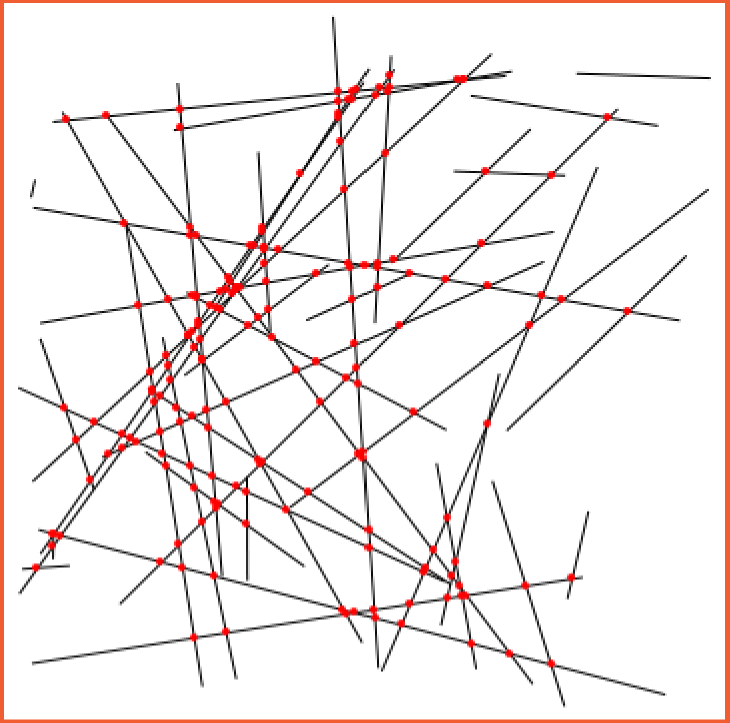
Lecture 5:
Sweepline technique
(and computational geometry)



THE UNIVERSITY OF SYDNEY

General techniques in this course

- Greedy algorithms [Lecture 3]
- Divide & Conquer algorithms [Lecture 4]
- Sweepline algorithms [today]
- Dynamic programming algorithms [4 and 11 Sep]
- Network flow algorithms [18 Sep and 9 oct]

Depth of interval
Segment Intersection
Convex Hull using Sweepline
Closest Pair using Sweepline
Visibility

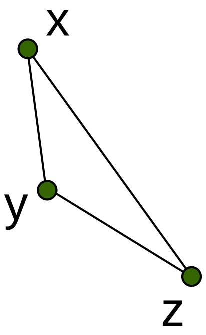
What is computational geometry?

The study of algorithms to solve problems stated in terms of geometry.

The problems we study in this lecture are defined in a metric space!

For every two points x and y in the metric space, there is a function $g(x,y) \geq 0$ which gives the distance between them as a nonnegative real number. A metric space must also satisfy

1. $g(x,y) = 0$ iff $x = y$,
2. $g(x,y) = g(y,x)$, and
3. the triangle inequality must hold $g(x,y) + g(y,z) \geq g(x,z)$.

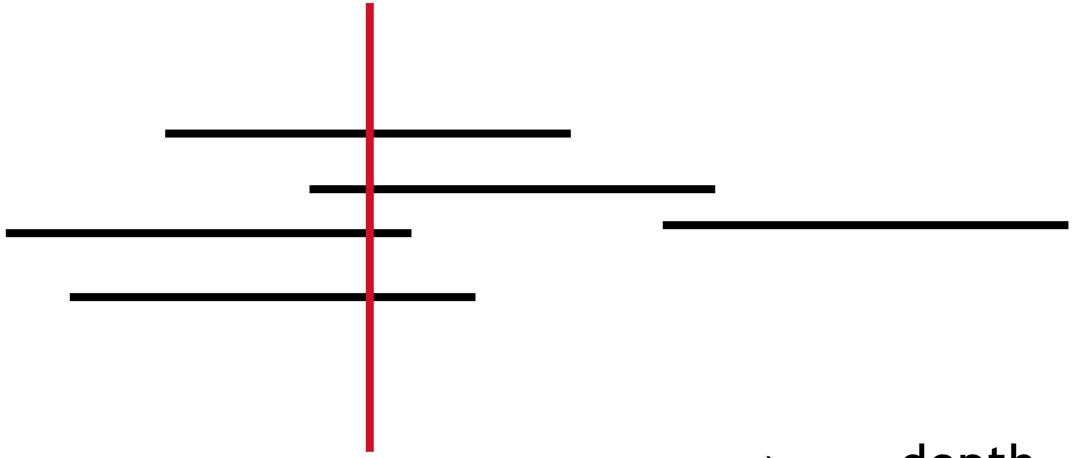


We will consider the Euclidean metric (L_2-metric).

Depth of interval

Given a set S of n intervals (in 1D) compute the depth of S.

The depth of S is the maximum number of intervals passing over any point.



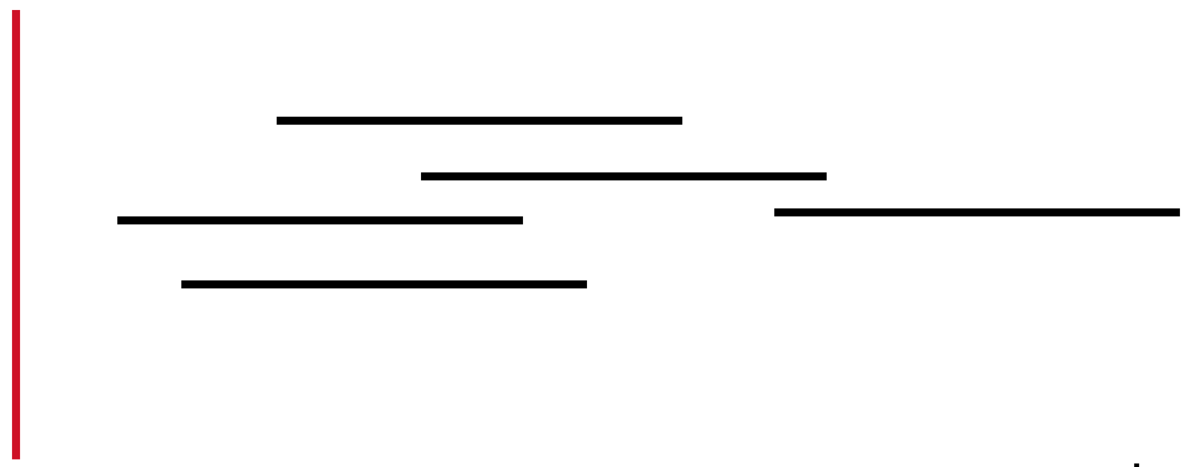
$\Rightarrow$ depth=4

Compute the depth using sweep-line, done in O(nlog n).

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine “sweeping” a vertical line from left to right while maintaining the current depth.

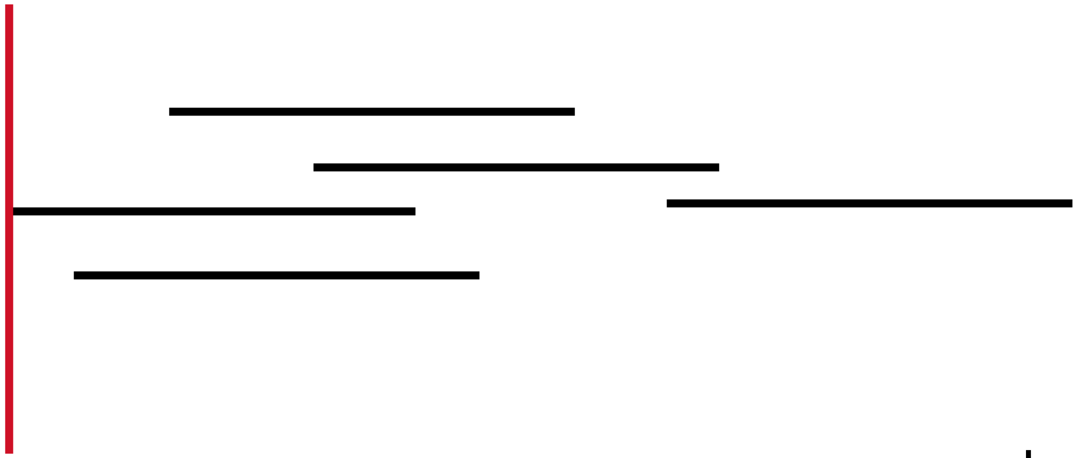


depth = 1

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine “sweeping” a vertical line from left to right while maintaining the current depth.

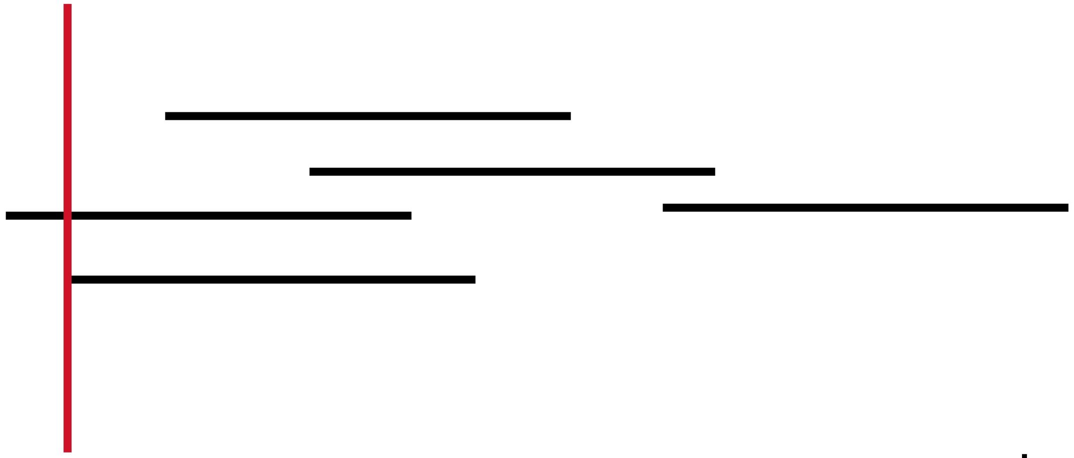


depth = 2

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine “sweeping” a vertical line from left to right while maintaining the current depth.

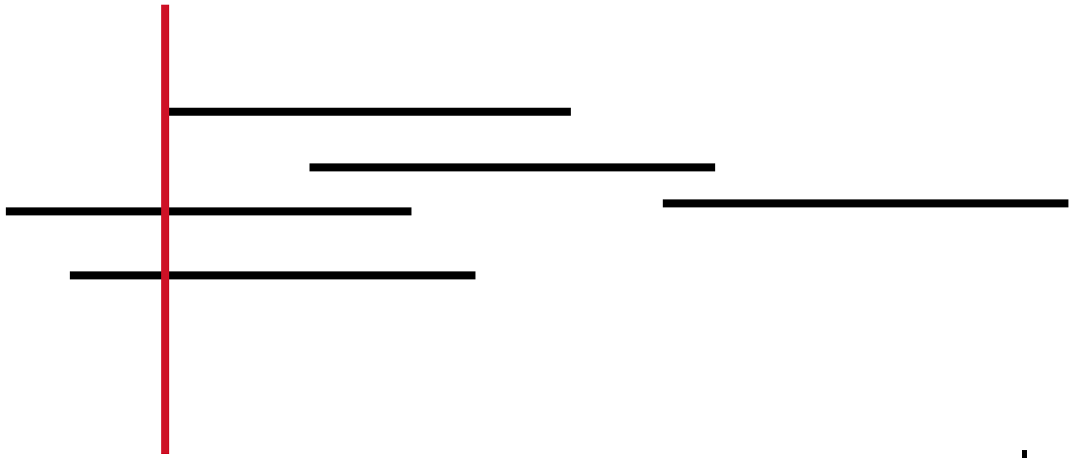


depth = 3

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine “sweeping” a vertical line from left to right while maintaining the current depth.

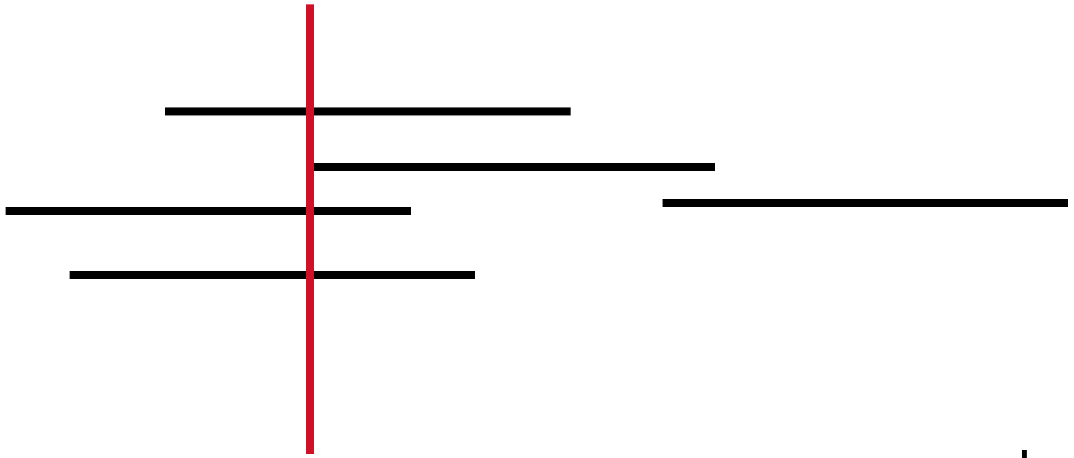


depth = 4

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine "sweeping" a vertical line from left to right while maintaining the current depth.

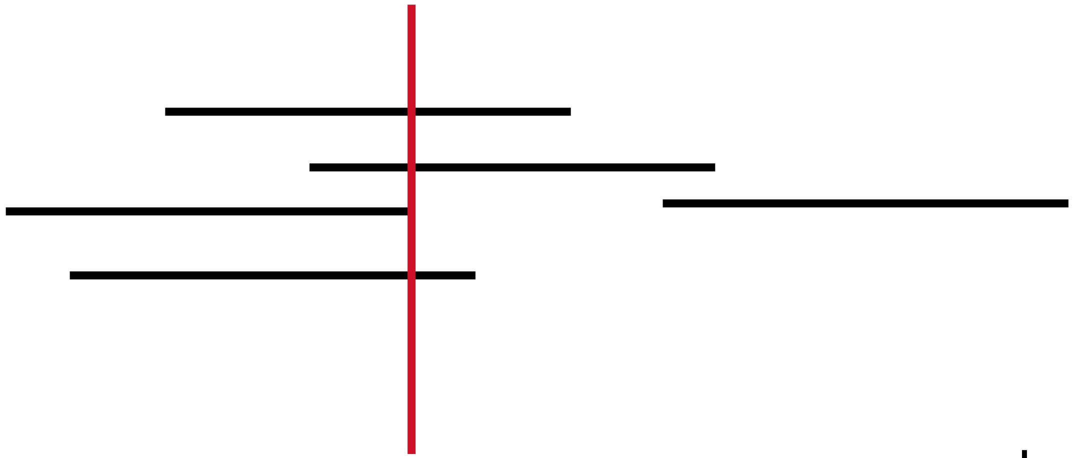


depth = 3

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine "sweeping" a vertical line from left to right while maintaining the current depth.



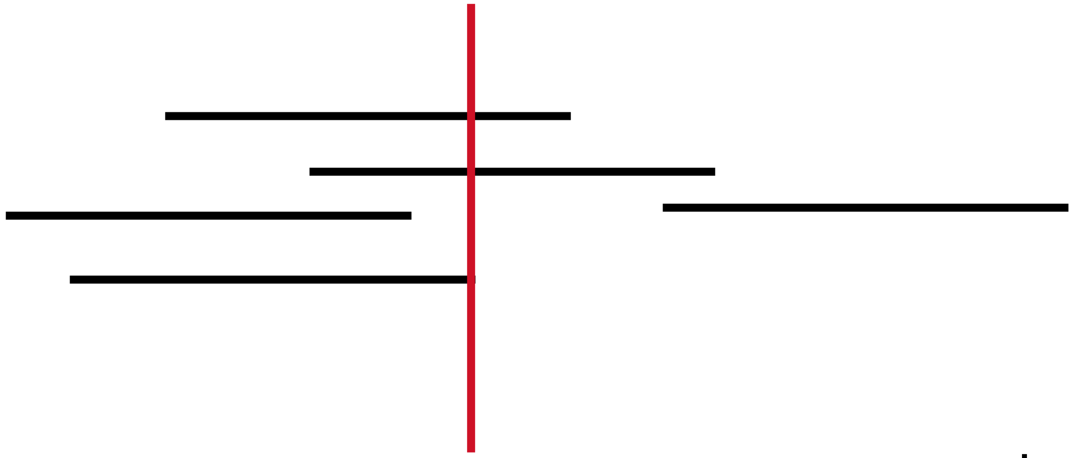
depth = 2

THE UNIVERSITY OF SYDNEY

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine "sweeping" a vertical line from left to right while maintaining the current depth.

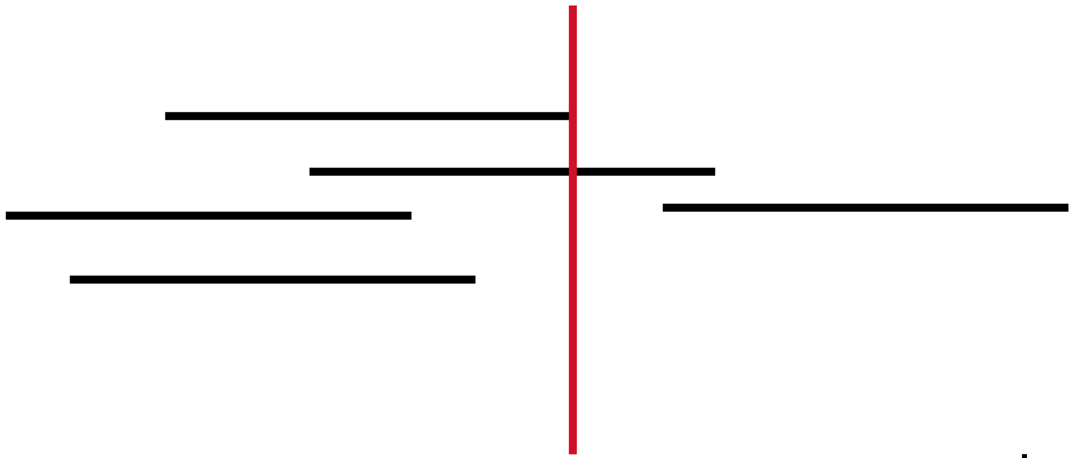


depth = 1

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine "sweeping" a vertical line from left to right while maintaining the current depth.

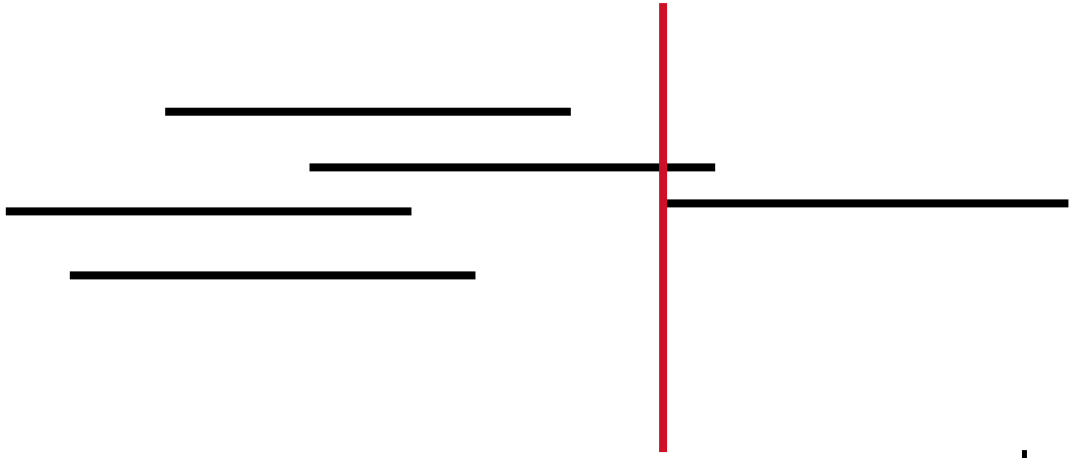


depth = 2

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine "sweeping" a vertical line from left to right while maintaining the current depth.

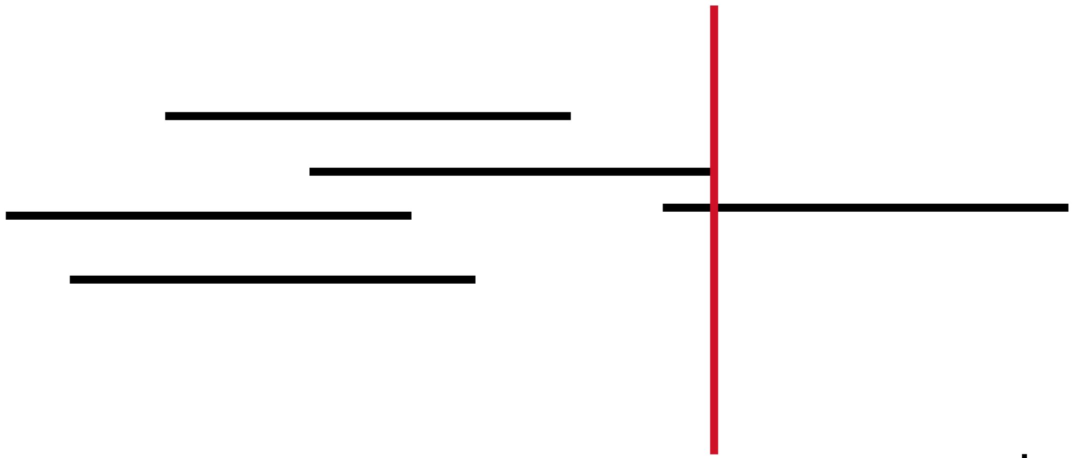


depth = 1

Depth of interval

How can we compute the depth?

The problem can be solved in O(n log n) using a sweepline approach. Imagine "sweeping" a vertical line from left to right while maintaining the current depth.



depth = 0

THE UNIVERSITY OF SYDNEY

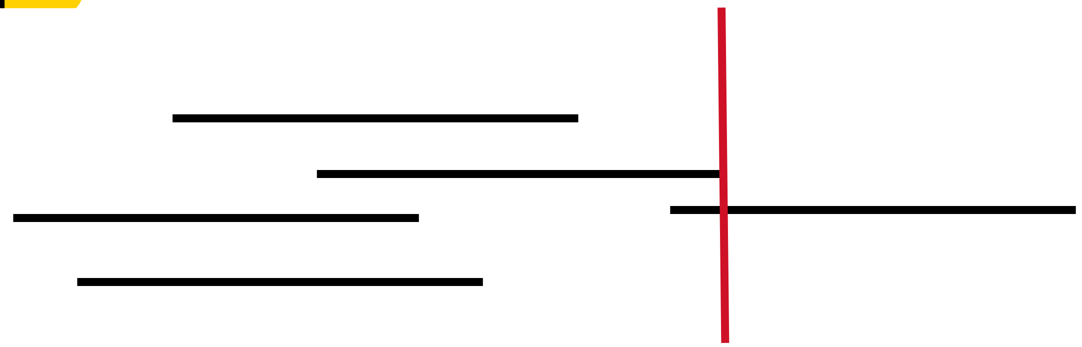
Depth of interval

If we can keep track of the current depth then we can easily also find the maximal depth.

