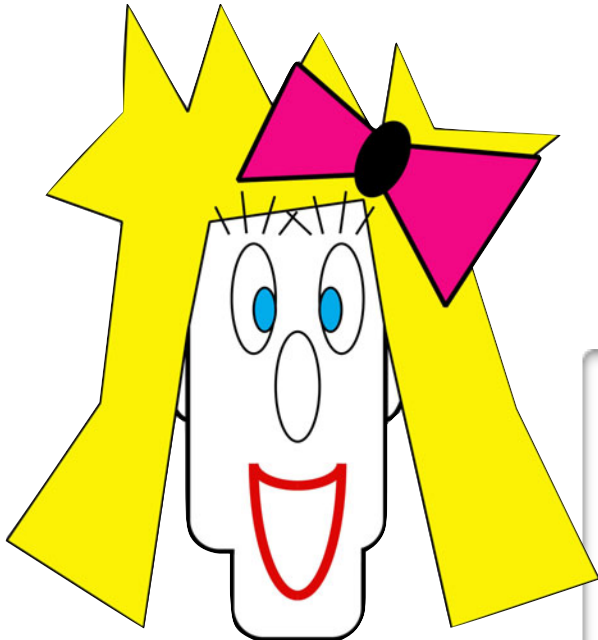
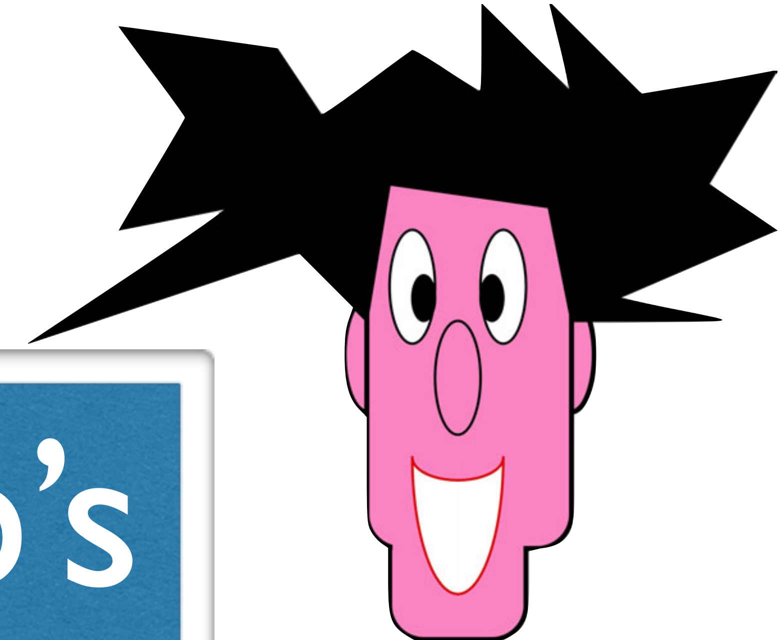
Winter 2016
COMP-250: Introduction to Computer Science

Lecture 15, March 8, 2016



Alice

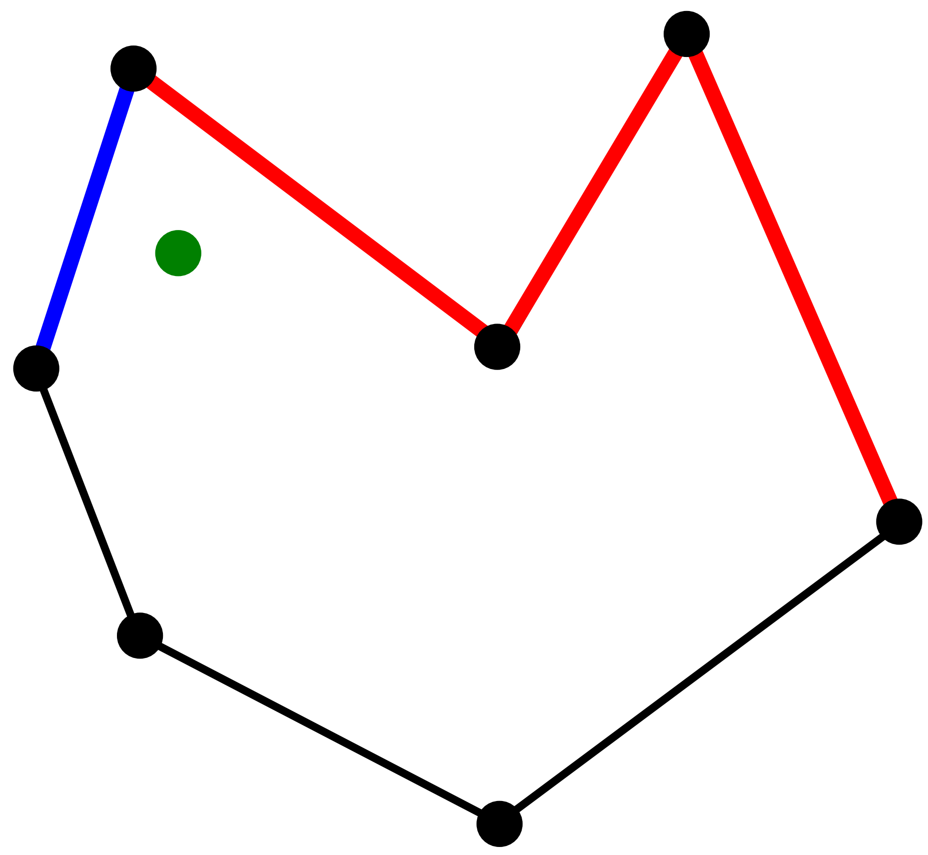
Alice and Bob's Adventures in GEOM-land...



Bob

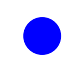
GEOMETRIC ALGORITHMS

- segment intersection
- orientation
- point inclusion
- simple closed path

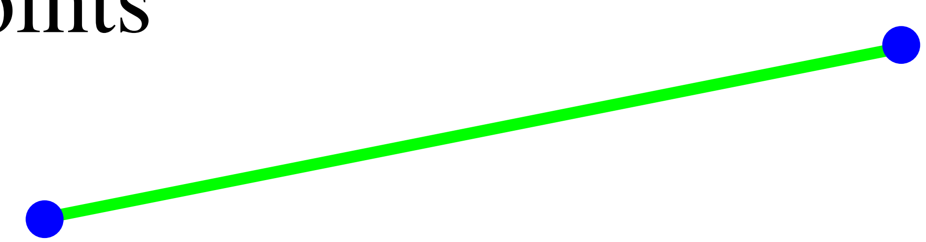


Basic Geometric Objects in the Plane

point: defined by a pair of coordinates (x,y)

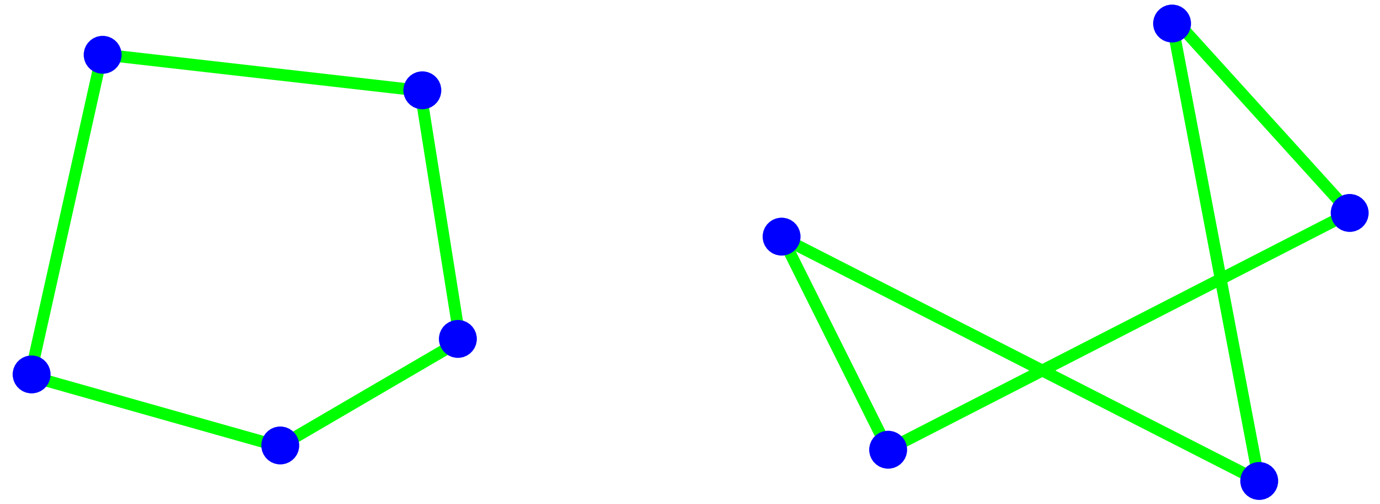


segment: portion of a straight line between two points



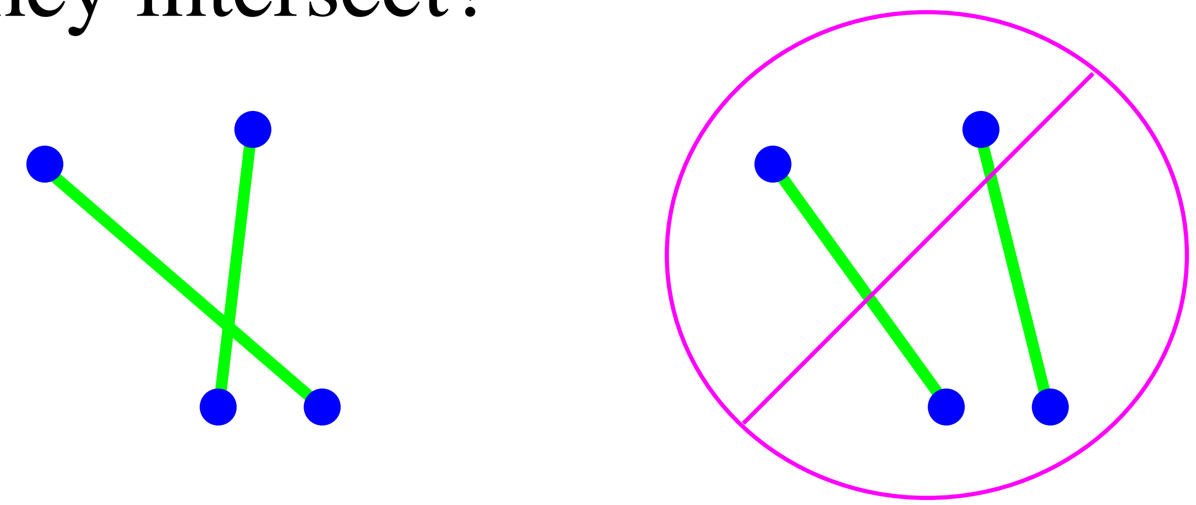
Basic Geometric Objects in the Plane

polygon: a circular sequence of points (vertices) and segments (edges) between them



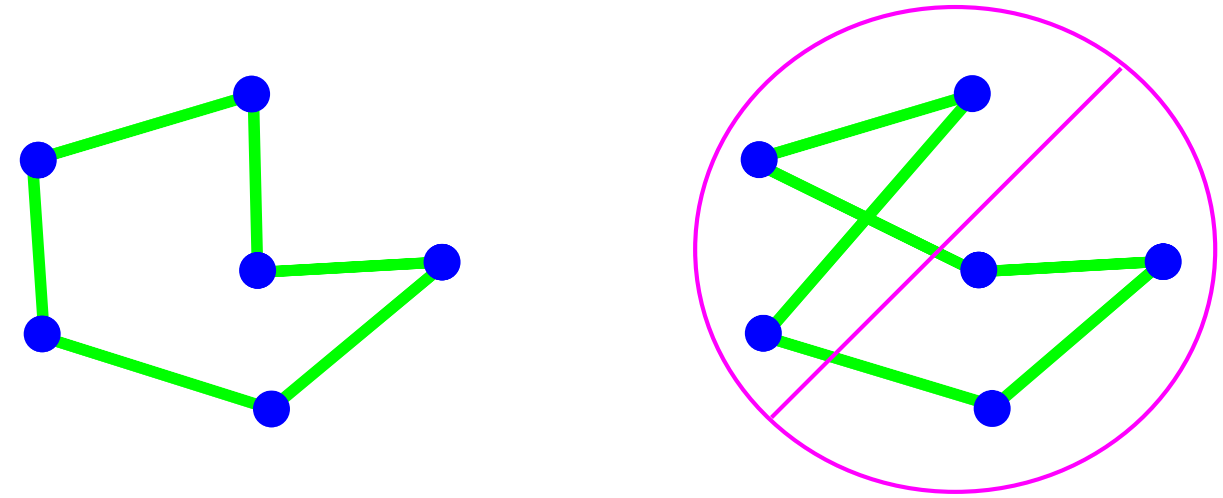
Some Geometric Problems

Segment intersection: Given two segments, do they intersect?



