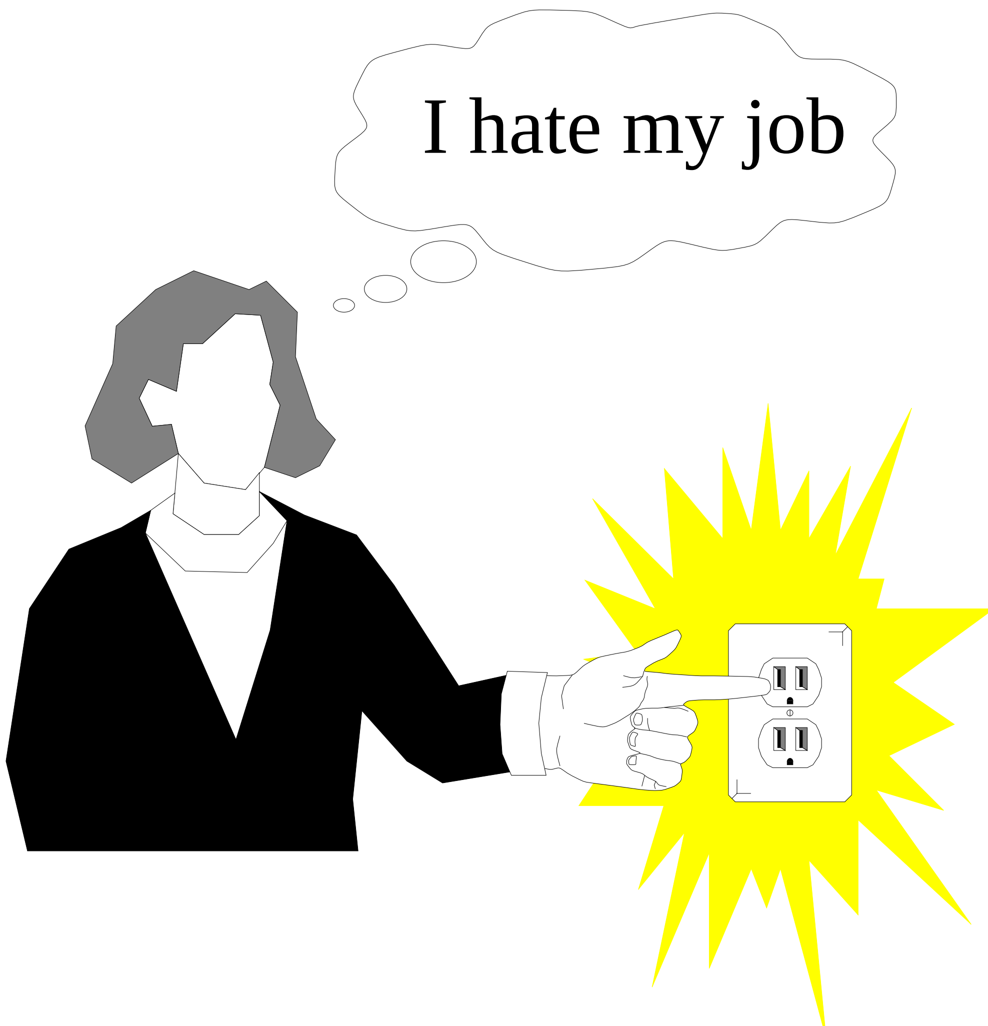
# GEOMETRIC ALGORITHMS

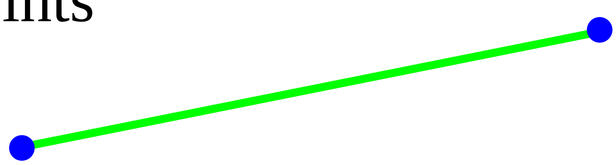
*Yowzer! They're electrifying!*



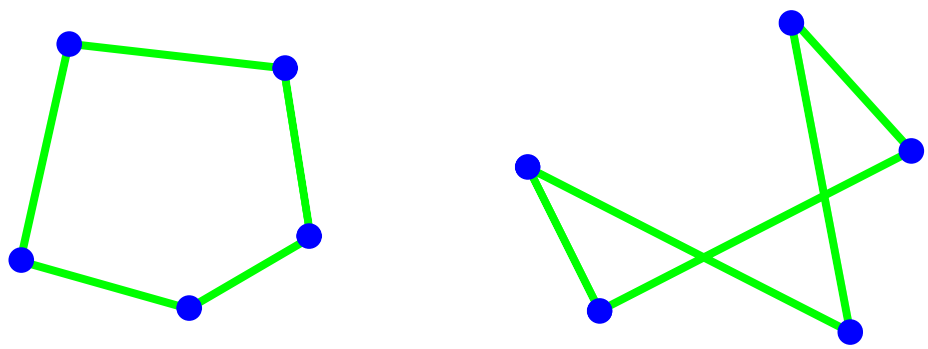
# Basic Geometric Objects in the Plane

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

*segment*: portion of a straight line between two points

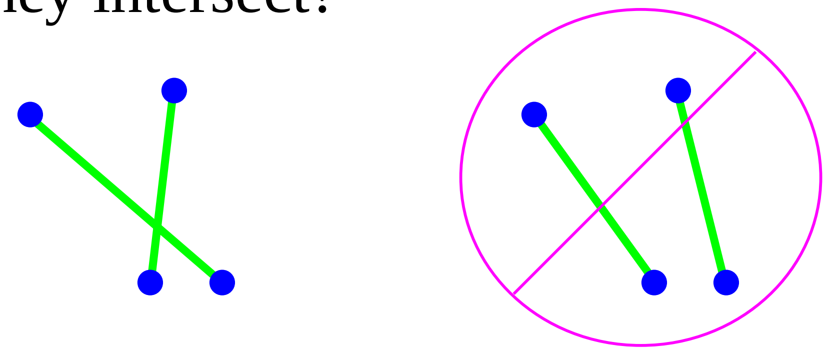


*polygon*: a circular sequence of points (vertices) and segments (edges) between them

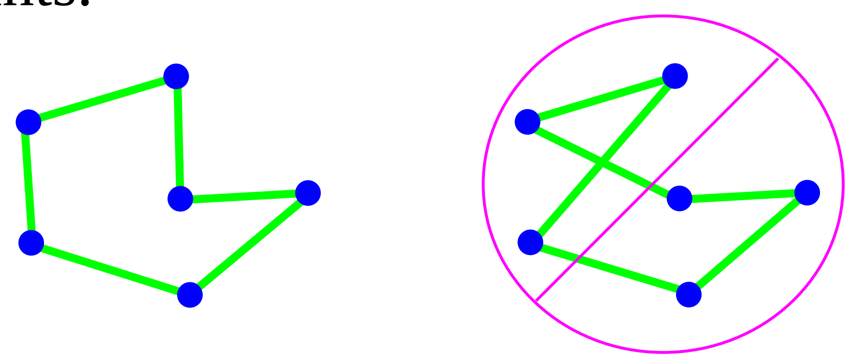


# Some Geometric Problems

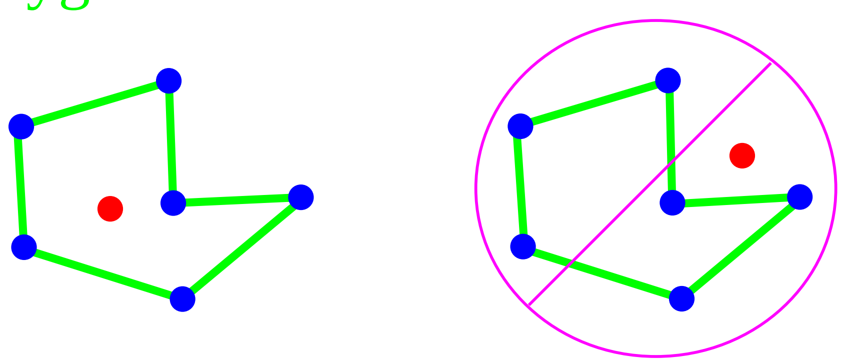