The points where a change of depth may occur are called the event points $\Rightarrow$ endpoints of the intervals

End Points: Changes happen

The sweepline status is the information stored with the sweepline $\Rightarrow$ current depth.



Depth of interval

1. Sort endpoints from left to right $p_1, \ldots, p_{2n}$ $O(n \log n)$
2. currentDepth=0
3. maxDepth=0
4. for i=1 to 2n do

    if p_i is left endpoint then

        currentDepth = currentDepth + 1

        if maxDepth < currentDepth then

            maxDepth = currentDepth

    else {if p_i is a right endpoint}

        currentDepth = currentDepth – 1

5. end {for}
6. Report maxDepth

$O(n)$

Summary: Depth of intervals

Theorem:

The depth of a set of n intervals in 1D can be computed in $O(n \log n)$ time using a sweepline algorithm.

Sweepline

Main idea:

Sweep an “imaginary" line L across the plane while

(1) maintaining the status of L and [current depth]

(2) fulfilling an invariant. [the maximum depth to the left of L has been computed]

维持一些条件不变

- The status of L only changes at certain discrete event points.

  [endpoints of segments]

- When the sweep line encounters an event point the status is updated in such a way that the invariant is guaranteed to hold after the event point has been processed.

  [updating the depth counter]

Prove correctness

Correctness usually follows immediately from the invariant and the event points.

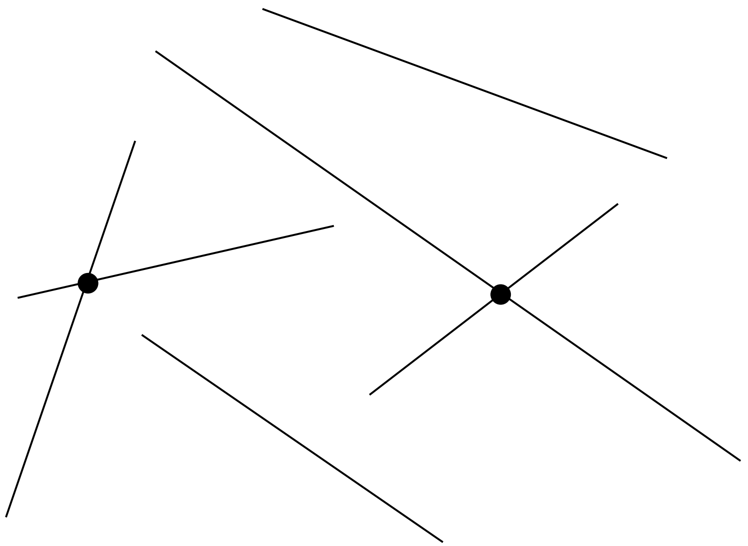
1) Prove that the status can't change between two consecutive event points and

   [if event points are correctly chosen this is usually trivial]

2) prove that the invariant holds before and after an event point is processed.

   [depth counter correct before new event and after an event has been processed]

Segment intersection

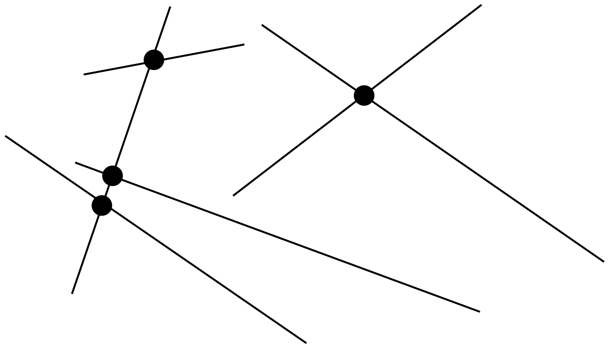


Segment intersection

Input: A set of n line segments $S=\{s_1, s_2, \dots, s_n\}$ in the plane, represented as pairs of endpoints.

Intersection detection:
Is there a pair of segments in S that intersect?

Intersection reporting:
Find all pairs of segments that intersect.

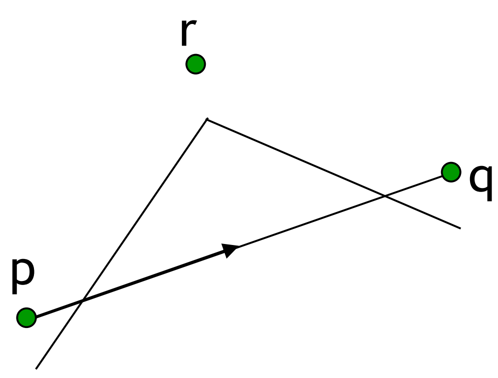


Check left turn a primitive?

How can we check if a point r lies to the left of a line pq?

$\Rightarrow$ Triangle $\Delta(p,q,r)$ is oriented counter-clockwise.

$p=(p_x,p_y)$, $q=(q_x,q_y)$ and $r=(r_x,r_y)$



$$D(p,q,r) = \begin{vmatrix} p_x & q_x & r_x \\ p_y & q_y & r_y \\ 1 & 1 & 1 \end{vmatrix}$$ Determinant

$= (q_x - p_x)(r_y - p_y) - (r_x - p_x)(q_y - p_y)$ [2 multiplications, 5 subtractions]

$\Delta(p,q,r)$ is oriented counter-clockwise iff $D(p,q,r) > 0$.

CCW(p,q,r) = true if D(p,q,r)>0 otherwise false

Segment intersection
Using determinant.

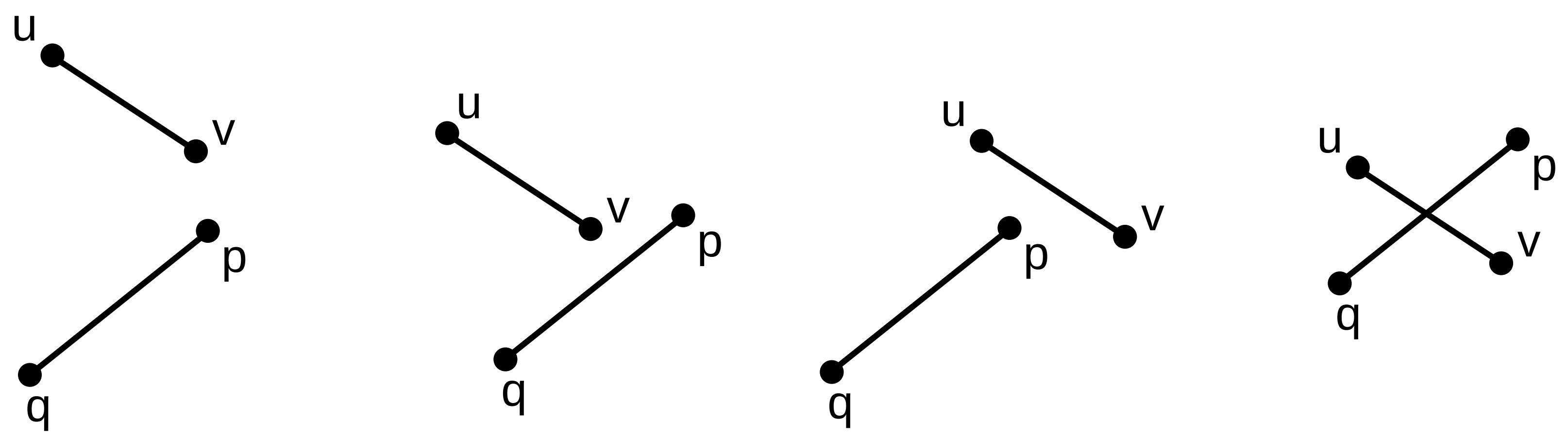
Testing intersection

Test if two segments (p,q) and (u,v) intersect.

boolean INTERSECT(Points u, v, p, q)

  return [(CCW(u, v, p) xor CCW (u, v, q)) and

    (CCW(p, q, u) xor CCW (p, q, v))]

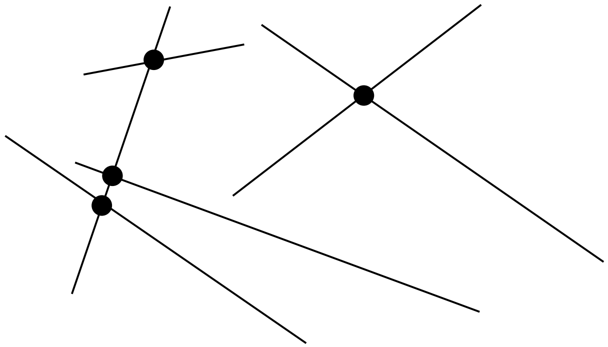


Segment intersection

Input: A set of n line segments $S=\{s_1, s_2, \dots, s_n\}$ in the plane, represented as pairs of endpoints.

Intersection detection:
Is there a pair of lines in S that intersect?

Intersection reporting:
Find all pairs of segments that intersect.



Brute force algorithm

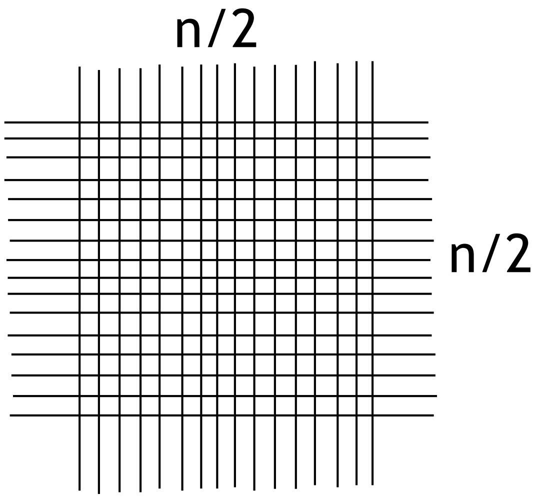
Check every possible pair of segments if they intersect

$\Rightarrow$ $O(n^2)$ time

Can we do better?

Detection? Maybe!

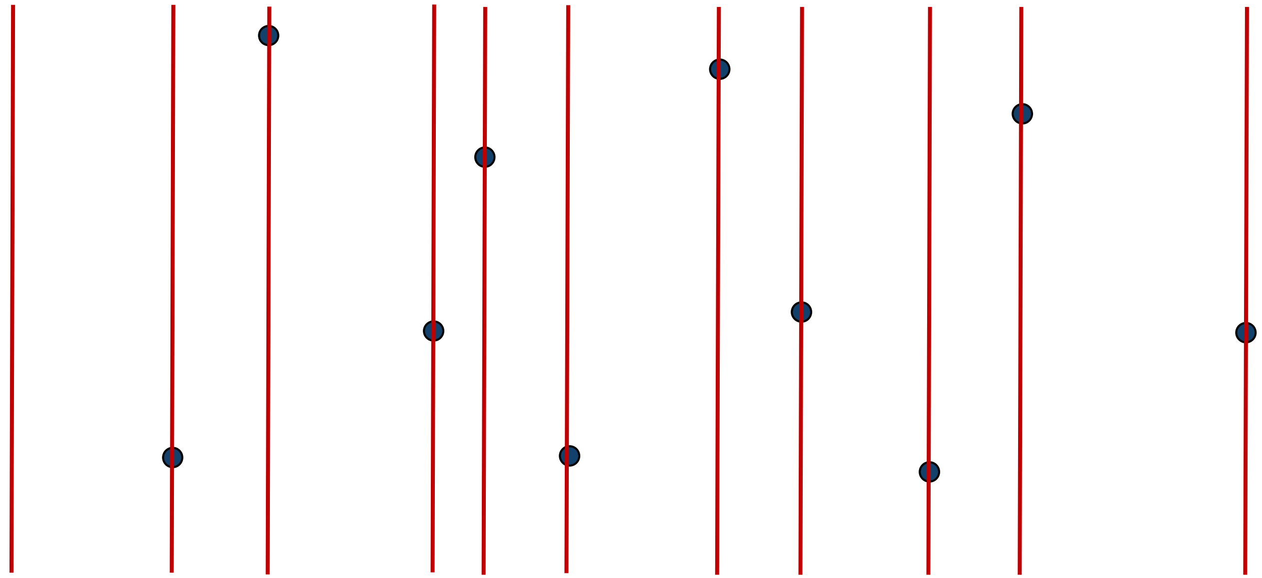
Reporting? Nope!



However, we can try to make the running time sensitive to the size of the output (h).

Design technique

- Simulate sweeping a vertical line from left to right across the plane.
- **Events**: Discrete points where sweep line status needs to be updated
- **Sweep line status**: Store information along sweep line
- Maintain **invariant**: At any point in time, to the left of sweep line everything has been properly processed.



Design technique

- Simulate sweeping a vertical line from left to right across the plane.
- **Events**: Discrete points where sweep line status needs to be updated
- **Sweep line status**: Store information along sweep line
- Maintain **invariant**: At any point in time, to the left of sweep line everything has been properly processed.

Algorithm Generic_Plane_Sweep:

Initialize **sweep line status** S at time x=-∞

Store initial events in **event queue** Q, a priority queue ordered by x-coordinate

while $Q \neq \emptyset$

// extract next event e:

e = Q.extractMin();

// handle event:

Update sweep line status

Discover new upcoming events and insert them into Q

Plane sweep algorithm: intersection detection

Plane sweep (general method):

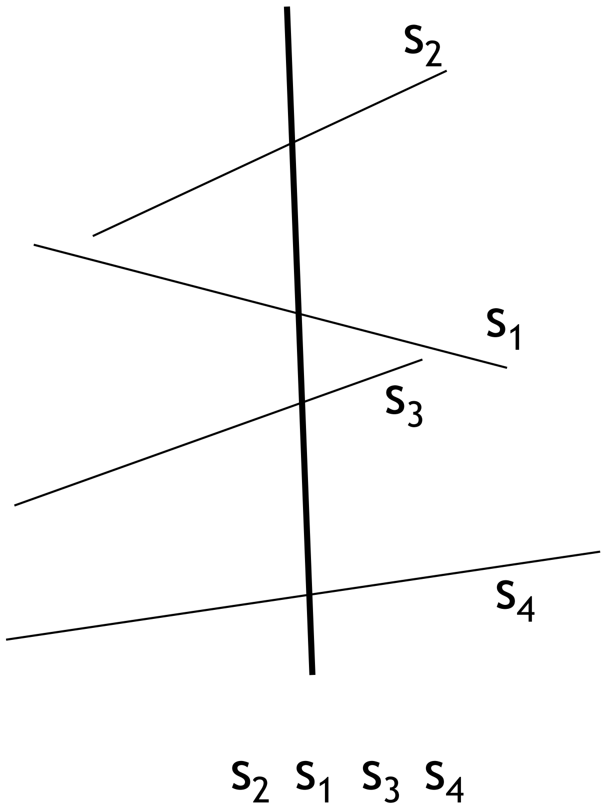
1. Sweep the input from left to right and stop at event points
2. Maintain invariant (status and structure)
3. At each event point restore invariant

Event points?

end points of the segments

Invariant:

- The order of the segments along the sweep line
- No intersections among segments encountered by the sweepline

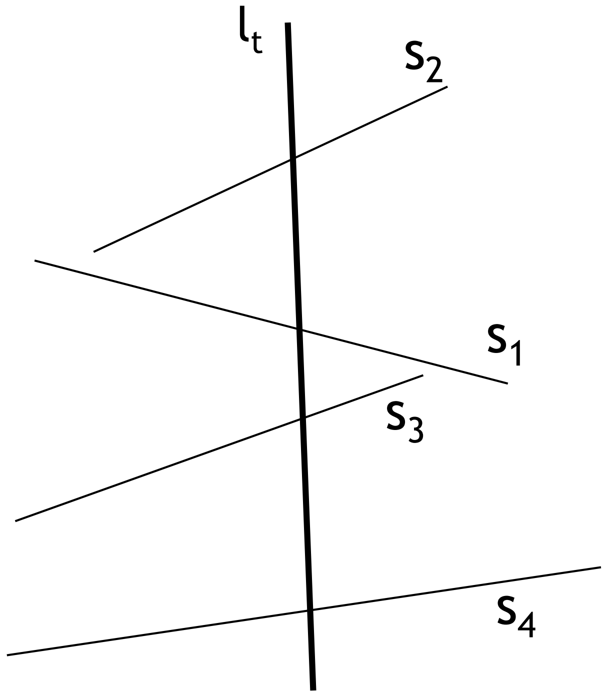


Plane sweep algorithm

l_t : the vertical line at x=t
S_t : the sequence of the segments that intersects l_t in order from top to bottom.

Idea:
Maintain S_t while l_t moves from left to right

Invariant:
- We know S_t
- No intersections to the left of L

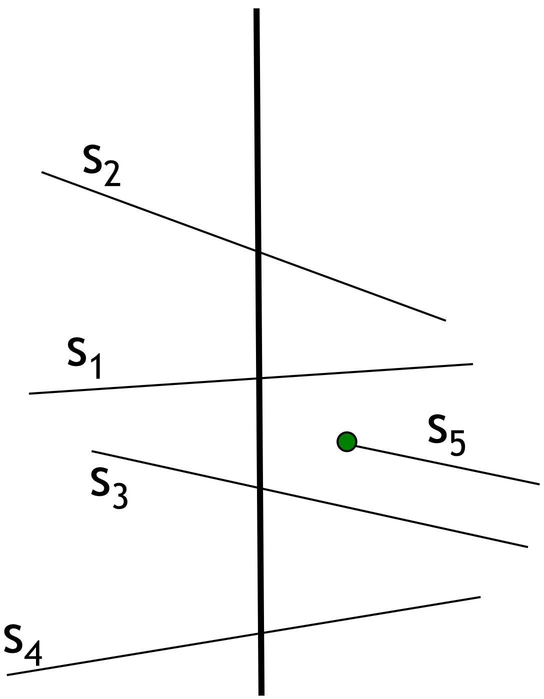


S_t: s_2 s_1 s_3 s_4

Event handling

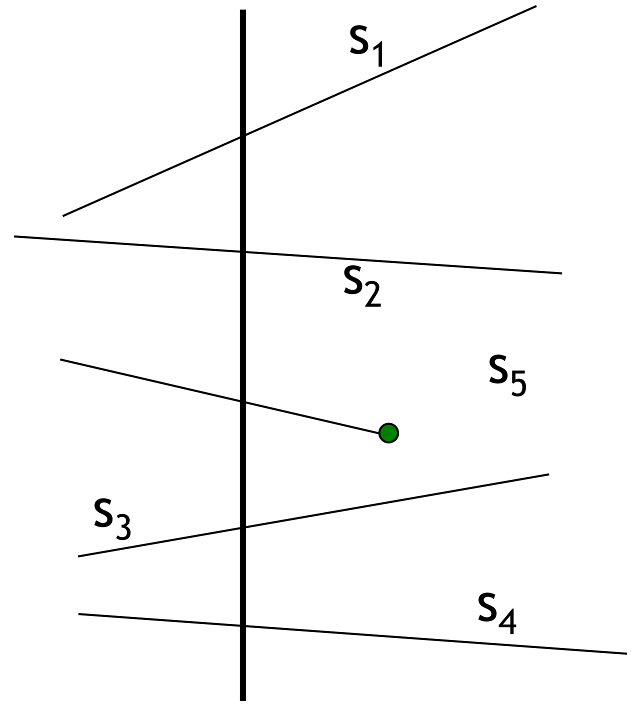
Initially: Let $t_1, t_2, \ldots, t_{2n}$ be the x-coordinates of the endpoints

Case 1: t_{i+1} is a left end point



$s_2 \ s_1 \ s_3 s_4$

Case 2: t_{i+1} is a right end point



$s_2 \ s_1 \ s_5 \ s_3 \ s_4$

Data structure

We need to store S_t in a data structure that supports fast insertions and deletions.

Structure: Balanced binary search trees

Each update can be done in O(log n) time

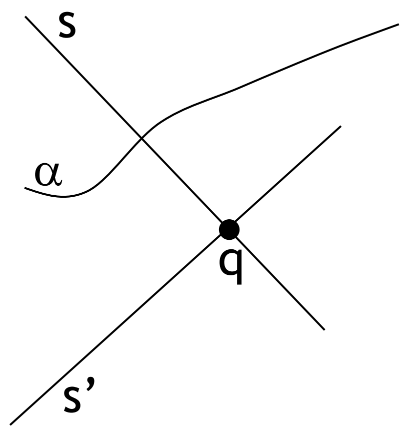
Problem: We did not check intersections!

Intersection points

Observation: Let q be the leftmost intersection point, where q is an intersection point between the segments s and s' with x-coordinate t then s and s' are adjacent in S_t.

Proof:

Assume the opposite, i.e., S_t = (....s...α...s'...)

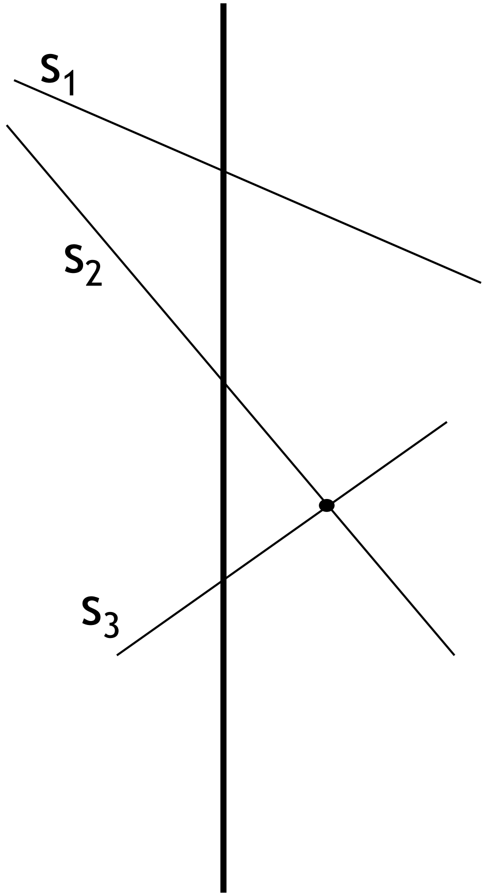


1. Right endpoint of α must be to the right of q.
2. If q below α then α intersects s to the left of q.
   $\Rightarrow$ contradicts that q is leftmost intersection
3. Similarly, q cannot lie above α

$\Rightarrow$ contradiction!

Intersection point

Conclusion: To detect an intersection we only need to check adjacent segments in S_t.



Algorithm

Algorithm DetectIntersection(S)

1. Store the segments S_t in a balanced binary search tree T w.r.t. the order along l_t.

2. When deleting a segment in T two segments become adjacent. We can find them in O(log n) time and check if they intersect.

3. When inserting a segment s_i in T it becomes adjacent to two segments. We can find them in O(log n) time and check if they intersect s_i.

4. If we find an intersection we're done!

Time complexity?

Algorithm - detection

Every endpoint is an event point $\Rightarrow$ 2n event points

Insert segment s

Add s to T:	O(log n)
Check neighbours:	2 × O(log n)

Delete segment s

Remove s from T:	O(log n)
Check new neighbours:	2 × O(log n)

Total: O(n log n)

Intersection reporting

The problem of reporting all intersections between a set of n segments has a lower bound of $O(n^2)$

How can we change the algorithm to report all intersections?

Event points = endpoints plus intersection points

Event Points S = {The endpoints of the segments and all segment intersections}

Where does the order along l_t change?

With the new event points we can run the algorithms as before (with minor modifications).

