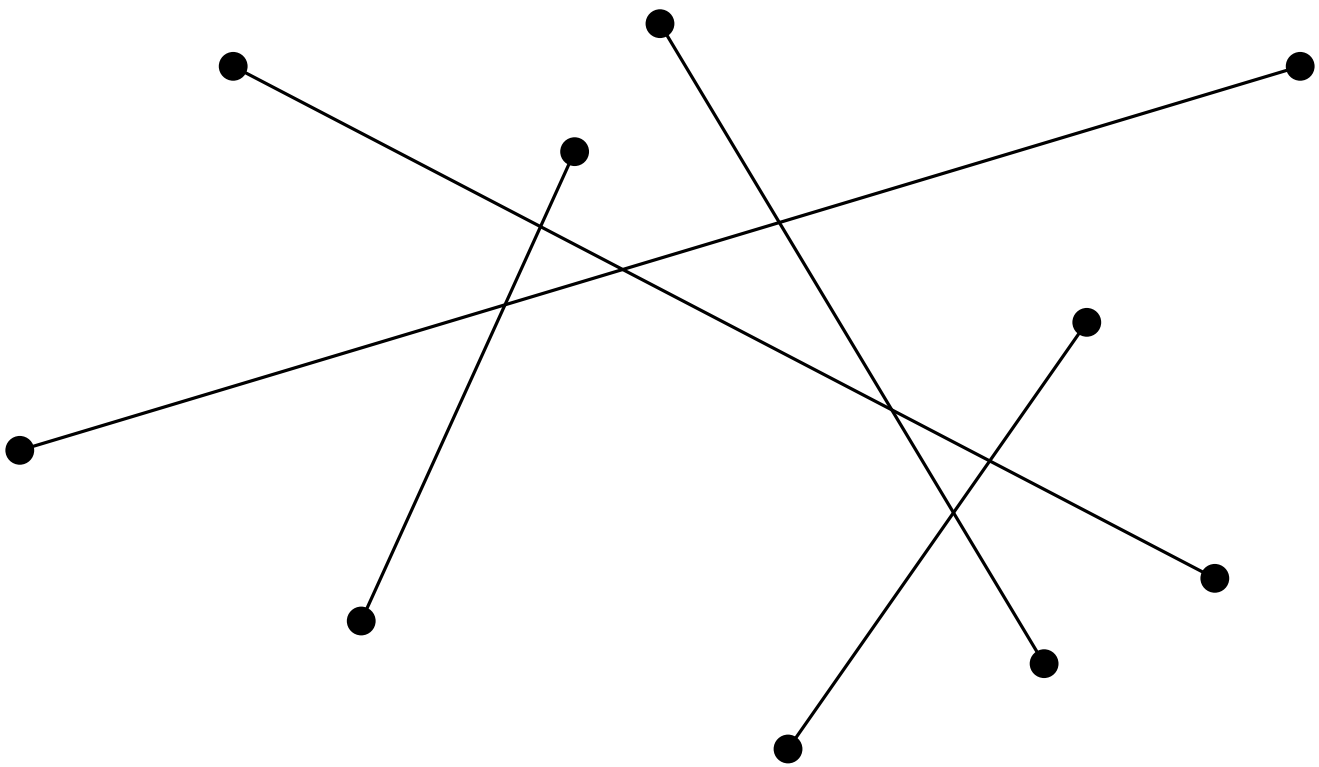
Intersection of Line Segments

Data Structures & Algorithms 2

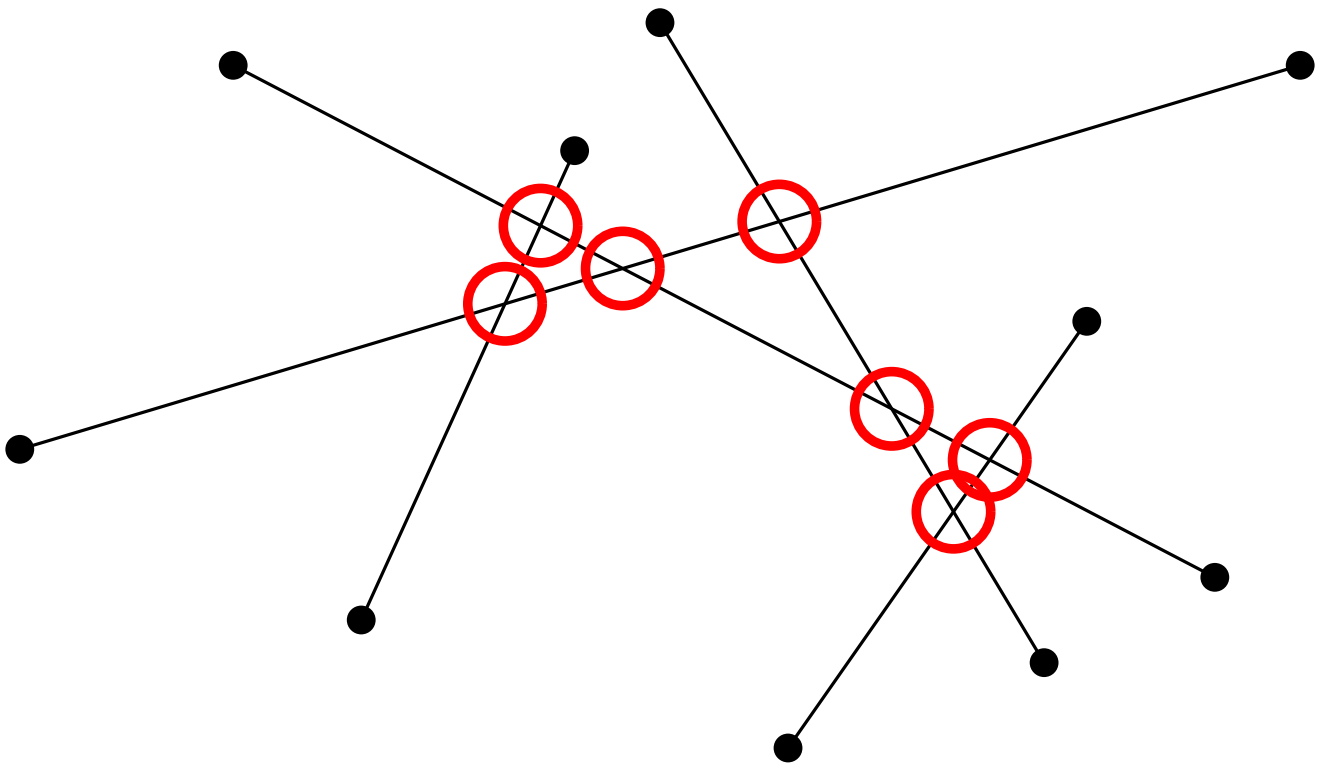
Definition

Given n line segments in the plane, find all of their intersections.



Definition

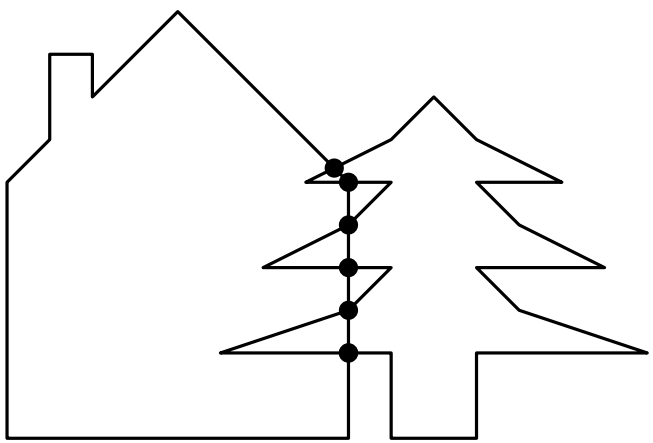
Given n line segments in the plane, find all of their intersections.

