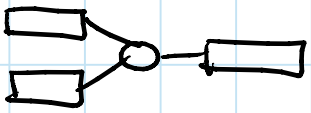
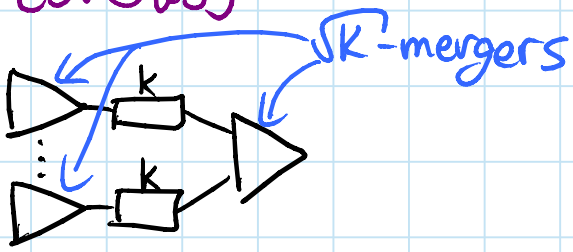
Computational geometry

Recall: Lazy funnelsort [BFO26]



binary merger



K-merger

- sort N items in $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ mem. trans.

Distribution sweeping [BFO26]

- use lazy funnelsort to drive divide & conquer
- replace binary merger by thinking about streams of inputs & output, adding extra data along the way

Problems: all solved in $O(\frac{N}{B}\log_{M/B}\frac{N}{B} + \frac{\text{output}}{B})$

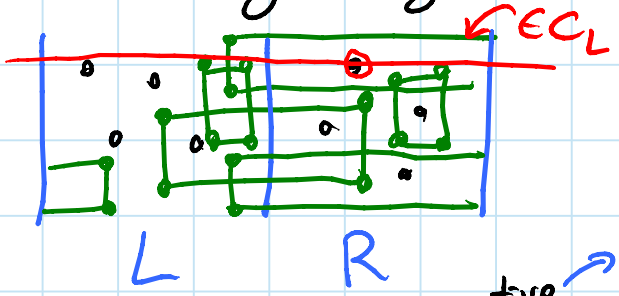
- measure of 2D rectangles
- batch orthogonal range queries
- orthogonal line segment intersection
- pairwise rectangle intersection
- line segment visibility from a point
- all Euclidean 2D nearest neighbors
- all maximal points in 3D

Batch orthogonal range searching:

- here: count # rectangles containing each point

① sort points & corners by x coordinate → sorting by y

② divide & conquer in x via lazy funnelsort where binary merger = upward sweep



→ stabbed by sweep line

- maintain C_L = # active rects. with corners in L & spanning all of R

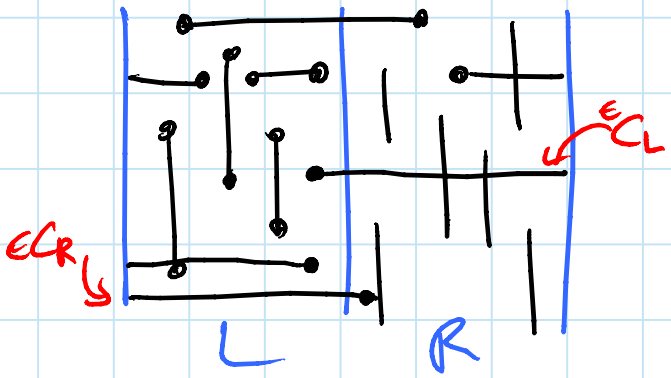
C_R = ~~ in R ~ spanning ~ L

- when encountering a point in L_a if $C_R > 0$ then increment counter by C_R

- reporting answers is harder/messier (see BF02b)

Orthogonal line segment intersection:

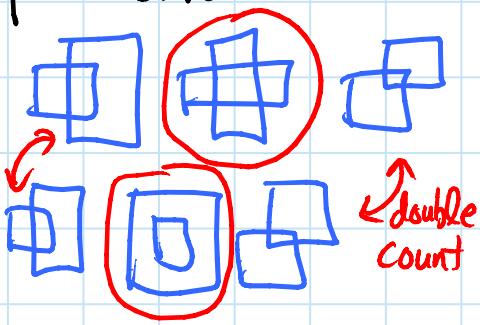
- here: count # vertical segments hit by each horizontal segment
  (again can solve reporting by same tricks)
- binary merger by upward sweep:
  - maintain c_L = # active vertical segments in L
    ↳ stabbed by sweep line
    c_R = # active vertical segments in R
  - if see horizontal segment that spans all of L then add c_L to its counter;
    all of R then add c_R to its counter



Pairwise rectangle intersection:

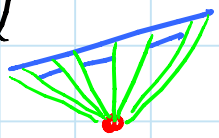
reduction to previous two problems:

- line segment intersection to find
- batched orthogonal range searching to find

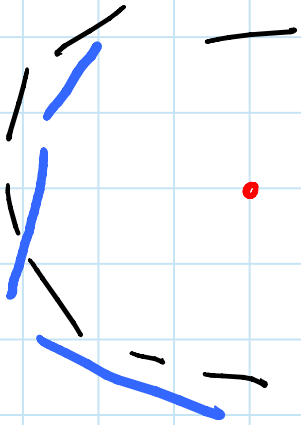


Line segment visibility from a point:

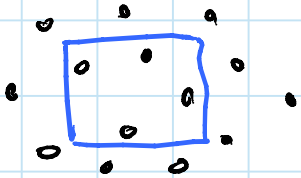
compute clockwise sequence of partial line segments visible from given point



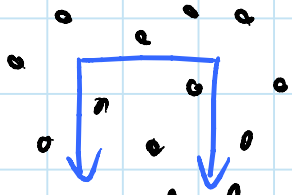
- divide & conquer (in original order) endpoints by lazy funnelsort. sorting by angle from point where binary merger
  = angular sweep
- just maintain active segment (if any) from L & from R



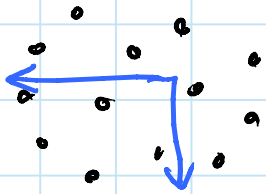
Orthogonal range searching: preprocess set of points to support reporting queries in $O(\log_B N + \frac{\text{output}}{B})$



4-sided



3-sided



2-sided

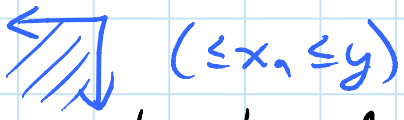
Internal memory:

	query	space
– k-d trees	$O(n^{1-1/d})$	$O(n)$
– range trees	$O(\lg^d n)$	$O(n \lg^{d-1} n)$
+ fractional cascading	$O(\lg^{d-1} n)$	

Cache-oblivious:

	query	space	update	ref.
– k-d trees:	$O((\frac{N}{B})^{1-1/d} + \frac{out}{B})$	$O(N)$	$O(\log_B^2 N)$	[AADH03]
– 2D range trees:				
– 4-sided:	$O(\log_B N + \frac{out}{B})$	$O(N \lg^2 N)$	∞	[AADH03]
* – 4-sided:	$O(\log_B N + \frac{out}{B})$	$O(N \frac{\lg^2 N}{\lg\lg N})$	∞	[ABFLO5]
– 4-sided counting:	$O(\log_B N)$	$O(N \lg N)$ $O(N)$ with bit tricks	∞	[ABFLO5]
* – 3-sided:	$O(\log_B N + \frac{out}{B})$	$O(N \lg N)$	∞	[ABFLO5, AZ06]
* – 2-sided:	$O(\log_B N + \frac{out}{B})$	$O(N)$	∞	[AZ06]
	$O(\lg N + \frac{out}{B})$	$O(N)$	$O(\frac{\lg^2 N}{\lg B})$ insert only	[AZ06]

2-sided: [AZ06]



- static search tree on points, keyed by y
- array of points, with duplication

Query: $(\le x, \le y)$

1. binary search for y in tree
2. follow pointer into array
3. scan array to the right until reach a point whose x coord > query x
   - output points in $(\le x, \le y)$

Claims:
- find all points in $(\le x, \le y)$
- # scanned points is $O(\#\text{output points})$
- array has size $O(N)$

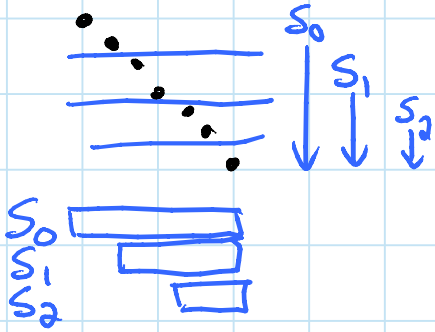
$\alpha > 1$

Density:
- query $(\le x, \le y)$ **dense** in S if scan $\le \alpha \cdot$ #outputs points in S
  - sorting S by x coord., starting at left
  - i.e. #points in $(\le x, *) \le \alpha \cdot$ #points in $(\le x, \le y)$
- else $(\le x, \le y)$ **sparse** in S

First try:

- let S_0 = all points (sorted by x)
- observation: $(\le x, \le y)$ is surely dense in S_0 for y large e.g. $y \ge$ max y coord.
- let y_1 = largest y where some query $(\le x, \le y_1)$ is sparse in S_0
- let S_1 = points in $(*, \le y_1)$ (sorted by x)
- let y_2 = ... in S_1
- repeat until S_i of constant size
- array = $S_0, S_1, S_2, \ldots, S_i$

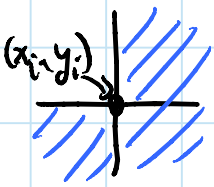
→ correct & fast queries but quadratic space:



Correct attempt: maximize common suffix

- let x_i = max. where $(\le x_i, \le y_i)$ is sparse for S_{i-1}
- let $P_{i-1} = S_{i-1} \cap (\le x_i, *)$
- let $S_i = S_{i-1} \cap ((*, \le y_i) \cup (> x_i, *))$
- array = $P_0, P_1, P_2, \ldots, P_{i-1}, S_i$

↑ $O(1)$ size



Proof of claims:

- correctness: the repeated elements always have x coord. < last seen point in any query
  - can avoid duplicates by focusing on monotone sequence of x coords.

