Algorithmics II (H)

Prof David Manlove

email `david.manlove@glasgow.ac.uk`

Lectures: Mondays 1300-1400 (ASBS 584)
Thursdays 1000-1100 (Rankine 108)

Examples classes: Thursdays 1500-1600 (ASBS 489)

Office hours: Tuesdays 1600-1700 (see Moodle page)

Pre-requisites

- Algorithmics I (H) (or equivalent)
- Java

Assessment

- Degree exam 70%
- Assessed exercise (1) 20%
- Quizzes 10%

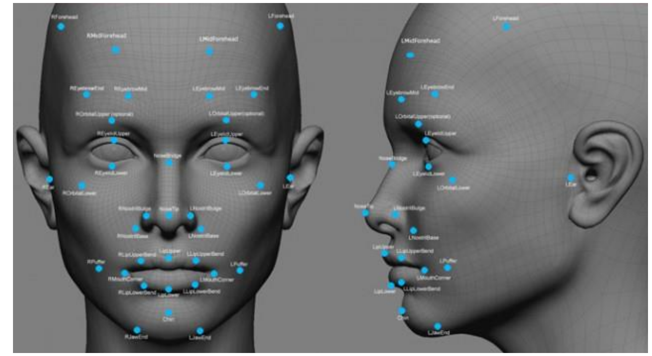
Overview of the course

- Primarily about *efficient algorithms*
- A huge range of problems / algorithms to choose from
  - examples chosen to provide a mixture of *theory* and *application*
  - many illustrate general algorithm design techniques
- Builds on Algorithmics I (H), which covered:
  - Sorting and tries
  - Strings and text algorithms
  - Graphs and graph algorithms
  - NP-completeness fundamentals
  - Computability
- We don't focus on:
  - Computability theory
  - NP-completeness (for its own sake)

Why study algorithms?

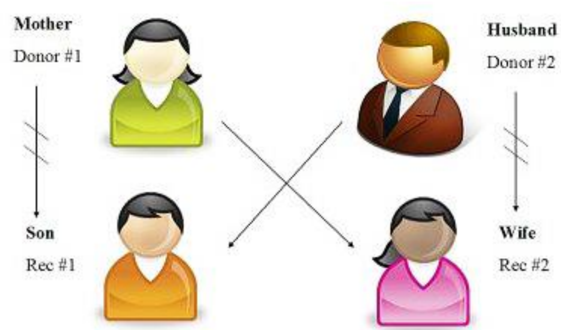
Google (Pagerank)

Heathrow: runway scheduling

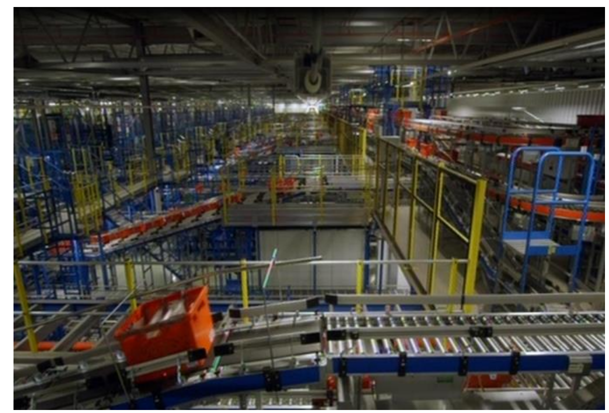


Facial recognition

Junior doctor allocation



Kidney exchange



Ocado: picking and packing groceries

"The Secret Rules of Modern Living: Algorithms"
First shown 24 Sep 2015, BBC4

Some examples of applications for which we will see efficient algorithms

- **File comparison:** find a minimal set of differences between two files – as in the Unix *diff* utility
- **Pattern matching:** Find occurrences in a file of text matching a given regular expression – as in *grep*
- **Communication networks:** Find the largest flow of data that we can broadcast through a network from a transmitter to a receiver
- **Allocation of scarce resources:** find the best way of allocating applicants to positions (or students to projects) taking preferences into account
- **Transportation:** find the best way to deliver items to addresses to minimise the total distance travelled

Why are *efficient* algorithms important?

