Geometric Search

- range search
- quad and kd trees
- intersection search
- VLSI rules check

References: Algorithms 2nd edition, Chapter 26-27

Intro to Algs and Data Structs, Section

---

range search
quad- and kD-trees
intersection search
VLSI rules checking

Overview

Types of data. Points, lines, planes, polygons, circles, ...
This lecture. Sets of N objects.

Geometric problems extend to higher dimensions.
- Good algorithms also extend to higher dimensions.
- Curse of dimensionality.

Basic problems.
- Range searching.
- Nearest neighbor.
- Finding intersections of geometric objects.

1D Range Search

Extension to symbol-table ADT with comparable keys.
- Insert key-value pair.
- Search for key k.
- How many records have keys between k_1 and k_2?
- Iterate over all records with keys between k_1 and k_2.

Application: database queries.

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

Geometric intuition.
- Keys are point on a line.
- How many points in a given interval?

1D Range search: implementations

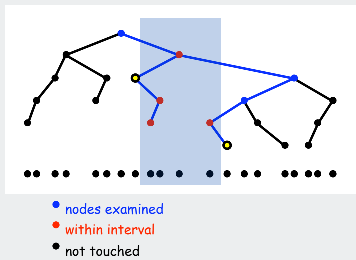
Range search. How many records have keys between k_1 and k_2?

Ordered array. Slow insert, binary search for k_1 and k_2 to find range.
Hash table. No reasonable algorithm (key order lost in hash).

BST. In each node x, maintain number of nodes in tree rooted at x.
Search for smallest element $\geq k_1$ and largest element $\leq k_2$.

	insert	count	range
ordered array	N	log N	R + log N
hash table	1	N	N
BST	log N	log N	R + log N

N = # records
R = # records that match

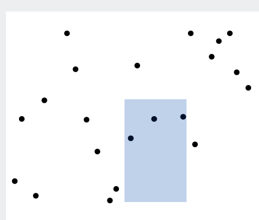


2D Orthogonal Range Search

Extension to symbol-table ADT with 2D keys.
- Insert a 2D key.
- Search for a 2D key.
- Range search: find all keys that lie in a 2D range?
- Range count: how many keys lie in a 2D range?

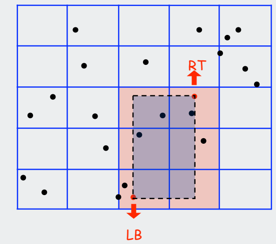
Applications: networking, circuit design, databases.

Geometric interpretation.
- Keys are point in the plane
- Find all points in a given h-v rectangle



2D Orthogonal range Ssearch: Grid implementation

Grid implementation. [Sedgewick 3.18]
- Divide space into M-by-M grid of squares.
- Create linked list for each square.
- Use 2D array to directly access relevant square.
- Insert: insert (x, y) into corresponding grid square.
- Range search: examine only those grid squares that could have points in the rectangle.

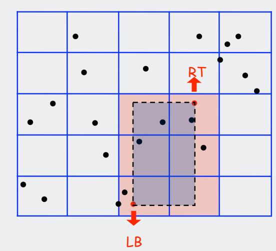


2D Orthogonal Range Search: Grid Implementation Costs

Space-time tradeoff.
- Space: $M^2 + N$.
- Time: $1 + N / M^2$ per grid cell examined on average.

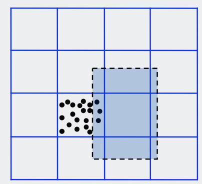
Choose grid square size to tune performance.
- Too small: wastes space.
- Too large: too many points per grid square.
- Rule of thumb: $\sqrt{N}$ by $\sqrt{N}$ grid.

Running time. [if points are evenly distributed]
- Initialize: O(N).
- Insert: O(1). $M \approx \sqrt{N}$
- Range: O(1) per point in range.



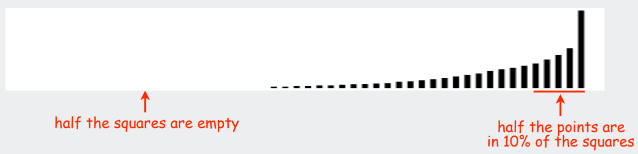
Clustering

Grid implementation. Fast, simple solution for well-distributed points.
Problem. Clustering is a well-known phenomenon in geometric data.



Ex: USA map data.

13,000 points, 1000 grid squares.



