Computational Geometry Lecture

Point Location

Chih-Hung Liu · Tamara Mchedlidze
20.06.2018
Motivation

Given a position \( p = (p_x, p_y) \) in a map, determine in which country \( p \) lies.
Motivation

Given a position \( p = (p_x, p_y) \) in a map, determine in which country \( p \) lies.
Motivation

Given a position $p = (p_x, p_y)$ in a map, determine in which country $p$ lies.

more precisely:
Find a data structure for efficiently answering such point location queries.
Motivation

Given a position \( p = (p_x, p_y) \) in a map, determine in which country \( p \) lies.

**more precisely:** Find a data structure for efficiently answering such point location queries.

The map is modeled as a subdivision of the plane into disjoint polygons.
Problem Setting
Problem Setting

**Goal:** Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.
Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.
Problem Setting

**Goal:** Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Think for 2 minutes!
**Goal:** Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

**Solution:** Partition $S$ at points into vertical slabs.
Problem Setting

**Goal:** Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

**Solution:** Partition $S$ at points into vertical slabs.
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query: find correct slab
Problem Setting

**Goal:** Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

**Solution:** Partition $S$ at points into vertical slabs.

**Query:** find correct slab
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:
- find correct slab
- search this slab
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:
- find correct slab
- search this slab
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:  
  - find correct slab
  - search this slab

\{ 2 \text{ binary searches} \}
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:  
- find correct slab
- search this slab

$O(\log n)$ time 

2 binary searches
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:
- find correct slab
- search this slab

But:

$O(\log n)$ time

2 binary searches
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:
- find correct slab
- search this slab

But: Space?

$O(\log n)$ time

2 binary searches
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:  
- find correct slab
- search this slab

But: Space? $\Theta(n^2)$
Problem Setting

Goal: Given subdivision $S$ of the plane with $n$ segments, construct data structure for fast point location queries.

Solution: Partition $S$ at points into vertical slabs.

Query:
- find correct slab
- search this slab

$O(\log n)$ time

But: Space? $\Theta(n^2)$

Question: lower bound example?
Observation: Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** *Trapezoidal map* $T(S)$
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** Trapezoidal map $T(S)$
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** *Trapezoidal map* $\mathcal{T}(S)$
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** *Trapezoidal map $T(S)$*

edge from each endpoint vertically up and down to neighboring segment
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** Trapezoidal map $T(S)$

- edge from each endpoint vertically up and down to neighboring segment
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** Trapezoidal map $T(S)$

- edge from each endpoint vertically up and down to neighboring segment
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** Trapezoidal map $T(S)$

edge from each endpoint vertically up and down to neighboring segment
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** Trapezoidal map $T(S)$

edge from each endpoint vertically up and down to neighboring segment
Observation: Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $S$ with lower complexity!

Solution: Trapezoidal map $\mathcal{T}(S)$
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** *Trapezoidal map $T(S)$*
Reducing the Complexity

**Observation:** Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

**Goal:** Find a suitable refinement of $S$ with lower complexity!

**Solution:** Trapezoidal map $T(S)$

Compare to slab partition
Reducing the Complexity

Observation: Slab partition is a refinement $S'$ of $S$ into (possibly degenerate) trapezoids.

Goal: Find a suitable refinement of $S$ with lower complexity!

Solution: Trapezoidal map $T(S)$

Assumption: $S$ is in general position, i.e., no two segment endpoints have the same $x$-coordinate
Notation

**Definition:** A *side* of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.
**Definition:** A *side* of a face of $T(S)$ is a segment of maximal length contained in face boundary.
Notation

**Definition:** A *side* of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.
Notation

Definition: A *side* of a face of $T(S)$ is a segment of maximal length contained in face boundary.
**Notation**

**Definition:** A *side* of a face of $T(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $T(S)$ has:
Notation

**Definition:** A side of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
Notation

**Definition:** A side of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- two non-vertical sides
**Notation**

**Definition:** A *side* of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.

![Diagram](image)

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- two non-vertical sides

**Left side:**

![Diagram](image)
### Notation

**Definition:** A side of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- two non-vertical sides

**Left side:**

- $\Delta$
- $\Delta$
Notation

**Definition:** A *side* of a face of $T(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\implies$ each face $\Delta$ of $T(S)$ has:
- one or two vertical sides
- two non-vertical sides

**Left side:**

- $\Delta$
- $\Delta$
- $\Delta$
Notation

**Definition:** A side of a face of $T(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $T(S)$ has:
- one or two vertical sides
- two non-vertical sides

**Left side:**

\[
\begin{align*}
\text{top}(\Delta) & \quad \text{bot}(\Delta) \\
\Delta & \\
\Delta & \\
\Delta & \\
\Delta & \\
\Delta & \\
\Delta & \\
\Delta &
\end{align*}
\]
**Notation**

**Definition:** A *side* of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- two non-vertical sides

**Left side:**

- $\Delta$
- $\Delta$
- $\Delta$
- $\Delta$
- $\Delta$
**Notation**

**Definition:** A side of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- two non-vertical sides

**Left side:**
- $\text{leftp}(\Delta)$
**Notation**

**Definition:** A *side* of a face of $\mathcal{T}(S)$ is a segment of maximal length contained in face boundary.

**Observation:** $S$ in general position $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- two non-vertical sides

**Left side:**

- $\text{leftp}(\Delta)$
- analogous: $\text{rightp}(\Delta)$
Complexity of the Trapezoidal Map

**Obs.:** A trapezoid $\Delta$ is uniquely defined by $\text{bot}(\Delta)$, $\text{top}(\Delta)$, $\text{leftp}(\Delta)$ and $\text{rightp}(\Delta)$.
Complexity of the Trapezoidal Map

Obs.: A trapezoid $\Delta$ is uniquely defined by $\text{bot}(\Delta)$, $\text{top}(\Delta)$, $\text{left}_p(\Delta)$ and $\text{right}_p(\Delta)$. 
Complexity of the Trapezoidal Map

Obs.: A trapezoid $\Delta$ is uniquely defined by $\text{bot}(\Delta)$, $\text{top}(\Delta)$, $\text{leftp}(\Delta)$ and $\text{rightp}(\Delta)$.

Lemma 1: The trapezoidal map $\mathcal{T}(S)$ of a set $S$ of $n$ segments in general position contains at most $6n + 4$ vertices and at most $3n + 1$ trapezoids.
Complexity of the Trapezoidal Map

**Obs.:** A trapezoid $\Delta$ is uniquely defined by $\text{bot}(\Delta)$, $\text{top}(\Delta)$, $\text{leftp}(\Delta)$ and $\text{rightp}(\Delta)$.

**Lemma 1:** The trapezoidal map $T(S)$ of a set $S$ of $n$ segments in general position contains at most $6n + 4$ vertices and at most $3n + 1$ trapezoids.
Search Structure

**Goal:** Compute the trapzoidal map \( \mathcal{T}(S) \) and simultaneously a data structure \( \mathcal{D}(S) \) for point location in \( \mathcal{T}(S) \).

\( \mathcal{T}(S) \)

\( \mathcal{D}(S) \) is a DAG with:

- **x-node** for point \( p \) tests left/right of \( p \)
- **y-node** for segment \( s \) tests above/below \( s \)
- **leaf node** for trapezoid \( \Delta \)
Incremental Algorithm

\text{TrapezoidalMap}(S)

\textbf{Input:} set $S = \{s_1, \ldots, s_n\}$ of crossing-free segments \\
\textbf{Output:} trapezoidal map $\mathcal{T}(S)$ and search structure $\mathcal{D}(S)$ \\
initialize $\mathcal{T}$ and $\mathcal{D}$ for $R = \text{BBox}(S)$

\textbf{for} $i \leftarrow 1$ \textbf{to} $n$ \textbf{do} \\
\hspace{1em} $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$ \\
\hspace{1em} $\mathcal{T} \leftarrow \mathcal{T} \setminus H$ \\
\hspace{1em} $\mathcal{T} \leftarrow \mathcal{T} \cup$ newly created trapezoids of $s_i$ \\
\hspace{1em} $\mathcal{D} \leftarrow$ replace leaves for $H$ by nodes and leaves for new trapezoids

return $(\mathcal{T}, \mathcal{D})$
Incremental Algorithm

TrapezoidalMap(S)

**Input:** set $S = \{s_1, \ldots, s_n\}$ of crossing-free segments  
**Output:** trapezoidal map $\mathcal{T}(S)$ and search structure $\mathcal{D}(S)$  
initialize $\mathcal{T}$ and $\mathcal{D}$ for $R = \text{BBox}(S)$

for $i \leftarrow 1$ to $n$ do

  $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$  
  $\mathcal{T} \leftarrow \mathcal{T} \setminus H$  
  $\mathcal{T} \leftarrow \mathcal{T} \cup$ newly created trapezoids of $s_i$  
  $\mathcal{D} \leftarrow$ replace leaves for $H$ by nodes and leaves for new trapezoids

return $(\mathcal{T}, \mathcal{D})$

**Problem:** Size of $\mathcal{D}$ and query time depend on insertion order
Incremental Algorithm

TrapezoidalMap($S$)