- Assume a problem has several algorithms with different time complexity functions
- We measure their running times on inputs of different sizes n
  - Times are in seconds (unless otherwise stated)
  - Assumes 10^9 operations per second

Complexity	Input size n			
	1,000	10,000	100,000	1,000,000
n	10^{-6}	10^{-5}	10^{-4}	10^{-3}
$n \log_2 n$	10^{-5}	1.3×10^{-4}	0.0016	0.02
n^2	0.001	0.1	10	16 mins
n^3	1	16 mins	11 days	32 yrs

Table 1. Polynomial-time algorithms

Why are *efficient* algorithms important (cont.)?

Complexity	Input size				
	10	20	30	40	50
2^n	10^{-6}	0.001	1.1	18 mins	4 months
n!	0.004	77 yrs	*	*	*

* means longer than the age of the Universe

Table 2. Exponential-time algorithms

- Clear distinction in rate of growth between polynomial-time algorithms and exponential-time algorithms
- Still true even if we assume a CPU 10x as fast, or 100x as fast…
- When we refer to *efficient* algorithms, we mean *polynomial-time* algorithms

Structure of the course

Split (roughly) into four quarters:

1. *Geometric algorithms*
   - Intersection of two line segments
   - Constructing a simple polygon
   - Finding the convex hull
   - Finding a closest pair of points
   - Intersection of horizontal and vertical line segments

2. *Text and string algorithms*
   - Introduction to suffix trees
   - Applications of suffix trees
   - Matching regular expressions
   - LCS of two strings using memoisation

Structure of the course (cont.)

3. Graph and matching algorithms

- Matching in bipartite graphs
- Network flow (2 lectures)
- Stable matching problems
- Floyd-Warshall algorithm

4. Algorithms for ‘hard’ problems

- Backtracking and branch-and-bound
- Integer programming and kidney exchange
- Pseudo-polynomial-time algorithms
- Constant-factor approximation algorithms
- Polynomial-time approximation schemes
- Limits to approximability

Resources

- **Lecture slides**
- **Some relevant texts**
  - M.H. Alsuwaiyel, *Algorithms: Design Techniques and Analysis*, World Scientific, 2016
  - T. Cormen et al, *Introduction to Algorithms*, 4th edn, MIT Press, 2022
  - M. Goodrich and R. Tamassia, *Algorithm Design: Foundations, Analysis, and Internet Examples*, Wiley, 2002
  - Link to reading list from the course website. (NB – background reading only!)
- **Tutorial exercises and solutions**
- **Moodle page** – course materials
  - https://moodle.gla.ac.uk/course/view.php?id=49793
- **Slido** – anonymous Q&A during lectures / examples classes
  - Event #64492 at slido.com or https://app.sli.do/event/blnurv0i

Good study practice

- Read slides ahead of lectures
- Come to lectures and examples classes!
- Make notes during lectures / examples classes
- Use Slido to ask / answer questions during lectures
- Review your notes after lectures
- Attempt tutorial questions before examples class
- Use office hours if required

Coursework

- **Assessed practical exercise will be handed out around 16 October with a deadline around 7 November (20%)**
  - **Based on the string and text algorithms section of the course**

- **Online quizzes aiming to incentivise engagement with the course (10%)**
  - **One quiz question per lecture / examples class**
  - **Designed to be simple, and answer should become clear during lecture / examples class**
  - **Correct answer: 1 mark; incorrect / missing answer: 0 marks**
  - **Answer via Moodle during the lecture / examples class**
  - **Best 24 (out of 29) quiz marks will be aggregated to give a band**
  - **If ill or have a job interview etc., let me know by email. Any affected quiz can be voided**
  - **See guidance notes on the quizzes on the Moodle page for more information**

Answering quiz questions

- **Refer to the quiz question and question number on the handout circulated at the lecture / examples class**
- **Navigate to the Moodle page for the course**
- **Under "Quizzes", click on the link for the relevant quiz question number**
- **You should see Quiz 1 which has 4 choices: A, B, C and D**
- **Select one as your answer to the question shown on the handout**
- **Review your answer if desired and then press "Submit all and finish"**
- **You can attempt the quiz multiple times, but only your last attempt will be counted**
- **The quiz can only be attempted during the lecture or examples class**

