Computational Geometry

Winter term 2016/17

Point Localization

or

Where am I?

Lecture #6

Joachim Spoerhase
What’s the Problem?
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!
What’s the Problem?

Task: Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!
What's the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!
Task: Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!
What’s the Problem?

Task: Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

Solution: Pre-proc: Partition $S$ into slabs induced by vertices.
What’s the Problem?

**Task:** Given a planar subdivision $\mathcal{S}$ with $n$ segments, preprocess $\mathcal{S}$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $\mathcal{S}$ into slabs induced by vertices.
What's the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query:
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query: – find right slab
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query: – find right slab
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query: – find right slab
– search slab
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query: – find right slab
– search slab
What's the Problem?

Task: Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

Solution: Pre-proc: Partition $S$ into slabs induced by vertices.

Query: \[
\begin{align*}
&\text{– find right slab} \\
&\text{– search slab}
\end{align*}
\] \{ 2 bin. searches! \}
What’s the Problem?

Task: Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

Solution: Pre-proc: Partition $S$ into slabs induced by vertices.

Query: – find right slab
 – search slab \quad \{ \text{2 bin. searches!} \quad O(\log n) \text{ time!} \}
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query: $\begin{cases} - \text{find right slab} \\ - \text{search slab} \end{cases}$ $\rightarrow$ 2 bin. searches! $O(\log n)$ time!

**But:**
What’s the Problem?

Task: Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

Solution: Pre-proc: Partition $S$ into slabs induced by vertices.

Query: $\begin{align*}
- \text{find right slab} \\
- \text{search slab}
\end{align*}$ \text{2 bin. searches!} $O(\log n)$ time!

But: Space?
What’s the Problem?

Task: Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

Solution: Pre-proc: Partition $S$ into slabs induced by vertices.

Query: \[
\begin{align*}
\text{– find right slab} \\
\text{– search slab}
\end{align*}
\] \quad \{ \text{2 bin. searches!} \}

\quad \text{O}(\log n) \quad \text{time!}

But: Space? $\Theta(n^2)$
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query:  
- find right slab
- search slab \[ \{ \text{2 bin. searches!} \} \]

$O(\log n)$ time!

**But:** Space? $\Theta(n^2)$  
**Task:** Give lower-bound example!
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query:  
- find right slab
- search slab

\[ \{ \text{2 bin. searches!} \} \quad O(\log n) \text{ time!} \]

**But:** Space? $\Theta(n^2)$ Pre-proc?
What’s the Problem?

**Task:** Given a planar subdivision $S$ with $n$ segments, preprocess $S$ to allow for fast point location queries!

**Solution:** Pre-proc: Partition $S$ into slabs induced by vertices.

Query: \[
\begin{align*}
\text{find right slab} & \quad \text{2 bin. searches!} \\
\text{search slab} & \quad O(\log n) \text{ time!}
\end{align*}
\]

**But:** Space? $\Theta(n^2)$  Pre-proc? $O(n^2 \log n)$
Decreasing the Complexity

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.
Decreasing the Complexity

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!
Decreasing the Complexity

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

Observation: The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

Task: Find “good” refinement of $S$ of low complexity!

Solution: Trapezoidal map $T(S)$
Decreasing the Complexity

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

**Observation:**  The slab partition of $S$ is a *refinement* $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:**  Find “good” refinement of $S$ of low complexity!

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

**Observation:** The slab partition of $S$ is a *refinement* $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

**Observation:** The slab partition of $S$ is a *refinement* $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

![Diagram of trapezoidal map](image)
Decreasing the Complexity

**Observation:** The slab partition of $S$ is a refinement $S'$ of $S$ that consists of (possibly degenerate) trapezoids.

**Task:** Find “good” refinement of $S$ of low complexity!

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

**Assumption:** $S$ is in *general position*, that is, no two vertices have the same $x$-coordinates.
Notation

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

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

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

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

**Observation:** $S$ in gen. pos. $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
Notation

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

**Observation:** $S$ in gen. pos. $\Rightarrow$ 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 maximum length contained in the boundary of the face.

**Observation:** $S$ in gen. pos. $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- exactly 2 non-vertical sides
**Notation**

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

**Observation:** $S$ in gen. pos. $\Rightarrow$ each face $\Delta$ of $T(S)$ has:
- one or two vertical sides
- exactly 2 non-vertical sides
Notation

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

**Observation:** $S$ in gen. pos. $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- exactly 2 non-vertical sides

**Left side:**
**Notation**

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

**Observation:** $S$ in gen. pos. $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- exactly 2 non-vertical sides

**Left side:**

[Diagram showing the left side of a face with vertical and non-vertical sides marked.]
Notation

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

**Observation:** $S$ in gen. pos. $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- exactly 2 non-vertical sides

**Left side:**
**Notation**

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

**Observation:** $S$ in gen. pos. $\Rightarrow$ each face $\Delta$ of $\mathcal{T}(S)$ has:
- one or two vertical sides
- exactly 2 non-vertical sides
Notation

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

Observation: $S$ in gen. pos. $\Rightarrow$ each face $\Delta$ of $T(S)$ has:
- one or two vertical sides
- exactly 2 non-vertical sides

Left side:

$\text{leftp}(\Delta)$
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$. 
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$. 
Observe: A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$. 
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$. 
Complexity of $\mathcal{T}(S)$

Observe: A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

Lemma. $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq \boxed{6n+4}$ vtc and $\leq \boxed{3n+1}$ trapezoids.
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq \mathbf{6}n + 4$ vtc and $\leq \mathbf{3}n + 1$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are

– endpts of segments in $S$
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by top($\Delta$), bot($\Delta$), leftp($\Delta$), and rightp($\Delta$).

