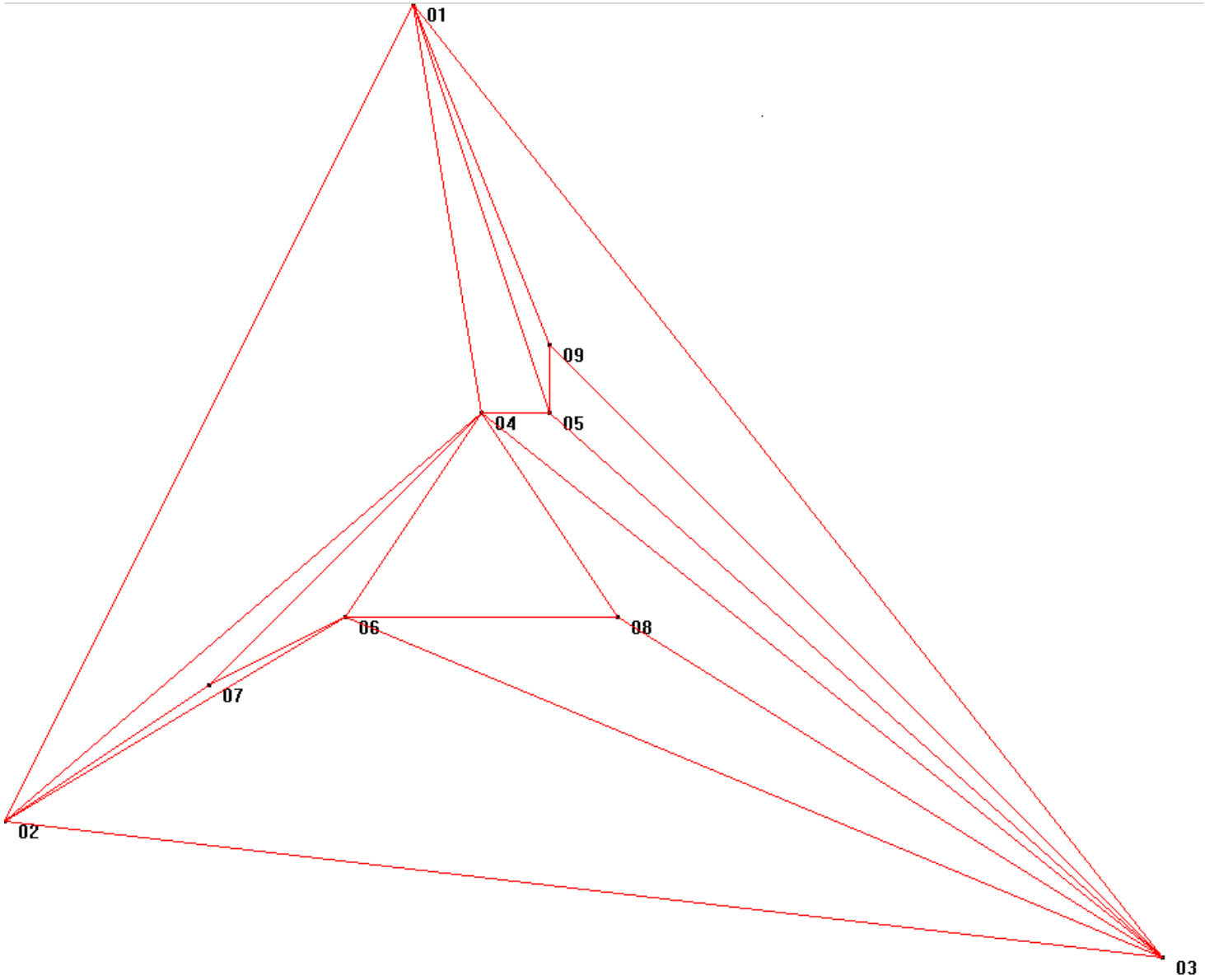
COS 397 – week 11. GEOMETRIC ALGORITHMS (CONTINUED)

Simple Triangulation of a set of points

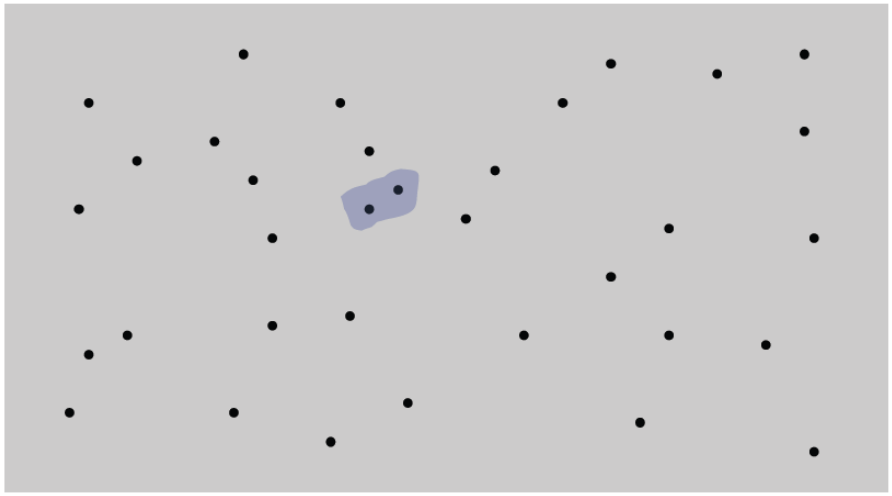
```
struct point{int x; int y; char c[3];};
point p[99];
int n;
void read_data();
void showL(int i, int j);
int ccw(point p0, point p1, point p2)
{ int dx1=p1.x-p0.x;
  int dy1=p1.y-p0.y;
  int dx2=p2.x-p0.x;
  int dy2=p2.y-p0.y;
  return dx1*dy2-dy1*dx2;
}
```

```
int sign(int x)
{
  if(x>0) return 1; if(x<0) return -1; return 0;
}
bool insideTriangle(point p, point a, point b, point c)
{
 int e=sign(ccw(p,a,b));
 if(e==0) return false;
 if(e!=sign(ccw(p,b,c))) return false;
 if(e!=sign(ccw(p,c,a))) return false;
 return true;
}
int find(int a, int b, int c)
{
  for(int k=1;k<=n;k++)
   if(insideTriangle(p[k],p[a],p[b],p[c])) return k;
  return 0;
}
```

```
void tri(int a, int b, int c)
{
  int d = find(a,b,c);
  if(d>0)
   {
     showL(a,d);
     showL(b,d);
     showL(c,d);
     tri(d,b,c);
     tri(a,d,c);
     tri(a,b,d);
   }
}
int main()
{
 Read_data();
 tri(1,2,3);
}
```

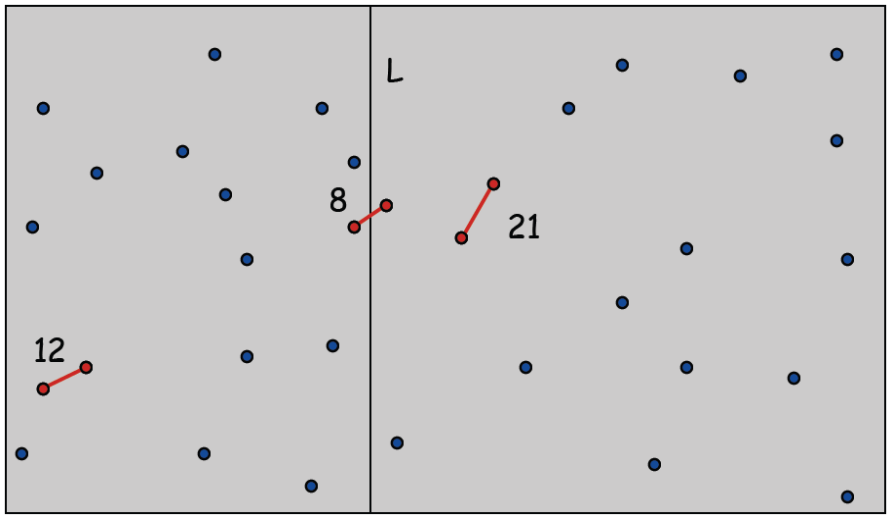

01
09
04
05
06
08
07
02
03

Closest Pair of Points



Algorithm.

