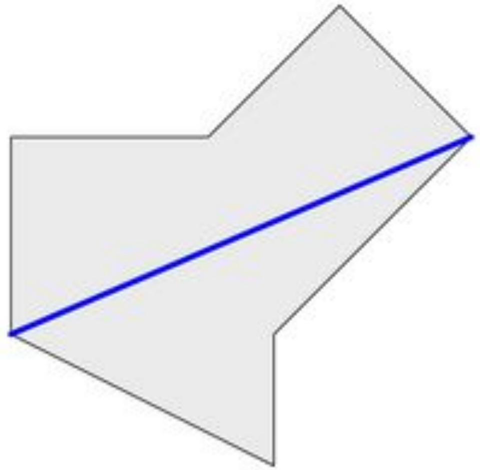
Geometry for Competitive Programming

Airport Construction: WF ‘17



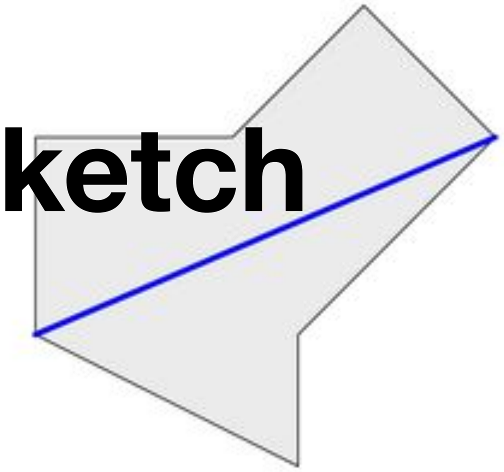
Input

The input starts with a line containing an integer n ($3 \le n \le 200$) specifying the number of vertices of the polygon. This is followed by n lines, each containing two integers x and y ($|x|, |y| \le 10^6$) that give the coordinates (x, y) of the vertices of the polygon in counter-clockwise order. The polygon is simple, i.e., its vertices are distinct and no two edges of the polygon intersect or touch, except that consecutive edges touch at their common vertex. In addition, no two consecutive edges are collinear.

Output

Display the length of the longest straight line segment that fits inside the polygon, with an absolute or relative error of at most 10^{-6}.

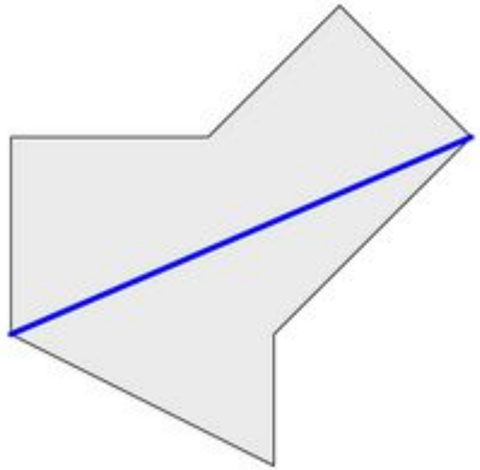
Airport Construction: Solution Sketch



Some initial observations:

- $n \leq 200$: $O(n^3)$ should be sufficient

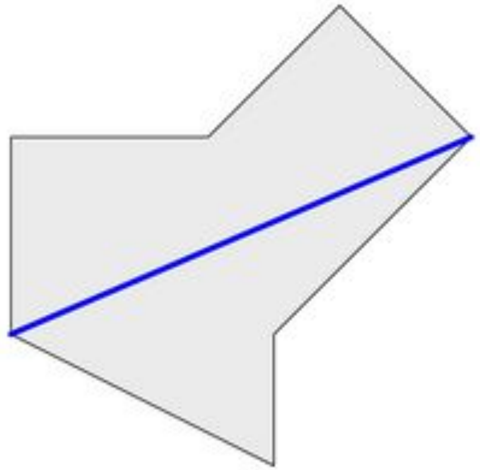
Airport Construction: Solution S



Some initial observations:

- $n \leq 200$: $O(n^3)$ should be sufficient
- coordinates are max 10^6 : quadratic formulas are OK

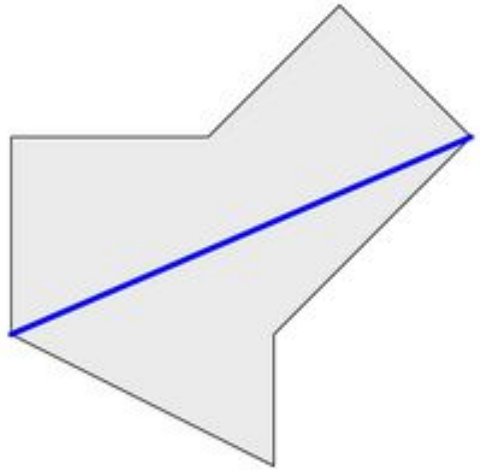
Airport Construction: Solution S



Some initial observations:

- $n \leq 200$: $O(n^3)$ should be sufficient
- coordinates are max 10^6 : quadratic formulas are OK
- the longest line segment must have its endpoints at polygon vertices
    - (why?)

Airport Construction: Solution S



```
maxlen = 0
for each pair of vertices p, q:
  extend pq until it intersects the polygon
    if the segment is inside the polygon
      maxlen = max( maxlen, len(p,q) )
```

This is an “easy” geometry problem for the WF level!

Airport Construction: Some Stats

Here are the scoreboard stats from World Finals 2017:

	A	B	C	D	E	F	G	H	I	J	K	L
Solved / Tries	35/710 (5%)	8/37 22%)	105/255 (41%)	31/311 (10%)	127/191 (66%)	123/174 (71%)	18/33 (55%)	0/18 (0%)	127/137 (93%)	1/14 (7%)	15/99 (15%)	27/150 (18%)
Average tries	7.32	2.64	2.11	4.78	1.50	1.38	1.65	2.00	1.07	3.50	2.83	3.49
Averages tries to solve	5.43	2.62	1.92	2.84	1.50	1.33	1.44	--	1.06	11.00	2.20	2.59

What happened?

Airport Construction: Some Stats

Here are the scoreboard stats from World Finals 2017:

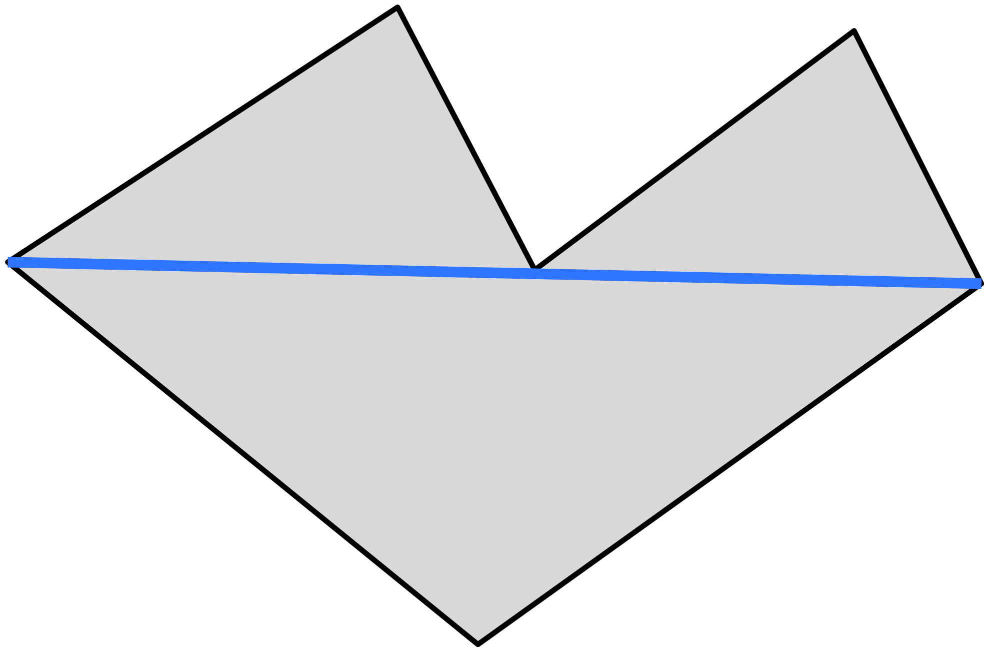
	A	B	C	D	E	F	G	H	I	J	K	L
Solved / Tries	35/710 (5%)	8/37 (22%)	105/255 (41%)	31/311 (10%)	127/191 (66%)	123/174 (71%)	18/33 (55%)	0/18 (0%)	127/137 (93%)	1/14 (7%)	15/99 (15%)	27/150 (18%)
Average tries	7.32	2.64	2.11	4.78	1.50	1.38	1.65	2.00	1.07	3.50	2.83	3.49
Averages tries to solve	5.43	2.62	1.92	2.84	1.50	1.33	1.44	--	1.06	11.00	2.20	2.59

What happened?

- it was Problem A so it was attempted by many teams
- it has some legitimate subtleties

The Subtleties

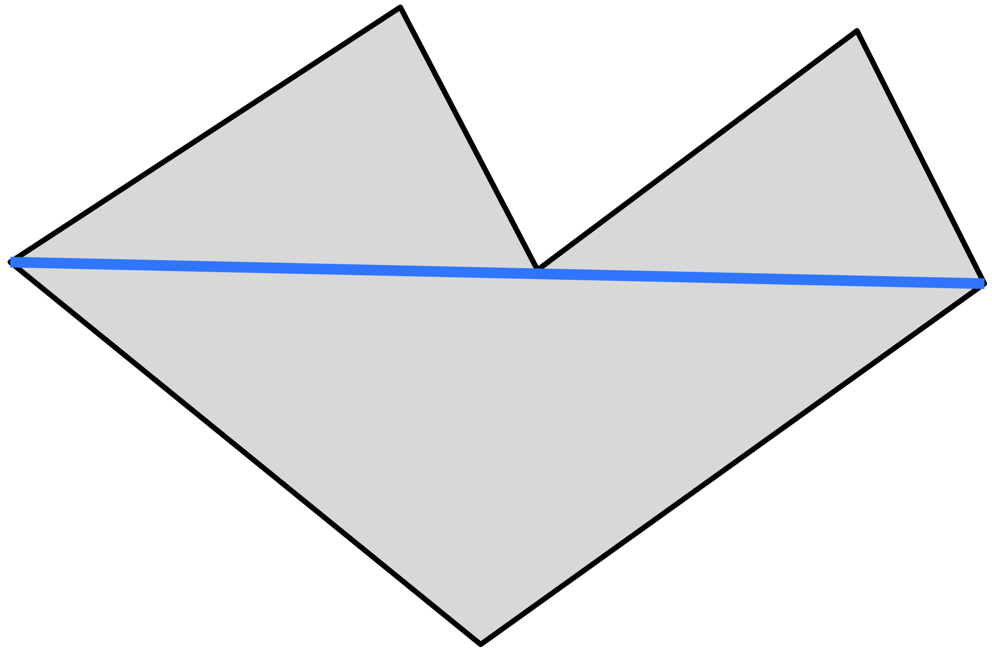
The solution could intersect other polygon vertices