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are
- endpts of segments in $S$ $\leq 2n$
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq \boxed{6n+4}$ vtc and $\leq \boxed{3n+1}$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are
- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by top($\Delta$), bot($\Delta$), leftp($\Delta$), and rightp($\Delta$).

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq$ vtc and $\leq$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are

- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by top($\Delta$), bot($\Delta$), leftp($\Delta$), and rightp($\Delta$).

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq \text{vtc}$ and $\leq \text{trapezoids}$.

**Proof.** The vertices of $\mathcal{T}(S)$ are
- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$
- vertices of $R$
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $	ext{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq \color{#FF8C00}6n+4$ vtc and $\leq \color{#FF8C00}3n+1$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are
- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$
- vertices of $R$ $4$
Complexity of $\mathcal{T}(S)$

**Observe:**  A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments \Rightarrow $\mathcal{T}(S)$ has $\leq$ vtc and $\leq$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are

- endpts of segments in $S$ \leq 2n
- endpts of vertical extensions \leq 2 \cdot 2n
- vertices of $R$ 4
Complexity of $\mathcal{T}(S)$

Observe: A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

Lemma. $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are
- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$
- vertices of $R$ $\leq \frac{6n + 4}{4}$
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are

- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$ $\leq 6n + 4$
- vertices of $R$ $\leq 4$
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are
- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$ \[ \leq 6n + 4 \]
- vertices of $R$ $\leq 4$

Bound $\#$trapezoids via Euler or directly (segments/leftp).
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by top($\Delta$), bot($\Delta$), lefTp($\Delta$), and rightp($\Delta$).

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are
- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$
- vertices of $R$

Bound $\#\text{trapezoids}$ via Euler or directly (segments/lefTp).

**Approach:**
Complexity of $\mathcal{T}(S)$

Observe: A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

Lemma. $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

Proof. The vertices of $\mathcal{T}(S)$ are

- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$ \[ \leq 6n + 4 \]
- vertices of $R$

Bound $\#\text{trapezoids}$ via Euler or directly (segments/leftp).

Approach: Construct tapezoidal map $\mathcal{T}(S)$ and point-location data structure $\mathcal{D}(S)$ for $\mathcal{T}(S)$ incrementally!
Complexity of $\mathcal{T}(S)$

**Observe:** A face $\Delta$ of $\mathcal{T}(S)$ is uniquely defined by $\text{top}(\Delta)$, $\text{bot}(\Delta)$, $\text{leftp}(\Delta)$, and $\text{rightp}(\Delta)$.

**Lemma.** $S$ planar subdivision in gen. pos., with $n$ segments $\Rightarrow \mathcal{T}(S)$ has $\leq 6n + 4$ vtc and $\leq 3n + 1$ trapezoids.

**Proof.** The vertices of $\mathcal{T}(S)$ are
- endpts of segments in $S$ $\leq 2n$
- endpts of vertical extensions $\leq 2 \cdot 2n$
- vertices of $R$ $\leq 4$

$\big\{ \leq 6n + 4 \big\}$

Bound $\#\text{trapezoids}$ via Euler or directly (segments/leftp).

**Approach:** Construct tapezoidal map $\mathcal{T}(S)$ and point-location data structure $\mathcal{D}(S)$ for $\mathcal{T}(S)$ *incrementally*! algorithm-design paradigm!
The 1d-Problem

Given a set \( S \) of \( n \) real numbers...
The 1d-Problem

Given a set $S$ of $n$ real numbers...
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \hspace{1cm} $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

$i \in \{1, \ldots, n\}$
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$i \in \{1, \ldots, n\}$

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$$S_i := \{s_1, \ldots, s_{i-1}\}, \quad l_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$$

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
The 1d-Problem

Given a set $S$ of $n$ real numbers... $i \in \{1, \ldots, n\}$

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \quad $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

– pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
– locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \quad $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
The 1d-Problem

Given a set $S$ of $n$ real numbers...

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
- build $D_i$:

$$
S_{i-1} := \{s_1, \ldots, s_{i-1}\}, \quad I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}
$$
The 1d-Problem

Given a set $S$ of $n$ real numbers...

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
- build $D_i$: 

\[ (\ell, r) \]

\[ D_{i-1} \]
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \quad $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
- build $D_i$:
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $I_{i-1} :=$ set of conn. comp. of $\mathbb{R} \setminus S_{i-1}$

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
- build $D_i$:
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $l_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $l_{i-1}$ containing $s_i$
- build $D_i$:

Problem:
The 1d-Problem

Given a set $S$ of $n$ real numbers...

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
- build $D_i$:

Problem: _loooong_ search paths!
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \hspace{1cm} I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

Solution:
- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
- build $D_i$:

Problem: *looong* search paths!
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, $l_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

Solution:
- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $l_{i-1}$ containing $s_i$
- build $D_i$:

Problem: looong search paths!
The 1d-Problem

Given a set $S$ of $n$ real numbers...

