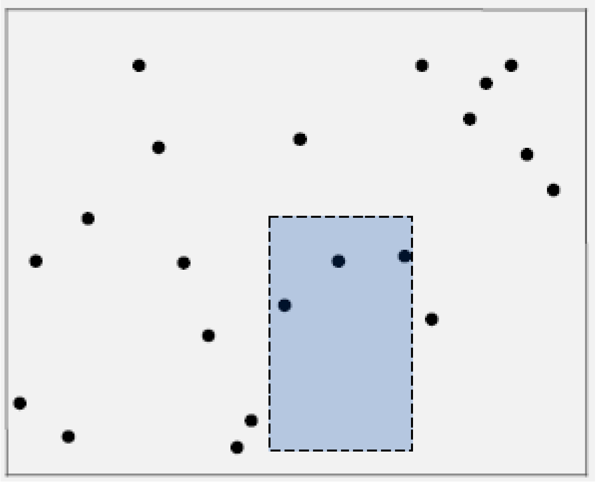
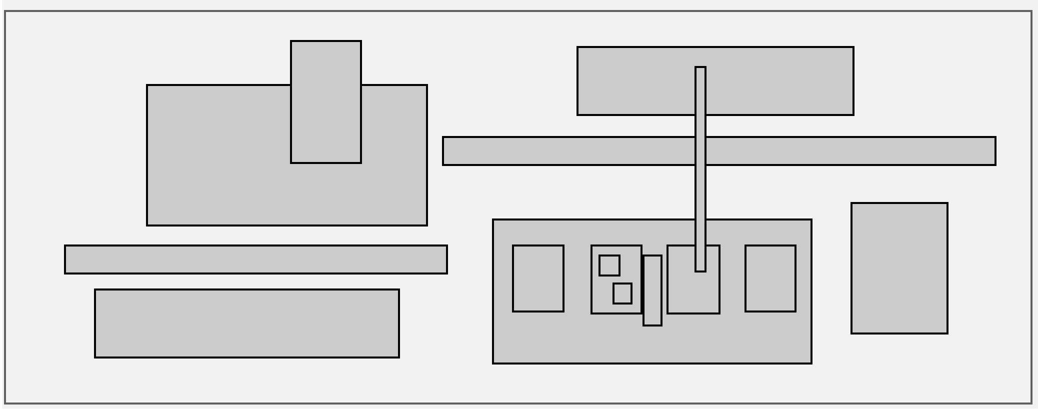
Lect10. Geometry Algorithm-2

Overview

- Types of data. Points, lines, intervals, circles, rectangles, polygons, ...
- Considering theme : Intersection among N objects.

- Example problems
  - 1D range search.
  - 2D range search.
  - Find all intersections among h-v segments.
  - Find all intersections among h-v rectangles.





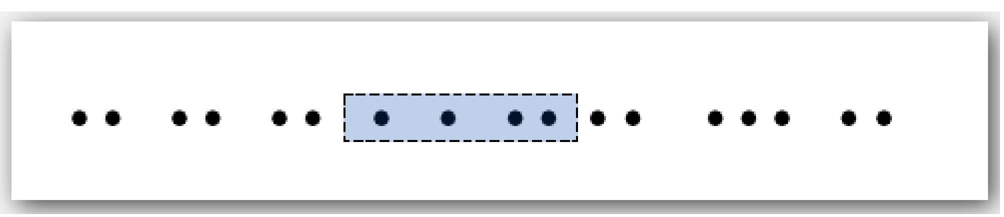
3

Range Search

1D Range Search

- Extension of ordered symbol table.
  - Insert key-value pair.
  - Search for key k.
  - Delete key k.
  - Range search: find all keys between k1 and k2.
  - Range count: number of keys between k1 and k2.
- Application. Database queries.
- Geometric interpretation.
  - Keys are point on a line.
  - Find/counts points in a given 1d interval.

```
insert B         B
insert D         B D
insert A         A B D
insert I         A B D I
insert H         A B D H I
insert F         A B D F H I
insert P         A B D F H I P
count G to K     2
search G to K    H I
```



1d range search: elementary implementations

- Unordered list. Fast insert, slow range search.
- Ordered array. Slow insert, binary search for $k1$ and $k2$ to do range search.

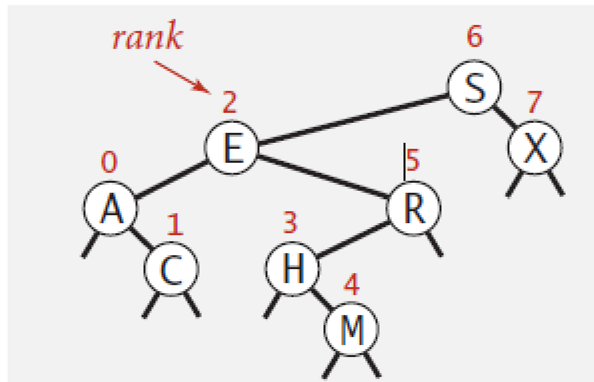
order of growth of running time for 1d range search

data structure	insert	range count	range search
unordered list	1	N	N
ordered array	N	$\log N$	$R + \log N$
goal	$\log N$	$\log N$	$R + \log N$

N = number of keys
R = number of keys that match

1d range count: BST implementation

- 1d range count. How many keys between lo and hi ?



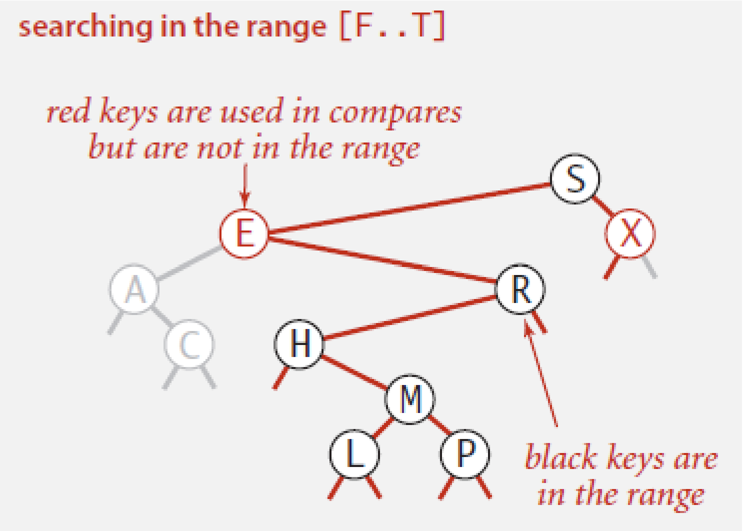

```
public int size(Key lo, Key hi)
{
   if (contains(hi)) return rank(hi) - rank(lo) + 1;
   else              return rank(hi) - rank(lo);
}
```

number of keys < hi

- Proposition. Running time proportional to log N.
- Pf. Nodes examined = search path to lo + search path to hi.

1d range search: BST implementation

- 1d range search. Find all keys between lo and hi.
  - Recursively find all keys in left subtree (if any could fall in range).
  - Check key in current node.
  - Recursively find all keys in right subtree (if any could fall in range).

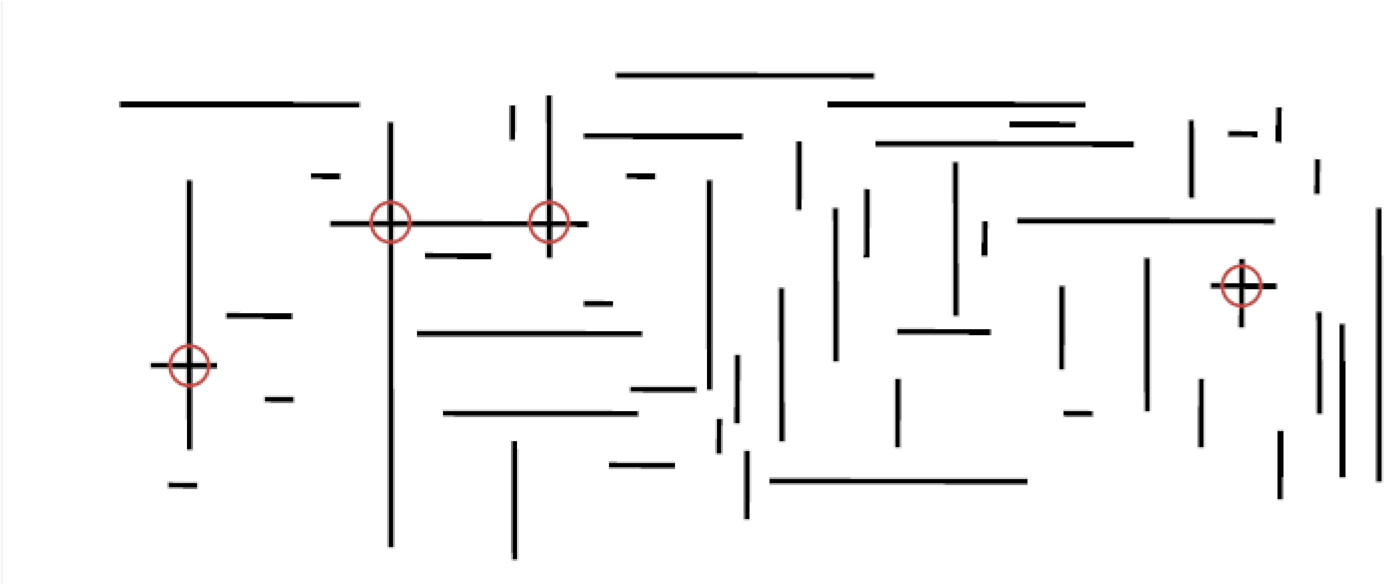


- Proposition. Running time proportional to $R + \log N$.
- Pf. Nodes examined = search path to lo + search path to hi + matches.

8 Line Segmentation Intersection

Orthogonal line segment intersection

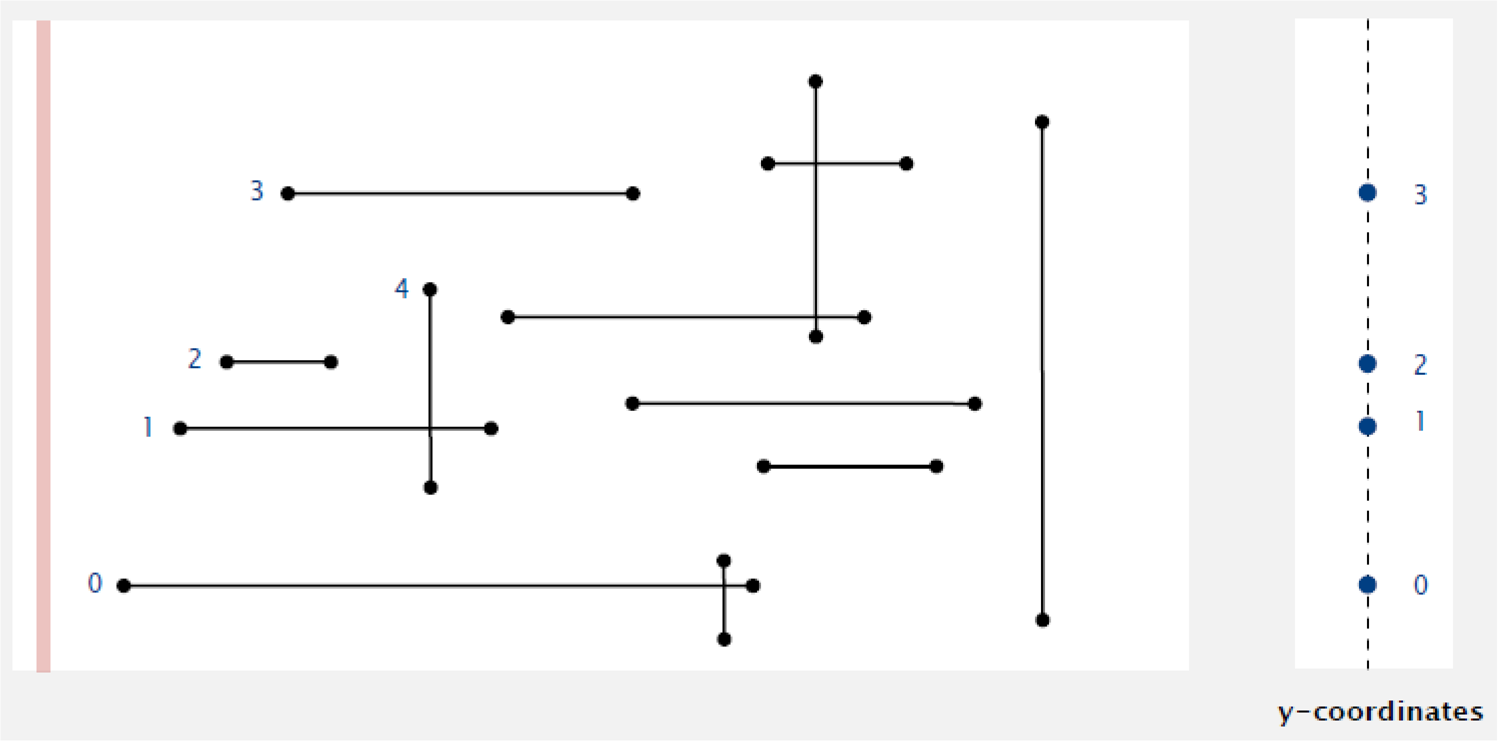
- *N* horizontal and vertical line segments, find all intersections.



- Quadratic algorithm. Check all pairs of line segments for intersection.
- Nondegeneracy assumption. All *x*- and *y*-coordinates are distinct.

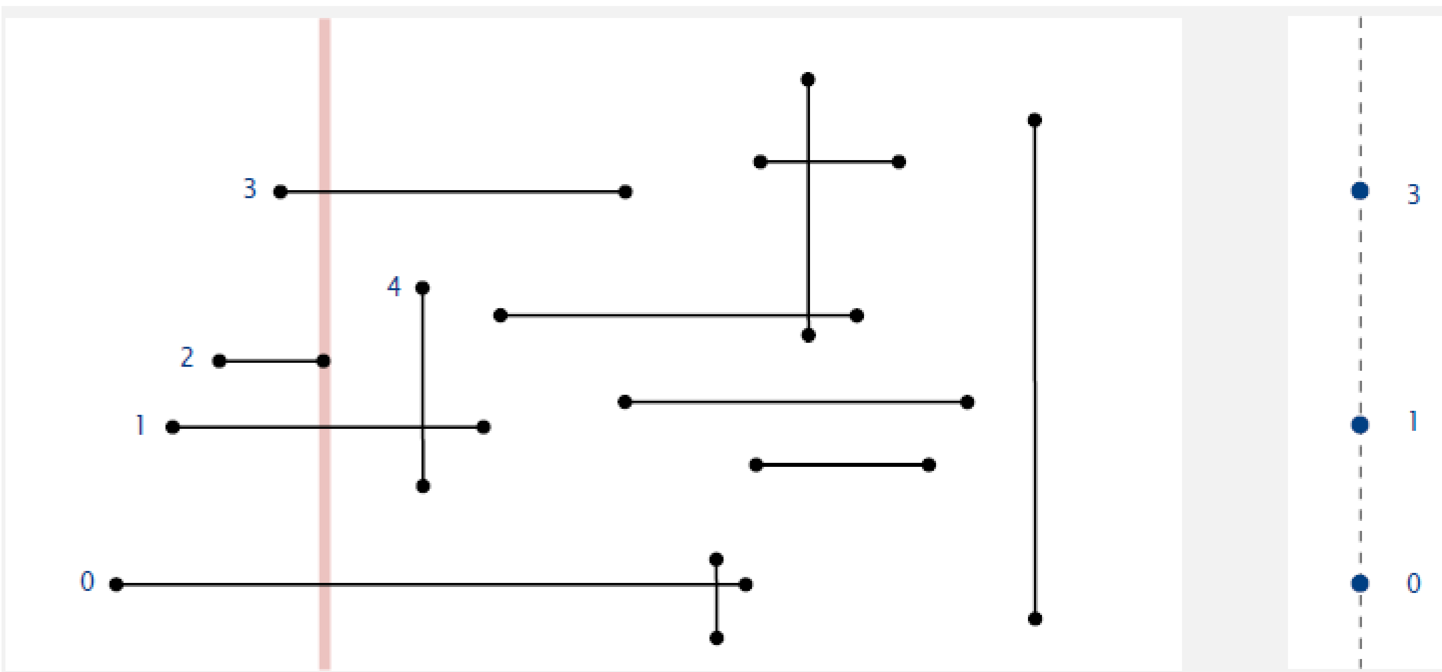
Orthogonal line segment intersection: sweep-line algorithm

- Sweep vertical line from left to right.
  - *x*-coordinates define events.
  - *h*-segment (left endpoint): insert *y*-coordinate into BST.



Orthogonal line segment intersection: sweep-line algorithm

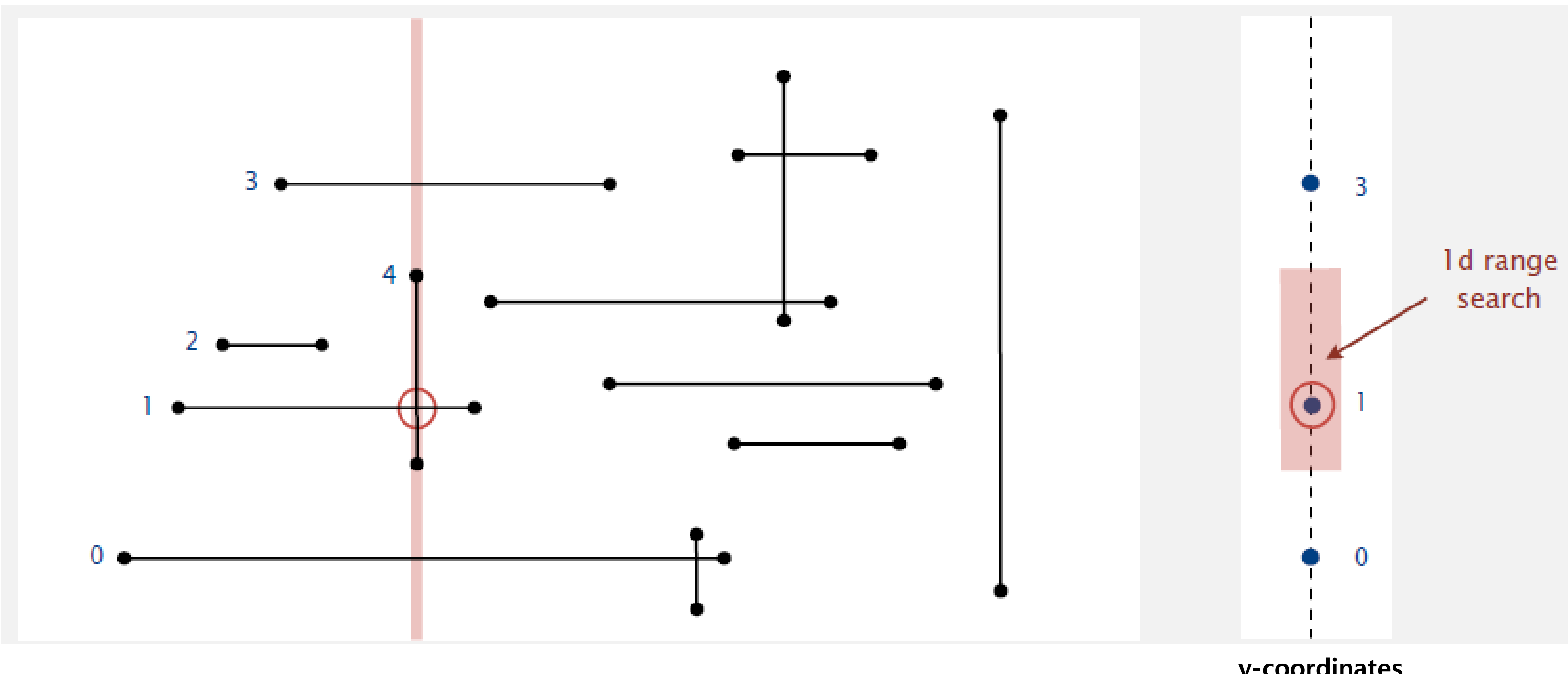
- Sweep vertical line from left to right.
  - x-coordinates define events.
  - h-segment (left endpoint): insert y-coordinate into BST.
  - h-segment (right endpoint): remove y-coordinate from BST.



y-coordinates

Orthogonal line segment intersection: sweep-line algorithm

- Sweep vertical line from left to right.
  - *x*-coordinates define events.
  - *h*-segment (left endpoint): insert *y*-coordinate into BST.
  - *h*-segment (right endpoint): remove *y*-coordinate from BST.
  - *v*-segment: range search for interval of *y*-endpoints.



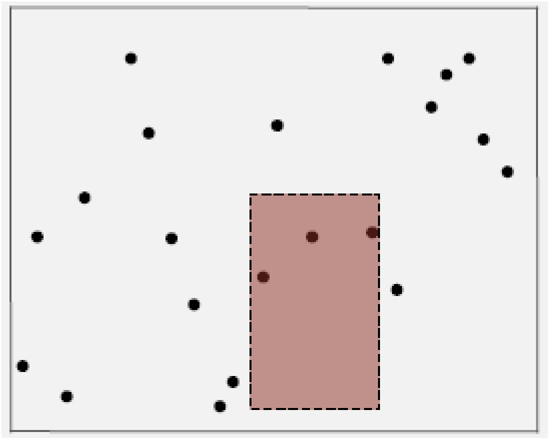
13

kd tree

2-d orthogonal range search

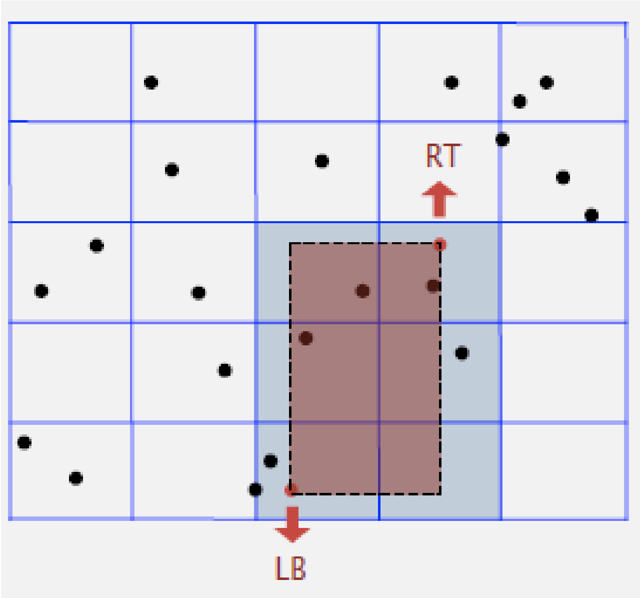
- Extension of ordered symbol-table to 2d keys.
  - Insert a 2d key.
  - Delete a 2d key.
  - Search for a 2d key.
  - Range search: find all keys that lie in a 2d range.
  - Range count: number of keys that lie in a 2d range.
- Applications. Networking, circuit design, databases, ...
- Geometric interpretation.
  - Keys are point in the plane.
  - Find/count points in a given *h-v* rectangle

rectangle is axis−aligned



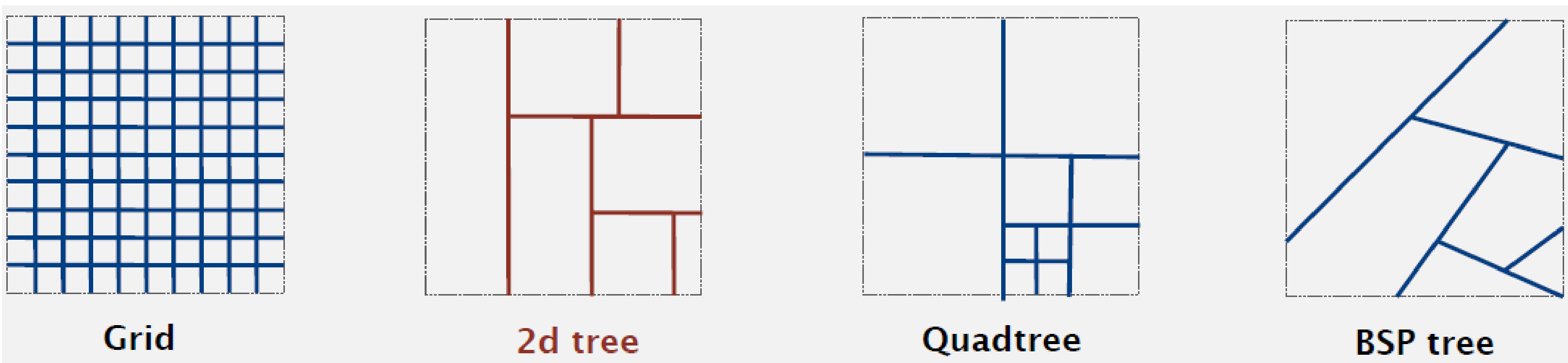
2d orthogonal range search: grid implementation

- Grid implementation.
  - Divide space into M-by-M grid of squares.
  - Create list of points contained in each square.
  - Use 2d array to directly index relevant square.
  - Insert: add (x, y) to list for corresponding square.
  - Range search: examine only squares that intersect 2d range query.



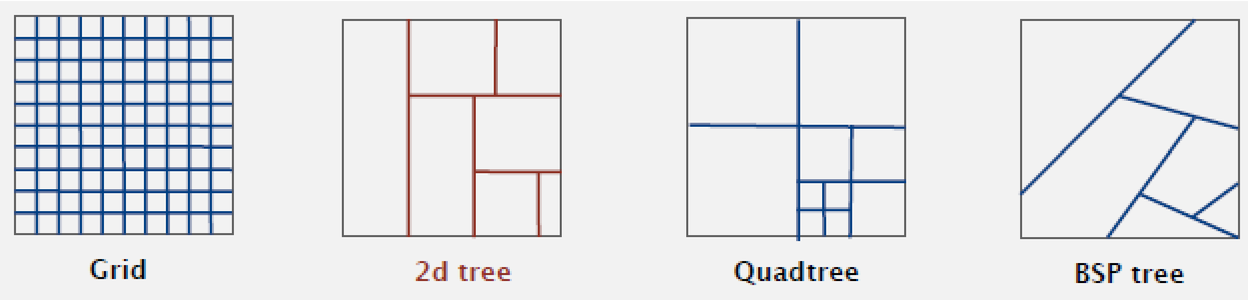
Space-partitioning trees

- Use a tree to represent a recursive subdivision of 2d space.
- Grid. Divide space uniformly into squares.
- 2d tree. Recursively divide space into two halfplanes.
- Quadtree. Recursively divide space into four quadrants.
- BSP tree. Recursively divide space into two regions.



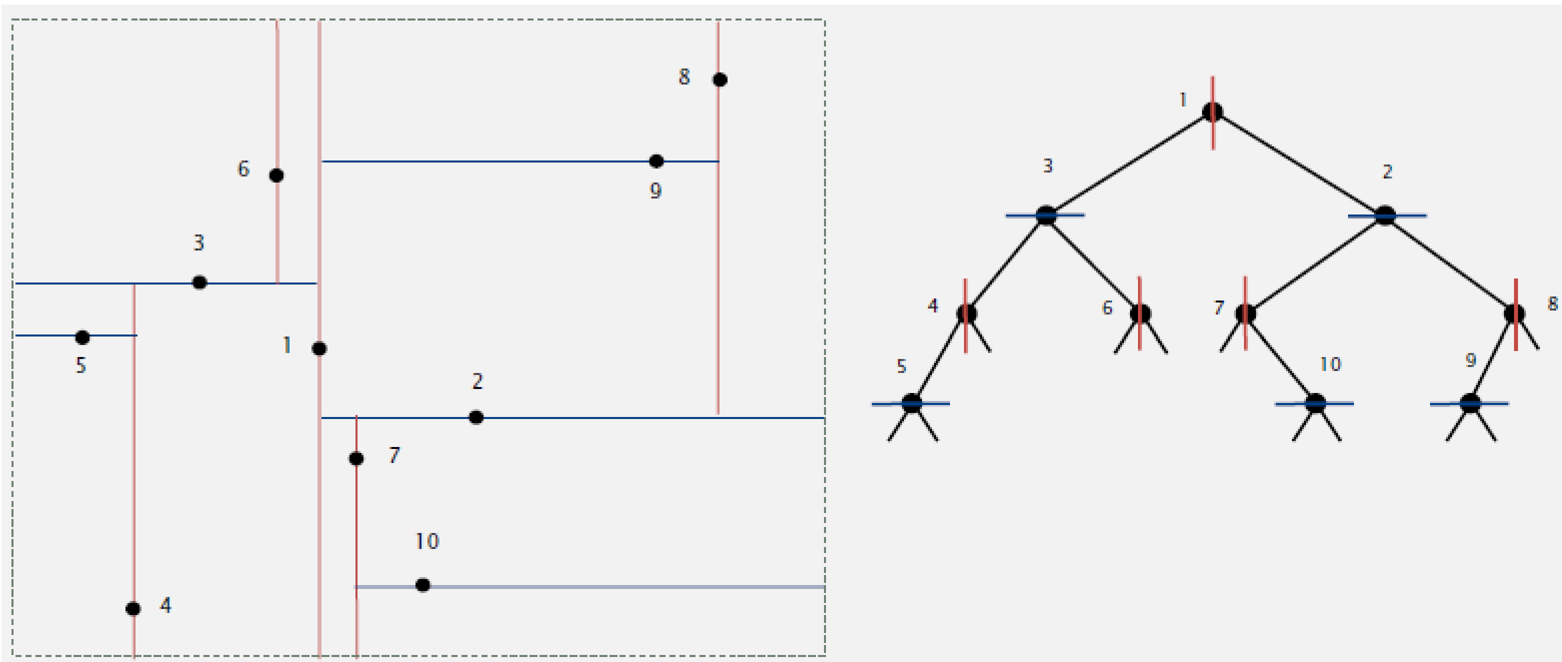
Space-partitioning trees: applications

- Applications.
  - Ray tracing.
  - 2d range search.
  - Flight simulators.
  - N-body simulation.
  - Collision detection.
  - Astronomical databases.
  - Nearest neighbor search.
  - Adaptive mesh generation.
  - Accelerate rendering in Doom.
  - Hidden surface removal and shadow casting.



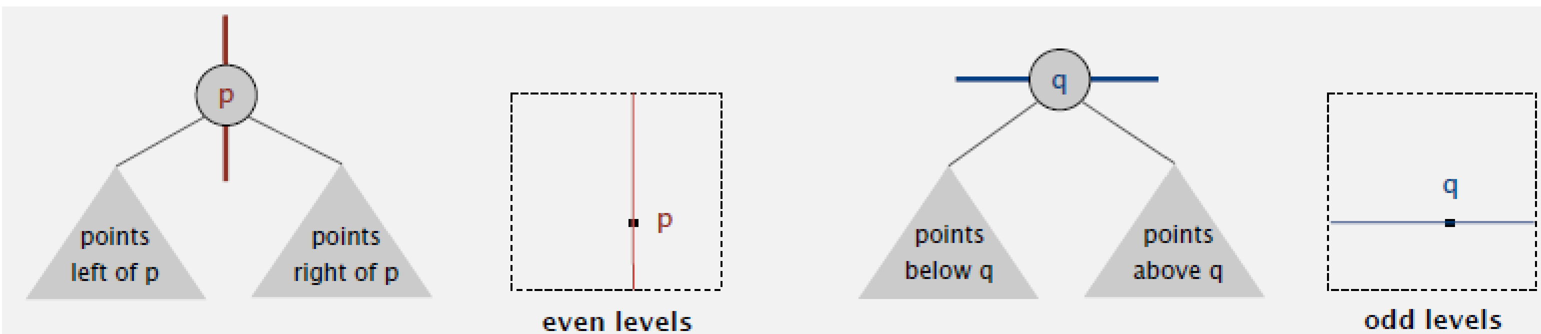
2d tree construction

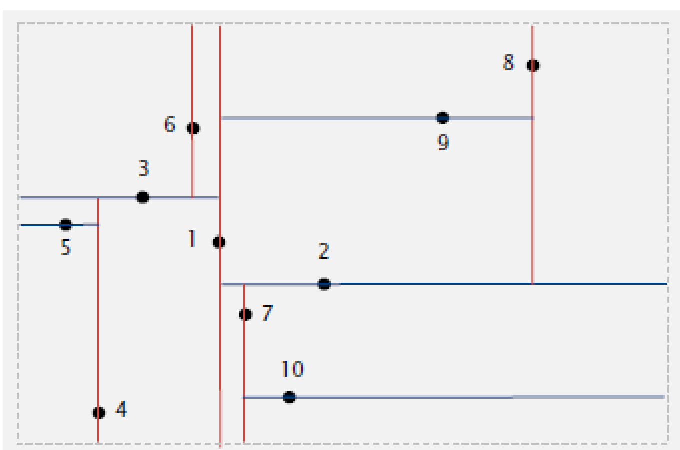
- Recursively partition plane into two halfplanes.

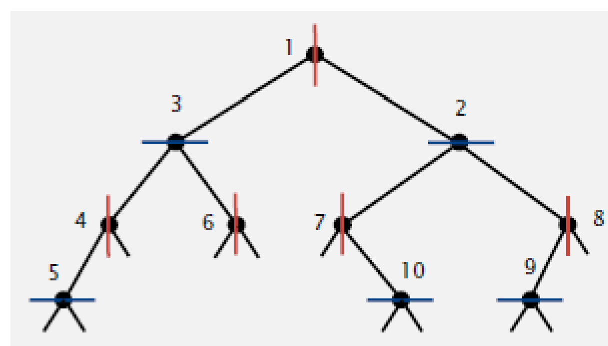


2d tree implementation

- Data structure. BST, but alternate using *x*- and *y*-coordinates as key.
  - Search gives rectangle containing point.
  - Insert further subdivides the plane.

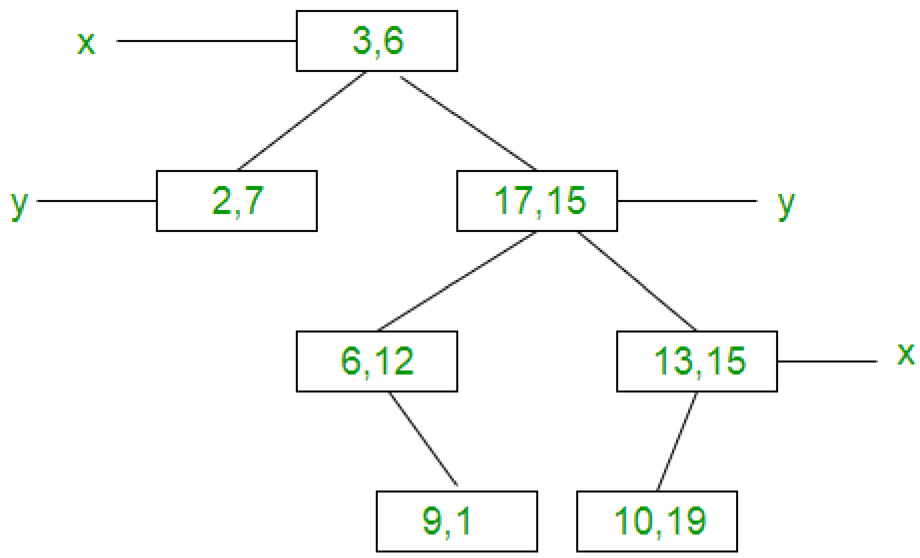






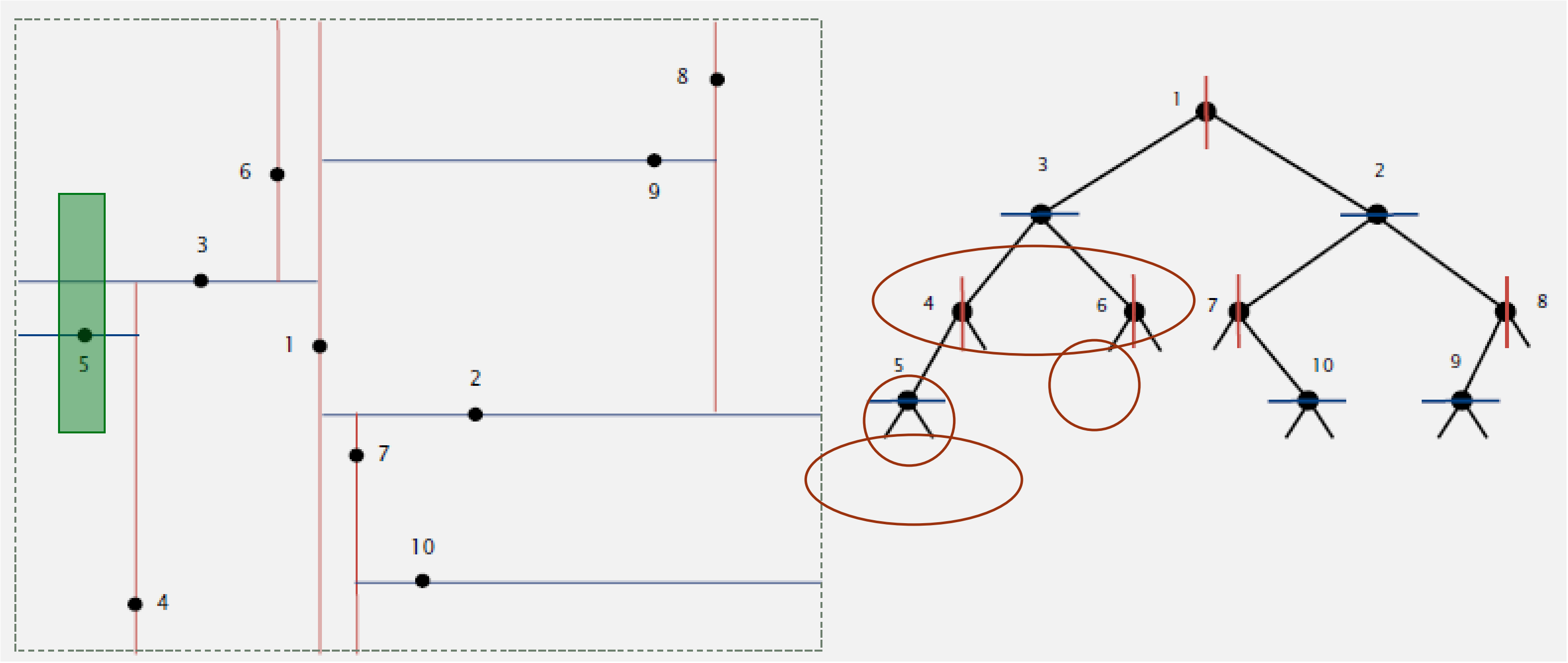
2d tree constuction

- Ex. Constucting 2d tree with (3,6), (17,15),(6,12),(9,1),(2,7),(10,19)
  1. (3,6) : since tree is empty, make it the root node
  2. (17, 15) : right child of root since 3 < 17 (x is key)
  3. (13,15) : 13>3 – right of root, next level- y is key, so compare 16 and 15. move to the right and there is no node. Insert node.
  4. (6,12) : 6>3-right, next 12 < 15, so move to left, and insert node since there is no node.
  5. Next : same approach….



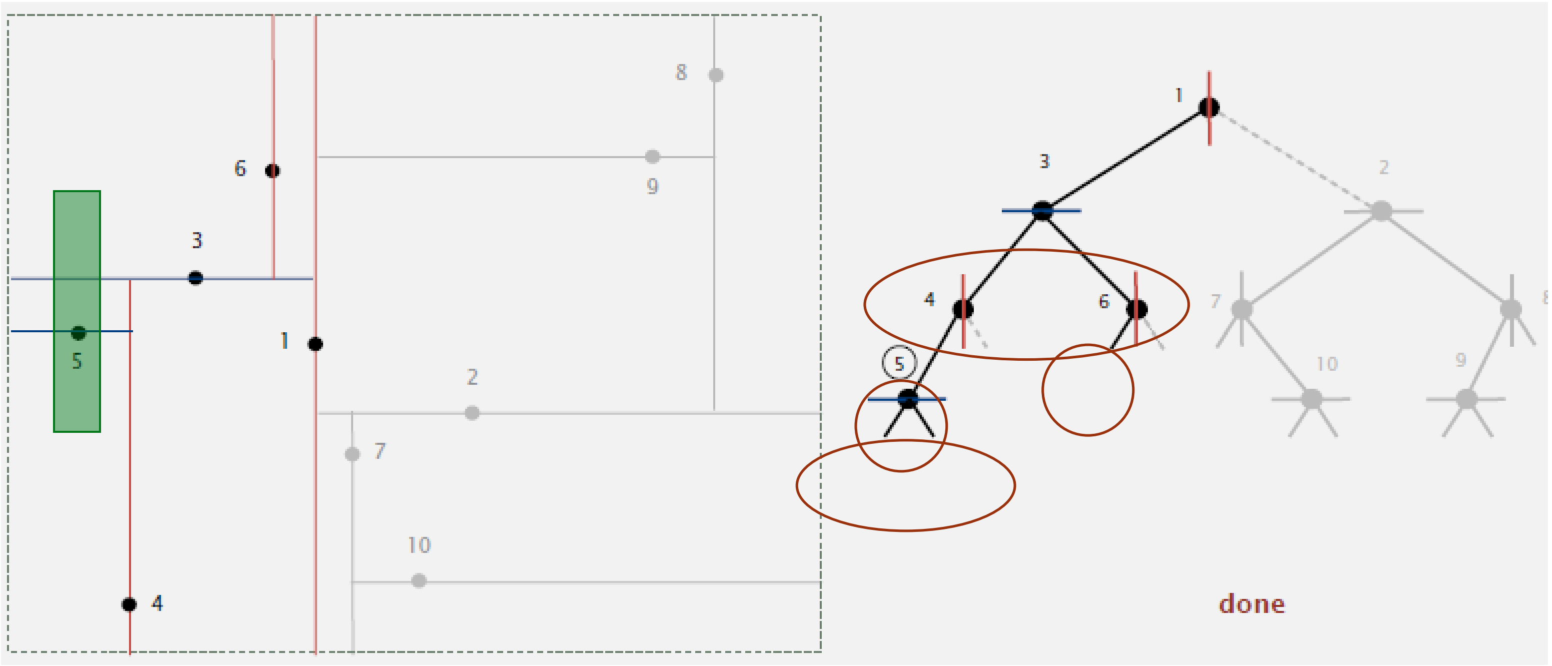
2d tree demo: range search

- Goal. Find all points in a query axis-aligned rectangle.
  - Check if point in node lies in given rectangle.
  - Recursively search left/bottom (if any could fall in rectangle).
  - Recursively search right/top (if any could fall in rectangle).



2d tree demo: range search

- Goal. Find all points in a query axis-aligned rectangle.
  - Check if point in node lies in given rectangle.
  - Recursively search left/bottom (if any could fall in rectangle).
  - Recursively search right/top (if any could fall in rectangle).

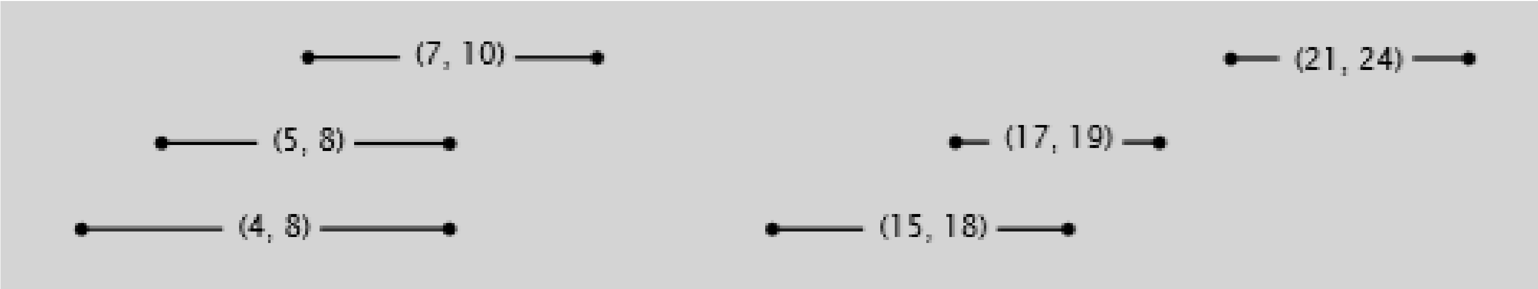


23 Interval Search Trees

1d interval search

- 1d interval search. Data structure to hold set of (overlapping) intervals.
  - Insert an interval(lo, hi)
  - Search for an interval(lo, hi)
  - Delete an interval(lo, hi)
  - Interval intersection query: given an interval(lo, hi), find all intervals (or one interval) in data structure that intersects(lo, hi)

- Q. Which intervals intersect(9,16)?
- A. (7,10) and (15,18)

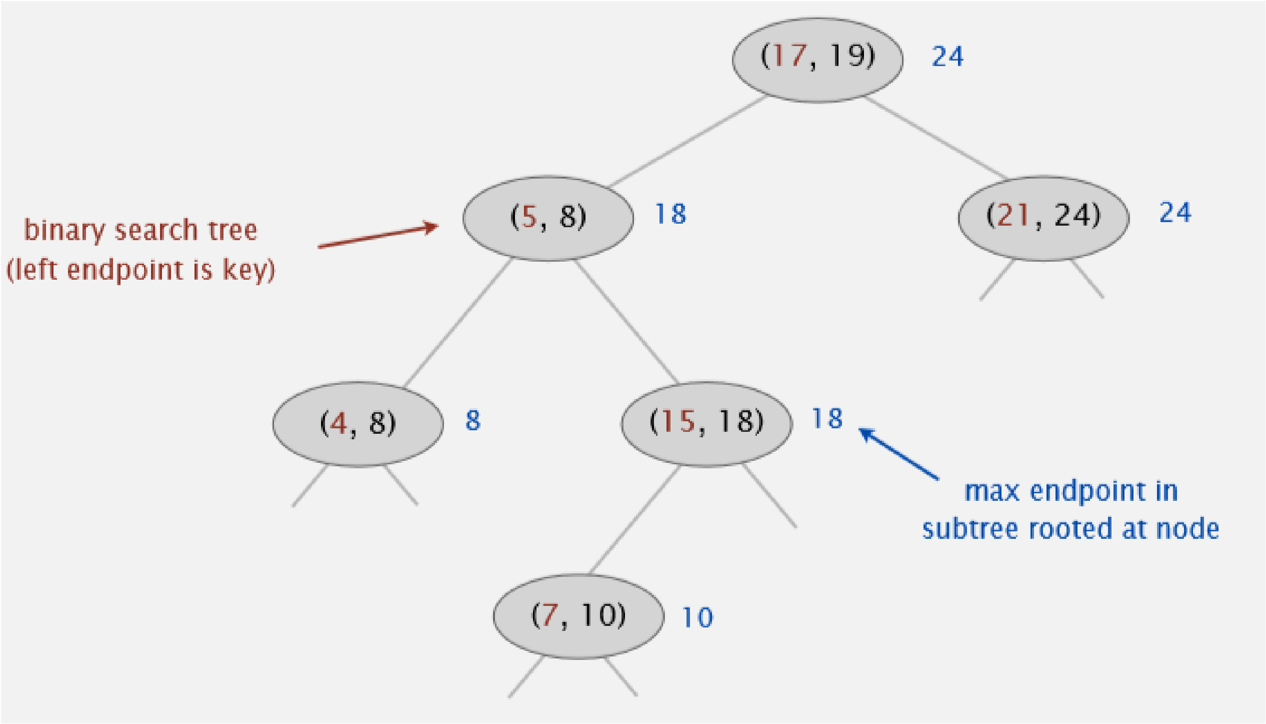


1d interval search API

public class	IntervalST<Key extends Comparable<Key>, Value>	
	IntervalST()	*create interval search tree*
void	put(Key lo, Key hi, Value val)	*put interval-value pair into ST*
Value	get(Key lo, Key hi)	*value paired with given interval*
void	delete(Key lo, Key hi)	*delete the given interval*
Iterable<Value>	intersects(Key lo, Key hi)	*all intervals that intersect (lo, hi)*

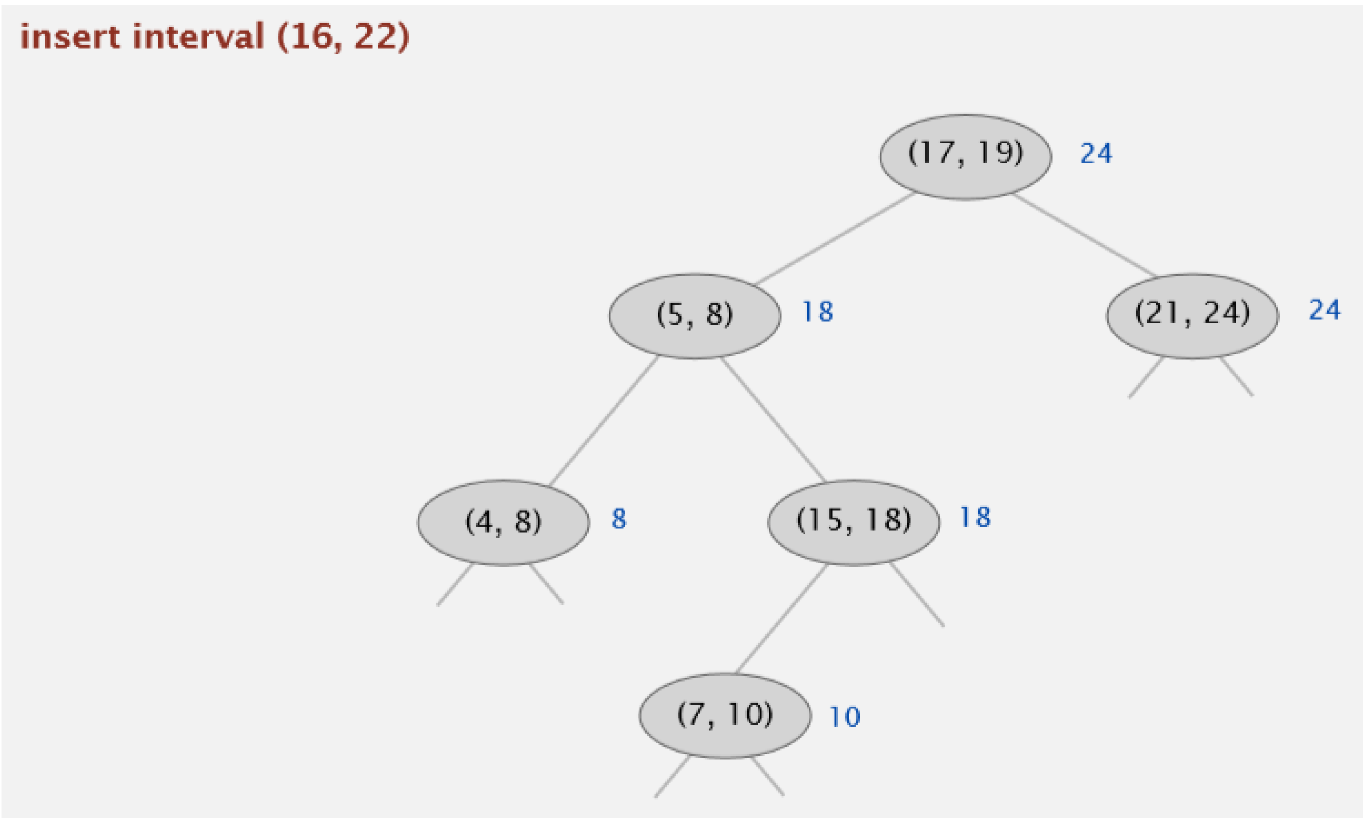
Interval search trees

- Create BST, where each node stores an interval (lo, hi).
  - Use left endpoint as BST key.
  - Store max endpoint in subtree rooted at node.



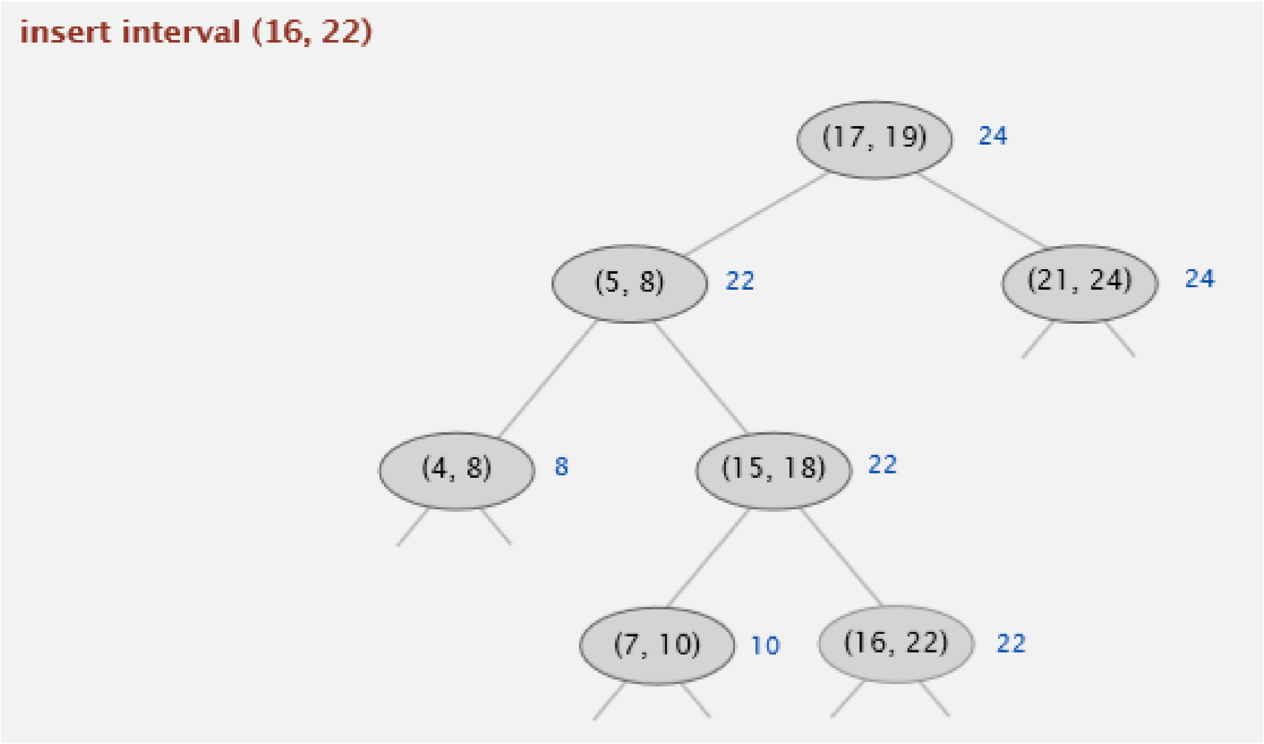
Interval search tree demo: insertion

- To insert an interval(lo, hi):
  - Insert into BST, using lo as the key.
  - Update max in each node on search path.



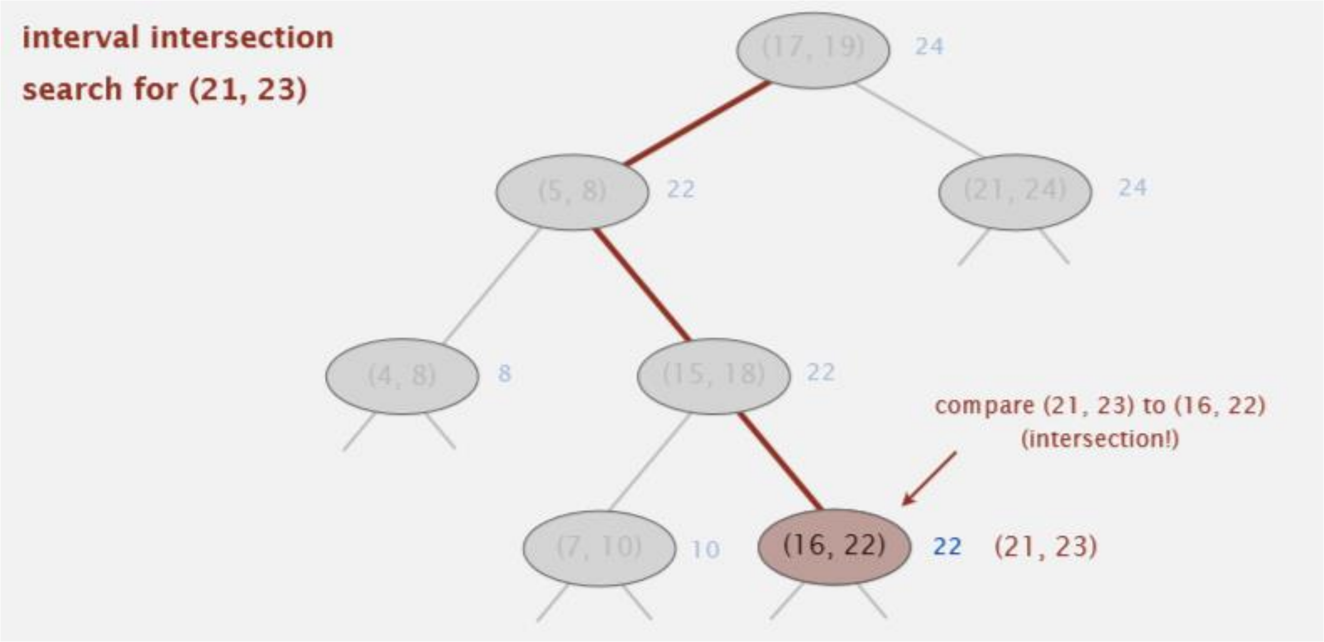
Interval search tree demo: insertion

- To insert an interval(lo, hi):
  - Insert into BST, using lo as the key.
  - Update max in each node on search path.



Interval search tree demo: intersection

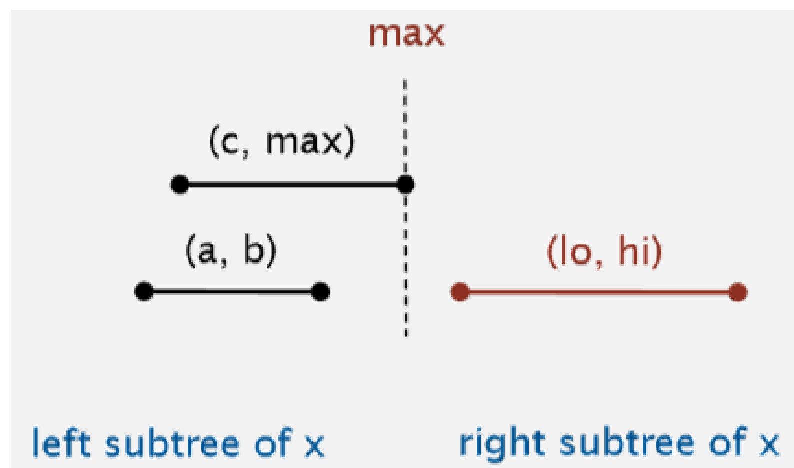
- To search for any one interval that intersects query interval (lo, hi) :
  - If interval in node intersects query interval, return it.
  - Else if left subtree is null, go right.
  - Else if max endpoint in left subtree is less than lo, go right.
  - Else go left.



Search for an intersecting interval: analysis

- To search for any one interval that intersects query interval (lo, hi) :
  - If interval in node intersects query interval, return it.
  - Else if left subtree is null, go right.
  - Else if max endpoint in left subtree is less than lo, go right.
  - Else go left.
- Case 1. If search goes right, then no intersection in left.
- Pf. Suppose search goes right and left subtree is non empty.
  - Since went right, we have max < lo.
  - For any interval (a, b) in left subtree of x, we have b ≤ max < lo.

  - Thus, (a, b) will not intersect (lo, hi).



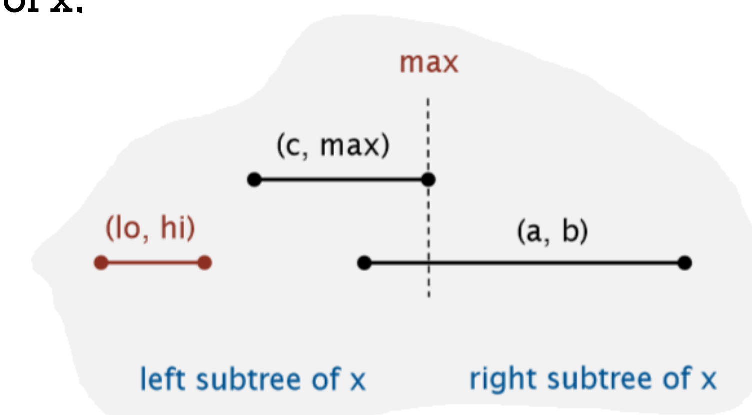
Search for an intersecting interval: analysis

- To search for any one interval that intersects query interval (lo, hi) :
  - If interval in node intersects query interval, return it.
  - Else if left subtree is null, go right.
  - Else if max endpoint in left subtree is less than lo, go right.
  - Else go left.
- Case 2. If search goes left, then there is either an intersection in left subtree or no intersections in either.
- Pf. Suppose no intersection in left.
  - Since went left, we have lo $\leq$ max.
  - Then for any interval (a, b) in right subtree of x,

  $hi < c \leq a \Rightarrow$ no intersection in right.

no intersections in left subtree

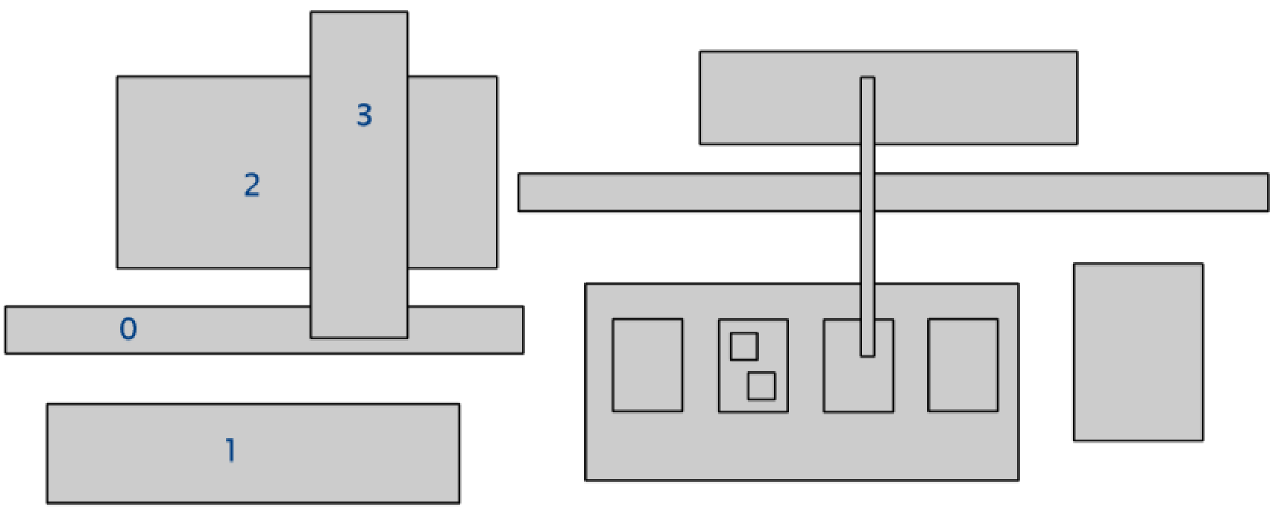
intervals sorted by left endpoint



32 Rectangle Intersection

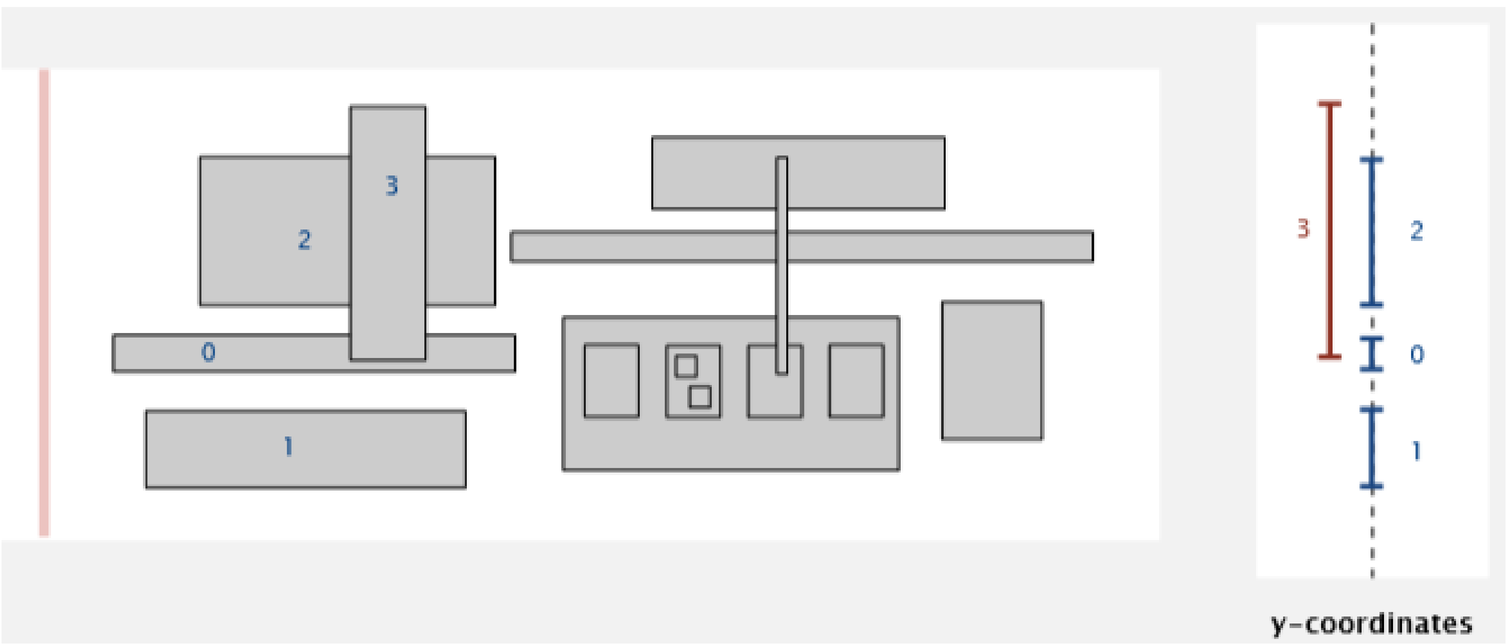
Orthogonal rectangle intersection

- Goal. Find all intersections among a set of N orthogonal rectangles.
- Quadratic algorithm. Check all pairs of rectangles for intersection.
- Assumption : All x- and y-coordinates are distinct.

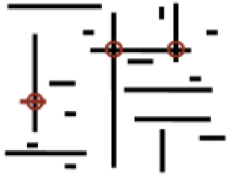
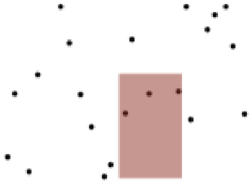
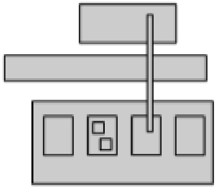


Orthogonal rectangle intersection: sweep-line algorithm

- Sweep vertical line from left to right.
  - x-coordinates of left and right endpoints define events.
  - Maintain set of rectangles that intersect the sweep line in an interval search tree (using y-intervals of rectangle).
  - Left endpoint: interval search for y-interval of rectangle; insert y-interval.
  - Right endpoint: remove y-interval.



Geometric applications of BSTs

problem	example	solution
1d range search		BST
2d orthogonal line segment intersection		sweep line reduces to 1d range search
kd range search		kd tree
1d interval search		interval search tree
2d orthogonal rectangle intersection		sweep line reduces to 1d interval search