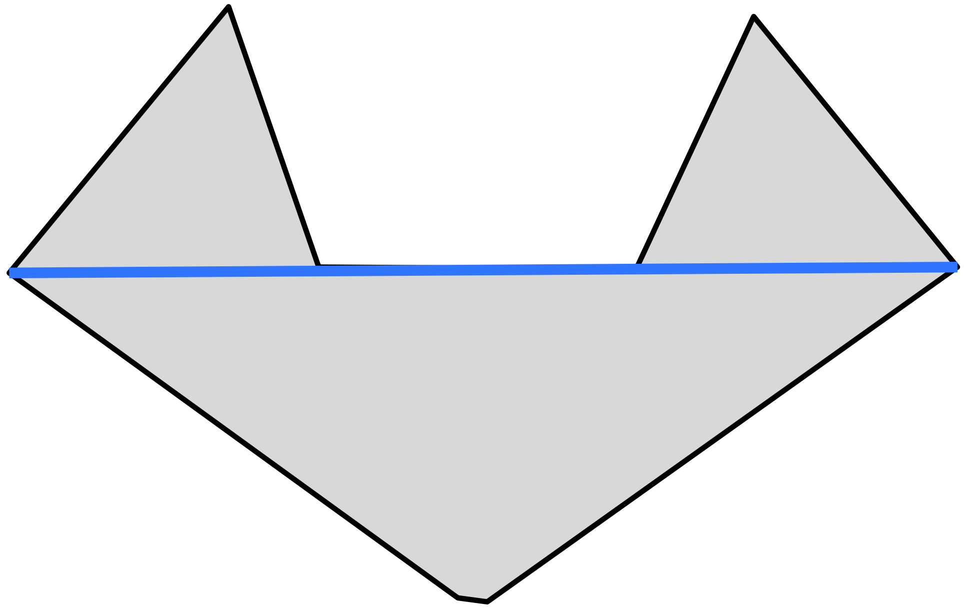


The Subtleties

The solution could intersect other polygon vertices

The solution could coincide with polygon edges



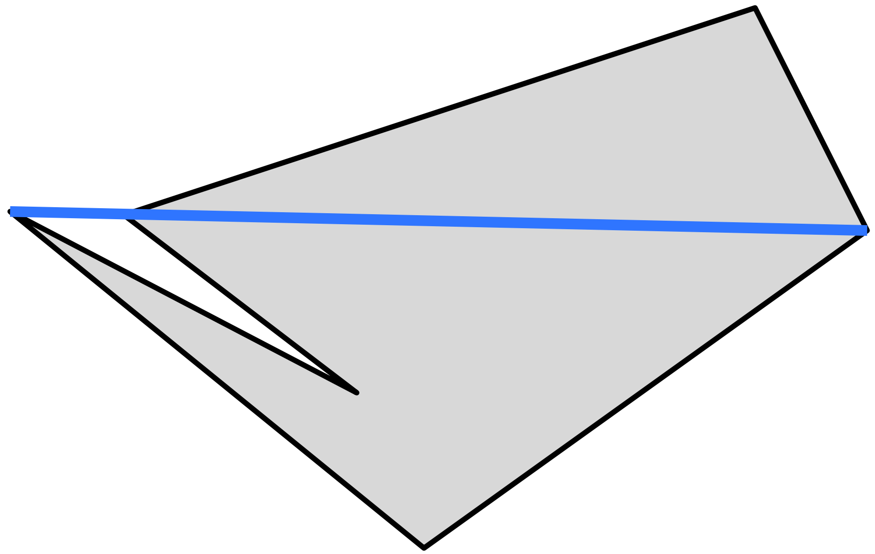


The Subtleties

The solution could intersect other polygon vertices

The solution could coincide with polygon edges

Illegal line segments might intersect only at vertices



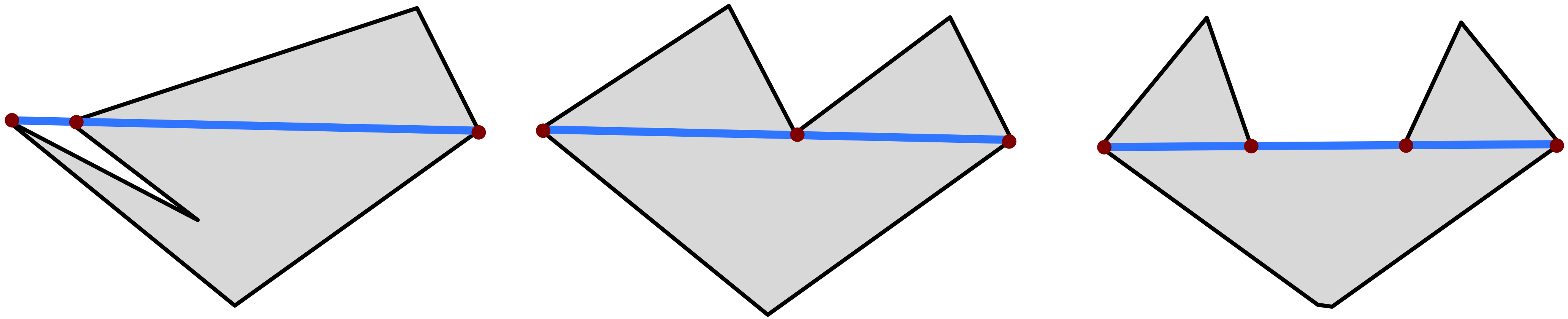
Uniform Solution to All Cases

Find where the line segment intersects polygon edges

- ignoring coincident edges

Break the segment into sub-segments and if check each is inside the polygon (by e.g. checking midpoint)

Compute longest consecutive run of segments inside



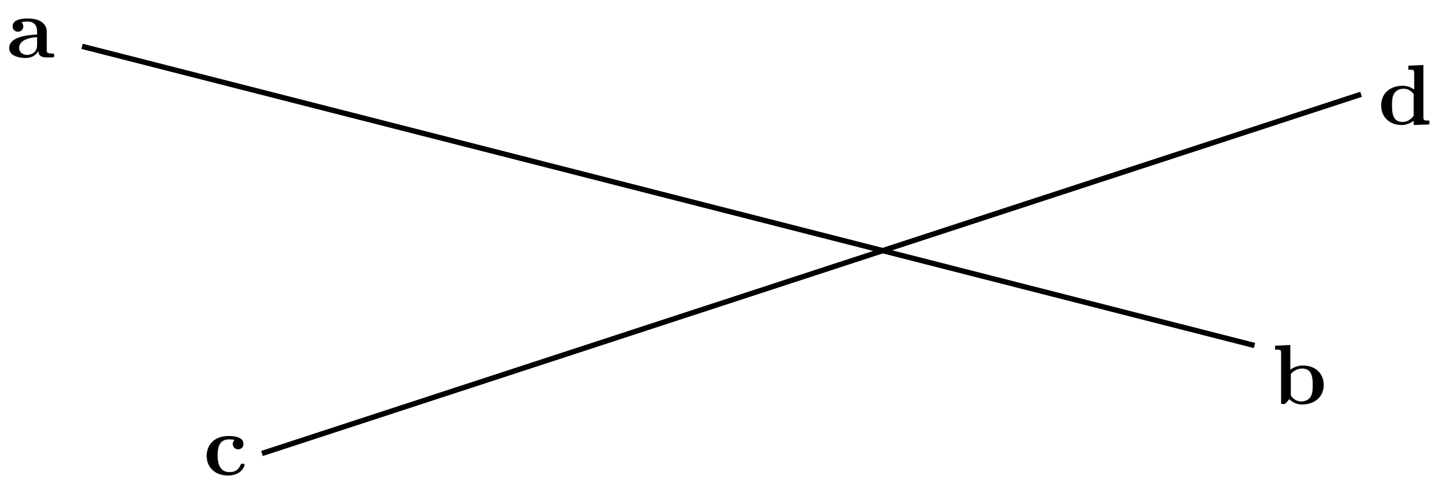
Segment-Segment Intersection Predicates

You should have these in your code book!

Segment-Segment Intersection Predicates

You should have these in your code book!

…but they are not hard to derive.



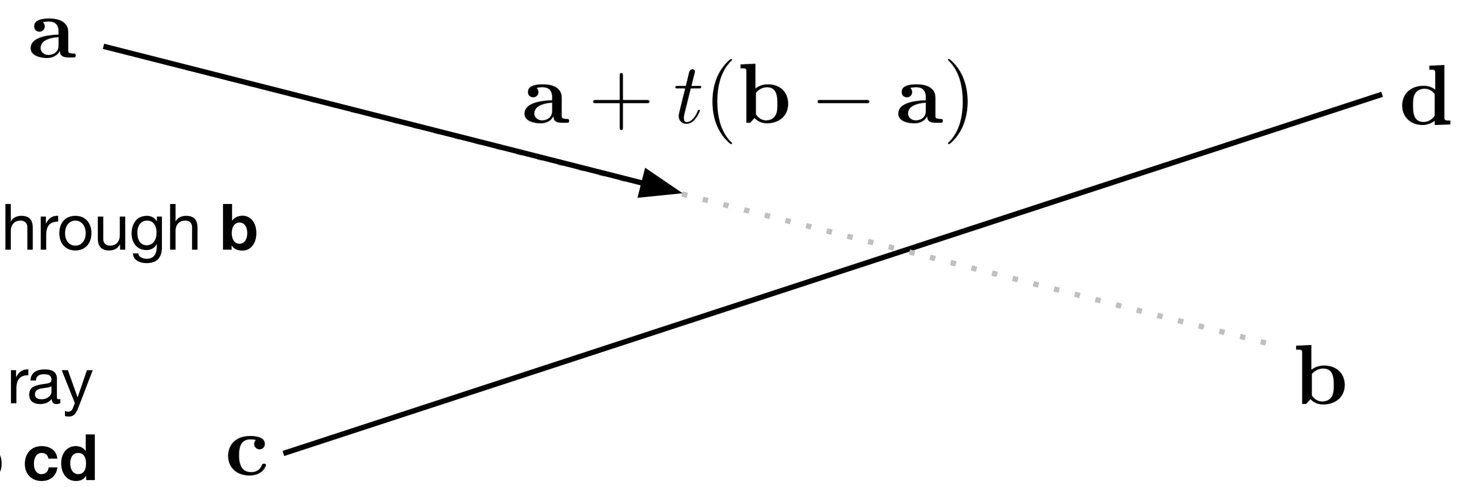
Segment-Segment Intersection Predicates

You should have these in your code book!

