CS 381 – FALL 2019

Week 4.1, Monday, Sept 9

Homework 2 Due: September 16th, 2019 @ 11:59PM on Gradescope

5.5 Integer Multiplication

Slides: Kevin Wayne

Motivation: Complex Multiplication

Complex multiplication. $(a + bi)(c + di) = x + yi$.

Grade-school. $x = ac - bd$, $y = bc + ad$.

4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

Our Prices Are Fantastic!
Multiplication: \$100 (reals only $\mathbb{R}$)
Addition: \$1 (reals only $\mathbb{R}$)

\$402 for Grade-School Approach: 4 multiplications, 2 additions

Complex Multiplication

Complex multiplication. $(a + bi)(c + di) = x + yi.$

Grade-school. $x = ac - bd,\ \ y = bc + ad.$

4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

Yes. [Gauss] $x = \underset{1}{ac} - \underset{2}{bd},\ \ y = \underset{3}{(a + b)(c + d)} - \underset{1}{ac} - \underset{2}{bd}.$

(y= ac + ad + bc + bd - $ac - bd = bc + ad$)

3 multiplications, 5 additions ($305)

Remark. Improvement if no hardware multiply.

Clicker Question

$$\boldsymbol{x = 2^{n/2} \cdot x_1 + x_0}$$
$$\boldsymbol{y = 2^{n/2} \cdot y_1 + y_0}$$

Suppose we have computed $x_0 y_0$ and $x_1 y_1$ how can we compute $\boldsymbol{x_0 y_1} + \boldsymbol{x_1 y_0}$ with only one additional multiplication (and O(1) addition/subtraction operations)?

A. Impossible! Two multiplications are necessary

B. $\boldsymbol{x_0 y_1} + \boldsymbol{x_1 y_0} = (x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1$

C. $\boldsymbol{x_0 y_1} + \boldsymbol{x_1 y_0} = (x_0 + y_1)(y_0 + x_1) - x_0 y_0 - x_1 y_1$

D. $\boldsymbol{x_0 y_1} + \boldsymbol{x_1 y_0} = (x_0 + y_0)(y_1 + x_1) - x_1 y_0 - x_0 y_1$

Clicker Question

$$\boldsymbol{x = 2^{n/2} \cdot x_1 + x_0}$$
$$\boldsymbol{y = 2^{n/2} \cdot y_1 + y_0}$$

Suppose we have computed $x_0 y_0$ and $x_1 y_1$ how can we compute $\boldsymbol{x_0 y_1 + x_1 y_0}$ with only one additional multiplication (and O(1) addition/subtraction operations)?

A. Impossible! Two multiplications are necessary

B. $\boldsymbol{x_0 y_1 + x_1 y_0} = (x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1$

C. $\boldsymbol{x_0 y_1 + x_1 y_0} = (x_0 + y_1)(y_0 + x_1) - x_0 y_0 - x_1 y_1$

D. $\boldsymbol{x_0 y_1 + x_1 y_0} = (x_0 + y_0)(y_1 + x_1) - x_1 y_0 - x_0 y_1$

Integer Addition

Addition. Given two n-bit integers x and y, compute $x + y$.
Grade-school. $\Theta(n)$ bit operations.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Remark. Grade-school addition algorithm is optimal.

Integer Multiplication

Multiplication. Given two n-bit integers x and y, compute $x \times y$.
Grade-school. $\Theta(n^2)$ bit operations.

```
                1 1 0 1 0 1 0 1
              × 0 1 1 1 1 1 0 1
              -----------------
                1 1 0 1 0 1 0 1
              0 0 0 0 0 0 0 0 0
            1 1 0 1 0 1 0 1 0
          1 1 0 1 0 1 0 1 0
        1 1 0 1 0 1 0 1 0
      1 1 0 1 0 1 0 1 0
    1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
-------------------------------
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1
```

Q. Is grade-school multiplication algorithm optimal?

Divide-and-Conquer Multiplication: Warmup

To multiply two n-bit integers x and y:

- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right)$$
$$= 2^n \cdot \underset{1}{x_1 y_1} + 2^{\frac{n}{2}} \cdot (\underset{2}{x_0 y_1} + \underset{3}{x_1 y_0}) + \underset{4}{x_0 y_0}$$

Ex. $x = \underbrace{1000}_{x_1}\underbrace{1101}_{x_0}$ $\quad y = \underbrace{1110}_{y_1}\underbrace{0001}_{y_0}$

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

Divide-and-Conquer Multiplication: Warmup

To multiply two n-bit integers x and y:

- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$

$$y = 2^{n/2} \cdot y_1 + y_0$$

$$xy = (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0)$$

$$= 2^n \cdot \underbrace{x_1 y_1}_{1} + 2^{\frac{n}{2}} \cdot (\underbrace{x_0 y_1}_{2} + \underbrace{x_1 y_0}_{3}) + \underbrace{x_0 y_0}_{4}$$

Bit Shifts: O(n) cheap

Ex. $x = \underbrace{1000}_{x_1}\underbrace{1101}_{x_0}$ $y = \underbrace{1110}_{y_1}\underbrace{0001}_{y_0}$

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

Master's Theorem: a = 4, b=2, c=1 $\left(\frac{a}{b^c}\right) > 1$, $O\left(n^{\log_b a}\right) = O(n^2)$

Karatsuba Multiplication

To multiply two n-bit integers x and y:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$

$$\begin{aligned} xy &= 2^n \cdot x_1y_1 + 2^{\frac{n}{2}} \cdot (x_0y_1 + x_1y_0) + x_0y_0 \\ &= 2^n \cdot \underset{1}{x_1y_1} + 2^{\frac{n}{2}} \cdot \Big(\underset{2}{(x_0 + x_1)(y_0 + y_1)} - \underset{3}{x_0y_0} - \underset{1}{x_1y_1}\Big) + \underset{3}{x_0y_0} \end{aligned}$$

$$T(n) = \underbrace{3T\left(\frac{n}{2}\right)}_{\text{Recursive calls (1), (2) and (3)}} + \underbrace{O(n)}_{\text{Add, Shift, Subtract}}$$

Clicker Question: Karatsuba Multiplication

The running time of Karatsuba is:

A. $\Theta(n \log n)$

B. $\Theta\left(n^{\log_3 2}\right)$

C. $\Theta(n^2)$

D. $\Theta\left(n^{\log_2 3}\right)$

E. $\Theta(n!)$

$$T(n) = \underbrace{3T\left(\frac{n}{2}\right)}_{\text{Recursive calls (1), (2) and (3)}} + \underbrace{O(n)}_{\text{Add, Shift, Subtract}}$$

The running time of Karatsuba is:

A. $\Theta(n \log n)$

B. $\Theta\left(n^{\log_3 2}\right)$

C. $\Theta(n^2)$

D. $\boldsymbol{\Theta\left(n^{\log_2 3}\right)}$

E. $\Theta(n!)$

$$T(n) = \underbrace{3T\left(\frac{n}{2}\right)}_{\substack{\text{Recursive calls} \\ \text{(1), (2) and (3)}}} + \underbrace{O(n)}_{\substack{\text{Add, Shift,} \\ \text{Subtract}}}$$

Karatsuba Multiplication

To multiply two n-bit integers x and y:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$

$$xy = 2^n \cdot x_1 y_1 + 2^{\frac{n}{2}} \cdot (x_0 y_1 + x_1 y_0) + x_0 y_0$$
$$= 2^n \cdot \underbrace{x_1 y_1}_{1} + 2^{\frac{n}{2}} \cdot \big(\underbrace{(x_0 + x_1)(y_0 + y_1)}_{2} - \underbrace{x_0 y_0}_{3} - \underbrace{x_1 y_1}_{1}\big) + \underbrace{x_0 y_0}_{3}$$

Theorem. [Karatsuba-Ofman 1962] Can multiply two n-bit integers in $O(n^{1.585})$ bit operations.

$$T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract,shift}} \Rightarrow T(n)$$

Master's Theorem: a = 3, b=2, c=1 $\left(\frac{a}{b^c}\right) > 1 \Rightarrow T(n) \in O\left(n^{\log_b a}\right)$

$[\log_2 3 < 1.585]$

Toom-3 Generalization

Split into 3 parts

$$a = 2^{2n/3} \cdot a_2 + 2^{\frac{n}{3}} \cdot a_1 + a_0$$
$$b = 2^{2n/3} \cdot b_2 + 2^{\frac{n}{3}} \cdot b_1 + b_0$$

Requires: 5 multiplications of n/3 bit numbers and O(1) additions, shifts

$$T(n) = 5 \cdot T\left(\frac{n}{3}\right) + O(n) \Rightarrow T(n) \in O\left(n^{\log_3 5}\right)$$

$$1.465 \approx \log_3 5 < \log_2 3 \approx 1.585$$

Toom-Cook Generalization (split into k parts): merge w/ (2k-1) mults

$$a = 2^{\frac{n(k-1)}{k}} \cdot a_{k-1} + \cdots + 2^{\frac{n}{k}} \cdot a_1 + a_0$$

$$b = 2^{\frac{n(k-1)}{k}} \cdot a_k + \cdots + 2^{\frac{n}{k}} \cdot a_1 + a_0$$

$$T_k(n) = (2k-1) \cdot T_k\left(\frac{n}{k}\right) + O(n) \Rightarrow T_k(n) \in O\left(n^{\log_k(2k-1)}\right)$$

$\forall \varepsilon > 0 \exists k$ s.t $T_k(n) \in O(n^{1+\varepsilon})$

$$\lim_{k\to\infty}(\log_k(2k-1)) = 1$$

Toom-3 Generalization

Split into 3 parts

$$a = 2^{2n/3} \cdot a_2 + 2^{\frac{n}{3}} \cdot a_1 + a_0$$
$$b = 2^{2n/3} \cdot b_2 + 2^{\frac{n}{3}} \cdot b_1 + b_0$$

Requires: 5 multiplications of n/3 bit numbers and O(1) additions, shifts

$$T(n) = 5 \cdot T\left(\frac{n}{3}\right) + O(n) \Rightarrow T(n) \in O\left(n^{\log_3 5}\right)$$

≈ 1.465

Schönhage-Strassen algorithm

$$T(n) \in O(n \ \log n \ \log\log n)$$

Only used for really big numbers: $a > 2^{2^{15}}$

State of the Art: $O(n \ \log n \ g(n))$ for increasing small $g(n) \ll \log\log n$

5.4 Closest Pair of Points

Closest Pair of Points

Closest pair. Given n points in the plane, find a pair 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

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

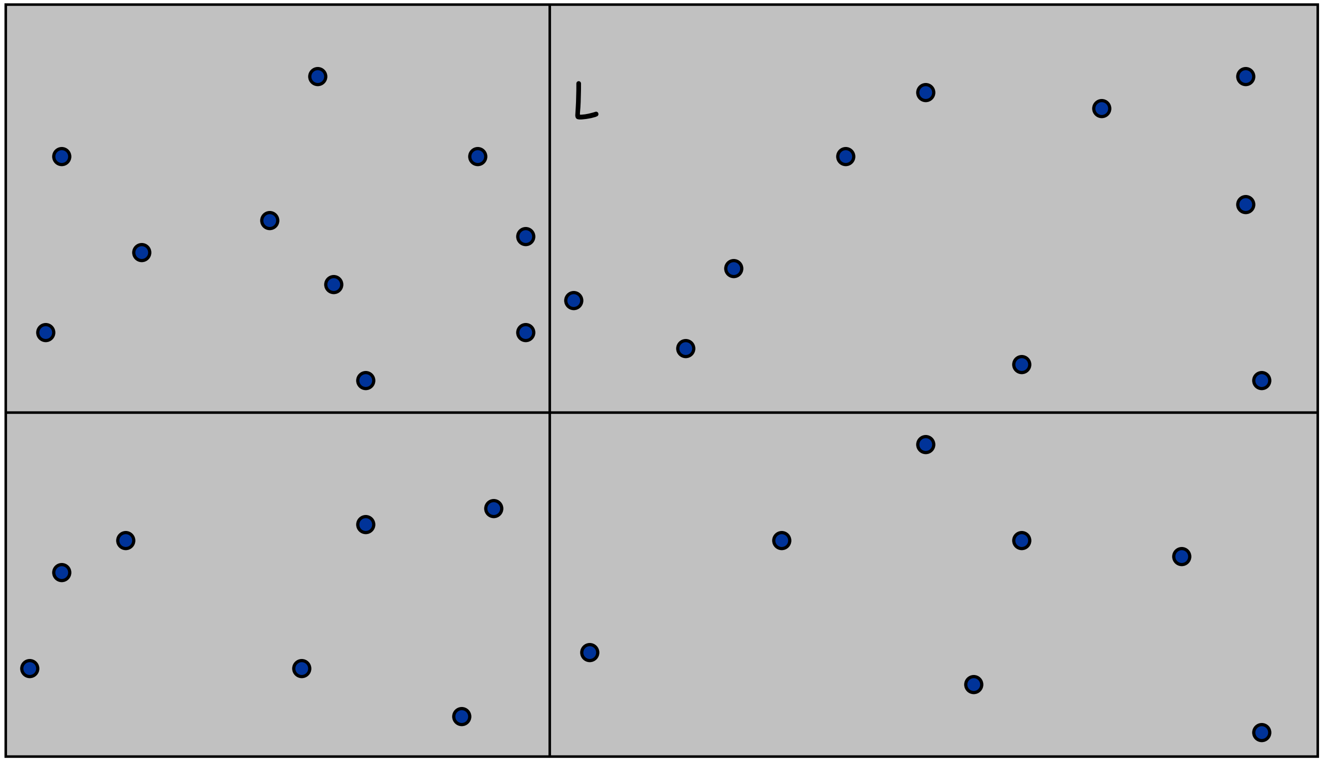
1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner

Closest Pair of Points: First Attempt

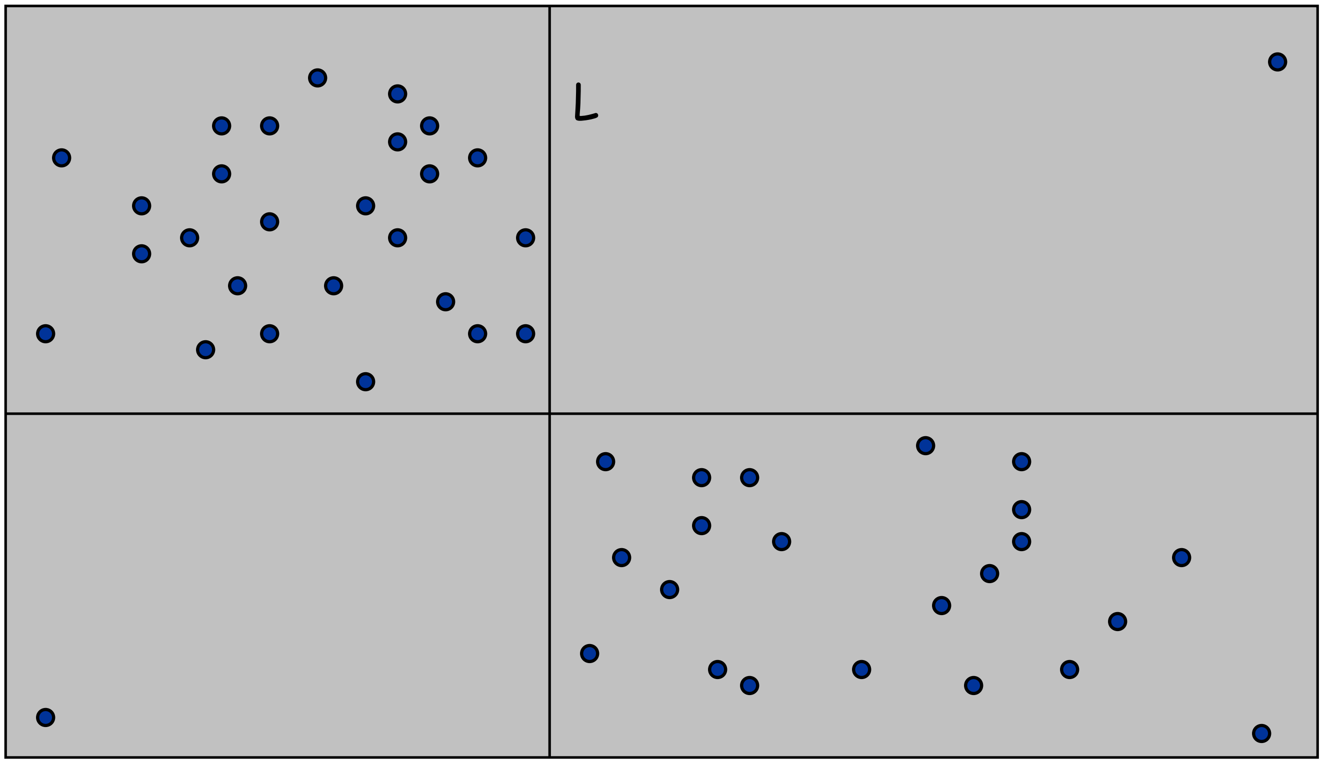
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