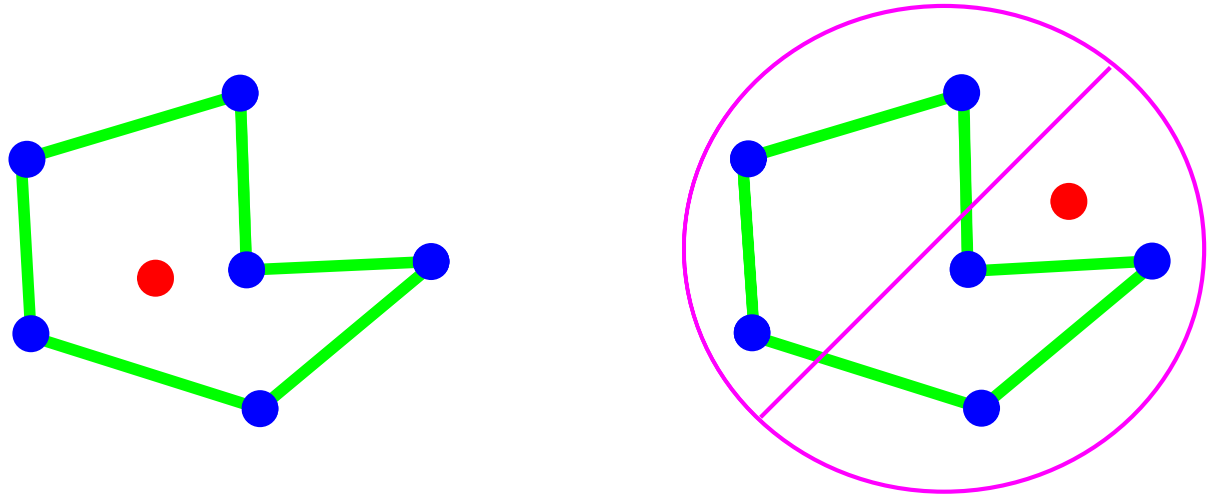
Some Geometric Problems

Simple closed path: Given a set of points, find a nonintersecting polygon with vertices on the points.



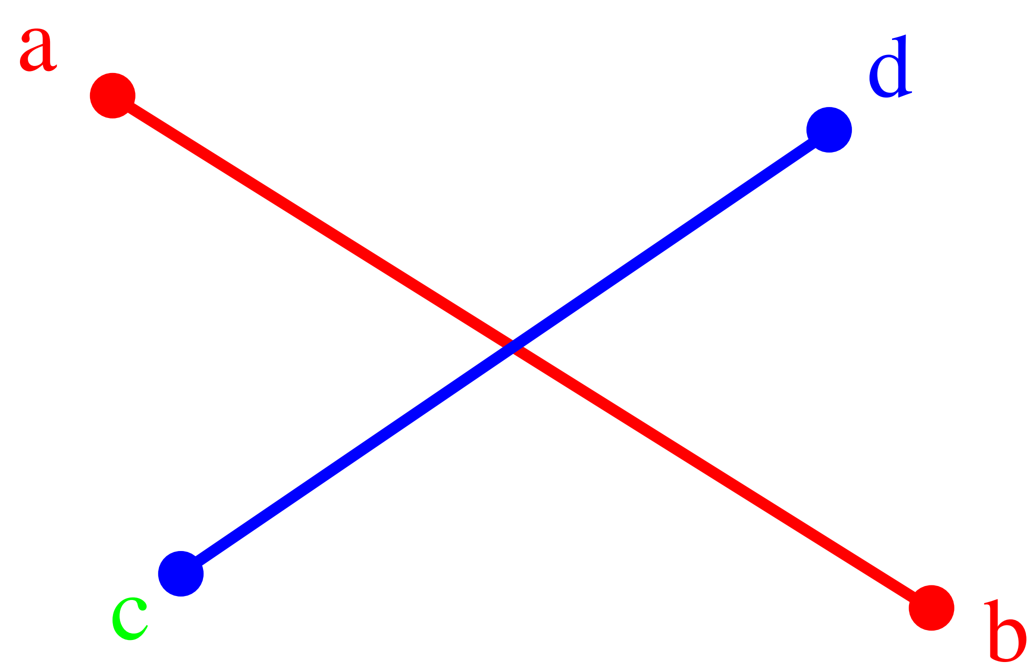
Some Geometric Problems

Inclusion in polygon: Is a point inside or outside a polygon?



An Apparently Simple Problem: Segment Intersection

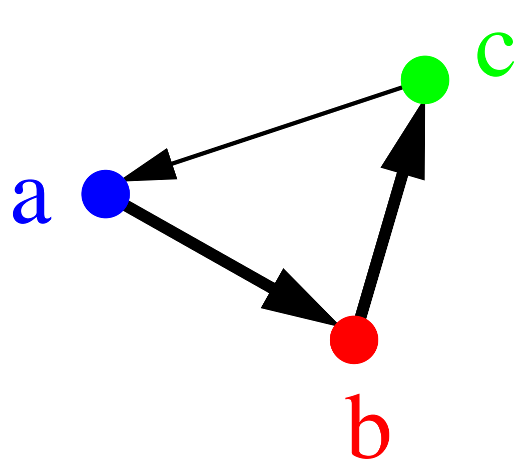
- Test whether segments (a,b) and (c,d) intersect.
  How do we do it?



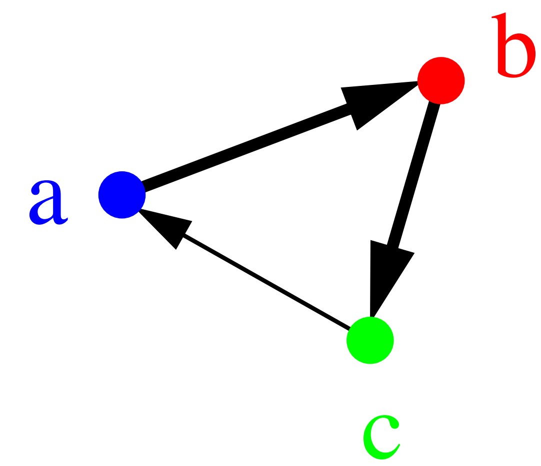
- We could start by writing down the equations of the lines through the segments, then test whether the lines intersect, then ...
- An alternative (and simpler) approach is based in the notion of **orientation** of an ordered triplet of points in the plane

Orientation in the Plane

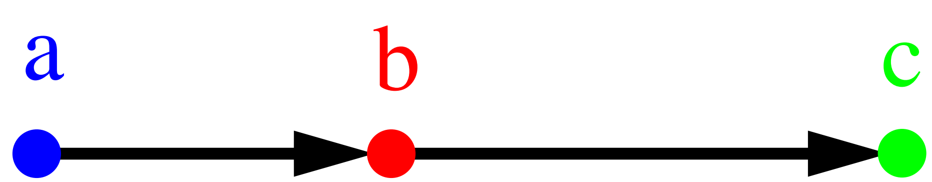
- The orientation of an ordered triplet of points in the plane can be



counterclockwise (**left turn**)



clockwise (**right turn**)

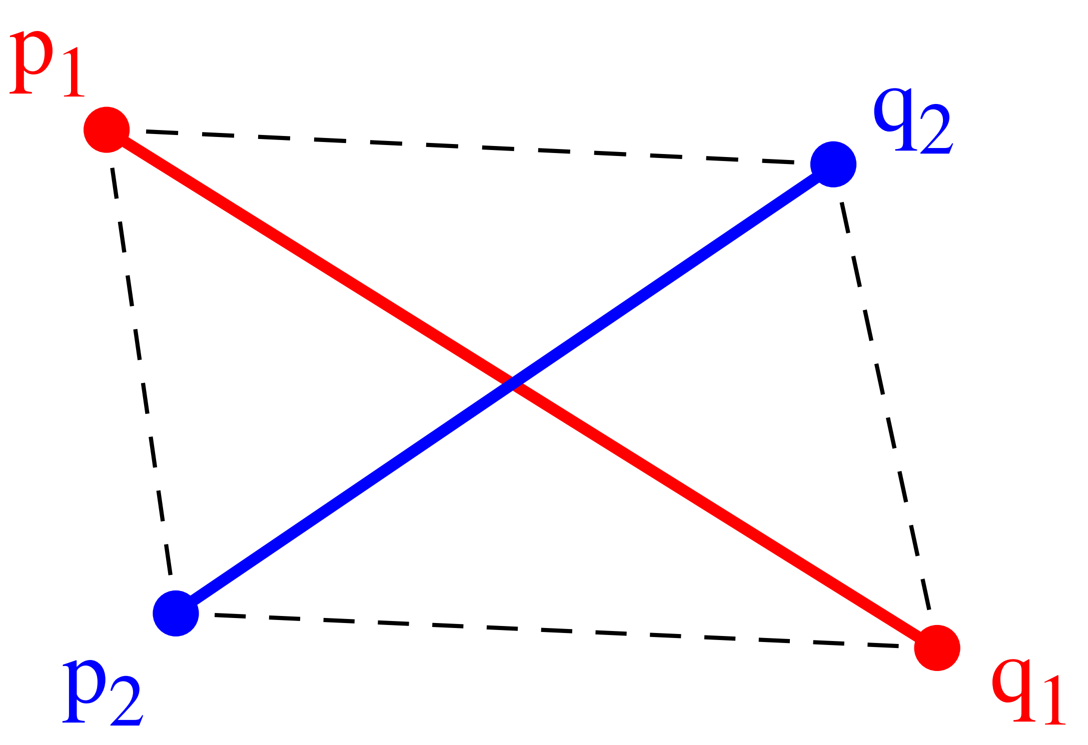


collinear (**no turn**)

Intersection and Orientation

Two segments (p_1,q_1) and (p_2,q_2) intersect if and only if one of the following two conditions is verified

