Solving Line Intersection (without Trigonometry and Division!)

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

\[(p_4 - p_1) \times (p_2 - p_1) = (-2, 2) \times (-4, 2) = 4\]
Outline

Introduction and Line Intersection

Convex Hull

Glimpse at (More) Advanced Algorithms
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
- ...

Do these lines intersect?
Cross Product (Area)

$p_1 = (2, 1)$
$p_2 = (1, 3)$

$p_1 \times p_2 = \det(\begin{array}{cc} x_1 & y_1 \\ x_2 & y_2 \end{array}) = x_1y_2 - x_2y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$

$p_2 \times p_1 = \det(\begin{array}{cc} y_1 & x_1 \\ y_2 & x_2 \end{array}) = y_1x_2 - y_2x_1 = -5$
Cross Product (Area)

How large is this area?

\( \vec{p}_1 + \vec{p}_2 = (3, 4) \)

\( \vec{p}_2 = (1, 3) \)

\( \vec{p}_1 = (2, 1) \)

\( \vec{p}_1 \times \vec{p}_2 = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1y_2 - x_2y_1 = 2 \cdot 3 - 1 \cdot 1 = 5 \)

\( \vec{p}_2 \times \vec{p}_1 = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = y_1x_2 - y_2x_1 = -5 \)
Cross Product (Area)

How large is this area?

$p_1 = (2, 1)$

$p_2 = (1, 3)$

$p_1 + p_2 = (3, 4)$

$p_1 \times p_2 = \begin{vmatrix}
  x_1 & x_2 \\
  y_1 & y_2
\end{vmatrix} = x_1y_2 - x_2y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$

$p_2 \times p_1 = -5$
**Cross Product (Area)**

How large is this area?

\[ p_1 + p_2 = (3, 4) \]

\[ p_1 = (2, 1) \]

\[ p_2 = (1, 3) \]

\[ (0, 0) \]

\[ \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1y_2 - x_2y_1 \]

\[ = 2 \cdot 3 - 1 \cdot 1 = 5 \]

\[ p_2 \times p_1 = y_1x_2 - y_2x_1 \]

\[ = -5 \]

Alternatively, one could take the dot-product (but not used here):

\[ p_1 \cdot p_2 = \|p_1\| \cdot \|p_2\| \cdot \cos(\phi) \]
Cross Product (Area)

How large is this area?

\[ p_1 + p_2 = (3, 4) \]

\[ p_1 = (2, 1) \]

\[ p_2 = (1, 3) \]

\[ (0, 0) \]

\[ p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \]
Cross Product (Area)

How large is this area?

\( p_1 + p_2 = (3, 4) \)

\( p_2 = (1, 3) \)

\( p_1 = (2, 1) \)

\( x \)

\( y \)

\( (0, 0) \)

\( p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 \)
Cross Product (Area)

How large is this area?

$p_1 = (2, 1)$

$p_2 = (1, 3)$

$p_1 + p_2 = (3, 4)$

$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1y_2 - x_2y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$
Cross Product (Area)

How large is this area?

$p_1 + p_2 = (3, 4)$

$p_1 = (2, 1)$

$p_2 = (1, 3)$

$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1y_2 - x_2y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$
Cross Product (Area)

How large is this area?

$p_1 + p_2 = (3, 4)$

$p_1 = (2, 1)$

$p_2 = (1, 3)$

$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1y_2 - x_2y_1 = 2 \cdot 3 - 1 \cdot 1 = 5$

$p_2 \times p_1$
Cross Product (Area)

How large is this area?

$p_1 = (2, 1)$

$p_2 = (1, 3)$

$p_1 + p_2 = (3, 4)$

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

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

How large is this area?

\[ p_1 + p_2 = (3, 4) \]

\[ p_2 = (1, 3) \]

\[ p_1 = (2, 1) \]

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

\[ p_2 \times p_1 = y_1 x_2 - y_2 x_1 = -(p_1 \times p_2) \]
Cross Product (Area)

How large is this area?

$p_1 + p_2 = (3, 4)$

$p_1 = (2, 1)$

$p_2 = (1, 3)$

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

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

How large is this area?

\[ p_1 + p_2 = (3, 4) \]

\[ p_1 = (2, 1) \]

\[ p_2 = (1, 3) \]

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

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

Alternatively, one could take the dot-product (but not used here):

\[ p_1 \cdot p_2 = \|p_1\| \cdot \|p_2\| \cdot \cos(\phi). \]
Cross Product in 3D

\[ \mathbf{p}_1 \times \mathbf{p}_2 = (0, 0, x_1 y_2 - x_2 y_1) \]

\[ (\mathbf{p}_1 \times \mathbf{p}_3) > 0 \]

\[ (\mathbf{p}_1 \times \mathbf{p}_3) < 0 \]
Cross Product in 3D

\[ p_1 \times p_2 = (0, 0, x_1y_2 - x_2y_1) \]
Cross Product in 3D

\[ p_1 \times p_2 = (0, 0, x_1y_2 - x_2y_1) \]
Cross Product in 3D

\[ p_1 \times p_2 = (0, 0, x_1 y_2 - x_2 y_1) \]
The cross product in 3D is defined as:

\[ p_1 \times p_2 = (0, 0, x_1 y_2 - x_2 y_1) \]
Cross Product in 3D

\[ p_1 \times p_2 = (0, 0, x_1y_2 - x_2y_1) \]

\[ (p_1 \times \cdot) > 0 \]
\[ (p_1 \times \cdot) < 0 \]
Using Cross product to determine Turns

\[
p_1 = (2, 1), \quad p_2 = (1, 3), \quad p_3 = (1, -1)
\]

\[
p_1 \times p_2 > 0: \text{left (counterclockwise) turn}
\]

\[
p_1 \times p_3 < 0: \text{right (clockwise) turn}
\]

Sign of cross product determines turn!

Cross product equals zero iff vectors are colinear
Using Cross product to determine Turns

\[ p_1 = (2, 1) \]
\[ p_2 = (1, 3) \]
\[ p_3 = (1, -1) \]

The sign of the cross product determines the turn:

- If \( p_1 \times p_2 > 0 \): left (counterclockwise) turn
- If \( p_1 \times p_3 < 0 \): right (clockwise) turn

Cross product equals zero iff vectors are colinear.
Using Cross product to determine Turns

\[
p_1 = (2, 1) \quad p_2 = (1, 3) \quad p_3 = (1, -1)
\]

\[p_1 \times p_2 > 0: \text{left (counterclockwise) turn}\]

Sign of cross product determines turn!