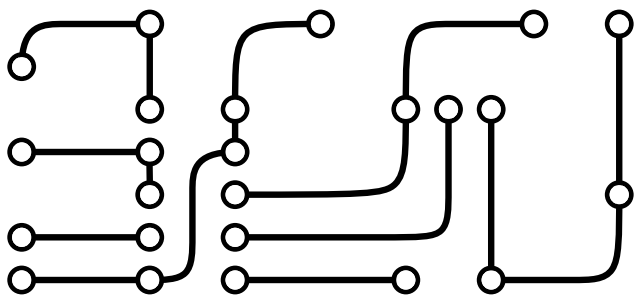


Definition

Given n line segments in the plane, find all of their intersections.

Applications:





- Hidden-line algorithm
- Printed circuit boards (test for shortcuts)
- Implicitly in geometric algorithms

Naive Approach

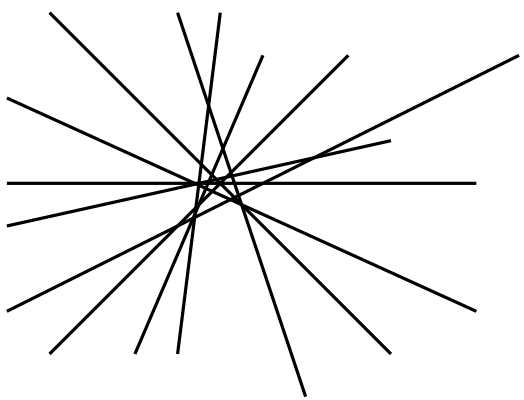
Observations:

- **Basic operation:** check a pair of segments for intersection; can be done in constant time

Naive Approach

Observations:

- **Basic operation:** check a pair of segments for intersection; can be done in constant time
- There exist up to $\Theta(n^2)$ intersections:



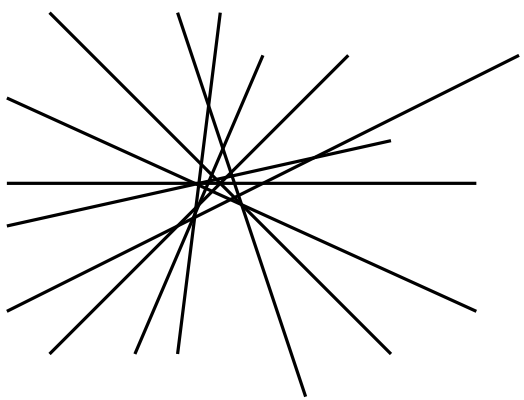
$\binom{n}{2} = \Theta(n^2)$ intersections

Thus in the worst case every algorithm needs $\Omega(n^2)$ time just to report the intersections.

Naive Approach

Observations:

- **Basic operation:** check a pair of segments for intersection; can be done in constant time
- There exist up to $\Theta(n^2)$ intersections:



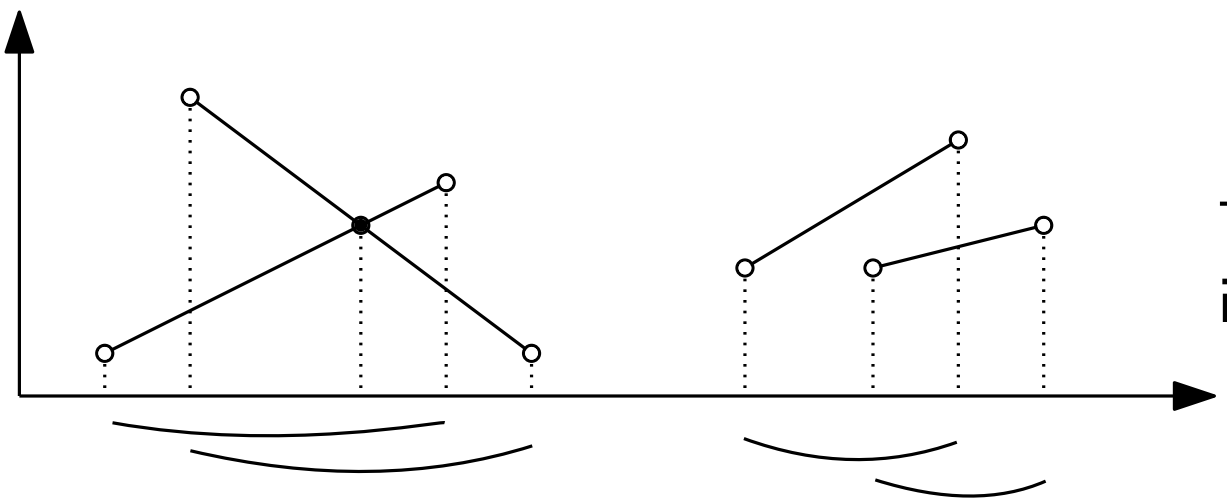
$\binom{n}{2} = \Theta(n^2)$ intersections

Thus in the worst case every algorithm needs $\Omega(n^2)$ time just to report the intersections.

- Runtime $T(n, k)$ should depend on k, the number of intersections! $\Rightarrow$ *output-sensitive* algorithm.

Plane-Sweep Idea

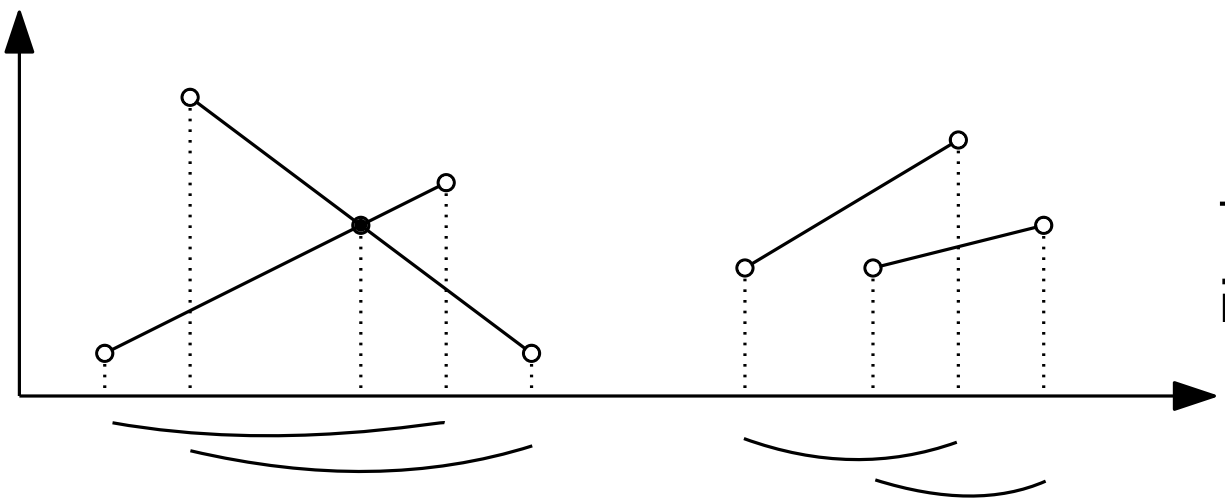
This algorithm uses the following simple observation: If two segments intersect, then their x-intervals overlap.



The inverse statement is not necessarily true

Plane-Sweep Idea

This algorithm uses the following simple observation: If two segments intersect, then their x-intervals overlap.

