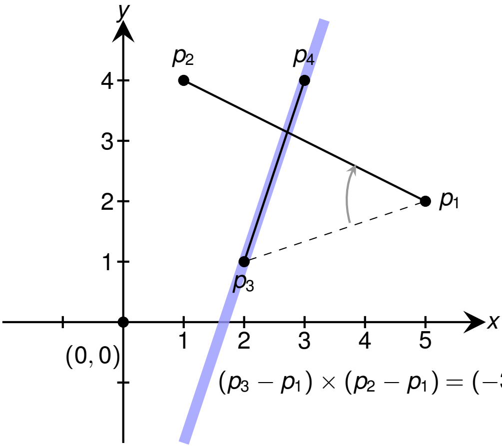
Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Solving Line Intersection



Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

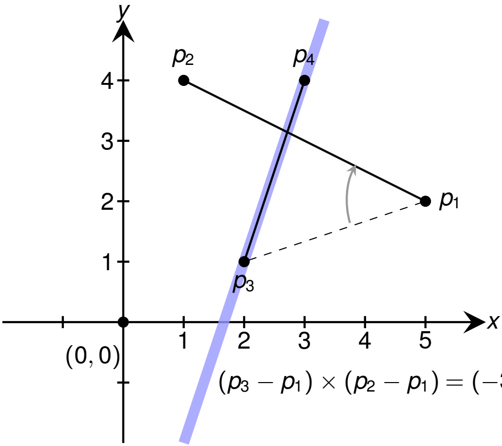
Solving Line Intersection



$$(p_3 - p_1) \times (p_2 - p_1) = (-3, -1) \times (-4, 2)$$

Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

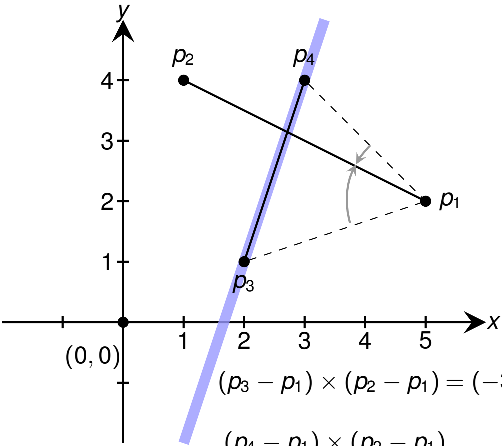
Solving Line Intersection



$(p_3 - p_1) \times (p_2 - p_1) = (-3,-1) \times (-4,2) = -10$

Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Solving Line Intersection

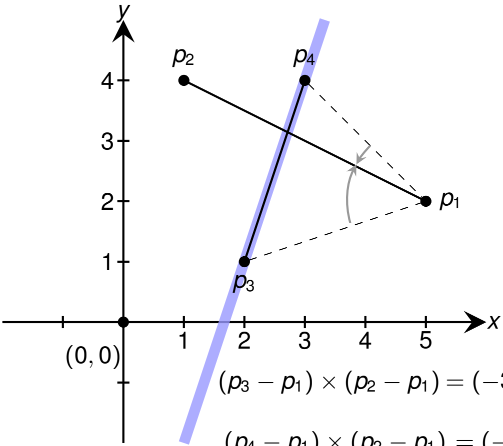


$$(p_3 - p_1) \times (p_2 - p_1) = (-3, -1) \times (-4, 2) = -10$$

$$(p_4 - p_1) \times (p_2 - p_1)$$

Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Solving Line Intersection

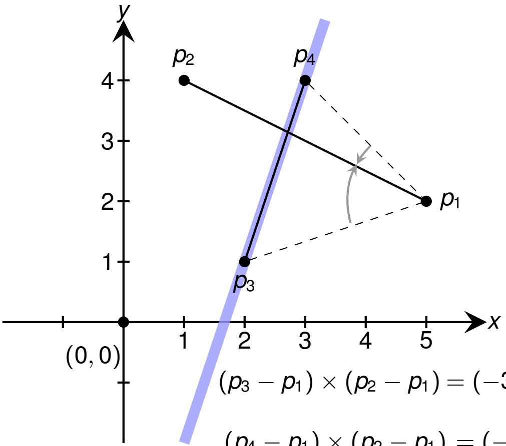


$$(p_3 - p_1) \times (p_2 - p_1) = (-3,-1) \times (-4,2) = -10$$

$$(p_4 - p_1) \times (p_2 - p_1) = (-2,2) \times (-4,2)$$

Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Solving Line Intersection

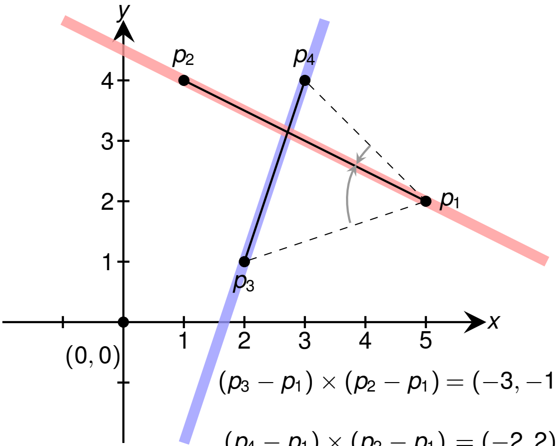


$$(p_3 - p_1) \times (p_2 - p_1) = (-3, -1) \times (-4, 2) = -10$$

$$(p_4 - p_1) \times (p_2 - p_1) = (-2, 2) \times (-4, 2) = 4$$

Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Solving Line Intersection



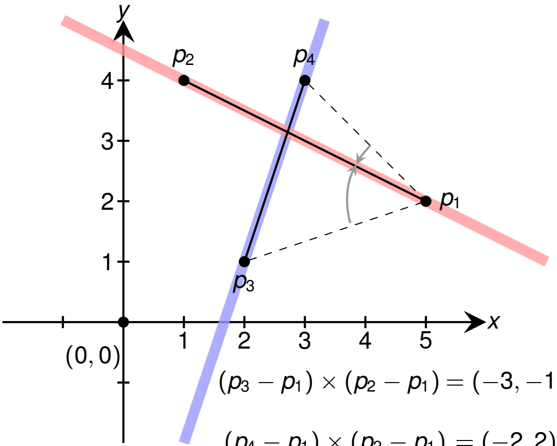
$$(p_3 - p_1) \times (p_2 - p_1) = (-3, -1) \times (-4, 2) = -10$$

$$(p_4 - p_1) \times (p_2 - p_1) = (-2, 2) \times (-4, 2) = 4$$

Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Opposite signs $\Rightarrow \overline{p_3 p_4}$ crosses (infinite) line through p_1 and p_2

Solving Line Intersection



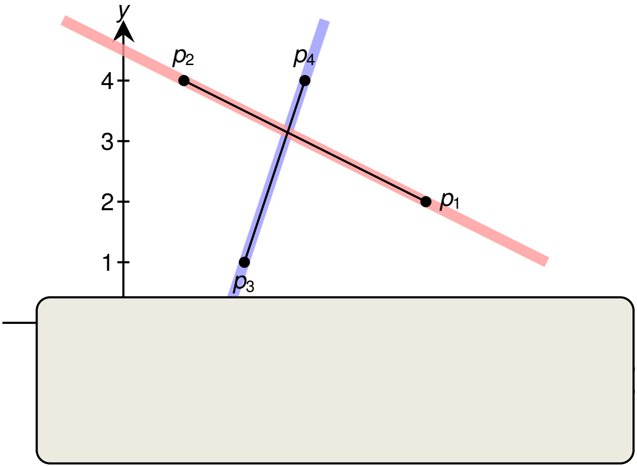
$$(p_3 - p_1) \times (p_2 - p_1) = (-3, -1) \times (-4, 2) = -10$$

$$(p_4 - p_1) \times (p_2 - p_1) = (-2, 2) \times (-4, 2) = 4$$

Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Opposite signs $\Rightarrow \overline{p_3 p_4}$ crosses (infinite) line through p_1 and p_2

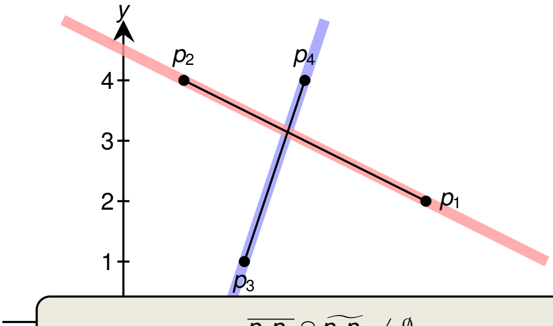
Solving Line Intersection



Opposite signs ⇒ $\overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Opposite signs ⇒ $\overline{p_3 p_4}$ crosses (infinite) line through p_1 and p_2

Solving Line Intersection

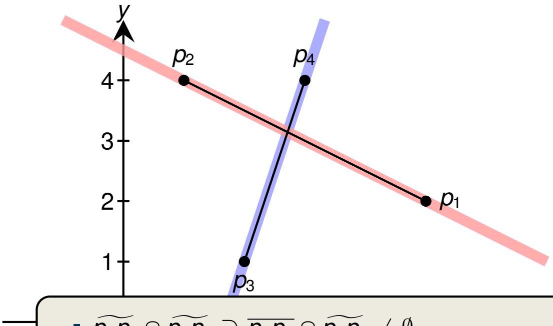


$$\overline{p_1 p_2} \cap \widetilde{p_3 p_4} \neq \emptyset$$
$$\widetilde{p_1 p_2} \cap \overline{p_3 p_4} \neq \emptyset$$

Opposite signs $\Rightarrow$ $\overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Opposite signs $\Rightarrow$ $\overline{p_3 p_4}$ crosses (infinite) line through p_1 and p_2

Solving Line Intersection

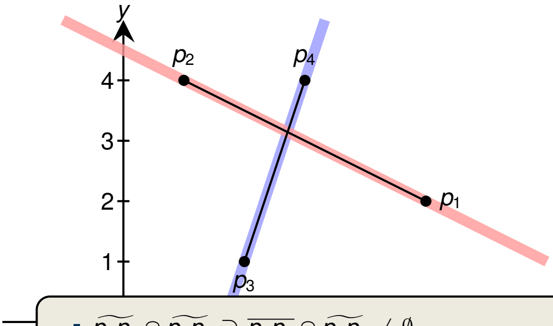


- $\widetilde{p_1 p_2} \cap \widetilde{p_3 p_4} \supseteq \overline{p_1 p_2} \cap \widetilde{p_3 p_4} \neq \emptyset$
- $\widetilde{p_1 p_2} \cap \widetilde{p_3 p_4} \supseteq \widetilde{p_1 p_2} \cap \overline{p_3 p_4} \neq \emptyset$

Opposite signs $\Rightarrow \overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Opposite signs $\Rightarrow \overline{p_3 p_4}$ crosses (infinite) line through p_1 and p_2

Solving Line Intersection

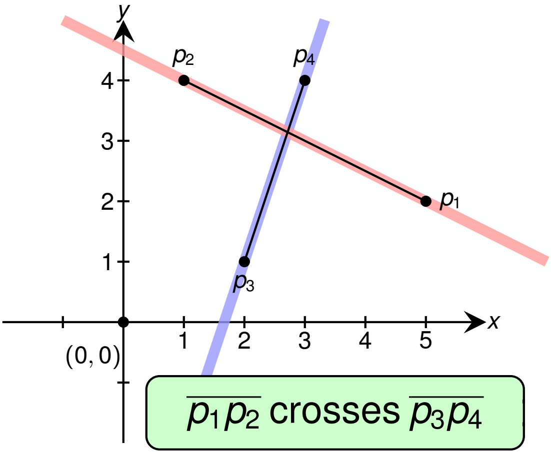


- $\widetilde{p_1p_2} \cap \widetilde{p_3p_4} \supseteq \overline{p_1p_2} \cap \widetilde{p_3p_4} \neq \emptyset$
- $\widetilde{p_1p_2} \cap \widetilde{p_3p_4} \supseteq \widetilde{p_1p_2} \cap \overline{p_3p_4} \neq \emptyset$
- Since $\widetilde{p_1p_2} \cap \widetilde{p_3p_4}$ consists of (at most) one point $\Rightarrow \overline{p_1p_2} \cap \overline{p_3p_4} \neq \emptyset$

Opposite signs $\Rightarrow \overline{p_1p_2}$ crosses (infinite) line through p_3 and p_4

Opposite signs $\Rightarrow \overline{p_3p_4}$ crosses (infinite) line through p_1 and p_2

Solving Line Intersection

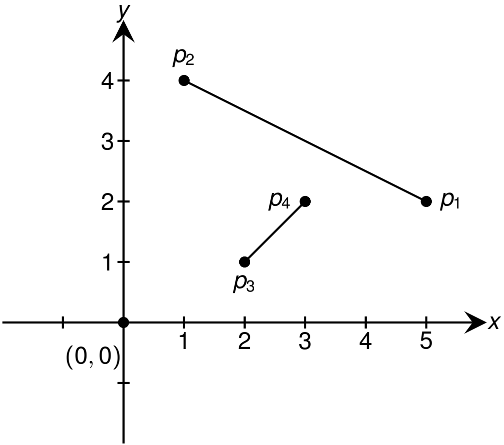


$\overline{p_1 p_2}$ crosses $\overline{p_3 p_4}$

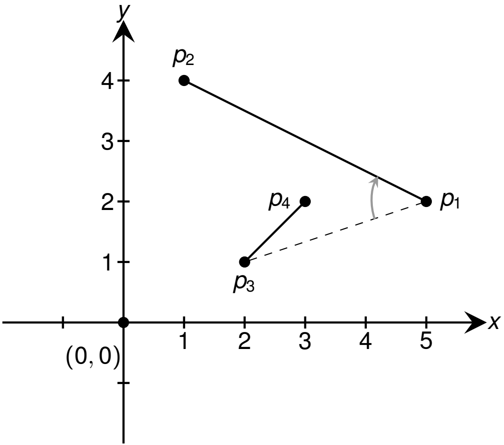
Opposite signs $\Rightarrow$ $\overline{p_1 p_2}$ crosses (infinite) line through p_3 and p_4

Opposite signs $\Rightarrow$ $\overline{p_3 p_4}$ crosses (infinite) line through p_1 and p_2

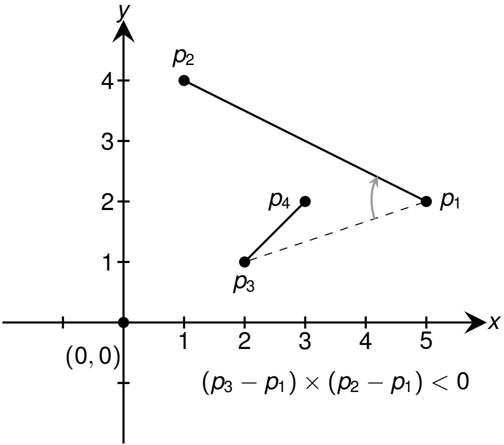
Solving Line Intersection



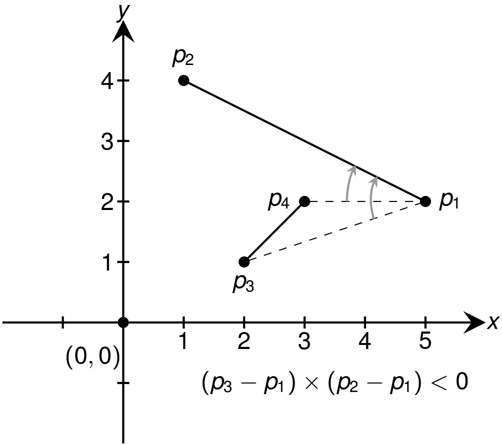
Solving Line Intersection



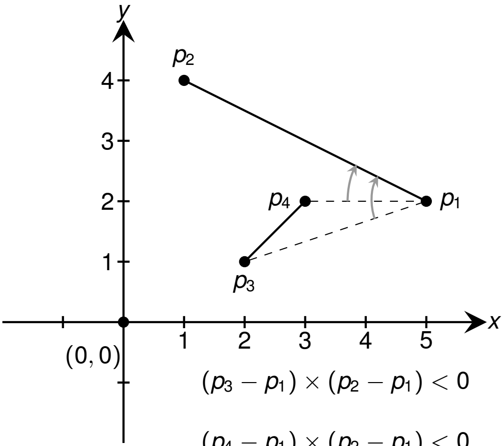
Solving Line Intersection



Solving Line Intersection



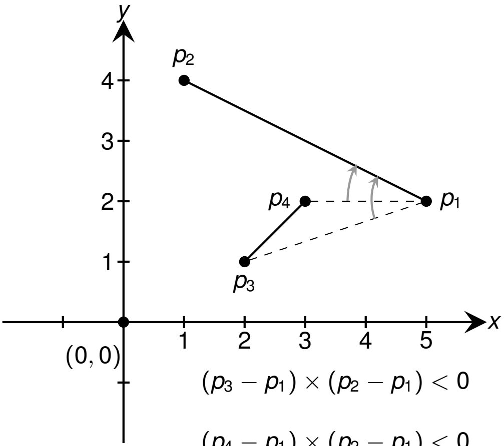
Solving Line Intersection



$$(p_3 - p_1) \times (p_2 - p_1) < 0$$

$$(p_4 - p_1) \times (p_2 - p_1) < 0$$

Solving Line Intersection

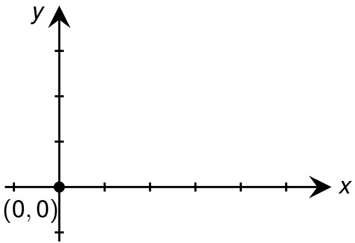


$$(p_3 - p_1) \times (p_2 - p_1) < 0$$

$$(p_4 - p_1) \times (p_2 - p_1) < 0$$

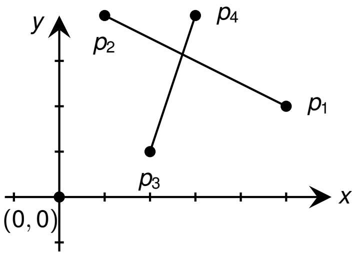
$\overline{p_1 p_2}$ does **not** cross $\overline{p_3 p_4}$

Solving Line Intersection



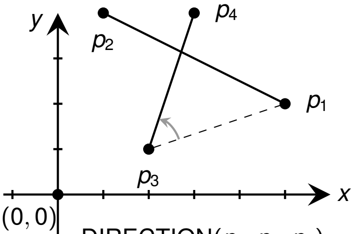

```
0: DIRECTION(p_i, p_j, p_k)
1:     return (p_k − p_i) × (p_j − p_i)
```

Solving Line Intersection



0: DIRECTION(p_i, p_j, p_k)
1: return $(p_k - p_i) \times (p_j - p_i)$

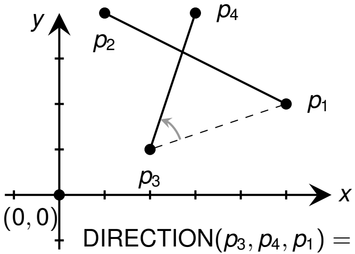
Solving Line Intersection



$$\text{DIRECTION}(p_3, p_4, p_1) = (p_1 - p_3) \times (p_4 - p_3)$$

```
DIRECTION(p_i, p_j, p_k)
    return (p_k − p_i) × (p_j − p_i)
```

Solving Line Intersection

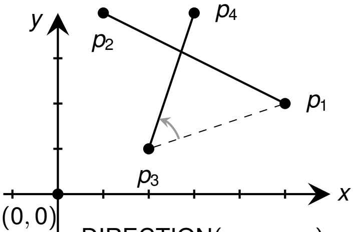


$$\text{DIRECTION}(p_3, p_4, p_1) = (p_1 - p_3) \times (p_4 - p_3)$$

DIRECTION(p_i, p_j, p_k)
    return $(p_k - p_i) \times (p_j - p_i)$

SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)
    $d_1 = \text{DIRECTION}(p_3, p_4, p_1)$
    $d_2 = \text{DIRECTION}(p_3, p_4, p_2)$
    $d_3 = \text{DIRECTION}(p_1, p_2, p_3)$
    $d_4 = \text{DIRECTION}(p_1, p_2, p_4)$
    If $d_1 \cdot d_2 < 0$ and $d_3 \cdot d_4 < 0$ return TRUE
        $\cdots$ (handle all degenerate cases)

Solving Line Intersection

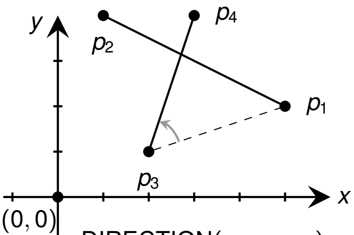


$$\text{DIRECTION}(p_3, p_4, p_1) = (p_1 - p_3) \times (p_4 - p_3)$$

```
DIRECTION(p_i, p_j, p_k)
    return (p_k − p_i) × (p_j − p_i)
```

```
SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)
    d_1 = DIRECTION(p_3, p_4, p_1)
    d_2 = DIRECTION(p_3, p_4, p_2)
    d_3 = DIRECTION(p_1, p_2, p_3)
    d_4 = DIRECTION(p_1, p_2, p_4)
    If d_1 · d_2 < 0 and d_3 · d_4 < 0 return TRUE
        ··· (handle all degenerate cases)
```

Solving Line Intersection



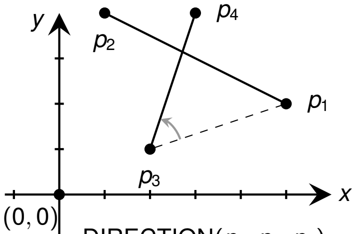
$$\text{DIRECTION}(p_3, p_4, p_1) = (p_1 - p_3) \times (p_4 - p_3)$$

```
DIRECTION(p_i, p_j, p_k)
    return (p_k − p_i) × (p_j − p_i)
```

```
SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)
    d_1 = DIRECTION(p_3, p_4, p_1)
    d_2 = DIRECTION(p_3, p_4, p_2)
    d_3 = DIRECTION(p_1, p_2, p_3)
    d_4 = DIRECTION(p_1, p_2, p_4)
    If d_1 · d_2 < 0 and d_3 · d_4 < 0 return TRUE
        · · · (handle all degenerate cases)
```

In total 4 satisfying conditions!

Solving Line Intersection



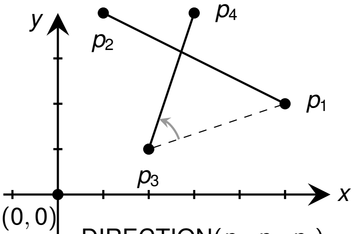
$$\text{DIRECTION}(p_3, p_4, p_1) = (p_1 - p_3) \times (p_4 - p_3)$$

```
DIRECTION(p_i, p_j, p_k)
    return (p_k − p_i) × (p_j − p_i)
```

```
SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)
    d_1 = DIRECTION(p_3, p_4, p_1)
    d_2 = DIRECTION(p_3, p_4, p_2)
    d_3 = DIRECTION(p_1, p_2, p_3)
    d_4 = DIRECTION(p_1, p_2, p_4)
    If d_1 · d_2 < 0 and d_3 · d_4 < 0 return TRUE
        · · · (handle all degenerate cases)
```

Lines could touch or be colinear

Solving Line Intersection

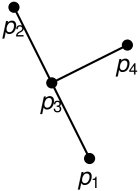


$$\text{DIRECTION}(p_3, p_4, p_1) = (p_1 - p_3) \times (p_4 - p_3)$$

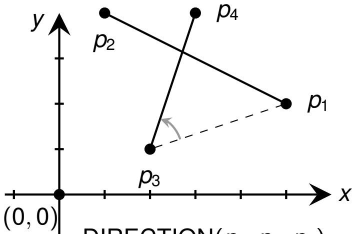
```
DIRECTION(p_i, p_j, p_k)
    return (p_k − p_i) × (p_j − p_i)
```

```
SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)
    d_1 = DIRECTION(p_3, p_4, p_1)
    d_2 = DIRECTION(p_3, p_4, p_2)
    d_3 = DIRECTION(p_1, p_2, p_3)
    d_4 = DIRECTION(p_1, p_2, p_4)
    If d_1 · d_2 < 0 and d_3 · d_4 < 0 return TRUE
        ··· (handle all degenerate cases)
```

Lines could touch or be colinear



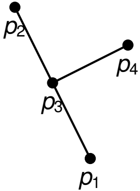
Solving Line Intersection

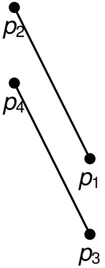


$$\text{DIRECTION}(p_3, p_4, p_1) = (p_1 - p_3) \times (p_4 - p_3)$$

```
DIRECTION(p_i, p_j, p_k)
    return (p_k − p_i) × (p_j − p_i)
```

```
SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)
    d_1 = DIRECTION(p_3, p_4, p_1)
    d_2 = DIRECTION(p_3, p_4, p_2)
    d_3 = DIRECTION(p_1, p_2, p_3)
    d_4 = DIRECTION(p_1, p_2, p_4)
    If d_1 · d_2 < 0 and d_3 · d_4 < 0 return TRUE
        ··· (handle all degenerate cases)
```





Lines could touch or be colinear

Outline

Introduction and Line Intersection

Convex Hull

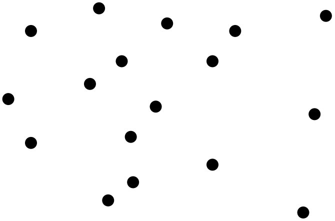
Glimpse at (More) Advanced Algorithms

Convex Hull

Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

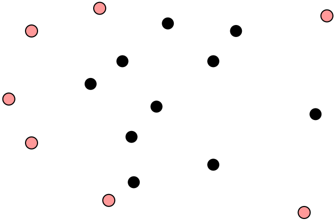
Convex Hull



Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

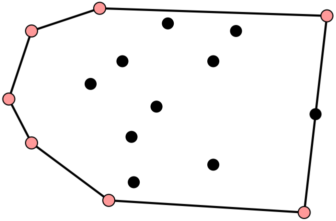
Convex Hull



Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

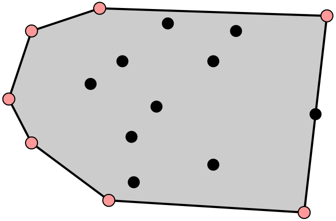
Convex Hull



Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

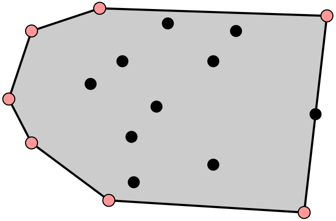
Convex Hull



Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Convex Hull

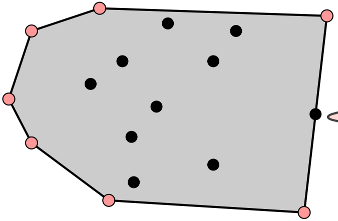


Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Smallest perimeter fence enclosing the points

Convex Hull

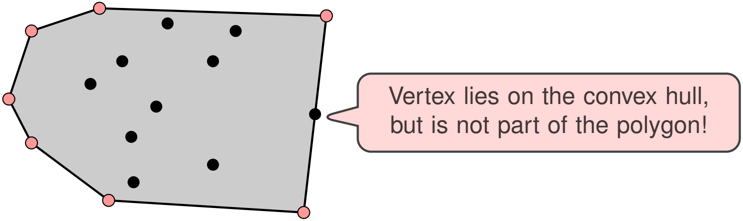


Vertex lies on the convex hull, but is not part of the polygon!

Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Convex Hull

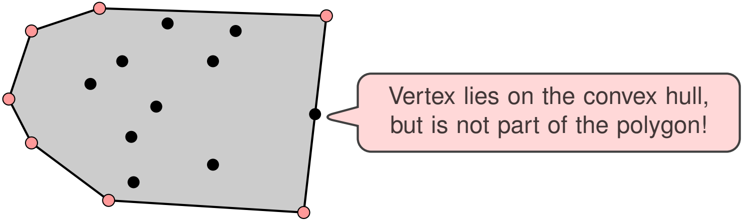


Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Convex Hull Problem

Convex Hull



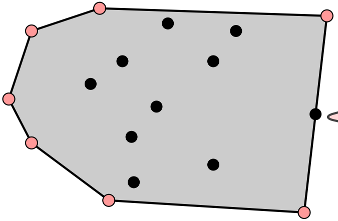
Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Convex Hull Problem

- Input: set of points Q in the Euclidean space

Convex Hull



Vertex lies on the convex hull, but is not part of the polygon!

Definition

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

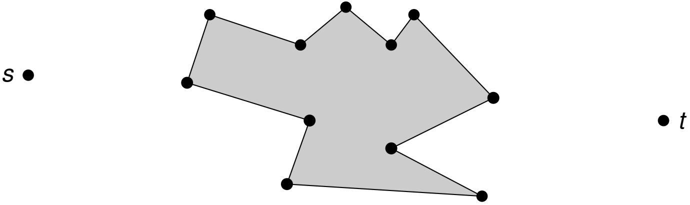
Convex Hull Problem

- Input: set of points Q in the Euclidean space
- Output: return points of the convex hull in counterclockwise order

Application of Convex Hull

Robot Motion Planning

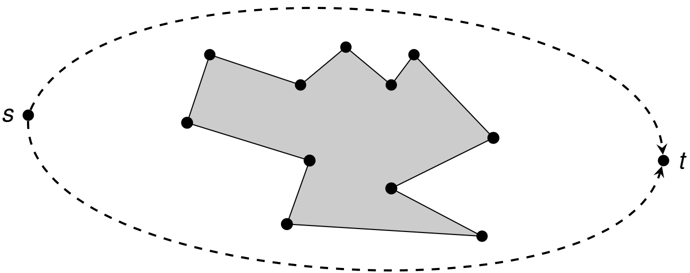
Find shortest path from s to t which avoids a polygonal obstacle.



Application of Convex Hull

Robot Motion Planning

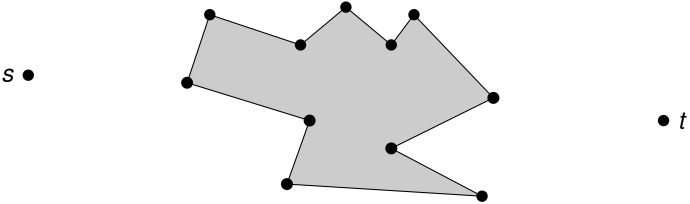
Find shortest path from s to t which avoids a polygonal obstacle.



Application of Convex Hull

Robot Motion Planning

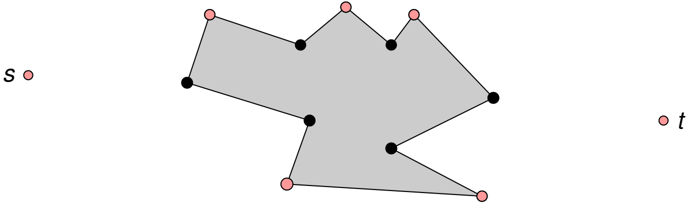
Find shortest path from s to t which avoids a polygonal obstacle.



Application of Convex Hull

Robot Motion Planning

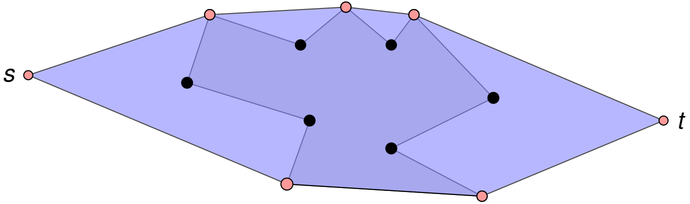
Find shortest path from s to t which avoids a polygonal obstacle.



