Line Segment Intersections

Benjamin Sach

slides inspired by Marc van Kreveld
Introduction

Problem Given $n$ line segments, find all the intersections...
Problem Given $n$ line segments, find all the intersections...
Introduction

**Problem** Given \( n \) line segments, find all the intersections...
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1,2,\ldots,n$
For $j= 1,2,\ldots,n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output $(i,j)$
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output $(i, j)$
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output ($i, j$)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$

For $j = 1, 2, \ldots, n$

If ($s_i$ intersects $s_j$) and ($i \neq j$)

output $(i, j)$
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1,2,...,n$
For $j = 1,2,...,n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output ($i,j$)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output $(i, j)$
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output $(i, j)$
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output ($i, j$)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let \( s_i \) denote the \( i \)-th line intersection.

For \( i = 1, 2, \ldots, n \)

For \( j = 1, 2, \ldots, n \)

If \( (s_i \text{ intersects } s_j) \) and \( (i \neq j) \)

output \((i, j)\)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j=1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output $(i, j)$
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output ($i, j$)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output ($i, j$)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output ($i, j$)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output ($i, j$)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$

For $j = 1, 2, \ldots, n$

If ($s_i$ intersects $s_j$) and ($i \neq j$)

output $(i, j)$

Given two line segments $s_i$ and $s_j$ described by their end point coordinates
deciding whether (and where) they intersect
takes $O(1)$ time
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If $(s_i$ intersects $s_j$) and ($i \neq j$)
output $(i, j)$

Given two line segments $s_i$ and $s_j$ described by their end point coordinates
deciding whether (and where) they intersect

takes $O(1)$ time

Why?
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let \( s_i \) denote the \( i \)-th line intersection

For \( i = 1, 2, \ldots, n \)
For \( j = 1, 2, \ldots, n \)
If (\( s_i \) intersects \( s_j \)) and (\( i \neq j \))
output (\( i, j \))

Given two line segments \( s_i \) and \( s_j \) described by their end point coordinates deciding whether (and where) they intersect
takes \( O(1) \) time

Why?

Any computation on two objects with \( O(1) \) space descriptions takes \( O(1) \) time
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output $(i, j)$
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, n$
If ($s_i$ intersects $s_j$) and ($i \neq j$)
output ($i, j$)

This algorithm runs in $O(n^2)$ time

(because checking pair of lines takes $O(1)$ time)
A simple algorithm

One simple approach to this problem is to test every pair of line segments...

Let $s_i$ denote the $i$-th line intersection

For $i = 1, 2, \ldots, n$

For $j = 1, 2, \ldots, n$

If ($s_i$ intersects $s_j$) and ($i \neq j$)

output ($i, j$)

This algorithm runs in $O(n^2)$ time

(because checking pair of lines takes $O(1)$ time)

...can we do better?
If there are $n$ line segments... how many intersections can there be?
If there are $n$ line segments... how many intersections can there be?

Here there are 10 line segments and 25 intersections.
If there are \( n \) line segments... how many intersections can there be?

Here there are 50 line segments and 625 intersections.
If there are $n$ line segments... how many intersections can there be?

Here there are 50 line segments and 625 intersections.

In general, there could be $(\frac{n}{2})^2$ intersections.
If there are $n$ line segments... 

how many intersections can there be?

Here there are 50 line segments and 625 intersections.

In general, there could be \((\frac{n}{2})^2\) intersections.

If we want to output all the intersections, 
we can’t possibly expect to do better than \(O(n^2)\) time in the worst-case.
Output sensitive algorithms

The time complexities of the algorithms we have seen so far (in this course) have only depended on the size on the input.
Output sensitive algorithms

The time complexities of the algorithms we have seen so far (in this course) have only depended on the size on the input.

The time complexity of the algorithm we will see in this lecture also depends on the size of the output.
Output sensitive algorithms

The time complexities of the algorithms we have seen so far (in this course) have only depended on the size on the input.