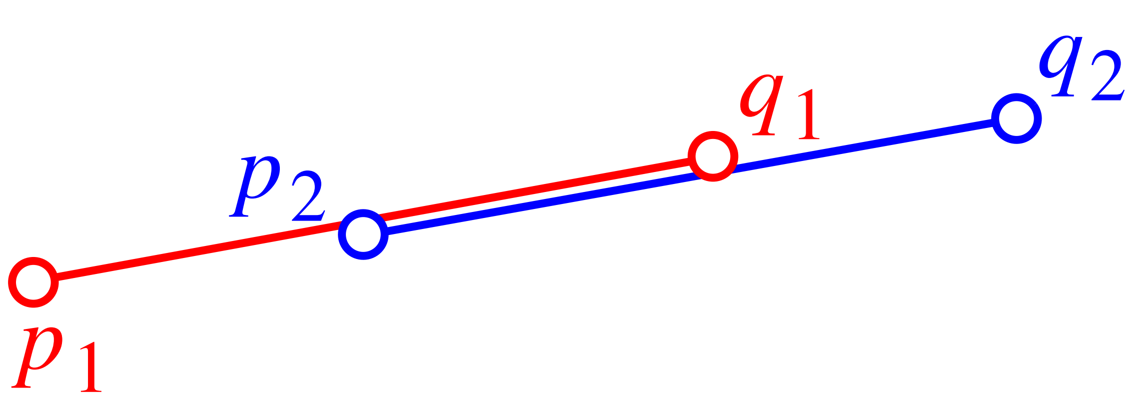
- general case:
  - (p_1,q_1,p_2) and (p_1,q_1,q_2) have different orientations **and**
  - (p_2,q_2,p_1) and (p_2,q_2,q_1) have different orientations



Intersection and Orientation

Two segments (p_1,q_1) and (p_2,q_2) intersect if and only if one of the following two conditions is verified

- special case
  - (p_1,q_1,p_2), (p_1,q_1,q_2), (p_2,q_2,p_1), and (p_2,q_2,q_1) are all collinear **and**
  - the *x*-projections of (p_1,q_1) and (p_2,q_2) intersect
  - the *y*-projections of (p_1,q_1) and (p_2,q_2) intersect



(p_1,q_1,p_2)

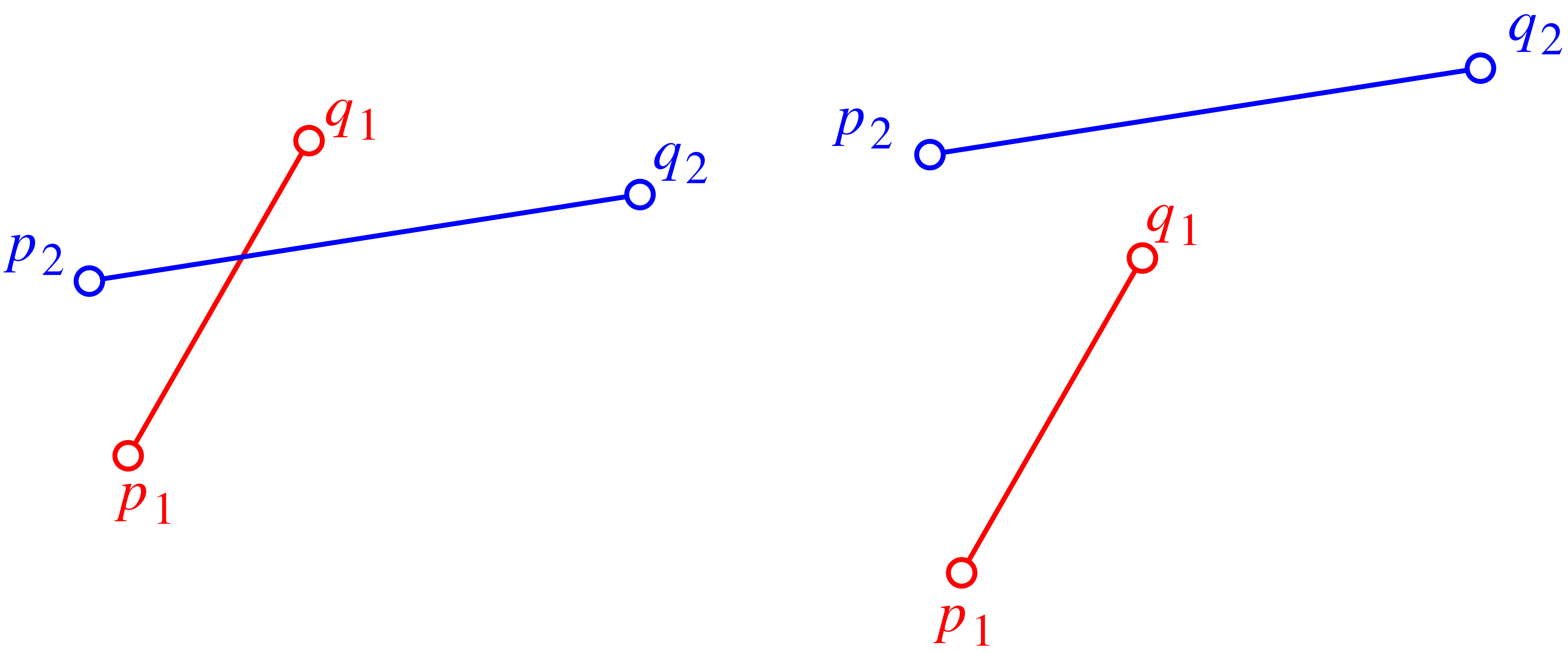
(p_1,q_1,q_2)

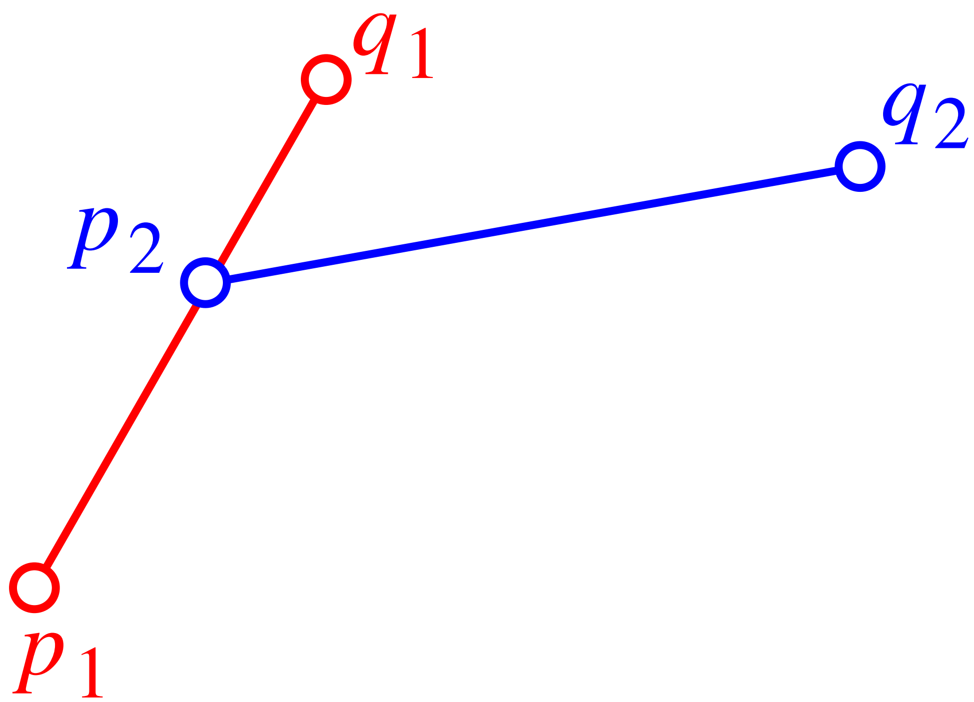
(p_2,q_2,p_1)

(p_2,q_2,q_1)

Examples (General Case)

- general case:
  - (p_1, q_1, p_2) and (p_1, q_1, q_2) have different orientations **and**
  - (p_2, q_2, p_1) and (p_2, q_2, q_1) have different orientations



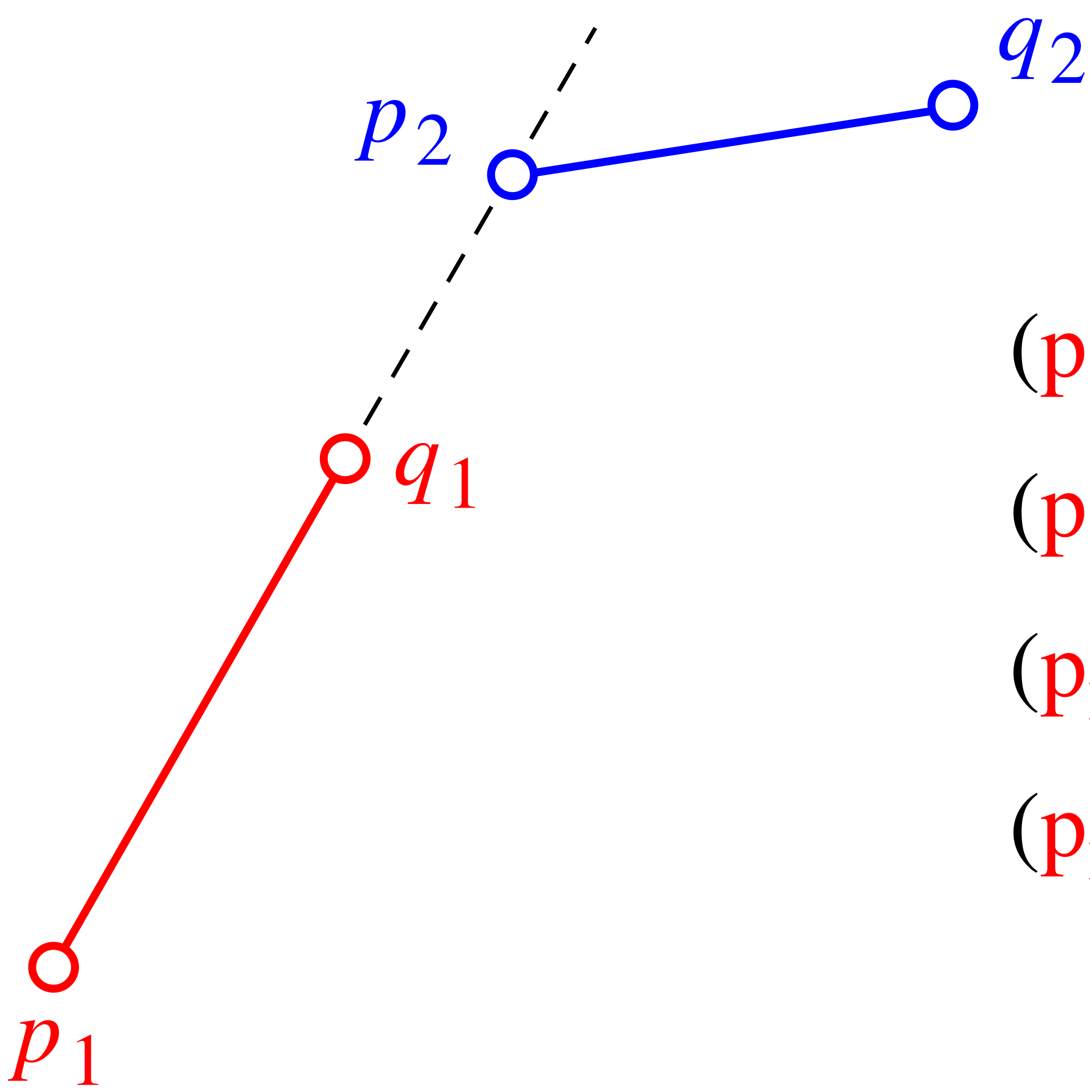


(p_1, q_1, p_2)

(p_1, q_1, q_2)