Formally:

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \quad $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

**Solution:** *random!*

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
- build $D_i$:

**Problem:** *looong search paths!*
The 1d-Problem

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \hspace{1cm} $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

**Solution:** random!

- pick an arbitrary point $s_i$ from $S \setminus S_{i-1}$
- locate $s_i$ in the search structure $D_{i-1}$ of $S_{i-1}$
- split interval $(\ell, r)$ of $I_{i-1}$ containing $s_i$
- build $D_i$:

**Problem:** looong search paths!
Thm. The randomized-incremental algorithm preprocesses a set $S$ of $n$ reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.
1d Result

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \quad $l_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

**Thm.** The randomized-incremental algorithm preprocesses a set $S$ of $n$ reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

**Proof.** Let $q \in \mathbb{R}$ (wlog. $q \notin S$) and $l_i(q) = \arg\{l \in l_i : q \in l\}$. 
1d Result

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \quad $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

**Thm.** The randomized-incremental algorithm preprocesses a set $S$ of $n$ reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

**Proof.** Let $q \in \mathbb{R}$ (wlog. $q \notin S$) and $l_i(q) = \arg\{l \in I_i : q \in l\}$.

$$E[\text{query time in } D_n] =$$
1d Result

Given a set $S$ of $n$ real numbers...

$$S_{i-1} := \{s_1, \ldots, s_{i-1}\}, \quad I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$$

**Thm.** The randomized-incremental algorithm preprocesses a set $S$ of $n$ reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

**Proof.** Let $q \in \mathbb{R}$ (wlog. $q \not\in S$) and $l_i(q) = \arg\{l \in I_i : q \in l\}$.

$$E[\text{query time in } D_n] = E[\text{length search path in } D_n] =$$
1d Result

Given a set $S$ of $n$ real numbers...

$S_{i-1} := \{s_1, \ldots, s_{i-1}\}$, \hspace{1cm} $I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1}$

**Thm.** The randomized-incremental algorithm preprocesses a set $S$ of $n$ reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

**Proof.** Let $q \in \mathbb{R}$ (wlog. $q \not\in S$) and $l_i(q) = \text{arg}\{l \in I_i : q \in l\}$.

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } D_n] = E[\text{length search path in } D_n] =$
1d Result

Given a set \( S \) of \( n \) real numbers... 

\[ S_{i-1} := \{s_1, \ldots, s_{i-1}\}, \quad I_{i-1} := \text{set of conn. comp. of } \mathbb{R} \setminus S_{i-1} \]

**Thm.** The randomized-incremental algorithm preprocesses a set \( S \) of \( n \) reals in \( O(n \log n) \) expected time such that a query takes \( O(\log n) \) expected time.

**Proof.** Let \( q \in \mathbb{R} \) (wlog. \( q \not\in S \)) and \( I_i(q) = \text{arg}\{l \in I_i : q \in l\} \).

Define random variable \( X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else}. \end{cases} \)

\[ E[\text{query time in } D_n] = E[\text{length search path in } D_n] = E[\sum_{i=1}^n X_i] = \]
1d Result

Given a set $S$ of $n$ real numbers...

Define random variable $X_i = \begin{cases} 
1 & \text{if } I_i(q) \neq I_{i-1}(q), \\
0 & \text{else.}
\end{cases}$

Let $q \in \mathbb{R}$ (wlog. $q \not\in S$) and $I_i(q) = \text{arg}\{l \in I_i : q \in l\}$.

The randomized-incremental algorithm preprocesses a set $S$ of $n$ reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.

Proof. Let $q \in \mathbb{R}$ (wlog. $q \not\in S$) and $I_i(q) = \text{arg}\{l \in I_i : q \in l\}$.

Define random variable $X_i = \begin{cases} 
1 & \text{if } I_i(q) \neq I_{i-1}(q), \\
0 & \text{else.}
\end{cases}$

$E[\text{query time in } D_n] = E[\text{length search path in } D_n] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \ ?$
Expected Query Time of $\mathcal{D}_n$

Define random variable $X_i = \begin{cases} 
1 & \text{if } l_i(q) \neq l_{i-1}(q), \\
0 & \text{else.}
\end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$
Expected Query Time of $\mathcal{D}_n$

$E[X_i] =

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] = \sum_{i=1}^{n} E[X_i] = ?$
Expected Query Time of $\mathcal{D}_n$

$$E[X_i] = P[X_i = 1] = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else}. \end{cases}$$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] = E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = ?$$
Expected Query Time of $\mathcal{D}_n$

$$E[X_i] = P[X_i = 1] =$$

$$= \text{probability that } l_i(q) \neq l_{i-1}(q)$$

Define random variable $X_i = \begin{cases} 
1 & \text{if } l_i(q) \neq l_{i-1}(q), \\
0 & \text{else.}
\end{cases}$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$$

$$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$$
Expected Query Time of $\mathcal{D}_n$

$$E[X_i] = P[X_i = 1] =$$

= probability that $l_i(q) \neq l_{i-1}(q)$, i.e., $s_i \in l_{i-1}(q)$.

Define random variable $X_i = \begin{cases} 
1 & \text{if } l_i(q) \neq l_{i-1}(q), \\
0 & \text{else.}
\end{cases}$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$$

$$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$$
Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

$= \text{probability that } I_i(q) \neq I_{i-1}(q), \text{ i.e., } s_i \in I_{i-1}(q).$

**Backwards analysis:**

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$
Expected Query Time of $D_n$

$E[X_i] = P[X_i = 1] =$

= probability that $l_i(q) \neq l_{i-1}(q)$, i.e., $s_i \in l_{i-1}(q)$.

Backwards analysis: Consider $S_i$ fixed.

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } D_n] = E[\text{length search path in } D_n] =

= E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = ?$
Expected Query Time of $\mathcal{D}_n$

$E[X_i] = P[X_i = 1] =$

= probability that $I_i(q) \neq I_{i-1}(q)$, i.e., $s_i \in I_{i-1}(q)$.

