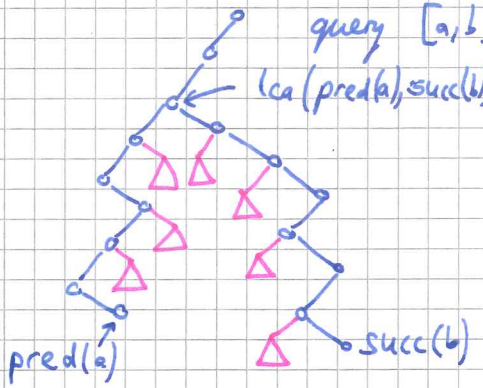
Lecture 6: Geometry

static: persistence
dynamic: retroactivity
} automatic

Orthogonal range searching

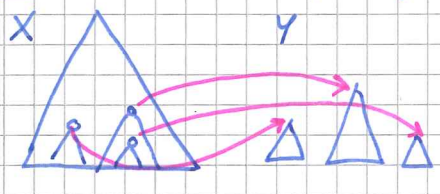
- maintaining n points in d-dimensional space (static or dynamic)
- query: existence / list / count all points in $[a_1, b_1] \times [a_2, b_2] \times \dots \times [a_d, b_d]$
  goal: $O(\log^d(n) + k)$ ← size of output

1D: balanced search tree (with values only in leaves)

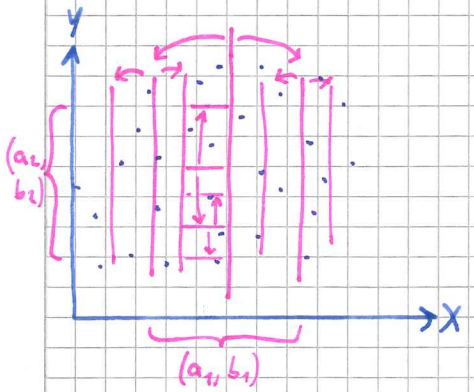


- each node stores - max of left subtree
  - size of tree (# leaves)
- maintaining balance with rotations
  → $O(\log n + k)$ time queries

2D: 1D-tree on X plus for each node: a separate 1D-tree on Y of all its points



- each point in $O(\log n)$ subtrees
  ↳ $\Theta(n \log n)$ space
- for query $([a_1, b_1] \times [a_2, b_2])$:
  1. X-query $([a_1, b_1])$ → $O(\log n)$ X-subtrees
  2. follow pointers → $O(\log n)$ y-subtrees
  3. $O(\log n)$ Y-queries $([a_2, b_2])$ → $O(\log^2 n)$ Y-subtrees
  ↳ $O(\log^2 n + k)$ query time
  ↳ static construction time: $\Theta(n \log n)$ (non-obvious)
  ↳ dynamic updates: $O(\log^2(n))$ (amortized, later today)

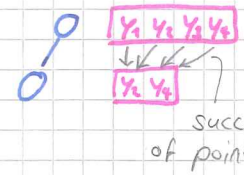


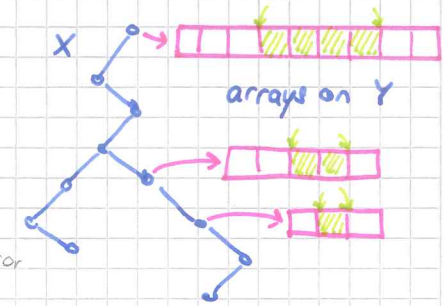
dD recursion on dimension: (no trick here)
$O(\log^d(n) + k)$ queries
$O(n \log^{d-1}(n))$ space & construction time
$O(\log^d(n))$ updates

Layered Range Tree (Gabow et al. 1984)

idea: reuse searches

- keep y-ordered subsets as sorted arrays
- arrays have pointers to arrays of children





for query:
- binary search in root for $[a_2, b_2]$
- keep track of the pointers while decomposing x-range

2D: $O(\log n)$ query dD $O(\log^{d-1}(n))$ query (no effect on construction time)

How to make this dynamic? Rotating a node close to the root is costly as many lower-dimensional trees have to be rebuilt.