Application of Convex Hull

Robot Motion Planning

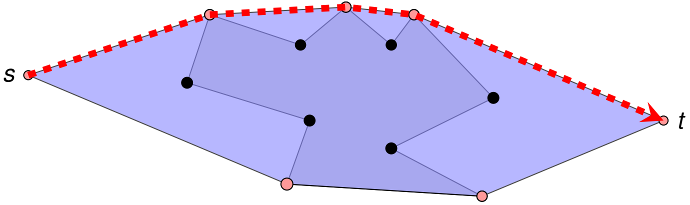
Find shortest path from s to t which avoids a polygonal obstacle.



Application of Convex Hull

Robot Motion Planning

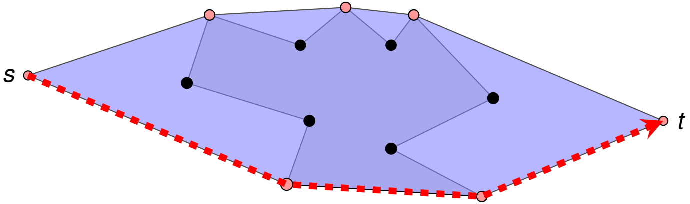
Find shortest path from s to t which avoids a polygonal obstacle.



Application of Convex Hull

Robot Motion Planning

Find shortest path from s to t which avoids a polygonal obstacle.

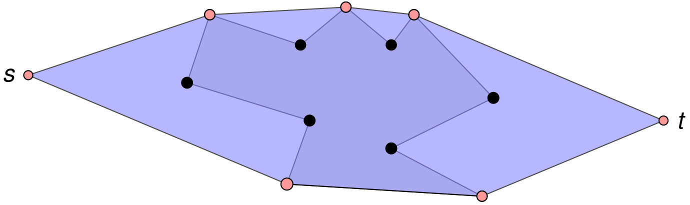


Application of Convex Hull

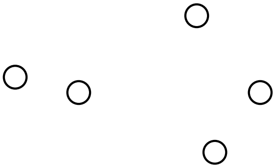
Robot Motion Planning

Find shortest path from s to t which avoids a polygonal obstacle.

can be solved by computing the Convex hull!



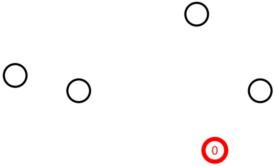
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate

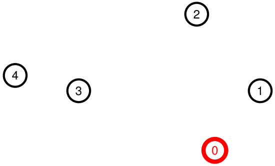
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate

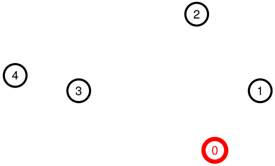
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle

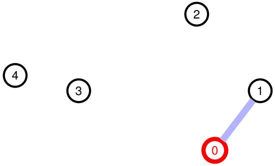
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull

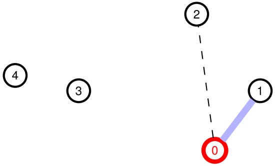
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull

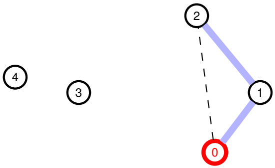
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine

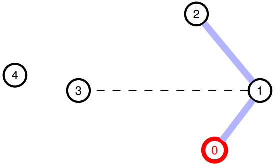
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine

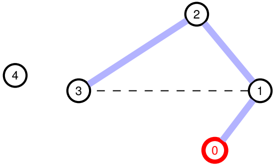
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine

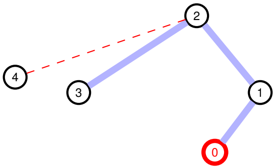
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓

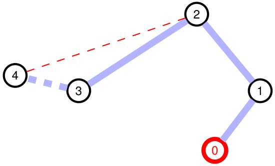
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise,

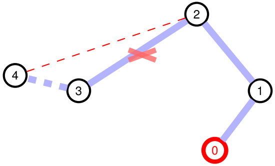
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise, keep on removing recent points until point can be added

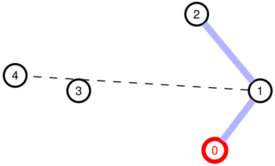
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise, keep on removing recent points until point can be added

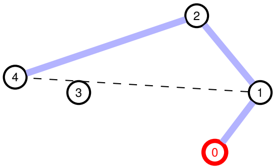
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise, keep on removing recent points until point can be added

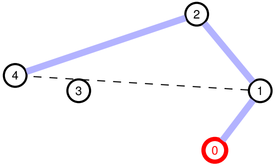
Graham's Scan



Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise, keep on removing recent points until point can be added

Graham's Scan

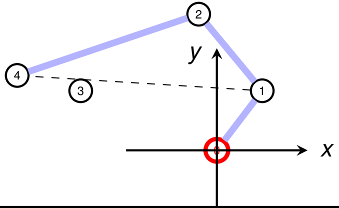


Efficient Sorting by comparing (not computing!) polar angles

Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise, keep on removing recent points until point can be added

Graham's Scan

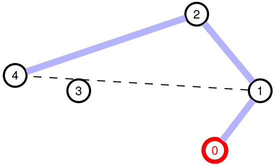


Efficient Sorting by comparing (not computing!) polar angles

Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise, keep on removing recent points until point can be added

Graham's Scan

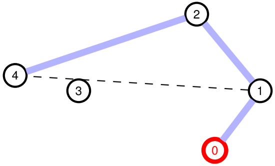


Efficient Sorting by comparing (not computing!) polar angles

Basic Idea

- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise, keep on removing recent points until point can be added

Graham's Scan



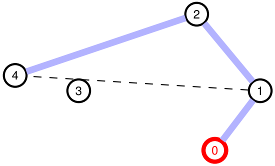
Use Cross Product!

Efficient Sorting by comparing (not computing!) polar angles

Basic Idea

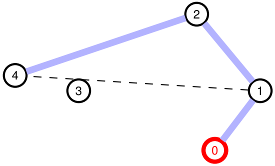
- Start with the point with smallest y-coordinate
- Sort all points increasingly according to their polar angle
- Try to add next point to the convex hull
  - If it does not introduce non-left turn, then fine ✓
  - Otherwise, keep on removing recent points until point can be added

Graham's Scan



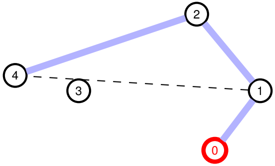
GRAHAM-SCAN(Q)
    Let p_0 be the point with minimum y-coordinate
    Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. p_0
    If $n < 2$ return false
    $S = \emptyset$
    PUSH(p_0,S)
    PUSH(p_1,S)
    PUSH(p_2,S)
    For $i = 3$ to n
        While angle of NEXT-TO-TOP(S),TOP(S),p_i makes a non-left turn
            POP(S)
        End While
        PUSH(p_i,S)
    End For
    Return S

Graham's Scan




```
GRAHAM-SCAN(Q)
    Let p_0 be the point with minimum y-coordinate
    Let (p_1, p_2, ..., p_n) be the other points sorted by polar angle w.r.t. p_0
    If n < 2 return false
    S = ∅
    PUSH(p_0,S)
    PUSH(p_1,S)
    PUSH(p_2,S)
    For i = 3 to n
        While angle of NEXT-TO-TOP(S),TOP(S),p_i makes a non-left turn
            POP(S)
        End While
        PUSH(p_i,S)
    End For
    Return S
```

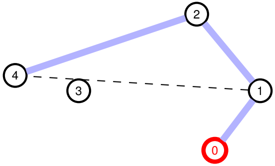
Graham's Scan



GRAHAM-SCAN(Q)
Let p_0 be the point with minimum y-coordinate
Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. p_0
If $n < 2$ return false
$S = \emptyset$
PUSH(p_0,S)
PUSH(p_1,S)
PUSH(p_2,S)
For $i = 3$ to n
  While angle of NEXT-TO-TOP(S),TOP(S),p_i makes a non-left turn
    POP(S)
  End While
  PUSH(p_i,S)
End For
Return S

Takes $O(n \log n)$ time

Graham's Scan



GRAHAM-SCAN(Q)

  Let p_0 be the point with minimum y-coordinate

  Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. p_0

  If $n < 2$ return false

  $S = \emptyset$

  PUSH(p_0,S)

  PUSH(p_1,S)

  PUSH(p_2,S)

  For $i = 3$ to n

    While angle of NEXT-TO-TOP(S),TOP(S),p_i makes a non-left turn

      POP(S)

    End While

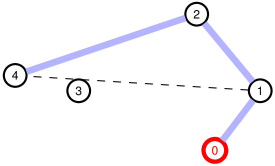
    PUSH(p_i,S)

  End For

  Return S

Takes $O(n \log n)$ time

Graham's Scan

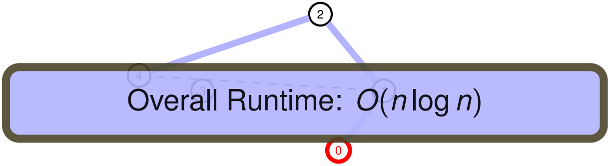



```
GRAHAM-SCAN(Q)
    Let p_0 be the point with minimum y-coordinate
    Let (p_1, p_2, ..., p_n) be the other points sorted by polar angle w.r.t. p_0
    If n < 2 return false
    S = ∅
    PUSH(p_0,S)
    PUSH(p_1,S)
    PUSH(p_2,S)
    For i = 3 to n
        While angle of NEXT-TO-TOP(S),TOP(S),p_i makes a non-left turn
            POP(S)
        End While
        PUSH(p_i,S)
    End For
    Return S
```

Takes $O(n \log n)$ time

Takes $O(n)$ time, since every point is part of a PUSH or POP at most once.

Graham's Scan



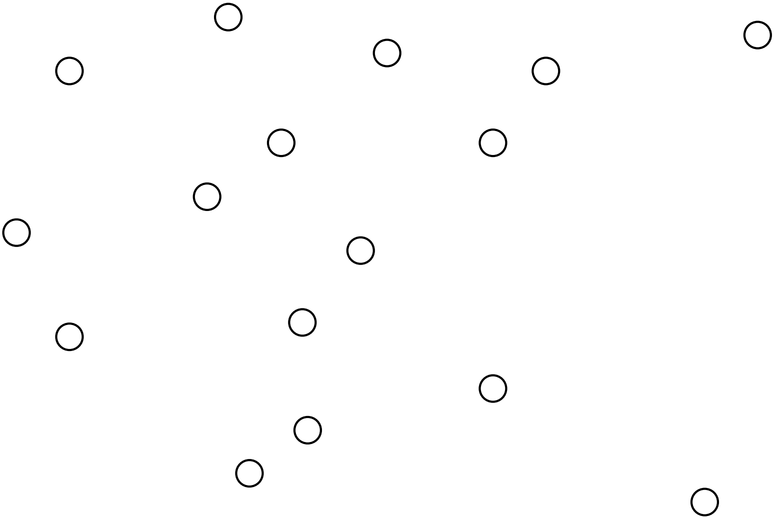
Overall Runtime: $O(n \log n)$

```
GRAHAM-SCAN(Q)
    Let p_0 be the point with minimum y-coordinate
    Let (p_1, p_2, ..., p_n) be the other points sorted by polar angle w.r.t. p_0
    If n < 2 return false
    S = ∅
    PUSH(p_0,S)
    PUSH(p_1,S)
    PUSH(p_2,S)
    For i = 3 to n
        While angle of NEXT-TO-TOP(S),TOP(S),p_i makes a non-left turn
            POP(S)
        End While
        PUSH(p_i,S)
    End For
    Return S
```

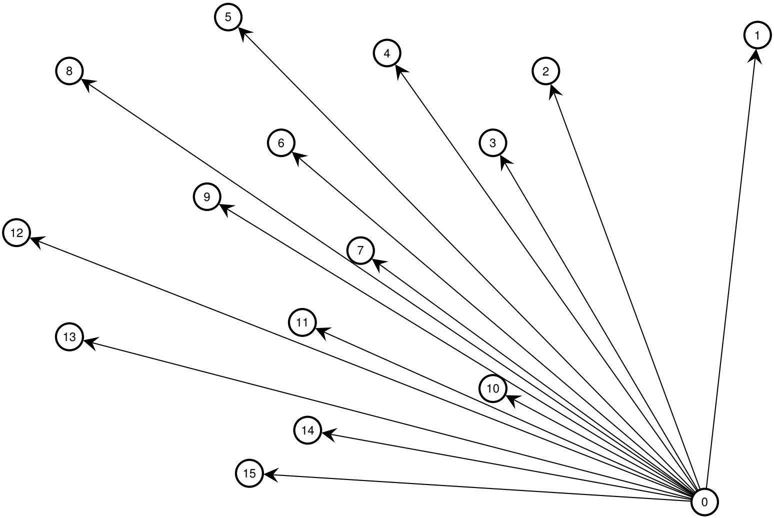
Takes $O(n \log n)$ time

Takes $O(n)$ time, since every point is part of a PUSH or POP at most once.