**Backwards analysis:** Consider $S_i$ fixed.

If we remove a randomly chosen pt from $S_i$,

Define random variable $X_i = \begin{cases} 1 & \text{if } I_i(q) \neq I_{i-1}(q), \\ 0 & \text{else}. \end{cases}$

$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$

$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$
Expected Query Time of $\mathcal{D}_n$

$$E[X_i] = P[X_i = 1] =$$

= probability that $l_i(q) \neq l_{i-1}(q)$, i.e., $s_i \in l_{i-1}(q)$.

**Backwards analysis:** Consider $S_i$ fixed.
If we remove a randomly chosen pt from $S_i$, what's the probability that the interval containing $q$ changes?

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] =$$

$$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$$
Expected Query Time of $D_n$

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[X_i] = P[X_i = 1] = \text{probability that } l_i(q) \neq l_{i-1}(q), \text{ i.e., } s_i \in l_{i-1}(q)$.

**Backwards analysis:** Consider $S_i$ fixed.

If we remove a randomly chosen pt from $S_i$, what’s the probability that the interval containing $q$ changes?

– we have $i$ choices, identically distributed

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else}. \end{cases}$

$E[\text{query time in } D_n] = E[\text{length search path in } D_n] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$
Expected Query Time of $\mathcal{D}_n$

$$E[X_i] = P[X_i = 1] = \text{probability that } l_i(q) \neq l_{i-1}(q), \text{ i.e., } s_i \in l_{i-1}(q).$$

**Backwards analysis:** Consider $S_i$ fixed.

If we *remove* a randomly chosen pt from $S_i$, what’s the probability that the interval containing $q$ changes?

– we have $i$ choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] = \sum_{i=1}^{n} 1 \cdot E[X_i] = \sum_{i=1}^{n} E[X_i] = ?$$
Expected Query Time of $D_n$

$$E[X_i] = P[X_i = 1] =$$

= probability that $l_i(q) \neq l_{i-1}(q)$, i.e., $s_i \in l_{i-1}(q)$.

**Backwards analysis:**

Consider $S_i$ fixed.
If we remove a randomly chosen pt from $S_i$, what’s the probability that the interval containing $q$ changes?
– we have $i$ choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 
1 & \text{if } l_i(q) \neq l_{i-1}(q), \\
0 & \text{else.} 
\end{cases}$

$$E[\text{query time in } D_n] = E[\text{length search path in } D_n] =$$

$$= E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$$
Expected Query Time of $\mathcal{D}_n$

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$$E[X_i] = P[X_i = 1] = \frac{2}{i}$$

$= \text{probability that } l_i(q) \neq l_{i-1}(q), \text{i.e., } s_i \in l_{i-1}(q).$

**Backwards analysis:** Consider $S_i$ fixed.

If we *remove* a randomly chosen pt from $S_i$, what’s the probability that the interval containing $q$ changes?

– we have $i$ choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$$E[\text{query time in } \mathcal{D}_n] = E[\text{length search path in } \mathcal{D}_n] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} E[X_i] = ?$$
Expected Query Time of $D_n$

$$E[X_i] = P[X_i = 1] = \frac{2}{i}$$

= probability that $l_i(q) \neq l_{i-1}(q)$, i.e., $s_i \in l_{i-1}(q)$.

**Backwards analysis:** Consider $S_i$ fixed.

If we *remove* a randomly chosen pt from $S_i$, what’s the probability that the interval containing $q$ changes?

– we have $i$ choices, identically distributed

– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else}. \end{cases}$

$$E[\text{query time in } D_n] = E[\text{length search path in } D_n] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$$
Expected Query Time of $D_n$

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[X_i] = P[X_i = 1] = \frac{2}{i}$

= probability that $l_i(q) \neq l_{i-1}(q)$, i.e., $s_i \in l_{i-1}(q)$.

**Backwards analysis:** Consider $S_i$ fixed.

If we remove a randomly chosen pt from $S_i$, what’s the probability that the interval containing $q$ changes?
– we have $i$ choices, identically distributed
– at most two of these change the interval

Define random variable $X_i = \begin{cases} 1 & \text{if } l_i(q) \neq l_{i-1}(q), \\ 0 & \text{else.} \end{cases}$

$E[\text{query time in } D_n] = E[\text{length search path in } D_n] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \frac{2}{i}$

$O(\log n)$
Thm. The randomized-incremental algorithm preprocesses a set $S$ of $n$ reals in $O(n \log n)$ expected time such that a query takes $O(\log n)$ expected time.
The 2d-Problem

**Approach:** randomized-incremental construction of $T$ and $D$
The 2d-Problem

**Approach:** randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$
The 2d-Problem

**Approach:** randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

![Trapezoidal map diagram]
The 2d-Problem

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

point-location data structure (DAG)
trapezoidal map
The 2d-Problem

**Approach:** randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

point-location data structure (DAG)
trapezoidal map
The 2d-Problem

**Approach:** randomized-incremental construction of $T$ and $D$

point-location data structure (DAG)  
trapezoidal map
The 2d-Problem

**Approach:** randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment $s$
The 2d-Problem

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$
- use $\mathcal{D}$ to locate left endpoint of next segment $s$
- “walk” along $s$ through $\mathcal{T}$
The 2d-Problem

**Approach:** randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment $s$
- “walk” along $s$ through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting $s$
The 2d-Problem

**Approach:** randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment $s$
- “walk” along $s$ through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting $s$
- construct new trapezoids of $\mathcal{T}$ (adjacent to $s$)

point-location data structure (DAG)  
trapezoidal map
The 2d-Problem

Approach: randomized-incremental construction of $\mathcal{T}$ and $\mathcal{D}$

- use $\mathcal{D}$ to locate left endpoint of next segment $s$
- “walk” along $s$ through $\mathcal{T}$
- destroy all trapezoids of $\mathcal{T}$ intersecting $s$
- construct new trapezoids of $\mathcal{T}$ (adjacent to $s$)
- update $\mathcal{D}$

point-location data structure (DAG)  
trapezoidal map
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