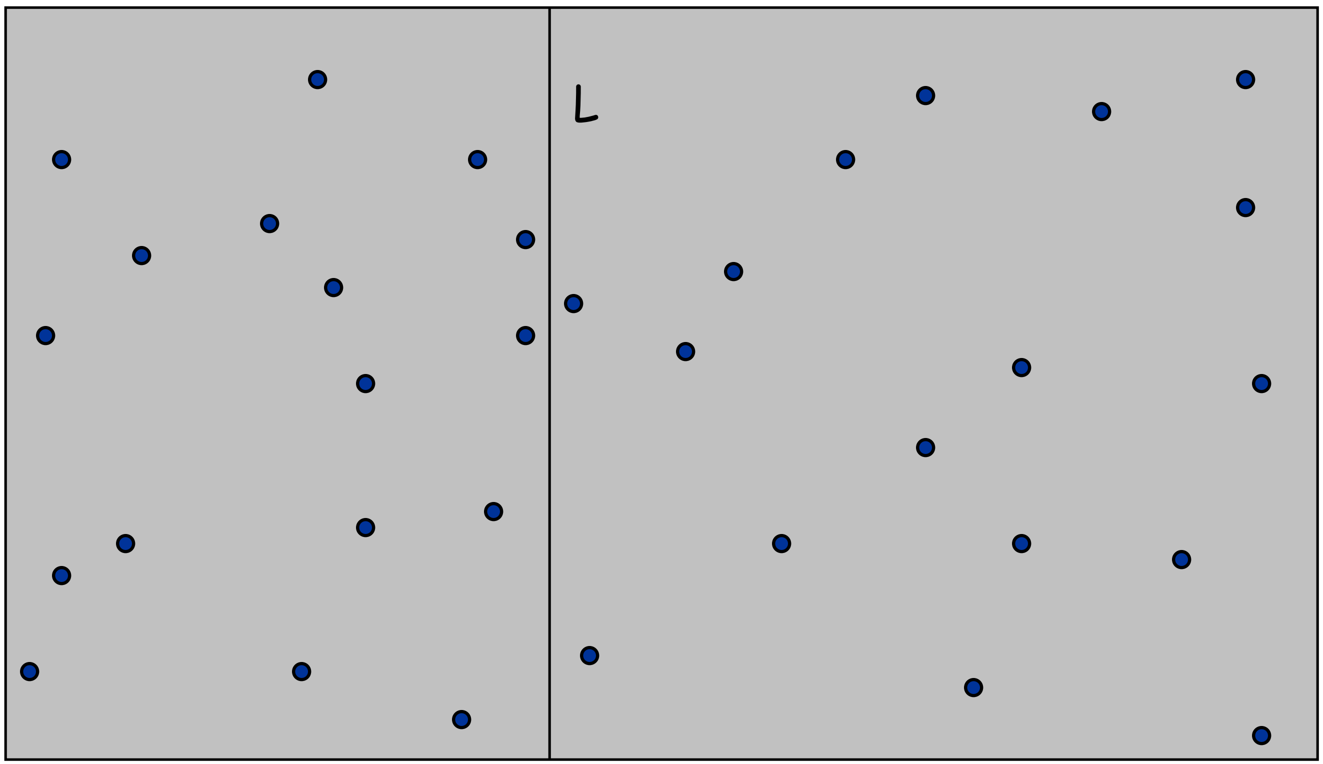
Obstacle. Impossible to ensure n/4 points in each piece.



Closest Pair of Points

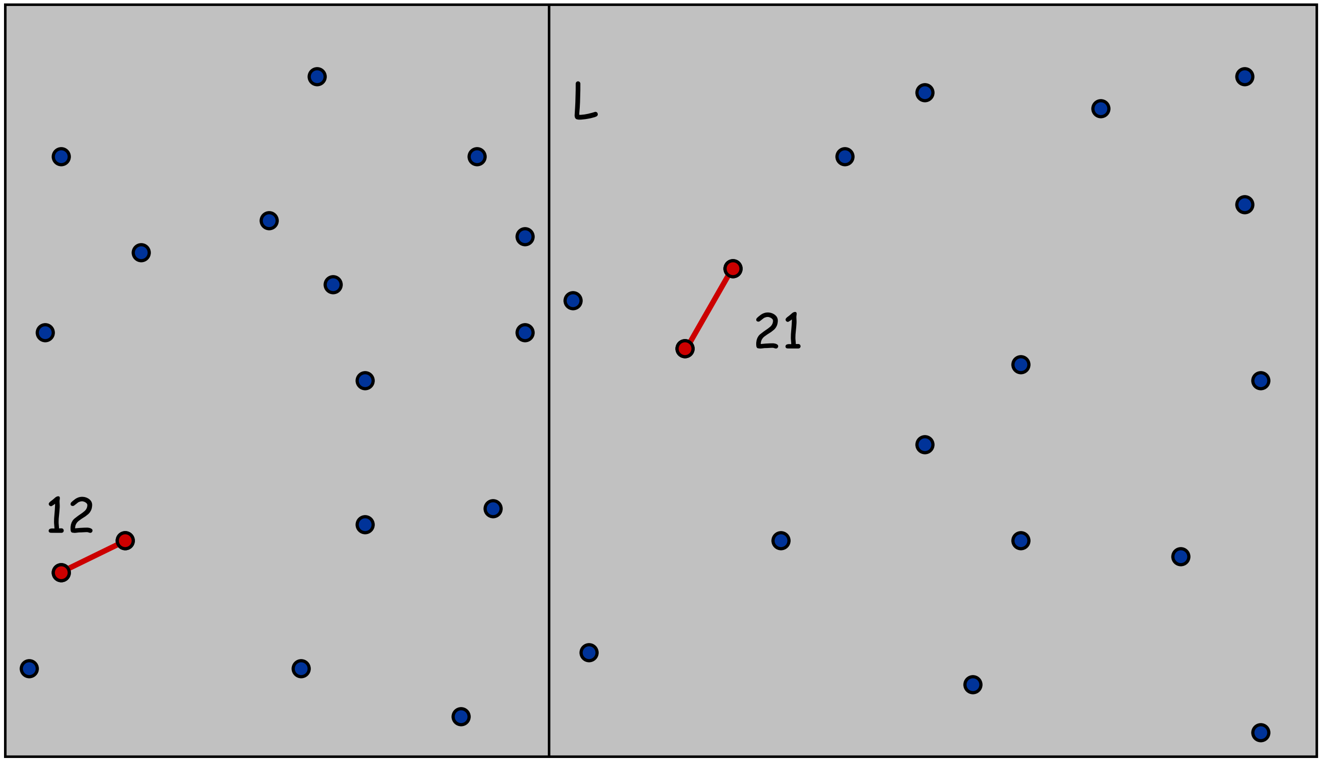
Algorithm.

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



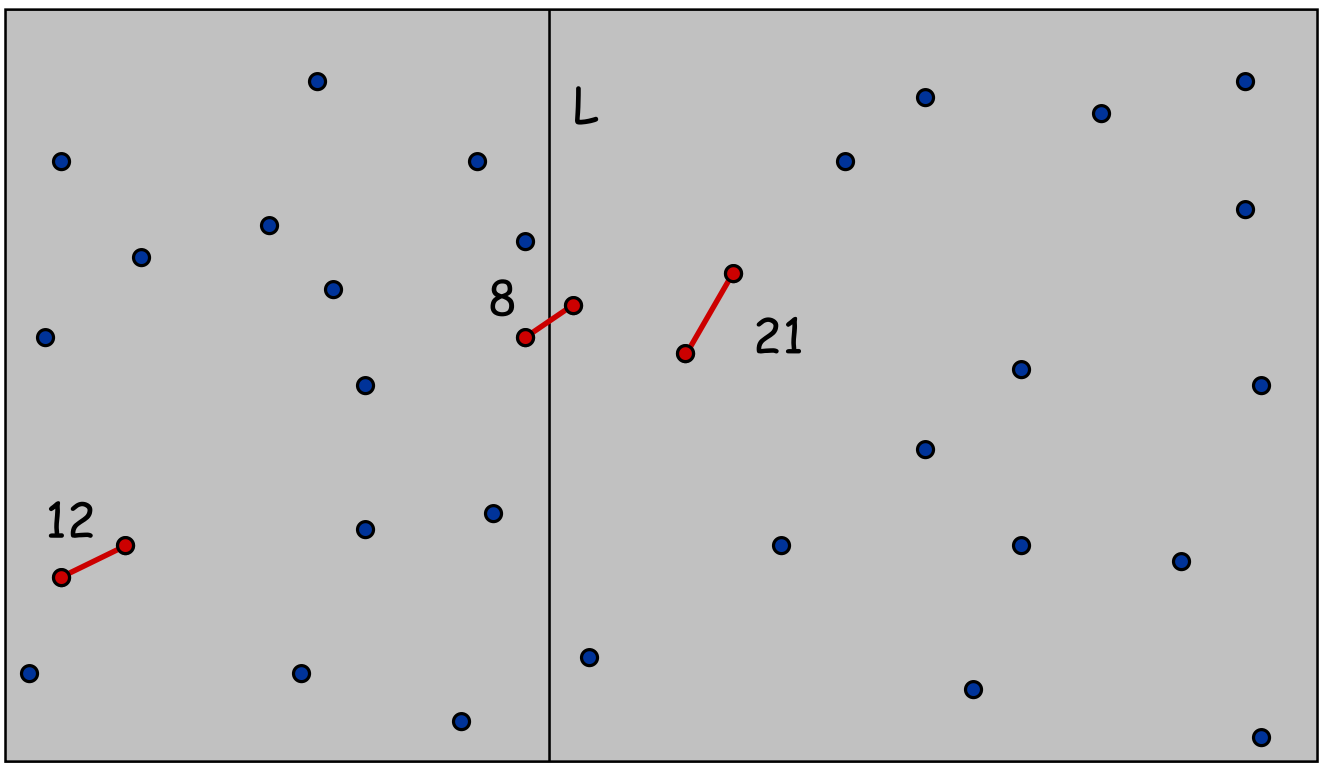
Algorithm.

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



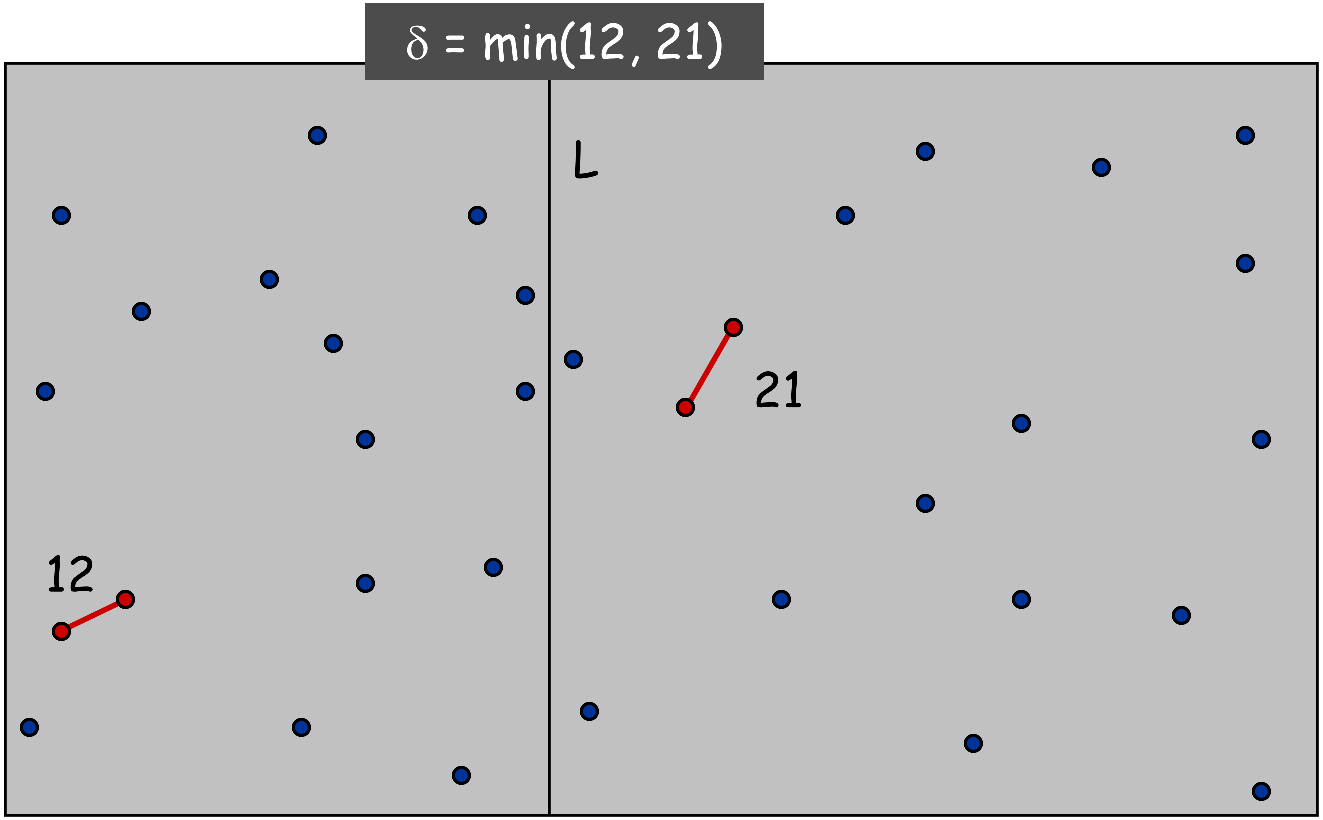
Closest Pair of Points

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. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.



Closest Pair of Points

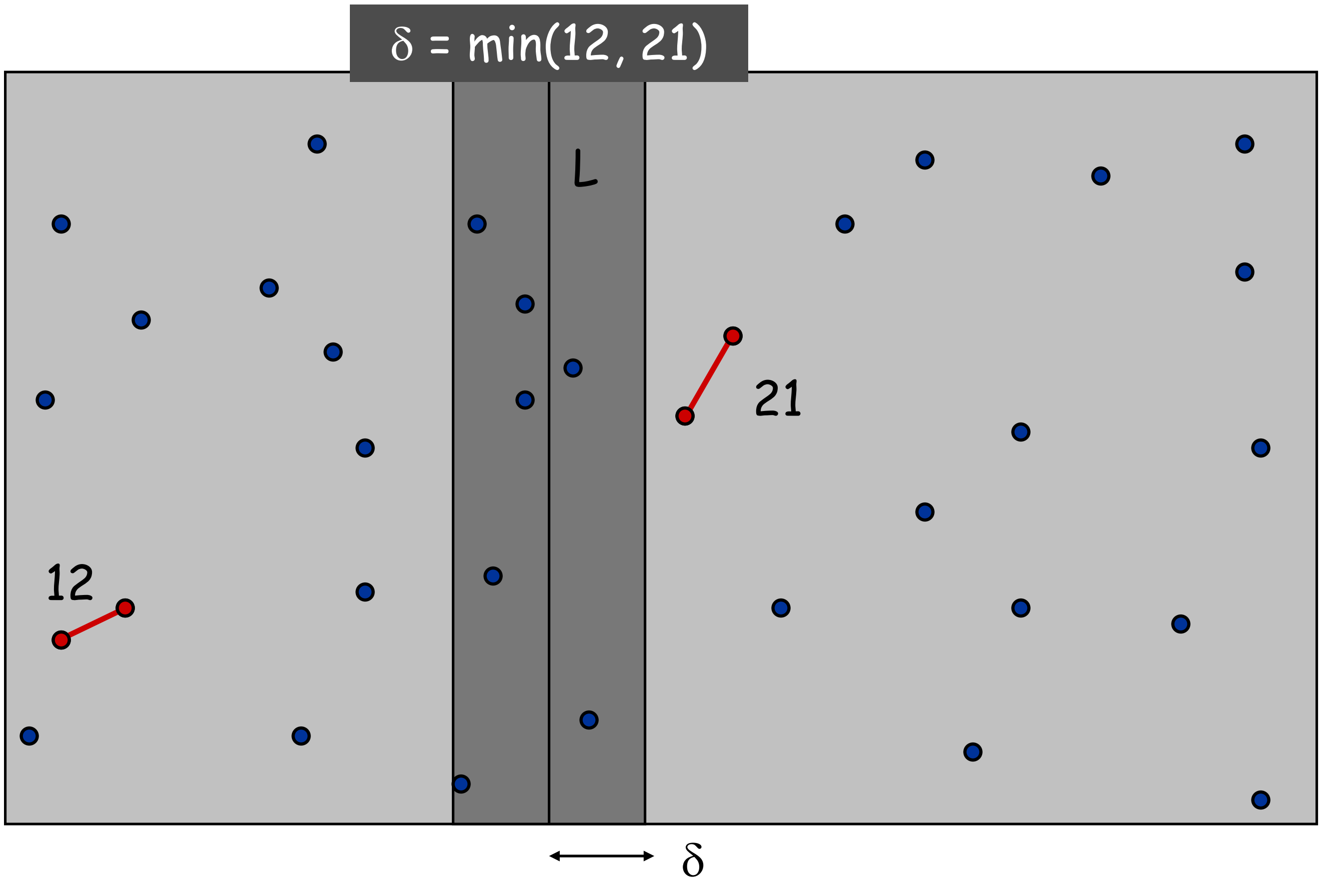
Find closest pair with one point in each side, assuming that distance < δ.



Closest Pair of Points

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

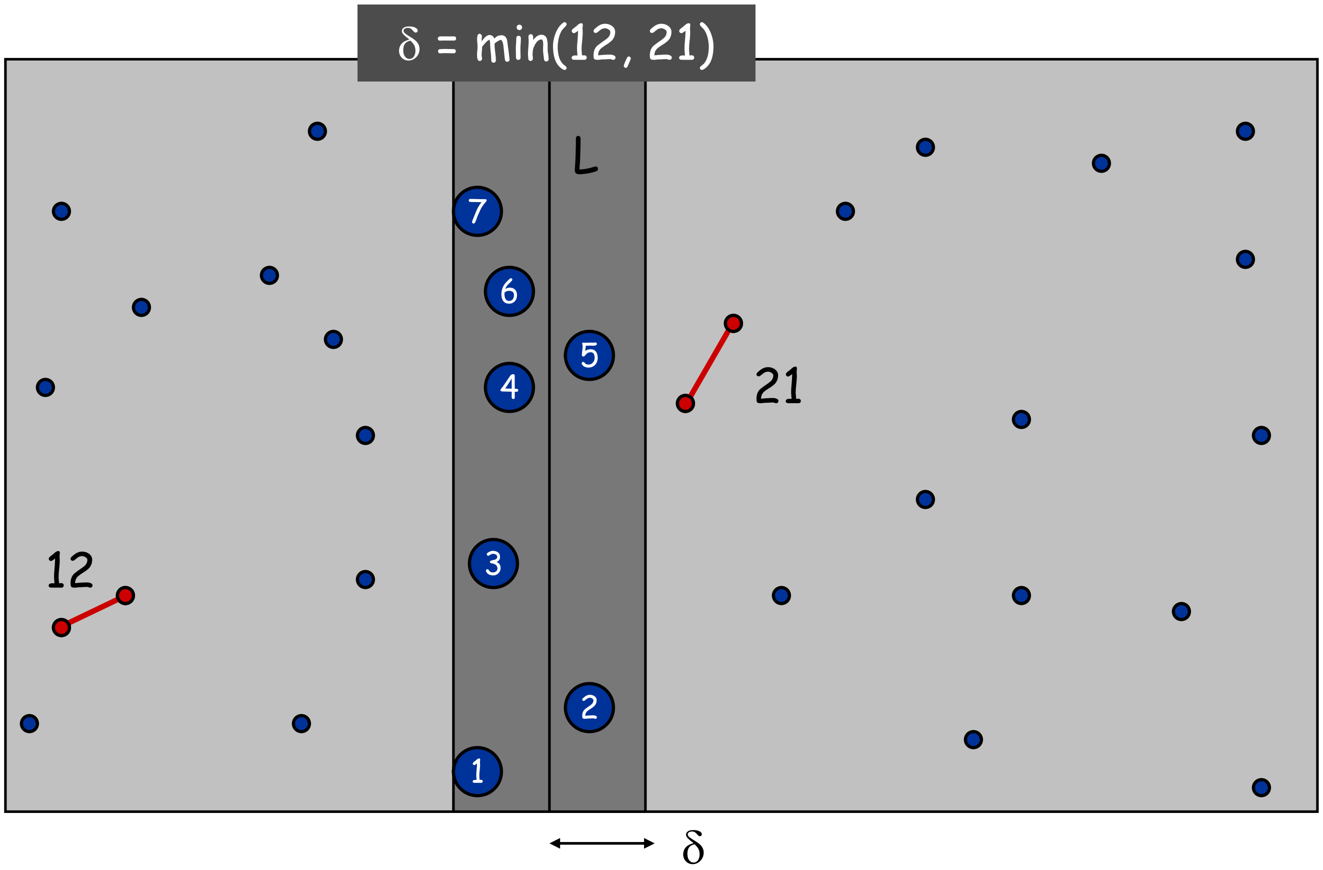
- Observation: only need to consider points within δ of line L.



Closest Pair of Points

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

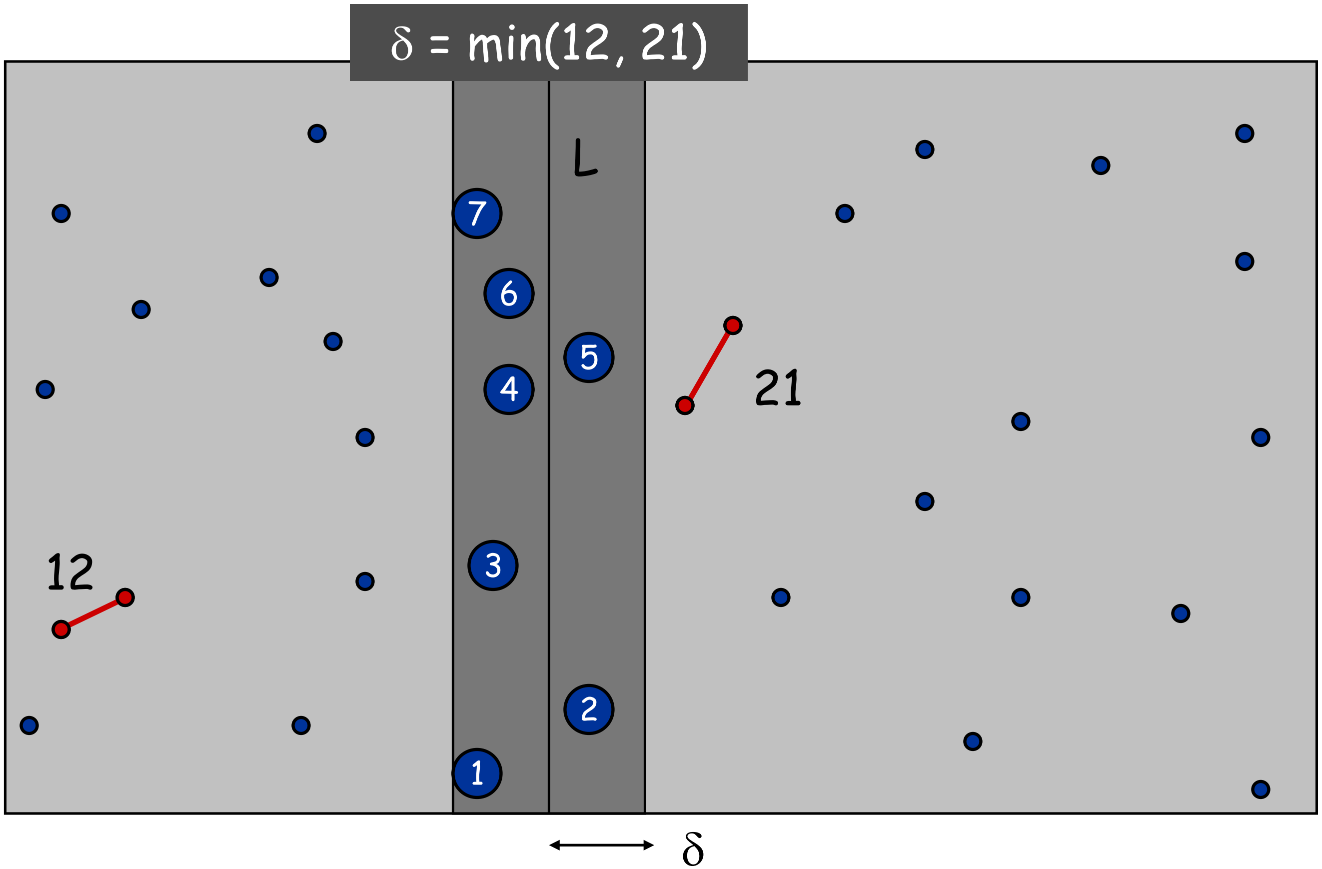
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.



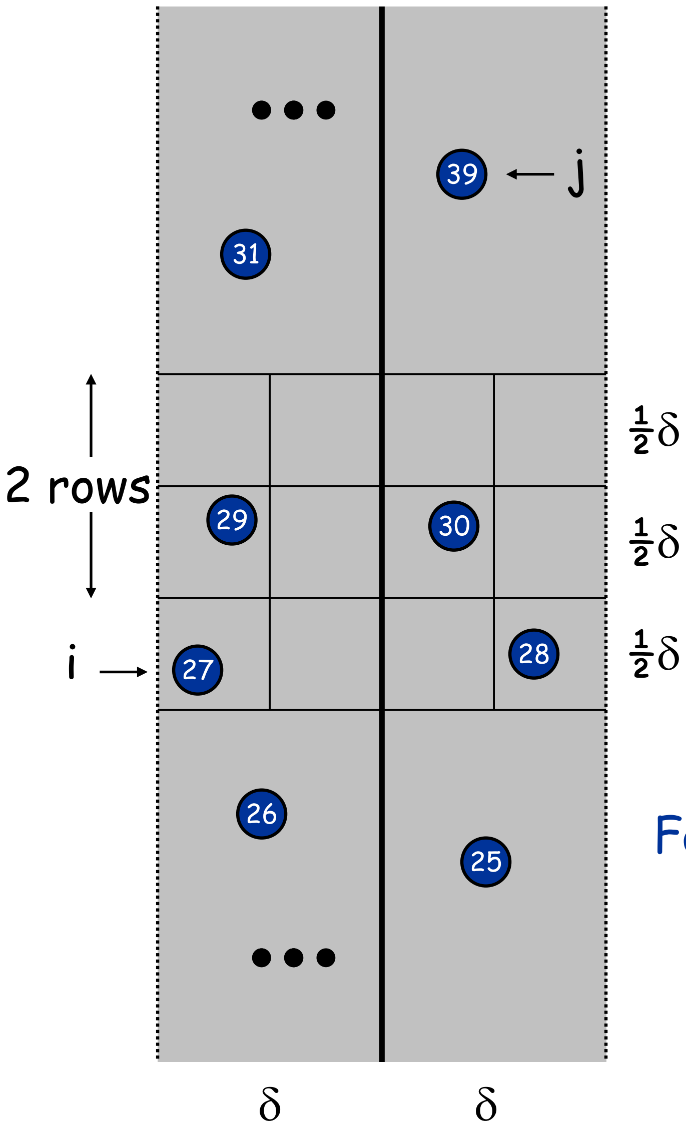
Closest Pair of Points

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!



Closest Pair of Points



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. 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.
}
```

Compute separation line L	$O(n \log n)$
Recursive calls	$2T(n / 2)$
Delete points	$O(n)$
Sort	$O(n \log n)$
Scan	$O(n)$

Closest Pair of Points

Recurrence of DQ algorithm

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

Use a generalization of case 2 in Master theorem

$f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.

- **$f(n)$ and $n^{\log_b a}$ grow at similar rates.**

Solution: **$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.**

How do we achieve O(n log n)? We will just give an outline, look at the book for details.

Solution (Idea)

- Use linear time median finding algorithm to determine line L and make the two recursive calls.

- Each recursive call returns its δ ***and*** the points in its region sorted by y coordinates.

- Merge: Sort y-lists by merging two returned sorted lists in time O(n). Same as mergesort!

Terminate recursion when n<4 or some other small constant and solve the resulting small problem by brute force.

$$T(n) \leq 2T\left(n/2\right) + O(n) \quad \Rightarrow \quad \mathrm{T}(n) = O(n \log n)$$