The time complexity of the algorithm we will see in this lecture also depends on the size of the output.

Let $k$ denote the number of line segment intersections ($k$ is not provided in the input).
Output sensitive algorithms

The time complexities of the algorithms we have seen so far (in this course) have only depended on the size on the input.

The time complexity of the algorithm we will see in this lecture also depends on the size of the output.

Let $k$ denote the number of line segment intersections $(k$ is not provided in the input).

Here $n = 17$ and $k = 3$. 
Output sensitive algorithms

The time complexities of the algorithms we have seen so far (in this course) have only depended on the size on the input.

The time complexity of the algorithm we will see in this lecture also depends on the size of the output.

Let $k$ denote the number of line segment intersections ($k$ is not provided in the input).

We will see an algorithm for line segment intersection which takes $O(n \log n + k \log n)$ time in the worst-case.

Here $n = 17$ and $k = 3$. 
Output sensitive algorithms

We will see an algorithm for line segment intersection which takes

\[ O(n \log n + k \log n) \]

time in the worst-case

If \( k \) is small...

For example if \( k \leq 2n \) we get an

\[ O(n \log n) \] worst-case time

If \( k \) is big...

For example if \( k \geq \left( \frac{n}{2} \right)^2 \) we get an

\[ O(n^2 \log n) \]

(worse than the simple algorithm)
Some simplifying restrictions

In the interest of simplicity, we don’t allow the input to contain any of the following:

- Horizontal line segments
- Overlapping line segments
- Two end points with the same \( y \)-coordinate
- Three (or more) lines segments which intersect at the same point

All of these restrictions can be removed making the algorithm slightly more involved (without changing the time complexity)
A first observation
A first observation

$s_i$
A first observation

$S_i$

$y$-span of $S_i$
A first observation

$s_i$

$y$-span of $s_i$

$s_j$
A first observation

$s_i$

$y$-span of $s_i$

$s_j$

$y$-span of $s_j$
A first observation

If \( s_i \) and \( s_j \) don’t have overlapping \( y \)-spans, they don’t intersect.
A first observation

If $s_i$ and $s_j$ don’t have overlapping $y$-spans, they don’t intersect.
A first observation

If $s_i$ and $s_j$ don’t have overlapping $y$-spans, they don’t intersect.

This suggests an overall approach to the problem...
A first observation

If $s_i$ and $s_j$ don’t have overlapping $y$-spans, they don’t intersect.

This suggests an overall approach to the problem…

sweep a horizontal line through the plane from top to bottom finding intersections as we go
A first observation

If \( s_i \) and \( s_j \) don’t have overlapping \( y \)-spans they don’t intersect.

This suggests an overall approach to the problem...

sweep a horizontal line through the plane from top to bottom finding intersections as we go
A first observation

If $s_i$ and $s_j$ don’t have overlapping $y$-spans, they don’t intersect.

This suggests an overall approach to the problem... sweep a horizontal line through the plane from top to bottom finding intersections as we go.
A first observation

If \( s_i \) and \( s_j \) don’t have overlapping \( y \)-spans, they don’t intersect.

This suggests an overall approach to the problem...

sweep a horizontal line through the plane from top to bottom
finding intersections as we go
A first observation

If $s_i$ and $s_j$ don’t have overlapping $y$-spans, they don’t intersect.

This suggests an overall approach to the problem...

sweep a horizontal line through the plane from top to bottom finding intersections as we go
A first observation

If $s_i$ and $s_j$ don't have overlapping $y$-spans, they don't intersect.

This suggests an overall approach to the problem... sweep a horizontal line through the plane from top to bottom finding intersections as we go.
A first observation

If $s_i$ and $s_j$ don’t have overlapping $y$-spans, they don’t intersect.

This suggests an overall approach to the problem...

\textit{sweep a horizontal line through the plane from top to bottom finding intersections as we go}
A first observation

If $s_i$ and $s_j$ don’t have overlapping y-spans, they don’t intersect.