$R = \text{BBox}(S); \mathcal{T}.\text{init}(); \mathcal{D}.\text{init}()$

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ to $n$ do

)
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

$R = \text{BBox}(S); \, \mathcal{T}.\text{init}(); \, \mathcal{D}.\text{init}()$

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

\textbf{for} $i = 1$ \textbf{to} $n$ \textbf{do}

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

\textbf{end for}
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

\[
\begin{align*}
\text{TrapezoidalMap(set } S \text{ of } n \text{ non-crossing segments)} \\
R &= \text{BBox}(S); \ \mathcal{T}.\text{init}(); \ \mathcal{D}.\text{init}() \\
(s_1, s_2, \ldots, s_n) &= \text{RandomPermutation}(S) \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\quad (\Delta_0, \ldots, \Delta_k) &= \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i) \\
\end{align*}
\]
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

\begin{align*}
\text{TrapezoidalMap(set } S \text{ of } n \text{ non-crossing segments)} \\
R &= \text{BBox}(S); \mathcal{T}.\text{init}(); \mathcal{D}.\text{init}() \\
(s_1, s_2, \ldots, s_n) &= \text{RandomPermutation}(S) \\
\text{for } i = 1 \text{ to } n \text{ do} \\
&\quad (\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}($\mathcal{T}$, $\mathcal{D}$, $s_i$) \\
&\quad \mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k) \\
&\quad \mathcal{D}.\text{remove leaves}(\Delta_0, \ldots, \Delta_k) \\
&\quad \mathcal{D}.\text{add leaves}(\text{new trapezoids incident to } s_i) \\
&\quad \mathcal{D}.\text{add new inner nodes})
\end{align*}
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

$R = \text{BBox}(S)$; $\mathcal{T}.\text{init}()$; $\mathcal{D}.\text{init}()$

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ to $n$ do

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

$\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k)$

$\mathcal{D}.\text{remove} \text{leaves}(\Delta_0, \ldots, \Delta_k)$

$\mathcal{D}.\text{add} \text{leaves}(\text{new trapezoids incident to } s_i)$

$\mathcal{D}.\text{add} \text{new inner nodes}()$
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

\[
\begin{align*}
\Delta_0 & \quad \Delta_1 \quad \Delta_2 \quad \Delta_3 \\
p_i \quad q_i \quad s_i
\end{align*}
\]

\[
\begin{align*}
\mathcal{T}(S_{i-1}) & \quad \mathcal{T}(S_i) \\

\text{TrapezoidalMap(set } S \text{ of } n \text{ non-crossing segments})
\end{align*}
\]

\[
R = \text{BBox}(S); \quad \mathcal{T}.\text{init()}; \quad \mathcal{D}.\text{init()}
\]

\[
(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)
\]

\[
\text{for } i = 1 \text{ to } n \text{ do}
\]

\[
(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)
\]

\[
\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k)
\]

\[
\mathcal{T}.\text{add(new trapezoids incident to } s_i)
\]

\[
\]
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

\[ R = \text{BBox}(S); \quad \mathcal{T}.\text{init}(); \quad \mathcal{D}.\text{init}() \]

\[(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)\]

\[\text{for } i = 1 \text{ to } n \text{ do}\]

\[ (\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i) \]

\[ \mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k) \]

\[ \mathcal{T}.\text{add}(\text{new trapezoids incident to } s_i) \]

\[ \]
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

$R = \text{BBox}(S); \mathcal{T}.\text{init}(); \mathcal{D}.\text{init}()$

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

\textbf{for } $i = 1$ \textbf{to } $n$ \textbf{do}

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

$\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k)$

$\mathcal{T}.\text{add}(\text{new trapezoids incident to } s_i)$
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

\[
\begin{align*}
\text{TrapezoidalMap(set } S \text{ of } n \text{ non-crossing segments)} & \\
R &= \text{BBox}(S); \mathcal{T}.\text{init(); } \mathcal{D}.\text{init()} \\
(s_1, s_2, \ldots, s_n) &= \text{RandomPermutation}(S) \\
\text{for } i = 1 \text{ to } n \text{ do} & \\
(\Delta_0, \ldots, \Delta_k) &= \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i) \\
\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k) & \\
\mathcal{T}.\text{add(new trapezoids incident to } s_i) & \\
\mathcal{D}.\text{remove_leaves}(\Delta_0, \ldots, \Delta_k) & \\
\end{align*}
\]
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

$R = \text{BBox}(S)$; $\mathcal{T}.\text{init}()$; $\mathcal{D}.\text{init}()$

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ to $n$ do

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

$\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k)$

$\mathcal{T}.\text{add}(\text{new trapezoids incident to } s_i)$

$\mathcal{D}.\text{remove} \text{leaves}(\Delta_0, \ldots, \Delta_k)$

}
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

$R = \text{BBox}(S); \; \mathcal{T}.\text{init}(); \; \mathcal{D}.\text{init}()$

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ to $n$ do

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

$\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k)$

$\mathcal{T}.\text{add}(\text{new trapezoids incident to } s_i)$

$\mathcal{D}.\text{remove\_leaves}(\Delta_0, \ldots, \Delta_k)$

$\mathcal{D}.\text{add\_leaves}(\text{new trapezoids incident to } s_i)$
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

\[
R = \text{BBox}(S); \quad \mathcal{T}.\text{init}(); \quad \mathcal{D}.\text{init}()
\]

\[(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)\]

\[\text{for } i = 1 \text{ to } n \text{ do} \]

\[\left(\Delta_0, \ldots, \Delta_k\right) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)\]

\[\mathcal{T}.\text{remove}(\Delta_0, \ldots, \Delta_k)\]

\[\mathcal{D}.\text{remove_leaves}(\Delta_0, \ldots, \Delta_k)\]

\[\mathcal{T}.\text{add(new trapezoids incident to } s_i)\]

\[\mathcal{D}.\text{add_leaves(new trapezoids incident to } s_i)\]
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