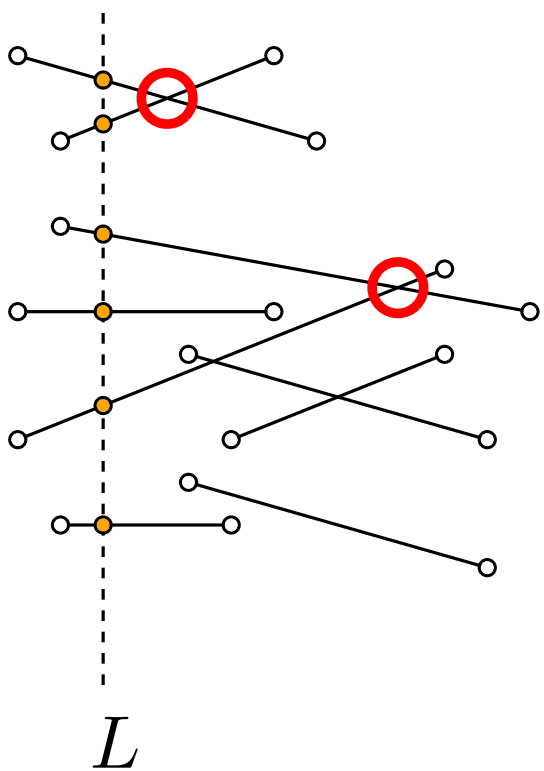


The inverse statement is not necessarily true

Idea: Scan from left to right through all x-values with a vertical line L. At every point we consider only those segments which are hit by L and check for intersections.

Plane-Sweep Algorithm

- Every position of L gives a y-order of the intersected segments.
- If two segments intersect, then they are neighbored on L at some point before (with respect to the x-order) the intersection.



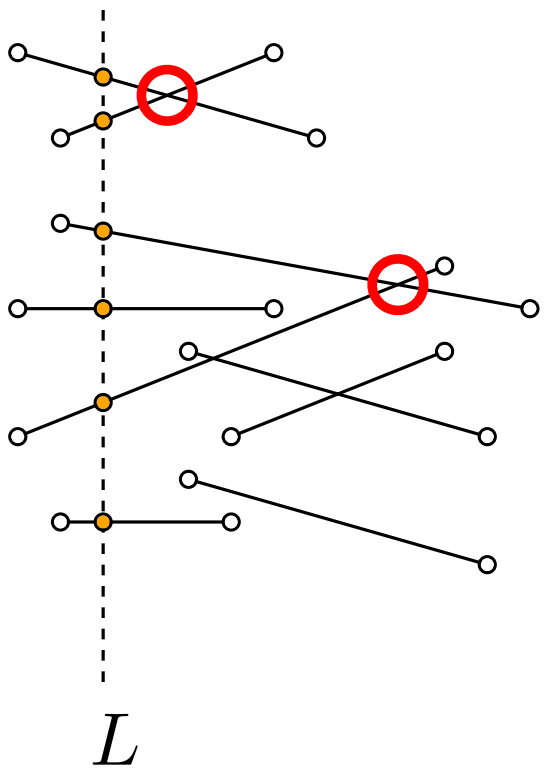
Plane-Sweep Algorithm

- Every position of L gives a y-order of the intersected segments.
- If two segments intersect, then they are neighbored on L at some point before (with respect to the x-order) the intersection.

Algorithm:

- Maintain y-order on L.
- Check y-neighbored segments for intersections

The plane-sweep is *event-based*, where an event is a change in the y-order.