**Segment intersection**: Given two segments, do they intersect?



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

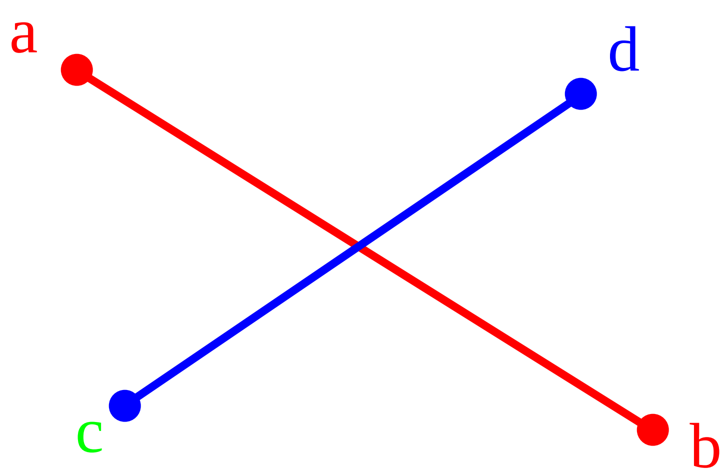


**Inclusion in polygon**: Is a point inside or outside a polygon?



# An Apparently Simple Problem: Segment Intersection

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

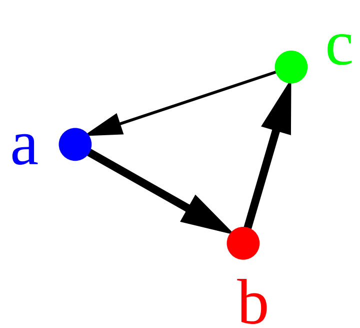



- We could start by writing down the equations of the lines through the segments, then test whether the lines intersect, then ...