Execution of Graham's Scan

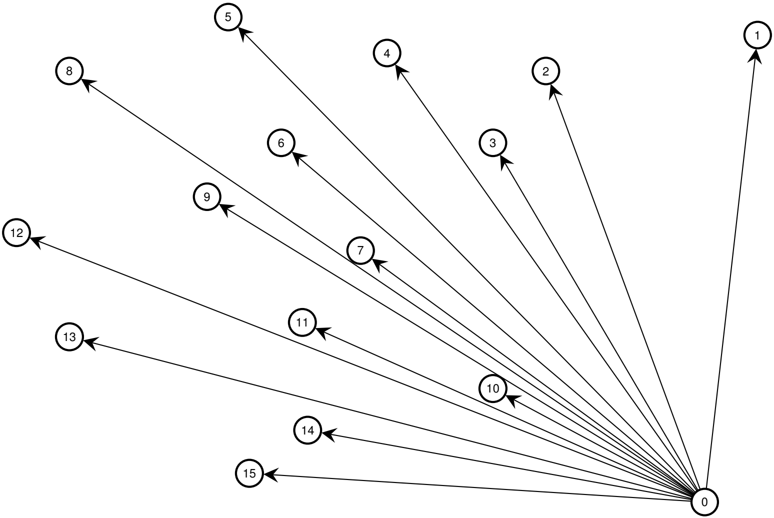


Execution of Graham's Scan

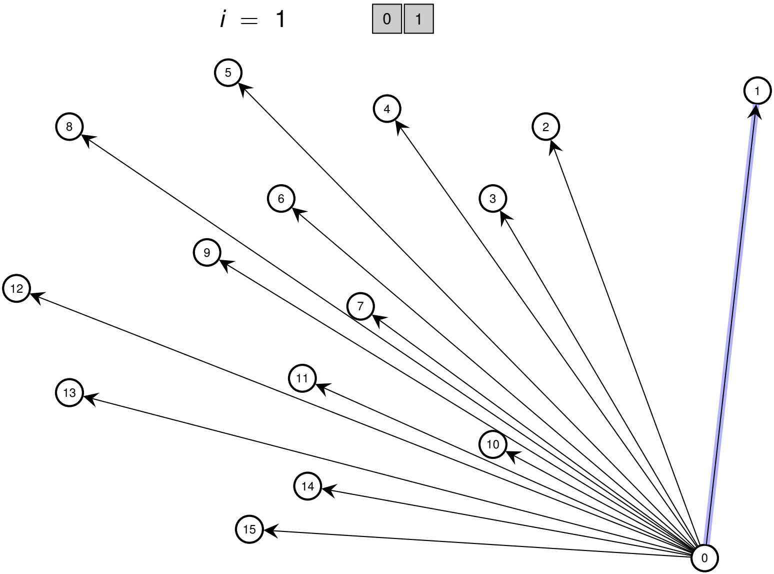


Execution of Graham's Scan

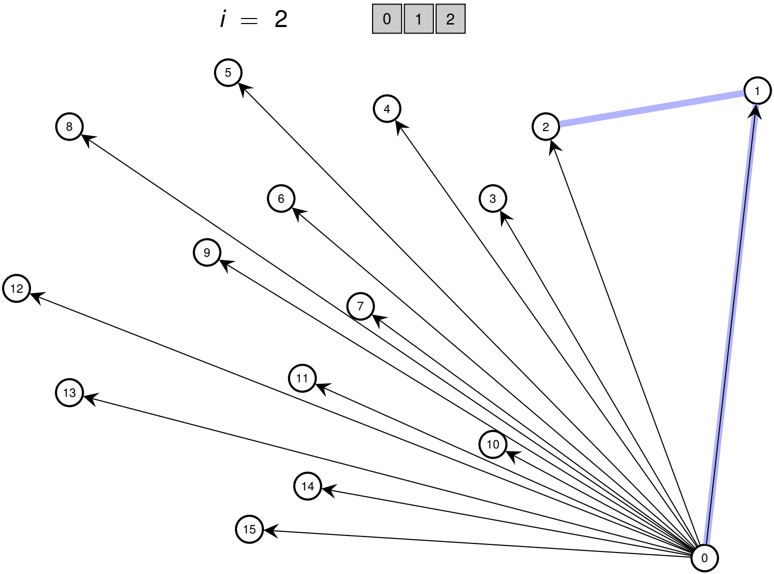
$i = 0$ $\boxed{0}$



Execution of Graham's Scan



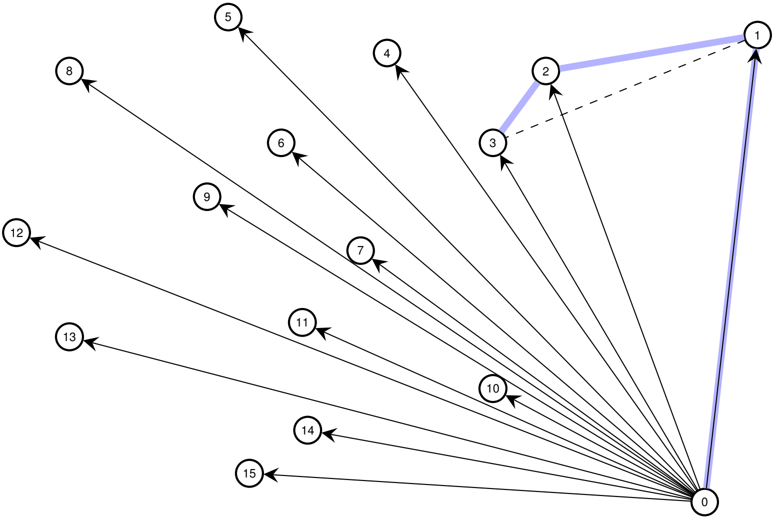
Execution of Graham's Scan



Execution of Graham's Scan

$i = 3$

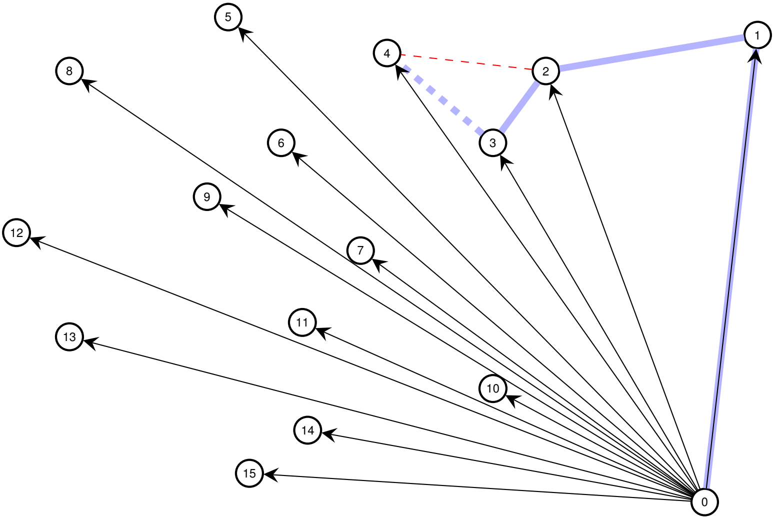
0	1	2	3



Execution of Graham's Scan

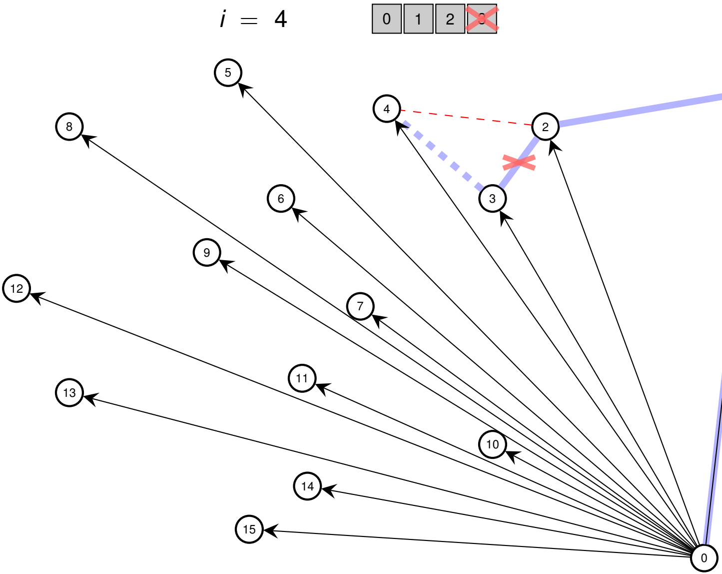
$i = 4$

0	1	2	3



Execution of Graham's Scan

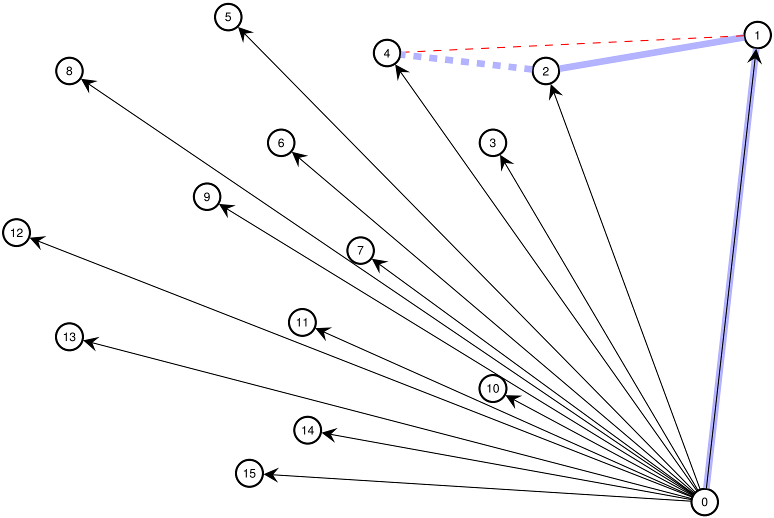
$i = 4$



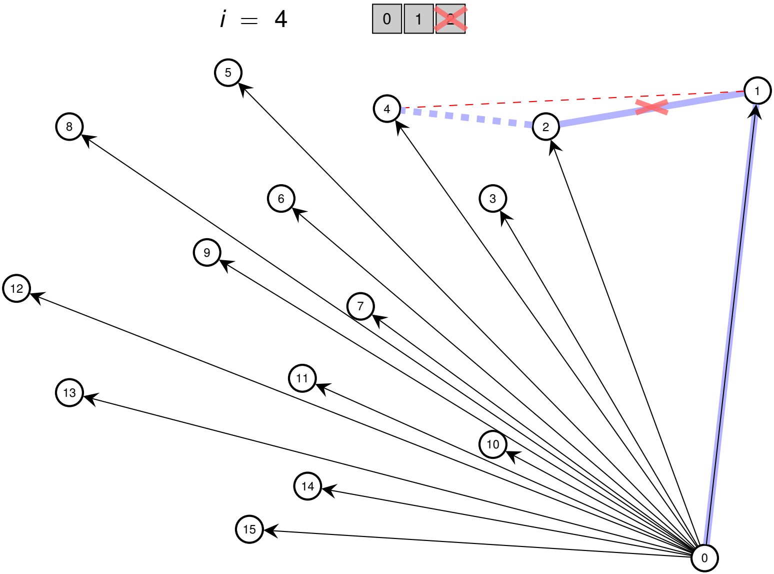
Execution of Graham's Scan

$i = 4$

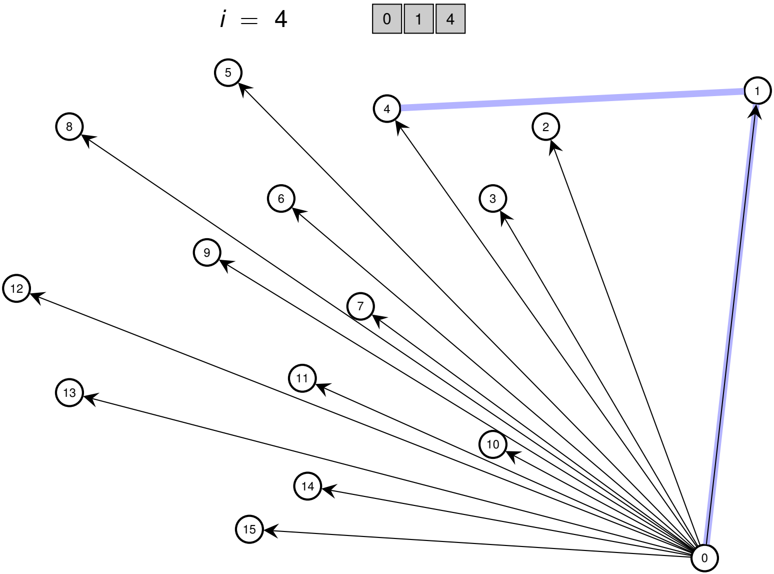
0	1	2



Execution of Graham's Scan



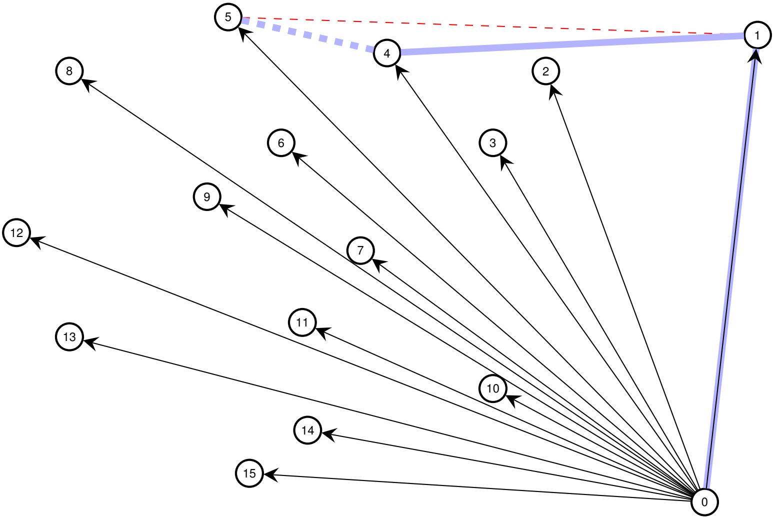
Execution of Graham's Scan



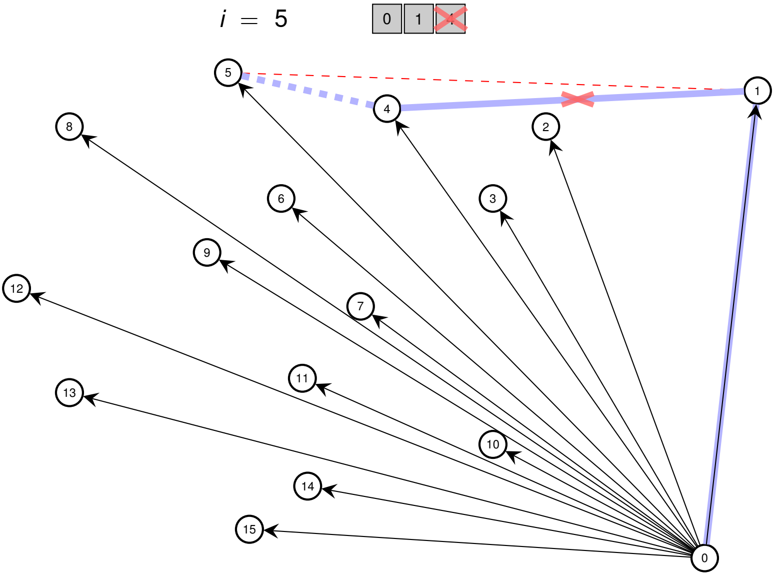
Execution of Graham's Scan

$i = 5$

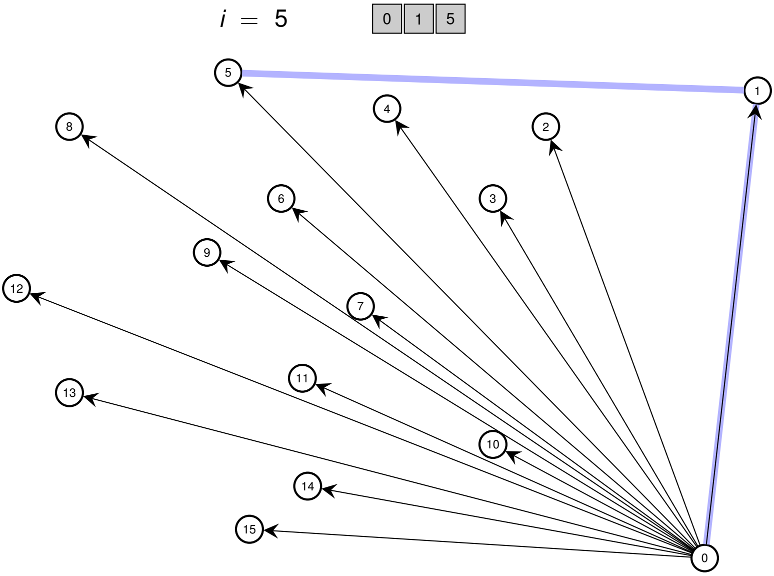
0	1	4



Execution of Graham's Scan



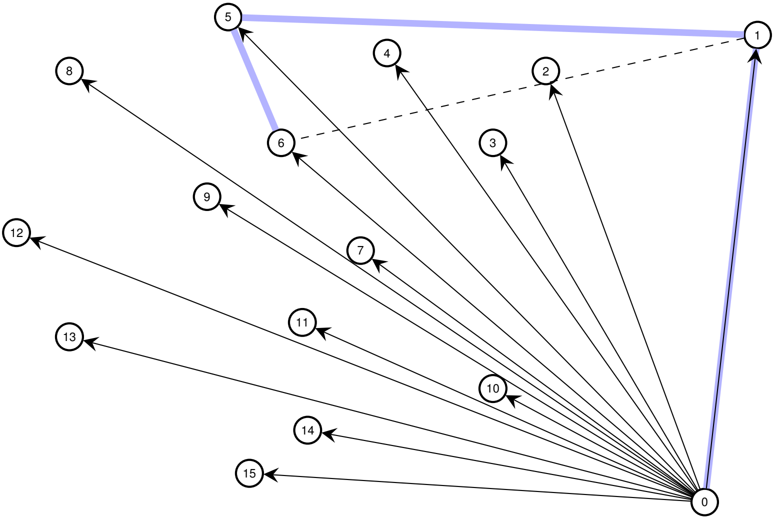
Execution of Graham's Scan



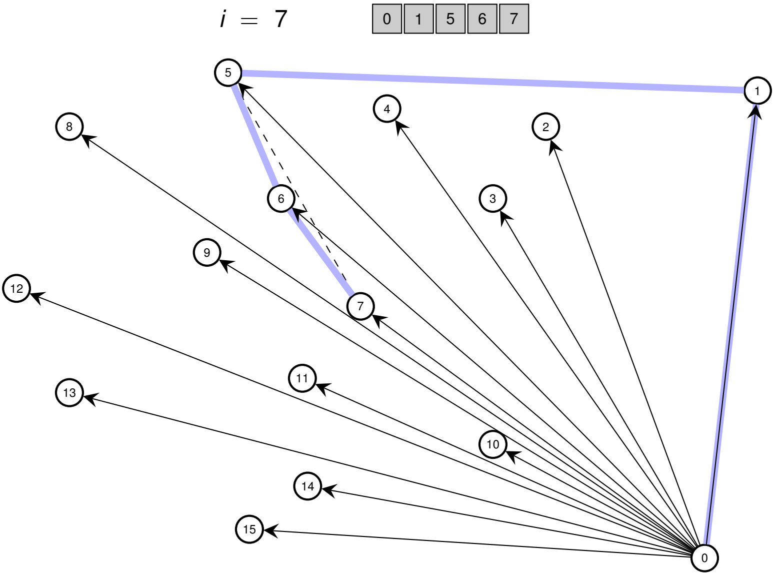
Execution of Graham's Scan

$i = 6$

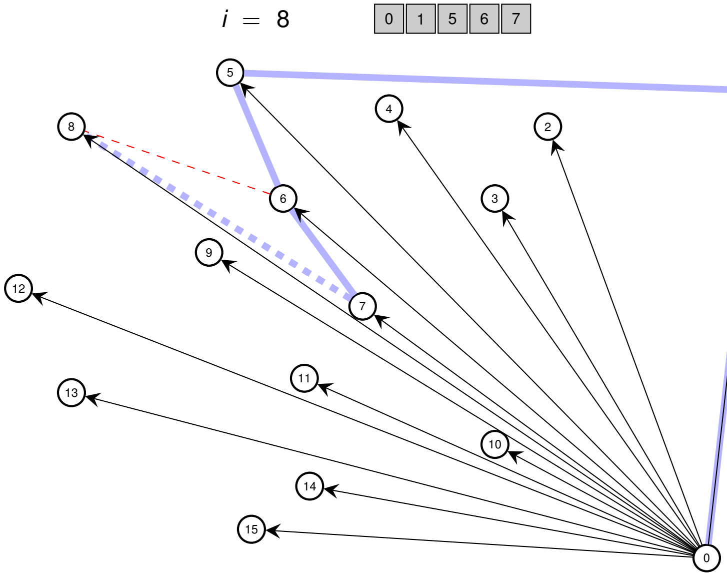
0	1	5	6



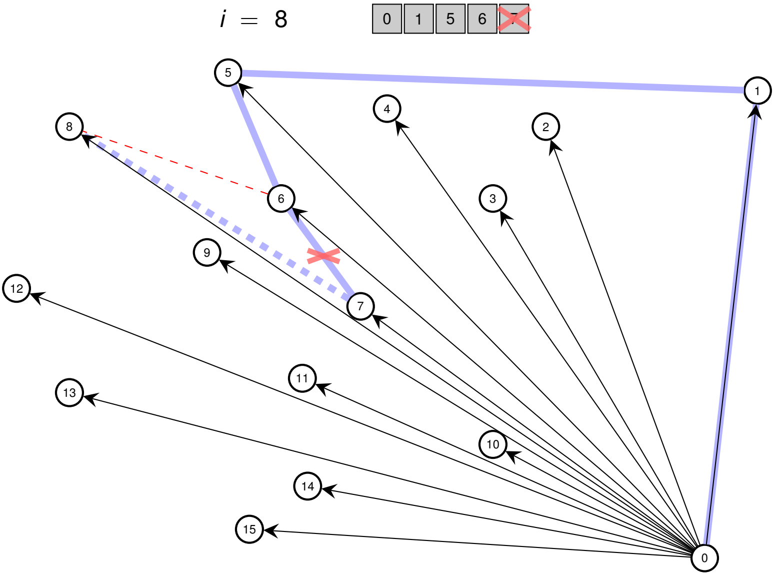
Execution of Graham's Scan



Execution of Graham's Scan



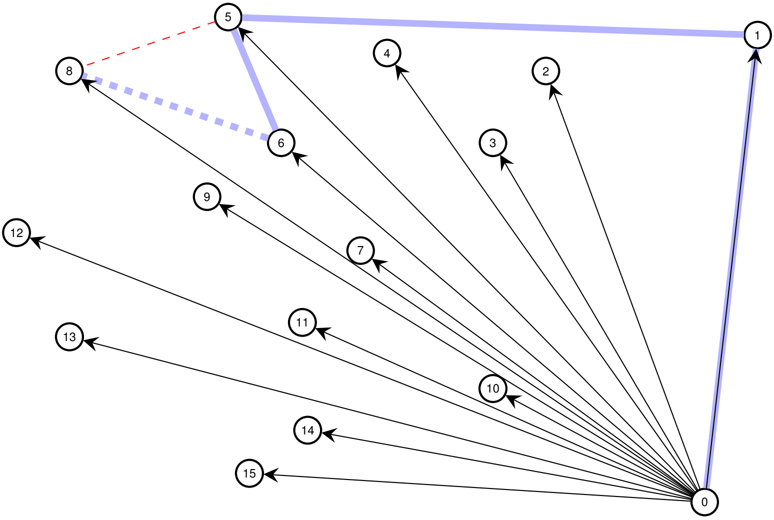
Execution of Graham's Scan



Execution of Graham's Scan

$i = 8$

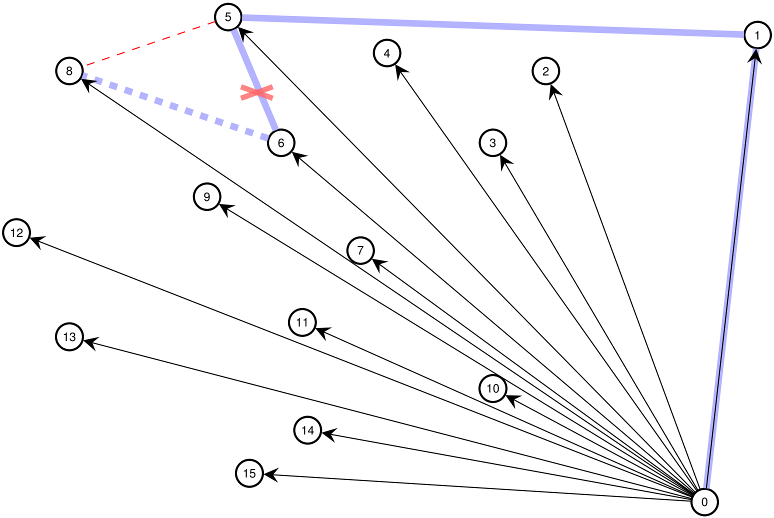
0	1	5	6



Execution of Graham's Scan

$i = 8$

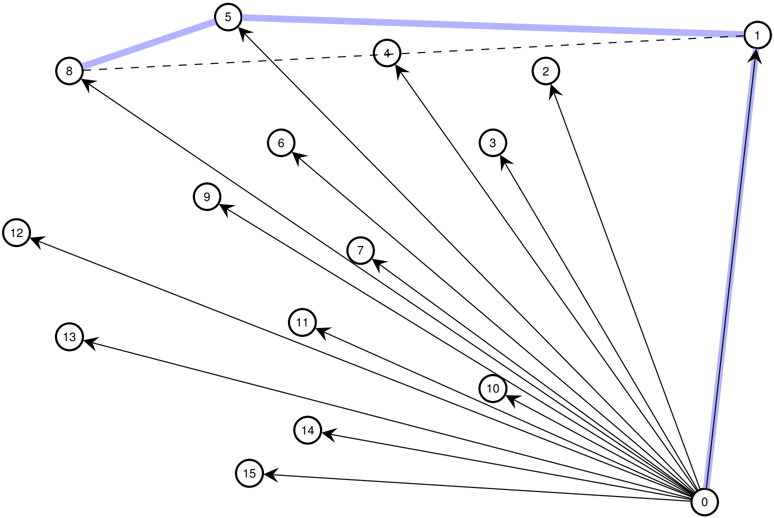
0	1	5	~~6~~



Execution of Graham's Scan

$i = 8$

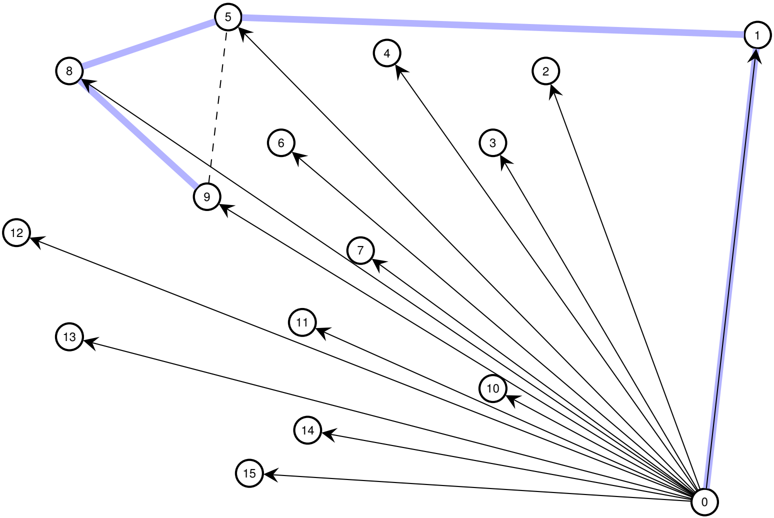
0	1	5	8



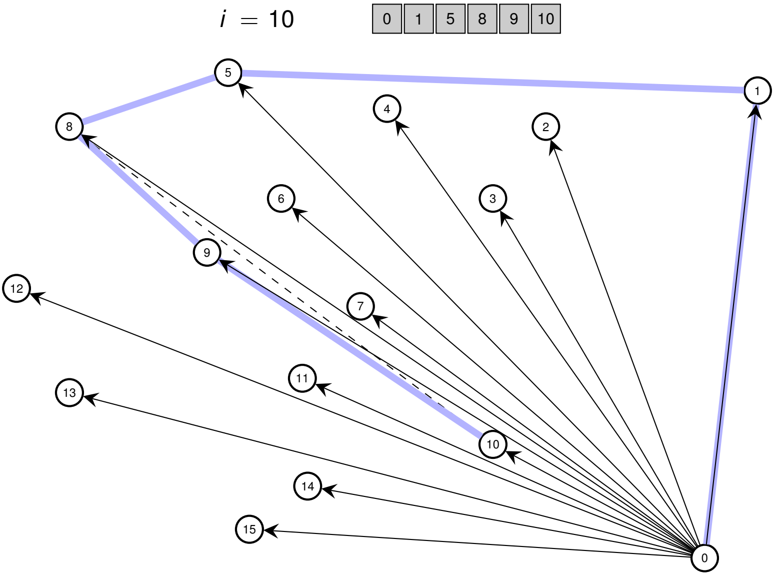
Execution of Graham's Scan

$i = 9$

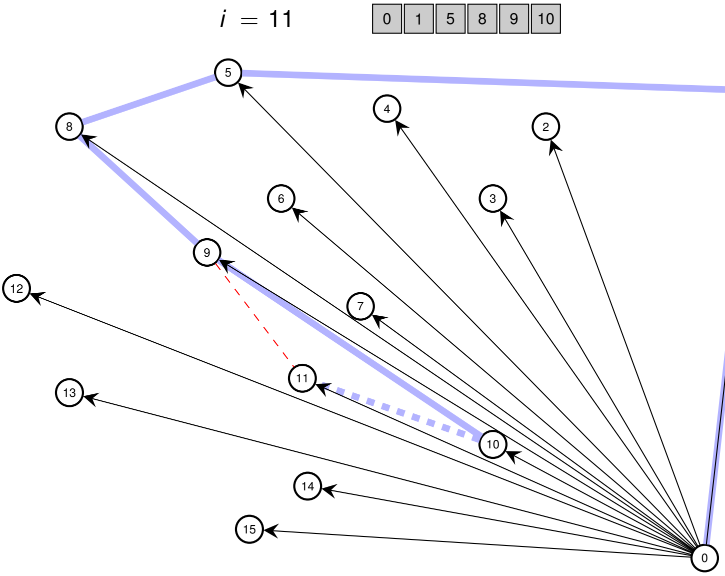
0	1	5	8	9



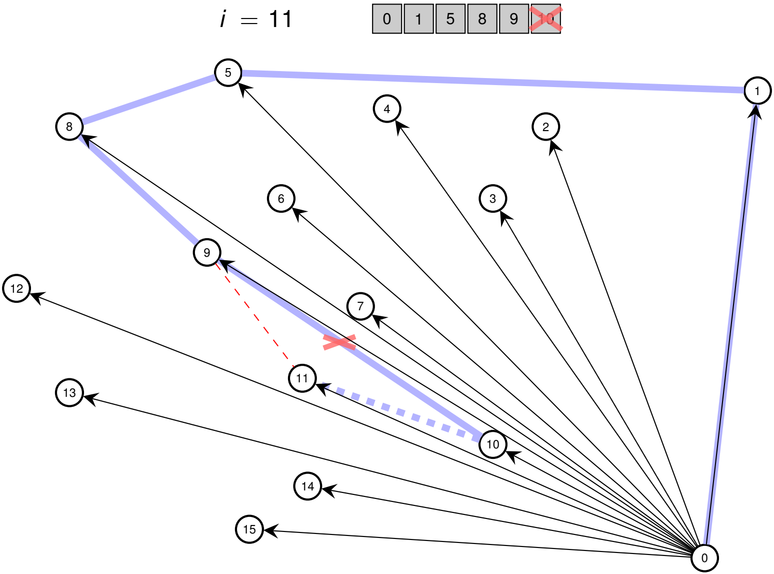
Execution of Graham's Scan



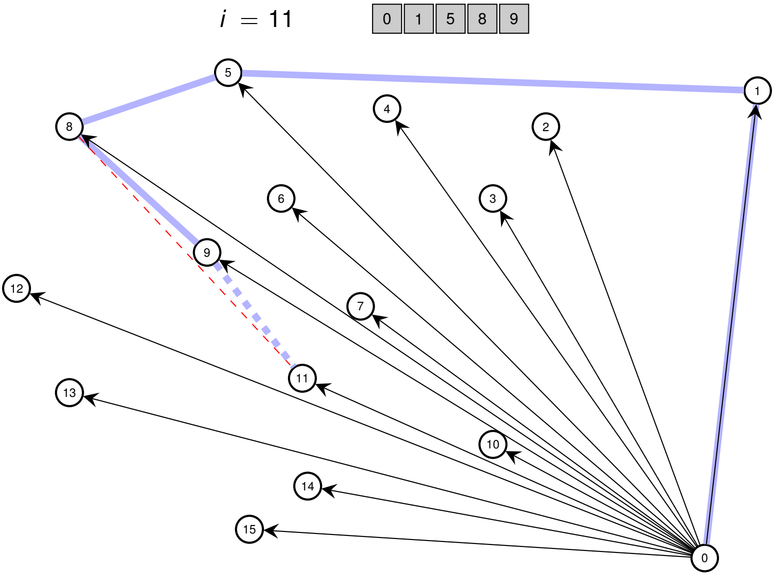
Execution of Graham's Scan



Execution of Graham's Scan



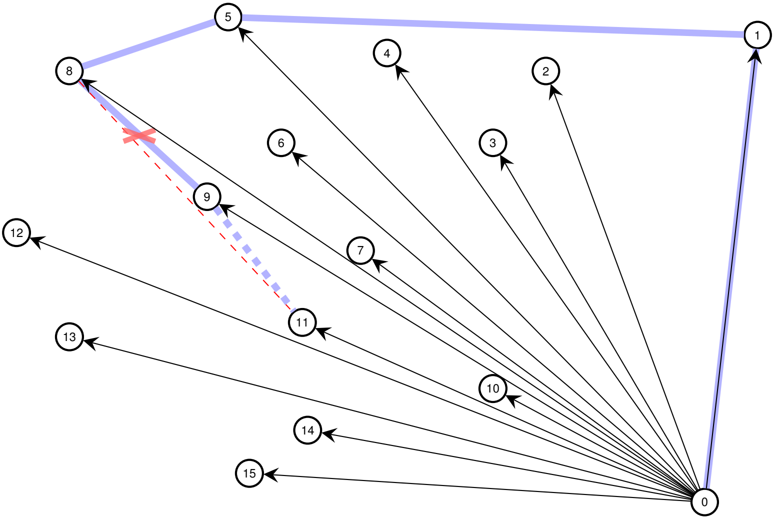
Execution of Graham's Scan



Execution of Graham's Scan

$i = 11$

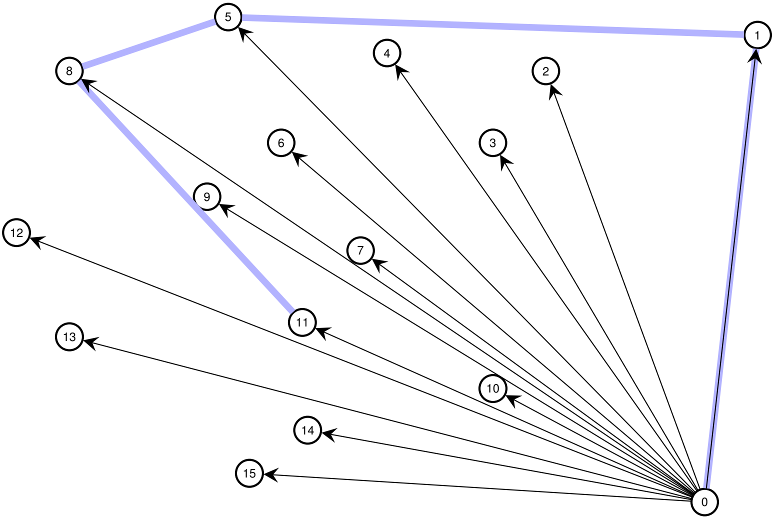
0	1	5	8	~~9~~



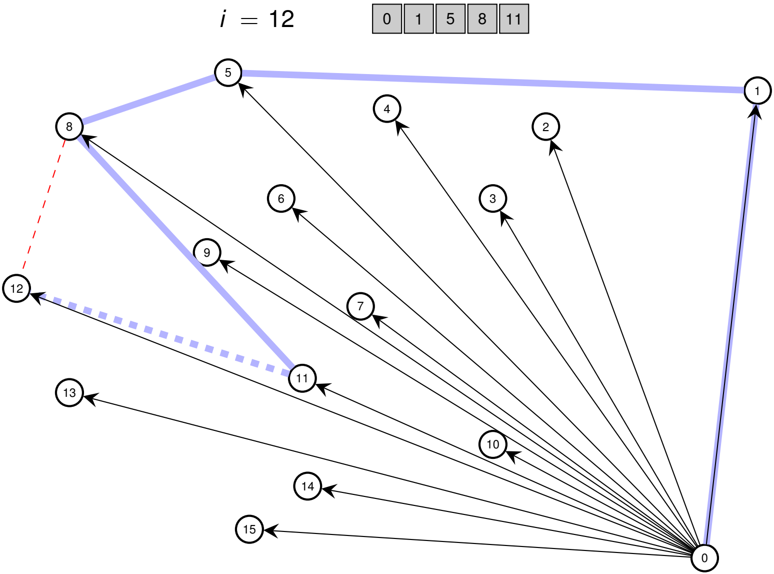
Execution of Graham's Scan

$i = 11$

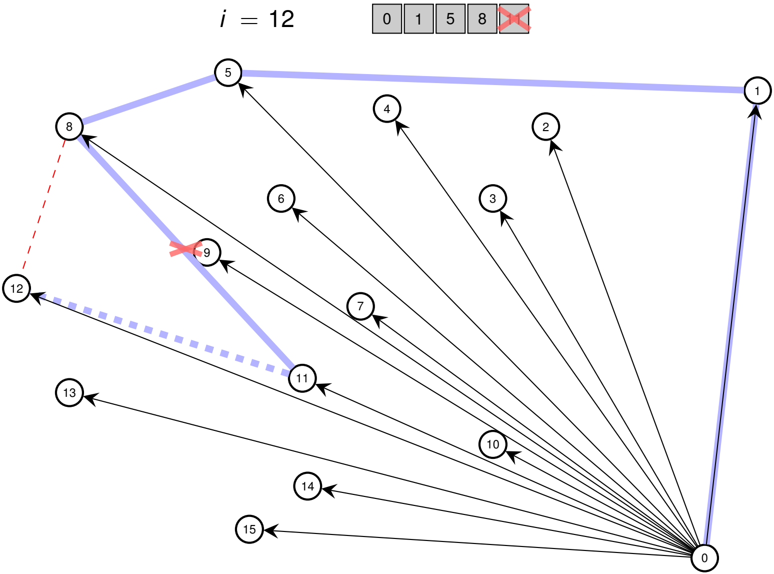
0	1	5	8	11



Execution of Graham's Scan



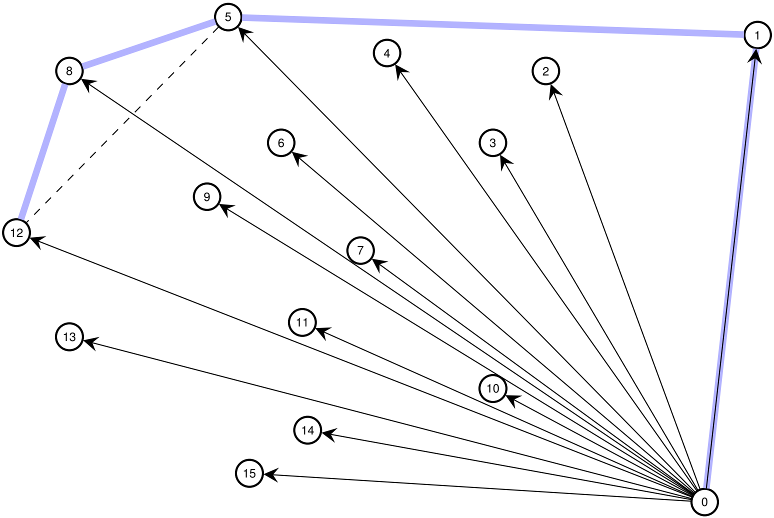
Execution of Graham's Scan



Execution of Graham's Scan

$i = 12$

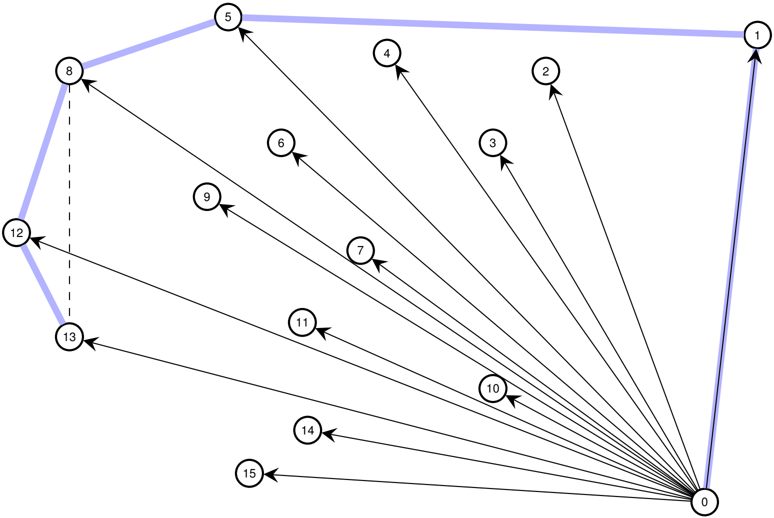
0	1	5	8	12



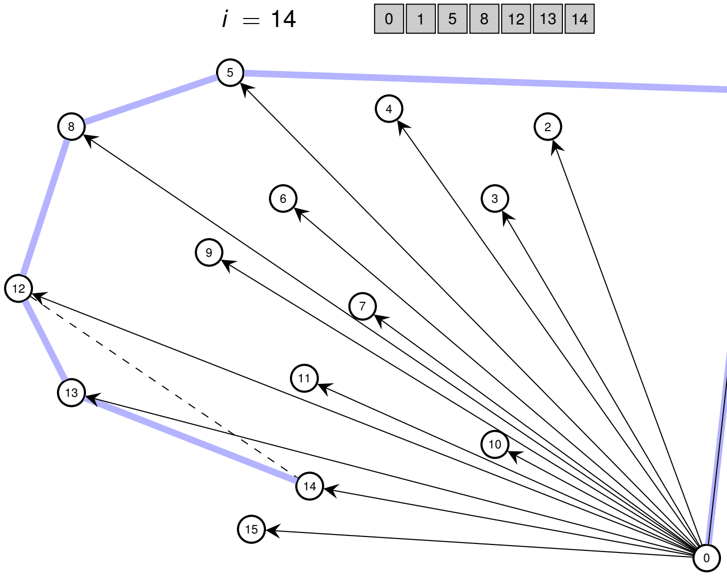
Execution of Graham's Scan

$i = 13$

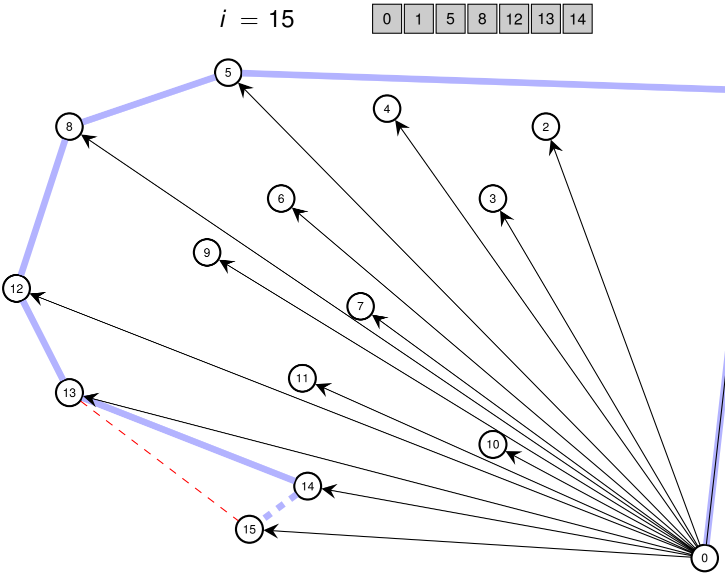
0	1	5	8	12	13



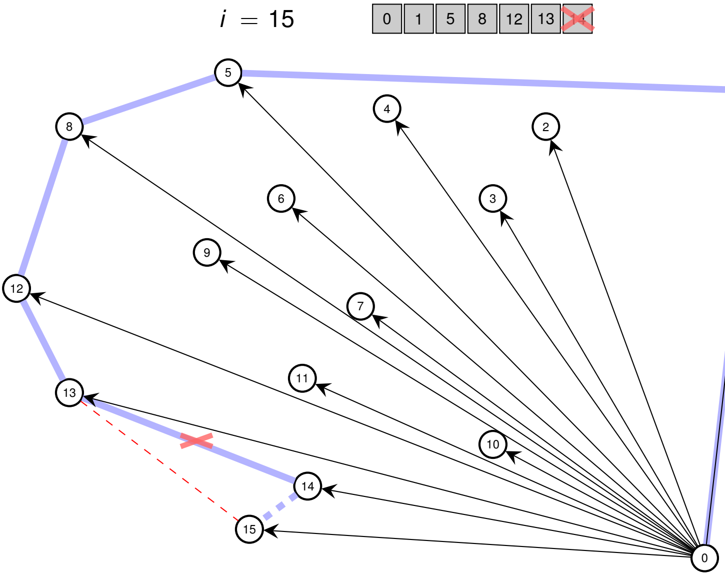
Execution of Graham's Scan



Execution of Graham's Scan



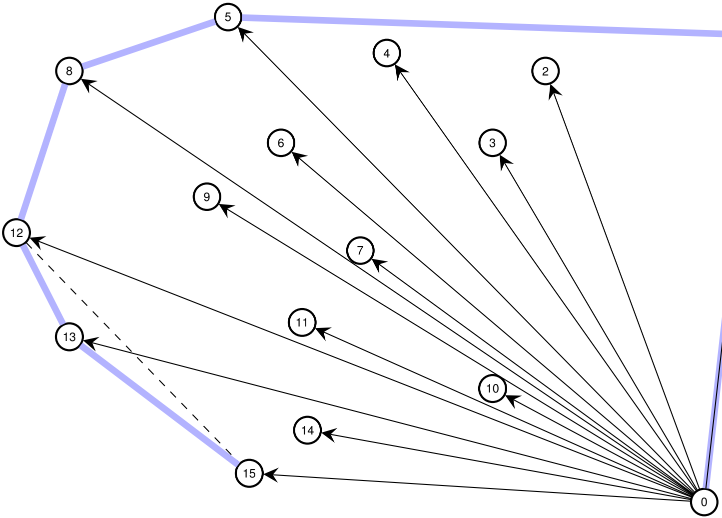
Execution of Graham's Scan



Execution of Graham's Scan

$i = 15$

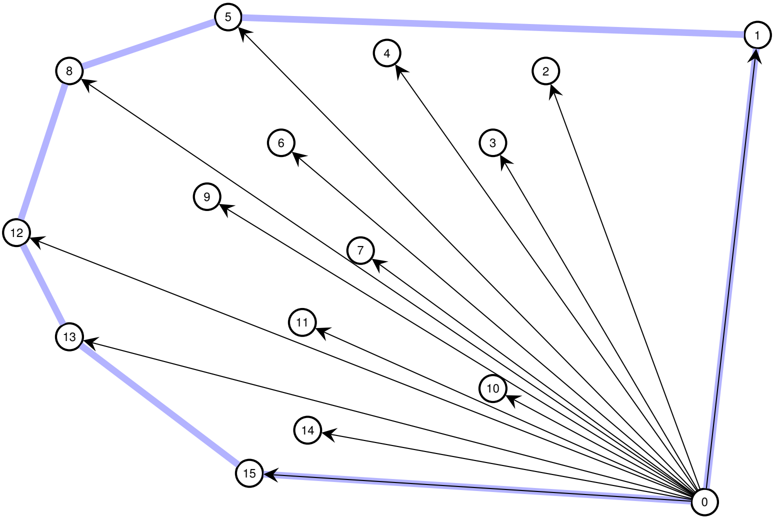
0	1	5	8	12	13	15



Execution of Graham's Scan

$i = 15$ ✓

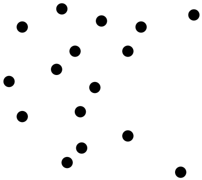
0	1	5	8	12	13	15



Jarvis' March (Gift wrapping)

Intuition

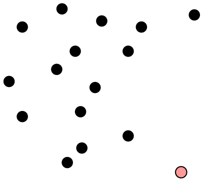
- Wrapping taut paper around the points



Jarvis' March (Gift wrapping)

Intuition

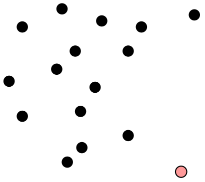
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point



Jarvis' March (Gift wrapping)

Intuition

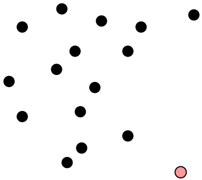
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point



Jarvis' March (Gift wrapping)

Intuition

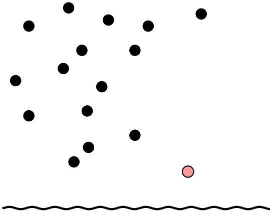
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point



Jarvis' March (Gift wrapping)

Intuition

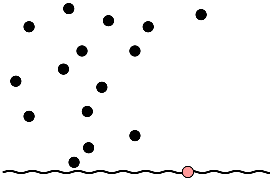
- Wrapping taut paper around the points
    1. Tape end of paper at lowest point



Jarvis' March (Gift wrapping)

Intuition

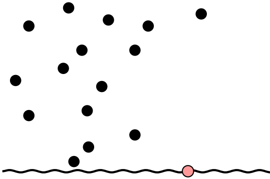
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point