This suggests an overall approach to the problem...

sweep a horizontal line through the plane from top to bottom
finding intersections as we go
A first observation

If $s_i$ and $s_j$ don't have overlapping $y$-spans, they don't intersect.

This suggests an overall approach to the problem...

sweep a horizontal line through the plane from top to bottom finding intersections as we go
A first observation

If $s_i$ and $s_j$ don’t have overlapping $y$-spans, they don’t intersect.

This suggests an overall approach to the problem... sweep a horizontal line through the plane from top to bottom finding intersections as we go.
A first observation

If $s_i$ and $s_j$ don’t have overlapping $y$-spans, they don’t intersect.

This suggests an overall approach to the problem... 

sweep a horizontal line through the plane from top to bottom

finding intersections as we go
Adjacent line segments and a second observation

These two line segments are adjacent at this $y$-coordinate
Adjacent line segments and a second observation

These two line segments are adjacent at this $y$-coordinate

(there is no line segment between them)
Adjacent line segments and a second observation
Adjacent line segments and a second observation

These two line segments
Adjacent line segments and a second observation

These two line segments are adjacent at this $y$-coordinate.
Adjacent line segments and a second observation

These two line segments are *adjacent* at this $y$-coordinate

but they aren’t *adjacent* at this $y$-coordinate
Adjacent line segments and a second observation
Adjacent line segments and a second observation

These two line segments
Adjacent line segments and a second observation

These two line segments never become *adjacent*
Adjacent line segments and a second observation

These two line segments never become *adjacent* so they can’t intersect
Adjacent line segments and a second observation

These two line segments never become *adjacent* so they can't intersect.

Two line segments \(s_i\) and \(s_j\) which are never adjacent don’t intersect.
The overall approach

The overall approach is to

*imagine a horizontal line passing through the plane from the top to the bottom*

*this is called a plane sweep*
The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom; this is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom this is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom; this is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom which is called a plane sweep.

We cannot hope to process the sweep line at every \( y \)-coordinate so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every \( y \)-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom; this is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between *interesting positions* called *event points*.

The event points are the end points of line segments and the *intersection points*.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to *imagine a horizontal line passing through the plane from the top to the bottom*.

**this is called a plane sweep**

We cannot hope to process the sweep line at every \( y \)-coordinate

so the sweep line jumps between *interesting positions* called **event points**

The **event points** are

the end points of line segments

and the *intersection points*

We will have to detect the

*intersection points* on the fly

(before we get to them)
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom; this is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom; this is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every \( y \)-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every \( y \)-coordinate so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points. We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to

*imagine a horizontal line passing through the plane from the top to the bottom*

*this is called a plane sweep*

We cannot hope to process the sweep line at every $y$-coordinate

so the sweep line jumps between *interesting positions*

*called event points*

The event points are

the end points of line segments

and the *intersection points*

We will have to detect the

*intersection points on the fly*

(before we get to them)
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom this is called a plane sweep

We cannot hope to process the sweep line at every $y$-coordinate so the sweep line jumps between interesting positions called event points

The event points are the end points of line segments and the intersection points

We will have to detect the intersection points on the fly (before we get to them)
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every y-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).
The overall approach

The overall approach is to imagine a horizontal line passing through the plane from the top to the bottom. This is called a plane sweep.

We cannot hope to process the sweep line at every $y$-coordinate, so the sweep line jumps between interesting positions called event points.

The event points are the end points of line segments and the intersection points.

We will have to detect the intersection points on the fly (before we get to them).

The number of event points is $O(n + k)$. 
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □).
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □).

this is first

this is last
The status of the sweep line

The *status* of the sweep line

is the set of line segments which currently intersect the sweep line

ordered from left to right by where they intersect

(i.e. in the order given by the □)
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □).

**Fact** the status of the sweep line can only change at event points (i.e. at an end point or an intersection).
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □).

**Fact** the status of the sweep line can only change at event points (i.e. at an end point or an intersection).
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the ■).

