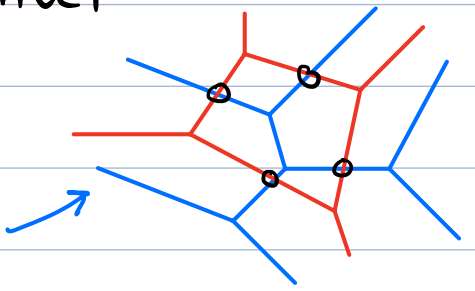
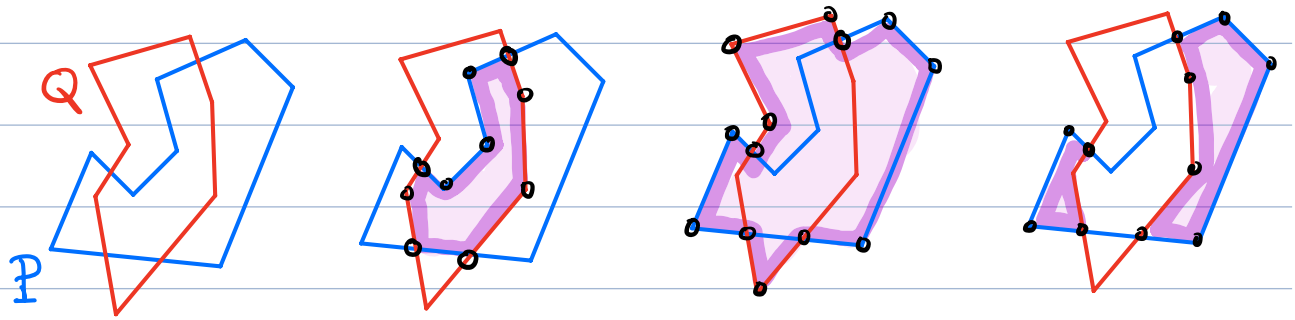
CMSC 754 - Computational Geometry
Lecture 4: Line Segment Intersection

Computing Intersections is fundamental to geometric computations.

- collision detection
- subdivision overlay
- Boolean operations on shapes - $\cup, \cap, \setminus, \ldots$





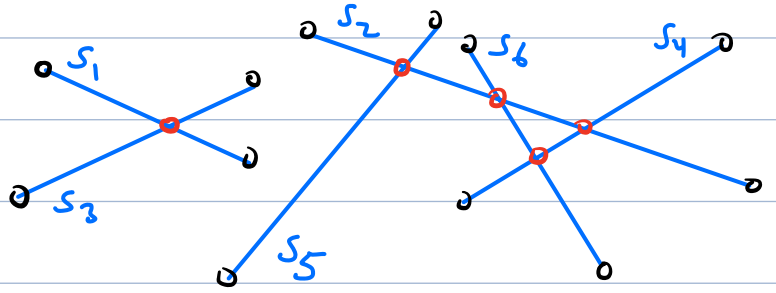
$P \cap Q$ $P \cup Q$ $P \setminus Q$

set subtraction

Line Segment Intersection:

Given a set $S = \{s_1, \ldots, s_n\}$ of line segments in $\mathbb{R}^2$ where $s_i = \overline{p_i q_i}$, report all intersecting pairs.

Example:

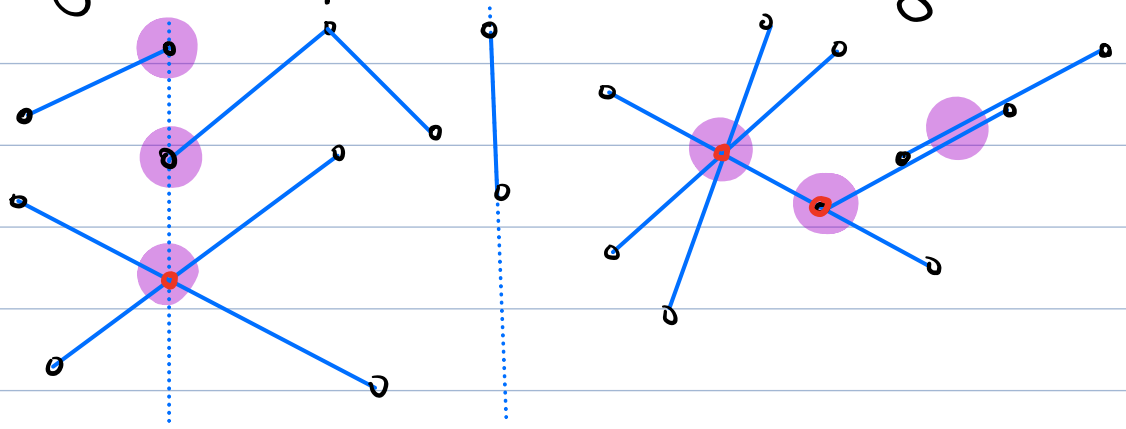


$\{(1,3), (2,5)$
$(2,6), (4,6), (2,4)\}$

{order doesn't matter

General Position Assumptions:

- No duplicate x-coordinates (endpoints or intersection points)
- No segment endpoint lies on another segment



Output Sensitivity: We express running time an func. of...

Input size n - (n segs, $2n$ endpts, $4n$ coordinates)

Output size m

$$0 \le m \le \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$$

Lower bound: $\Omega(m + n \log n)$

- m: Output size
- $n \log n$: Follows from a lower bound on element uniqueness in algeb. dec. tree model

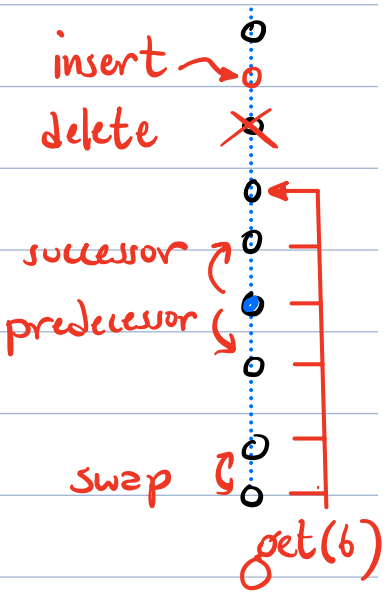
This lecture: Plane sweep algorithm

$O((n+m) \log n)$ time

suboptimal by $\log n$ factor on m

Utility Data Structures:

Ordered Dictionary: Supports insert, delete, find, predecessor, successor, get-k^{th}, + swap adjacent



All in $O(\log n)$ time + $O(n)$ space

Priority Queue - Stores object o + priority x

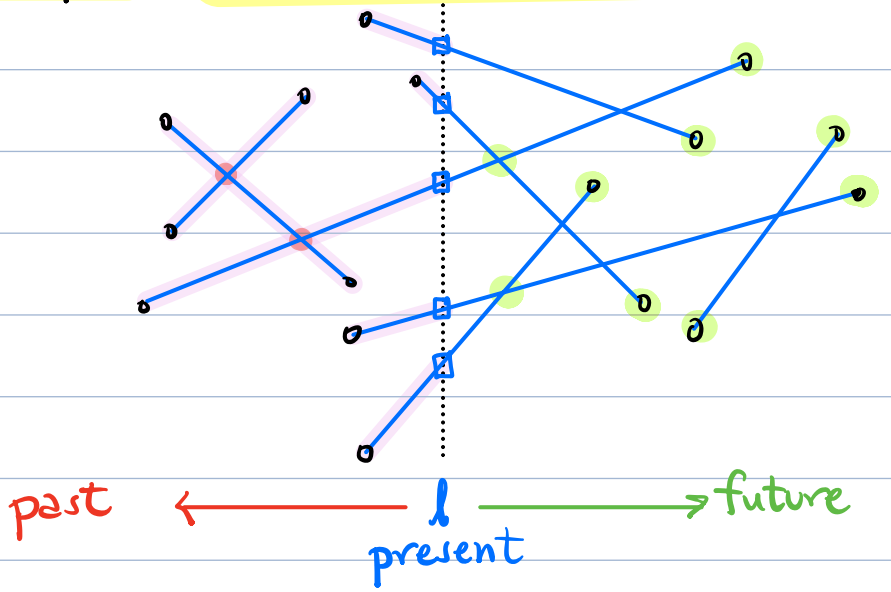
ref ← enqueue(o, x)