- Divide: draw vertical line L so that roughly N/2 points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return the best of 3 solutions.



Find closest pair with one point in each side, assuming that distance < δ.

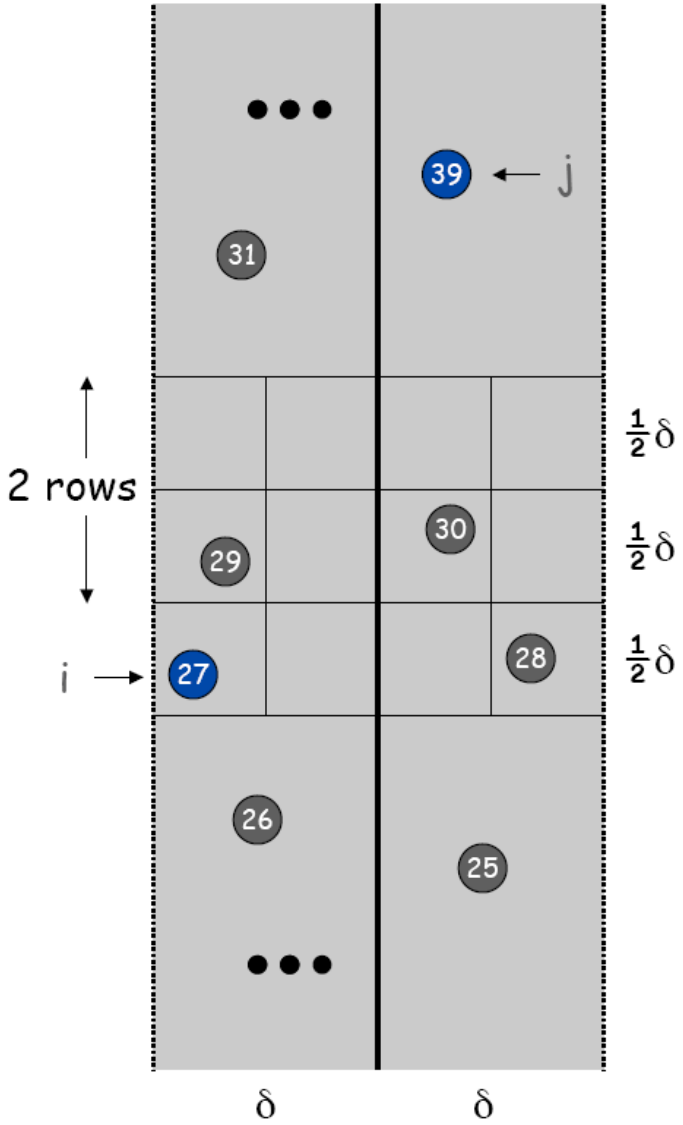
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ - strip by their y coordinate.
- Only check distances of those within 11 positions in a sorted list**!**

Let s_i be the point in the 2δ-strip, with the i-th smallest y-coordinate.
Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ.

Proof.

No two points lie in same δ/2-by- δ/2 box.
Two points at least 2 rows apart have distance $\geq 2(\delta/2)$

Fact. Still true if we replace 12 with 7.



Closest Pair Algorithm

```
Closest-Pair(p1, …, pn) {
   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

O(N log N)

2T(N / 2)

O(N)

O(N log N)

O(N)

Closest Pair of Points: Analysis

Running time.

$$T(N) \leq 2T(N/2) + O(N \log N) \Rightarrow T(N) = O(N \log^2 N)$$

Upper bound. Can be improved to O(N log N).

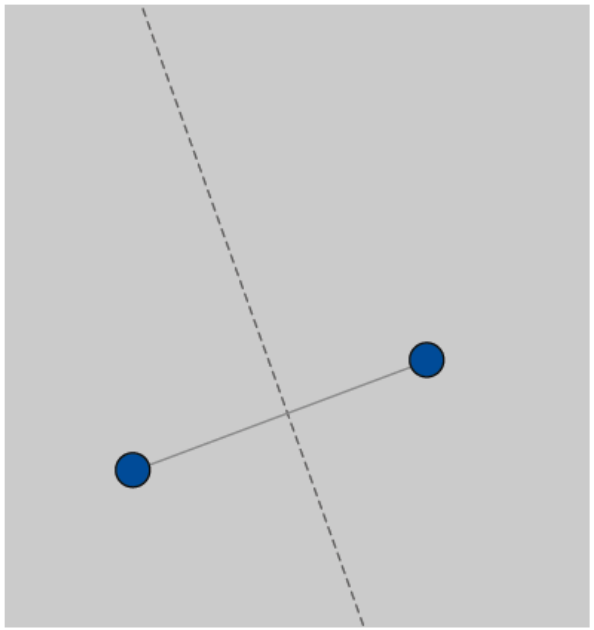
Lower bound. In quadratic decision tree model, any algorithm for closest pair requires
O(N log N) steps.

Nearest Neighbor

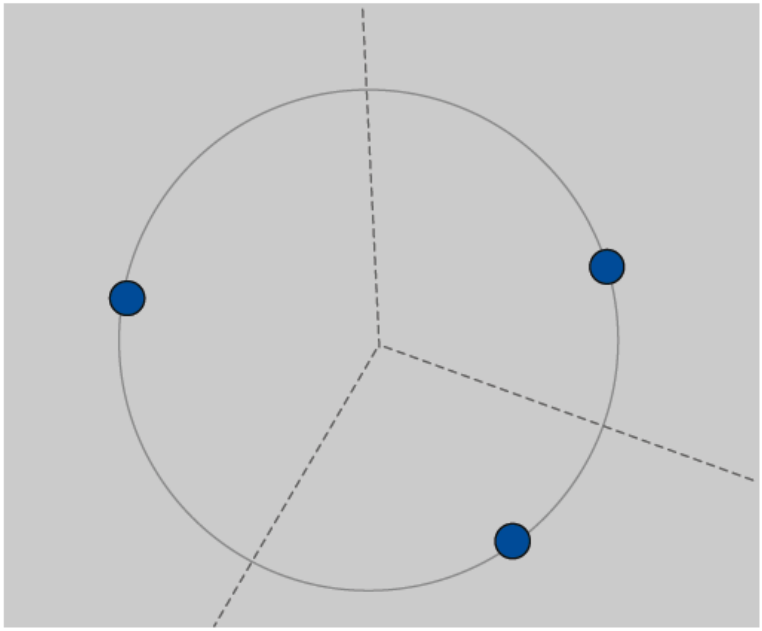
Nearest neighbor problem. Given a query point p, which one of original N points is closest to p?