(p_2, q_2, p_1)

(p_2, q_2, q_1)



(p_1, q_1, p_2)

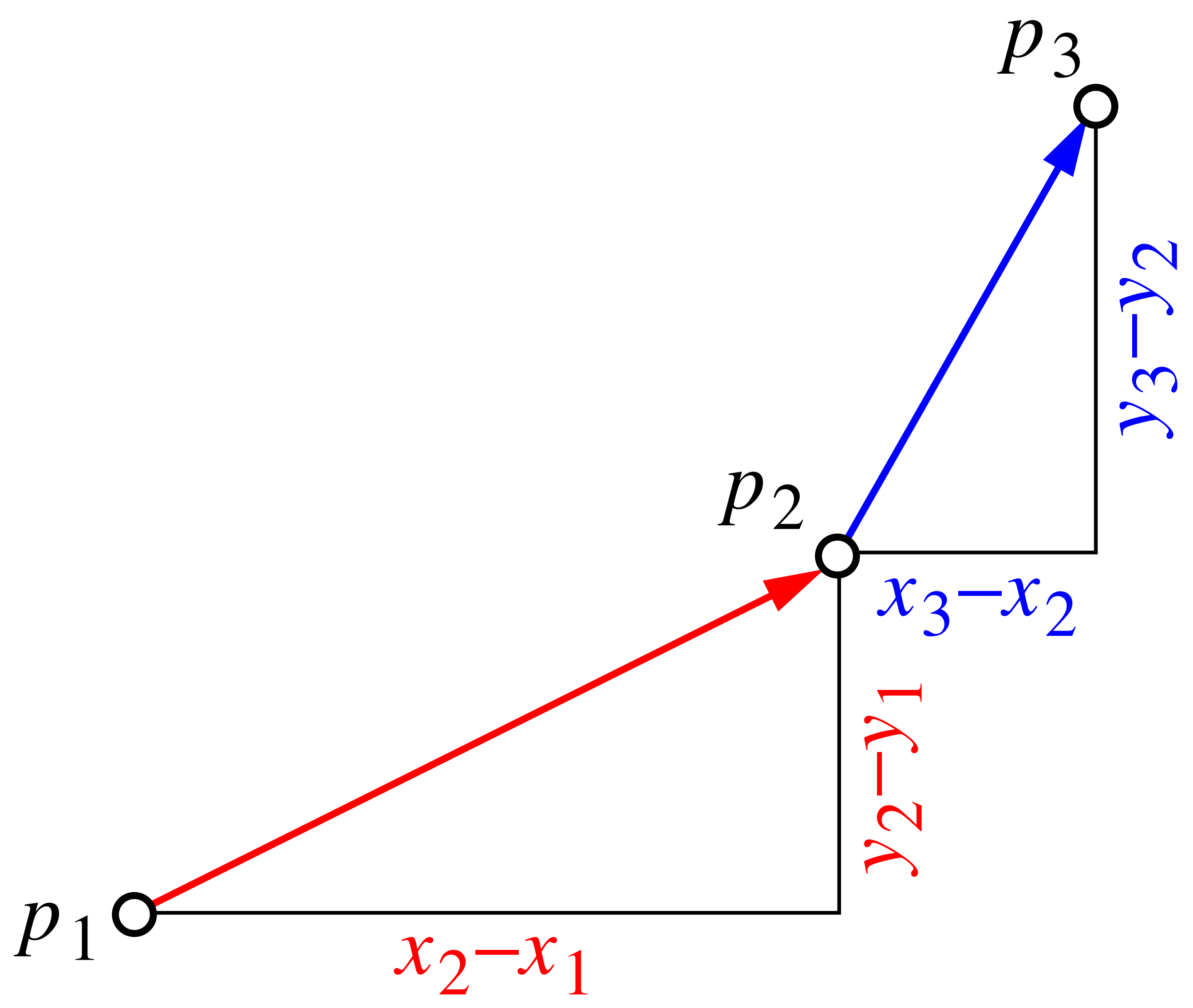
(p_1, q_1, q_2)

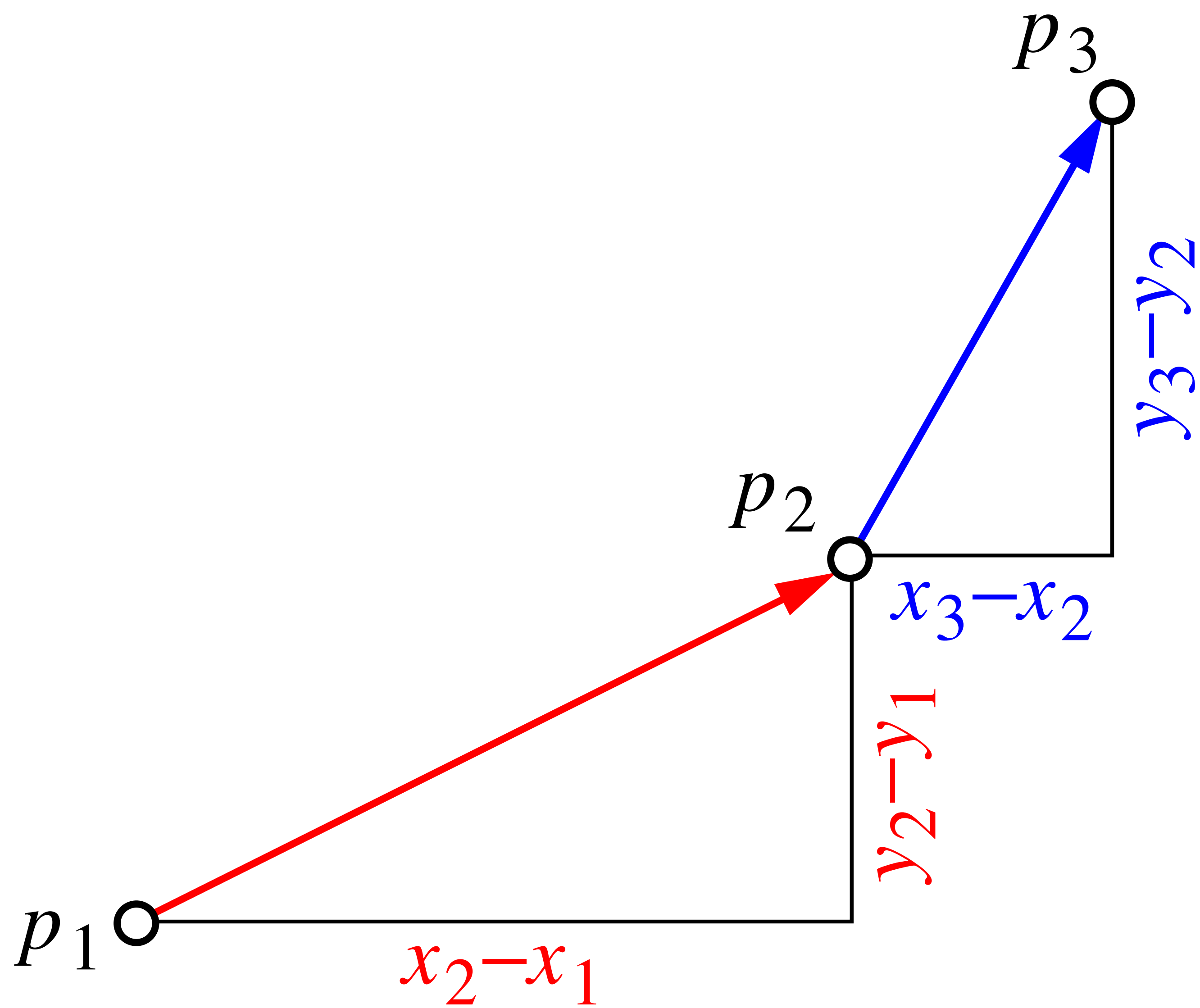
(p_2, q_2, p_1)

(p_2, q_2, q_1)

How to Compute the Orientation

- slope of segment (p_1,p_2): $\sigma = (y_2 - y_1) / (x_2 - x_1)$
- slope of segment (p_2,p_3): $\tau = (y_3 - y_2) / (x_3 - x_2)$



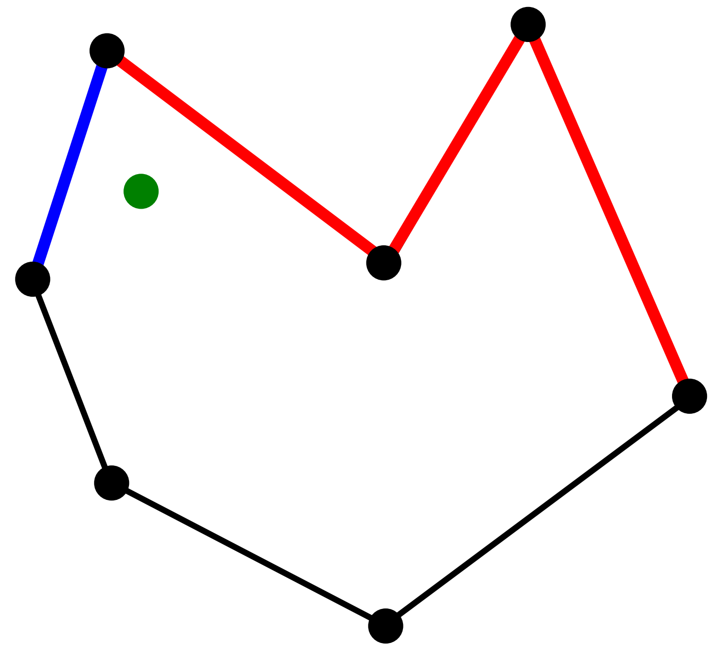


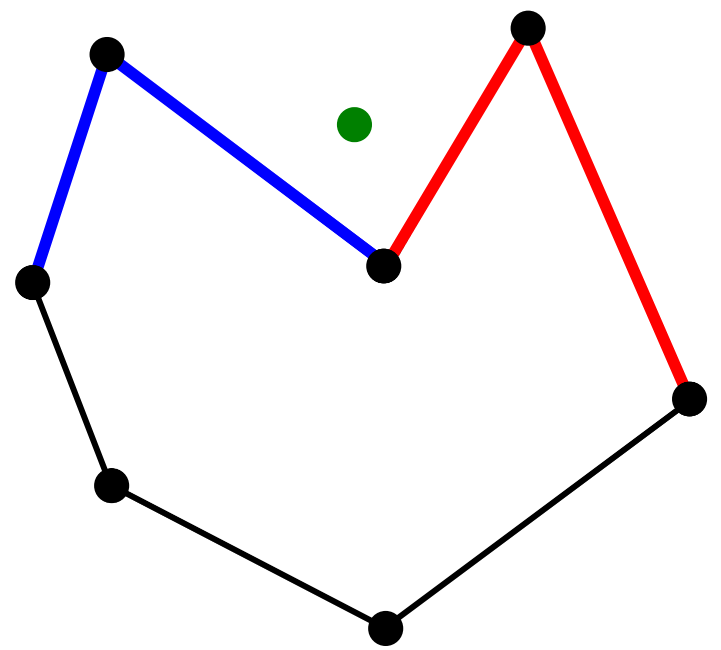
- Orientation test
  - counterclockwise (left turn): $\sigma < \tau$
  - clockwise (right turn): $\sigma > \tau$
  - collinear (left turn): $\sigma = \tau$

- The orientation depends on whether the expression
  $(y_2 - y_1)\ (x_3 - x_2) - (y_3 - y_2)\ (x_2 - x_1)$
  is positive, negative, or zero.