Jarvis' March (Gift wrapping)

Intuition

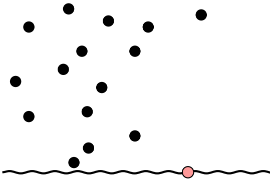
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

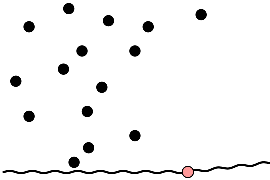
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

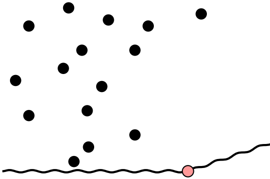
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

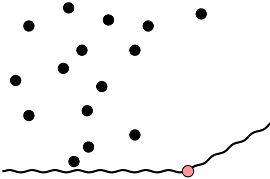
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

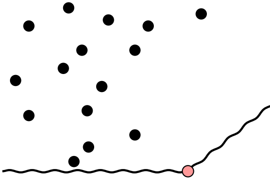
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

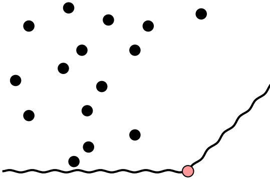
- Wrapping taut paper around the points
    1. Tape end of paper at lowest point
    2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

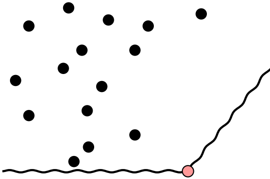
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

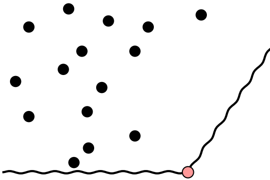
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

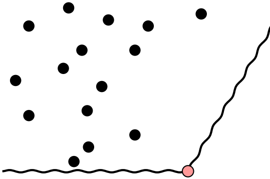
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

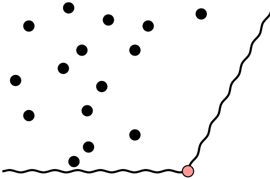
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

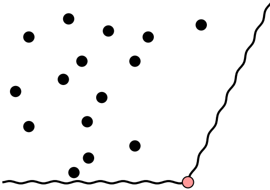
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

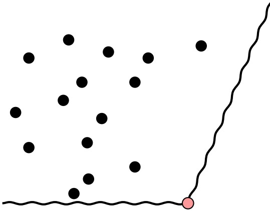
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

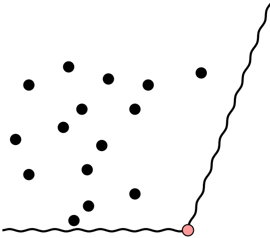
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

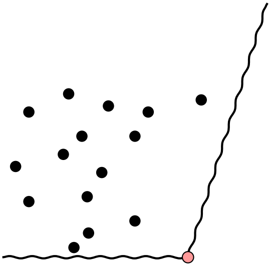
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

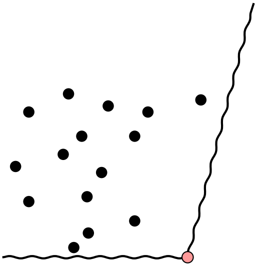
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

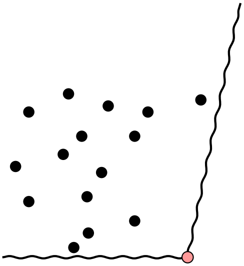
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

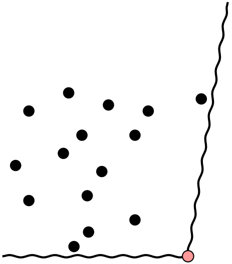
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

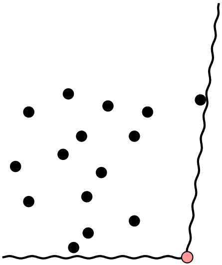
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

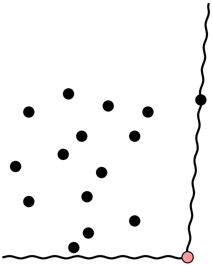
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

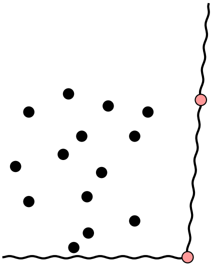
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point



Jarvis' March (Gift wrapping)

Intuition

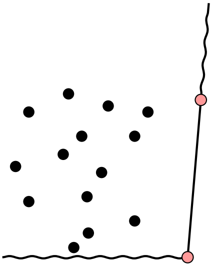
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



Jarvis' March (Gift wrapping)

Intuition

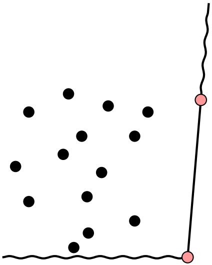
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



Jarvis' March (Gift wrapping)

Intuition

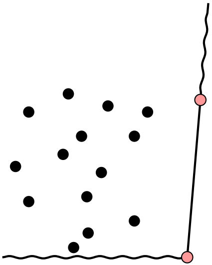
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



Jarvis' March (Gift wrapping)

Intuition

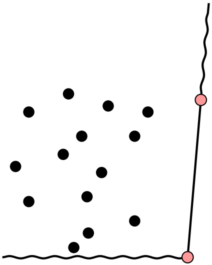
- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



Jarvis' March (Gift wrapping)

Intuition

- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



Algorithm

1. Let p_0 be the lowest point
2. Next point the one with smallest angle w.r.t. p_0

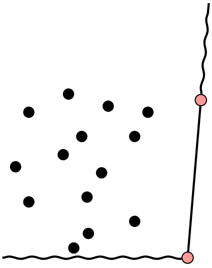
Jarvis' March (Gift wrapping)

Intuition

- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2

Algorithm

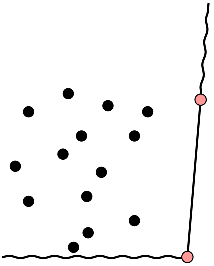
1. Let p_0 be the lowest point
2. Next point the one with smallest angle w.r.t. p_0
3. Continue until highest point p_k



Jarvis' March (Gift wrapping)

Intuition

- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



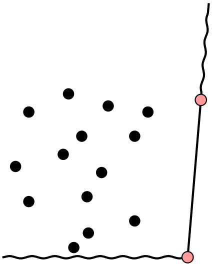
Algorithm

1. Let p_0 be the lowest point
2. Next point the one with smallest angle w.r.t. p_0
3. Continue until highest point p_k
4. Next point the one with smallest angle w.r.t. p_k

Jarvis' March (Gift wrapping)

Intuition

- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



Algorithm

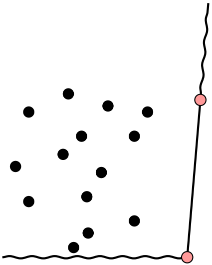
1. Let p_0 be the lowest point
2. Next point the one with smallest angle w.r.t. p_0
3. Continue until highest point p_k
4. Next point the one with smallest angle w.r.t. p_k

Here, we rotate the coordinate system by 180!

Jarvis' March (Gift wrapping)

Intuition

- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



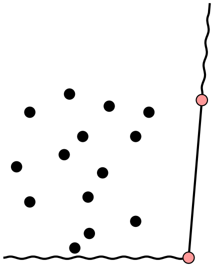
Algorithm

1. Let p_0 be the lowest point
2. Next point the one with smallest angle w.r.t. p_0
3. Continue until highest point p_k
4. Next point the one with smallest angle w.r.t. p_k
5. Continue until p_0 is reached

Jarvis' March (Gift wrapping)

Intuition

- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



Algorithm

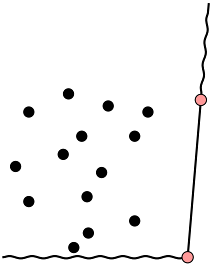
1. Let p_0 be the lowest point
2. Next point the one with smallest angle w.r.t. p_0
3. Continue until highest point p_k
4. Next point the one with smallest angle w.r.t. p_k
5. Continue until p_0 is reached

Runtime: $O(n \cdot h)$, where h is no. points on convex hull.

Jarvis' March (Gift wrapping)

Intuition

- Wrapping taut paper around the points
  1. Tape end of paper at lowest point
  2. Pull paper to the right until it touches a point
  3. Tape paper and go to 2



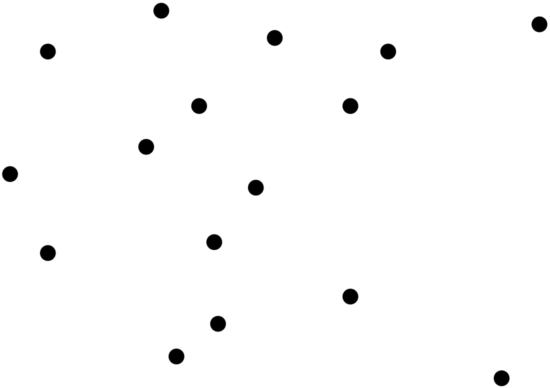
Algorithm

1. Let p_0 be the lowest point
2. Next point the one with smallest angle w.r.t. p_0
3. Continue until highest point p_k
4. Next point the one with smallest angle w.r.t. p_k
5. Continue until p_0 is reached

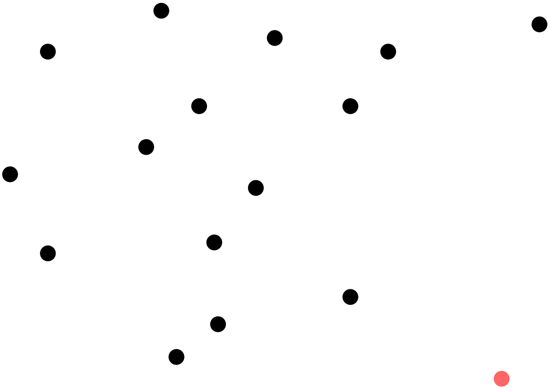
Runtime: $O(n \cdot h)$, where h is no. points on convex hull.

Output sensitive algorithm!