Lists are too long, even though average length is short.
Need data structure that gracefully adapts to data.

range searching
quad- and kD-trees
intersection
VLSI rules checking

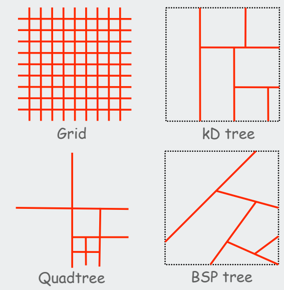
Space Partitioning Trees

Use a tree to represent a recursive subdivision of d-dimensional space.

BSP tree. Recursively divide space into two regions.
Quadtree. Recursively divide plane into four quadrants.
Octree. Recursively divide 3D space into eight octants.
kD tree. Recursively divide k-dimensional space into two half-spaces.

Applications.

- Ray tracing.
- Flight simulators.
- N-body simulation.
- Collision detection.
- Astronomical databases.
- Adaptive mesh generation.
- Accelerate rendering in Doom.
- Hidden surface removal and shadow casting.



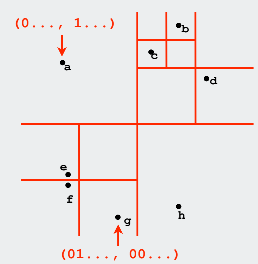
Quadtree

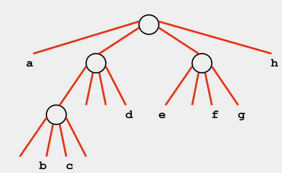
Recursively partition plane into 4 quadrants.

Implementation: 4-way tree.

actually a trie
partitioning on bits of coordinates

```
public class QuadTree
{
   private Quad quad;
   private Value value;
   private QuadTree NW, NE, SW, SE;
}
```



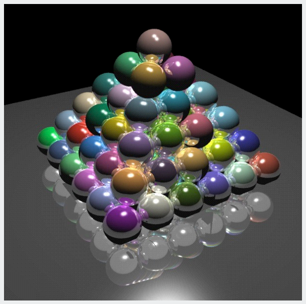
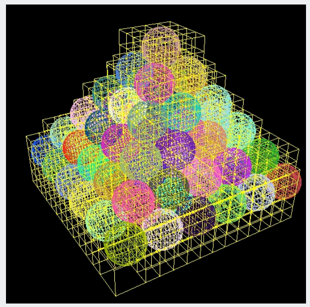


Primary reason to choose quad trees over grid methods:
good performance in the presence of clustering

Curse of Dimensionality

Range search / nearest neighbor in k dimensions?
Main application. Multi-dimensional databases.

3D space. Octrees: recursively divide 3D space into 8 octants.
100D space. Centrees: recursively divide into 2^{100} centrants???

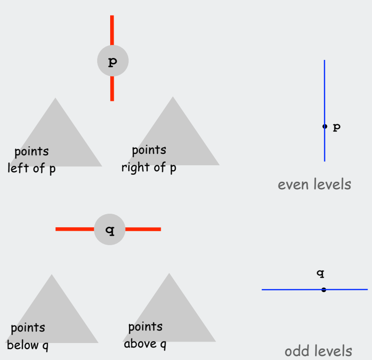
Raytracing with octrees
http://graphics.cs.ucdavis.edu/~gregorsk/graphics/275.html

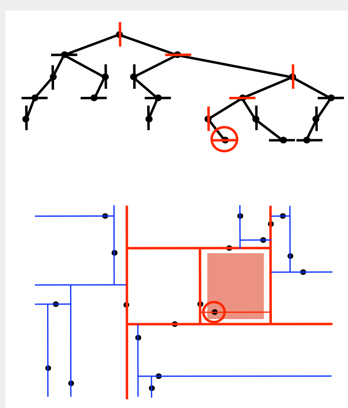
2D Trees

Recursively partition plane into 2 halfplanes.

Implementation: BST, but alternate using x and y coordinates as key.

- Search gives rectangle containing point.
- Insert further subdivides the plane.





Near Neighbor Search

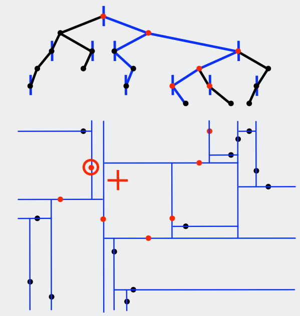
Useful extension to symbol-table ADT for records with metric keys.

- Insert a k dimensional point.
- Near neighbor search: given a point p, which point in data structure is nearest to p?

Need concept of distance, not just ordering.

kD trees provide fast, elegant solution.

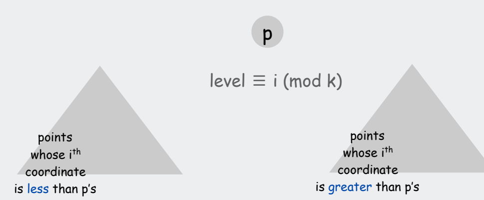
- Recursively search subtrees that could have near neighbor (may search both).
- O(log N) ?

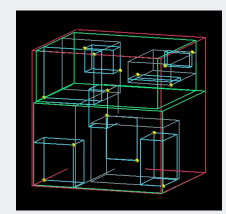


kD Trees

kD tree. Recursively partition k-dimensional space into 2 halfspaces.

Implementation: BST, but cycle through dimensions ala 2D trees.





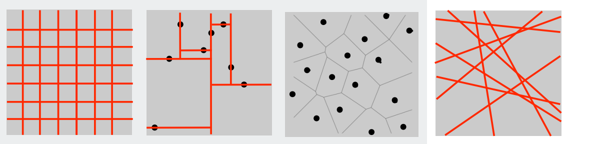
Efficient, simple data structure for processing k-dimensional data.

- adapts well to clustered data.
- adapts well to high dimensional data.
- widely used.
- discovered by an undergrad in an algorithms class!

Summary

Basis of many geometric algorithms: search in a planar subdivision.

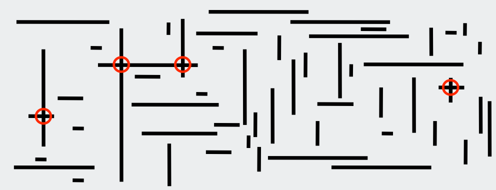
	grid	2D tree	Voronoi diagram	intersecting lines
basis	√N h-v lines	N points	N points	√N lines
representation	2D array of N lists	N-node BST	N-node multilist	~N-node BST
cells	~N squares	N rectangles	N polygons	~N triangles
search cost	1	log N	log N	log N
extend to kD?	too many cells	easy	cells too complicated	use (k-1)D hyperplane



Search for intersections

Problem. Find all intersecting pairs among set of N geometric objects.
Applications. CAD, games, movies, virtual reality.

Simple version: 2D, all objects are horizontal or vertical line segments.



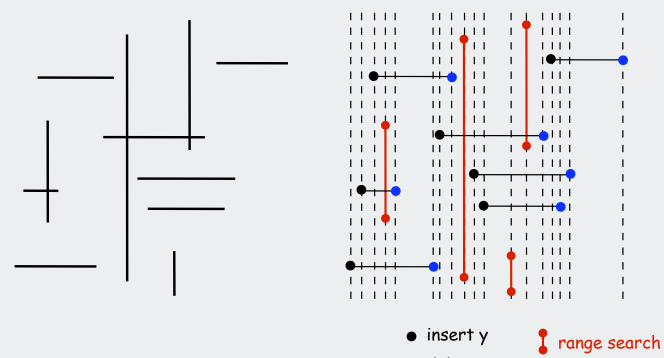
Brute force. Test all $\Theta(N^2)$ pairs of line segments for intersection.
Sweep line. Efficient solution extends to 3D and general objects.

range search
quad- and kD-trees
intersection search
VLSI rules checking

Orthogonal segment intersection search: Sweep-line algorithm

Sweep vertical line from left to right.

- x-coordinates define events.
- left endpoint of h-segment: insert y coordinate into ST.
- right endpoint of h-segment: remove y coordinate from ST.
- v-segment: range search for interval of y endpoints.



Orthogonal Segment Intersection: Sweep Line Algorithm

Reduces 2D orthogonal segment intersection search to 1D range search!

Running time of sweep line algorithm.

- Put x-coordinates on a PQ (or sort). O(N log N)
- Insert y-coordinate into SET. O(N log N)
- Delete y-coordinate from SET. O(N log N)
- Range search. O(R + N log N)

N = # line segments
R = # intersections

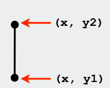
Efficiency relies on judicious use of data structures.

Immutable H-V segment ADT

```
public final class SegmentHV implements Comparable<SegmentHV>
{
   public final int x1, y1;
   public final int x2, y2;

   public SegmentHV(int x1, int y1, int x2, int y2)
   {  ...  }
   public boolean isHorizontal()
   {  ...  }
   public boolean isVertical()
   {  ...  }
   public int compareTo(SegmentHV b)
   {  ...  }
   public String toString()
   {  ...  }
}
```

compare by x-coordinate; break ties by y-coordinate

horizontal segment



vertical segment

Sweep Line Event

```
public class Event implements Comparable<Event>
{
   private int time;
   private SegmentHV segment;

   public Event(int time, SegmentHV segment)
   {
      this.time    = time;
      this.segment = segment;
   }

   public int compareTo(Event b)
   {
      return a.time - b.time;
   }
}
```


Sweep-line algorithm: Initialize events

```
MinPQ<Event> pq = new MinPQ<Event>();

for (int i = 0; i < N; i++)
{
   if (segments[i].isVertical())
   {
      Event e = new Event(segments[i].x1, segments[i]);
      pq.insert(e);
   }
   else if (segments[i].isHorizontal())
   {
      Event e1 = new Event(segments[i].x1, segments[i]);
      Event e2 = new Event(segments[i].x2, segments[i]);
      pq.insert(e1);
      pq.insert(e2);
   }
}
```

initialize PQ

vertical segment

horizontal segment

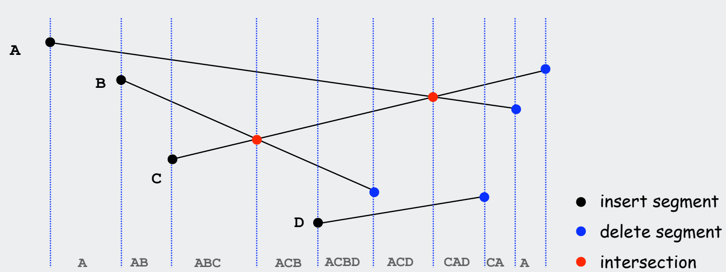
Sweep-line algorithm: Simulate the sweep line

```
int INF = Integer.MAX_VALUE;

SET<SegmentHV> set = new SET<SegmentHV>();

while (!pq.isEmpty())
{
   Event e = pq.delMin();
   int sweep = e.time;
   SegmentHV segment = e.segment;

   if (segment.isVertical())
   {
      SegmentHV seg1, seg2;
      seg1 = new SegmentHV(-INF, segment.y1, -INF, segment.y1);
      seg2 = new SegmentHV(+INF, segment.y2, +INF, segment.y2);
      for (SegmentHV seg : set.range(seg1, seg2))
          System.out.println(segment + " intersects " + seg);
   }

   else if (sweep == segment.x1) set.add(segment);
   else if (sweep == segment.x2) set.remove(segment);
}
```


General line segment intersection search

Use horizontal sweep line moving from left to right.

- Maintain order of segments that intersect sweep line by y-coordinate.
- Intersections can only occur between adjacent segments.
- Add/delete line segment ⇒ one new pair of adjacent segments.
- Intersection ⇒ swap adjacent segments.



order of segments

Line Segment Intersection: Implementation

Efficient implementation of sweep line algorithm.

- Maintain PQ of important x-coordinates: endpoints and intersections.
- Maintain SET of segments intersecting sweep line, sorted by y.
- O(R log N + N log N).

to support "next largest" and "next smallest" queries

Implementation issues.

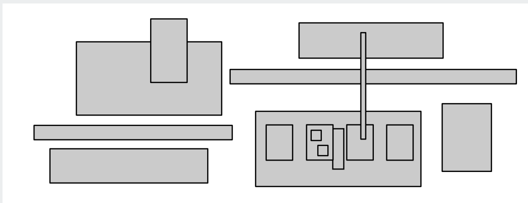
- Degeneracy.
- Floating point precision.
- Use PQ, not presort (intersection events are unknown ahead of time).

range searching
quad- and kD-trees
intersection
VLSI rules checking

Algorithms and Moore's Law

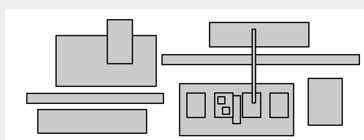
Rectangle intersection search. Find all intersections among h-v rectangles.

Application. Design-rule checking in VLSI circuits.

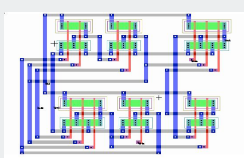


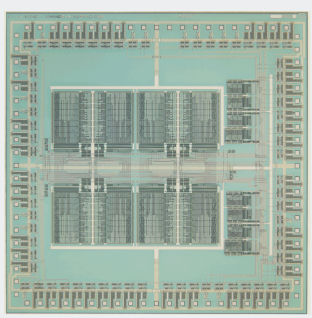
Algorithms and Moore's Law

Early 1970s: microprocessor design became a geometric problem.
- Very Large Scale Integration (VLSI).
- Computer-Aided Design (CAD).



Design-rule checking:
- certain wires cannot intersect
- certain spacing needed between different types of wires
- debugging = rectangle intersection search





Algorithms and Moore's Law

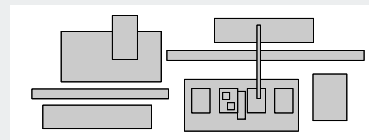
"Moore's Law." Processing power doubles every 18 months.
- 197x: need to check N rectangles.
- 197(x+1.5): need to check 2N rectangles on a 2x-faster computer.

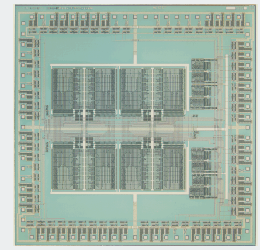
Bootstrapping: we get to use the faster computer for bigger circuits

But bootstrapping is not enough if using a quadratic algorithm
- 197x: takes M days.
- 197(x+1.5): takes (4M)/2 = 2M days. (!)

quadratic algorithm

2x-faster computer

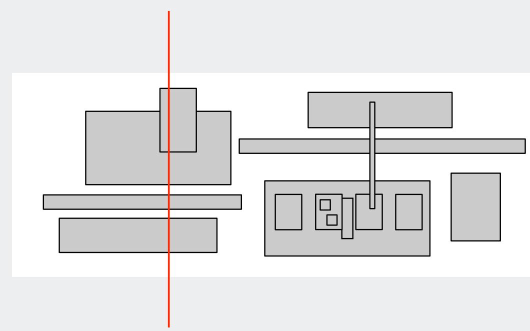




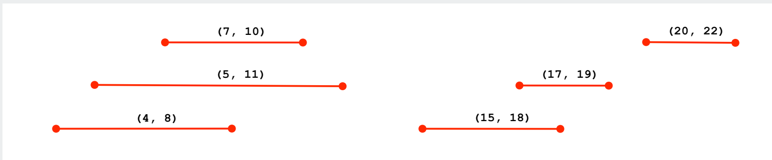
O(N log N) CAD algorithms are necessary to sustain Moore's Law.

Rectangle intersection search

Move a vertical "sweep line" from left to right.
- Sweep line: sort rectangles by x-coordinate and process in this order, stopping on left and right endpoints.
- Maintain set of intervals intersecting sweep line.
- Key operation: given a new interval, does it intersect one in the set?



Interval Search Trees

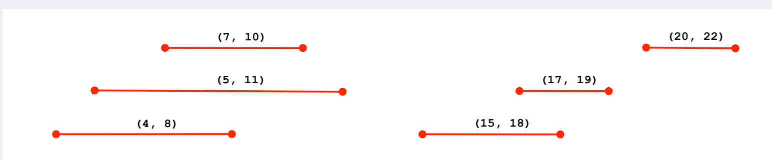


Support following operations.

- Insert an interval `(lo, hi)`.
- Delete the interval `(lo, hi)`.
- Search for an interval that intersects `(lo, hi)`.

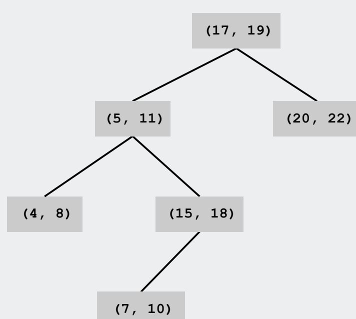
Non-degeneracy assumption. No intervals have the same x-coordinate.

Interval Search Trees

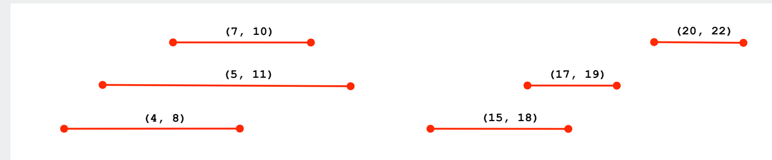


Interval tree implementation with BST.

- Each BST node stores one interval.
- use `lo` endpoint as BST key.

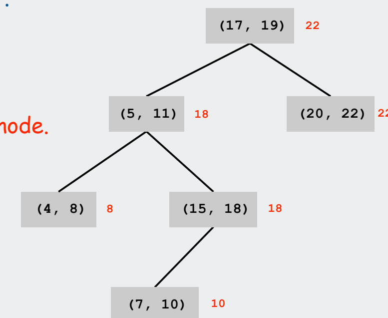


Interval Search Trees



Interval tree implementation with BST.

- Each BST node stores one interval.
- BST nodes sorted on `lo` endpoint.
- Additional info: store and maintain max endpoint in subtree rooted at node.



Finding an intersecting interval

Search for an interval that intersects `(lo, hi)`.

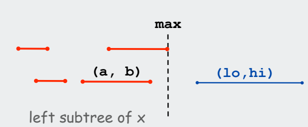
```
Node x = root;
while (x != null)
{
   if (x.interval.intersects(lo, hi)) return x.interval;
   else if (x.left == null)   x = x.right;
   else if (x.left.max < lo)  x = x.right;
   else                       x = x.left;
}
return null;
```

Case 1. If search goes right, no overlap in left.

- `(x.left == null)` ⇒ trivial.
- `(x.left.max < lo)` ⇒ for any interval `(a, b)` in left subtree of x, we have $b \le max < lo$.

defn of max

reason for going right



Finding an Intersecting Interval

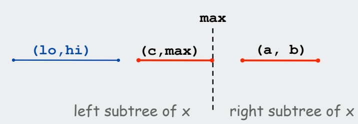
Search for an interval that intersects `(lo, hi)`.

```
Node x = root;
while (x != null)
{
   if (x.interval.intersects(lo, hi)) return x.interval;
   else if (x.left == null)   x = x.right;
   else if (x.left.max < lo)  x = x.right;
   else                       x = x.left;
}
return null;
```

Case 2. If search goes left, then either (i) there is an intersection in left subtree or (ii) no intersections in either subtree.
Pf. Suppose no intersection in left. Then for any interval `(a, b)` in right subtree, $a \ge c > hi$ $\Rightarrow$ no intersection in right.

intervals sorted by left endpoint

no intersection in left subtree



Interval Search Tree: Analysis

Implementation. Use a red-black tree to guarantee performance.

can maintain auxiliary information using log N extra work per op

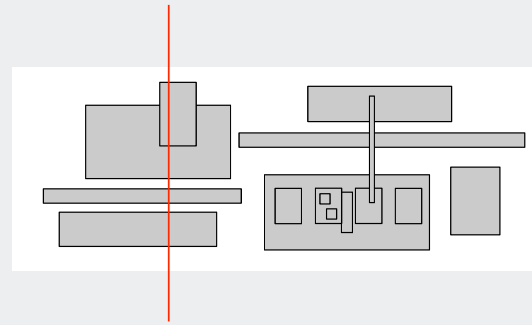
Operation	Worst case
insert interval	log N
delete interval	log N
find an interval that intersects (lo, hi)	log N
find all intervals that intersect (lo, hi)	R log N

N = # intervals
R = # intersections

Rectangle intersection sweep-line algorithm: Review

Move a vertical "sweep line" from left to right.

- Sweep line: sort rectangles by x-coordinates and process in this order.
- Store set of rectangles that intersect the sweep line in an interval search tree (using y-interval of rectangle).
- Left side: interval search for y-interval of rectangle, insert y-interval.
- Right side: delete y-interval.



VLSI Rules checking: Sweep-line algorithm (summary)

Reduces 2D orthogonal rectangle intersection search to 1D interval search!

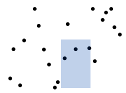
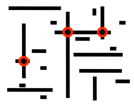
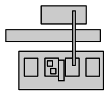
Running time of sweep line algorithm.

- Sort by x-coordinate. O(N log N)
- Insert y-interval into ST. O(N log N)
- Delete y-interval from ST. O(N log N)
- Interval search. O(R log N)

N = # line segments
R = # intersections

Efficiency relies on judicious extension of BST.

Geometric search summary: Algorithms of the day

1D range search		BST
kD range search		kD tree
1D interval intersection search		interval tree
2D orthogonal line intersection search		sweep line reduces to 1D range search
2D orthogonal rectangle intersection search		sweep line reduces to 1D interval intersection search