Today: Memory Hierarchies meet Geometry
- distribution sweeping via Lazy Funnelsort
- orthogonal 2D range searching:
  - batched
  - online

Lazy Funnelsort: [Brodal & Fagerberg - ICALP 2002]
- K-funnel: merges K sorted lists of total size $\Theta(K^3)$ in $O(K^3 \log M/B \cdot (K/B + K))$ mem. transf.

```
output
buffer
```

```
size $K^3$
```

```
V\k funnel
```

```
\{size $K^{3/2}$ \} total buffers = $K^2$
```

```
\{ total $\geq K^3$
```

- recursive layout: each \( \triangle \) stored consecutive
- fill buffer by merging 2 child buffers; if one empties, recursively fill it
- $N^{1/3}$-way mergesort with $N^{1/3}$-funnel merger sorts in $O(N^{1/3} \log M/B \cdot N/B)$ (as needed in L8 prio. queue) assuming tall cache
Distribution sweeping: [Brodal & Fagerberg – ICALP 2003]
- 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 $O\left(\frac{N}{b} \log_b \frac{N}{b} + \frac{\text{output}}{b}\right)$
- 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: given \( N \) points & \( N \) rectangles, report which points are in which rectangles
- first count \# rectangles containing each pt: 

\[ \text{split by } x, \text{ sorted by } y, \text{ merging } y \text{ orders} \]

- maintain \( c_L = \# \text{active rectangles} \)

\[ \text{stabbed by sweep line} \]

with left corners in \( L \) & spanning \( R \)
(right corners are right of \( R \))

- symmetrically \( c_R = \# \text{active rectangles} \)
with right corners in \( R \) & spanning \( L \)

- when encountering a point in \( L \), add \( c_R \) to its counter

- similarly compute \# outputs from each merge
- allocate that much space for reporting pass
- split up recursion into \( O(N) \)-space parts
(necessary for analysis to work out - see Brodal & Fagerberg)
Orthogonal 2D range searching: preprocess set of points to support reporting queries in $O(\log_B N + \frac{\text{out}}{B})$.

- query: $O(\log_B N + \frac{\text{out}}{B})$
- Space:
  - 2-sided: $O(N)$
  - 3-sided: $O(N \log^2 N)$
  - 4-sided: $O(N \frac{\log^2 N}{\log \log N})$

(static)
- compare with RAM: $O(N \frac{\log N}{\log \log N})$ space

[Arge & Zeh - SoCG 2006]
[Arge, Brodal, Fagerberg]
[Laustsen - SoCG 2005]

[L3]
2-sided: \( [A206] \)
- static search tree on points, keyed by \( y \)
- array of points, with duplication

Query: \( \leq x, \leq y \)
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 unique points in \( \leq x, \leq y \)

Claims:
- find all points in \( \leq x, \leq y \)
- \# scanned points is \( O(\#\text{output points}) \)
- array has size \( O(N) \) \( \alpha \geq 1 \)

Density:
- query \( \leq x, \leq y \) dense in \( S \)
  if \( \#\text{points in } (\leq x, *) \leq \alpha \cdot \#\text{points in } (\leq x, \leq y) \)
  i.e. sorting \( S \) by \( x \) & scanning \((-\infty, x) \)
  visits \# points \( \leq \alpha \cdot \#\text{outputs points in } S \)
- else \( \leq x, \leq y \) sparse in \( S \)

\[ \text{NEXT TIME: USE } \alpha = 2 \]
First try:
- let $S_0 = \text{all points } (\text{sorted by } x)$
- observation: $(\leq x, \leq y)$ is surely dense in $S_0$ for $y$ large e.g. $y \geq \max y \text{ coord.}$
- let $y_i = \text{largest } y \text{ where some query } (\leq x, \leq y_i) \text{ is sparse in } S_{i-1}$
- let $S_i = S_{i-1} \cap (\ast, \leq y_i)$ (sorted by $x$)
- repeat until $S_k$ of constant size
- array = $S_0 \ast S_1 \ast S_2 \ast \ldots \ast S_k$
- correct & fast queries but quadratic space:

Correct attempt: maximize common suffix
- define $y_i$ (but not $S_i$) as before
- let $x_i = \max x \text{ where } (\leq x_i, \leq y_i) \text{ is sparse for } S_{i-1}$
- let $P_{i-1} = S_{i-1} \cap (\leq x_i, \ast)$
- let $S_i = S_{i-1} \cap ((\ast, \leq y_i) \cup (> x_i, \ast))$
- array = $P_0 \ast P_1 \ast P_2 \ldots \ast P_{i-1} \ast S_i$
- $O(1)$ 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.

- 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

- query time: repetition is geometric series
  $\Rightarrow$ lose only $O(1) \times$

- can be computed in $O(\frac{N}{B} \log_2 B \cdot \frac{N}{B})$ [Brodal]
3-sided: $O(\log_b N + \frac{\text{output}}{B})$ query: $O(N \lg N)$ space

- just like structure 3 in L4:
- static search tree where leaves = points, keyed by $x$:

\[ \text{VEB} \]

$N$ points

stores two 2-sided structures for $\leq$ & $\geq$, on points in the subtree

\[ \Rightarrow O(N \lg N) \text{ space} \]

query $([x_1, x_2], \leq y_a)$:
- find $\text{lca}(l, r)$ (VEB analysis)
- query $([x_1, x_2], \leq y_a)$ in left child
- query $([x_1, x_2], \leq y_a)$ 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 of external memory]
4-sided: [ABFL05]
$O(\log_B N + \frac{\text{output}}{B})$ query; $O(N \frac{\log^2 N}{\log_B N})$ space

- static search tree on leaves = points, keyed by $y$

- conceptually contract $\frac{1}{3} \log \log n$-height subtrees into $\sqrt{\log n}$-degree nodes:

  $\Rightarrow$ height $= O(\frac{\log n}{\sqrt{\log n}})$

- for each such node, store
  - two 3-sided structures for on points in subtree
  - $\log n$ static search trees, keyed by $x$, on points in each interval of children

- query $([x_1, x_2], [y_1, y_2])$:

  - find lca($y_1, y_2$) in tree
  - query $([x_1, x_2], y_1)$ in (left) child $\exists y_1$
  - query $([x_1, x_2], y_2)$ in (right) child $\exists y_2$
  - query $([x_1, x_2], \ast)$ in children in between

- space:

  $O(N \log N \frac{\log N}{\log_B N})$

  \[3\text{-sided} \quad \frac{\text{#repetitions of element}}{\text{tree \#trees}}\]