**Fact** the status of the sweep line can only change at event points (i.e. at an end point or an intersection)

*the ■ move but the order stays the same*
The status of the sweep line

The *status* of the sweep line

is the set of line segments which currently intersect the sweep line

ordered from left to right by where they intersect

(i.e. in the order given by the ■)

**Fact** the status of the sweep line can only change at event points

(i.e. at an end point or an intersection)

*the ■ move but the order stays the same*
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □).

**Fact** the status of the sweep line can only change at event points (i.e. at an end point or an intersection)

*the □ move but the order stays the same*

We will store the status of the sweep line in a data structure which allows efficient updates (more details later)
The status of the sweep line

The *status* of the sweep line
is the set of line segments which currently intersect the sweep line
ordered from left to right by where they intersect
(i.e. in the order given by the )
The status of the sweep line

The *status* of the sweep line

is the set of line segments which currently intersect the sweep line

ordered from left to right by where they intersect

(i.e. in the order given by the □)

The status of the sweep line tells us which line segments are currently adjacent
The status of the sweep line

The *status* of the sweep line

is the set of line segments which currently intersect the sweep line

ordered from left to right by where they intersect

(i.e. in the order given by the )

Why is this useful?

The status of the sweep line tells us which line segments are currently adjacent
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the orange ■).

The status of the sweep line tells us which line segments are currently adjacent.

**Why is this useful?**

Two line segments which *never* become adjacent cannot intersect.
The status of the sweep line

The status of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □).

Why is this useful?

The status of the sweep line tells us which line segments are currently adjacent.

Why is this useful?

two line segments which never become adjacent cannot intersect.

We will detect each upcoming intersection when the corresponding line segments first become adjacent.
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □).

Why is this useful?

The status of the sweep line tells us which line segments are currently adjacent.

*Why is this useful?*

Two line segments which *never* become adjacent cannot intersect.

We will detect each upcoming intersection when the corresponding line segments first become adjacent.
The status of the sweep line

The *status* of the sweep line is the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the ■).

Why is this useful?

The status of the sweep line tells us which line segments are currently adjacent.

*Why is this useful?*

Two line segments which *never* become adjacent cannot intersect.

We will detect each upcoming intersection when the corresponding line segments first become adjacent.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the top of a line segment, we insert it into the status data structure (at the appropriate place).
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the top of a line segment, we insert it into the status data structure (at the appropriate place).
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the top of a line segment, we insert it into the status data structure (at the appropriate place).

We then check whether this segment will intersect either of the adjacent segments.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the top of a line segment, we insert it into the status data structure (at the appropriate place).

We then check whether this segment will intersect either of the adjacent segments.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the top of a line segment, we insert it into the status data structure (at the appropriate place).

We then check whether this segment will intersect either of the adjacent segments.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the top of a line segment, we insert it into the status data structure (at the appropriate place).

We then check whether this segment will intersect either of the adjacent segments.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the top of a line segment, we insert it into the status data structure (at the appropriate place).

We then check whether this segment will intersect either of the adjacent segments.

*If either pair will intersect, we’ve found a new event point.*
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the top of a line segment, we insert it into the status data structure (at the appropriate place).

We then check whether this segment will intersect either of the adjacent segments. If either pair will intersect, we’ve found a new event point.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the bottom of a line segment, we delete it from the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the bottom of a line segment, we delete it from the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the bottom of a line segment, we delete it from the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the bottom of a line segment, we delete it from the status data structure.

This makes a pair of segments adjacent, so we check whether they will intersect.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the bottom of a line segment, we delete it from the status data structure.

This makes a pair of segments adjacent, so we check whether they will intersect.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the bottom of a line segment, we delete it from the status data structure.

This makes a pair of segments adjacent, so we check whether they will intersect.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the bottom of a line segment, we delete it from the status data structure.

This makes a pair of segments adjacent, so we check whether they will intersect.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is the bottom of a line segment, we delete it from the status data structure.

This makes a pair of segments adjacent, so we check whether they will intersect.