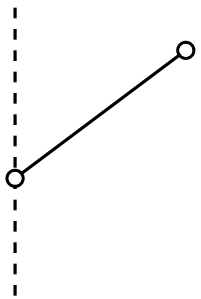


Sweep Events

There are three different kinds of events:

Sweep Events

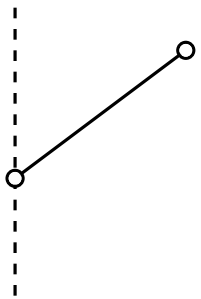
There are three different kinds of events:



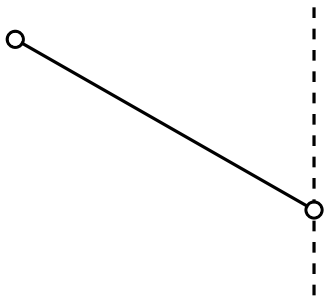
Left endpoint:
insert segment

Sweep Events

There are three different kinds of events:



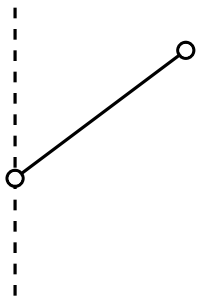
Left endpoint:
insert segment



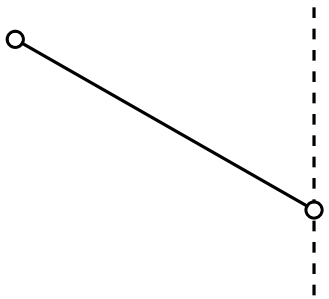
Right endpoint:
remove segment

Sweep Events

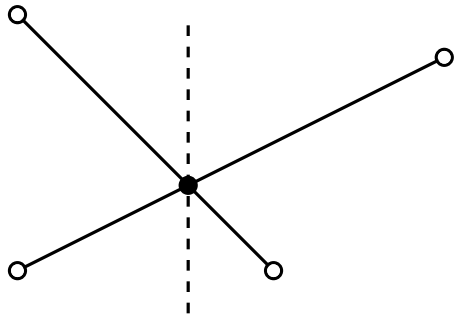
There are three different kinds of events:



Left endpoint:
insert segment



Right endpoint:
remove segment



Intersection:
switch y-neighbored
segments

Implementation - Data Structures

Data structur **X**: Contains x-coordinates of the currently known events, that have not yet been reached by L (start- and endpoints, known intersections).
Operations: Inserting, remove x-minimum $\Rightarrow$ queue, use for example heap as data structure

Data structure **Y**: (ordering on L) Contains y-ordered set of segments that intersect L.
Operations: Inserting (startpoints), removing (endpoints), switching of neighbors (intersection) $\Rightarrow$ dictionary, use for example a (2-4)-tree as data structure

Implementation - Pseudocode

$X = \varnothing, Y = \varnothing$

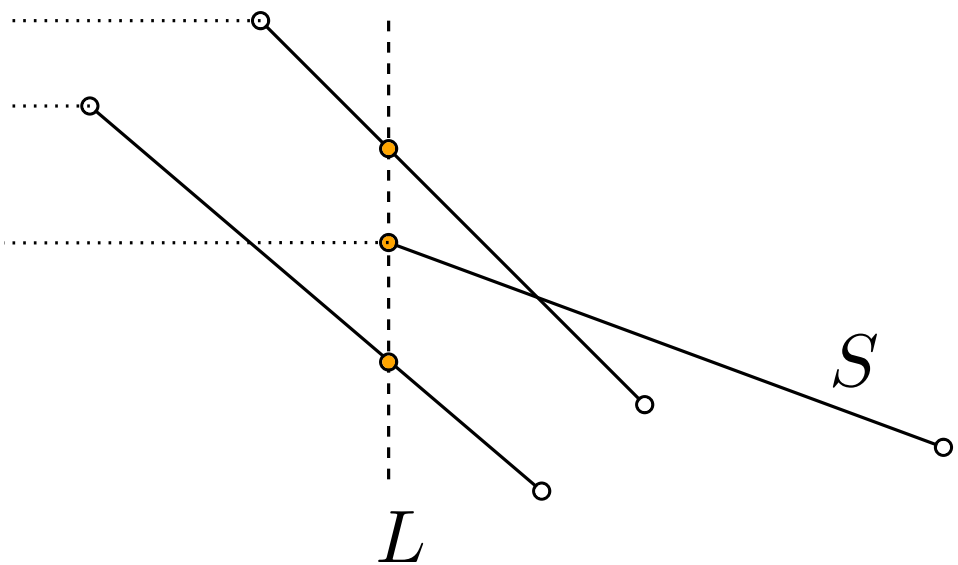
Insert x-coordinates of the start- and endpoints of all segments into X.