Point Inclusion

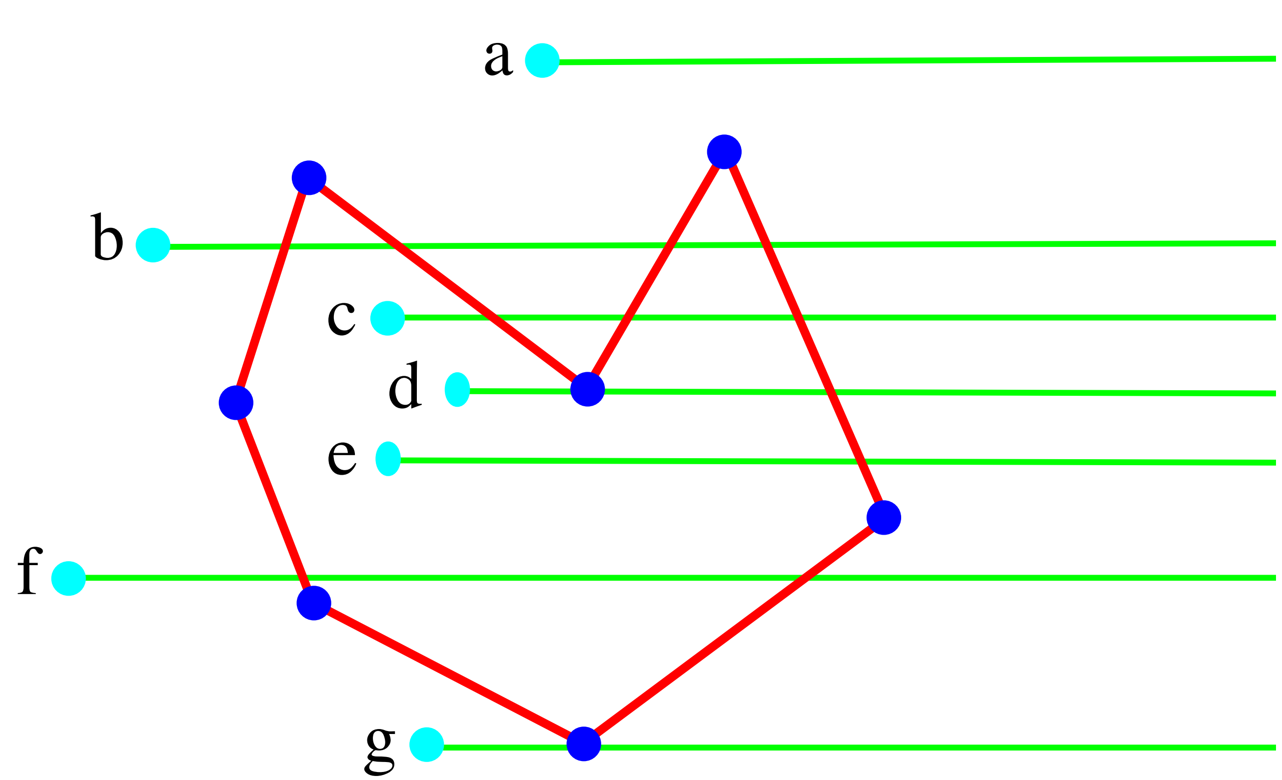
- given a polygon and a point, is the point inside or outside the polygon?
- orientation helps solving this problem in linear time





Point Inclusion — Part II

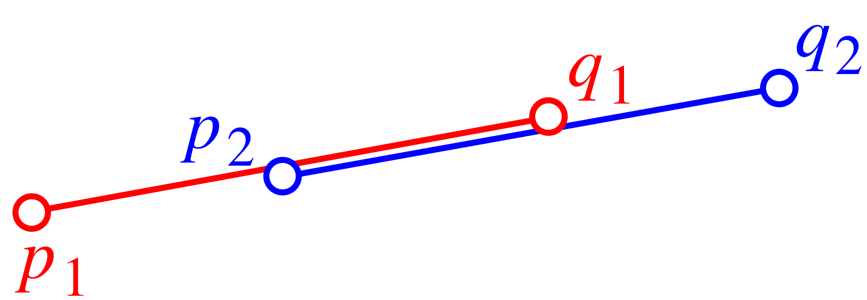
- Draw a horizontal line to the right of each point and extend it to infinity
- Count the number of times a line intersects the polygon. We have:
  - even number $\Rightarrow$ point is outside
  - odd number $\Rightarrow$ point is inside
- Why?



- What about points d and g ?? Degeneracy!

Degeneracy

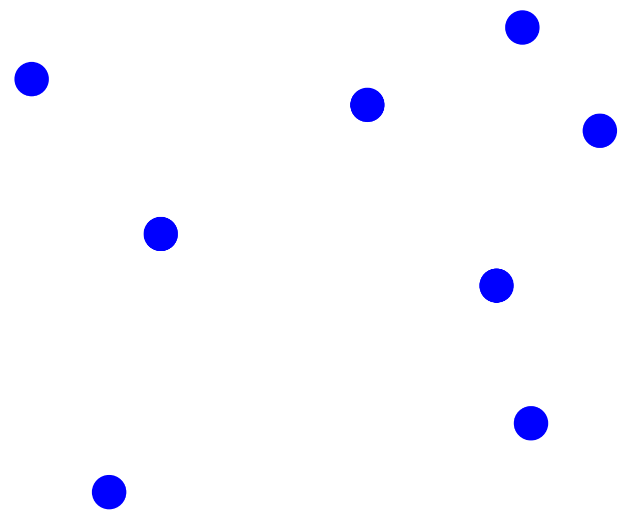
- Degeneracies are input configurations that involve tricky special cases.
- When implementing an algorithm, degeneracies should be taken care of separately -- the general algorithm might fail to work.
- For example, in the previous example where we had to determine whether two segments intersect, we have degeneracy if two segments are collinear.



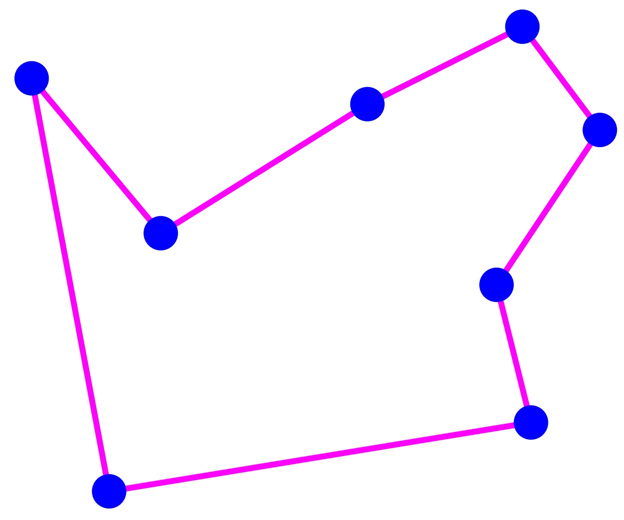
- The general algorithm of checking for orientation would fail to distinguish whether the two segments intersect. Hence, this case should be dealt with separately.

Simple Closed Path — Part I

- Problem: Given a set of points ...

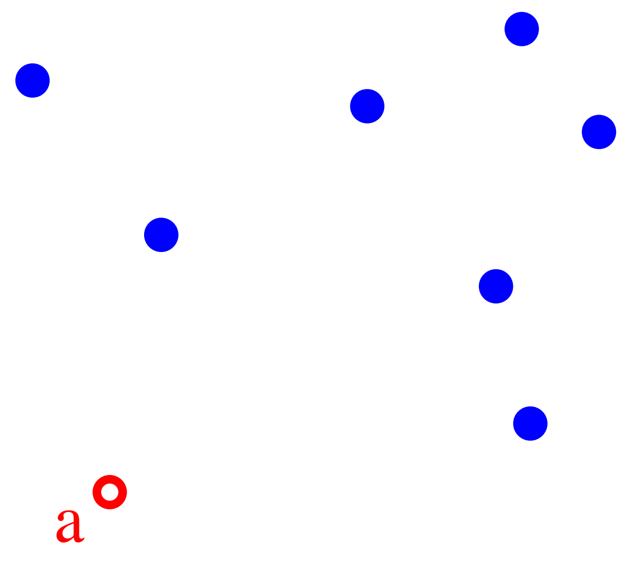


- “Connect the dots” without crossings

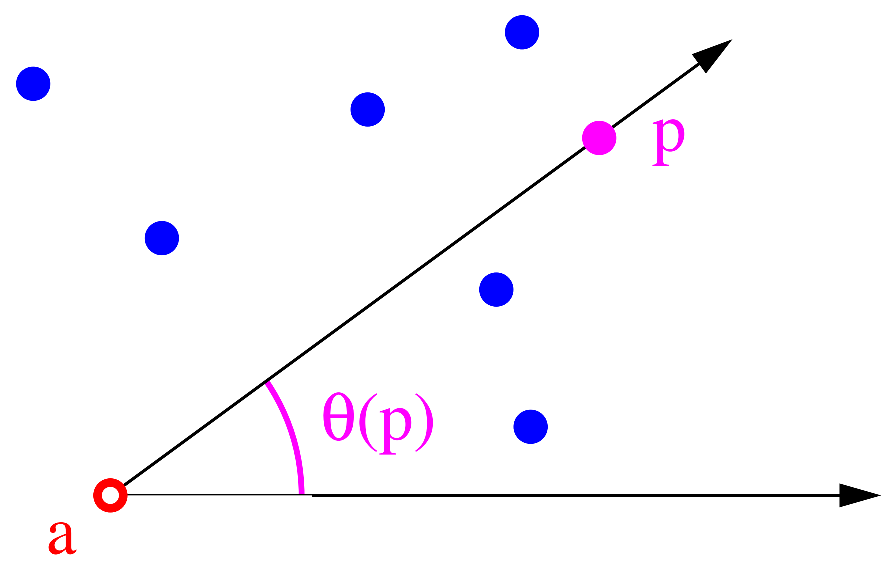


Simple Closed Path — Part II

- Pick the bottommost point a as the anchor point

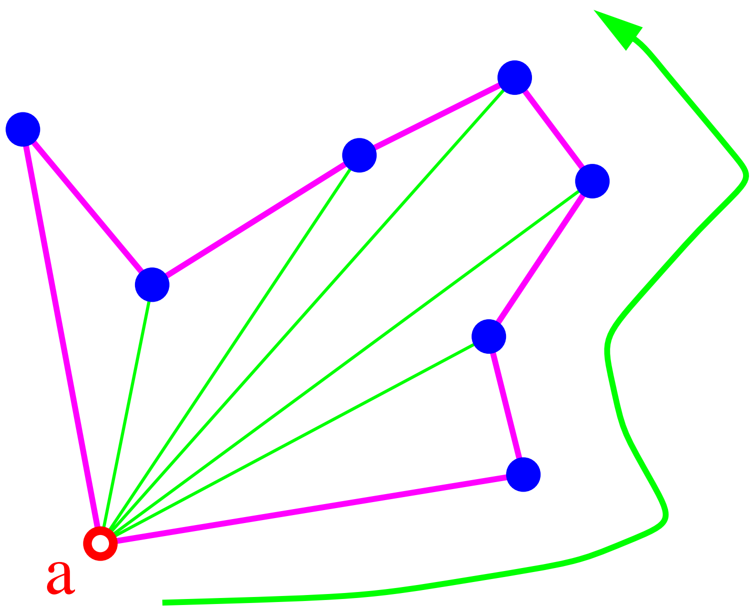


- For each point p, compute the angle θ(p) of the segment (a,p) with respect to the x-axis:



Simple Closed Path — Part III

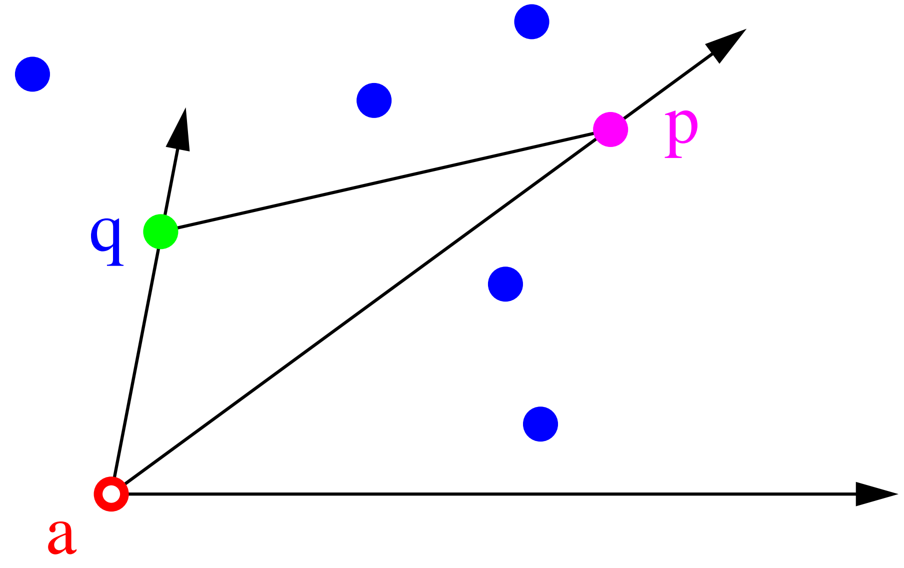
- Traversing the points by increasing angle yields a simple closed path:



- The question is: how do we compute angles?
  - We could use trigonometry (e.g., arctan).
  - However, the computation would be inefficient since trigonometric functions are not in the normal instruction set of a computer and need a call to a math-library routine.
  - Observation:, we don't care about the actual values of the angles. We just want to sort by angle.
  - Idea: use orientation to compare angles without actually computing them!!

Simple Closed Path — Part IV

- Orientation can be used to compare angles without actually computing them ... Cool!



$\theta(p) < \theta(q) \Leftrightarrow$ orientation of (a,p,q) is counterclockwise

- We can sort the points by angle by using any "sorting-by-comparison" algorithm (e.g., heapsort or merge-sort) and replacing angle comparisons with orientation tests

- We obtain an O(N log N)-time algorithm for the simple closed path problem on N points

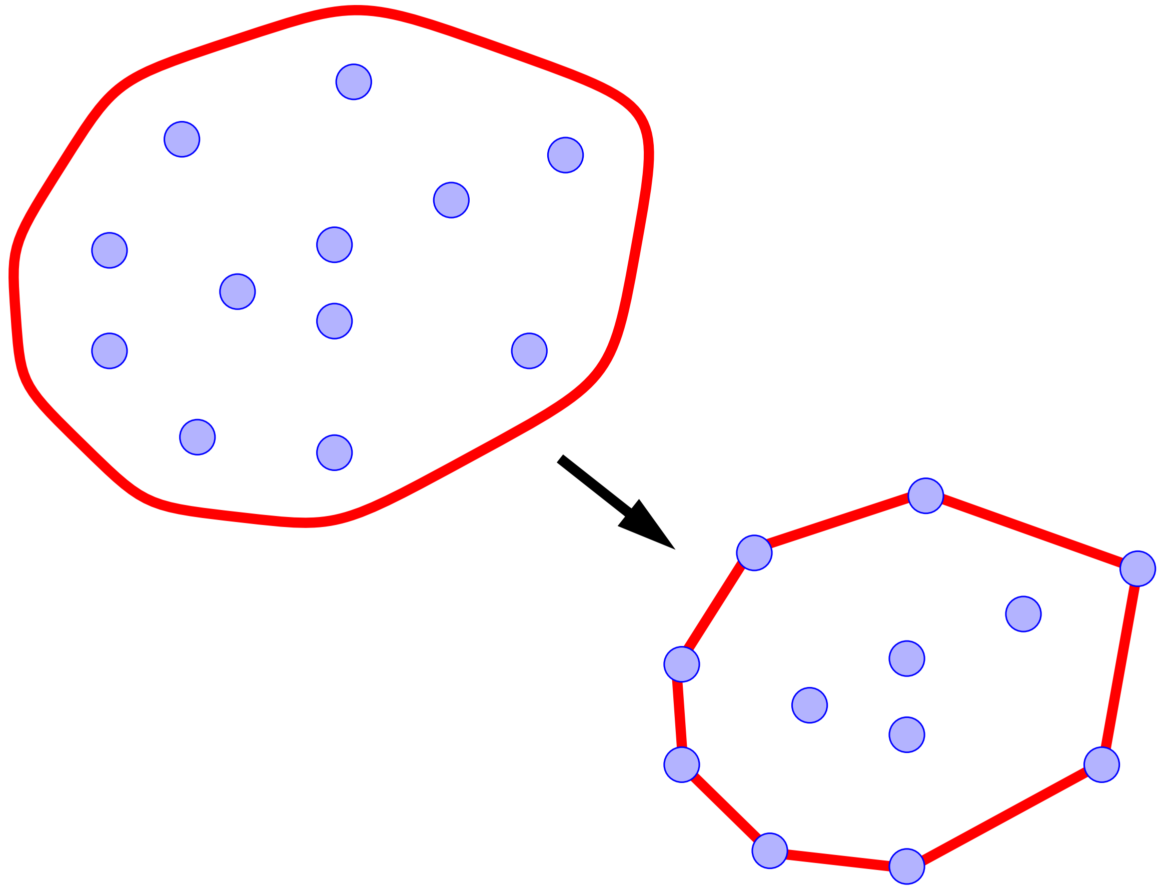
Convex HULL

- Convexity
- Package-Wrap Algorithm
- Graham Scan

What is the Convex Hull?

Let ***S*** be a set of points in the plane.

Intuition: Imagine the points of ***S*** as being pegs; the ***convex hull*** of ***S*** is the shape of a rubber-band stretched around the pegs.



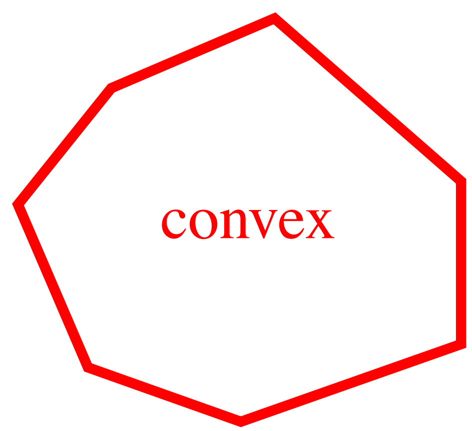
Formal definition: the ***convex hull*** of ***S*** is the smallest convex polygon that contains all the points of ***S***

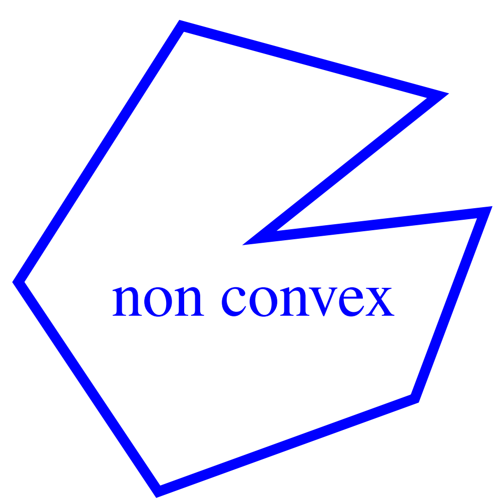
Convexity

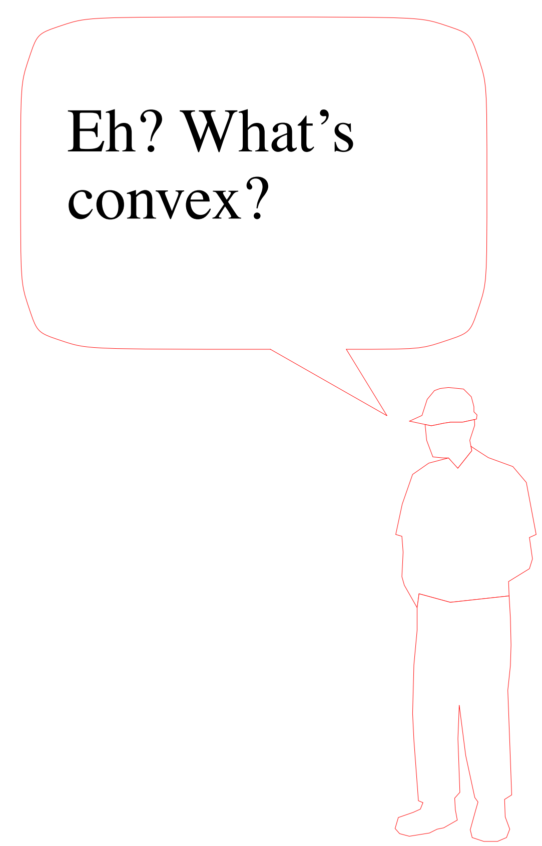
You know what ***convex*** means, right?

A polygon ***P*** is said to be ***convex*** if:

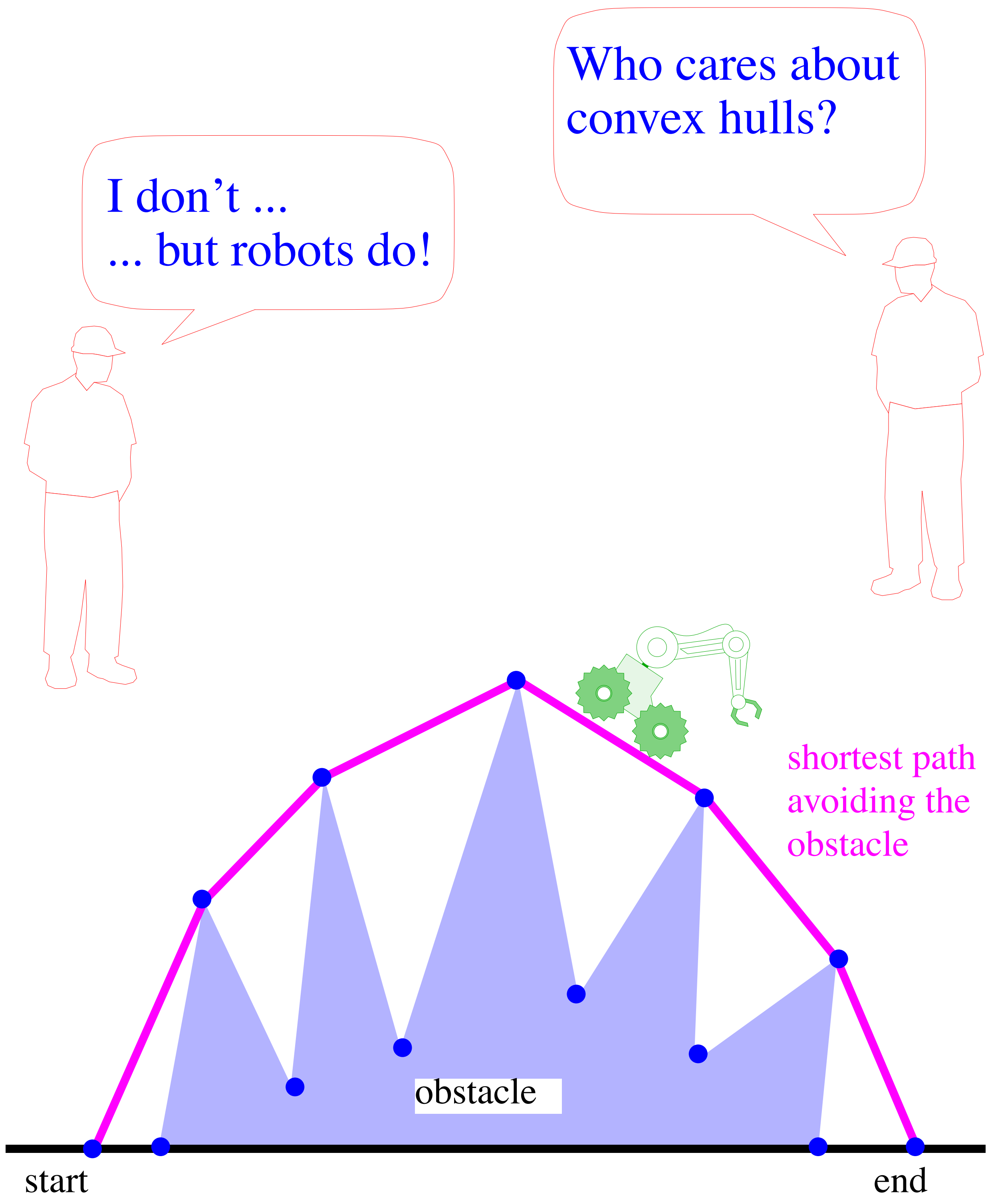
1. ***P*** is non-intersecting; and
2. for any two points ***p*** and ***q*** on the boundary of ***P***, segment ***pq*** lies entirely inside ***P***



