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

Backwards analysis
- compute expected time to insert $S_{i-1} \rightarrow S_i$
- backwards: time to delete $S_i \rightarrow S_{i-1}$
- conditions on $S_i$
- but generally analysis doesn’t care what $S_i$ is.

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 \( \text{conv}(S_{i-1}) \rightarrow \text{conv}(S_i) \)
  – 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 $\text{conv}(S_i)$ hit by $p_0p$
- say $p$ cuts this edge
- To update $p_i$ in $\text{conv}(S_{i-1})$:
  - if $p_i$ inside, discard
  - delete new non hull vertices and edges
  - 2 vertices $v_1, v_2$ of $\text{conv}(S_{i-1})$ become $p_i$-neighbors
  - other vertices unchanged.
- To implement:
  - detect changes by moving out from edge cut by $p_0p$.
  - for each hull edge deleted, must update cut-pointers to $p_i\overrightarrow{v_1}$ or $p_i\overrightarrow{v_2}$

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 $\text{conv}(S_i)$) to get $S_{i-1}$
    * same number of pointers updated
    * expected number $O(n/i)$
      * what $\Pr[\text{update } p]$?
      * $\Pr[\text{delete cut edge of } p]$
      * $\Pr[\text{delete endpoint edge of } p]$
      * $2/i$
    * 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.
Linear programming.

- define

- assumptions:
  - nonempty, bounded polyhedron
  - minimizing $x_1$
  - unique minimum, at a vertex
  - exactly $d$ constraints per vertex

- definitions:
  - hyperplanes $H$
  - basis $B(H)$
  - optimum $O(H)$

- Simplex
  - exhaustive polytope search:
  - walks on vertices
  - runs in $O(n^{d/2})$ time in theory
  - often great in practice

- polytime algorithms exist, but bit-dependent!

- OPEN: strongly polynomial LP

- goal today: polynomial algorithms for small $d$

Randomized incremental algorithm

$$T(n) \leq T(n-1,d) + \frac{d}{n}(O(dn) + T(n-1,d-1)) = O(d!n)$$