o ← extractMin() - Remove object with min priority

delete(ref)

Sweep-line Algorithm:

Sweep a vertical line l from left to right. Update output + internal data structures at each event

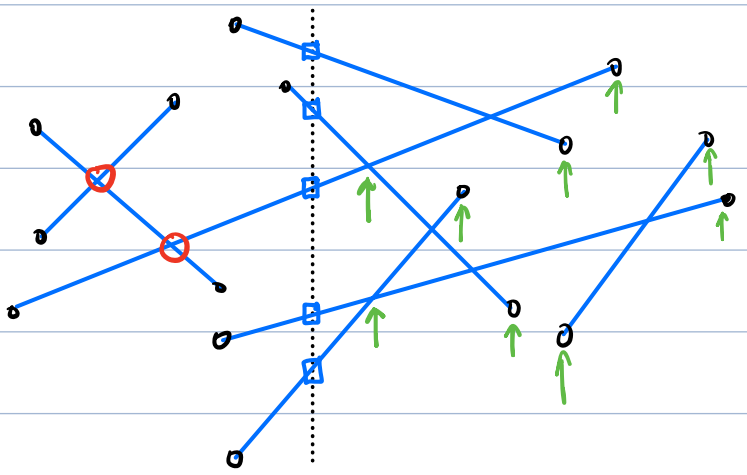


What we store:

Past - (Partial) solution to left of l

Present - Current "sweep-line status" on l

Future - (Known) Events to right of l



What we store (specifically for segment intersection)

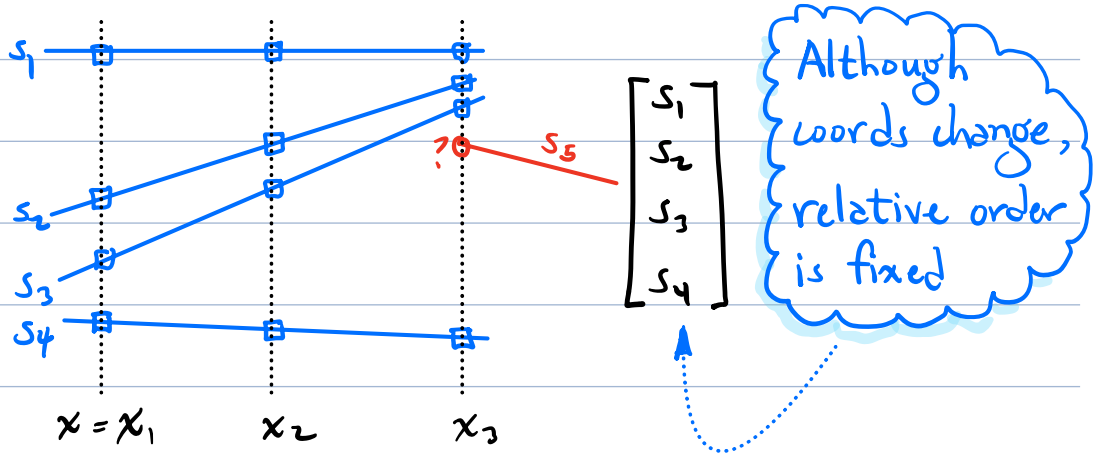
Past: List of intersecting pairs (so far)

Present: Segments that intersect l,
sorted (top to bottom, say)
Stored in ordered dictionary
called sweep-line status

Future: Priority queue with known future events
- Segment endpoints to right of l
- "Imminent" intersection pts to right of l
- Priority = x-coordinate of each event
- Number of imminent intersection events = $O(n)$

Sweep-line Status:

- As l moves, all coordinates of intersections on the sweep line change.
- Much too slow to update them all!