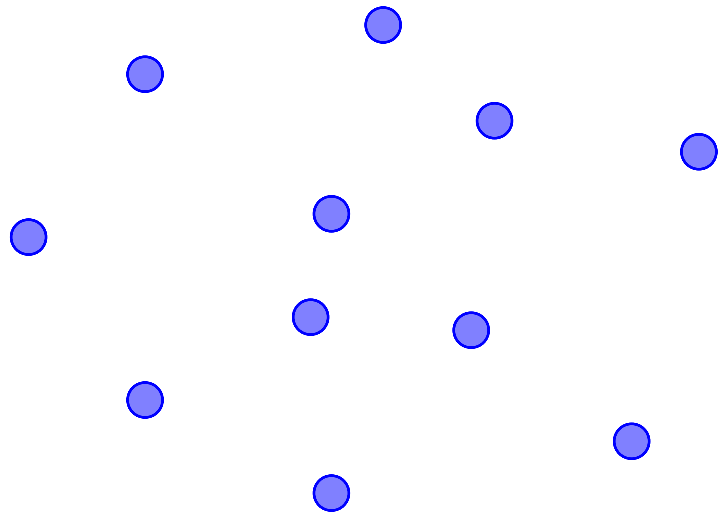


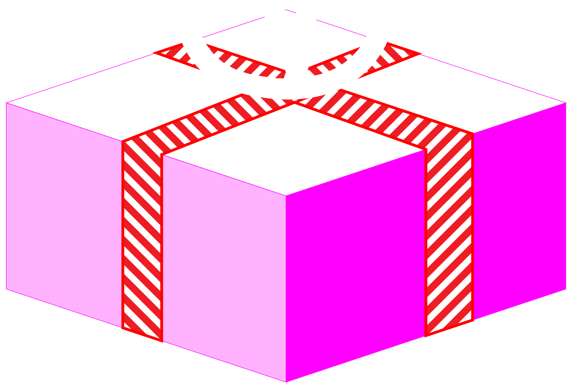


Why Convex Hulls?



The Package Wrapping Algorithm

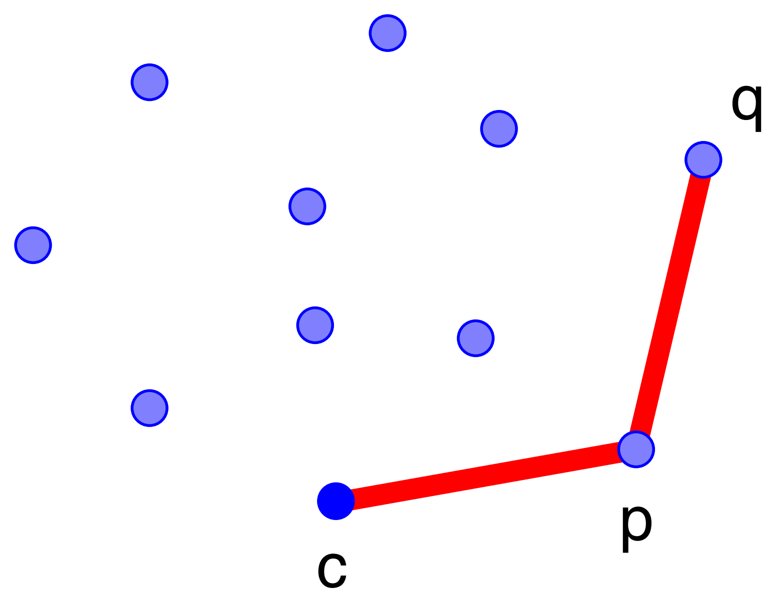




Package Wrap

- given the current point, how do we compute the next point?
- set up an orientation tournament using the current point as the anchor-point...
- the next point is selected as the point that beats all other points at CCW orientation, i.e., for any other point, we have

orientation(c, p, q) = CCW



Time Complexity of Package Wrap

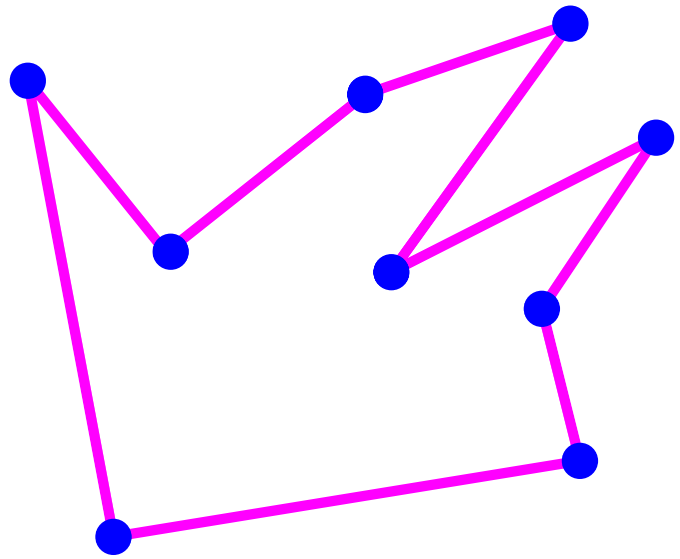
- For every point on the hull we examine all the other points to determine the next point
- Notation:
  - N: number of points
  - M: number of hull points ($M \leq N$)
- Time complexity:
  - $\Theta(MN)$
- Worst case: $\Theta(N^2)$
  - all the points are on the hull ($M=N$)
- Average case: $\Theta(N \log N)$ — $\Theta(N^{4/3})$
  - for points randomly distributed inside a ***square***, $M = \Theta(\log N)$ on average
  - for points randomly distributed inside a ***circle***, $M = \Theta(N^{1/3})$ on average

Package Wrap has worst-case time complexity $O(N^2)$

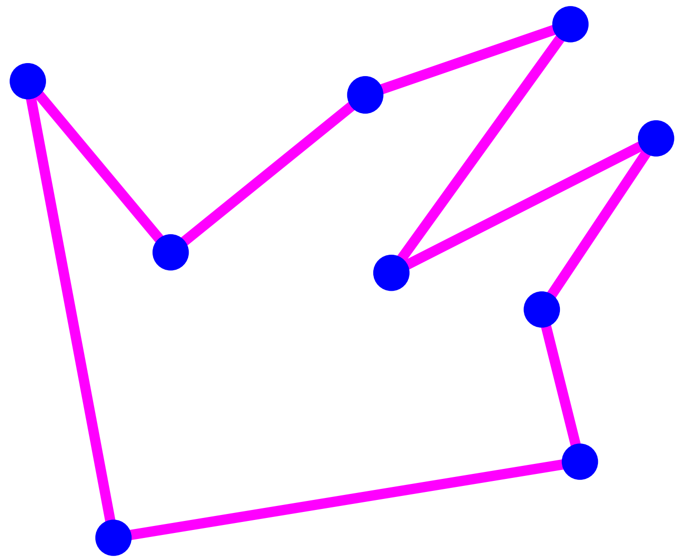
Which is bad...

Graham Scan

- Form a simple polygon (connect the dots as before)

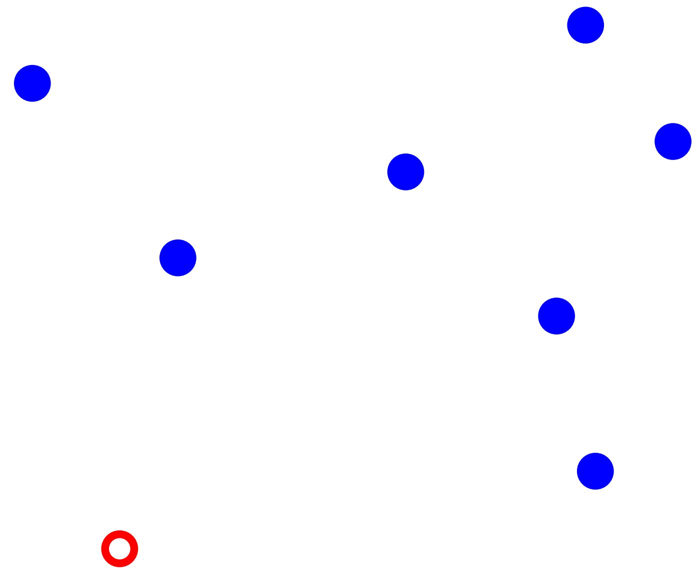


- Remove points at concave angles



Graham Scan

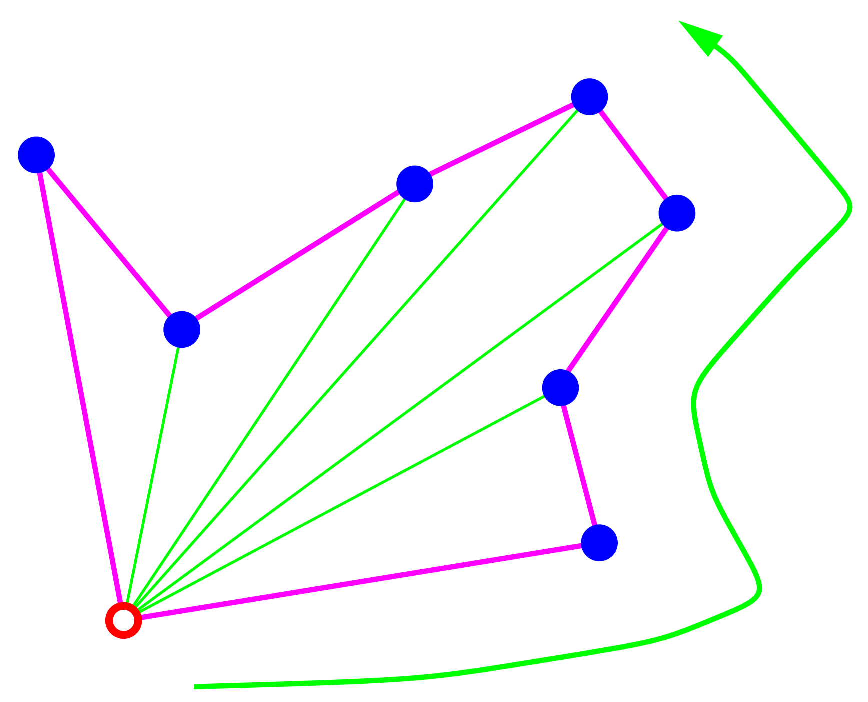
How Does it Work?



Start with the lowest point (anchor point)

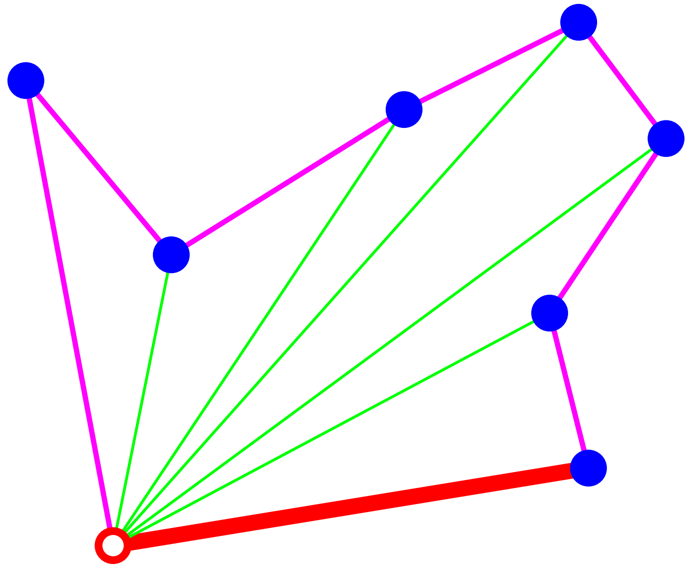
Graham Scan: Phase 1

Now, form a closed simple path traversing the points by increasing angle with respect to the anchor point



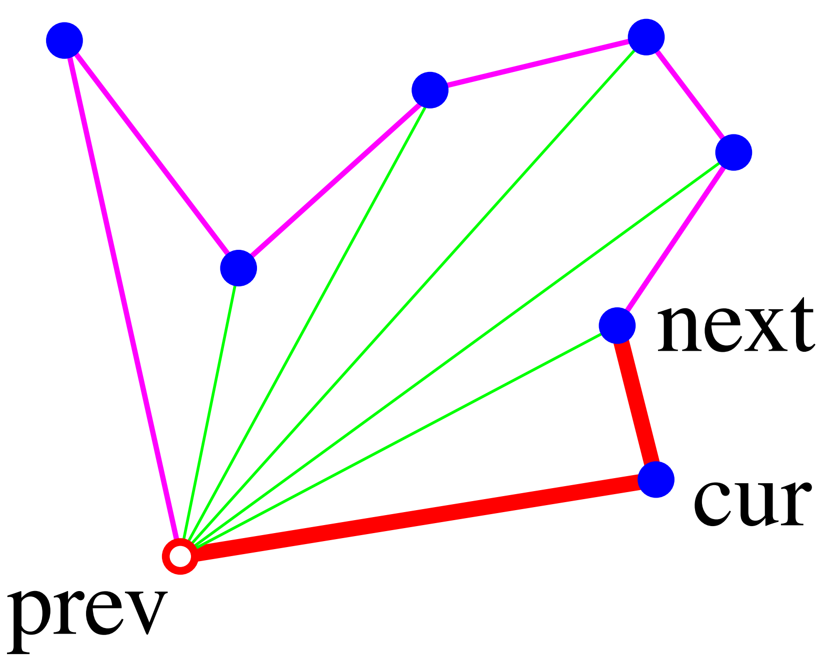
Graham Scan: Phase 2

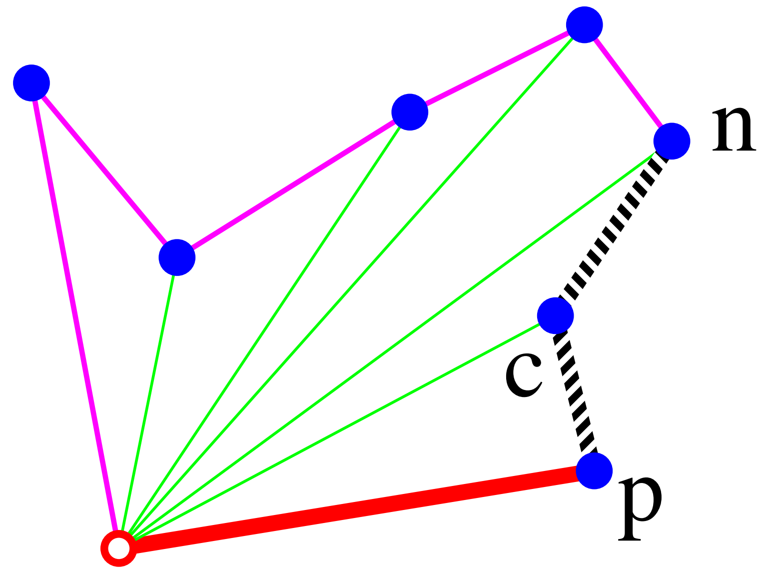
The anchor point and the next point on the path must be on the hull (why?)



Graham Scan: Phase 2

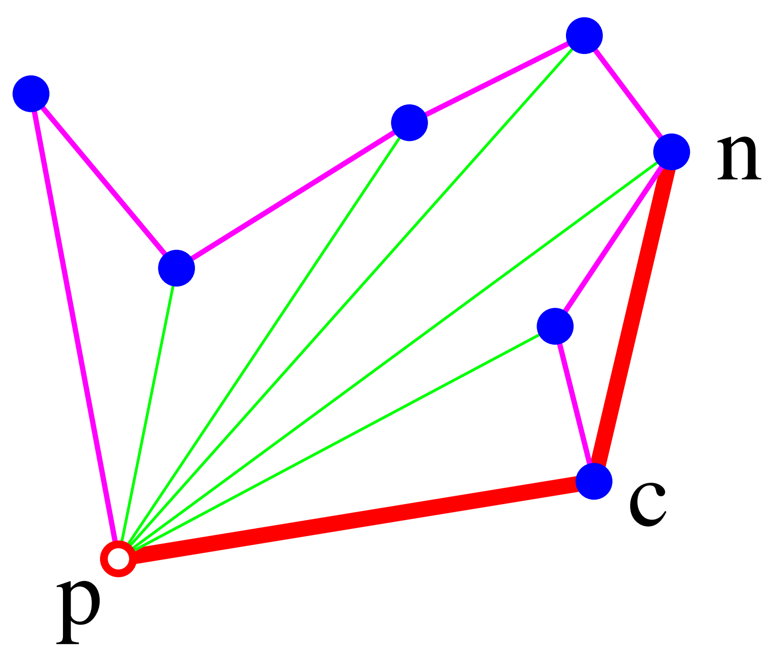
- keep the path and the hull points in two sequences
- elements are removed from the beginning of the path sequence and are inserted and deleted from the end of the hull sequence
- orientation is used to decide whether to accept or reject the next point

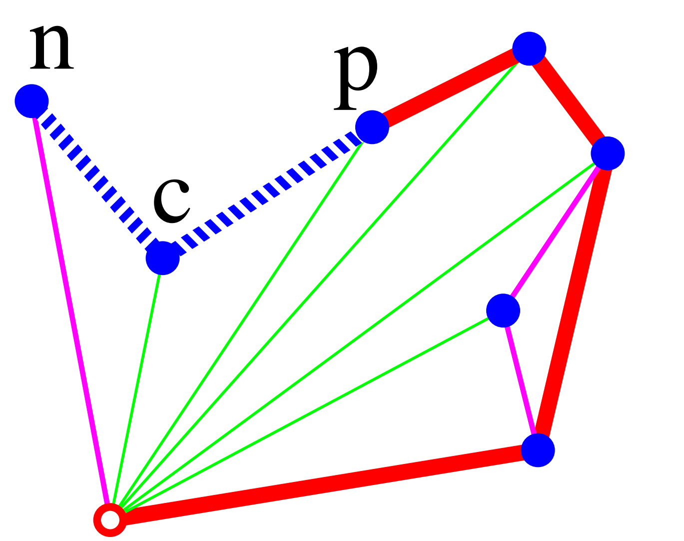




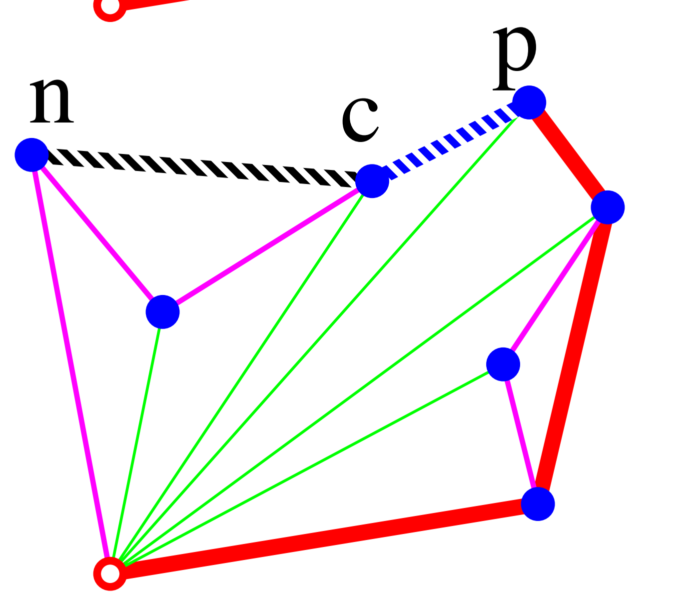
(p,c,n) is a right turn!

Discard c

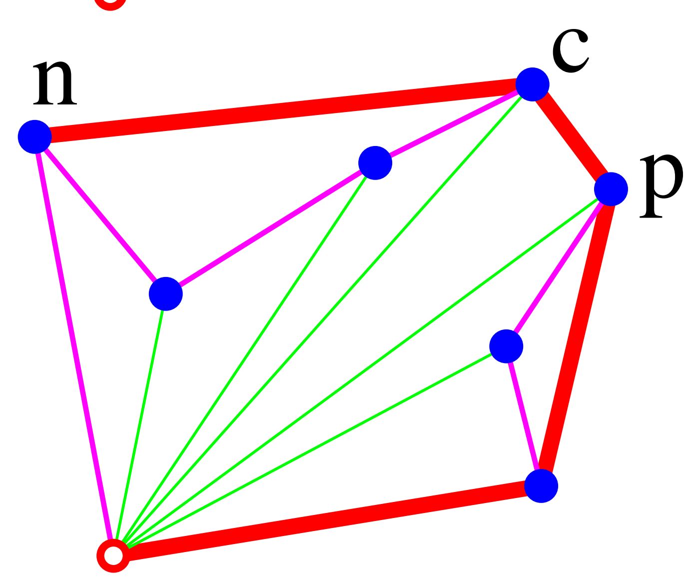




(p,c,n) is a right turn!



(p,c,n) is a right turn!



Time Complexity of Graham Scan

- Phase 1 takes time O(N logN)
  - points are sorted by angle around the anchor
- Phase 2 takes time O(N)
  - each point is inserted into the sequence exactly once, and
  - each point is removed from the sequence at most once
- Total time complexity O(N log N)

Winter 2016
COMP-250: Introduction to Computer Science

Lecture 15, March 8, 2016