Computational Geometry

Sweep Line Algorithms

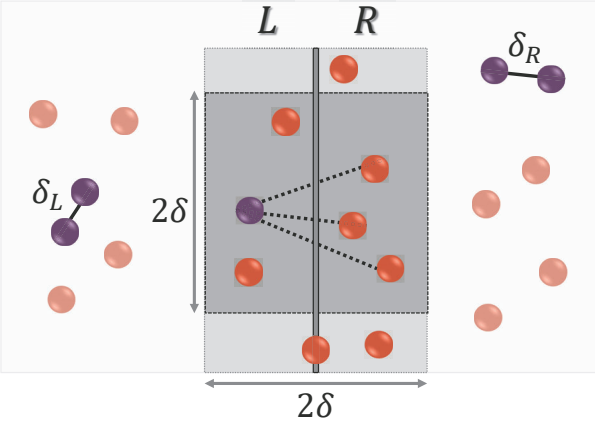
CS 336: Design and Analysis of Algorithms

Closest Points in the Plane

Given: Given a set of points (x_i, y_i).
Goal: Find the pair of closest points.

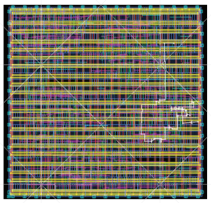
1. Divide points into two groups L and R.
2. Find closest pairs in L and R:
   a. $[p_L, q_L, \delta_L]$ = Closest_Pair (L);
   b. $[p_R, q_R, \delta_R]$ = Closest_Pair (R);
3. Combine solutions.

Find closest points in 2δ-strip



- Find the closest points in 2δ-strip on the opposite sides of the cut.
- Compare each vertex in the δ-strip in L with every vertex in the the δ-strip R.

VLSI Design



Modern processors:

- Billions of transistors.
- Built on 10-32nm technology.
- Many layers. Each layer is 2 dimensional.

Many computational problems. E.g., verifying that different elements are separated by the necessary distance. Running time must be at most $O(n \log n)$.

How to find intersections?

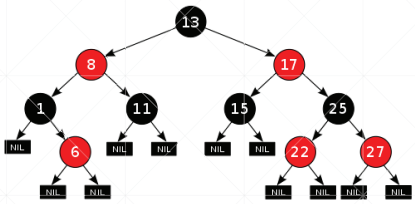
- Question 1:
  - Given a collection of circles of radius r find whether any of them intersect.

How to find intersections?

- Question 1:
  - Given a collection of circles of radius r find whether any of them intersect.
  - What if circles have different radii?

Red–black Trees

- Search: O(log n)
- Insert: O(log n)
- Delete: O(log n)



- What elements can we store in a red-black tree?
  - Can I create RedBlackTree<MyClass>?

Red-black Tree

```
template < class Key,                    // map::key_type
           class T,                      // map::mapped_type
           class Compare = less<Key>,    // map::key_compare
           class Alloc = allocator<pair<const Key,T> >>
class map;
```

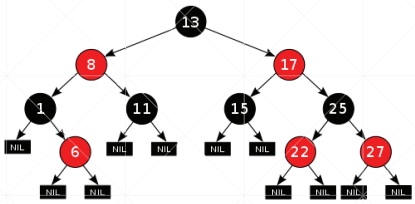
Red-black Tree

```
template < class Key,                    // map::key_type
           class T,                      // map::mapped_type
           class Compare = less<Key>,    // map::key_compare
           class Alloc = allocator<pair<const Key,T> >>
class map;
```

- Red-black trees store elements from ***totally ordered*** sets.
  a. If $a \leq b$ and $b \leq a$, then $a = b$.
  b. If $a \leq b$ and $b \leq c$, then $a \leq c$ (transitivity).
  c. $a \leq b$ or $b \leq a$ (totality).

Red–black Trees

- Search: O(log n)
- Insert: O(log n)
- Delete: O(log n)



- Red-black trees maintain ordered sets.
- For any x in the tree, we can find elements ajacent with x (according to the ordering): Left/Below(x) and Right/Above(x).