…but they are not hard to derive.

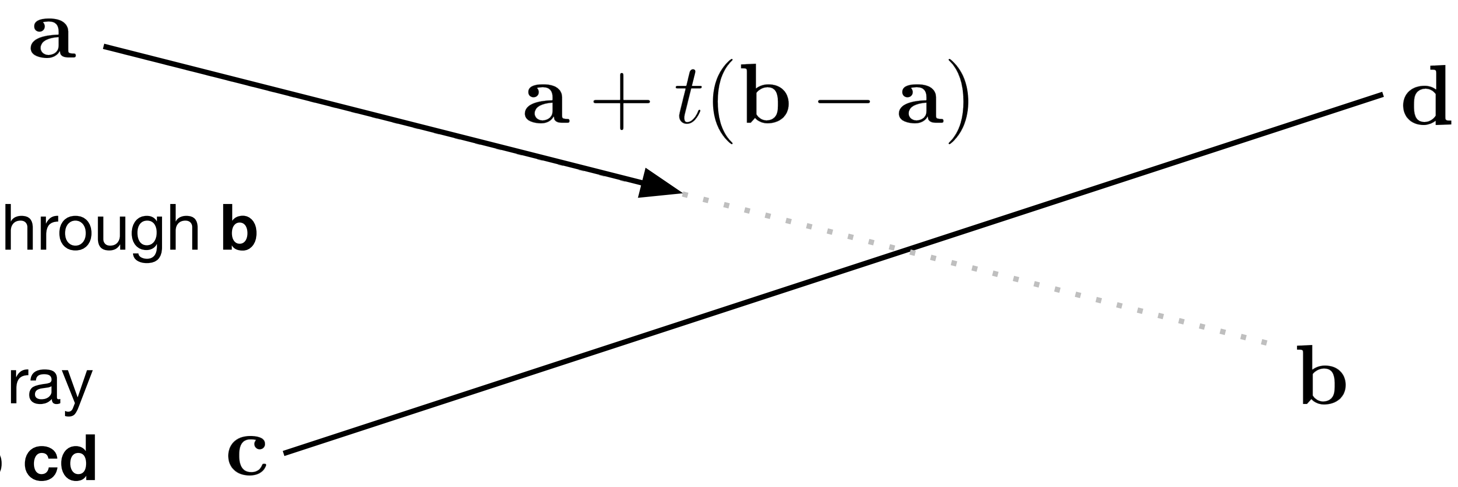
Consider ray traveling from **a** through **b**

Intersection is value of *t* when ray endpoint becomes parallel to **cd**



Segment-Segment Intersection Predicates

$$[\mathbf{a} + t(\mathbf{b} - \mathbf{a}) - \mathbf{c}] \times (\mathbf{d} - \mathbf{c}) = 0$$



Consider ray traveling from **a** through **b**

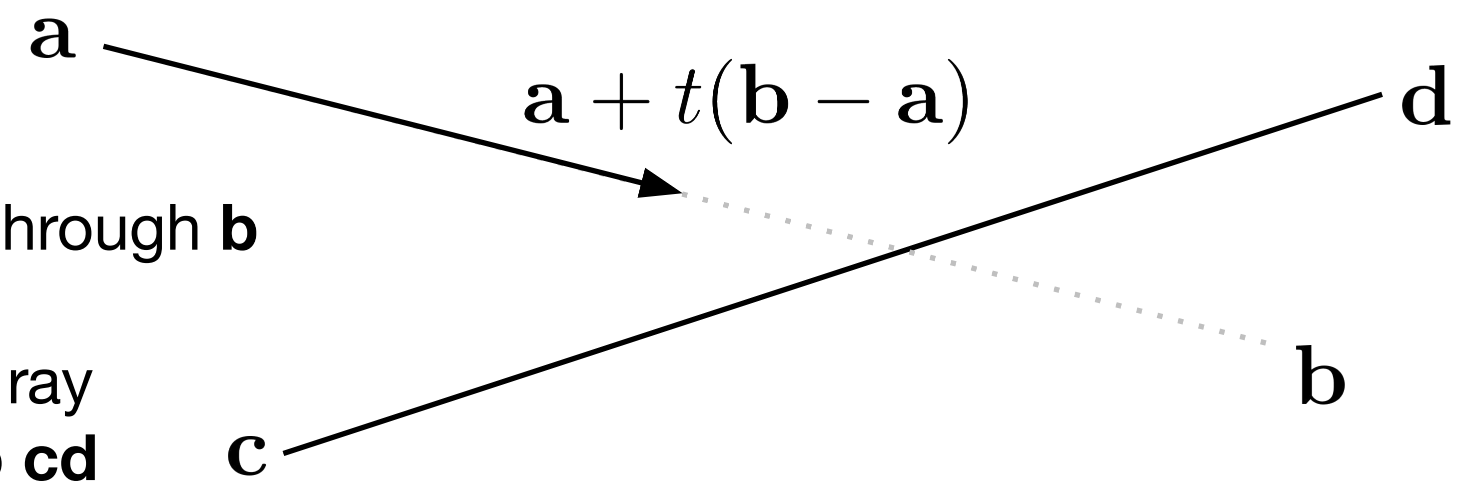
Intersection is value of *t* when ray endpoint becomes parallel to **cd**

Segment-Segment Intersection Predicates

$$[\mathbf{a} + t(\mathbf{b} - \mathbf{a}) - \mathbf{c}] \times (\mathbf{d} - \mathbf{c}) = 0$$

2D cross product:

$$\mathbf{v} \times \mathbf{w} = v_x w_y - v_y w_x$$



Consider ray traveling from **a** through **b**

Intersection is value of *t* when ray endpoint becomes parallel to **cd**

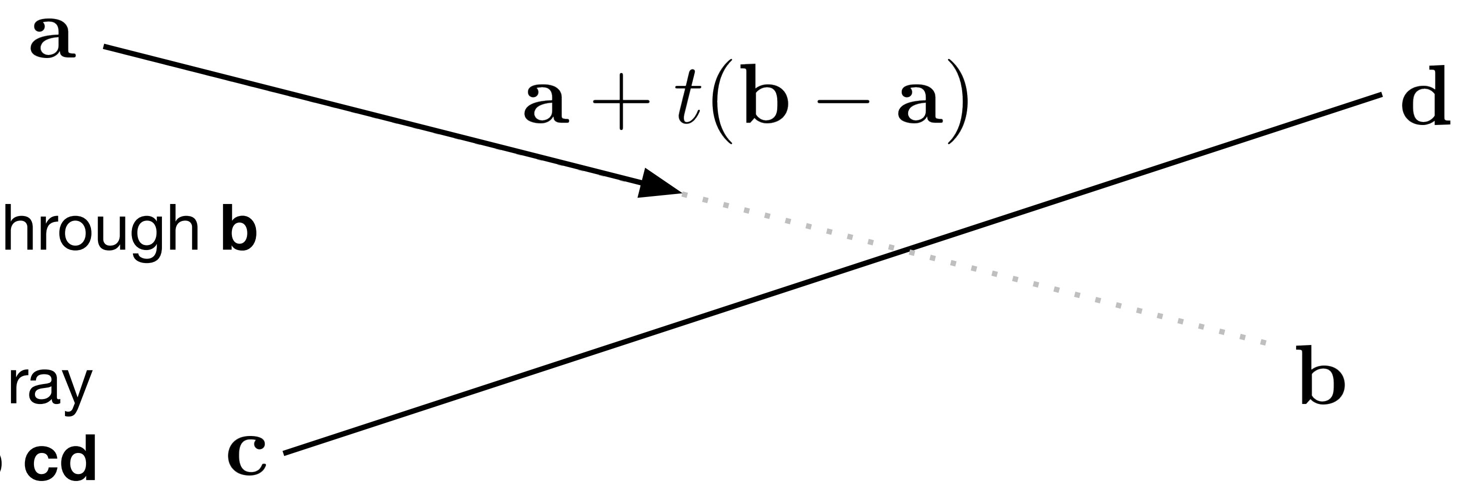
Segment-Segment Intersection Predicates

$$[\mathbf{a} + t(\mathbf{b} - \mathbf{a}) - \mathbf{c}] \times (\mathbf{d} - \mathbf{c}) = 0$$

$$t = \frac{(\mathbf{c} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})}{(\mathbf{b} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})}$$

2D cross product:

$$\mathbf{v} \times \mathbf{w} = v_x w_y - v_y w_x$$



Consider ray traveling from **a** through **b**

Intersection is value of *t* when ray endpoint becomes parallel to **cd**

Segment-Segment Intersection Predicates

$$[\mathbf{a} + t(\mathbf{b} - \mathbf{a}) - \mathbf{c}] \times (\mathbf{d} - \mathbf{c}) = 0$$

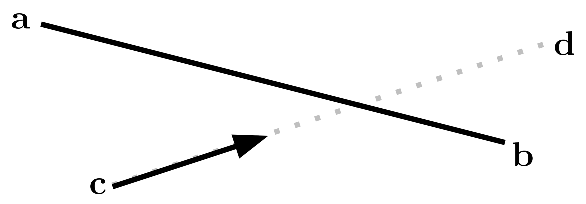
$$t = \frac{(\mathbf{c} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})}{(\mathbf{b} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})}$$

2D cross product:

$$\mathbf{v} \times \mathbf{w} = v_x w_y - v_y w_x$$

- $t < 0$ or $t > 1$: definitely no intersection
- $0 < t < 1$: maybe intersection on interior
- $t = 0$ or $t = 1$: maybe intersection at endpoint

repeat check for other segment



Segment-Segment Intersection Predicates

$$[\mathbf{a} + t(\mathbf{b} - \mathbf{a}) - \mathbf{c}] \times (\mathbf{d} - \mathbf{c}) = 0$$

$$t = \frac{(\mathbf{c} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})}{(\mathbf{b} - \mathbf{a}) \times (\mathbf{d} - \mathbf{c})}$$