If they will intersect, we’ve found a new event point.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is an intersection point, we swap the two line segments in the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is an **intersection point**, we **swap** the two line segments in the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is an intersection point, we swap the two line segments in the status data structure.

This gives two new pairs of adjacent segments to check.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is an intersection point, we swap the two line segments in the status data structure.

This gives two new pairs of adjacent segments to check.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is an intersection point, we swap the two line segments in the status data structure.

This gives two new pairs of adjacent segments to check.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is an intersection point, we swap the two line segments in the status data structure.

This gives two new pairs of adjacent segments to check.

*If either pair will intersect, we’ve found a new event point.*
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is:

- the top of a line segment, we **insert** it into the status data structure.
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure

If the event point is:

- the top of a line segment, we insert it into the status data structure
- the bottom of a line segment, we delete it from the status data structure
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is:

- the top of a line segment, we **insert** it into the status data structure
- the bottom of a line segment, we **delete** it from the status data structure
- an intersection point, we **swap** the two line segments in the status data structure
Updating the sweep line

Every time the sweep line moves to the next event point, we update the status data structure.

If the event point is:
- the top of a line segment, we **insert** it into the status data structure
- the bottom of a line segment, we **delete** it from the status data structure
- an intersection point, we **swap** the two line segments in the status data structure

*and we always check whether we have discovered any new event points (specifically intersections)*
How do we keep track of the event points?

At the start of the algorithm, we are aware of $2n$ event points, one for each end of each line segment.
How do we keep track of the event points?

At the start of the algorithm, we are aware of $2n$ event points, one for each end of each line segment. The sweep line starts at the top event point (the one with the highest $y$ value).
How do we keep track of the event points?

At the start of the algorithm, we are aware of $2n$ event points, one for each end of each line segment.

The sweep line starts at the top event point (the one with the highest $y$ value).

After processing an event point, the sweep line moves down to the next event point.
How do we keep track of the event points?

At the start of the algorithm, we are aware of $2n$ event points, one for each end of each line segment.

The sweep line starts at the top event point (the one with the highest $y$ value).

After processing an event point, the sweep line moves down to the next event point.

However, in processing an event point, the algorithm may discover new event points (specifically intersections).
How do we keep track of the event points?

At the start of the algorithm, we are aware of $2n$ event points, one for each end of each line segment.

The sweep line starts at the top event point (the one with the highest $y$ value).

After processing an event point, the sweep line moves down to the next event point.

However, in processing an event point, the algorithm may discover new event points (specifically intersections).

We keep track of the event points using a Priority Queue.
How do we keep track of the event points?

At the start of the algorithm, we are aware of $2n$ event points, one for each end of each line segment.

The sweep line starts at the top event point (the one with the highest $y$ value).

After processing an event point, the sweep line moves down to the next event point.

However, in processing an event point, the algorithm may discover new event points (specifically intersections).

We keep track of the event points using a Priority Queue.

Every event point is INSERTED as it is discovered (with its $y$ value as the key).
How do we keep track of the event points?

At the start of the algorithm, we are aware of $2n$ event points, one for each end of each line segment.

The sweep line starts at the top event point (the one with the highest $y$ value).

After processing an event point, the sweep line moves down to the next event point.

However, in processing an event point, the algorithm may discover new event points (specifically intersections).

We keep track of the event points using a Priority Queue.

Every event point is INSERTED as it is discovered (with its $y$ value as the key).

We can then use DELETEMN to recover the next event point.
Can we miss out on an intersection?

If \( s_i \) and \( s_j \) intersect

they must become adjacent at some \( y \)-coordinate

(before they intersect)
Can we miss out on an intersection?

If $s_i$ and $s_j$ intersect

they must become adjacent at some $y$-coordinate

*(before they intersect)*

In particular, they must become adjacent at some *event point*

with a higher $y$-coordinate
Can we miss out on an intersection?

