Dynamic Data Structures

e.g. in 1D, \(O(\log n)\) query, insert, delete time
by balanced search trees (AVL, red-black, splay, ...)

Dynamic 2D Convex Hull

DS to support insert, delete, query
- find extreme CH vertex along query direction
  - membership:
    - test if query pt is inside CH
  - intersect CH with query line
  - find next/prev vertex to query CH vertex.

Suffices to consider upper hull (UH)
issue: insert/delete can change UH drastically

Method 1 (insert-only): Preparata'80
maintain UH in a balanced search tree
\textbf{insert}(q):
1. find 2 tangent pts \(t_1, t_2\) to \(q\)
2. split \(t_1, t_2\)
to get sublists \(L_1, L_2, L_3\)
3. merge \(L_1\) & \(L_2\)

Split/merge: \(O(\log n)\) time e.g. by red-black tree

How to find tangent pt \(t_i\) to \(q\):
idea - binary search
if \(q\) left of \(P_m\) then search left
else if \(q\) above \(P_m\) then search left
else search right
Method 2 (Fully dynamic): Overmars-van Leeuwen '80

Subproblem: Given 2 vertically separated UTIs A, B
find bridge between A, B

First Sol'n: nested binary search

If \( l_m \) intersects \( B \), then search left in A
else search right in A.

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

Better Sol'n: Simultaneous binary search
Case 1. bk above lm $\Rightarrow$ search left in A

Case 2. am above lk $\Rightarrow$ search right in B

Case 3. lk below lm and am below lk

if lm is lower than lk at $l$ then search left in B else search right in A

each iteration removes half of A or B

$\Rightarrow \leq 2 \log n$ iterations $\Rightarrow O(\log n)$ time

Data Structure (Hull Tree)

divide by median vertical line $l$

store bridge

recursively on left & right

Space $O(n)$
insert/delete(q):
  if q is left of e
    insert/delete on left subtree
  else  right
    recompute bridge ← O(log n) time

⇒ O(log² n)

query: O(log n) for extreme pt query, membership query, ...
      [next/prev O(1) time with more effort]

issue - how to maintain balance?

Option 1 - weight balancing
  whenever size(left subtree of v) & size(right subtree of v)
  differs by more than const factor

rebuild tree at v ← O(size of tree at v)

insert/delete remains O(log² n) amortized
  (i.e. total time for any sequence of n insert/delete is O(n log² n))

Option 2 - rotation

AVL/red-black tree - O(1) rotations per insert/delete
  each rotation requires recomputing 2 bridges

⇒ insert/delete still O(log² n) recomputing 2 bridges