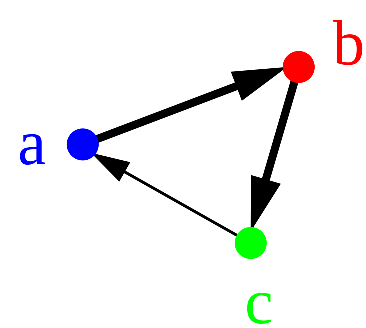
- An alternative (and simpler) approach is based in the notion of **orientation** of an ordered triplet of points in the plane

# Orientation in the Plane

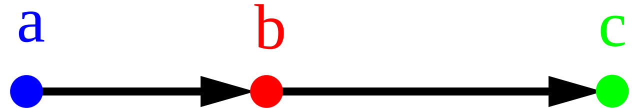
- The orientation of an ordered triplet of points in the plane can be
  - **counterclockwise** (**left turn**)
  - **clockwise** (**right turn**)
  - **collinear** (**no turn**)
- Examples:




**counterclockwise** (**left turn**)




**clockwise** (**right turn**)

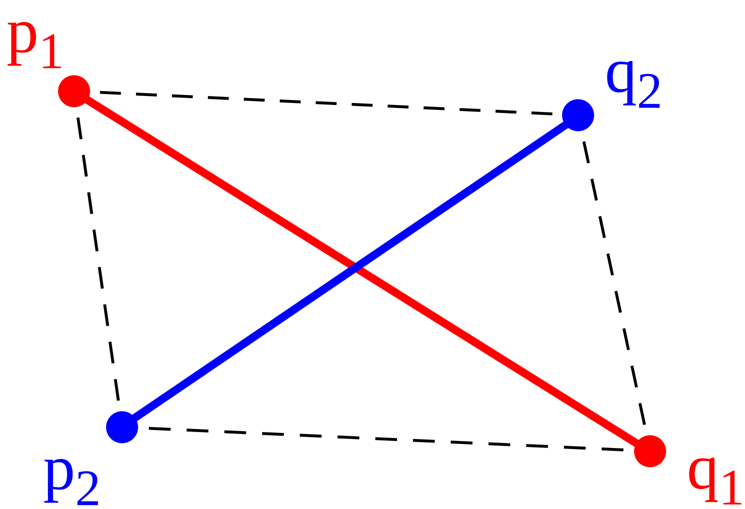



**collinear** (**no turn**)

# Intersection and Orientation

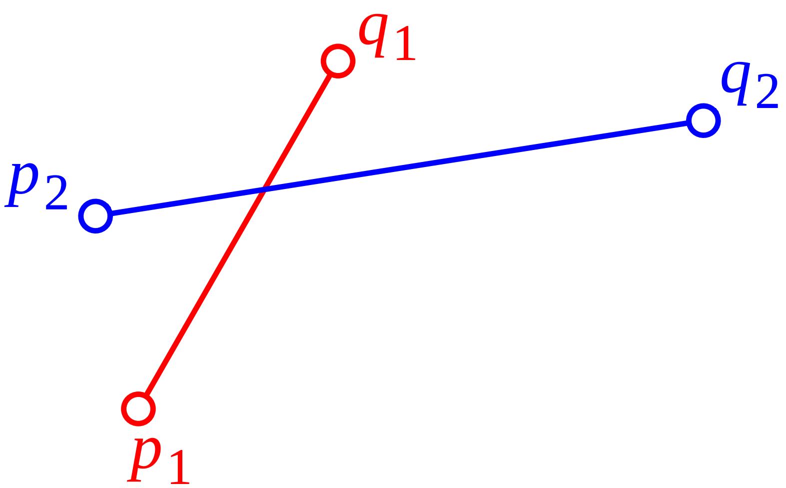
Two segments ($p_1$,$q_1$) and ($p_2$,$q_2$) intersect if and only if one of the following two conditions is verified

- 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
- 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