If $s_i$ and $s_j$ intersect

they must become adjacent at some $y$-coordinate

(before they intersect)

In particular, they must become adjacent at some event point

with a higher $y$-coordinate
Can we miss out on an intersection?

If \( s_i \) and \( s_j \) intersect

they must become adjacent at some \( y \)-coordinate

(before they intersect)

In particular, they must become adjacent at some \textit{event point}

with a higher \( y \)-coordinate

This is because the status of the sweep line doesn’t change between event points
Can we miss out on an intersection?

If $s_i$ and $s_j$ intersect
they must become adjacent at some $y$-coordinate
(before they intersect)

In particular, they must become adjacent at some event point
with a higher $y$-coordinate

This is because the status of the sweep line doesn’t change between event points

The formal proof then follows by induction
Can we find the same event point twice?

Consider the line segments shown...
Can we find the same event point twice?

Consider the line segments shown...
Can we find the same event point twice?

Consider the line segments shown...

when we process this event point,
Can we find the same event point twice?

Consider the line segments shown... when we process this event point,

we discover this event point
Can we find the same event point twice?

Consider the line segments shown...

when we process this event point,

we rediscover this event point
Can we find the same event point twice?

Consider the line segments shown…

when we process this event point,

we rediscover this event point again
Can we find the same event point twice?

Consider the line segments shown... we rediscover this event point again and again.
Can we find the same event point twice?
Can we find the same event point twice?

That is, we *can* discover the same event point more than once.
Can we find the same event point twice?

That is, we *can* discover the same event point more than once.

There are (at least) two ways to deal with this:
Can we find the same event point twice?

That is, we *can* discover the same event point more than once.

There are (at least) two ways to deal with this:

1. Check whether we already found the new point - by looking it up in the priority queue.
Can we find the same event point twice?

That is, we can discover the same event point more than once.

There are (at least) two ways to deal with this:

1. Check whether we already found the new point - by looking it up in the priority queue

2. Don’t worry about it (INSERT it anyway) but make sure that when we process an event point, we only process it once - by checking that the priority queue didn’t return the same event point as last time.
Can we find the same event point twice?

That is, we *can* discover the same event point more than once.

There are (at least) two ways to deal with this:

1. Check whether we already found the new point
   - by looking it up in the priority queue

2. Don’t worry about it (INSERT it anyway)
   but make sure that when we process an event point, we only process it once
   - by checking that the priority queue didn’t return the same event point as last time

*either approach gives the same time complexity*
How do we implement the status data structure?

We need a data structure to store the status of the sweep line

i.e. the set of line segments which currently intersect the sweep line
ordered from left to right by where they intersect
(i.e. in the order given by the □ )
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line

i.e. the set of line segments which currently intersect the sweep line
ordered from left to right by where they intersect
(i.e. in the order given by the □)

The operations we need are

**INSERT, DELETE, FIND and PREDECESSOR/SUCCESSOR**
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line

i.e. the set of line segments which currently intersect the sweep line
ordered from left to right by where they intersect
(i.e. in the order given by the ■)

The operations we need are

**INSERT, DELETE, FIND** and **PREDECESSOR/SUCCESSOR**
How do we implement the status data structure?

We need a data structure to store the status of the sweep line

- i.e. the set of line segments which currently intersect the sweep line
- ordered from left to right by where they intersect

(i.e. in the order given by the ■)

The operations we need are

**INSERT, DELETE, FIND and PREDECESSOR/SUCCESSOR**

we also need **SWAP** but this can be done using

**INSERT** and **DELETE**
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line

i.e. the set of line segments which currently intersect the sweep line
ordered from left to right by where they intersect
(i.e. in the order given by the ■)

The operations we need are

**INSERT, DELETE, FIND** and **PREDECESSOR/SUCCESSOR**
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line

i.e. the set of line segments which currently intersect the sweep line

ordered from left to right by where they intersect

(i.e. in the order given by the □)

The operations we need are

**INSERT, DELETE, FIND** and **Predecessor/Successor**

We will use a self-balancing tree (e.g. a 2-3-4 tree) as the status data structure
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line

i.e. the set of line segments which currently intersect the sweep line

ordered from left to right by where they intersect

(i.e. in the order given by the □)

The operations we need are

*Insert, Delete, Find* and *Predecessor/Successor*

We will use a self-balancing tree (e.g. a 2-3-4 tree) as the status data structure

which supports these operations in $O(\log n)$ time
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line

i.e. the set of line segments which currently intersect the sweep line
ordered from left to right by where they intersect
(i.e. in the order given by the ■)

The operations we need are

**INSERT, DELETE, FIND and PREDECESSOR/SUCCESSOR**

We will use a self-balancing tree (e.g. a 2-3-4 tree) as the status data structure
which supports these operations in \( O(\log n) \) time

*however*, we need to be careful about the *keys*
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line

  i.e. the set of line segments which currently intersect the sweep line
  ordered from left to right by where they intersect
  (i.e. in the order given by the ■ )
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line

i.e. the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □)

We insert each line segment $s_i$ into the self-balancing search tree using the description of $s_i$ (i.e. its end points) as the key
How do we implement the status data structure?

We need a data structure to store the status of the sweep line. i.e. the set of line segments which currently intersect the sweep line ordered from left to right by where they intersect (i.e. in the order given by the □)

We insert each line segment $s_i$ into the self-balancing search tree using the description of $s_i$ (i.e. its end points) as the key.

This may seem odd as we normally think of a key as being an integer.
How do we implement the status data structure?

We need a data structure to store the *status* of the sweep line
i.e. the set of line segments which currently intersect the sweep line
ordered from left to right by where they intersect
(i.e. in the order given by the ■)

We insert each line segment \( s_i \) into the self-balancing search tree using
the description of \( s_i \) (i.e. its end points) as the key

This may seem odd as we normally think of a key as being an integer

Actually, all we require is that the keys have an order
and we can compare two keys in \( O(1) \) time
Time Complexity (sketch)

The algorithm moves the sweep line $O(n + k)$ times,

once for each event point
Time Complexity (sketch)

The algorithm moves the sweep line $O(n + k)$ times,
   once for each event point

If the status data structure and priority queue structures
   are implemented so that their operations take $O(\log n)$ time
   *(e.g. with a self-balancing tree and a binary heap, respectively)*
Time Complexity (sketch)

The algorithm moves the sweep line $O(n + k)$ times, once for each event point.

If the status data structure and priority queue structures are implemented so that their operations take $O(\log n)$ time

(e.g. with a self-balancing tree and a binary heap, respectively)

The overall complexity then becomes $O(n \log n + k \log n)$ as claimed.
The algorithm moves the sweep line $O(n + k)$ times, once for each event point.

If the status data structure and priority queue structures are implemented so that their operations take $O(\log n)$ time (e.g. with a self-balancing tree and a binary heap, respectively),

the overall complexity then becomes $O(n \log n + k \log n)$ as claimed.

This is because we do a $O(n + k)$ operations on each data structure while moving the sweep line.
Summary

We have seen an algorithm for line segment intersection which runs in $O(n \log n + k \log n)$ time

where $n$ is the number of line segments and $k$ is the number of intersections.

The approach is to move a horizontal line through the plane which jumps between all the interesting positions.

The efficiency relies on using a Self-balancing search tree and a Binary Heap.

We put quite a few restrictions on the input, fixing these is fiddly but not difficult.

In the original paper, they suggest adding random noise to the points to avoid the restrictions.
Dealing with the restrictions

In the interest of simplicity, we didn’t allow the input to contain any of the following:

- Horizontal line segments
- Overlapping line segments
  (merge these then postprocess)
- Two end points with the same y-coordinate
  (split ties using the x-coordinate)
- Three (or more) lines segments which intersect at the same point
  (swap becomes reverse)

All of these restrictions can be removed making the algorithm slightly more involved
(hints are given for the interested)