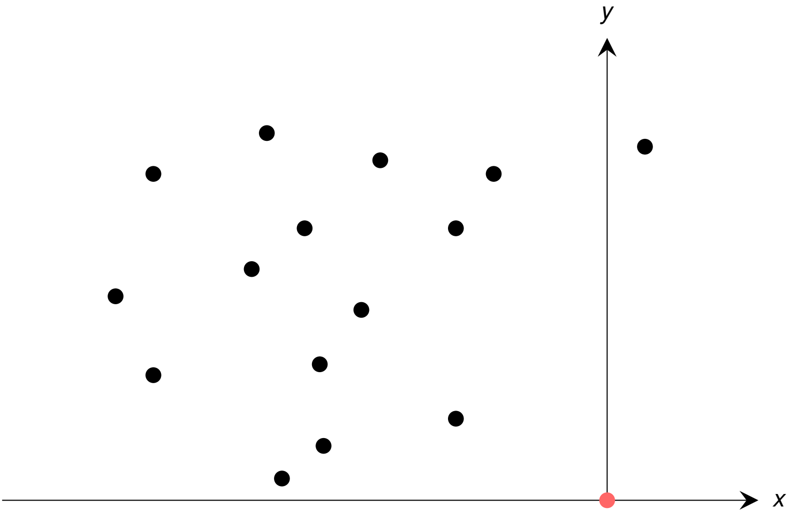
Execution of Jarvis' March



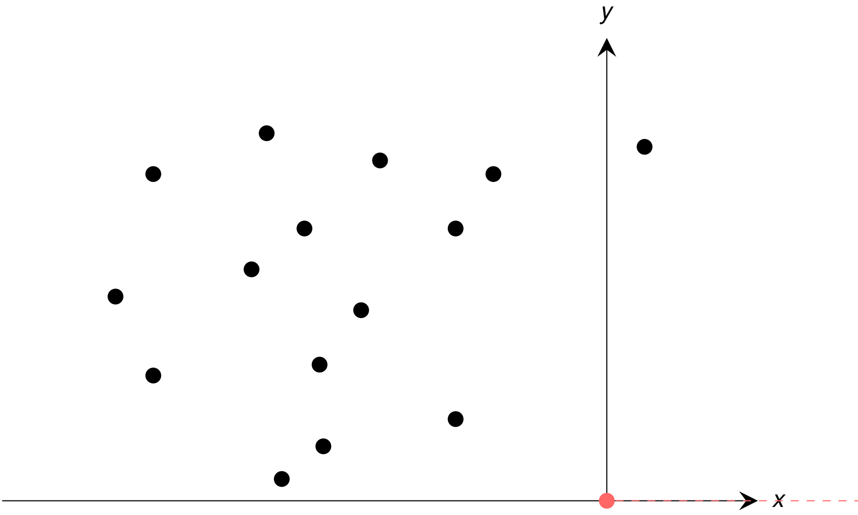
Execution of Jarvis' March



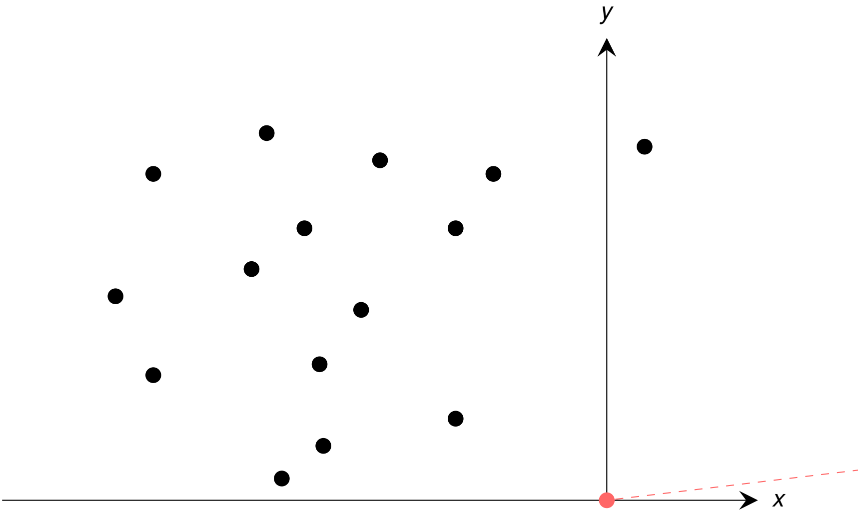
Execution of Jarvis' March



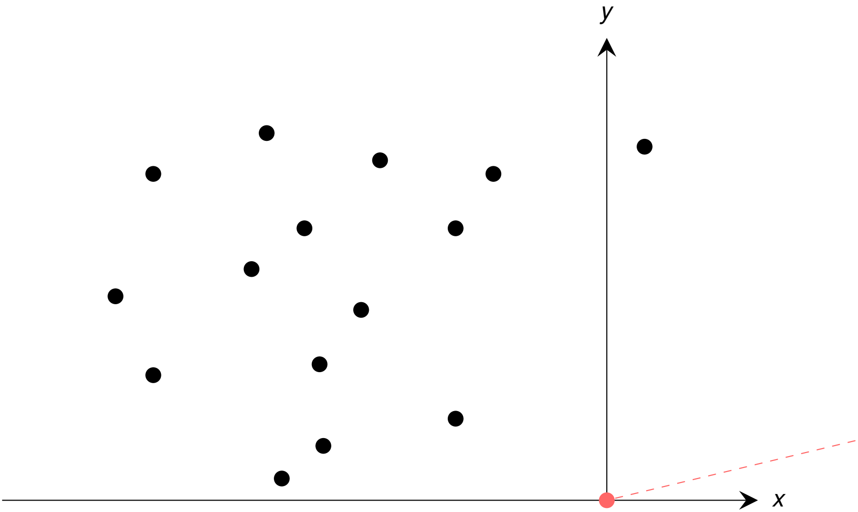
Execution of Jarvis' March



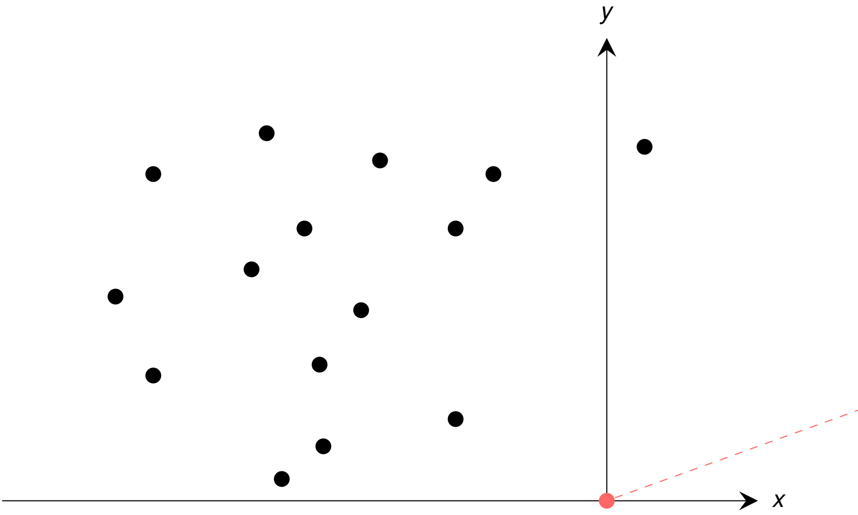
Execution of Jarvis' March



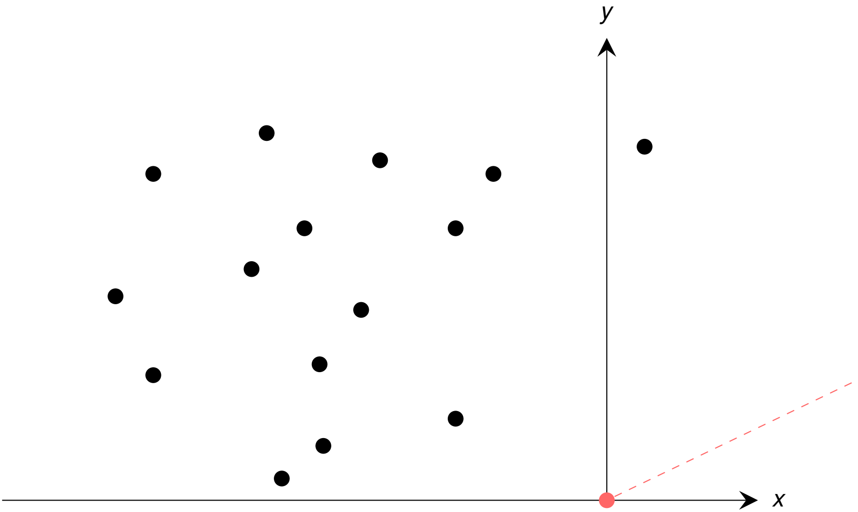
Execution of Jarvis' March



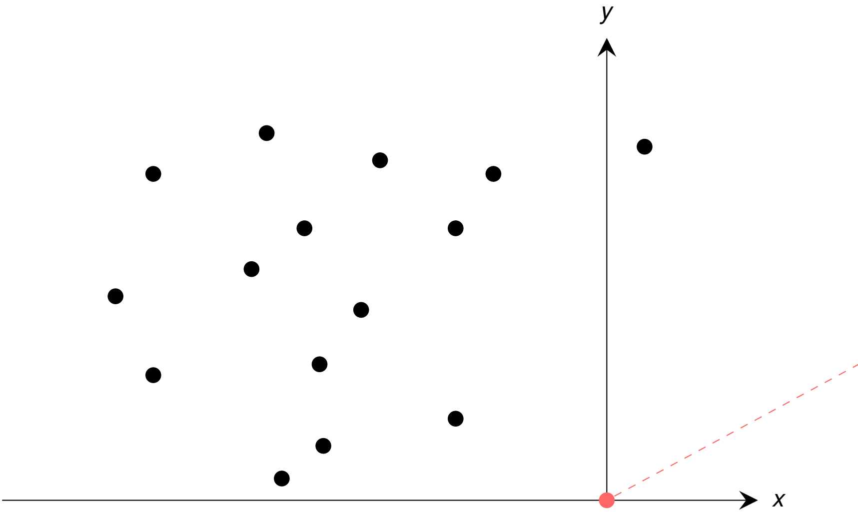
Execution of Jarvis' March



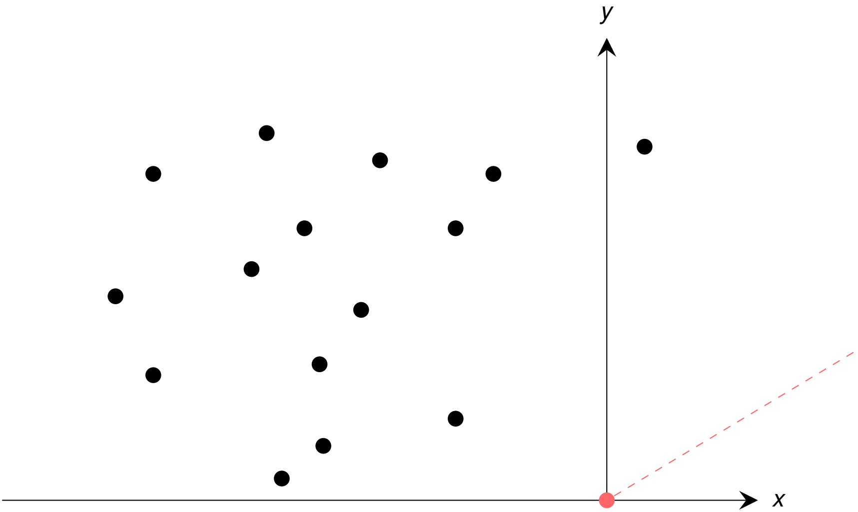
Execution of Jarvis' March



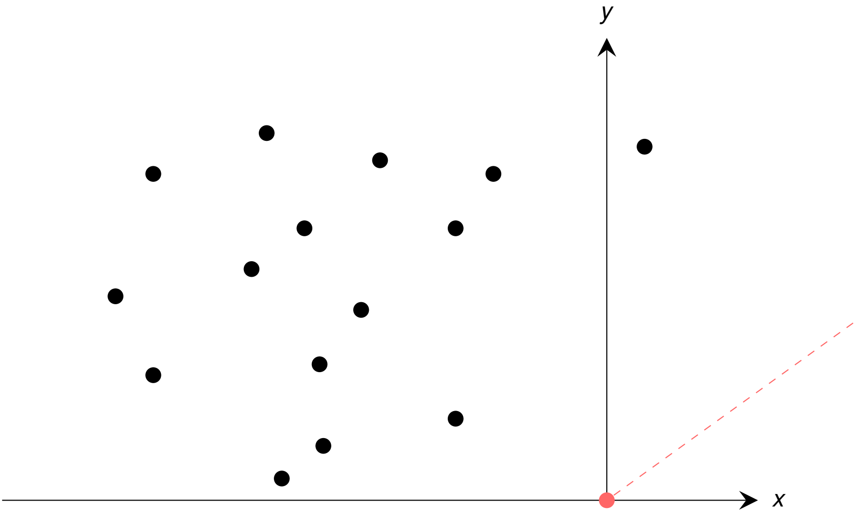
Execution of Jarvis' March



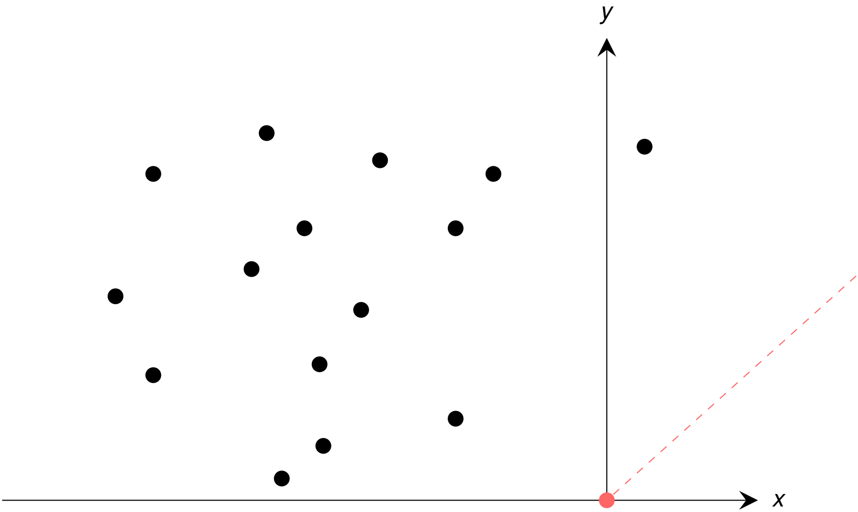
Execution of Jarvis' March



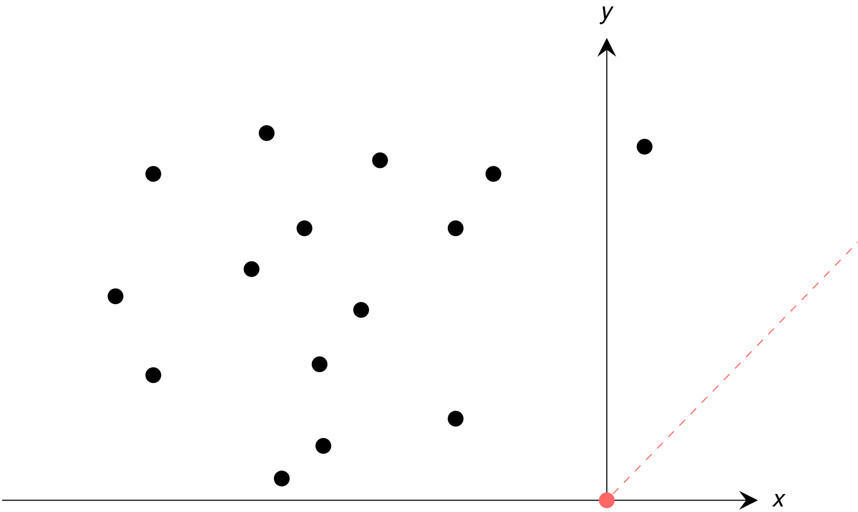
Execution of Jarvis' March



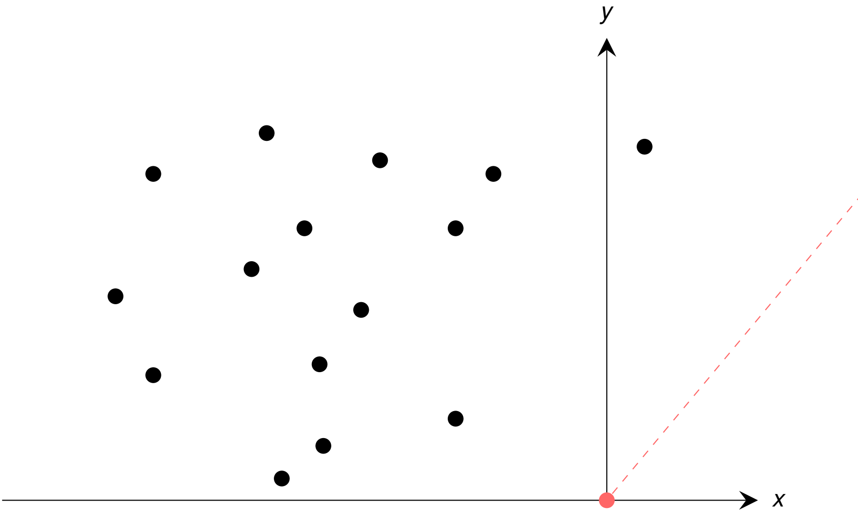
Execution of Jarvis' March



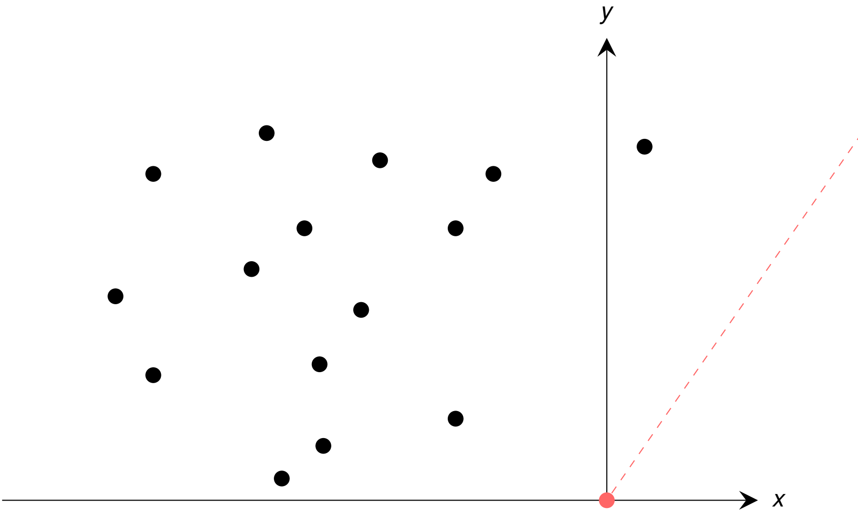
Execution of Jarvis' March



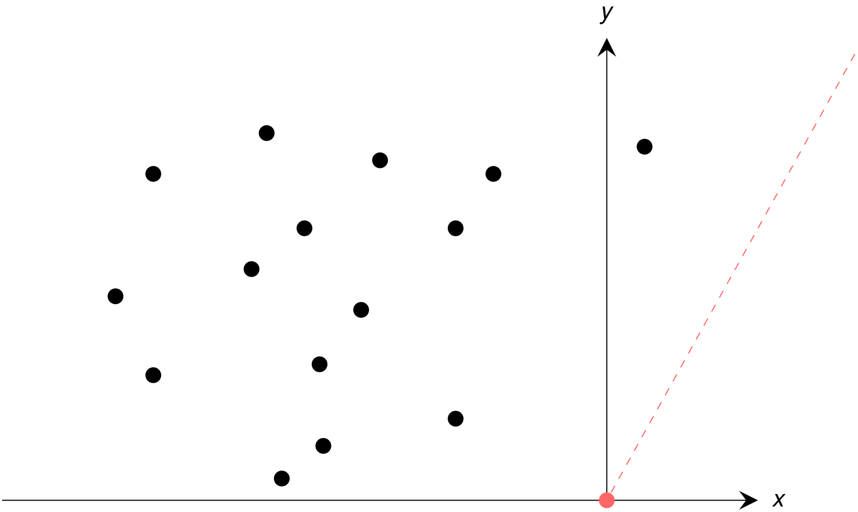
Execution of Jarvis' March



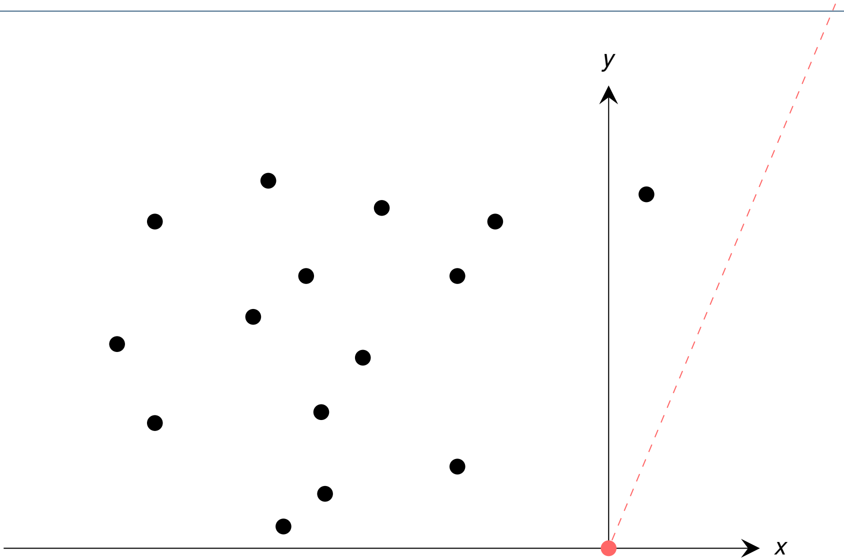
Execution of Jarvis' March



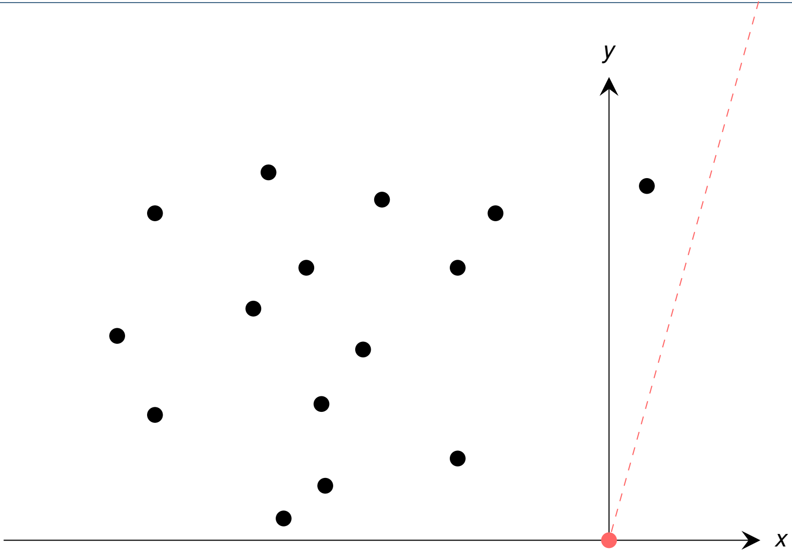
Execution of Jarvis' March



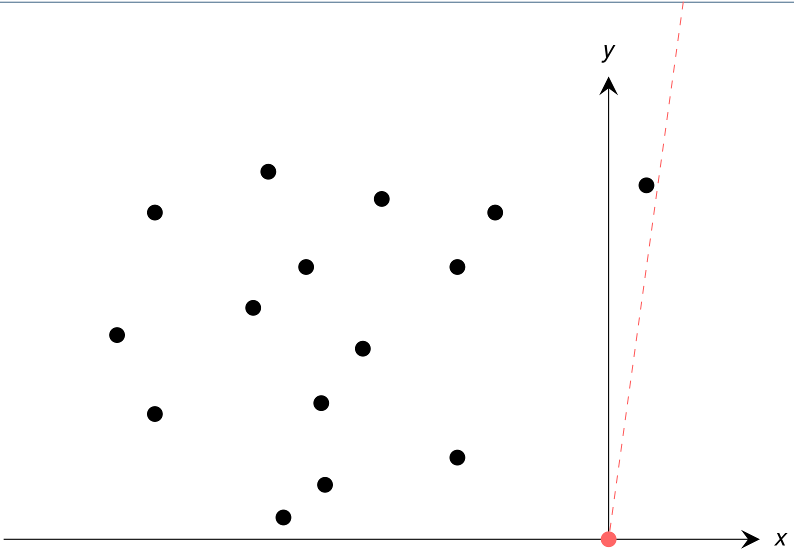
Execution of Jarvis' March



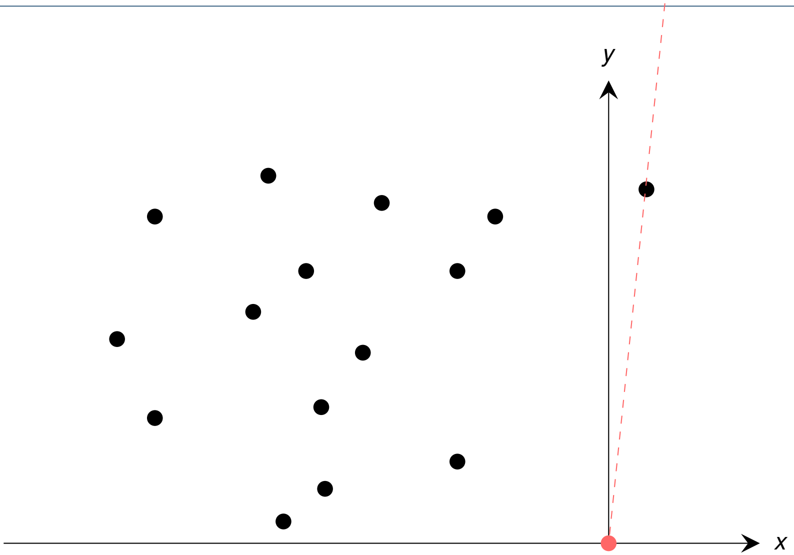
Execution of Jarvis' March



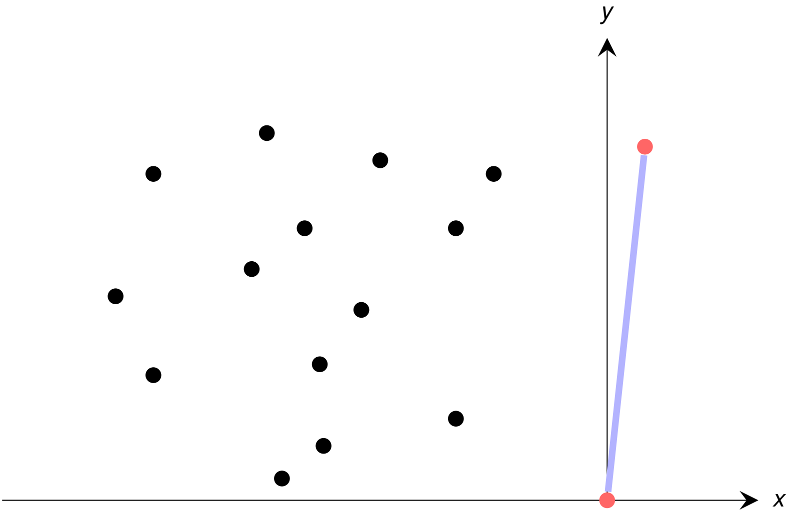
Execution of Jarvis' March



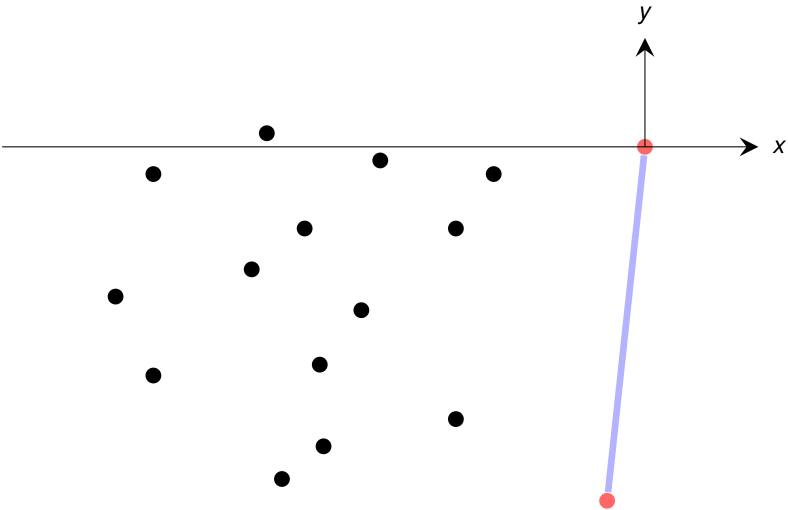
Execution of Jarvis' March



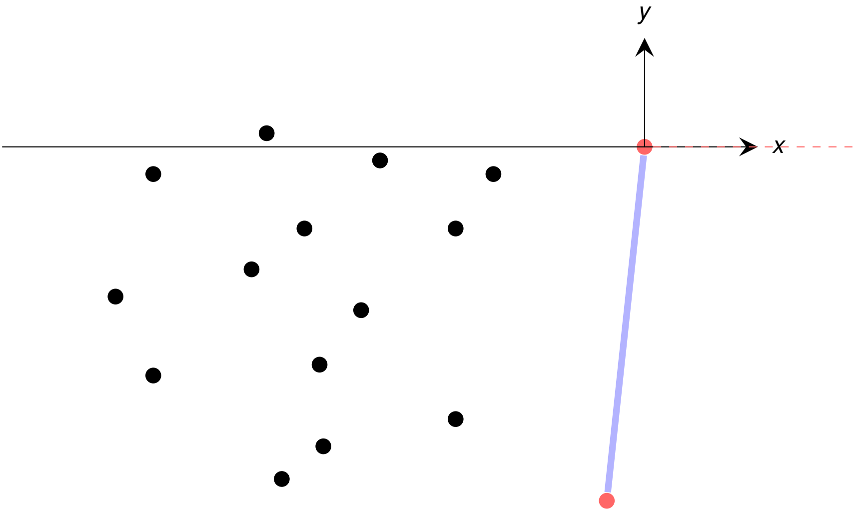
Execution of Jarvis' March



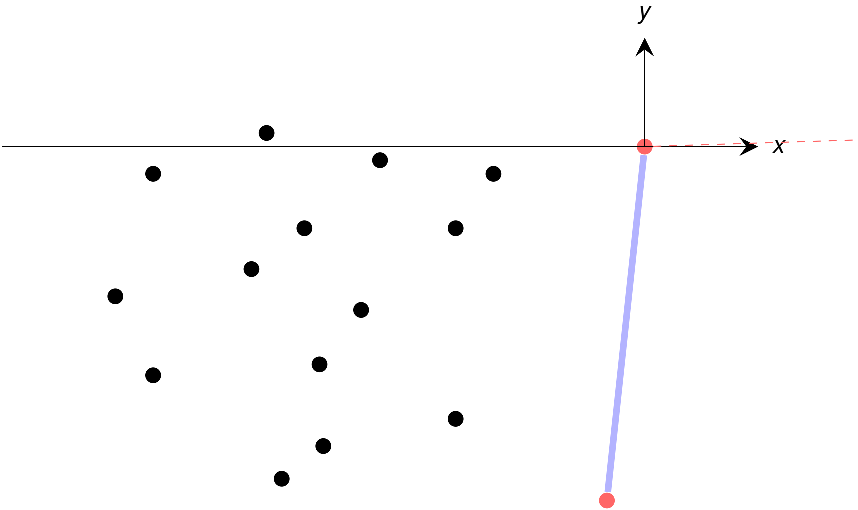
Execution of Jarvis' March



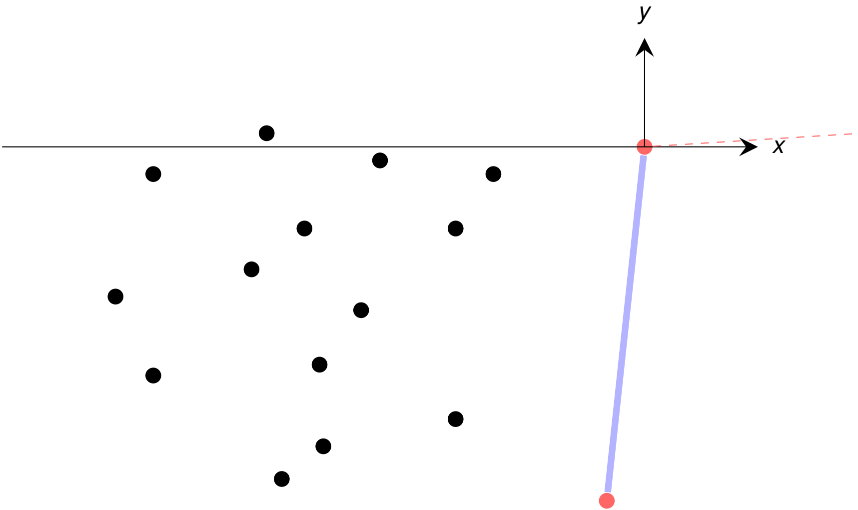
Execution of Jarvis' March



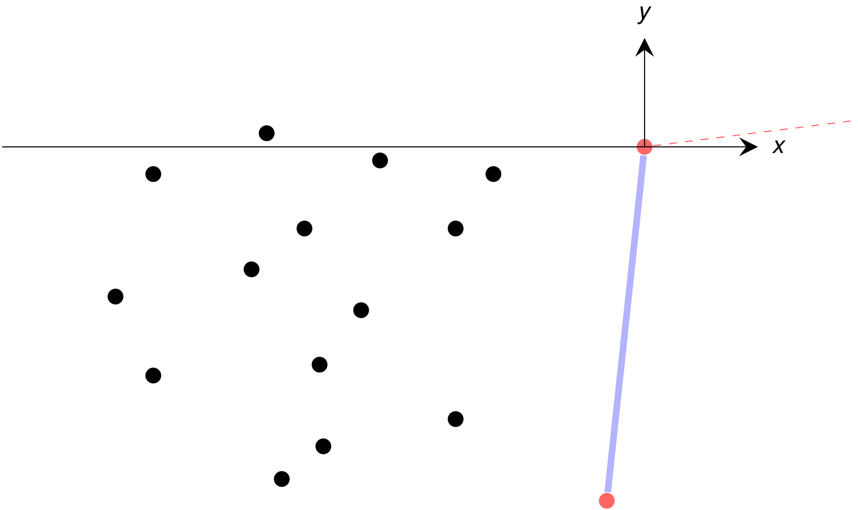
Execution of Jarvis' March



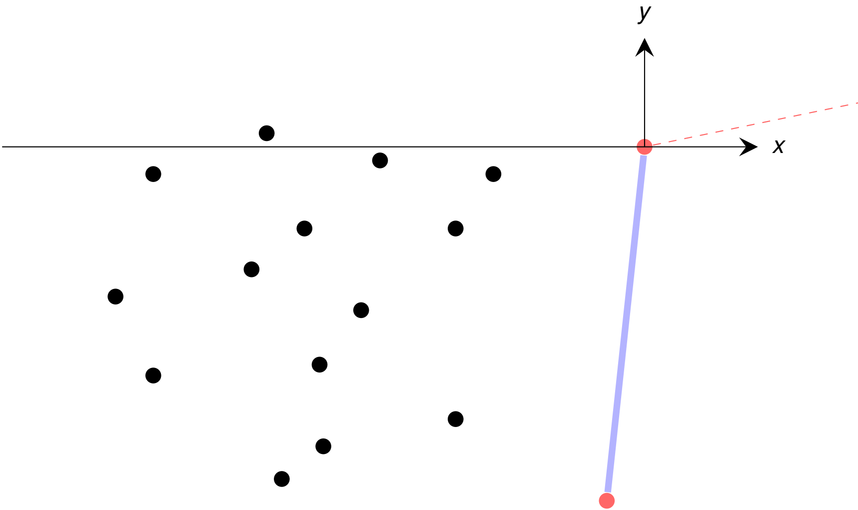
Execution of Jarvis' March



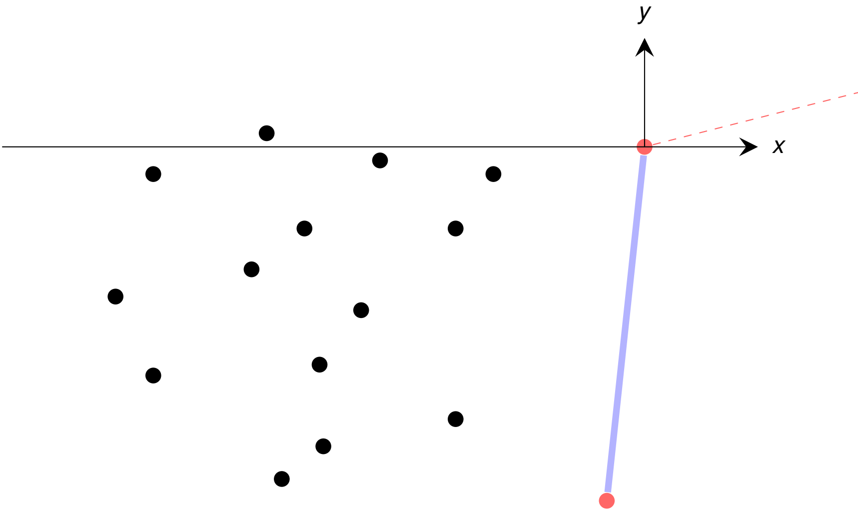
Execution of Jarvis' March



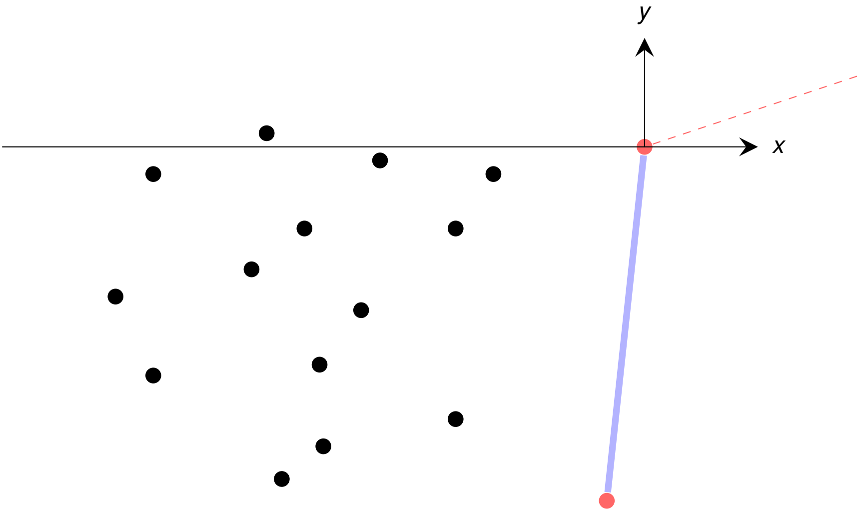
Execution of Jarvis' March



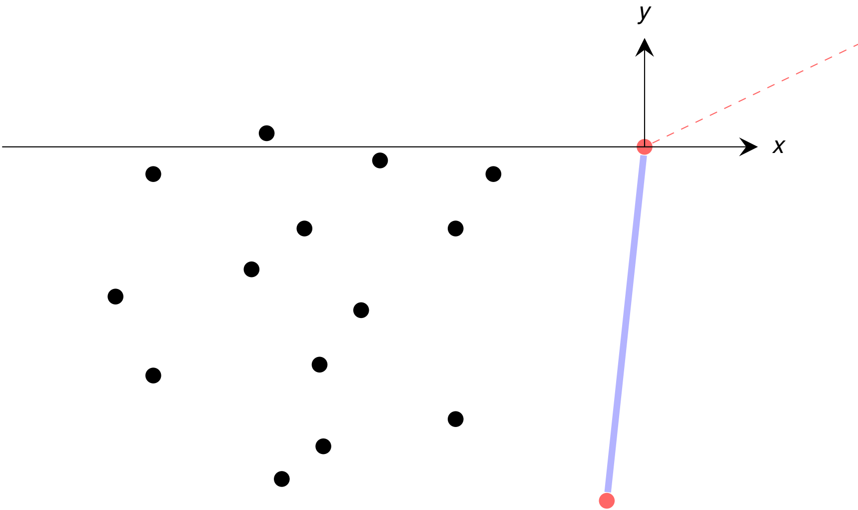
Execution of Jarvis' March



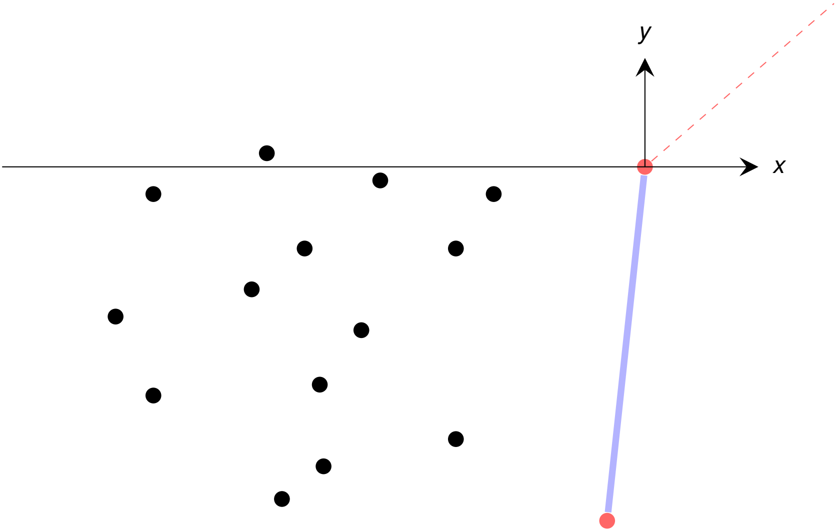
Execution of Jarvis' March



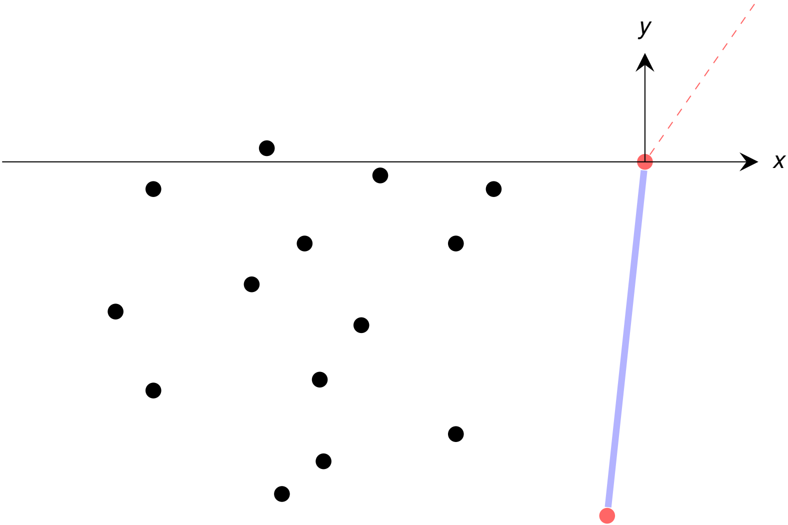
Execution of Jarvis' March



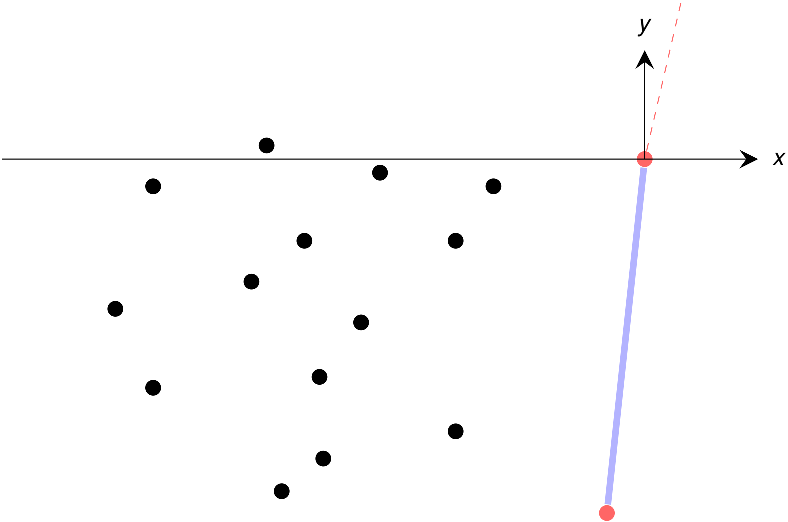
Execution of Jarvis' March



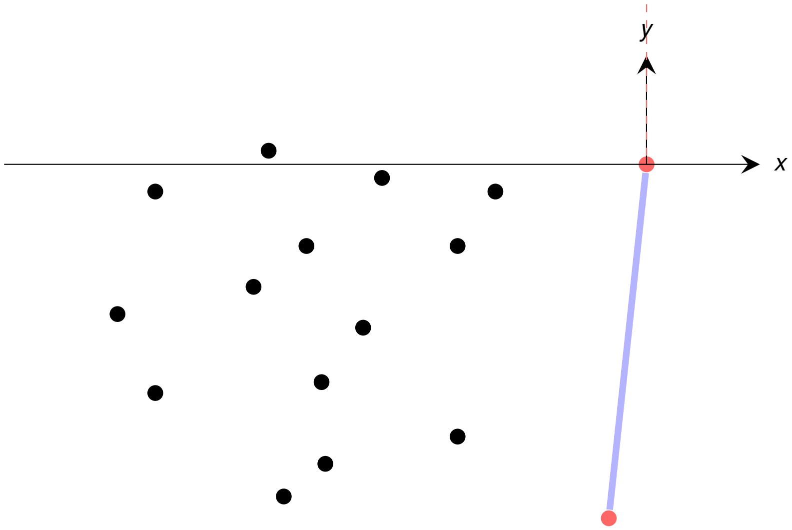
Execution of Jarvis' March



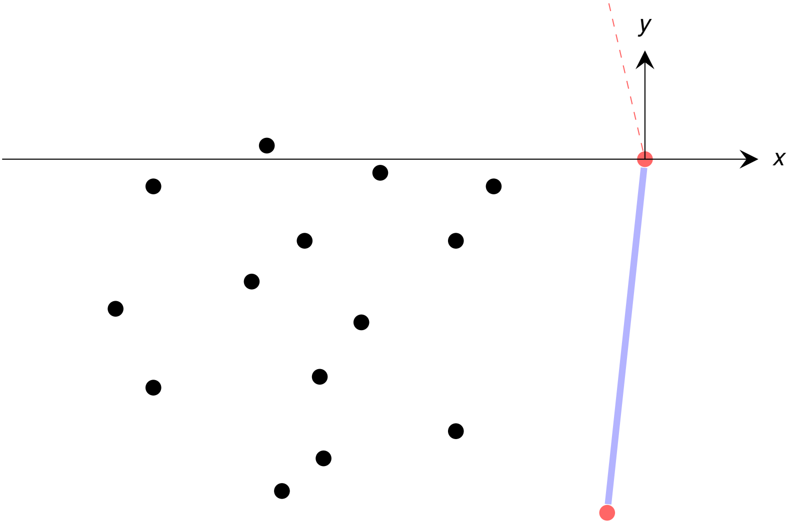
Execution of Jarvis' March



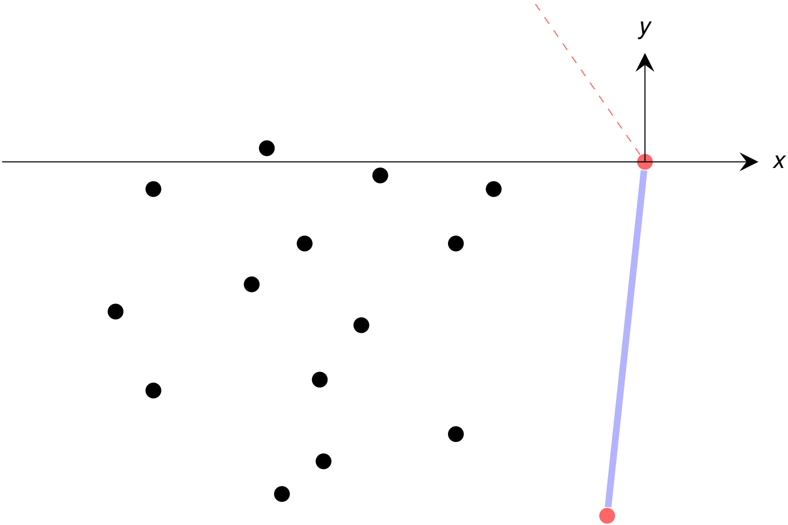
Execution of Jarvis' March



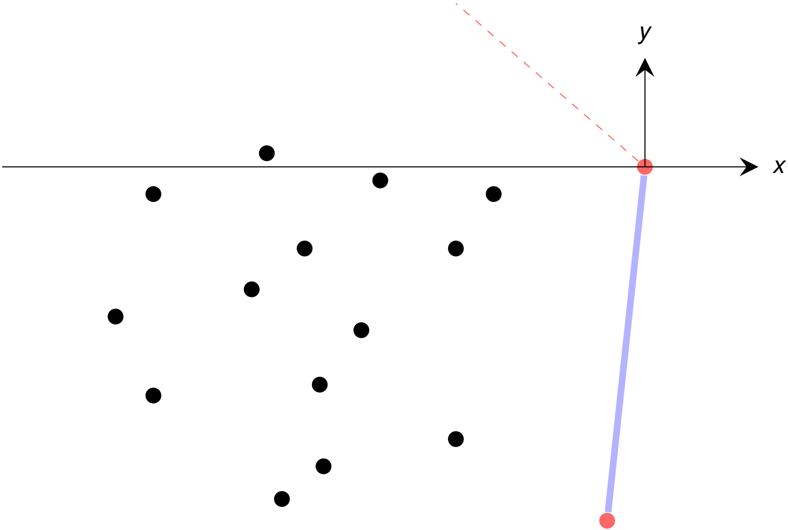
Execution of Jarvis' March



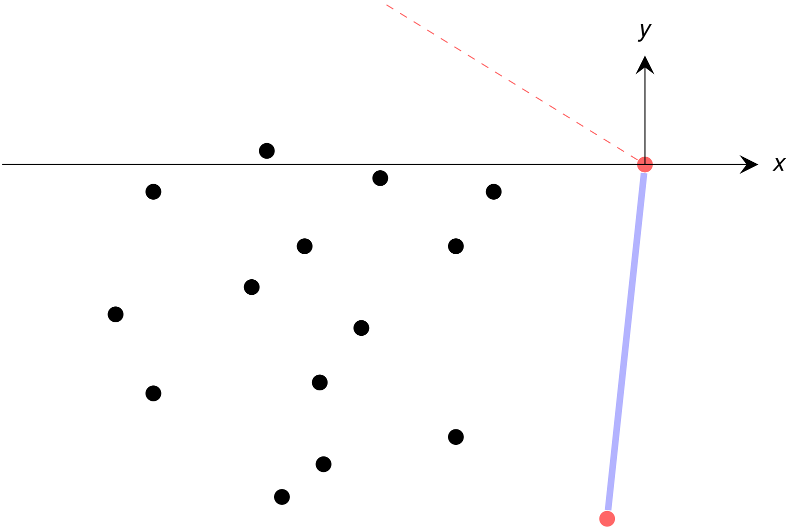
Execution of Jarvis' March



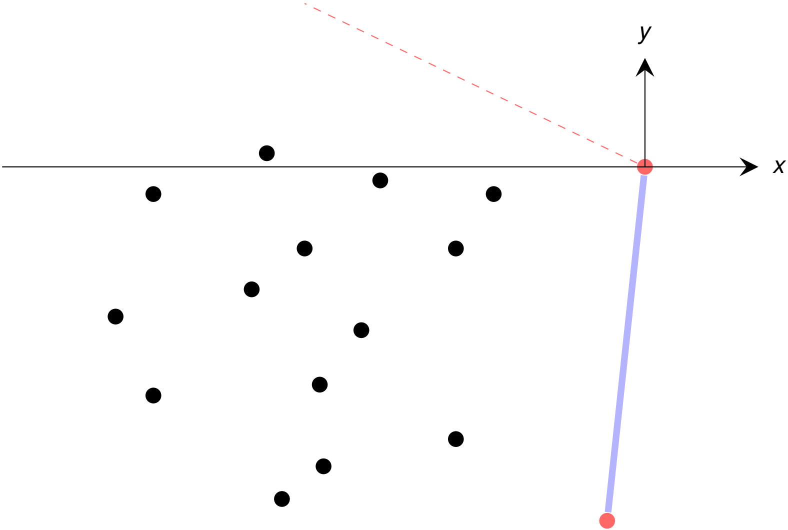
Execution of Jarvis' March



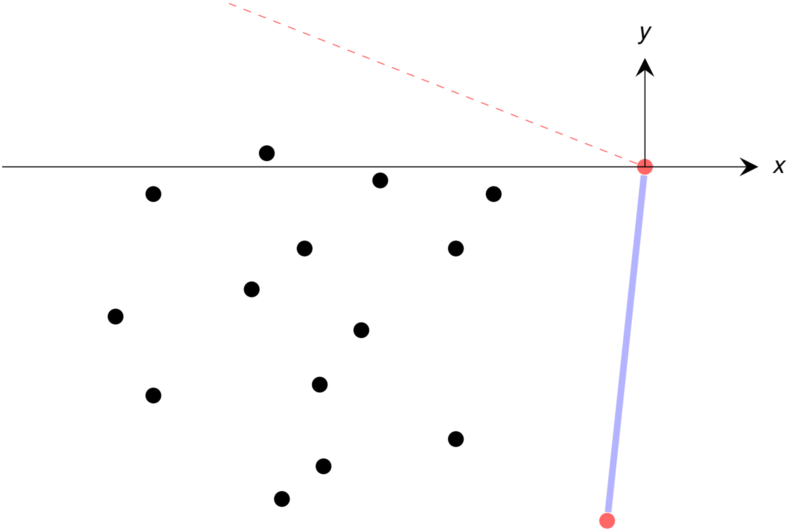
Execution of Jarvis' March



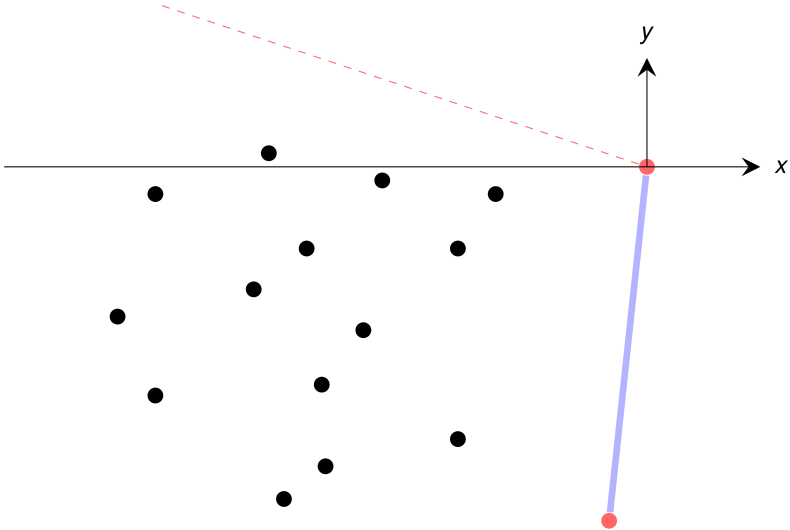
Execution of Jarvis' March



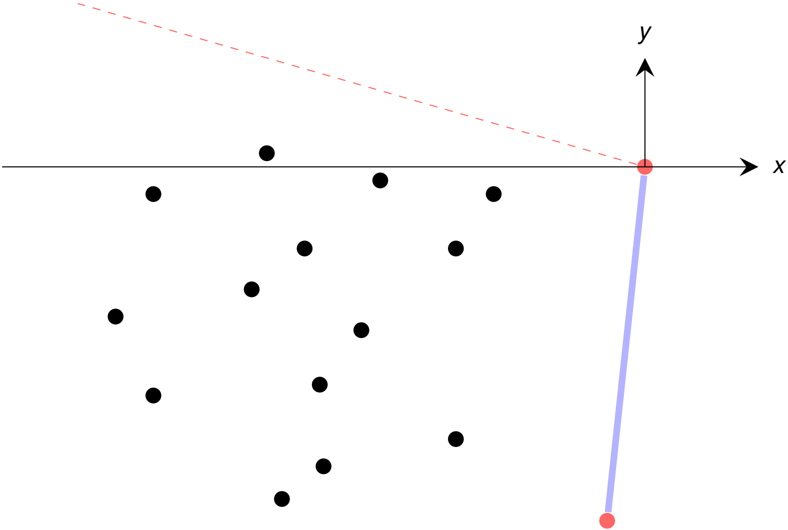
Execution of Jarvis' March



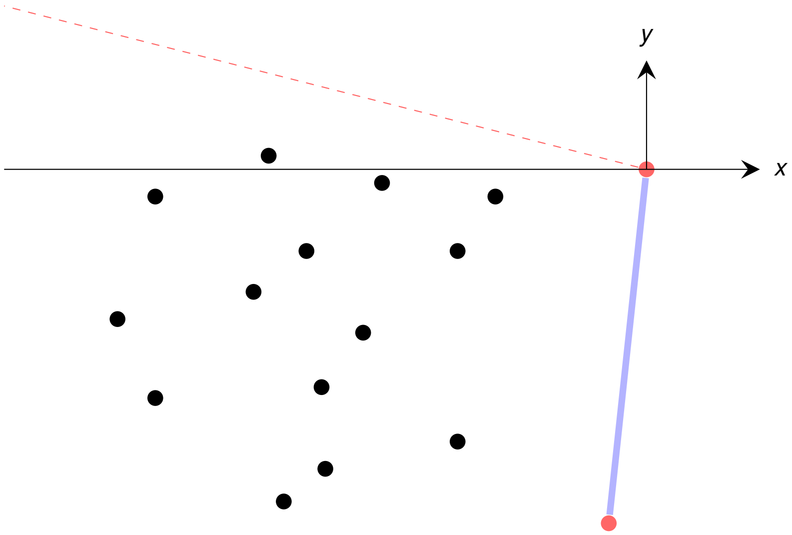
Execution of Jarvis' March



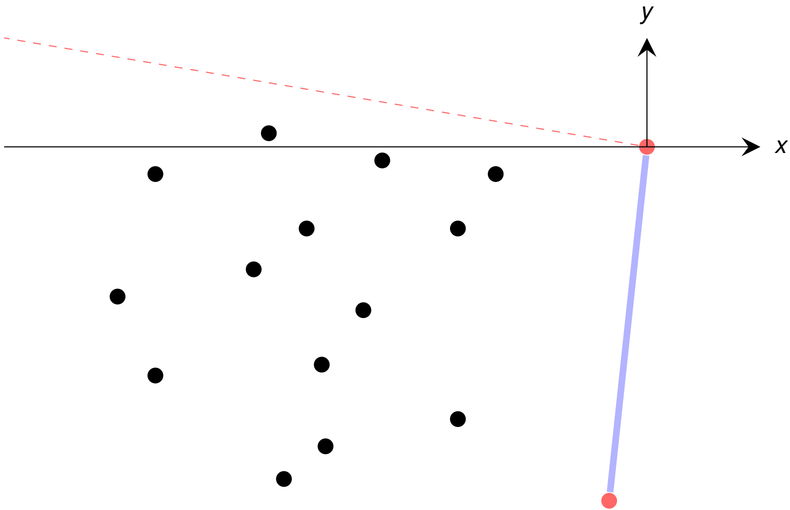
Execution of Jarvis' March



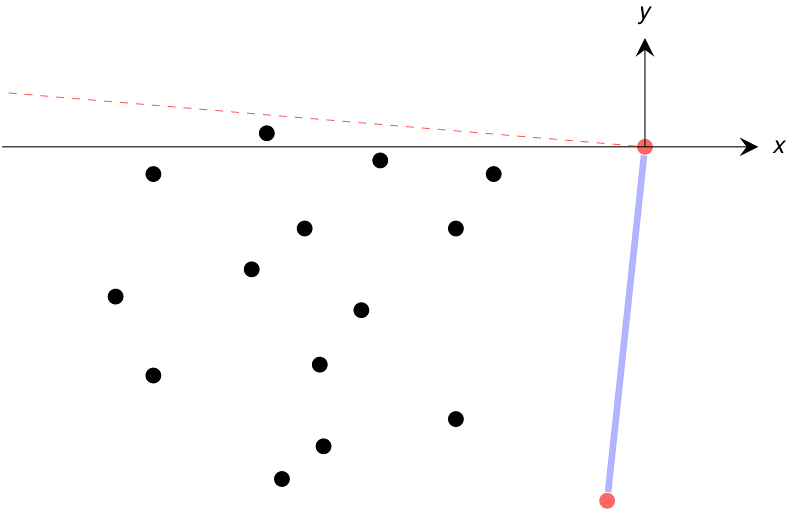
Execution of Jarvis' March



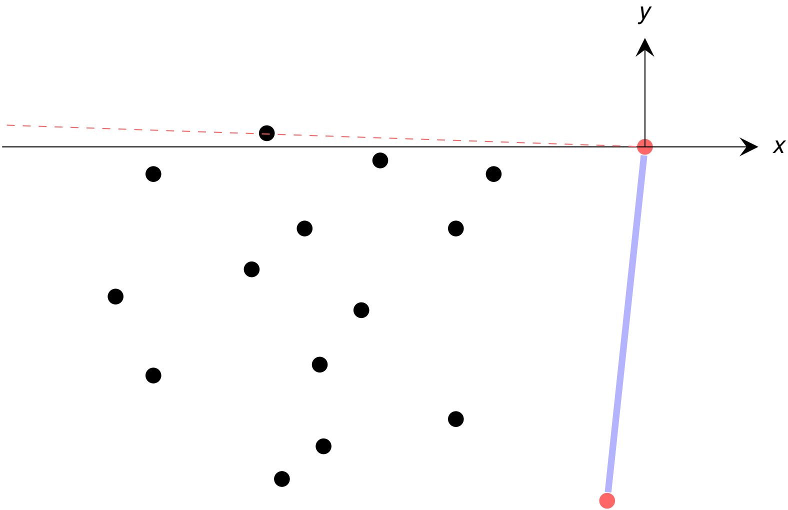
Execution of Jarvis' March



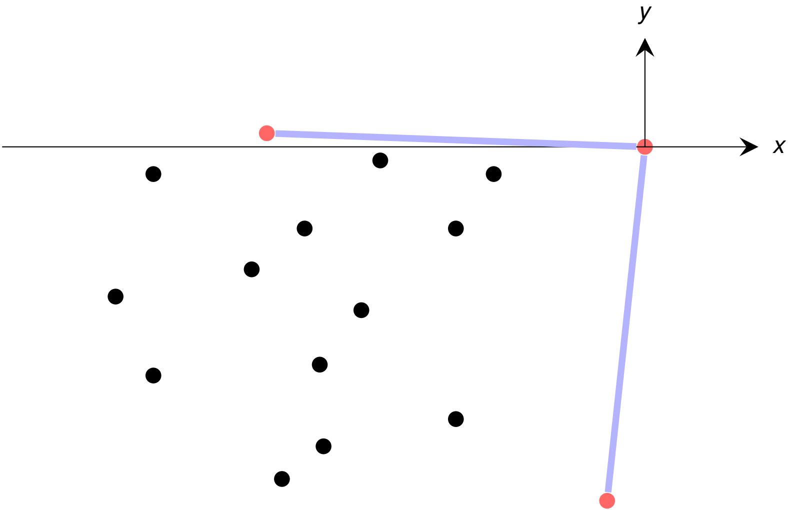
Execution of Jarvis' March



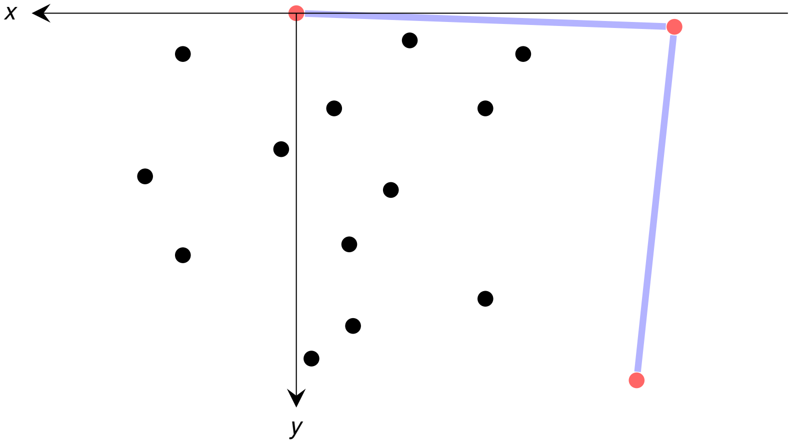
Execution of Jarvis' March



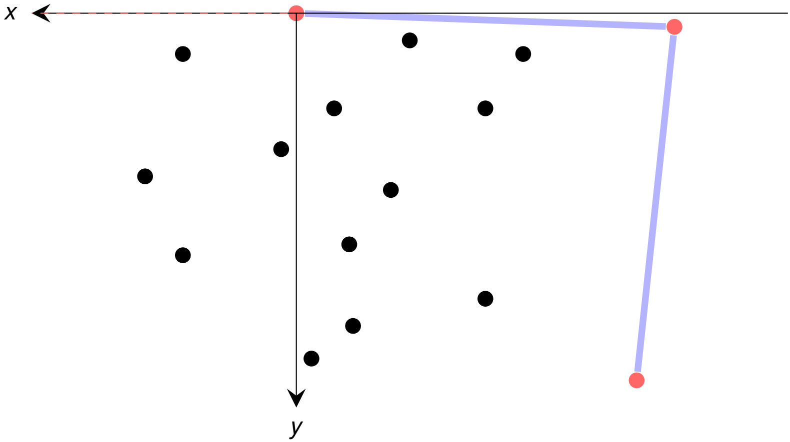
Execution of Jarvis' March



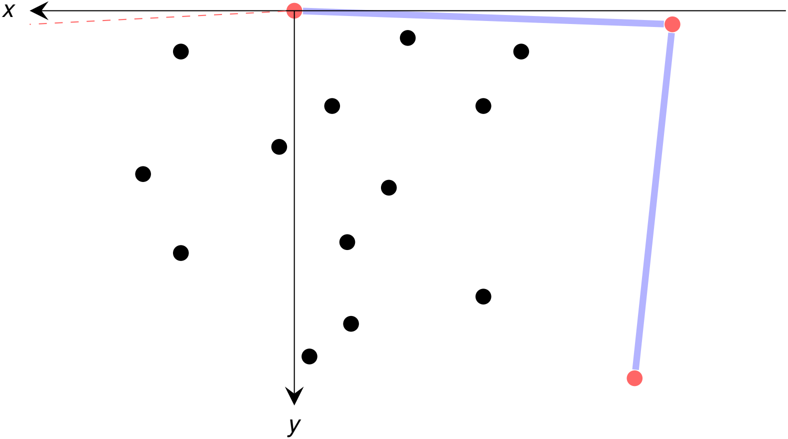
Execution of Jarvis' March



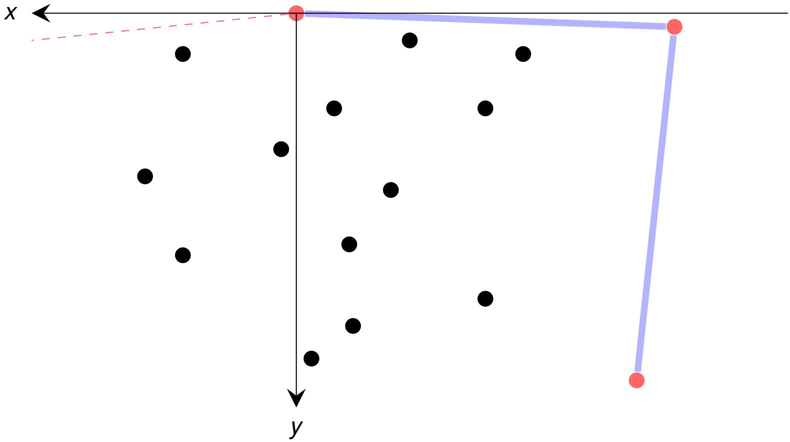
Execution of Jarvis' March



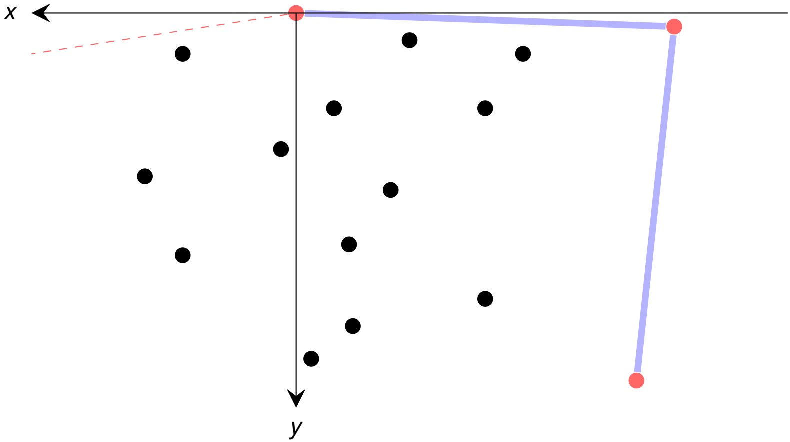
Execution of Jarvis' March



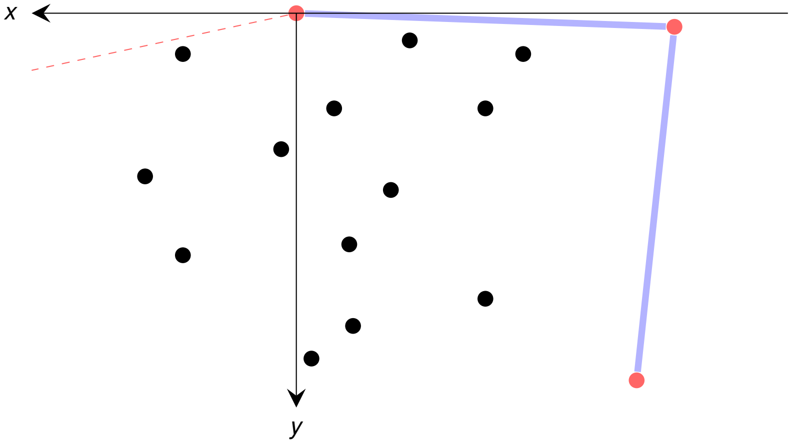
Execution of Jarvis' March



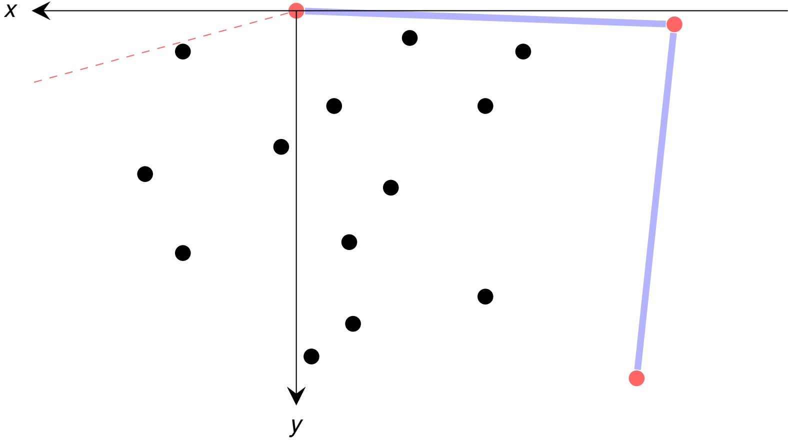
Execution of Jarvis' March



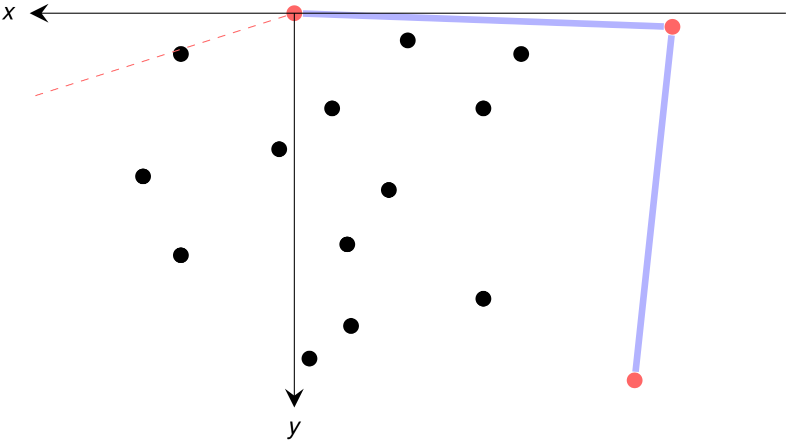
Execution of Jarvis' March



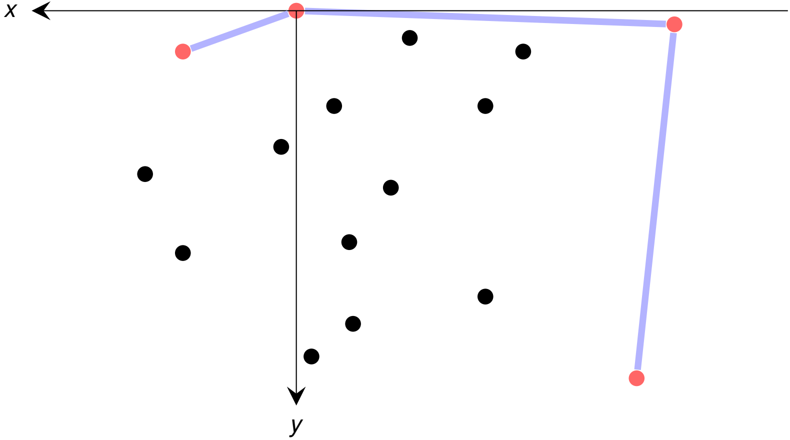
Execution of Jarvis' March



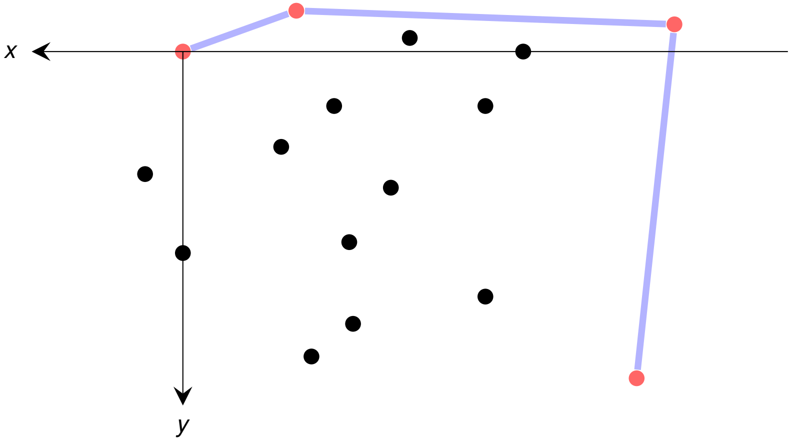
Execution of Jarvis' March



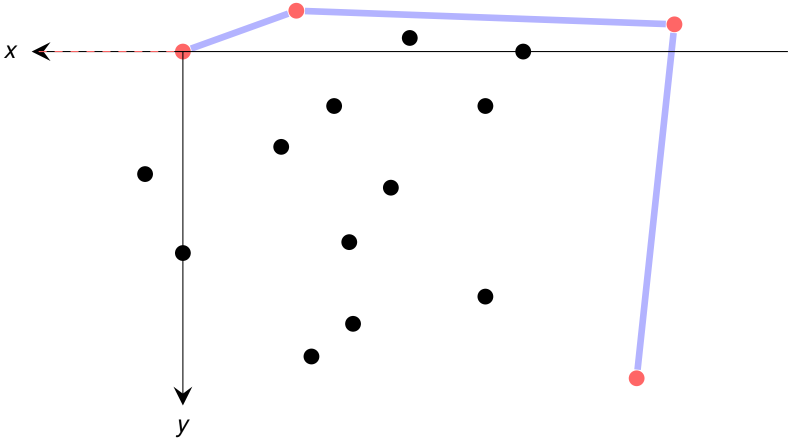
Execution of Jarvis' March



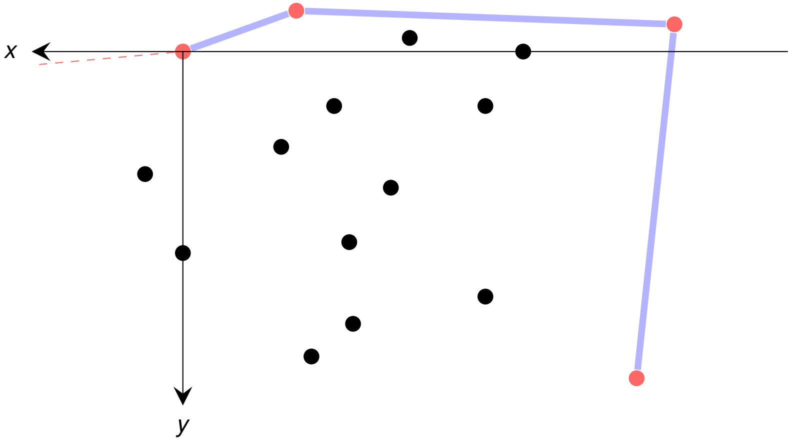
Execution of Jarvis' March



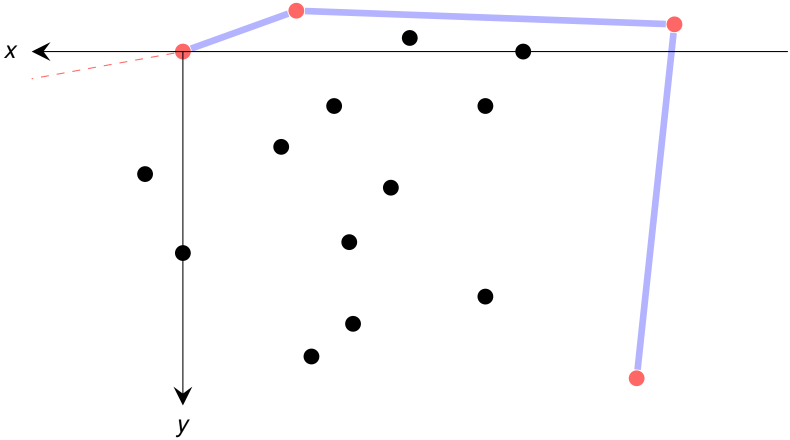
Execution of Jarvis' March



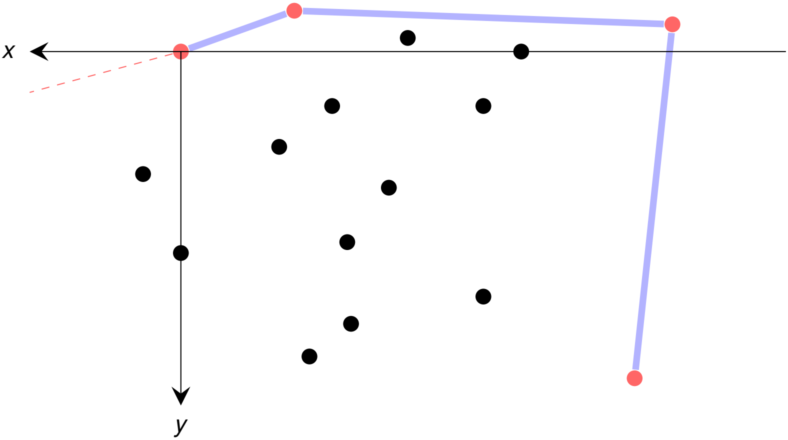
Execution of Jarvis' March



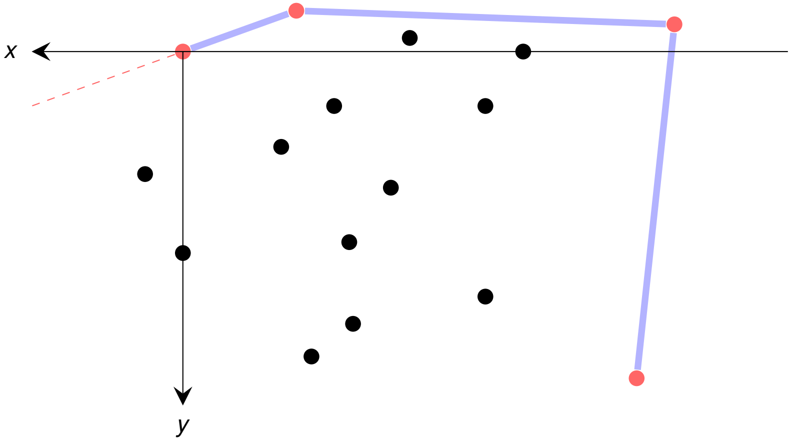
Execution of Jarvis' March



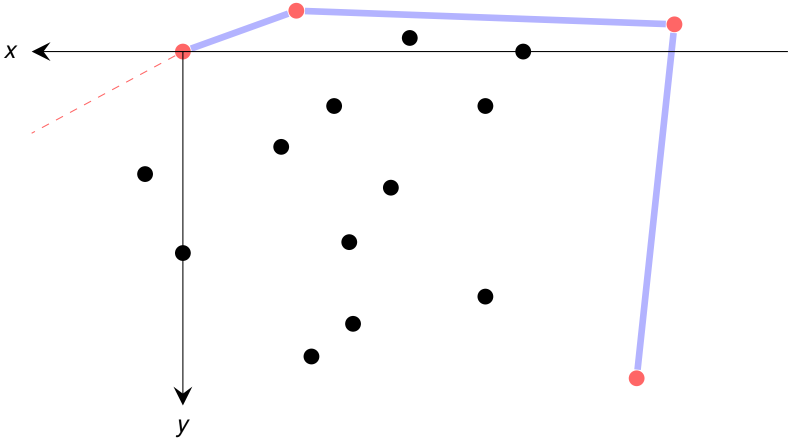
Execution of Jarvis' March



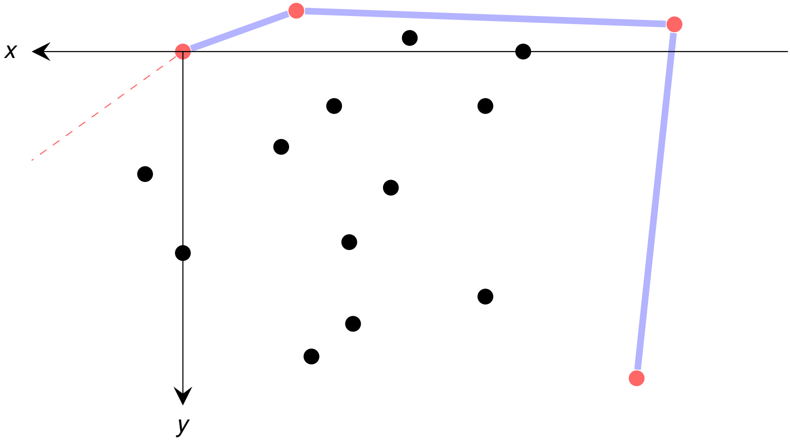
Execution of Jarvis’ March



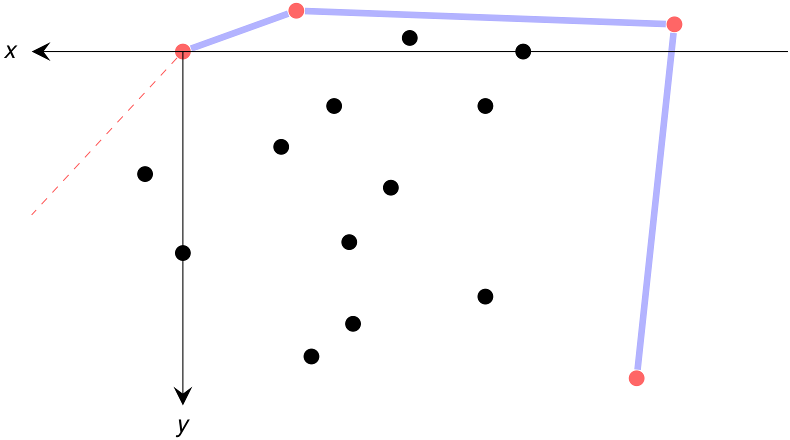
Execution of Jarvis' March



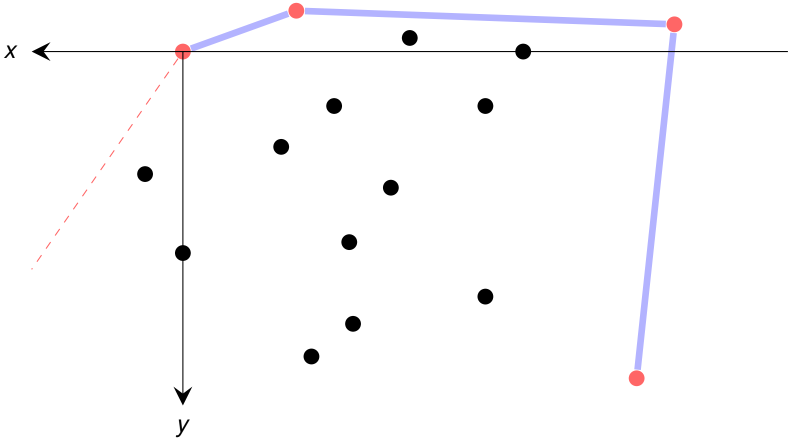
Execution of Jarvis' March



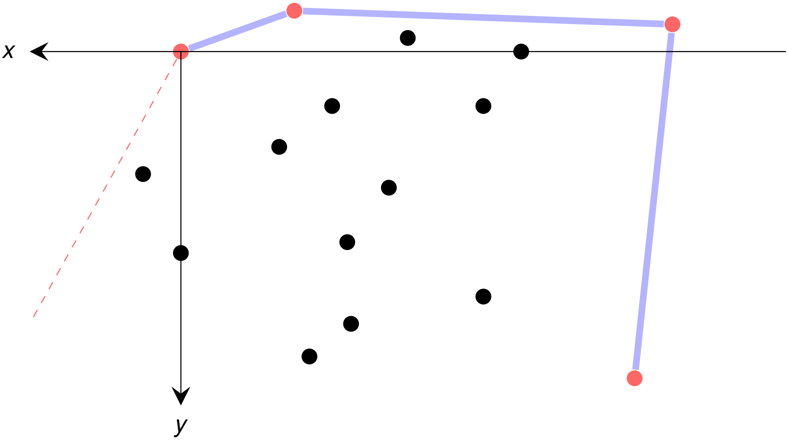
Execution of Jarvis' March



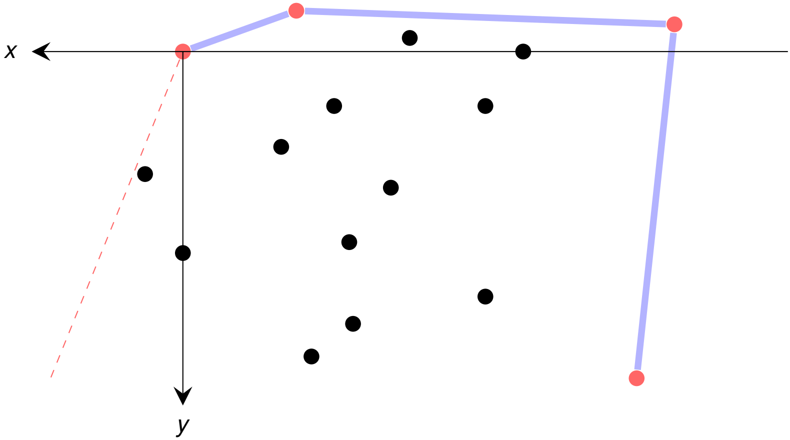
Execution of Jarvis' March



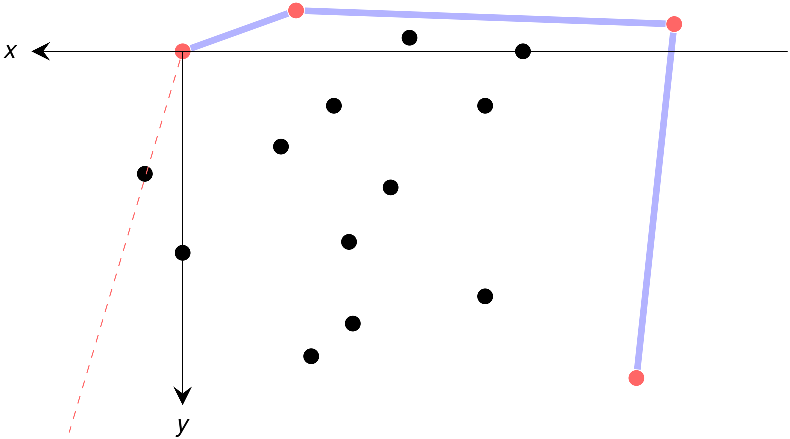
Execution of Jarvis' March



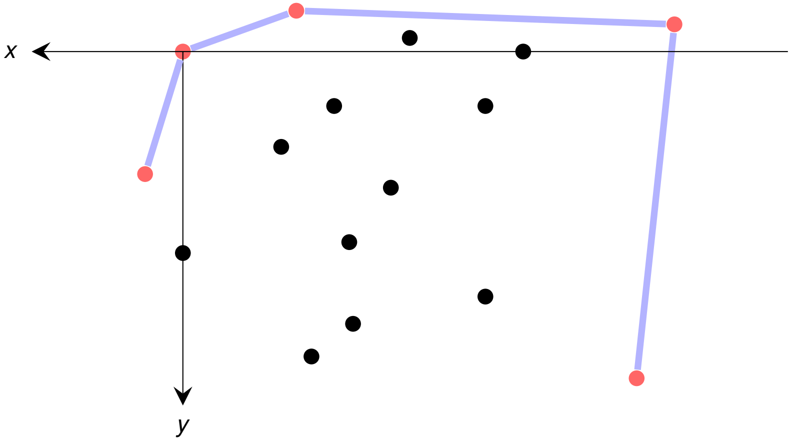
Execution of Jarvis' March



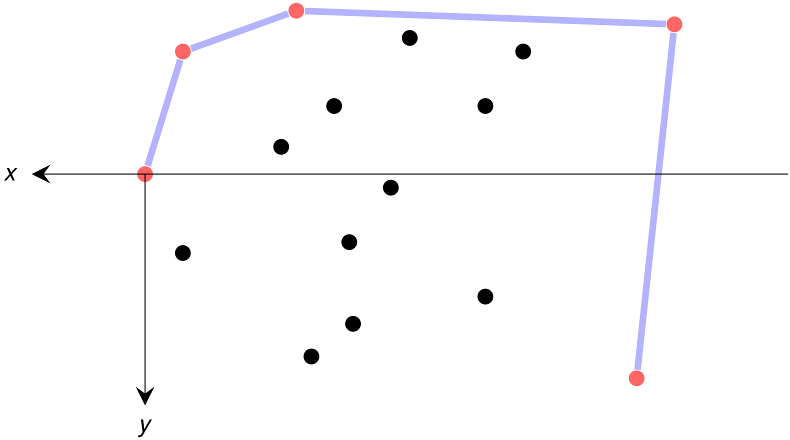
Execution of Jarvis' March



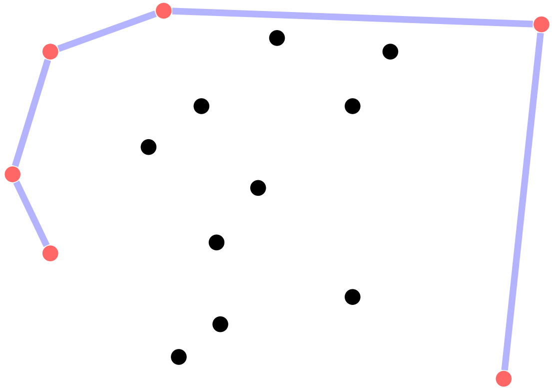
Execution of Jarvis' March



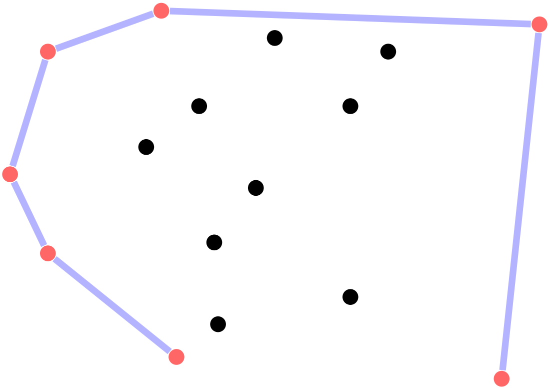
Execution of Jarvis' March



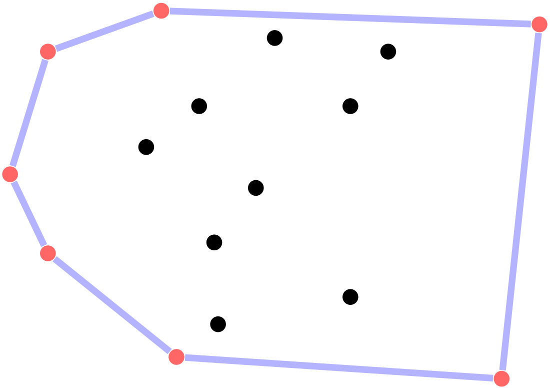
Execution of Jarvis' March



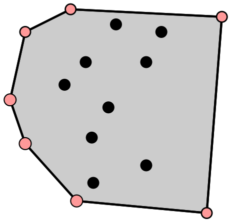
Execution of Jarvis' March



Execution of Jarvis' March

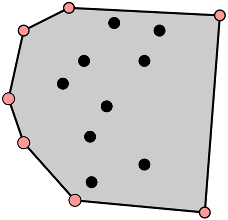


Computing Convex Hull: Summary



Computing Convex Hull: Summary

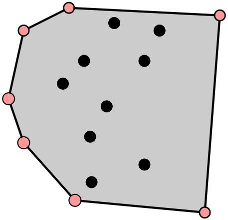
Graham's Scan



Computing Convex Hull: Summary

Graham's Scan

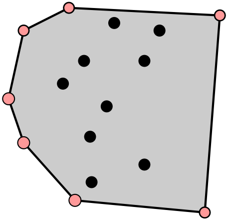
- natural backtracking algorithm



Computing Convex Hull: Summary

Graham's Scan

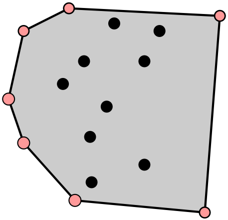
- natural backtracking algorithm