Running time: $O(n \log n + h \log n)$

h-># of output (size of output)

Segment intersection

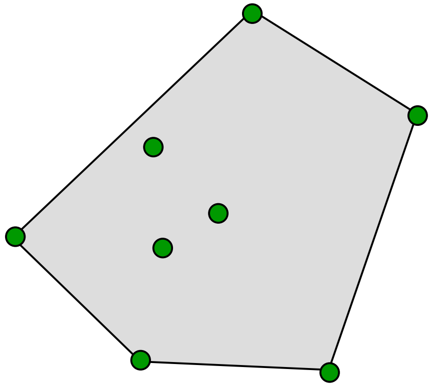
Sweep-line technique

[Shamos & Hoey'75], [Lee & Preparata'77], [Bentley & Ottman'79]

Intersection reporting

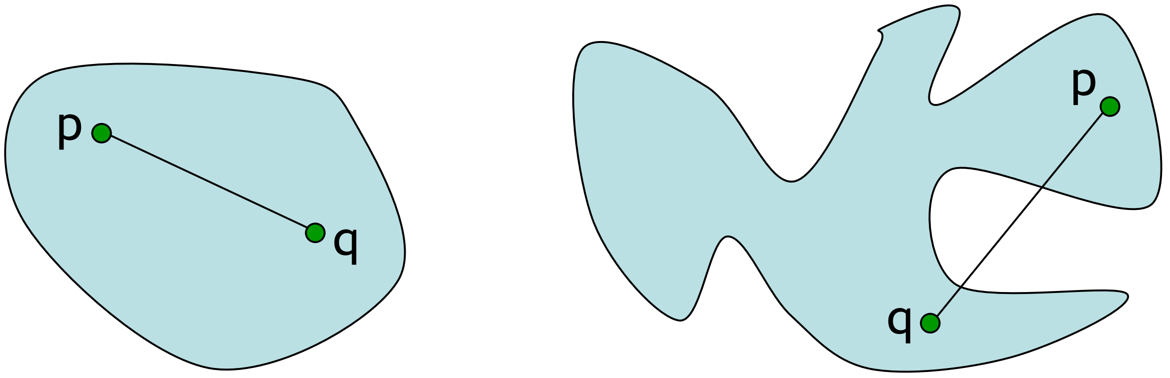
- O(n log n + h log n) time — [Bentley & Ottmann'79]
- $O(n \log^2 n / \log\log n + h)$ — [Chazelle'86]
- O(n log n + h) — [Chazelle & Edelsbrunner'88]
- O(n log n + h) — [Balaban'95] (also works for curves)

Convex hulls and the sweep line technique



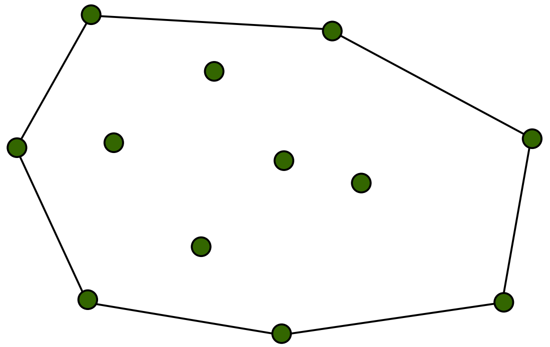
Convex hull

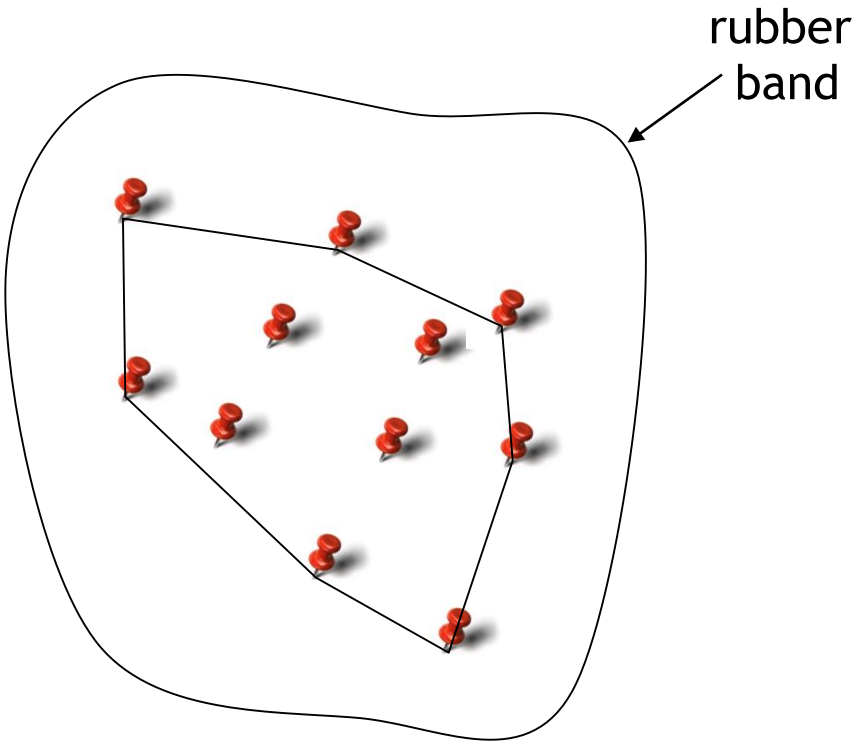
A subset S of the plane is convex if for every pair of points p,q in S the straight line segment pq is completely contained in S.



Convex hull

The convex hull of a point set is the smallest convex set containing S.





We only want to find the hull! 壳

CH algorithm 1

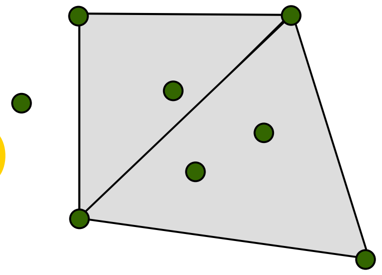
Definition:

The convex set of a set of point S in d dimensions is the union of all convex combinations of (d+1) points of S.

d=2: Convex combination of 3 points $\Rightarrow$ a triangle!

Definition implies an algorithm:

A point that does not lie in the interior of any triangle of S is a CH vertex.



CH algorithm 1

Algorithm CH1(S)

1. for every possible triple of points x,y,z in S do
2. for every point p in S do
3. if q lies within the triangle (x,y,z) then
4. discard q from S

Time complexity?

Step 1 is performed $O(n^3)$ times
Step 2 is performed n times/iteration
Step 3 and step 4 cost O(1) /iteration

Total time: $O(n^4)$

In one iteration a linear number of points along the convex hull computed so far might be visited but on average only a constant number of nodes are visited.

CH algorithm 1: running time

Assumption: 10^8 instructions per second

Input size: 1 million points = 10^6 points

$\Rightarrow$ running time ~ $n^4/10^8 = 10^{16}$ seconds ~ 317 million years

CH in 1 second: 100 points

CH algorithm 2

Definition:
The CH of S is the intersection of all halfspaces that contain S.

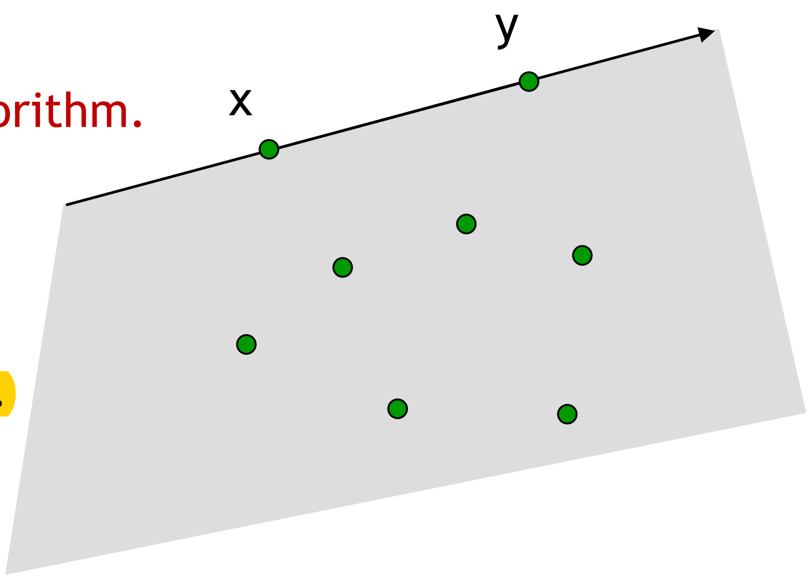
Why is the intersection of two convex sets a convex set? Union?

union of two convex sets is not a convex set.

This definition implies a second algorithm.

Consider an edge xy of CH(S).

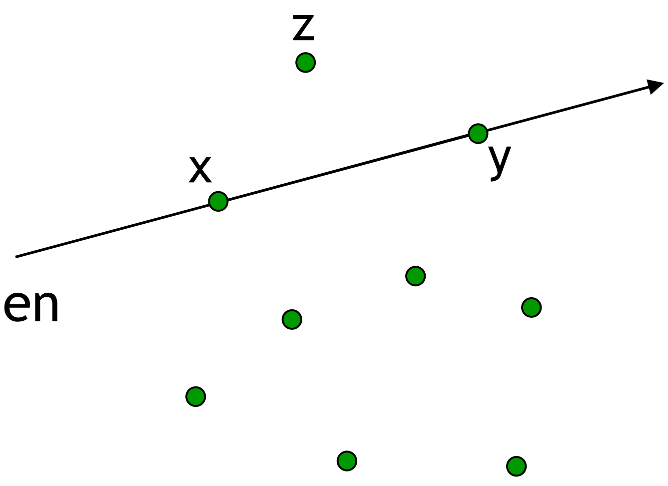
All points of S must lie to the right of the directed line through x and y.



CH algorithm 2

Algorithm CH2(S)

1. for every ordered pair x,y in S do
2. valid $\leftarrow$ true
3. for every point z in S-{x,y} do
4. if z lies to the left of xy then
5. valid $\leftarrow$ false
6. if valid then
7. add xy to CH
8. Sort the edges in CH



Time complexity?

Steps 1-2, 6-7 : $O(n^2)$ times
Steps 3-5 : (n-2) times/iteration
Step 8 : O(n log n)

Total time: $O(n^3)$

CH algorithm 2: running time

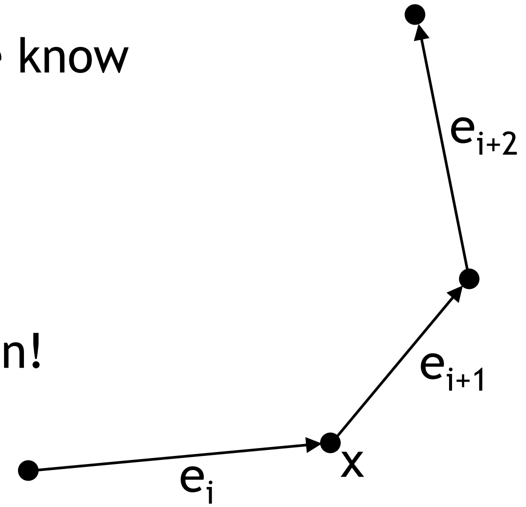
Assumption: 10^8 instructions per second

Input size: 1 million points = 10^6 points
$\Rightarrow$ running time ~ $n^3/10^8 = 10^{10}$ seconds ~ 317 years

CH in 1 second: 464 points

CH algorithm 3 (Gift Wrapping)

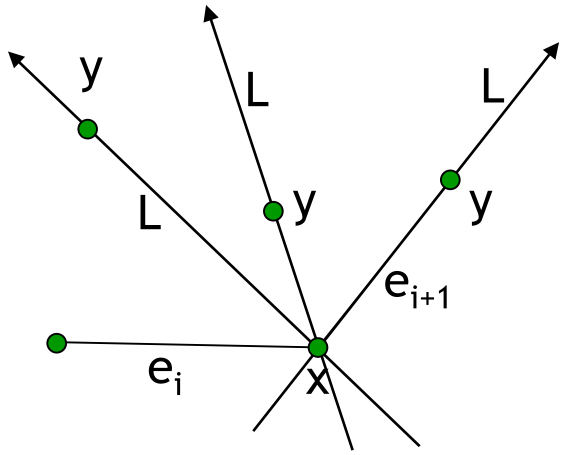
Can we compute the CH faster? Is there anything we know about the CH that we haven't used?



The edges in the CH are linked into a convex polygon!

If we found an edge on the CH with endpoint at x then the next edge must start at x.

Idea:
Draw a line L through x and a point y. Are there any points to the right of L?
If not (x,y) is an edge of CH.

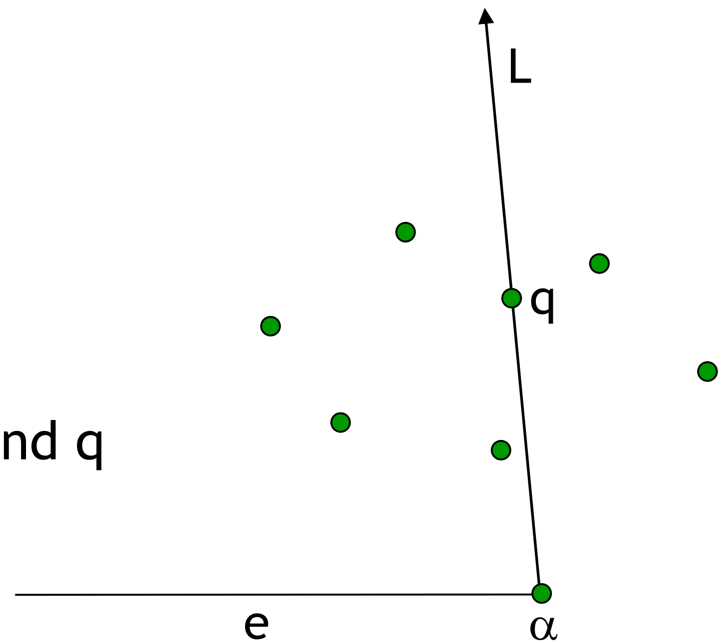


Start point?

CH algorithm 3

Algorithm CH3(S)

```
find lowest point p in S
e ← ( (-∞,p_y),p)
α ← p
 repeat
    valid ← true
    for every point q in S-{α} do
          L ←  directed line through α and q
          for every point r in S-{α,q} do
             if r to the right of L then
                 valid ← false
     if valid then
          add αq to CH
          α ← q
 until α == p
```



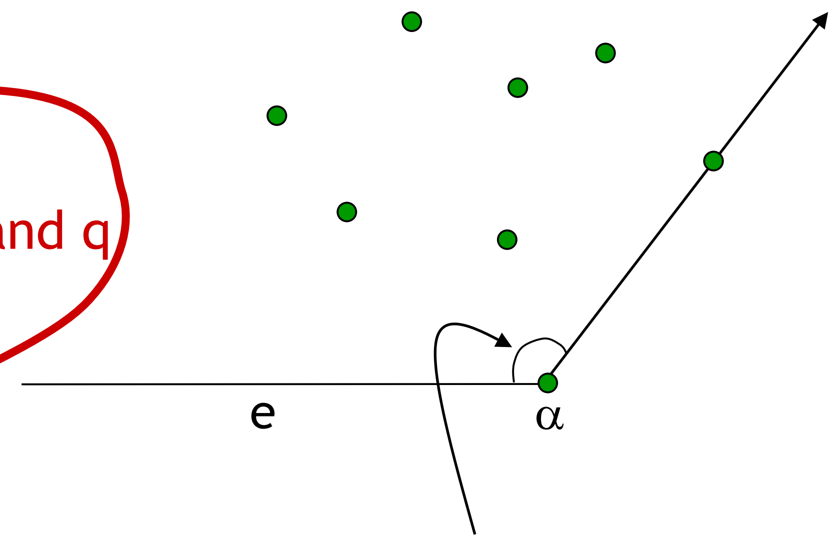
Time complexity: $O(n^3)$

CH algorithm 3

Algorithm CH3(S)

1. find lowest point p in S
2. e ← ($(-\infty,p_y)$,p)
3. α ← p
4. repeat
5. valid ← true
6. for every point q in S-{α} do
7. L ← directed line through α and q
8. for every point r in S-{α,q} do
9. if r to the right of L then
10. valid ← false
11. if valid then
12. add αq to CH
13. α ← q
14. until α == p

Can this be done faster?

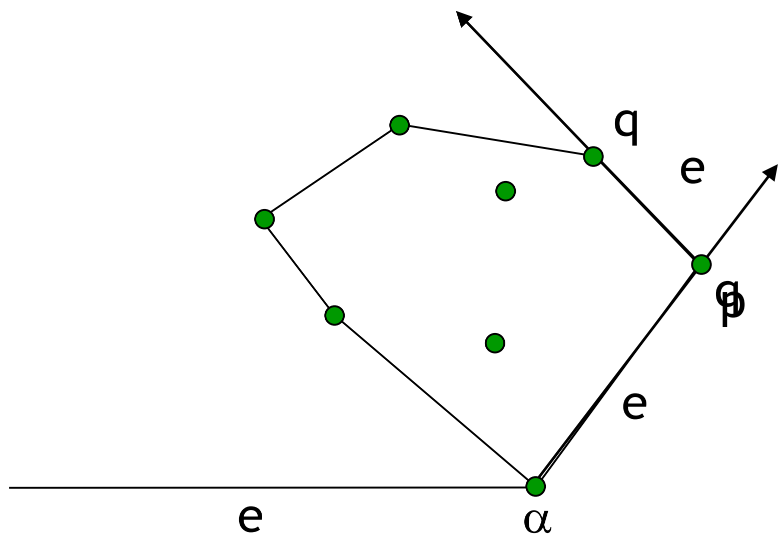


Looking for the point with largest angle!

CH algorithm 4

Algorithm CH4(S)

1. find lowest point p in S
2. $\alpha \leftarrow (-\infty, p_y)$
3. e $\leftarrow$ (α,p)
4. repeat
5. maxAngle $\leftarrow$ 0
6. for every point q in S do
7. if $\angle$(e,(p,q)) > maxAngle then
8. nextPoint $\leftarrow$ q
9. maxAngle $\leftarrow$ $\angle$(e,(p,q))
10. e $\leftarrow$ (p,nextPoint)
11. p $\leftarrow$ q
12. until α == p



Time complexity: $O(n^2)$

What if the number of points on the CH is small?

CH algorithm 4: running time

Assumption: 10^8 instructions per second

Input size: 1 million points = 10^6 points
$\Rightarrow$ running time ~ $n^2/10^8 = 10^4$ seconds ~ 3 hours

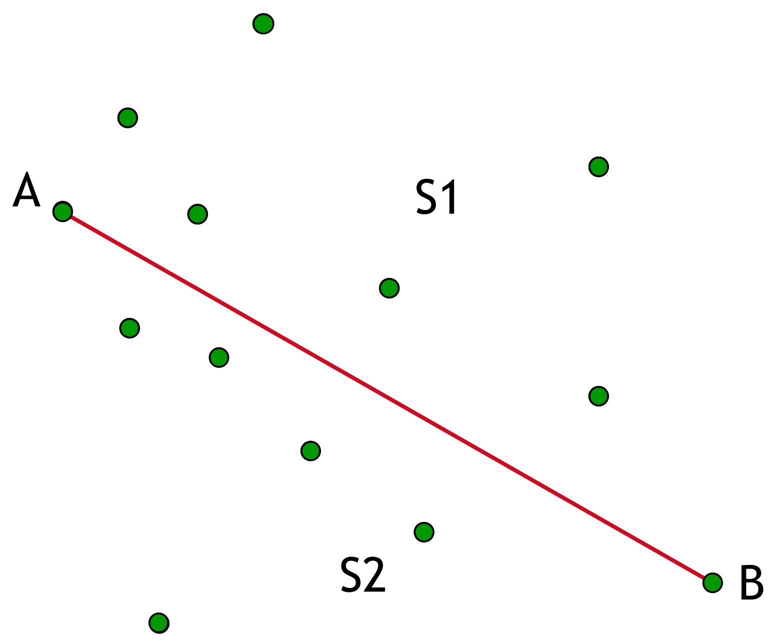
CH in 1 second: 10,000 points

CH algorithm 5 - QuickHull

Divide-and-Conquer approach

QuickHull(S)

1. A = leftmost point of S
2. B = rightmost point of S
3. S1 = {points in S above AB}
4. S2 = {points in S below AB}
5. FindHull(S1,A,B)
6. FindHull(S2,B,A)

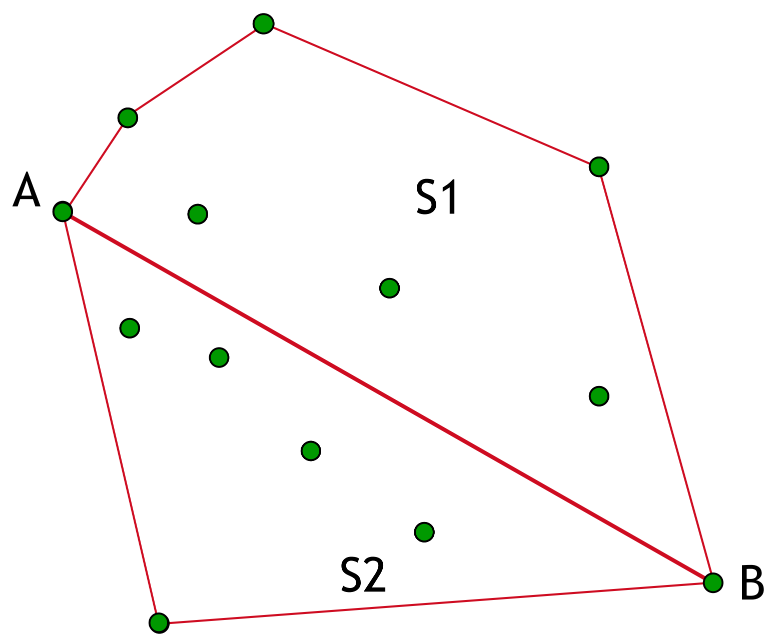


CH algorithm 5 - QuickHull

Divide-and-Conquer approach

QuickHull(S)

1. A = leftmost point of S
2. B = rightmost point of S
3. S1 = {points in S above AB}
4. S2 = {points in S below AB}
5. FindHull(S1,A,B)
6. FindHull(S2,B,A)

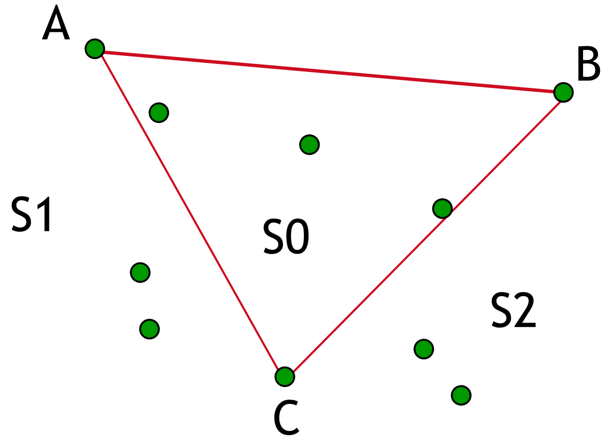


CH algorithm 5 - QuickHull

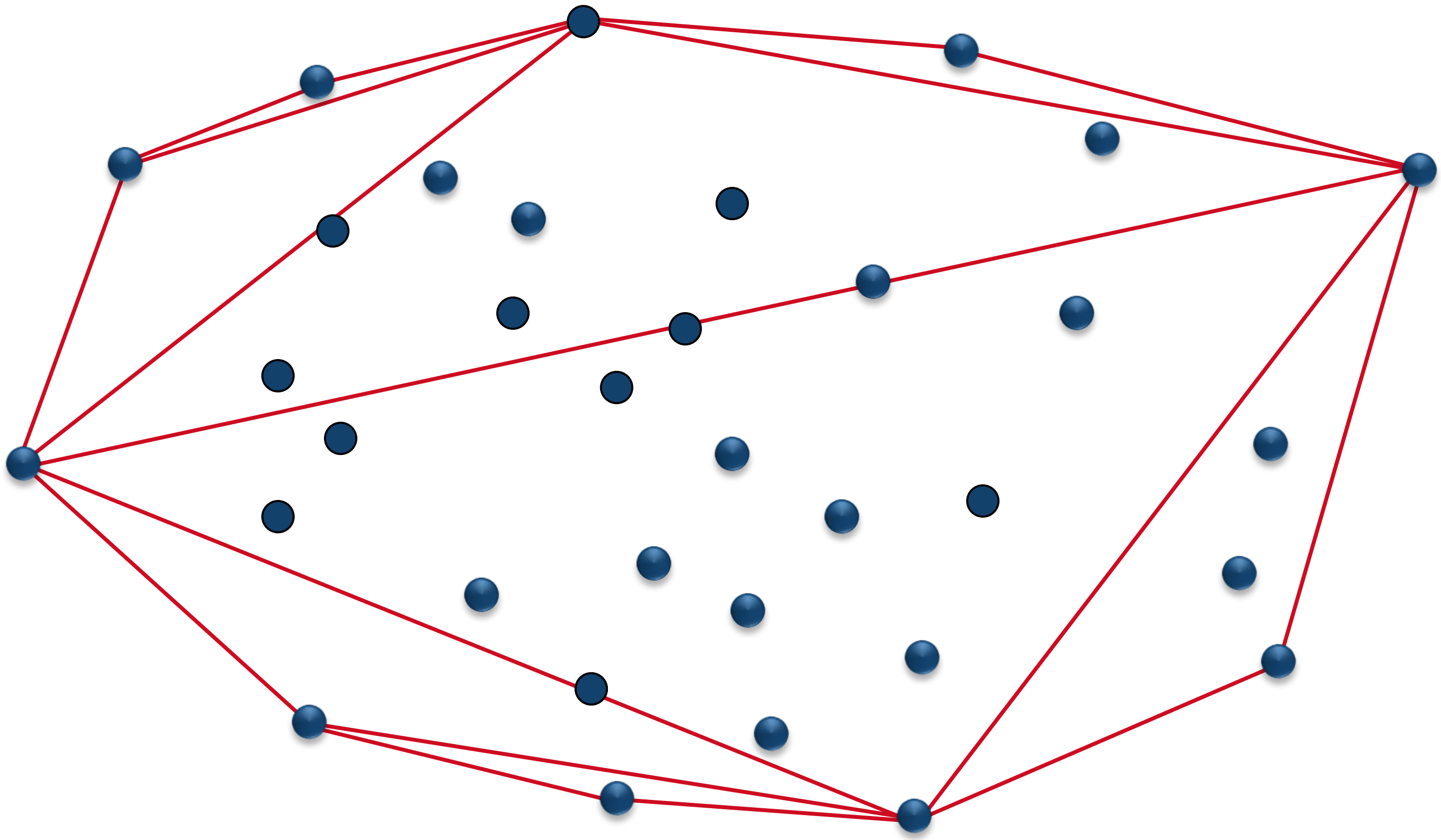
FindHull(S, A, B)

If S not empty then

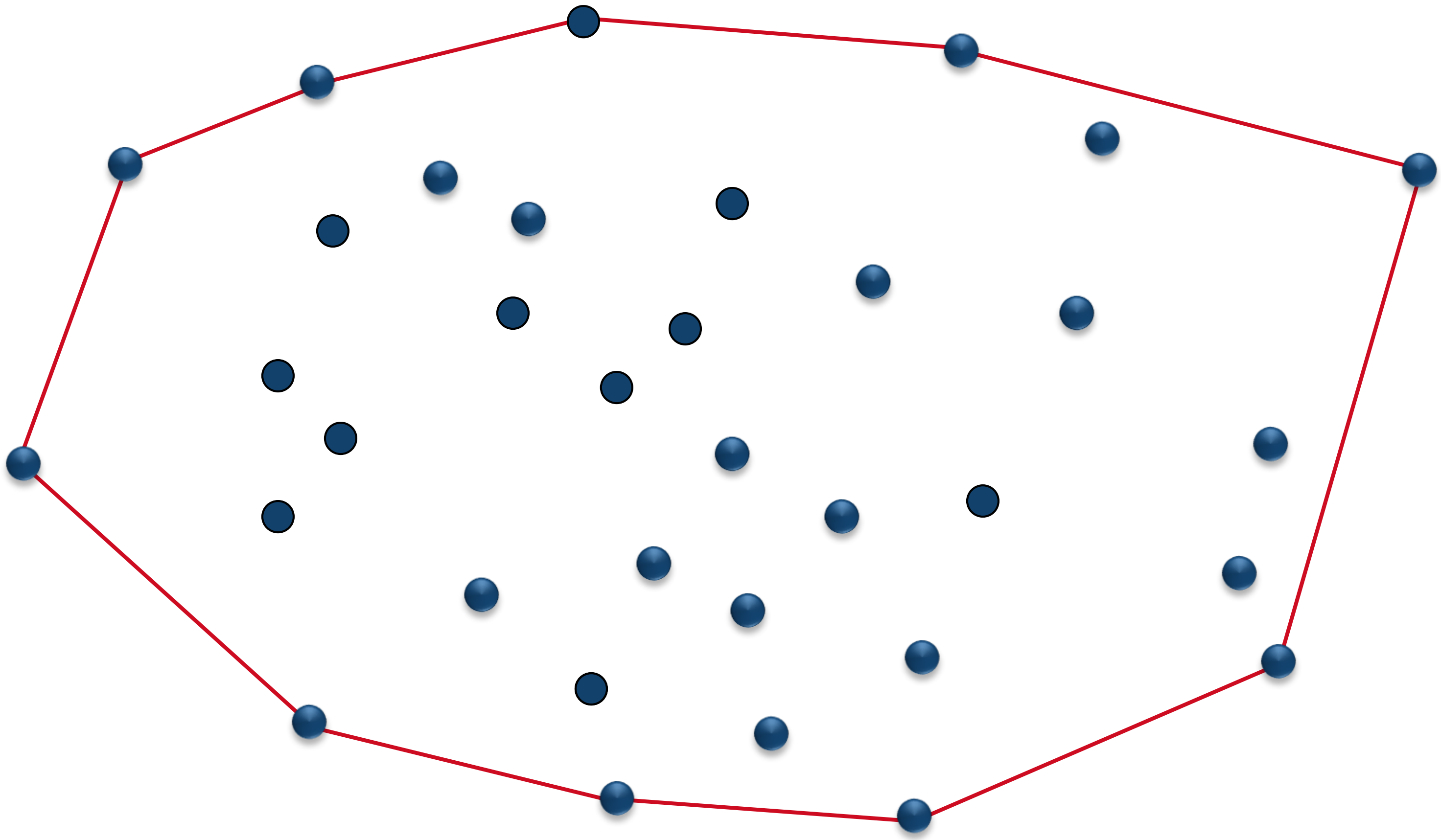
1. Find farthest point C in S from AB
2. Add C to convex hull between A and B
3. S0={points inside ABC}
4. S1={points to the right of AC}
5. S2={points to the right of CB}
6. FindHull(S1, A, C)
7. FindHull(S2, C, B)



CH algorithm 5 - QuickHull



CH algorithm 5 - QuickHull



CH algorithm 5 - QuickHull

QuickHull

- Compute A and B — $O(n)$ time
- FindHull(S_1,A,B) — $T(|S1|)$ time
- FindHull(S_2,B,A) — $T(|S2|)$ time

$$T(n)=T(|S_1|)+T(|S_2|)+O(n)$$

Worst Case:

$$T(n) = T(n-2) + O(n)$$
$$= T(n-3) + O(n) + O(n)$$
$$= \ldots = O(n^2)$$

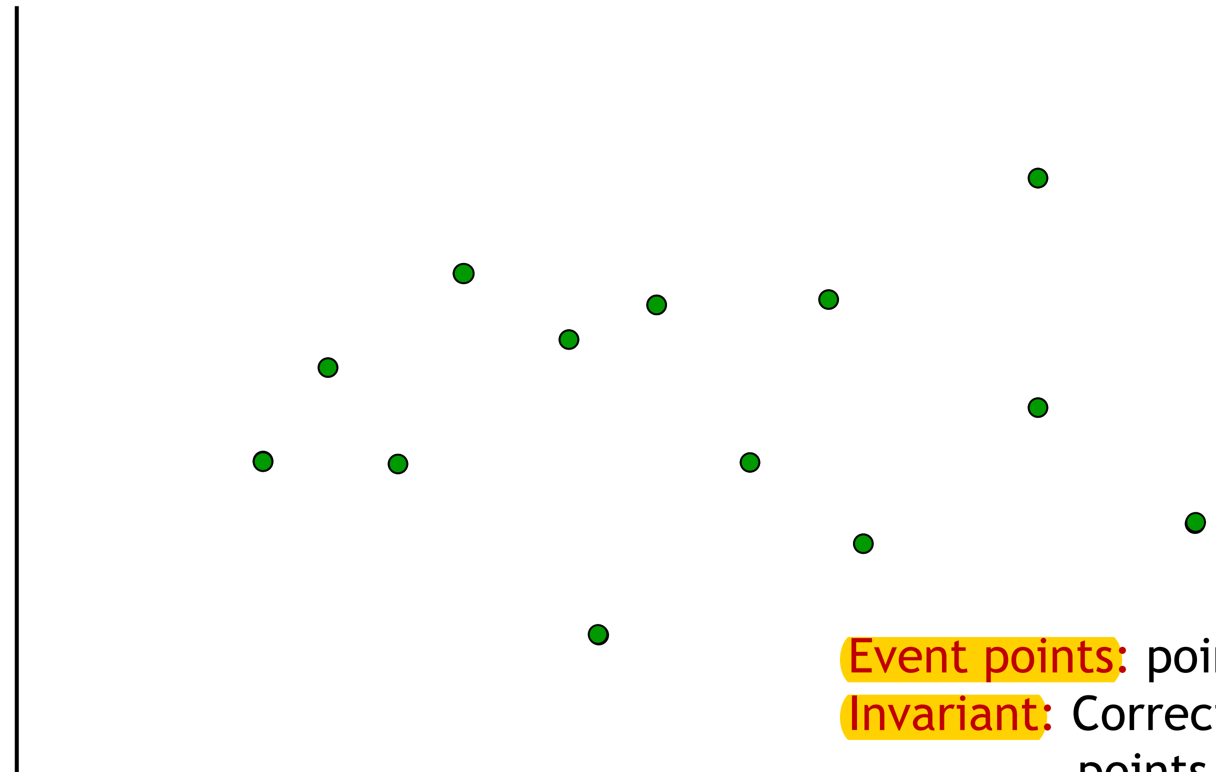
What if points are “nicely” distributed?

$$T(n) < T(n/2) + T(n/2) + O(n)$$
$$= O(n \log n) \qquad \text{Why?}$$

THE UNIVERSITY OF SYDNEY

CH algorithm 6 – sweep line approach

Idea: Maintain hull while adding the points one by one, from left to right ⇔ sweep the point from left to right

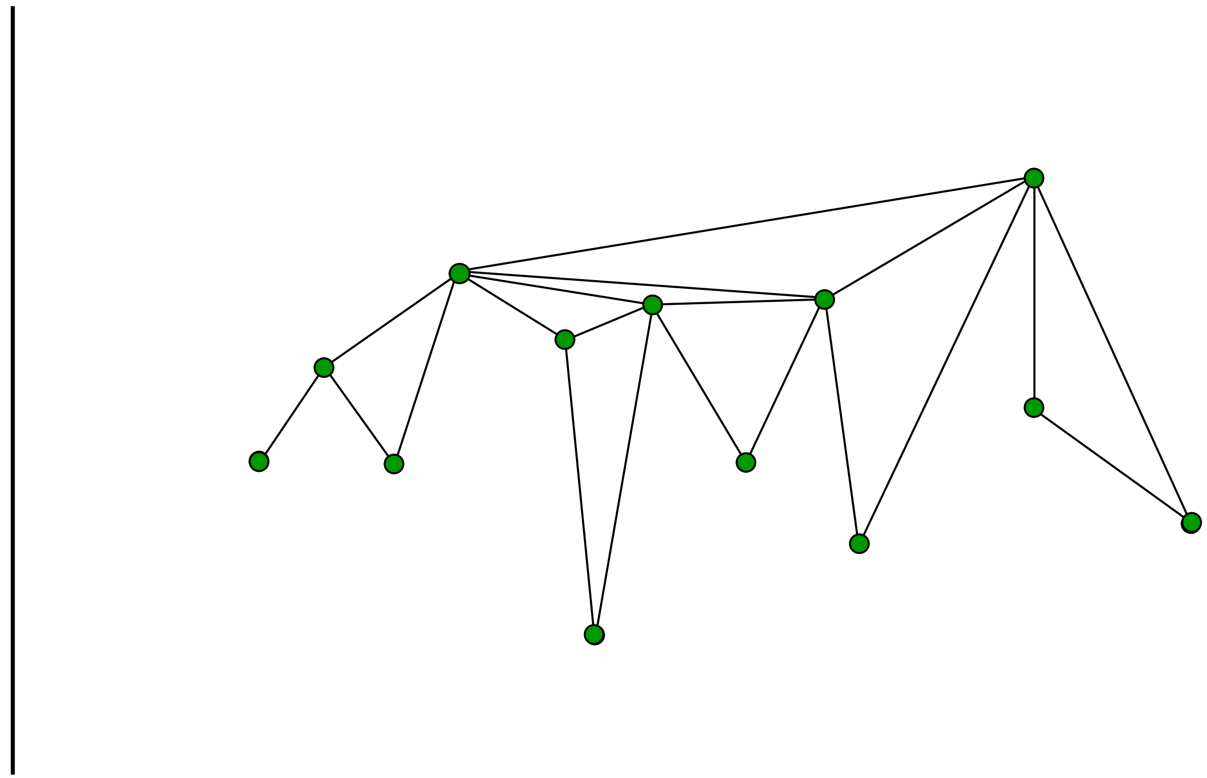


Event points: points

Invariant: Correct upper CH of points to the left of L

CH algorithm 6 – sweep line approach

Idea: Maintain hull while adding the points one by one, from left to right ⇔ sweep the point from left to right

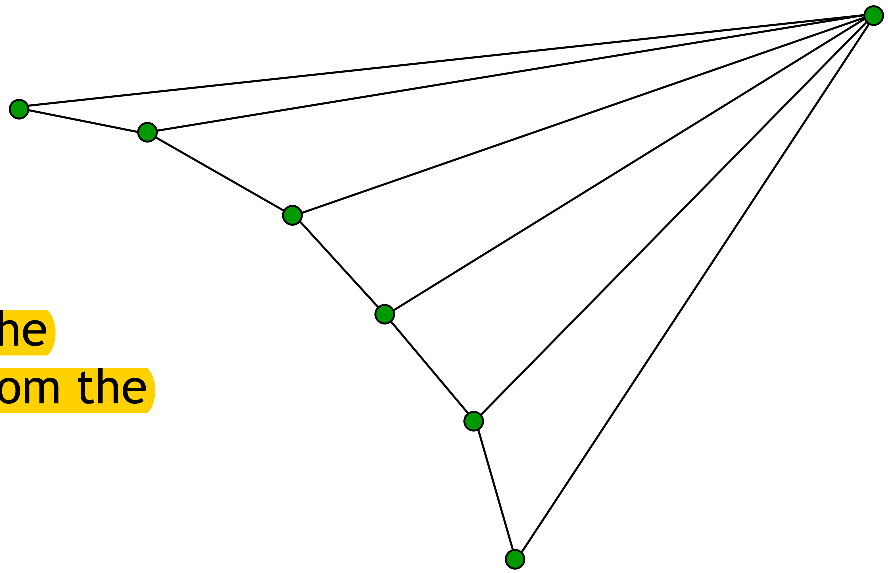


CH algorithm 6 - sweep line approach

Running time?

O(n) per insertion $\Rightarrow$ O(n^2) in total

Can it be that bad?



A point is only added to the hull once and removed from the hull once!

CH algorithm 6 - sweep line approach

Algorithm CH6(S)

1. sort the points in S from left to right $\langle p_1, p_2, \ldots, p_n \rangle$
2. $L_{upper} \leftarrow \langle p_1, p_2 \rangle$
3. for $i \leftarrow 3$ to n do
4. append p_i to L_{upper}
5. while $|L_{upper}| > 2$ and the last three points (q_1, q_2, q_3) turn left do
6. Delete q2 from L_{upper}
7. $L_{lower} \leftarrow \langle p_1, p_2 \rangle$

…

13. $L \leftarrow join(L_{upper}, L_{lower})$
14. return L

Time complexity: O(n log n)

CH algorithm 6: running time

Assumption: 10^8 instructions per second

Input size: 1 million points = 10^6 points

$\Rightarrow$ running time ~ $n \log n / 10^8$ = 0.2 seconds

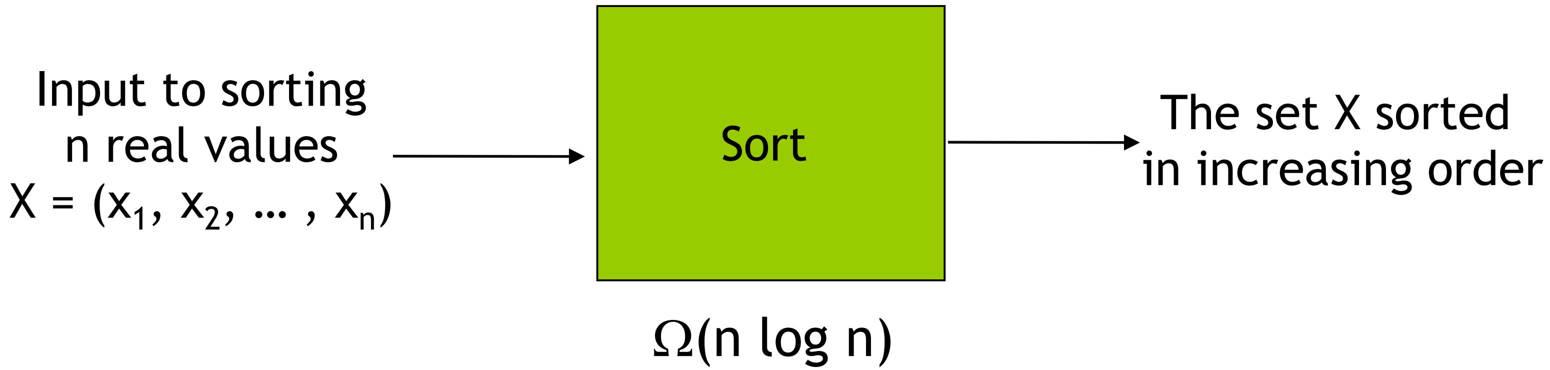
CH in 1 second: 4.5M points

Lower bound

Can we do better than $O(n \log n)$?

Prove a lower bound! Use a reduction from Sorting.

Sorting = $\Omega(n \log n)$ in the algebraic decision tree model

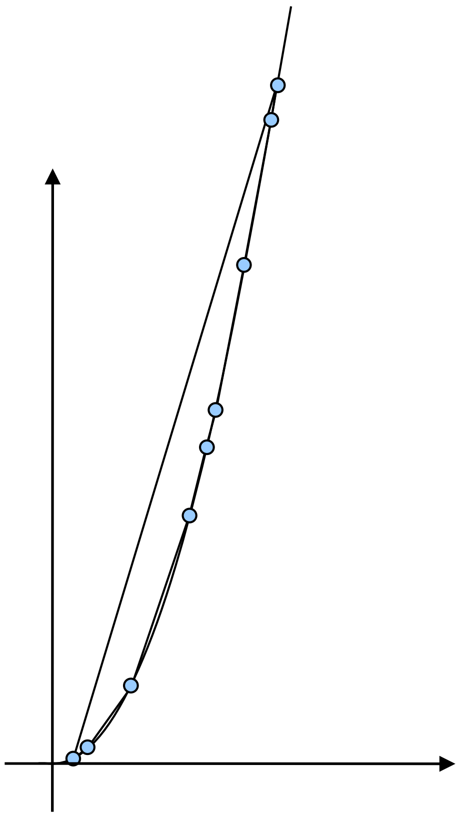


Lower bound

For each value x_i in X construct a point

$$p_i = (x_i, x_i^2)$$

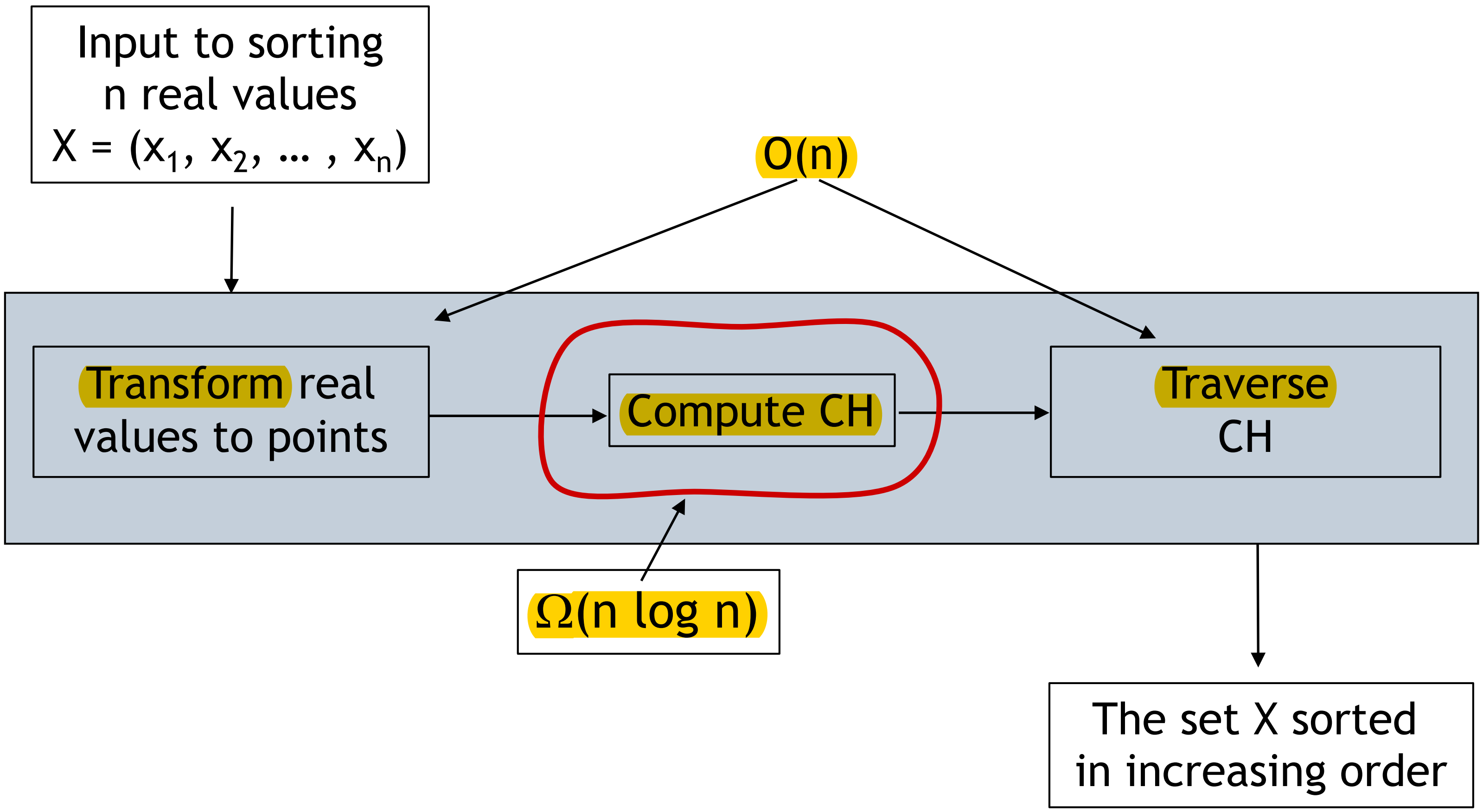
$P = (p_1, p_2, \dots, p_n)$



Compute CH of P

Find the leftmost point p in the CH.
Traverse the CH counter-clockwise from p and output the vertices in the order they are encountered. -> Points in sorted order!

