Planar Point Location

store planar subdivision with \( n \) vertices

s.t. given query pt \( q \), find region containing \( q \)

- equivalently: find line segment immediately above \( q \).

by Euler's formula,

# edges/faces = \( O(n) \)

---

Method 0:

divide into \( n \) vertical slabs

store y-sorted list in each slab

\( \Rightarrow \) query time \( O(\log n) \)

(binary search in \( x \) + binary search in \( y \))

space \( O(n^2) \)

preproc time \( O(n^2 \log n) \)

---

Method 1: Segment Tree

given \( n \) segs intersecting slab \( \sigma \),

divide by median \( x \)

remove all long segs in \( \sigma \)

& store them in y-sorted list

recurse in left & right subslabs

---

Def: \( s \) is long in \( \sigma \) if it completely cuts across \( \sigma \)
Each seg is stored $O(\log n)$ times

$\Rightarrow$ Space $O(n \log n)$

(Prepare time $O(n \log n)$)

Query algo:

Binary search in each slab containing $q$:

$\Rightarrow O(\log n)$ binary searches

$\Rightarrow O(\log^2 n)$ time
Method 2: Segment Tree + "Fractional Cascading" (Chazelle, Guibas '86)

Idea: To speed up query,

- pass fraction \( b \) of the list of each node \( u \)
  - to the list of each child \( v \) of \( u \)

\[ \text{e.g. } b = 3 \]

\[ L(u) = \text{original g-sorted list at } u \]

\[ L^+(u) = \text{"augmented" list} \]

\[ \text{Sample}(L) = \text{take one after every } 5^{th} \text{ element in } L \]

For each child \( v \) of \( u \),

- define \( L^+(v) = L(v) \cup \text{Sample}(L^+(u)) \)

- Store ptrs between \( L^+(v) \) and \( L(v) \), \( \text{Sample}(L^+(u)) \).

If we know succ of \( q \) in \( L^+(v) \),

\[ \Rightarrow \text{know succ of } q \text{ in } L(v) \]

in \( O(1) \) time

\[ &\text{and in } \text{Sample}(L^+(u)) \]

\[ \Rightarrow \text{find succ in } L^+(u) \text{ in } O(b) \text{ time} \]

Query time \( O(\log n + (\log n)^2) = O(\log n) \)

Visit binary search at leaf
\[
\sum_u L(u) \left( 1 + \frac{2}{b} + \left(\frac{2}{b}\right)^2 + \left(\frac{2}{b}\right)^3 + \ldots \right) = O\left( \sum_u L(u) \right) \text{ for } b > 2
= \boxed{O(n \log n)}
\]

[can reduce space to \(O(n)\) with additional ideas.]

Method 3: Trapezoid Tree (Preparata '81)
idea: tree of vertical trapezoids instead of slabs

Given \( n \) segs intersecting trapezoid \( T \):

- if no long segs, divide \( T \) by median \( x \)
- else divide \( T \) by all its long segs

multi-leaf node!

\[ O(\log n) \]

space as before

Query: naively \( O(\log n \cdot \log n) = O(\log^2 n) \)

binary search at multi-leaf node
Fact: given \( n \) elements \( y_1, \ldots, y_m \) in 1D,
with weights \( w_1, \ldots, w_m \), \( \sum w_i = W \),
can find pred/succ \( y_i \) of any query pt \( q \)
in \( O(\log \frac{W}{w_i} + 1) = O(\log W - \log w_i + 1) \) time.

**Pf:** by weighted binary search (using weighted median)

**Query algm:**
at each multi-deg node, do weighted search
with weight \( (z) = \# \) segs intersecting \( z \)

\[
\Rightarrow \text{total query time } O\left(\log W_0 - \log W_1 + 1 \right.
\]
\[
+ \log W_1 - \log W_2 + 1 \}
\]
\[
+ \ldots \right)
\]
\[
= O\left(\log W_0 + \log n \right)
\]
\[
= O\left(\log n \right)
\]

(Rmk: related to BSP tree)
(Rmk: for tree-like structures, \( \Omega(n \log n) \) space)

**Method 4:** Persistent Search Tree (Sarnak-Tarjan '86)

idea - back to Method 0 (slabs)

sweep from left to right
if in list and pt,
insert to list
and store in balanced search tree
if right empty,
delete

remember history of all changes to search tree
& "persistence" - ability to query in past
Method 5: Planar Separators (Lipton-Tarjan '77)

**Theorem:** Given a triangulated planar graph $G$ with $n$ faces, can find subset $R$ of $O(m)$ edges that divide $G$ into 2 regions with $\leq 2^m$ faces.

**Generalized Theorem:** For any given $b$, can find subset $R$ of $O(b)$ edges that divide $G$ into regions with $\leq b$ faces.

(P.f. apply Theorem recursively...)

**Query Time:** $O(\log n + \log n) = O(\log n)$
- Initial binary search
- Time to query in search tree

**Space:** $O(n \log n)$

**Preproc Time:** $O(n \log n)$

(can do rotations...)

(by flattening nodes to do less copying)