Geometric Algorithms

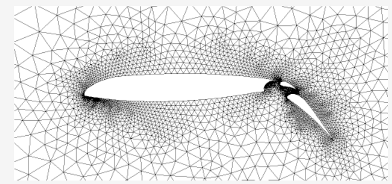
- primitive operations
- convex hull
- closest pair
- voronoi diagram

References:
Algorithms in C (2nd edition), Chapters 24-25
`http://www.cs.princeton.edu/algs4/71primitives`
`http://www.cs.princeton.edu/algs4/72hull`

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).



airflow around an aircraft wing

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

History.

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

- primitive operations
- convex hull
- closest pair
- voronoi diagram

Geometric primitives

Point: two numbers (x, y).

Line: two numbers a and b [ax + by = 1] any line not through origin

Line segment: two points.

Polygon: sequence of points.

Primitive operations.

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

Other geometric shapes.

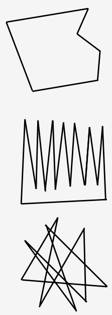
- Triangle, rectangle, circle, sphere, cone, …
- 3D and higher dimensions sometimes more complicated.

Intuition

Warning: intuition may be misleading.

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

Q. Is a given polygon simple? ← no crossings



x	1	6	5	8	7	2
y	7	8	6	4	2	1

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

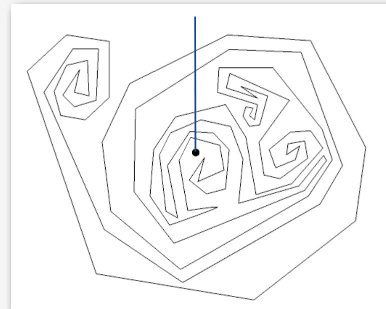
x	1	10	3	7	2	8	8	3	4
y	6	5	15	1	11	3	14	2	16

we think of this — *algorithm sees this*

Polygon inside, outside

Jordan curve theorem. [Veblen 1905] Any continuous simple closed curve cuts the plane in exactly two pieces: the inside and the outside.

Q. Is a point inside a simple polygon?



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

Application. Draw a filled polygon on the screen.

Polygon inside, outside

Jordan curve theorem. [Veblen 1905] Any continuous simple closed curve cuts the plane in exactly two pieces: the inside and the outside.

Q. Is a point inside a simple polygon?

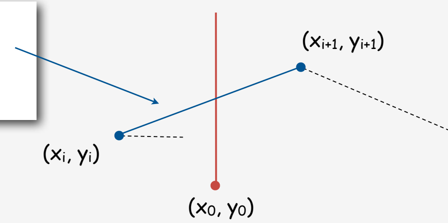
Application. Draw a filled polygon on the screen.

Polygon inside, outside: crossing number

Q. Does line segment intersect ray?

$$y_0 = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x_0 - x_i) + y_i$$

$$x_i \le x_0 \le x_{i+1}$$

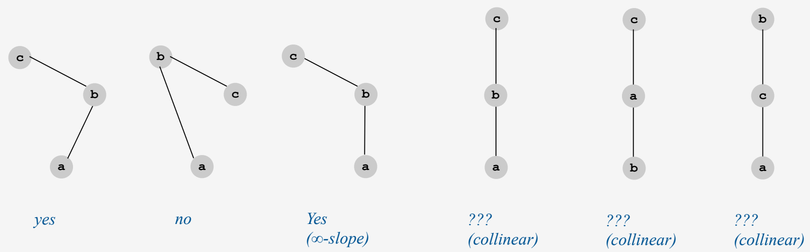



```
public boolean contains(double x0, double y0)
{
    int crossings = 0;
    for (int i = 0; i < N; i++)
    {
       double  slope = (y[i+1] - y[i]) / (x[i+1] - x[i]);
       boolean cond1 = (x[i] <= x0) && (x0 < x[i+1]);
       boolean cond2 = (x[i+1] <= x0) && (x0 < x[i]);
       boolean above = (y0 < slope * (x0 - x[i]) + y[i]);
       if ((cond1 || cond2)  && above ) crossings++;
    }
    return ( crossings % 2 != 0 );
 }
```

Implementing ccw

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

- Analog of comparisons in sorting.
- Idea: compare slopes.



Lesson. Geometric primitives are tricky to implement.

- Dealing with degenerate cases.
- Coping with floating point precision.

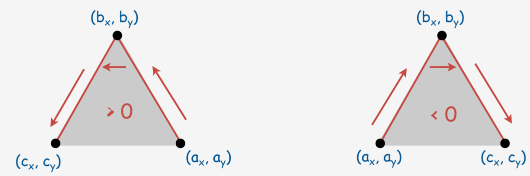
Implementing ccw

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

- Determinant gives twice area of triangle.

$$2 \times Area(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$$

- 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.



Immutable point data type

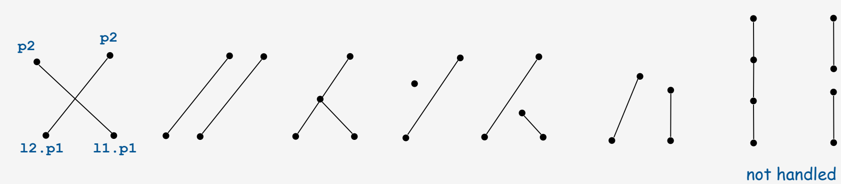
```
public class Point
{
   private final int x;
   private final int y;

   public Point(int x, int y)
   {  this.x = x; this.y = y;  }

   public double distanceTo(Point that)
   {
      double dx = this.x - that.x;
      double dy = this.y - that.y;
      return Math.sqrt(dx*dx + dy*dy);
   }

   public static int ccw(Point a, Point b, Point c)
   {
      double area2 = (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
      if      (area2 < 0) return -1;
      else if (area2 > 0) return +1;
      else                return  0;
   }

   public static boolean collinear(Point a, Point b, Point c)
   {  return ccw(a, b, c) == 0;  }
}
```

cast to `long` to avoid overflowing an `int`

Sample ccw client: Line intersection

Intersect. Given two line segments, do they intersect?

- Idea 1: find intersection point using algebra and check.
- Idea 2: check if the endpoints of one line segment are on different "sides" of the other line segment (4 calls to ccw).



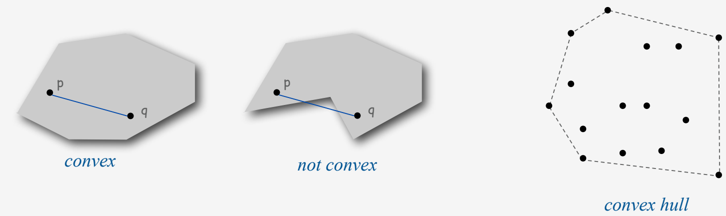

```
public static boolean intersect(Line l1, Line l2)
{
   int test1 = Point.ccw(l1.p1, l1.p2, l2.p1) * Point.ccw(l1.p1, l1.p2, l2.p2);
   int test2 = Point.ccw(l2.p1, l2.p2, l1.p1) * Point.ccw(l2.p1, l2.p2, l1.p2);
   return (test1 <= 0) && (test2 <= 0);
}
```

- primitive operations
- **convex hull**
- closest pair
- voronoi diagram

Convex hull

A set of points is convex if for any two points p and q in the set, the line segment $\overline{pq}$ is completely in the set.

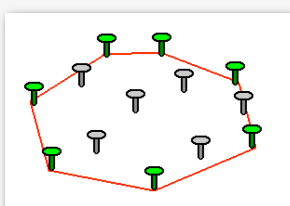
Convex hull. Smallest convex set containing all the points.



Properties.
- "Simplest" shape that approximates set of points.
- Shortest perimeter fence surrounding the points.
- Smallest area convex polygon enclosing the points.

Mechanical solution

Mechanical algorithm. Hammer nails perpendicular to plane; stretch elastic rubber band around points.

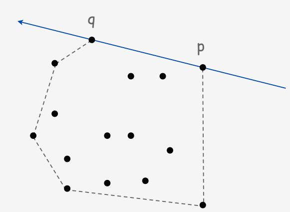


http://www.dfanning.com/math_tips/convexhull_1.gif

Brute-force algorithm

Observation 1.
Edges of convex hull of P connect pairs of points in P.

Observation 2.
p-q is on convex hull if all other points are counterclockwise of $\overrightarrow{pq}$.

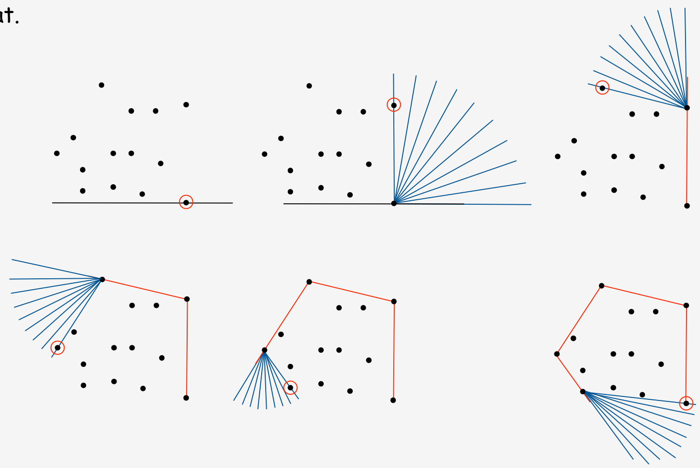


O(N^3) algorithm. For all pairs of points p and q in P:
- Compute `ccw(p, q, x)` for all other `x` in P.
- p-q is on hull if all values are positive.

Package wrap (Jarvis march)

Package wrap.

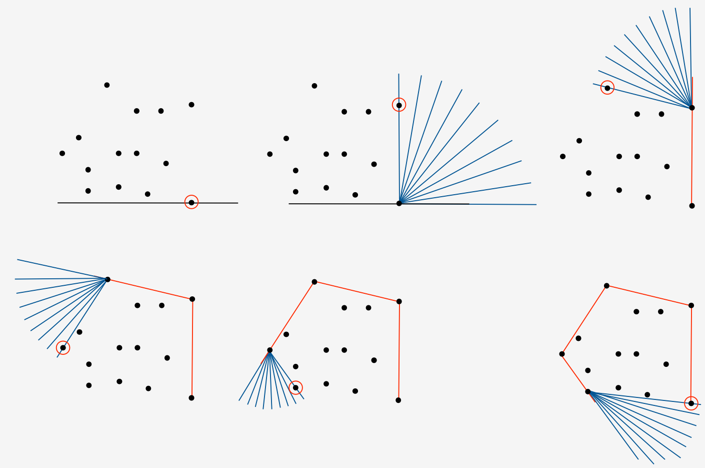
- Start with point with smallest y-coordinate.
- Rotate sweep line around current point in ccw direction.
- First point hit is on the hull.
- Repeat.



Package wrap (Jarvis march)

Implementation.

- Compute angle between current point and all remaining points.
- Pick smallest angle larger than current angle.
- $\Theta(N)$ per iteration.



How many points on the hull?

Parameters.

- N = number of points.
- h = number of points on the hull.

Package wrap running time. $\Theta(N\,h)$.

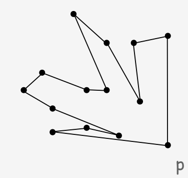
How many points on hull?

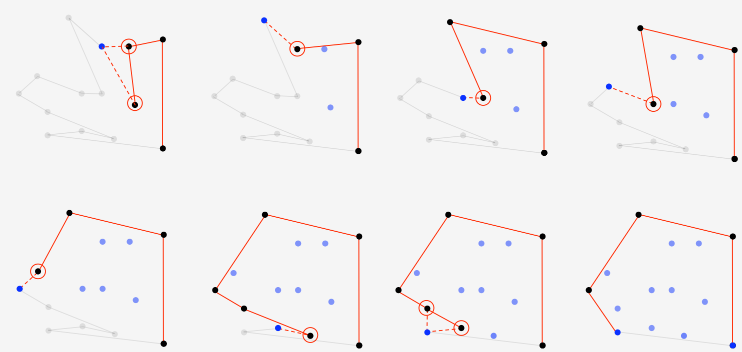
- Worst case: h = N.
- Average case: difficult problems in stochastic geometry.
  - in a disc: $h = N^{1/3}$
  - in a convex polygon with O(1) edges: h = log N

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 would create a clockwise turn.





Graham scan: implementation

Implementation.

- Input: `p[1], p[2], …, p[N]` are points.
- Output: `M` and rearrangement so that `p[1], p[2], …, p[M]` is convex hull.

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

points[0] = points[N];   // sentinel
int M = 2;
for (int i = 3; i <= N; i++)
{
   while (Point.ccw(p[M-1], p[M], p[i]) <= 0) M--;
   M++;
   swap(points, M, i);
}
```

discard points that would create clockwise turn

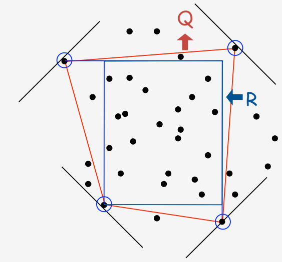
add i to putative hull

Running time. O(N log N) for sort and $\underbrace{\text{O(N) for rest}}_{\text{why?}}$.

Quick elimination

Quick elimination.

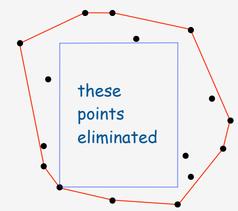
- Choose a quadrilateral Q or rectangle R with 4 points as corners.
- Any point inside cannot be on hull.
  - 4 ccw tests for quadrilateral
  - 4 compares for rectangle



Three-phase algorithm.

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

In practice: eliminates almost all points in linear time.



Convex hull algorithms costs summary

Asymptotic cost to find h-point hull in N-point set.

algorithm	running time
package wrap	N h
Graham scan	N log N
quickhull	N log N
mergehull	N log N
sweep line	N log N
quick elimination	N †
marriage-before-conquest	N log h

output sensitive

† assumes "reasonable" point distribution

Convex hull: lower bound

Models of computation.

- Compare-based: compare coordinates.
  (impossible to compute convex hull in this model of computation)

```
(a.x < b.x) || ((a.x == b.x) && (a.y < b.y)))
```

- Quadratic decision tree model: compute any quadratic function of the coordinates and compare against 0.

```
(a.x*b.y - a.y*b.x + a.y*c.x - a.x*c.y + b.x*c.y - c.x*b.y) < 0
```

higher degree polynomial tests don't help either [Ben-Or, 1983]

Proposition. [Andy Yao, 1981] In quadratic decision tree model, any convex hull algorithm requires Ω(N log N) ops.

even if hull points are not required to be output in counterclockwise order

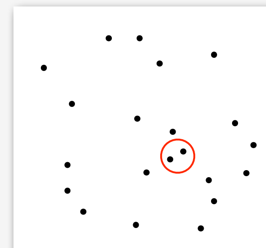
primitive operations
convex hull
closest pair
voronoi diagram

Closest pair problem

Input. N points in the plane.
Output. Pair of points with smallest Euclidean distance between them.

Fundamental geometric primitive.
- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired fast algorithms for these problems



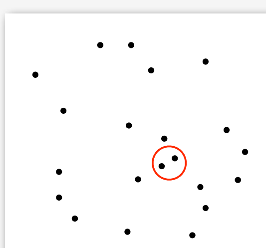
Closest pair problem

Input. N points in the plane
Output. Pair of points with smallest Euclidean distance between them.

Brute force. Check all pairs with N^2 distance calculations.

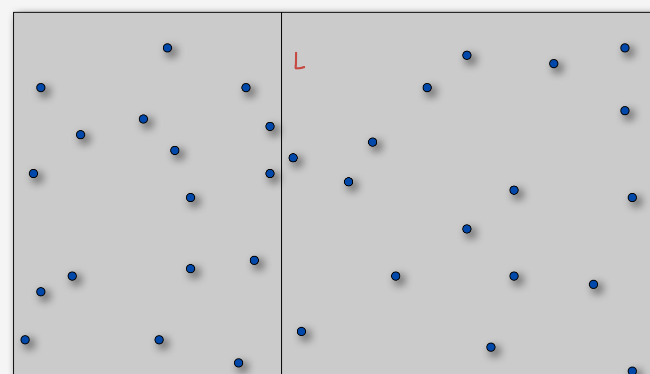
1-D version. Easy N log N algorithm if points are on a line.

Degeneracies complicate solutions.
[assumption for lecture: no two points have same x-coordinate]



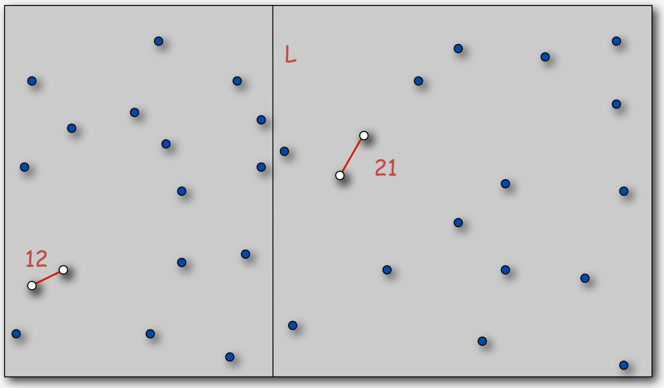
Divide-and-conquer algorithm

- Divide: draw vertical line L so that roughly $\frac{1}{2}N$ points on each side.



Divide-and-conquer algorithm

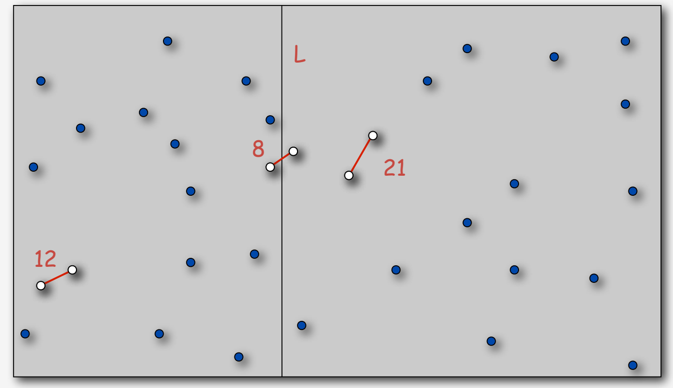
- Divide: draw vertical line L so that roughly $\frac{1}{2}N$ points on each side.
- Conquer: find closest pair in each side recursively.



Divide-and-conquer algorithm

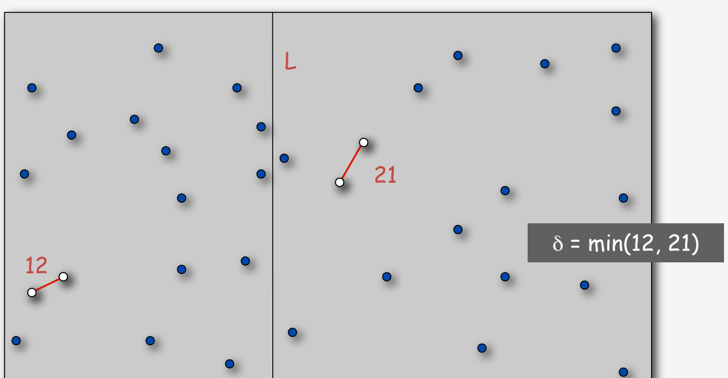
- Divide: draw vertical line L so that roughly $\frac{1}{2}N$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.

seems like $\Theta(N^2)$



How to find closest pair with one point in each side?

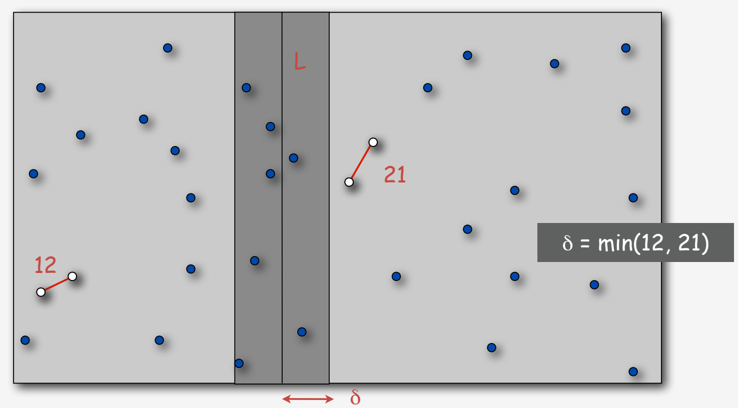
Find closest pair with one point in each side, assuming that distance < δ.



How to find closest pair with one point in each side?

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

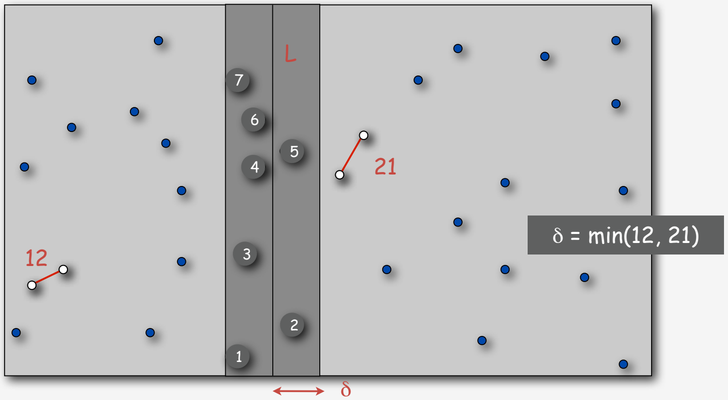
- Observation: only need to consider points within δ of line L.



How to find closest pair with one point in each side?

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

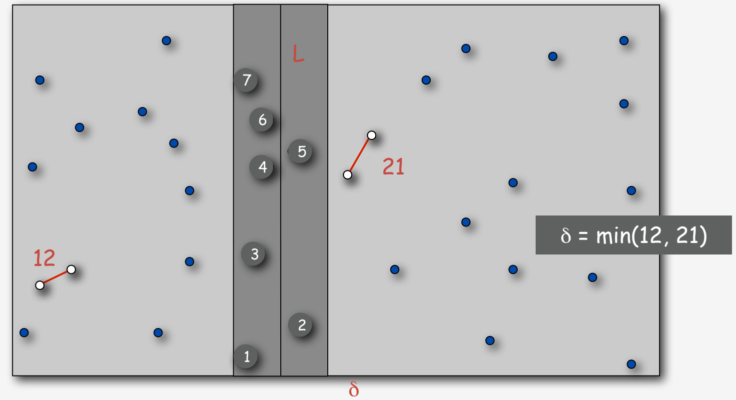
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.



How to find closest pair with one point in each side?

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 sorted list!



How to find closest pair with one point in each side?

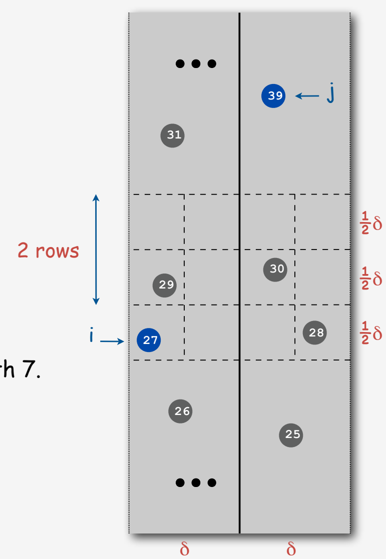
Def. Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate.

Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ.

Pf.

- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. ▪

Fact. Claim remains true if we replace 12 with 7.



Divide-and-conquer algorithm

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

   δ₁ = Closest-Pair(left half)
   δ₂ = Closest-Pair(right half)                                 2T(N / 2)
   δ  = min(δ₁, δ₂)

   Delete all points further than δ from separation line L      O(N)

   Sort remaining points by y-coordinate.                        O(N log N)

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

   return δ.
}
```

Divide-and-conquer algorithm: analysis

Running time recurrence. $T(N) \le 2T(N/2) + O(N \log N)$.

Solution. $T(N) = O(N (\log N)^2)$.

Remark. Can be improved to $O(N \log N)$.

avoid sorting by y-coordinate from scratch

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

$(x_1 - x_2)^2 + (y_1 - y_2)^2$

Summary

Ingenious algorithms enable solution of large instances for numerous fundamental geometric problems.

problem	brute	clever
convex hull	N^2	$N \log N$
closest pair	N^2	$N \log N$
Voronoi	?	$N \log N$
Delauney triangulation	N^4	$N \log N$
Euclidean MST	N^2	$N \log N$

asymptotic time to solve a 2D problem with N points

Note. 3D and higher dimensions test limits of our ingenuity.