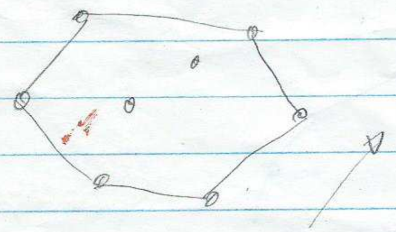
Dynamic Geometric Data Structures

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.

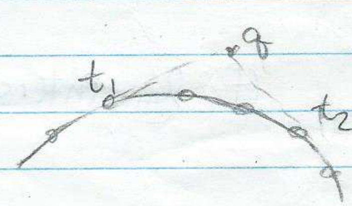
Suffices to consider upper hull (UH)
issue: insert/delete can change UH drastically

Method 1 (Insert-Only): Preparata '80

maintain UH in a balanced search tree

insert(q):

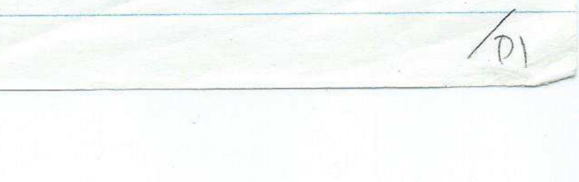
1. find 2 tangent pts t_1, t_2 to q
2. split at p_1, p_2 to get sublists L_1, L_2, L_3
3. merge L_1 & L_2



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

How to find tangent pt t_1 to q:

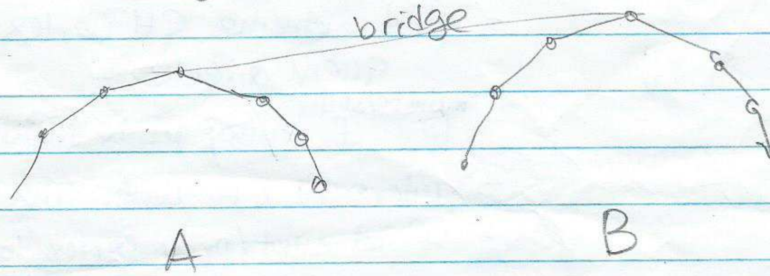
idea - binary search
if q left of p_m then search left
else if q above ℓ_m then search left
else search right



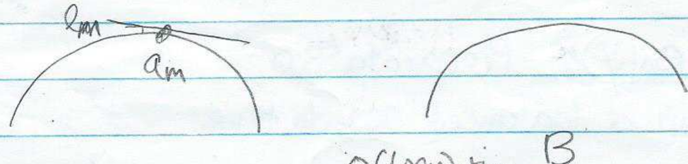
$\Rightarrow$ insert time $O(\log n)$
query time $O(\log n)$ [next/prev $O(1)$]
space $O(n)$

Method 2 (Fully dynamic): Overmars-van Leeuwen '80

Subproblem Given 2 vertically separated UHs A, B
find bridge between A, B



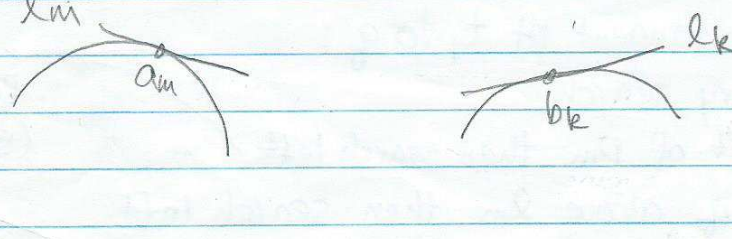
First Sol'n: nested binary search



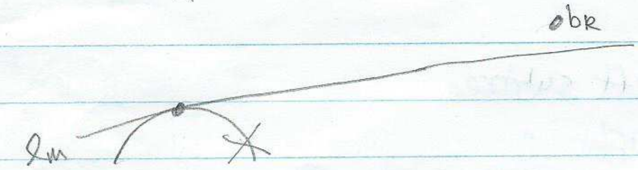
$O(\log n)$ time
if l_m intersects B then search left in A
else search right in A

