Yan-Bin Jia, Iowa State University

Priority Search Trees

Outline:

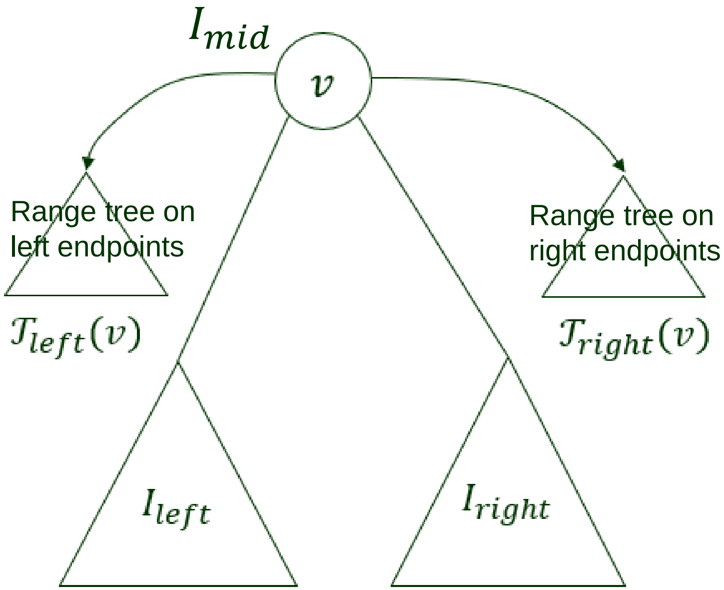
I. Heap-based point queries

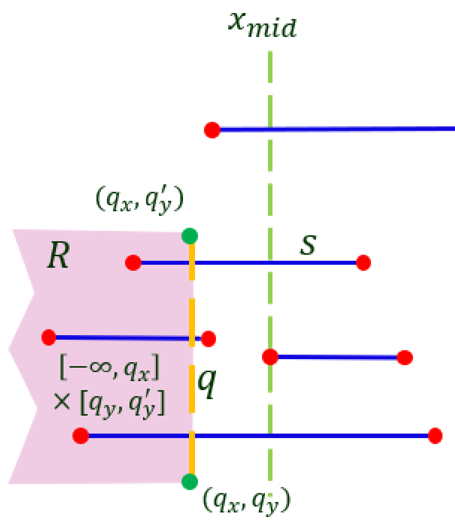
II. Structure of a PST

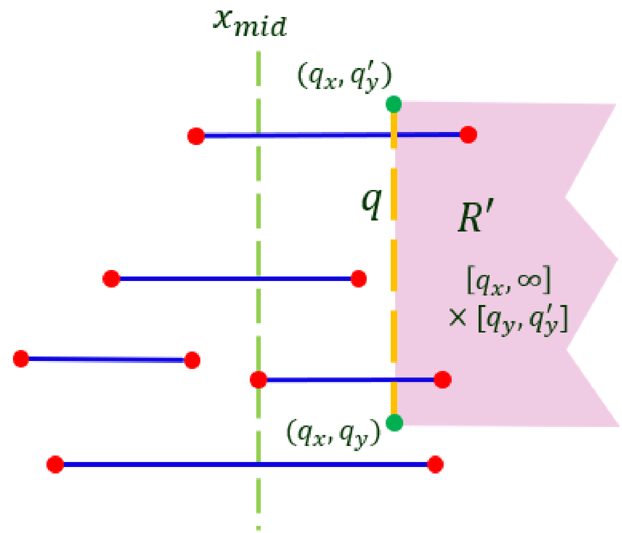
III. Query

IV. Correctness and running time

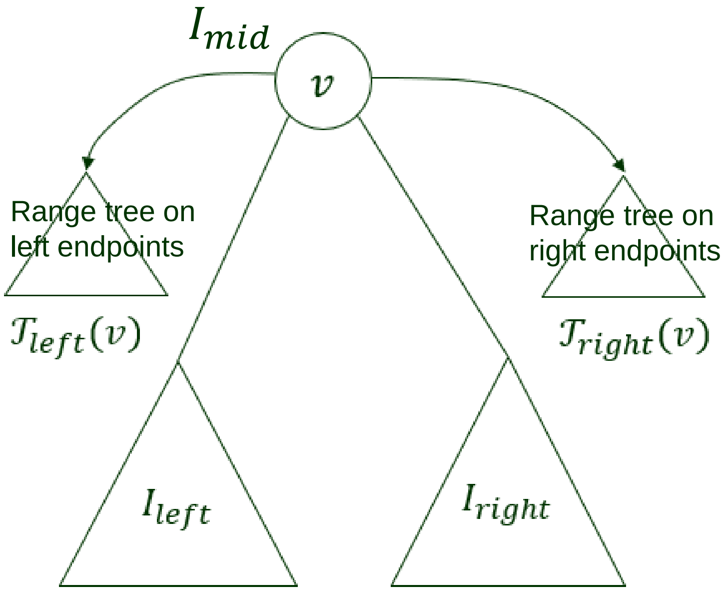
Windowing Queries with an Interval Tree

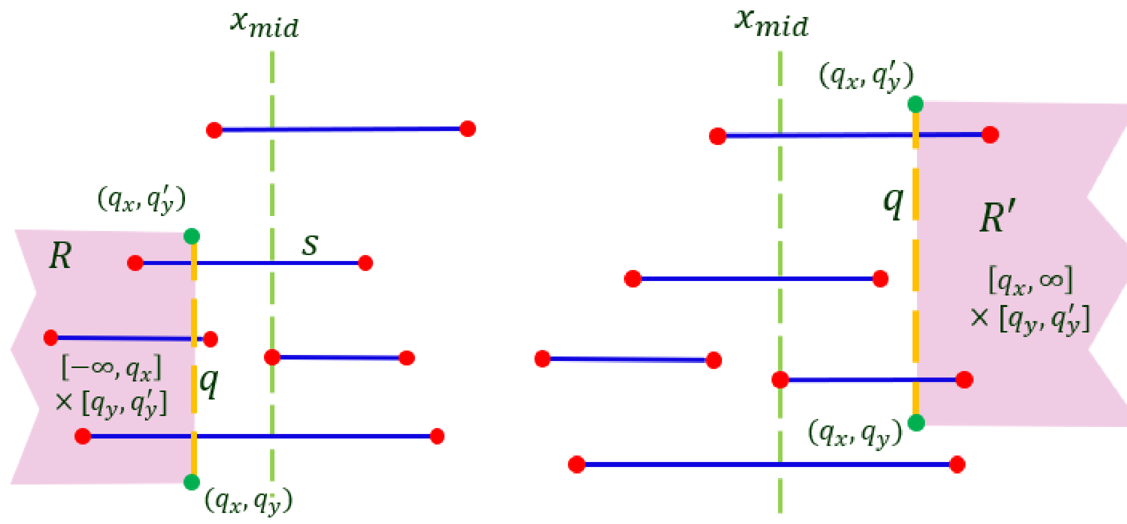






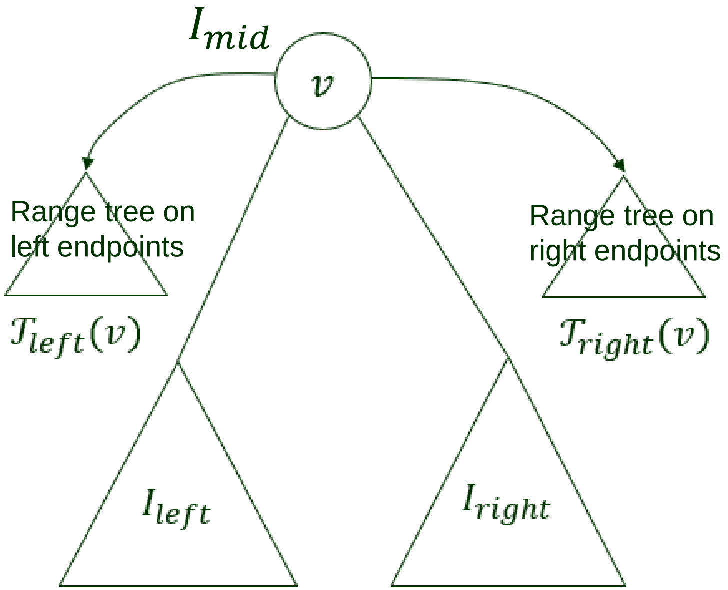
Windowing Queries with an Interval Tree

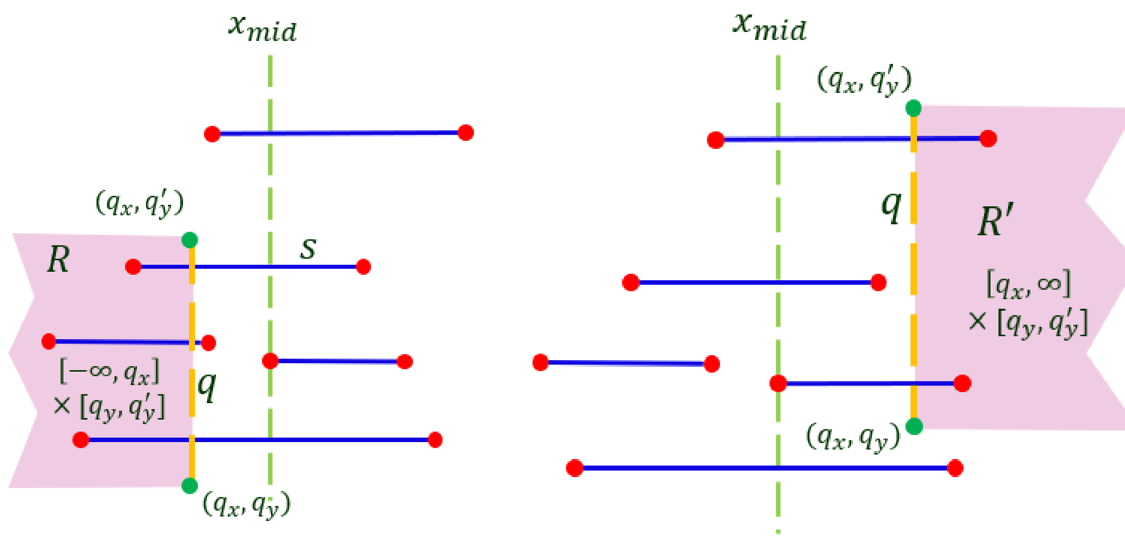




♣ Complicated data structure due to the uses of range tree and fractional cascading for efficiency.

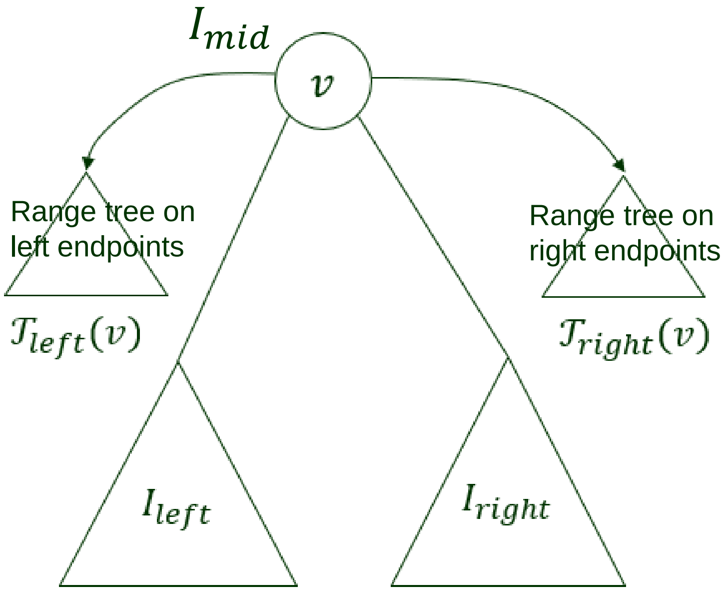
Windowing Queries with an Interval Tree

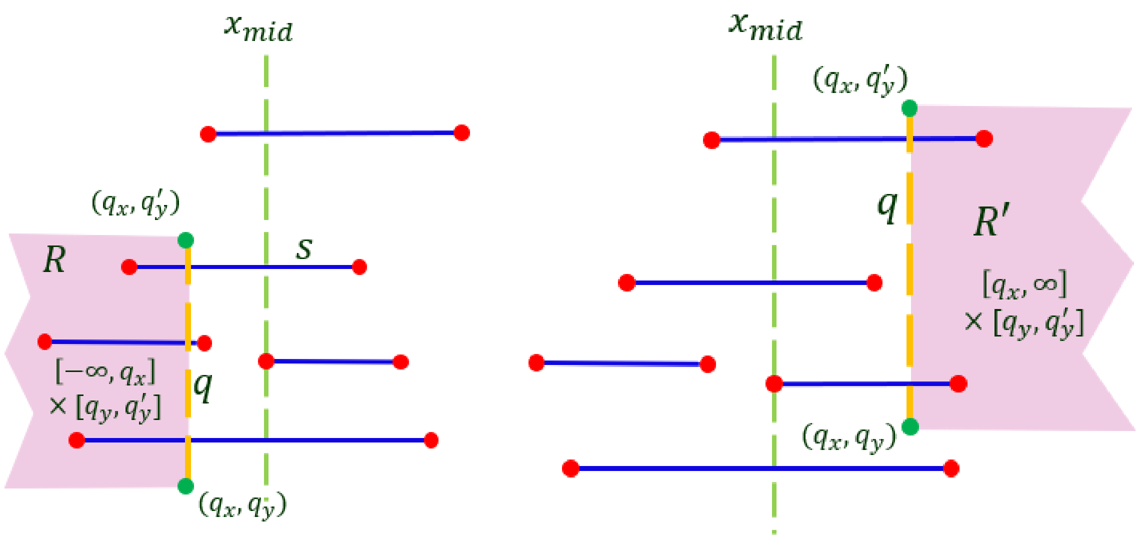




- ♣ Complicated data structure due to the uses of range tree and fractional cascading for efficiency.
- ♣ High storage: $O(n \log n)$

Windowing Queries with an Interval Tree

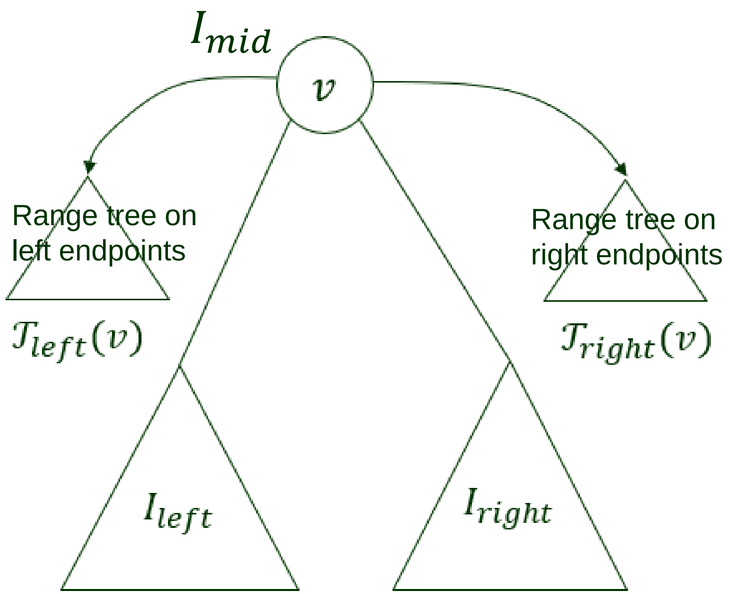


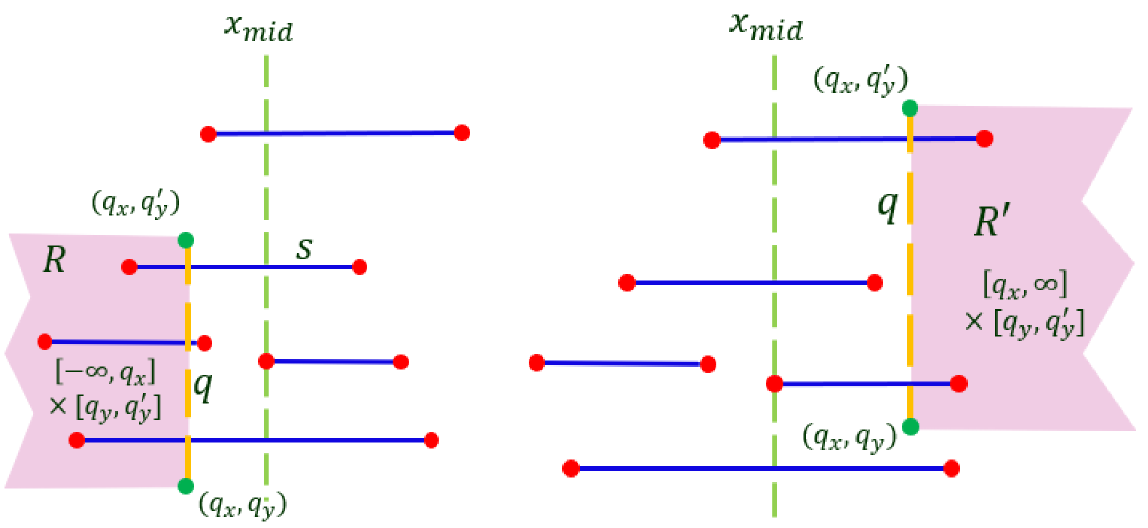


- ♣ Complicated data structure due to the uses of range tree and fractional cascading for efficiency.
- ♣ High storage: $O(n \log n)$

Improvement?

Windowing Queries with an Interval Tree



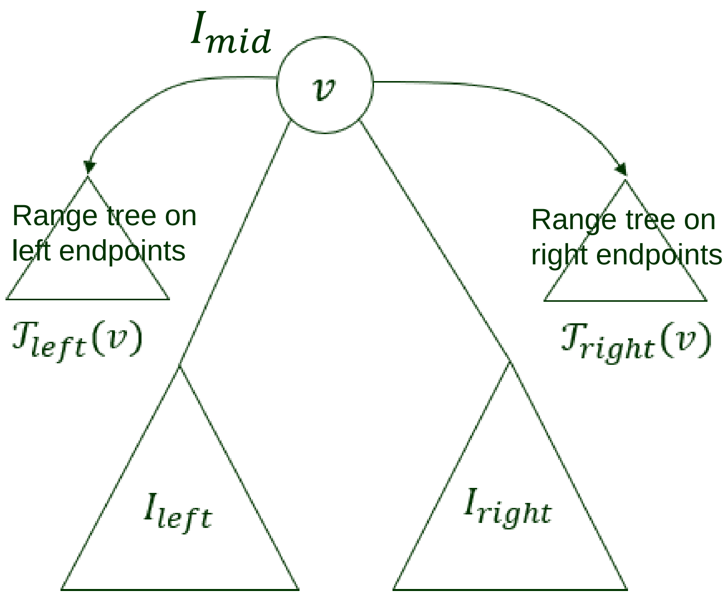


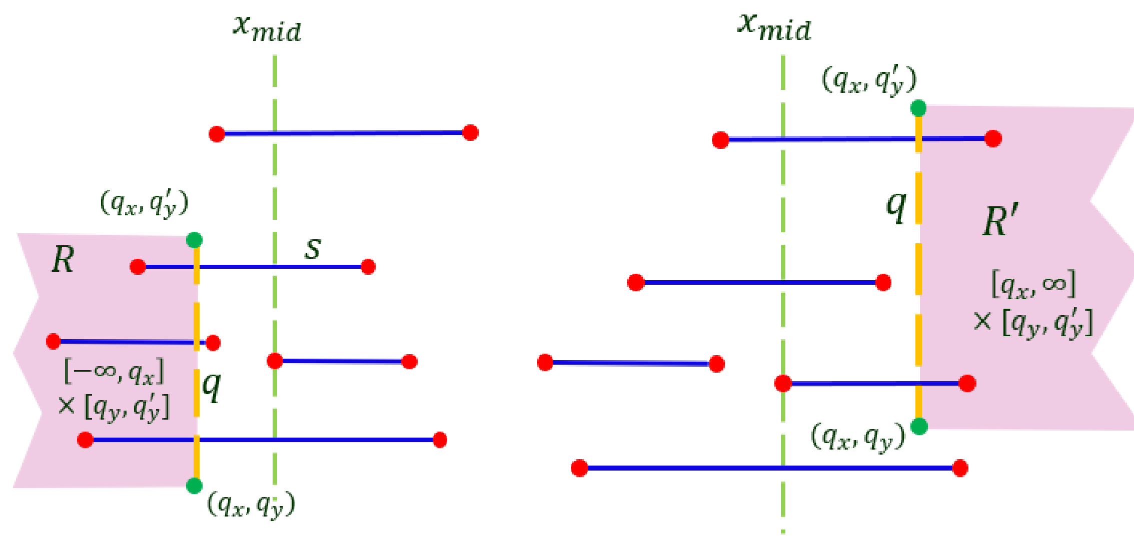
- ♣ Complicated data structure due to the uses of range tree and fractional cascading for efficiency.
- ♣ High storage: $O(n \log n)$

Improvement?

- ♦ Explore that the query range is unbounded on *one side* ($-\infty$ or ∞ over x).

Windowing Queries with an Interval Tree



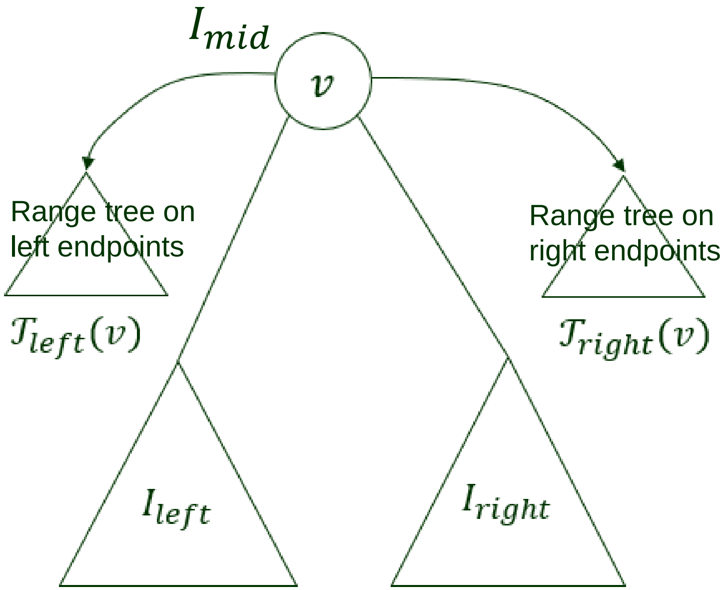


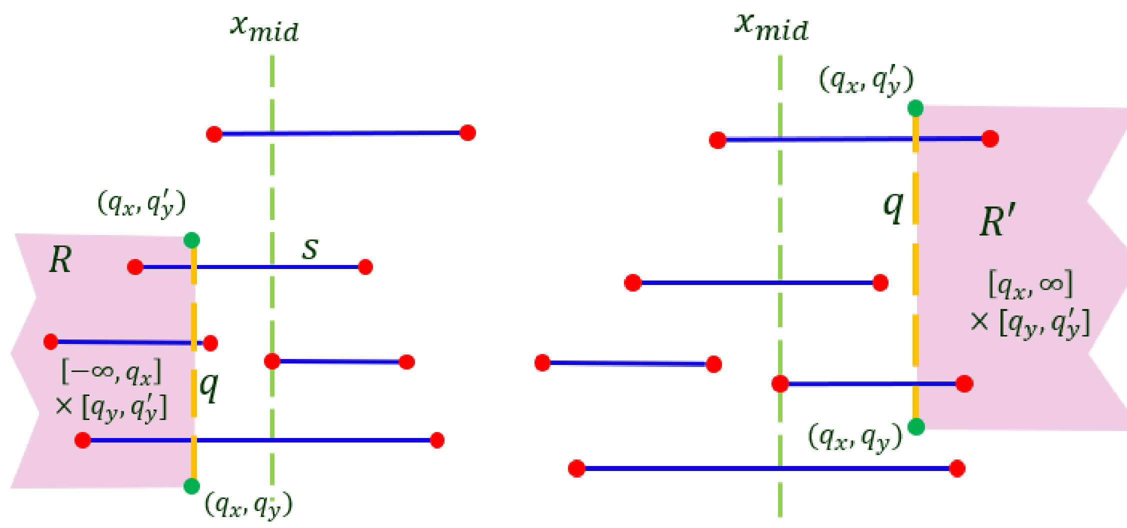
- ♣ Complicated data structure due to the uses of range tree and fractional cascading for efficiency.
- ♣ High storage: $O(n \log n)$

Improvement?

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

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

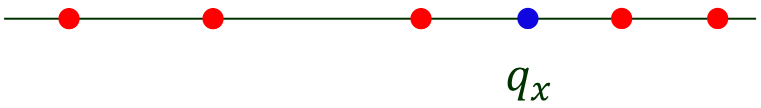
I. Query Problem

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

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

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

Range: $(-\infty, q_x]$



I. Query Problem

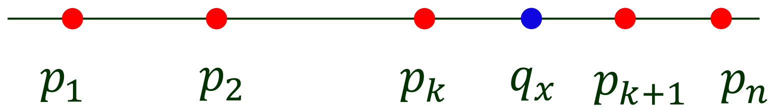
Point set: $P = \{p_1, p_2, \dots, p_n\}$

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 $p_1 < p_2 < \cdots < p_n$.



I. Query Problem

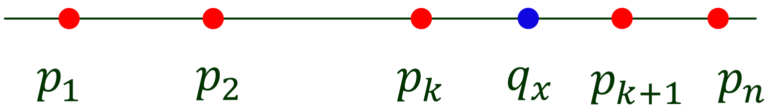
Point set: $P = \{p_1, p_2, \dots, p_n\}$

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

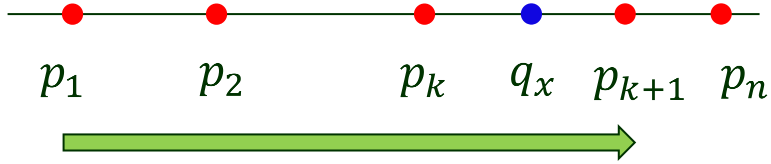
Point set: $P = \{p_1, p_2, \dots, p_n\}$

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

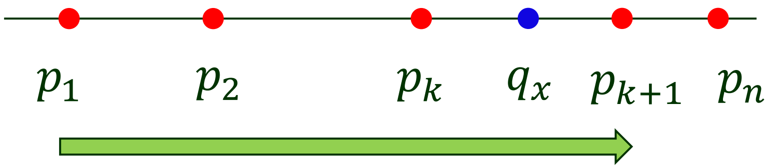
Point set: $P = \{p_1, p_2, \dots, p_n\}$

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 $p_1, p_2, \dots, p_k$ during the walk



I. Query Problem

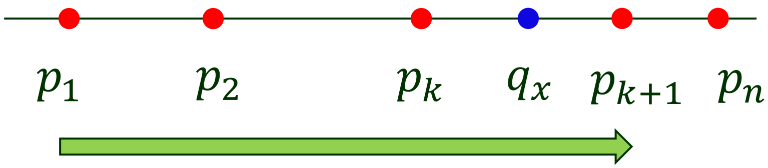
Point set: $P = \{p_1, p_2, \dots, p_n\}$

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 $p_1, p_2, \dots, p_k$ 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 $[q_y, q'_y]$.

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.

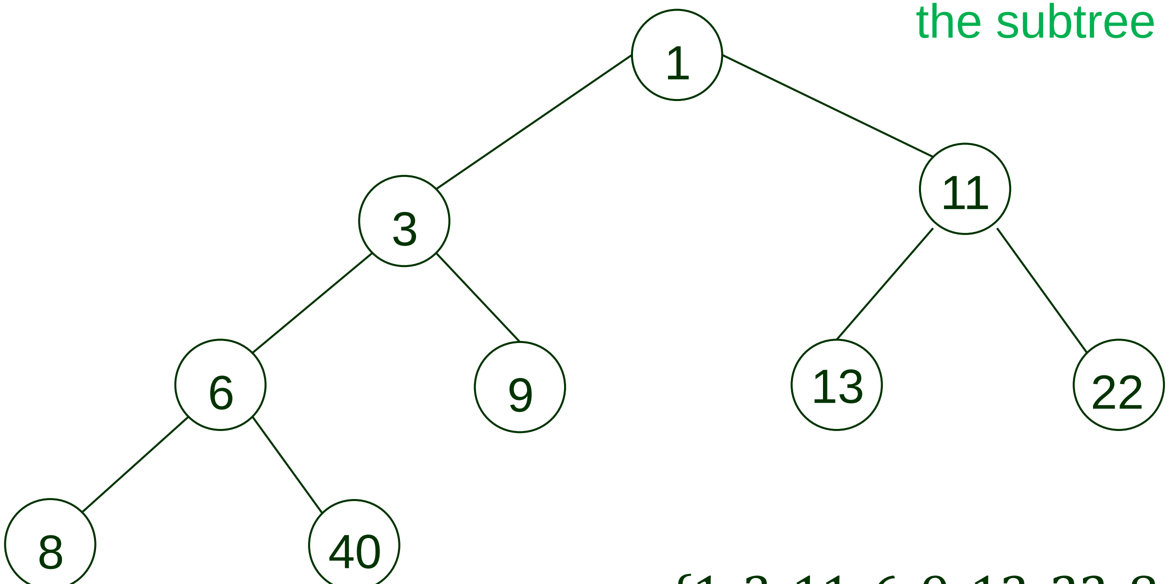
Moving on to 2D Query

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.

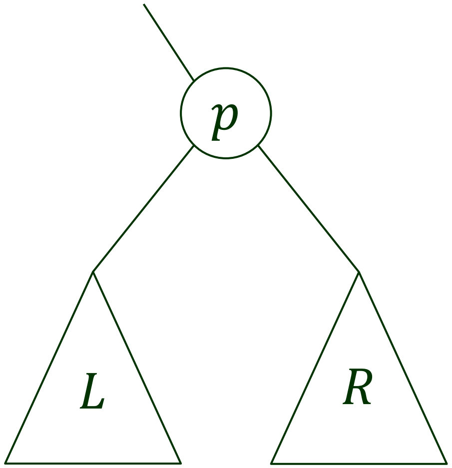


$\{1, 3, 11, 6, 9, 13, 22, 8, 40\}$

First Query Range Handled by a Heap

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

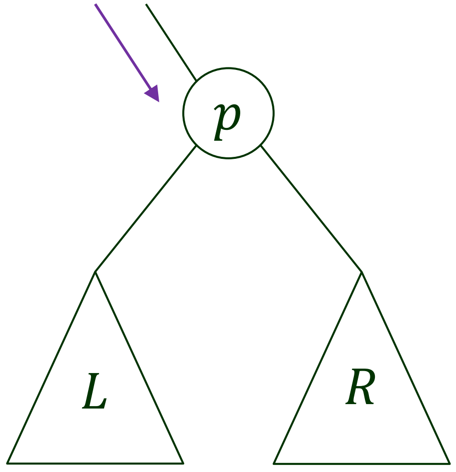
Query range: $(-\infty, q_x]$



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.

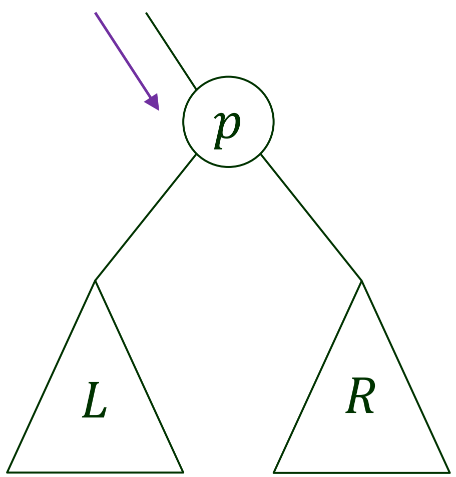


First Query Range Handled by a Heap

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

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

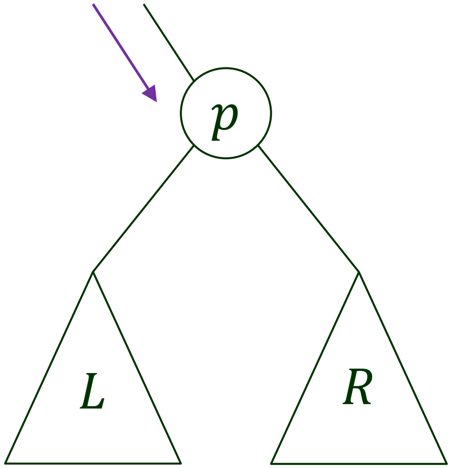
- Walk down the tree.
- At each node with stored point $p = (p_x, p_y)$ check if $p_x \leq q_x$.



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 $p_x \leq q_x$.



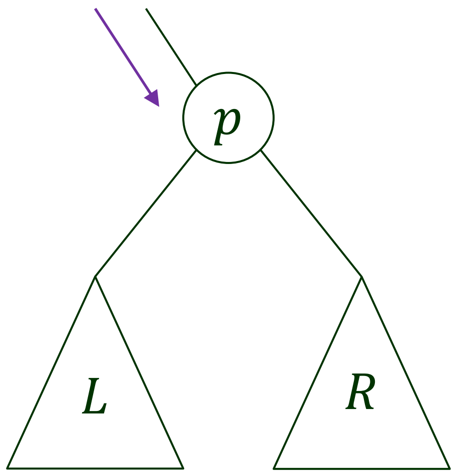
- If yes, report p and continue in both subtrees L and R.

First Query Range Handled by a Heap

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

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

- Walk down the tree.
- At each node with stored point $p = (p_x, p_y)$ check if $p_x \le q_x$.



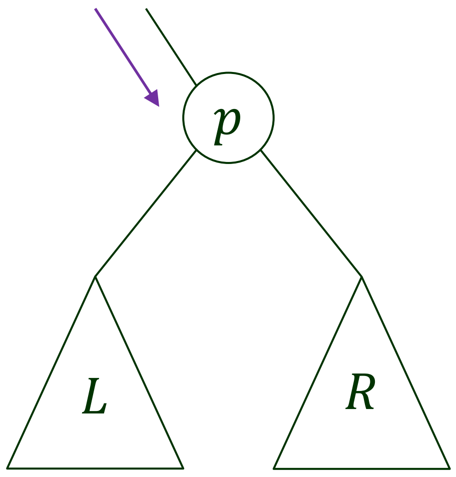
- ♦ If yes, report p and continue in both subtrees L and R.
- ♦ Otherwise ($p_x > q_x$), abort the subtree $\mathcal{T}(p)$ rooted at p.

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 $p_x \le q_x$.



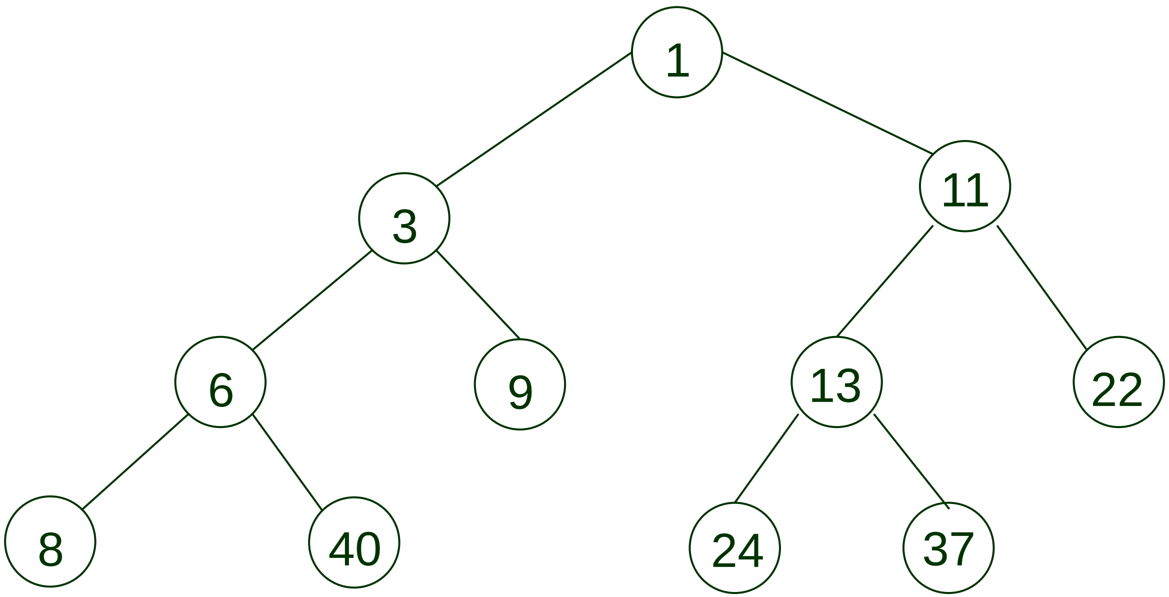
- ♦ If yes, report p and continue in both subtrees L and R.
- ♦ Otherwise ($p_x > q_x$), abort the subtree $\mathcal{T}(p)$ rooted at p.

Every node $r = (r_x, r_y) \neq p$ in $\mathcal{T}(p)$ satisfies

$$r_x \ge p_x > q_x$$

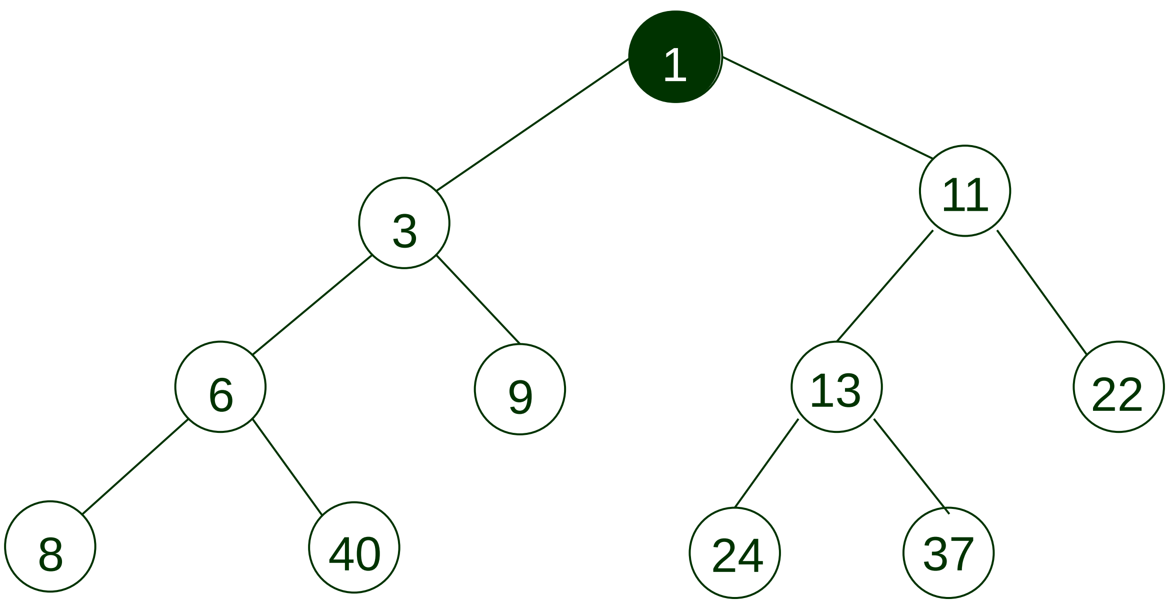
Example

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



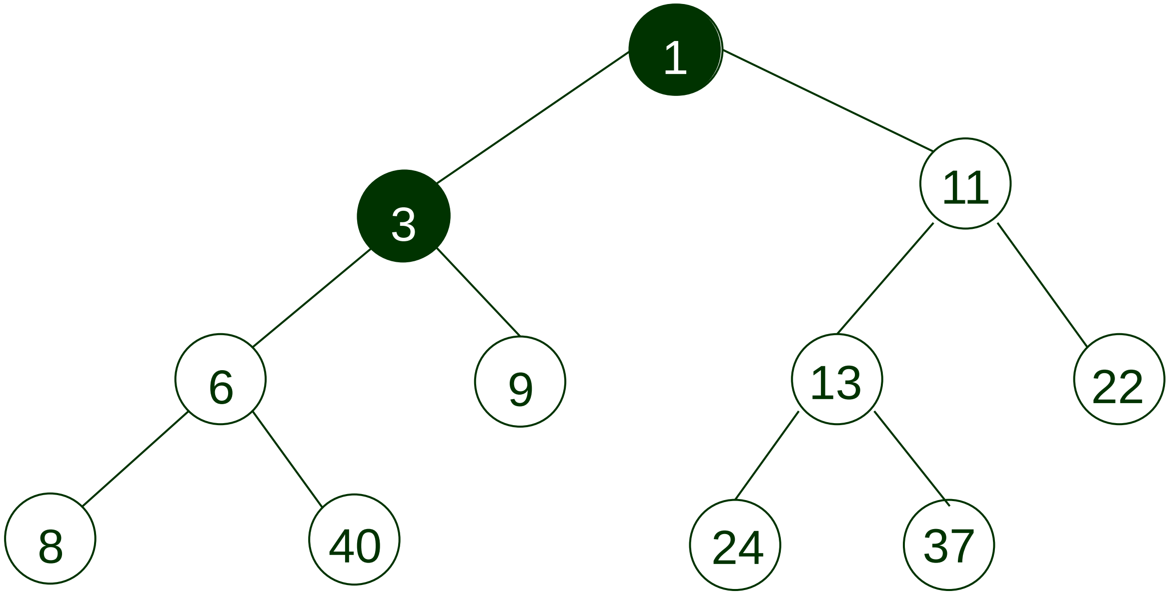
Example

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



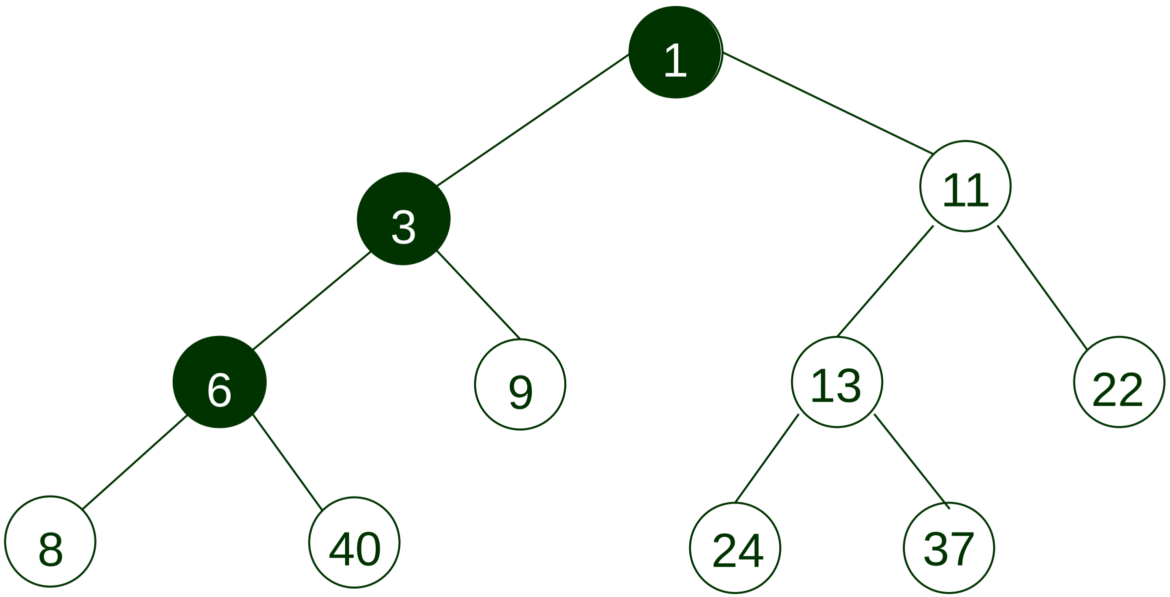
Example

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



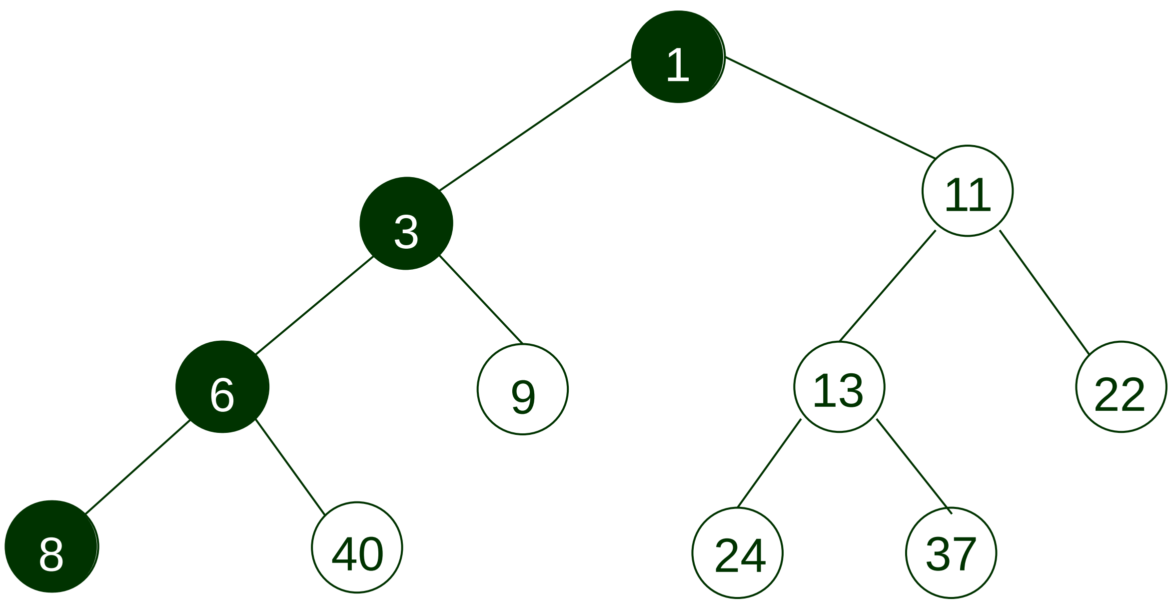
Example

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



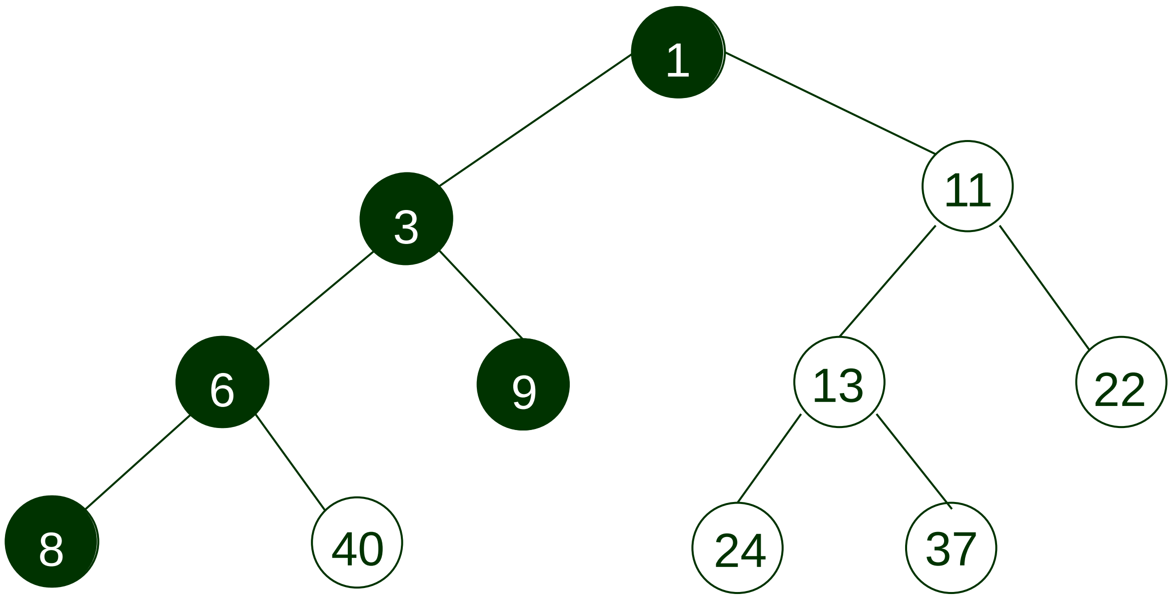
Example

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



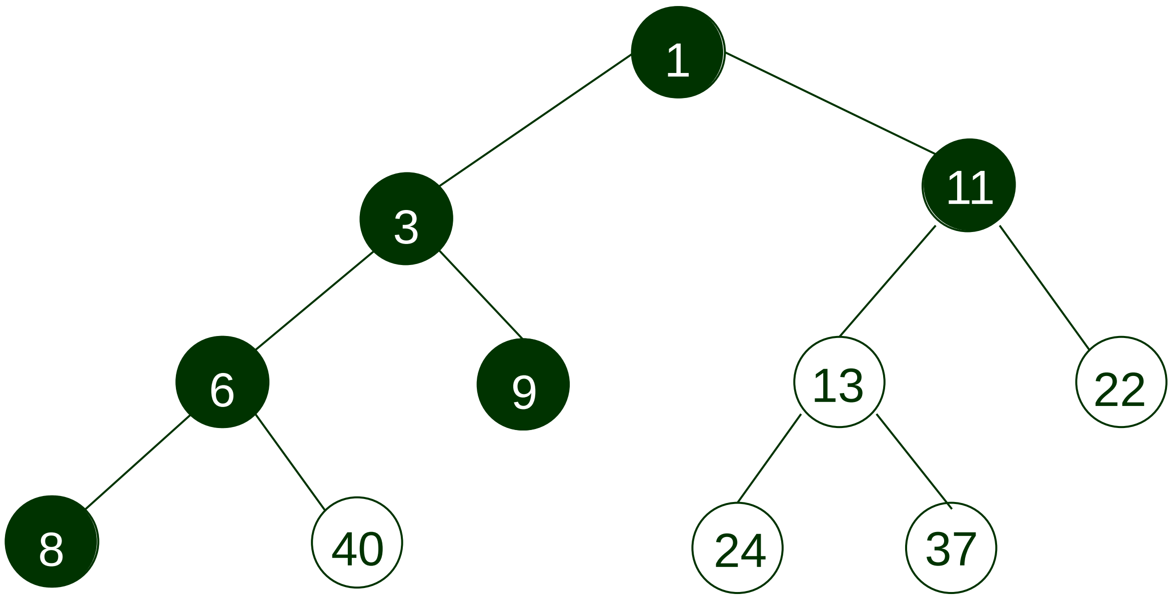
Example

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



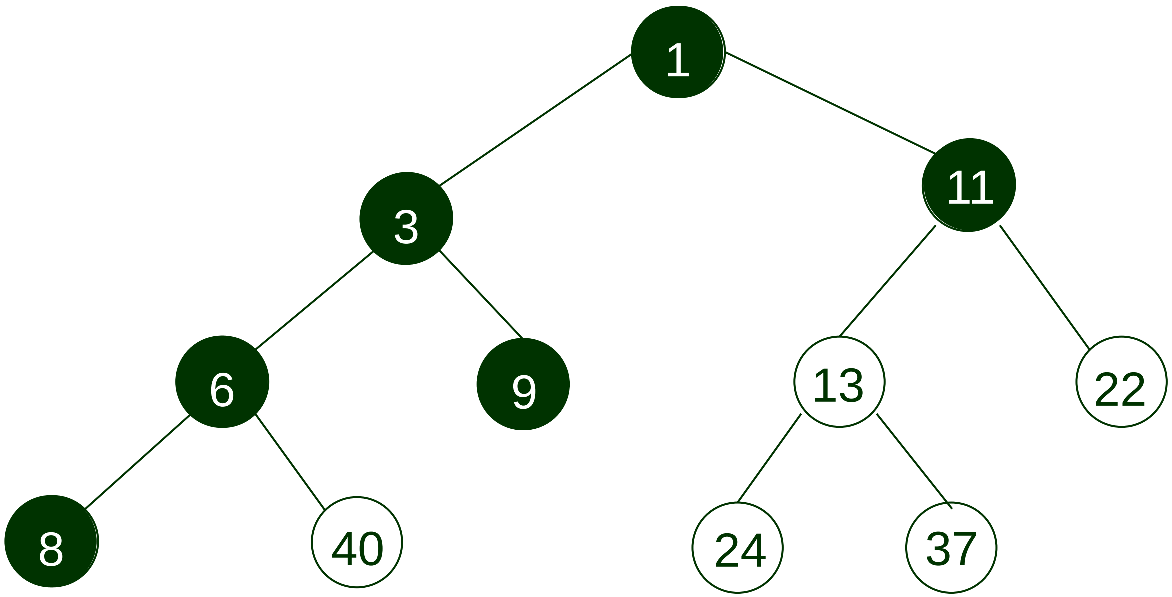
Example

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

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

Both Query Ranges Handled by a Heap

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

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

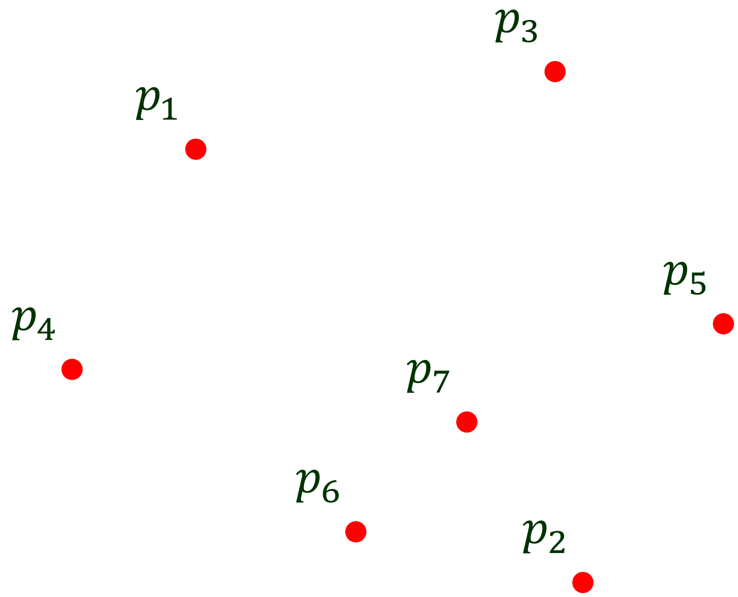
- A set can be represented by many heaps, each representing a way of partitioning the set.

♣ How to integrate the information about the y-coordinate without using the associated structures?

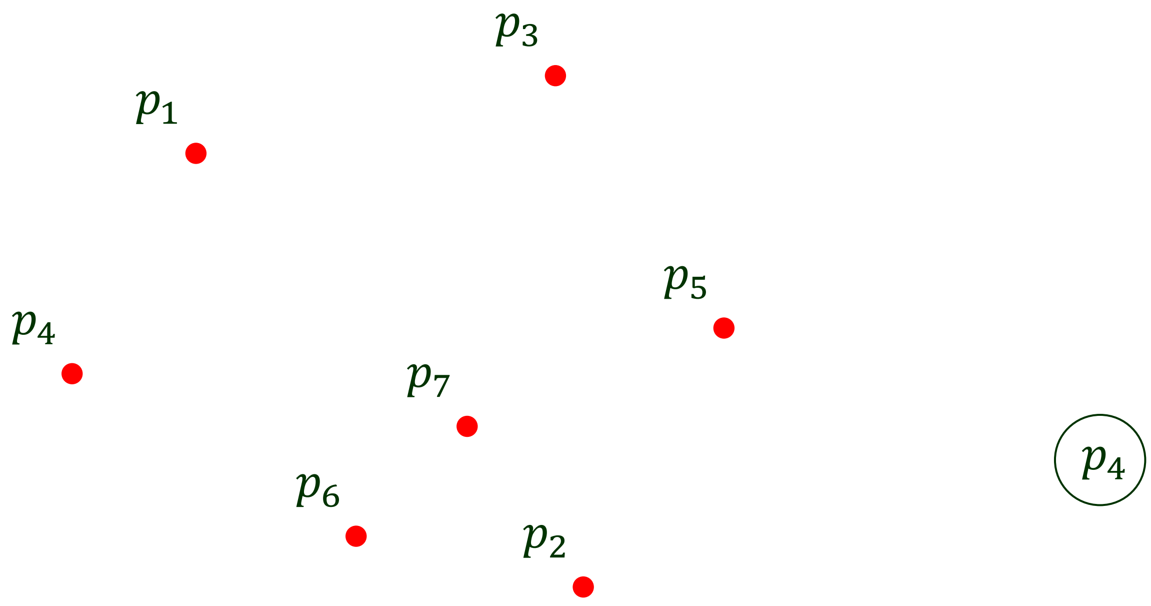
- We can choose a heap that partitions the set according to the y-coordinate.

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)

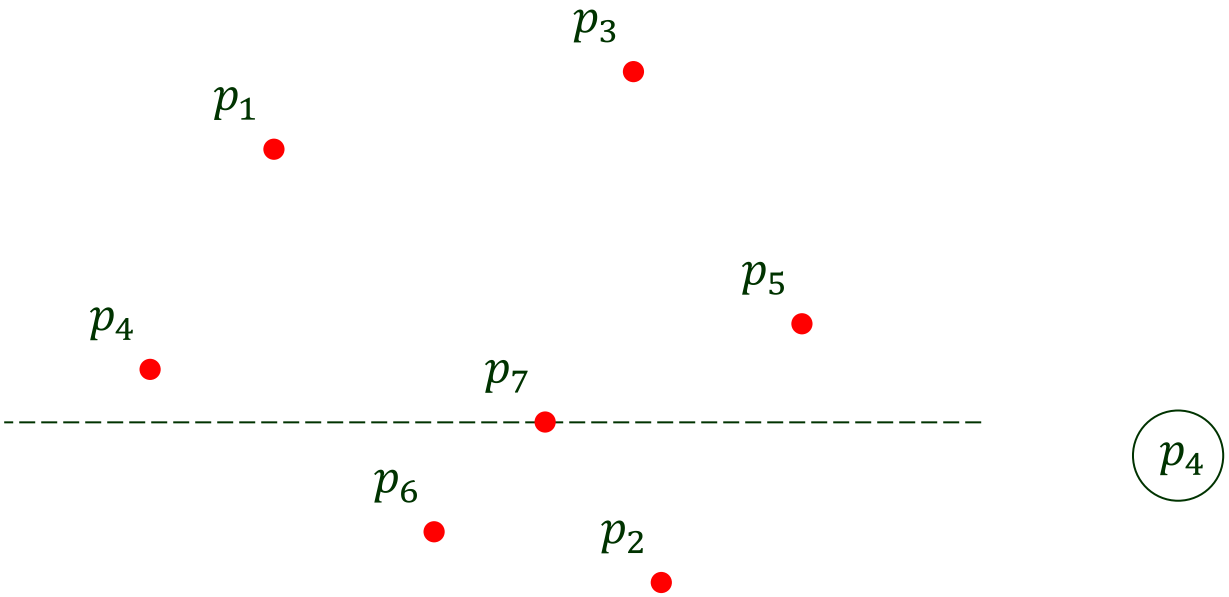


II. Priority Search Tree (PST)



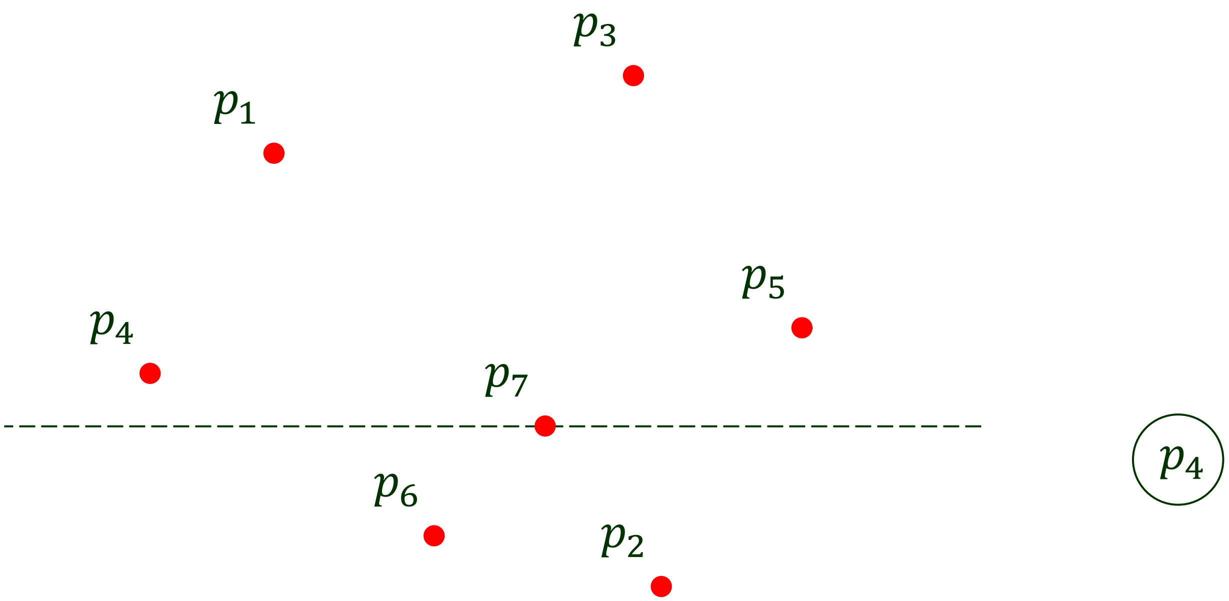
- p_4 is at the root because it has the smallest x-coordinate.

II. Priority Search Tree (PST)



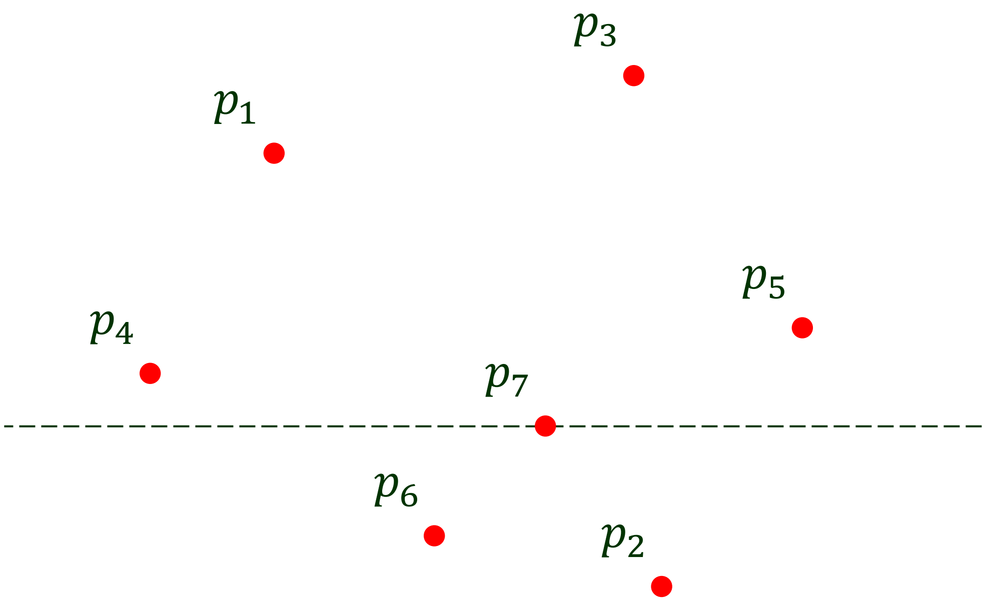
- p_4 is at the root because it has the smallest x-coordinate.
- Of the remaining 6 points, p_7 has the median y-coordinate.

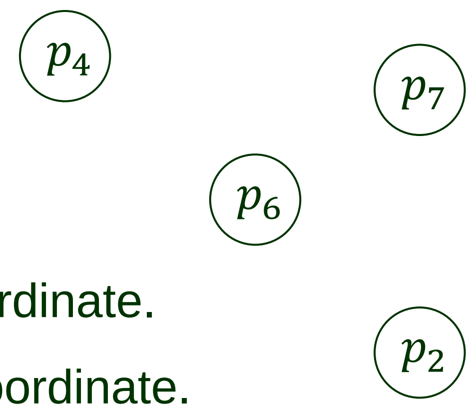
II. Priority Search Tree (PST)



- 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.

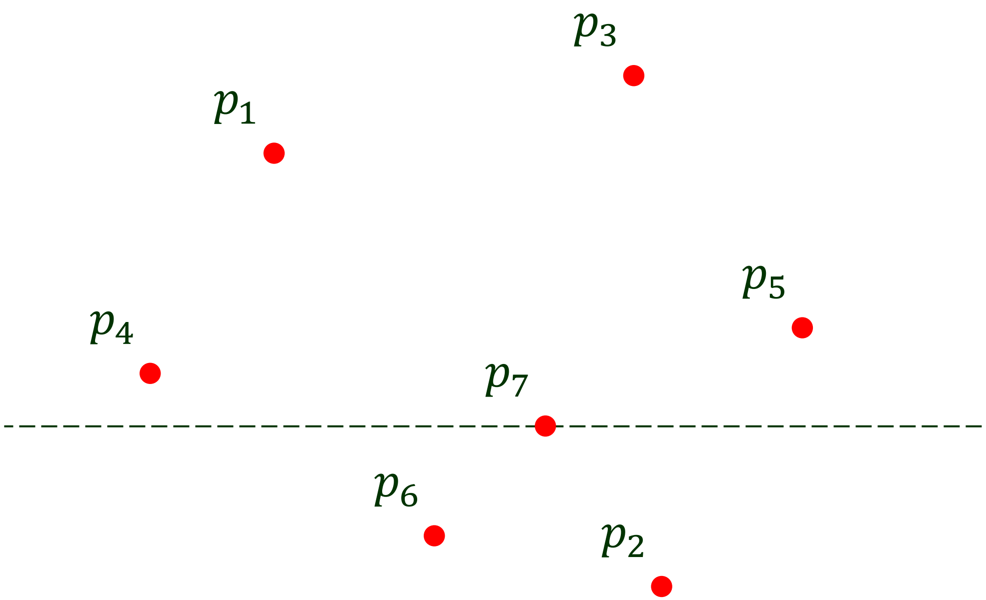
II. Priority Search Tree (PST)

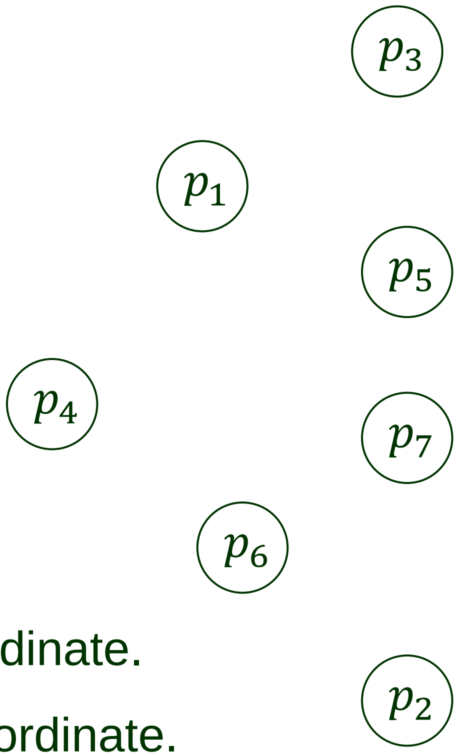




- p_4 is at the root because it has the smallest x-coordinate.
- Of the remaining 6 points, p_7 has the median y-coordinate.
- This median splits them into two groups: p_2, p_6, p_7 stored in the left (lower) subtree and p_1, p_3, p_5 stored in the right (upper) subtree.

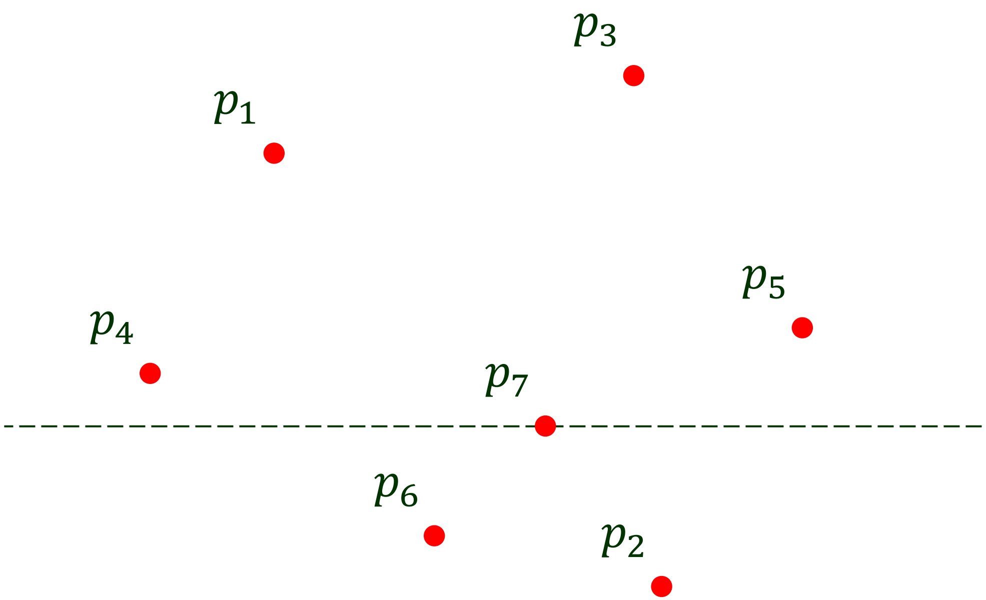
II. Priority Search Tree (PST)

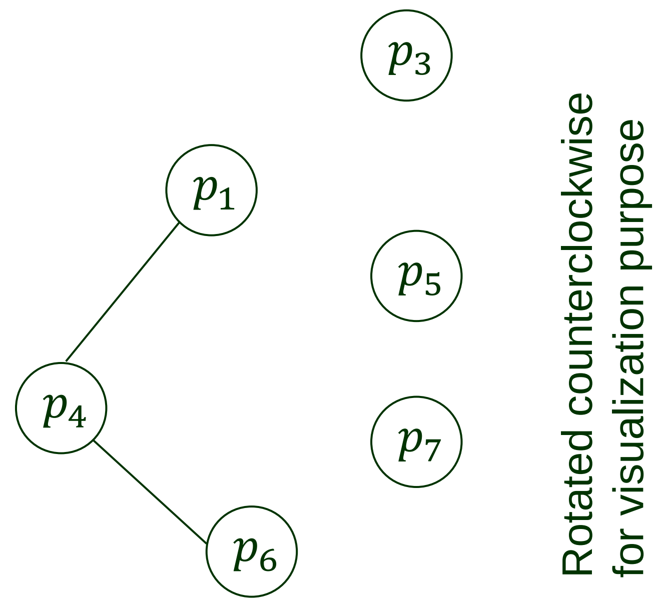


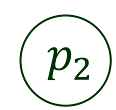


- p_4 is at the root because it has the smallest x-coordinate.
- Of the remaining 6 points, p_7 has the median y-coordinate.
- This median splits them into two groups: p_2, p_6, p_7 stored in the left (lower) subtree and p_1, p_3, p_5 stored in the right (upper) subtree.

II. Priority Search Tree (PST)

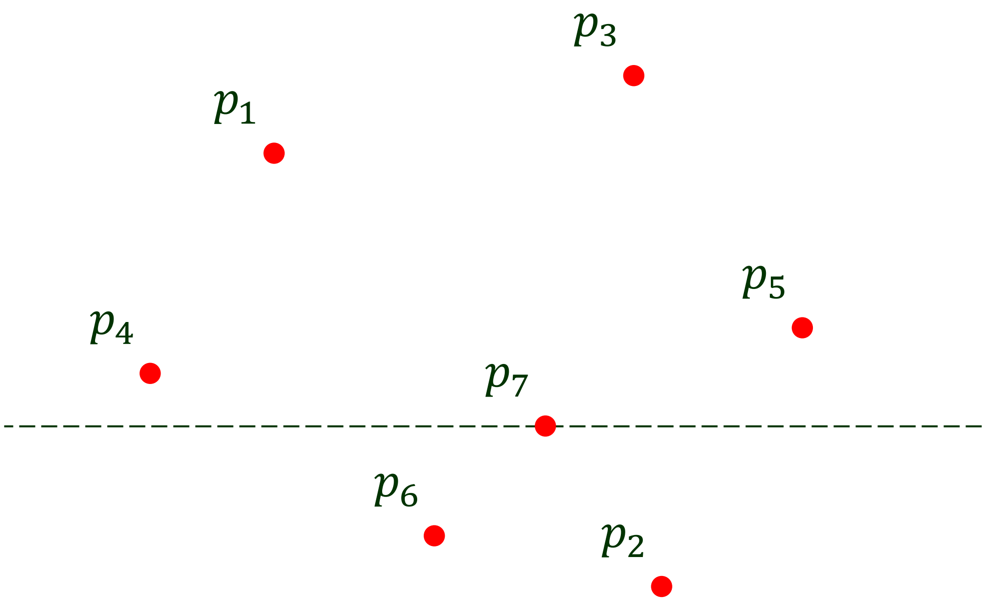


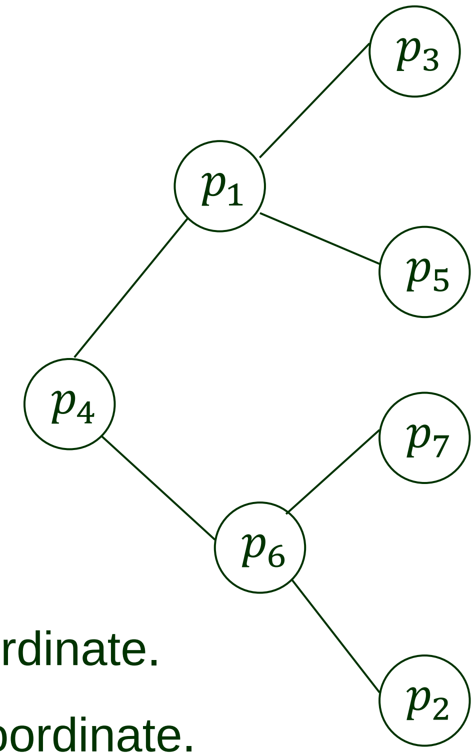




- p_4 is at the root because it has the smallest x-coordinate.
- Of the remaining 6 points, p_7 has the median y-coordinate.
- This median splits them into two groups: p_2, p_6, p_7 stored in the left (lower) subtree and p_1, p_3, p_5 stored in the right (upper) subtree.
- p_6 and p_1 are the roots of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

II. Priority Search Tree (PST)





Rotated counterclockwise
for visualization purpose

- 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.
  - $P_{below} = \{p \in P \setminus \{p_{min}\} \mid p_y \leq y_{mid}\}$.

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,
  - $p_{min} \in P$ with the smallest x-coordinate.
  - y_{mid} : median y-coordinate of the remaining points.
  - $P_{below} = \{p \in P \setminus \{ p_{min}\} \mid p_y \leq y_{mid}\}$.
  - $P_{above} = \{p \in P \setminus \{ p_{min}\} \mid p_y > y_{mid}\}$.

Formal Definition of the PST

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

(Easily removable with lexicographic ordering)

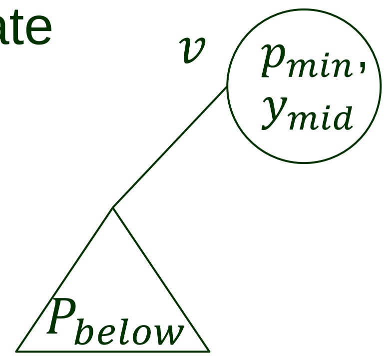
- If $|P| = 1$, then the tree has one node.
- Otherwise,
  - $p_{min} \in P$ with the smallest x-coordinate.
  - y_{mid} : median y-coordinate of the remaining points.
  - $P_{below} = \{p \in P \setminus \{\, p_{min}\} \mid p_y \leq y_{mid}\}$.
  - $P_{above} = \{p \in P \setminus \{\, p_{min}\} \mid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

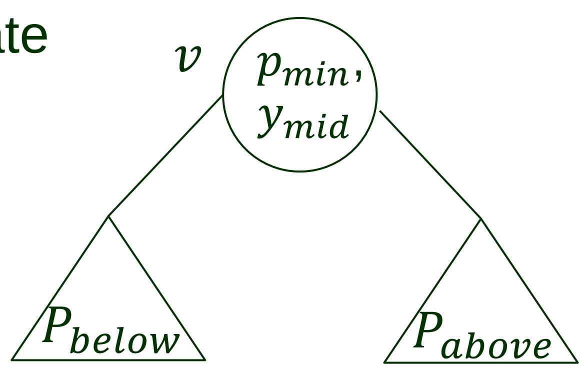


Formal Definition of the PST

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.
  - $P_{below} = \{p \in P \setminus \{ p_{min}\} \mid p_y \le y_{mid}\}$.
  - $P_{above} = \{p \in P \setminus \{ p_{min}\} \mid p_y > y_{mid}\}$.
  - Create

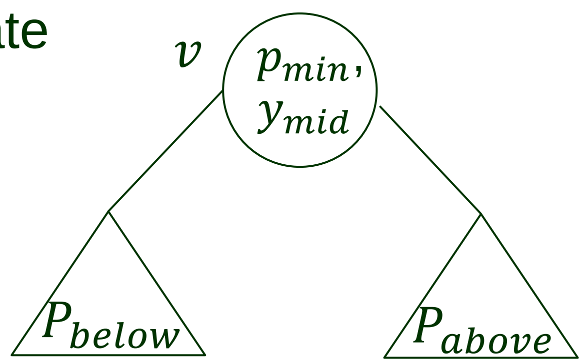


Formal Definition of the PST

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

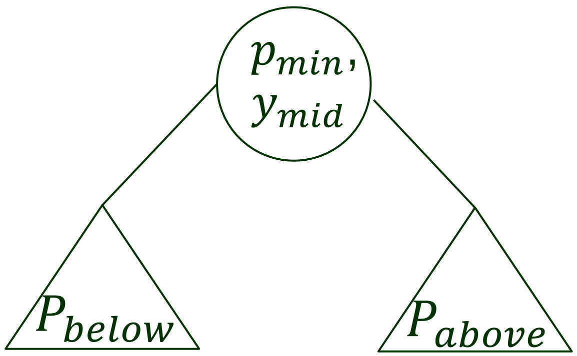
(Easily removable with lexicographic ordering)

- 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



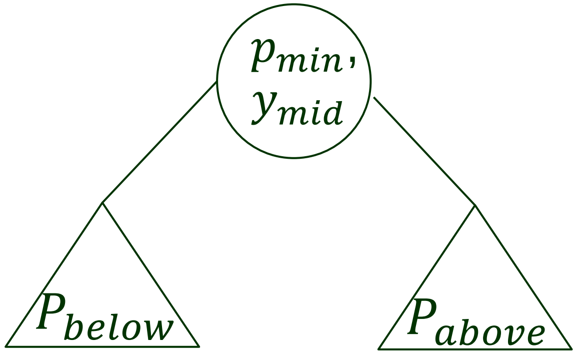
- Points are not stored at leaves only.
- Every node stores a different point.

Construction Time



$O(n \log n)$ if recursively (top-down)

Construction Time



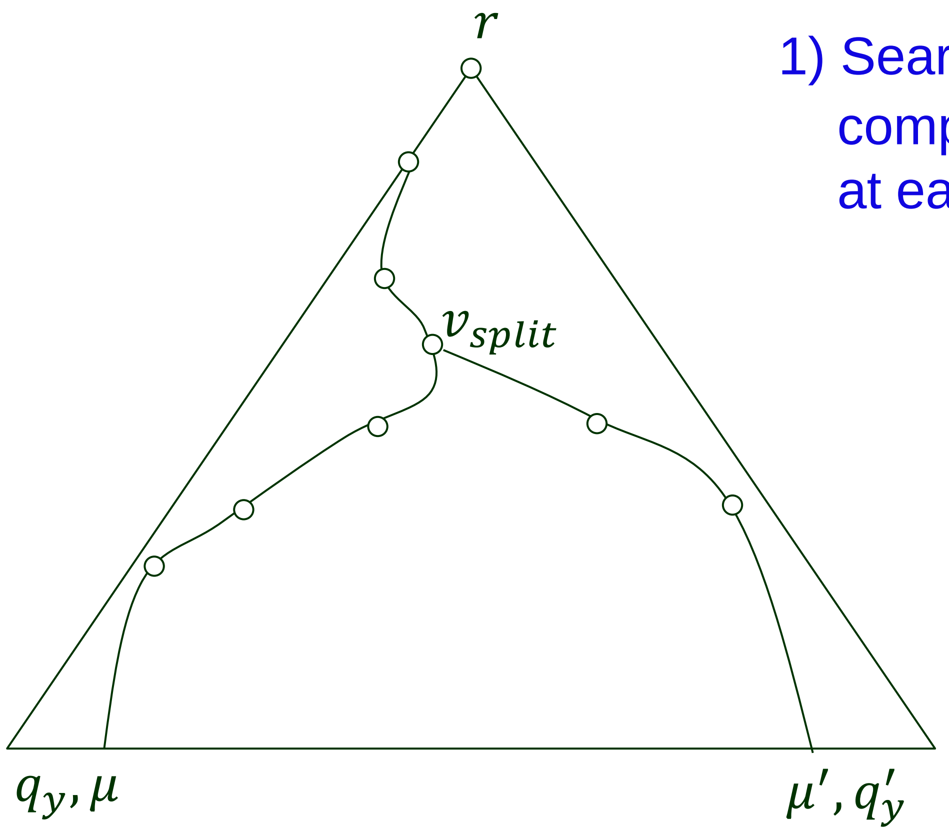
$O(n \log n)$ if recursively (top-down)

$O(n)$ if

- the points are pre-sorted on y-coordinate, and
- constructed bottom-up in the way of building a heap.

III. Query

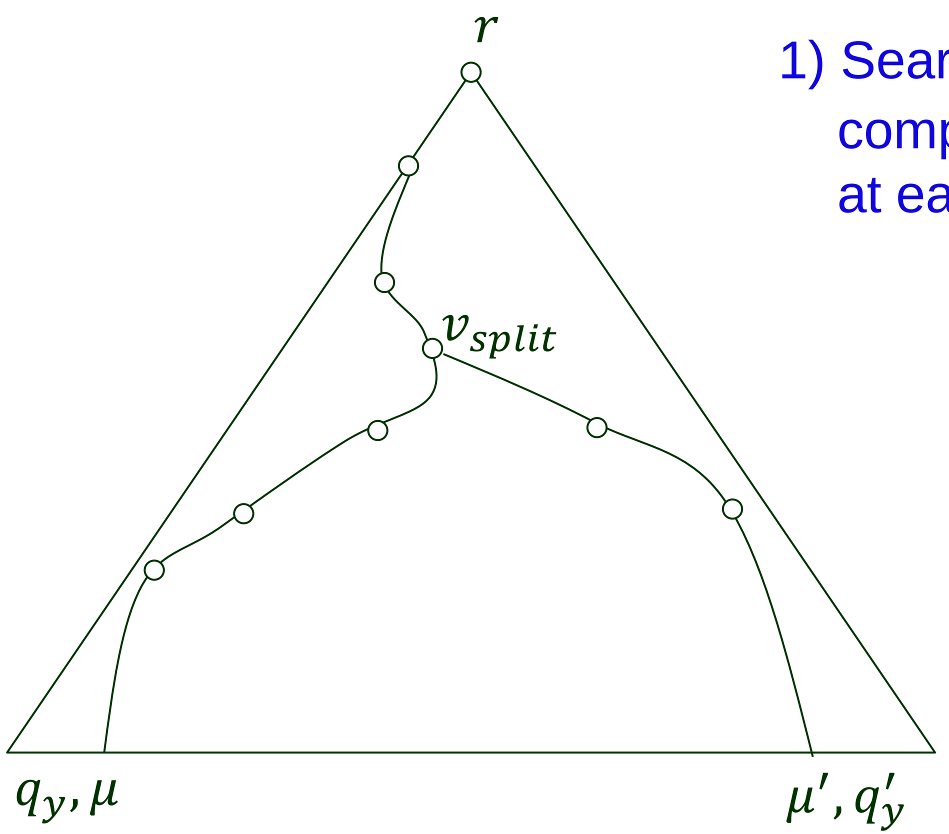
Query range: $(-\infty, q_x] \times [q_y, q_y']$



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: $(-\infty, q_x] \times [q_y, q_y']$

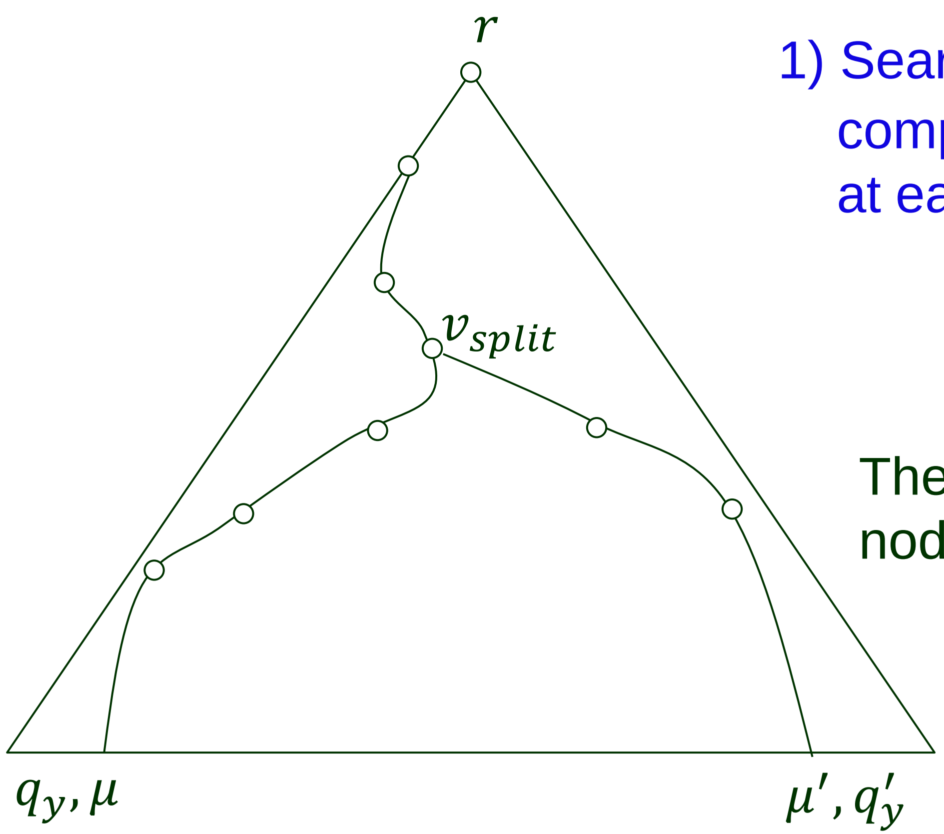


1) Search the PST with q_y and q_y' by comparing them with the y_{mid} value at each node.

1D range searching

III. Query

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



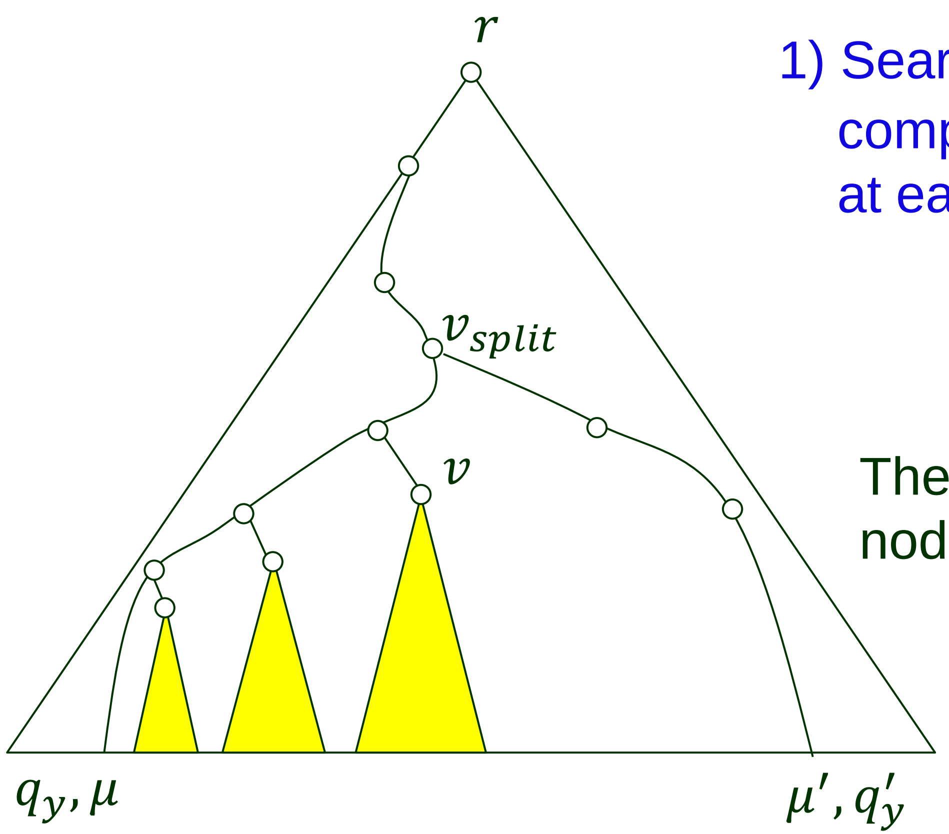
1) Search the PST with q_y and q'_y by comparing them with the y_{mid} value at each node.

1D range searching

The two searches end at the nodes μ and μ', respectively.

III. Query

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



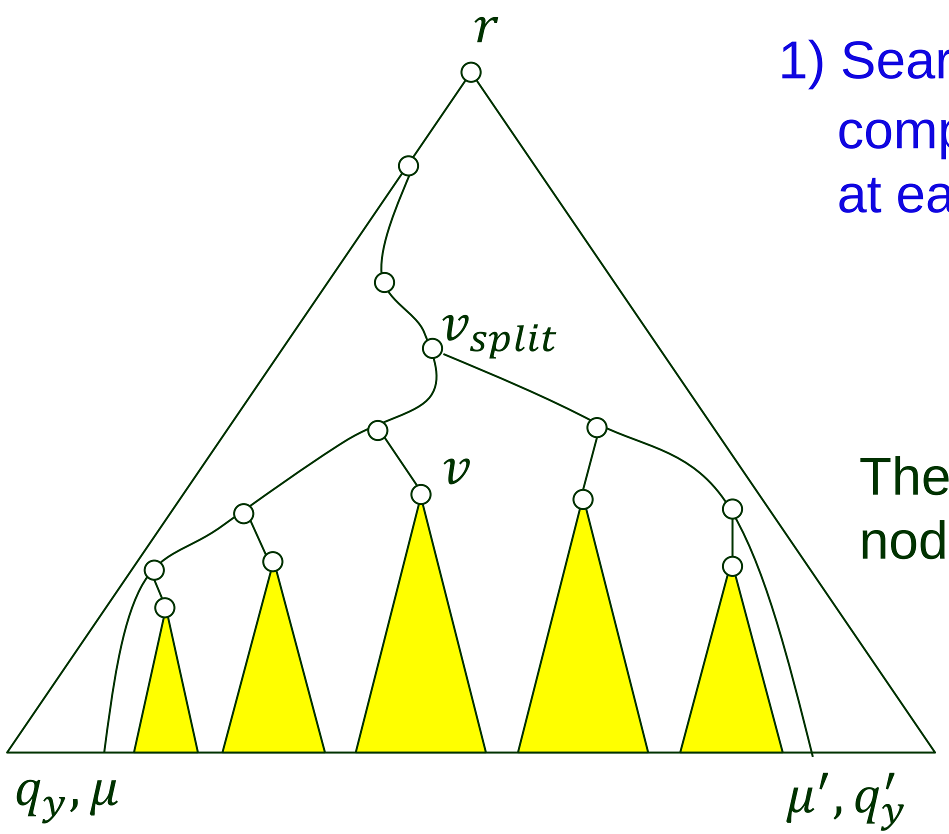
1) Search the PST with q_y and q'_y by comparing them with the y_{mid} value at each node.

1D range searching

The two searches end at the nodes μ and μ', respectively.

III. Query

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



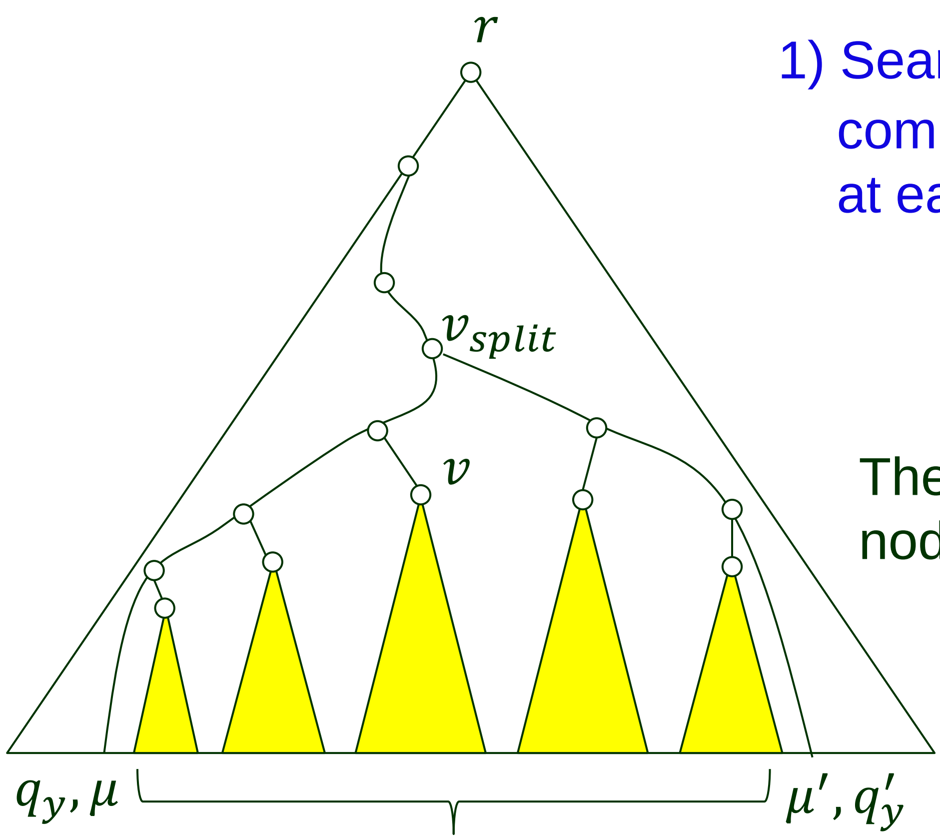
1) Search the PST with q_y and q'_y by comparing them with the y_{mid} value at each node.

1D range searching

The two searches end at the nodes μ and μ', respectively.

III. Query

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



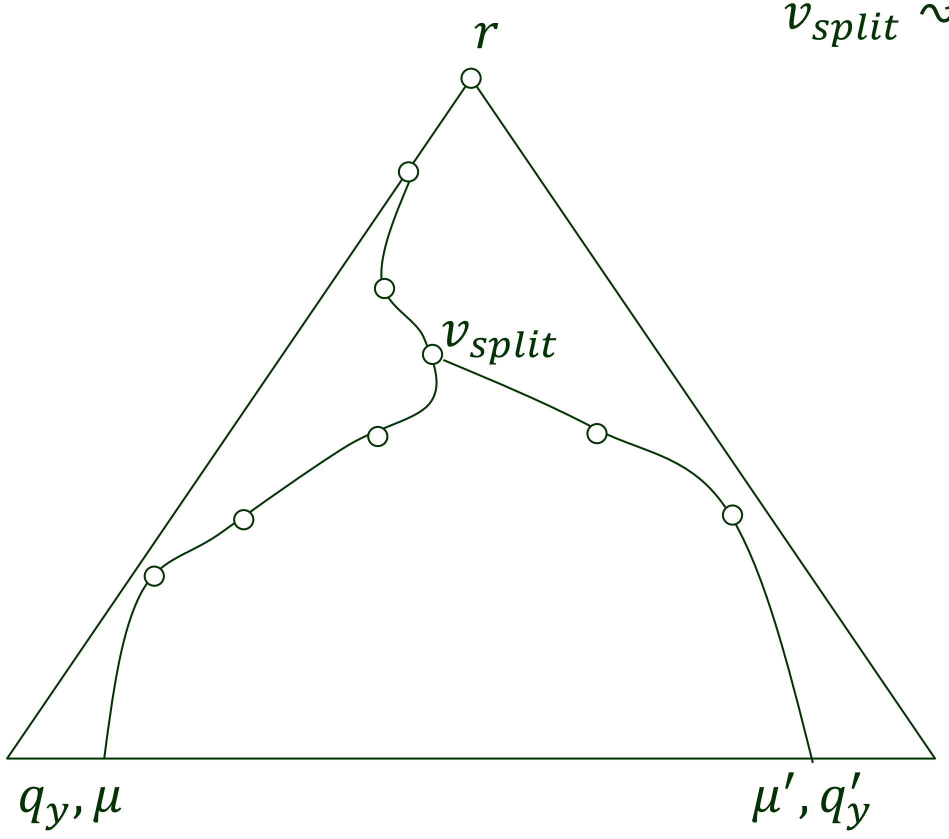
Selected subtrees for future searches based on x-coordinates

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 μ and μ', respectively.

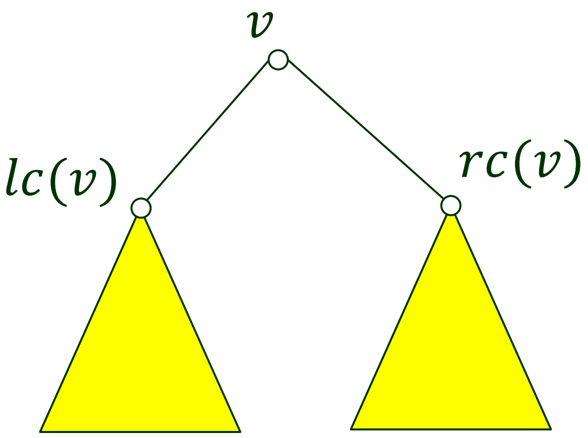
Nodes on the Search Paths



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

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

Search in the Selected Subtrees

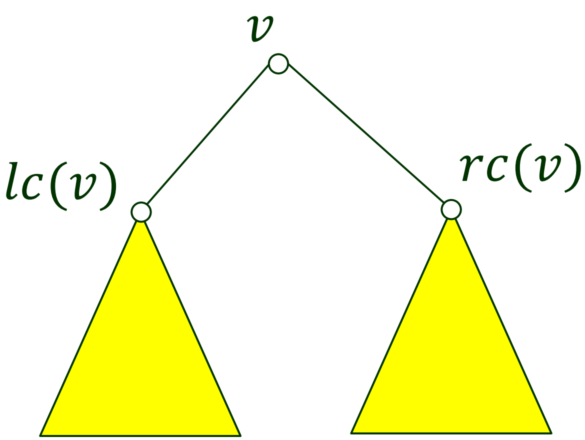


min heap
on x-coordinate

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.



min heap
on x-coordinate

```
ReportInSubtree(v, q_x)
// incorrect in the text for omitting
// the case of v as a leaf
if (p(v))_x ≤ q_x
    then report p(v)
    if v is not a leaf
        then ReportInSubtree(lc(v), q_x)
             ReportInSubtree(rc(v), q_x)
```

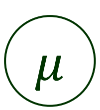
IV. Correctness of `ReportInSubtree()`

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, 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 μ in $\mathcal{T}(v)$ such that its stored point $p(\mu)$ satisfies $\big(p(\mu)\big)_x \le q_x$.

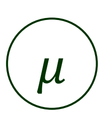


IV. Correctness of `ReportInSubtree()`

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 μ in $\mathcal{T}(v)$ such that its stored point $p(\mu)$ satisfies $\big(p(\mu)\big)_x \le q_x$.

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

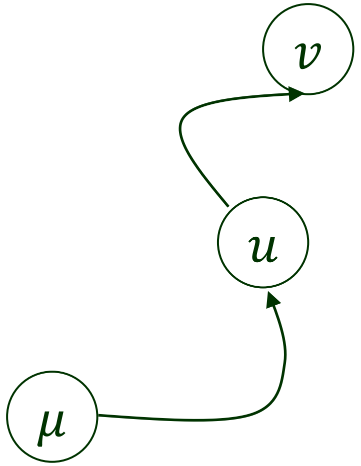


IV. Correctness of `ReportInSubtree()`

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

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

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

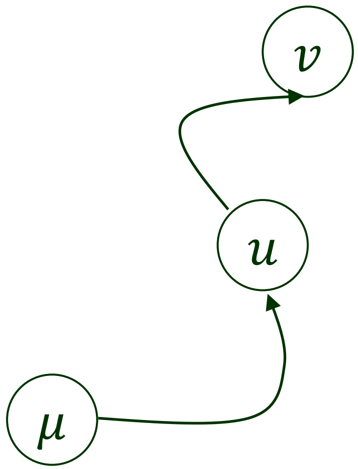


IV. Correctness of `ReportInSubtree()`

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

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

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



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

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.

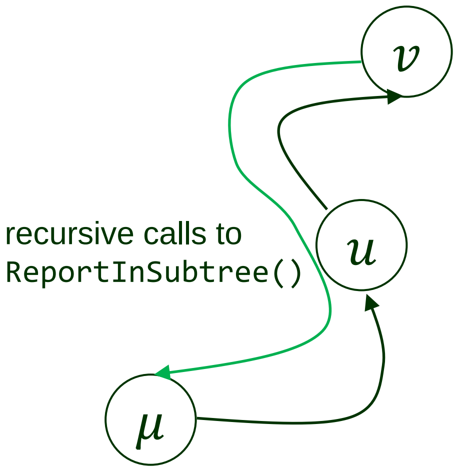
Proof Consider a node μ in $\mathcal{T}(v)$ such that its stored point $p(\mu)$ satisfies $\big(p(\mu)\big)_x \le q_x$.

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

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

$$\Downarrow$$

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 $\mathcal{T}(v)$ whose x-coordinate is at most q_x.

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

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

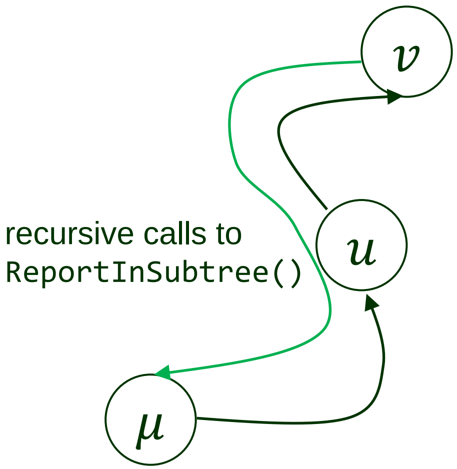
$$q_x \ge \big(p(\mu)\big)_x > \cdots > \big(p(u)\big)_x > \cdots > \big(p(v)\big)_x$$

$$\Downarrow$$

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

$$\Downarrow$$

$p(\mu)$ 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 μ in $\mathcal{T}(v)$ such that its stored point $p(\mu)$ satisfies $\big(p(\mu)\big)_x \le q_x$.

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

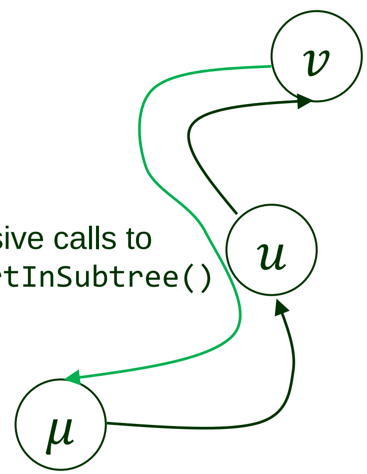
$$q_x \ge \big(p(\mu)\big)_x > \cdots > \big(p(u)\big)_x > \cdots > \big(p(v)\big)_x$$

$$\Downarrow$$

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

$$\Downarrow$$

$p(\mu)$ is reported.



recursive calls to `ReportInSubtree()`

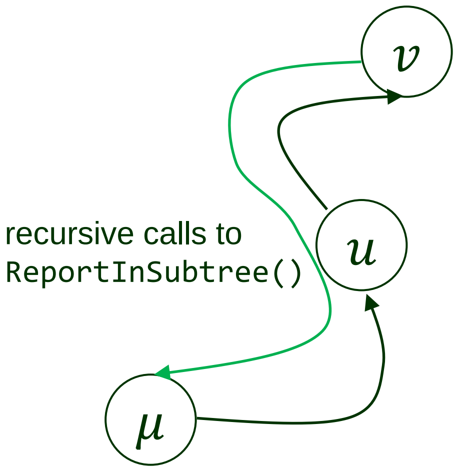
The time $O(1+k_v)$ follows from $O(1)$ effort spent on each node .

IV. Correctness of `ReportInSubtree()`

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

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

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



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

$$\Downarrow$$

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

$$\Downarrow$$

$p(\mu)$ is reported.

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

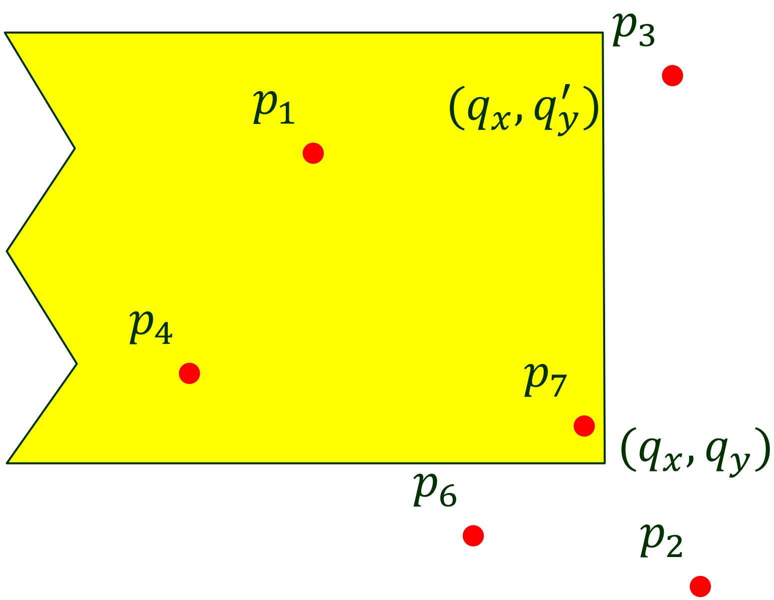
Query Algorithm

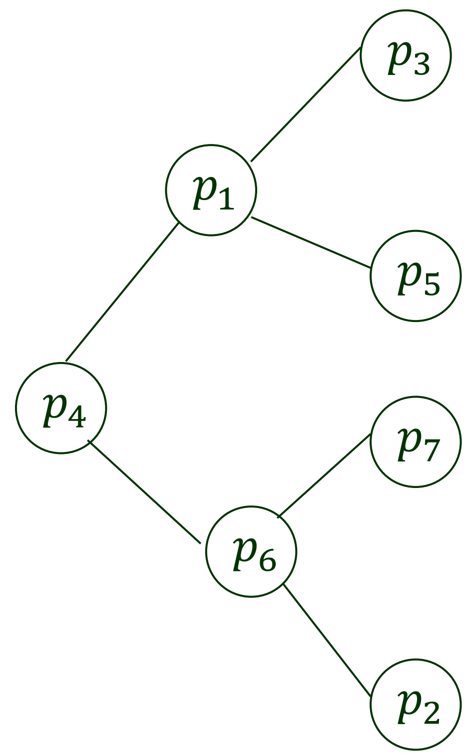
QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q'_y]$)

// r is the root of $\mathcal{T}$

1. search with q_y and q'_y in $\mathcal{T}$, 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 $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow \mu'$
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 $v_{split} \rightsquigarrow \mu$
7.   **do if** the path goes left at v
8.   **then** ReportInSubtree($rc(v)$, q_x)
9. **for** each node v on $v_{split} \rightsquigarrow \mu'$
10.   **do if** the path goes right at v
11.   **then** ReportInSubtree($lc(v)$, q_x)

Example of Execution



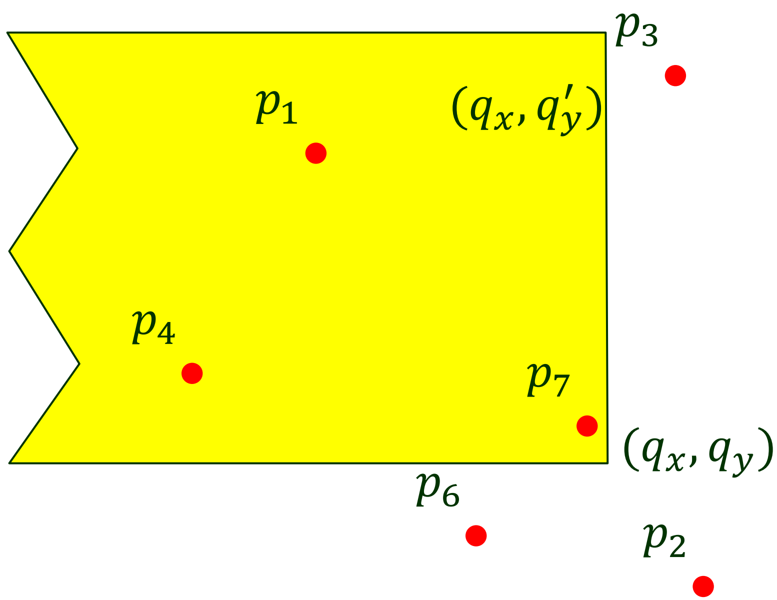


QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q_y']$)
// r is the root of $\mathcal{T}$
search with q_y and q_y' in $\mathcal{T}$, ending at the nodes μ and μ'
let v_{split} be the node where the two paths split.
for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow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

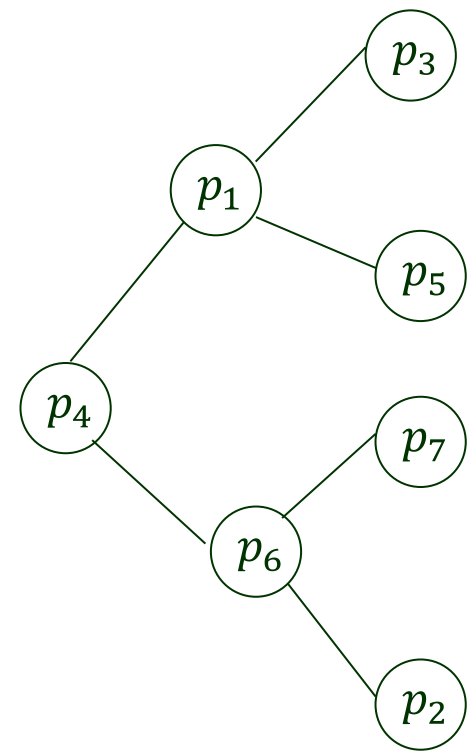
ReportInSubtree(v, q_x)

if $(p(v))_x \leq q_x$
    then report $p(v)$
if v is not a leaf
    then ReportInSubtree($lc(v)$, q_x)
        ReportInSubtree($rc(v)$, q_x)

Example of Execution



$v_{split} = p_4$

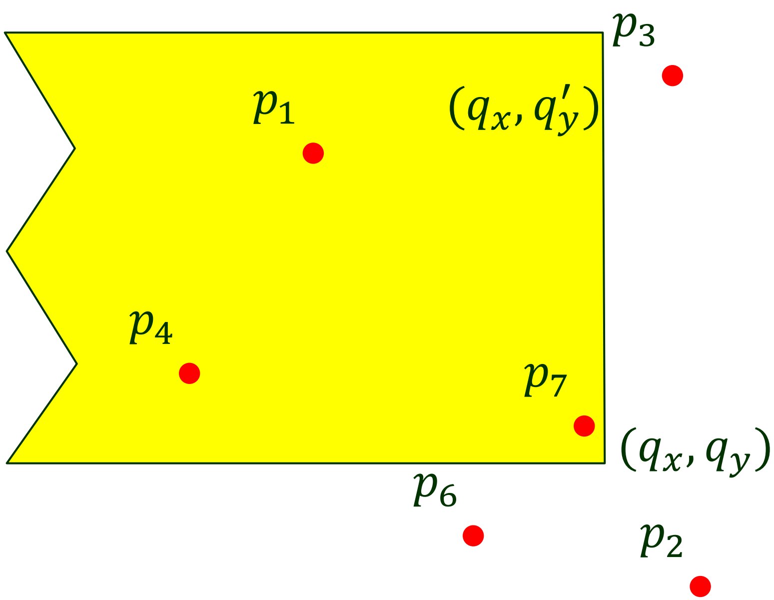


QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q'_y]$)
// r is the root of $\mathcal{T}$
search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'
let v_{split} be the node where the two paths split.
for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow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

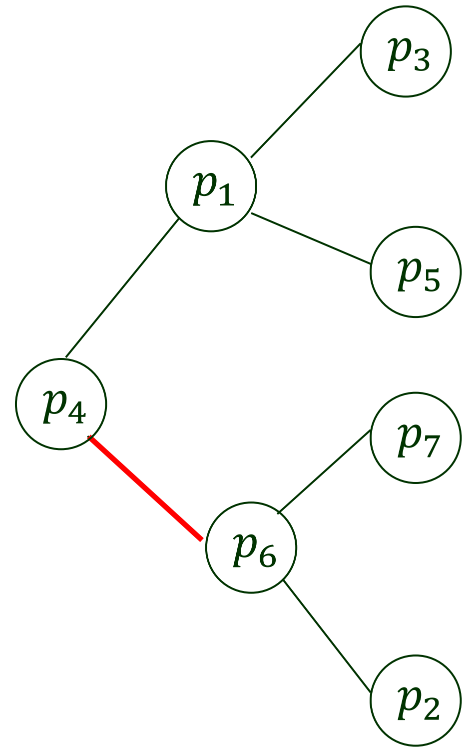
ReportInSubtree(v, q_x)

if $(p(v))_x \leq q_x$
    then report $p(v)$
if v is not a leaf
    then ReportInSubtree($lc(v)$, q_x)
        ReportInSubtree($rc(v)$, q_x)

Example of Execution



$v_{split} = p_4$



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

// r is the root of $\mathcal{T}$

search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'

let v_{split} be the node where the two paths split.

for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow \mu'$

    do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$

        then report $p(v)$

for each node v on $v_{split} \rightsquigarrow \mu$

    do if the path goes left at v

        then ReportInSubtree($rc(v)$, q_x)

for each node v on $v_{split} \rightsquigarrow \mu'$

    do if the path goes right at v

        then ReportInSubtree($lc(v)$, q_x)

ReportInSubtree(v, q_x)

if $(p(v))_x \leq q_x$

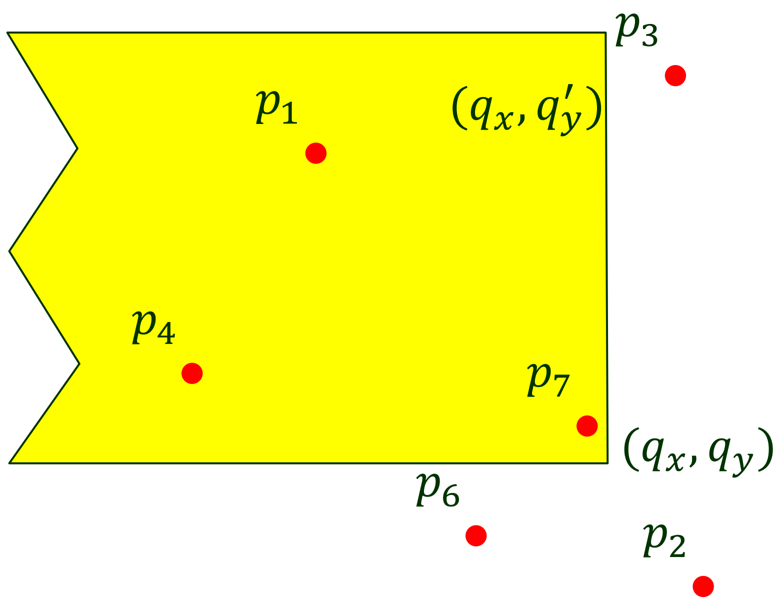
    then report $p(v)$

if v is not a leaf

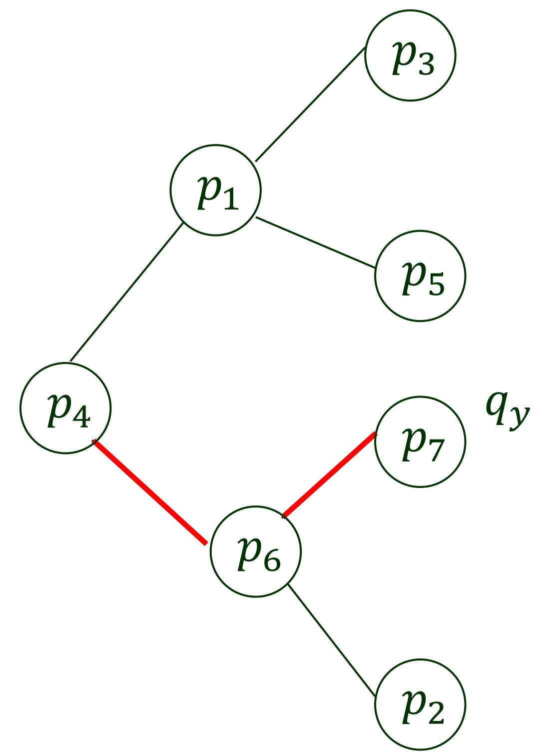
    then ReportInSubtree($lc(v)$, q_x)

        ReportInSubtree($rc(v)$, q_x)

Example of Execution



$v_{split} = p_4$

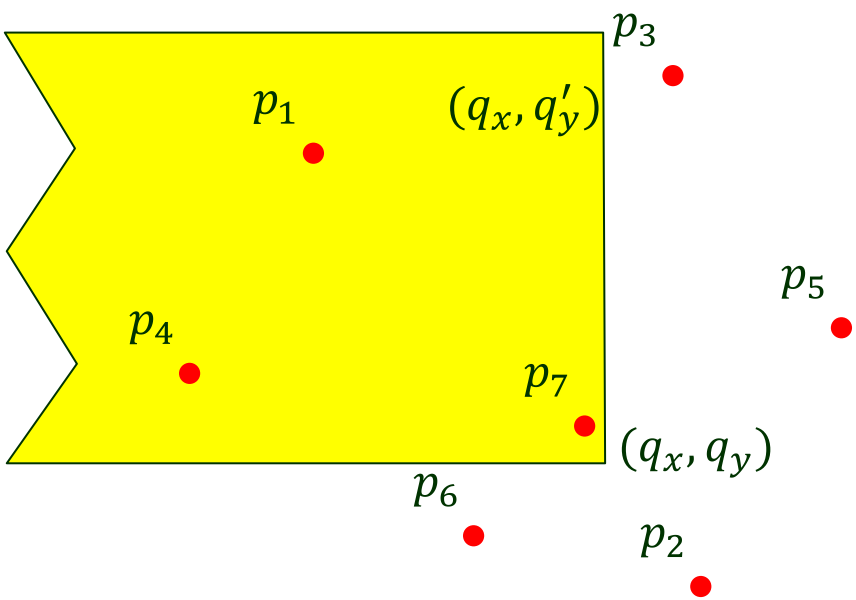


QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q'_y]$)
// r is the root of $\mathcal{T}$
search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'
let v_{split} be the node where the two paths split.
for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow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

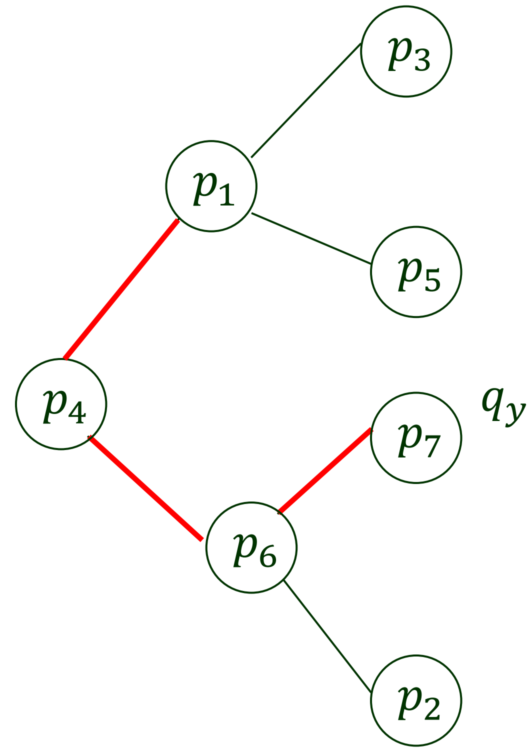
ReportInSubtree(v,q_x)

if $(p(v))_x \le q_x$
    then report $p(v)$
if v is not a leaf
    then ReportInSubtree($lc(v)$, q_x)
        ReportInSubtree($rc(v)$, q_x)

Example of Execution



$v_{split} = p_4$

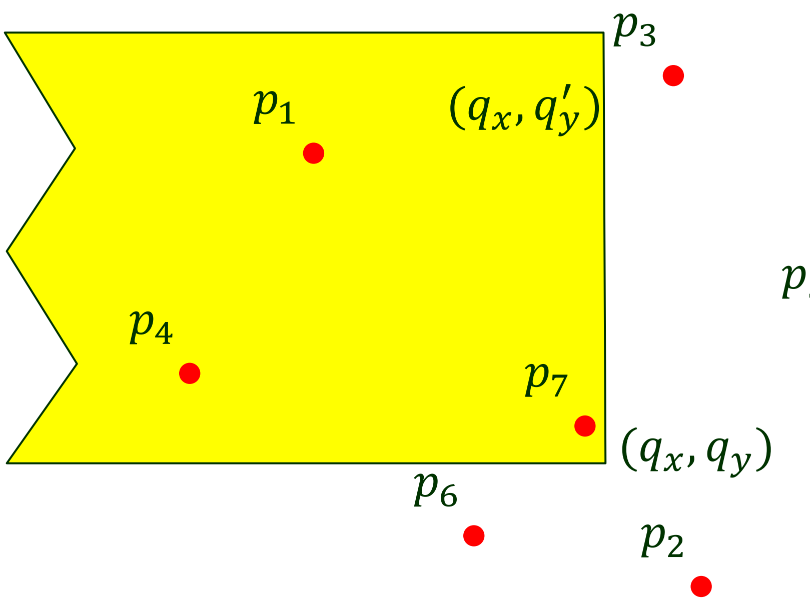


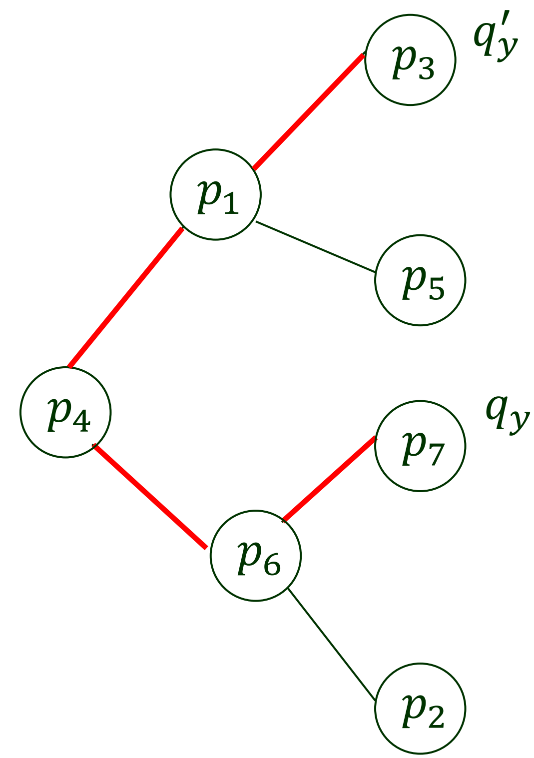
QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q'_y]$)
// r is the root of $\mathcal{T}$
search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'
let v_{split} be the node where the two paths split.
for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow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

ReportInSubtree(v,q_x)

if $(p(v))_x \leq q_x$
  then report $p(v)$
if v is not a leaf
  then ReportInSubtree($lc(v)$, q_x)
    ReportInSubtree($rc(v)$, q_x)

Example of Execution





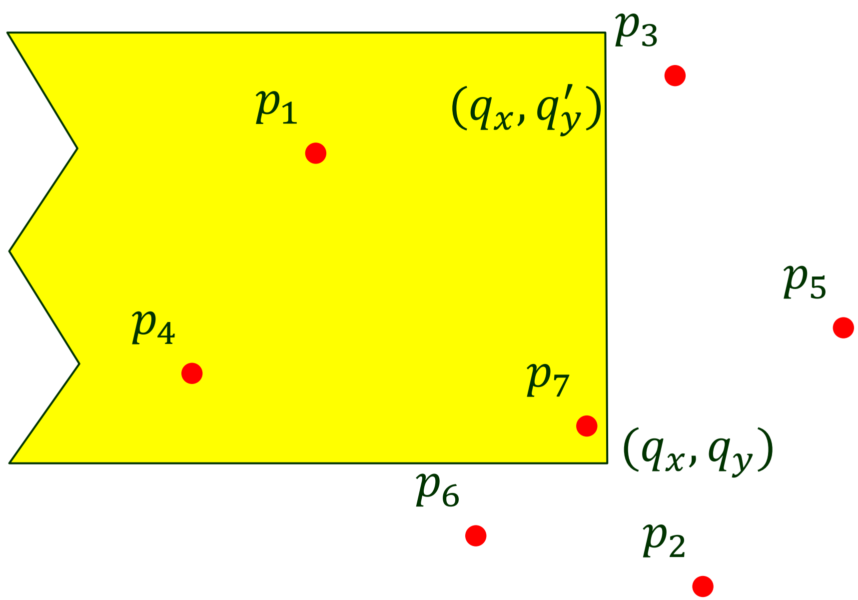
$$v_{split} = p_4$$

```
QueryPrioSearchTree(𝒯, (−∞, q_x] × [q_y, q'_y])
// r is the root of 𝒯
    search with q_y and q'_y in 𝒯, ending at the nodes μ and μ'
    let v_split be the node where the two paths split.
    for each node v on the path r ⇝ μ or v_split ⇝ μ'
        do if p(v) ∈ (−∞, q_x] × [q_y, q'_y]
              then report p(v)
    for each node v on v_split ⇝ μ
        do if the path goes left at v
              then ReportInSubtree(rc(v), q_x)
    for each node v on v_split ⇝ μ'
        do if the path goes right at v
              then ReportInSubtree(lc(v), q_x)
```

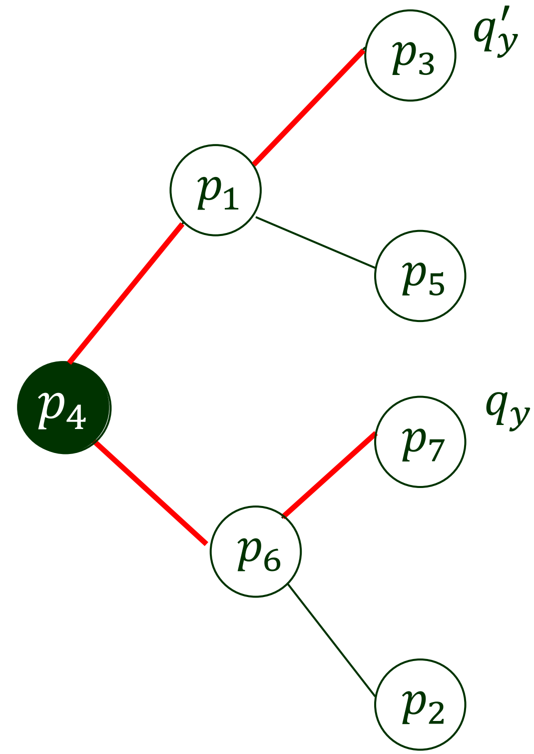
```
ReportInSubtree(v, q_x)

    if (p(v))_x ≤ q_x
        then report p(v)
    if v is not a leaf
        then ReportInSubtree(lc(v), q_x)
             ReportInSubtree(rc(v), q_x)
```

Example of Execution



$v_{split} = p_4$

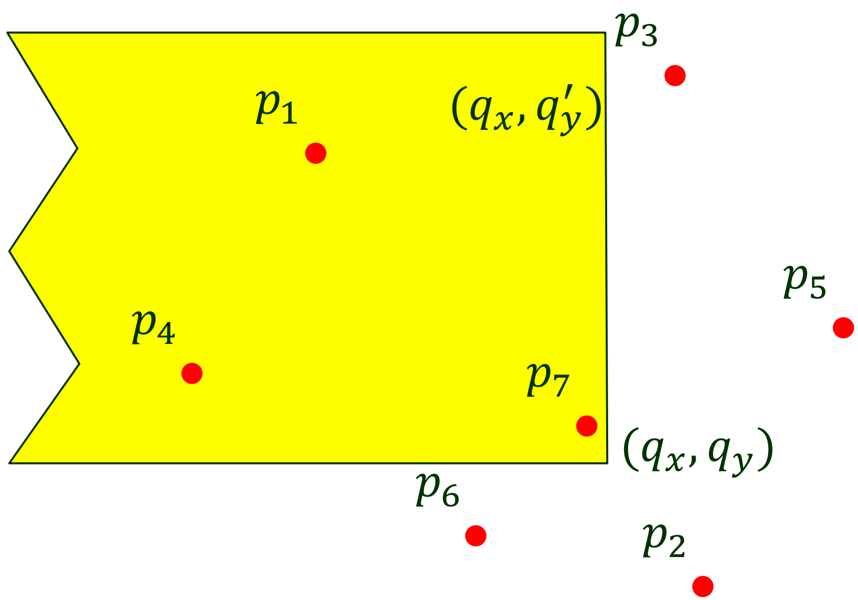


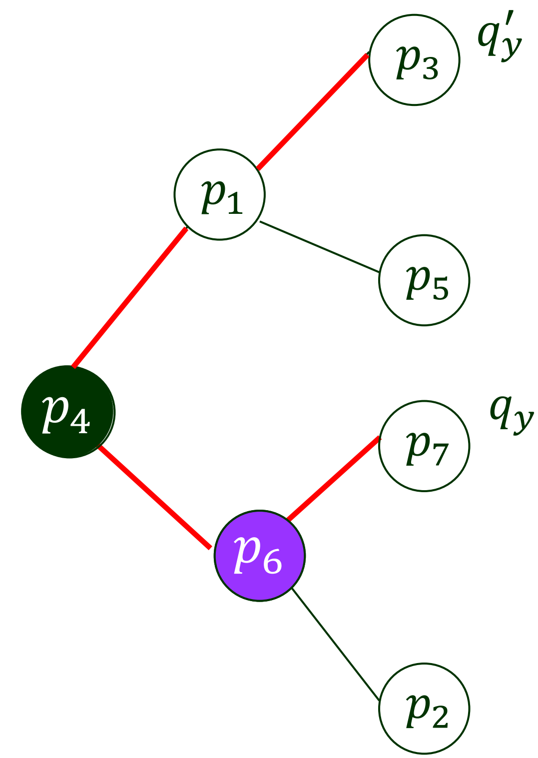
QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q'_y]$)
// r is the root of $\mathcal{T}$
    search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'
    let v_{split} be the node where the two paths split.
    for each node v on the path $r \rightsquigarrow \mu$ or $v_{split}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

ReportInSubtree(v,q_x)

    if $(p(v))_x \le q_x$
        then report $p(v)$
    if v is not a leaf
        then ReportInSubtree($lc(v)$, q_x)
            ReportInSubtree($rc(v)$, q_x)

Example of Execution





$v_{split} = p_4$

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

// r is the root of $\mathcal{T}$

search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'

let v_{split} be the node where the two paths split.

for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow \mu'$

    do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$

        then report $p(v)$

for each node v on $v_{split} \rightsquigarrow \mu$

    do if the path goes left at v

        then ReportInSubtree($rc(v)$, q_x)

for each node v on $v_{split} \rightsquigarrow \mu'$

    do if the path goes right at v

        then ReportInSubtree($lc(v)$, q_x)

ReportInSubtree(v, q_x)

if $(p(v))_x \le q_x$

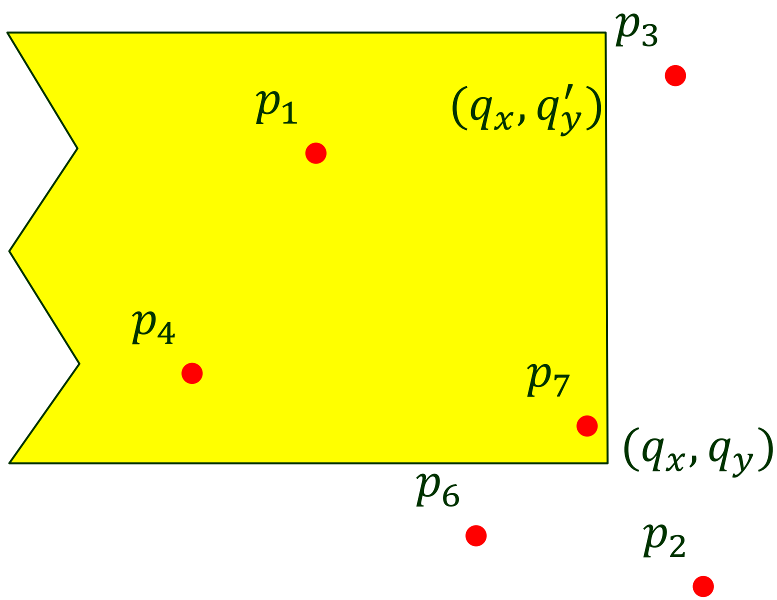
    then report $p(v)$

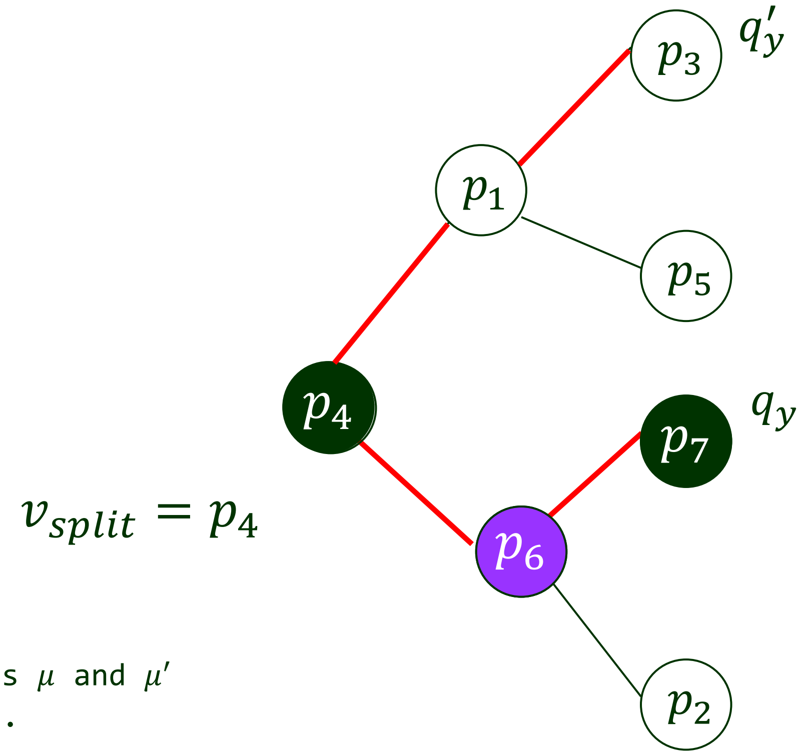
if v is not a leaf

    then ReportInSubtree($lc(v)$, q_x)

        ReportInSubtree($rc(v)$, q_x)

Example of Execution



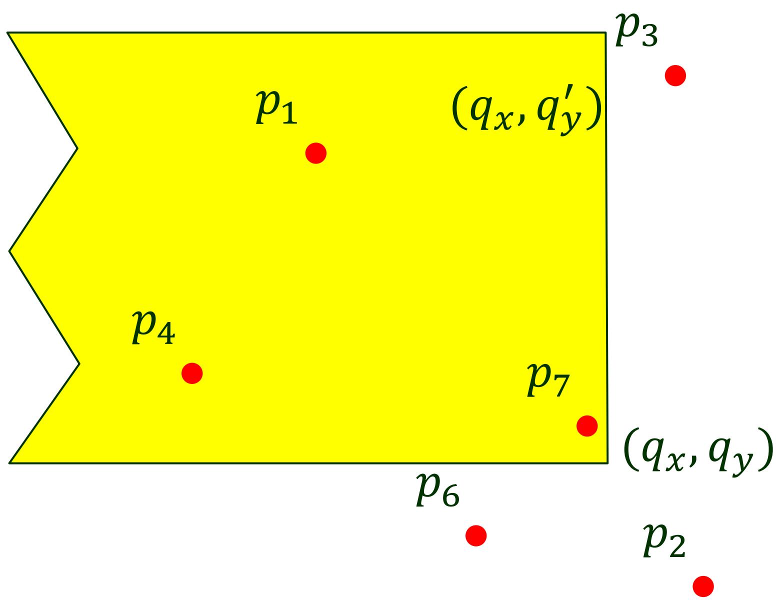


QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q'_y]$)
// r is the root of $\mathcal{T}$
search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'
let v_{split} be the node where the two paths split.
for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow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

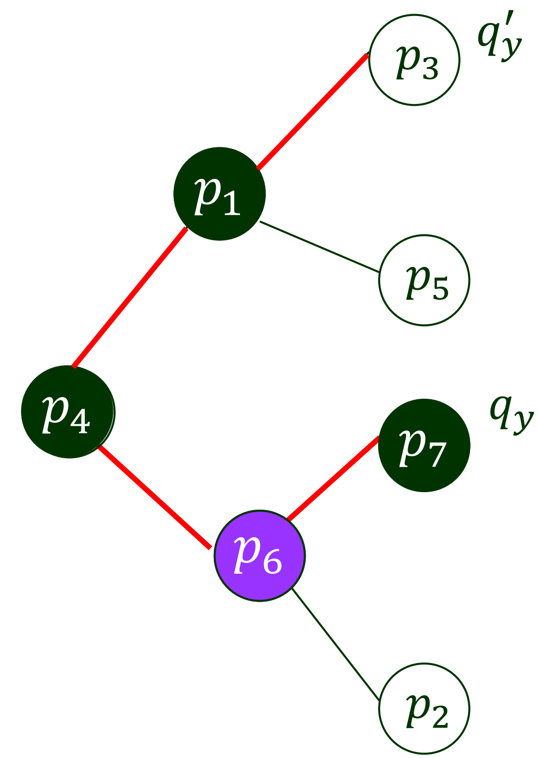
ReportInSubtree(v,q_x)

if $(p(v))_x \le q_x$
    then report $p(v)$
if v is not a leaf
    then ReportInSubtree($lc(v)$, q_x)
        ReportInSubtree($rc(v)$, q_x)

Example of Execution



$v_{split} = p_4$

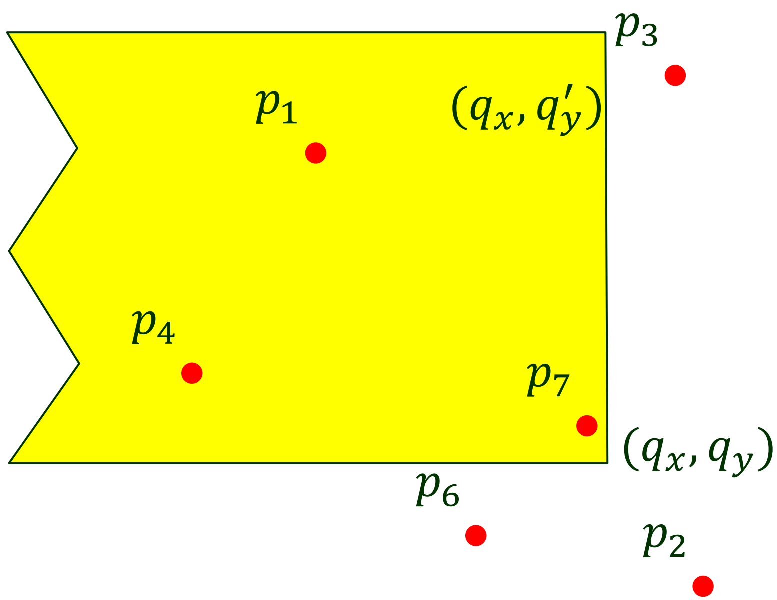


QueryPrioSearchTree($\mathcal{T}$, $(-\infty, q_x] \times [q_y, q_y']$)
// r is the root of $\mathcal{T}$
search with q_y and q_y' in $\mathcal{T}$, ending at the nodes μ and μ'
let v_{split} be the node where the two paths split.
for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow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

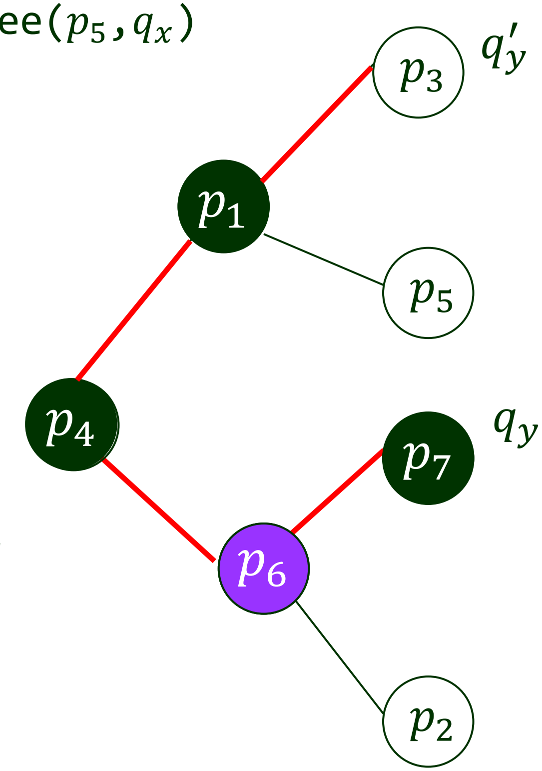
ReportInSubtree(v,q_x)

if $(p(v))_x \leq q_x$
    then report $p(v)$
if v is not a leaf
    then ReportInSubtree($lc(v)$, q_x)
        ReportInSubtree($rc(v)$, q_x)

Example of Execution



ReportInSubtree(p_5,q_x)



$v_{split} = p_4$

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

// r is the root of $\mathcal{T}$

search with q_y and q_y' in $\mathcal{T}$, ending at the nodes μ and μ'

let v_{split} be the node where the two paths split.

for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow \mu'$

  do if $p(v) \in (-\infty, q_x] \times [q_y, q_y']$

    then report $p(v)$

for each node v on $v_{split} \rightsquigarrow \mu$

  do if the path goes left at v

    then ReportInSubtree($rc(v)$, q_x)

for each node v on $v_{split} \rightsquigarrow \mu'$

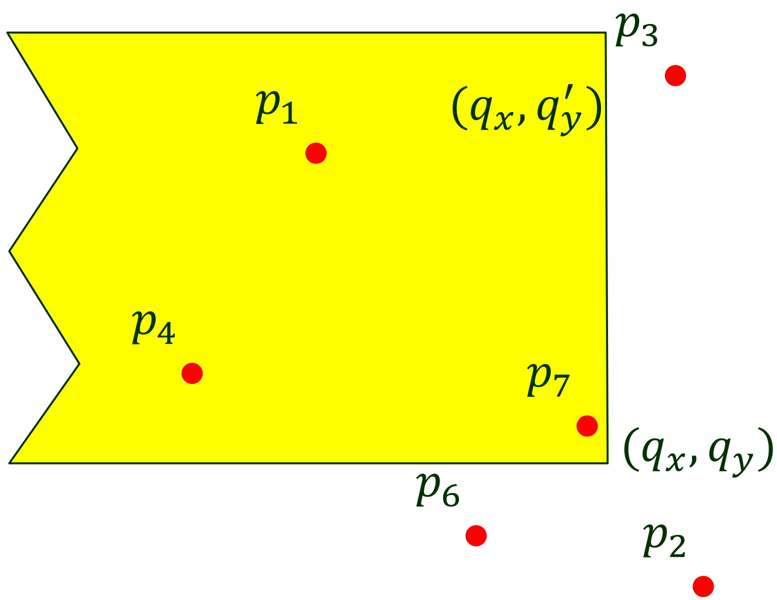
  do if the path goes right at v

    then ReportInSubtree($lc(v)$, q_x)

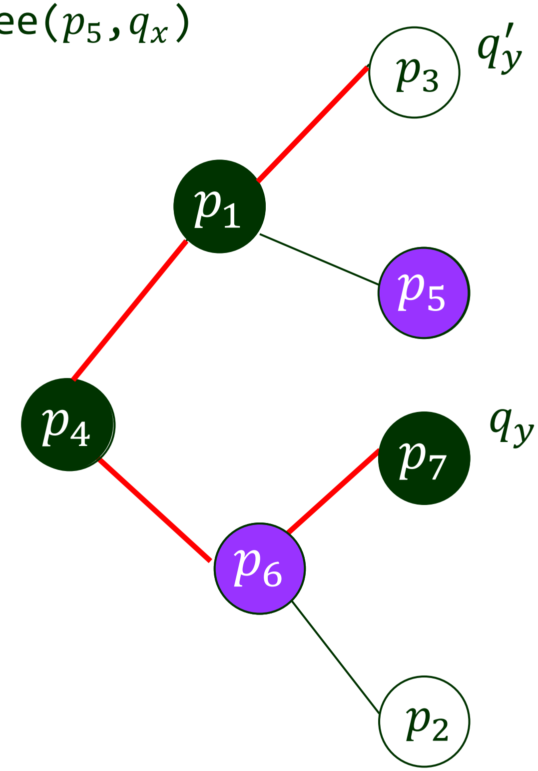
ReportInSubtree(v,q_x)

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



ReportInSubtree(p_5, q_x)



$v_{split} = p_4$

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

// r is the root of $\mathcal{T}$

search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'

let v_{split} be the node where the two paths split.

for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow \mu'$

    do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$

        then report $p(v)$

for each node v on $v_{split} \rightsquigarrow \mu$

    do if the path goes left at v

        then ReportInSubtree$(rc(v), q_x)$

for each node v on $v_{split} \rightsquigarrow \mu'$

    do if the path goes right at v

        then ReportInSubtree$(lc(v), q_x)$

ReportInSubtree(v, q_x)

if $(p(v))_x \leq q_x$

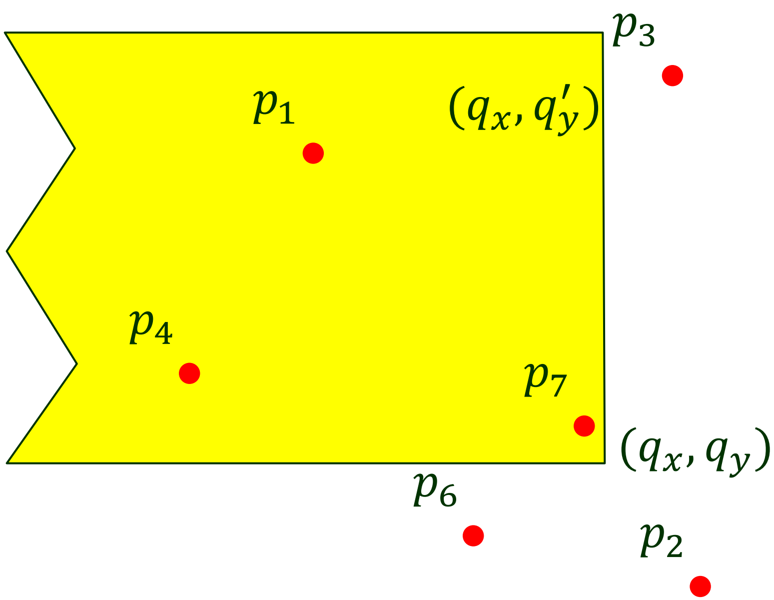
    then report $p(v)$

if v is not a leaf

    then ReportInSubtree$(lc(v), q_x)$

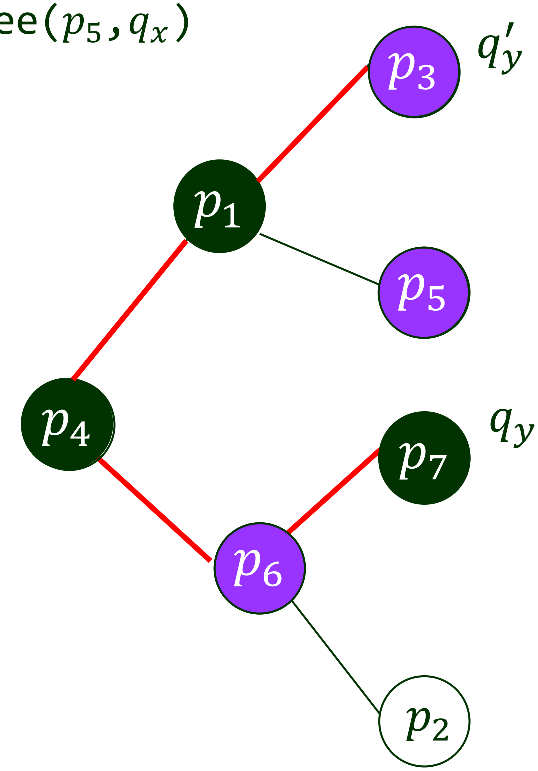
        ReportInSubtree$(rc(v), q_x)$

Example of Execution



ReportInSubtree(p_5,q_x)

$v_{split} = p_4$



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

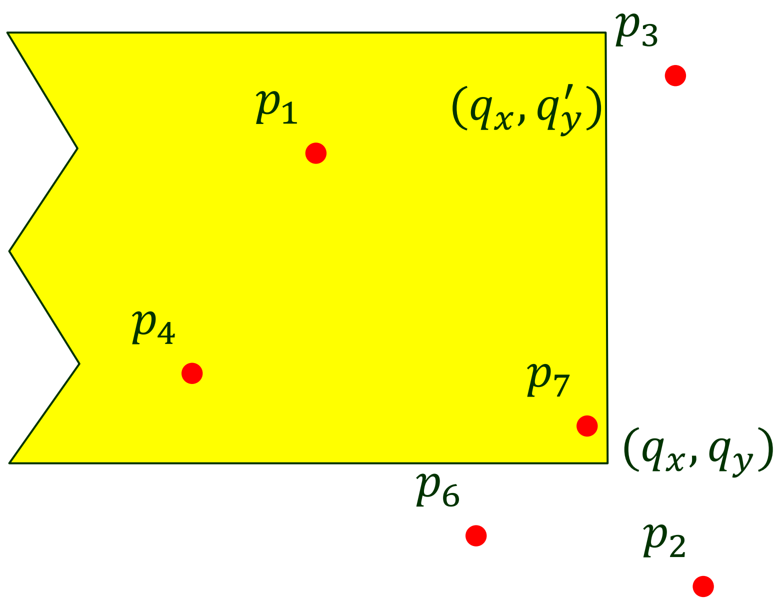
// r is the root of $\mathcal{T}$

search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'
let v_{split} be the node where the two paths split.
for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow \mu'$
  do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$
    then report $p(v)$
for each node v on $v_{split} \rightsquigarrow \mu$
  do if the path goes left at v
    then ReportInSubtree($rc(v)$, q_x)
for each node v on $v_{split} \rightsquigarrow \mu'$
  do if the path goes right at v
    then ReportInSubtree($lc(v)$, q_x)

ReportInSubtree(v,q_x)

if $(p(v))_x \leq q_x$
  then report $p(v)$
if v is not a leaf
  then ReportInSubtree($lc(v)$, q_x)
    ReportInSubtree($rc(v)$, q_x)

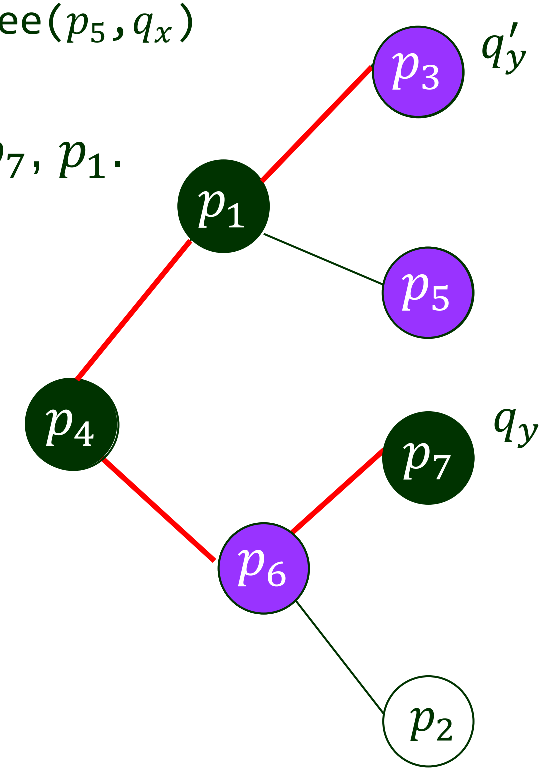
Example of Execution



ReportInSubtree(p_5,q_x)

Returns p_4, p_7, p_1.

$v_{split} = p_4$



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

ReportInSubtree(v,q_x)

if $(p(v))_x \leq q_x$
    then report $p(v)$
if v is not a leaf
    then ReportInSubtree($lc(v)$, q_x)
        ReportInSubtree($rc(v)$, q_x)

Running Time

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

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

if $\left(p(v)\right)_x \leq q_x$
    then report $p(v)$
    if v is not a leaf
        then ReportInSubtree($lc(v)$, q_x)
             ReportInSubtree($rc(v)$, q_x)

Time cost breaks down to two parts:

Running Time

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

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

if $\left(p(v)\right)_x \le q_x$
    then report $p(v)$
    if v is not a leaf
        then ReportInSubtree($lc(v)$, q_x)
            ReportInSubtree($rc(v)$, q_x)

Time cost breaks down to two parts:

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

Running Time

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

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

if $\big(p(v)\big)_x \leq q_x$
    then report $p(v)$
    if v is not a leaf
        then ReportInSubtree($lc(v)$, q_x)
            ReportInSubtree($rc(v)$, q_x)

Time cost breaks down to two parts:

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

Running Time

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

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

if $(p(v))_x \le q_x$
  then report $p(v)$
  if v is not a leaf
    then ReportInSubtree($lc(v)$, q_x)
      ReportInSubtree($rc(v)$, q_x)

Time cost breaks down to two parts:

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

Running Time

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

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

if $\big(p(v)\big)_x \leq q_x$
    then report $p(v)$
    if v is not a leaf
        then ReportInSubtree($lc(v)$, q_x)
             ReportInSubtree($rc(v)$, q_x)

Time cost breaks down to two parts:

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

\# reported points

Running Time

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

// r is the root of $\mathcal{T}$

search with q_y and q'_y in $\mathcal{T}$, ending at the nodes μ and μ'

let v_{split} be the node where the two paths split.

for each node v on the path $r \rightsquigarrow \mu$ or $v_{split} \rightsquigarrow \mu'$

    do if $p(v) \in (-\infty, q_x] \times [q_y, q'_y]$

        then report $p(v)$

for each node v on $v_{split} \rightsquigarrow \mu$

    do if the path goes left at v

        then `ReportInSubtree`$(rc(v),\ q_x)$

for each node v on $v_{split} \rightsquigarrow \mu'$

    do if the path goes right at v

        then `ReportInSubtree`$(lc(v),\ q_x)$

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

if $\big(p(v)\big)_x \le q_x$

    then report $p(v)$

    if v is not a leaf

        then `ReportInSubtree`$(lc(v),\ q_x)$

            `ReportInSubtree`$(rc(v),\ q_x)$

Time cost breaks down to two parts:

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

reported points

$$O(\log n + k)$$

Summary on PST

Min heap over the x-coordinate.

Binary search tree over the y-coordinate.

Storage	$O(n)$
Construction time	$O(n \log n)$
Query time	$O(\log n + k)$