**Input:** set $S = \{s_1, \ldots, s_n\}$ of crossing-free segments  
**Output:** trapezoidal map $\mathcal{T}(S)$ and search structure $\mathcal{D}(S)$

initialize $\mathcal{T}$ and $\mathcal{D}$ for $R = \text{BBox}(S)$

$S \leftarrow \text{RandomPermutation}(S)$

for $i \leftarrow 1$ to $n$ do

$H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$

$\mathcal{T} \leftarrow \mathcal{T} \setminus H$

$\mathcal{T} \leftarrow \mathcal{T} \cup$ newly created trapezoids of $s_i$

$\mathcal{D} \leftarrow$ replace leaves for $H$ by nodes and leaves for new trapezoids

return $(\mathcal{T}, \mathcal{D})$

**Problem:** Size of $\mathcal{D}$ and query time depend on insertion order

**Solution:** Randomization!
Randomized Incremental Algorithm

**Invariant:** \( \mathcal{T} \) is trapezoidal map for \( S_i = \{s_1, \ldots, s_i\} \) and \( \mathcal{D} \) is corresponding search structure
Randomized Incremental Algorithm

**Invariant:** \( \mathcal{T} \) is trapezoidal map for \( S_i = \{s_1, \ldots, s_i\} \) and \( \mathcal{D} \) is corresponding search structure

**Initialization:** \( \mathcal{T} = \mathcal{T}(\emptyset) = \mathcal{R} \) and \( \mathcal{D} = (\mathcal{R}, \emptyset) \)
Randomized Incremental Algorithm

**Invariant:** $\mathcal{T}$ is trapezoidal map for $S_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

**Initialization:** $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $\mathcal{D} = (R, \emptyset)$

**Step 1:** $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$
Randomized Incremental Algorithm

**Invariant:** \( \mathcal{T} \) is trapezoidal map for \( S_i = \{s_1, \ldots, s_i\} \) and \( \mathcal{D} \) is corresponding search structure

**Initialization:** \( \mathcal{T} = \mathcal{T}(\emptyset) = R \) and \( \mathcal{D} = (R, \emptyset) \)

**Step 1:** \( H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\} \)
Randomized Incremental Algorithm

**Invariant:** $\mathcal{T}$ is trapezoidal map for $S_i = \{s_1, \ldots, s_i\}$ and $D$ is corresponding search structure

**Initialization:** $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $D = (R, \emptyset)$

**Step 1:** $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$

![Diagram](image-url)
Randomized Incremental Algorithm

Invariant: $\mathcal{T}$ is trapezoidal map for $S_i = \{s_1, \ldots, s_i\}$ and $\mathcal{D}$ is corresponding search structure

Initialization: $\mathcal{T} = \mathcal{T}(\emptyset) = R$ and $\mathcal{D} = (R, \emptyset)$

Step 1: $H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\}$

Task: How do you find the set $H$ of trapezoids from left to right?
Randomized Incremental Algorithm

Invariant: \( \mathcal{T} \) is trapezoidal map for \( S_i = \{s_1, \ldots, s_i\} \) and \( \mathcal{D} \) is corresponding search structure

Initialization: \( \mathcal{T} = \mathcal{T}(\emptyset) = R \) and \( \mathcal{D} = (R, \emptyset) \)

Step 1: \( H \leftarrow \{\Delta \in \mathcal{T} \mid \Delta \cap s_i \neq \emptyset\} \)

Task: How do you find the set \( H \) of trapezoids from left to right?

\[
\Delta_0 \leftarrow \text{FindTrapezoid}(p_i, \mathcal{D}); j \leftarrow 0
\]

\begin{algorithmic}
\While {right endpoint \( q_i \) right of \( \text{rightp}(\Delta_j) \)}
\If {rightp(\( \Delta_j \)) above \( s_i \)}
\State \( \Delta_{j+1} \leftarrow \) lower right neighbor of \( \Delta_j \)
\Else
\State \( \Delta_{j+1} \leftarrow \) upper right neighbor of \( \Delta_j \)
\EndIf
\State \( j \leftarrow j + 1 \)
\EndWhile
\begin{algorithmic}
\Return \( \Delta_0, \ldots, \Delta_j \)
\end{algorithmic}
Updating $\mathcal{T}(S)$ and $\mathcal{D}(S)$