# 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

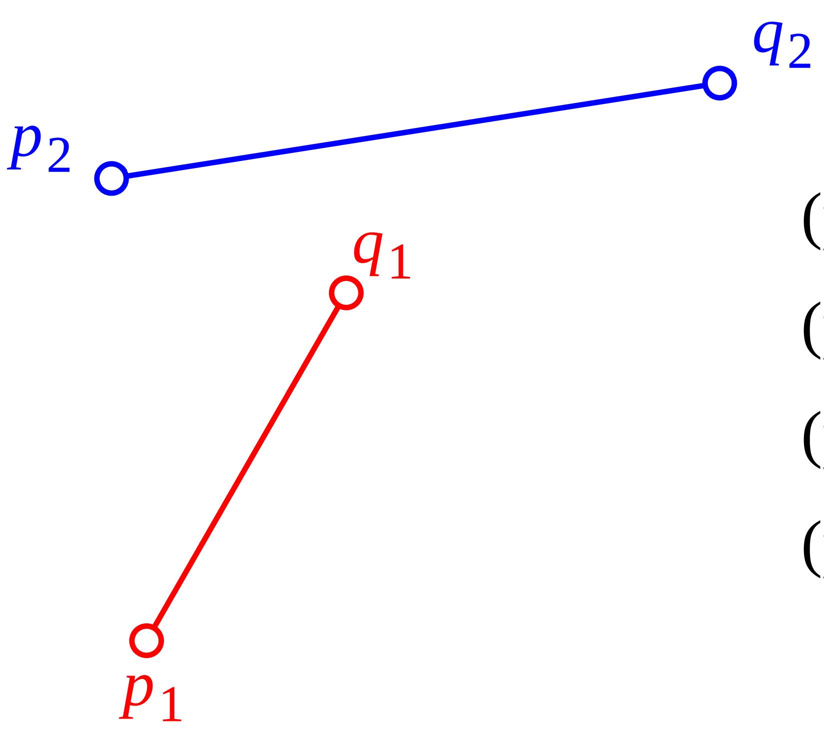



($p_1$,$q_1$,$p_2$)

($p_1$,$q_1$,$q_2$)

($p_2$,$q_2$,$p_1$)

($p_2$,$q_2$,$q_1$)




($p_1$,$q_1$,$p_2$)

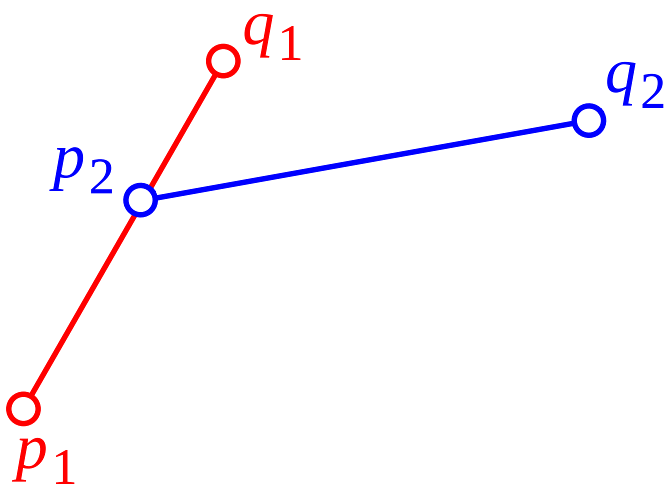
($p_1$,$q_1$,$q_2$)

($p_2$,$q_2$,$p_1$)

($p_2$,$q_2$,$q_1$)

# 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

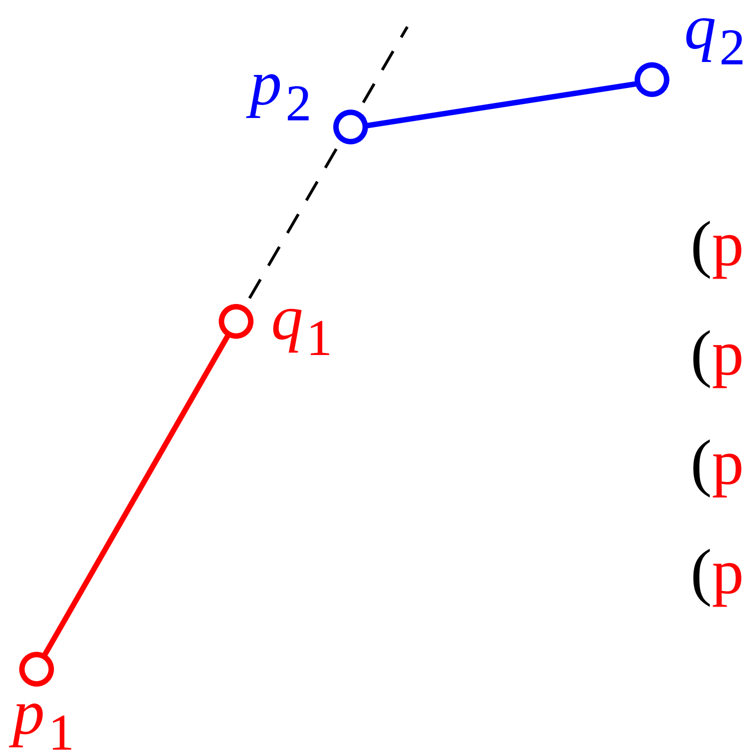



($p_1$,$q_1$,$p_2$)

($p_1$,$q_1$,$q_2$)

($p_2$,$q_2$,$p_1$)

($p_2$,$q_2$,$q_1$)




($p_1$,$q_1$,$p_2$)