2D cross product:

$$\mathbf{v} \times \mathbf{w} = v_x w_y - v_y w_x$$

- $t < 0$ or $t > 1$: definitely no intersection
- $0 < t < 1$: maybe intersection on interior
- $t = 0$ or $t = 1$: maybe intersection at endpoint

repeat check for other segment

Note: these checks **do not** require division

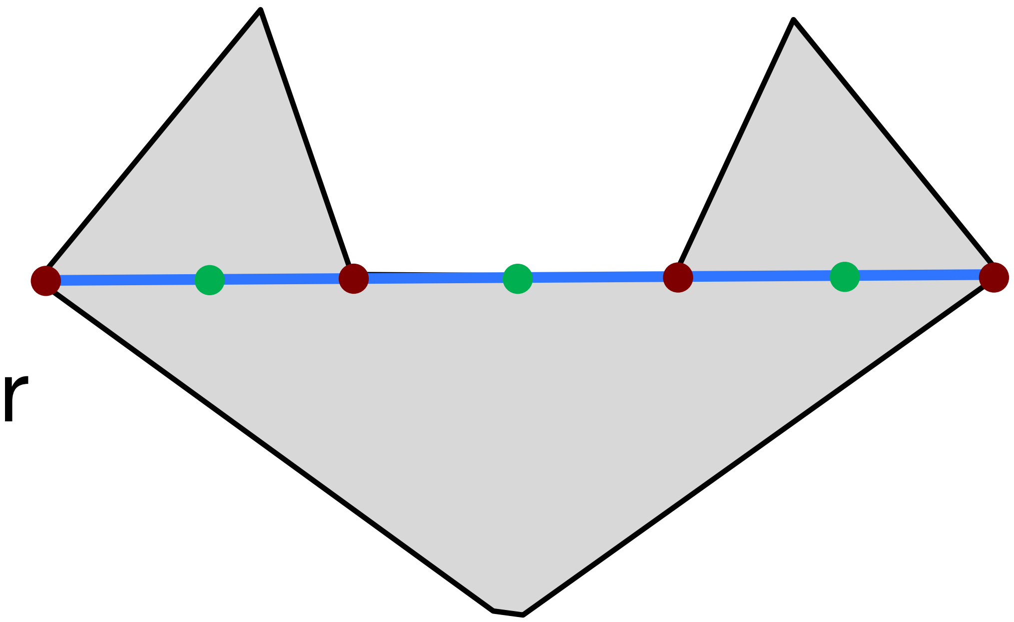
- they **do** require magnitudes quadratic in the coords

Point in Polygon Predicate

Again, must be robust to testing points exactly on the polygon boundary

Have book code for this too!

We will discuss a winding number solution later



Solving Airport Construction

You need:

- to realize that there are nontrivial (literal) edge cases
- robust segment-segment intersection predicates
- robust point-in-polygon predicate

Solving Airport Construction

You need:

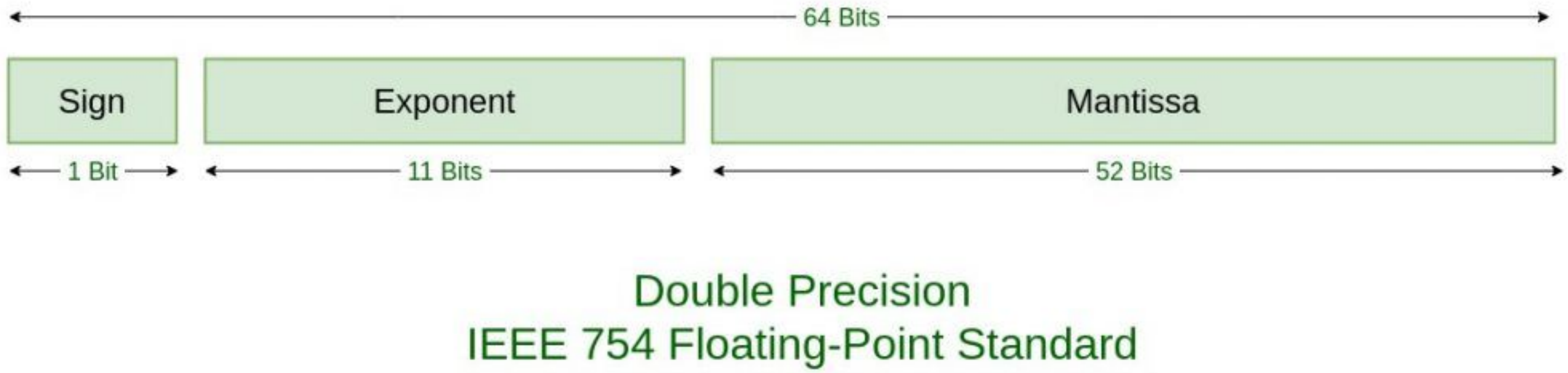
- to realize that there are nontrivial (literal) edge cases
- robust segment-segment intersection predicates
- robust point-in-polygon predicate

Are there any potential overflow or precision issues?

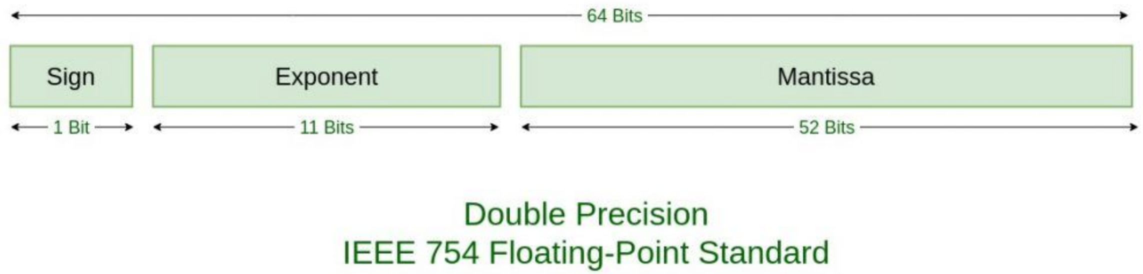
- can you use `doubles`?

The Minimum Every Competitive Programmer Needs To Know About Floating Point

Structure of a **`double`**: $\pm a \cdot 2^b$

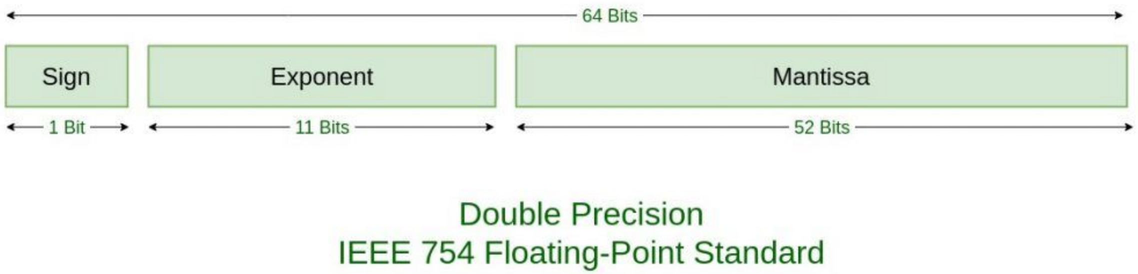


IEEE Floating Point



Integers up to 2^{52} (about 15 decimal digits) can be **exactly** represented

IEEE Floating Point

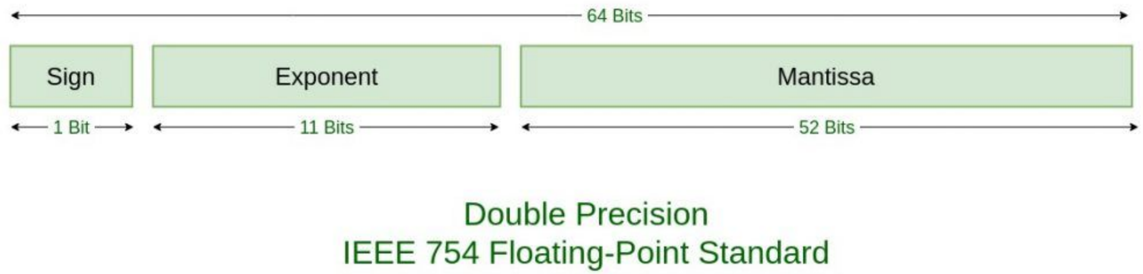


Integers up to 2^{52} (about 15 decimal digits) can be **exactly** represented

Fractions with power-of-2 denominators (e.g.: averages, midpoints) can be **exactly** represented

Arithmetic operations (`+`,`-`,`*`,`/`,`sqrt`) and comparisons (**including** `==`) Just Work so long as all intermediate values stay in the exactly representable range

IEEE Floating Point



Integers up to 2^{52} (about 15 decimal digits) can be **exactly** represented

Fractions with power-of-2 denominators (e.g.: averages, midpoints) can be **exactly** represented

Other numbers have up to 52 **bits of precision**. This goes down as errors accumulate in intermediate calculations

More Precision

`long double` sometimes* lets you use 80-bit floating point numbers with 64 bits of mantissa precision

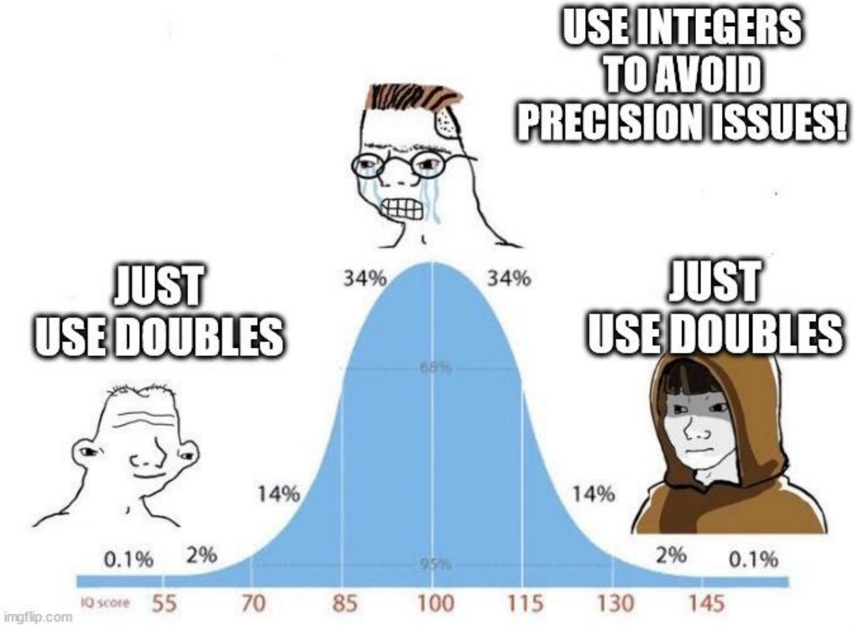
*: non-standard, but supported by e.g. GCC on x64

More Precision

`long double` sometimes* lets you use 80-bit floating point numbers with 64 bits of mantissa precision

- these can exactly represent **all** `int64`s!