Algorithm	Preprocess	Query
Brute	1	N
Goal	N log N	log N

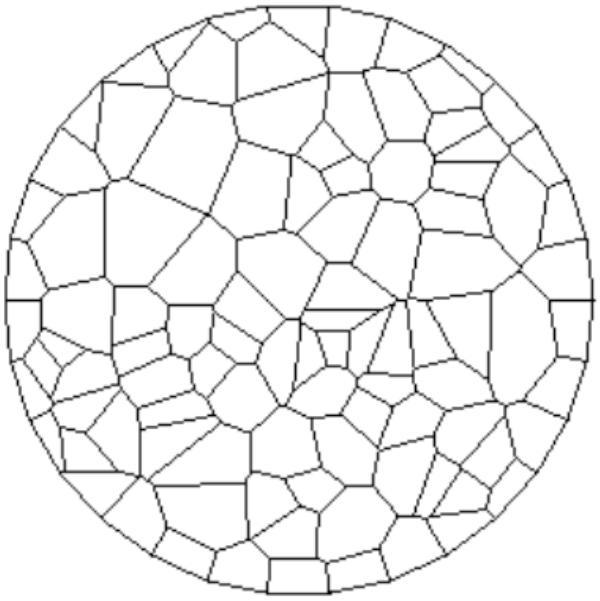
Voronoi region. Set of all points closest to a given point.
Voronoi diagram. Planar subdivision delineating Voronoi regions.
Fact. Voronoi edges are perpendicular bisector segments.

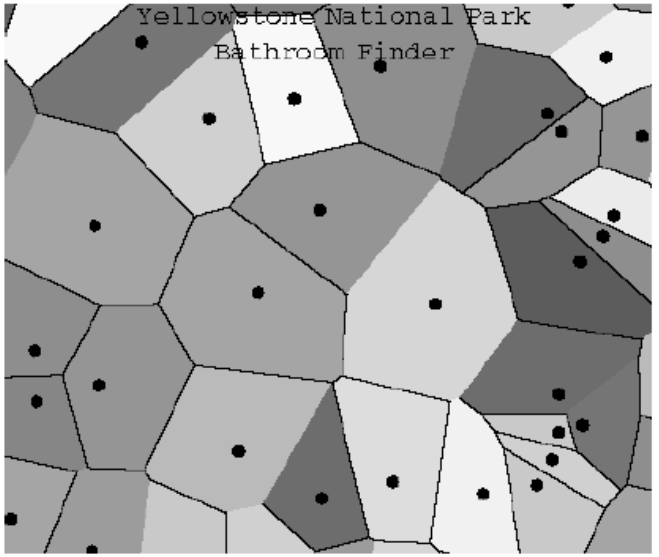


Voronoi of 2 points
(perpendicular bisector)



Voronoi of 3 points
(passes through circumcenter)





Voronoi Diagram: Applications

Toxic waste dump problem. N homes in a region. Where to locate nuclear power plant so that it is far away from any home as possible?

Answer: looking for largest empty circle (center must lie on Voronoi diagram)

Path planning. Circular robot must navigate through environment with N obstacle points. How to minimize risk of bumping into an obstacle?

Answer: robot should stay on Voronoi diagram of obstacles

Voronoi Diagram: More Applications:

Anthropology. Identify influence of clans and chiefdoms on geographic regions.
Astronomy. Identify clusters of stars and clusters of galaxies.
Biology, Ecology, Forestry. Model and analyze plant competition.
Cartography. Piece together satellite photographs into large "mosaic" maps.
Crystallography. Study Wigner-Setiz regions of metallic sodium.
Data visualization. Nearest neighbor interpolation of 2D data.
Finite elements. Generating finite element meshes which avoid small angles.
Fluid dynamics. Vortex methods for inviscid incompressible 2D fluid flow.
Geology. Estimation of ore reserves in a deposit using info from bore holes.
Geo-scientific modeling. Reconstruct 3D geometric figures from points.
Marketing. Model market of US metro area at individual retail store level.
Metallurgy. Modeling "grain growth" in metal films.
Physiology. Analysis of capillary distribution in cross-sections of muscle tissue.

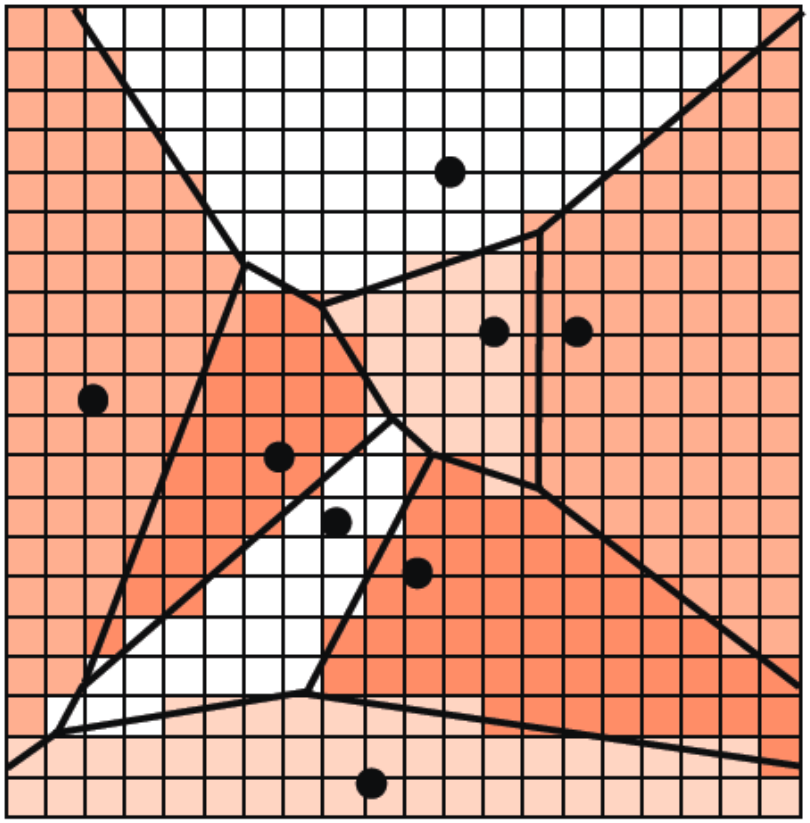
Robotics. Path planning for robot to minimize risk of collision. Typography. Character recognition, beveled and carved lettering. Zoology. Model and analyze the territories of animals.

Scientific Rediscoveries

Year	Discoverer	Discipline	Name
1644	Descartes	Astronomy	"Heavens"
1850	Dirichlet	Math	Dirichlet tesselation
1908	Voronoi	Math	Voronoi diagram
1909	Boldyrev	Geology	area of influence polygons
1911	Thiessen	Meteorology	Thiessen polygons
1927	Niggli	Crystallography	domains of action
1933	Wigner-Seitz	Physics	Wigner-Seitz regions
1958	Frank-Casper	Physics	atom domains
1965	Brown	Ecology	area of potentially available
1966	Mead	Ecology	plant polygons
1985	Hoofd et al.	Anatomy	capillary domains

Discretized Voronoi. Solve nearest neighbor problem on an N-by-N grid.