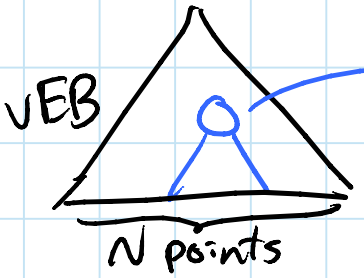
- query time: repetion is geometric series
  $\Rightarrow$ lose only $O(1)\times$
- space: $|P_{i-1} \cap S_i| \leq \frac{1}{\alpha} \cdot |P_{i-1}|$
  because (x_i, y_i) is sparse in S_{i-1}
  $\Rightarrow$ charge storing P_{i-1} to $P_{i-1} \setminus S_i$
  $\Rightarrow$ each point charged only once.
  factor $\frac{1}{1-\frac{1}{\alpha}} = \frac{\alpha}{\alpha-1}$
  $\Rightarrow \leq \frac{\alpha}{\alpha-1} \cdot N$ space

- can be computed in $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ [Brodal]

3-sided: [AZ06]

$O(\log_B N + \frac{\text{output}}{B})$ query; $O(N \lg N)$ space

- static search tree where leaves = points, keyed by x:



stores two 2-sided structures for ↖ & ↗ on points in the subtree

$\Rightarrow O(N \lg N)$ space

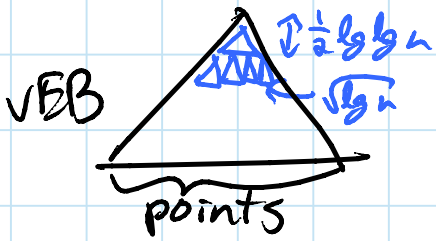
query $(\in [l, r], \leq y)$:
- find $\text{lca}(l, r)$ (vEB analysis)
- query $(\geq l, \leq y)$ in left child
- query $(\leq r, \leq y)$ in right child

OPEN: 3-sided range queries
$O(\log_B N + \frac{\text{output}}{B})$ query
$O(N)$ space
i.e. match persistent B-tree

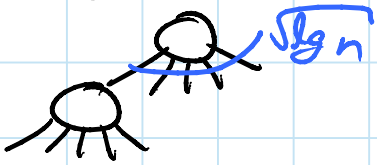
4-sided: [ABFLO5]

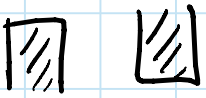
$O(\log_B N + \frac{\text{output}}{B})$ query; $O(N \frac{\lg^2 N}{\lg\lg N})$ space

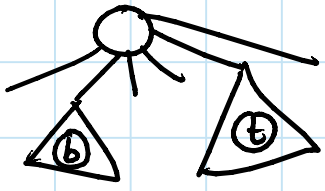
- static search tree where leaves = points, keyed by y



- conceptually contract $\frac{1}{2}\lg\lg n$-height subtrees into $\sqrt{\lg n}$-degree nodes:
  $\Rightarrow$ height $= O(\frac{\lg n}{\lg\lg n})$



- for each such node, store
  - two 3-sided structures for  on points in subtree
  - $\lg n$ static search trees, keyed by x, on points in each interval of children
- query $([l,r],[b,t])$:
  - find $\text{lca}(b,t)$ in tree
  - query $([l,r], \geq b)$ in (left) child $\ni b$
  - query $([l,r], \leq t)$ in (right) child $\ni t$
  - query $([l,r], *)$ in children in between



- space:

$$O(\underbrace{N \lg N}_{\text{3-sided}} \underbrace{\frac{\lg N}{\lg\lg N}}_{\#\text{repetitions of element}})$$

tree, #trees

Other geometry results:

- R-trees: [AdH05]
  storing rectangles, not points
  - point query: $O((\frac{N}{B})^{\varepsilon})$ $\forall \varepsilon > 0$
  - rectangle query: $O(\sqrt{\frac{N}{B}} + \frac{\text{output}}{B})$
  - assuming max. overlap (point) $\leq B$

- output-sensitive convex hull: → size of hull
  - lower bound: $\Omega(\frac{N}{B} \log_{M/B} \frac{H}{B})$
  - upper bound: $O(\frac{N}{B} \log_{M/B} \frac{H}{B} + \frac{N}{B} \lg\lg \frac{N}{B})$ [AF07]

- static planar point location: [BCR02]
  $O(\log_B N)$ query
  $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ preprocessing
  $O(N)$ space