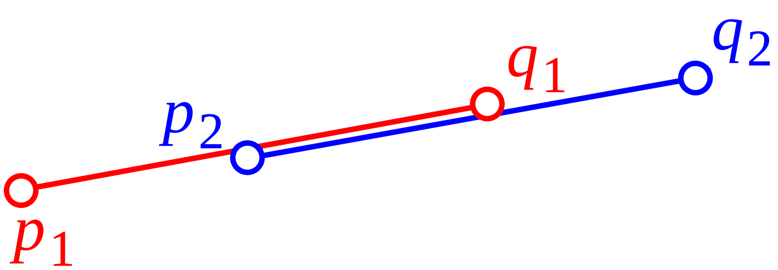
($p_1$,$q_1$,$q_2$)

($p_2$,$q_2$,$p_1$)

($p_2$,$q_2$,$q_1$)

# Examples (Special Case)

- 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

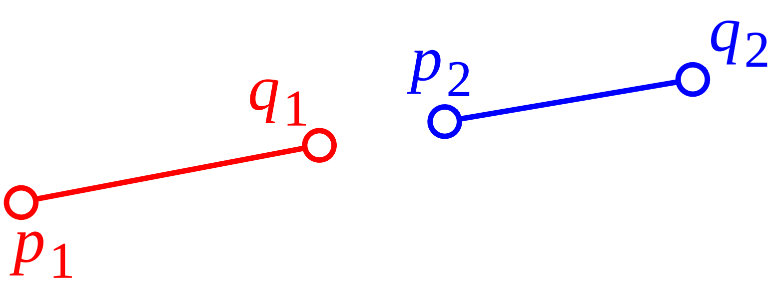



$(p_1,q_1,p_2)$

$(p_1,q_1,q_2)$

$(p_2,q_2,p_1)$

$(p_2,q_2,q_1)$




$(p_1,q_1,p_2)$

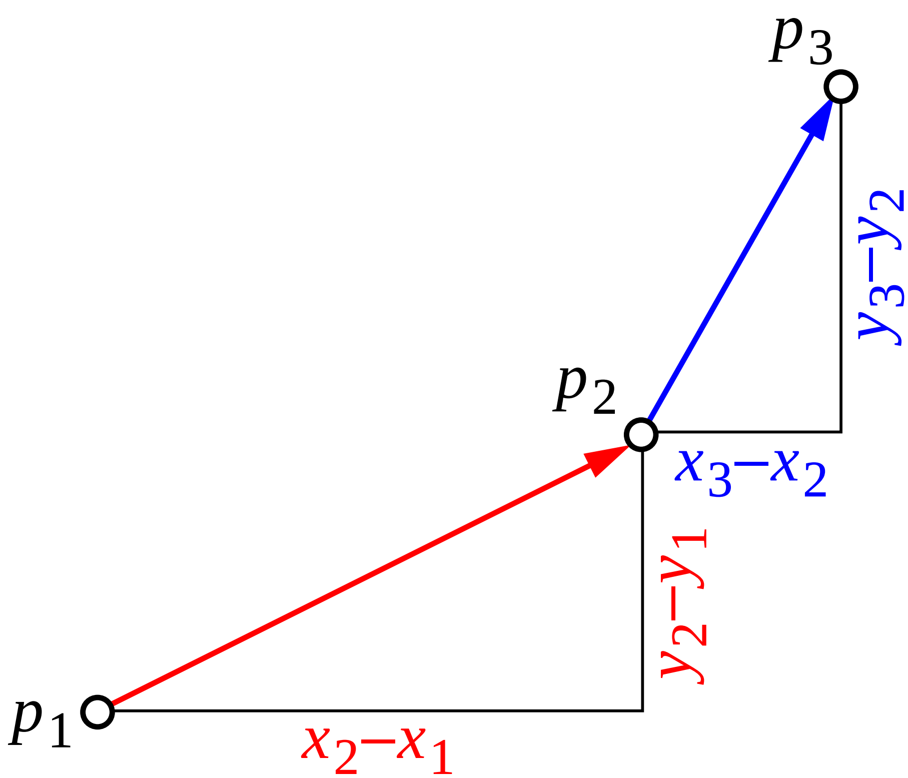
$(p_1,q_1,q_2)$

$(p_2,q_2,p_1)$

$(p_2,q_2,q_1)$

# How to Compute the Orientation

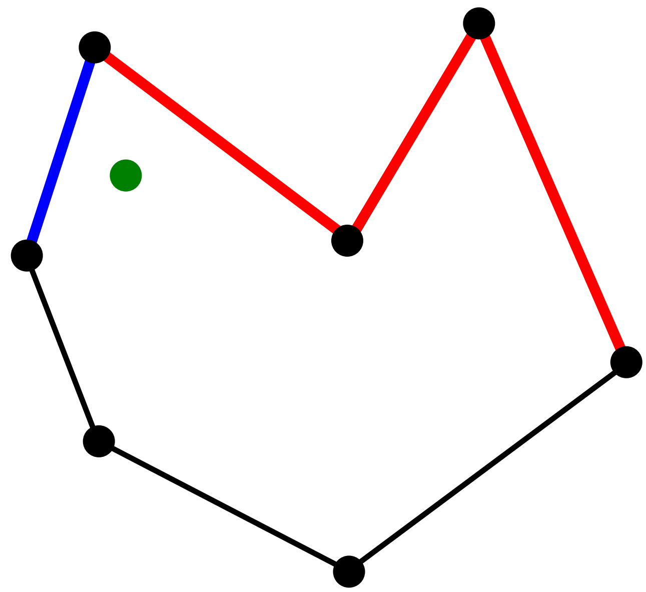
- slope of segment $(p_1,p_2)$: $\sigma = (y_2-y_1) / (x_2-x_1)$