Lower bound


Input to sorting
n real values
X = (x1, x2, … , xn)
O(n)
Transform real values to points
Compute CH
Traverse CH
Ω(n log n)
The set X sorted in increasing order

What is known?

Preparata & Hong'77 $O(n \log n)$

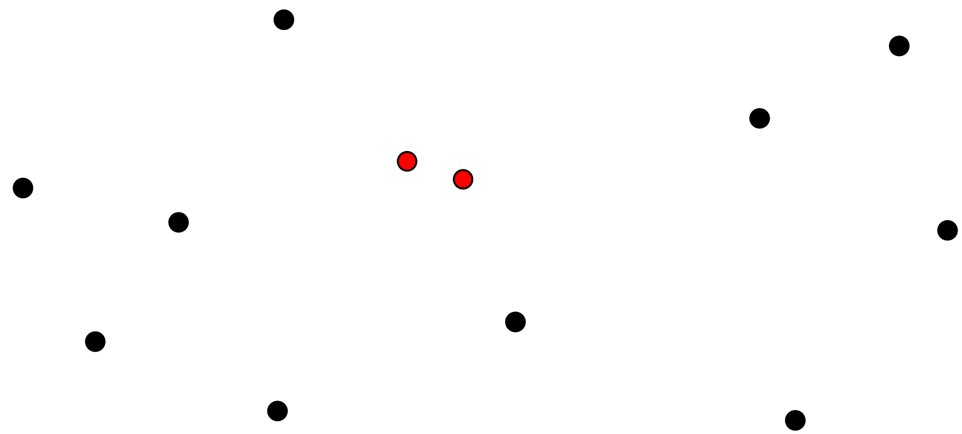
Kirkpatrick & Seidel'86 $O(n \log h)$

Dynamic convex hull
Brodal & Jacob'02 $O(\log n)$ time/update

[This was an open problem since 1981]

d dimensions
Chazelle'93 $\Theta(n \log n + n^{\lfloor d/2 \rfloor})$

Closest pair using a sweepline approach

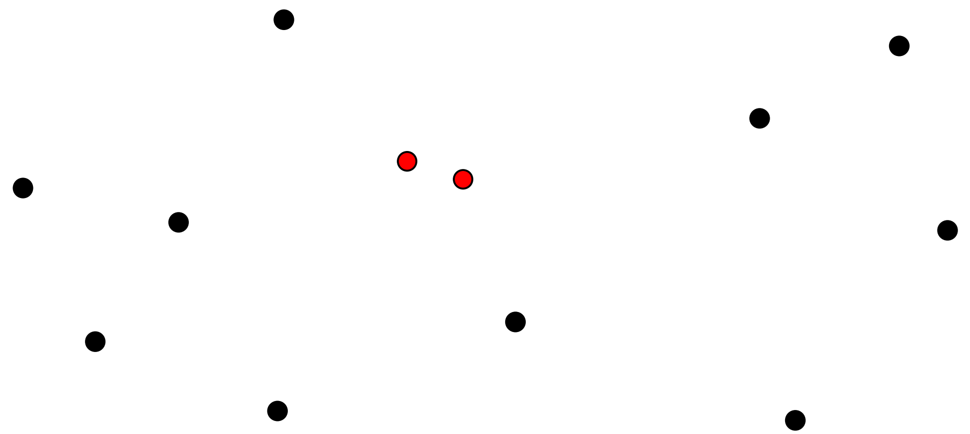


Time Complexity O(nlogn)

Closest Pair

Input: A set of n points $S=\{s_1, s_2, \ldots, s_n\}$ in the plane.

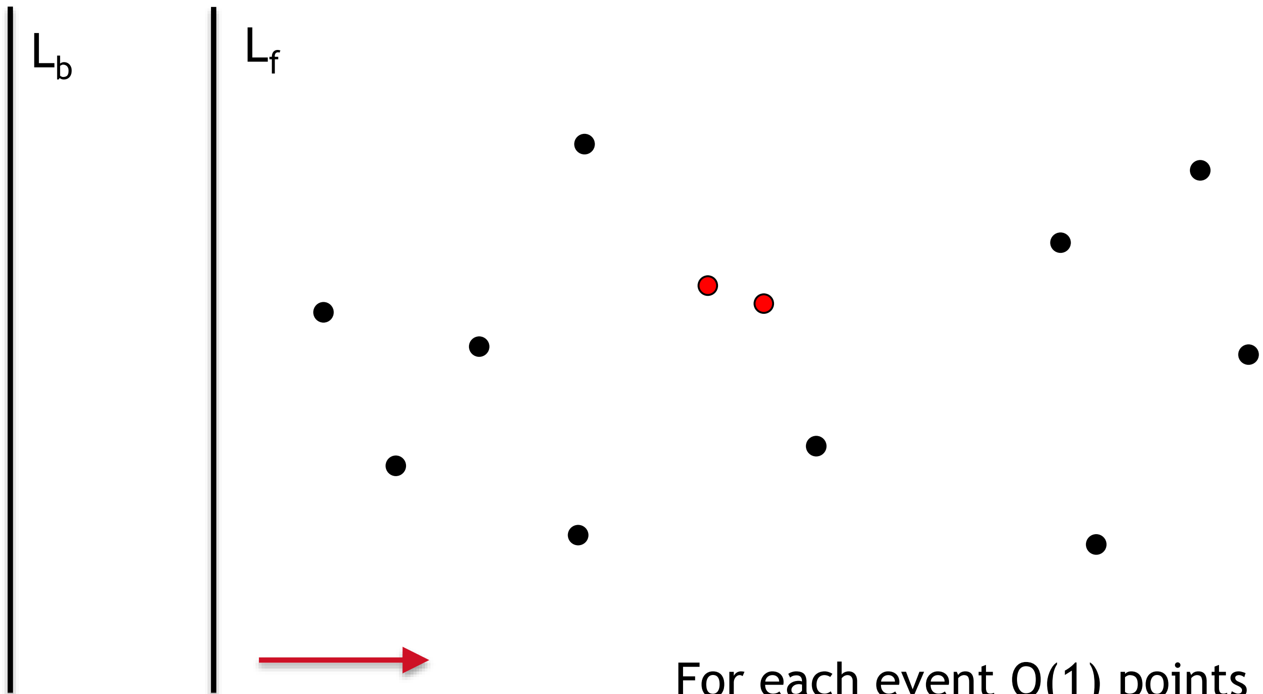
Aim: Report the closest pair in S.



Closest Pair

The sweepline algorithm utilies two sweeplines

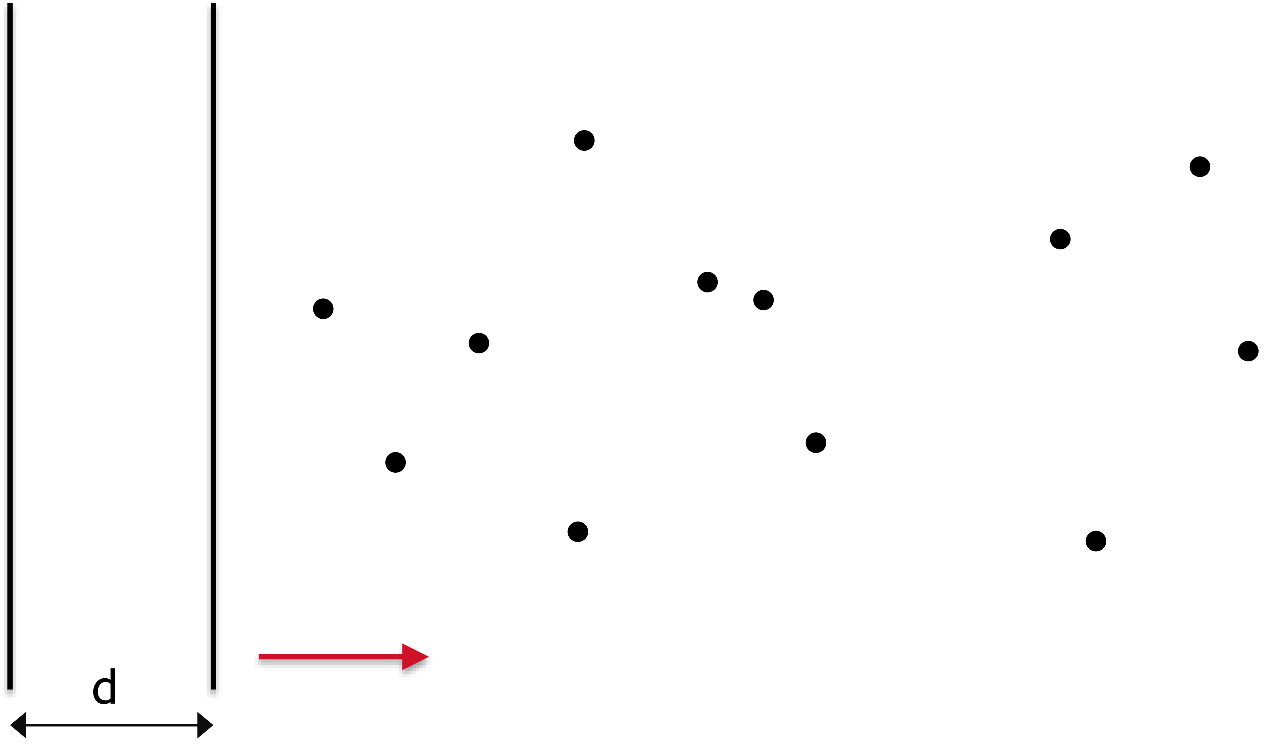
Idea: We will use two parallel vertical sweep-lines: the front L_f and the back L_b.



For each event O(1) points are checked

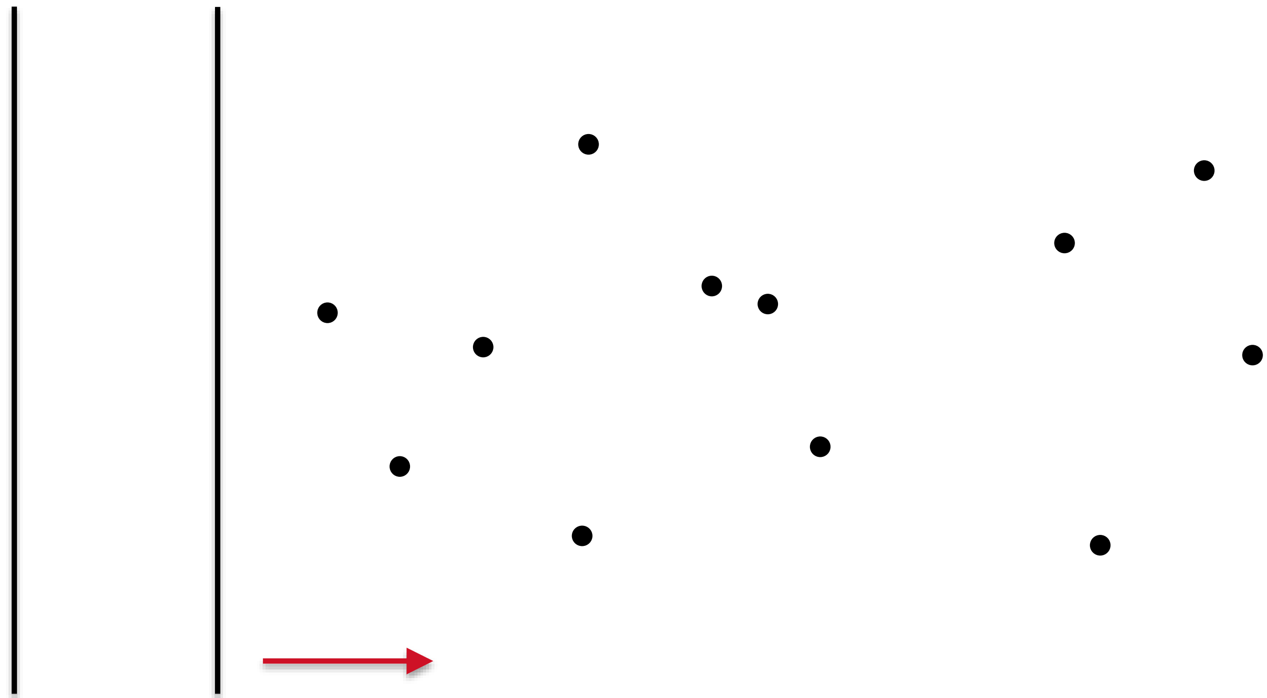
Closest Pair

Invariant 1: The closest pair among the points to the left of L_f, and the distance d between this pair.



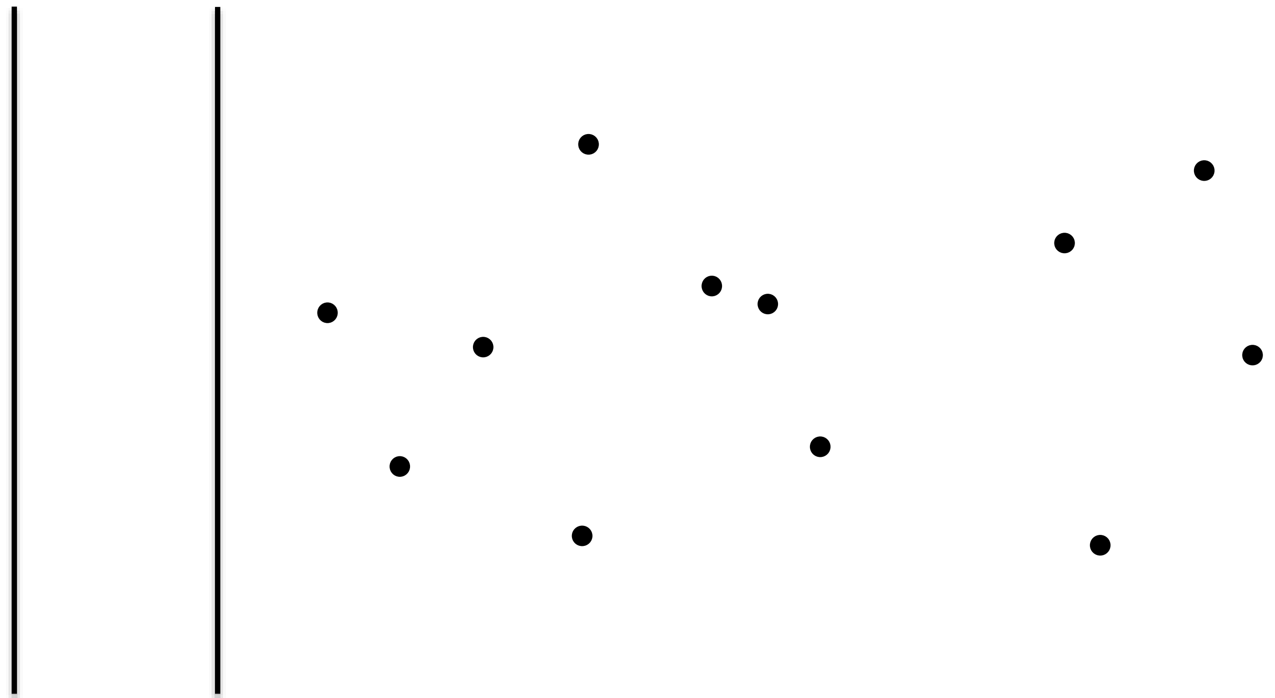
Closest Pair

Invariant 2: A balanced binary search tree T storing all the points in S between L_f and L_b ordered from top to bottom. (status structure)



Closest Pair

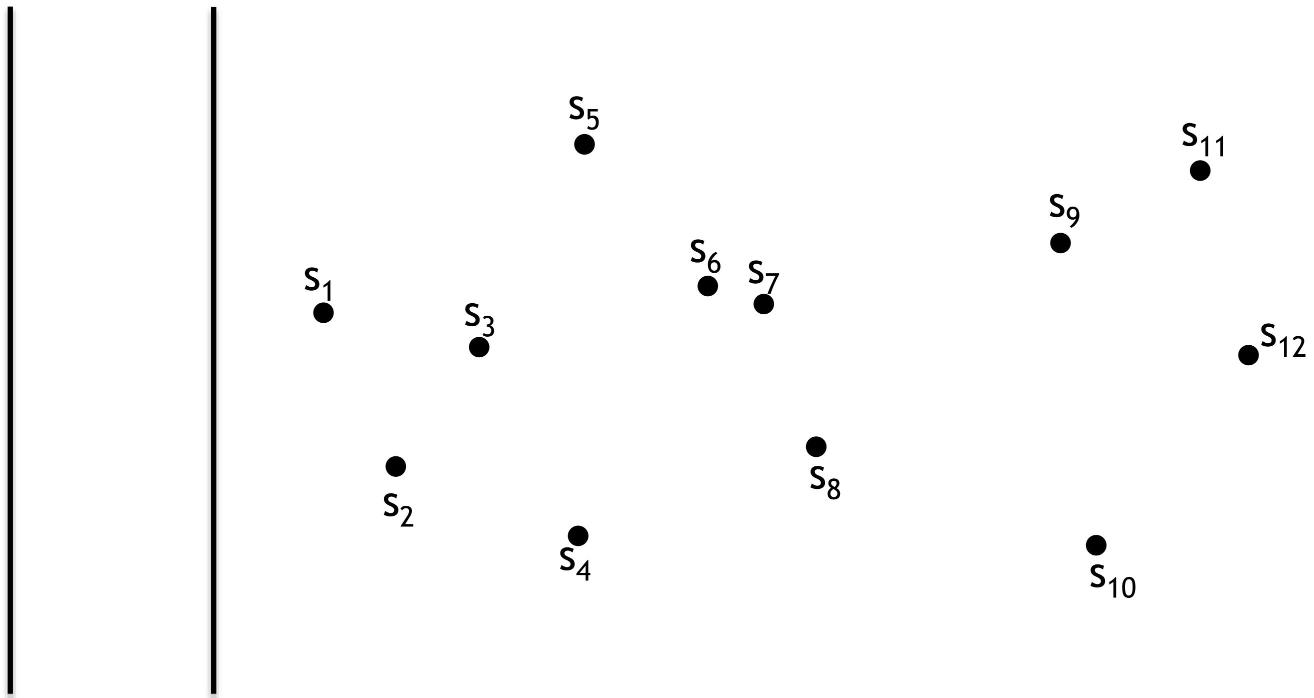
Initialise: T=∅ and d=∞.



Closest Pair

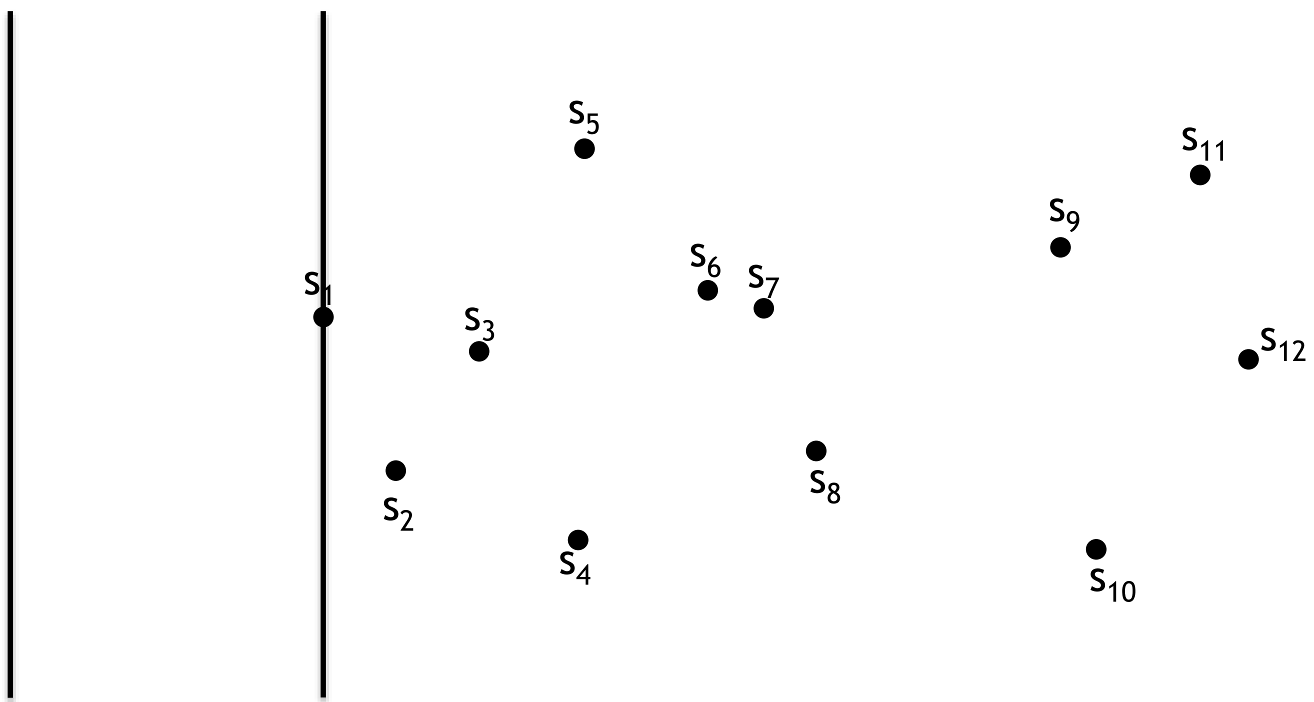
T= $\varnothing$

d= ∞



Closest Pair

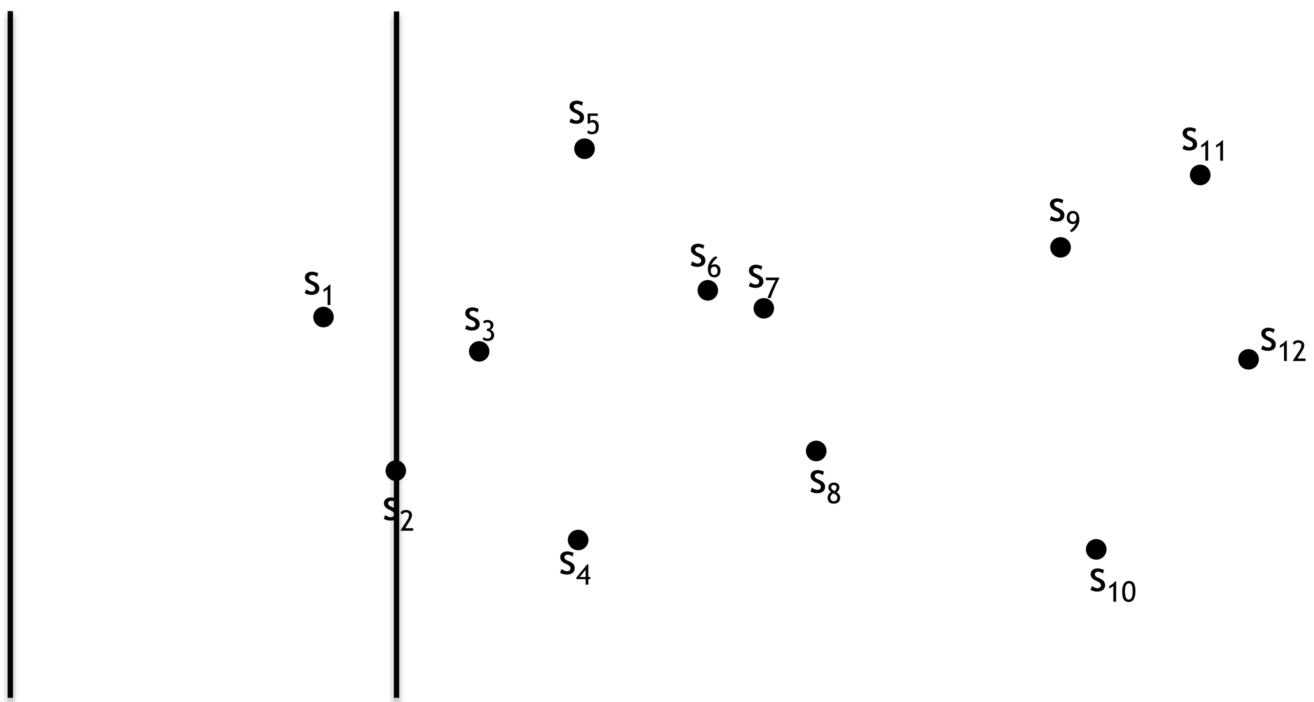
T= s_1
d= ∞



Closest Pair

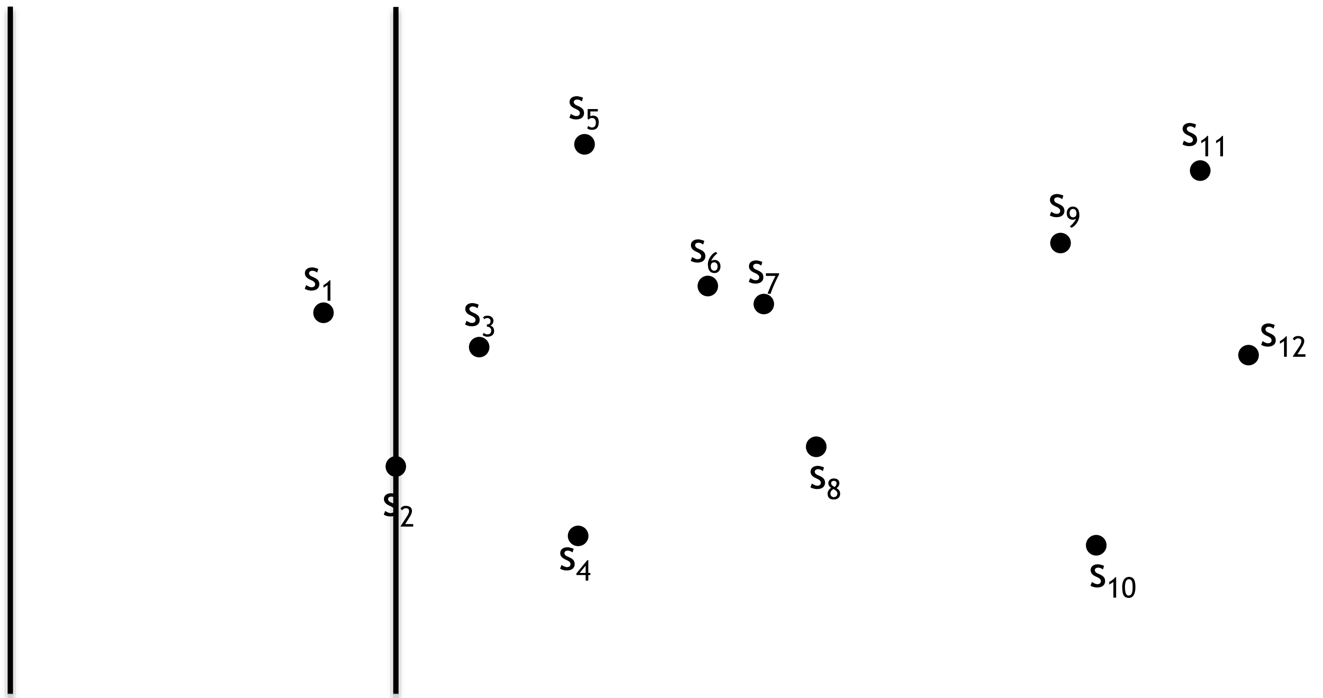
T= s_1 s_2

d= ∞



Closest Pair

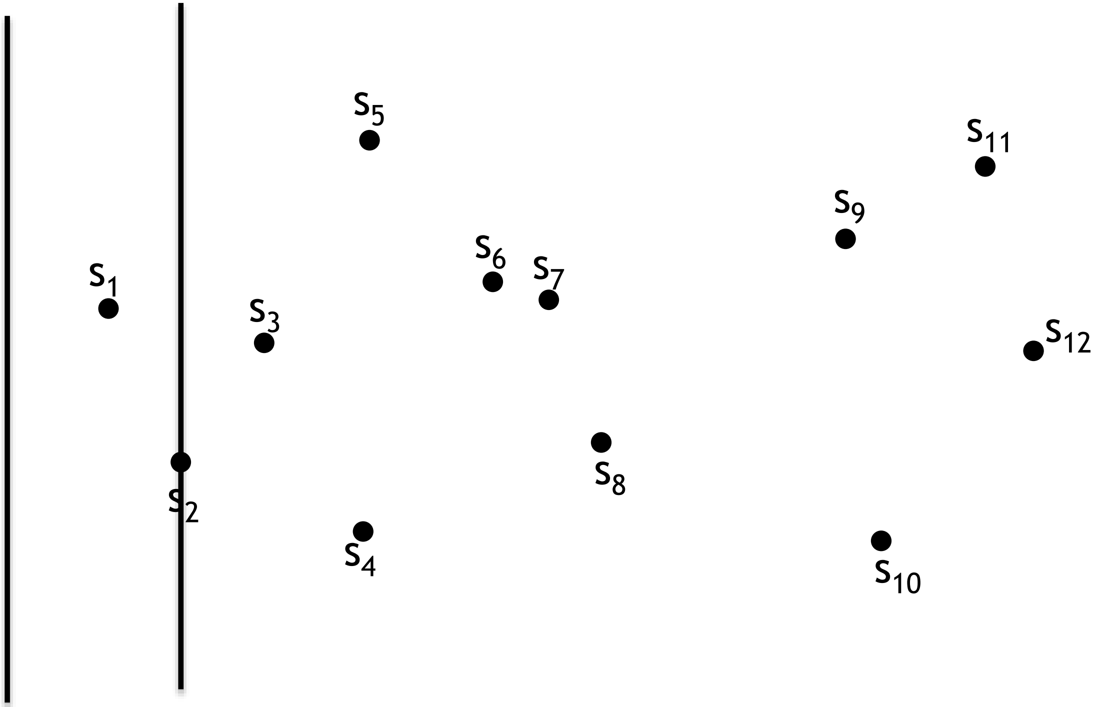
T= $s_1\ s_2$
d= $|s_1 s_2|$



Closest Pair

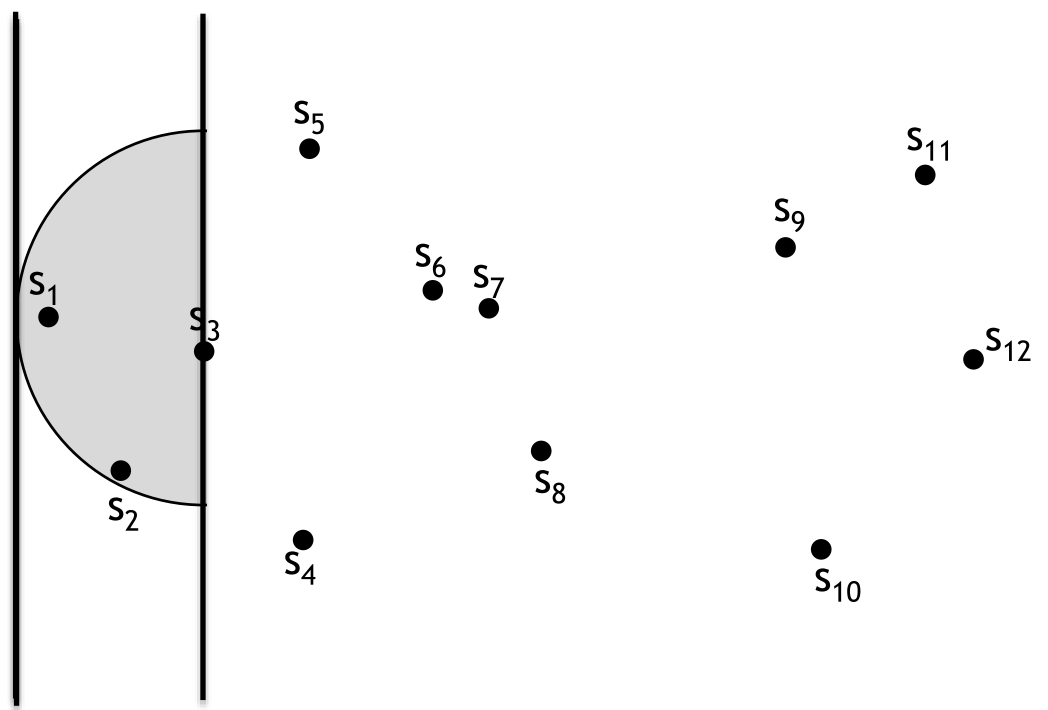
T= s_1 s_2

d= $|s_1 s_2|$



Closest Pair

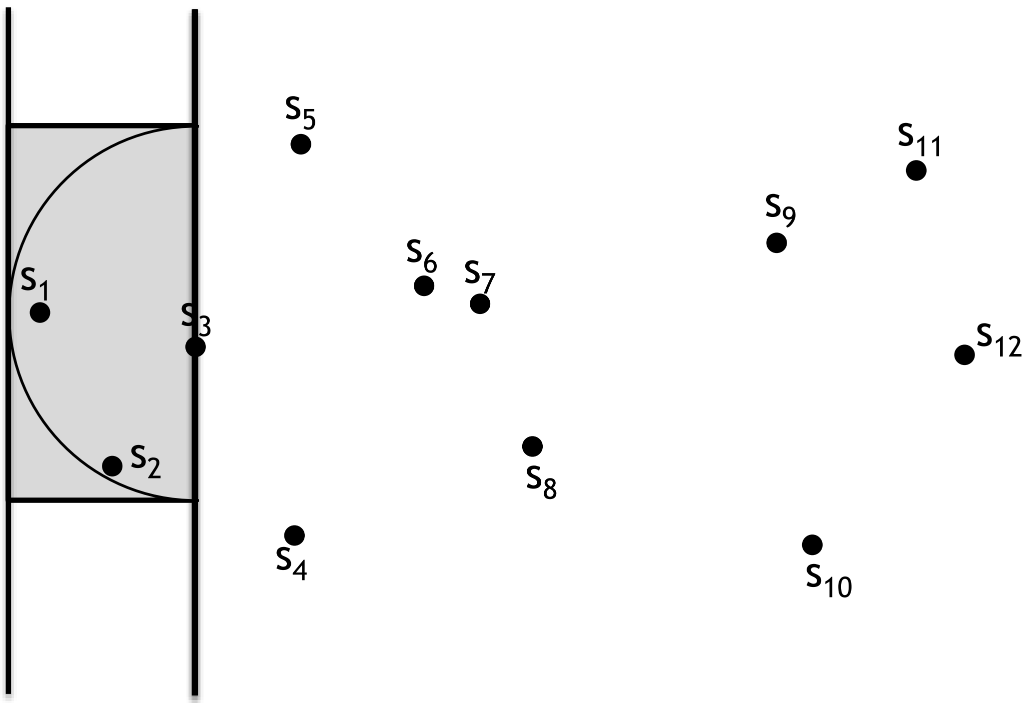
T= s_1 s_2
d= $|s_1 s_2|$



Closest Pair

T= s_1 s_2

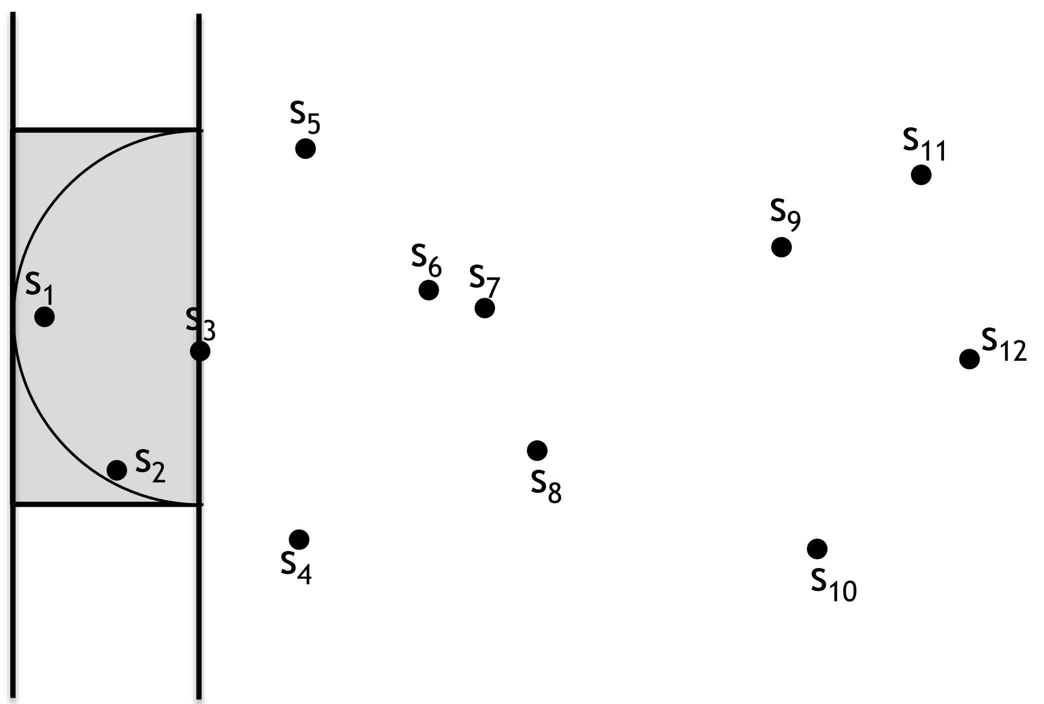
d= $|s_1 s_2|$



Closest Pair

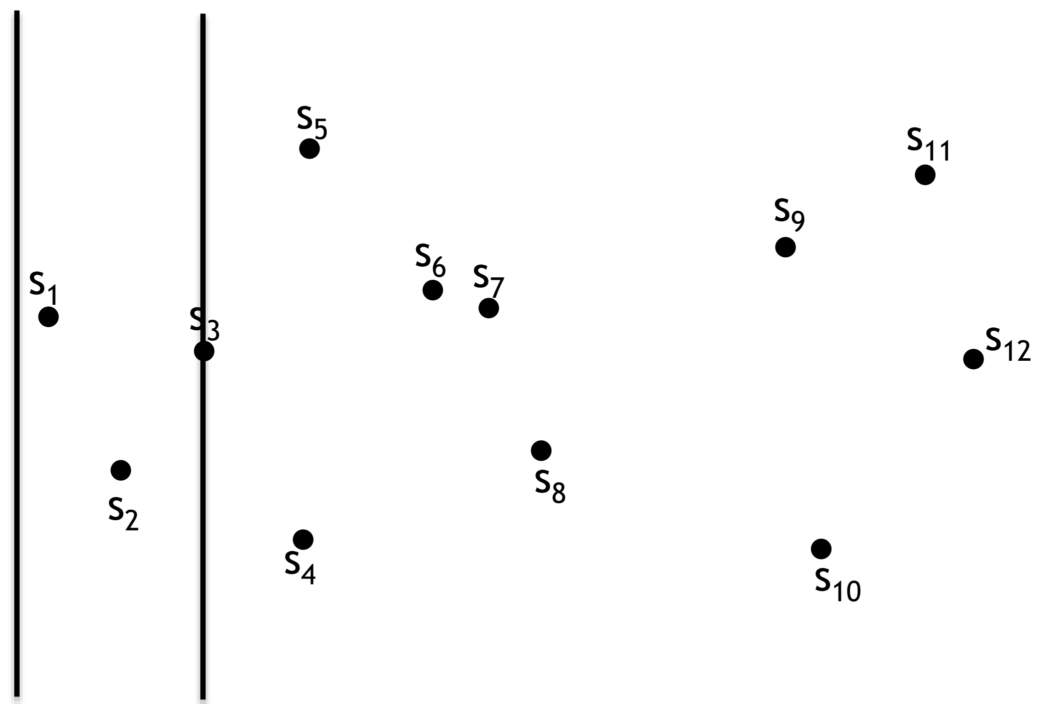
$T = s_1\ s_2$
$d = |s_1 s_2|$

Check all points in T within vertical distance d of s_3.