*: non-standard, but supported by e.g. GCC on x64



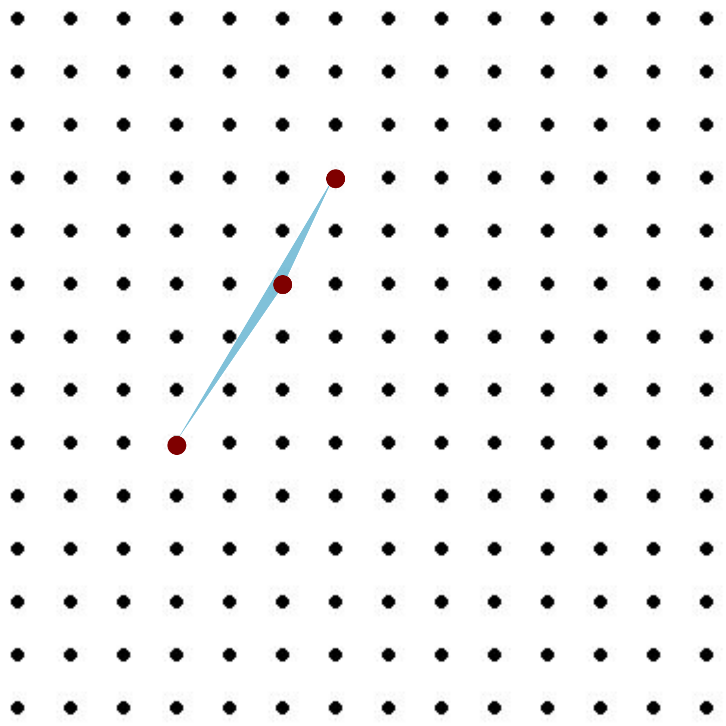
The Minimum Every Competitive Programmer Needs To Know About Floating Point

The rabbit hole goes deeper:

- extended-precision FPU registers
- denormalized numbers
- IEEE rounding modes
- `errno` and hardware traps
- signed `0` and `inf`
- signaling and quiet `nan`

One Precision Pitfall

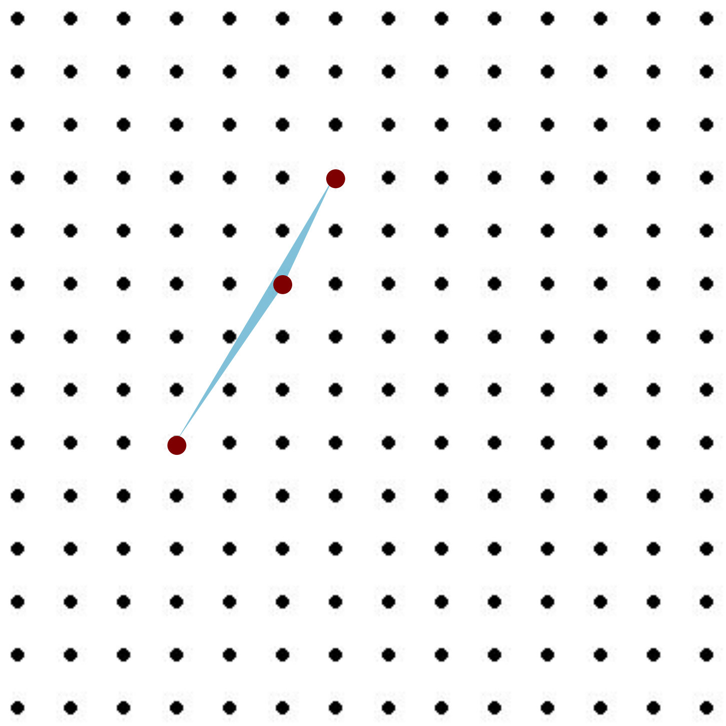
Given points on an $N \times N$ integer grid:



What is the smallest possible triangle area?

One Precision Pitfall

Given points on an $N \times N$ integer grid:



What is the smallest possible triangle area? $\frac{1}{2}$

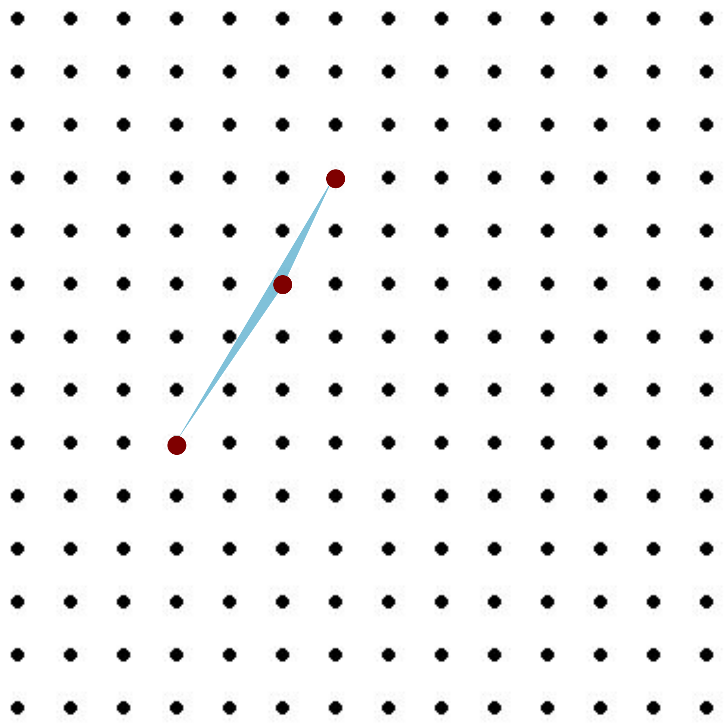
- for triangle $(x_0, y_0), (x_1, y_1), (x_2, y_2)$,

$$\text{Area} = \tfrac{1}{2} \det \begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}$$

$$= \frac{(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)}{2}$$

One Precision Pitfall

Given points on an $N \times N$ integer grid:

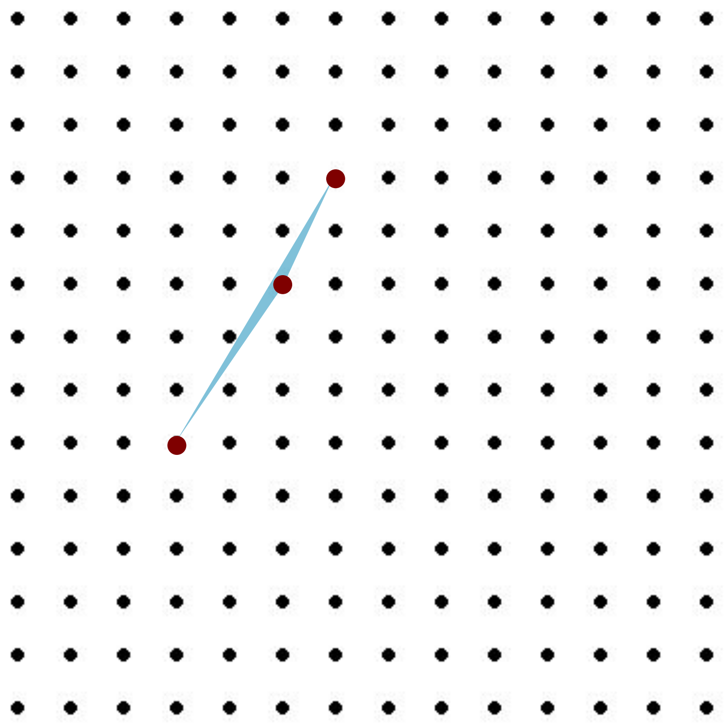


What is the smallest possible triangle area? $\frac{1}{2}$

What is the smallest possible angle?

One Precision Pitfall

Given points on an $N \times N$ integer grid:



What is the smallest possible triangle area? $\frac{1}{2}$

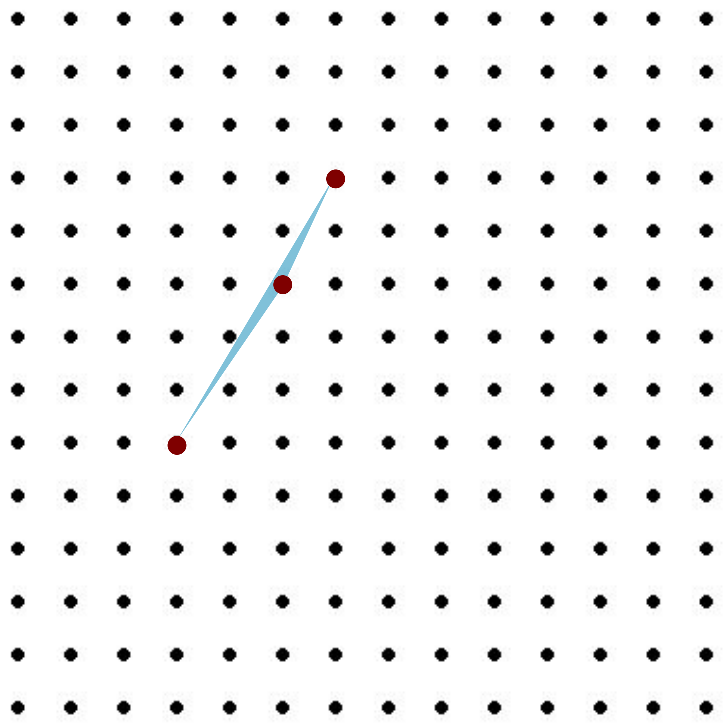
What is the smallest possible angle?

$$\text{Area} = \frac{1}{2} AB \sin\theta$$

$$\theta \approx \sin\theta = \frac{2\text{Area}}{AB}$$

One Precision Pitfall

Given points on an $N \times N$ integer grid:



What is the smallest possible triangle area? $\frac{1}{2}$

What is the smallest possible angle?

$$\text{Area} = \frac{1}{2} AB \sin\theta$$

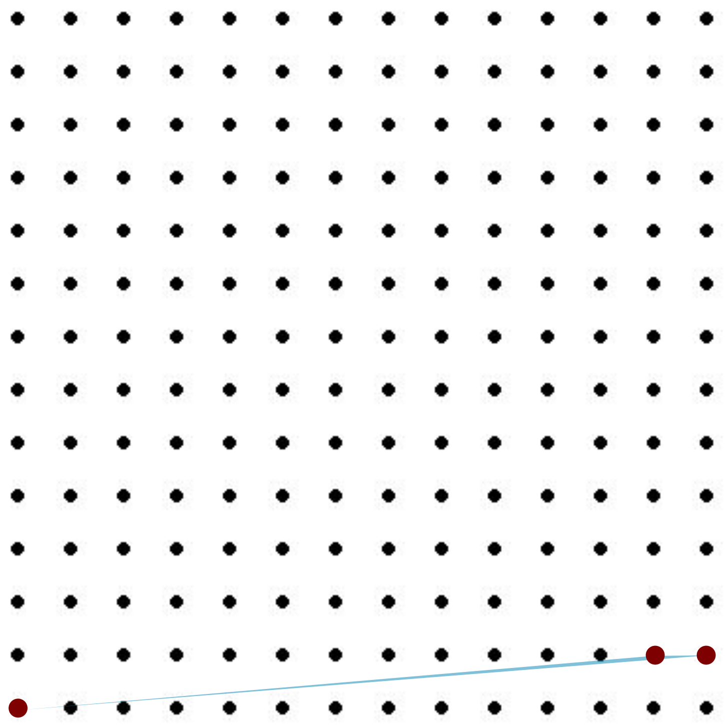
$$\theta \approx \sin\theta = \frac{2\text{Area}}{AB}$$

at least 1/2

at most $O(N^2)$

One Precision Pitfall

Given points on an $N \times N$ integer grid:



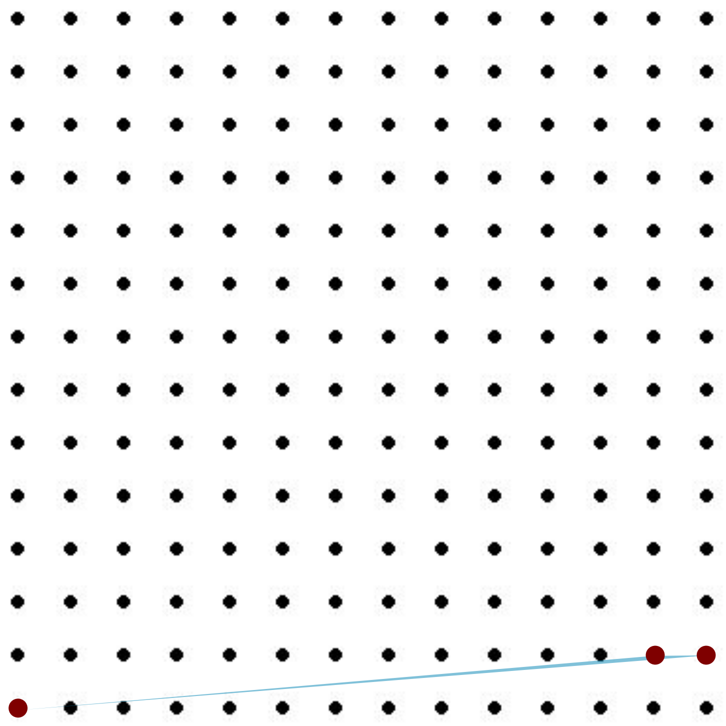
What is the smallest possible triangle area? $\frac{1}{2}$

What is the smallest possible angle?

$O(1/N^2)$

One Precision Pitfall

Given points on an $N \times N$ integer grid:

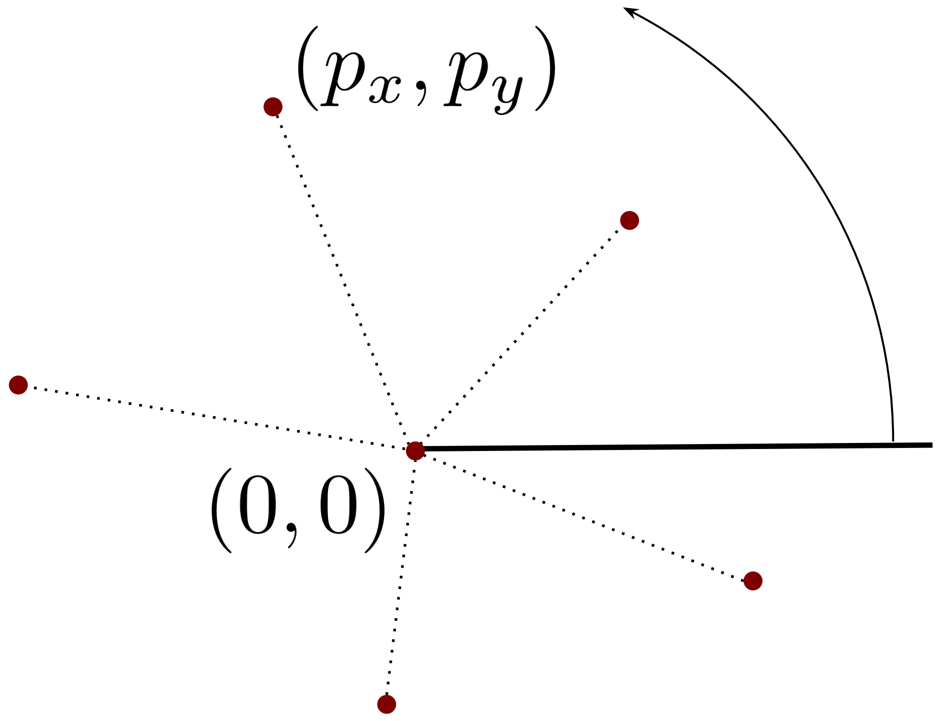


What is the smallest possible triangle area? $\frac{1}{2}$

What is the smallest possible angle?
$O(1/N^2)$

Punchline: if N is too large ($N \gtrsim 2^{26}$), `doubles` **cannot** reliably tell angles apart

Angle Sweeps



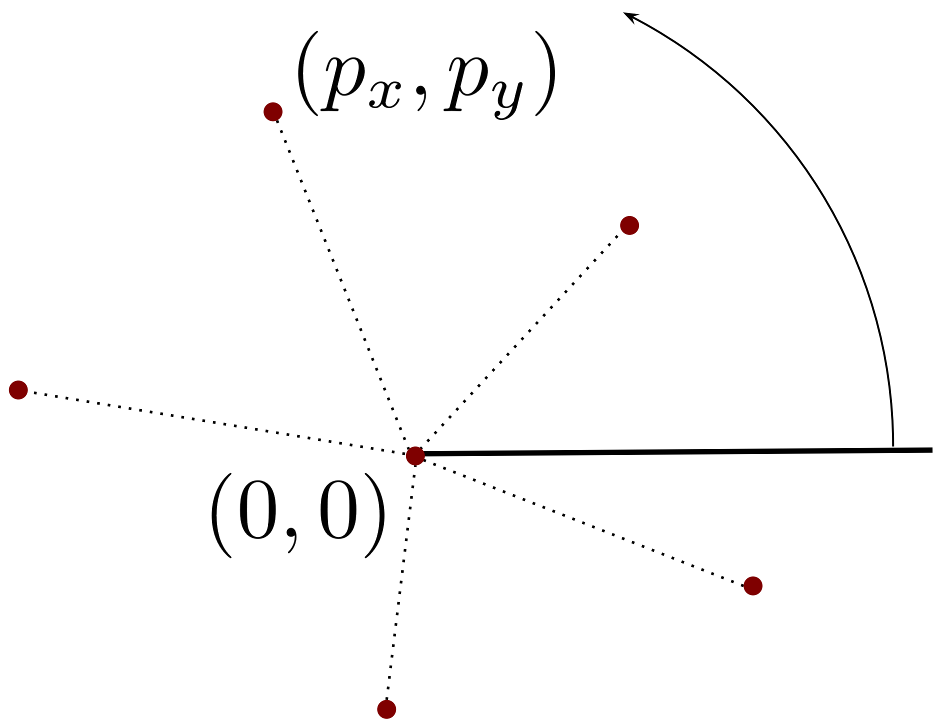
Sorting points around origin (easy way):

- simply sort by $p_\theta = \mathrm{atan2}(p_y, p_x)$

Use when:

- slight errors don't matter when angles are very similar
- detecting identical angles isn't needed

Angle Sweeps



Sorting points around origin (easy way):

- simply sort by $p_\theta = \mathrm{atan2}(p_y, p_x)$

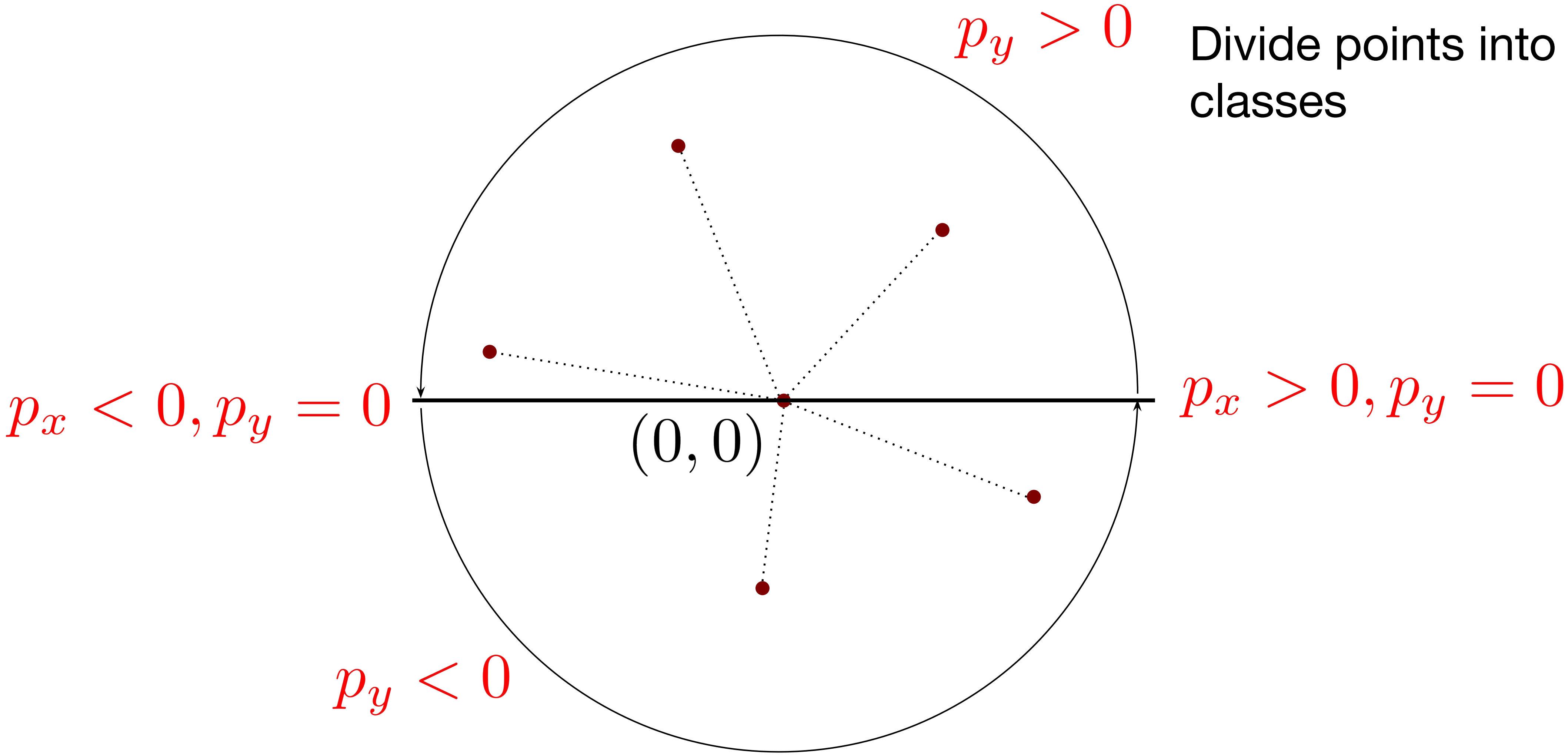
Use when:

- slight errors don't matter when angles are very similar
- detecting identical angles isn't needed

Example:

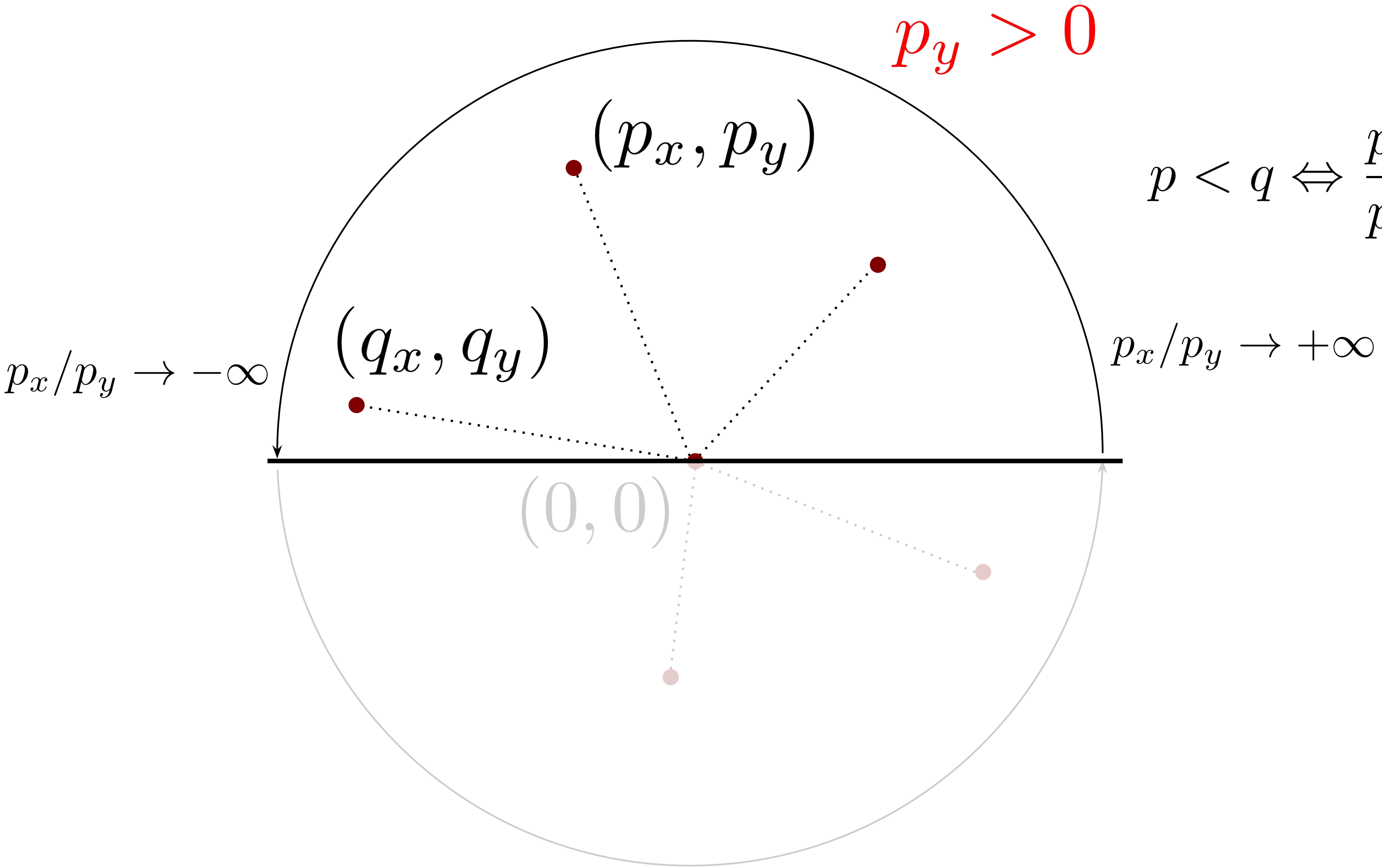
- compute the fraction of the circle covered by a union of angle intervals, up to some tolerance

When Exact Sorting Is Needed



Divide points into four classes

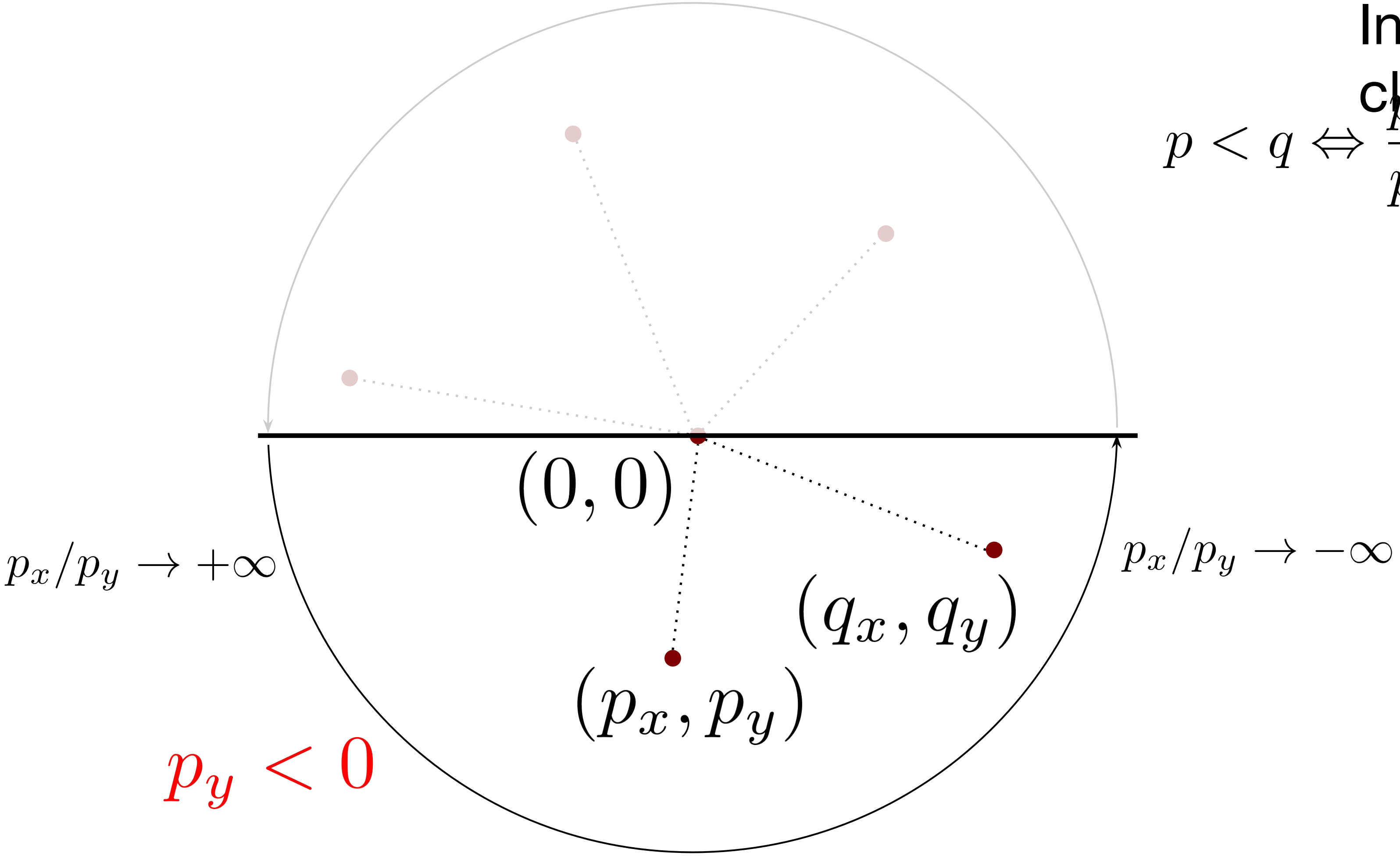
When Exact Sorting Is Needed



In top class:

$$p < q \Leftrightarrow \frac{p_x}{p_y} > \frac{q_x}{q_y} \Leftrightarrow p_y q_x < p_x q_y$$