Cross product equals zero iff vectors are colinear

7: Geometric Algorithms
Using Cross product to determine Turns

\[ p_1 = (2, 1), \quad p_2 = (1, 3), \quad p_3 = (1, -1) \]

\[ p_1 \times p_2 > 0: \text{ left (counterclockwise) turn} \]

Sign of cross product determines turn!

Cross product equals zero iff vectors are colinear.
Using Cross product to determine Turns

\[ p_1 = (2, 1) \]
\[ p_2 = (1, 3) \]
\[ p_3 = (1, -1) \]

\[ p_1 \times p_2 > 0: \text{ left (counterclockwise) turn} \]
\[ p_1 \times p_3 < 0: \text{ right (clockwise) turn} \]
Using Cross product to determine Turns

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

$p_1 \times p_2 > 0$: 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

\[ p_1 = (2, 1) \]
\[ p_2 = (1, 3) \]
\[ p_3 = (1, -1) \]

\[ p_1 \times p_2 > 0: \text{ left (counterclockwise) turn} \]
\[ p_1 \times p_3 < 0: \text{ right (clockwise) turn} \]

Sign of cross product determines turn!
Using Cross product to determine Turns

Using Cross product to determine Turns

\[ p_1 = (2, 1), \quad p_2 = (1, 3), \quad p_3 = (1, -1) \]

- \( p_1 \times p_2 > 0 \): 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)

\[
\begin{align*}
\mathbf{p}_0 &= (2, 1) \\
\mathbf{p}_1 &= (4, 2) \\
\mathbf{p}_2 &= (3, 4) \\
\mathbf{p}_3 &= (3, 0)
\end{align*}
\]

\[
\mathbf{p}_1 - \mathbf{p}_0 \times \mathbf{p}_2 - \mathbf{p}_0 > 0: \text{ left turn}
\]

\[
\mathbf{p}_1 - \mathbf{p}_0 \times \mathbf{p}_3 - \mathbf{p}_0 < 0: \text{ right turn}
\]
Using Cross product to determine Turns (origin shifted)

\[ \mathbf{p}_0 = (2, 1) \]
\[ \mathbf{p}_1 = (4, 2) \]
\[ \mathbf{p}_2 = (3, 4) \]
\[ \mathbf{p}_3 = (3, 0) \]

\[ (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) > 0: \text{left turn} \]
\[ (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_3 - \mathbf{p}_0) < 0: \text{right turn} \]
Using Cross product to determine Turns (origin shifted)

\[ \mathbf{p}_0 = (2, 1) \]

\[ \mathbf{p}_1 = (4, 2) \]

\[ \mathbf{p}_2 = (3, 4) \]

\[ \mathbf{p}_3 = (3, 0) \]

\[(\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) > 0: \text{ left turn} \]

\[(\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_3 - \mathbf{p}_0) < 0: \text{ right turn} \]
Using Cross product to determine Turns (origin shifted)

\[
\begin{align*}
\mathbf{p}_0 &= (2, 1) \\
\mathbf{p}_1 &= (4, 2) \\
\mathbf{p}_2 &= (3, 4) \\
\mathbf{p}_3 &= (3, 0)
\end{align*}
\]

\[
(\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) > 0: \text{ left turn}
\]

\[
(2, 1) \times (1, 3) = 5
\]

\[
(\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_3 - \mathbf{p}_0) < 0: \text{ right turn}
\]
Using Cross product to determine Turns (origin shifted)

\[
\begin{align*}
\mathbf{p}_0 &= (2, 1) \\
\mathbf{p}_1 &= (4, 2) \\
\mathbf{p}_2 &= (3, 4) \\
\mathbf{p}_3 &= (3, 0)
\end{align*}
\]

\[
(\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) > 0: \text{ left turn}
\]

\[
(2, 1) \times (1, 3) = 5
\]
Using Cross product to determine Turns (origin shifted)

\[ p_0 = (2, 1) \]
\[ p_1 = (4, 2) \]
\[ p_2 = (3, 4) \]
\[ p_3 = (3, 0) \]

\[ (p_1 - p_0) \times (p_2 - p_0) > 0: \text{ left turn} \]
\[ (p_1 - p_0) \times (p_3 - p_0) < 0: \text{ right turn} \]

\[ (2, 1) \times (1, 3) = 5 \]
Using Cross product to determine Turns (origin shifted)

\[ p_0 = (2, 1) \]
\[ p_1 = (4, 2) \]
\[ p_2 = (3, 4) \]
\[ p_3 = (3, 0) \]

\[ (p_1 - p_0) \times (p_2 - p_0) > 0: \text{ left turn} \]
\[ (2, 1) \times (1, 3) = 5 \]
\[ (p_1 - p_0) \times (p_3 - p_0) < 0: \text{ right turn} \]
\[ (2, 1) \times (1, -1) = -3 \]
Using Cross product to determine Turns (origin shifted)

\[
\begin{align*}
(p_1 - p_0) \times (p_2 - p_0) &> 0: \text{ left turn} \\
(2, 1) \times (1, 3) &= 5 \\
(p_1 - p_0) \times (p_3 - p_0) &< 0: \text{ right turn} \\
(2, 1) \times (1, -1) &= -3
\end{align*}
\]
Solving Line Intersection

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

\[(p_2 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3) = -10
\]

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

\[(p_3 - p_1) \times (p_2 - p_1) < 0
\]

\[(p_4 - p_1) \times (p_2 - p_1) < 0
\]

\(p_1 p_2 \text{ does not cross } p_3 p_4\)
Solving Line Intersection

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

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

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

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

Since \(\tilde{p}_1p_2 \cap \tilde{p}_3p_4\) consists of (at most) one point, \(p_1p_2 \cap p_3p_4 \neq \emptyset\).
Solving Line Intersection

\[(0,0)\]

\[\begin{align*}
(p_3 - p_1) \times (p_4 - p_3) &= (3,1) \times (1,3) = 8 \\
(p_2 - p_3) \times (p_4 - p_3) &= (0,0) \times (1,3) = 0 \\
(p_4 - p_1) \times (p_2 - p_1) &= (-1,3) \times (-4,2) = -10 \\
(p_3 - p_1) \times (p_2 - p_1) &= (-3,-1) \times (-4,2) = 10
\end{align*}\]

Opposite signs \(\Rightarrow\) \(p_1p_2\) crosses (infinite) line through \(p_3\) and \(p_4\)