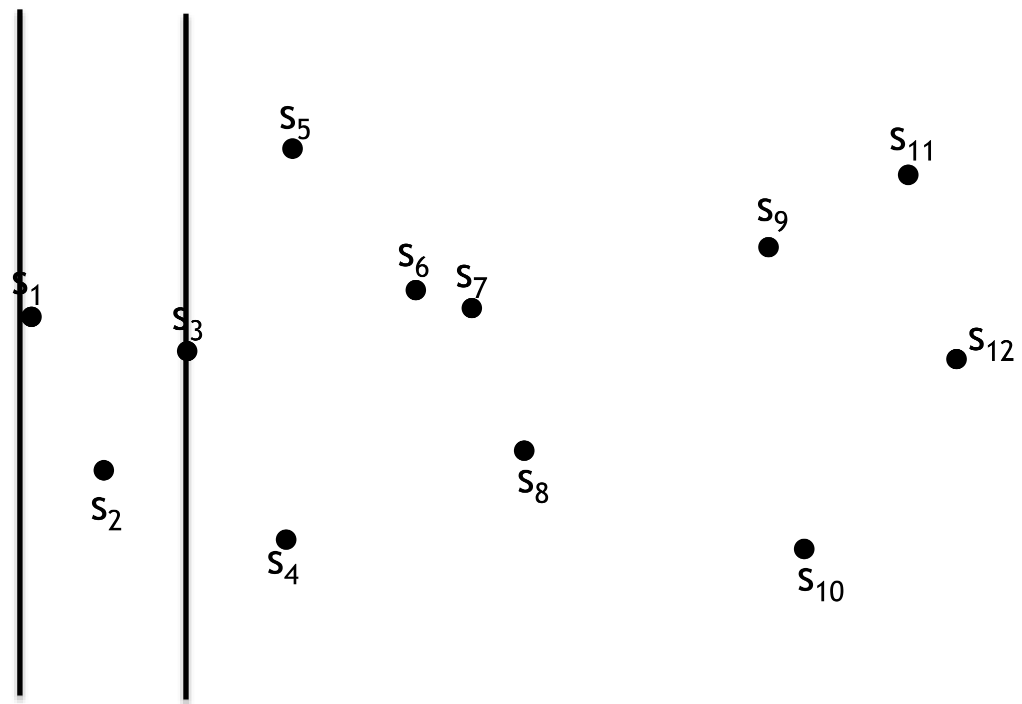
Closest Pair

$T = s_1\ s_2\ s_3$
$d = |s_2 s_3|$



Closest Pair

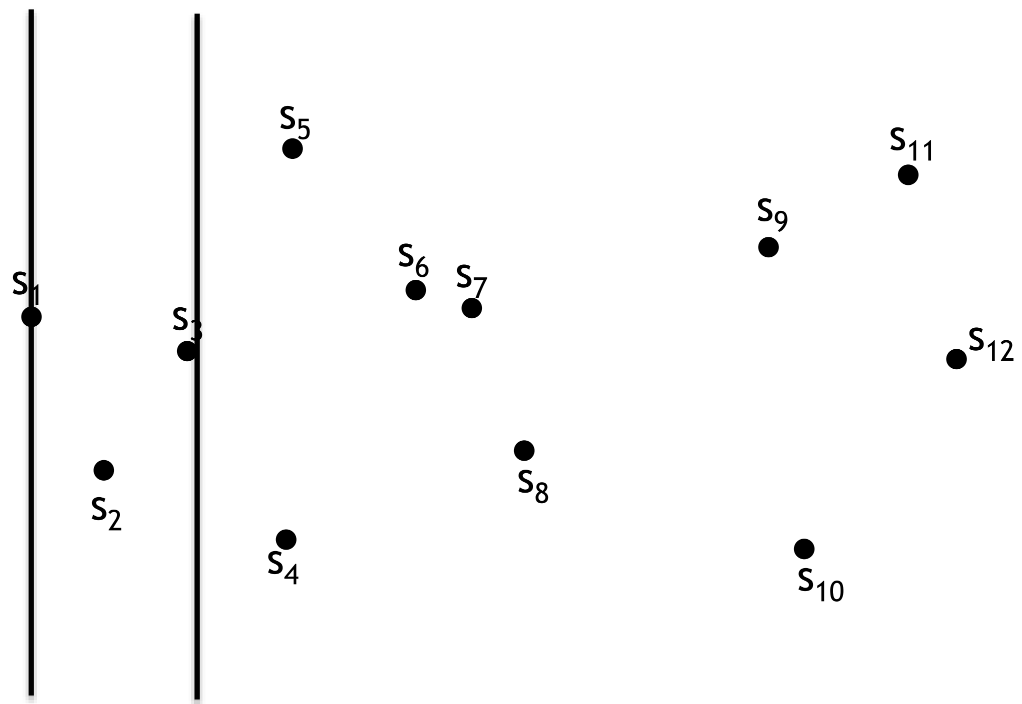
T= s_2 s_3
d= $|s_2 s_3|$



Closest Pair

$T = s_3$

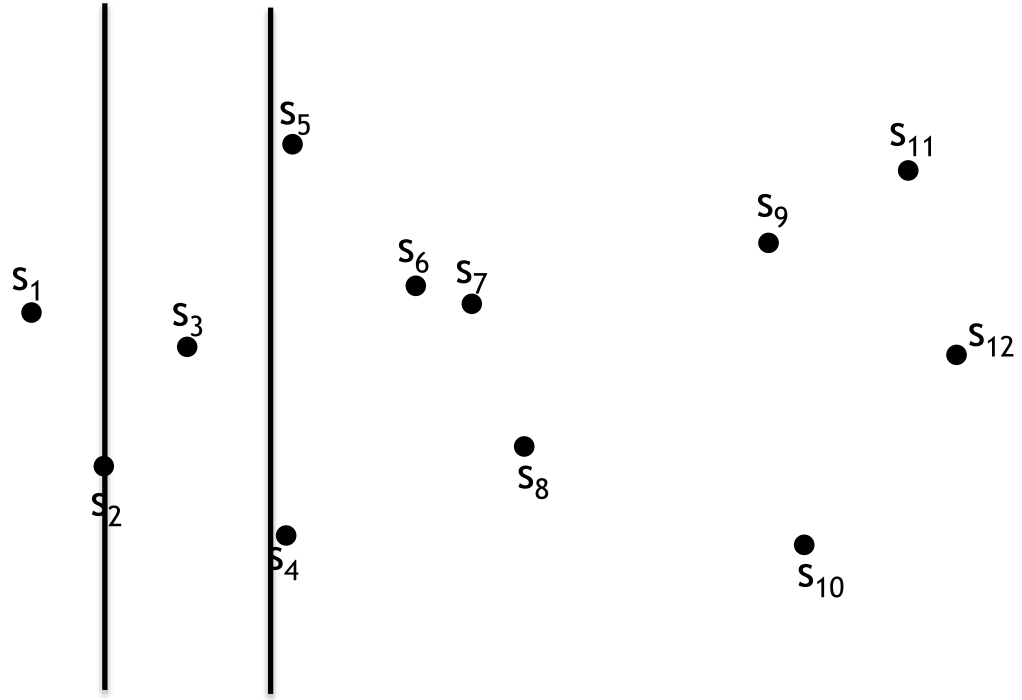
$d = |s_2 s_3|$



Closest Pair

T= s_3 s_4

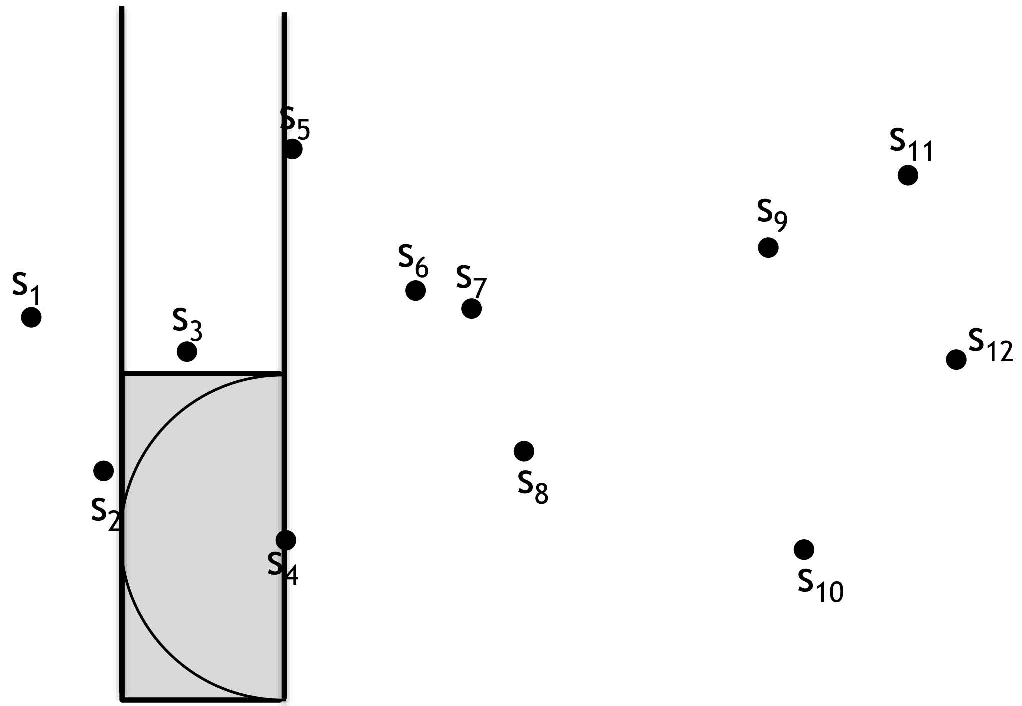
d= $|s_2s_3|$



Closest Pair

$T = s_3\ s_4$

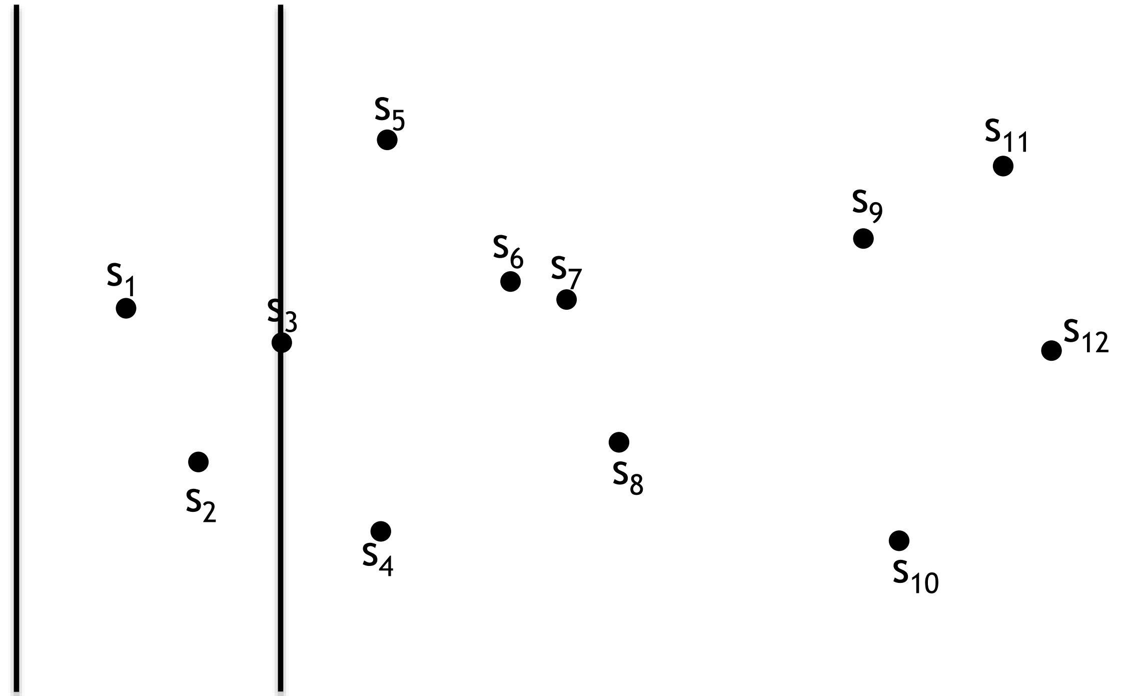
$d = |s_2 s_3|$



Two events:

1. L_f encounters a point s

2. L_b encounters a point s



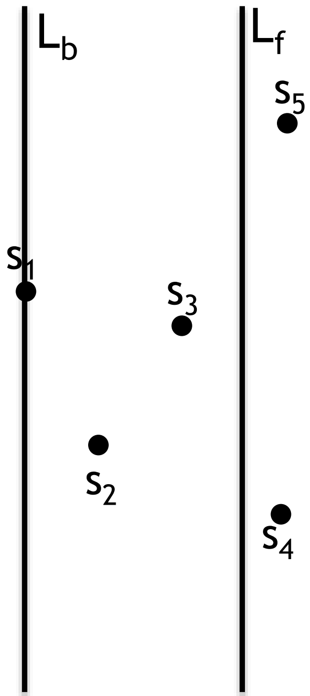
Closest Pair

Two events:

1. L_f encounters a point s

2. L_b encounters a point s

Remove s from T.



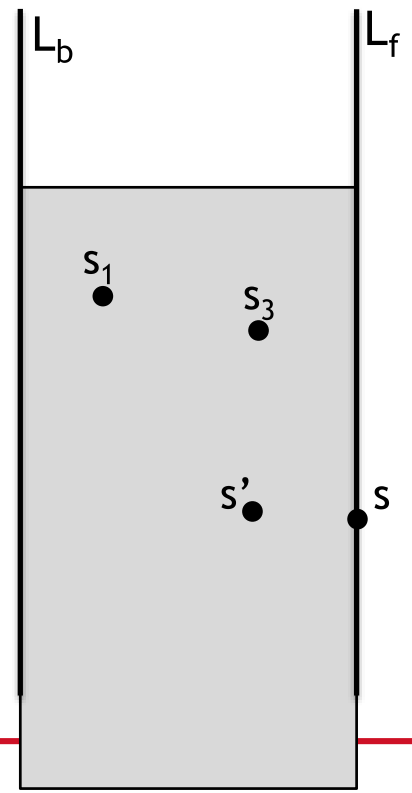
Closest Pair

Two events:

1. L_f encounters a point s

2. L_b encounters a point s

a. Find the point s' closest to s in-between L_b and L_f within vertical distance d from s.
b. If |ss'| < d then
   i. set d = |ss'|
   ii. CP=(s,s')
   iii. Sweep L_b and update T
c. Insert s into T.



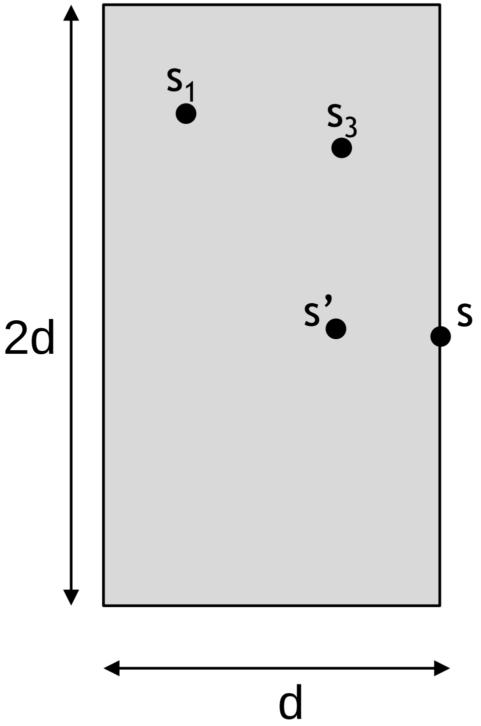
Running time?

Closest Pair

a. Find the point s' closest to s in-between L_b and L_f within vertical distance d from s.

Recall: Our search is constrained to the bounding box of size $d \times 2d$.

Question: How many points can there be in the bounding box?

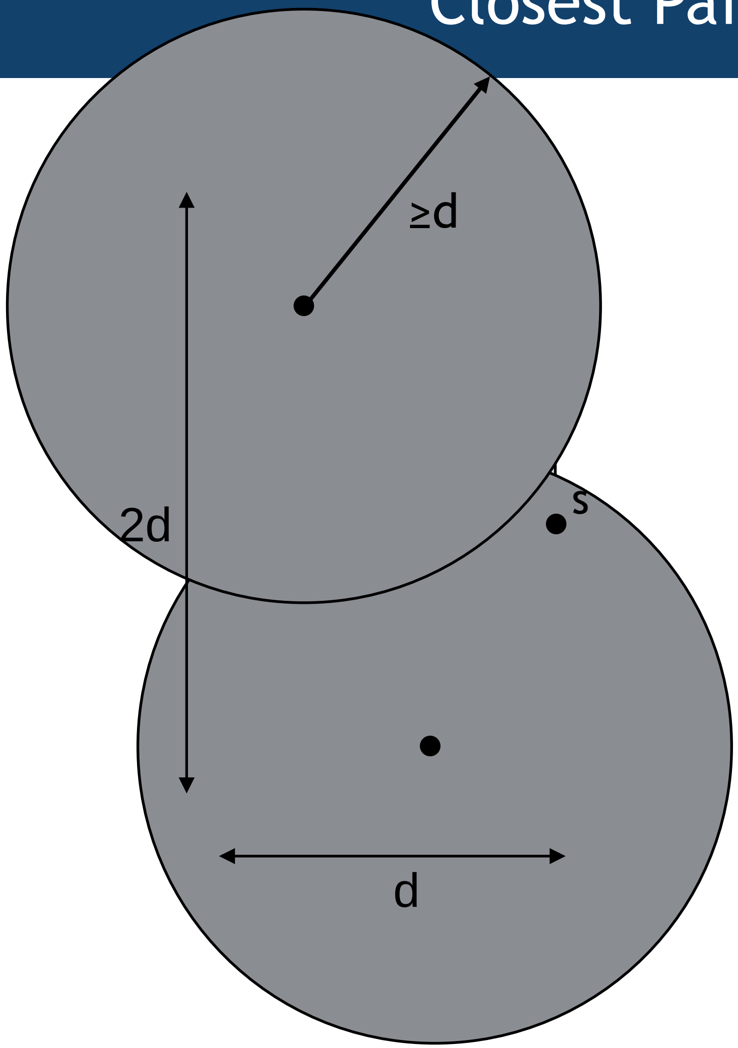


Closest Pair

a. Find the point s' closest to s in-between L_b and L_f within vertical distance d from s.

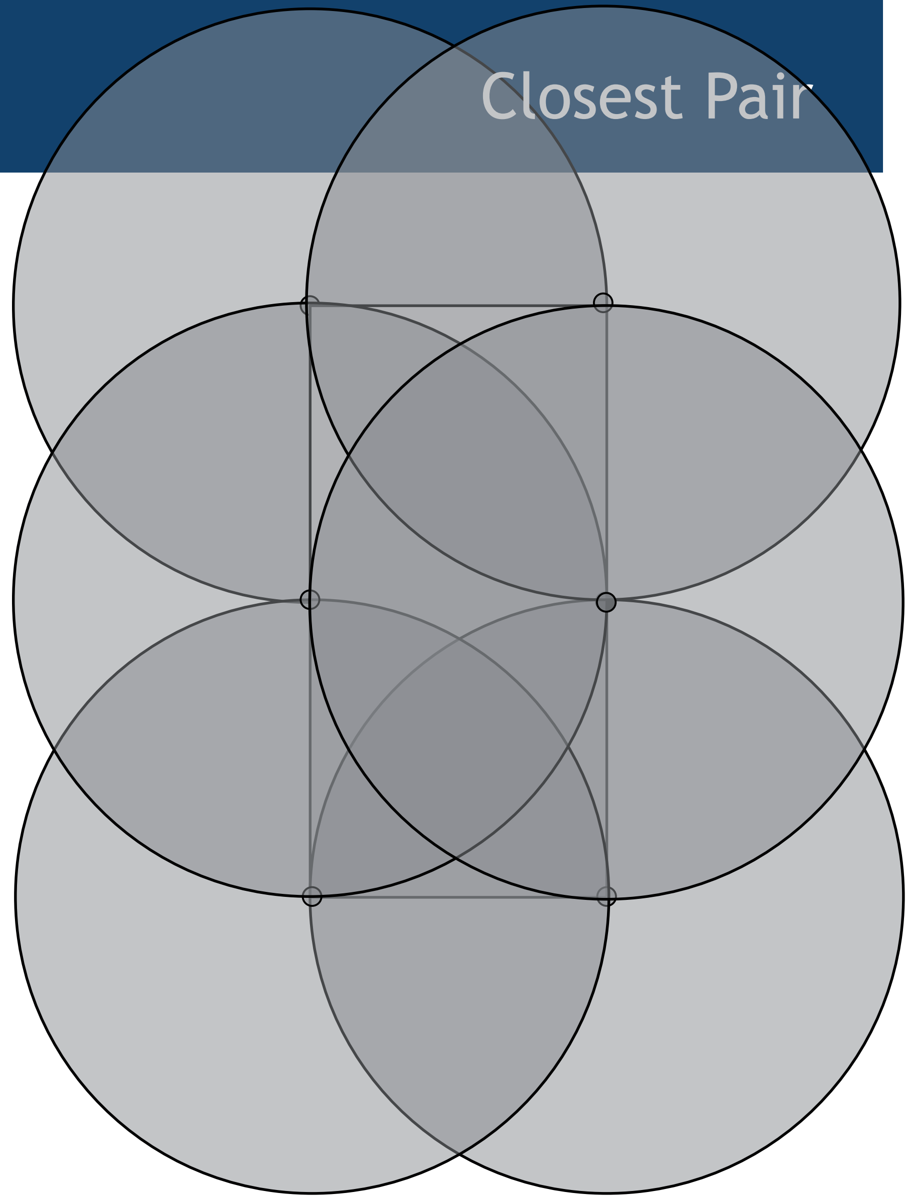
Recall: Our search is constrained to the bounding box of size $d \times 2d$.

Question: How many points can there be in the bounding box?



a. Find the point s' closest to s in-between L_b and L_f.

Recall: Our search is constrained to the bounding box of size $d \times 2d$.



Question: How many points can there be in the bounding box?

Answer: 6

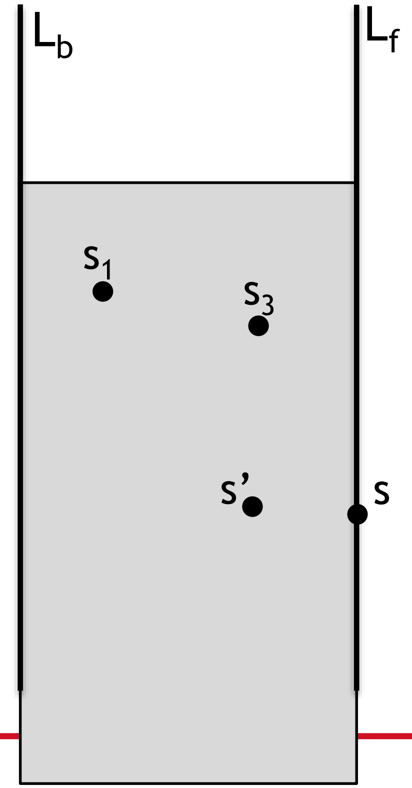
Closest Pair

Two events:

1. L_f encounters a point s

2. L_b encounters a point s

   a. Find the point s' closest to s in-between L_b and L_f within vertical distance d from s.
   b. If |ss'| < d then
      i. set d = |ss'|
      ii. CP=(s,s')
      iii. Sweep L_b and update T
   c. Insert s into T.



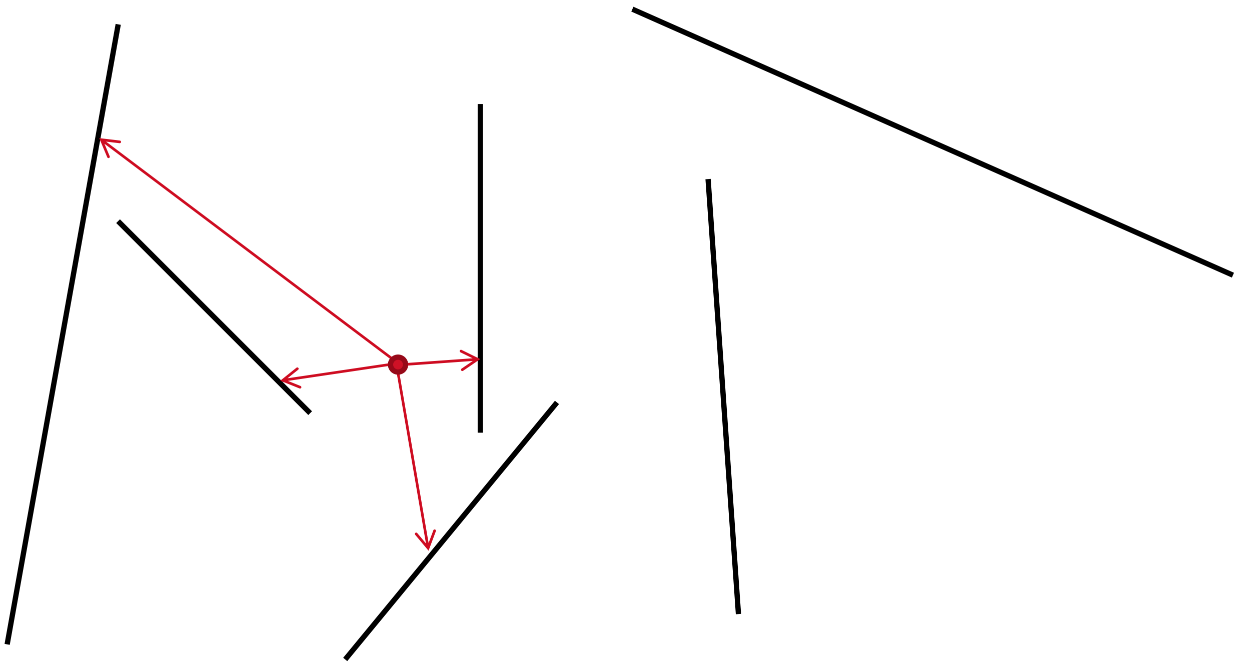
Running time?
O(n log n)

Closest Pair

Theorem:

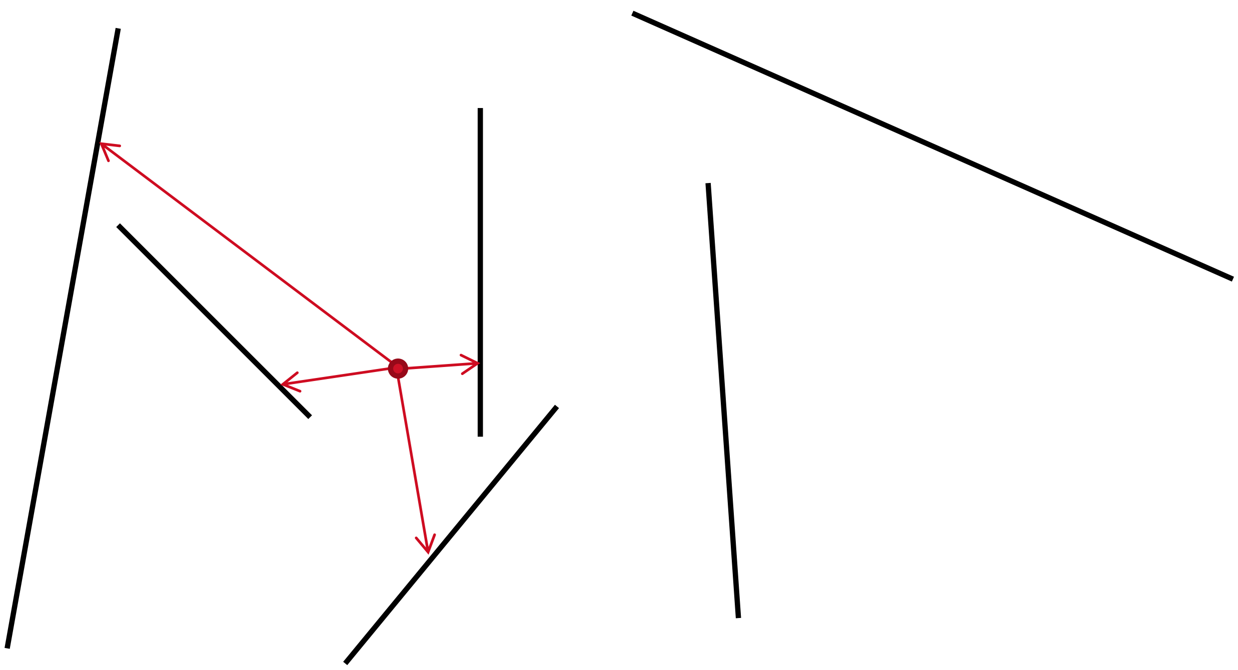
Given a set S of n points in the plane, the closest pair in S can be computed in O(n log n) time.