How to check whether intervals on the real line intersect?

- **Define the ordering:** $[x_i', x_i''] < [x_j', x_j'']$ if $x_i'' < x_j'$.
- Is it a total ordering?
  a. $[x_i', x_i''] \leq [x_j', x_j'']$ and $[x_j', x_j''] \leq [x_i', x_i'']$, then
  $$x_j'' \leq x_i' \leq x_i'' \leq x_j'$$
  b. $[x_i', x_i''] < [x_j', x_j'']$ and $[x_j', x_j''] < [x_k', x_k'']$, then
  $$x_i'' < x_j' \leq x_j'' < x_k'$$

How to check whether intervals on the real line intersect?

- **Define the ordering:** $[x_i', x_i''] < [x_j', x_j'']$ if $x_i'' < x_j'$.
- Is it a total ordering?
  c. if $[x_i', x_i''] \not\leq [x_j', x_j'']$ and $[x_j', x_j''] \not\leq [x_k', x_k'']$, then the intervals $[x_i', x_i'']$ and $[x_j', x_j'']$ intersect! (if and only if!)

Algorithm for the Real Line

- Insert intervals one by one in the red-black tree.
- After each insertion check that the inserted interval does not intersect with its *left* and *right* neighbors .

Claim: If the algorithm doesn't find any violations, then intervals do not intersect.

Algorithm for the Real Line

Claim: If the algorithm doesn't find any violations, then intervals do not intersect.

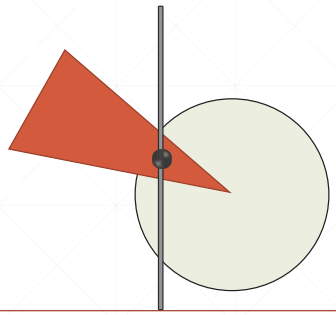
Proof: All intervals are ordered in the tree. Hence,

$$[x_1', x_1''] < [x_2', x_2''] < \cdots < [x_n', x_n'']$$

and, consequently, $[x_i', x_i''] < [x_j', x_j'']$ for $i < j$.

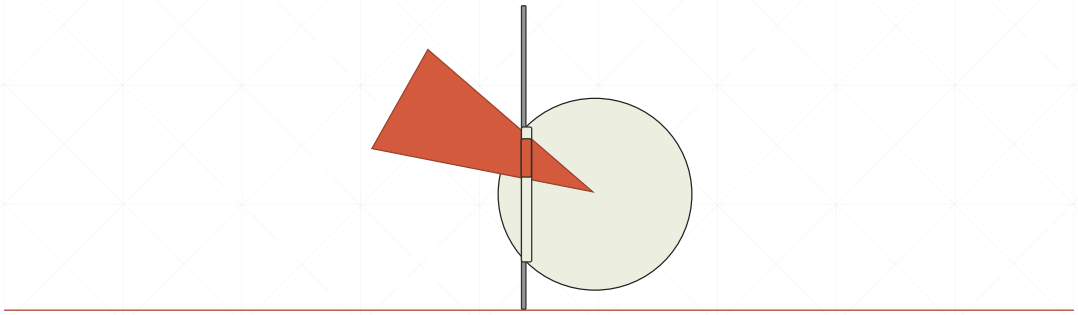
Algorithm for Objects in 2D

- Two geometric objects ***A*** and ***B*** intersect if there is a line parallel to the y-axis that the sections of ***A*** and ***B*** intersect.



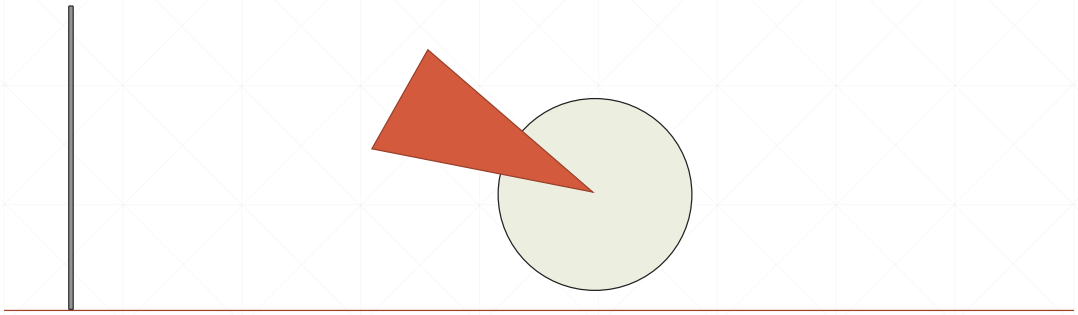
Algorithm for Objects in 2D

- Two geometric objects ***A*** and ***B*** intersect if there is a line parallel to the y-axis that the sections of ***A*** and ***B*** intersect.



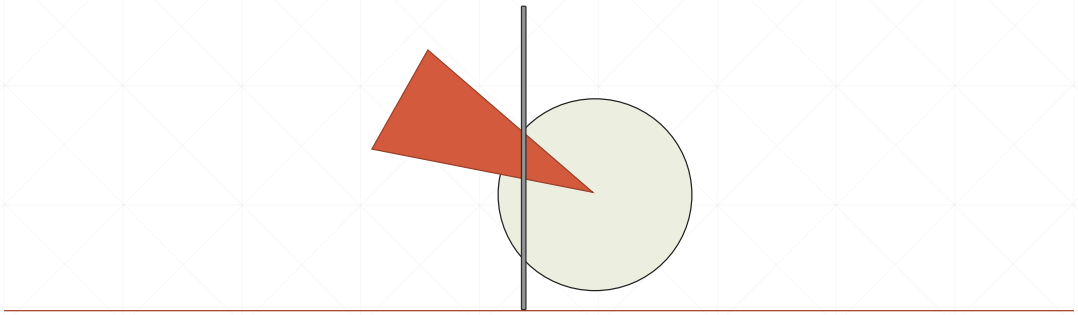
Sweep Line

- Two geometric objects ***A*** and ***B*** intersect if there is a line parallel to the y-axis that the sections of ***A*** and ***B*** intersect.



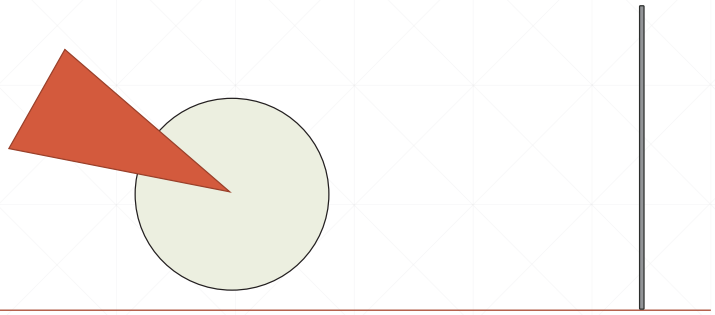
Sweep Line

- Two geometric objects ***A*** and ***B*** intersect if there is a line parallel to the y-axis that the sections of ***A*** and ***B*** intersect.



Sweep Line

- Two geometric objects ***A*** and ***B*** intersect if there is a line parallel to the y-axis that the sections of ***A*** and ***B*** intersect.

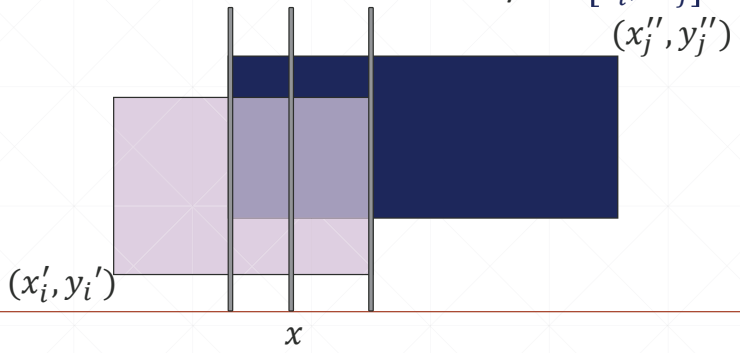


How to implement the sweep line algorithm?

- **Challenge:** The number of possible lines is infinite.
- **Solution:** Consider lines only for important x – "event points".
- For axis-parallel rectangles, event points are the x-coordinates of the left and right edges.

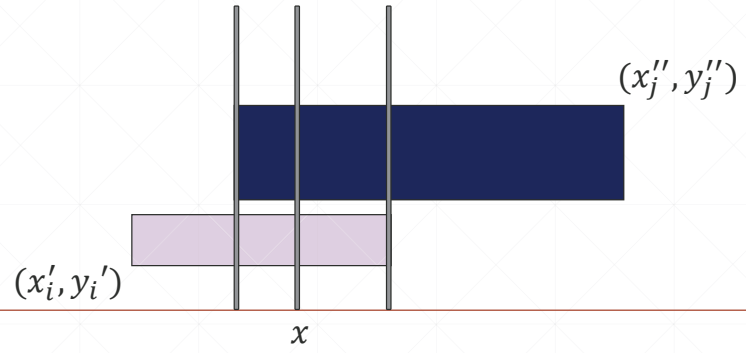
Intersection of Axis-parallel Rectangles

- If rectangles $((x_i', y_i') - (x_i'', y_i''))$ and $\left((x_j', y_j') - (x_j'', y_j'')\right)$ intersect their sections intersect for every $x \in [x_i', x''_j]$.



Intersection of Axis-parallel Rectangles

- If the section of A is less than the section of B for some x, then this inequality holds for all $x \in [x_j'', x_i']$.



Algorithm

- Create a list E of all event points:
  - For each rectangle i, add x_i' and x_i'' to the list.
  - Sort E from left to right.
- Create a red-black tree T.
- For each event point x in E:
  - Add all rectangles that "start" at x to T (i.e. $x_i' = x$).
  - After each insertion, check that the inserted rectangle doesn't intersect with its *Above* and *Below* neighbors.
  - Remove all rectangles that "end" at x from T (i.e. $x_i'' = x$).

In the tree each rectangle is represented by $[y_i', y_i'']$.

Loop Invariant 1

Claim: For every x before the algorithm terminates, tree T contains all *active* rectangles at time x.

- Rectangle i is active at "time" x, if $x_i' \leq x < x_i''$.

Why? Because we add a rectangle whenever it becomes active and remove it once it gets inactive.

Loop Invariant 2

For every x, the order of active rectangles in the tree is correct.

It is sufficient to check that

$$R > Below(R) \text{ and } R < Above(R)$$

Then, by transitivity, R is "greater" than any rectangle below and less than any rectangle "above".

The order of any two active rectangles does not change over time. So, we only need to verify that the inequality above holds when we insert R in the tree.

*Rectangle i is active at "time" x, if $x \in [x_i' \leq x < x_i'']$.

Proof of Correctness

- If no two rectangles intersect, then the algorithm will not find any intersecting rectangles.
- If two rectangles intersect, we pick a pair of intersecting rectangles $i < j$ with the smallest j. There are no intersections before x_j'. At $x = x_j'$, we add rectangle j and as argued earlier find the intersection.

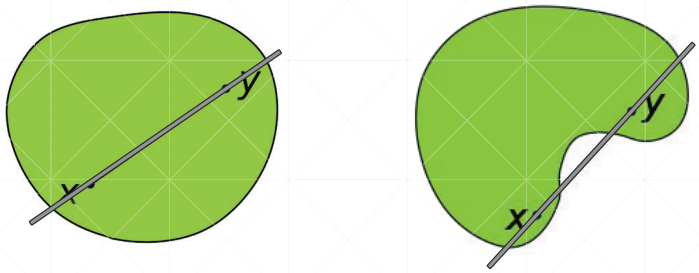
Running Time

- Creating a list of all events: $O(n)$
- Sorting all events: $O(n \log n)$
- Insert and remove operations on T: $O(n \log n)$

Total running time: $O(n \log n)$

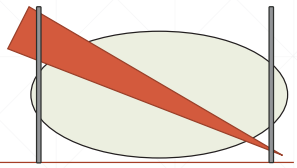
How about other Convex Objects

- If A is a convex set, then it intersects with any line by an interval, point or empty set.



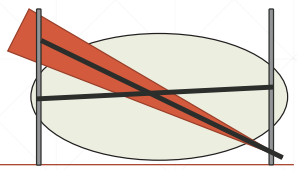
How about other Convex Objects

- If A is a convex set, then it intersects with any line by an interval, point or empty set.
- Let's define $A < B$ at time x^*, if for the line $x = x^*$ the section of A is less than the section of B.
- Can $A < B$ for some x and $B < A$ for others?

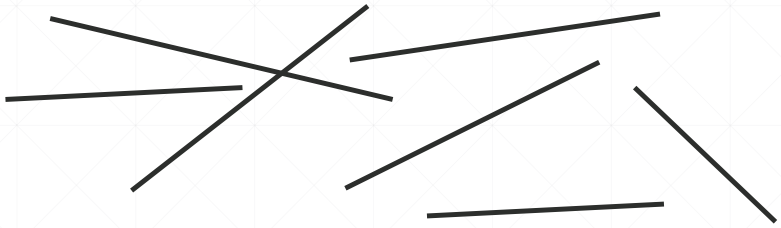


How about other Convex Objects

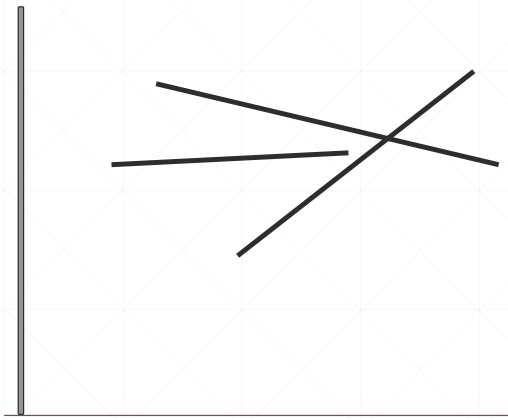
- If A is a convex set, then it intersects with any line by an interval, point or empty set.
- Let's define $A < B$ at time x^*, if for the line $x = x^*$ the section of A is less than the section of B.
- Can $A < B$ for some x and $B < A$ for others? Yes.
- Should A and B in this case intersect? Yes.



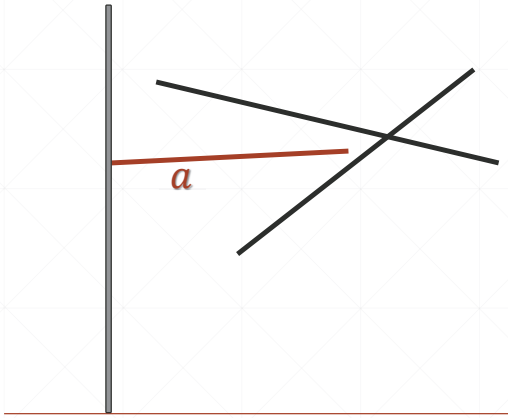
Algorithm for Intervals?



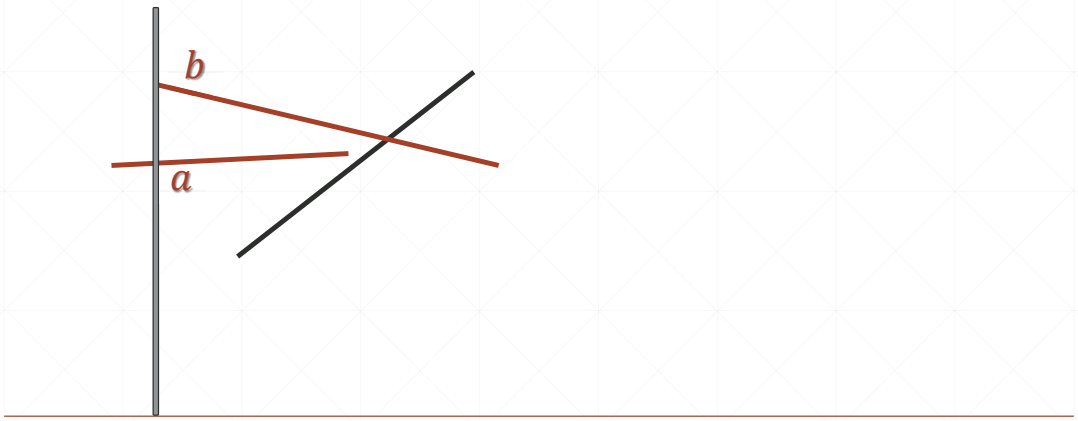
Algorithm for Intervals?



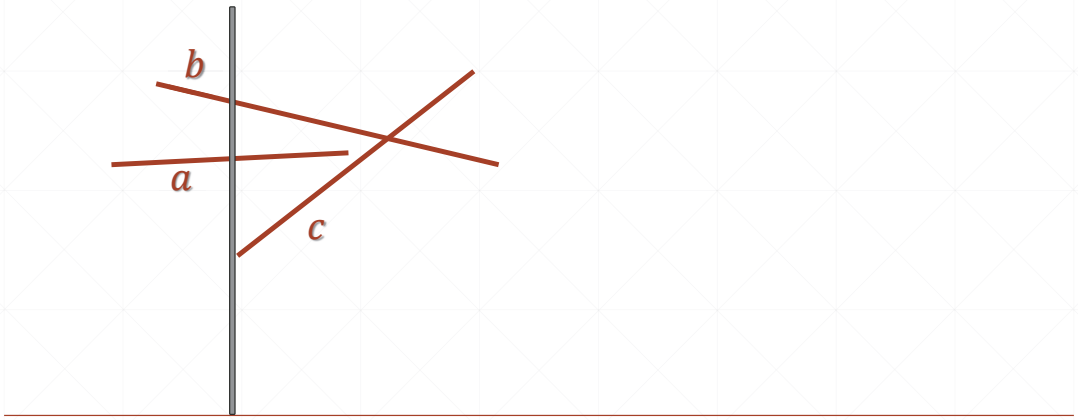
Algorithm for Intervals?



Algorithm for Intervals?



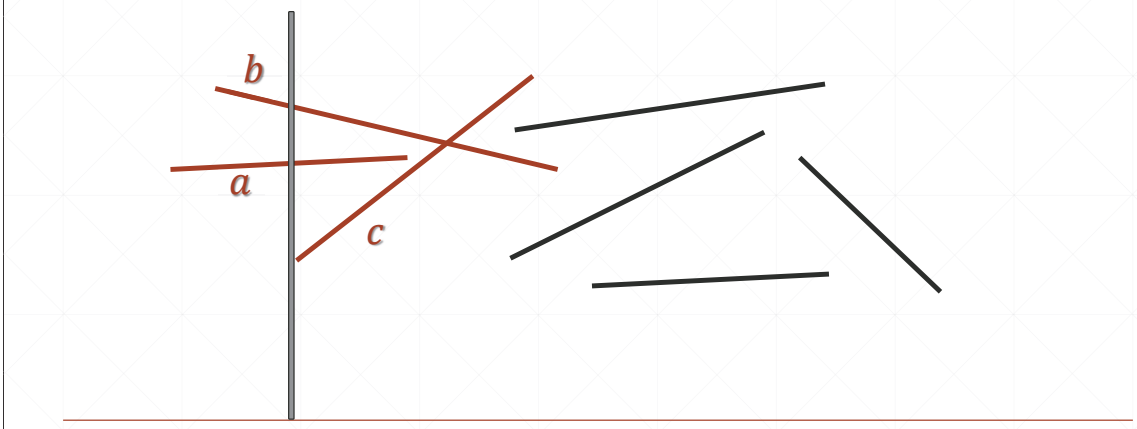
Algorithm for Intervals?