$R = \text{BBox}(S); \mathcal{T}.\text{init}(); \mathcal{D}.\text{init}()$

$(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1 \text{ to } n$ do

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

$\mathcal{T}.\text{remove}((\Delta_0, \ldots, \Delta_k))$

$\mathcal{T}.\text{add(new trapezoids incident to } s_i)$

$\mathcal{D}.\text{remove_leaves}((\Delta_0, \ldots, \Delta_k))$

$\mathcal{D}.\text{add_leaves(new trapezoids incident to } s_i)$

$\mathcal{D}.\text{add_new_inner_nodes()}$
Walking through $\mathcal{T}$ and Updating $\mathcal{D}$

TrapezoidalMap(set $S$ of $n$ non-crossing segments)

$R = \text{BBox}(S); \mathcal{T}\text{.init(); } \mathcal{D}\text{.init()}

(s_1, s_2, \ldots, s_n) = \text{RandomPermutation}(S)$

for $i = 1$ to $n$ do

$(\Delta_0, \ldots, \Delta_k) = \text{FollowSegment}(\mathcal{T}, \mathcal{D}, s_i)$

$\mathcal{T}\text{.remove}(\Delta_0, \ldots, \Delta_k)$

$\mathcal{T}\text{.add(new trapezoids incident to } s_i)$

$\mathcal{D}\text{.remove\_leaves}(\Delta_0, \ldots, \Delta_k)$

$\mathcal{D}\text{.add\_leaves(new trapezoids incident to } s_i)$

$\mathcal{D}\text{.add\_new\_inner\_nodes()}$
The 2d-Result

**Theorem.** TrapezoidalMap($S$) computes $\mathcal{T}(S)$ for a set of $n$ line segments in general position and a search structure $\mathcal{D}$ for $\mathcal{T}(S)$ in $O(n \log n)$ expected time.
The 2d-Result

**Theorem.** TrapezoidalMap(S) computes $T(S)$ for a set of $n$ line segments in general position and a search structure $D$ for $T(S)$ in $O(n \log n)$ expected time. The expected size of $D$ is $O(n)$ and the expected query time is $O(\log n)$. 
The 2d-Result

**Theorem.** TrapezoidalMap($S$) computes $\mathcal{T}(S)$ for a set of $n$ line segments in general position and a search structure $\mathcal{D}$ for $\mathcal{T}(S)$ in $O(n \log n)$ expected time. The expected size of $\mathcal{D}$ is $O(n)$ and the expected query time is $O(\log n)$.

**Invariant:** Before step $i$, $\mathcal{T}$ is a trapezoidal map for $S_{i-1}$ and $\mathcal{D}$ is a valid search structure for $\mathcal{T}$.

**Proof.**
- Correctness by loop invariant.
- Query time similar to 1d analysis.
  $\Rightarrow$ construction time
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$. 
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from D.root to } q)$. 
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$. 

$\Rightarrow T(q) = O(\text{length of the path from } D\text{.root to } q)$. 
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.  
⇒ $T(q) = O(\text{length of the path from } D.\text{root to } q)$.  

height($D$) increases by at most 3 in each step.
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.

height($D$) increases by at most 3 in each step. $\Rightarrow T(q) \leq$
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.  
$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.  

height($D$) increases by at most 3 in each step.  $\Rightarrow T(q) \leq 3n$. 
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.

height($D$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $D$:...
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.

height($D$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $D$:

$\Rightarrow$ average of $T(q)$ over
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.  
height($D$) increases by at most 3 in each step.  $\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $D$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.

height($D$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $D$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of $S$)
**Query Time**

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D\text{.root to } q)$. 

height($D$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the *expected* behaviour of $D$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of $S$)

$X_i := \# \text{ nodes that are added to the query path in iteration } i$. 
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$. 
$\Rightarrow T(q) = O(\text{length of the path from } \mathcal{D}.\text{root to } q)$. 

height($\mathcal{D}$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $\mathcal{D}$: 
$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of $S$) 

$X_i := \# \text{ nodes that are added to the query path in iteration } i$. 
$S$ and $q$ are fixed.
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$. 

height($D$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $D$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of $S$)

$X_i := \# \text{ nodes that are added to the query path in iteration } i$.

$S$ and $q$ are fixed.

$\Rightarrow X_i$ random variable that depends only on insertion order of $S$. 
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.

height($D$) increases by at most 3 in each step. $\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $D$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of $S$)

$X_i := \# \text{ nodes that are added to the query path in iteration } i$. $S$ and $q$ are fixed.

$\Rightarrow X_i$ random variable that depends only on insertion order of $S$.

$\Rightarrow$ expected path length from $D.\text{root to } q$ is
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.

height($D$) increases by at most 3 in each step.  $\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $D$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of $S$)

$X_i := \#\text{ nodes that are added to the query path in iteration } i$.

$S$ and $q$ are fixed.

$\Rightarrow X_i$ random variable that depends only on insertion order of $S$.