Opposite signs \(\Rightarrow\) \(p_3p_4\) crosses (infinite) line through \(p_1\) and \(p_2\)

Since \(\tilde{p}_1p_2\cap\tilde{p}_3p_4\) consists of (at most) one point \(\Rightarrow\) \(p_1p_2\cap p_3p_4\neq\emptyset\)

\[\tilde{p}_1p_2\cap\tilde{p}_3p_4\supset\tilde{p}_1p_2\cap p_3p_4\neq\emptyset\]
Solving Line Intersection

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

\[(p_2 - p_3) \times (p_4 - p_3) = (-1, 3) \times (1, 3) = -10\]

Since \(p_1p_2 \cap p_3p_4\) consists of (at most) one point \(\Rightarrow p_1p_2 \cap p_3p_4 \neq \emptyset\)

\(p_1p_2\) does not cross \(p_3p_4\)
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 \(\Rightarrow\) \(\overrightarrow{p_1p_2}\) crosses (infinite) line through \(p_3\) and \(p_4\)
Solving Line Intersection

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

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

\[(p_4 - p_1) \times (p_2 - p_1) = (-2, 2) \times (-4, 2) = -10\]

\[(p_3 - p_4) \times (p_2 - p_4) = (2, 2) \times (-4, 2) = 4\]

Opposite signs \(\Rightarrow p_1p_2\) crosses (infinite) line through \(p_3\) and \(p_4\)

Opposite signs \(\Rightarrow p_3p_4\) crosses (infinite) line through \(p_1\) and \(p_2\)

\[\overrightarrow{p_1p_2} \cap \overrightarrow{p_3p_4} \supseteq \overrightarrow{p_1p_2} \cap \overrightarrow{p_3p_4} \neq \emptyset\]

Opposite signs \(\Rightarrow p_1p_2\) does not cross \(p_3p_4\)

Opposite signs \(\Rightarrow p_3p_4\) does not cross \(p_1p_2\)
Solving Line Intersection

\[ (p_3 - p_1) \times (p_2 - p_1) = (3, 1) \times (1, 3) = 8 \]

\[ (p_4 - p_1) \times (p_2 - p_1) = (-1, 3) \times (-4, 2) = -10 \]

\[ \text{Opposite signs} \Rightarrow p_1p_2 \text{ crosses } p_3p_4 \]

\[ \text{(infinite) line through } p_3 \text{ and } p_4 \]
Solving Line Intersection

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

Opposite signs \(\Rightarrow\) \(p_1p_2\) 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 ⇒ \(\overline{p_1p_2}\) 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 \(\Rightarrow p_1p_2\) crosses (infinite) line through \(p_3\) and \(p_4\)
(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 \(\Rightarrow\) \overrightarrow{p_1p_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 \(\Rightarrow\) \(\overline{p_1p_2}\) 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 \(\Rightarrow\) \(p_1p_2\) crosses (infinite) line through \(p_3\) and \(p_4\)

Opposite signs \(\Rightarrow\) \(p_3p_4\) crosses (infinite) line through \(p_1\) and \(p_2\)

(0, 0)
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 \(\Rightarrow\) \(\overrightarrow{p_1p_2}\) crosses (infinite) line through \(p_3\) and \(p_4\)

Opposite signs \(\Rightarrow\) \(\overrightarrow{p_3p_4}\) crosses (infinite) line through \(p_1\) and \(p_2\)
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 \( \Rightarrow p_1 p_2 \) crosses (infinite) line through \( p_3 \) and \( p_4 \)

Opposite signs \( \Rightarrow p_3 p_4 \) crosses (infinite) line through \( p_1 \) and \( p_2 \)
Solving Line Intersection

\[ (p_1 - p_3) \times (p_4 - p_3) = 3 \times 1 = 3 \times 1 = \underline{8} \]

\[ (p_2 - p_3) \times (p_4 - p_3) = -1 \times 3 = -3 \times 2 = \underline{-6} \]

\[ (p_3 - p_1) \times (p_2 - p_1) = -3 \times -1 = 3 \times 2 = \underline{-10} \]

\[ (p_4 - p_1) \times (p_2 - p_1) = -2 \times -1 = 2 \times 2 = \underline{4} \]

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

\[
\begin{align*}
(p_1 - p_3) \times (p_4 - p_3) &= (3, 1) \times (1, 3) = 8 \\
(p_2 - p_3) \times (p_4 - p_3) &= (-1, 3) \times (1, 3) = -6 \\
(p_4 - p_1) \times (p_2 - p_1) &= (-2, 2) \times (-4, 2) = 4
\end{align*}
\]

Opposite signs \(\Rightarrow\) \(p_1p_2\) crosses (infinite) line through \(p_3\) and \(p_4\)

Opposite signs \(\Rightarrow\) \(p_3p_4\) crosses (infinite) line through \(p_1\) and \(p_2\)
Solving Line Intersection

\[ \begin{align*}
(p_1 - p_3) \times (p_4 - p_3) & = (3, 1) \times (1, 3) = 8 \\
(p_2 - p_3) \times (p_4 - p_3) & = (-1, 3) \times (1, 3) = -6 \\
(p_4 - p_1) \times (p_2 - p_1) & = (-2, 2) \times (-4, 2) = 4
\end{align*} \]

Opposite signs \(\Rightarrow\) \(p_1p_2\) crosses (infinite) line through \(p_3\) and \(p_4\)

Opposite signs \(\Rightarrow\) \(p_3p_4\) crosses (infinite) line through \(p_1\) and \(p_2\)

Since \(\overline{p_1p_2} \cap \overline{p_3p_4}\) consists of (at most) one point
\[ \Rightarrow \overline{p_1p_2} \cap \overline{p_3p_4} \neq \emptyset \]
Solving Line Intersection

\[ \text{Opposite signs} \Rightarrow \overline{p_1p_2} \text{ crosses } \overline{p_3p_4} \]

\[ \text{(infinite) line through } p_3 \text{ and } p_4 \]

\[ \text{Opposite signs} \Rightarrow \overline{p_3p_4} \text{ crosses } \overline{p_1p_2} \]

\[ \text{(infinite) line through } p_1 \text{ and } p_2 \]
Solving Line Intersection

\[(0, 0)\]

\[\begin{align*}
p_2 & \quad p_4 \quad p_1 \\
p_3 & \quad (p_1 - p_3) \times (p_4 - p_3) = (3, 1) \times (1, 3) = 8 \\
(p_2 - p_3) \times (p_4 - p_3) & = (-1, 3) \times (1, 3) = -6 \\
(p_3 - p_1) \times (p_2 - p_1) & = (-3, -1) \times (-4, 2) = -10 \\
(p_4 - p_1) \times (p_2 - p_1) & = (-2, 2) \times (-4, 2) = 4
\end{align*}\]