while $X \neq \varnothing$:

1. Get minimum m of X and remove it from X.
2. IF m left endpoint THEN insert its segment into Y
   ELSE IF m right endpoint THEN remove its segment from Y
   ELSE (m intersection) switch the order of the intersecting segments in Y
3. FOR all new neighboring pairs in Y (at most two):
   IF neighboring pair intersects in p AND p is to the right of L
   THEN report p and insert x-coordinate of p into X

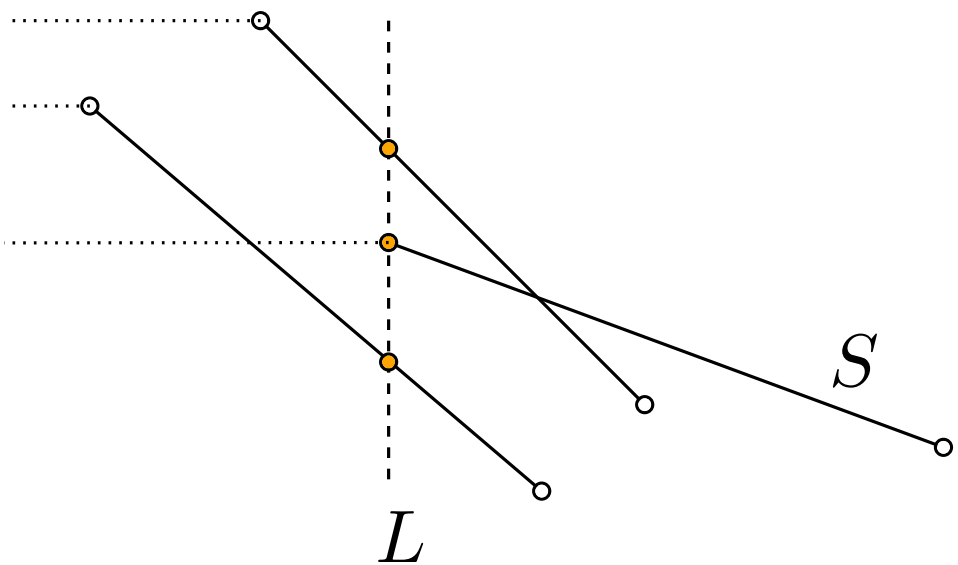
Implementation - Details

Details: When inserting a segment into the search-tree Y, we compare the y-coordinates of the segments intersected with L. We compute them for the current position of L on-line:



Implementation - Details

Details: When inserting a segment into the search-tree Y, we compare the y-coordinates of the segments intersected with L. We compute them for the current position of L on-line:



Neighbors in Y are easier to find, if the values in the search tree are 'linked' in order. This can be done by linking the leaves of the (2-4)-tree by pointers.

Analysis

n segments, k intersections, $0 \leq k \leq \binom{n}{2} = \Theta(n^2)$

- **In X**: Per segment we insert two events, per intersection one. We later remove all of these events.

  $\Rightarrow O(n+k)$ space, and
  $O((n+k)\log(n+k)) = O((n+k)\log n)$ time

Analysis

n segments, k intersections, $0 \leq k \leq \binom{n}{2} = \Theta(n^2)$

- **In X**: Per segment we insert two events, per intersection one. We later remove all of these events.

  $\Rightarrow O(n+k)$ space, and
  $O((n+k)\log(n+k)) = O((n+k)\log n)$ time

- **In Y**: We insert and remove every segment exactly once. For every intersection we switch a pair of segments. $O(1)$ per switch, if we link intersections to their segments (linked leaves in the 2-4-tree).

