Geometry

Model

- RAM
- operations on reals, including sqrts.
- (why OK)
- line segment intersections
- DISCRETE randomization

Applications:

- graphics of course
- any domain where few variables, many constraints

Point location in line arrangements

setup:

- n lines in plane
- gives $O(n^2)$ convex regions
- goal: given point, find containing region.
- for convenience, use *triangulated* $T(L)$
- triangulation introduces $O(n^2)$ segments (planar graph)
- assume all inside a bounding triangle

how about a binary space partition?

- single line splits input into two groups of n-1 rays
- search time (depth) could be n

A good algorithm:

- choose r random lines R, triangulate
- inside each triangle, some lines.
- **good** if each triangle has only $an(\log r)/r$ lines in it
- will show good with prob. 1/2
- recurse in each triangle—halves lines

Lookup method: $O(\log n)$ time.
Proof of **good**

- As with cut sampling, consider individual "problem" events, show unlikely
- Let Δ be all triplets of L-intersections
- when $\delta \in \Delta$ is bad:
  - let $I(\delta)$ be number of lines hitting δ
  - let $G(\delta)$ be lines that induce δ (at most 6)
  - for bad δ, must have all lines of $G(\delta)$ in R (call this $B_1(\delta)$), no lines of $I(\delta)$ in R (call this $B_2(\delta)$.
- bound prob. of bad δ:
  - we know
  $$\Pr[\delta] \le \Pr[B_1(\delta)] \Pr[B_2(\delta) \mid B_1(\delta)]$$
  (why not equal? Because triangulation may not create triangle from δ)
  - Given $B_1(\delta)$, still need $r - |G(\delta)| \ge r - 6 \ge r/2$ drawings (assuming $r > 12$)
  - prob. none picked is at most
  $$(1 - \frac{|I(\delta)|}{n})^{r/2} \le e^{-rI(\delta)/2n}$$
  - Only care if $I(\delta) > an(\log r)/r$—large triplets
  - $\Pr[B_2(\delta) \mid B_1(\delta)] \le r^{-a/2}$ for large triplet
- prob. some bad at most
$$r^{-a/2} \sum_{\delta} \Pr[B_1(\delta)]$$
- sum is expected number of large triplets.
  - at most r^2 points in sample
  - at most $(r^2)^3 = r^6$ triplets in sample
  - expectation at most r^6
  - choose $a > 12$, deduce result.

Construction time:

- Recurrence
$$T(n) \le n^2 + cr^2 T(an\frac{\log r}{r}) = O(n^{2+\epsilon(r)})$$
- ϵ decreasing with r
- by choosing large r, arbitrarily close to $O(n^2)$

Randomized incremental construction

Special sampling idea:

- Sample all *except* one item
- hope final addition makes small or no change

Method:

- process items in order
- average case analysis
- randomize order to achieve average case
- e.g. binary tree for sorting

Randomized incremental sorting

- Funny implementation of quicksort
- repeated insert of item into so-far-sorted
- each yet-uninserted item points to "destination interval" in current partition
- bidirectional pointers (interval points back to all contained items)
- when insert x to I,
  - splits interval I (x is "pivot" for I)
  - must update all I-pointers to one of two new intervals
  - finding items in I easy (since back pointers)
  - work proportional to size of I
- If analyze insertions, bigger intervals more likely to update; lots of quadratic terms.

Backwards analysis

- run algorithm backwards
- at each step, choose random element to un-insert
- find expected work
- works because:
  - condition on what first i objects are
  - which is i^{th} is random
  - discover didn't actually matter what first i items are.

Apply analysis to Sorting:

- at step i, delete random of i sorted elements
- un-update pointers in adjacent intervals
- each pointer has $2/i$ chance of being un-updated
- expected work $O(n/i)$.
- true *whichever* are i elements.
- sum over i, get $O(n \log n)$
- compare to trouble analyzing insertion
  - large intervals more likely to get new insertion
  - for some prefixes, must do $n - i$ updates at step i.

Convex Hulls

Define

- assume no 3 points on straight line.
- output:
  - points and edges on hull
  - in counterclockwise order
  - can leave out edges by hacking implementation

$\Omega(n \log n)$ lower bound via sorting
algorithm (RIC):

- random order p_i
- insert one at a time (to get S_i)
- update $conv(S_{i-1}) \rightarrow conv(S_i)$
  - new point stretches convex hull
  - remove new non-hull points
  - revise hull structure

Data structure:

- point p_0 inside hull (how find? centroid of 3 vertices.)
- for each p, edge of $conv(S_i)$ hit by $\vec{p_0 p}$

- say p *cuts* this edge
- To update p_i in $conv(S_{i-1})$:
  - if p_i inside, discard
  - delete new non hull vertices and edges
  - 2 vertices v_1, v_2 of $conv(S_{i-1})$ become p_i-neighbors
  - other vertices unchanged.
- To implement:
  - detect changes by moving out from edge cut by $\vec{p_0 p}$.
  - for each hull edge deleted, must update cut-pointers to $\vec{p_i v_1}$ or $\vec{p_i v_2}$

Runtime analysis

- deletion cost of edges:
  - charge to creation cost
  - 2 edges created per step
  - total work $O(n)$
- pointer update cost
  - proportional to number of pointers crossing a deleted cut edge
  - **backwards** analysis
    * run backwards
    * delete random point of S_i (**not** $conv(S_i)$) to get S_{i-1}
    * same number of pointers updated
    * expected number $O(n/i)$
      · what Pr[update p]?
      · Pr[delete cut edge of p]
      · Pr[delete endpoint edge of p]
      · $2/i$
    * deduce $O(n \log n)$ runtime

Book studies 3d convex hull using same idea, time $O(n \log n)$, also gets voronoi diagram and Delauney triangulations.