- Dynamic comparator - Rather than storing y-coords explicitly, store the equation for each line that intersects the sweep line:

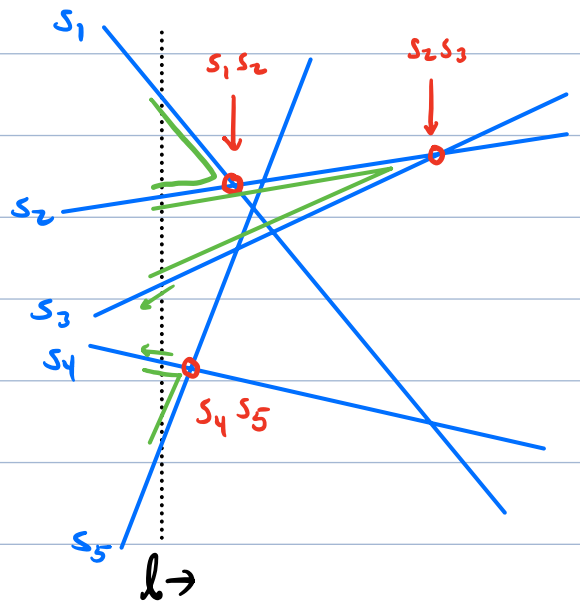
$$s_i : y = a_i x + b_i$$

- As sweep line moves, can evaluate the actual y-coords on demand:

E.g. $l : x = x_0$ $\quad \forall i,\ y_i = a_i x_0 + b_i$

Future Events - Stored in priority queue (sorted by x)

- All segment endpts to right of sweep line
- "Imminent intersections" - Intersections to right of sweep line of pairs of segments that are consecutive on sweep line



Note: $s_3 s_4$ occurs in the past, so ignored

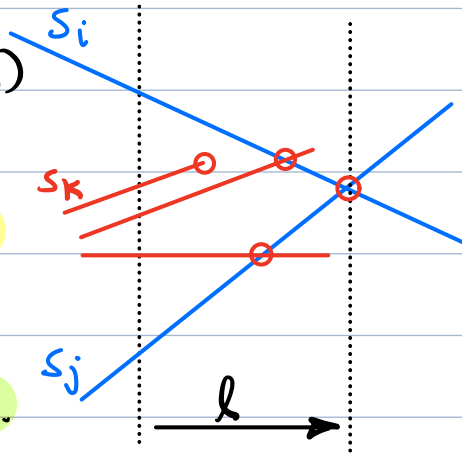
Why? Consecutive pairs are easy to detect + update

- At most $n-1 = O(n)$ intersection events in the priority queue (not $O(n^2)$)
- **Concern:** Since we only store limited cases, we might miss some intersections

Lemma: If the next event is an intersection, the two intersecting segments are consecutive on the current sweep line.

Proof: (Contradiction)

- Suppose not
- Let $s_i s_j$ be next event, but there is s_k between them.



Either:

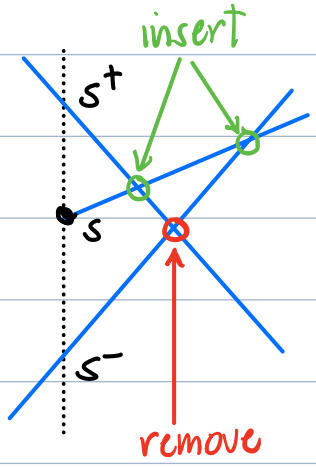
- s_k right endpt
- s_k hits s_i
- s_k hits s_j

Before $s_i s_j$ intersect □

Plane-Sweep Algorithm Input: $S = \{s_1, \ldots, s_n\}$ $s_i = (p_i, q_i)$

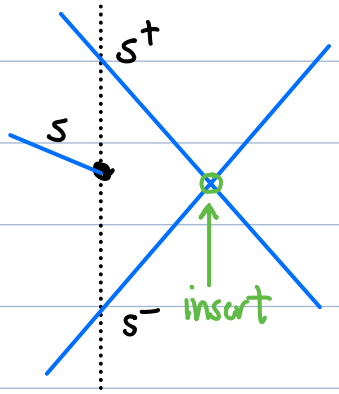
- Insert all segment endpts into priority queue (sorted by x-coordinate)
- while (queue is non-empty) {
  - extract next event
  - cases:

① Left endpoint of segment s:



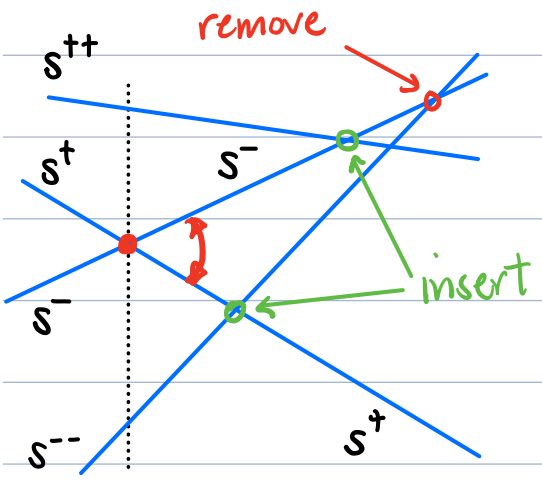
- Insert s into sweep-line status (dictionary) based on y-coord
- Let s^+ & s^- be segs immediately above and below (using pred. & succ. ops)
- If s^+, s^- had an intersection event in priority queue, remove it (since s^+ & s^- no longer consecutive)
- If s & s^+ or s & s^- intersect, add these events to priority queue

② Right endpoint of segment s:



- Let s^+ & s^- be segments above & below
- Delete s from sweep-line status
- If s^+ & s^- intersect to right, add this event to priority queue (now consecutive)

③ Intersection between segments s^+ & s^-



- Let s^{++} & s^{--} be segs immed. above & below intersection (pred & succ)
- Remove from priority queue any events for s^+ & s^{++} or s^- & s^{--} (no longer consecutive)
- Swap s^+ & s^- on sweep-line status
- Check for intersections of s^- & s^{++} and s^+ & s^{--} to right of sweep line & add new events to priority queue (now consecutive)

Correctness: (sketch)

- Sweep-line status stores all segments intersecting sweep line in proper order
- Priority queue stores:
  - All segment endpoints to right of sweep line
  - All intersection events to right of sweep line involving consecutive pairs

Proof is straightforward exercise in induction.

Need to show that, assuming all requirements hold just prior to each event, they still hold after.

Running time: n = no. of segments m = no. of intersects

Number of events = $2n + m = O(n+m)$

($2n$: endpts, m: intersects)

Time per event:

- Extract_min from priority queue } $O(\log n)$
- $O(1)$ dictionary ops (insert, delete, pred, succ, swap) } $O(\log n)$
- $O(1)$ priority queue ops (add/remove event) } $O(\log n)$

Total time = $O((n+m)\log n)$

Space = $O(n)$ working storage for data structures + $O(m)$ for output

Summary:

- Plane sweep algorithm for reporting line segment intersections
- $O((n+m)\log n)$ time + $O(n)$ working space [+ $O(m)$ output size]

- Can we do better?
  - → $O(m + n \log n)$ is achievable (covered later)