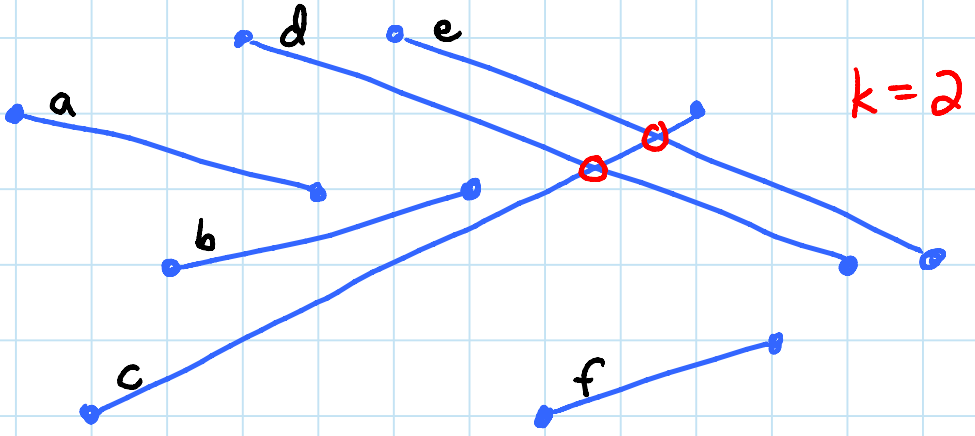
6.006	Lecture 22	Dec. 6. 2016

TODAY: Computational Geometry
- line-segment intersection
- sweep-line technique
- closest pair of points

Motivation: Collision Detection
(video games, physical simulation, etc.)
- consider one moment in time (frame, timestep)
- let's stick to 2D (but same ideas work in 3D)

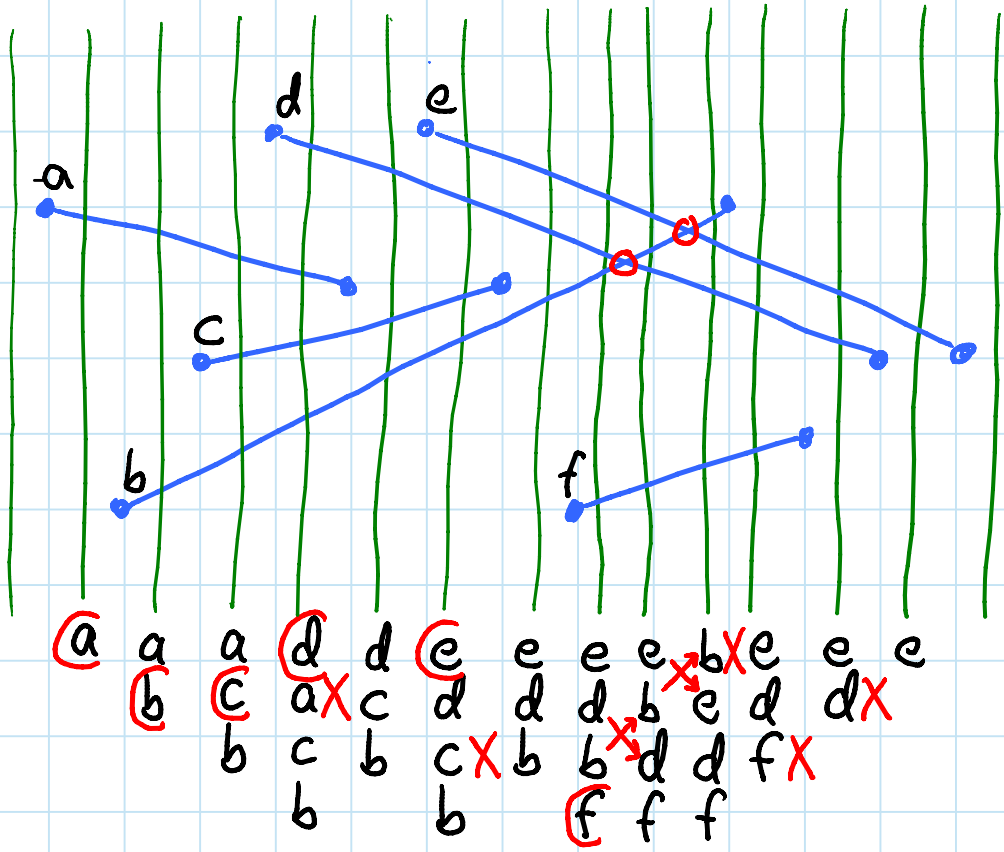
Line-segment intersection:
given n line segments in 2D, find the k intersections



