Computational Geometry

Simplex Range Searching
Lecture #11

[Comp. Geom A&A : Chapter 16]
Range-Counting Query
Range-Counting Query

area affected by the construction of a new airport

Figure from *Computational Geometry: Algorithms and Applications*, De Berg et al., 3rd edition, Springer 2008.
Range-Counting Query

area affected by the construction of a new airport

Observation:
Query range depends on, e.g., dominant wind directions

Range-Counting Query

Observation:
Query range depends on, e.g., dominant wind directions

⇒ non-orthogonal
Non-orthogonal range queries

Query range:
Non-orthogonal range queries

Query range:
Non-orthogonal range queries

Query range:
Non-orthogonal range queries

Query range:
Non-orthogonal range queries

Query range:

Problem: Given a set $P$ of $n$ points, preprocess $P$ such that half-space range-counting queries can be answered quickly.
Non-orthogonal range queries

Query range:  

Problem: Given a set $P$ of $n$ points, preprocess $P$ such that *half-space range-counting queries* can be answered quickly.

Task: Design a data structure for the 1-dim. case:
Non-orthogonal range queries

Query range:

Problem: Given a set $P$ of $n$ points, preprocess $P$ such that half-space range-counting queries can be answered quickly.

Task: Design a data structure for the 1-dim. case:

– Given a number $x$, return $|P \cap [x, \infty)|$. 
Non-orthogonal range queries

Query range:  

Problem: Given a set $P$ of $n$ points, preprocess $P$ such that half-space range-counting queries can be answered quickly.

Task: Design a data structure for the 1-dim. case:

- Given a number $x$, return $|P \cap [x, \infty)|$.
- Consider $P$ static / dynamic!
The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:
The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution: • use balanced binary search trees
The 1-Dimensional Case

**Task:** Design a data structure for the 1-dimensional case!

**Solution:**
- use balanced binary search trees
- augment each node with the number of nodes in its subtree

The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution: • use balanced binary search trees
• augment each node with the number of nodes in its subtree

[see Cormen et al., Introduction to Algorithms, MIT press, 3rd ed., 2009]
The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution: • use balanced binary search trees  
• augment each node with the number of nodes in its subtree [see Cormen et al., Introduction to Algorithms, MIT press, 3rd ed., 2009]
The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:
- use balanced binary search trees
- augment each node with the number of nodes in its subtree  
  [see Cormen et al., Introduction to Algorithms, MIT press, 3rd ed., 2009]
The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution: • use balanced binary search trees
• augment each node with the number of nodes in its subtree

[see Cormen et al., Introduction to Algorithms, MIT press, 3rd ed., 2009]
The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution: • use balanced binary search trees
• augment each node with the number of nodes in its subtree

Lesson: On each level, visit \( \leq 1 \) subtree recursively!

The 1-Dimensional Case

Task: Design a data structure for the 1-dimensional case!

Solution:
- use balanced binary search trees
- augment each node with the number of nodes in its subtree [see Cormen et al., *Introduction to Algorithms*, MIT press, 3rd ed., 2009]

Lesson: On each level, visit $\leq 1$ subtree recursively!
Generalizing to 2 Dimensions

Any ideas?
Generalizing to 2 Dimensions

Any ideas?
Generalizing to 2 Dimensions

Partition the input!
Generalizing to 2 Dimensions

Partition the input! Query...
Generalizing to 2 Dimensions

Partition the input!  Query... in a partition tree
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*
Generalizing to 2 Dimensions

Partition the input! Query... in a partition tree... recursively!
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*... recursively!
Generalizing to 2 Dimensions

Partition the input!  Query... in a *partition tree* ... recursively!

**Definition:**  \( \Psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\} \) is a simplicial partition (of size \( r \)) for \( S \) if
Generalizing to 2 Dimensions

Partition the input! Query... in a partition tree... recursively!

Definition: \( \psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\} \) is a simplicial partition (of size \( r \)) for \( S \) if
- \( S \) is partitioned by \( S_1, \ldots, S_r \) and
- for \( 1 \leq i \leq r \), \( t_i \) is a triangle and \( S_i \subset t_i \).
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*... recursively!

**Definition:** \( \psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\} \) is a *simplicial partition* (of size \( r \)) for \( S \) if
- \( S \) is partitioned by \( S_1, \ldots, S_r \) and
- for \( 1 \leq i \leq r \), \( t_i \) is a triangle and \( S_i \subset t_i \).
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree* ... recursively!

**Definition:** $\psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$ is a *simplicial partition* (of size $r$) for $S$ if
- $S$ is partitioned by $S_1, \ldots, S_r$ and
- for $1 \leq i \leq r$, $t_i$ is a triangle and $S_i \subset t_i$.

$\psi(S)$ is *fine* if $|S_i| \leq 2\frac{|S|}{r}$ for every $1 \leq i \leq r$. 
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*... recursively!

**Definition:** The *crossing number* of $\ell$ (w.r.t. $\Psi(S)$) is the number of triangles $t_1, \ldots, t_r$ crossed by $\ell$. 
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*... recursively!

**Definition:** The *crossing number* of \( \ell \) (w.r.t. \( \Psi(S) \)) is the number of triangles \( t_1, \ldots, t_r \) crossed by \( \ell \).

The *crossing number* of \( \Psi(S) \) is the maximum crossing number over all possible lines.
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree* ... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree* ... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree* ... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree* ... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Generalizing to 2 Dimensions

Partition the input! Query... in a *partition tree*... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Generalizing to 2 Dimensions

Partition the input! Query... in a partition tree... recursively!

Theorem. For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Generalizing to 2 Dimensions

Partition the input! Query... in a partition tree... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Generalizing to 2 Dimensions

Partition the input! Query... in a partition tree... recursively!

**Theorem.** For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Generalizing to 2 Dimensions

Partition the input! Query... in a partition tree... recursively!

Theorem. For any set $S$ of $n$ pts and any $1 \leq r \leq n$, a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$ exists.

For any $\varepsilon > 0$, such a partition can be built in $O(n^{1+\varepsilon})$ time.
Example for a Query

point set $S$
Example for a Query

point set $S$

partition by triangles
Example for a Query

point set $S$

$h \ldots$ query range

partition by triangles
Example for a Query

point set $S$

$h \ldots$ query range

partition tree for $S$

partition by triangles
Example for a Query

point set $S$

$h \ldots$ query range

partition by triangles

partition tree for $S$

- $v_1$
- $v_2$
- $v_3$
- $v_4$
- $v_5$
- $v_6$
- $v_7$

- selected node
- visited node
Example for a Query

point set $S$

$h\ldots$ query range

partition by triangles

partition tree for $S$

bullet = selected node

gray = visited node
Example for a Query

point set $S$

$h$... query range

partition tree for $S$

partition by triangles

$V_1$, $V_2$, $V_3$, $V_4$, $V_5$, $V_6$, $V_7$

recursively visited subtrees

$\bullet$ = selected node
$\circ$ = visited node
Query Algorithm

$\textbf{SelectInHalfplane}(\text{half-plane } h, \text{ partit. tree } \mathcal{T} \text{ for pt set } S)$

$N \leftarrow \emptyset \quad \{ \text{set of selected nodes} \}$
Query Algorithm

**SelectInHalfplane** (half-plane h, partit. tree T for pt set S)

\[
N \leftarrow \emptyset \quad \{ \text{set of selected nodes} \}
\]

if \( T = \{ \mu \} \) then

else

return \( N \) \quad \{ \text{with } S \cap h = \bigcup_{\nu \in N} S(\nu) \}
**Query Algorithm**

**SelectInHalfplane** (half-plane $h$, partit. tree $\mathcal{T}$ for pt set $S$)

$N \leftarrow \emptyset$ \hspace{1cm} \{ set of selected nodes \}

\begin{align*}
\text{if } \mathcal{T} = \{\mu\} \text{ then} & \quad \text{if point stored at } \mu \text{ lies in } h \text{ then} \\
& \quad \quad \quad \quad N \leftarrow \{\mu\} \\
\text{else} & \quad \text{return } N \quad \{ \text{with } S \cap h = \bigcup_{\nu \in N} S(\nu) \}
\end{align*}
**Query Algorithm**

\[\text{SELECTINHALFPLANE}(\text{half-plane } h, \text{ partit. tree } T \text{ for pt set } S)\]