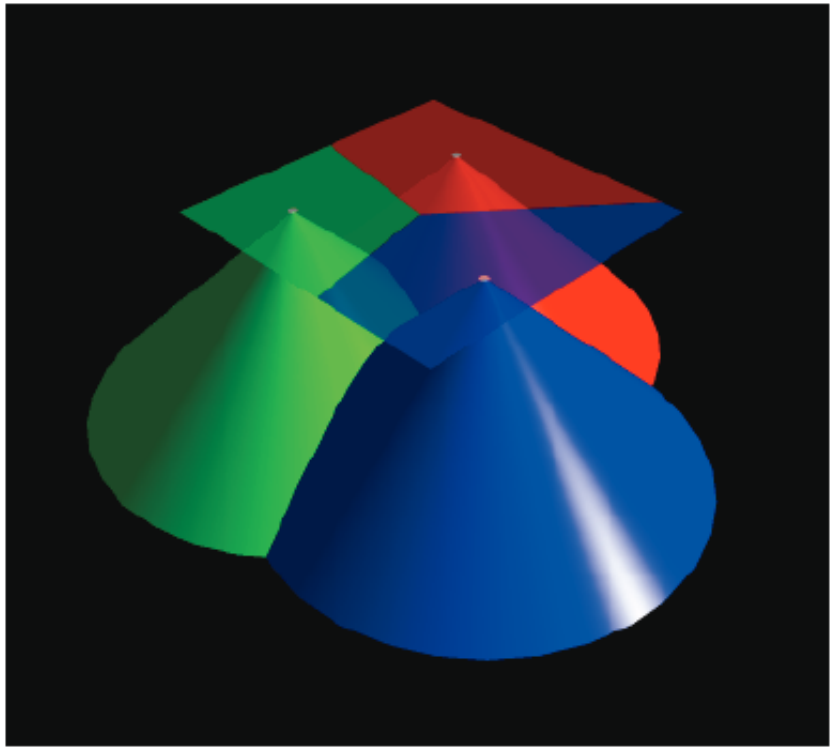
Brute force. For each grid cell, maintain closest point. When adding a new point to Voronoi, update N^2 cells.

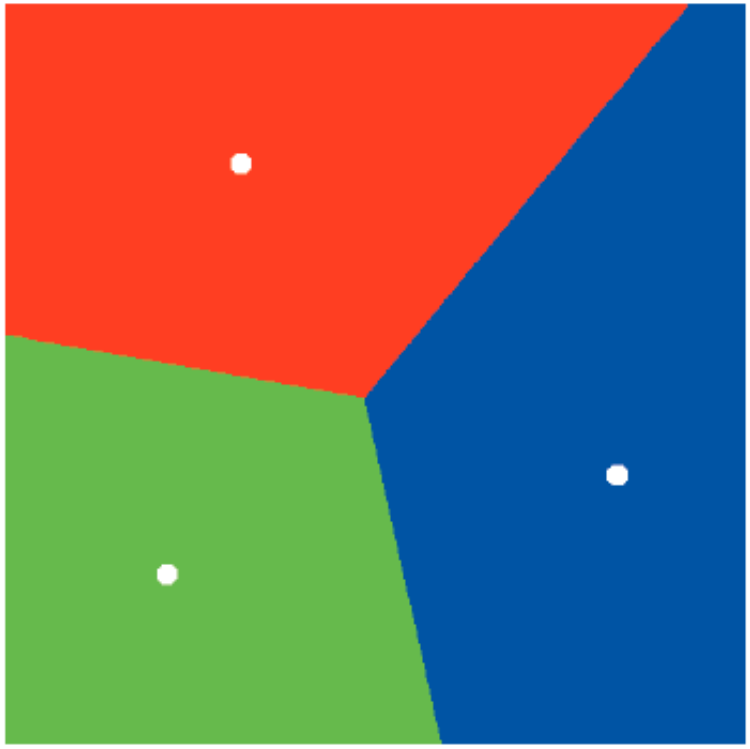


Hoff's algorithm. Align apex of a right circular cone with sites.

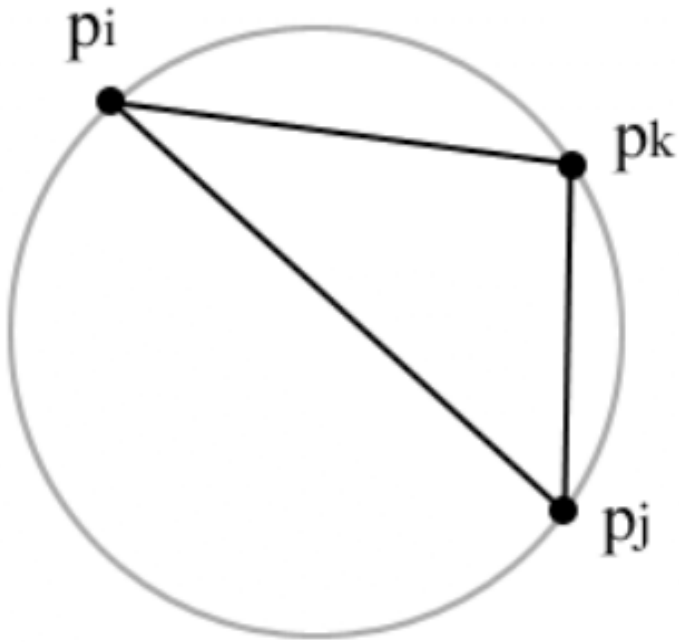
- Minimum envelope of cone intersections projected onto plane is the Voronoi diagram.
- View cones in different colors → render Voronoi.

Implementation. Draw cones using standard graphics hardware!

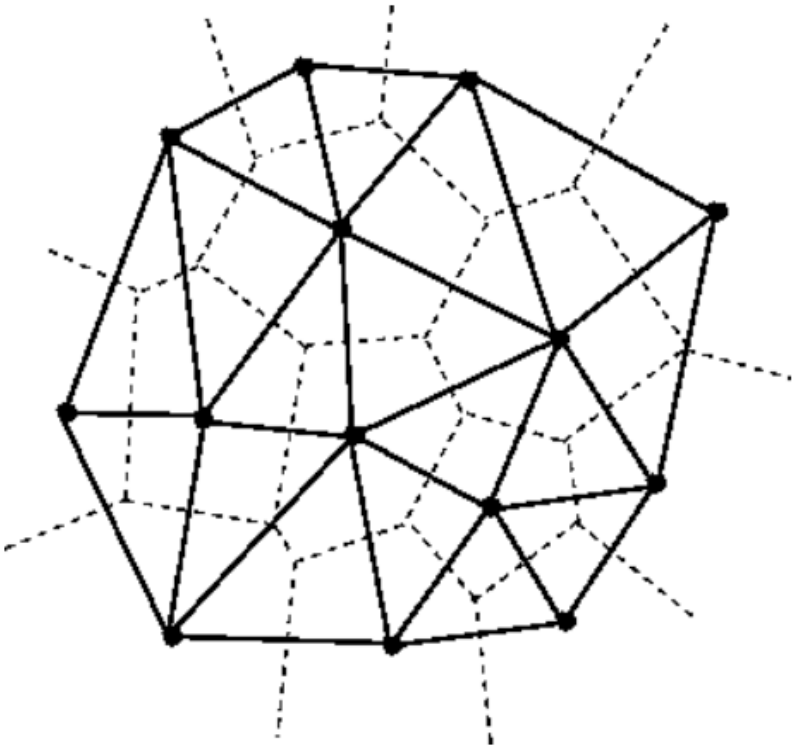




Delaunay triangulation. Triangulation of N points such that no point is inside circumcircle of any other triangle.



Fact 0. It exists and is unique (assuming no degeneracy).
Fact 1. Dual of Voronoi (connect adjacent points in Voronoi diagram).
Fact 2. No edges cross
Fact 3. Maximizes the minimum angle for all triangular elements.
Fact 4. Boundary of Delaunay triangulation is convex hull.
Fact 5. Shortest Delaunay edge connects closest pair of points.



Delaunay

Voronoi

Summary. Many fundamental geometric problems require ingenuity to solve large instances.

Problem	Brute	Cleverness
convex hull	N^2	N log N
closest pair	N^2	N log N
furthest pair	N^2	N log N
Delaunay triangulation	N^4	N log N
polygon triangulation	N^2	N

asymptotic time to solve a 2D problem with N points

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 k1 and k2?
- Iterate over all records with keys between k1 and k2.

Application: database queries.

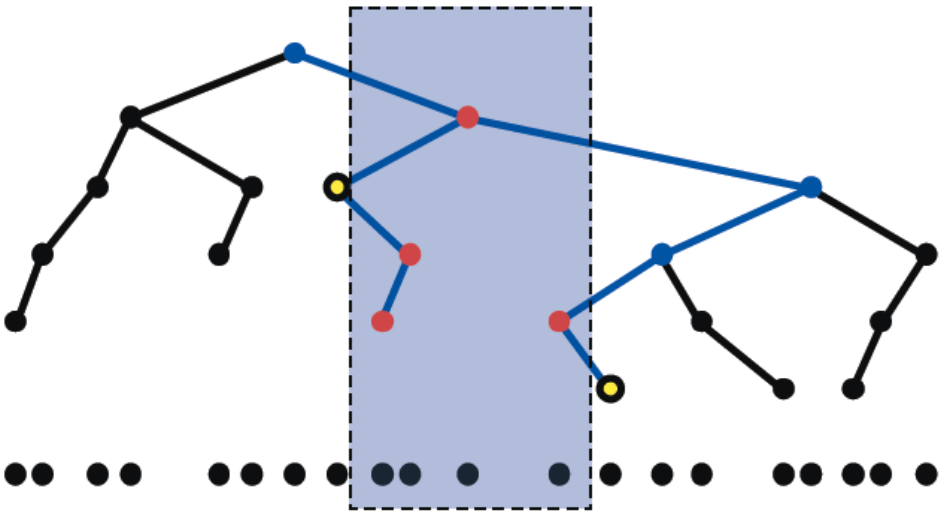
insert B	B
insert D	B D
insert A	A B D
insert I	A B D I
insert H	A B D H I
insert F	A B D F H I
insert P	A B D F H I P
count G to K	2
search G to K	H I

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 k1 and k2?
Ordered array. Slow insert, binary search for k1 and k2 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 the smallest element ≥ k1 and the largest element ≤ k2.

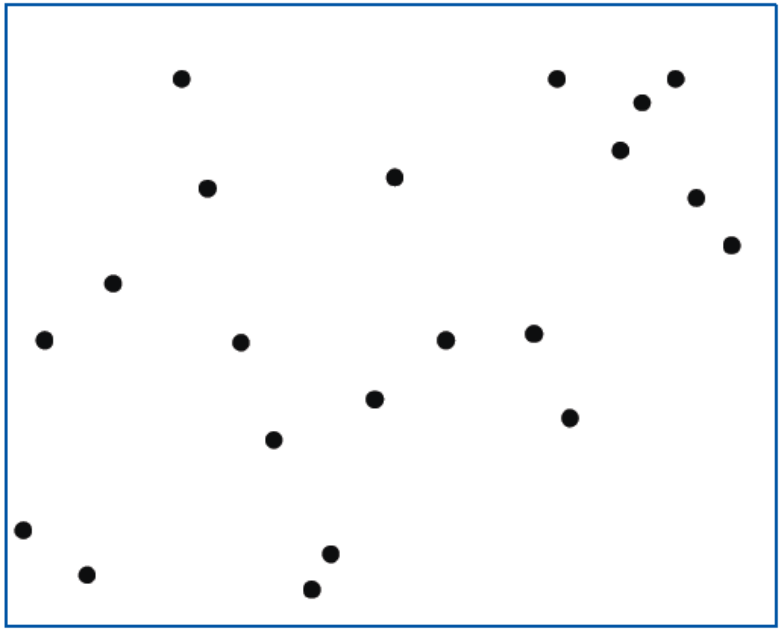


- nodes examined
- within interval
- not touched

	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.

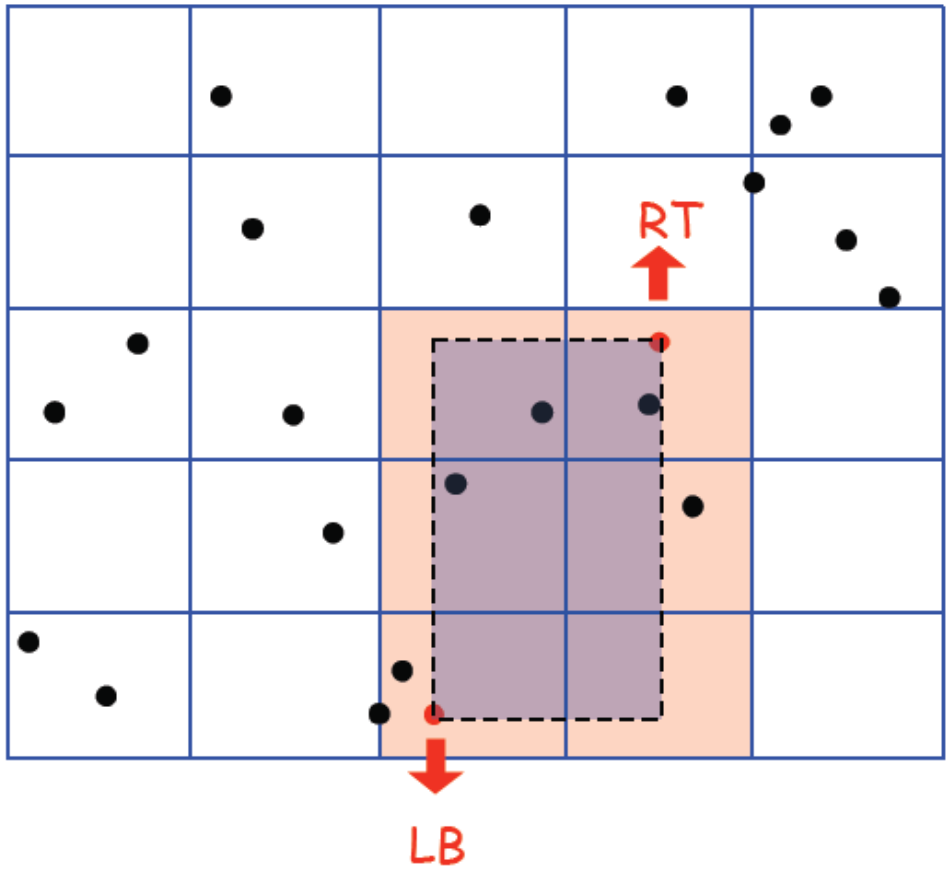
Geometric interpretation.

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

2D Orhtogonal Range Search: Grid Implementation

Grid implementation.

- 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.



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.

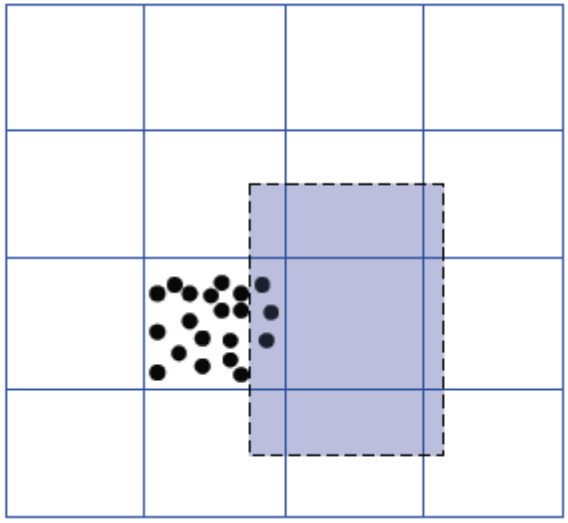
Running time. [if points are evenly distributed]

- Initialize: O(N).
- Insert: O(1).
- 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.



Example: USA map data.

- 80,000 points, 20,000 grid squares.
- Half the grid squares are empty.
- Half the points have ≥ 10 others in same grid square.
- Ten percent have ≥ 100 others in same grid square.

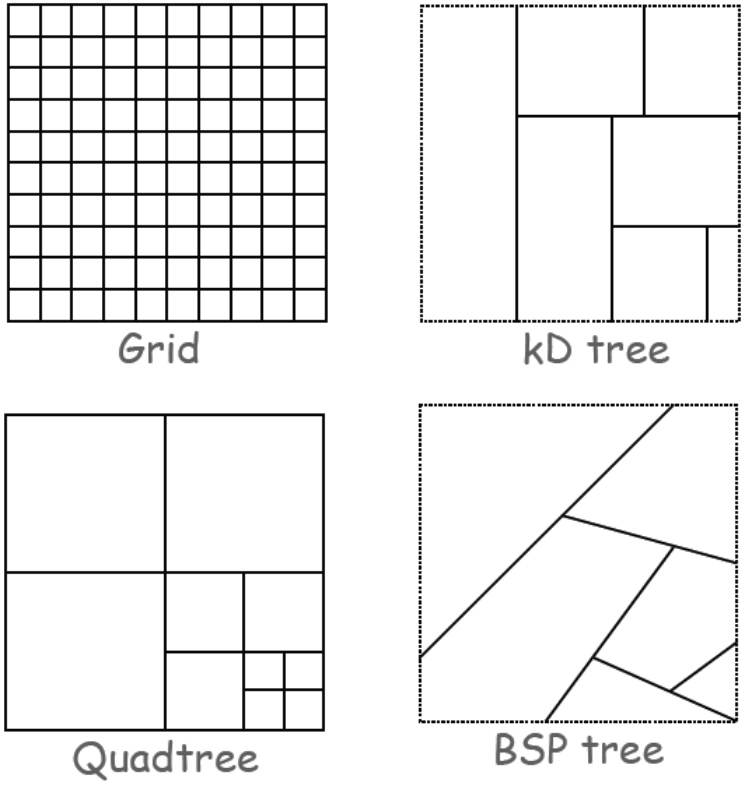
Need data structure that gracefully adapts to data.

Space partitioning tree. Use a tree to represent the recursive hierarchical subdivision of d-dimensional space.

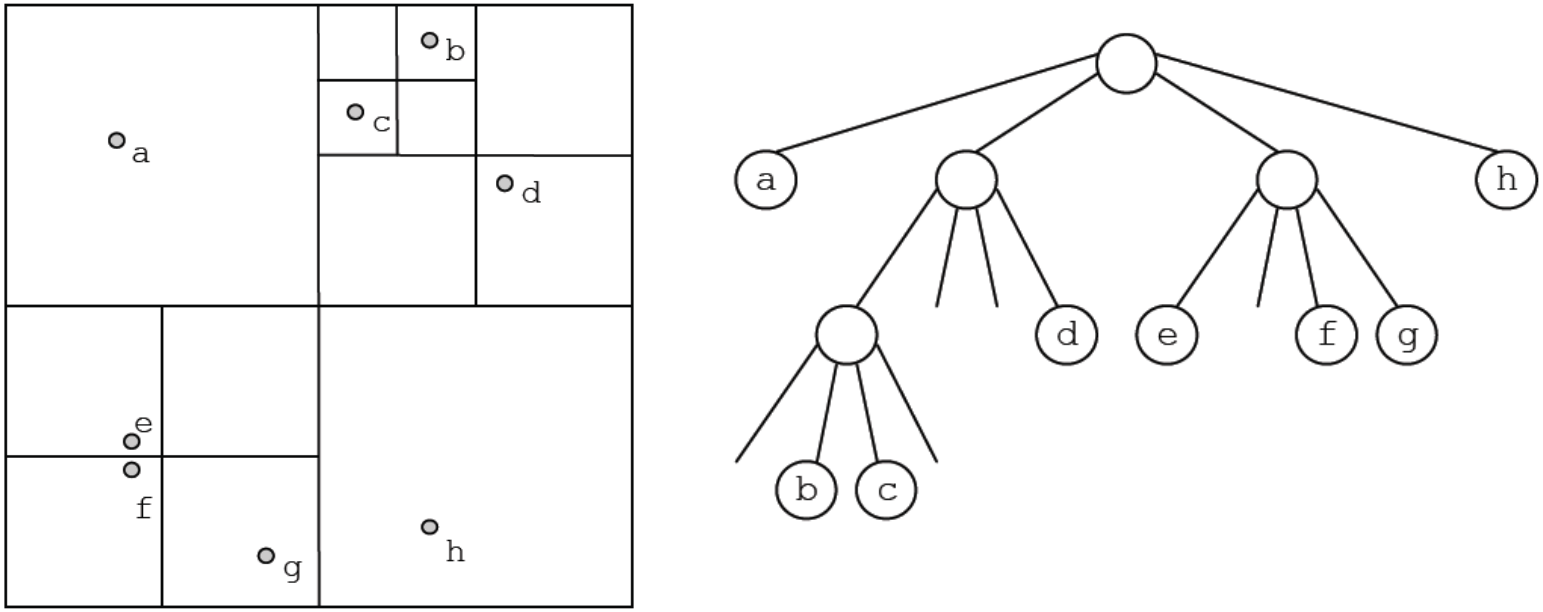
- BSP tree. Recursively divide space into two regions.
- Quadtree. Recursively divide plane into four quadrants.
- Octree. Recursively divide 3D space into eight octants.
- kD tree. Recursively divide k-dimensional space into two half-spaces.

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.

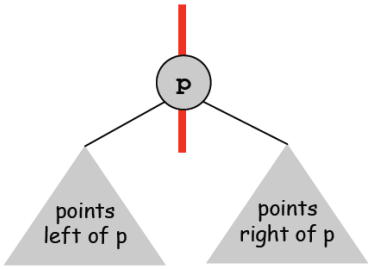


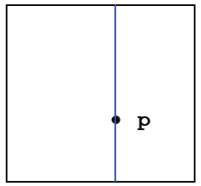
Quad tree.
Recursively partition plane into 4 quadrants.
Implementation: 4-way tree.
Good clustering performance is a primary reason to choose quad trees over grid methods.



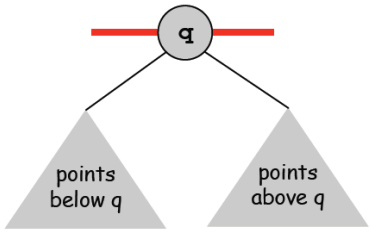
2D tree. Recursively partition plane into 2 halfplanes.
Implementation: BST, but alternate using x and y coordinates as key.

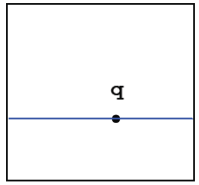
- Search gives rectangle containing point.
- Insert further subdivides the plane.



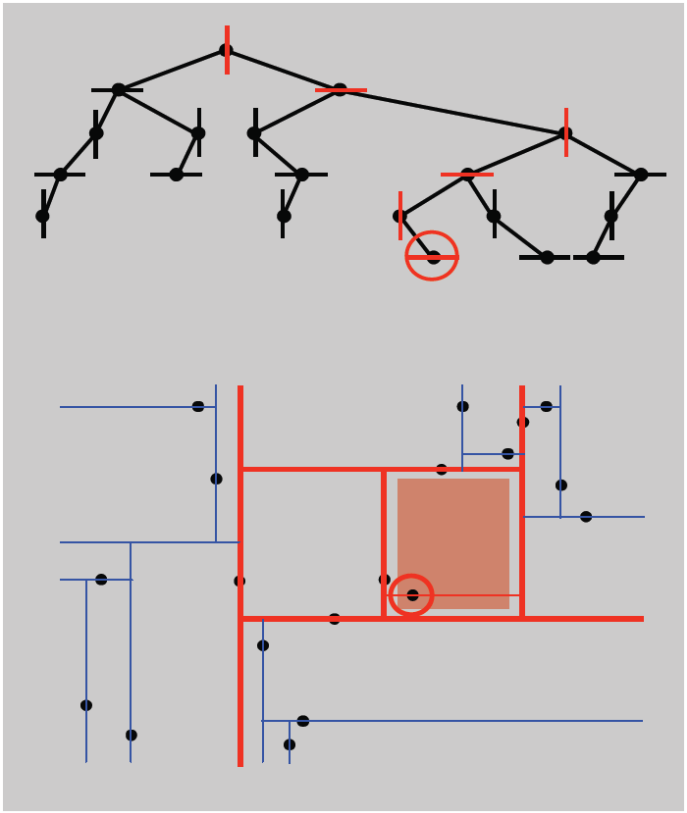


even levels



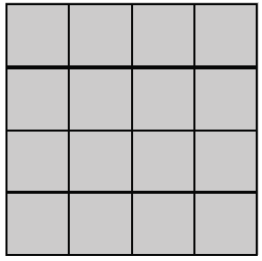
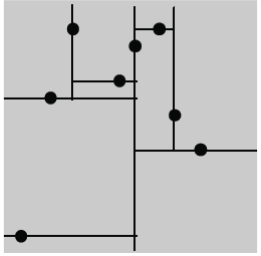
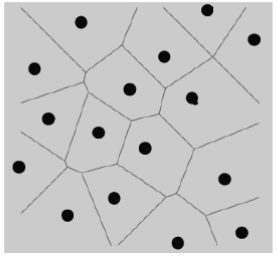
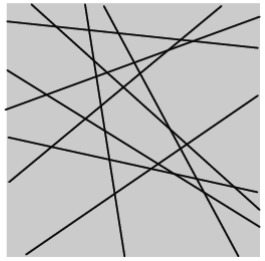


odd levels



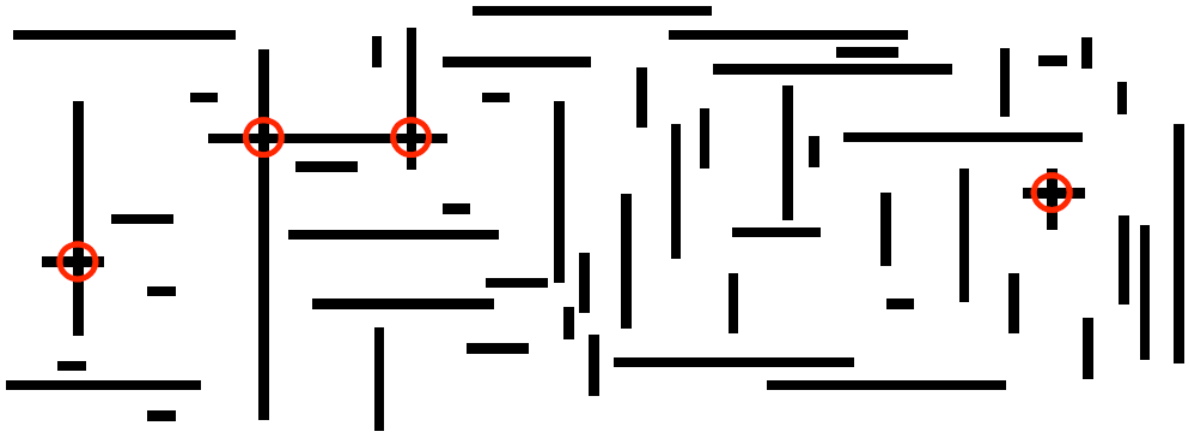
Basis of many geometric algorithms: search in a planar subdivision.

	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

Geometric Intersection

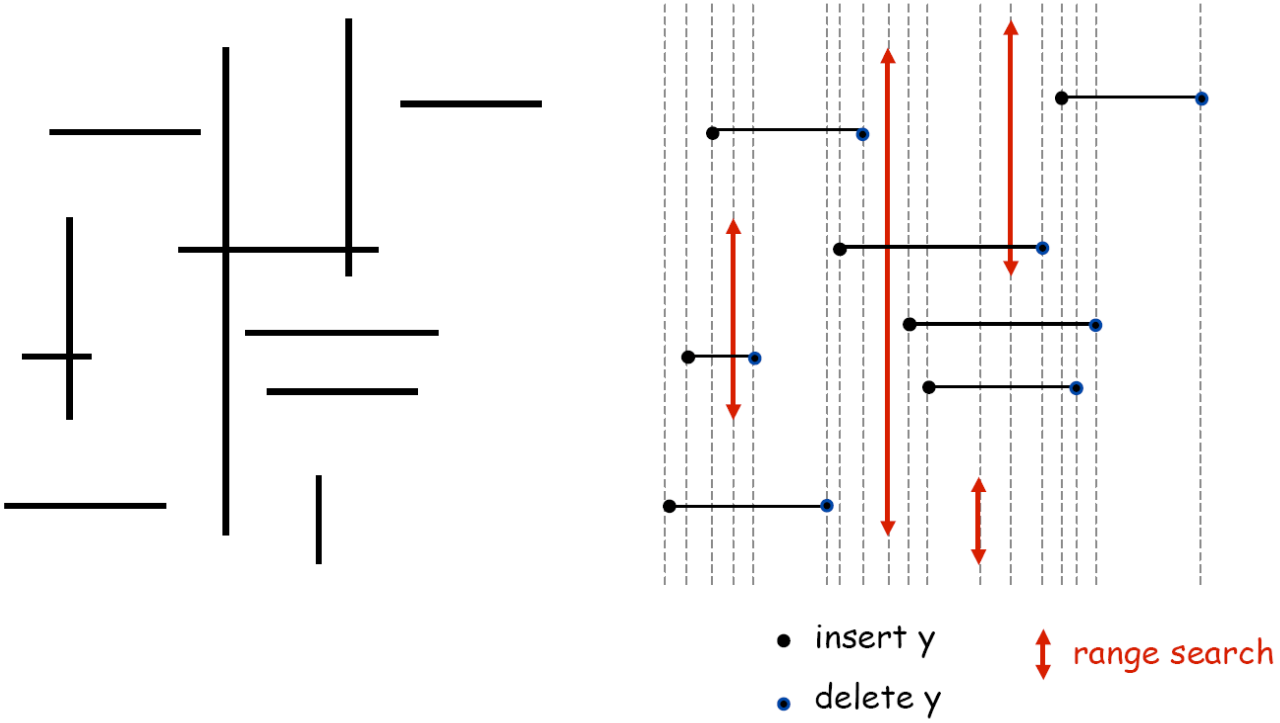
Problem. Find all intersecting pairs among set of N geometric objects.
Applications. CAD, games, movies, virtual reality.
Simple version: 2D, all objects are horizontal or vertical line segments.



Brute force. Test all $O(N^2)$ pairs of line segments for intersection.

