ORTHOGONAL RANGE SEARCHING(2)

فاطمه محبی

9009544

Algorithm SeasrchKdtree(v.R)

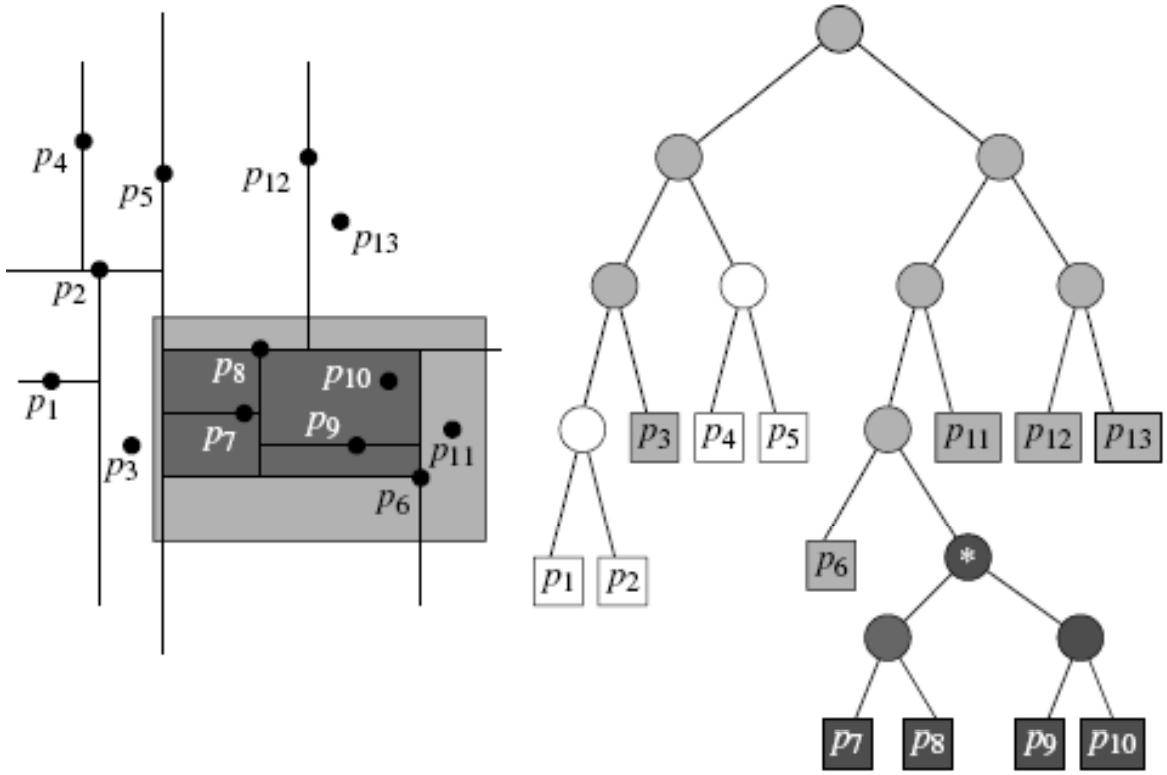
Input. The root of (a subtree of) a kd-tree. And a range R.

Output. All point 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 SeasrchKdtree(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 SeasrchKdtree(rc(v) .R)
```

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.

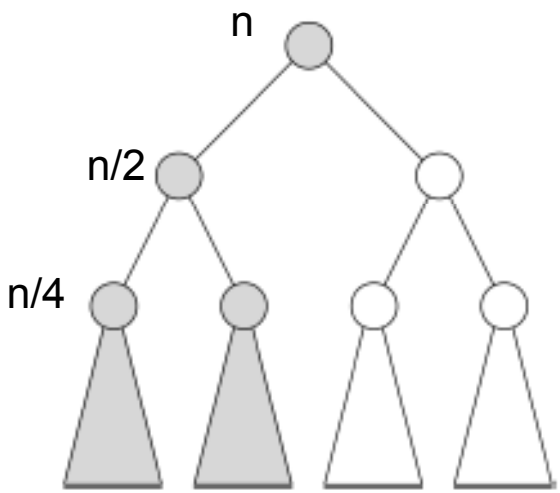


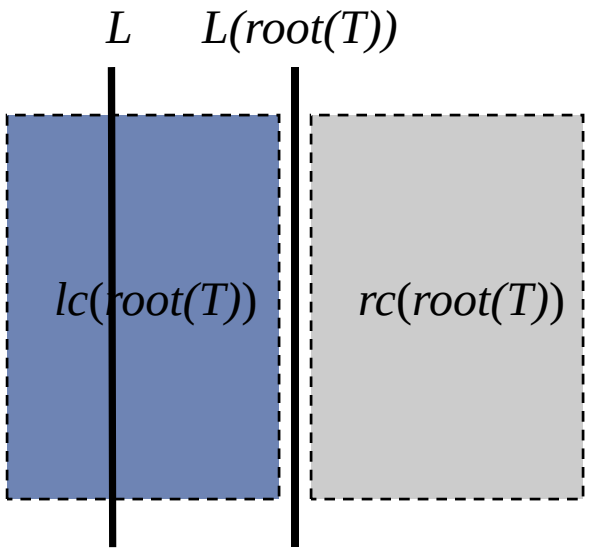
Proof

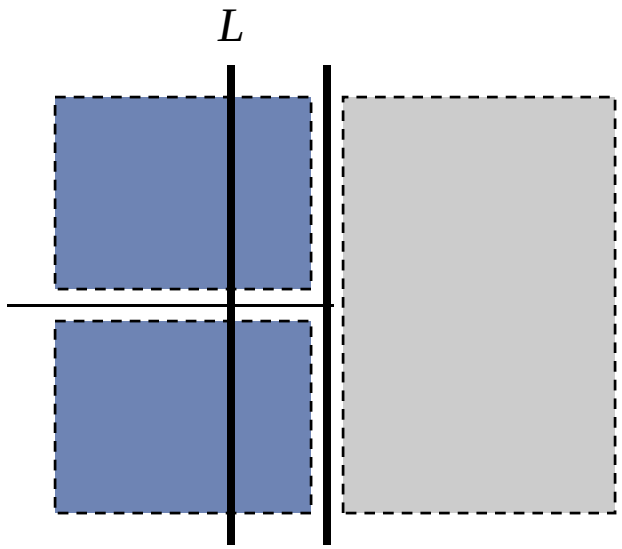
- The time to traverse a subtree and report the points stored in its leaves is linear in the number of reported points. The total time required for traversing subtrees in steps 4 and 8 is O(k).
- The number of nodes visited by the query algorithm that are not in one of the traversed subtrees. Q(n) The number of regions intersected by any vertical(horizontal) line.

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

the total number of regions intersected by the boundary of a rectangular query range is bounded by O(√n).


n
n/2
n/4


L
L(root(T))
lc(root(T))
rc(root(T))


L

Theorem 5.5

A kd-tree for a set P of n points in the plane uses O(n) storage and can be build 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.

Kd-tree for 3- or higher-dimensional space

the construction algorithm is very similar to the planer case: at the root the point set is partitioned based on the first coordinate of the points. At the children of the root the partition is based on the second coordinate, at nodes at depth two on the third coordinate, and so on, until at depth d-1 we partition on the last coordinate. At depth d we start all over again. The recursion stops when there is only one point left.

Storage: O(n)

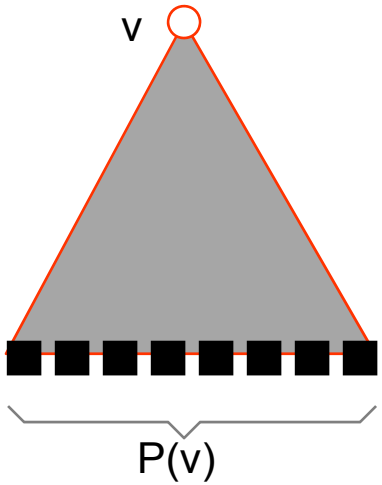
Construction time: O(n log n)

Query time: $O(n^{1-1/d} + k)$

Range trees

Kd-trees have $O(\sqrt{n} + k)$ query time. The range tree, which has a better query time $O((\log n)^2 + k)$. The price we have to pay for this improvement is an increase in storage from O(n) for kd-trees to O(nlogn) for range trees.

Let's call the subset of points stored in the leaves of the subtree rooted at a node v the canonical subset of v (p(v)).

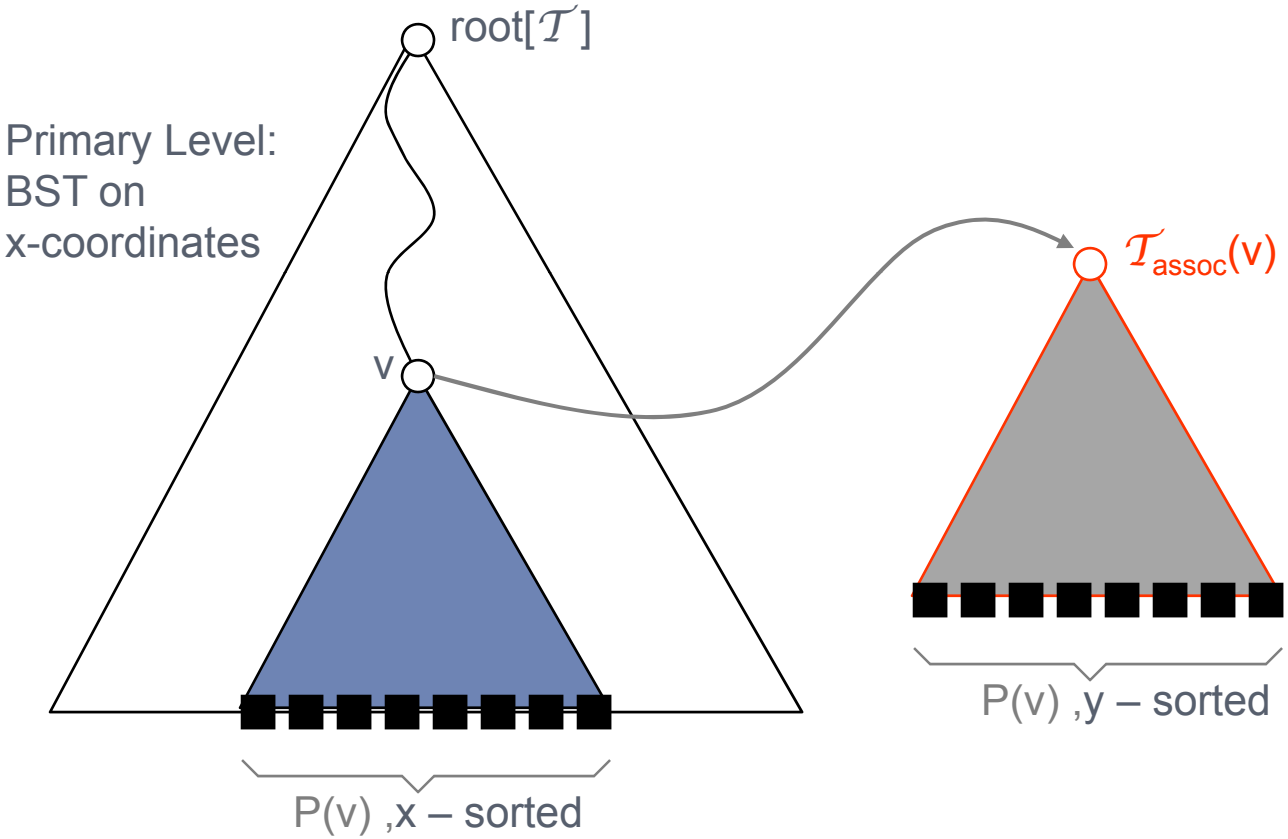


2-dimensional range tree

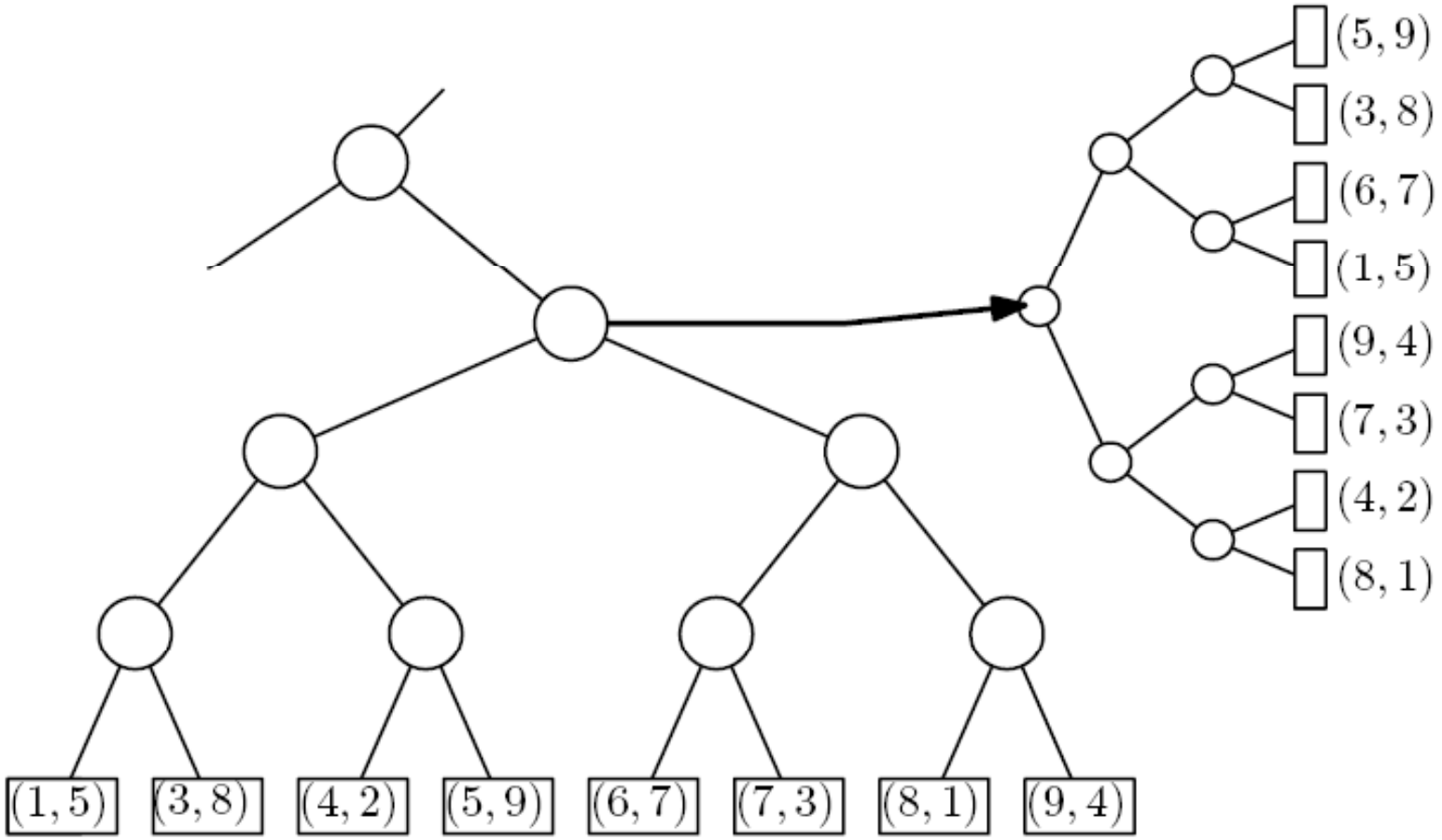
As in the pervious section, we assume that no two pionts have the sam x- or y-coordinate.

- The main tree is a balanced binary search tree $\mathcal{T}$ build on x-coordinateof the points in P.
- For any internal or leaf node v in $\mathcal{T}$, the canonicla subset p(v) is stored in a balanced binary search tree $\mathcal{T}_{\text{assoc}}(\text{v})$ on the y-coordinate of the points. The node v stores a pointer to the root of $\mathcal{T}_{\text{assoc}}(\text{v})$, which is called the associated structure of v.

A 2-dimensional range tree



A example of 2-dimensional range tree



Algorithm BUILD2DRANGETREE(*P*)

Input. A set *P* of points in the plane.

Output. The root of a 2-dimensional range tree.

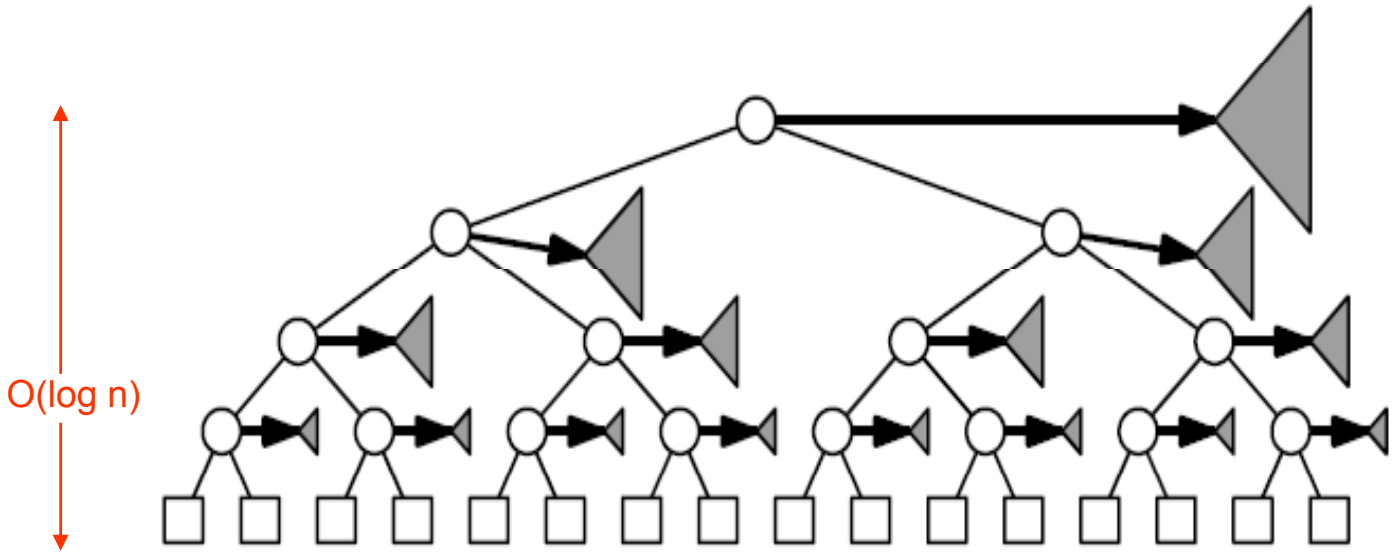
1. Construct the associated structure: Build a binary search tree T_{assoc} on the set p_y of *y*-coordinates of the points in *P*. Store at the leaves of T_{assoc} not just the *y*-coordinate of the points in p_y, but the points themselves.
2. **if** *P* contains only one point
3. **then** Create a leaf *v* storing this point, and make T_{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. $v_{left} \leftarrow$ BUILD2DRANGETREE(p_{left})
6. $v_{right} \leftarrow$ 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 T_{assoc} the associated structure of *v*.
8. **return** *v*

Lemma 5.6

A range tree on a set of n points in the plane requires O(n log n) storage.

Proof

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). The total amount of storage is O(n log n).

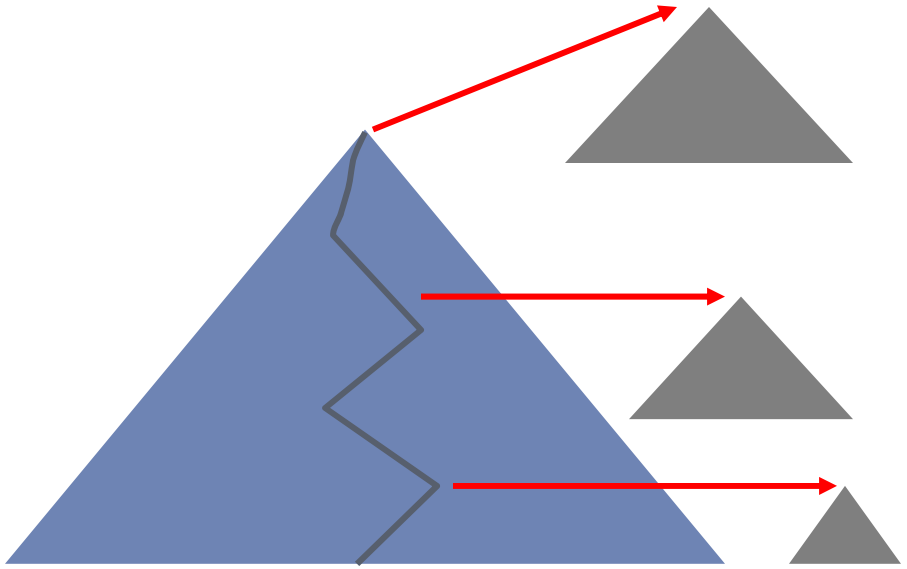


Construction time

constructing the associated structure in line 1 would take O(nlogn) time, but we can do better if the points are presorted on y-coordinate: then the BST can be constructed bottom-up in linear time. This way the time we spend at a node in the main tree $\mathcal{T}$ is linear in the size of its canonical subset.

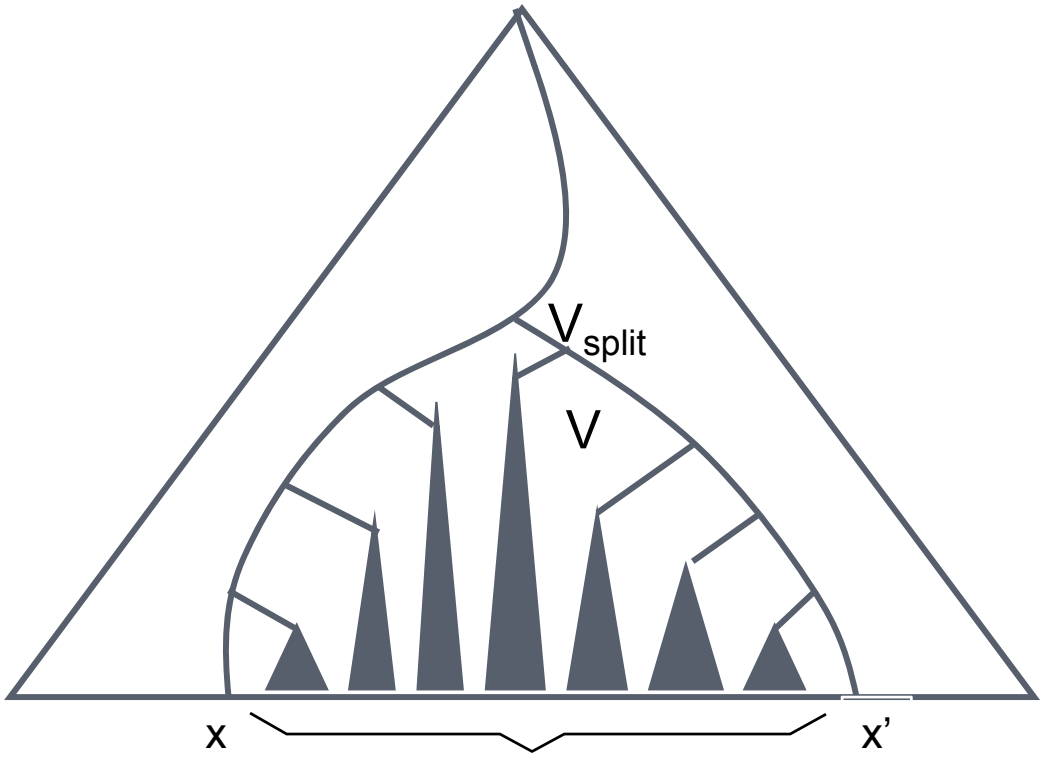
- The total construction time is the same as the amount of storage. Namely O(n log n).
- The presorting takes O(n log n) time.

The total construction time is O(n log n).



2-dimensional range query

The query algorithm first selects O(log n) canonical subsets that together contain the points whose x-coordinate lie in the range [x : x']. This can be done with the 1-dimensional query algorithm. Of those subsets, we then report the points whose y-coordinate lie in the range [y : y'].



O(log n) canonical sub-trees

Algorithm 2DRANGEQUERY($\mathcal{T}$.[x:x']×[y:y'])

Input. A 2-dimensional range tree $\mathcal{T}$ and a range [x:x']×[y:y'].

Output. All points in $\mathcal{T}$ that lie in the range.

1. v_{split} ← FINDSLPITNODE($\mathcal{T}$,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 ←lc(v_{split})
6. **While** v is not a leaf
7. **do if** $x \leq x_v$
8. **then** 1DRANGEQUERY($\mathcal{T}_{assoc}$(rc(v)), [y:y'])
9. v ←lc(v)
10. **else** v ←rc(v)
11. Check if the point stored at v must be reported.
12. Similarly, follow the path from rc(v_{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.

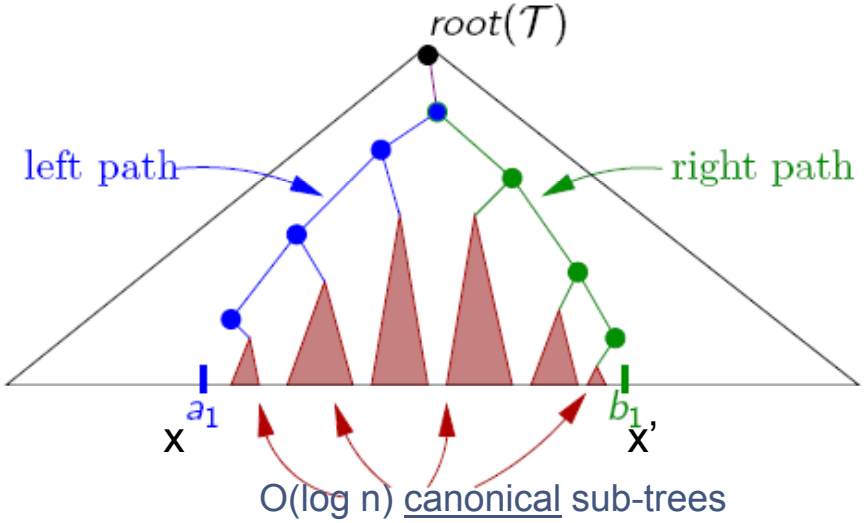
Lemma 5.7

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

Proof

- At left and right paths. Total cost O(log n).
- Line 8 called at roots of canonical sub-trees, a total of O(log n) times. The time that Each call takes is:

  $O(K_v + \log |T_{assoc}(v)|)=O(K_v + \log n)$ time.
- Total Query Time: $O(\log n + \sum_v(K_v + \log n)) = O(\sum_v K_v + \log^2 n) = O(K + \log^2 n)$

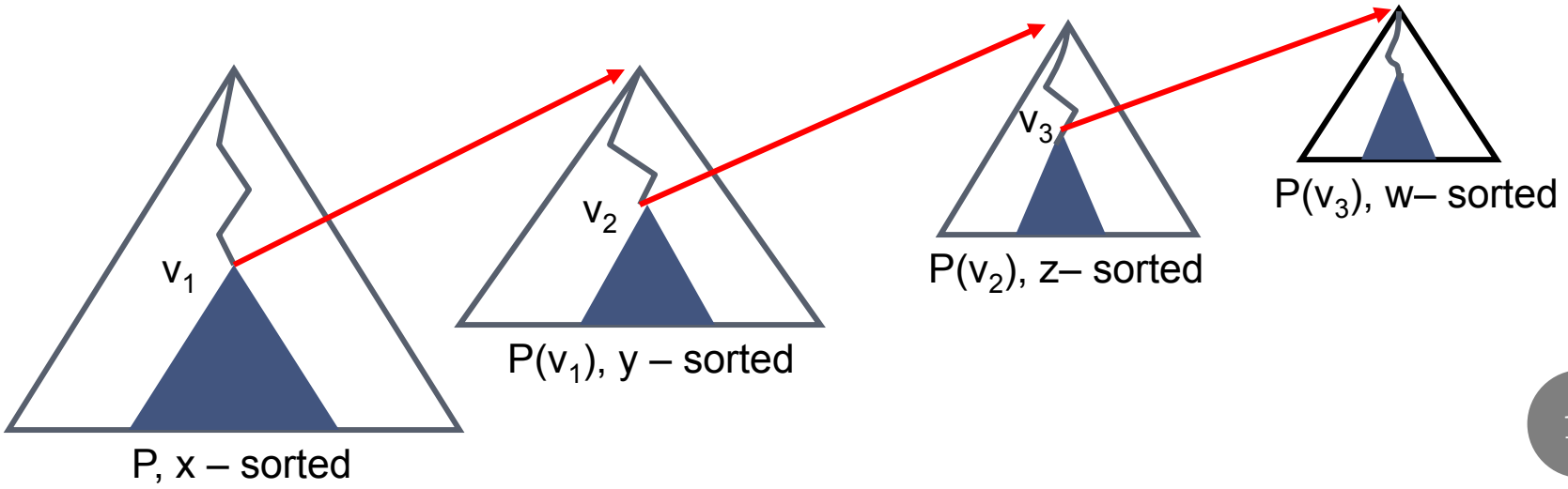


Theorem 5.8

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 n)^2 + k) time, where k is the number of reported points.

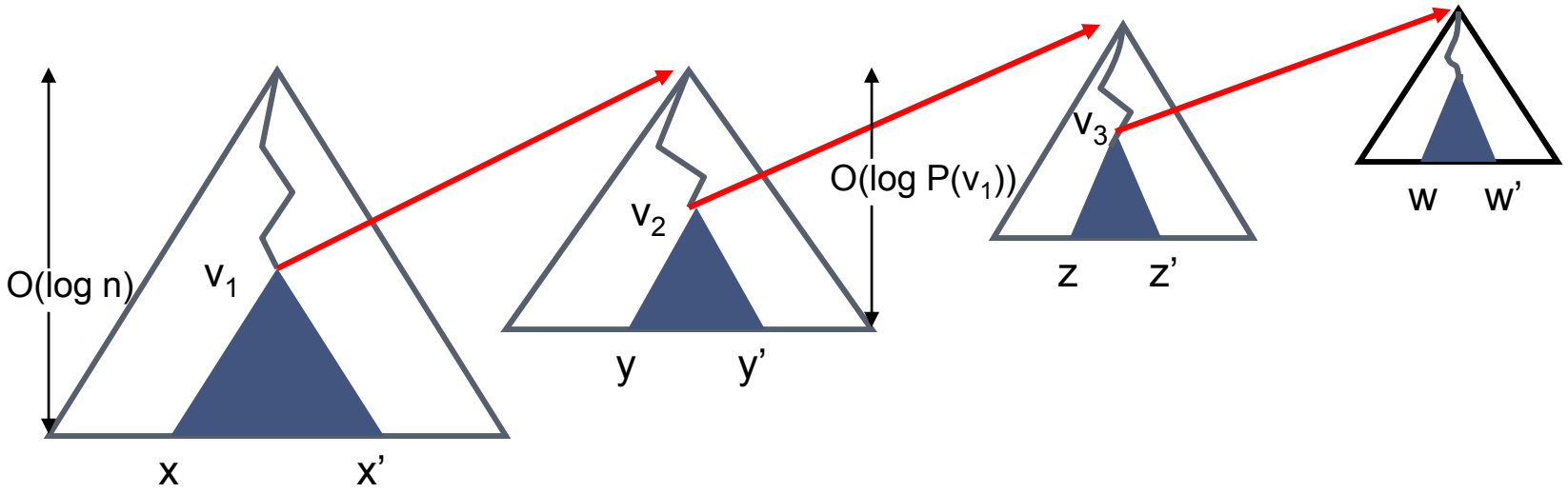
Higher-dimensional range tree

P be a set of points in d-dimensional space



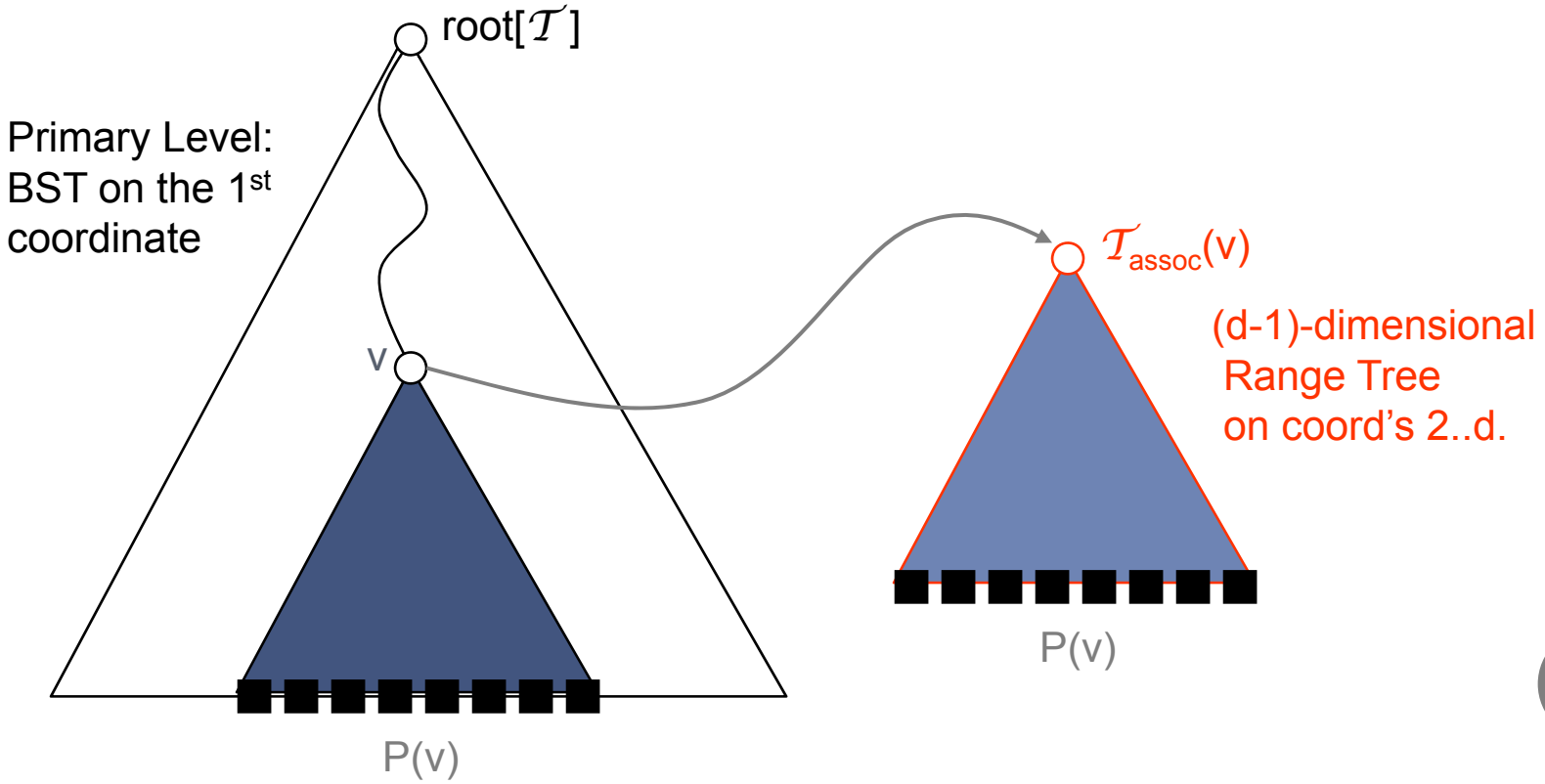
Higher-dimensional range query

The query algorithm is very similar to the 2-dimensional case.



Theorem 5.9

Let P be a set of n points in d-dimensional space, where d>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.



Proof

- Let $T_d(n)$ denote the construction time for a range tree on a set of n points in d-dimensional space.
  - $T_2(n)=O(n \log n)$.
  - $T_d(n)=O(n \log n)+O(\log n). T_{d-1}(n)$.

This recurrence solves to $O(n \log^{d-1} n)$.

- The bound on the amount of storage follows in the same way.
- Let $Q_d(n)$ denote the time spent in querying a d-dimensional range tree on n points. not counting the time to report points.
  - $Q_2(n)=O(\log^2 n)$.
  - $Q_d(n)=O(\log n)+O(\log n). Q_{d-1}(n)$.

This recurrence easily solves to $Q_d(n)=O(\log^d n)$.

- Add the time needed to report points: O(k)

The total query time is $O(\log^d n + k)$.

General sets of points

Until now we imposed the restriction that no two points have equal x- or y-coordinate.

- **Composite Number Space:** (lexicographic order)

  $(a,b) \rightarrow (a \mid b)$

  $(a \mid b) < (a' \mid b') \Leftrightarrow a<a' \text{ or } (a=a' \ \& \ b<b')$

- $p = (p_x , p_y) \rightarrow p' = ((p_x \mid p_y) , (p_y \mid p_x))$

  $R=[x:x'] \times [y:y'] \rightarrow R' = [(x \mid -\infty) : (x' \mid +\infty)] \times [(y \mid -\infty) : (y' \mid +\infty)]$

- Note: no two points in the composite space have the same value at any coordinate (unless they are identical points).

Lemma 5.10

Let P be a point and R a rectangular range. Then

$p \in R \Leftrightarrow p' \in R'$.

Proof

$$x \le p_x \le x' \quad \& \quad y \le p_y \le y'$$

$$(x \mid -\infty) \le (p_x \mid p_y) \le (x' \mid +\infty) \quad \& \quad (y \mid -\infty) \le (p_y \mid p_x) \le (y' \mid +\infty)$$

The approach of using composite number can also be used in higher dimensions.

END