Data structures so far - keys from a totally ordered set
- e.g. numbers in 1D
- accessed by comparisons

Other possibilities
- integers - arithmetic & bit operations give faster alg.
- strings - in CS 240
- geometric data - e.g. pts. or regions in plane
  - not totally ordered, or k-dim. space

2 problems
1. process points, query region to find points in \( R \)
   "Range Search" query region to find points in \( R \)
2. process regions, query point to find region(s)
   "Point Location" query point to find region(s)

Range searching
- preprocess set of points in \( k \) dimensions
  to handle range query
  
  3 measures - \( P \) - preprocessing time
  \( S \) - space
  \( Q \) - query time (\( \geq \) size of output)
  \( U \) - update time for dynamic case - pts can be added/deleted

\( k = 1 \)

rectangle is an interval
In 1-dim  \( P = O(n \log n) \)  \( S = O(n) \)  \( Q = O(\log n + t) \)
\[ U = O(\log n) \]
output size
use sorted list
use balanced binary search trees

\( k = 2 \) even static case is interesting
\( CS 240 \): quad trees, k-d trees, range trees

\[
\sqrt{n} = 2^{\log n / 2}
\]

quad tree  
divide squares  
into 4 subsquares  
repeat until each  
square has 0 or 1 pts.

k-d tree  
alternately divide  
pts in half vertically  
then horizontally.

\[
P = O(n \log n) \quad S = O(n) \quad Q = \Theta(\sqrt{n} + t)
\]
output size
quad trees have the same run-times. \( \sqrt{n} \) is bad!
**Range Trees:**

- improve \( Q \) at the expense of \( S \)
- Make a balanced binary search tree
- Leaves = points sorted by \( x \)-coord

\[ D(v) = \text{descendants of node } v \text{ associated with slab from } x = v_L \text{ to } x = v_R \]

At node \( v \), attach array \( A(v) \) — pts in \( D(v) \) sorted by \( y \)

- \( S \in \mathcal{O}(\log n) \) — each point is in \( D(v) \) for \( \log n \) \( v \)'s
- \( P \in \mathcal{O}(n\log n) \) — sort by \( x \) to make tree
  - sort by \( y \) to make lists \( A(v) \)

How to query rectangle \( R \):

- search tree for \( x_1 \) and \( x_2 \)
- the points we want are at leaves between \( x_1 \) and \( x_2 \)
  but we must filter by \( y \)
Look at nodes $z$ — $O(\log n)$ of them
- right children of nodes on search path root $\rightarrow x_1$
- left children of nodes on search path root $\rightarrow x_2$
They correspond to disjoint slabs whose union is $[x_1 \cdots x_2]$.

For each $z$ (each slab),
do binary search in $A(z)$ to get points between $y_1$ and $y_2$.
$O(\log n + \text{output})$ per slab.

Since the slabs are disjoint, we don't repeat output.
Thus $Q \in O(\log^2 n + t)$, $t = \text{output size}$.

**Fractional cascading**
- improve $Q$ from $O(\log^2 n + t)$ to $O(\log n + t)$
- idea: in each $x$-slab we repeat search for same $y_1, y_2$
  That's wasteful!

Consider node $z$ child $w$.

Consider node $z$

\[ A(z) : \begin{array}{cccccc}
1 & 3 & 4 & 7 & 11 & 12 \\
\end{array} \]

\[ A(w) : \begin{array}{cccc}
1 & 4 & 7 & 11 \\
\end{array} \]

Keep ptrs from each element in $z$'s list
to corresponding element (or next higher)
in $w$'s list.

Gives $O(\log n + t)$

We search once for $y_1, y_2$
in list of root and follow ptrs.
Point Location
- Plane divided into disjoint polygonal regions
  preprocess to query: given a pt, which region contains it.

Regions might come from “closest to center”

In 1-D

Balanced binary search tree

P = O(n log n)
S = O(n)
Q = O(log n)
In 2D

Divide into slabs by adding vertical line at every pt.

given query pt \( x \), find correct slab \( O(\log n) \)
then do binary search by \( y \) — \( O(\log n) \)
(still works even though lines not constant \( y \))

\[ Q = O(\log n) \] — excellent!

Space \( S = \Theta(n^2) \)

Each of \( \frac{n}{2} \) lines is in \( \frac{n}{2} \) slabs

From one slab to next — few changes.

Make a BST for leftmost slab and update for subsequent slabs.

Total # updates to BST is \( O(n) \) — every segment gets inserted once and deleted once.
Idea: update a BST and search it in the past.
\[
\begin{array}{c|c|c|c}
\text{BST}_1 & \text{BST}_2 & \text{BST}_3 \\
\hline
\text{t=1} & \text{t=2} & \text{t=3} \\
\hline
\end{array}
\Rightarrow \text{time } t
\]
called "Persistent Data Structure".
e.g. Facebook friends, query in past.
Partial Persistence — query in past
— update only most recent
Full Persistence — can change past (Inception!)

Driscoll, Tarjan '89.
add partial persistence to any data structure (this is hard to do)
for planar point location this gives:
\[
\begin{align*}
P &= O(n \log n) \\
S &= O(n) \\
Q &= O(\log n)
\end{align*}
\]
\[
\begin{array}{c}
\text{same as 1-D.}
\end{array}
\]