Basic notions about (geometric) algorithms

Rodrigo Silveira

What we will need

Knowledge about

- Geometry
- Combinatorics
- Algorithms
- Data structures

What we will need

Knowledge about

- Geometry
- Combinatorics
- Algorithms
- Data structures

Analysis of algorithms

- Time requirements (*An* $O(n^2 \log n)$*-time algorithm*)
- Space requirements (*An algorithm that uses* $O(n^2)$ *space*)
- Simplicity
- Robustness
- Correctness!

Model of computation

How long does it take to run an algorithm?

```
sum ← 0
for i = 1 to n do
    sum ← sum + i
end for
```

Model of computation

How long does it take to run an algorithm?

```
sum ← 0
for i = 1 to n do
    sum ← sum + i
end for
```

Actual running time (seconds) vs theoretical running time

Depends on...

- Computer
- Programming language
- A concrete implementation (thus on the programmer's skills!)

Defines a "standard computer":

- Units of space (variables)
- Units of time (primitive operations)
- We study time and spatial *complexity*

Theoretical simplification that allows us to **compare algorithms**

The model in computational geometry: Real RAM

The Real RAM

Primitive operations with real numbers take constant time

The model in computational geometry: Real RAM

The Real RAM

Primitive operations with real numbers take constant time

Space units

- Each memory unit (variable) can hold a real number $(2, 1/3, \pi, \dots)$
- Thus to store a number you need 1 unit, for an array with n numbers you need n units, and so on.
- Accessing a variable takes constant time

The model in computational geometry: Real RAM

The Real RAM

Primitive operations with real numbers take constant time

Space units

- Each memory unit (variable) can hold a real number $(2, 1/3, \pi, \dots)$
- Thus to store a number you need 1 unit, for an array with n numbers you need n units, and so on.
- Accessing a variable takes constant time

Primitive operations (take constant time)

- Comparing two real numbers $(<, \leq, =, \neq, >, \geq)$
- Arithmetic operations $(+, -, *, /)$
- Usual analytic functions $(\log x,\ \sqrt{x},\ \cos(x))$
- Maybe even taking the integer part of a real number $(\lfloor x \rfloor)$ (less standard)

Asymptotic Analysis

How we measure time and space

In principle, we count units of time and space, as functions of the *size* of the input.

Asymptotic Analysis

How we measure time and space

In principle, we count units of time and space, as functions of the *size* of the input.

```
{Sums numbers from 1 to n}
sum ← 0
for i = 1 to n do
    sum ← sum + i
end for
return sum
```

How many units of time does this algorithm take?

Asymptotic Analysis

How we measure time and space

In principle, we count units of time and space, as functions of the *size* of the input.

```
{Sums numbers from 1 to n}
sum ← 0
for i = 1 to n do
    sum ← sum + i
end for
return sum
```

How many units of time does this algorithm take?

We neglect constants

We are interested in the "order of magnitude" of the running time or space.

If it is n, $17n + 24$ or $0.3n$, we will simply say it is "in the order of n"

- We study the asymptotic behavior...
- ...and use **asymptotic notation**

Asymptotic notation

Our way to express running time and space usage of algorithms

Let f and g be two positive, real-valued functions of the (integer) variable n

- If $\mathbf{f(n)} \in \mathbf{O(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$
  Informally: f is bounded above by g, asymptotically

Asymptotic notation

Our way to express running time and space usage of algorithms

Let f and g be two positive, real-valued functions of the (integer) variable n

- If $\mathbf{f(n) \in O(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$
  Informally: f is bounded above by g, asymptotically

- If $\mathbf{f(n) \in \Omega(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \geq c \cdot g(n)$
  Informally: f is bounded below by g, asymptotically

Asymptotic notation

Our way to express running time and space usage of algorithms

Let f and g be two positive, real-valued functions of the (integer) variable n

- If $\mathbf{f(n) \in O(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$
  Informally: f is bounded above by g, asymptotically

- If $\mathbf{f(n) \in \Omega(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \geq c \cdot g(n)$
  Informally: f is bounded below by g, asymptotically

- If $\mathbf{f(n) \in \Theta(g(n))}$, then $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$
  Informally: f is bounded above and below by g, asymptotically

Asymptotic notation

Our way to express running time and space usage of algorithms

Let f and g be two positive, real-valued functions of the (integer) variable n

- If $\mathbf{f(n) \in O(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$
  Informally: f is bounded above by g, asymptotically

- If $\mathbf{f(n) \in \Omega(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \geq c \cdot g(n)$
  Informally: f is bounded below by g, asymptotically