Feedback provided during the course

- Lectures
  - Answers will be given to questions asked via Slido
- Tutorial questions:
  - Solutions will be worked through at examples classes
  - Solutions to all questions will be provided via Moodle
  - Feedback will be given on responses provided verbally or via Slido
- Quiz questions:
  - You will receive feedback on your answers (whether correct, what correct answer is) and an overall grade at the end of the course
  - Solutions to previous quiz questions will be worked through at examples classes
- Assessed exercise:
  - General feedback on the assessed exercise will be given via a document circulated after marking is complete
  - Individual feedback will also be given:
    - Overall mark
    - Completed marksheet giving mark breakdown and comments
    - Annotated code (in some cases)

Part 1

Geometric Algorithms

- Determining whether two line segments intersect
- Constructing a simple polygon from a set of points
- Finding the convex hull of a set of points
  - the smallest convex polygon that includes them all
- Finding a closest pair among a set of points
- Finding all the intersections of a set of horizontal and vertical line segments

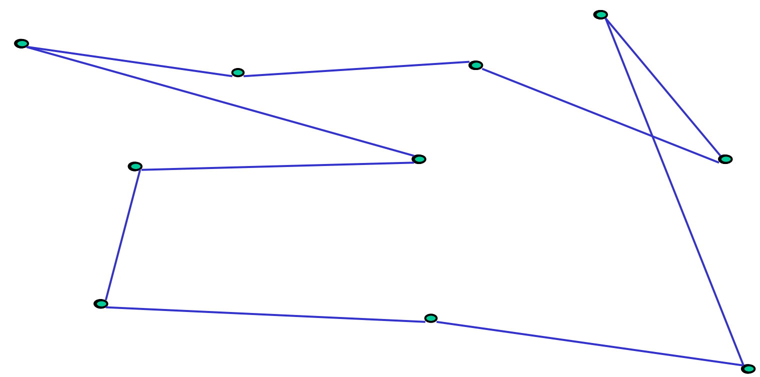
Geometric Objects

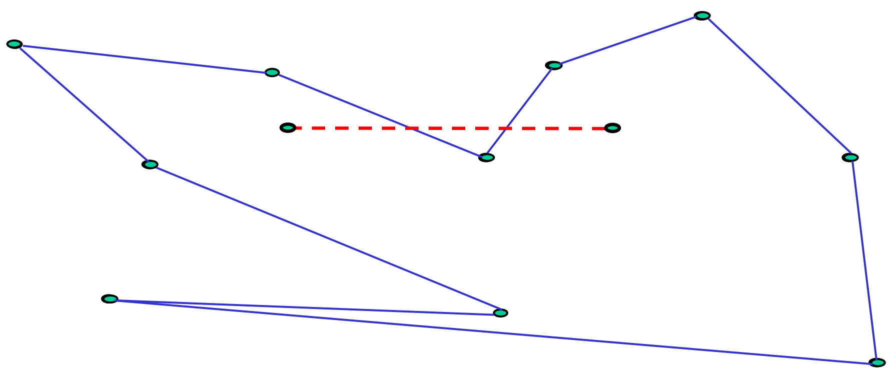
Some definitions

- **point**: a pair of coordinates (x, y)
- **line**: an (ordered) pair of points p, q denoted —p—q—
- **line segment**: a pair of end-points p, q denoted p—q
- **path**: a sequence of distinct points $p_1, \ldots, p_n$
- **polygon**: a path $p_1, \ldots, p_n$ with $p_1 = p_n$
- **simple polygon**: no self-intersections
  - encloses a region, its **inside**
- **convex polygon**
  - p, q inside $\Rightarrow$ p—q inside

Example polygons

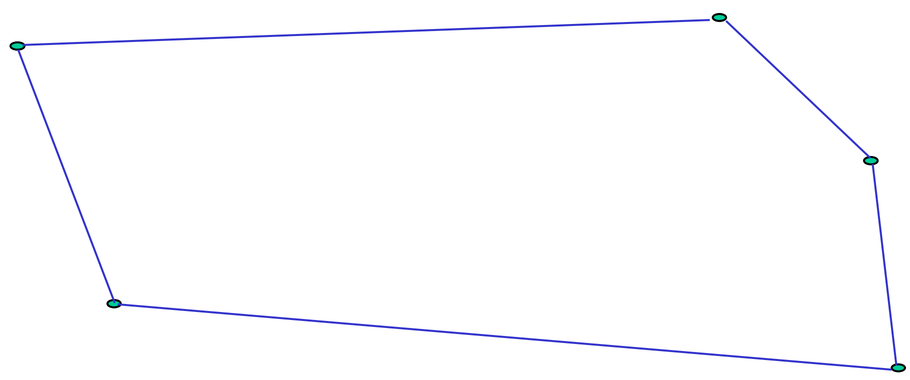
Polygon;
not simple





Simple polygon; not
convex

Convex polygon



Geometric Representations

- **point**: use `Point2D.Double`
- **line (segment)**:

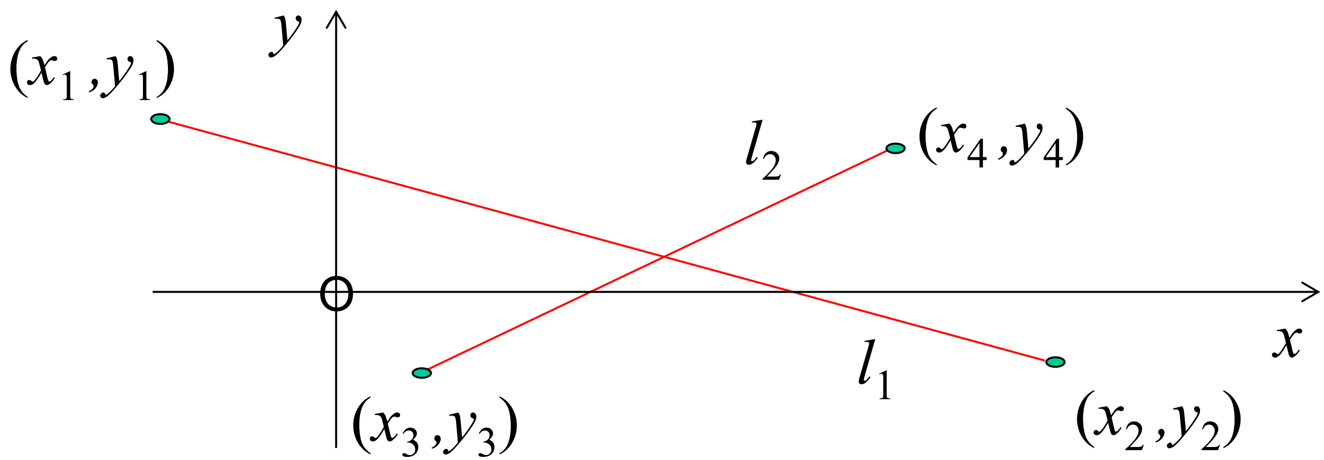
```
public class Line {
    public Point2D.Double p1;
    public Point2D.Double p2;
}
```

- **set of points**:

```
public class PointSet {
    public Point2D.Double [] pArray;
}
```

- **path / polygon**: order of points in array is important
  - abuse of syntax: if p is a `PointSet` object we will refer to p[k] etc.

Problem 1: determining if two line segments intersect



"High school" method: compute equation of each line:

$$(x_2-x_1)(y-y_1)-(y_2-y_1)(x-x_1)=0$$
$$(x_4-x_3)(y-y_3)-(y_4-y_3)(x-x_3)=0$$

Solve to find point of intersection (if any) – (p,q)
Check whether $x_1 \le p \le x_2$ and $x_3 \le p \le x_4$

Involves division! (Relatively costly and potentially inaccurate due to floating point representation)

Can we do it without using division?