Visibility



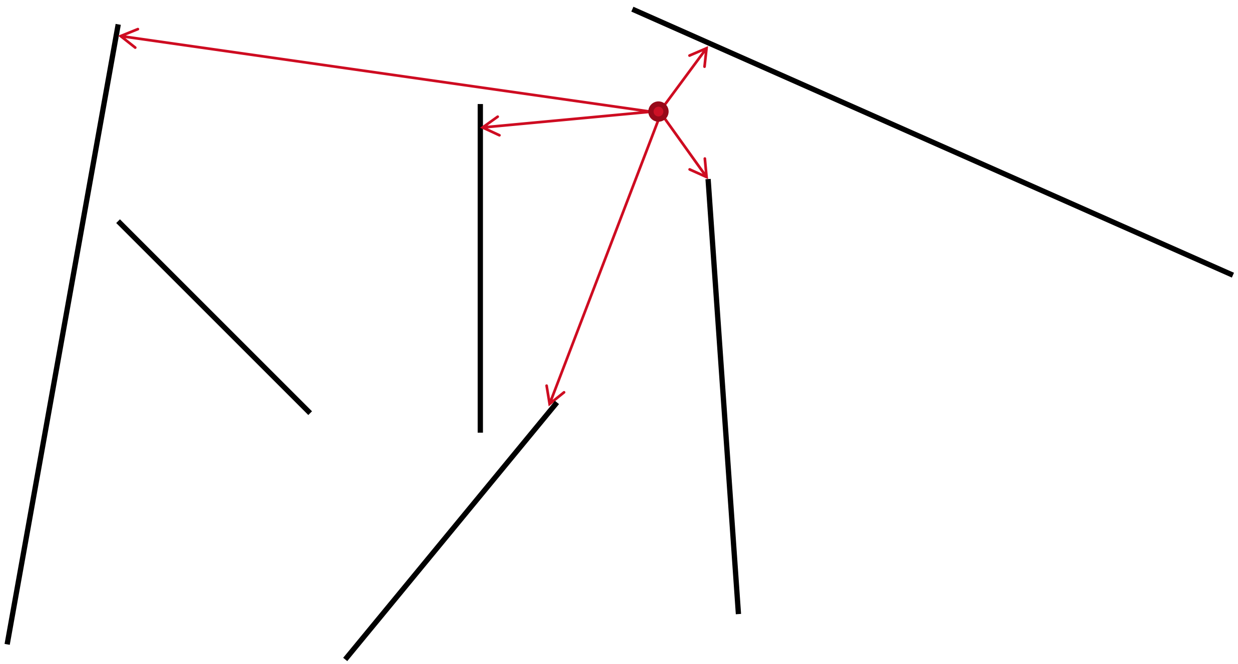
Another sweepline algorithm

Problem: Let S be a set of n disjoint line segments in the plane, and let p be a point not on any line segment of S. Determine all line segments of S that p can see



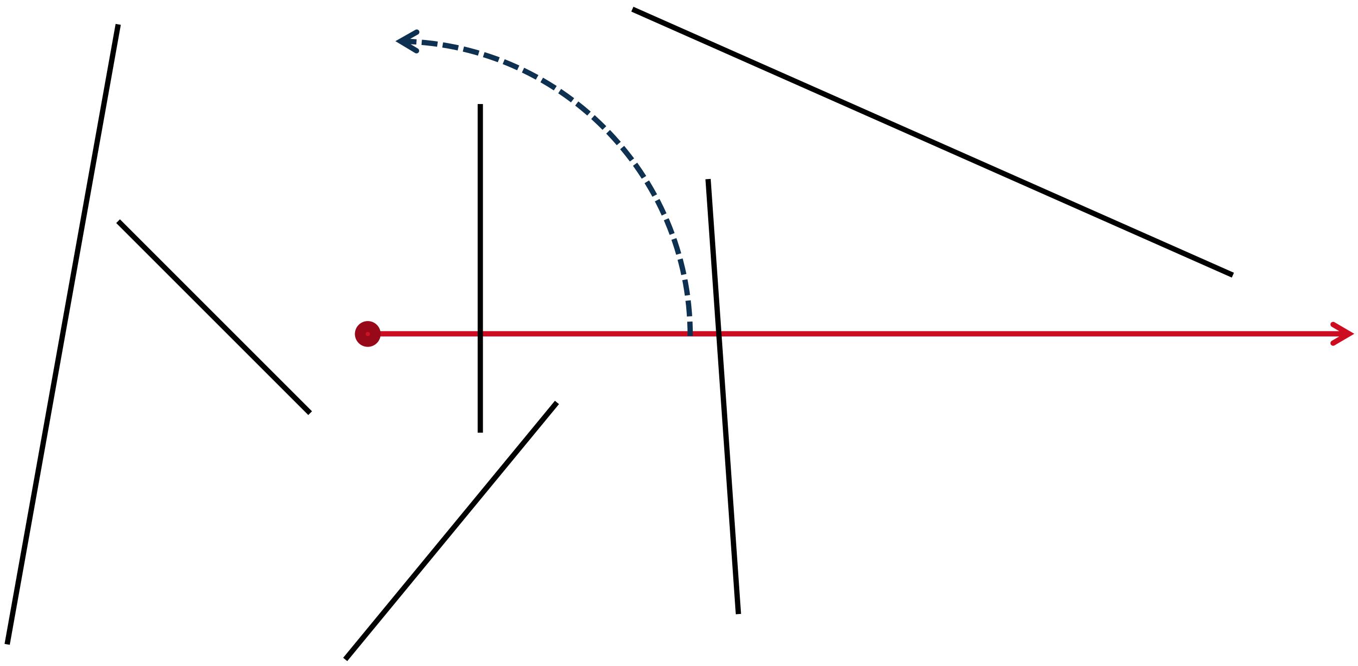
Another sweepline algorithm

Problem: Let S be a set of n disjoint line segments in the plane, and let p be a point not on any line segment of S. Determine all line segments of S that p can see



Another sweepline algorithm

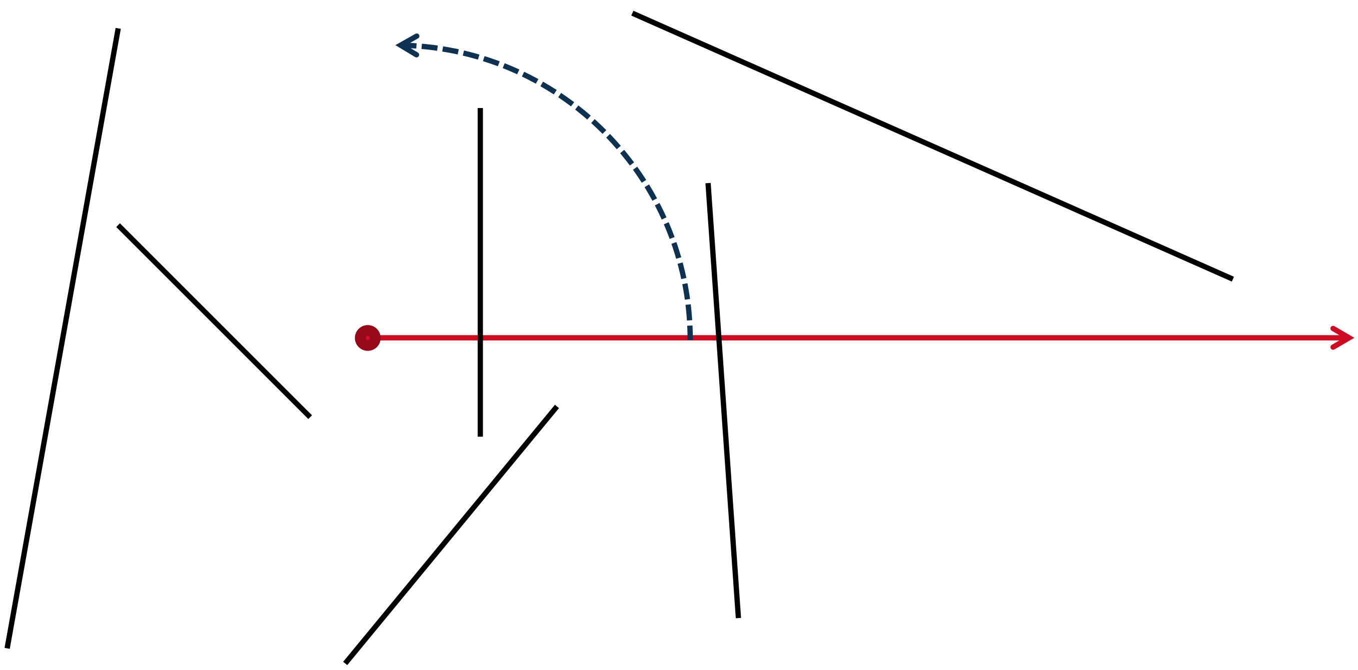
Use sweepline approach. How?



Another sweepline algorithm

Event points?

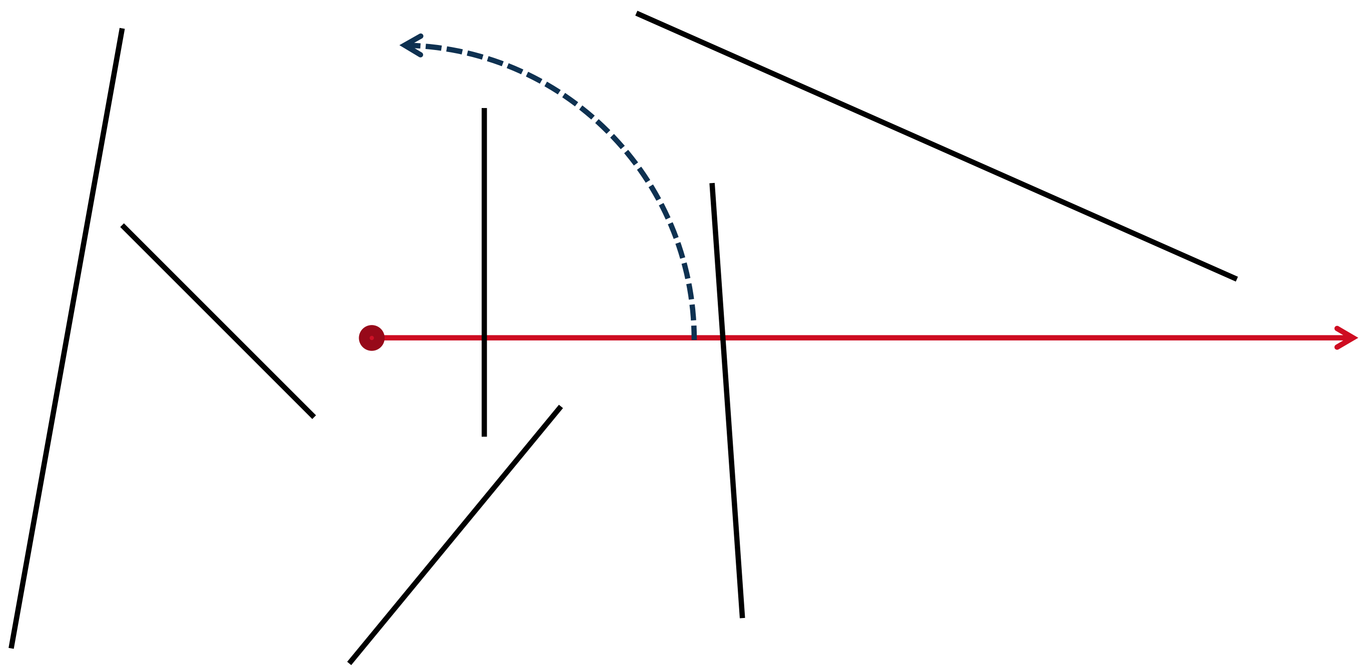
Consider a radial sweep. When can q see a new segment?



Another sweepline algorithm

Event points? Endpoints of segments

What do we want to keep track of during the sweep?



Another sweepline algorithm

Event points? Endpoints of segments

What do we want to keep track of during the sweep?

The segment q sees in that direction (which one is that?)

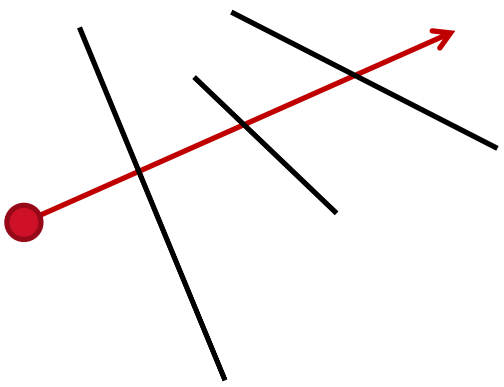
Another sweepline algorithm

Event points? Endpoints of segments

What do we want to keep track of during the sweep?

The segment q sees in that direction (which one is that?)

The order of the segments along the ray?



Another sweepline algorithm

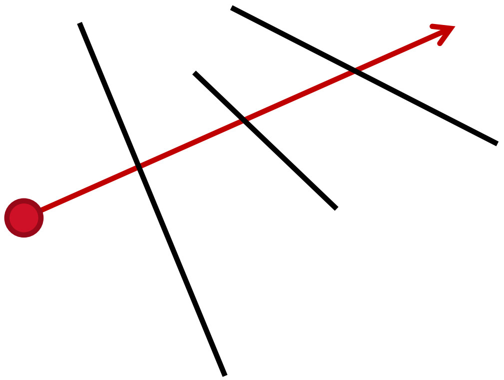
Event points? Endpoints of segments

What do we want to keep track of during the sweep?

The segment q sees in that direction (which one is that?)

The order of the segments along the ray?

Invariant?



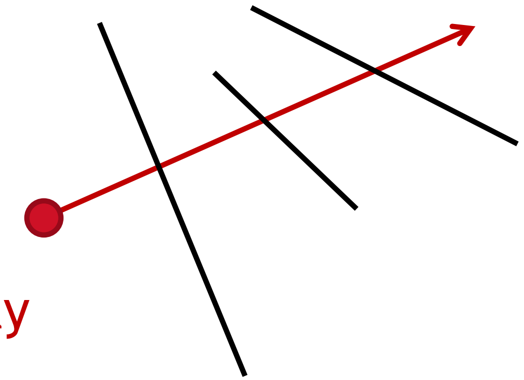
Another sweepline algorithm

Event points? Endpoints of segments

What do we want to keep track of during the sweep?

The segment q sees in that direction (which one is that?)

The order of the segments along the ray?



Invariant?

The segments seen so far.

The order of the segments intersecting the ray

Another sweepline algorithm

Event points? Endpoints of segments

What do we want to keep track of during the sweep?

- The segment q sees in that direction (which one is that?)
- The order of the segments along the ray?

Invariant?

- The segments seen so far.
- The order of the segments intersecting the ray

Handle event?

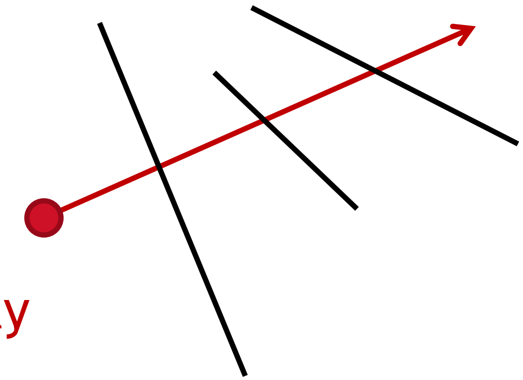
Another sweepline algorithm

Event points? Endpoints of segments

What do we want to keep track of during the sweep?

The segment q sees in that direction (which one is that?)

The order of the segments along the ray?



Invariant?

The segments seen so far.

The order of the segments intersecting the ray

Handle event?

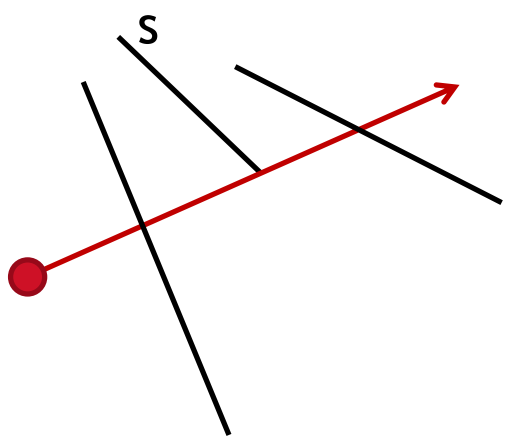
Two cases:

first endpoint of segment

last endpoint of segment

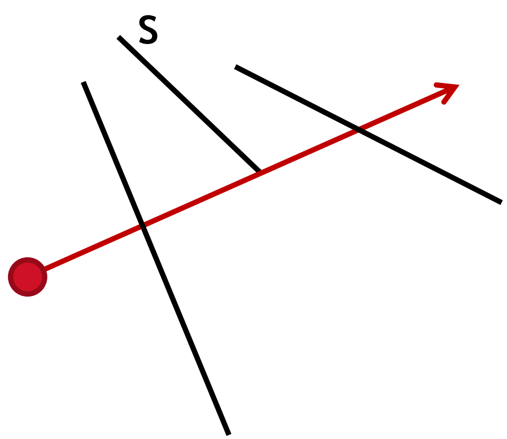
Another sweepline algorithm

First endpoint: Assume order of segment along the ray is stored in a data structure D.



Another sweepline algorithm

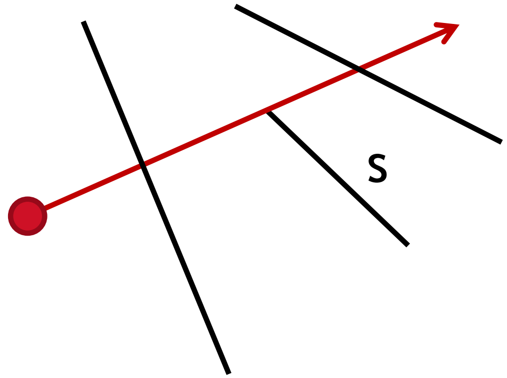
First endpoint: Assume order of segment along the ray is stored in a data structure D.



1. Insert new segment s in D.
2. If s is the first segment hit by the ray then report s.

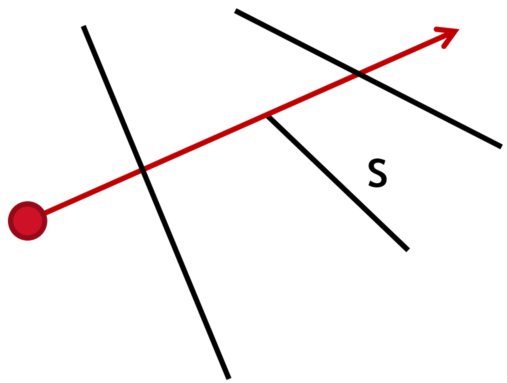
Another sweepline algorithm

Second endpoint: Assume order of segment along the ray is stored in a data structure D.



Another sweepline algorithm

Second endpoint: Assume order of segment along the ray is stored in a data structure D.



1. Remove s from D
2. If s was the first segment in D report new first segment in D.

Another sweepline algorithm

Time complexity?

Number of event points?

Another sweepline algorithm

Time complexity?

Number of event points = 2n

Another sweepline algorithm

Time complexity?

Number of event points = 2n

Time to handle an event?

Which data structure should we use for D?

Another sweepline algorithm

Time complexity?

Number of event points = 2n

Time to handle an event?

Which data structure should we use for D?

binary search tree (insert/delete/query=O(log n))

Another sweepline algorithm

Time complexity?

Number of event points = 2n

Time to handle an event?

Which data structure should we use for D?

binary search tree (insert/delete/query=O(log n))

First endpoint: insert + query = O(log n)

Second endpoint: delete + query = O(log n)

Another sweepline algorithm

Time complexity?

Number of event points = 2n

Time to handle an event?

Which data structure should we use for D?

binary search tree (insert/delete/query=O(log n)

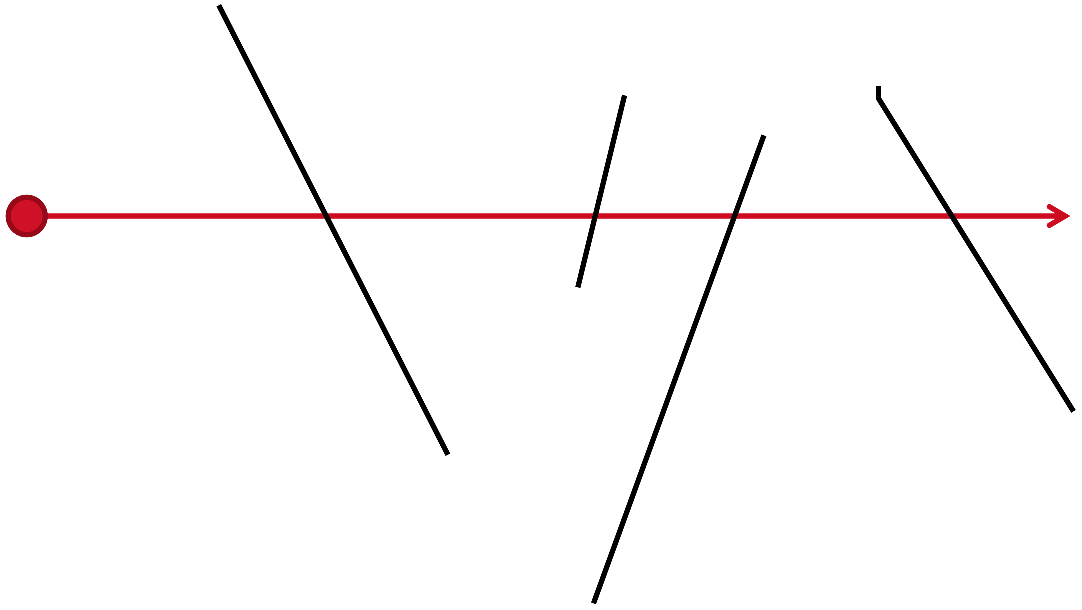
First endpoint: insert + query = O(log n)

Second endpoint: delete + query = O(log n)

Total: O(n log n)

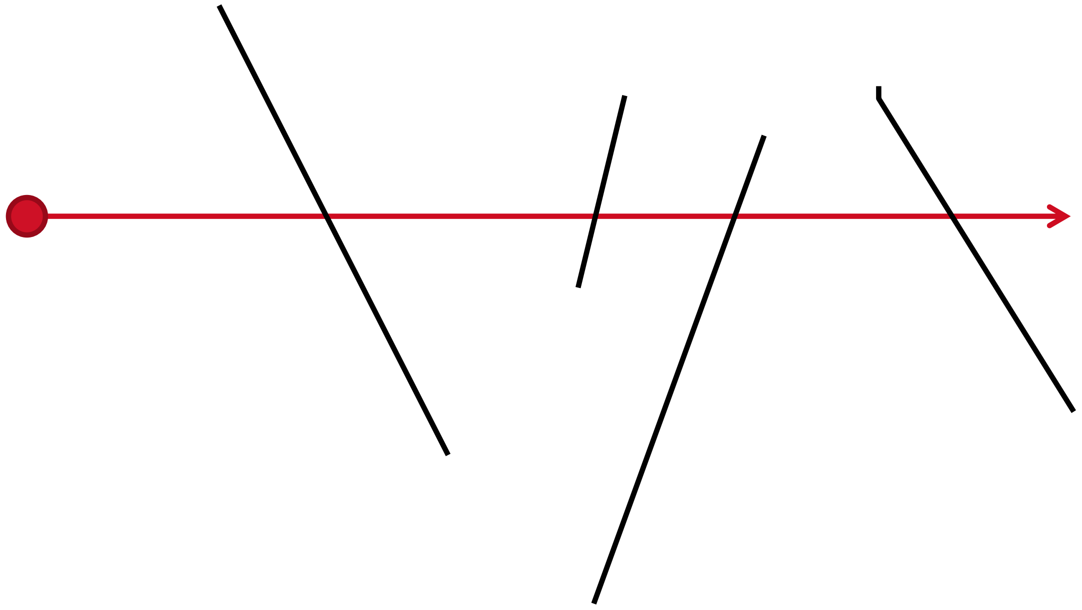
Another sweepline algorithm

What about initialization? How do we start the sweep?



Another sweepline algorithm

What about initialization? How do we start the sweep?



Find all segments intersecting the ray. Sort wrt distance from q.

Time: O(n log n)

Reading list

- Segment intersection
  http://www.cs.tufts.edu/comp/163/notes05/seg_intersection_handout.pdf

- Closest pair
  http://www.cs.mcgill.ca/~cs251/ClosestPair/ClosestPairPS.html

- Convex hull
  https://www.cs.duke.edu/courses/fall08/cps230/Lectures/L-20.pdf
  and Section 1.6 in
  http://jeffe.cs.illinois.edu/teaching/compgeom/notes/01-convexhull.pdf