$\Rightarrow$ expected path length from $D.\text{root to } q$ is

$$\mathbb{E}[\sum_{i=1}^{n} X_i] =$$
Query Time

Let $T(q)$ be the query time for a fixed query pt $q$.

$\Rightarrow T(q) = O(\text{length of the path from } D.\text{root to } q)$.

The height of $D$ increases by at most 3 in each step.

$\Rightarrow T(q) \leq 3n$.

We are interested in the expected behaviour of $D$:

$\Rightarrow$ average of $T(q)$ over all $n!$ insertion orders (permut. of $S$)

$X_i := \# \text{ nodes that are added to the query path in iteration } i$.

$S$ and $q$ are fixed.

$\Rightarrow X_i$ random variable that depends only on insertion order of $S$.

$\Rightarrow$ expected path length from $D.\text{root}$ to $q$ is

$E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = ?$
Query Time (cont’d)

\[ p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \]
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \)

\[ \Rightarrow E[X_i] = \]
Query Time (cont’d)

\[ p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \]

\[ \Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \]
Query Time (cont’d)

\( p_i \) = prob. that the search path \( \Pi_q \) of \( q \) in \( \mathcal{D} \) contains a node that was created in iteration \( i \).

\[ \Rightarrow \mathbb{E}[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = \]
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \)

\[ \Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i \]
Query Time (cont’d)

$p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } D \text{ contains a node that was created in iteration } i.$

$\Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i$

$\Delta_q(S_i) := \text{trapezoid in } \mathcal{T}(S_i) \text{ that contains } q.$
Query Time (cont’d)

\( p_i = \) prob. that the search path \( \Pi_q \) of \( q \) in \( D \) contains a node that was created in iteration \( i \).

\[ \Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i \]

\( \Delta_q(S_i) := \) trapezoid in \( T(S_i) \) that contains \( q \).

**Key idea:** Iteration \( i \) contributes a node to \( \Pi_q \) iff
Query Time (cont’d)

\( p_i = \) prob. that the search path \( \Pi_q \) of \( q \) in \( D \) contains a node that was created in iteration \( i \).

\[ \Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i \]

\( \Delta_q(S_i) := \) trapezoid in \( T(S_i) \) that contains \( q \).

**Key idea:** Iteration \( i \) contributes a node to \( \Pi_q \) iff

\[ \Delta_q(S_{i-1}) \neq \Delta_q(S_i). \]
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \)

\[ \Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i \]

\( \Delta_q(S_i) := \text{trapezoid in } T(S_i) \text{ that contains } q. \)

**Key idea:** Iteration \( i \) contributes a node to \( \Pi_q \) iff

\[ \Delta_q(S_i-1) \neq \Delta_q(S_i). \]

In this case \( \Delta_q(S_i) \) must have been created in iteration \( i \).
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } D \text{ contains a node that was created in iteration } i. \)

\[ \Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i \]

\( \Delta_q(S_i) := \text{trapezoid in } T(S_i) \text{ that contains } q. \)

**Key idea:** Iteration \( i \) contributes a node to \( \Pi_q \) iff

\[ \Delta_q(S_{i-1}) \neq \Delta_q(S_i). \]

In this case \( \Delta_q(S_i) \) must have been created in iteration \( i. \)

\[ \Rightarrow \Delta := \Delta_q(S_i) \text{ is adjacent to the new segment } s_i. \]
Query Time (cont’d)

\[ p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \]

\[ \Rightarrow \mathbb{E}[X_i] = \sum_{j=0}^{3} j \cdot \mathbb{P}[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot \mathbb{P}[X_i \geq 1] = 3p_i \]

\[ \Delta_q(S_i) := \text{trapezoid in } T(S_i) \text{ that contains } q. \]

**Key idea:** Iteration \( i \) contributes a node to \( \Pi_q \) iff

\[ \Delta_q(S_{i-1}) \neq \Delta_q(S_i). \]

In this case \( \Delta_q(S_i) \) must have been created in iteration \( i \).

\[ \Rightarrow \Delta := \Delta_q(S_i) \text{ is adjacent to the new segment } s_i. \]

\[ \Rightarrow \text{top}(\Delta) = s_i, \text{bot}(\Delta) = s_i, \text{leftp}(\Delta) \in s_i, \text{ or rightp}(\Delta) \in s_i. \]
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \)

\[
\Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i
\]

\( \Delta_q(S_i) := \text{trapezoid in } T(S_i) \text{ that contains } q. \)

**Key idea:** Iteration \( i \) contributes a node to \( \Pi_q \) iff

\[
\Delta_q(S_{i-1}) \neq \Delta_q(S_i).
\]

In this case \( \Delta_q(S_i) \) must have been created in iteration \( i. \)

\[
\Rightarrow \Delta := \Delta_q(S_i) \text{ is adjacent to the new segment } s_i.
\]

\[
\Rightarrow \text{top}(\Delta) = s_i, \text{ bot}(\Delta) = s_i, \text{ leftp}(\Delta) \in s_i, \text{ or rightp}(\Delta) \in s_i.
\]

**Trick:** \( T(S_i) \) (and thus \( \Delta \)) is uniquely determined by \( S_i. \).
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } D \text{ contains a node that was created in iteration } i. \)

\[ \Rightarrow E[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i \]

\( \Delta_q(S_i) := \text{trapezoid in } \mathcal{T}(S_i) \text{ that contains } q. \)

**Key idea:** Iteration \( i \) contributes a node to \( \Pi_q \) iff

\[ \Delta_q(S_{i-1}) \neq \Delta_q(S_i). \]