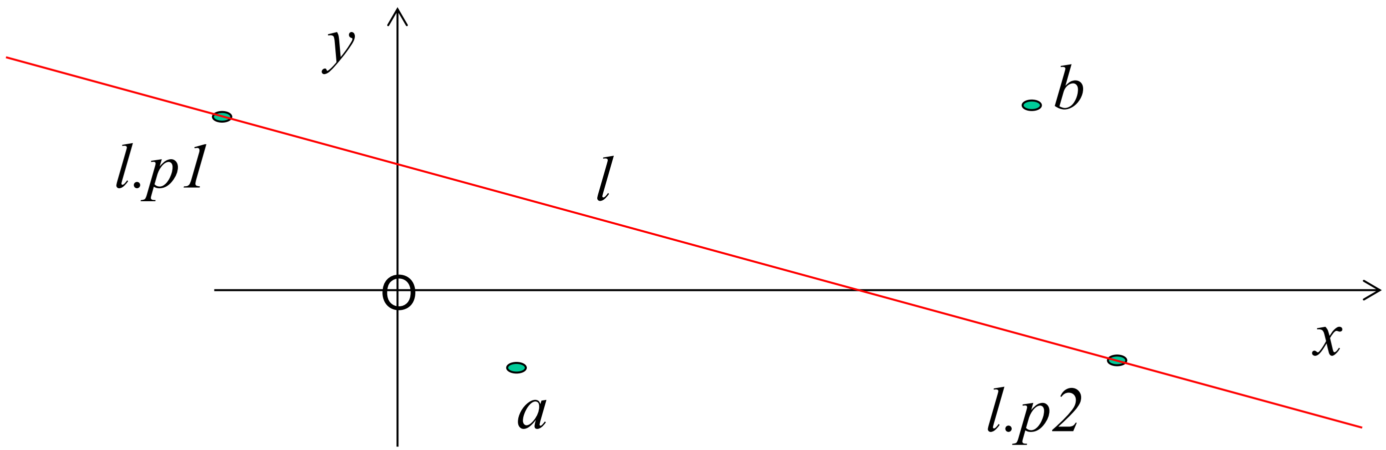
Problem 1: determining whether two line segments p—q and r—s intersect

Split the problem into two subproblems:

1. **Determine whether points p, q lie on opposite sides of the line through points r and s, and whether points r, s lie on opposite sides of the line through points p and q (`onOppositeSides` tests)**

2. **Determine whether the smallest rectangles whose sides are parallel to the x and y axes containing each of p—q and r—s intersect (`boundingBox` test)**

Subproblem 1: `onOppositeSides` test



Determine whether points a and b are on opposite sides of line *l* = —p1—p2—.

Again, note that equation of line *l* is

```
(l.p2.x - l.p1.x)(y - l.p1.y) -
                (l.p2.y - l.p1.y)(x - l.p1.x) = 0 (*)
```

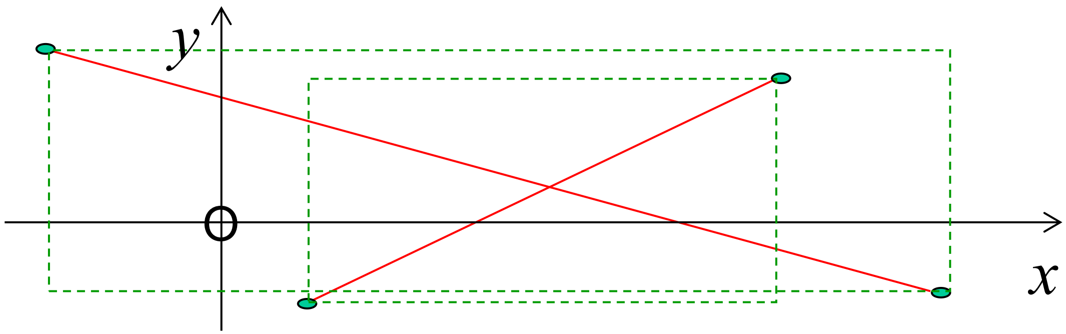
a and b lie on opposite sides of *l* if and only if LHS of (*) has opposite signs when substituted with each of a and b.

Algorithm `onOppositeSides`

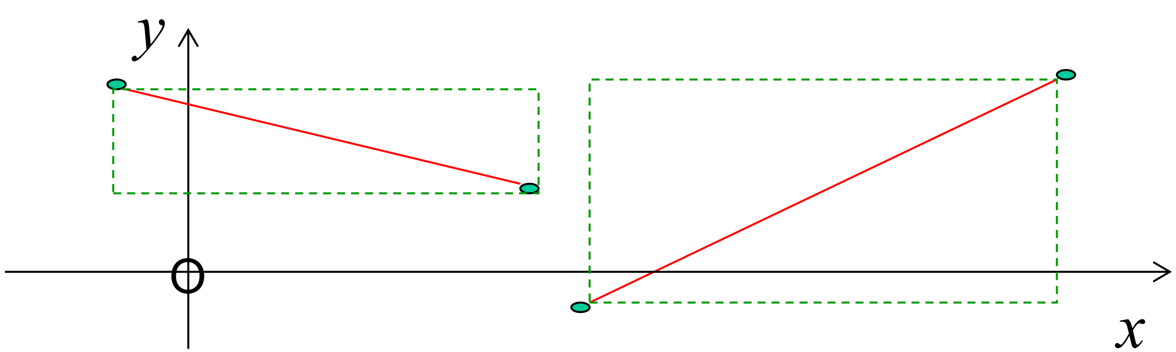
```
/** Input: two points a, b, and a line l = —p1—p2—
  * Output: true if a and b lie on opposite sides of l,
    or if a or b lies on l; false otherwise */

private boolean onOppositeSides
        (Point2D.Double a, Point2D.Double b, Line l) {

  double g, h;

  g = (l.p2.x - l.p1.x) * (a.y - l.p1.y)
                  - (l.p2.y - l.p1.y) * (a.x - l.p1.x);

  h = (l.p2.x - l.p1.x) * (b.y - l.p1.y)
                  - (l.p2.y - l.p1.y) * (b.x - l.p1.x);

  return g * h <= 0.0;

}
```