- slope of segment $(p_2,p_3)$: $\tau = (y_3-y_2) / (x_3-x_2)$

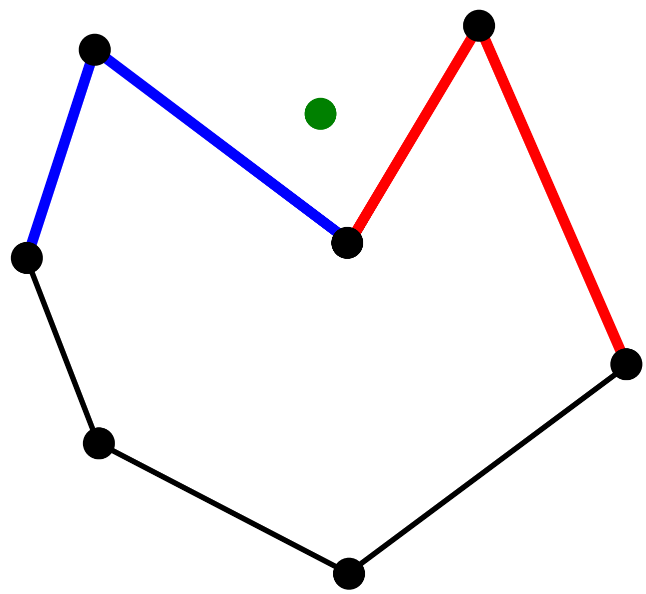



- Orientation test
  - counterclockwise (left turn): $\sigma < \tau$
  - clockwise (right turn): $\sigma > \tau$
  - collinear (left turn): $\sigma = \tau$
- The orientation depends on whether the expression $(y_2-y_1)(x_3-x_2) - (y_3-y_2)(x_2-x_1)$ is positive, negative, or null.

# Point Inclusion

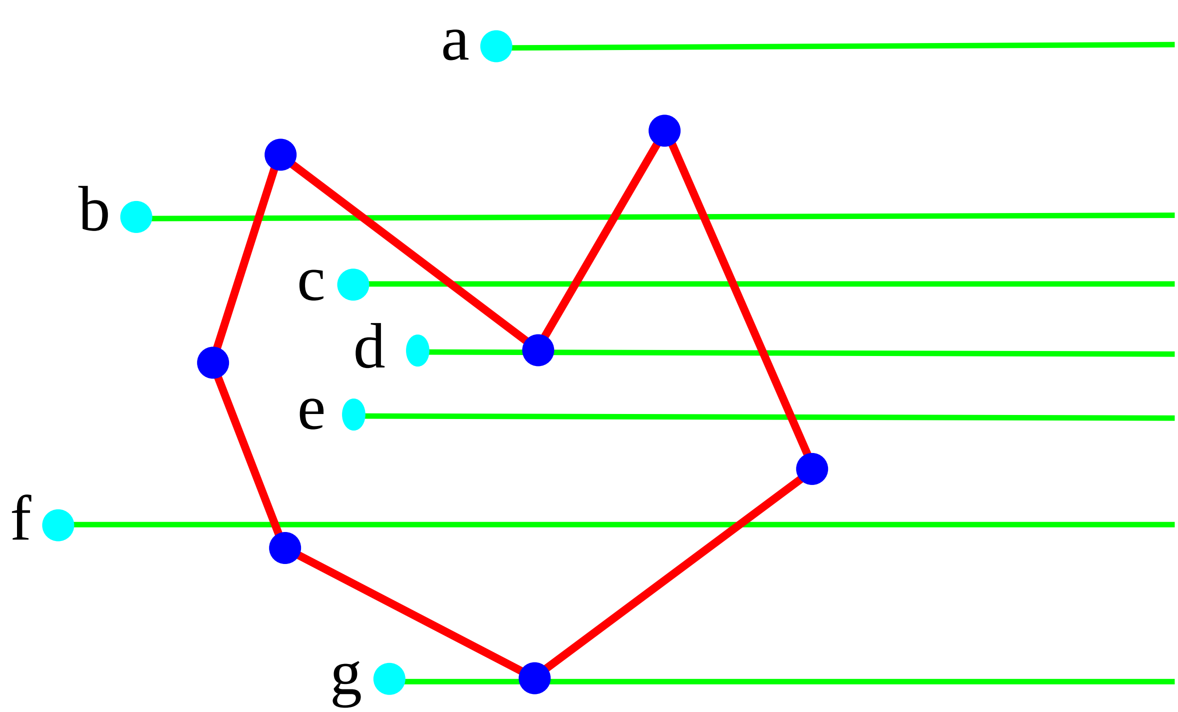
- given a polygon and a point, is the point inside or outside the polygon?
- orientation helps solving this problem in linear time