Sweep line. Efficient solution extends to 3D and general objects.
Sweep vertical line from left to right.

- Event times: x-coordinates of h-v line segments.
- 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

Sweep line: reduces 2D orthogonal segment intersection problem to 1D range searching!

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.

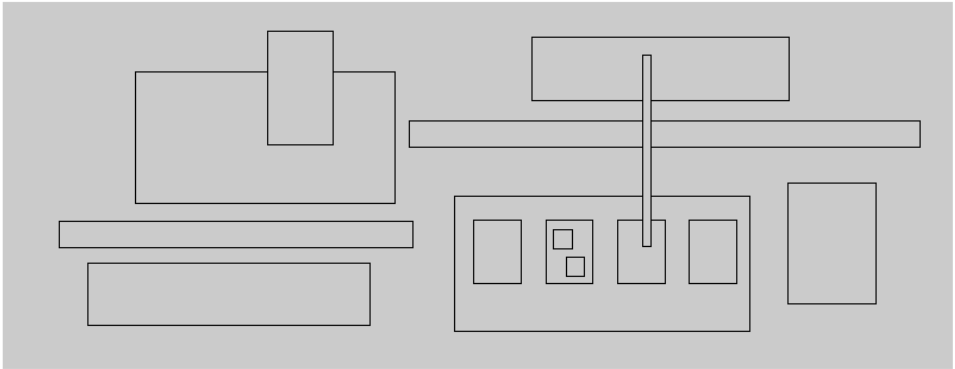
VLSI Rules Checking

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

Application. VLSI rules checking in microprocessor design.

Early 1970s: microprocessor design became a geometric problem.

- Very Large Scale Integration (VLSI).
- Computer-Aided Design (CAD).
- Design-rule checking.



"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.

Quadratic algorithm. Compare each rectangle against all others.

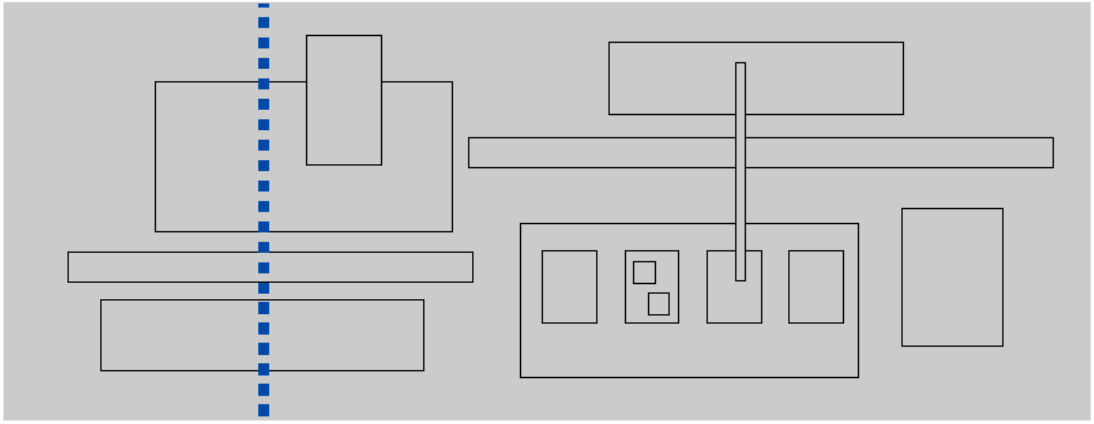
- 197x: takes M days.
- 197(x+1.5): takes (4M)/2 = 2M days. (!)

Need O(N log N) CAD algorithms to sustain Moore’s Law.

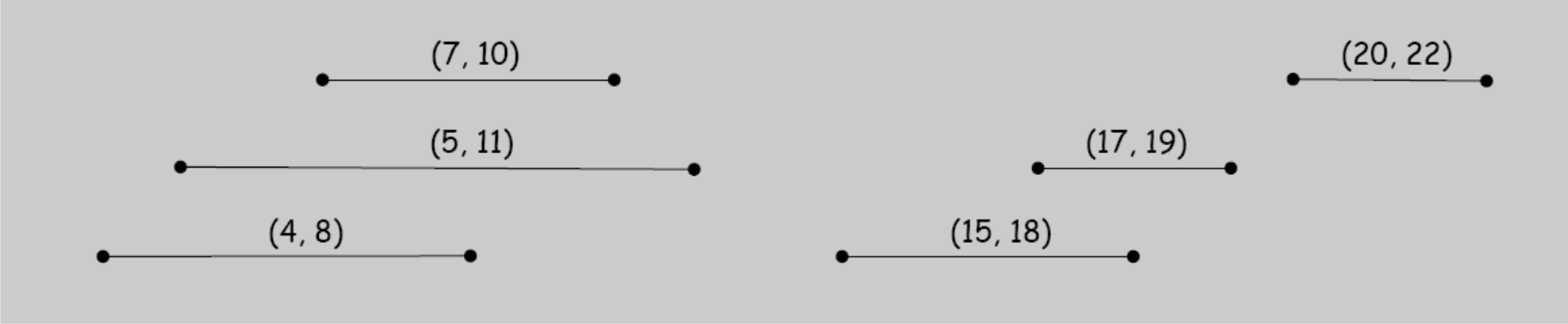
VLSI Database Problem

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



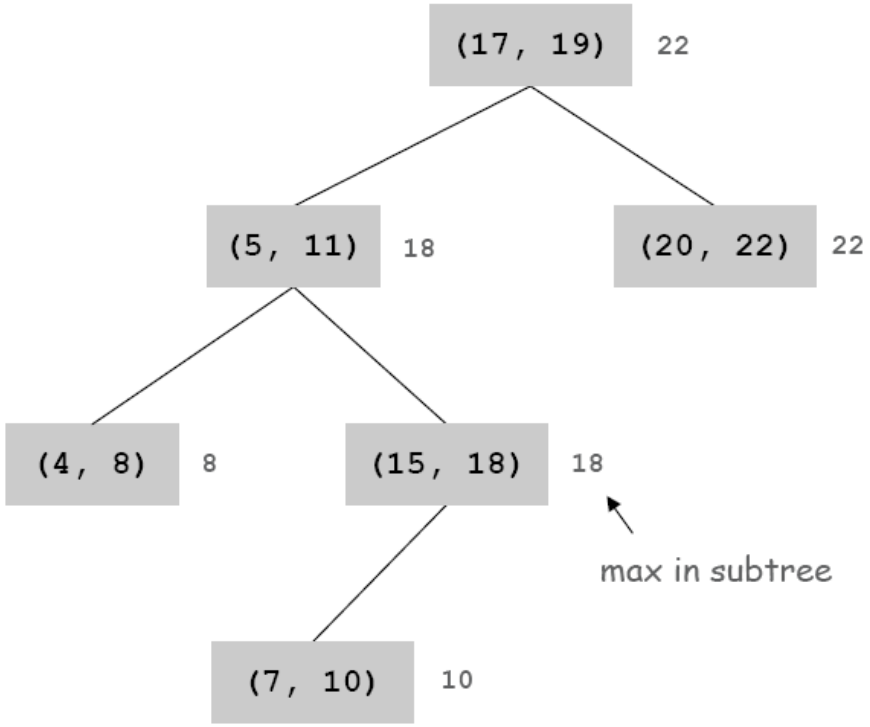
Support following operations.

- 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 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.



VLSI Database Sweep Line Algorithm:
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.

Sweep line: reduces 2D orthogonal rectangle intersection problem to 1D interval searching!