Dynamization with augmentation via weight-balanced BB[α] trees

for each node x: $\begin{cases} size(left(x)) \geq \alpha \cdot size(x) \\ size(right(x)) \geq \alpha \cdot size(x) \end{cases}$ for $\alpha \in (0.5, 1) \Rightarrow$ height $\leq \log_{\frac{1}{1-\alpha}}(n)$

whenever some x becomes unbalanced $\Rightarrow$ rebuild subtree of x in perfect balance

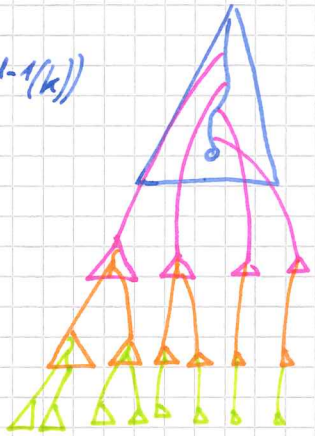
why can we afford it? rebuilding tree of size $k \Rightarrow$ imbalance was $\Theta(k)$ so we can pay for rebuilding from previous ins/del into this subtree
↳ amortized cost of ins/del is $\Theta(\log n)$

useful for:

Dynamic Range Tree

- rebuilding of k points in d dimensions costs $\Theta(k \log^{d-1}(k))$
- charge for a single ins/del:
  - $\Theta(\log n)$ updates costing $\Theta(\log^{d-1}(n))$ on 1st-dimension
  - $\Theta(\log^2 n)$ updates costing $\Theta(\log^{d-2}(n))$ on 2nd-dimension
  - ⋮
  - $\Theta(\log^d(n))$ updates costing $\Theta(1)$ on d^{th}-dimension

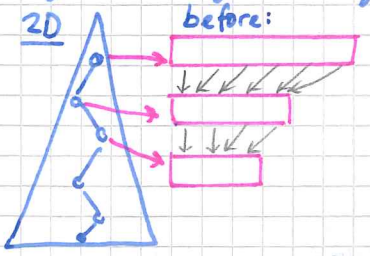
  ↳ $\Theta(d \log^d n) = \Theta(\log^d n)$ amortized updates



So we get dynamization at no extra cost!
Also for layered range trees?

Dynamic Layered Range Tree

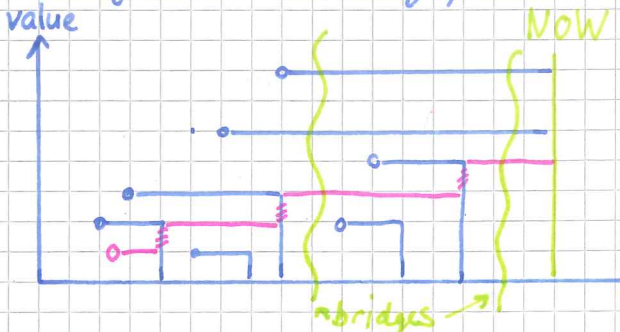
2D



BBST on top → still allows $O(\log n)$ search
linked list → easy to update
for all others

$\Rightarrow$ [query in $\Theta(\log^{d-1}(n) + k)$, update in $\Theta(\log^d(n))$]

Partially retroactive priority queue in $O(\log(n))$ per operation



L-view: insert = rightward ray
delete-min = upward ray

e.g. insert(t, "insert(t)") (in pink)
= delete("delete min")
which inserts $\min\{k, k' : k' \text{ deleted at time } t\}$

bridge at time t: iff $Q_t \subseteq Q_{now}$

t' is bridge preceding time t → $\max\{k' \in Q_{now} : k' \text{ inserted} \geq t'\}$

T_1 - store Q_{now} as balanced BST (by value)

T_2 - store bal. BST on leaves = inserts (by time) and aggregate:
$\forall$ node x: $\max\{k' \in Q_{now} : k' \in \text{subtree of } x\}$

T_3 - store bal BST on leaves = updates

$$\begin{cases} +1 & \text{insert}(k) \quad k \notin Q_{now} \\ -1 & \text{delete-min} \\ 0 & \text{insert}(k) \quad k \in Q_{now} \end{cases}$$

nodes aggregate: - subtree sums
- min/max of prefix sums

↳ this allows us to find times that are bridges as points with prefix sum = 0 in T_3

↳ determine change to Q_{now} using T_2 → update T_1 → update T_2, T_3 all in $O(\log n)$