Subproblem 2: `boundingBox` test

The bounding box of a line segment p—q is the smallest rectangle containing p—q whose sides are parallel to the x and y axes



Example: two line segments together with their bounding boxes



Clearly two line segments cannot intersect if their bounding boxes do not

```
Algorithm boundingBox
Input:  Line segments l1 and l2
Output: true if the rectangles containing l1 and l2
        intersect, and false otherwise
```

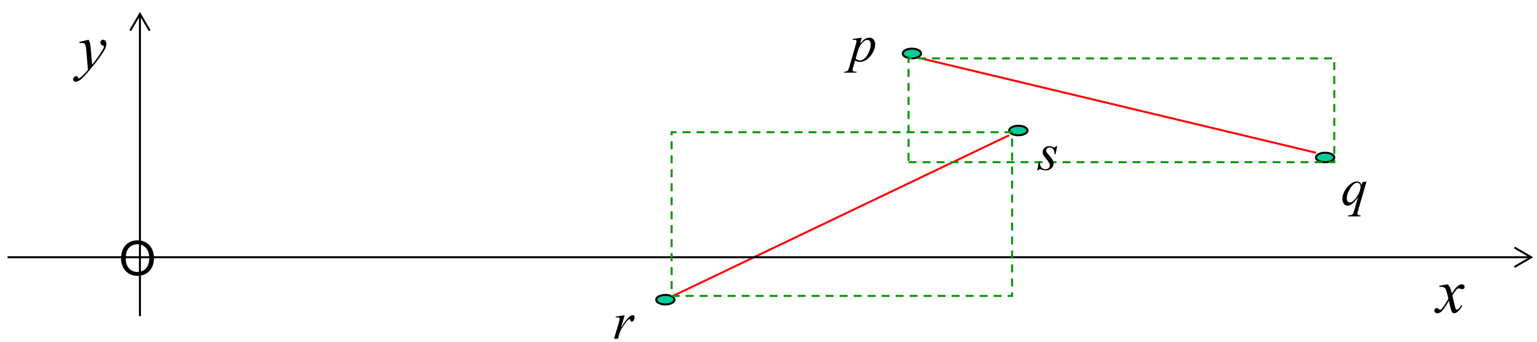
(Implementation is an exercise)

Putting the two tests together

Line segments **p—q** and **r—s** intersect if and only if

```
onOppositeSides(p,q,—r—s—) ∧
        onOppositeSides(r,s,—p—q—) ∧
                boundingBox(p—q,r—s)
```

Example: line segments **p—q** and **r—s** do not intersect



`onOppositeSides(p,q,—r—s—)` returns **true**

`onOppositeSides(r,s,—p—q—)` returns **false**

`boundingBox(p—q,r—s)` returns **true**

Algorithm `intersect`

```
/** Input: two line segments l1 (p—q) and l2 (r—s)
  * Output: returns true if the two line segments have
    a point in common; returns false otherwise */

public boolean intersect (Line l1, Line l2 ) {
   Point2D.Double p = l1.p1;
   Point2D.Double q = l1.p2;
   Point2D.Double r = l2.p1;
   Point2D.Double s = l2.p2;
   return (onOppositeSides(p, q, l2) &&
           onOppositeSides(r, s, l1) &&
           boundingBox(p—q, r—s));
}
```

Clearly O(1) complexity

Exercise: the `onOppositeSides` tests on their own are sufficient to determine whether two line segments intersect, except in one special case – can you find it?