- If $\mathbf{f(n) \in \Theta(g(n))}$, then $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$
  Informally: f is bounded above and below by g, asymptotically

- If $\mathbf{f(n) \in o(g(n))}$, then $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$
  Informally: f is dominated by g, asymptotically

Asymptotic notation

Our way to express running time and space usage of algorithms

Let f and g be two positive, real-valued functions of the (integer) variable n

- If $\mathbf{f(n) \in O(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$
  Informally: f is bounded above by g, asymptotically
- If $\mathbf{f(n) \in \Omega(g(n))}$, then there is an integer n_0 and some constant $c > 0$ such that $\forall n \geq n_0$, $f(n) \geq c \cdot g(n)$
  Informally: f is bounded below by g, asymptotically
- If $\mathbf{f(n) \in \Theta(g(n))}$, then $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$
  Informally: f is bounded above and below by g, asymptotically
- If $\mathbf{f(n) \in o(g(n))}$, then $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$
  Informally: f is dominated by g, asymptotically

Examples: $(2n^2 + 5n) \in O(n^2)$, and $(2n^2 + 5n) \in \Omega(n^2)$, thus $(2n^2 + 5n) \in \Theta(n^2)$

Linear functions are in $\Theta(n)$, quadratic functions are all in $\Theta(n^2)$.
Constant functions are all in $\Theta(1)$.

Common functions used for running times

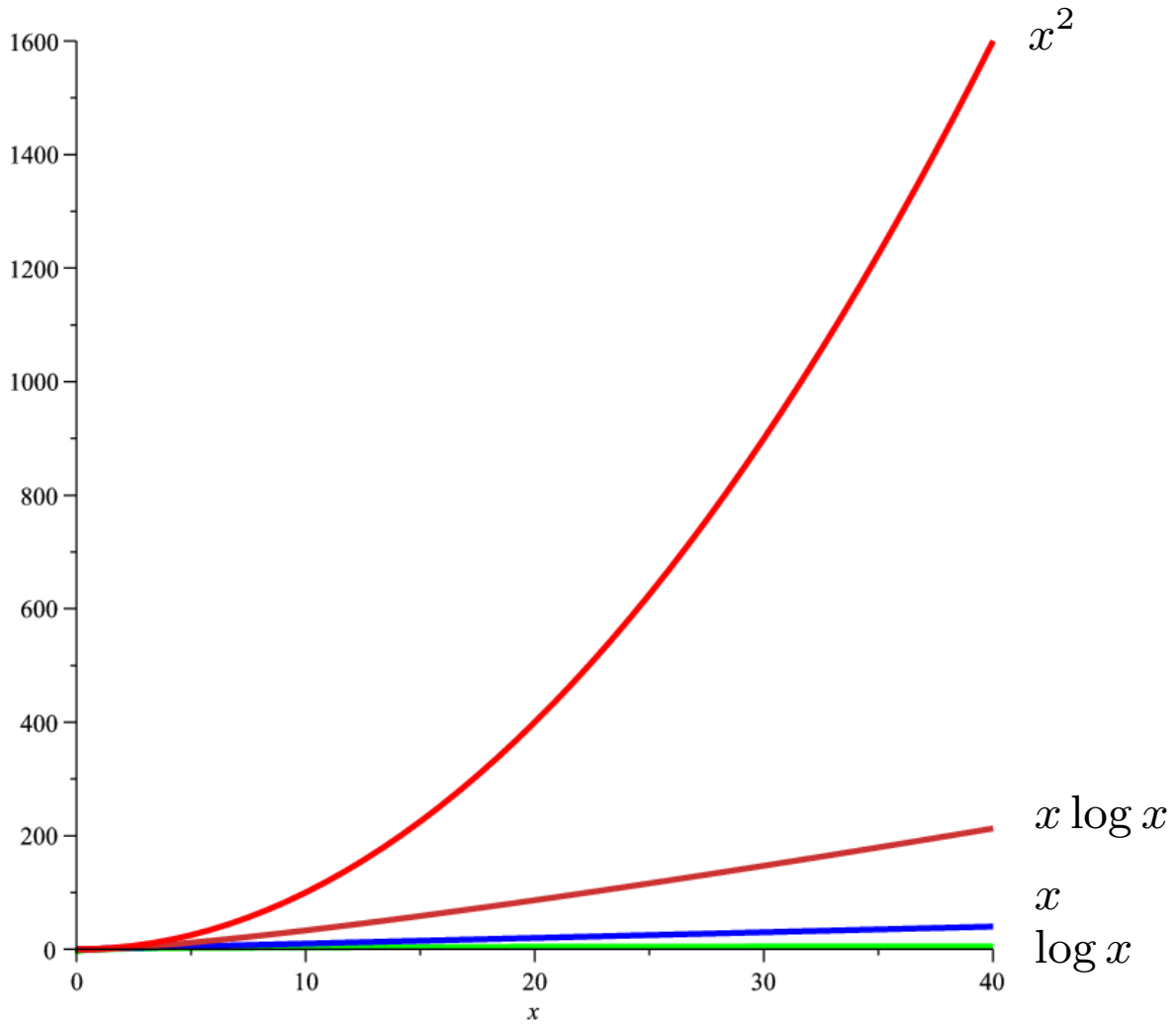
Typical running times of geometric algorithms use functions like:

- $O(\log n)$
- $O(\sqrt{n})$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- $O(n^3)$

Common functions used for running times

Typical running times of geometric algorithms use functions like:

- $O(\log n)$
- $O(\sqrt{n})$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- $O(n^3)$



Common functions used for running times

Comparison of execution times

Assuming a speed of 1 millon operations per second

Cost	$n = 10$	$n = 20$	$n = 100$	$n = 18000$
$\log n$	0.004 ms	0.005 ms	0.007 ms	0.018 ms
n	0.01 ms	0.02 ms	0.1 ms	18 ms
$n \log n$	0.033 ms	0.09 ms	0.66 ms	254 ms
n^2	0.1 ms	0.4 ms	10 ms	5 min 24 sec
n^4	10 ms	160 ms	1 min 40 sec	3 328 years
2^n	1 ms	1.05 sec	2.7×10^6 UA	
$n!$	3.6 sec	76 000 years	2×10^{134} UA	
n^n	2 h 48 min	220 UA	2×10^{176} UA	

UA: age of the universe (15 thousand millions years)

Running time analysis

Running time of an algorithm

Often the exact running time of an algorithm depends on the contents on the input (and not only on its size)

[(2,3);(-4,3);(-1,2);$\cdots$;(-4,7)]

[(-4,7);(-4,3);(-1,2);$\cdots$;(2,3)]

{Find first point with positive x-coordinate, in array with n points}
$i \leftarrow 0$
while points[i].x≤ 0 **do**
  $i \leftarrow i + 1$
end while
return points[i]

(Assuming there is at least one such point)

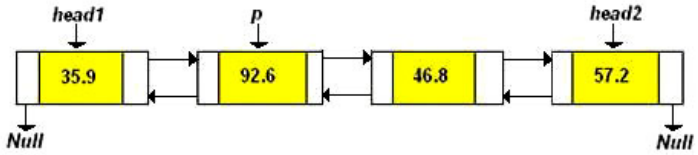
Worst-case running time

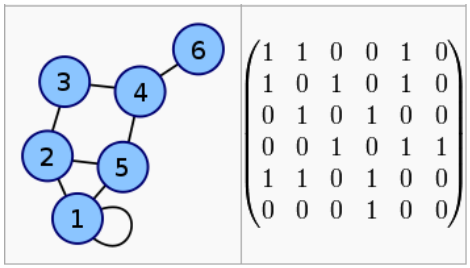
In general, we will analyze the **worst-case running time**. We will say that an algorithm has running time $O(f(n))$ if the running time is $O(f(n))$ **for any possible input**.

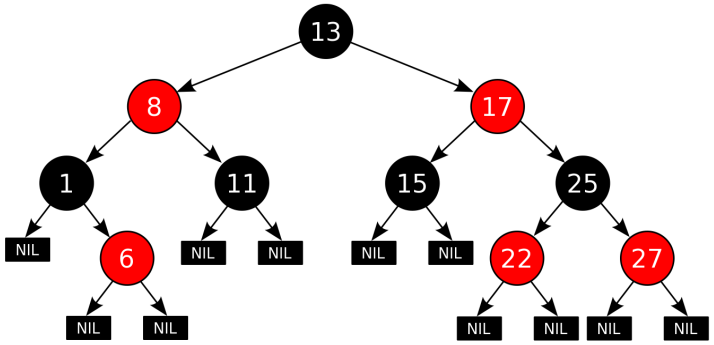
Some useful data structures

Basic data structures used for geometric algorithms include:

- List, queues
- Priority queues
- Data structures for representing graphs (adjacency matrix/list)
- Search trees (usually balanced binary search trees, like AVLs or red-black trees)
- Dictionaries