- cross-product avoids computing polar angles



Computing Convex Hull: Summary

Graham's Scan

- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

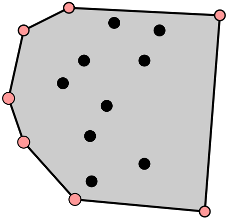


Computing Convex Hull: Summary

Graham's Scan

- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

Jarvis' March



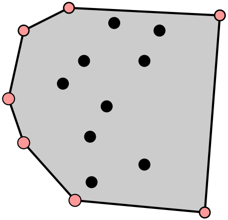
Computing Convex Hull: Summary

Graham's Scan

- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

Jarvis' March

- proceeds like wrapping a gift



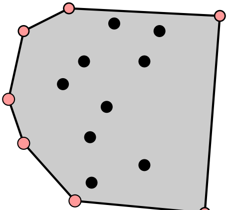
Computing Convex Hull: Summary

Graham's Scan

- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

Jarvis' March

- proceeds like wrapping a gift
- Runtime $O(nh) \rightsquigarrow$ output-sensitive



Computing Convex Hull: Summary

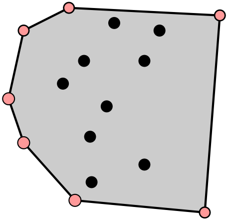
Graham's Scan

- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

Jarvis' March

- proceeds like wrapping a gift
- Runtime $O(nh) \rightsquigarrow$ output-sensitive

Improves Graham's scan only if $h = O(\log n)$



Computing Convex Hull: Summary

Graham's Scan

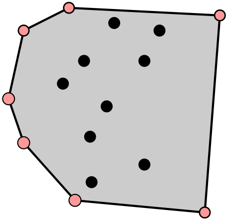
- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

Jarvis' March

- proceeds like wrapping a gift
- Runtime $O(nh) \rightsquigarrow$ output-sensitive

Improves Graham's scan only if $h = O(\log n)$

There exists an algorithm with $O(n \log h)$ runtime!



Computing Convex Hull: Summary

Graham's Scan

- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

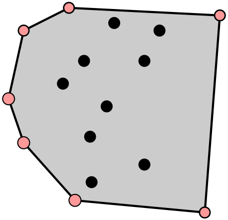
Jarvis' March

- proceeds like wrapping a gift
- Runtime $O(nh) \rightsquigarrow$ output-sensitive

Improves Graham's scan only if $h = O(\log n)$

There exists an algorithm with $O(n \log h)$ runtime!

Lessons Learned



Computing Convex Hull: Summary

Graham's Scan

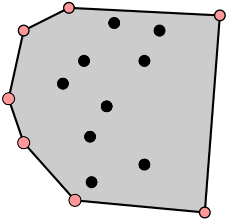
- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

Jarvis' March

- proceeds like wrapping a gift
- Runtime $O(nh) \rightsquigarrow$ output-sensitive

Improves Graham's scan only if $h = O(\log n)$

There exists an algorithm with $O(n \log h)$ runtime!



Lessons Learned

- cross product very powerful tool
  (avoids trigonometry and divison!)

Computing Convex Hull: Summary

Graham's Scan

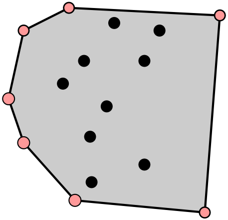
- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\rightsquigarrow O(n \log n)$

Jarvis' March

- proceeds like wrapping a gift
- Runtime $O(nh) \rightsquigarrow$ output-sensitive

Improves Graham's scan only if $h = O(\log n)$

There exists an algorithm with $O(n \log h)$ runtime!



Lessons Learned

- cross product very powerful tool
  (avoids trigonometry and divison!)
- take care of degenerate cases

Outline

Introduction and Line Intersection

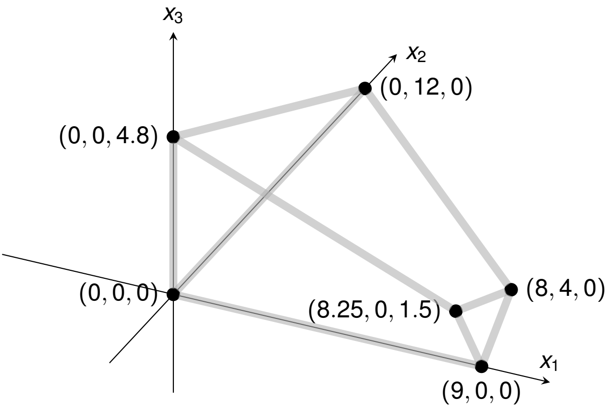
Convex Hull

Glimpse at (More) Advanced Algorithms

$$
\begin{array}{llcccccc}
\text{maximize} & 3x_1 & + & x_2 & + & 2x_3 & & \\
\text{subject to} & & & & & & & \\
& x_1 & + & x_2 & + & 3x_3 & \leq & 30 \\
& 2x_1 & + & 2x_2 & + & 5x_3 & \leq & 24 \\
& 4x_1 & + & x_2 & + & 2x_3 & \leq & 36 \\
& & & x_1, x_2, x_3 & & & \geq & 0
\end{array}
$$

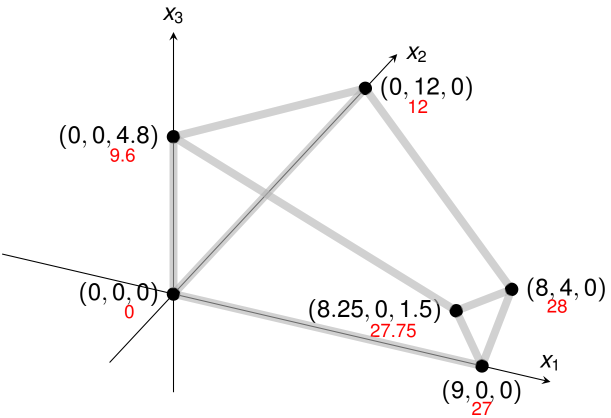
Go to End

Linear Programming and Simplex



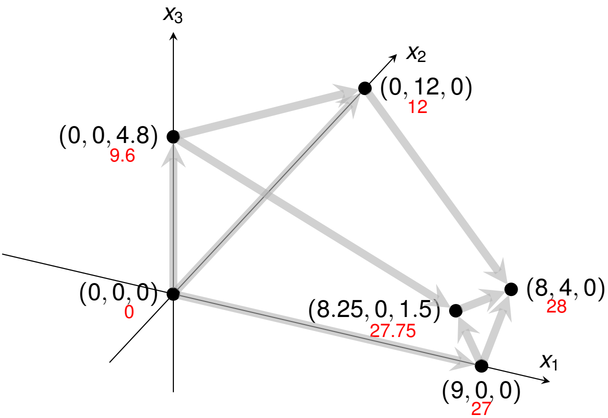
$$
\begin{array}{llllllll}
\text{maximize} & 3x_1 & + & x_2 & + & 2x_3 & & \\
\text{subject to} & & & & & & & \\
 & x_1 & + & x_2 & + & 3x_3 & \leq & 30 \\
 & 2x_1 & + & 2x_2 & + & 5x_3 & \leq & 24 \\
 & 4x_1 & + & x_2 & + & 2x_3 & \leq & 36 \\
 & & & x_1, x_2, x_3 & & & \geq & 0
\end{array}
$$

Go to End



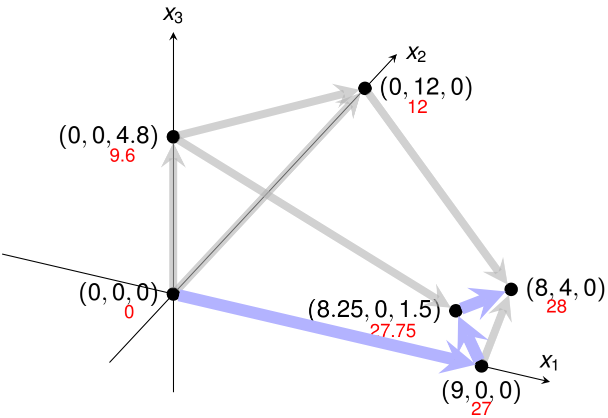
$$
\begin{array}{llllllll}
\text{maximize} & 3x_1 & + & x_2 & + & 2x_3 & & \\
\text{subject to} & & & & & & & \\
 & x_1 & + & x_2 & + & 3x_3 & \leq & 30 \\
 & 2x_1 & + & 2x_2 & + & 5x_3 & \leq & 24 \\
 & 4x_1 & + & x_2 & + & 2x_3 & \leq & 36 \\
 & & & x_1, x_2, x_3 & & & \geq & 0
\end{array}
$$

Go to End



$$\begin{array}{lrcrcrcr}
\text{maximize} & 3x_1 & + & x_2 & + & 2x_3 & & \\
\text{subject to} & & & & & & & \\
& x_1 & + & x_2 & + & 3x_3 & \leq & 30 \\
& 2x_1 & + & 2x_2 & + & 5x_3 & \leq & 24 \\
& 4x_1 & + & x_2 & + & 2x_3 & \leq & 36 \\
& & & x_1, x_2, x_3 & & & \geq & 0
\end{array}$$

Go to End



maximize	$3x_1$	$+$	x_2	$+$	$2x_3$			
subject to								
	x_1	$+$	x_2	$+$	$3x_3$	$\leq$	30
	$2x_1$	$+$	$2x_2$	$+$	$5x_3$	$\leq$	24
	$4x_1$	$+$	x_2	$+$	$2x_3$	$\leq$	36
			x_1, x_2, x_3			$\geq$	0

Go to End

The Original Article (1954)

SOLUTION OF A LARGE-SCALE TRAVELING-SALESMAN PROBLEM*

G. DANTZIG, R. FULKERSON, AND S. JOHNSON

The Rand Corporation, Santa Monica, California

(Received August 9, 1954)

It is shown that a certain tour of 49 cities, one in each of the 48 states and Washington, D. C., has the shortest road distance.

THE TRAVELING-SALESMAN PROBLEM might be described as follows: Find the shortest route (tour) for a salesman starting from a given city, visiting each of a specified group of cities, and then returning to the original point of departure. More generally, given an n by n symmetric matrix $D=(d_{IJ})$, where d_{IJ} represents the 'distance' from I to J, arrange the points in a cyclic order in such a way that the sum of the d_{IJ} between consecutive points is minimal. Since there are only a finite number of possibilities (at most $\frac{1}{2}(n-1)!$) to consider, the problem is to devise a method of picking out the optimal arrangement which is reasonably efficient for fairly large values of n. Although algorithms have been devised for problems of similar nature, e.g., the optimal assignment problem,[3,7,8] little is known about the traveling-salesman problem. We do not claim that this note alters the situation very much; what we shall do is outline a way of approaching the problem that sometimes, at least, enables one to find an optimal path and prove it so. In particular, it will be shown that a certain arrangement of 49 cities, one in each of the 48 states and Washington, D. C., is best, the d_{IJ} used representing road distances as taken from an atlas.