  $\Rightarrow O(n)$ space and $O(n \log n + k)$ time

Analysis

n segments, k intersections, $0 \leq k \leq \binom{n}{2} = \Theta(n^2)$

- **In X**: Per segment we insert two events, per intersection one. We later remove all of these events.

  $\Rightarrow O(n+k)$ space, and
  $O((n+k)\log(n+k)) = O((n+k)\log n)$ time

- **In Y**: We insert and remove every segment exactly once. For every intersection we switch a pair of segments. $O(1)$ per switch, if we link intersections to their segments (linked leaves in the 2-4-tree).

  $\Rightarrow O(n)$ space and $O(n \log n + k)$ time

  In total: $O((n+k)\log n)$ **time and** $O(n+k)$ **space.**

Remarks

- Detecting an intersection multiple times does not impact the analysis - charge it to the events.

Remarks

- Detecting an intersection multiple times does not impact the analysis - charge it to the events.
- If we insert for every segment only the first not yet reached intersection into X, the algorithm uses only $O(n)$ space with the same running time.

Remarks

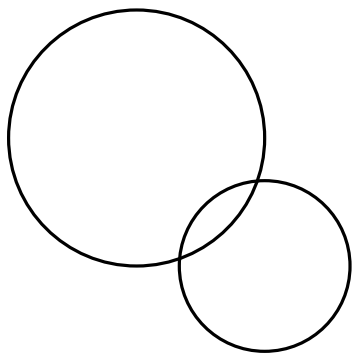
- Detecting an intersection multiple times does not impact the analysis - charge it to the events.
- If we insert for every segment only the first not yet reached intersection into X, the algorithm uses only $O(n)$ space with the same running time.
- Time can be reduced to $O(n \log n + k)$ (with $O(n)$ space), [Balaban, 1995].

Remarks

- Detecting an intersection multiple times does not impact the analysis - charge it to the events.
- If we insert for every segment only the first not yet reached intersection into X, the algorithm uses only $O(n)$ space with the same running time.
- Time can be reduced to $O(n \log n + k)$ (with $O(n)$ space), [Balaban, 1995].
- The algorithm works as intersection-**detector** in time $O(n \log n)$ and optimal space $O(n)$ (set $k = 0$ or $k = 1$).

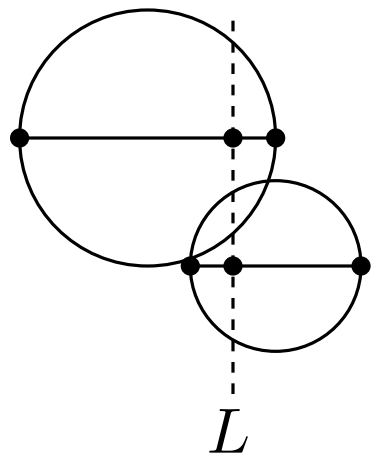
Remarks

- Intersection-detector for discs is very similar:



Remarks

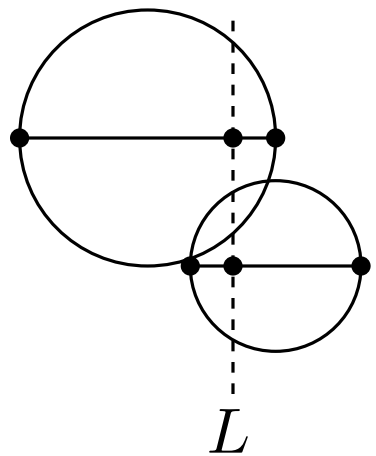
- Intersection-detector for discs is very similar:



- Check neighbored horizontal diameters of the disks.

Remarks

- Intersection-detector for discs is very similar:



- Check neighbored horizontal diameters of the disks.
- Only as detector for intersections useful, since not all intersections will be reported.
  If an intersection exists, there exist two discs where the horizontal diameters are at some point neighbored on L.