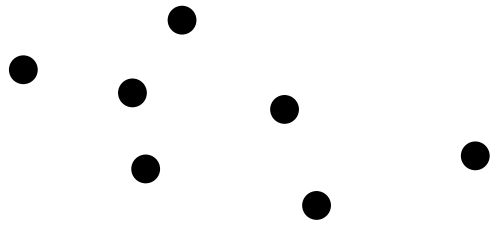




Basic geometric elements

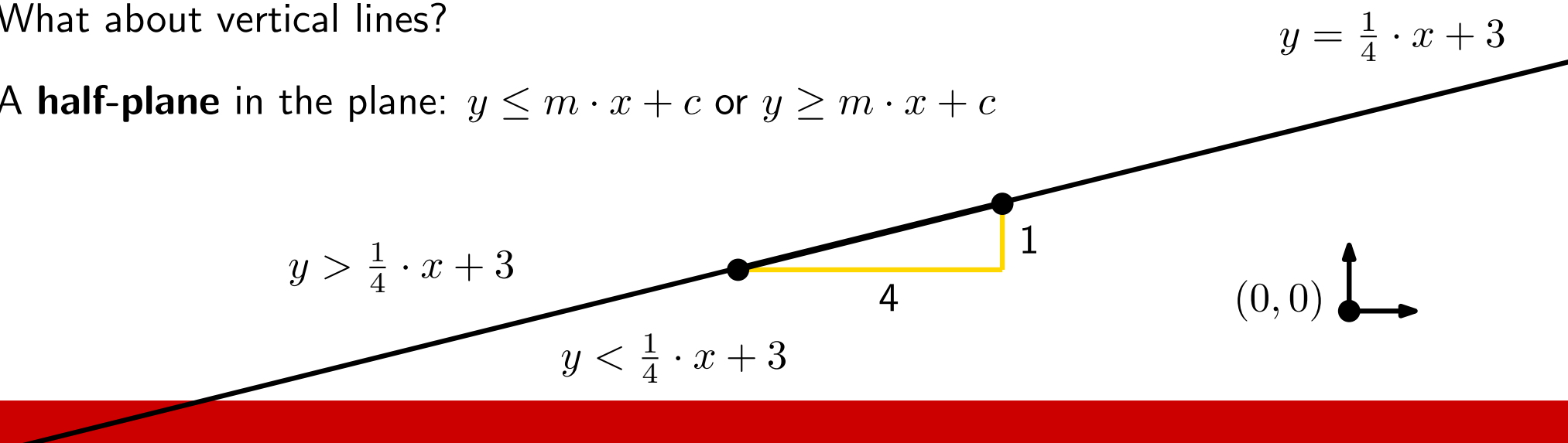
Where they live

- Plane (two-dimensional), $\mathbb{R}^2$
- Space (three-dimensional), $\mathbb{R}^3$
- Space (higher-dimensional), $\mathbb{R}^d$



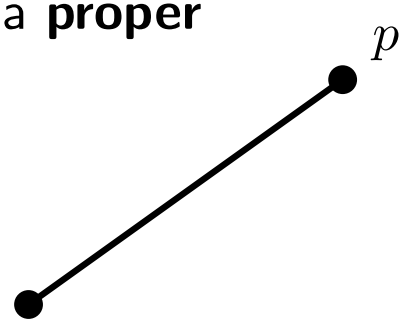
Who they are

- A **point** in the plane, 3-dimensional space, higher-dimensional space.
  $p = (p_x, p_y)$, $p = (p_x, p_y, p_z)$, $p = (p_1, p_2, \ldots, p_d)$
- A **line** in the plane: $y = m \cdot x + c$; representation by m and c
  What about vertical lines?
- A **half-plane** in the plane: $y \leq m \cdot x + c$ or $y \geq m \cdot x + c$



Line segments

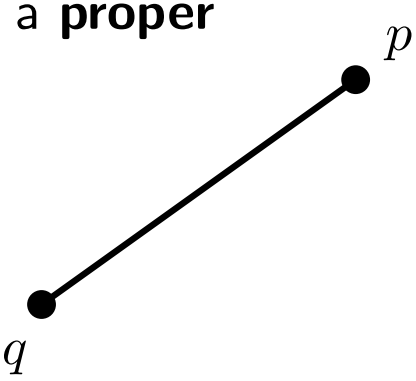
- A **line segment** $\overline{pq}$ is defined by its two endpoints p and q.
- Line segments are assumed to be **closed** $=$ with endpoints, not **open**
- Two line segments **intersect** if they have some point in common. It is a **proper intersection** if it is exactly one interior point of each line segment