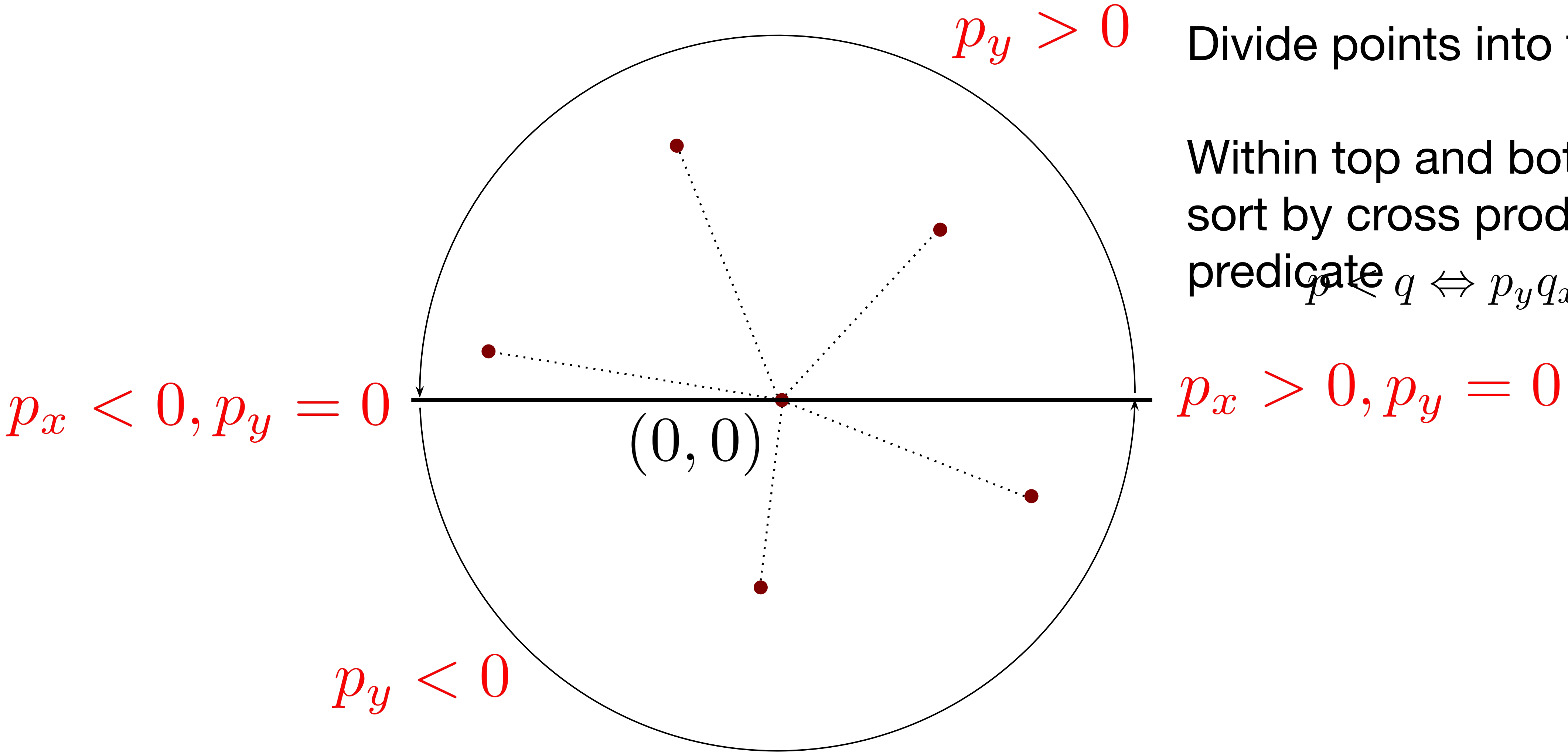
When Exact Sorting Is Needed



In bottom class:

$$p < q \Leftrightarrow \frac{p_x}{p_y} > \frac{q_x}{q_y} \Leftrightarrow p_y q_x < p_x q_y$$

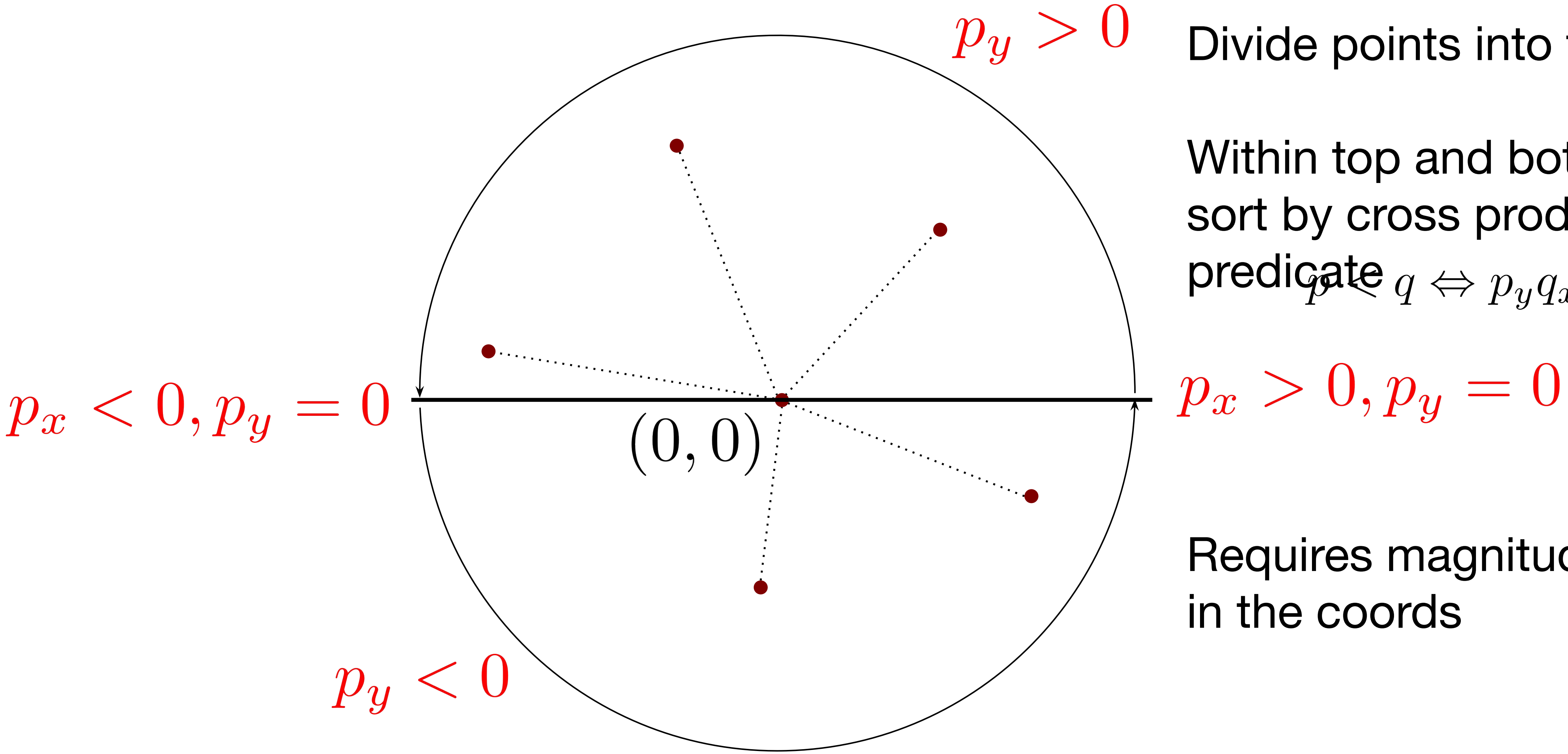
When Exact Sorting Is Needed



Divide points into four classes

Within top and bottom class, sort by cross product predicate $p < q \Leftrightarrow p_y q_x < p_x q_y$

When Exact Sorting Is Needed

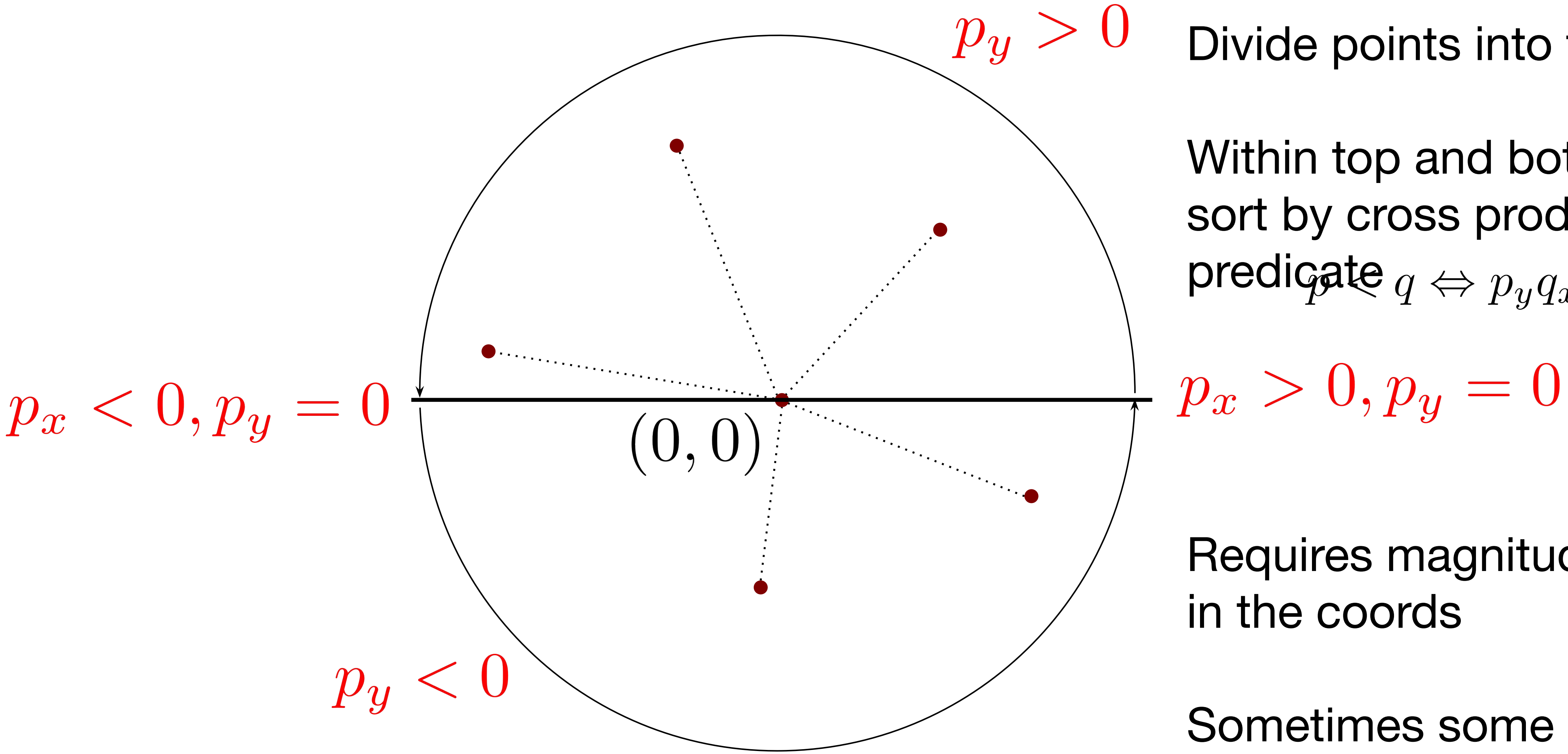


Divide points into four classes

Within top and bottom class, sort by cross product predicate $p < q \Leftrightarrow p_y q_x < p_x q_y$

Requires magnitudes quadratic in the coords

When Exact Sorting Is Needed



Divide points into four classes

Within top and bottom class, sort by cross product predicate $p < q \Leftrightarrow p_y q_x < p_x q_y$

Requires magnitudes quadratic in the coords

Sometimes some classes are impossible, simplifying this

Geometry Toolbox Checklist

Exact Predicates:
- point on segment
- point in polygon
- segment-segment intersection
- segment-circle intersection

Formulas / Subroutines:
- basic trig
- polygon area
- segment-segment distance
- segment-circle distance
- circle-circle intersection points
- segments tangent to two circles
- circumscribed and inscribed circles

Algorithms and Data Structures: