Range searching with
Range Trees

#### 1D

• Balanced BinarySearchTree

• Build: O(n lg n)

• Space: O(n)

• Range queries: O(lg n +k)

#### 2D

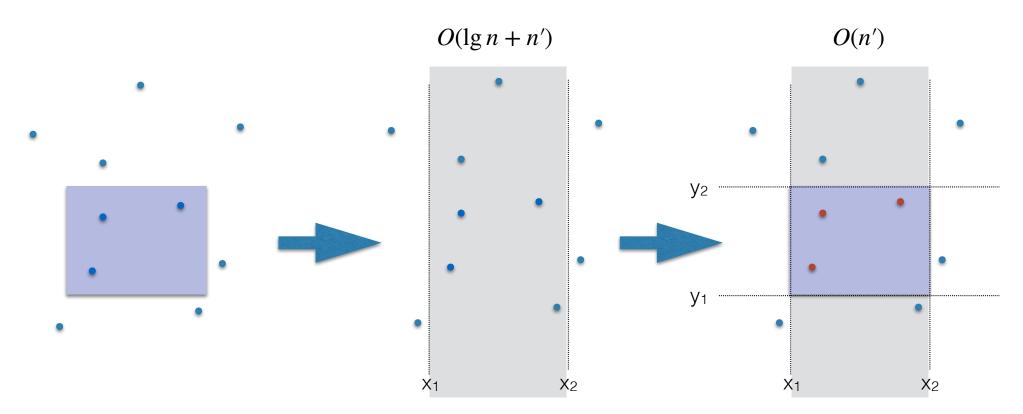
- kd-trees
  - Build: O(n lg n)
  - Space: O(n)
  - Range queries:  $O(\sqrt{n} + k)$

- Range trees
  - Build: O(n lg n)
  - Space: O(n lg n)
  - Range queries: O(lg n + k)

Different trade-offs

### Towards range trees

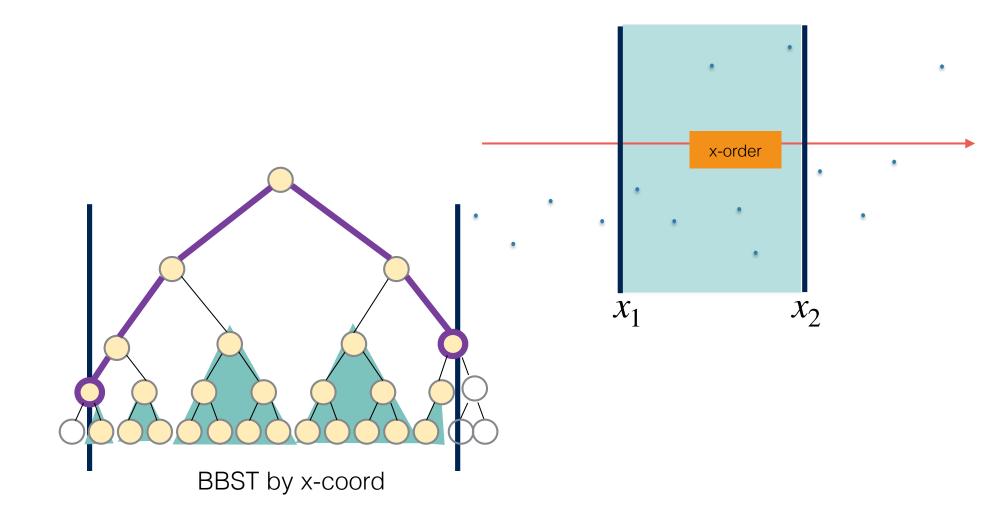
- Build BBST by x-coord
- Range queries: find all points in  $[x_1, x_2] \times [y_1, y_2]$ 
  - Use BBST to find all points with the x-coordinates in [x<sub>1</sub>, x<sub>2</sub>]
  - Traverse these points and find those with y-coord in [y<sub>1</sub>, y<sub>2</sub>]

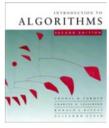


Slow if n' is large but k is small

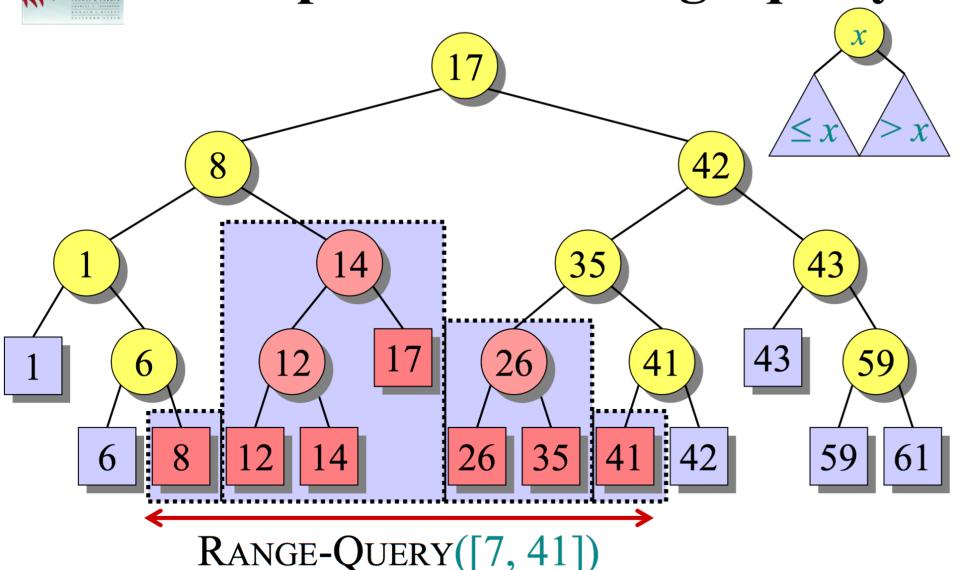
#### A closer look

• Use BBST to find all points with the x-coordinates in [x1, x2]



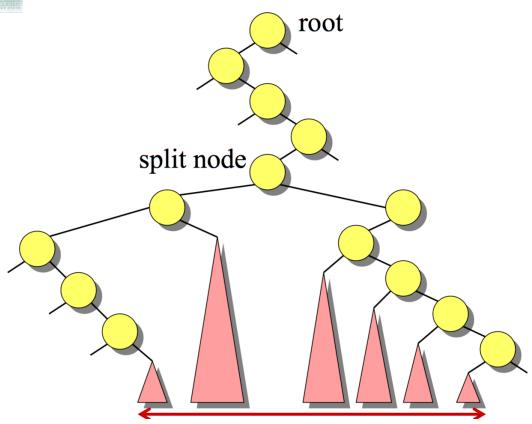


Example of a 1D range query





# General 1D range query

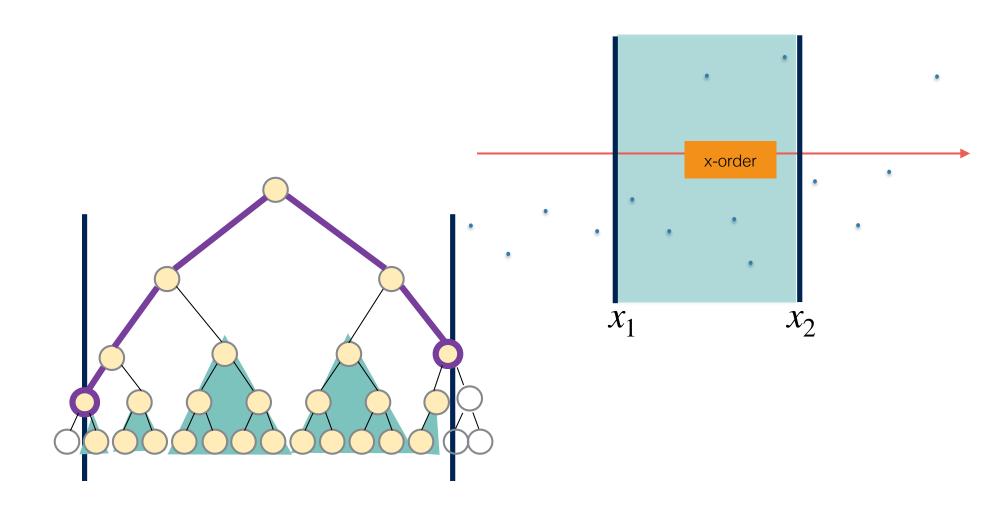


The k points in the range sit in  $O(\lg n)$  subtrees

#### A closer look

• Use BBST to find all points with the x-coordinates in [x1, x2]

The points in  $[x_1, x_2]$  sit in  $O(\lg n)$  subtrees

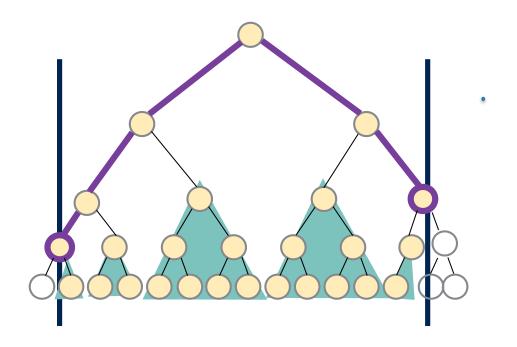


#### A closer look

• Use BBST to find all points with the x-coordinates in [x1, x2] The points in  $[x_1, x_2]$  sit in  $O(\lg n)$  subtrees

Of all these points, we need those with the y-coordinates in [y<sub>1</sub>, y<sub>2</sub>]

For each subtree we need all points in  $[y_1,y_2]$ 



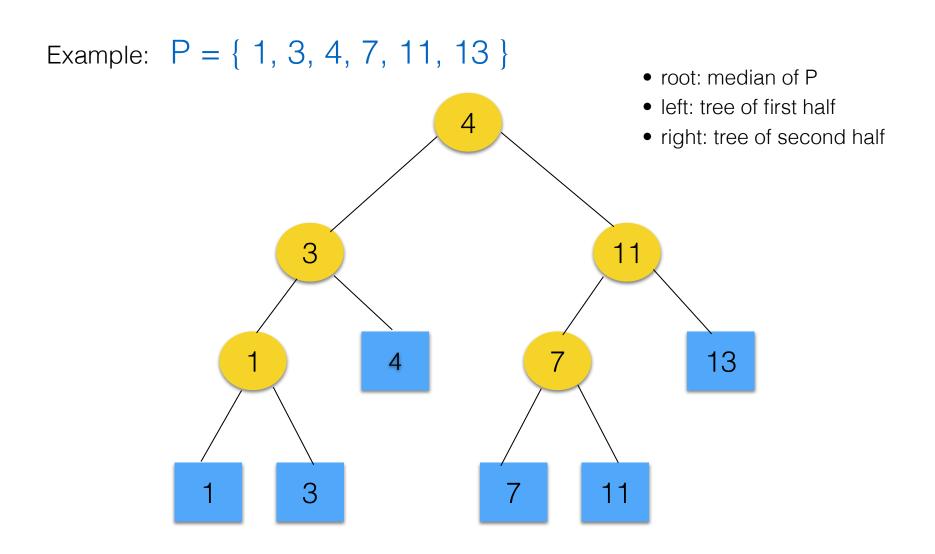
For each of these subtrees we'll build data structure for searching on y-coord

What is a good data structure for searching on y?

A BBST by y-coord!

#### **BBST**

• We'll use a variant of BBSTs that store all data in leaves (it makes details simpler)



#### Class work

- Show the BBST with all data in leaves for P = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- Write pseudocode for the algorithm to build BBST(P)
- root: median of P
- left: tree of first half
- right: tree of second half

//create and return a BBST of P with all data in leaves
BuildBBST ( P )

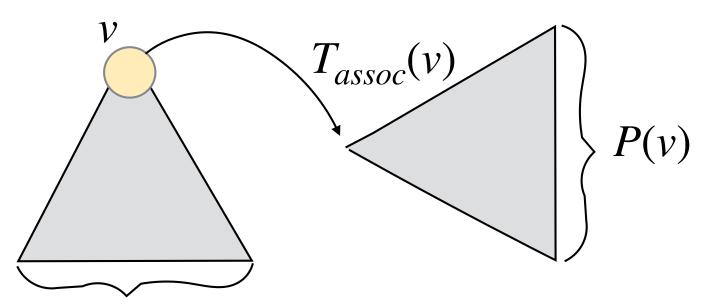
- Running time analysis in general.
- Running time analysis if P is given sorted.