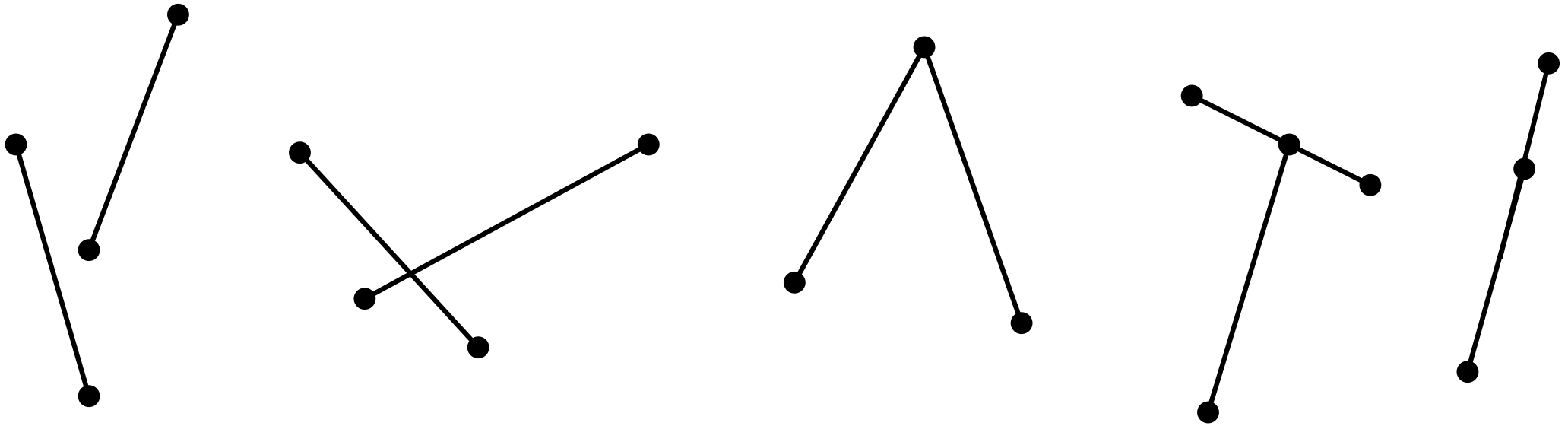
- **Question:** In how many ways can two segments intersect?

Line segments

- A **line segment** $\overline{pq}$ is defined by its two endpoints p and q.
- Line segments are assumed to be **closed** $=$ with endpoints, not **open**
- Two line segments **intersect** if they have some point in common. It is a **proper intersection** if it is exactly one interior point of each line segment

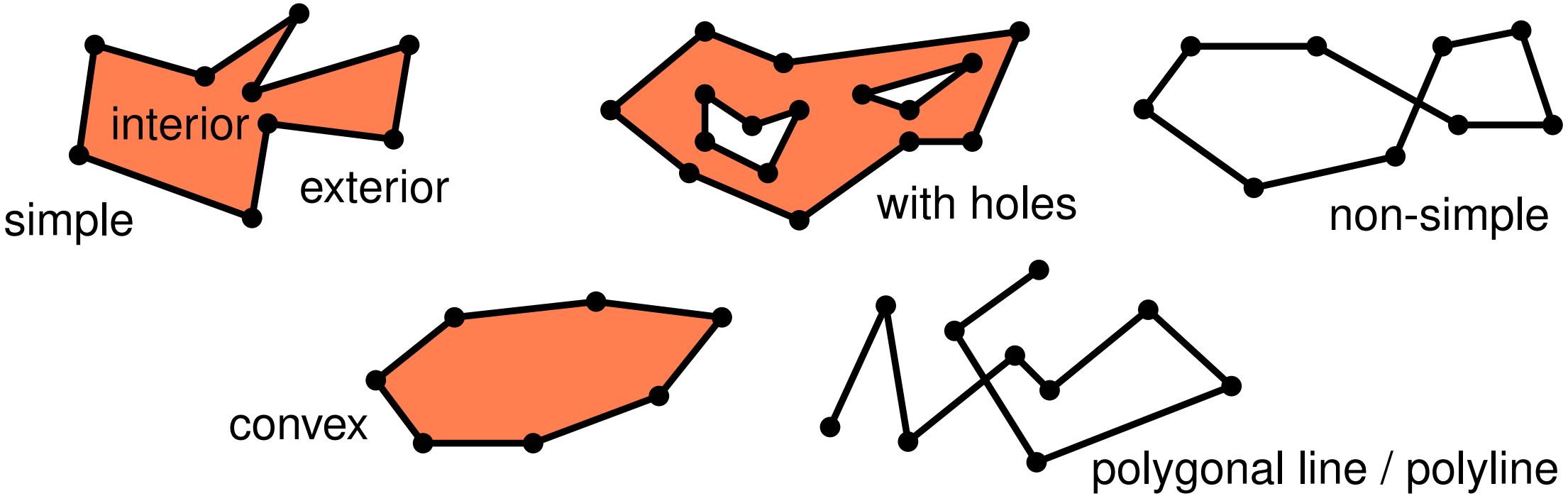


- **Question:** In how many ways can two segments intersect?



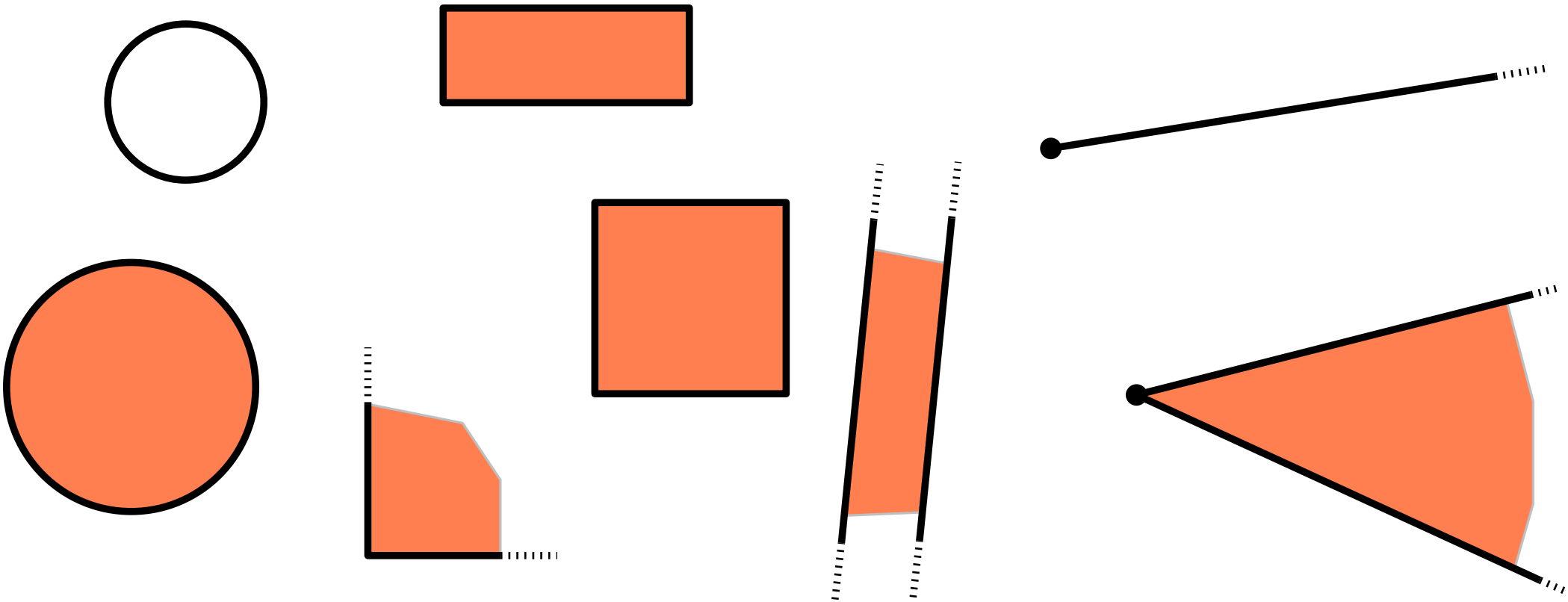
Polygons

- A **polygon** is a connected region of the plane bounded by a sequence of line segments.
- The line segments of a polygon are called its **edges**, the endpoints of those edges are the **vertices**.
- Many flavors: simple, convex, with holes, non-simple, etc.



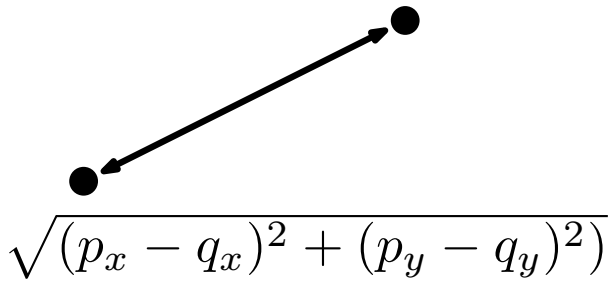
Other popular shapes

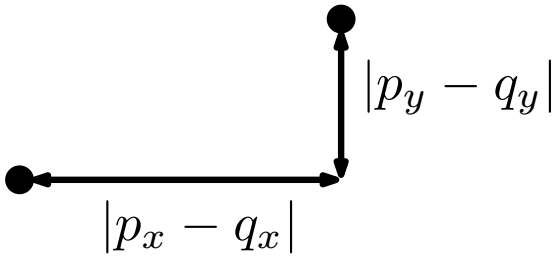
- **Circle** (only the boundary), and **disk** (boundary plus the interior).
- **Rectangle**, **square**, **quadrant**.
- **Slabs**, **half-lines**, **wedges**.



Distances

- The distance between two points is usually the **Euclidean distance**:
  $\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$
- There are other popular options, for example, the **Manhattan distance** (L_1 metric).



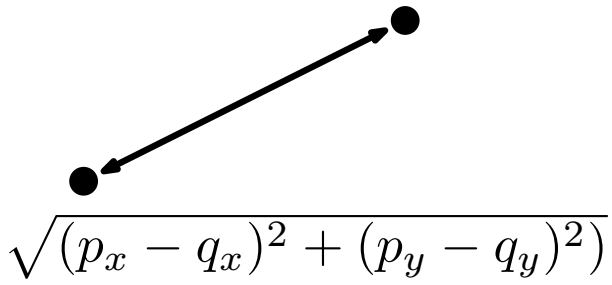


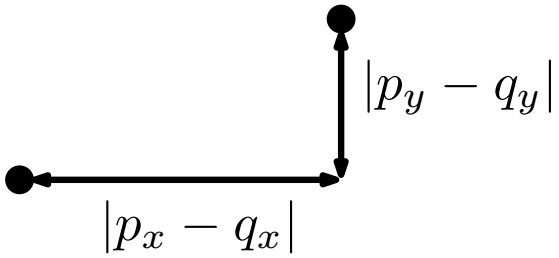
Distances

- The distance between two points is usually the **Euclidean distance**:

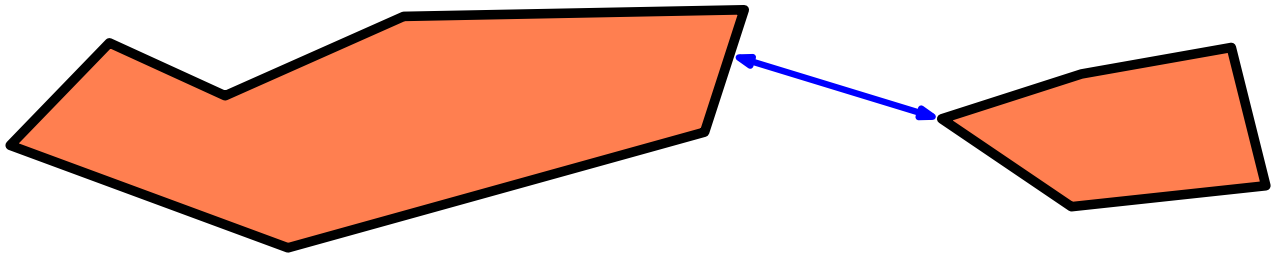
  $\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$

- There are other popular options, for example, the **Manhattan distance** (L_1 metric).





- The distance between two geometric objects other than points usually refers to the minimum distance between two points that are part of these objects

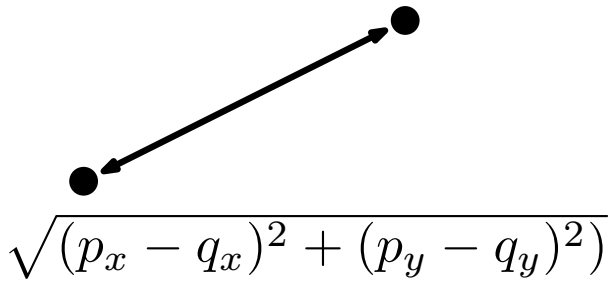


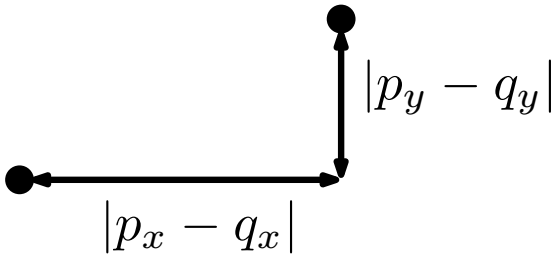
Distances

- The distance between two points is usually the **Euclidean distance**:

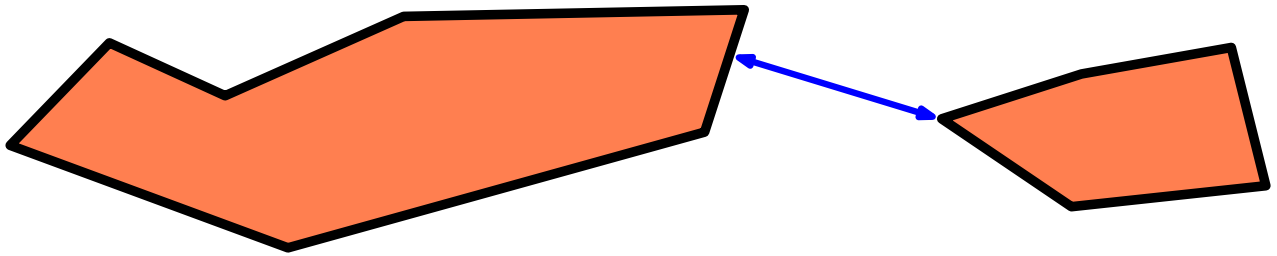
  $\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$

- There are other popular options, for example, the **Manhattan distance** (L_1 metric).





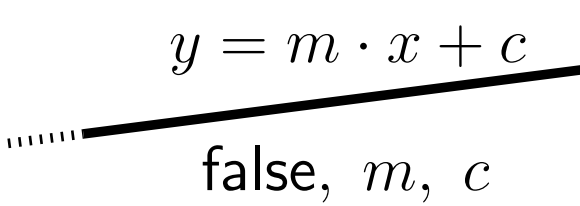
- The distance between two geometric objects other than points usually refers to the minimum distance between two points that are part of these objects



- **Question:** How can the distance between two line segments be realized?

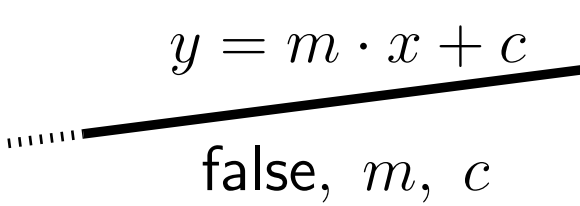
Description size

- A point in the plane can be represented using two reals
- A line in the plane can be represented using two reals and a Boolean (for example)
- A line segment can be represented by two points, so four reals
- A circle (or disk) requires three reals to store it (center, radius)
- A rectangle requires four reals to store it

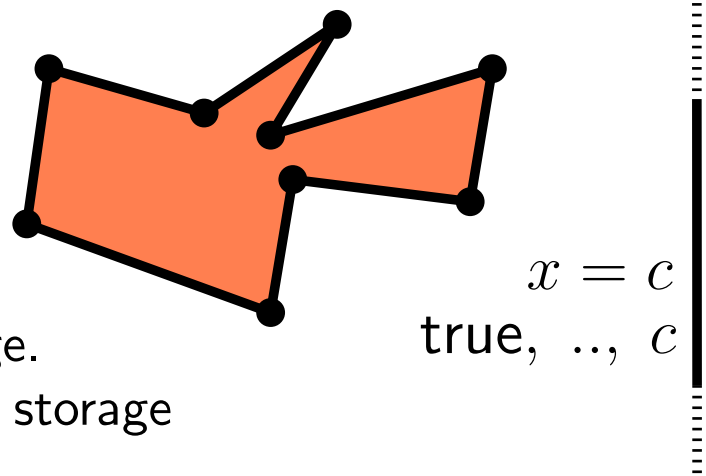


Description size

- A point in the plane can be represented using two reals
- A line in the plane can be represented using two reals and a Boolean (for example)
- A line segment can be represented by two points, so four reals
- A circle (or disk) requires three reals to store it (center, radius)
- A rectangle requires four reals to store it



- A simple polygon in the plane can be represented using $2n$ reals if it has n vertices (and necessarily, n edges)
- A set of n points requires $2n$ reals
- A set of n line segments requires $4n$ reals
- A point, line, circle, ... requires $O(1)$, or constant, storage.
  A simple polygon with n vertices requires $O(n)$, or linear, storage



Questions?

Questions?