In this case \( \Delta_q(S_i) \) must have been created in iteration \( i. \)

\[ \Rightarrow \Delta := \Delta_q(S_i) \text{ is adjacent to the new segment } s_i. \]

\[ \Rightarrow \text{top}(\Delta) = s_i, \text{bot}(\Delta) = s_i, \text{leftp}(\Delta) \in s_i, \text{ or rightp}(\Delta) \in s_i. \]

**Trick:** \( \mathcal{T}(S_i) \) (and thus \( \Delta \)) is uniquely determined by \( S_i \).

Consider \( S_i \subseteq S \text{ fixed.} \)
Query Time (cont’d)

\[ p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \]

\[ \Rightarrow \mathbf{E}[X_i] = \sum_{j=0}^{3} j \cdot P[X_i = j] \leq \sum_{j=0}^{3} 3 \cdot P[X_i \geq 1] = 3p_i \]

\[ \Delta_q(S_i) := \text{trapezoid in } \mathcal{T}(S_i) \text{ that contains } q. \]

**Key idea:** Iteration \( i \) contributes a node to \( \Pi_q \) iff

\[ \Delta_q(S_{i-1}) \neq \Delta_q(S_i). \]

In this case \( \Delta_q(S_i) \) must have been created in iteration \( i. \)

\[ \Rightarrow \Delta := \Delta_q(S_i) \text{ is adjacent to the new segment } s_i. \]

\[ \Rightarrow \text{top}(\Delta) = s_i, \text{bot}(\Delta) = s_i, \text{leftp}(\Delta) \in s_i, \text{or rightp}(\Delta) \in s_i. \]

**Trick:** \( \mathcal{T}(S_i) \) (and thus \( \Delta \)) is uniquely determined by \( S_i. \)

Consider \( S_i \subseteq S \text{ fixed.} \)

\[ \Rightarrow \Delta \text{ does not depend on insertion order.} \]
Query Time (cont’d)

$p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } D \text{ contains a node that was created in iteration } i.$

i.e., prob that $\Delta$ changes when inserting $s_i$.

**Aim:** bound $p_i$. 

Query Time (cont’d)

$p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i$.

i.e., prob that $\Delta$ changes when inserting $s_i$.

**Aim:** bound $p_i$.

**Tool:**
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \)

i.e., prob that \( \Delta \) changes when inserting \( s_i \).

**Aim:** bound \( p_i \).

**Tool:** *Backwards analysis!*
Query Time (cont’d)

\[ p_i = \text{prob. that the search path } \pi_q \text{ of } q \text{ in } D \text{ contains a node that was created in iteration } i. \]
\[ \text{i.e., prob that } \Delta \text{ changes when inserting } s_i. \]

**Aim:** bound \( p_i \).

**Tool:** Backwards analysis!

\[ p_i = \text{prob that } \Delta \text{ changes when } s_i \text{ is removed} \]
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \)

i.e., prob that \( \Delta \) changes when inserting \( s_i \).

**Aim:** bound \( p_i \).

**Tool:** Backwards analysis!

\( p_i = \text{prob that } \Delta \text{ changes when } s_i \text{ is removed} \)

Four cases:
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } D \text{ contains a node that was created in iteration } i. \)

i.e., prob that \( \Delta \) changes when inserting \( s_i \).

**Aim:** bound \( p_i \).

**Tool:** Backwards analysis!

\( p_i = \text{prob that } \Delta \text{ changes when } s_i \text{ is removed} \)

Four cases:

\[
P(\text{top}(\Delta) = s_i) = ?
\]
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \)

i.e., prob that \( \Delta \) changes when inserting \( s_i \).

**Aim:** bound \( p_i \).

**Tool:** *Backwards analysis!*

\( p_i = \text{prob that } \Delta \text{ changes when } s_i \text{ is removed} \)

Four cases:

\[
\Pr(\text{top}(\Delta) = s_i) = \frac{1}{i} \text{ (since exactly one of } i \text{ segments is top}(\Delta)).
\]
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } \mathcal{D} \text{ contains a node that was created in iteration } i. \)

i.e., prob that \( \Delta \) changes when inserting \( s_i \).

**Aim:** bound \( p_i \).

**Tool:** Backwards analysis!

\( p_i = \text{prob that } \Delta \text{ changes when } s_i \text{ is removed} \)

Four cases:

\[
P(\text{top}(\Delta) = s_i) = \frac{1}{i} \text{ (since exactly one of } i \text{ segments is } \text{top}(\Delta)).
\]

\( \Rightarrow p_i \leq \frac{4}{i} \)

\( \Rightarrow \mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i] \leq \sum_{i=1}^{n} 3 \cdot p_i \)

\( = 12 \sum_{i=1}^{n} \frac{1}{i} \leq O(\log n) \)
Query Time (cont’d)

\( p_i = \text{prob. that the search path } \Pi_q \text{ of } q \text{ in } D \text{ contains a node that was created in iteration } i. \)
i.e., \( \text{prob that } \Delta \text{ changes when inserting } s_i. \)

Aim: bound \( p_i. \)

Tool: Backwards analysis!

\( p_i = \text{prob that } \Delta \text{ changes when } s_i \text{ is removed} \)

Four cases:

\[
\Pr(\text{top}(\Delta) = s_i) = \frac{1}{i} \quad \text{(since exactly one of } i \text{ segments is top}(\Delta)).
\]

\[
\Rightarrow p_i \leq \frac{4}{i}
\]

\[
\Rightarrow \mathbb{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{E}[X_i] \leq \sum_{i=1}^{n} 3 \cdot p_i
\]

\[
= 12 \sum_{i=1}^{n} \frac{1}{i} \leq O(\log n)
\]