Priority Search Trees

Outline:

I. Heap-based point queries

II. Structure of a PST

III. Query

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

Range tree on left endpoints
\( T_{left}(v) \)

Range tree on right endpoints
\( T_{right}(v) \)

\( I_{left} \)

\( I_{right} \)

Region \( R \)

\( [\infty, q_x] \times [q_y, q'_y] \)

Query \( q \)

\( (q_x, q_y) \)

Region \( R' \)

\( [q_x, \infty] \times [q_y, q'_y] \)

Point \( (q_x, q'_y) \)

Point \( (q_x, q_y) \)

Point \( x_{mid} \)
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:

- Range tree on left endpoints $T_{left}(v)$
- Range tree on right endpoints $T_{right}(v)$
- $l_{left}$
- $I_{right}$

Diagram:

- $R$: $\left[ -\infty, q_x \right] \times [q_y, q_y']$
- $q$: $(q_x, q_y)$
- $R'$: $[q_x, \infty] \times [q_y, q_y']$
- $x_{mid}$
- $(q_x, q_y')$
- $(q_x, q_y)$
Windowing Queries with an Interval Tree

- High storage: $O(n \log n)$
- Complicated data structure due to the use 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

- 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 $O(n)$. 
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: $P = \{p_1, p_2, ..., p_n\}$

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

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

Range: $(-\infty, q_x]$
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 $p_1 < p_2 < \cdots < p_n$. 

```
p_1  p_2  p_k  q_x  p_{k+1}  p_n
```
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: \( P = \{p_1, p_2, ..., p_n\} \)

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 \( p_{k+1} > q_x \).
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.
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.

{1, 3, 11, 6, 9, 13, 22, 8, 40}
First Query Range Handled by a Heap

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

Diagram: 

- Node $p$ with pointers to $L$ and $R$.
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 \( R \).
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 \( T(p) \) rooted at \( p \).
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 \( p \).

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

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

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

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

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

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

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

\((-\infty, 12]\) 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: $P = \{p_1, p_2, ..., p_n\}$

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

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

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.
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?
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?

- We can choose a heap that partitions the set according to the $y$-coordinate.
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,
  - $p_{\text{min}} \in P$ with the smallest $x$-coordinate.
  - $y_{\text{mid}}$: median $y$-coordinate of the remaining points.
  - $P_{\text{below}} = \{ p \in P \setminus \{ p_{\text{min}} \} \mid p_y \leq y_{\text{mid}} \}$. 
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,
  - $p_{min} \in P$ 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 $v(p_{min}, y_{mid})$.
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.
  - $P_{\text{below}} = \{p \in P \setminus \{p_{\text{min}}\} \mid p_y \leq y_{\text{mid}}\}$.
  - $P_{\text{above}} = \{p \in P \setminus \{p_{\text{min}}\} \mid p_y > y_{\text{mid}}\}$.
  - Create $v$ with $p_{\text{min}}, y_{\text{mid}}$
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,
  - $p_{\text{min}} \in P$ with the smallest $x$-coordinate.
  - $y_{\text{mid}}$: median $y$-coordinate of the remaining points.
  - $P_{\text{below}} = \{ p \in P \setminus \{ p_{\text{min}} \} \mid p_y \leq y_{\text{mid}} \}$.
  - $P_{\text{above}} = \{ p \in P \setminus \{ p_{\text{min}} \} \mid p_y > y_{\text{mid}} \}$.
  - Create

\[
\begin{align*}
&v \quad p_{\text{min}}, \\
&P_{\text{below}} \\
&P_{\text{above}}
\end{align*}
\]
Construction Time

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

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

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

\[ p_{\text{below}} \quad p_{\text{above}} \]

\[ O(n \log n) \] 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 \(y_{mid}\) 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.
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.
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.
Check every node $v$ on either of the paths $v_{\text{split}} \sim \mu$ or $v_{\text{split}} \sim \mu'$ to see if

$$p(v) \in (-\infty, q_x] \times [q_y, q_y']$$
Search in the Selected Subtrees

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.

ReportInSubtree($v, q_x$)

1. If $v$ is not a leaf and $(p(v))_x \leq q_x$
2. then report $p(v)$
3. ReportInSubtree($lc(v), q_x$)
4. ReportInSubtree($rc(v), q_x$)
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 $q_x$. 
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 $μ$ in $𝑇(𝑣)$ such that its stored point $p(μ)$ satisfies $(p(μ))_x ≤ 𝑞_x$. 

$μ$
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 $\mu \rightsquigarrow v$ 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($v, q_x$) 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.

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

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

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

\(p(\mu)\) is reported.
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 \(\mu \sim v\) 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 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.
Query Algorithm

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

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 \( v_{split} \sim \mu \) 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

QueryPrioSearchTree(T, (−∞, q_x] × [q_y, q'_y])

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 v_split ∼ 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)
Example of Execution

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

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 $v_{split} \sim \mu$ or $v_{split} \sim \mu'$
   4. do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$ then report $p(v)$
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($lc(v), q_x$)

$v_{split} = p_4$

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

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 $v_{\text{split}} \sim \mu$ 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($lc(v)$, $q_x$)

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

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 $v_{split} \sim \mu$ 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. do if the path goes right at $v$
         9. then ReportInSubtree($lc(v)$, $q_x$)

$v_{split} = p_4$

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

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 $v_{split} \sim \mu$ 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($lc(v)$, $q_x$)

$v_{split} = p_4$

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**($T, (-∞, q_x] \times [q_y, q'_y]$)

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 $v_{split} \sim \mu$ or $v_{split} \sim \mu'$
   4. do if $p(v) \in (-∞, q_x] \times [q_y, q'_y]$
   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($v$), $q_x$)

$v_{split} = p_4$

**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**(rc($v$), $q_x$)
5. **ReportInSubtree**(rc($v$), $q_x$)
Example of Execution

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

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 $v_{split} \sim \mu$ 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($v$), $q_x$)

$v_{split} = p_4$

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

1. search with $q_y$ and $q'_y$ in $T$, ending at the nodes $\mu$ and $\mu'$
2. let $\nu_{\text{split}}$ be the node where the two paths split.
3. for each node $v$ on the path $\nu_{\text{split}} \sim \nu$ or $\nu_{\text{split}} \sim \mu'$
   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 $\nu_{\text{split}} \sim \mu$
   7. do if the path goes left at $v$
   8. then ReportInSubtree($rc(v), q_x$)
9. for each node $v$ on $\nu_{\text{split}} \sim \mu'$
   10. do if the path goes right at $v$
   11. then ReportInSubtree($lc(v), q_x$)

$v_{\text{split}} = p_4$

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

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 $v_{split} \sim \mu$ 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($lc(v), q_x$)

$v_{split} = p_4$

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

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 $v_{split} \sim \mu$ or $v_{split} \sim \mu'$
   
   4. do if $p(v) \in \langle -\infty, q_x \rangle \times [q_y, q'_y]$ then report $p(v)$
5. 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($lc(v)$, $q_x$)

$v_{split} = p_4$

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

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 $v_{\text{split}} \sim \mu$ 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($lc(v), q_x$)

$v_{\text{split}} = p_4$

ReportInSubtree($p_5, q_x$)

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

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

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 $v_{split} \sim \mu$ 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 $v$
          then ReportInSubtree($p(v)$, $q_x$)
   9. for each node $v$ on $v_{split} \sim \mu'$
      10. do if the path goes right at $v$
          then ReportInSubtree($p(v)$, $q_x$)

ReportInSubtree($p_5, q_x$)

1. if $(p(v))_x \leq q_x$
   2. then report $p(v)$
3. if $v$ is not a leaf
   4. then ReportInSubtree($p(v)$, $q_x$)
5. ReportInSubtree($p_5$, $q_x$)

$v_{split} = p_4$
QueryPrioSearchTree($T, (-\infty, q_x] \times [q_y, q'_y]$)

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 $v_{split} \sim \mu$ 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. do if the path goes right at $v$
            9. then ReportInSubtree($lc(v), q_x$)

Example of Execution
Example of Execution

QueryPrioSearchTree(T, (-∞, q_x] × [q_y, q'_y])

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 v_{split} ∼ 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(p_5, q_x)

Returns p_4, p_7, p_1.

ν_{split} = p_4

ReportInSubtree(v, q_x)

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)
Running Time

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

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 \(v_{\text{split}} \sim \mu\) 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:
Running Time

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

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 \( v_{\text{split}} \sim \mu \) 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 \( \text{ReportInSubtree}(rc(v), q_x) \)
9. for each node \( v \) on \( v_{\text{split}} \sim \mu' \)
10. do if the path goes right at \( v \)
11. then \( \text{ReportInSubtree}(lc(v), q_x) \)

Time cost breaks down to two parts:

- number of nodes on the path \( v_{\text{split}} \sim \mu \) or \( v_{\text{split}} \sim \mu' \)
Running Time

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

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 \(v_{split} \leadsto \mu\) or \(v_{split} \leadsto \mu'\)
   4. do if \(p(v) \in (-\infty, q_x] \times [q_y, q'_y]\)
      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)

Time cost breaks down to two parts:

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

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

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 $v_{split} \rightarrow \mu$ or $v_{split} \rightarrow \mu'$
4. \hspace{1em} do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$
5. \hspace{2em} then report $p(v)$
6. for each node $v$ on $v_{split} \rightarrow \mu$
7. \hspace{1em} do if the path goes left at $v$
8. \hspace{2em} then $\text{ReportInSubtree}(rc(v), q_x)$
9. for each node $v$ on $v_{split} \rightarrow \mu'$
10. \hspace{1em} do if the path goes right at $v$
11. \hspace{2em} then $\text{ReportInSubtree}(lc(v), q_x)$

Time cost breaks down to two parts:

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

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

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 \(v_{\text{split}} \sim \mu\) 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 \(v_{\text{split}} \sim \mu\) or \(v_{\text{split}} \sim \mu'\) \(O(\log n)\)
- number of recursive calls to ReportInSubtree() \(O(k)\)

\# reported points
Running Time

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

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 \( v_{\text{split}} \sim \mu \) or \( v_{\text{split}} \sim \mu' \)
4. \hspace{1em} do if \( p(v) \in (-\infty, q_x) \times [q_y, q'_y] \)
5. \hspace{2em} then report \( p(v) \)
6. for each node \( v \) on \( v_{\text{split}} \sim \mu \)
7. \hspace{1em} do if the path goes left at \( v \)
8. \hspace{2em} then \text{ReportInSubtree}(rc(v), q_x)
9. for each node \( v \) on \( v_{\text{split}} \sim \mu' \)
10. \hspace{1em} do if the path goes right at \( v \)
11. \hspace{2em} then \text{ReportInSubtree}(lc(v), q_x)

Time cost breaks down to two parts:

- number of nodes on the path \( v_{\text{split}} \sim \mu \) or \( v_{\text{split}} \sim \mu' \) \( O(\log n) \)
- number of recursive calls to \text{ReportInSubtree()} \( O(k) \)
- # reported points
# Summary on PST

<table>
<thead>
<tr>
<th></th>
<th>Storage</th>
<th>Construction time</th>
<th>Query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min heap over the $x$-coordinate.</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
<td>$O(\log n + k)$</td>
</tr>
<tr>
<td>Binary search tree over the $y$-coordinate.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>