\[N \leftarrow \emptyset \quad \{ \text{set of selected nodes} \} \]

\[
\begin{align*}
&\text{if } T = \{\mu\} \text{ then} \\
&\quad \text{if point stored at } \mu \text{ lies in } h \text{ then} \\
&\qquad N \leftarrow \{\mu\}
\end{align*}
\]

\[
\begin{align*}
&\text{else} \\
&\quad \text{foreach child } \nu \text{ of the root of } T \text{ do}
\end{align*}
\]

\[
\begin{align*}
&\quad \text{if } t(\nu) \subset h \text{ then} \\
&\qquad N \leftarrow N \cup \{\nu\}
\end{align*}
\]

\[
\begin{align*}
&\quad \text{else} \\
&\qquad \text{if } t(\nu) \cap h \neq \emptyset \text{ then} \\
&\qquad \quad N \leftarrow N \cup \text{SELECTINHALFPLANE}(h, T_\nu)
\end{align*}
\]

\[
\begin{align*}
&\text{return } N \quad \{ \text{with } S \cap h = \bigcup_{\nu \in N} S(\nu) \}
\end{align*}
\]
Query Algorithm

\[ \text{SelectInHalfplane}(\text{half-plane } h, \text{ partit. tree } T \text{ for pt set } S) \]

\[ N \leftarrow \emptyset \quad \{ \text{set of selected nodes} \} \]

\[ \text{if } T = \{ \mu \} \text{ then} \]

\[ \quad \text{if point stored at } \mu \text{ lies in } h \text{ then} \]

\[ \quad \quad N \leftarrow \{ \mu \} \]

\[ \text{else} \]

\[ \quad \text{foreach child } \nu \text{ of the root of } T \text{ do} \]

\[ \quad \quad \text{if } t(\nu) \subset h \text{ then} \]

\[ \quad \quad \quad \text{else} \]

\[ \text{return } N \quad \{ \text{with } S \cap h = \bigcup_{\nu \in N} S(\nu) \} \]
**Query Algorithm**

\[ \text{SELECTINHALFPLANE}(\text{half-plane } h, \text{ partit. tree } T \text{ for pt set } S) \]

\[ N \leftarrow \emptyset \quad \{ \text{set of selected nodes} \} \]

\[ \text{if } T = \{ \mu \} \text{ then} \]
\[ \quad \text{if point stored at } \mu \text{ lies in } h \text{ then} \]
\[ \quad \quad N \leftarrow \{ \mu \} \]

**else**

\[ \quad \text{foreach child } \nu \text{ of the root of } T \text{ do} \]
\[ \quad \quad \text{if } t(\nu) \subseteq h \text{ then} \]
\[ \quad \quad \quad N \leftarrow N \cup \{ \nu \} \]
\[ \quad \quad \text{else} \]
\[ \quad \quad \quad \text{else} \]

**return** \[ N \quad \{ \text{with } S \cap h = \bigcup_{\nu \in N} S(\nu) \} \]
Query Algorithm

\texttt{SELECTINHALFPLANE}(\text{half-plane } h, \text{ partit. tree } \mathcal{T} \text{ for pt set } S)
\begin{align*}
N & \leftarrow \emptyset \quad \{ \text{ set of selected nodes } \} \\
\text{if } \mathcal{T} = \{ \mu \} \text{ then} & \\
& \quad \begin{aligned}
& \text{if point stored at } \mu \text{ lies in } h \text{ then} \\
& \quad \quad N \leftarrow \{ \mu \}
\end{aligned} \\
\text{else} & \\
& \quad \begin{aligned}
& \text{foreach child } \nu \text{ of the root of } \mathcal{T} \text{ do} \\
& \quad \quad \begin{aligned}
& \text{if } t(\nu) \subset h \text{ then} \\
& \quad \quad \quad N \leftarrow N \cup \{ \nu \}
\end{aligned} \\
& \text{else} \\
& \quad \quad \quad \begin{aligned}
& \text{if } t(\nu) \cap h \neq \emptyset \text{ then}
\end{aligned}
\end{aligned}
\end{aligned}
\text{return } N \quad \{ \text{ with } S \cap h = \bigcup_{\nu \in N} S(\nu) \}
Query Algorithm

\textbf{SELECTINHALFPLANE}(half-plane \(h\), partit. tree \(T\) for pt set \(S\))

\(N \leftarrow \emptyset\) \{ set of selected nodes \}

\begin{align*}
\text{if } T = \{\mu\} \text{ then} & \\
& \text{if point stored at } \mu \text{ lies in } h \text{ then} \\
& \quad N \leftarrow \{\mu\} \\
\text{else} & \\
& \quad \text{foreach child } \nu \text{ of the root of } T \text{ do} \\
& \quad \quad \text{if } t(\nu) \subset h \text{ then} \\
& \quad \quad \quad N \leftarrow N \cup \{\nu\} \\
& \quad \quad \text{else} \\
& \quad \quad \quad \text{if } t(\nu) \cap h \neq \emptyset \text{ then} \\
& \quad \quad \quad \quad N \leftarrow N \cup \text{SELECTINHALFPLANE}(h, T_\nu) \\
\end{align*}

return \(N\) \{ with \(S \cap h = \bigcup_{\nu \in N} S(\nu)\) \}
Query Algorithm

\begin{align*}
\text{\textbf{SelectInHalfplane}}(\text{half-plane } h, \text{ partit. tree } \mathcal{T} \text{ for pt set } S) \\
N &\leftarrow \emptyset \quad \{ \text{set of selected nodes} \} \\
\text{if } \mathcal{T} = \{\mu\} \text{ then} \\
&\quad \text{if point stored at } \mu \text{ lies in } h \text{ then} \\
&\quad \quad N \leftarrow \{\mu\} \\
\text{else} \\
&\quad \text{foreach child } \nu \text{ of the root of } \mathcal{T} \text{ do} \\
&\quad \quad \text{if } t(\nu) \subset h \text{ then} \\
&\quad \quad \quad N \leftarrow N \cup \{\nu\} \\
&\quad \quad \text{else} \\
&\quad \quad \quad \text{if } t(\nu) \cap h \neq \emptyset \text{ then} \\
&\quad \quad \quad \quad N \leftarrow N \cup \text{SelectInHalfplane}(h, \mathcal{T}_\nu) \\
\text{return } N \quad \{ \text{with } S \cap h = \bigcup_{\nu \in N} S(\nu) \}\end{align*}

Task:
Turn this into a range counting query algorithm!
Analysis of the Partition Tree

Let $S$ be a set of $n$ points in the plane. \textit{Recall:}

\textbf{Theorem.} For any $r$ with $1 \leq r \leq n$, $S$ has a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$. For any $\varepsilon > 0$, such a partition can be computed in $O(n^{1+\varepsilon})$ time.
Analysis of the Partition Tree

Let $S$ be a set of $n$ points in the plane.

**Theorem.** For any $r$ with $1 \leq r \leq n$, $S$ has a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$. For any $\varepsilon > 0$, such a partition can be computed in $O(n^{1+\varepsilon})$ time.

**Lemma.** A partition tree for $S$ can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.
Analysis of the Partition Tree

Let $S$ be a set of $n$ points in the plane.  

**Theorem.** For any $r$ with $1 \leq r \leq n$, $S$ has a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$. For any $\varepsilon > 0$, such a partition can be computed in $O(n^{1+\varepsilon})$ time.

**Lemma.** A partition tree for $S$ can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.

**Lemma.** For any $\varepsilon > 0$, there is a partition tree $T$ for $S$ s.t.:
Analysis of the Partition Tree

Let $S$ be a set of $n$ points in the plane. \( \text{Recall:} \)

**Theorem.** For any $r$ with $1 \leq r \leq n$, $S$ has a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$. For any $\varepsilon > 0$, such a partition can be computed in $O(n^{1+\varepsilon})$ time.

**Lemma.** A partition tree for $S$ can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.

**Lemma.** For any $\varepsilon > 0$, there is a partition tree $\mathcal{T}$ for $S$ s.t.:

- for a query half-plane $h$, $\text{SelectInHalfplane}$ selects in $O(n^{1/2+\varepsilon})$ time
- a set $N$ of $O(n^{1/2+\varepsilon})$ nodes of $\mathcal{T}$
Analysis of the Partition Tree

Let $S$ be a set of $n$ points in the plane.  

**Theorem.** For any $r$ with $1 \leq r \leq n$, $S$ has a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$. For any $\varepsilon > 0$, such a partition can be computed in $O(n^{1+\varepsilon})$ time.

**Lemma.** A partition tree for $S$ can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.

**Lemma.** For any $\varepsilon > 0$, there is a partition tree $\mathcal{T}$ for $S$ s.t.:

- for a query half-plane $h$,
- $\text{SelectInHalfplane}$ selects in $O(n^{1/2+\varepsilon})$ time
- a set $N$ of $O(n^{1/2+\varepsilon})$ nodes of $\mathcal{T}$
- with the property that $h \cap S = \bigcup_{\nu \in N} S(\nu)$. 

*Recall:*
Analysis of the Partition Tree

Let $S$ be a set of $n$ points in the plane.

Recall:

**Theorem.** For any $r$ with $1 \leq r \leq n$, $S$ has a fine simplicial partition of size $r$ and crossing number $O(\sqrt{r})$. For any $\varepsilon > 0$, such a partition can be computed in $O(n^{1+\varepsilon})$ time.

**Lemma.** A partition tree for $S$ can be constructed in $O(n^{1+\varepsilon})$ time. The tree uses $O(n)$ storage.

**Lemma.** For any $\varepsilon > 0$, there is a partition tree $T$ for $S$ s.t.:

- for a query half-plane $h$, $\text{SelectInHalfplane}$ selects in $O(n^{1/2+\varepsilon})$ time
- a set $N$ of $O(n^{1/2+\varepsilon})$ nodes of $T$
- with the property that $h \cap S = \bigcup_{\nu \in N} S(\nu)$.

**Corollary.** Half-plane range counting queries can be answered in $O(n^{1/2+\varepsilon})$ time using $O(n)$ space & $O(n^{1+\varepsilon})$ prep.
Back to *Triangular* Range Queries

Any ideas?
Back to *Triangular* Range Queries

**Any ideas?** Just use `SELECTINHALFPLANE`!
Back to *Triangular Range Queries*

**Any ideas?** Just use `SELECTINHALFPLANE`!

**Theorem:** Given a set $S$ of $n$ pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.
Back to *Triangular Range Queries*

**Any ideas?**  Just use `SELECT_IN_HALFPLANE`!

**Theorem:**  Given a set \( S \) of \( n \) pts in the plane, for any \( \varepsilon > 0 \), a triangular range-counting query can be answered in \( O(n^{1/2+\varepsilon}) \) time using a partition tree. The tree can be built in \( O(n^{1+\varepsilon}) \) time and uses \( O(n) \) space.
Back to *Triangular* Range Queries

**Any ideas?** Just use `SELECTINHALFPLANE`!

**Theorem:** Given a set $S$ of $n$ pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.

The tree can be built in $O(n^{1+\varepsilon})$ time and uses $O(n)$ space.

The points inside the query range can be reported in $O(k)$ additional time, where $k$ is the number of reported pts.
Back to *Triangular Range Queries*

**Any ideas?** Just use `SELECTINHALFPLANE`!

**Theorem:** Given a set $S$ of $n$ pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree.

The tree can be built in $O(n^{1+\varepsilon})$ time and uses $O(n)$ space.

The points inside the query range can be reported in $O(k)$ additional time, where $k$ is the number of reported pts.

**Can we do better?**
Back to *Triangular Range Queries*

Any ideas?  Just use `SelectInHalfplane`!

**Theorem:** Given a set $S$ of $n$ pts in the plane, for any $\varepsilon > 0$, a triangular range-counting query can be answered in $O(n^{1/2+\varepsilon})$ time using a partition tree. The tree can be built in $O(n^{1+\varepsilon})$ time and uses $O(n)$ space.

The points inside the query range can be reported in $O(k)$ additional time, where $k$ is the number of reported pts.

Can we do better?

Use cutting trees! (Chapter 16.3)
Any ideas? Just use \texttt{SELECTINHALFPLANE}!

Theorem:Given a set \( S \) of \( n \) pts in the plane, for any \( \varepsilon > 0 \), a triangular range-counting query can be answered in \( O(n^{1/2+\varepsilon}) \) time using a partition tree. The tree can be built in \( O(n^{1+\varepsilon}) \) time and uses \( O(n) \) space. The points inside the query range can be reported in \( O(k) \) additional time, where \( k \) is the number of reported pts.

Can we do better? Use cutting trees! (Chapter 16.3) Query time \( O(\log^3 n) \), prep. & storage \( O(n^{2+\varepsilon}) \).
Multi-Level Partition Trees

Idea: Store with each internal node not just a number,
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, $|S(v)|$.
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, but another data structure!
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 

\[ |S(v)| \]
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$.

Hint:
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$.

Hint:
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$.

Hint:

$$p_{\text{left}}(s)$$

$$s$$

$$p_{\text{right}}(s)$$
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$.

**Hint:**

\[ \text{left}(s) \]

\[ \text{right}(s) \]
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$.

**Hint:**
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$.

Hint:
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$.

**Hint:**

$\left| S(v) \right|$
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 

$s'(p_{left}(s'))$

$p_{right}(s')$

$S(v)$
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 

$p_{\text{left}}(s')$ $p_{\text{right}}(s')$
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 

![Diagram showing line segments and partitioning lines $p_{left}(s')$ and $p_{right}(s')$.]
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line \( \ell \).
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 

\[ p_{\text{left}}(s') \]

\[ p_{\text{right}}(s') \]
Multi-Level Partition Trees

Idea: Store with each internal node not just a number, but another data structure!

Task: Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 

\[ |S(v)| \]
Multi-Level Partition Trees

**Idea:** Store with each internal node not just a number, but another data structure!

**Task:** Design a fast data structure for line segments that counts all segments intersecting a query line $\ell$. 

$|S(v)|$
Query Algorithm

SelectIntSegments(line \( \ell \), two-level partition tree \( \mathcal{T} \) for \( S \))

\[
N \leftarrow \emptyset \\
\text{if } \mathcal{T} = \{\mu\} \text{ then} \\
\quad \text{if segment stored in } \mu \text{ intersects } \ell \text{ then } N \leftarrow \{\mu\} \\
\text{else} \\
\quad \text{foreach child } \nu \text{ of } \mathcal{T}'s \text{ root do} \\
\quad \quad \text{if } t(\nu) \subset \ell^+ \text{ then} \\
\quad \quad \quad N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, \mathcal{T}_{\nu}^{\text{assoc}}) \\
\quad \quad \text{else} \\
\quad \quad \quad \text{if } t(\nu) \cap \ell \neq \emptyset \text{ then} \\
\quad \quad \quad \quad N \leftarrow N \cup \text{SelectIntSegments}(\ell, \mathcal{T}_{\nu}) \\
\text{return } N
\]

- first-level tree stores \( P_{\text{right}}(S) \)
- second-level trees store subsets of \( P_{\text{left}}(S) \)
Query Algorithm

SelectIntSegments(line ℓ, two-level partition tree T for S)

N ← ∅
if T = {µ} then
  if segment stored in µ intersects ℓ then
    N ← {µ}
else
  foreach child ν of T’s root do
    if t(ν) ⊂ ℓ⁺ then
      N ← N ∪ SelectInHalfplane(ℓ⁻, T_assoc
    else
      if t(ν) ∩ ℓ ̸= ∅ then
        N ← N ∪ SelectIntSegments(ℓ, T_ν)

return N

For S' ⊆ S, let
\[ P_{\text{right}}(S') = \{ p_{\text{right}}(s) \mid s \in S' \} \]
left
\[ P_{\text{left}}(S) = \{ p_{\text{left}}(s) \mid s \in S \} \]

stores \( P_{\text{left}}(S_{\text{seg}}(ν)) \), where
\[ S_{\text{seg}}(ν) = \{ s \mid p_{\text{right}}(s) \in S(ν) \} \]

For \( S' \subseteq S \), let
\[ P_{\text{right}}(S') = \{ p_{\text{right}}(s) \mid s \in S' \} \]
left
\[ P_{\text{left}}(S) = \{ p_{\text{left}}(s) \mid s \in S \} \]

stores \( P_{\text{left}}(S_{\text{seg}}(ν)) \), where
\[ S_{\text{seg}}(ν) = \{ s \mid p_{\text{right}}(s) \in S(ν) \} \]
Query Algorithm

SelectIntSegments(line $\ell$, two-level partition tree $T$ for $S$)

$$N \leftarrow \emptyset$$

if $T = \{\mu\}$ then
  if segment stored in $\mu$ intersects $\ell$ then
    $N \leftarrow \{\mu\}$
  else
    foreach child $\nu$ of $T$’s root do
      if $t(\nu) \subset \ell^+$ then
        $N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, T^{\text{assoc}}_\nu)$
      else
        if $t(\nu) \cap \ell \neq \emptyset$ then
          $N \leftarrow N \cup \text{SelectIntSegments}(\ell, T_\nu)$

return $N$

For $S' \subseteq S$, let $P_{\leftarrow}(S') = \{p_{\leftarrow}(s) \mid s \in S'\}$

stores $P_{\leftarrow}(S_{\text{seg}}(\nu))$, where $S_{\text{seg}}(\nu) = \{s \mid p_{\right}(s) \in S(\nu)\}$
Query Algorithm

SelectIntSegments(line $\ell$, two-level partition tree $\mathcal{T}$ for $S$)

$$N \leftarrow \emptyset$$

if $\mathcal{T} = \{ \mu \}$ then

if segment stored in $\mu$ intersects $\ell$ then

$N \leftarrow \{ \mu \}$

else

foreach child $\nu$ of $\mathcal{T}$'s root do

if $t(\nu) \subset \ell^+$ then

$N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, \mathcal{T}\nu^{\text{assoc}})$

else

if $t(\nu) \cap \ell \neq \emptyset$ then

$N \leftarrow N \cup \text{SelectIntSegments}(\ell, \mathcal{T}\nu)$

return $N$

For $S' \subseteq S$, let

$P_{\text{right}}(S') = \{ p_{\text{right}}(s) \mid s \in S' \}$

$P_{\text{left}}$ stores $P_{\text{right}}(S)$

$P_{\text{left}}$ stores subsets of $P_{\text{left}}(S)$

stores $P_{\text{left}}(S_{\text{seg}}(\nu))$, where

$S_{\text{seg}}(\nu) = \{ s \mid p_{\text{right}}(s) \in S(\nu) \}$

$\bigcup_{\nu \in N} S(\nu) = \{ s \in S \mid p_{\text{right}}(s) \text{ above } \ell \text{ and } p_{\text{left}}(s) \text{ below } \ell \}$. 
Query Algorithm

SelectIntSegments(line $\ell$, two-level partition tree $T$ for $S$)

$N \leftarrow \emptyset$

if $T = \{\mu\}$ then

if segment stored in $\mu$ intersects $\ell$ then $N \leftarrow \{\mu\}$

else

foreach child $\nu$ of $T$'s root do

if $t(\nu) \subset \ell^+$ then

$N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, T\nu\text{assoc})$

else

if $t(\nu) \cap \ell \neq \emptyset$ then

$N \leftarrow N \cup \text{SelectIntSegments}(\ell, T\nu)$

return $N$

\[ \bigcup_{\nu \in N} S(\nu) = \{s \in S \mid p_{\text{right}}(s) \text{ above } \ell \text{ and } p_{\text{left}}(s) \text{ below } \ell\}. \]
Query Algorithm

SelectIntSegments(line $\ell$, two-level partition tree $T$ for $S$)

\[ N \leftarrow \emptyset \]

if $T = \{ \mu \}$ then

\[ \text{if segment stored in } \mu \text{ intersects } \ell \text{ then } N \leftarrow \{ \mu \} \]

else

\[
\text{foreach child } \nu \text{ of } T \text{'s root do}
\]

\[ \text{if } t(\nu) \subset \ell^+ \text{ then} \]

\[ N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, T_{\nu}^{\text{assoc}}) \]

else

\[ \text{if } t(\nu) \cap \ell \neq \emptyset \text{ then} \]

\[ N \leftarrow N \cup \text{SelectIntSegments}(\ell, T_{\nu}) \]

return $N$

\[ \text{For } S' \subseteq S, \text{ let} \]
\[ P_{\text{right}}(S') = \{ p_{\text{right}}(s) \mid s \in S' \}\]
\[ P_{\text{left}}(S') = \{ p_{\text{left}}(s) \mid s \in S' \} \]

stores $P_{\text{left}}(S_{\text{seg}}(\nu))$, where
\[ S_{\text{seg}}(\nu) = \{ s \mid p_{\text{right}}(s) \in S(\nu) \} \]

stores $P_{\text{right}}(S_{\text{seg}}(\nu))$, where
\[ S_{\text{seg}}(\nu) = \{ s \mid p_{\text{right}}(s) \in S(\nu) \} \]

\[ \bigcup_{\nu \in N} S(\nu) = \{ s \in S \mid p_{\text{right}}(s) \text{ above } \ell \text{ and } p_{\text{left}}(s) \text{ below } \ell \} \]
Query Algorithm

SelectIntSegments(line \( \ell \), two-level partition tree \( \mathcal{T} \) for \( S \))

\[
N \leftarrow \emptyset \\
\text{if } \mathcal{T} = \{ \mu \} \text{ then} \\
\quad \text{if segment stored in } \mu \text{ intersects } \ell \text{ then } N \leftarrow \{ \mu \} \\
\text{else} \\
\quad \text{foreach child } \nu \text{ of } \mathcal{T}'s \text{ root do} \\
\quad \quad \text{if } t(\nu) \subset \ell^+ \text{ then} \\
\quad \quad \quad \ N \leftarrow N \cup \text{SelectInHalfplane}(\ell^-, \mathcal{T}_\nu^{\text{assoc}}) \\
\quad \quad \text{else} \\
\quad \quad \quad \text{if } t(\nu) \cap \ell \neq \emptyset \text{ then} \\
\quad \quad \quad \quad \ N \leftarrow N \cup \text{SelectIntSegments}(\ell, \mathcal{T}_\nu) \\
\text{return } N
\]

\[
\bigcup_{\nu \in N} S(\nu) = \{ s \in S \mid p_{\text{right}}(s) \text{ above } \ell \text{ and } p_{\text{left}}(s) \text{ below } \ell \}\]
Results

Lemma: A 2-level partition tree for line-intersection queries among a set of $n$ segments uses $O(n \log n)$ storage.
Results

**Lemma:** A 2-level partition tree for line-intersection queries among a set of $n$ segments uses $O(n \log n)$ storage.

**Lemma:** Let $S$ be a set of $n$ segments in the plane. For any $\varepsilon > 0$, there is a 2-level partition tree $T$ for $S$ s.t.
Results

Lemma: A 2-level partition tree for line-intersection queries among a set of \( n \) segments uses \( O(n \log n) \) storage.

Lemma: Let \( S \) be a set of \( n \) segments in the plane. For any \( \varepsilon > 0 \), there is a 2-level partition tree \( T \) for \( S \) s.t.

- given a query line \( \ell \), we can select \( O(n^{1/2+\varepsilon}) \)
  nodes from \( T \) whose canonical subsets represent the segments intersected by \( \ell \).
Results

Lemma: A 2-level partition tree for line-intersection queries among a set of \( n \) segments uses \( O(n \log n) \) storage.

Lemma: Let \( S \) be a set of \( n \) segments in the plane. For any \( \varepsilon > 0 \), there is a 2-level partition tree \( T \) for \( S \) s.t.

\[ O(n^{1/2+\varepsilon}) \]

– given a query line \( \ell \), we can select \( O(n^{1/2+\varepsilon}) \) nodes from \( T \) whose canonical subsets represent the segments intersected by \( \ell \).

– The selection takes \( O(n^{1/2+\varepsilon}) \) time.
Results

**Lemma:** A 2-level partition tree for line-intersection queries among a set of \( n \) segments uses \( O(n \log n) \) storage.

**Lemma:** Let \( S \) be a set of \( n \) segments in the plane. For any \( \varepsilon > 0 \), there is a 2-level partition tree \( T \) for \( S \) s.t.

- given a query line \( \ell \), we can select \( O(n^{1/2+\varepsilon}) \) nodes from \( T \) whose canonical subsets represent the segments intersected by \( \ell \).
- The selection takes \( O(n^{1/2+\varepsilon}) \) time.

**Corollary:** Let \( S \) be a set of \( n \) segments in the plane. After \( O(...) \)-time preprocessing, we can count the number of segments in \( S \) intersected by a query line in \( O(n^{1/2+\varepsilon}) \) time.