Opposite signs \(\Rightarrow \) \(p_1 p_2\) crosses (infinite) line through \(p_3\) and \(p_4\)

Opposite signs \(\Rightarrow \) \(p_3 p_4\) crosses (infinite) line through \(p_1\) and \(p_2\)

\[\sim p_1 p_2 \cap \sim p_3 p_4 \supseteq p_1 p_2 \cap p_3 p_4 \neq \emptyset\]

Since \(\sim p_1 p_2 \cap \sim p_3 p_4\) consists of (at most) one point \(\Rightarrow \) \(p_1 p_2 \cap p_3 p_4 \neq \emptyset\)

\(p_1 p_2\) does not cross \(p_3 p_4\)
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 \(\Rightarrow p_1 p_2\) crosses \((infinite)\) line through \(p_3 p_4\)

Opposite signs \(\Rightarrow p_3 p_4\) crosses \((infinite)\) line through \(p_1 p_2\)

\[\tilde{p}_1 p_2 \cap \tilde{p}_3 p_4 \supset \tilde{p}_1 p_2 \cap p_3 p_4 \neq \emptyset\]
Solving Line Intersection

\[(p_3 - p_1) \times (p_2 - p_1) < 0\]
Solving Line Intersection

\[ (p_3 - p_1) \times (p_2 - p_1) < 0 \]

\( \Rightarrow \) \( p_1 \) \( p_2 \) crosses \( (\infty) \) line through \( p_3 \) and \( p_4 \)

\( \Rightarrow \) \( p_3 \) \( p_4 \) crosses \( (\infty) \) line through \( p_1 \) and \( p_2 \)

\( \Rightarrow \) \( \tilde{p}_1 \) \( \tilde{p}_2 \) \( \cap \) \( \tilde{p}_3 \) \( \tilde{p}_4 \) \( \supseteq \) \( p_1 \) \( p_2 \) \( \cap \) \( \tilde{p}_3 \) \( \tilde{p}_4 \) \( \neq \) \( \emptyset \)

\( \Rightarrow \) \( \tilde{p}_1 \) \( \tilde{p}_2 \) \( \cap \) \( \tilde{p}_3 \) \( \tilde{p}_4 \) \( \supseteq \) \( \tilde{p}_1 \) \( \tilde{p}_2 \) \( \cap \) \( p_3 \) \( p_4 \) \( \neq \) \( \emptyset \)

Since \( \tilde{p}_1 \) \( \tilde{p}_2 \) \( \cap \) \( \tilde{p}_3 \) \( \tilde{p}_4 \) consists of (at most) one point

\( \Rightarrow \) \( p_1 \) \( p_2 \) \( \cap \) \( p_3 \) \( p_4 \) \( \neq \) \( \emptyset \)

\( p_1 \) \( p_2 \) does not cross \( p_3 \) \( p_4 \)
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

\[
(p_3 - p_1) \times (p_2 - p_1) < 0
\]

\[
(p_4 - p_1) \times (p_2 - p_1) < 0
\]

\(\overline{p_1p_2}\) does not cross \(\overline{p_3p_4}\)
Solving Line Intersection

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

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

0: \( \text{DIRECTION}(p_i, p_j, p_k) \)
1: \( \text{return } (p_k - p_i) \times (p_j - p_i) \)
Solving Line Intersection

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

0: SEGMENTS-INTERSECT($p_1, p_2, p_3, p_4$)
1: $d_1 = \text{DIRECTION}(p_3, p_4, p_1)$
2: $d_2 = \text{DIRECTION}(p_3, p_4, p_2)$
3: $d_3 = \text{DIRECTION}(p_1, p_2, p_3)$
4: $d_4 = \text{DIRECTION}(p_1, p_2, p_4)$
5: If $d_1 \cdot d_2 < 0$ and $d_3 \cdot d_4 < 0$ return TRUE
6: ⋮ (handle all degenerate cases)
Solving Line Intersection

DIRECTION\((p_3, p_4, p_1)\) = \((p_1 - p_3) \times (p_4 - p_3)\)

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

0: SEGMENTS-INTERSECT\((p_1, p_2, p_3, p_4)\)
1: \(d_1 =\) DIRECTION\((p_3, p_4, p_1)\)
2: \(d_2 =\) DIRECTION\((p_3, p_4, p_2)\)
3: \(d_3 =\) DIRECTION\((p_1, p_2, p_3)\)
4: \(d_4 =\) DIRECTION\((p_1, p_2, p_4)\)
5: If \(d_1 \cdot d_2 < 0\) and \(d_3 \cdot d_4 < 0\) return TRUE
6: ⋯ (handle all degenerate cases)

In total 4 satisfying conditions!

Lines could touch or be colinear
Solving Line Intersection

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

0: \text{DIRECTION}(p_i, p_j, p_k)
1: \quad \text{return } (p_k - p_i) \times (p_j - p_i)

0: \text{SEGMENTS-INTERSECT}(p_1, p_2, p_3, p_4)
1: \quad d_1 = \text{DIRECTION}(p_3, p_4, p_1)
2: \quad d_2 = \text{DIRECTION}(p_3, p_4, p_2)
3: \quad d_3 = \text{DIRECTION}(p_1, p_2, p_3)
4: \quad d_4 = \text{DIRECTION}(p_1, p_2, p_4)
5: \quad \text{If } d_1 \cdot d_2 < 0 \text{ and } d_3 \cdot d_4 < 0 \text{ return TRUE}
6: \quad \cdots \text{ (handle all degenerate cases)}

In total 4 satisfying conditions!
Solving Line Intersection

\[
DIRECTION(p_3, p_4, p_1) = (p_1 - p_3) \times (p_4 - p_3)
\]

0: \(DIRECTION(p_i, p_j, p_k)\)
1: \(\text{return } (p_k - p_i) \times (p_j - p_i)\)

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

Lines could touch or be colinear
Solving Line Intersection

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

\[ \text{DIRECTION}(p_i, p_j, p_k) = (p_k - p_i) \times (p_j - p_i) \]

\[ \text{SEGMENTS-INTERSECT}(p_1, p_2, p_3, p_4) \]

1. \( d_1 = \text{DIRECTION}(p_3, p_4, p_1) \)
2. \( d_2 = \text{DIRECTION}(p_3, p_4, p_2) \)
3. \( d_3 = \text{DIRECTION}(p_1, p_2, p_3) \)
4. \( d_4 = \text{DIRECTION}(p_1, p_2, p_4) \)
5. If \( d_1 \cdot d_2 < 0 \) and \( d_3 \cdot d_4 < 0 \) return TRUE
6. \( \cdots \) (handle all degenerate cases)

Lines could touch or be colinear
Solving Line Intersection

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

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

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

Lines could touch or be colinear
Introduction and Line Intersection

Convex Hull

Glimpse at (More) Advanced Algorithms
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

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.
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.
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.
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

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.
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**

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

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 \textit{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*.
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-SCAN(Q)

1: Let $p_0$ be the point with minimum $y$-coordinate
2: Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. $p_0$
3: If $n < 2$ return false
4: $S = \emptyset$
5: PUSH($p_0$, $S$)
6: PUSH($p_1$, $S$)
7: PUSH($p_2$, $S$)
8: For $i = 3$ to $n$
9: While angle of NEXT-TO-TOP($S$), TOP($S$), $p_i$ makes a non-left turn
10: POP($S$)
11: End While
12: PUSH($p_i$, $S$)
13: End For
14: Return $S$
```

Takes $O(n \log n)$ time, since every point is part of a PUSH or POP at most once. Overall Runtime: $O(n \log n)$.
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-SCAN(Q)

1. Let $p_0$ be the point with minimum y-coordinate
2. Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. $p_0$
3. If $n < 2$ return false
4. $S = \emptyset$
5. PUSH($p_0$, $S$)
6. PUSH($p_1$, $S$)
7. PUSH($p_2$, $S$)
8. For $i = 3$ to $n$
   9. While angle of NEXT-TO-TOP(S), TOP(S), $p_i$ makes a non-left turn
      10. POP(S)
   11. End While
   12. PUSH($p_i$, S)
13. End For
14. Return S
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

### Algorithm

1. **GRAHAM-SCAN**(\( Q \))
2. Let \( p_0 \) be the point with minimum \( y \)-coordinate
3. Let \( (p_1, p_2, \ldots, p_n) \) be the other points sorted by polar angle w.r.t. \( p_0 \)
4. If \( n < 2 \) return false
5. \( S = \emptyset \)
6. Push \( p_0 \) to \( S \)
7. Push \( p_1 \) to \( S \)
8. Push \( p_2 \) to \( S \)
9. For \( i = 3 \) to \( n \)
   - While angle of NEXT-TO-TOP(\( S \)), TOP(\( S \)), \( p_i \) makes a non-left turn
     - Pop \( S \)
10. Push \( p_i \) to \( S \)
11. End For
12. Return \( S \)

### Time Complexity
- Takes \( \mathcal{O}(n \log n) \) time
- Takes \( \mathcal{O}(n) \) time, since every point is part of a PUSH or POP at most once.
- **Overall Runtime:** \( \mathcal{O}(n \log n) \)
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

- 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

Basic Idea

\[
\begin{align*}
\text{GRAHAM-SCAN}(Q) &= 0 \\
\text{Let } p_0 &= \text{the point with minimum } y \text{-coordinate} \\
\text{Let } (p_1, p_2, \ldots, p_n) &= \text{the other points sorted by polar angle w.r.t. } p_0 \\
\text{If } n < 2 \text{ return false} \\
S &= \emptyset \\
PUSH(p_0, S) \\
PUSH(p_1, S) \\
PUSH(p_2, S) \\
\text{For } i = 3 \text{ to } n \text{ \textbf{While angle of NEXT-TO-TOP}(S),TOP(S), } p_i \text{ makes a non-left turn} \\
\text{POP}(S) \\
\text{End While} \\
PUSH(p_i, S) \\
\text{End For} \\
\text{Return } S
\end{align*}
\]

Takes $O(n \log n)$ time

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

Overall Runtime: $O(n \log n)$
Graham’s Scan

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 ✓

Basic Idea

Algorithm:

1. Let $p_0$ be the point with minimum $y$-coordinate
2. Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. $p_0$
3. If $n < 2$ return false
4. $S = \emptyset$
5. PUSH($p_0$, $S$)
6. PUSH($p_1$, $S$)
7. PUSH($p_2$, $S$)
8. For $i = 3$ to $n$
9.   While angle of NEXT - TO - TOP($S$), TOP($S$), $p_i$ makes a non-left turn
10.  POP($S$)
11. End While
12. PUSH($p_i$, $S$)
13. End For
14. 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.
Overall Runtime: $O(n \log n)$
Graham’s Scan

- 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

---

Takes $O(n \log n)$ time

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

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

- 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

Efficient Sorting by comparing (not computing!) polar angles

```
0: GRAHAM-SCAN(Q)
1: Let $p_0$ be the point with minimum $y$-coordinate
2: Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. $p_0$
3: If $n < 2$ return false
4: $S = \emptyset$
5: PUSH($p_0$, S)
6: PUSH($p_1$, S)
7: PUSH($p_2$, S)
8: For $i = 3$ to $n$
9: While angle of NEXT-TO-TOP(S), TOP(S), $p_i$ makes a non-left turn
10: POP(S)
11: End While
12: PUSH($p_i$, S)
13: End For
14: 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.

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

**Efficient Sorting by comparing (not computing!) polar angles**
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

Efficient Sorting by comparing (not computing!) polar angles

---

Takes $O(n \log n)$ time

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

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

Use Cross Product!
Graham’s Scan

0: GRAHAM-SCAN(Q)
1:   Let \( p_0 \) be the point with minimum \( y \)-coordinate
2:   Let \( (p_1, p_2, \ldots, p_n) \) be the other points sorted by polar angle w.r.t. \( p_0 \)
3:   If \( n < 2 \) return false
4:   \( S = \emptyset \)
5:   PUSH\((p_0, S)\)
6:   PUSH\((p_1, S)\)
7:   PUSH\((p_2, S)\)
8:   For \( i = 3 \) to \( n \)
9:      While angle of NEXT-TO-TOP\((S)\),TOP\((S)\),\( p_i \) makes a non-left turn
10:     POP\((S)\)
11:   End While
12:   PUSH\((p_i, S)\)
13:   End For
14:   Return \( S \)
Graham’s Scan

0: GRAHAM-SCAN(Q)
1: Let $p_0$ be the point with minimum $y$-coordinate
2: Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. $p_0$
3: If $n < 2$ return false
4: $S = \emptyset$
5: PUSH($p_0$, S)
6: PUSH($p_1$, S)
7: PUSH($p_2$, S)
8: For $i = 3$ to $n$
9:     While angle of NEXT-TO-TOP(S), TOP(S), $p_i$ makes a non-left turn
10:         POP(S)
11:     End While
12:     PUSH($p_i$, S)
13: End For
14: Return S
Graham's Scan

0: GRAHAM-SCAN(Q)
1: Let \( p_0 \) be the point with minimum \( y \)-coordinate
2: Let \((p_1, p_2, \ldots, p_n)\) be the other points sorted by polar angle w.r.t. \( p_0 \)
3: If \( n < 2 \) return false
4: \( S = \emptyset \)
5: PUSH\((p_0,S)\)
6: PUSH\((p_1,S)\)
7: PUSH\((p_2,S)\)
8: For \( i = 3 \) to \( n \)
9: While angle of NEXT-TO-TOP(S),TOP(S),\( p_i \) makes a non-left turn
10: POP(S)
11: End While
12: PUSH\((p_i,S)\)
13: End For
14: Return S

Takes \( O(n \log n) \) time
Graham’s Scan

0: GRAHAM-SCAN(Q)
1: Let $p_0$ be the point with minimum $y$-coordinate
2: Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. $p_0$
3: If $n < 2$ return false
4: $S = \emptyset$
5: PUSH($p_0$, $S$)
6: PUSH($p_1$, $S$)
7: PUSH($p_2$, $S$)
8: For $i = 3$ to $n$
9: \hspace{1em} While angle of NEXT-TO-TOP($S$), TOP($S$), $p_i$ makes a non-left turn
10: \hspace{2em} POP($S$)
11: \hspace{1em} End While
12: \hspace{1em} PUSH($p_i$, $S$)
13: End For
14: Return $S$

Takes $O(n \log n)$ time
Graham’s Scan

0: GRAHAM-SCAN(Q)
1: Let $p_0$ be the point with minimum $y$-coordinate
2: Let $(p_1, p_2, \ldots, p_n)$ be the other points sorted by polar angle w.r.t. $p_0$
3: If $n < 2$ return false
4: $S = \emptyset$
5: PUSH($p_0$, S)
6: PUSH($p_1$, S)
7: PUSH($p_2$, S)
8: For $i = 3$ to $n$
9: While angle of NEXT-TO-TOP(S), TOP(S), $p_i$ makes a non-left turn
10: POP(S)
11: End While
12: PUSH($p_i$, S)
13: End For
14: 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

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

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

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

\[ i = 0 \]
Execution of Graham’s Scan

\[ i = 1 \]

0 1

\[ \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9 \\
10 \\
11 \\
12 \\
13 \\
14 \\
15 \\
\end{array} \]
Execution of Graham’s Scan

\[ i = 2 \]

\[ 0 \quad 1 \quad 2 \]
Execution of Graham’s Scan

\[ i = 3 \]
Execution of Graham’s Scan

\[ i = 4 \]

0 1 2 3

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

✓
Execution of Graham’s Scan

\[ i = 4 \]

Diagram showing the execution of Graham's Scan with points labeled 0 to 15.
Execution of Graham’s Scan

\[ i = 4 \]

0 1 2

\[ 0 \quad 1 \quad 2 \]
Execution of Graham’s Scan

\[ i = 4 \]
Execution of Graham’s Scan

\[ i = 4 \]

\[
\begin{array}{c|c|c}
0 & 1 & 4 \\
\end{array}
\]
Execution of Graham’s Scan

\[ i = 5 \]

\[ 0 \quad 1 \quad 4 \]
Execution of Graham’s Scan

\[ i = 5 \]
Execution of Graham’s Scan

\[ i = 5 \]

\[ 0 \quad 1 \quad 5 \]
Execution of Graham’s Scan

\[ i = 6 \]

\[ 0 \ 1 \ 5 \ 6 \]

\[ 0 \ 1 \ 2 \ 3 \ 4 \]

\[ 5 \]

\[ 6 \]

\[ 7 \]

\[ 8 \]

\[ 9 \]

\[ 10 \]

\[ 11 \]

\[ 12 \]

\[ 13 \]

\[ 14 \]

\[ 15 \]
Execution of Graham’s Scan

\[ i = 7 \]

\[ 0 \quad 1 \quad 5 \quad 6 \quad 7 \]
Execution of Graham’s Scan

\[ i = 8 \]

\[ 0 \quad 1 \quad 5 \quad 6 \quad 7 \]
Execution of Graham’s Scan

\[ i = 8 \]

0 1 5 6

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

0
Execution of Graham’s Scan

\[ i = 8 \]

\[
\begin{align*}
0 & \quad 1 & \quad 5 & \quad 6 \\
1 & \quad 2 & \quad 3 & \quad 4 \\
6 & \quad 5 & \quad 8 & \quad 12 \\
9 & \quad 10 & \quad 11 & \quad 13 \\
14 & \quad 15 & \quad 0 & \quad 1
\end{align*}
\]
Execution of Graham’s Scan

\[ i = 8 \]

\[
\begin{array}{ccc}
0 & 1 & 5 \\
\end{array}
\]
Execution of Graham’s Scan

\[ i = 8 \]

\[ \begin{array}{cccc}
0 & 1 & 5 & 8 \\
\end{array} \]
Execution of Graham’s Scan

\[ i = 9 \]

Diagram showing the execution of Graham's Scan algorithm with vertices numbered 0 to 15.
Execution of Graham’s Scan

\[ i = 10 \]

\[ 0 \ 1 \ 5 \ 8 \ 9 \ 10 \]
Execution of Graham’s Scan

\[ i = 11 \]

Diagram showing the execution of Graham's Scan algorithm.
Execution of Graham’s Scan

\[ i = 11 \]

![Diagram of Graham's Scan execution](image)
Execution of Graham’s Scan

\[ i = 11 \]

\[
\begin{array}{c}
0 \\
1 \\
5 \\
8 \\
9 \\
\end{array}
\]
Execution of Graham’s Scan

\[ i = 11 \]
Execution of Graham’s Scan

\[ i = 11 \]

\[
\begin{array}{c}
0 & 1 & 5 & 8 & 11 \\
\end{array}
\]
Execution of Graham’s Scan

\[ i = 12 \]
Execution of Graham’s Scan

\[ i = 12 \]

0 1 5 8
Execution of Graham’s Scan

\[ i = 12 \]
Execution of Graham’s Scan

\[ i = 13 \]

0 1 5 8 12 13

\begin{figure}
\centering
\includegraphics[width=\textwidth]{graham_scan_diagram.png}
\end{figure}
Execution of Graham’s Scan

\[ i = 14 \]
Execution of Graham’s Scan

\[ i = 15 \]
Execution of Graham’s Scan

\[ i = 15 \]
Execution of Graham’s Scan

\[ i = 15 \checkmark \]
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

Algorithm

Here, we rotate the coordinate system by 180°.

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

Output sensitive algorithm!
Jarvis’ March (Gift wrapping)

Intuition

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

Algorithm

Here, we rotate the coordinate system by 180°.

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

Output sensitive algorithm!

7: Geometric Algorithms
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

Algorithm

Here, we rotate the coordinate system by 180°.

Runtime: $O(n \cdot h)$, where $h$ is the number of points on the convex hull.

Output sensitive algorithm!
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

Intuition

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

Algorithm

Here, we rotate the coordinate system by $180^\circ$.

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

Output sensitive algorithm!
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

Intuition

- Let $p_0$ be the lowest point
- Next point the one with smallest angle w.r.t. $p_0$
- Continue until highest point $p_k$
- Next point the one with smallest angle w.r.t. $p_k$
- Continue until $p_0$ is reached

Algorithm

Here, we rotate the coordinate system by 180°.

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

Output sensitive algorithm!
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

Algorithm

Here, we rotate the coordinate system by 180°.

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

Output sensitive algorithm!
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

Intuition

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

Algorithm

Here, we rotate the coordinate system by 180°

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

Output sensitive algorithm!
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

Algorithm

Here, we rotate the coordinate system by 180°.

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

Output sensitive algorithm!
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

Algorithm

Here, we rotate the coordinate system by 180°.

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

Output sensitive algorithm!
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

Algorithm
Here, we rotate the coordinate system by 180°

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

Output sensitive algorithm!
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

Algorithm

Here, we rotate the coordinate system by 180°.

Runtime: $O(n \cdot h)$, where $h$ is the number of points on the convex hull.

Output sensitive algorithm!
**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

---

**Algorithm**
Here, we rotate the coordinate system by 180°.

**Runtime:** $O(n \cdot h)$, where $h$ is the number of points on the convex hull.

Output sensitive algorithm!
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

Diagram:

Here, we rotate the coordinate system by 180°.

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

Output sensitive algorithm!
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

**Algorithm**

Here, we rotate the coordinate system by 180°.

**Runtime:** $O(n \cdot h)$, where $h$ is the number of points on the convex hull.

Output sensitive algorithm!
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

Algorithm

Here, we rotate the coordinate system by 180°.

Runtime: $O(n \cdot h)$, where $h$ is the number of points on the convex hull.

Output sensitive algorithm!

7: Geometric Algorithms
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

Here, we rotate the coordinate system by 180°.

Runtime: $O(n \cdot h)$, where $h$ is the number of points on the convex hull.

Output sensitive algorithm!
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

Here, we rotate the coordinate system by 180°.

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

Output sensitive algorithm!
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)

<table>
<thead>
<tr>
<th>Intuition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Wrapping taut paper around the points</td>
</tr>
<tr>
<td>1. Tape end of paper at lowest point</td>
</tr>
<tr>
<td>2. Pull paper to the right until it touches a point</td>
</tr>
<tr>
<td>3. Tape paper and go to 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let $p_0$ be the lowest point</td>
</tr>
<tr>
<td>2. Next point the one with smallest angle w.r.t. $p_0$</td>
</tr>
<tr>
<td>3. Continue until highest point $p_k$</td>
</tr>
</tbody>
</table>
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

Diagram showing a set of points on a 2D plane.
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

7: Geometric Algorithms
Execution of Jarvis’ March
Execution of Jarvis’ March
Execution of Jarvis’ March

7: Geometric Algorithms
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

- **Natural Backtracking Algorithm**
  - Cross-product avoids computing polar angles
  - Runtime dominated by sorting: $O(n \log n)$

- **Graham's Scan**
  - Proceeds like wrapping a gift
  - Runtime: $O(nh)$, output-sensitive

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

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

- **Cross Product** is a very powerful tool (avoids trigonometry and division!)
- Take care of degenerate cases

---

**Lessons Learned**

7: Geometric Algorithms
Computing Convex Hull: Summary

Graham's Scan

- Natural backtracking algorithm
- Cross-product avoids computing polar angles
- Runtime dominated by sorting: $O(n \log n)$

- Graham's Scan proceeds like wrapping a gift
- Runtime $O(nh) \Rightarrow$ output-sensitive

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

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

- Cross product very powerful tool (avoids trigonometry and division!)

Lessons Learned
Computing Convex Hull: Summary

- **Graham's Scan**
  - natural backtracking algorithm

- **Runtime** $O(n \log n)$

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

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

- Cross product: very powerful tool (avoids trigonometry and division!)

- Take care of degenerate cases

**Lessons Learned**
Computing Convex Hull: Summary

Graham's Scan
- natural backtracking algorithm
- cross-product avoids computing polar angles

Lessons Learned

7: Geometric Algorithms

T.S.
Computing Convex Hull: Summary

Graham's Scan

- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\sim O(n \log n)$
Computing Convex Hull: Summary

- **Graham’s Scan**
  - natural backtracking algorithm
  - cross-product avoids computing polar angles
  - Runtime dominated by sorting $\sim O(n \log n)$

- **Jarvis’ March**

Lessons Learned

7: Geometric Algorithms
Computing Convex Hull: Summary

Graham’s Scan
- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\Rightarrow O(n \log n)$

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 $\approx O(n \log n)$

**Jarvis’ March**
- proceeds like wrapping a gift
- Runtime $O(nh) \approx$ output-sensitive
Computing Convex Hull: Summary

Graham’s Scan
- natural backtracking algorithm
- cross-product avoids computing polar angles
- Runtime dominated by sorting $\sim O(n \log n)$

Jarvis’ March
- proceeds like wrapping a gift
- Runtime $O(nh) \sim \text{output-sensitive}$

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 $\sim O(n \log n)$

**Jarvis’ March**
- proceeds like wrapping a gift
- Runtime $O(nh)$ $\sim$ 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 $\sim O(n \log n)$

Jarvis' March
- proceeds like wrapping a gift
- Runtime $O(nh) \sim$ 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 \( \sim O(n \log n) \)

**Jarvis’ March**
- proceeds like wrapping a gift
- Runtime \( O(nh) \) \( \sim \) 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 division!)
Computing Convex Hull: Summary

**Graham's Scan**
- natural **backtracking** algorithm
- **cross-product avoids** computing polar angles
- Runtime dominated by sorting $\sim O(n \log n)$

**Jarvis' March**
- proceeds like **wrapping a gift**
- Runtime $O(nh) \sim$ 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 division!)
- take care of **degenerate cases**
Outline

Introduction and Line Intersection

Convex Hull

Glimpse at (More) Advanced Algorithms
maximize \[ 3x_1 + x_2 + 2x_3 \]

subject to

\[
\begin{align*}
30 & \geq x_1 + x_2 + 3x_3 \\
24 & \geq 2x_1 + 2x_2 + 5x_3 \\
36 & \geq 4x_1 + x_2 + 2x_3 \\
0 & \geq x_1, x_2, x_3
\end{align*}
\]
Linear Programming and Simplex

maximize 3x₁ + x₂ + 2x₃

subject to

x₁ + x₂ + 3x₃ ≤ 30
2x₁ + 2x₂ + 5x₃ ≤ 24
4x₁ + x₂ + 2x₃ ≤ 36
x₁, x₂, x₃ ≥ 0
maximize \( 3x_1 + x_2 + 2x_3 \)
subject to
\[
\begin{align*}
x_1 + x_2 + 3x_3 & \leq 30 \\
2x_1 + 2x_2 + 5x_3 & \leq 24 \\
4x_1 + x_2 + 2x_3 & \leq 36 \\
x_1, x_2, x_3 & \geq 0
\end{align*}
\]
maximize $3x_1 + x_2 + 2x_3$
subject to

$$x_1 + x_2 + 3x_3 \leq 30$$
$$2x_1 + 2x_2 + 5x_3 \leq 24$$
$$4x_1 + x_2 + 2x_3 \leq 36$$

$x_1, x_2, x_3 \geq 0$
maximize $3x_1 + x_2 + 2x_3$

subject to

$x_1 + x_2 + 3x_3 \leq 30$
$2x_1 + 2x_2 + 5x_3 \leq 24$
$4x_1 + x_2 + 2x_3 \leq 36$
$x_1, x_2, x_3 \geq 0$
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,\textsuperscript{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.
4. Cleveland, Ohio
7. Indianapolis, Ind.
8. Chicago, Ill.
9. Milwaukee, Wis.
10. Minneapolis, Minn.
11. Pierre, S. D.
12. Bismarck, N. D.
13. Helena, Mont.
15. Portland, Ore.
16. Boise, Idaho
17. Salt Lake City, Utah
18. Carson City, Nev.
19. Los Angeles, Calif.
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.
40. Washington, D. C.
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.
### 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.

<table>
<thead>
<tr>
<th>Cities</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>170</td>
</tr>
<tr>
<td>2</td>
<td>171</td>
</tr>
<tr>
<td>3</td>
<td>172</td>
</tr>
<tr>
<td>4</td>
<td>173</td>
</tr>
<tr>
<td>5</td>
<td>174</td>
</tr>
<tr>
<td>6</td>
<td>175</td>
</tr>
<tr>
<td>7</td>
<td>176</td>
</tr>
<tr>
<td>8</td>
<td>177</td>
</tr>
<tr>
<td>9</td>
<td>178</td>
</tr>
<tr>
<td>10</td>
<td>179</td>
</tr>
<tr>
<td>11</td>
<td>180</td>
</tr>
<tr>
<td>12</td>
<td>181</td>
</tr>
<tr>
<td>13</td>
<td>182</td>
</tr>
<tr>
<td>14</td>
<td>183</td>
</tr>
<tr>
<td>15</td>
<td>184</td>
</tr>
<tr>
<td>16</td>
<td>185</td>
</tr>
<tr>
<td>17</td>
<td>186</td>
</tr>
<tr>
<td>18</td>
<td>187</td>
</tr>
<tr>
<td>19</td>
<td>188</td>
</tr>
<tr>
<td>20</td>
<td>189</td>
</tr>
<tr>
<td>21</td>
<td>190</td>
</tr>
<tr>
<td>22</td>
<td>191</td>
</tr>
<tr>
<td>23</td>
<td>192</td>
</tr>
<tr>
<td>24</td>
<td>193</td>
</tr>
<tr>
<td>25</td>
<td>194</td>
</tr>
<tr>
<td>26</td>
<td>195</td>
</tr>
<tr>
<td>27</td>
<td>196</td>
</tr>
<tr>
<td>28</td>
<td>197</td>
</tr>
<tr>
<td>29</td>
<td>198</td>
</tr>
<tr>
<td>30</td>
<td>199</td>
</tr>
<tr>
<td>31</td>
<td>200</td>
</tr>
<tr>
<td>32</td>
<td>201</td>
</tr>
<tr>
<td>33</td>
<td>202</td>
</tr>
<tr>
<td>34</td>
<td>203</td>
</tr>
<tr>
<td>35</td>
<td>204</td>
</tr>
<tr>
<td>36</td>
<td>205</td>
</tr>
<tr>
<td>37</td>
<td>206</td>
</tr>
<tr>
<td>38</td>
<td>207</td>
</tr>
<tr>
<td>39</td>
<td>208</td>
</tr>
<tr>
<td>40</td>
<td>209</td>
</tr>
<tr>
<td>41</td>
<td>210</td>
</tr>
</tbody>
</table>

7: Geometric Algorithms

T.S. 22
The (Unique) Optimal Tour (699 Units \(\approx 12,345\) miles)

This tour has a length of 12,345 miles when the adjusted units are expressed in 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, 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>
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Solution Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_2_1</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_42_1</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_3_2</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_4_3</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_5_4</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_6_5</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_7_6</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_8_7</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_9_8</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_10_9</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_11_10</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_12_11</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_13_12</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_14_13</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_15_14</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_16_15</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_17_16</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_18_17</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_19_18</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_20_19</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_21_20</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_22_21</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_23_22</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_24_23</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_25_24</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_26_25</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_27_26</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_28_27</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_29_28</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_30_29</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_31_30</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_32_31</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_33_32</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_34_33</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_35_34</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_36_35</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_37_36</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_38_37</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_39_38</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_40_39</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_41_40</td>
<td>1.00000</td>
</tr>
<tr>
<td>x_42_41</td>
<td>1.00000</td>
</tr>
</tbody>
</table>

All other variables in the range 1-861 are 0.
Iteration 10, Branch b $x(13, 12) = 0$: Objective 701
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