### And finally, the 2D Range Tree!

P: set of points

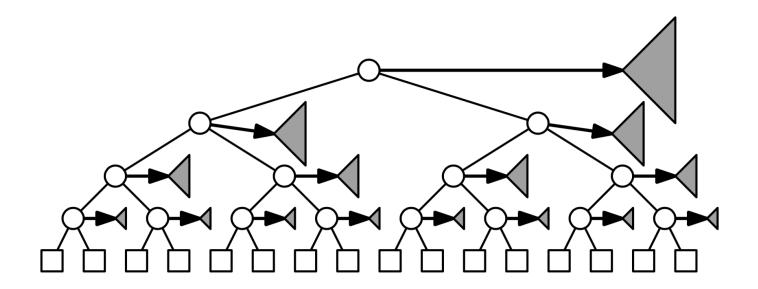
RangeTree(P) is

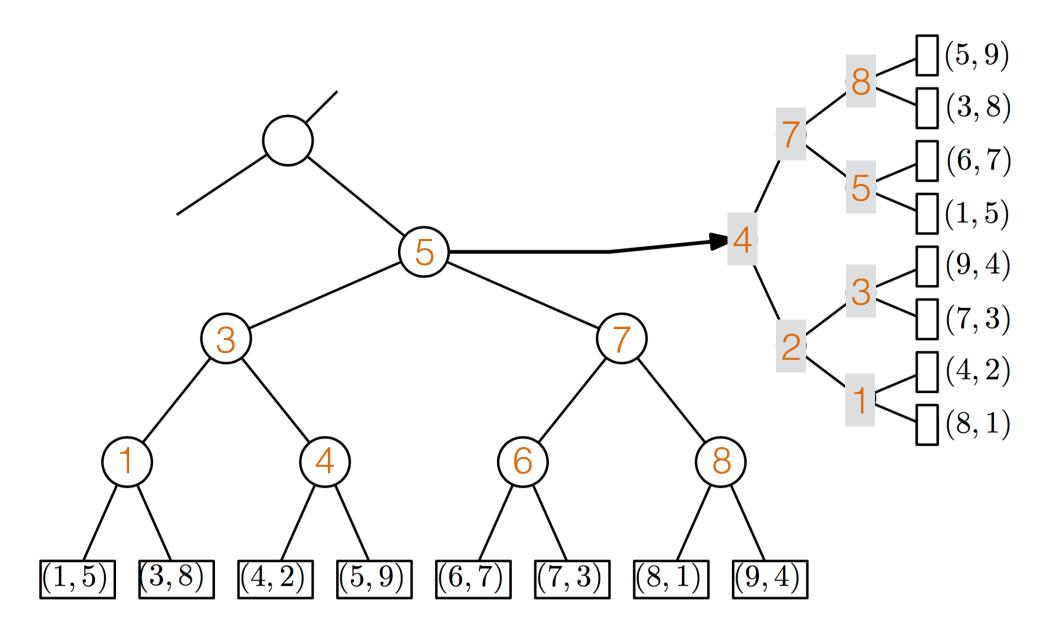
- A BBST T of P ordered by x-coord
- Each node v in T stores an associated structure  $T_{assoc}$  that's a BBST of P(v) ordered by y-coord



P(v): all points in subtree rooted at v

Every internal node stores a whole tree in an *associated* structure, on *y*-coordinate





# The 2D Range Tree

#### Questions

- How to build one and how fast?
- How much space does it use?
- How to answer range queries and how fast?

### Building a 2D Range Tree

 $//P = \{p_1, p_2, \dots, p_n\}$ . Assume P sorted by x-coord.

#### Algorithm Build2DRT(P)

- 1. if P contains only one point:
  - create a leaf v storing this point, create its T<sub>assoc</sub> and return v
- 2. else
  - Construct the associated structure: build a BBST on P ordered by the ycoordinates, and call its root T<sub>assoc</sub> (P)
  - Partition P into 2 sets w.r.t. the median coordinate x<sub>middle</sub>:
    - $P_{left} = \{p \text{ in } P \text{ with } p_x \leq x_{middle}\}, P_{right} = \dots$
  - V<sub>left</sub> = Build2DRT(P<sub>left</sub>)
  - Vright = Build2DRT(Pright)
  - Create a node v storing  $x_{middle}$ , make  $v_{left}$  its left child, make  $v_{right}$  its right child, make  $T_{assoc}$  its associate structure
  - return v

### Building a 2D Range Tree

#### Running time:

- Let T(n) be the time of **Build2DRT(P)**, where P has n points
- We know that building a BBST on an unsorted set of keys takes  $O(n \lg n)$
- Then  $T(n) = 2T(n/2) + O(n \lg n)$
- This solves to  $T(n) = O(n \lg^2 n)$

• Hmm. Remember our trick?

