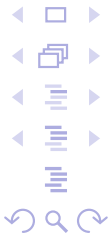
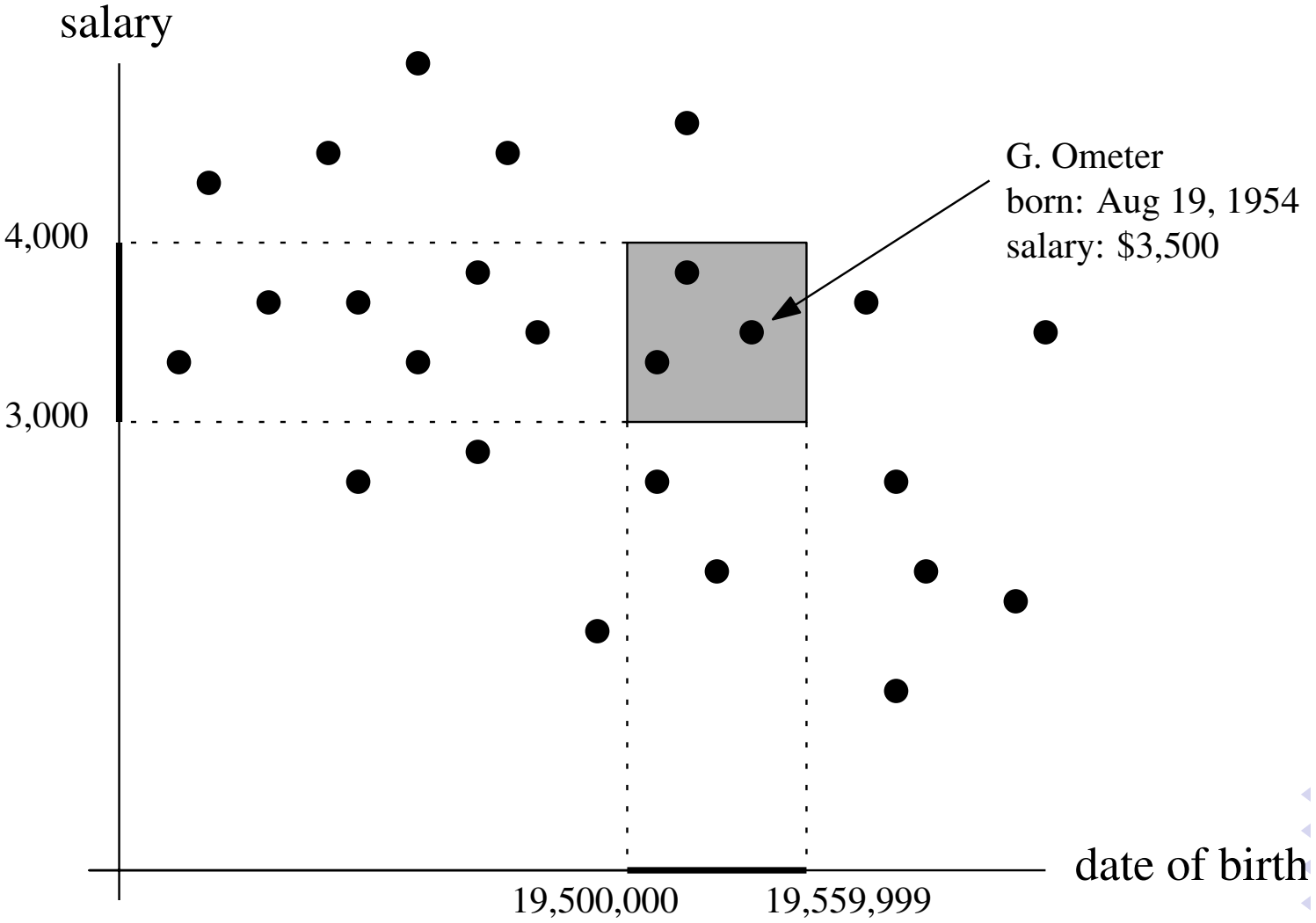
Orthogonal Range Searching

1397-2



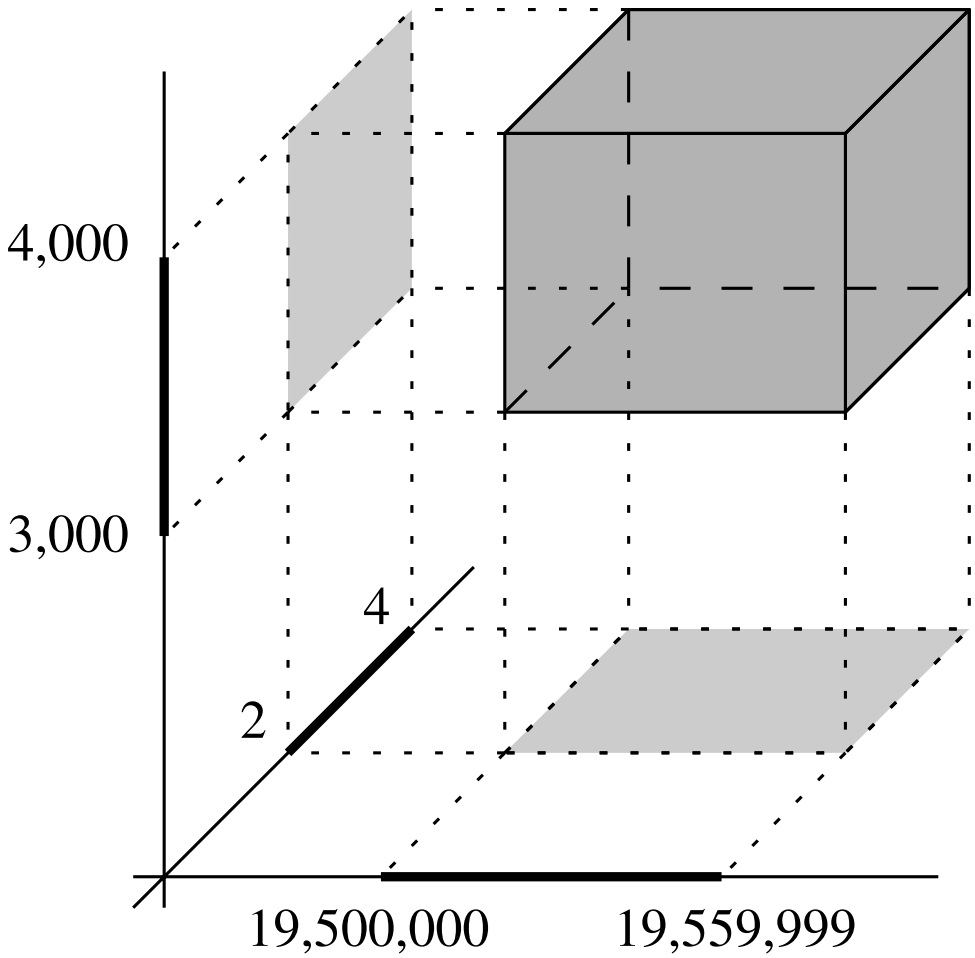
Motivation:

Querying a Database



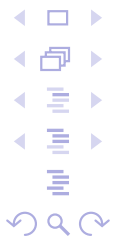
Motivation:

Querying a Database



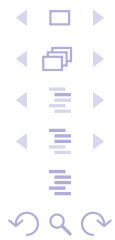
1D Range Searching

- Input: A set $P = \{p_1, p_2, \ldots, p_n\}$ of point on real line (1D).
- Query: Given the interval $[x, x']$, report all the point in the interval.
- Data Structure: Balanced Binary Search Tree



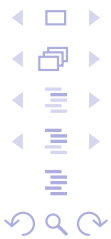
1D Range Searching

- Input: A set $P = \{p_1, p_2, \ldots, p_n\}$ of point on real line (1D).
- Query: Given the interval $[x, x']$, report all the point in the interval.
- Data Structure: Balanced Binary Search Tree



1D Range Searching

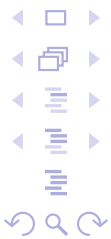
- Input: A set $P = \{p_1, p_2, \ldots, p_n\}$ of point on real line (1D).
- Query: Given the interval $[x, x']$, report all the point in the interval.
- Data Structure: Balanced Binary Search Tree



1D Range Searching

- Input: A set $P = \{p_1, p_2, \ldots, p_n\}$ of point on real line (1D).
- Query: Given the interval $[x, x']$, report all the point in the interval.
- Data Structure: Balanced Binary Search Tree

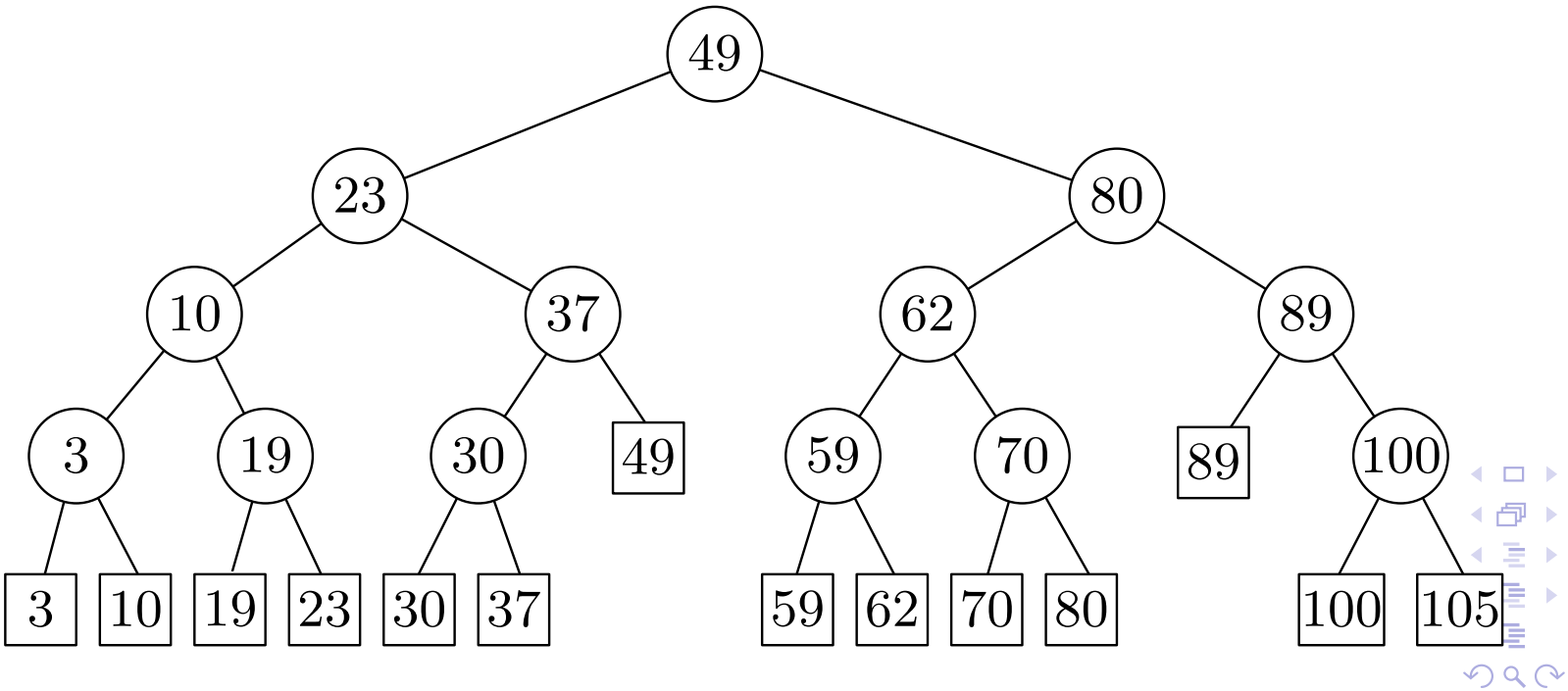
Example: $P = \{3, 10, 19, 23, 30, 37, 49, 59, 62, 70, 80, 89, 100, 105\}$



1D Range Searching

- Input: A set $P = \{p_1, p_2, \ldots, p_n\}$ of point on real line (1D).
- Query: Given the interval $[x, x']$, report all the point in the interval.
- Data Structure: Balanced Binary Search Tree

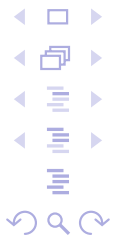
Example: $P = \{3, 10, 19, 23, 30, 37, 49, 59, 62, 70, 80, 89, 100, 105\}$



1D Range Searching

FINDSPLITNODE($\mathcal{T}, x, x'$)
Input. A tree $\mathcal{T}$ and two values x and x' with $x \leqslant x'$.
Output. The node v where the paths to x and x' split, or the leaf where both paths end.

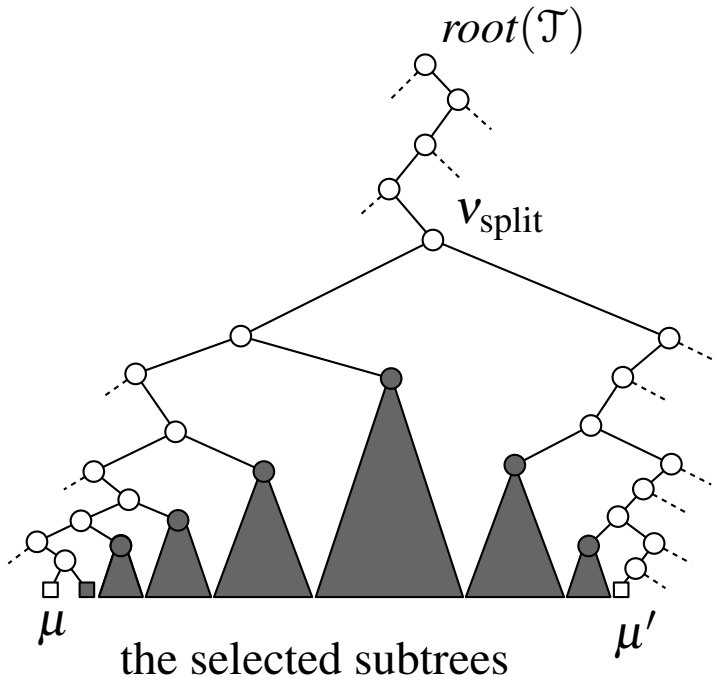
1. $v \leftarrow root(\mathcal{T})$
2. **while** v is not a leaf **and** ($x' \leqslant x_v$ **or** $x > x_v$)
3. **do if** $x' \leqslant x_v$
4. **then** $v \leftarrow lc(v)$
5. **else** $v \leftarrow rc(v)$
6. **return** v



1D Range Searching

FINDSPLITNODE($\mathcal{T}, x, x'$)
Input. A tree $\mathcal{T}$ and two values x and x' with $x \leqslant x'$.
Output. The node v where the paths to x and x' split, or the leaf where both paths end.

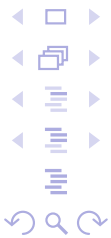
1. $v \leftarrow root(\mathcal{T})$
2. **while** v is not a leaf **and** ($x' \leqslant x_v$ **or** $x > x_v$)
3. **do if** $x' \leqslant x_v$
4. **then** $v \leftarrow lc(v)$
5. **else** $v \leftarrow rc(v)$
6. **return** v



1D Range Searching

Algorithm 1DRANGEQUERY($\mathcal{T}, [x : x']$)
Input. A binary search tree $\mathcal{T}$ and a range $[x : x']$.
Output. All points stored in $\mathcal{T}$ that lie in the range.

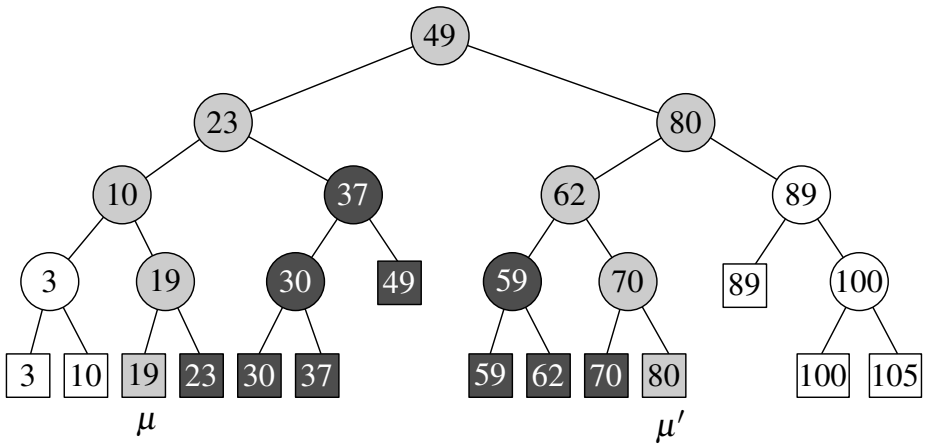
1. $v_{\text{split}} \leftarrow$FINDSPLITNODE($\mathcal{T}, x, x'$)
2. **if** v_{split} is a leaf
3. **then** Check if the point stored at v_{split} must be reported.
4. **else** ($*$ Follow the path to x and report the points in subtrees right of the path. $*$)
5. $v \leftarrow lc(v_{\text{split}})$
6. **while** v is not a leaf
7. **do if** $x \leqslant x_v$
8. **then** REPORTSUBTREE($rc(v)$)
9. $v \leftarrow lc(v)$
10. **else** $v \leftarrow rc(v)$
11. Check if the point stored at the leaf v must be reported.
12. Similarly, follow the path to x', report the points in subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.



1D Range Searching

Algorithm 1DRANGEQUERY($\mathcal{T}, [x : x']$)
Input. A binary search tree $\mathcal{T}$ and a range $[x : x']$.
Output. All points stored in $\mathcal{T}$ that lie in the range.

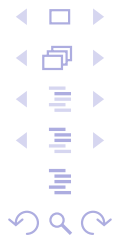
1. $v_{\text{split}} \leftarrow$FINDSPLITNODE($\mathcal{T}, x, x'$)
2. **if** v_{split} is a leaf
3. **then** Check if the point stored at v_{split} must be reported.
4. **else** ($*$ Follow the path to x and report the points in subtrees right of the path. $*$)
5. $v \leftarrow lc(v_{\text{split}})$
6. **while** v is not a leaf
7. **do if** $x \leqslant x_v$
8. **then** REPORTSUBTREE($rc(v)$)
9. $v \leftarrow lc(v)$
10. **else** $v \leftarrow rc(v)$
11. Check if the point stored at the leaf v must be reported.
12. Similarly, follow the path to x', report the points in subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.



1D Range Searching

Lemma 5.1

Algorithm 1DRANGEQUERY reports exactly those points that lie in the query range.

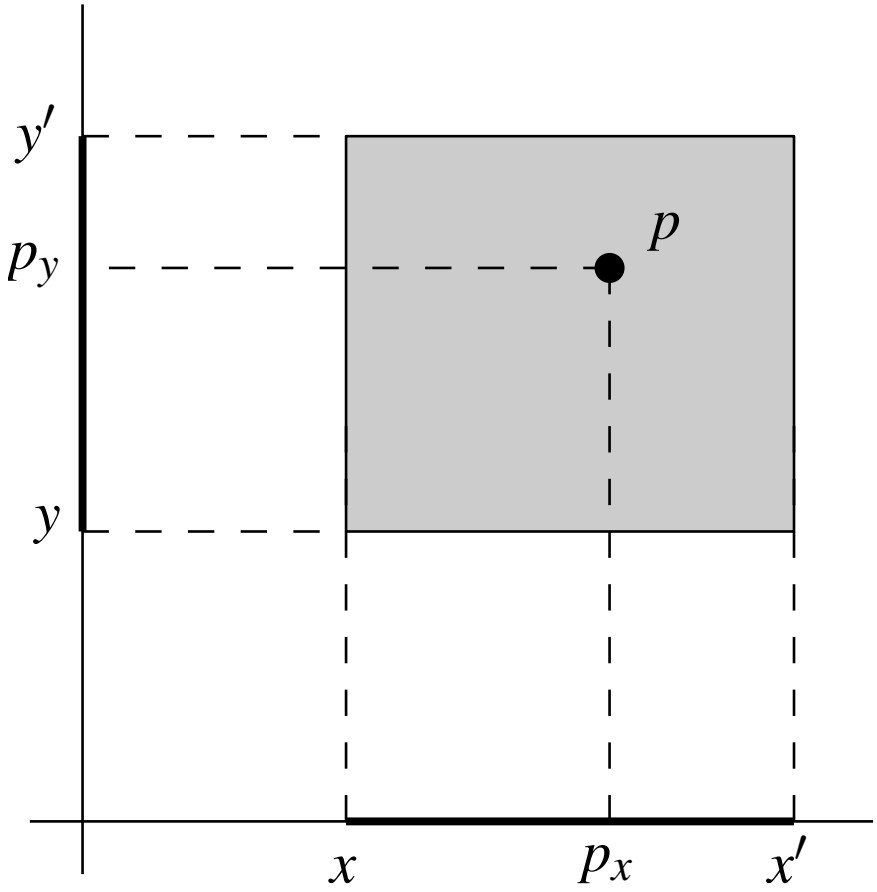


Lemma 5.1

Algorithm 1DRANGEQUERY reports exactly those points that lie in the query range.

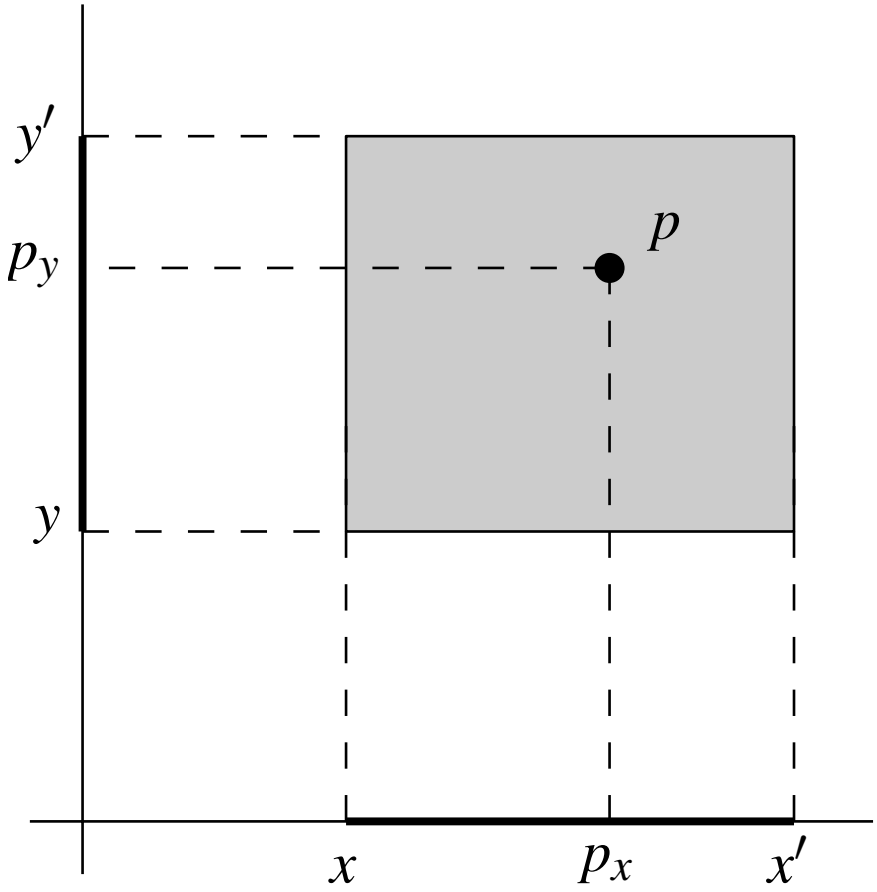
Theorem 5.2 *Let P be a set of n points in 1-dimensional space. The set P can be stored in a balanced binary search tree, which uses $O(n)$ storage and has $O(n \log n)$ construction time, such that the points in a query range can be reported in time $O(k + \log n)$, where k is the number of reported points.*

2D Range Searching



Can we generalize the 1-dimensional structure to 2-dimension?

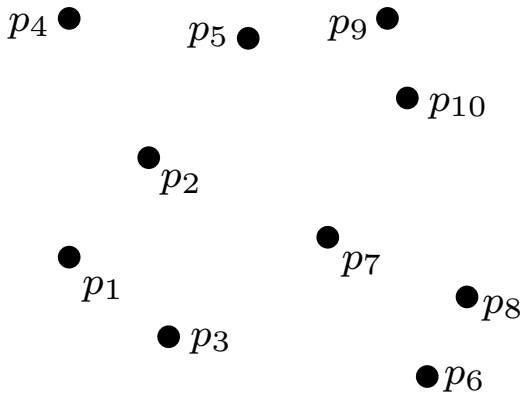
2D Range Searching



Can we generalize the 1-dimensional structure to 2-dimension?

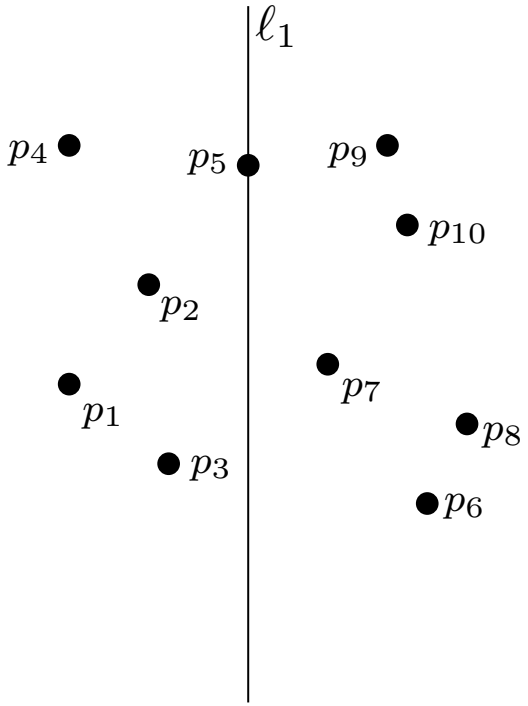
2D Range Searching

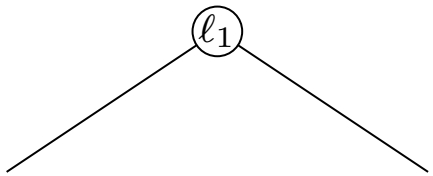
KD-tree



2D Range Searching

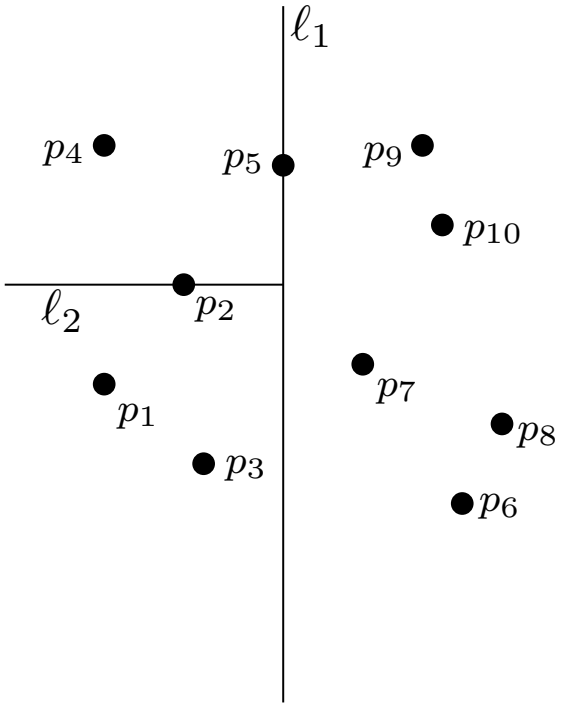
KD-tree

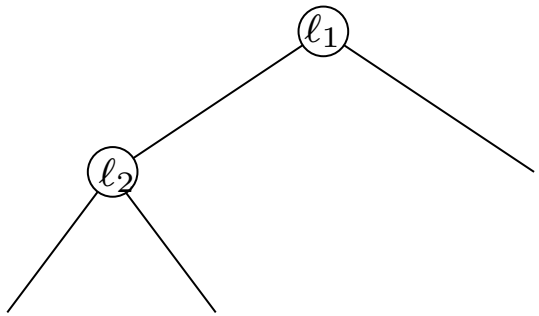




2D Range Searching

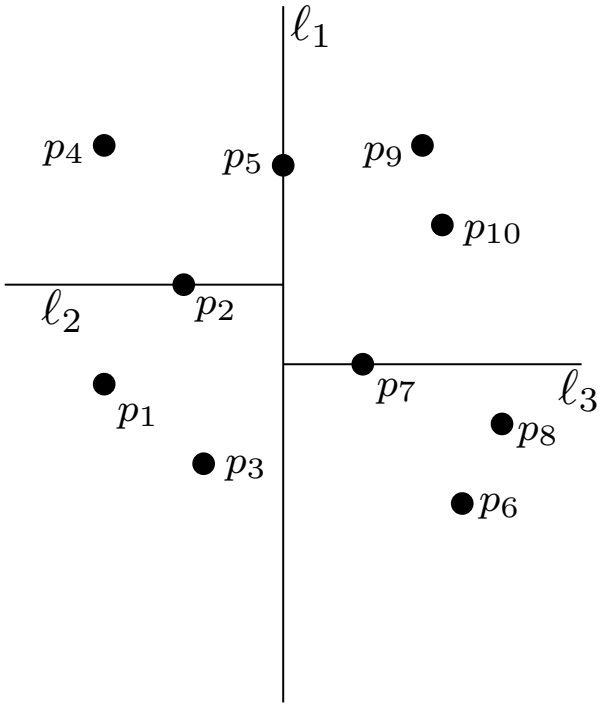
KD-tree

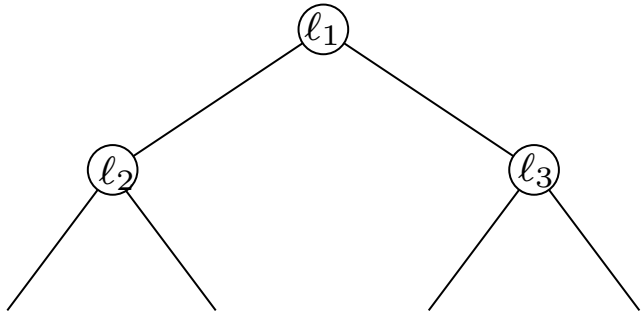




2D Range Searching

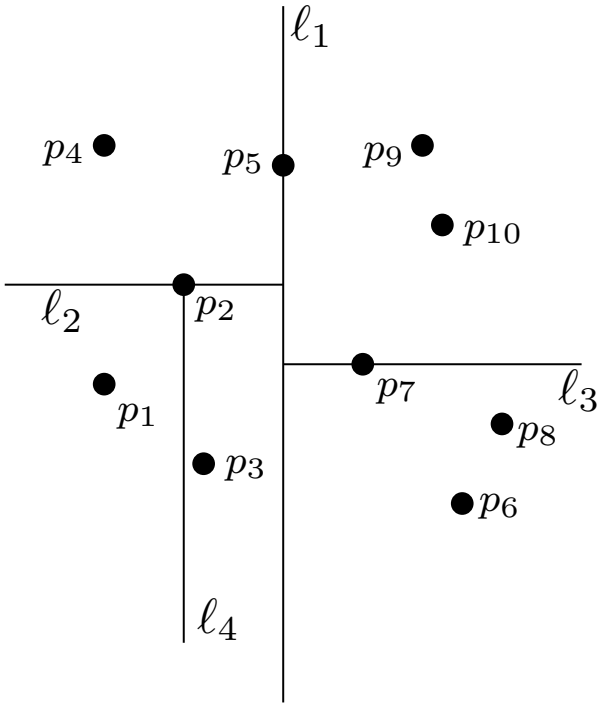
KD-tree

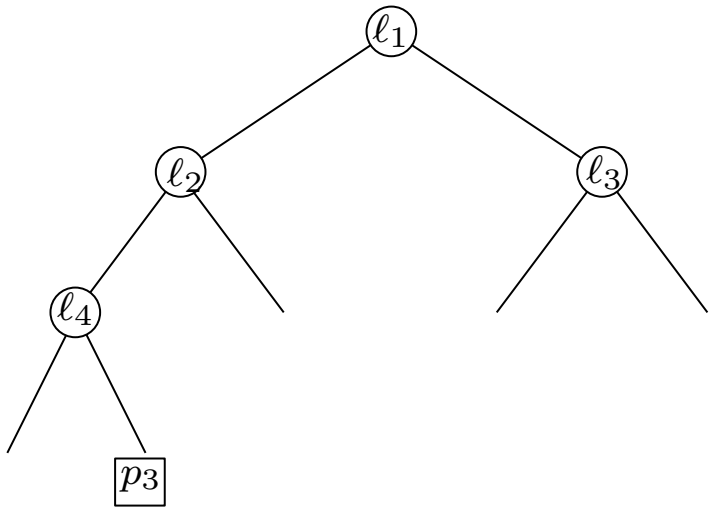




2D Range Searching

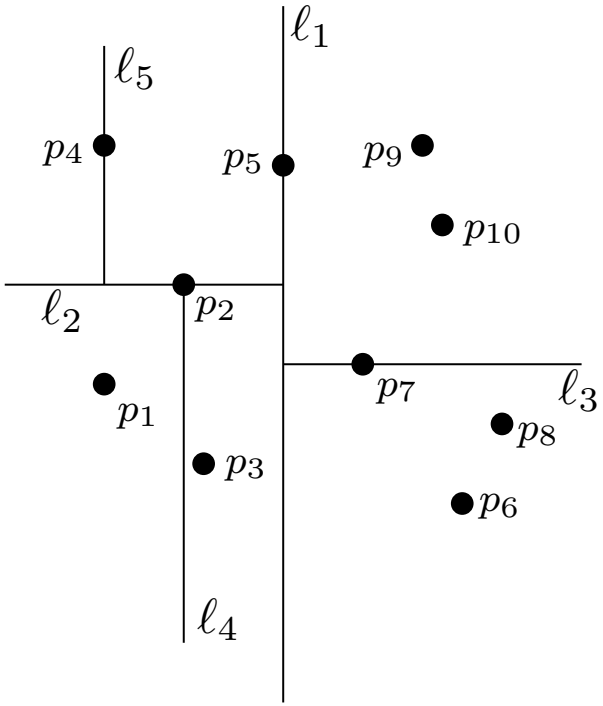
KD-tree

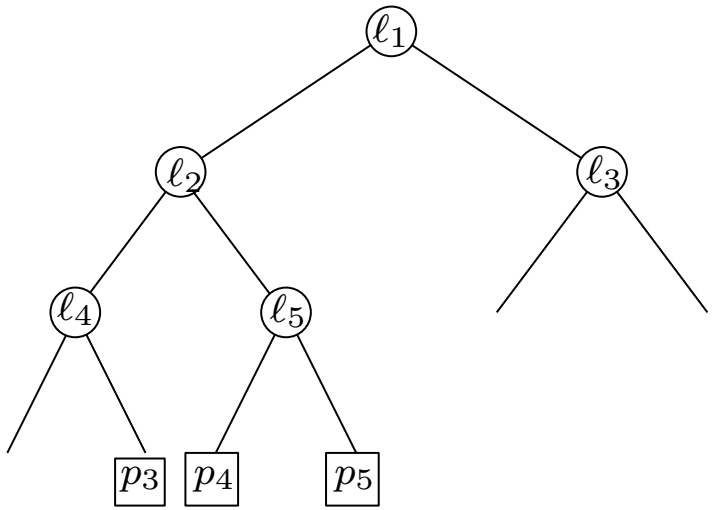




2D Range Searching

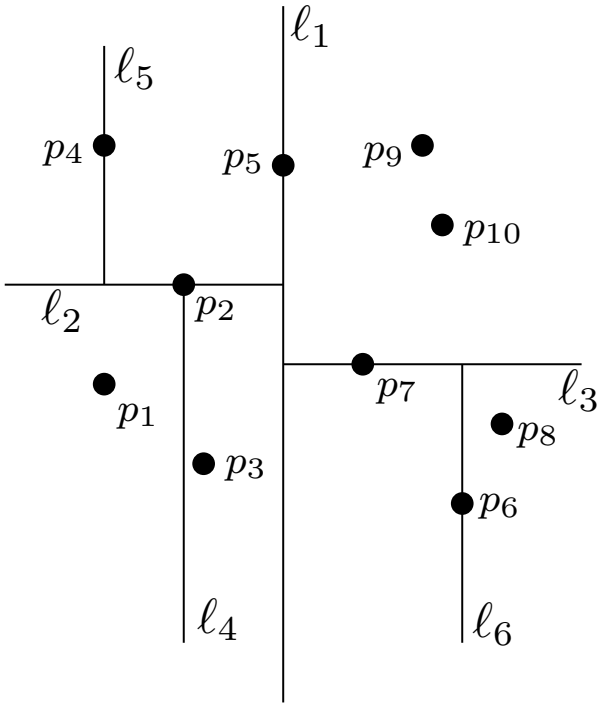
KD-tree

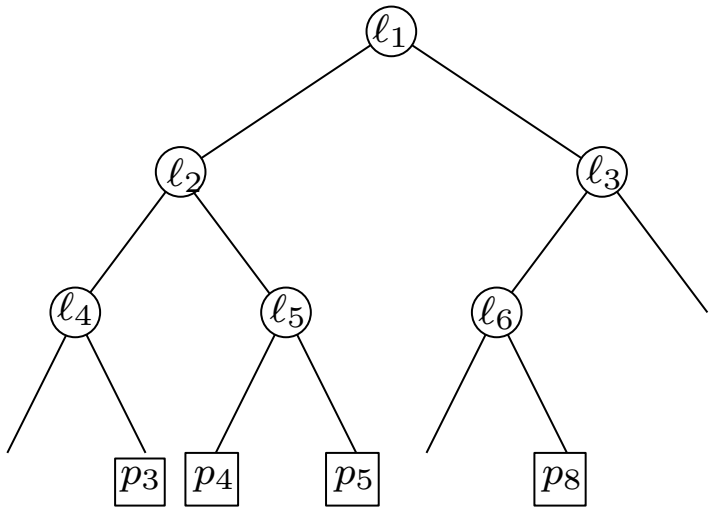




2D Range Searching

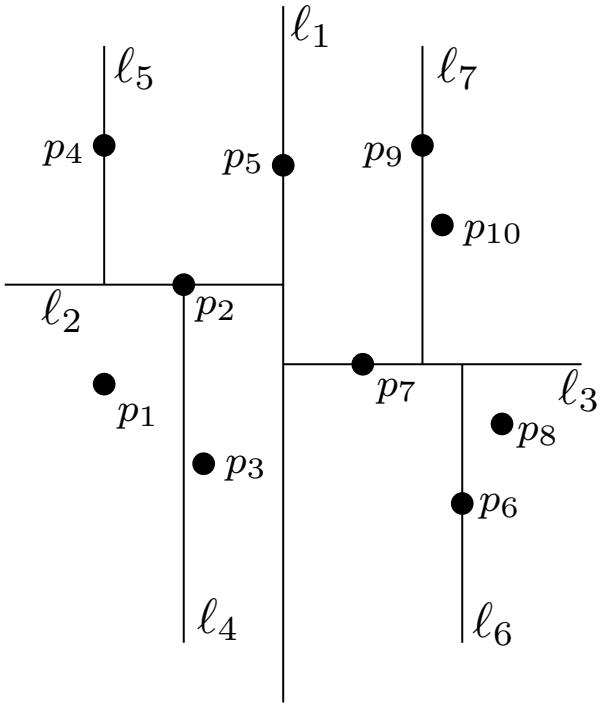
KD-tree

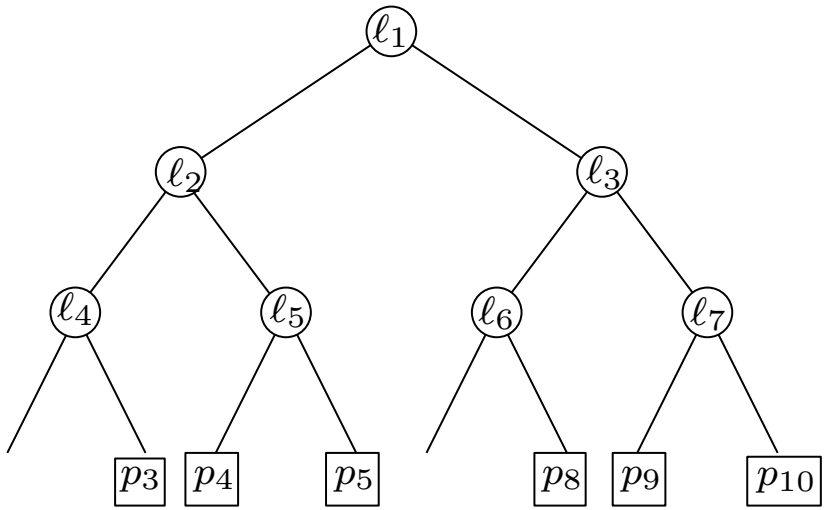




2D Range Searching

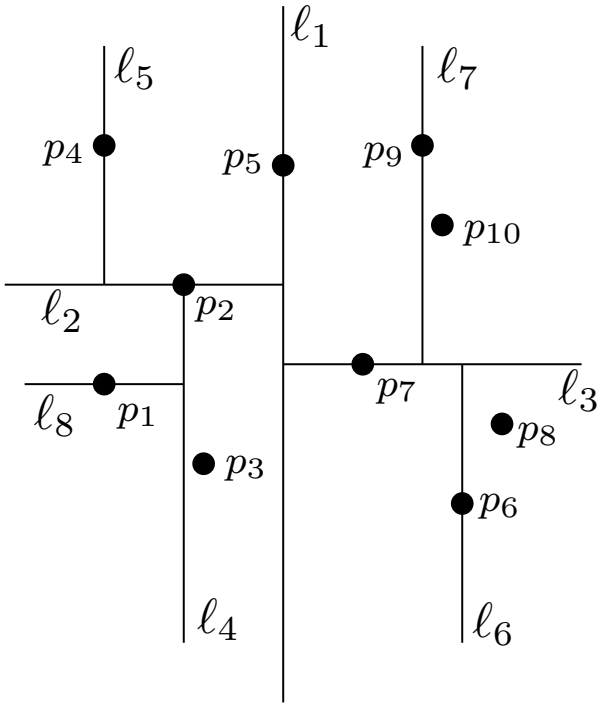
KD-tree

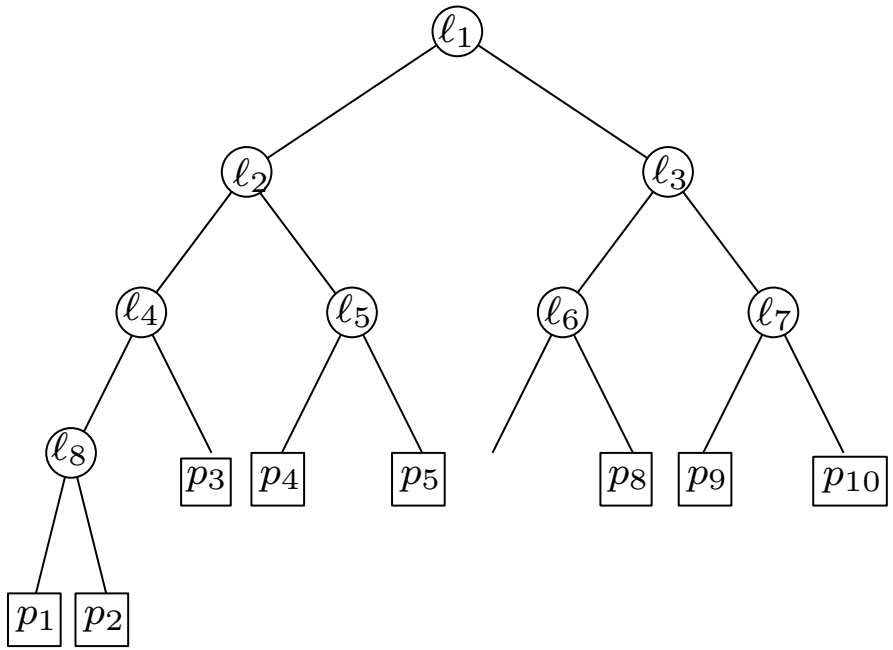




2D Range Searching

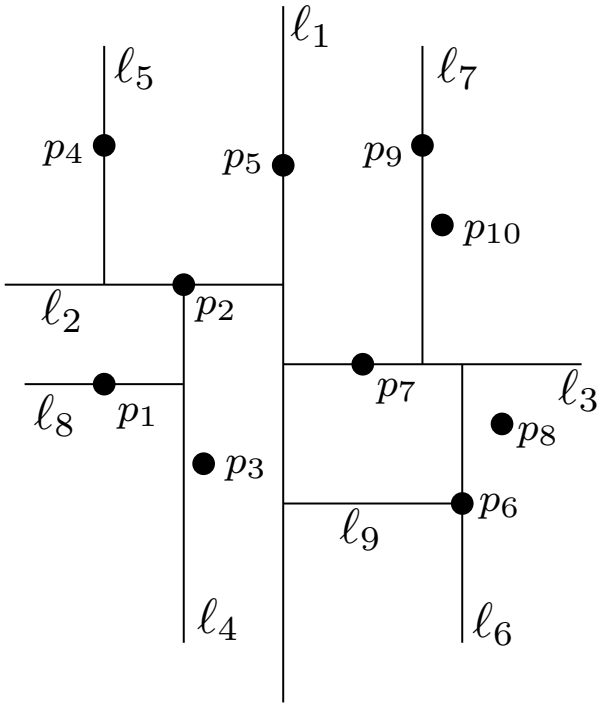
KD-tree

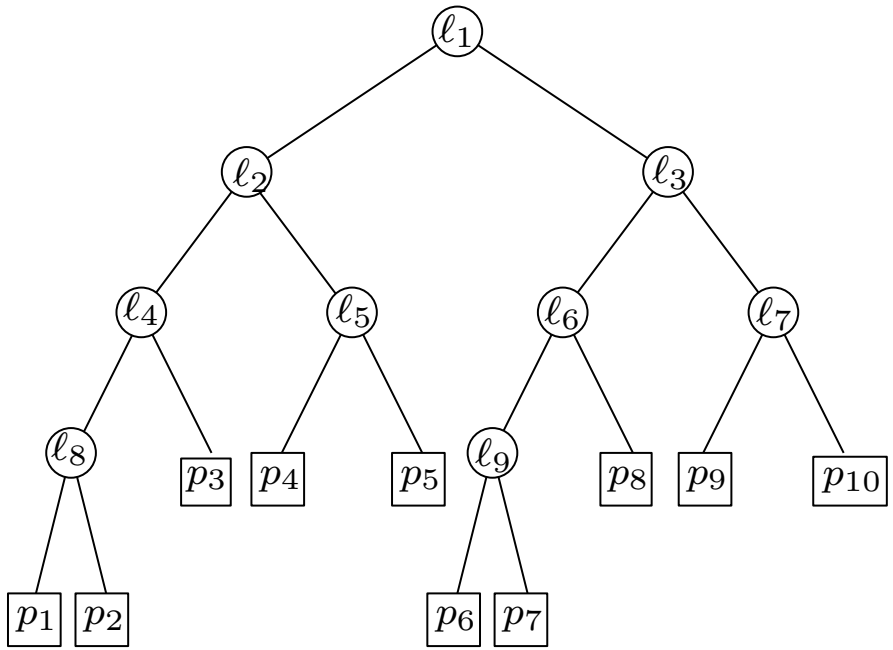




2D Range Searching

KD-tree





Algorithm BUILDKDTREE(P, *depth*)
Input. A set of points P and the current depth *depth*.
Output. The root of a kd-tree storing P.

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** *depth* is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of ℓ or on ℓ, and let P_2 be the set of points to the right of ℓ.
5. **else** Split P into two subsets with a horizontal line ℓ through the median y-coordinate of the points in P. Let P_1 be the set of points below ℓ or on ℓ, and let P_2 be the set of points above ℓ.
6. $v_{\text{left}} \leftarrow$ BUILDKDTREE(P_1, *depth* $+ 1$)
7. $v_{\text{right}} \leftarrow$ BUILDKDTREE(P_2, *depth* $+ 1$)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v

Algorithm BUILDKDTREE($P, depth$)
Input. A set of points P and the current depth $depth$.
Output. The root of a kd-tree storing P.

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** $depth$ is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of ℓ or on ℓ, and let P_2 be the set of points to the right of ℓ.
5. **else** Split P into two subsets with a horizontal line ℓ through the median y-coordinate of the points in P. Let P_1 be the set of points below ℓ or on ℓ, and let P_2 be the set of points above ℓ.
6. $v_{\text{left}} \leftarrow$ BUILDKDTREE($P_1, depth+1$)
7. $v_{\text{right}} \leftarrow$ BUILDKDTREE($P_2, depth+1$)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v

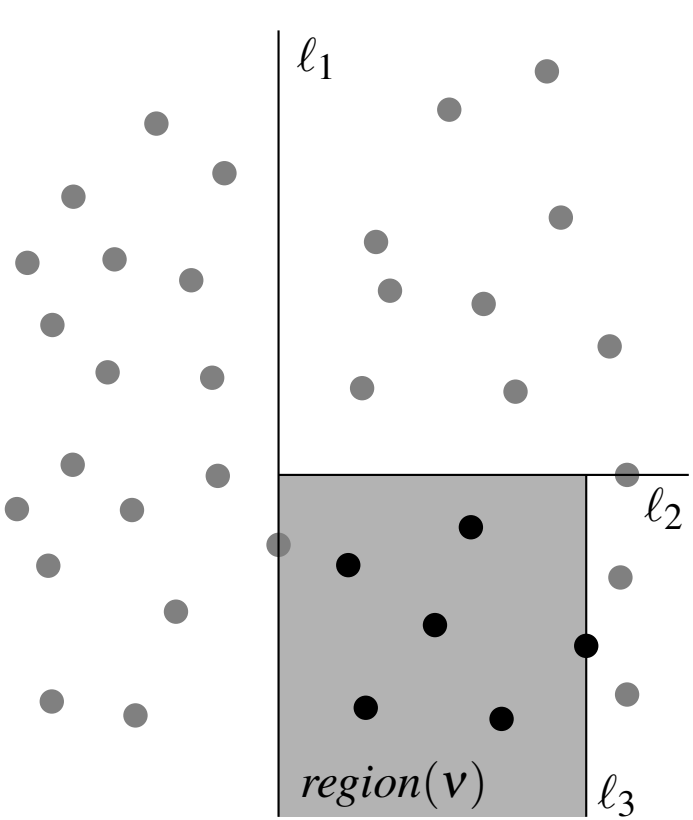
Time complexity: $T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ O(n) + 2T(\frac{n}{2}) & \text{if } n > 1 \end{cases}$

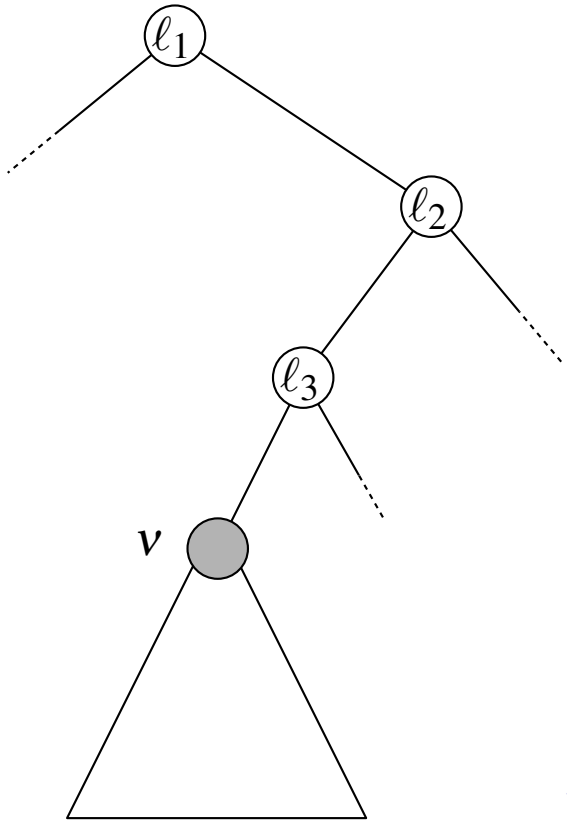
Algorithm BUILDKDTREE($P, depth$)
Input. A set of points P and the current depth $depth$.
Output. The root of a kd-tree storing P.

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** $depth$ is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of ℓ or on ℓ, and let P_2 be the set of points to the right of ℓ.
5. **else** Split P into two subsets with a horizontal line ℓ through the median y-coordinate of the points in P. Let P_1 be the set of points below ℓ or on ℓ, and let P_2 be the set of points above ℓ.
6. $v_{\text{left}} \leftarrow$ BUILDKDTREE($P_1, depth+1$)
7. $v_{\text{right}} \leftarrow$ BUILDKDTREE($P_2, depth+1$)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v

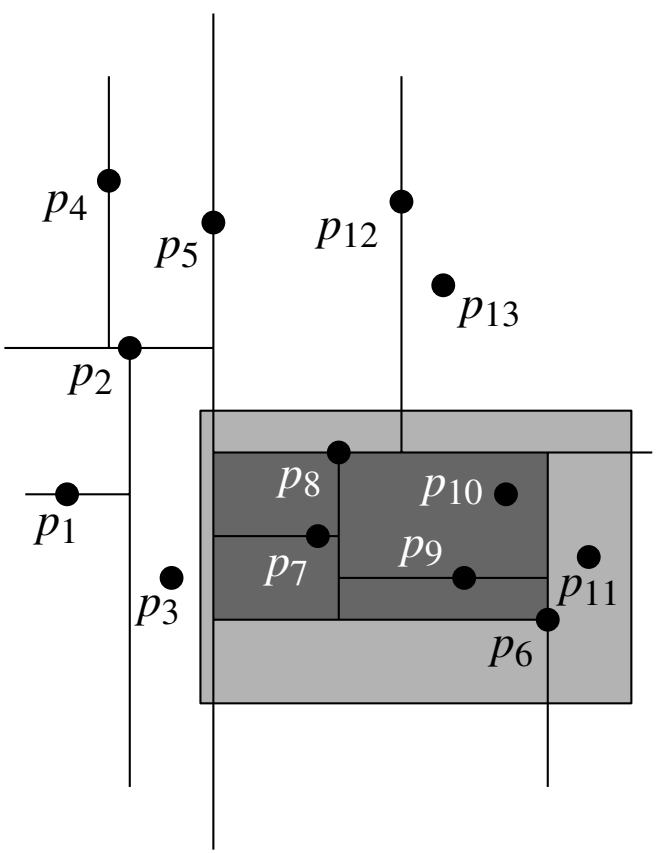
Lemma 5.3 *A kd-tree for a set of n points uses $O(n)$ storage and can be constructed in $O(n \log n)$ time.*

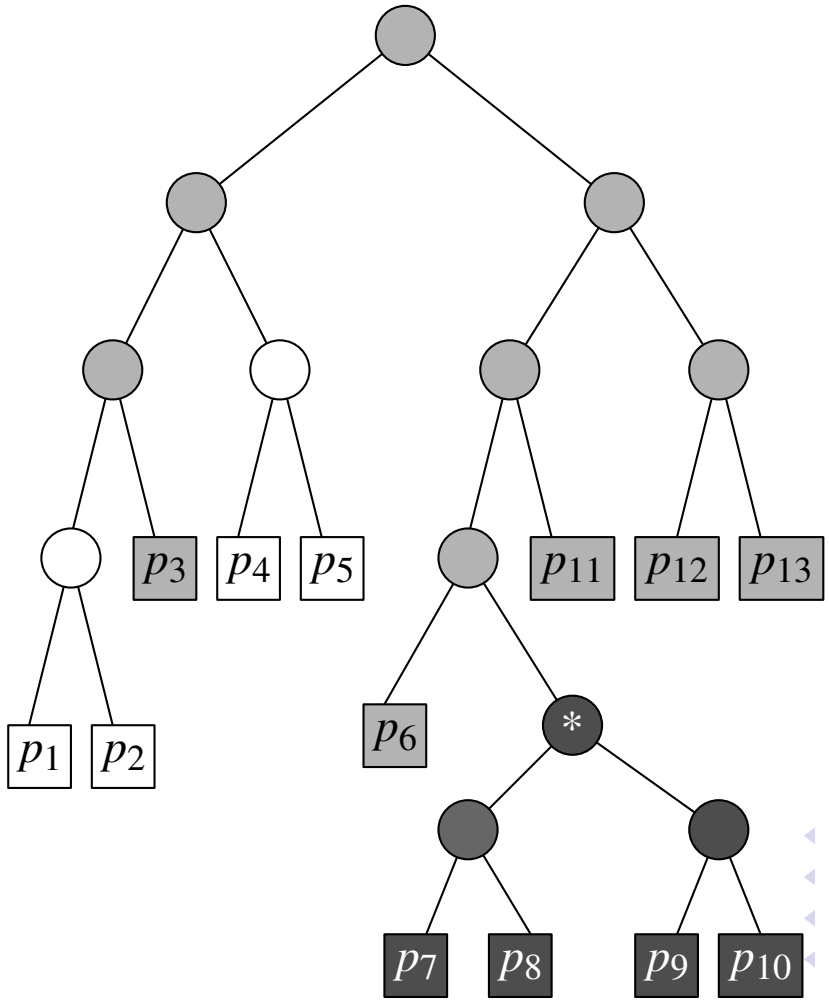
Nodes in the tree vs. Regions in the plane





Query on kd-tree





Query on kd-tree

Algorithm SEARCHKDTREE(v, R)
Input. The root of (a subtree of) a kd-tree, and a range R.
Output. All points at leaves below v that lie in the range.

```
1.  if v is a leaf
2.     then Report the point stored at v if it lies in R.
3.     else  if region(lc(v)) is fully contained in R
4.              then REPORTSUBTREE(lc(v))
5.              else  if region(lc(v)) intersects R
6.                       then SEARCHKDTREE(lc(v), R)
7.           if region(rc(v)) is fully contained in R
8.              then REPORTSUBTREE(rc(v))
9.              else  if region(rc(v)) intersects R
10.                      then SEARCHKDTREE(rc(v), R)
```

How to test whether the query range R intersects the region corresponding to a node?

Solution 1: Store region of each node in the node.

Solution 2:

How to test whether the query range R intersects the region corresponding to a node?

Solution 1: Store region of each node in the node.

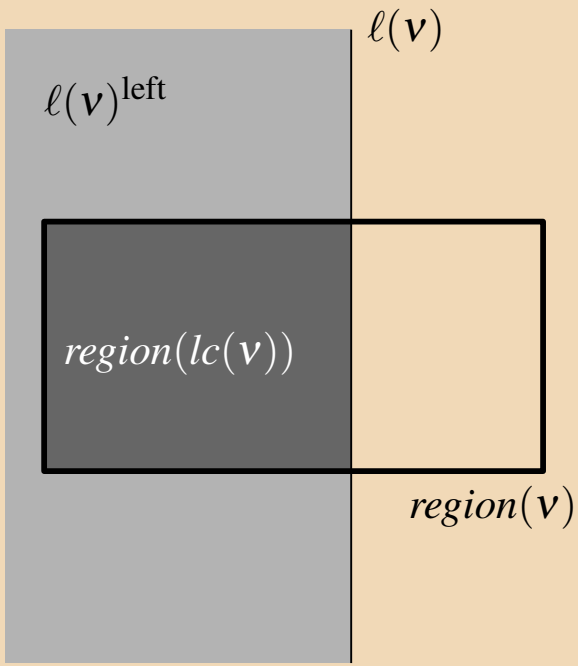
Solution 2:

Yazd Univ.

How to test whether the query range R intersects the region corresponding to a node?

Solution 1: Store region of each node in the node.

Solution 2:



$$region(lc(v)) = region(v) \cap \ell(v)^{\text{left}},$$

Lemma 5.4 *A query with an axis-parallel rectangle in a kd-tree storing n points can be performed in $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

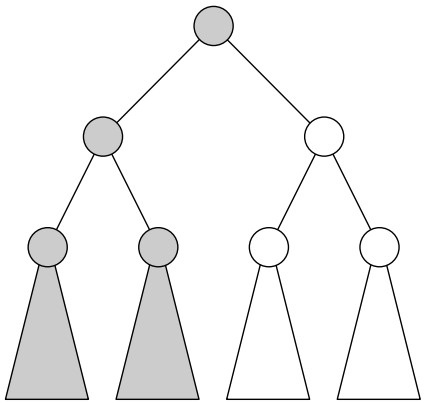
Proof.

- traversing subtrees and reporting points: Linear time.

Lemma 5.4 *A query with an axis-parallel rectangle in a kd-tree storing n points can be performed in $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

Proof.

- traversing subtrees and reporting points: Linear time.

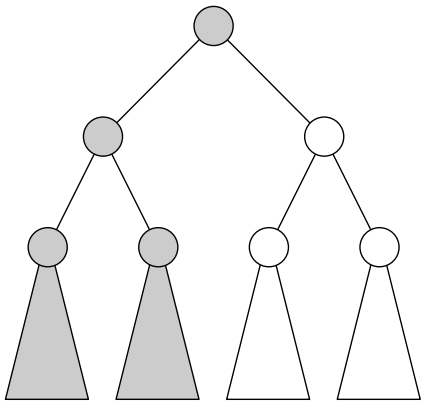


$$Q(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2 + 2Q(\frac{n}{4}) & \text{if } n > 1 \end{cases}$$

Lemma 5.4 *A query with an axis-parallel rectangle in a kd-tree storing n points can be performed in $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

Proof.

- traversing subtrees and reporting points: Linear time.



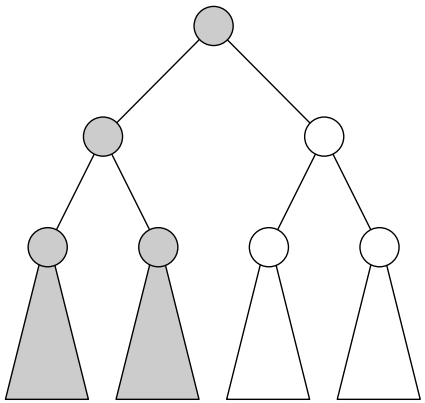
$$Q(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2 + 2Q(\frac{n}{4}) & \text{if } n > 1 \end{cases}$$

$$Q(n) = O(\sqrt{n})$$

Lemma 5.4 *A query with an axis-parallel rectangle in a kd-tree storing n points can be performed in $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

Proof.

- traversing subtrees and reporting points: Linear time.



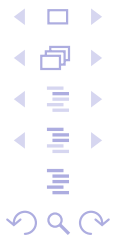
$$Q(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2 + 2Q(\frac{n}{4}) & \text{if } n > 1 \end{cases}$$

Theorem 5.5 *A kd-tree for a set P of n points in the plane uses $O(n)$ storage and can be built in $O(n \log n)$ time. A rectangular range query on the kd-tree takes $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

Theorem 5.5 *A kd-tree for a set P of n points in the plane uses $O(n)$ storage and can be built in $O(n \log n)$ time. A rectangular range query on the kd-tree takes $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

Can we generalize the results to d-dimension?

Yes.
Time complexity: $\mathcal{O}(n \log n)$ construction time, $\mathcal{O}(n)$ space, and $\mathcal{O}(n^{1-1/d}+k)$ query time



Theorem 5.5 *A kd-tree for a set P of n points in the plane uses $O(n)$ storage and can be built in $O(n \log n)$ time. A rectangular range query on the kd-tree takes $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

Can we generalize the results to d-dimension?

Yes.

Time complexity: $\mathcal{O}(n \log n)$ construction time, $\mathcal{O}(n)$ space, and $\mathcal{O}(n^{1-1/d}+k)$ query time

Theorem 5.5 *A kd-tree for a set P of n points in the plane uses $O(n)$ storage and can be built in $O(n \log n)$ time. A rectangular range query on the kd-tree takes $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

Can we generalize the results to d-dimension?

Yes.

Time complexity: $\mathcal{O}(n \log n)$ construction time, $\mathcal{O}(n)$ space, and $\mathcal{O}(n^{1-1/d}+k)$ query time

Theorem 5.5 *A kd-tree for a set P of n points in the plane uses $O(n)$ storage and can be built in $O(n \log n)$ time. A rectangular range query on the kd-tree takes $O(\sqrt{n}+k)$ time, where k is the number of reported points.*

Can we generalize the results to d-dimension?

Yes.

Time complexity: $\mathcal{O}(n \log n)$ construction time, $\mathcal{O}(n)$ space, and $\mathcal{O}(n^{1-1/d}+k)$ query time

Question:

Can we do the range searching faster?

Answer: Yes, but by increasing the storage complexity.

Time Complexity: $\mathcal{O}(\log^2 n + k)$

Space Complexity: $\mathcal{O}(n \log n)$.

Question:

Can we do the range searching faster?

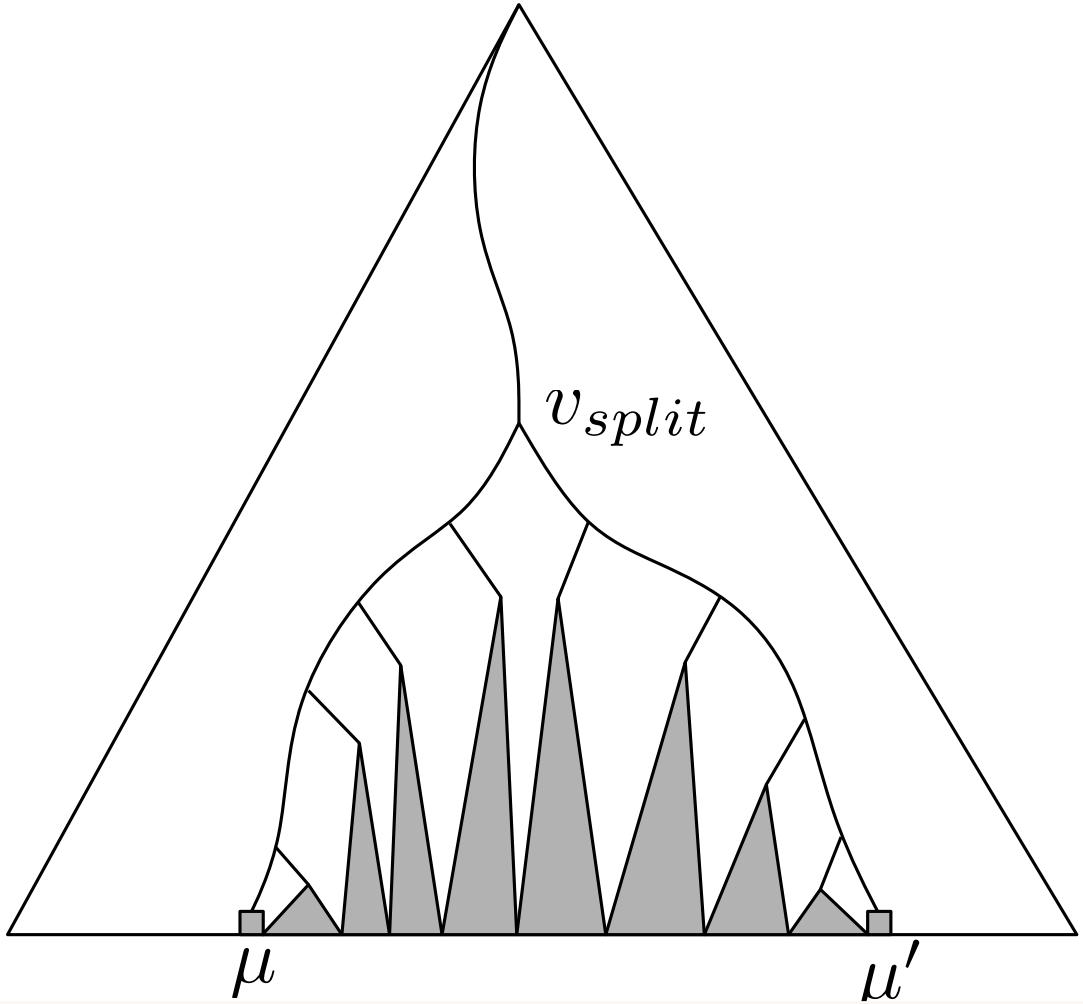
Answer: Yes, but by increasing the storage complexity.

Time Complexity: $\mathcal{O}(\log^2 n + k)$

Space Complexity: $\mathcal{O}(n \log n)$.

2D Range Searching

Range Trees

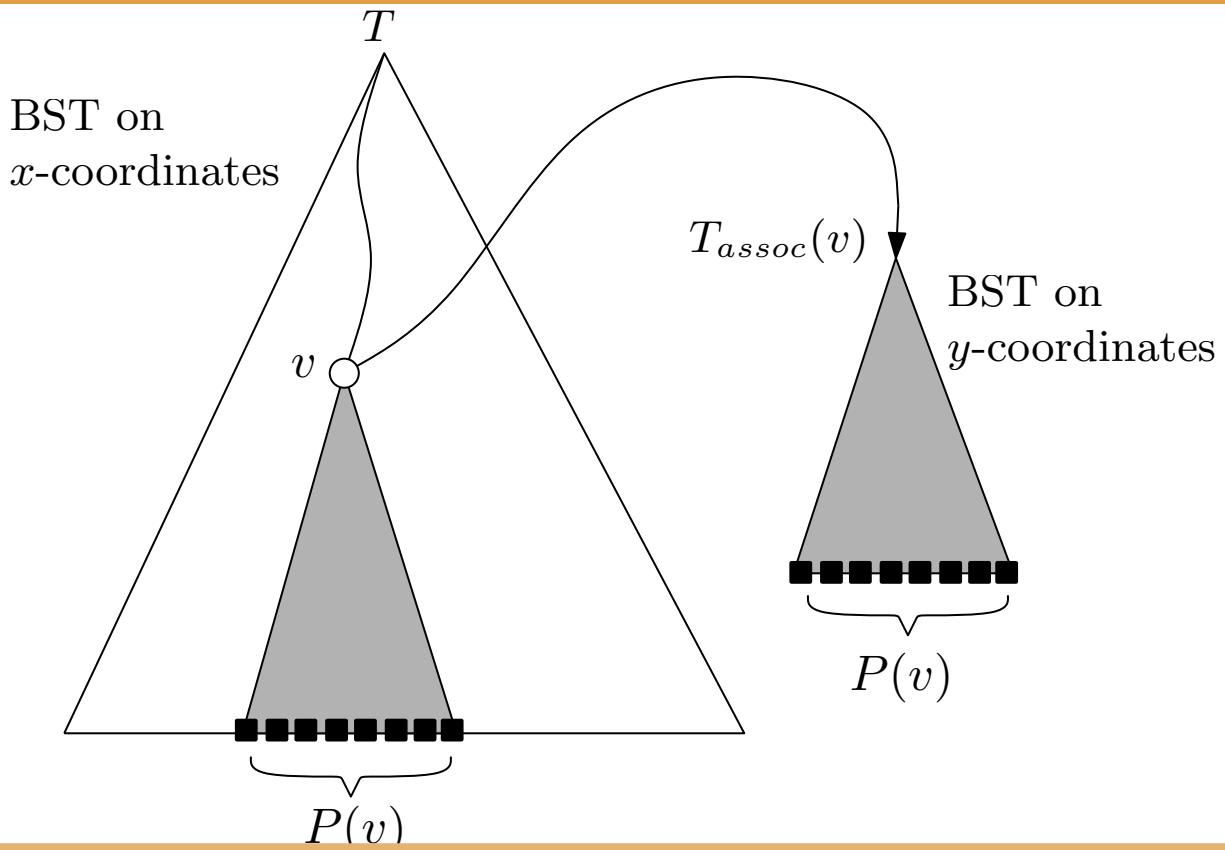


Range tree

- A 2-layered data structure
- Main tree (layer 1): BST on x-coordinates

2D Range Searching

Range Trees

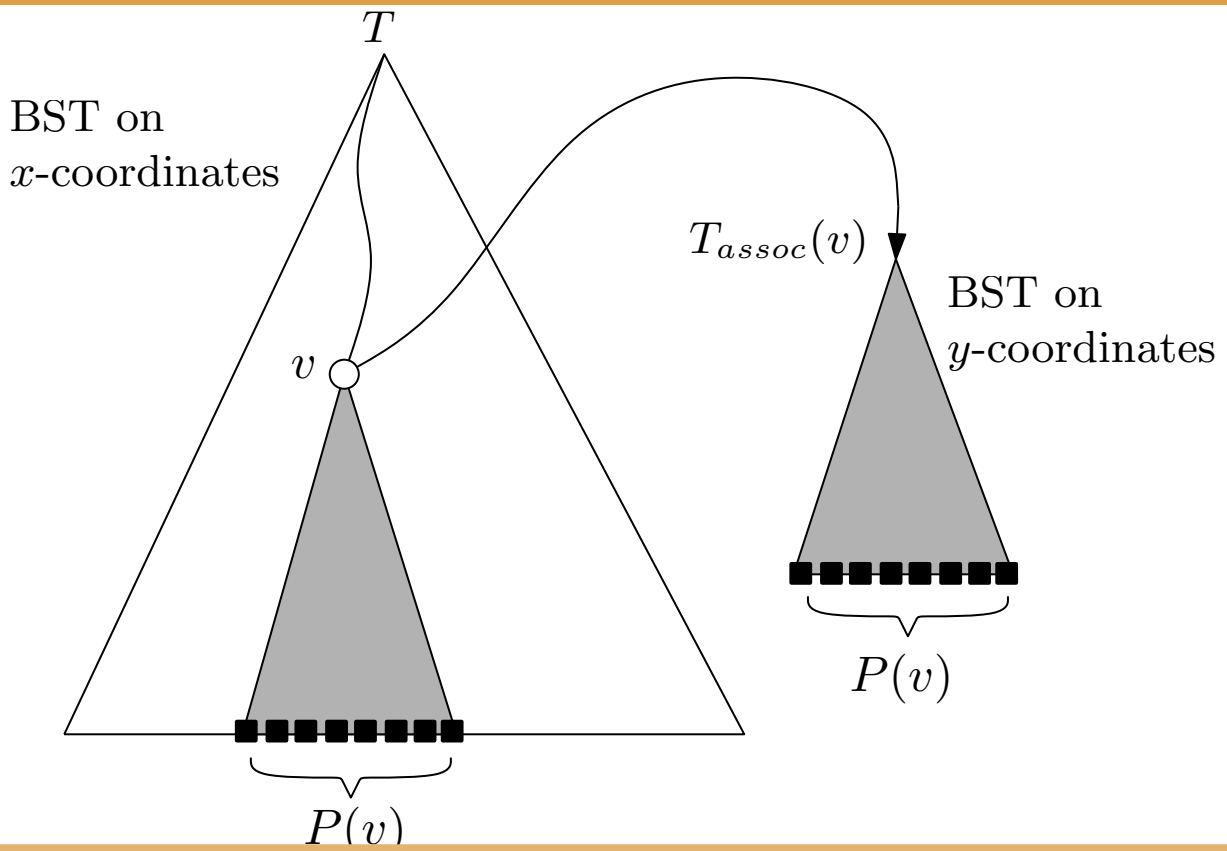


Range tree

- A 2-layered data structure
- Main tree (layer 1): BST on x-coordinates
- Layer 2: For any internal and leaf node: a BST on $P(v)$

2D Range Searching

Range Trees



Range tree

- A 2-layered data structure
- Main tree (layer 1): BST on x-coordinates
- Layer 2: For any internal and leaf node: a BST on $P(v)$

Algorithm BUILD2DRANGETREE(P)
Input. A set P of points in the plane.
Output. The root of a 2-dimensional range tree.

1. Construct the associated structure: Build a binary search tree $\mathcal{T}_{\text{assoc}}$ on the set P_y of y-coordinates of the points in P. Store at the leaves of $\mathcal{T}_{\text{assoc}}$ not just the y-coordinate of the points in P_y, but the points themselves.
2. **if** P contains only one point
3. **then** Create a leaf v storing this point, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of v.
4. **else** Split P into two subsets; one subset P_{left} contains the points with x-coordinate less than or equal to x_{mid}, the median x-coordinate, and the other subset P_{right} contains the points with x-coordinate larger than x_{mid}.
5. $v_{\text{left}} \leftarrow$ BUILD2DRANGETREE(P_{left})
6. $v_{\text{right}} \leftarrow$ BUILD2DRANGETREE(P_{right})
7. Create a node v storing x_{mid}, make v_{left} the left child of v, make v_{right} the right child of v, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of v.
8. **return** v

Algorithm BUILD2DRANGETREE(P)
Input. A set P of points in the plane.
Output. The root of a 2-dimensional range tree.

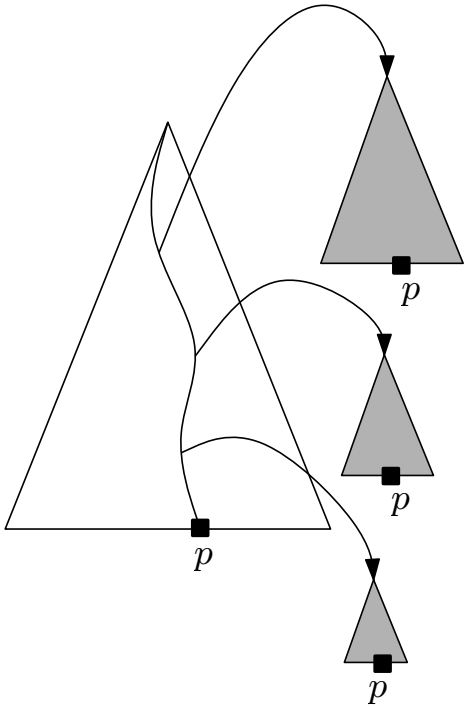
1. Construct the associated structure: Build a binary search tree $\mathcal{T}_{\text{assoc}}$ on the set P_y of y-coordinates of the points in P. Store at the leaves of $\mathcal{T}_{\text{assoc}}$ not just the y-coordinate of the points in P_y, but the points themselves.
2. **if** P contains only one point
3. **then** Create a leaf v storing this point, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of v.
4. **else** Split P into two subsets; one subset P_{left} contains the points with x-coordinate less than or equal to x_{mid}, the median x-coordinate, and the other subset P_{right} contains the points with x-coordinate larger than x_{mid}.
5. $v_{\text{left}} \leftarrow$ BUILD2DRANGETREE(P_{left})
6. $v_{\text{right}} \leftarrow$ BUILD2DRANGETREE(P_{right})
7. Create a node v storing x_{mid}, make v_{left} the left child of v, make v_{right} the right child of v, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of v.
8. **return** v

Time Complexity: $\mathcal{O}(n \log n)$.

Space Complexity:

Lemma 5.6 *A range tree on a set of n points in the plane requires $O(n \log n)$ storage.*

Proof. A point p in P is stored only in the associated structure of nodes on the path in $\mathcal{T}$ towards the leaf containing p. Hence, for all nodes at a given depth of $\mathcal{T}$, the point p is stored in exactly one associated structure. Because 1-dimensional range trees use linear storage it follows that the associated structures of all nodes at any depth of $\mathcal{T}$ together use $O(n)$ storage. The depth of $\mathcal{T}$ is $O(\log n)$. Hence, the total amount of storage required is bounded by $O(n \log n)$. □



1D Range Searching:

Algorithm 1DRANGEQUERY($\mathcal{T}, [x : x']$)
Input. A binary search tree $\mathcal{T}$ and a range $[x : x']$.
Output. All points stored in $\mathcal{T}$ that lie in the range.

1. $v_{\text{split}} \leftarrow$FINDSPLITNODE($\mathcal{T}, x, x'$)
2. **if** v_{split} is a leaf
3. **then** Check if the point stored at v_{split} must be reported.
4. **else** ($*$ Follow the path to x and report the points in subtrees right of the path. $*$)
5. $v \leftarrow lc(v_{\text{split}})$
6. **while** v is not a leaf
7. **do if** $x \leqslant x_v$
8. **then** REPORTSUBTREE($rc(v)$)
9. $v \leftarrow lc(v)$
10. **else** $v \leftarrow rc(v)$
11. Check if the point stored at the leaf v must be reported.
12. Similarly, follow the path to x', report the points in subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.

2D Range Searching:

Algorithm 2DRANGEQUERY($\mathcal{T}, [x : x'] \times [y : y']$)
Input. A 2-dimensional range tree $\mathcal{T}$ and a range $[x : x'] \times [y : y']$.
Output. All points in $\mathcal{T}$ that lie in the range.

1. $v_{\text{split}} \leftarrow$FINDSPLITNODE($\mathcal{T}, x, x'$)
2. **if** v_{split} is a leaf
3. **then** Check if the point stored at v_{split} must be reported.
4. **else** ($*$ Follow the path to x and call 1DRANGEQUERY on the subtrees right of the path. $*$)
5. $v \leftarrow lc(v_{\text{split}})$
6. **while** v is not a leaf
7. **do if** $x \leqslant x_v$
8. **then** 1DRANGEQUERY($\mathcal{T}_{\text{assoc}}(rc(v)), [y : y']$)
9. $v \leftarrow lc(v)$
10. **else** $v \leftarrow rc(v)$
11. Check if the point stored at v must be reported.
12. Similarly, follow the path from $rc(v_{\text{split}})$ to x', call 1DRANGEQUERY with the range $[y : y']$ on the associated structures of subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.

Lemma 5.7 *A query with an axis-parallel rectangle in a range tree storing n points takes $O(\log^2 n + k)$ time, where k is the number of reported points.*

Proof. At each node v in the main tree $\mathcal{T}$ we spend constant time to decide where the search path continues, and we possibly call 1DRANGEQUERY. Theorem 5.2 states that the time we spend in this recursive call is $O(\log n + k_v)$, where k_v is the number of points reported in this call. Hence, the total time we spend is

$$\sum_v O(\log n + k_v),$$

where the summation is over all nodes in the main tree $\mathcal{T}$ that are visited. Notice that the sum $\sum_v k_v$ equals k, the total number of reported points. Furthermore, the search paths of x and x' in the main tree $\mathcal{T}$ have length $O(\log n)$. Hence, $\sum_v O(\log n) = O(\log^2 n)$. The lemma follows. □

Lemma 5.7 *A query with an axis-parallel rectangle in a range tree storing n points takes $O(\log^2 n + k)$ time, where k is the number of reported points.*

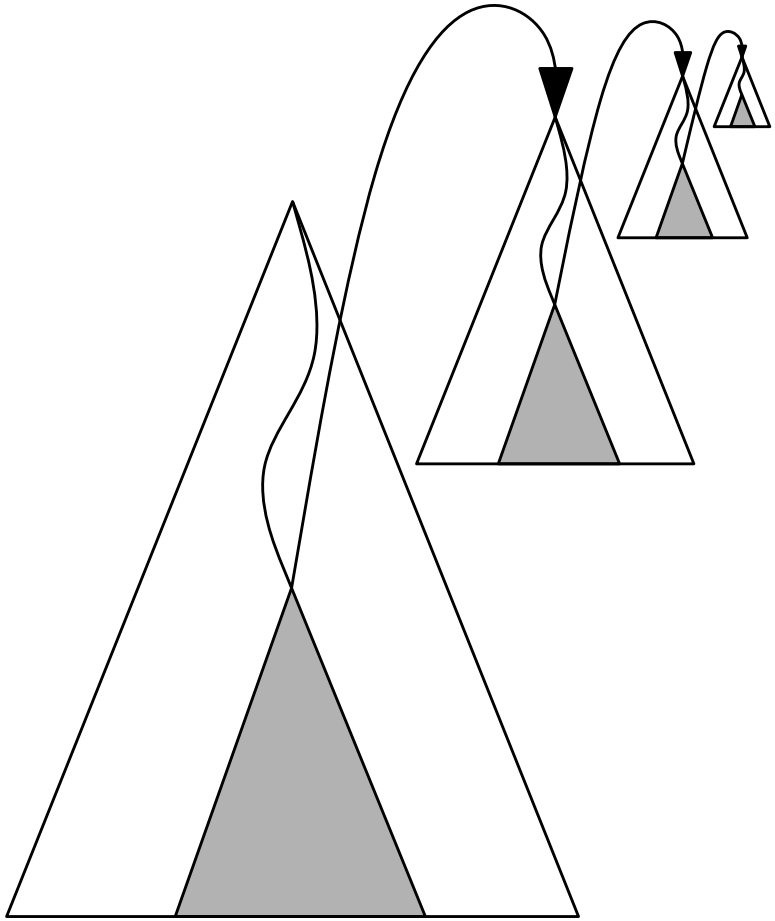
Proof. At each node v in the main tree $\mathcal{T}$ we spend constant time to decide where the search path continues, and we possibly call 1DRANGEQUERY. Theorem 5.2 states that the time we spend in this recursive call is $O(\log n + k_v)$, where k_v is the number of points reported in this call. Hence, the total time we spend is

$$\sum_v O(\log n + k_v),$$

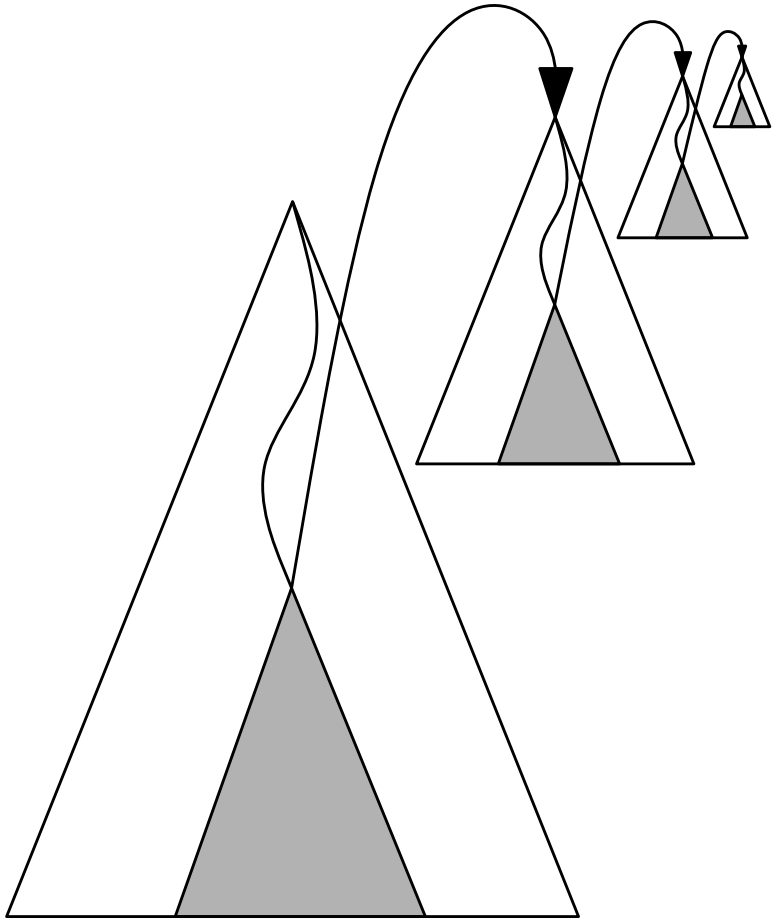
where the summation is over all nodes in the main tree $\mathcal{T}$ that are visited. Notice that the sum $\sum_v k_v$ equals k, the total number of reported points. Furthermore, the search paths of x and x' in the main tree $\mathcal{T}$ have length $O(\log n)$. Hence, $\sum_v O(\log n) = O(\log^2 n)$. The lemma follows. □

Theorem 5.8 *Let P be a set of n points in the plane. A range tree for P uses $O(n \log n)$ storage and can be constructed in $O(n \log n)$ time. By querying this range tree one can report the points in P that lie in a rectangular query range in $O(\log^2 n + k)$ time, where k is the number of reported points.*

Higher-Dimensional Range Searching



Higher-Dimensional Range Searching



Theorem 5.9 *Let P be a set of n points in d-dimensional space, where $d \geqslant 2$. A range tree for P uses $O(n \log^{d-1} n)$ storage and it can be constructed in $O(n \log^{d-1} n)$ time. One can report the points in P that lie in a rectangular query range in $O(\log^d n + k)$ time, where k is the number of reported points.*

END.