**Step 2: Update $\mathcal{T}$ and $\mathcal{D}$**

- Case 1: $s_i \subset \Delta_0$

\[
\begin{align*}
\Delta_0 & \quad p_i \quad s_i \quad q_i \\
\mathcal{T}(S_{i-1}) & \rightarrow \\
\mathcal{T}(S_i) & \quad A \quad B \quad C \quad D
\end{align*}
\]
Updating $\mathcal{T}(S)$ and $\mathcal{D}(S)$

**Step 2:** Update $\mathcal{T}$ and $\mathcal{D}$

- **Case 1:** $s_i \subset \Delta_0$

$$D(S_{i-1}) \quad \Rightarrow \quad \mathcal{D}(S_i)$$
Updating $\mathcal{T}(S)$ and $\mathcal{D}(S)$

**Step 2: Update $\mathcal{T}$ and $\mathcal{D}$**

- **Case 1:** $s_i \subset \Delta_0$
- **Case 2:** $|\mathcal{T} \cap s_i| \geq 2$
Updating $\mathcal{T}(S)$ and $\mathcal{D}(S)$

**Step 2:** Update $\mathcal{T}$ and $\mathcal{D}$

- Case 1: $s_i \subset \Delta_0$
- Case 2: $|\mathcal{T} \cap s_i| \geq 2$

![Diagram showing the updating process of $\mathcal{T}(S_{i-1})$ and $\mathcal{D}(S_{i-1})$ to $\mathcal{T}(S_i)$ and $\mathcal{D}(S_i)$]
Updating $\mathcal{T}(S)$ and $\mathcal{D}(S)$

**Step 2: Update $\mathcal{T}$ and $\mathcal{D}$**

- Case 1: $s_i \subset \Delta_0$
- Case 2: $|\mathcal{T} \cap s_i| \geq 2$
**Analysis**

**Thm 1:** The algorithm computes the trapezoidal map $\mathcal{T}(S)$ and the search structure $\mathcal{D}$ for a set $S$ of $n$ segments in expected $O(n \log n)$ time. The expected size of $\mathcal{D}$ is $O(n)$ and the expected query time is $O(\log n)$. 
Analysis

**Thm 1:** The algorithm computes the trapezoidal map $\mathcal{T}(S)$ and the search structure $\mathcal{D}$ for a set $S$ of $n$ segments in expected $O(n \log n)$ time. The expected size of $\mathcal{D}$ is $O(n)$ and the expected query time is $O(\log n)$.

**Observations:**
- worst case: size of $\mathcal{D}$ is quadratic and query time is linear
- hope: that happens rarely!
- consider expected time and size over all $n!$ permutations of $S$
- the theorem holds independently of the input set $S$
Analysis

**Thm 1:** The algorithm computes the trapezoidal map $T(S)$ and the search structure $D$ for a set $S$ of $n$ segments in expected $O(n \log n)$ time. The expected size of $D$ is $O(n)$ and the expected query time is $O(\log n)$.

**Observations:**
- worst case: size of $D$ is quadratic and query time is linear
- hope: that happens rarely!
- consider expected time and size over all $n!$ permutations of $S$
- the theorem holds independently of the input set $S$

**Proof:**
- define random variables and consider their expected values
- perform *backward analysis*
- → details on blackboard
Worst-Case Consideration

So far: expected query time for arbitrary point is $O(\log n)$

But: each permutation could have a very bad (worst case) query point
Worst-Case Consideration

So far: expected query time for arbitrary point is $O(\log n)$

But: each permutation could have a very bad (worst case) query point

Lemma 2: Let $S$ be a set of $n$ crossing-free segments, let $q$ be a query point and let $\lambda > 0$. Then

$$\Pr[\text{search path for } q \text{ longer than } 3\lambda \ln(n + 1)] \leq 1/(n + 1)^{\lambda \ln 1.25 - 1}.$$  

No proof. (or see Chapter 6.4)
Worst-Case Consideration

So far: expected query time for arbitrary point is $O(\log n)$

But: each permutation could have a very bad (worst case) query point

**Lemma 2:** Let $\mathcal{S}$ be a set of $n$ crossing-free segments, let $q$ be a query point and let $\lambda > 0$. Then
\[
\Pr[\text{search path for } q \text{ longer than } 3\lambda \ln(n + 1)] \\
\leq \frac{1}{(n + 1)^{\lambda \ln 1.25 - 1}}.
\]

**Lemma 3:** Let $\mathcal{S}$ be a set of $n$ crossing-free segments and $\lambda > 0$. Then
\[
\Pr[\text{max. search path in } \mathcal{D} \text{ longer than } 3\lambda \ln(n + 1)] \\
\leq \frac{2}{(n + 1)^{\lambda \ln 1.25 - 3}}.
\]
Worst-Case Consideration

