Outline:

I. Heap-based point queries

II. Structure of a PST

III. Query

IV. Correctness and running time
Windowing Queries with an Interval Tree

\[ I_{mid} \]

Range tree on left endpoints

\[ J_{left}(v) \]

Range tree on right endpoints

\[ J_{right}(v) \]

\[ I_{left} \]

\[ I_{right} \]

\[ (q_x, q_y) \]

\[ (q_x, q'_y) \]

\[ R' \]

\[ [q_x, \infty] \times [q_y, q'_y] \]
Windowing Queries with an Interval Tree

Complicated data structure due to the uses of range tree and fractional cascading for efficiency.
Windowing Queries with an Interval Tree

- Complicated data structure due to the uses of range tree and fractional cascading for efficiency.

- High storage: $O(n \log n)$
Windowing Queries with an Interval Tree

- High storage: $O(n \log n)$
- Complicated data structure due to the uses of range tree and fractional cascading for efficiency.

Improvement?
Windowing Queries with an Interval Tree

- High storage: $O(n \log n)$
- Complicated data structure due to the uses of range tree and fractional cascading for efficiency.

**Improvement?**

- Explore that the query range is unbounded on one side ($-\infty$ or $\infty$ over $x$).
Windowing Queries with an Interval Tree

- Complicated data structure due to the uses of range tree and fractional cascading for efficiency.

- High storage: $O(n \log n)$

Improvement?

- Explore that the query range is unbounded on one side ($-\infty$ or $\infty$ over $x$).
- Use a simpler data structure to cut down storage to $O(n)$.
Windowing Queries with an Interval Tree

- High storage: $O(n \log n)$
- Complicated data structure due to the uses of range tree and fractional cascading for efficiency.

**Improvement?**
- Explore that the query range is unbounded on one side ($-\infty$ or $\infty$ over $x$).
- Use a simpler data structure to cut down storage to $O(n)$.

Priority search tree
I. Query Problem

Point set: $P = \{p_1, p_2, \ldots, p_n\}$

Query range: $(-\infty, q_x] \times [q_y, q'_y]$
I. Query Problem

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)

Query range: \((−∞, q_x] \times [q_y, q_y']\)

Let us think small to start with the 1D case first.

Range: \((−∞, q_x]\)
I. Query Problem

Point set: $P = \{p_1, p_2, \ldots, p_n\}$

Query range: $(-\infty, q_x] \times [q_y, q'_y]$

Let us think small to start with the 1D case first.

Range: $(-\infty, q_x]$

- Order the points $p_1 < p_2 < \cdots < p_n$. 

\[ \text{\includegraphics[width=\textwidth]{diagram}} \]
I. Query Problem

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)

Query range: \((\neg\infty, q_x] \times [q_y, q'_y]\)

Let us think small to start with the 1D case first.

Range: \((\neg\infty, q_x]\)

- Order the points \( p_1 < p_2 < \cdots < p_n \).
- Start at the leftmost point and walk toward right until \( p_{k+1} > q_x \).
I. Query Problem

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)

Query range: \( (-\infty, q_x] \times [q_y, q'_y] \)

Let us think small to start with the 1D case first.

Range: \( (-\infty, q_x] \)

- Order the points \( p_1 < p_2 < \ldots < p_n \).
- Start at the leftmost point and walk toward right until \( p_{k+1} > q_x \).
I. Query Problem

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)

Query range: \((-\infty, q_x] \times [q_y, q'_y]\)

Let us think small to start with the 1D case first.

Range: \((-\infty, q_x]\)

- Order the points \( p_1 < p_2 < \ldots < p_n \).
- Start at the leftmost point and walk toward right until \( p_{k+1} > q_x \).
- Report \( p_1, p_2, \ldots, p_k \) during the walk.
I. Query Problem

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)

Query range: \( (-\infty, q_x] \times [q_y, q'_y] \)

Let us think small to start with the 1D case first.

Range: \( (-\infty, q_x] \)

- Order the points \( p_1 < p_2 < \cdots < p_n \).

- Start at the leftmost point and walk toward right until \( p_{k+1} > q_x \).

- Report \( p_1, p_2, \ldots, p_k \) during the walk

\[ O(1 + k) \]
Moving on to 2D Query

Among the points with $x$-coordinates in $(-\infty, q_x]$, select those whose $y$-coordinates are in $[q_y, q_y']$. 

\{1, 3, 11, 6, 9, 13, 22, 8, 40\}
Moving on to 2D Query

Among the points with $x$-coordinates in $(-\infty, q_x]$, select those whose $y$-coordinates are in $[q_y, q'_y]$.

- How to exploit $(-\infty, q_x]$ being half-open?
Moving on to 2D Query

Among the points with $x$-coordinates in $(-\infty, q_x]$, select those whose $y$-coordinates are in $[q_y, q'_y]$.

- How to exploit $(-\infty, q_x]$ being half-open?

- Use a min heap.

Property: Every internal node stores the minimum value of the subtree rooted at the node.
Moving on to 2D Query

Among the points with $x$-coordinates in $(-\infty, q_x]$, select those whose $y$-coordinates are in $[q_y, q'_y]$.

- How to exploit $(-\infty, q_x]$ being half-open?

- Use a min heap.

Property: Every internal node stores the minimum value of the subtree rooted at the node.

\[
\begin{align*}
&\text{Property} & \text{Every internal node stores the minimum value of the subtree rooted at the node.} \\
&1 & \\
&3 & 11 \\
&6 & 9 & 13 & 22 \\
&8 & 40 & & \\
\{1, 3, 11, 6, 9, 13, 22, 8, 40\}
\end{align*}
\]
First Query Range Handled by a Heap

Point set: $P = \{p_1, p_2, \ldots, p_n\}$

Query range: $(-\infty, q_x]$
First Query Range Handled by a Heap

Point set: \( P = \{p_1, p_2, ..., p_n\} \)  
Query range: \((−\infty, q_x] \)

• Walk down the tree.
First Query Range Handled by a Heap

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)  \hspace{1cm} Query range: \((-\infty, q_x]\)

- Walk down the tree.
- At each node with stored point \( p = (p_x, p_y) \) check if \( p_x \leq q_x \).
First Query Range Handled by a Heap

Point set: $P = \{p_1, p_2, ..., p_n\}$

Query range: $(-\infty, q_x]$

- Walk down the tree.

- At each node with stored point $p = (p_x, p_y)$ check if $p_x \leq q_x$.

  - If yes, report $p$ and continue in both subtrees $L$ and $R$. 
First Query Range Handled by a Heap

Point set: $P = \{p_1, p_2, ..., p_n\}$

Query range: $(-\infty, q_x]$  

- Walk down the tree.

- At each node with stored point $p = (p_x, p_y)$ check if $p_x \leq q_x$.

  - If yes, report $p$ and continue in both subtrees $L$ and $R$.

  - Otherwise ($p_x > q_x$), abort the subtree $T(p)$ rooted at $p$. 

First Query Range Handled by a Heap

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)

Query range: \((-\infty, q_x]\)

- Walk down the tree.

- At each node with stored point \( p = (p_x, p_y) \) check if \( p_x \leq q_x \).

- If yes, report \( p \) and continue in both subtrees \( L \) and \( R \).

- Otherwise \((p_x > q_x)\), abort the subtree \( \mathcal{T}(p) \) rooted at \( p \).

Every node \( r = (r_x, r_y) \neq p \) in \( \mathcal{T}(p) \) satisfies 
\[
    r_x \geq p_x > q_x
\]
Example

$(-\infty, 12]$ on the heap below.
Example

$(-\infty, 12]$ on the heap below.

```
        1
       / \
       3   11
      / \  /  \
     6   9 13  22
    / \  /  /  \
   8  40 24 37
```
Example

\((-\infty, 12]\) on the heap below.
Example

$(-\infty, 12]$ on the heap below.
$(-\infty, 12]$ on the heap below.
Example

$(-\infty, 12]$ on the heap below.
$(-\infty, 12]$ on the heap below.
Example

$(-\infty, 12]$ on the heap below.

Report 1, 3, 6, 8, 9, 11.
Both Query Ranges Handled by a Heap

Point set: \( P = \{ p_1, p_2, \ldots, p_n \} \)

Query range: \( (-\infty, q_x] \times [q_y, q'_y] \)
Both Query Ranges Handled by a Heap

Point set: \( P = \{p_1, p_2, ..., p_n\} \)  
Query range: \((-\infty, q_x] \times [q_y, q'_y]\)

- A set can be represented by many heaps, each representing a way of partitioning the set.
Both Query Ranges Handled by a Heap

Point set: $P = \{p_1, p_2, ..., p_n\}$

Query range: $(-\infty, q_x] \times [q_y, q'_y]$

- A set can be represented by many heaps, each representing a way of partitioning the set.

- How to integrate the information about the $y$-coordinate without using the associated structures?
Both Query Ranges Handled by a Heap

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)

Query range: \((−\infty, q_x] \times [q_y, q'_y]\)

- A set can be represented by many heaps, each representing a way of partitioning the set.

- How to integrate the information about the \( y \)-coordinate without using the associated structures?

- We can choose a heap that partitions the set according to the \( y \)-coordinate.
Both Query Ranges Handled by a Heap

Point set: \( P = \{p_1, p_2, \ldots, p_n\} \)  

Query range: \((-\infty, q_x] \times [q_y, q'_y]\)

- A set can be represented by many heaps, each representing a way of partitioning the set.

- How to integrate the information about the \( y \)-coordinate without using the associated structures?

- We can choose a heap that partitions the set according to the \( y \)-coordinate.

Split the remainder of the set into two subsets such that the points in one subset have their \( y \)-coordinates less than those of the points in the other subset.
II. Priority Search Tree (PST)
II. Priority Search Tree (PST)

- $p_4$ is at the root because it has the smallest $x$-coordinate.
II. Priority Search Tree (PST)

- $p_4$ is at the root because it has the smallest $x$-coordinate.
- Of the remaining 6 points, $p_7$ has the median $y$-coordinate.
II. Priority Search Tree (PST)

- $p_4$ is at the root because it has the smallest $x$-coordinate.
- Of the remaining 6 points, $p_7$ has the median $y$-coordinate.
- This median splits them into two groups: $p_2, p_6, p_7$ stored in the left (lower) subtree and $p_1, p_3, p_5$ stored in the right (upper) subtree.
II. Priority Search Tree (PST)

- \( p_4 \) is at the root because it has the smallest \( x \)-coordinate.
- Of the remaining 6 points, \( p_7 \) has the median \( y \)-coordinate.
- This median splits them into two groups: \( p_2, p_6, p_7 \) stored in the left (lower) subtree and \( p_1, p_3, p_5 \) stored in the right (upper) subtree.
II. Priority Search Tree (PST)

- $p_4$ is at the root because it has the smallest $x$-coordinate.
- Of the remaining 6 points, $p_7$ has the median $y$-coordinate.
- This median splits them into two groups: $p_2$, $p_6$, $p_7$ stored in the left (lower) subtree and $p_1$, $p_3$, $p_5$ stored in the right (upper) subtree.
II. Priority Search Tree (PST)

- $p_4$ is at the root because it has the smallest $x$-coordinate.
- Of the remaining 6 points, $p_7$ has the median $y$-coordinate.
- This median splits them into two groups: $p_2, p_6, p_7$ stored in the left (lower) subtree and $p_1, p_3, p_5$ stored in the right (upper) subtree.
- $p_6$ and $p_1$ are the roots of the two subtrees because they have the smallest $x$-coordinates in their groups, and so on.
II. Priority Search Tree (PST)

• $p_4$ is at the root because it has the smallest $x$-coordinate.

• Of the remaining 6 points, $p_7$ has the median $y$-coordinate.

• This median splits them into two groups: $p_2, p_6, p_7$ stored in the left (lower) subtree and $p_1, p_3, p_5$ stored in the right (upper) subtree.

• $p_6$ and $p_1$ are the roots of the two subtrees because they have the smallest $x$-coordinates in their groups, and so on.
Formal Definition of the PST

**Assumption** No two points have the same $x$- or $y$-coordinate.

(Easily removable with lexicographic ordering)
Formal Definition of the PST

Assumption No two points have the same $x$- or $y$-coordinate.
(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
Formal Definition of the PST

**Assumption** No two points have the same $x$- or $y$-coordinate.

(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{min} \in P$ with the smallest $x$-coordinate.
Formal Definition of the PST

Assumption No two points have the same $x$- or $y$-coordinate.
(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{min} \in P$ with the smallest $x$-coordinate.
  - $y_{mid}$: median $y$-coordinate of the remaining points.
Formal Definition of the PST

Assumption No two points have the same $x$- or $y$-coordinate.
(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{min} \in P$ with the smallest $x$-coordinate.
  - $y_{mid}$: median $y$-coordinate of the remaining points.
  - $P_{below} = \{ p \in P \setminus \{ p_{min} \} | p_y \leq y_{mid} \}$. 
Formal Definition of the PST

Assumption
No two points have the same $x$- or $y$-coordinate.
(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{\text{min}} \in P$ with the smallest $x$-coordinate.
  - $y_{\text{mid}}$ : median $y$-coordinate of the remaining points.
  - $P_{\text{below}} = \{ p \in P \setminus \{p_{\text{min}}\} \mid p_y \leq y_{\text{mid}} \}$.
  - $P_{\text{above}} = \{ p \in P \setminus \{p_{\text{min}}\} \mid p_y > y_{\text{mid}} \}$.
Formal Definition of the PST

Assumption No two points have the same $x$- or $y$-coordinate.
(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{\text{min}} \in P$ with the smallest $x$-coordinate.
  - $y_{\text{mid}}$: median $y$-coordinate of the remaining points.
  - $P_{\text{below}} = \{p \in P \setminus \{p_{\text{min}}\} | p_y \leq y_{\text{mid}}\}$.
  - $P_{\text{above}} = \{p \in P \setminus \{p_{\text{min}}\} | p_y > y_{\text{mid}}\}$.
  - Create $\nu(p_{\text{min}}, y_{\text{mid}})$
Formal Definition of the PST

Assumption No two points have the same $x$- or $y$-coordinate.

(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{min} \in P$ with the smallest $x$-coordinate.
  - $y_{mid}$ : median $y$-coordinate of the remaining points.
  - $P_{below} = \{p \in P \setminus \{p_{min}\} | p_y \leq y_{mid}\}$.
  - $P_{above} = \{p \in P \setminus \{p_{min}\} | p_y > y_{mid}\}$.
  - Create

\[ v \quad p_{min}, \quad y_{mid} \]

\[ P_{below} \]
Formal Definition of the PST

Assumption No two points have the same $x$- or $y$-coordinate.

(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{min} \in P$ with the smallest $x$-coordinate.
  - $y_{mid}$ : median $y$-coordinate of the remaining points.
  - $P_{below} = \{ p \in P \setminus \{ p_{min}\} \mid p_y \leq y_{mid}\}$.
  - $P_{above} = \{ p \in P \setminus \{ p_{min}\} \mid p_y > y_{mid}\}$.
  - Create
Formal Definition of the PST

Assumption No two points have the same $x$- or $y$-coordinate.

(Easily removable with lexicographic ordering)

- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{min} \in P$ with the smallest $x$-coordinate.
  - $y_{mid}$ : median $y$-coordinate of the remaining points.
  - $P_{below} = \{p \in P \setminus \{p_{min}\} \mid p_y \leq y_{mid}\}$.
  - $P_{above} = \{p \in P \setminus \{p_{min}\} \mid p_y > y_{mid}\}$.

Create

$$
\begin{array}{c}
\text{\Large{$v$}} \\
\text{\Large{$p_{min}, y_{mid}$}} \\
\text{$P_{below}$} \\
\text{$P_{above}$}
\end{array}$$

- Points are not stored at leaves only.
- Every node stores a different point.
Construction Time

\( p_{\text{min}, \ y_{\text{mid}}} \)

\( O(n \log n) \) if recursively (top-down)
Construction Time

\[ p_{\text{min}}, y_{\text{mid}} \]

- \( O(n \log n) \) if recursively (top-down)
- \( O(n) \) if
  - the points are pre-sorted on \( y \)-coordinate, and
  - constructed bottom-up in the way of building a heap.
III. Query

Query range: \((-\infty, q_x] \times [q_y, q'_y]\)

1) Search the PST with \(q_y\) and \(q'_y\) by comparing them with the \(y_{\text{mid}}\) value at each node.
III. Query

Query range: \((-\infty, q_x] \times [q_y, q'_y]\)

1) Search the PST with \(q_y\) and \(q'_y\) by comparing them with the \(y_{mid}\) value at each node.

1D range searching
III. Query

Query range: \((-\infty, q_x] \times [q_y, q_y']\)

1) Search the PST with \(q_y\) and \(q_y'\) by comparing them with the \(y_{\text{mid}}\) value at each node.

1D range searching

The two searches end at the nodes \(\mu\) and \(\mu'\), respectively.
Query range: $(-\infty, q_x] \times [q_y, q'_y]$

1) Search the PST with $q_y$ and $q'_y$ by comparing them with the $y_{mid}$ value at each node.

1D range searching

The two searches end at the nodes $\mu$ and $\mu'$, respectively.
III. Query

Query range: \((-\infty, q_x] \times [q_y, q_y']\)

1) Search the PST with \(q_y\) and \(q_y'\) by comparing them with the \(y_{mid}\) value at each node.

1D range searching

The two searches end at the nodes \(\mu\) and \(\mu'\), respectively.
III. Query

Query range: \((-\infty, q_x] \times [q_y, q_y']\)

1) Search the PST with \(q_y\) and \(q_y'\) by comparing them with the \(y_{\text{mid}}\) value at each node.

1D range searching

The two searches end at the nodes \(\mu\) and \(\mu'\), respectively.

Selected subtrees for future searches based on \(x\)-coordinates.
Nodes on the Search Paths

Check every node $v$ on every one of the three paths, $r \sim v_{\text{split}}$, $v_{\text{split}} \sim \mu$ and $v_{\text{split}} \sim \mu'$ to see if

$$p(v) \in (\infty, q_x] \times [q_y, q'_y]$$
2) Search every selected subtree based on $x$-coordinate as in a one-dimensional array.
Search in the Selected Subtrees

2) Search every selected subtree based on $x$-coordinate as in a one-dimensional array.

```java
ReportInSubtree(v, q_x)
// incorrect in the text for omitting // the case of $v$ as a leaf
1. if $(p(v))_x \leq q_x$
2. then report $p(v)$
3. if $v$ is not a leaf
4. then ReportInSubtree(lc(v), q_x)
5. ReportInSubtree(rc(v), q_x)
```
IV. Correctness of ReportInSubtree()

**Lemma** ReportInSubtree(v, qx) reports in \( O(1 + k_v) \) time all the \( k_v \) points in the subtree \( \mathcal{T}(v) \) whose \( x \)-coordinate is at most \( q_x \).
IV. Correctness of \texttt{ReportInSubtree()}

**Lemma** \texttt{ReportInSubtree}(v, q_x) reports in $O(1 + k_v)$ time all the $k_v$ points in the subtree $T(v)$ whose $x$-coordinate is at most $q_x$.

**Proof** Consider a node $\mu$ in $T(v)$ such that its stored point $p(\mu)$ satisfies \((p(\mu))_x \leq q_x\).
IV. Correctness of ReportInSubtree()

**Lemma** ReportInSubtree\((v, q_x)\) reports in \(O(1 + k_v)\) time all the \(k_v\) points in the subtree \(\mathcal{T}(v)\) whose \(x\)-coordinate is at most \(q_x\).

**Proof** Consider a node \(\mu\) in \(\mathcal{T}(v)\) such that its stored point \(p(\mu)\) satisfies \((p(\mu))_x \leq q_x\).

- Along the (upward) path \(\mu \bowtie v\) the \(x\)-coordinates of the stored points decrease.
IV. Correctness of ReportInSubtree()

**Lemma** ReportInSubtree($v, q_x$) reports in $O(1 + k_v)$ time all the $k_v$ points in the subtree $T(v)$ whose $x$-coordinate is at most $q_x$.

**Proof** Consider a node $\mu$ in $T(v)$ such that its stored point $p(\mu)$ satisfies $(p(\mu))_x \leq q_x$.

- Along the (upward) path $\mu \leadsto v$ the $x$-coordinates of the stored points decrease.
IV. Correctness of ReportInSubtree()

**Lemma** ReportInSubtree(𝑣, 𝑞ₓ) reports in $O(1 + k_v)$ time all the $k_v$ points in the subtree $T(𝑣)$ whose $x$-coordinate is at most $q_x$.

**Proof** Consider a node $\mu$ in $T(𝑣)$ such that its stored point $p(\mu)$ satisfies $(p(\mu))_x \leq q_x$.

- Along the (upward) path $\mu \rightsquigarrow v$ the $x$-coordinates of the stored points decrease.

\[ q_x \geq (p(\mu))_x > \cdots > (p(u))_x > \cdots > (p(v))_x \]
IV. Correctness of ReportInSubtree()

**Lemma** ReportInSubtree(v,qₓ) reports in $O(1 + k_v)$ time all the $k_v$ points in the subtree $T(v)$ whose $x$-coordinate is at most $q_x$.

**Proof** Consider a node $\mu$ in $T(v)$ such that its stored point $p(\mu)$ satisfies $(p(\mu))_x \leq q_x$.

- Along the (upward) path $\mu \leadsto v$ the $x$-coordinates of the stored points decrease.

$$q_x \geq (p(\mu))_x > \cdots > (p(u))_x > \cdots > (p(v))_x$$

Recursive calls to ReportInSubtree() are invoked at all the nodes on the downward path $v \leadsto \mu$. 

![Diagram](https://via.placeholder.com/150)
IV. Correctness of ReportInSubtree()

Lemma ReportInSubtree\((v, q_x)\) reports in \(O(1 + k_v)\) time all the \(k_v\) points in the subtree \(T(v)\) whose \(x\)-coordinate is at most \(q_x\).

Proof Consider a node \(\mu\) in \(T(v)\) such that its stored point \(p(\mu)\) satisfies \((p(\mu))_x \leq q_x\).

- Along the (upward) path \(\mu \leadsto v\) the \(x\)-coordinates of the stored points decrease.

\[
q_x \geq (p(\mu))_x > \cdots > (p(u))_x > \cdots > (p(v))_x
\]

Recursive calls to ReportInSubtree are invoked at all the nodes on the downward path \(v \leadsto \mu\).

\(p(\mu)\) is reported.
IV. Correctness of ReportInSubtree()

**Lemma** \(\text{ReportInSubtree}(v, q_x)\) reports in \(O(1 + k_v)\) time all the \(k_v\) points in the subtree \(T(v)\) whose \(x\)-coordinate is at most \(q_x\).

**Proof** Consider a node \(\mu\) in \(T(v)\) such that its stored point \(p(\mu)\) satisfies \((p(\mu))_x \leq q_x\).

- Along the (upward) path \(\mu \sim v\) the \(x\)-coordinates of the stored points decrease.

\[
q_x \geq (p(\mu))_x > \cdots > (p(u))_x > \cdots > (p(v))_x
\]

Recursive calls to \(\text{ReportInSubtree}\) are invoked at all the nodes on the downward path \(v \sim \mu\).

\(p(\mu)\) is reported.

The time \(O(1 + k_v)\) follows from \(O(1)\) effort spent on each node.
IV. Correctness of ReportInSubtree()

**Lemma** ReportInSubtree($v, q_x$) reports in $O(1 + k_v)$ time all the $k_v$ points in the subtree $T(v)$ whose $x$-coordinate is at most $q_x$.

**Proof** Consider a node $\mu$ in $T(v)$ such that its stored point $p(\mu)$ satisfies $(p(\mu))_x \leq q_x$.

- Along the (upward) path $\mu \leadsto v$ the $x$-coordinates of the stored points decrease.
  
  $q_x \geq (p(\mu))_x > \cdots > (p(u))_x > \cdots > (p(v))_x$

  Recursive calls to ReportInSubtree are invoked at all the nodes on the downward path $v \leadsto \mu$.

  $p(\mu)$ is reported.

The time $O(1 + k_v)$ follows from $O(1)$ effort spent on each node.
Query Algorithm

QueryPrioSearchTree(\(T, (-\infty, q_x] \times [q_y, q'_y]\))

// \(r\) is the root of \(T\)

1. search with \(q_y\) and \(q'_y\) in \(T\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{\text{split}}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \sim \mu\) or \(v_{\text{split}} \sim \mu'\)
4. 
   do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
5.       then report \(p(v)\)
6. for each node \(v\) on \(v_{\text{split}} \sim \mu\)
7. 
   do if the path goes left at \(v\)
8.       then ReportInSubtree(rc(v), q_x)
9. for each node \(v\) on \(v_{\text{split}} \sim \mu'\)
10. do if the path goes right at \(v\)
11.       then ReportInSubtree(lc(v), q_x)
Example of Execution

QueryPrioSearchTree(\( \mathcal{T} \), \((-\infty, q_x] \times [q_y, q'_y]\))

// r is the root of \( \mathcal{T} \)
1. search with \( q_y \) and \( q'_y \) in \( \mathcal{T} \), ending at the nodes \( \mu \) and \( \mu' \)
2. let \( v_{\text{split}} \) be the node where the two paths split.
3. for each node \( v \) on the path \( r \leadsto \mu \) or \( v_{\text{split}} \leadsto \mu' \)
4. do if \( p(v) \in (-\infty, q_x] \times [q_y, q'_y] \)
5. then report \( p(v) \)
6. for each node \( v \) on \( v_{\text{split}} \leadsto \mu \)
7. do if the path goes left at \( v \)
8. then ReportInSubtree(\( rc(v) \), \( q_x \))
9. for each node \( v \) on \( v_{\text{split}} \leadsto \mu' \)
10. do if the path goes right at \( v \)
11. then ReportInSubtree(\( lc(v) \), \( q_x \))
Example of Execution

QueryPrioSearchTree(\(T, (-\infty, q_x] \times [q_y, q'_y]\))
// \(r\) is the root of \(T\)
1. search with \(q_y\) and \(q'_y\) in \(T\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{split}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \sim \mu\) or \(v_{split} \sim \mu'\)
4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
5. then report \(p(v)\)
6. for each node \(v\) on \(v_{split} \sim \mu\)
7. do if the path goes left at \(v\)
8. then ReportInSubtree(\(rc(v), q_x\))
9. for each node \(v\) on \(v_{split} \sim \mu'\)
10. do if the path goes right at \(v\)
11. then ReportInSubtree(\(lc(v), q_x\))

\(v_{split} = p_4\)
Example of Execution

QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q_y']$)

// $r$ is the root of $\mathcal{T}$
1. search with $q_y$ and $q_y'$ in $\mathcal{T}$, ending at the nodes $\mu$ and $\mu'$
2. let $v_{\text{split}}$ be the node where the two paths split.
3. for each node $v$ on the path $r \sim \mu$ or $v_{\text{split}} \sim \mu'$
4. do if $p(v) \in (-\infty, q_x] \times [q_y, q_y']$
5. then report $p(v)$
6. for each node $v$ on $v_{\text{split}} \sim \mu$
7. do if the path goes left at $v$
8. then ReportInSubtree($rc(v)$, $q_x$)
9. for each node $v$ on $v_{\text{split}} \sim \mu'$
10. do if the path goes right at $v$
11. then ReportInSubtree($lc(v)$, $q_x$)

$v_{\text{split}} = p_4$
Example of Execution

```
QueryPrioSearchTree(𝒯, (−∞, qₓ] × [qᵧ, qᵧ′])
// r is the root of 𝒯
1. search with qᵧ and qᵧ′ in 𝒯, ending at the nodes μ and μ′
2. let v_split be the node where the two paths split.
3. for each node v on the path r ↝ μ or v_split ↝ μ′
   4. do if p(v) ∈ (−∞, qₓ] × [qᵧ, qᵧ′]
      5. then report p(v)
   6. for each node v on v_split ↝ μ
      7. do if the path goes left at v
      8. then ReportInSubtree(rc(v), qₓ)
   9. for each node v on v_split ↝ μ′
      10. do if the path goes right at v
      11. then ReportInSubtree(lc(v), qₓ)
```

ReportInSubtree(v, qₓ)

1. if (p(v))ₓ ≤ qₓ
2. then report p(v)
3. if v is not a leaf
4. then ReportInSubtree(lc(v), qₓ)
5. ReportInSubtree(rc(v), qₓ)
Example of Execution

QueryPrioSearchTree(\(T, (-\infty, q_x] \times [q_y, q'_y]\))

// r is the root of \(T\)
1. search with \(q_y\) and \(q'_y\) in \(T\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{split}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \leadsto \mu\) or \(v_{split} \leadsto \mu'\)
4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
5. then report \(p(v)\)
6. for each node \(v\) on \(v_{split} \leadsto \mu\)
7. do if the path goes left at \(v\)
8. then ReportInSubtree\(\(rc(v), q_x)\)
9. for each node \(v\) on \(v_{split} \leadsto \mu'\)
10. do if the path goes right at \(v\)
11. then ReportInSubtree\(\(lc(v), q_x)\)

ReportInSubtree\(\(v, q_x)\)

1. if \((p(v))_x \leq q_x\)
2. then report \(p(v)\)
3. if \(v\) is not a leaf
4. then ReportInSubtree\(\(lc(v), q_x)\)
5. ReportInSubtree\(\(rc(v), q_x)\)
Example of Execution

QueryPriorSearchTree(\(T, (-\infty, q_x] \times [q_y, q'_y]\))

// r is the root of T
1. search with \(q_y\) and \(q'_y\) in \(T\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{split}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \leadsto \mu\) or \(v_{split} \leadsto \mu'\)
   4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
      5. then report \(p(v)\)
6. for each node \(v\) on \(v_{split} \leadsto \mu\)
   7. do if the path goes left at \(v\)
      8. then ReportInSubtree(rc(v), q_x)
9. for each node \(v\) on \(v_{split} \leadsto \mu'\)
10. do if the path goes right at \(v\)
    11. then ReportInSubtree(lc(v), q_x)
Example of Execution

```plaintext
QueryPrioSearchTree(\(T, (−\infty, q_x] \times [q_y, q'_y]\))
// r is the root of T
1. search with \(q_y\) and \(q'_y\) in \(T\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{split}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \sim \mu\) or \(v_{split} \sim \mu'\)
4. do if \(p(v) \in (−\infty, q_x] \times [q_y, q'_y]\)
5. then report \(p(v)\)
6. for each node \(v\) on \(v_{split} \sim \mu\)
7. do if the path goes left at \(v\)
8. then ReportInSubtree(rc(v), q_x)
9. for each node \(v\) on \(v_{split} \sim \mu'\)
10. do if the path goes right at \(v\)
11. then ReportInSubtree(lc(v), q_x)
```

ReportInSubtree(\(v, q_x\))
1. if \((p(v))_x \leq q_x\)
2. then report \(p(v)\)
3. if \(v\) is not a leaf
4. then ReportInSubtree(lc(v), q_x)
5. ReportInSubtree(rc(v), q_x)
Example of Execution

QueryPrioSearchTree(\(\mathcal{T}, (-\infty, q_x] \times [q_y, q'_y]\))

// r is the root of \(\mathcal{T}\)
1. search with \(q_y\) and \(q'_y\) in \(\mathcal{T}\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{\text{split}}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \sim \mu\) or \(v_{\text{split}} \sim \mu'\)
   4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
      5. then report \(p(v)\)
6. for each node \(v\) on \(v_{\text{split}} \sim \mu\)
   7. do if the path goes left at \(v\)
      8. then ReportInSubtree(r(v), \(q_x\))
9. for each node \(v\) on \(v_{\text{split}} \sim \mu'\)
   10. do if the path goes right at \(v\)
      11. then ReportInSubtree(l(v), \(q_x\))
12. ReportInSubtree(v, \(q_x\))

// r is the root of \(\mathcal{T}\)
1. if \((p(v))_x \leq q_x\)
2. then report \(p(v)\)
3. if \(v\) is not a leaf
   4. then ReportInSubtree(rc(v), \(q_x\))
5. ReportInSubtree(lc(v), \(q_x\))
QueryPrioSearchTree($T$, $(-\infty, q_x] \times [q_y, q'_y]$)

// $r$ is the root of $T$
1. search with $q_y$ and $q'_y$ in $T$, ending at the nodes $\mu$ and $\mu'$
2. let $v_{split}$ be the node where the two paths split.
3. for each node $v$ on the path $r \sim \mu$ or $v_{split} \sim \mu'$
   4. do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$ then report $p(v)$
   5. for each node $v$ on $v_{split} \sim \mu$
      6. do if the path goes left at $v$
         7. then ReportInSubtree($rc(v)$, $q_x$)
   8. for each node $v$ on $v_{split} \sim \mu'$
      9. do if the path goes right at $v$
         10. then ReportInSubtree($lc(v)$, $q_x$)
11. ReportInSubtree($v$, $q_x$)

Example of Execution

$v_{split} = p_4$
Example of Execution

```
QueryPrioSearchTree(T, (-∞, q_x] × [q_y, q'_y])
// r is the root of T
1. search with q_y and q'_y in T, ending at the nodes μ and μ'
2. let v_split be the node where the two paths split.
3. for each node v on the path r ↝ μ or v_split ↝ μ'
   4. do if p(v) ∈ (-∞, q_x] × [q_y, q'_y]
   5. then report p(v)
4. do if the path goes left at v
   7. ReportInSubtree(rc(v), q_x)
5. if v is not a leaf
   3. then ReportInSubtree(lc(v), q_x)
6. do if the path goes right at v
9. ReportInSubtree(lc(v), q_x)
5. ReportInSubtree(rc(v), q_x)
```

Example of Execution

QueryPrioSearchTree(\(\mathcal{T}\), \((-\infty, q_x] \times [q_y, q'_y]\))

// \(r\) is the root of \(\mathcal{T}\)
1. search with \(q_y\) and \(q'_y\) in \(\mathcal{T}\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{\text{split}}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \sim \mu\) or \(v_{\text{split}} \sim \mu'\)
   4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
   5. then report \(p(v)\)
6. for each node \(v\) on \(v_{\text{split}} \sim \mu\)
7. do if the path goes left at \(v\)
8. then ReportInSubtree(\(rc(v), q_x\))
9. for each node \(v\) on \(v_{\text{split}} \sim \mu'\)
10. do if the path goes right at \(v\)
11. then ReportInSubtree(\(lc(v), q_x\))
Example of Execution

QueryPrioSearchTree(\(\mathcal{T}, (-\infty, q_x] \times [q_y, q'_y]\))

// \(r\) is the root of \(\mathcal{T}\)

1. search with \(q_y\) and \(q'_y\) in \(\mathcal{T}\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{split}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \sim \mu\) or \(v_{split} \sim \mu'\)
   4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
      5. then report \(p(v)\)
   6. for each node \(v\) on \(v_{split} \sim \mu\)
      7. do if the path goes left at \(v\)
         8. then ReportInSubtree(rc(v), \(q_x\))
   9. for each node \(v\) on \(v_{split} \sim \mu'\)
      10. do if the path goes right at \(v\)
           11. then ReportInSubtree(lc(v), \(q_x\))
Example of Execution

\[ (q_x, q'_y) \]

\[ (q_x, q_y) \]

\[ v_{\text{split}} = p_4 \]

```latex
\text{QueryPrioSearchTree}(\mathcal{T}, (-\infty, q_x] \times [q_y, q'_y])
// r is the root of \mathcal{T}
1. search with \( q_y \) and \( q'_y \) in \( \mathcal{T} \), ending at the nodes \( \mu \) and \( \mu' \)
2. let \( v_{\text{split}} \) be the node where the two paths split.
3. for each node \( v \) on the path \( r \leadsto \mu \) or \( v_{\text{split}} \leadsto \mu' \)
4. do if \( p(v) \in (-\infty, q_x] \times [q_y, q'_y] \)
5. then report \( p(v) \)
6. for each node \( v \) on \( v_{\text{split}} \leadsto \mu \)
7. do if the path goes left at \( v \)
8. then ReportInSubtree(\( rc(v), q_x \))
9. for each node \( v \) on \( v_{\text{split}} \leadsto \mu' \)
10. do if the path goes right at \( v \)
11. then ReportInSubtree(\( lc(v), q_x \))
```
Example of Execution

\[
(p_1, \ldots, p_7) \quad (q_x, q_y', q_y)
\]

Returns \(p_4, p_7, p_1\).

\[
v_{split} = p_4
\]

QueryPrioSearchTree(\(T, (-\infty, q_x] \times [q_y, q_y']\))

// \(r\) is the root of \(T\)
1. search with \(q_y\) and \(q_y'\) in \(T\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{split}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \sim \mu\) or \(v_{split} \sim \mu'\)
   4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q_y']\)
   5. then report \(p(v)\)
6. for each node \(v\) on \(v_{split} \sim \mu\)
   7. do if the path goes left at \(v\)
   8. then ReportInSubtree(\(rc(v), q_x\))
9. for each node \(v\) on \(v_{split} \sim \mu'\)
   10. do if the path goes right at \(v\)
   11. then ReportInSubtree(\(lc(v), q_x\))

ReportInSubtree(\(p_5, q_x\))
Running Time

**QueryPrioSearchTree(𝑇, (−∞, 𝑞_𝑥] × [𝑞_य, 𝑞_य'])**

// 𝑟 is the root of 𝑇
1. search with 𝑞_𝑦 and 𝑞_𝑦' in 𝑇, ending at the nodes 𝜇 and 𝜇'
2. let 𝑣_split be the node where the two paths split.
3. for each node 𝑣 on the path 𝑟 ∼ 𝜇 or 𝑣_split ∼ 𝜇'
   4. do if 𝑝(𝑣) ∈ (−∞, 𝑞_𝑥] × [𝑞_𝑦, 𝑞_𝑦']
      5. then report 𝑝(𝑣)
6. for each node 𝑣 on 𝑣_split ∼ 𝜇
   7. do if the path goes left at 𝑣
      8. then ReportInSubtree(𝑟𝑐(𝑣), 𝑞_𝑥)
9. for each node 𝑣 on 𝑣_split ∼ 𝜇'
10. do if the path goes right at 𝑣
11. then ReportInSubtree(𝑙𝑐(𝑣), 𝑞_𝑥)

---

**ReportInSubtree(𝑣, 𝑞_𝑥) // 𝑂(𝑘)**

1. if (𝑝(𝑣))_𝑥 ≤ 𝑞_𝑥
2. then report 𝑝(𝑣)
3. if 𝑣 is not a leaf
4. then ReportInSubtree(𝑙𝑐(𝑣), 𝑞_𝑥)
5. ReportInSubtree(𝑟𝑐(𝑣), 𝑞_𝑥)

Time cost breaks down to two parts:
Running Time

QueryPrioSearchTree($T$, $(-\infty, q_x] \times [q_y, q'_y]$)

// $r$ is the root of $T$
1. search with $q_y$ and $q'_y$ in $T$, ending at the nodes $\mu$ and $\mu'$
2. let $v_{split}$ be the node where the two paths split.
3. for each node $v$ on the path $r \leadsto \mu$ or $v_{split} \leadsto \mu'$
4. \hspace{5pt} do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$ \hspace{5pt} then report $p(v)$
5. \hspace{5pt} for each node $v$ on $v_{split} \leadsto \mu$
6. \hspace{10pt} do if the path goes left at $v$
7. \hspace{15pt} then ReportInSubtree($rc(v)$, $q_x$)
8. \hspace{5pt} for each node $v$ on $v_{split} \leadsto \mu'$
9. \hspace{10pt} do if the path goes right at $v$
10. \hspace{15pt} then ReportInSubtree($lc(v)$, $q_x$)

ReportInSubtree($v, q_x$) \hspace{5pt} // $O(k)$

1. \hspace{5pt} if $(p(v))_x \leq q_x$
2. \hspace{10pt} then report $p(v)$
3. \hspace{5pt} if $v$ is not a leaf
4. \hspace{10pt} then ReportInSubtree($lc(v)$, $q_x$)
5. \hspace{10pt} ReportInSubtree($rc(v)$, $q_x$)

Time cost breaks down to two parts:

- number of nodes on the path $r \leadsto \mu$ or $v_{split} \leadsto \mu'$
Running Time

QueryPrioSearchTree($T$, $(-\infty, q_x] \times [q_y, q'_y]$)

// $r$ is the root of $T$
1. search with $q_y$ and $q'_y$ in $T$, ending at the nodes $\mu$ and $\mu'$
2. let $v_{split}$ be the node where the two paths split.
3. for each node $v$ on the path $r \sim \mu$ or $v_{split} \sim \mu'$
   4. do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$ then report $p(v)$
5. for each node $v$ on $v_{split} \sim \mu$
   6. do if the path goes left at $v$
5. then ReportInSubtree($rc(v)$, $q_x$)
9. for each node $v$ on $v_{split} \sim \mu'$
10. do if the path goes right at $v$
11. then ReportInSubtree($lc(v)$, $q_x$)

ReportInSubtree($v, q_x$) // $O(k)$

1. if $(p(v))_x \leq q_x$
2. then report $p(v)$
3. if $v$ is not a leaf
4. then ReportInSubtree($lc(v)$, $q_x$)
5. ReportInSubtree($rc(v)$, $q_x$)

Time cost breaks down to two parts:

- number of nodes on the path $r \sim \mu$ or $v_{split} \sim \mu'$ $O(\log n)$
Running Time

QueryPrioSearchTree(\(\mathcal{T}, (-\infty, q_x] \times [q_y, q'_y]\))

// \(r\) is the root of \(\mathcal{T}\)
1. search with \(q_y\) and \(q'_y\) in \(\mathcal{T}\), ending at the nodes \(\mu\) and \(\mu'\)
2. let \(v_{\text{split}}\) be the node where the two paths split.
3. for each node \(v\) on the path \(r \sim \mu\) or \(v_{\text{split}} \sim \mu'\)
   4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
      5. then report \(p(v)\)
6. for each node \(v\) on \(v_{\text{split}} \sim \mu\)
   7. do if the path goes left at \(v\)
      8. then ReportInSubtree(\(rc(v), q_x\))
9. for each node \(v\) on \(v_{\text{split}} \sim \mu'\)
10. do if the path goes right at \(v\)
11. then ReportInSubtree(\(lc(v), q_x\))

Time cost breaks down to two parts:
- number of nodes on the path \(r \sim \mu\) or \(v_{\text{split}} \sim \mu'\) \(O(\log n)\)
- number of recursive calls to ReportInSubtree().
Running Time

QueryPrioSearchTree($T$, $(-\infty, q_x] \times [q_y, q'_y]$)

// $r$ is the root of $T$
1. search with $q_y$ and $q'_y$ in $T$, ending at the nodes $\mu$ and $\mu'$
2. let $v_{split}$ be the node where the two paths split.
3. for each node $v$ on the path $r \sim \mu$ or $v_{split} \sim \mu$
   4. do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$
      then report $p(v)$
   5. for each node $v$ on $v_{split} \sim \mu$
      do if the path goes left at $v$
      then ReportInSubtree($rc(v)$, $q_x$)
   9. for each node $v$ on $v_{split} \sim \mu'$
      do if the path goes right at $v$
      then ReportInSubtree($lc(v)$, $q_x$)

ReportInSubtree($v, q_x$) // $O(k)$

1. if $(p(v))_x \leq q_x$
2. then report $p(v)$
3. if $v$ is not a leaf
4. then ReportInSubtree($lc(v)$, $q_x$)
5. ReportInSubtree($rc(v)$, $q_x$)

Time cost breaks down to two parts:

- number of nodes on the path $r \sim \mu$ or $v_{split} \sim \mu'$ $O(\log n)$
- number of recursive calls to ReportInSubtree(). $O(k)$

# reported points
Running Time

```
QueryPrioSearchTree(𝒯, (−∞, qx] × [qy, qy′])
// r is the root of 𝒯
1. search with qy and qy′ in 𝒯, ending at the nodes μ and μ′
2. let 𝑣split be the node where the two paths split.
3. for each node 𝑣 on the path r ∼ μ or 𝑣split ∼ μ′
   4. do if p(𝑣) ∈ (−∞, qx] × [qy, qy′]
      then report p(𝑣)
   5. for each node 𝑣 on 𝑣split ∼ μ
      do if the path goes left at 𝑣
      then ReportInSubtree(rc(𝑣), qx)
   6. for each node 𝑣 on 𝑣split ∼ μ′
      do if the path goes right at 𝑣
      then ReportInSubtree(lc(𝑣), qx)
```

Time cost breaks down to two parts:

- number of nodes on the path r ∼ μ or 𝑣split ∼ μ′  \( O(\log n) \)
- number of recursive calls to ReportInSubtree()  \( O(k) \)

\( O(\log n + k) \) # reported points
Summary on PST

- Min heap over the $x$-coordinate.
- Binary search tree over the $y$-coordinate.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Construction time</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Query time</td>
<td>$O(\log n + k)$</td>
</tr>
</tbody>
</table>