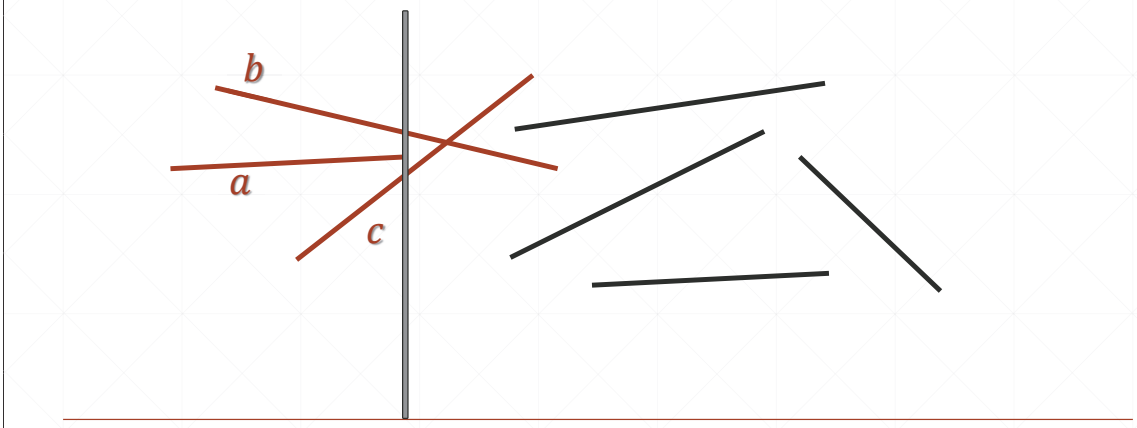
Algorithm

- Create a list E of all event points:
- Create a red-black tree T.
- For each event point x in E:
  - Insert all intervals that "start" at x to T (i.e. $x_i' = x$).
  - After each insertion, check that the inserted interval doesn't intersect with its *Above* and *Below* neighbors.
  - Remove all intervals that "end" at x from T (i.e. $x_i'' = x$).
  - After each deletion, check that the *Above* and *Below* neighbors of the deleted rectangle do not intersect.

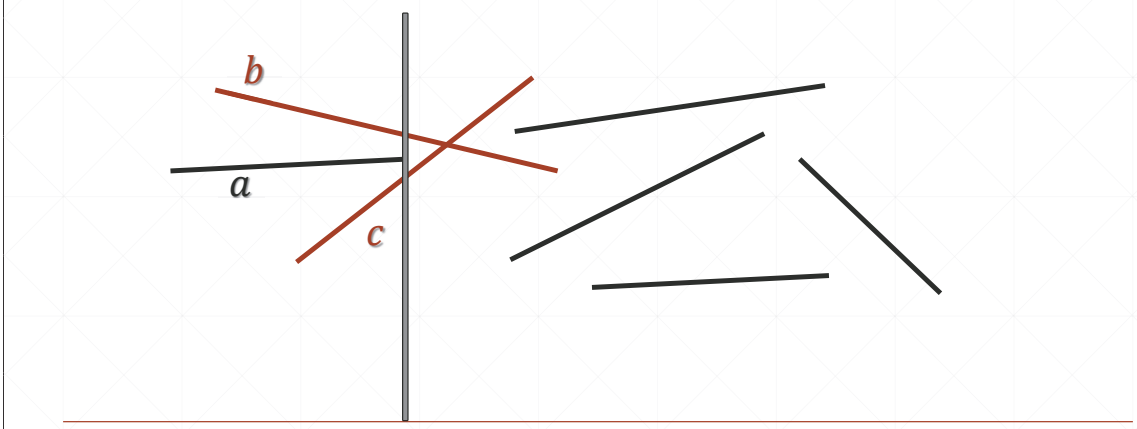
Algorithm for Intervals?



Algorithm for Intervals?



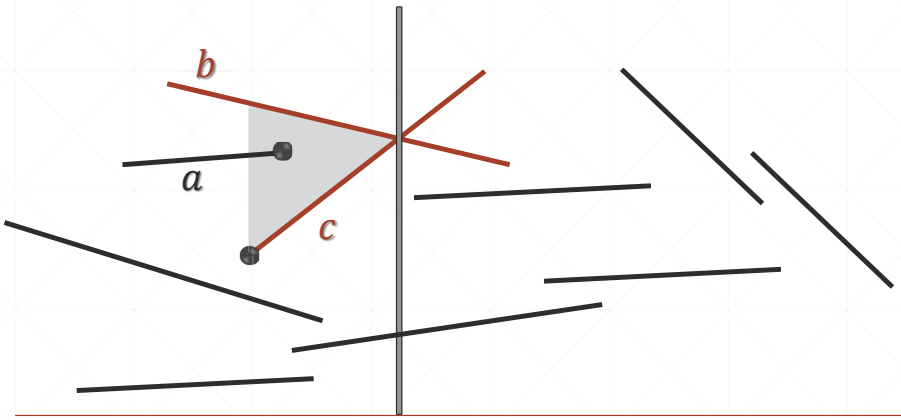
Algorithm for Intervals?



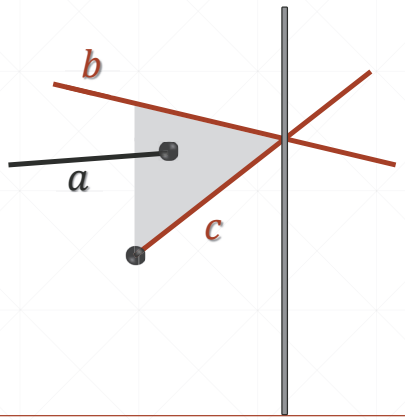
Loop Invariant

- For every* x before the algorithm terminates, the order of intervals in T is correct.
- At time x_i', when we insert interval i in T, we know that i is comparable to all other intervals j and its order in the tree is correct.
- The order between i and j can get wrong only after intervals i and j intersect.
- Let us look at the left most intersection!

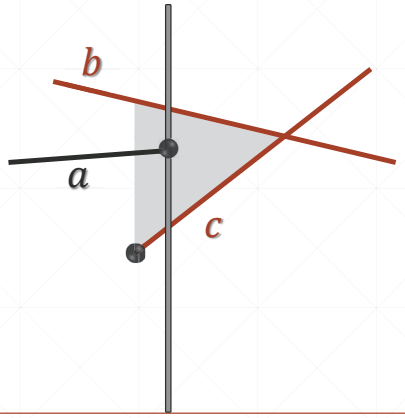
Last Event before the Intersection



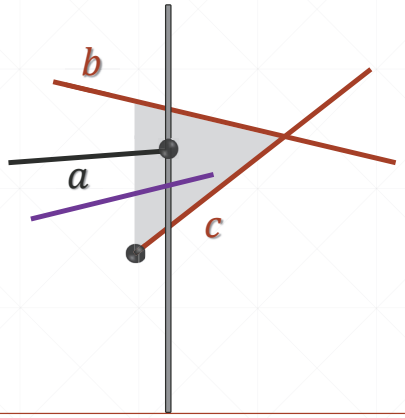
Last Event before the intersection



Last Event before the intersection

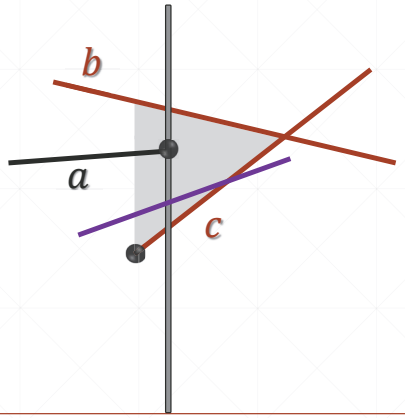


Last Event before the intersection



- Purple interval needs to leave the gray area.

Last Event before the intersection



- Purple interval needs to leave the gray area.
- Then there is another intersection to the left of ours.

Loop Invariant

- For every* x before the algorithm terminates, the order of intervals in T is correct.
- At time x_i', when we insert interval i in T, we know that i is comparable to all other intervals j and its order in the tree is correct.
- The order between i and j can get wrong only after intervals i and j intersect. But we will find that interaction at the previous event-point in the “gray area”.

Other Convex Objects?