So far: expected query time for arbitrary point is $O(\log n)$

But: each permutation could have a very bad (worst case) query point

Lemma 2: Let $\mathcal{S}$ be a set of $n$ crossing-free segments, let $q$ be a query point and let $\lambda > 0$. Then

$$\Pr[\text{search path for } q \text{ longer than } 3\lambda \ln(n + 1)] \leq 1/(n + 1)^{\lambda \ln 1.25 - 1}.$$

Lemma 3: Let $\mathcal{S}$ be a set of $n$ crossing-free segments and $\lambda > 0$. Then

$$\Pr[\text{max. search path in } D \text{ longer than } 3\lambda \ln(n + 1)] \leq 2/(n + 1)^{\lambda \ln 1.25 - 3}.$$

Thm 2: Let $\mathcal{S}$ be a subdivision of the plane with $n$ edges. There is a search structure for point location within $\mathcal{S}$ that has $O(n)$ space and $O(\log n)$ query time.
Degenerate Inputs

Two assumptions:
- no two segment endpoints have the same $x$-coordinates
- always unique answers (left/right) on the search path
Degenerate Inputs

Two assumptions:

- no two segment endpoints have the same $x$-coordinates
- always unique answers (left/right) on the search path

**solution:** symbolic shear transformation

\[ \varphi : (x, y) \mapsto (x + \varepsilon y, y) \]
Degenerate Inputs

Two assumptions:
- no two segment endpoints have the same $x$-coordinates
- always unique answers (left/right) on the search path

**solution:** symbolic shear transformation

$$\varphi : (x, y) \mapsto (x + \varepsilon y, y)$$

Here $\varepsilon > 0$ is chosen such that the $x$-order $<$ of the points does not change.
Degenerate Inputs

- Effect 1: lexicographic order
- Effect 2: affine map $\varphi$ maintains point–line relations
Degenerate Inputs

- Effect 1: lexicographic order
- Effect 2: affine map $\varphi$ maintains point–line relations
- Run algorithm for $\varphi S = \{ \varphi s \mid s \in S \}$ and $\varphi p$. 
Degenerate Inputs

- Effect 1: lexicographic order
- Effect 2: affine map \( \varphi \) maintains point–line relations
- Run algorithm for \( \varphi S = \{ \varphi s \mid s \in S \} \) and \( \varphi p \).

Two basic operations for constructing \( \mathcal{T} \) and \( \mathcal{D} \):
1. Is \( q \) left or right of the vertical line through \( p \)?
2. Is \( q \) above or below the segment \( s \)?
Degenerate Inputs

- Effect 1: lexicographic order
- Effect 2: affine map \( \varphi \) maintains point–line relations
- Run algorithm for \( \varphi S = \{ \varphi s \mid s \in S \} \) and \( \varphi p \).
- Two basic operations for constructing \( T \) and \( D \):
  1. is \( q \) left or right of the vertical line through \( p \)?
  2. is \( q \) above or below the segment \( s \)?
- Locating a point \( q \) in \( T(S) \) works by locating \( \varphi q \) in \( T(\varphi S) \).

→ see Chapter 6.3 in [De Berg et al. 2008]
Discussion

Are there similar methods for higher dimensions?
Discussion

Are there similar methods for higher dimensions?

The currently best three-dimensional data structure uses $O(n \log n)$ space and $O(\log^2 n)$ query time [Snoeyink ’04]. Whether linear space and $O(\log n)$ query time is possible is an open question. In even higher dimensions efficient methods are known only for special hyper plane subdivisions.
Discussion

Are there similar methods for higher dimensions?

The currently best three-dimensional data structure uses $O(n \log n)$ space and $O(\log^2 n)$ query time [Snoeyink ’04]. Whether linear space and $O(\log n)$ query time is possible is an open question. In even higher dimensions efficient methods are known only for special hyper plane subdivisions.

Are there dynamic data structures that allow insertions and deletions?
Discussion

Are there similar methods for higher dimensions?

The currently best three-dimensional data structure uses $O(n \log n)$ space and $O(\log^2 n)$ query time [Snoeyink ’04]. Whether linear space and $O(\log n)$ query time is possible is an open question. In even higher dimensions efficient methods are known only for special hyper plane subdivisions.

Are there dynamic data structures that allow insertions and deletions?

Dynamic data structures for point location are well known, see the survey by [Chiang, Tamassia ’92]. A more recent example by [Arge et al. ’06] needs $O(n)$ space, $O(\log n)$ query time and $O(\log n)$ update time (insertions).