#### Building a 2D Range Tree

• Common trick: pre-sort P and pass it as argument

```
//P_x is set of points sorted by x-coord 
//P_y is set of points sorted by y-coord 
Build2DRT(P_x, P_y)
```

Maintain the sorted sets through recursion

```
P_{left}-sorted-by-x, P_{left}-sorted-by-y, P_{right}-sorted-by-x, P_{right}-sorted-by-y
```

- If the keys are in order, a BBST can be built in O(n)
- We have T(n) = 2T(n/2) + O(n) which solves to  $O(n \lg n)$

**Theorem**: A 2d-range tree for a set of n points can be built in  $\Theta(n \lg n)$  time.

#### Class work

Show the range tree for

$$p_1 = (1,4), p_2 = (5,8), p_3 = (4,1), p_4 = (7,3), p_5 = (3,2), p_6 = (2,6), p_7 = (8,7)$$

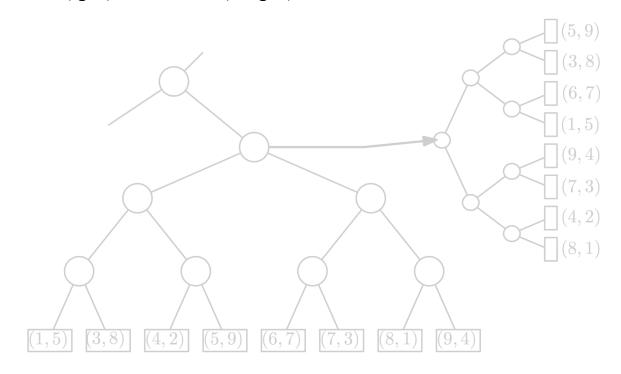
#### The 2D Range Tree: Space

**Theorem**: A 2d-range tree for a set of n points in the plane has  $\Theta(n \lg n)$  size.

#### Two arguments:

• At each level in the tree, each point is stored exactly once (in the associated structure of precisely one node). So every level stores all points and uses O(n) space => O( n lg n)

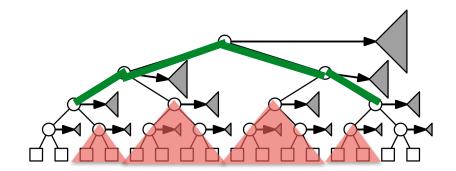
Or: Each point p is stored in the associated structures of all nodes on the path from root to p. So one point is stored  $O(\lg n)$  times =>  $O(n \lg n)$ 



### Range queries with the 2D Range Tree

- Find the split node  $x_{split}$  where the search paths for  $x_1$  and  $x_2$  split
- Follow path root to x<sub>1</sub>: for each node v to the right of the path, query its associated structure T<sub>assoc</sub>(v) with [y<sub>1</sub>,y<sub>2</sub>]
- Follow path root to x<sub>2</sub>: for each node v to the
   left of the path, query its associated structure
   T<sub>assoc</sub>(v) with [y<sub>1</sub>,y<sub>2</sub>]
- (Also, while traversing these paths, check and report the nodes on the path)

Every internal node stores a whole tree in an associated structure, on y-coordinate



How long does this take?

### Range queries with the 2D Range Tree

- There are  $O(\lg n)$  subtrees in between the paths
- We query each one of them using its associated structure
- Querying  $T_{assoc}$  takes  $O(\lg n_v + k')$

Every internal node stores a whole tree in an associated structure, on y-coordinate

• Overall it takes  $\sum O(\lg n_v + k') = O(\lg^2 n + k)$ 

 $n_{v}$ : number of points in  $T_{\rm assoc}$ 

k': number of points in  $T_{assoc}$  that are in [y1,y2]

**Theorem**: A 2d-range tree for a set of n points answers range queries in  $O(\lg^2 n + k)$  time.

We like logs!! Also, it is known how to improve this to  $O(\lg n + k)$  time.