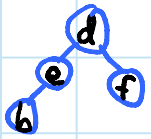
Obvious algorithm: $\Theta(n^2)$ ~ optimal if $k = \Theta(n^2)$!
for each pair (l, l') of line segments:
check for intersection — $\Theta(1)$
↳ intersect lines, check if within segments

Sweep-line technique:

- idea: sweep a vertical line from left to right & maintain intersection of sweep line with input



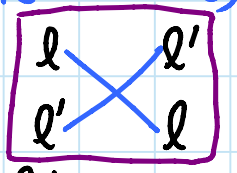
- discrete-event simulation of continuous sweep
- events when intersection changes:
  - segment endpoints → insert/delete
  - intersections → order flips
    - ↳ only need to consider adjacent segments crossed by sweep line (adjacent before crossing)



- maintain sweep-line intersection as AVL tree ordered by (current) y coordinate
- maintain events in priority queue (heap) ordered by x coordinate (sweep time)

Algorithm:

```
T = empty AVL tree
Q = build-heap({l.left, l.right for segment l})
until Q empty:
    event = Q.delete-min()
    if event is l.left:
        insert l into T (key = l.left.y)
        Q.add(meet(l, T.predecessor(l)),   ← if they
              meet(l, T.successor(l)))     ← intersect
    elif event is l.right:
        Q.add(meet(T.predecessor(l), T.successor(l)))
        delete l from T (l stores pointer to T node)
    elif event is meet(l,l'):
        output intersection between l & l'
        swap contents of T nodes for l & l'
        Q.add(meet(l, T.predecessor(l)),   ← if they
              meet(l', T.successor(l)))    ← intersect
```



Time: $O(\#\text{events} \cdot \lg(\#\text{events}))$

↳ $2n$ endpoint events

↳ k intersection events

$= O((n+k)\lg(n+k))$ ($n+k$: $O(n^2)$)

$= O((n+k)\lg n)$ ← output sensitive: better for small k

Best algorithm: $O(n \lg n + k)$

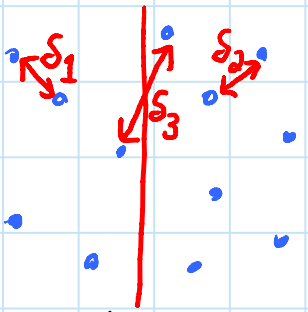
Closest pair of points: (~collision of unit disks)

given n points in 2D, find the two closest

Obvious algorithm: $\Theta(n^2)$

min(distance(p,q) for point p for point $q \neq p$)

Divide & conquer algorithm:



- sort points by x coordinate
- δ_1 = recurse(points[:n/2])
- δ_2 = recurse(points[n/2:])
- δ_3 = min. distance between left & right halves
- return $\min\{\delta_1, \delta_2, \delta_3\}$

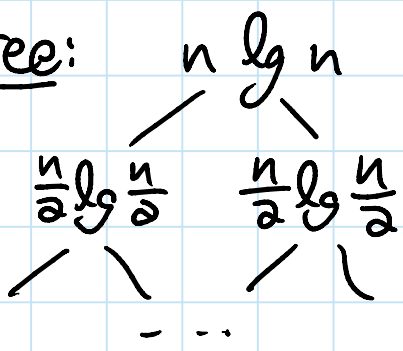
Strip merge:

- restrict to points within $\min\{\delta_1, \delta_2\} = \delta$ of middle line $x = (\text{points}[n/2-1].x + \text{points}[n/2])/2$
- sort these points by y
- each left point needs to consider ≤ 3 right points because right points $\geq \delta_2$ from each other
- ditto for each right point
- can compute δ_3 in $O(n)$ time by merging 2 sorted arrays

Time: $T(n) = 2T(n/2) + \underbrace{\Theta(n \lg n)}_{\text{y sort}}$

$= \Theta(n \lg^2 n)$

Recursion tree:



— $n \lg n$

— $n \lg \frac{n}{2} = n \lg n - n \approx n \lg n$

Best algorithm: $\Theta(n \lg n)$

- presort points by x & by y in two crosslinked arrays
- when recursing, project y array too — $O(n)$
- before merging, map to y array — $O(n)$

$\Rightarrow T(n) = 2T(n/2) + O(n)$

$= O(n \lg n)$