Travelling Salesman Problem: The 42 (49) Cities

1. Manchester, N. H.
2. Montpelier, Vt.
3. Detroit, Mich.
4. Cleveland, Ohio
5. Charleston, W. Va.
6. Louisville, Ky.
7. Indianapolis, Ind.
8. Chicago, Ill.
9. Milwaukee, Wis.
10. Minneapolis, Minn.
11. Pierre, S. D.
12. Bismarck, N. D.
13. Helena, Mont.
14. Seattle, Wash.
15. Portland, Ore.
16. Boise, Idaho
17. Salt Lake City, Utah
18. Carson City, Nev.
19. Los Angeles, Calif.
20. Phoenix, Ariz.
21. Santa Fe, N. M.
22. Denver, Colo.
23. Cheyenne, Wyo.
24. Omaha, Neb.
25. Des Moines, Iowa
26. Kansas City, Mo.
27. Topeka, Kans.
28. Oklahoma City, Okla.
29. Dallas, Tex.
30. Little Rock, Ark.
31. Memphis, Tenn.
32. Jackson, Miss.
33. New Orleans, La.
34. Birmingham, Ala.
35. Atlanta, Ga.
36. Jacksonville, Fla.
37. Columbia, S. C.
38. Raleigh, N. C.
39. Richmond, Va.
40. Washington, D. C.
41. Boston, Mass.
42. Portland, Me.

A. Baltimore, Md.
B. Wilmington, Del.
C. Philadelphia, Penn.
D. Newark, N. J.
E. New York, N. Y.
F. Hartford, Conn.
G. Providence, R. I.

Road Distances

TABLE I

Road Distances between Cities in Adjusted Units

The figures in the table are mileages between the two specified numbered cities, less 11, divided by 17, and rounded to the nearest integer.

2	8																																								
3	39	45																																						
4	37	47	9																																					
5	50	49	21	15																																				
6	61	62	21	20	17																																			
7	58	60	16	17	18	6																																		
8	59	60	15	20	26	17	10																																	
9	62	66	20	25	31	22	15	5																																
10	81	81	40	44	50	41	35	24	20																															
11	103	107	62	67	72	63	57	46	41	23																														
12	108	117	66	71	77	68	61	51	46	26	11																													
13	145	149	104	108	114	106	99	88	84	63	49	40																												
14	181	185	140	144	150	142	135	124	120	99	85	76	35																											
15	187	191	146	150	156	142	137	130	125	105	90	81	41	10																										
16	161	170	120	124	130	115	110	104	105	90	72	64	34	31	27																									
17	142	146	101	104	111	97	91	85	86	75	51	59	29	53	48	21																								
18	174	178	133	138	143	129	123	117	118	107	83	84	54	46	35	26	31																							
19	185	186	142	143	140	130	126	124	128	118	93	101	72	69	58	58	43	26																						
20	164	165	120	123	124	106	106	105	110	104	86	97	71	93	82	62	42	45	22																					
21	137	139	94	96	94	80	78	77	84	77	56	64	65	90	87	58	36	68	50	30																				
22	117	122	77	80	83	68	62	60	61	50	34	42	49	82	77	60	30	62	70	49	21																			
23	114	118	73	78	84	69	63	57	59	48	28	36	43	77	72	45	27	59	69	55	27	5																		
24	85	89	44	48	53	41	34	28	29	22	23	35	69	105	102	74	56	88	99	81	54	32	29																	
25	77	80	36	40	46	34	27	19	21	14	29	40	77	114	111	84	64	96	107	87	60	40	37	8																
26	87	89	44	46	46	30	28	29	32	27	36	47	78	116	112	84	66	98	95	75	47	36	39	12	11															
27	91	93	48	50	48	34	32	33	36	30	34	45	77	115	110	83	63	97	91	72	44	32	36	9	15	3														
28	105	106	62	63	64	47	46	49	54	48	46	59	85	119	115	88	66	98	79	59	31	36	42	28	33	21	20													
29	111	113	69	71	66	51	53	56	61	57	59	71	96	130	126	98	75	98	85	62	38	47	53	39	42	29	30	12												
30	91	92	50	51	46	30	34	38	43	49	60	71	103	141	136	109	90	115	99	81	53	61	62	36	34	24	28	20	20											
31	83	85	42	43	38	22	26	32	36	51	63	75	106	142	140	112	93	126	108	88	60	64	66	39	36	27	31	28	28	8										
32	89	91	55	55	50	34	39	44	49	63	76	87	120	155	150	123	100	123	109	86	62	71	78	52	49	39	44	35	24	15	12									
33	95	97	64	63	56	42	49	56	60	75	86	97	126	160	155	128	104	128	113	90	67	76	82	62	59	49	53	40	29	25	23	11								
34	74	81	44	43	35	23	30	39	44	62	78	89	121	159	155	127	108	136	124	101	75	79	81	54	50	42	46	43	39	23	14	14	21							
35	67	69	42	41	31	25	32	41	46	64	83	90	130	164	160	133	114	146	134	111	85	84	86	59	52	47	51	53	49	32	24	24	30	9						
36	74	76	61	60	42	44	51	60	66	83	102	110	147	185	179	155	133	159	146	122	98	105	107	79	71	66	70	70	60	48	40	36	33	25	18					
37	57	59	46	41	25	30	36	47	52	71	93	98	136	172	172	148	126	158	147	124	121	97	99	71	65	59	63	67	62	46	38	37	43	23	13	17				
38	45	46	41	34	20	34	38	48	53	73	96	99	137	176	178	151	131	163	159	135	108	102	103	73	67	64	69	75	72	54	46	49	54	34	24	29	12			
39	35	37	35	26	18	34	36	46	51	70	93	97	134	171	176	151	129	161	163	139	118	102	101	71	65	65	70	84	78	58	50	56	62	41	32	38	21	9		
40	29	33	30	21	18	35	33	40	45	65	87	91	117	166	171	144	125	157	156	139	113	95	97	67	60	62	67	79	82	62	53	59	66	45	38	45	27	15	6	
41	3	11	41	37	47	57	55	58	63	83	105	109	147	186	188	164	144	176	182	161	134	119	116	86	78	84	88	101	108	88	80	86	92	71	64	71	54	41	32	25	
42	5	12	55	41	53	64	61	61	66	84	111	113	150	186	192	166	147	180	188	167	140	124	119	90	87	90	94	107	114	77	86	92	98	80	74	77	60	48	38	32	6
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**

The (Unique) Optimal Tour (699 Units ≈ 12,345 miles)

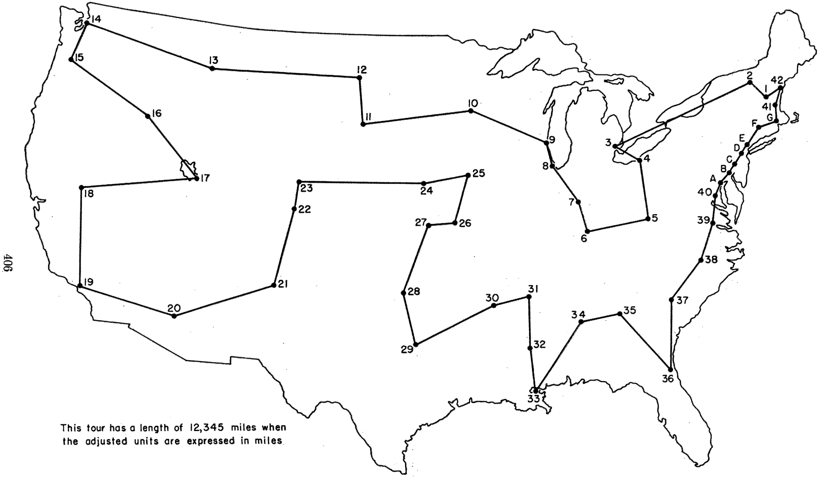
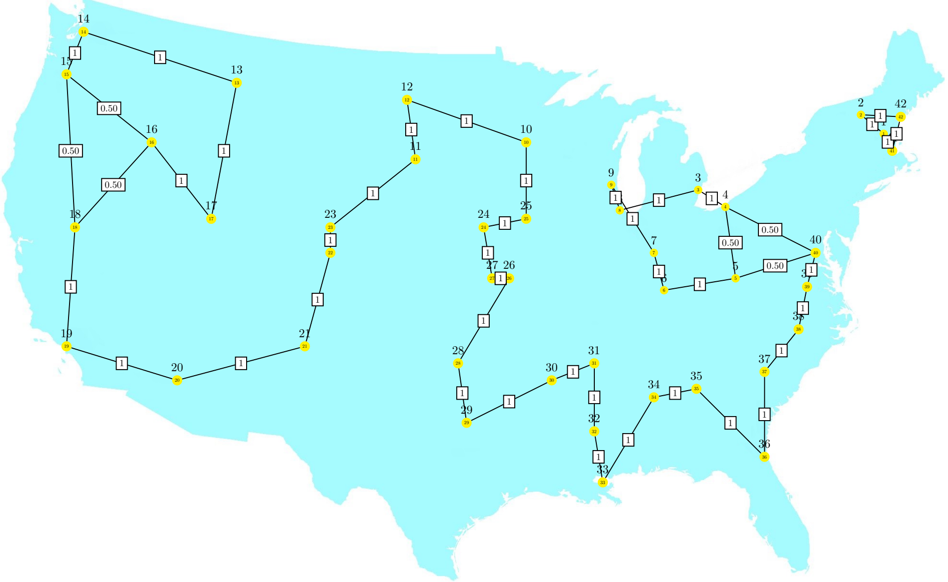
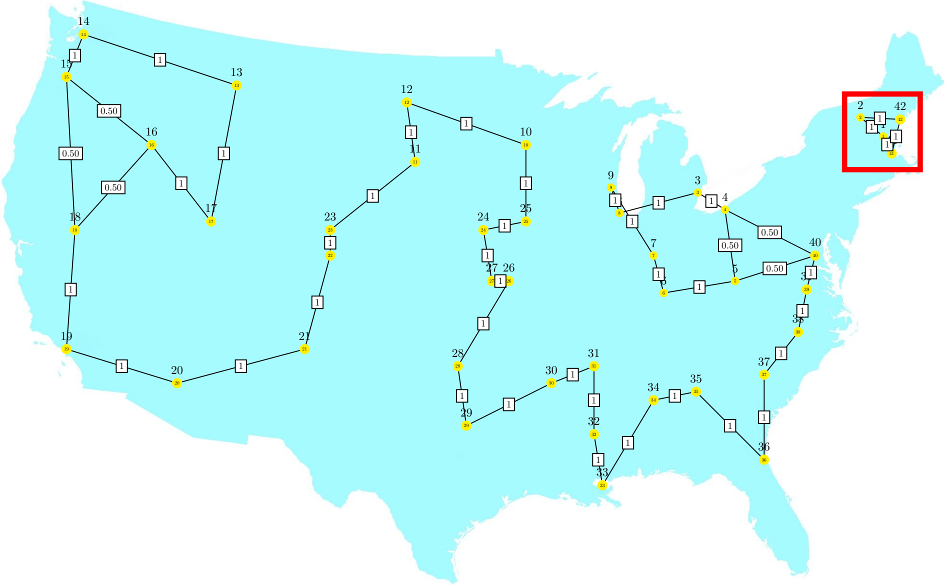


FIG. 16. The optimal tour of 49 cities.

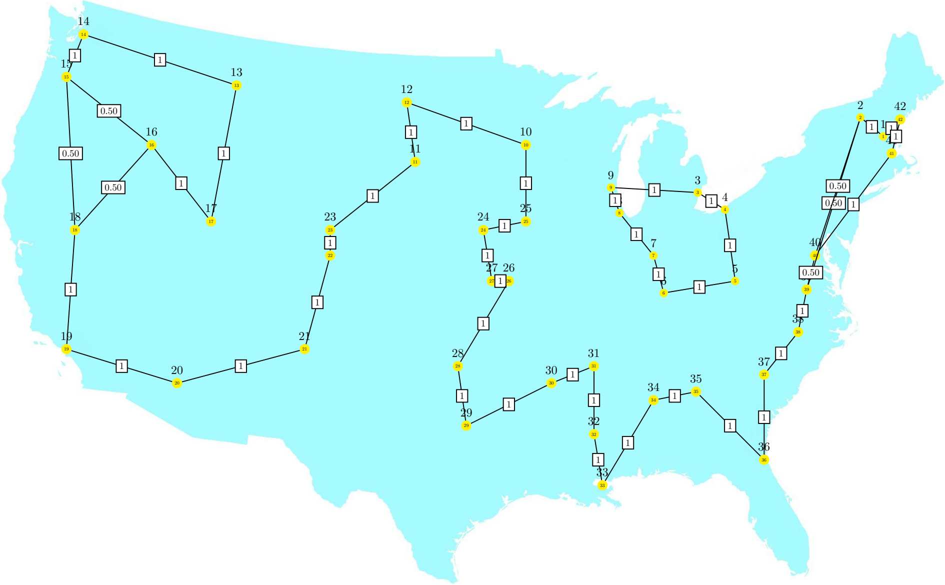
Iteration 1: Objective 641



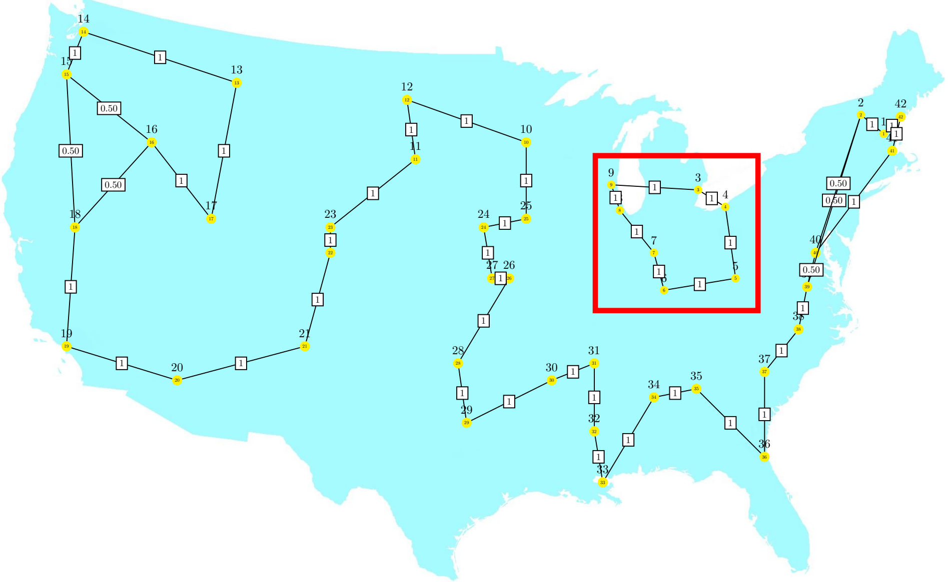
Iteration 1: Objective 641, Eliminate Subtour $1, 2, 41, 42$



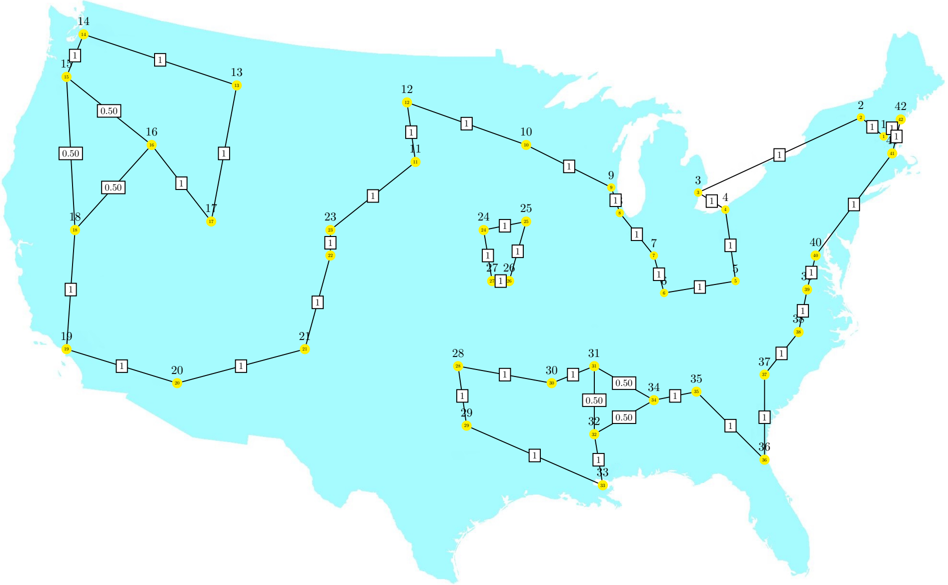
Iteration 2: Objective 676



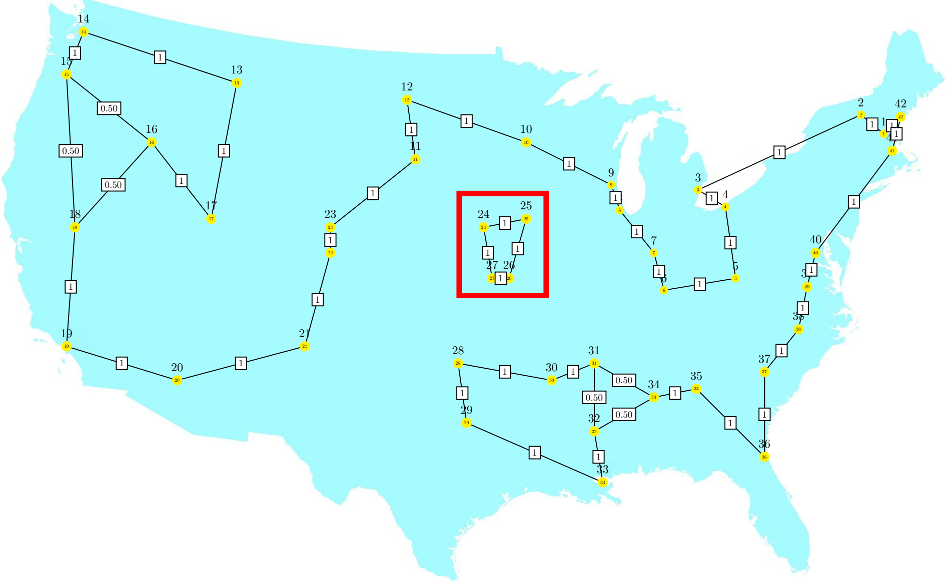
Iteration 2: Objective 676, Eliminate Subtour $3 - 9$



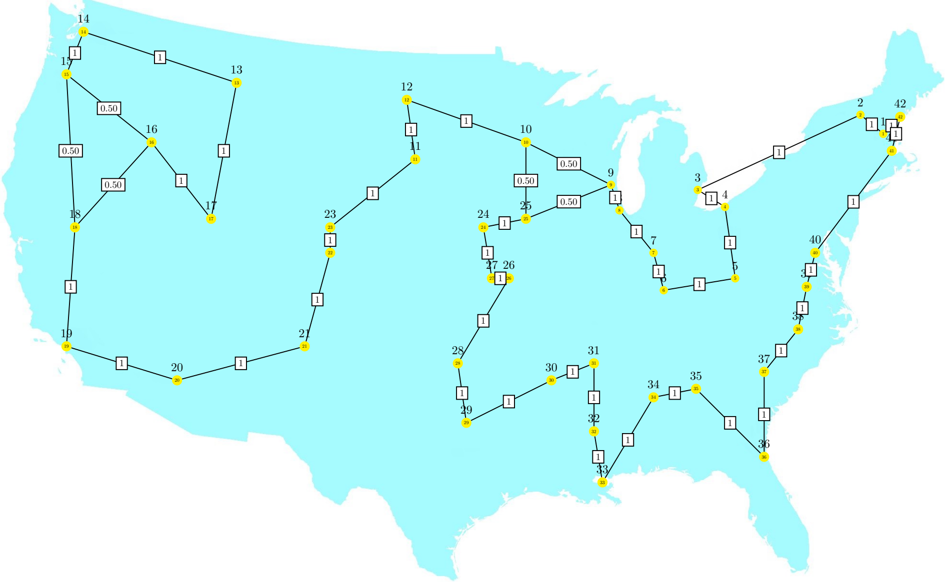
Iteration 3: Objective 681



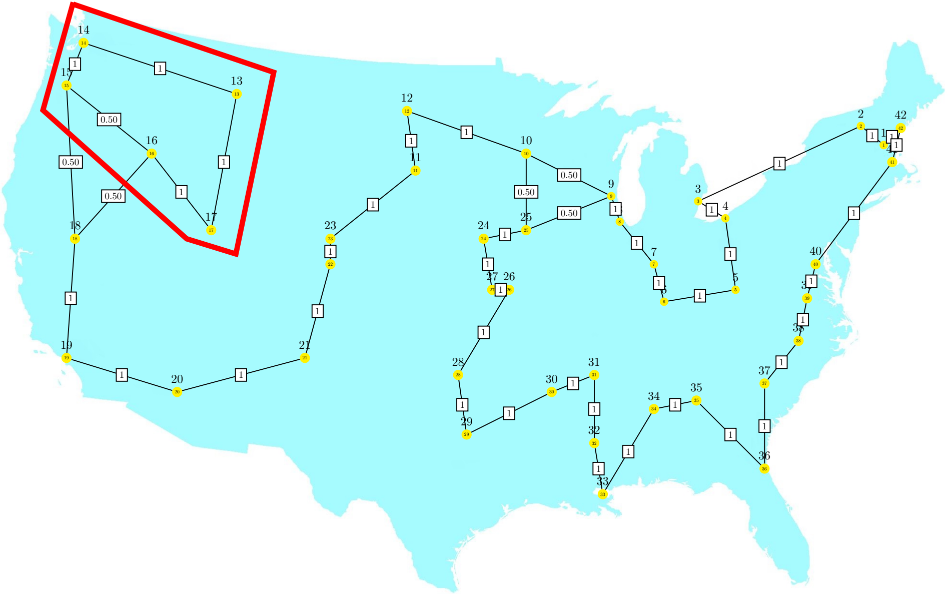
Iteration 3: Objective 681, Eliminate Subtour $24, 25, 26, 27$



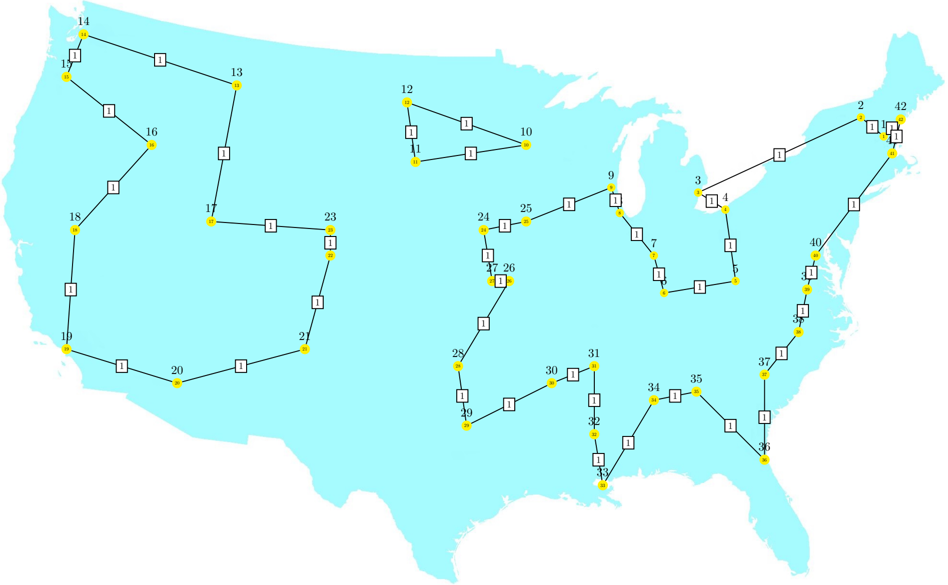
Iteration 4: Objective 682.5



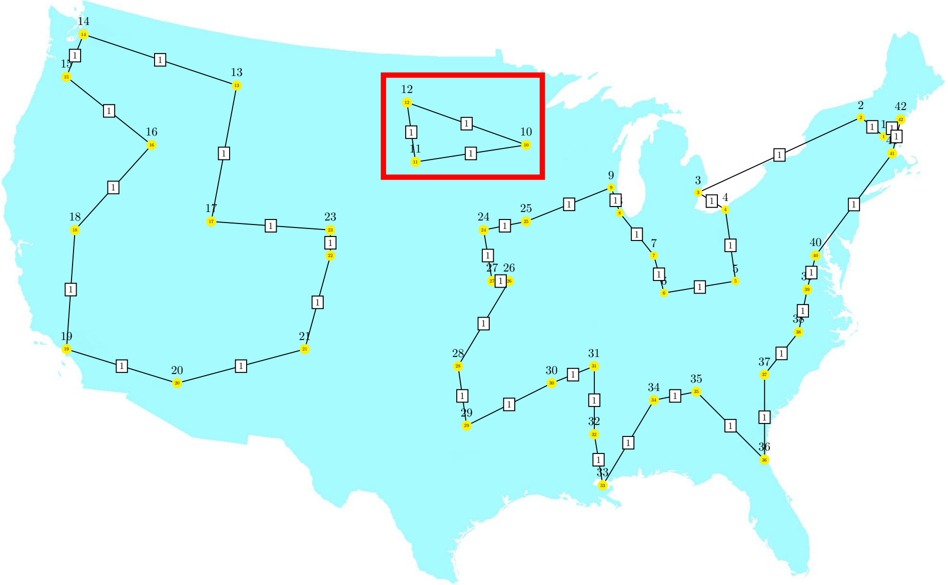
Iteration 4: Objective 682.5, Eliminate Small Cut by $13 - 17$



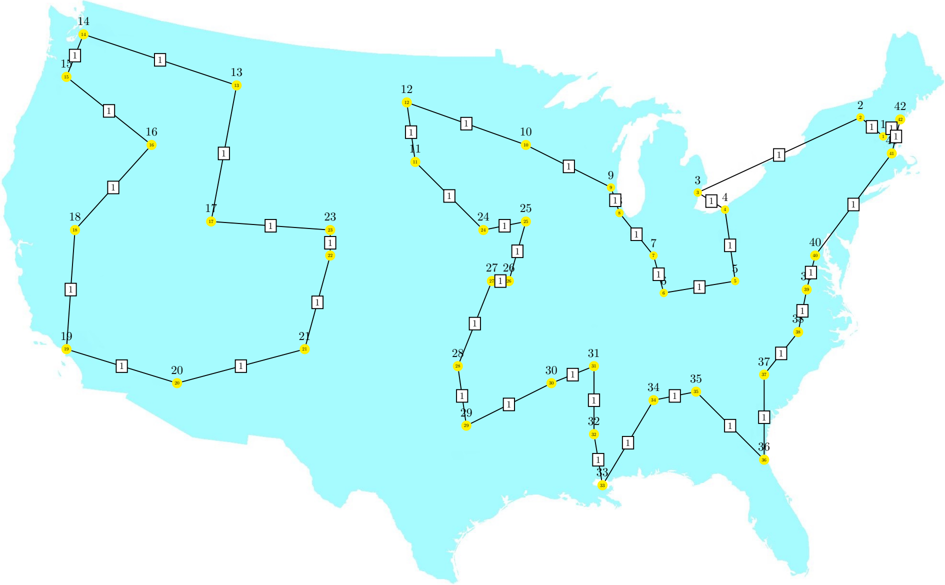
Iteration 5: Objective 686



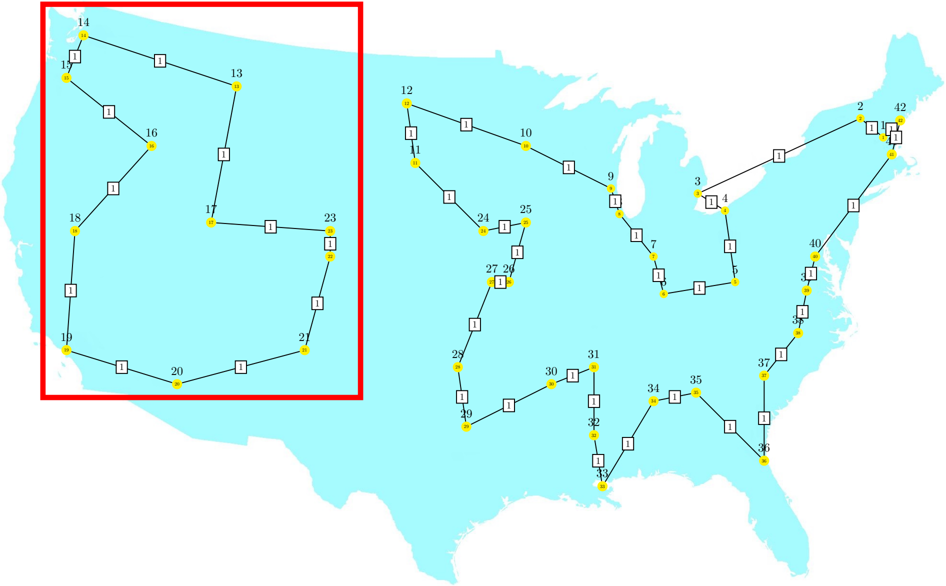
Iteration 5: Objective 686, Eliminate Subtour $10, 11, 12$



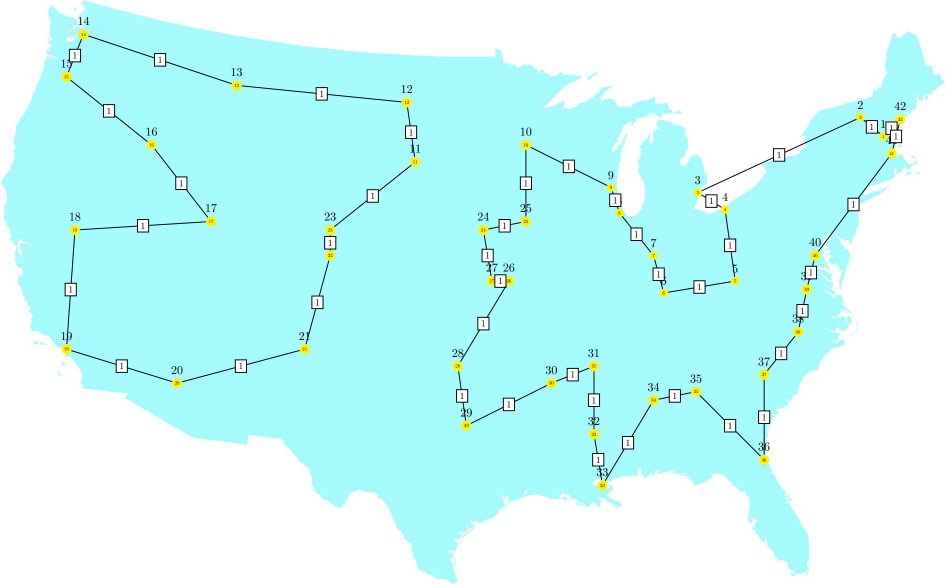
Iteration 6: Objective 686



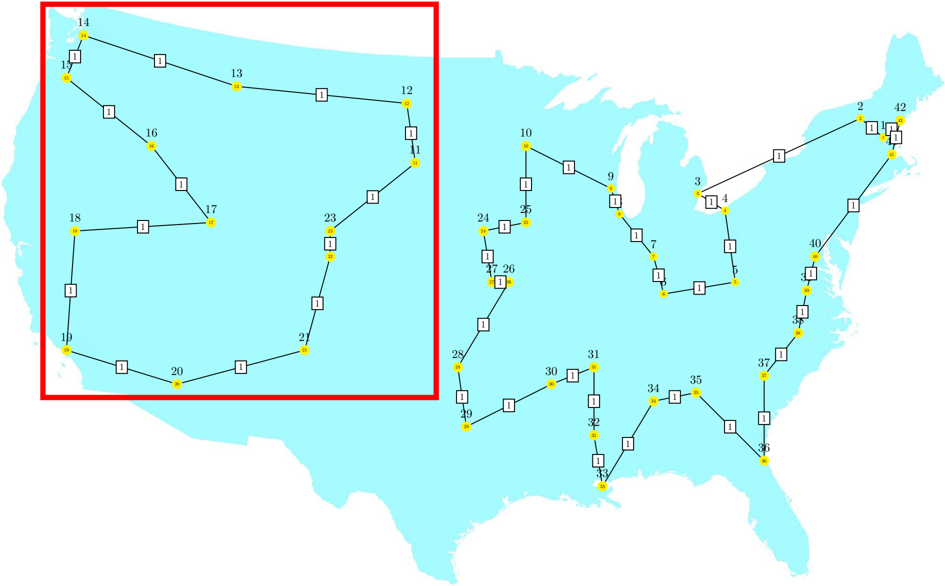
Iteration 6: Objective 686, Eliminate Subtour $13 - 23$



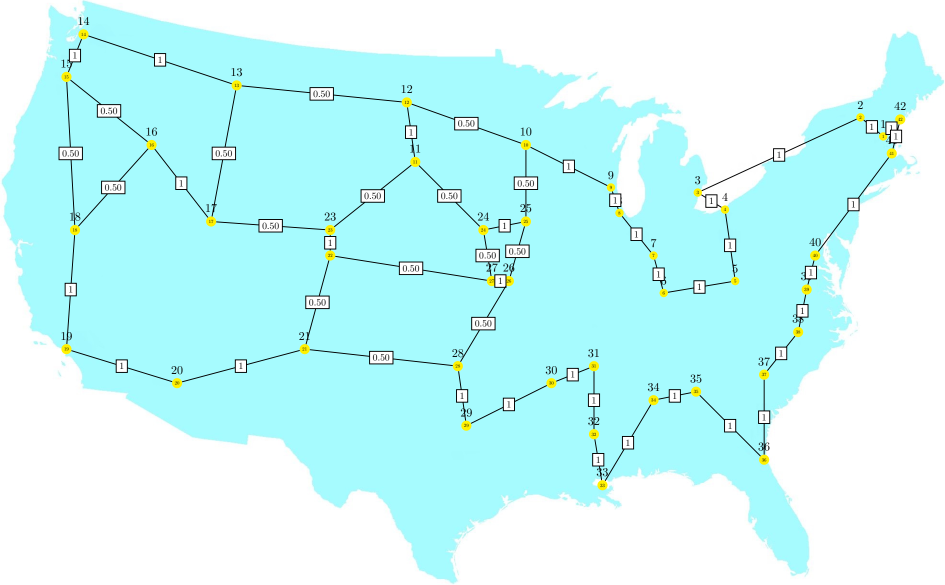
Iteration 7: Objective 688



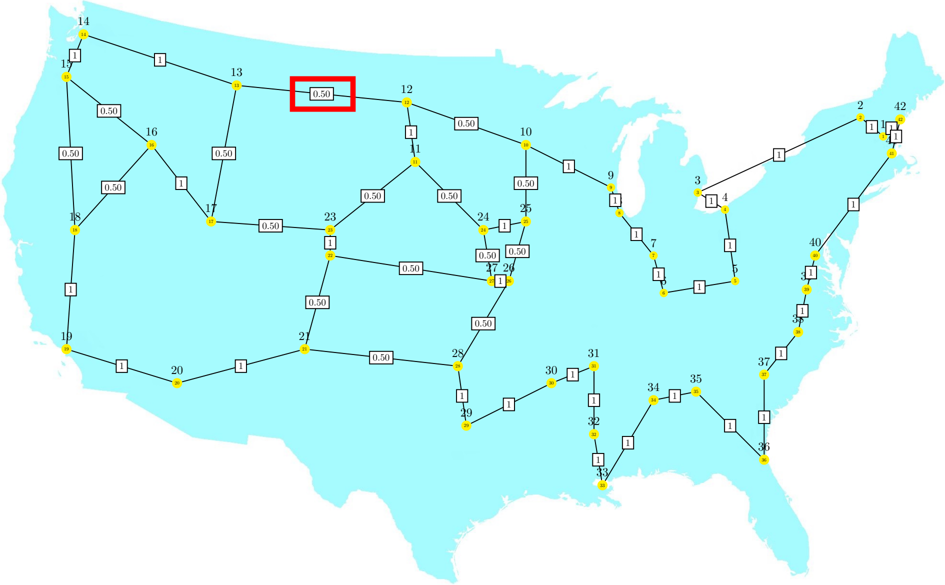
Iteration 7: Objective 688, Eliminate Subtour $11 - 23$



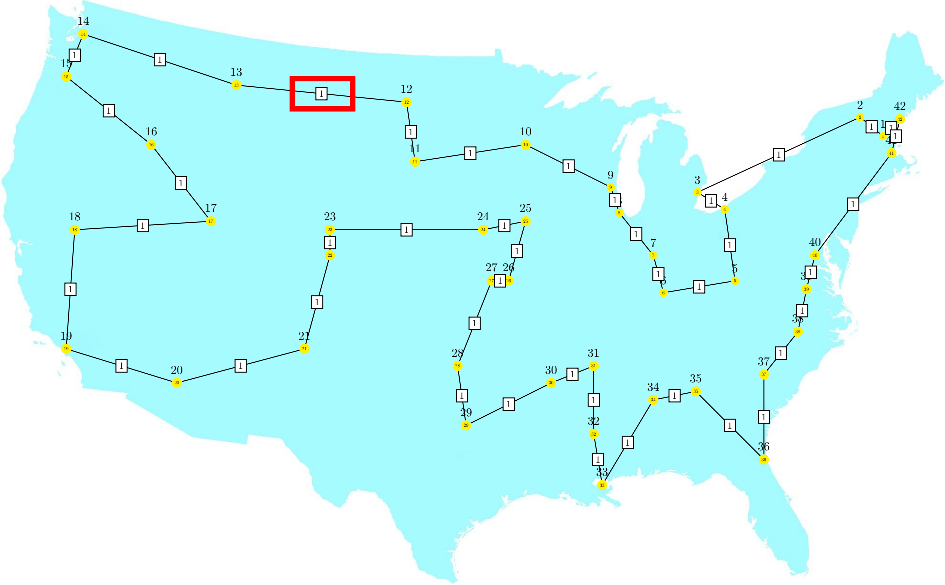
Iteration 8: Objective 697



Iteration 8: Objective 697, Branch on $x(13, 12)$



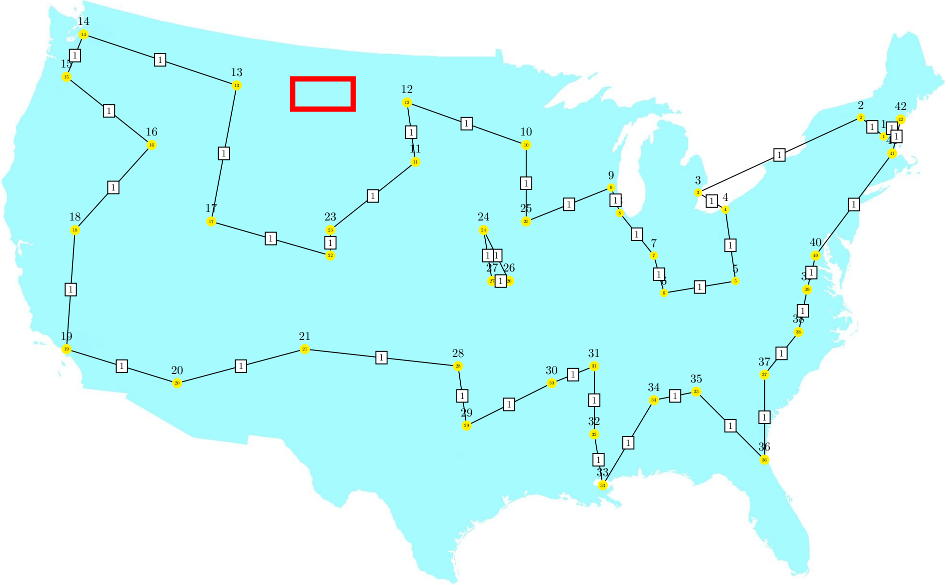
Iteration 9, Branch a $x(13, 12) = 1$: Objective 699 (Valid Tour)



```
Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.6.1.0
  with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2014.  All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX> read tsp.lp
Problem 'tsp.lp' read.
Read time = 0.00 sec. (0.06 ticks)
CPLEX> primopt
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 49 rows, 860 columns, and 2483 nonzeros.
Presolve time = 0.00 sec. (0.36 ticks)

Iteration log . . .
Iteration:     1    Infeasibility =            33.999999
Iteration:    26    Objective     =          1510.000000
Iteration:    90    Objective     =           923.000000
Iteration:   155    Objective     =           711.000000

Primal simplex - Optimal:  Objective =  6.9900000000e+02
Solution time =    0.00 sec.  Iterations = 168 (25)
Deterministic time = 1.16 ticks  (288.86 ticks/sec)

CPLEX>
```

```
CPLEX> display solution variables -
Variable Name           Solution Value
x_2_1                         1.000000
x_42_1                        1.000000
x_3_2                         1.000000
x_4_3                         1.000000
x_5_4                         1.000000
x_6_5                         1.000000
x_7_6                         1.000000
x_8_7                         1.000000
x_9_8                         1.000000
x_10_9                        1.000000
x_11_10                       1.000000
x_12_11                       1.000000
x_13_12                       1.000000
x_14_13                       1.000000
x_15_14                       1.000000
x_16_15                       1.000000
x_17_16                       1.000000
x_18_17                       1.000000
x_19_18                       1.000000
x_20_19                       1.000000
x_21_20                       1.000000
x_22_21                       1.000000
x_23_22                       1.000000
x_24_23                       1.000000
x_25_24                       1.000000
x_26_25                       1.000000
x_27_26                       1.000000
x_28_27                       1.000000
x_29_28                       1.000000
x_30_29                       1.000000
x_31_30                       1.000000
x_32_31                       1.000000
x_33_32                       1.000000
x_34_33                       1.000000
x_35_34                       1.000000
x_36_35                       1.000000
x_37_36                       1.000000
x_38_37                       1.000000
x_39_38                       1.000000
x_40_39                       1.000000
x_41_40                       1.000000
x_42_41                       1.000000
All other variables in the range 1-861 are 0.
```

Iteration 10, Branch b $x(13,12) = 0$: Objective 701



The End

Thank you for attending this course &
Best wishes for the rest of your Tripos!

- Don't forget to visit the online feedback page!
- Please send comments on the slides to:
  `tms41@cam.ac.uk`