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

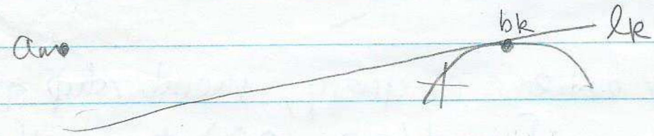
Better Sol'n: simultaneous binary search



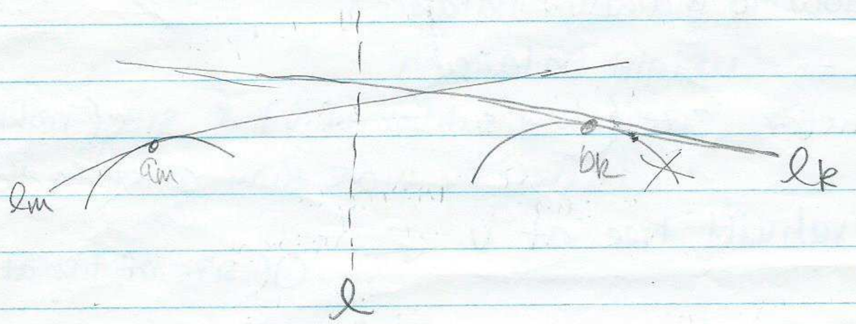
Case 1. b_k above l_m $\Rightarrow$ search left in A



Case 2. a_m above l_k $\Rightarrow$ search right in B



Case 3. b_k below l_m and a_m below l_k

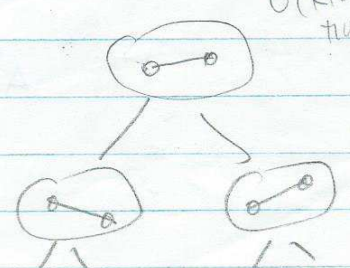


if l_m is lower than l_k at l then search left in B
else search right in A

each iteration removes half of A or B
$\Rightarrow$ $\leq 2 \log n$ iterations $\Rightarrow$ $O(\log n)$ time

(OPEN - k convex polygons in det $O(k \log n)$ time)

Data Structure (Hull Tree)
divide by median vertical line l
store bridge
recurse on left & right



Space $O(n)$

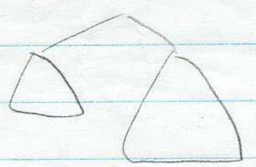
D2

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

$\Rightarrow \boxed{O(\log^2 n)}$

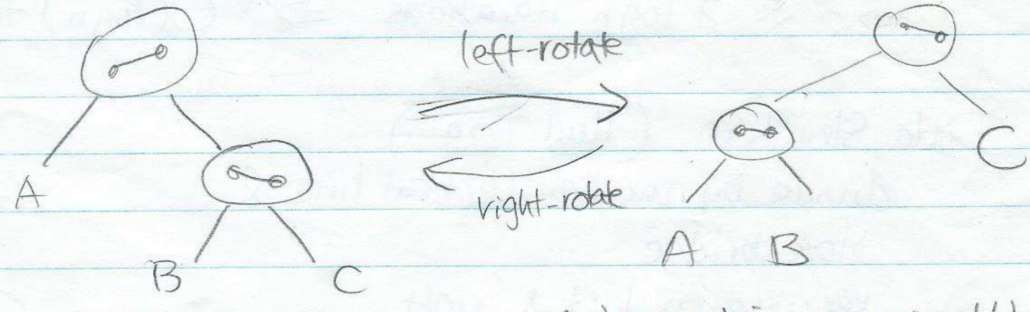
query: $\boxed{O(\log n)}$ for extreme pt query, membership query, ...
[next/prev $O(1)$ time with more effort]

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 $O(\log^2 n)$ <u>amortized</u>
(i.e. total time for any sequence of n inserts/deletes is $O(n \log^2 n)$)

option 2 - rotation



AVL / red-black tree - $O(1)$ rotations per insert/delete
each rotation requires recomputing 2 bridges
$\Rightarrow$ insert/delete still $O(\log^2 n)$