#### 1D

• Balanced BinarySearchTree

• Build: O(n lg n)

• Space: O(n)

• Range queries: O(lg n +k)

#### 2D

- kd-trees
  - Build: O(n lg n)
  - Space: O(n)
  - Range queries:  $O(\sqrt{n} + k)$

- Range trees
  - Build: O(n lg n)
  - Space: O(n lg n)
  - Range queries: O(lg n + k)

Different trade-offs

## Kd-tree vs Range Tree: Does it really matter?

#### in 2D

n	$\log n$	$\log^2 n$	$\sqrt{n}$
16	4	16	4
64	6	36	8
256	8	64	16
1024	10	100	32
4096	12	144	64
16384	14	196	128
65536	16	256	256
1M	20	400	1K
16M	24	576	4K

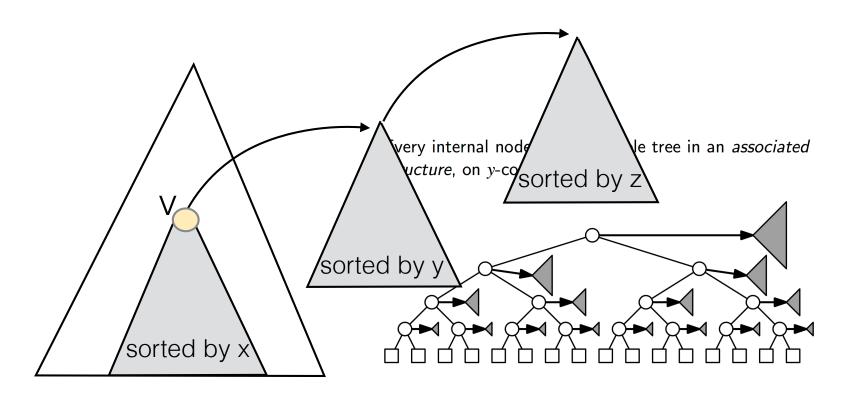




P: set of points in 3D

3DRangeTree(P) is

- A BBST T of P ordered by x-coord
- Each node v in T stores an associated structure  $T_{assoc}$  that's a 2D range tree for P(v)



Build time:  $O(n \lg^2 n)$ 

- Think recursively
- Let  $B_3(n)$  be the time to build a 3D Range Tree of n points
- Find a recurrence for  $B_3(n)$ 
  - Think about how we build it: we build an associated structure for P that's a 2D range tree; then we build recursively a 3D range tree for the left and right half of the points
  - $B_3(n) = 2B_3(n/2) + B_2(n)$
  - This solves to  $O(n \lg^2 n)$

Size:  $O(n \lg^2 n)$ 

- Why? we can thinks of this in two ways:
- An associated structure for n points uses  $O(n \lg n)$  space. Each point is stored in all associated structures of all its ancestors =>  $O(n \lg^2 n)$
- Or, recursively
  - Let  $S_3(n)$  be the size of a 3D Range Tree of n points
  - Find a recurrence for  $S_3(n)$
  - We build an associated structure for P that's a 2D range tree; then we build recursively a 3D range tree for the left and right half of the points
  - $S_3(n) = 2S_3(n/2) + S_2(n)$
  - This solves to  $O(n \lg^2 n)$

#### Query:

- Query BBST on x-coord to find  $O(\lg n)$  nodes (roots of subtrees)
- Then perform a 2D range query in each node

#### Time:

- Let  $Q_3(n)$  be the time to answer a 3D range query
- Find a recurrence for  $Q_3(n)$ 
  - $Q_3(n) = O\lg n + O(\lg n) \times Q_2(n)$
  - This solves to  $O(n \lg^3 n + k)$

# Kd-tree vs Range Tree

4D

n	$\log n$	$\log^4 n$	$n^{3/4}$
1024	10	10,000	181
65,536	16	65,536	4096
1M	20	160,000	32,768
1G	30	810,000	5,931,641
1T	40	2,560,000	1G

screen shot from Mark van Kreveld slides, http://www.cs.uu.nl/docs/vakken/ga/slides5b.pdf)

#### Class work

Show the 3D-range tree for the set of points below:

