Outline:

I. Windowing query

II. Interval tree

III. Construction and query

IV. Query with a line segment
I. Windowing
1. Windowing
I. Windowing
I. Windowing
Windowing Query

Determine the part of the map that lie in the window and display them.
Windowing Query

Determine the part of the map that lie in the window and display them.

Applications

- Geographic maps
- Computer graphics (e.g., flight simulation – part of landscape within sight)
- Design of circuit boards – zooming in onto a certain portion
Windowing Query

Determine the part of the map that lie in the window and display them.

Applications

- Geographic maps
- Computer graphics (e.g., flight simulation – part of landscape within sight)
- Design of circuit boards – zooming in onto a certain portion

Brute-force approach: Check every single feature to see if it is inside the window.
Windowing Query

Determine the part of the map that lie in the window and display them.

Applications

- Geographic maps
- Computer graphics (e.g., flight simulation – part of landscape within sight)
- Design of circuit boards – zooming in onto a certain portion

Brute-force approach: Check every single feature to see if it is inside the window.

Too slow because the amount of data is huge!
Windowing Query

Determine the part of the map that lie in the window and display them.

Applications

- Geographic maps
- Computer graphics (e.g., flight simulation – part of landscape within sight)
- Design of circuit boards – zooming in onto a certain portion

Brute-force approach: Check every single feature to see if it is inside the window.

Too slow because the amount of data is huge!

Use some data structure to store the map and allow quick retrieval of its portion inside a window.
<table>
<thead>
<tr>
<th></th>
<th>Windowing Query</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>segments, polygons, curves</td>
<td>points</td>
</tr>
<tr>
<td>search space</td>
<td>2D or 3D</td>
<td>$k$-D</td>
</tr>
</tbody>
</table>
Formulating the Problem

\[ S = \{s_1, s_2, \ldots, s_n\}: \text{set of } n \text{ axis-parallel segments} \]
Formulating the Problem

\[ S = \{s_1, s_2, \ldots, s_n\} : \text{set of } n \text{ axis-parallel segments} \]
Formulating the Problem

\[ S = \{s_1, s_2, ..., s_n\}: \text{set of } n \text{ axis-parallel segments} \]

Query window: \( W = [x, x'] \times [y, y'] \)
with boundary \( \partial W \)

Possible cases of a segment \( s \):

- \( s \) inside \( W \);
- \( s \) intersects \( W \) once;
- \( s \) intersects \( W \) twice;
- \( s \) overlaps \( \partial W \).
Segments with $\geq 1$ Endpoints in $\mathcal{W}$

Range query over $2n$ endpoints

$O(n \log n)$ storage

$O(n \log^2 n + k)$ query time
Segments with $\geq 1$ Endpoints in $\mathcal{W}$

Range query over $2n$ endpoints

$O(n \log n)$ storage

$O(n \log^2 n + k)$ query time

fractional cascading

$O(n \log n)$
Segments with $\geq 1$ Endpoints in $\mathcal{W}$

Range query over $2n$ endpoints

$O(n \log n)$ storage

$O(n \log^2 n + k)$ query time

\[\downarrow\] fractional cascading

\[O(n \log n)\]

\[\blacklozenge\] Not to report a segment twice.
Segments with No Endpoint Inside $W$

Segments that pass through $W$ may

- crosses $W$ twice;
- contains one boundary edge.
Segments with No Endpoint Inside $\mathcal{W}$

Segments that pass through $\mathcal{W}$ may

- crosses $\mathcal{W}$ twice;
- contains one boundary edge.

♦ Need only consider horizontal segments.
Segments with No Endpoint Inside $\mathcal{W}$

Segments that pass through $\mathcal{W}$ may

- crosses $\mathcal{W}$ twice;
- contains one boundary edge.

- Need only consider horizontal segments.

The case of vertical segments is similar.
Segments with No Endpoint Inside $W$

Segments that pass through $W$ may

- crosses $W$ twice;
- contains one boundary edge.

* Need only consider horizontal segments.

The case of vertical segments is similar.

* Need only know how to efficiently check if such a segment intersects the left boundary edge.
Segments with No Endpoint Inside $W$

Segments that pass through $W$ may

- crosses $W$ twice;
- contains one boundary edge.

- Need only consider horizontal segments.

The case of vertical segments is similar.

- Need only know how to efficiently check if such a segment intersects the left boundary edge.

It’s similar to check if it intersects the right boundary edge.
II. Query Segment as a Full Line

We first consider the *simpler* version of query when the left boundary edge of $W$ is a *full line*.

\[
\begin{align*}
l : & \quad x = q_x \\
\end{align*}
\]
II. Query Segment as a Full Line

We first consider the *simpler* version of query when the left boundary edge of $\mathcal{W}$ is a *full line*.

$$l: \quad x = q_x$$
II. Query Segment as a Full Line

We first consider the *simpler* version of query when the left boundary edge of $W$ is a *full line*.

**l:** $x = q_x$

Segment $s$: $[x, x'] \times y$ intersects $l$.

$x \leq q_x \leq x'$
II. Query Segment as a Full Line

We first consider the *simpler* version of query when the left boundary edge of \( W \) is a *full line*.

\[ l: x = q_x \]

Segment \( s: [x, x'] \times y \) intersects \( l \).

\[ x \leq q_x \leq x' \]

1D problem

Given a set of intervals \( I = \{ [x_1, y_1], [x_2, y_2], \ldots, [x_n, y_n] \} \), report those that contain the query point \( q_x \).
Partitioning the Interval Set

\[ x_{\text{mid}}: \text{median of } 2n \text{ interval endpoints.} \]
Partitioning the Interval Set

\( x_{mid} \): median of \( 2n \) interval endpoints.
Partitioning the Interval Set

\( x_{\text{mid}} \): median of \( 2n \) interval endpoints.

\[ I_{\text{mid}} = \{ [x_j, x'_j] \in I \mid x_j \leq x_{\text{mid}} \leq x'_j \} \]
Partitioning the Interval Set

\( x_{\text{mid}} \): median of 2n interval endpoints.

\[
I_{\text{mid}} = \{ [x_j, x'_j] \in I \mid x_j \leq x_{\text{mid}} \leq x'_j \}
\]
Partitioning the Interval Set

\( x_{\text{mid}} \): median of \( 2n \) interval endpoints.

\[
I_{\text{mid}} = \{ [x_j, x'_j] \in I \mid x_j \leq x_{\text{mid}} \leq x'_j \}
\]

\[
I_{\text{left}} = \{ [x_j, x'_j] \in I \mid x'_j < x_{\text{mid}} \}
\]
Partitioning the Interval Set

**Partitioning the Interval Set**

\[ x_{\text{mid}}: \text{median of } 2n \text{ interval endpoints.} \]

\[ I_{\text{mid}} = \left\{ [x_j, x'_j] \in I \mid x_j \leq x_{\text{mid}} \leq x'_j \right\} \]

\[ I_{\text{left}} = \left\{ [x_j, x'_j] \in I \mid x'_j < x_{\text{mid}} \right\} \]
Partitioning the Interval Set

\( x_{\text{mid}} \): median of \( 2n \) interval endpoints.

\[ I_{\text{mid}} = \left\{ [x_j, x'_j] \in I \mid x_j \leq x_{\text{mid}} \leq x'_j \right\} \]

\[ I_{\text{left}} = \left\{ [x_j, x'_j] \in I \mid x'_j < x_{\text{mid}} \right\} \]

\[ I_{\text{right}} = \left\{ [x_j, x'_j] \in I \mid x_j > x_{\text{mid}} \right\} \]
Partitioning the Interval Set

$x_{mid}$: median of $2n$ interval endpoints.

$$I_{mid} = \left\{ [x_j, x'_j] \in I \mid x_j \leq x_{mid} \leq x'_j \right\}$$

$$I_{left} = \left\{ [x_j, x'_j] \in I \mid x'_j < x_{mid} \right\}$$

$$I_{right} = \left\{ [x_j, x'_j] \in I \mid x_j > x_{mid} \right\}$$
Interval Tree

- $I_{mid}$
- $I_{left}$
- $I_{right}$
- $x_{mid}$
Interval Tree

$I_{mid}$

$I_{left}$

$x_{mid}$

$I_{right}$

$v$
Interval Tree

$I_{mid}$

$I_{left}$

$I_{right}$

$x_{mid}$

$v$

$I_{mid}$
Interval Tree

$I_{mid}$

$I_{left}$

$I_{right}$

$x_{mid}$

$I_{mid}$

$I_{left}$

$I_{right}$

$lc(v)$

$v$
Interval Tree

\[ I_{\text{mid}} \]

\[ I_{\text{left}} \]

\[ I_{\text{right}} \]

\[ x_{\text{mid}} \]
Interval Tree

Subtrees $lc(v)$ and $rc(v)$ are recursively defined.
Interval Tree

Subtrees $lc(v)$ and $rc(v)$ are recursively defined.

$O(\log n)$ depth
Storage of $I_{mid}$

$I_{mid}$ is stored in two lists at $v$:
Storage of $I_{mid}$

$I_{mid}$ is stored in two lists at $v$:

- $\mathcal{L}_{\text{left}}(v)$: sorted in the increasing order on the left endpoints of the intervals.
Storage of $I_{mid}$

$I_{mid}$ is stored in two lists at $v$:

- $L_{left}(v)$: sorted in the increasing order on the left endpoints of the intervals.
- $L_{right}(v)$: sorted in the decreasing order on their right endpoints.
Storage of $I_{mid}$

$I_{mid}$ is stored in two lists at $v$:

- $\mathcal{L}_{left}(v)$: sorted in the increasing order on the left endpoints of the intervals.
- $\mathcal{L}_{right}(v)$: sorted in the decreasing order on their right endpoints.

\[
\mathcal{L}_{left}: 3, 4, 5 \quad \quad \quad \quad \mathcal{L}_{right}: 5, 3, 4
\]
Total Storage

$I_{left}, I_{mid}, I_{right}$ are disjoint.

Every interval appears in only one set $I_{mid}$ (stored once in each of the two sorted lists).

- All associated lists require $O(n)$ storage.
- The tree uses $O(n)$ storage.
- The tree has height $O(\log n)$. 
III. Construction of the Interval Tree

ConstructIntervalTree($I$)

1. if $I = \emptyset$
2. then return an empty leaf
3. else compute $x_{mid}$
4. store it in a new node $v$
5. compute $I_{mid}, I_{left}, I_{right}$
6. construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
7. store them at $v$
8. $lc(v) \leftarrow$ ConstructIntervalTree($I_{left}$)
9. $rc(v) \leftarrow$ ConstructIntervalTree($I_{right}$)
10. return $v$
### III. Construction of the Interval Tree

ConstructIntervalTree($I$)

1. if $I = \emptyset$
2. then return an empty leaf
3. else compute $x_{mid}$
4. store it in a new node $v$
5. compute $I_{mid}, I_{left}, I_{right}$
6. construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
7. store them at $v$
8. $lc(v) \leftarrow$ ConstructIntervalTree($I_{left}$)
9. $rc(v) \leftarrow$ ConstructIntervalTree($I_{right}$)
10. return $v$

Presort the points and maintain ordered lists through recursive calls.
III. Construction of the Interval Tree

ConstructInveralTree(I)

1. if $I = \emptyset$
2. then return an empty leaf
3. else compute $x_{mid}$ \hspace{1cm} // $O(n)$
4. store it in a new node $v$
5. compute $I_{mid}, I_{left}, I_{right}$
6. construct sorted lists $\mathcal{L}_{left}(v)$ and $\mathcal{L}_{right}(v)$
7. store them at $v$
8. $lc(v) \leftarrow \text{ConstructIntervalTree}(I_{left})$
9. $rc(v) \leftarrow \text{ConstructIntervalTree}(I_{right})$
10. return $v$

Presort the points and maintain ordered lists through recursive calls.
III. Construction of the Interval Tree

ConstructIntervalTree($I$)

1. if $I = \emptyset$
2. then return an empty leaf
3. else compute $x_{\text{mid}}$ \quad \quad \quad \quad \quad \quad // $O(n)$
4. store it in a new node $v$
5. compute $I_{\text{mid}}, I_{\text{left}}, I_{\text{right}}$ \quad \quad \quad \quad // $O(n)$
6. construct sorted lists $L_{\text{left}}(v)$ and $L_{\text{right}}(v)$
7. store them at $v$
8. $lc(v) \leftarrow \text{ConstructIntervalTree} (I_{\text{left}})$
9. $rc(v) \leftarrow \text{ConstructIntervalTree} (I_{\text{right}})$
10. return $v$

Presort the points and maintain ordered lists through recursive calls.
III. Construction of the Interval Tree

ConstructIntervalTree(I)

1. if $I = \emptyset$
2. then return an empty leaf
3. else compute $x_{\text{mid}}$  
   // $O(n)$
4. store it in a new node $v$
5. compute $I_{\text{mid}}, I_{\text{left}}, I_{\text{right}}$  
   // $O(n)$
6. construct sorted lists $L_{\text{left}}(v)$ and $L_{\text{right}}(v)$
7. store them at $v$  
   // $O(n)$
8. $lc(v) \leftarrow \text{ConstructIntervalTree}(I_{\text{left}})$
9. $rc(v) \leftarrow \text{ConstructIntervalTree}(I_{\text{right}})$
10. return $v$

Presort the points and maintain ordered lists through recursive calls.
III. Construction of the Interval Tree

ConstructIntervalTree($I$)

1. if $I = \emptyset$
2. then return an empty leaf
3. else compute $x_{mid}$ // $O(n)$
4. store it in a new node $v$
5. compute $I_{mid}, I_{left}, I_{right}$ // $O(n)$
6. construct sorted lists $L_{left}(v)$ and $L_{right}(v)$
7. store them at $v$ // $O(n)$
8. $lc(v) \leftarrow \text{ConstructIntervalTree}(I_{left})$
9. $rc(v) \leftarrow \text{ConstructIntervalTree}(I_{right})$
10. return $v$

Presort the points and maintain ordered lists through recursive calls.

$\implies O(n)$ time before recursions on lines 8 and 9
III. Construction of the Interval Tree

ConstructIntervalTree($I$)

1. \textbf{if} $I = \emptyset$
2. \quad \textbf{then} return an empty leaf
3. \quad \textbf{else} compute $x_{\text{mid}}$ \hspace{1cm} // $O(n)$
4. \quad store it in a new node $v$
5. \quad compute $I_{\text{mid}}, I_{\text{left}}, I_{\text{right}}$ \hspace{1cm} // $O(n)$
6. \quad construct sorted lists $L_{\text{left}}(v)$ and $L_{\text{right}}(v)$
7. \quad store them at $v$ \hspace{1cm} // $O(n)$
8. $lc(v) \leftarrow \text{ConstructIntervalTree}(I_{\text{left}})$
9. $rc(v) \leftarrow \text{ConstructIntervalTree}(I_{\text{right}})$
10. return $v$

Presort the points and maintain ordered lists through recursive calls.

\implies O(n) \text{ time before recursions on lines 8 and 9}

\implies \text{Total construction time: } O(n \log n)
Query

Find all intervals containing $q_x$.

$q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$. 
Find all intervals containing $q_x$. 

- $q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$. 
- $q_x < x_{mid}(v)$
Query

Find all intervals containing $q_x$.

- $q_x = x_{\text{mid}}$ ends the query after reporting all intervals in $I_{\text{mid}}$.
- $q_x < x_{\text{mid}}(v)$
Find all intervals containing $q_x$.

- $q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$.
- $q_x < x_{mid}(v) \iff$ Search $I_{mid}$ and $I_{left}$.

$\forall q_x < x_{mid}(v) < q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$. 

$\forall q_x < x_{mid}(v) < q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$.
Find all intervals containing $q_x$.

- $q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$.
- $q_x < x_{mid}(v) \implies$ Search $I_{mid}$ and $I_{left}$.

$\ell_{left}(v)$

$I_{mid}$

$\ell_{right}(v)$

$I_{left}$

$I_{right}$

$s_1$

$s_2$

$s_3$

$s_4$

$s_5$
Find all intervals containing $q_x$.

- $q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$.
- $q_x < x_{mid}(v) \iff$ Search $I_{mid}$ and $I_{left}$.

\[ L_{left}(v) = \langle s_2, s_1, s_3 \rangle \]
Find all intervals containing $q_x$.

- $q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$.
- $q_x < x_{mid}(v) \iff$ Search $I_{mid}$ and $I_{left}$.

At $v$, start at the leftmost endpoint of $\mathcal{L}_{left}$.
Find all intervals containing $q_x$.

- $q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$.
- $q_x < x_{mid}(v) \iff$ Search $I_{mid}$ and $I_{left}$.

• At $v$, start at the leftmost endpoint of $L_{left}$.
• Report all the intervals containing $q_x$. 
Query

Find all intervals containing $q_x$.

- $q_x = x_{mid}$ ends the query after reporting all intervals in $I_{mid}$.
- $q_x < x_{mid}(v) \iff$ Search $I_{mid}$ and $I_{left}$.

At $v$, start at the leftmost endpoint of $\mathcal{L}_{left}$.

Report all the intervals containing $q_x$. 
Query

Find all intervals containing $q_x$.

- $q_x = x_{\text{mid}}$ ends the query after reporting all intervals in $I_{\text{mid}}$.
- $q_x < x_{\text{mid}}(v) \iff$ Search $I_{\text{mid}}$ and $I_{\text{left}}$.

\[ L_{\text{left}}(v) = \langle s_2, s_1, s_3 \rangle \]

- At $v$, start at the leftmost endpoint of $L_{\text{left}}$.
- Report all the intervals containing $q_x$.
- Stop at an interval that does not contain $q_x$ (i.e., its left endpoint is to the right of $q_x$).
Find all intervals containing \( q_x \).

- \( q_x = x_{mid} \) ends the query after reporting all intervals in \( I_{mid} \).
- \( q_x < x_{\text{mid}}(v) \) \( \iff \) Search \( I_{\text{mid}} \) and \( I_{\text{left}} \).

\[
L_{\text{left}}(v) = \langle s_2, s_1, s_3 \rangle
\]

At \( v \), start at the leftmost endpoint of \( L_{\text{left}} \).

Report all the intervals containing \( q_x \).

Stop at an interval that does not contain \( q_x \) (i.e., its left endpoint is to the right of \( q_x \)).

Query the left subtree.
Query (cont’d)

- \( q_x > x_{\text{mid}}(v) \)  \( \iff \)  Search \( I_{\text{mid}} \) and \( I_{\text{right}} \).

- At \( v \), start at the rightmost endpoint of \( L_{\text{right}} \).
- Report all the intervals containing \( q_x \).
- Stop at an interval does not contain \( q_x \).
- Query the right subtree.

\[ L_{\text{left}} = (s_3, s_1) \]

\[ L_{\text{right}} = (s_3, s_1) \]
Query (cont’d)

- $q_x > x_{mid}(v) \quad \Rightarrow \quad$ Search $I_{mid}$ and $I_{right}$.

- At $v$, start at the rightmost endpoint of $\mathcal{L}_{right}$.
- Report all the intervals containing $q_x$.
- Stop at an interval does not contain $q_x$.
- Query the right subtree.
Query (cont’d)

- $q_x > x_{\text{mid}}(v)$ \implies \text{Search } I_{\text{mid}} \text{ and } I_{\text{right}}.$

- At $v$, start at the rightmost endpoint of $L_{\text{right}}$.
- Report all the intervals containing $q_x$.
- Stop at an interval does not contain $q_x$.
- Query the right subtree.

**Query time:** $O(\log n + k)$
Query (cont’d)

- $q_x > x_{\text{mid}}(v)$  $\implies$  Search $I_{\text{mid}}$ and $I_{\text{right}}$.

- At $v$, start at the rightmost endpoint of $\mathcal{L}_{\text{right}}$.
- Report all the intervals containing $q_x$.
- Stop at an interval does not contain $q_x$.
- Query the right subtree.

Query time: $O(\log n + k)$

#reported intervals
IV. Query Segment with Finite Length

Query object: a vertical line segment \( q \): \( q_x \times [q_y, q_y'] \)

\( S = \{s_1, s_2, ..., s_n\} \): set of \( n \) horizontal segments
IV. Query Segment with Finite Length

Query object: a vertical line segment $q$: $q_x \times [q_y, q_y']$

$S = \{s_1, s_2, ..., s_n\}$: set of $n$ horizontal segments

$S: [s_x, s_x'] \times s_y$

$(s_x, s_y)$  \hspace{2cm} $(s_x, s_y)$
IV. Query Segment with Finite Length

Query object: a vertical line segment $q$: $q_x \times [q_y, q_y']$

$S = \{s_1, s_2, \ldots, s_n\}$: set of $n$ horizontal segments

$s$ intersects $q$ iff

i) $s_x \leq q_x \leq s_x'$

ii) $q_y \leq s_y \leq q_y'$
Modified Query

Modify the (earlier described) query with a vertical line object.

\[
(q_x, q_y) \quad \text{s} \quad (q_x, q_y')
\]

\[
x_{\text{mid}}
\]
Modified Query

Modify the (earlier described) query with a vertical line object.

- $q$ to the left of $x_{mid}$, i.e., $q_x \leq x_{mid}$.
Modify the (earlier described) query with a vertical line object.

- $q$ to the left of $x_{mid}$, i.e., $q_x \leq x_{mid}$.

- Every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query if

$$s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y$$
Modified Query

Modify the (earlier described) query with a vertical line object.

- $q$ to the left of $x_{\text{mid}}$, i.e., $q_x \leq x_{\text{mid}}$.

- Every segment $s = [s_x, s'_x] \times s_y \in I_{\text{mid}}$ (in the increasing order of left endpoint) satisfies the query if
  
  \[ s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y \]
  
  \[ x_{\text{mid}} \leq s'_x \]
Modified Query

Modify the (earlier described) query with a vertical line object.

- \( q \) to the left of \( x_{\text{mid}} \), i.e., \( q_x \leq x_{\text{mid}} \).

- Every segment \( s = [s_x, s_x'] \times s_y \in I_{\text{mid}} \) (in the increasing order of left endpoint) satisfies the query if

\[
\begin{align*}
s_x &\leq q_x \leq s_x' \quad \text{and} \quad q_y \leq s_y \leq q_y' \\
x_{\text{mid}} &\leq s_x' \\
q_x &\leq x_{\text{mid}}
\end{align*}
\]
Modified Query

Modify the (earlier described) query with a vertical line object.

- $q$ to the left of $x_{mid}$, i.e., $q_x \leq x_{mid}$.
- Every segment $s = [s_x, s_x'] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query if

$$s_x \leq [q_x \leq s_x'] \quad \text{and} \quad q_y \leq s_y \leq q_y'$$

$$x_{mid} \leq s_x'$$

$$q_x \leq x_{mid}$$
Modified Query

Modify the (earlier described) query with a vertical line object.

- $q$ to the left of $x_{mid}$, i.e., $q_x \leq x_{mid}$.

- Every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query if

  $$s_x \leq q_x \leq s'_x \quad \text{and} \quad q_y \leq s_y \leq q'_y$$

  $$x_{mid} \leq s'_x \quad \text{and} \quad q_x \leq x_{mid}$$
Modified Query

Modify the (earlier described) query with a vertical line object.

- $q$ to the left of $x_{\text{mid}}$, i.e., $q_x \leq x_{\text{mid}}$.

- Every segment $s = [s_x, s_x'] \times s_y \in I_{\text{mid}}$ (in the increasing order of left endpoint) satisfies the query if

$$s_x \leq q_x \leq s_x' \quad \text{and} \quad q_y \leq s_y \leq q_y'$$

$$x_{\text{mid}} \leq s_x'$$

$$q_x \leq x_{\text{mid}}$$

$$(s_x, s_y) \in R: \left[ -\infty, q_x \right] \times \left[ q_y, q_y' \right]$$

Left endpoint
Modify the (earlier described) query with a vertical line object.

- \( q \) to the left of \( x_{\text{mid}} \), i.e., \( q_x \leq x_{\text{mid}} \).

- Every segment \( s = [s_x, s_x'] \times s_y \in I_{\text{mid}} \) (in the increasing order of left endpoint) satisfies the query if

\[
\begin{align*}
    &x_{\text{mid}} \leq s_x' \\
    &q_x \leq x_{\text{mid}} \\
    &q_y \leq s_y \leq q_y' \\
\end{align*}
\]

\( (s_x, s_y) \in R: [-\infty, q_x] \times [q_y, q_y'] \)
Modify the (earlier described) query with a vertical line object.

- $q$ to the left of $x_{mid}$, i.e., $q_x \leq x_{mid}$.

- Every segment $s = [s_x, s'_x] \times s_y \in I_{mid}$ (in the increasing order of left endpoint) satisfies the query if

$$s_x \leq q_x \leq s'_x \quad \text{and} \quad q_y \leq s_y \leq q'_y$$

$$x_{mid} \leq s'_x$$

$$q_x \leq x_{mid}$$

$$(s_x, s_y) \in R: [-\infty, q_x] \times [q_y, q'_y]$$

Rectangular range query – 2D range tree
Modify the (earlier described) query with a vertical line object.

- **q** to the left of \( x_{\text{mid}} \), i.e., \( q_x \leq x_{\text{mid}} \).

- Every segment \( s = [s_x, s_x'] \times s_y \in I_{\text{mid}} \) (in the increasing order of left endpoint) satisfies the query if

\[
\begin{align*}
\begin{array}{c}
\text{Left endpoint} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\text{Rectangular range query – 2D range tree} \\
\end{array}
\end{align*}
\]
\( q \) to the right of \( x_{\text{mid}} \), i.e., \( q_x > x_{\text{mid}} \).
• $q$ to the right of $x_{mid}$, i.e., $q_x > x_{mid}$.

• Every segment $s \in I_{mid}$ satisfies the query if

\[ s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y \]
Modified Query (cont’d)

- $q$ to the right of $x_{\text{mid}}$, i.e., $q_x > x_{\text{mid}}$.

- Every segment $s \in I_{\text{mid}}$ satisfies the query if

$$s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y$$

$$s_x \leq x_{\text{mid}}$$
• $q$ to the right of $x_{\text{mid}}$, i.e., $q_x > x_{\text{mid}}$.

- Every segment $s \in I_{\text{mid}}$ satisfies the query if
  
  $$s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y$$

  $$s_x \leq x_{\text{mid}}$$
  $$x_{\text{mid}} < q_x$$
• $q$ to the right of $x_{\text{mid}}$, i.e., $q_x > x_{\text{mid}}$.

• Every segment $s \in I_{\text{mid}}$ satisfies the query if

\[ s_x \leq q_x \leq s'_x \text{ and } q_y \leq s_y \leq q'_y \]

\[
\begin{align*}
&s_x \leq q_x \\
&x_{\text{mid}} < q_x
\end{align*}
\]
- $q$ to the right of $x_{\text{mid}}$, i.e., $q_x > x_{\text{mid}}$.

- Every segment $s \in I_{\text{mid}}$ satisfies the query if

\[
\begin{align*}
    s_x & \leq q_x \leq s'_x, \\
    q_y & \leq s_y \leq q'_y
\end{align*}
\]

\[
\begin{align*}
    s_x & \leq x_{\text{mid}}, \\
    x_{\text{mid}} & < q_x
\end{align*}
\]
Modified Query (cont’d)

- $q$ to the right of $x_{\text{mid}}$, i.e., $q_x > x_{\text{mid}}$.

- Every segment $s \in I_{\text{mid}}$ satisfies the query if $s_x \leq q_x \leq s'_x$ and $q_y \leq s_y \leq q'_y$

$$(s'_x, s_y) \in R' : [q_x, \infty] \times [q_y, q'_y]$$

Right endpoint
- $q$ to the right of $x_{mid}$, i.e., $q_x > x_{mid}$.

- Every segment $s \in I_{mid}$ satisfies the query if $s_x \leq q_x \leq s_x'$ and $q_y \leq s_y \leq q_y'$.

$s_x \leq x_{mid}$

$x_{mid} < q_x$

$(s_x', s_y) \in R': [q_x, \infty] \times [q_y, q_y']$

Right endpoint
Modified Query (cont’d)

- $q$ to the right of $x_{mid}$, i.e., $q_x > x_{mid}$.

- Every segment $s \in I_{mid}$ satisfies the query if

$$s_x \leq q_x \leq s_x'$$ and

$$q_y \leq s_y \leq q_y'$$

- Query the right subtree.

$(s_x', s_y) \in R': [q_x, \infty] \times [q_y, q_y']$

Right endpoint
Data Structure

Main structure: interval tree $\mathcal{T}$ on $x$-coordinates.

At each node $v$, replace

$$\mathcal{L}_{left}(v) \quad \mathcal{L}_{right}(v)$$

$I_{left}$ \quad $I_{right}$
Data Structure

Main structure: interval tree $T$ on $x$-coordinates.

At each node $v$, replace

- $L_{left}(v)$ with a range tree $T_{left}(v)$ on the left endpoints of $I_{mid}(v)$. 

Main structure: interval tree $T$ on $x$-coordinates.

At each node $v$, replace

- $L_{left}(v)$ with a range tree $T_{left}(v)$ on the left endpoints of $I_{mid}(v)$. 
Main structure: interval tree $\mathcal{T}$ on $x$-coordinates.

At each node $v$, replace

- $\mathcal{L}_{\text{left}}(v)$ with a range tree $\mathcal{T}_{\text{left}}(v)$ on the left endpoints of $I_{\text{mid}}(v)$.
- $\mathcal{L}_{\text{right}}(v)$ with a range tree $\mathcal{T}_{\text{right}}(v)$ on the right endpoints of $I_{\text{mid}}(v)$.
Main structure: interval tree $\mathcal{T}$ on $x$-coordinates.

At each node $v$, replace

- $\mathcal{L}_{left}(v)$ with a range tree $\mathcal{T}_{left}(v)$ on the left endpoints of $I_{mid}(v)$.
- $\mathcal{L}_{right}(v)$ with a range tree $\mathcal{T}_{right}(v)$ on the right endpoints of $I_{mid}(v)$.
Query with $[-\infty, q_x] \times [q_y, q_y']$ performed on $T_{left}(v)$. 
Range Queries

- Query with \([-\infty, q_x] \times [q_y, q_y']\) performed on $T_{left}(v)$.

Each range tree $T_{left}(v)$ (or $T_{right}(v)$) has

- a main tree on the $x$-coordinate
- a secondary structure associated with every internal node on the $y$-coordinate.
Range Queries

- Query with $[-\infty, q_x] \times [q_y, q_y']$ performed on $T_{left}(v)$.
- Query with $[q_x, \infty] \times [q_y, q_y']$ performed on $T_{right}(v)$.

Each range tree $T_{left}(v)$ (or $T_{right}(v)$) has

- a main tree on the $x$-coordinate
- a secondary structure associated with every internal node on the $y$-coordinate.
Query Time

Perform a query on the range tree $T_{left}(v)$ instead of traversing the sorted list $L_{left}(v)$. 
Query Time

Perform a query on the range tree $T_{left}(v)$ instead of traversing the sorted list $L_{left}(v)$.

$\circ O(\log n)$ nodes on of the search path in the interval tree.
Query Time

Perform a query on the range tree $T_{left}(v)$ instead of traversing the sorted list $L_{left}(v)$.

- $O(\log n)$ nodes on the search path in the interval tree.
- at each such node $v$, $O(\log^2 n + k_v)$ time spent on the range tree $T_{left}(v)$ or $T_{right}(v)$. 
Query Time

Perform a query on the range tree $T_{left}(v)$ instead of traversing the sorted list $L_{left}(v)$.

- $O(\log n)$ nodes on of the search path in the interval tree.
- at each such node $v$, $O(\log^2 n + k_v)$ time spent on the range tree $T_{left}(v)$ or $T_{right}(v)$.

# reported segments in the range tree
Query Time

Perform a query on the range tree $T_{left}(v)$ instead of traversing the sorted list $L_{left}(v)$.

- $O(\log n)$ nodes on of the search path in the interval tree.

- at each such node $v$, $O(\log^2 n + k_v)$ time spent on the range tree $T_{left}(v)$ or $T_{right}(v)$.

$\#$ reported segments
in the range tree

Total query time: $O(\log^3 n + k_v \log n)$
Query Time

Perform a query on the range tree $\mathcal{T}_{left}(v)$ instead of traversing the sorted list $\mathcal{L}_{left}(v)$.

- $O(\log n)$ nodes on of the search path in the interval tree.

- at each such node $v$, $O(\log^2 n + k_v)$ time spent on the range tree $\mathcal{T}_{left}(v)$ or $\mathcal{T}_{right}(v)$.

Number of reported segments in the range tree $\mathcal{T}_{left}(v)$ or $\mathcal{T}_{right}(v)$.

Total query time: $O(\log^3 n + k_v \log n)$

Reduced to (using fractional cascading): $O(\log^2 n + k_v)$
Query Time

Perform a query on the range tree $T_{\text{left}}(v)$ instead of traversing the sorted list $L_{\text{left}}(v)$.

- $O(\log n)$ nodes on of the search path in the interval tree.
- at each such node $v$, $O(\log^2 n + k_v)$ time spent on the range tree $T_{\text{left}}(v)$ or $T_{\text{right}}(v)$.

# reported segments in the range tree

Total query time: $O(\log^3 n + k_v \log n)$

(reduced to (using fractional cascading))

$O(\log^2 n + k_v)$

Storage: $O(n \log n)$
Query Time

Perform a query on the range tree $T_{left}(v)$ instead of traversing the sorted list $L_{left}(v)$.

- $O(\log n)$ nodes on of the search path in the interval tree.
- at each such node $v$, $O(\log^2 n + k_v)$ time spent on the range tree $T_{left}(v)$ or $T_{right}(v)$.

Total query time: $O(\log^3 n + k_v \log n)$

reduced to (using fractional cascading)

$O(\log^2 n + k_v)$

Storage: $O(n \log n)$

Construction time: $O(n \log n)$