# Point Inclusion — Part II

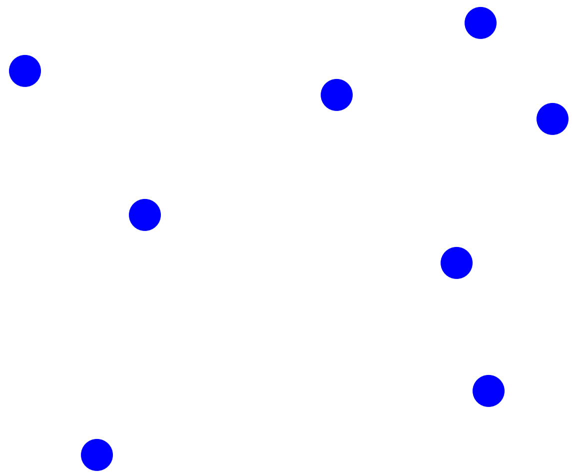
- Draw a horizontal line to the right of each point and extend it to infinity
- Count the number of times a line intersects the polygon. We have:
  - even number $\Rightarrow$ point is outside
  - odd number $\Rightarrow$ point is inside
- Why?



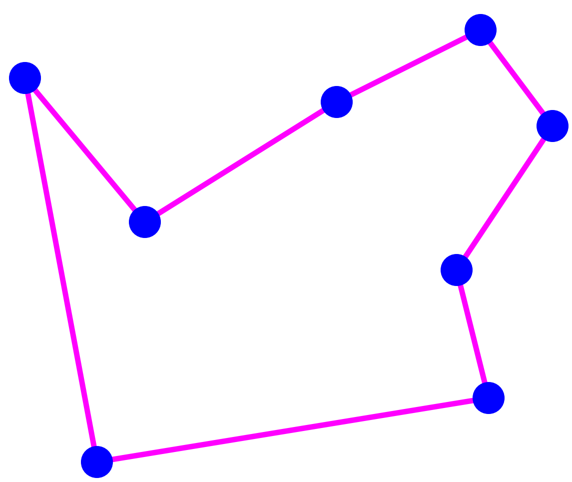

- What about points d and g ?? Degeneracy!

# Simple Closed Path — Part I

- Problem:  Given a set of points ...

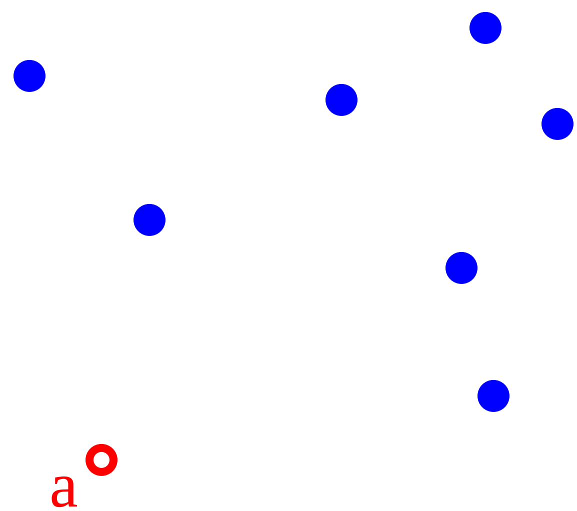


- "Connect the dots" without crossings

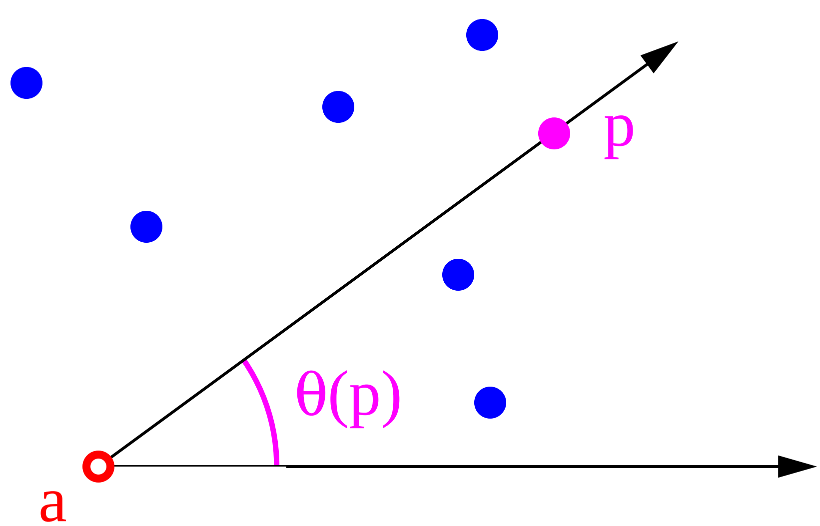


# Simple Closed Path — Part II

- Pick the bottommost point a as the anchor point

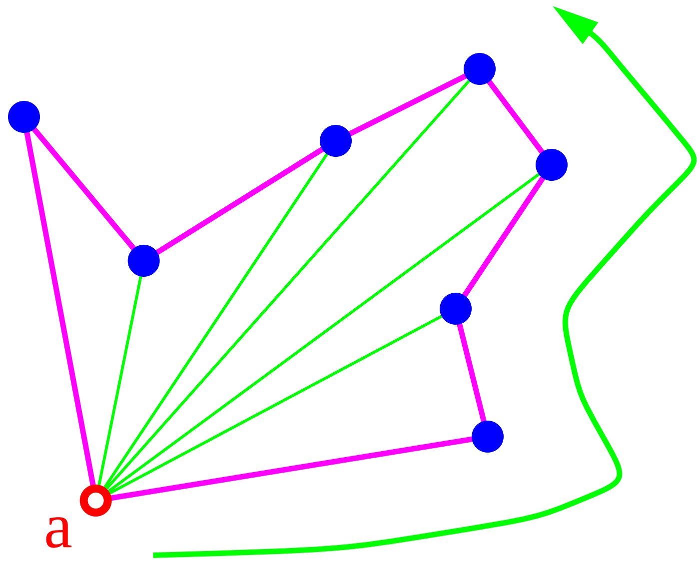



- For each point p, compute the angle q(p) of the segment (a,p) with respect to the x-axis:



# Simple Closed Path — Part III

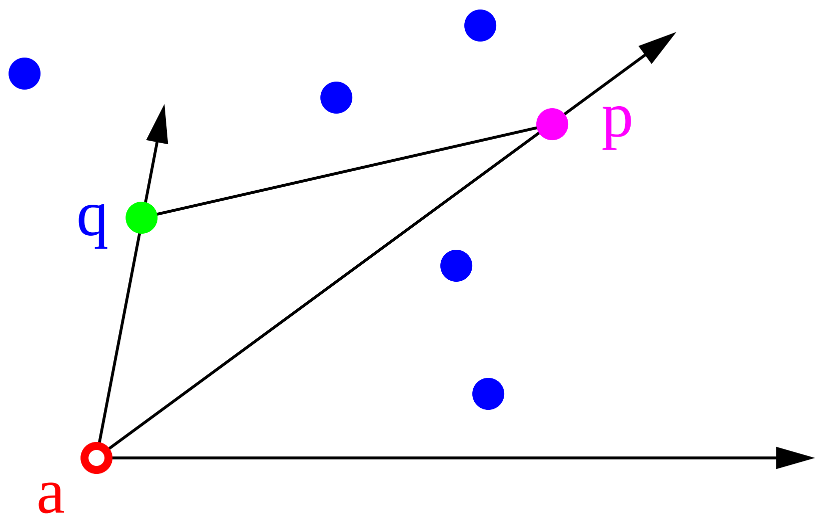
- Traversing the points by increasing angle yields a simple closed path:




- 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 the orientation to compare angles without actually computing them!!

# Simple Closed Path — Part IV

- the orientationcan be used to compare angles without actually computing them ... Cool!




$\theta(p) < \theta(q) \iff \text{orientation}(a,p,q) = \text{CCW}$

- 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

# Graham Scan Algorithm

**Algorithm** Scan($S$, $a$):

**Input:** A sequence $S$ of points in the plane beginning with point $a$ such that:
1) $a$ is a vertex of the convex hull of the points of $S$
2) the remaining points of $S$ are counterclockwise around $a$.

**Output:** Sequence $S$ from which the points that are not vertices of the convex hull have been removed.

```
S.insertLast(a)              {add a copy of a at the end of S}
prev ← S.first()             {so that prev = a initially}
curr ← S.after(prev)         {the next point is on the}
                             {current convex chain}
repeat
   next ← S.after(curr)      {advance}
   if points (point(prev), point(curr), point(next))
make a left turn then
      prev ← curr
   else
      S.remove(curr)         {point curr is on the convex hull}
      prev ← S.before(prev)
   curr ← S.after(prev)
until curr = S.last()
S.remove(S.last())           {remove the copy of a}
```