Geometric Algorithms

Convex hull

Geometric primitives

Closest pair of points

Voronoi

Reference: Chapters 24-25, Algorithms in C, 2nd Edition, Robert Sedgewick.

Geometric Algorithms

Applications.

- Data mining.
- VLSI design.
- Computer vision.
- Mathematical models.
- Astronomical simulation.
- Geographic information systems.
- Computer graphics (movies, games, virtual reality).
- Models of physical world (maps, architecture, medical imaging).

Reference: http://www.ics.uci.edu/~eppstein/geom.html

History.

- Ancient mathematical foundations.
- Most geometric algorithms less than 25 years old.

Geometric Primitives

Point: two numbers (x, y).
Line: two numbers a and b [ax + by = 1] ⬅ any line not through origin
Line segment: four numbers (x_1, y_1), (x_2, y_2).
Polygon: sequence of points.

Primitive operations.

- Distance between two points.
- Compare slopes of two lines.
- Given three points p_1, p_2, p_3, is p_1-p_2-p_3 a counterclockwise turn?
- Do two line segments intersect?
- Is a point inside a polygon?

Other geometric shapes.

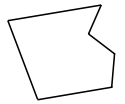
- Triangle, rectangle, circle, sphere, . . .
- 3D and higher dimensions sometimes more complicated.

Warning: Intuition May Mislead

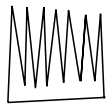
Warning: intuition may be misleading.

- Humans have spatial intuition in 2D and 3D.

Is a given polygon convex?



1	6	5	8	7	2
7	8	6	4	2	1



1	15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	2	18	4	18	4	19	4	19	4	20	3	20	3	20

1	10	3	7	2	8	8	3	4
6	5	15	1	11	3	14	2	16

we think of this

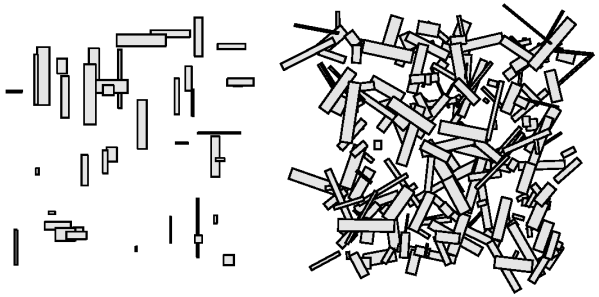
algorithm sees this

Warning: Intuition May Mislead

Warning: intuition may be misleading.

- Humans have spatial intuition in 2D and 3D.

Intersections among set of rectangles.



we think of this

algorithm sees this

Warning: Intuition May Mislead

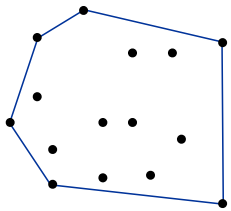
Warning: intuition may be misleading.

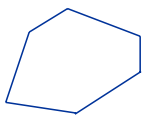
- Humans have spatial intuition in 2D and 3D.
- Computers do not.
- Neither has good intuition in higher dimensions!

Convex Hull

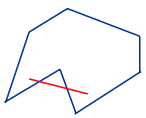
Convex hull.

- Shortest fence surrounding the points.
- Smallest (convex) polygon enclosing the points.
- Intersection of halfspaces defined by point pairs.





convex



not convex

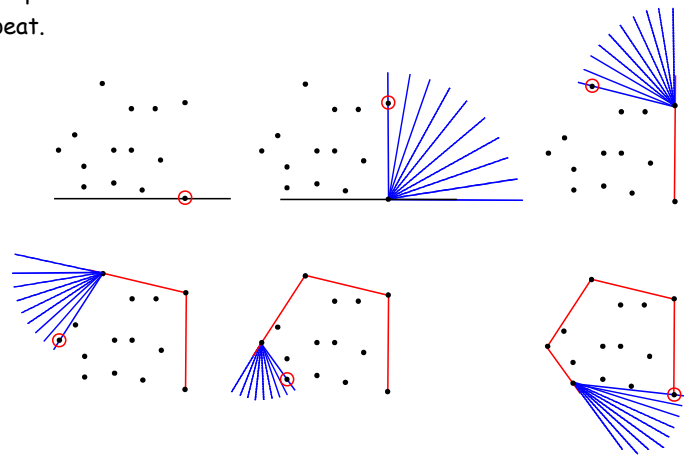
Parameters.

- N = # points.
- M = # points on the hull.

Package Wrap

Package wrap.

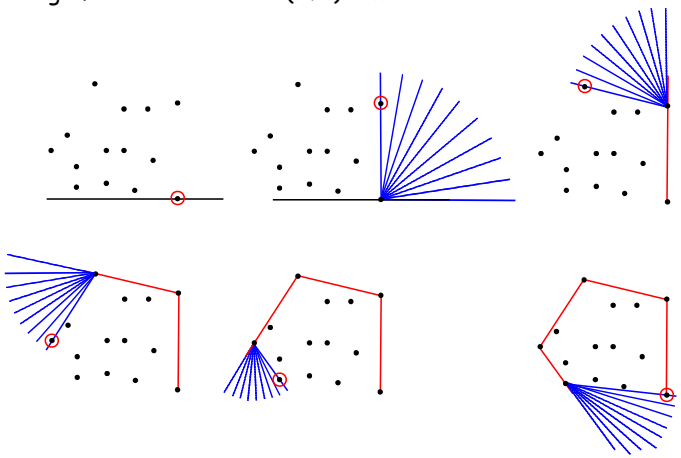
- Start with point with smallest y-coordinate.
- Rotate sweep line around current point in ccw direction.
- First point hit is on the hull.
- Repeat.



Package Wrap

Implementation.

- Compute angle between current point and all remaining points.
- Pick smallest angle larger than current angle.
- 2D analog of selection sort: $\Theta(MN)$ time.



How Many Points on the Hull?

Parameters.

- N = # points.
- M = # points on the hull.

How many points on hull?

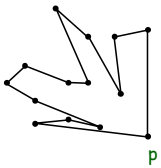
- Worst case: N.
- Average case: difficult problems in stochastic geometry.

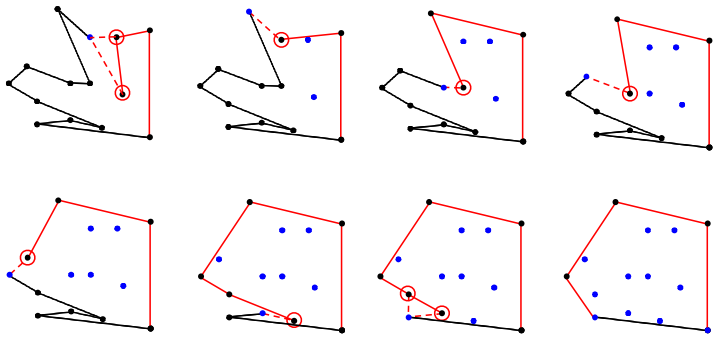
Uniform on a circle: N.
Uniform in a convex polygon with $O(1)$ edges: log N.
Uniform in a disc: $N^{1/3}$.

Graham Scan: Example

Graham scan.

- Choose point p with smallest y-coordinate.
- Sort points by polar angle with p to get simple polygon.
- Consider points in order, and discard those that cause a clockwise turn.





Graham Scan: Example

Implementation.

- Input: `p[1], p[2], . . ., p[N]` are points.
- Output: `M` and rearrangement so that `p[1], ..., p[M]` is convex hull.
- Given three points a,b, and c, is a-b-c a counterclockwise turn?
- Total cost: $O(N \log N)$ for sort and $O(N)$ for rest.

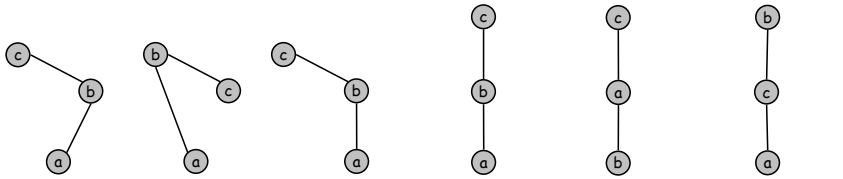
```
// preprocess so that p[1] has smallest y-coordinate
// sort by angle with p[1]

points[0] = points[N];  // sentinel
int M = 3;
for (int i = 4; i <= N; i++) {
   while (Point.ccw(p[M], p[M-1], p[i]) >= 0) {
      M--;              // back up to include i on hull
   }
   M++;
   swap(points, M, i);  // add i to putative hull
}
```

Implementing CCW

CCW: Given three point a, b, and c, is a-b-c a counterclockwise turn?

- Idea: compare slopes.
- Difficulty: degeneracy.



Lesson.

- Geometric primitives are tricky to implement.
- Need to handle all degenerate cases.

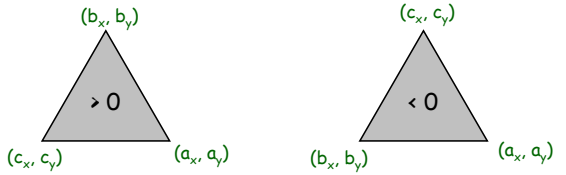
Implementing CCW

CCW: Given three point a, b, and c, is a-b-c a counterclockwise turn?

- Plays same role as comparisons in sorting.
- Determinant gives twice area of triangle.

$$\begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = a_x b_y - a_y b_x + a_y c_x - a_x c_y + b_x c_y - c_x b_y$$

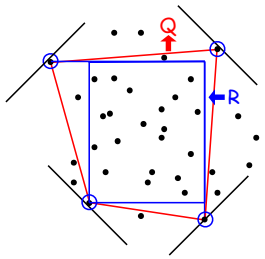
- If area > 0 then a-b-c is counterclockwise.
  If area < 0, then a-b-c is clockwise.
  If area = 0, then a-b-c are collinear.



Quick Elimination

Quick elimination.

- Choose a quadrilateral Q or rectangle R with 4 points as corners.
- If point is inside, can eliminate.
  - 4 CCW tests for quadrilateral
  - 4 comparisons for rectangle

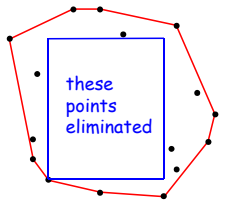


Three-phase algorithm

- Pass through all points to compute R.
- Eliminate points inside R.
- Find convex hull of remaining points.

Impact.

- Almost all points are eliminated if points are random: O(N).
- Improve performance of any convex hull algorithm.



Convex Hull Algorithms Costs Summary

"Guaranteed" asymptotic cost to find M-point hull in N-point set.

Algorithm	Running Time
Package Wrap	N M
Graham Scan	N log N
Quickhull	N log N
Mergehull	N log N
Sweep Line	N log N
Quick Elimination	N *
Best in Theory	N log M

* assumes "reasonable" point distribution

Closest Pair of Points

Given N points in the plane, find a pair that is closest together.

- For concreteness, we assume Euclidean distances.
- Foundation of then-fledgling field of computational geometry.
- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.

Brute force solution.

- Check all pairs of points p and q.
- $\Theta(N^2)$ comparisons.

1-D version (points on a line). O(N log N) easy.

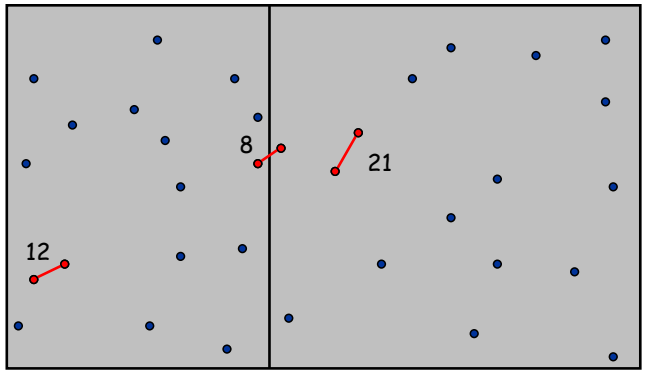
Assumption. No two points have same x coordinate.

⬆ solely to make presentation cleaner

Closest Pair of Points

Algorithm.

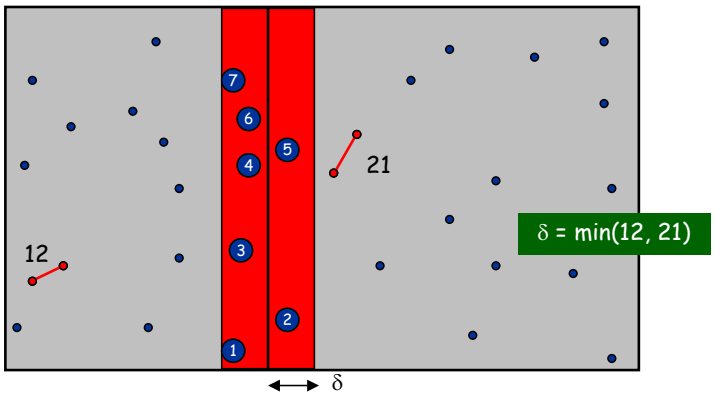
- Divide: draw vertical line so that roughly N / 2 points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return: best of 3 solutions.



Closest Pair of Points

Key step: find closest pair with one point in each side.

- Extra information: closest pair entirely in one side had distance δ.
- Observation: only need to consider points within δ of line.
- Sort points in 2δ-strip by their y coordinate.
- Only check distances of those within 6 positions in sorted list!



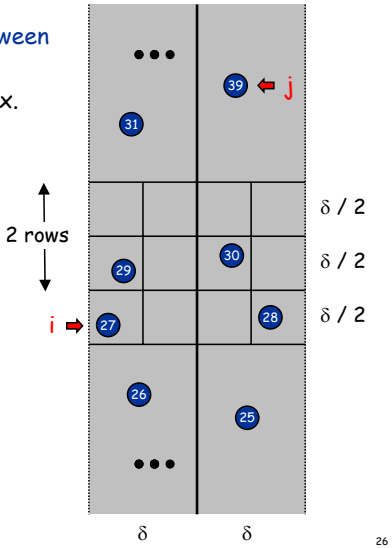
Closest Pair of Points

`s[]` = array of points in the 2δ-strip, sorted by their y-coordinate.

Fact: if $|i - j| \geq 12$, then the distance between `s[i]` and `s[j]` is at least δ.

- No two points lie in same $\delta/2$ by $\delta/2$ 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 of Points

Closest pair algorithm.

Compute separation line x = x_{med} such that half the points have x coordinate less than x_{med}, and half are greater.	O(N log N)
δ_1 = ClosestPair(left half) δ_2 = ClosestPair(right half) δ = min (δ_1 , δ_2)	2T(N / 2)
Delete all points further than δ from separation line.	O(N)
Sort remaining points in strip by y coordinate.	O(N log N)
Scan in y order, and compute distance between each point and next 6 neighbors. If any of these distances is less than δ , update δ .	O(N)
Return δ.	

δ = ClosestPair (p_1, p_2 , . . . , p_N)

Closest Pair of Points

Running time.

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

Can we achieve O(N log N)?

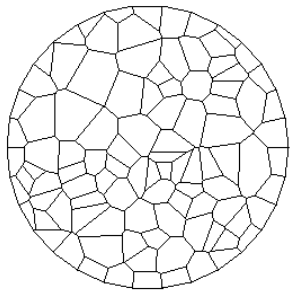
- Yes. Don't sort points in strip from scratch each time.
- Each recursive call should return two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sorting is accomplished by merging two already sorted lists.

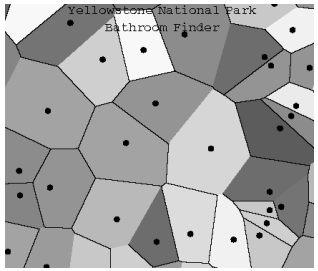
$$T(N) = 2T(N/2) + O(N) \Rightarrow T(N) = O(N \log N)$$

Nearest Neighbor

Input: N Euclidean points.

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

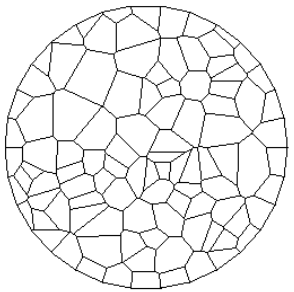


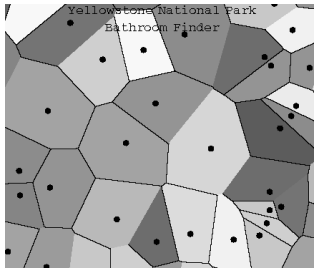


Brute force: O(N) time per query.
Goal: O(N log N) preprocessing, O(log N) per query.

Voronoi Diagram / Dirichlet Tesselation

Input: N Euclidean points.
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.





Quintessential nearest neighbor data structure.

Applications of Voronoi Diagrams

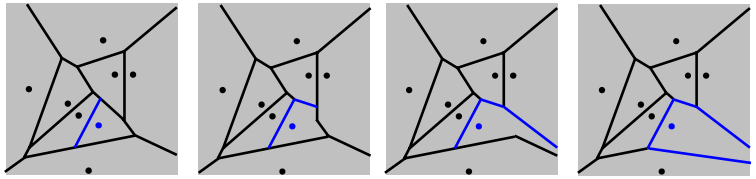
Anthropology. Identify influence of clans and chiefdoms on geographic regions.
Astronomy. Identify clusters of stars and clusters of galaxies.
Biology, Ecology, Forestry. Model and analyze plant competition.
Cartography. Piece together satellite photographs into large "mosaic" maps.
Crystallography. 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 at individual retail store level.
Metallurgy. Modeling "grain growth" in metal films.
Physiology. Analysis of capillary distribution in cross-sections of muscle tissue.
Typography. Character recognition, beveled and carved lettering.
Zoology. Model and analyze the territories of animals.

References: http://voronoi.com, http://www.ics.uci.edu/~eppstein/geom.html

Adding a Point to Voronoi Diagram

Challenge: compute Voronoi.

Basis for incremental algorithms: region containing point gives points to check to compute new Voronoi region boundaries.

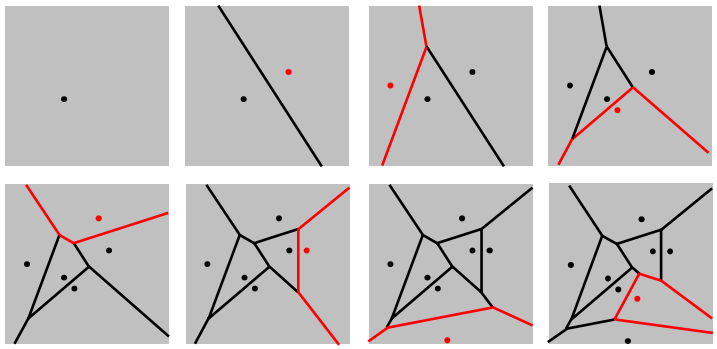


How to represent the Voronoi diagram? Use multilist associating each point with its Voronoi neighbors.

Randomized Incremental Voronoi Algorithm

Add points (in random order).

- Find region containing point. ⬅ use Voronoi itself
- Update neighbor regions, create region for new point.

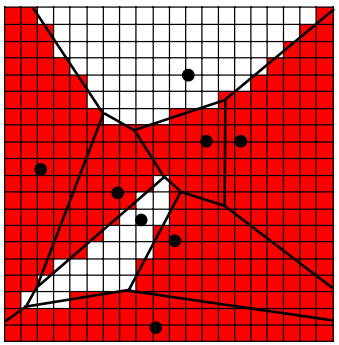


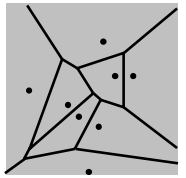
- Running time: O(N log N) on average.

Discretized Voronoi Diagram

Use grid approach to answer near-neighbor queries in constant time.

- Approach 1: provide approximate answer (to within grid size).
- Approach 2: keep list of points to check in grid squares.
- Computation not difficult (move outward from points).

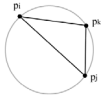




Delaunay Triangulation

Input: N Euclidean points.

Delaunay triangulation: triangulation such that no point is inside *circumcircle* of any other triangle.



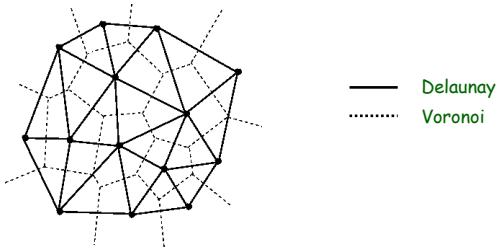
Fact 1: Dual of Voronoi (connect adjacent points in Voronoi diagram).

Fact 2: No edges cross $\Rightarrow$ O(N) edges.

Fact 3: Maximizes the minimum angle for all triangular elements.

Fact 4: Boundary of Delaunay triangulation is convex hull.

Fact 5: Closest pair of of Delaunay graph is closest pair.



Some Geometric Algorithms

Running time to solve a 2D problem with N points.

Problem	Brute Force	Cleverness
convex hull	N^2	N log N
closest pair	N^2	N log N
nearest neighbor	N	log N
polygon triangulation	N^2	N log N
furthest pair	N^2	N log N