Objectives

- Wrap up Counting inversions
- Divide and conquer
  - Closest pair of points
  - Integer multiplication
  - Matrix multiplication

Counting Inversions: Implementation

Get out handouts

```
Sort-and-Count(L)
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (i_A, A) = Sort-and-Count(A)
   (i_B, B) = Sort-and-Count(B)
   (i, L) = Merge-and-Count(A, B)

   total_inversions = i_A + i_B + i
   return total_inversions and the sorted list L
```

- Merge-and-Count
  - Pre-condition. A and B are sorted.
  - Post-condition. L is sorted.

Counting Inversions: Implementation

```
Sort-and-Count(L)
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (iA, A) = Sort-and-Count(A)
   (iB, B) = Sort-and-Count(B)
   (i, L) = Merge-and-Count(A, B)

   total_inversions = iA + iB + i
   return total_inversions and the sorted list L
```

Recurrence relation?
Runtime of algorithm?

- Merge-and-Count
  - Pre-condition. A and B are sorted.
  - Post-condition. L is sorted.

Analysis

Recurrence Relation:

$T(n) \leq 2T(n/2) + O(n)$

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

```
Sort-and-Count(L)
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (iA, A) = Sort-and-Count(A)       T(n/2)
   (iB, B) = Sort-and-Count(B)       T(n/2)
   (i, L) = Merge-and-Count(A, B)  O(n)

   total_inversions = iA + iB + i
   return total_inversions and the sorted list L
```

CLOSEST PAIR OF POINTS

Computational Geometry

- Algorithms and data structures for geometrical objects
  - Points, line segments, polygons, etc.
  - Common motivator: large data sets → efficiency
- Some Applications
  - Graphics
  - Robotics
    - motion planning and visibility problems
  - Geographic information systems (GIS)
    - geometrical location and search, route planning

Closest Pair of Points

- Closest pair. Given *n* points in the plane, find a pair with smallest Euclidean distance between them.
  - Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired
fast algorithms for these problems

- Brute force?

March 13, 2019 CSCI211 - Sprenkle 7

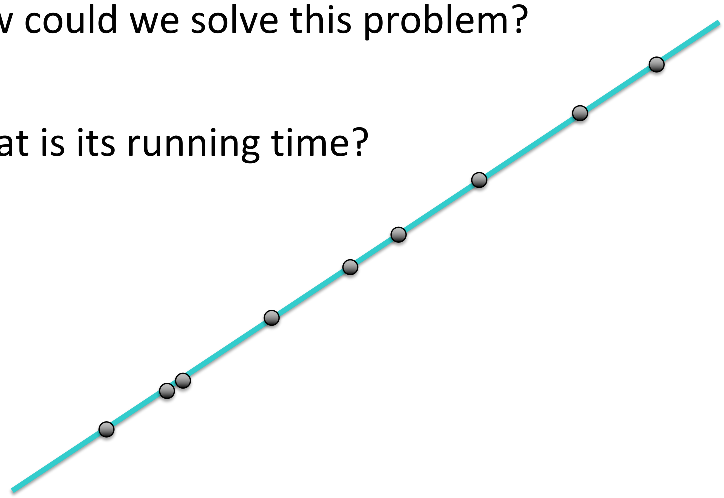
Closest Pair of Points

- Closest pair. Given *n* points in the plane, find a pair with smallest Euclidean distance between them.
  - Special case of nearest neighbor, Euclidean MST, Voronoi.
- Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons

March 13, 2019 CSCI211 - Sprenkle 8

Simplify: All Points on a Line

- How could we solve this problem?
- What is its running time?



Simplify: All Points on a Line

- How could we solve this problem?
  - Sort the points
    - Monotonically increasing x/y coordinates
    - No closer points than neighbors in sorted list
  - Step through, looking at the distances between each pair
- What is its running time?
  - O(n logn)

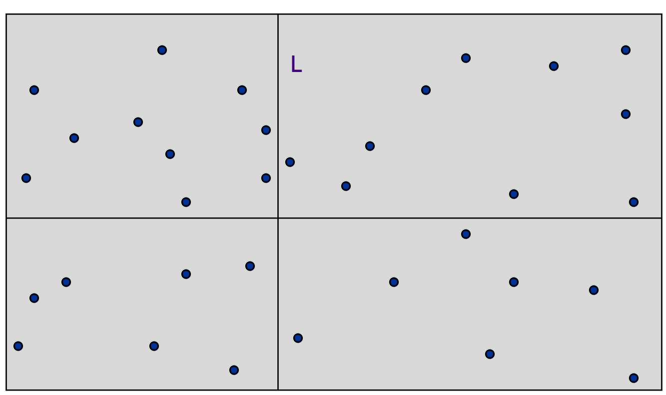
Why won't this work for 2D?

Closest Pair of Points

- Closest pair. Given *n* points in the plane, find a pair with smallest Euclidean distance between them.
  - Special case of nearest neighbor, Euclidean MST, Voronoi.
- 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

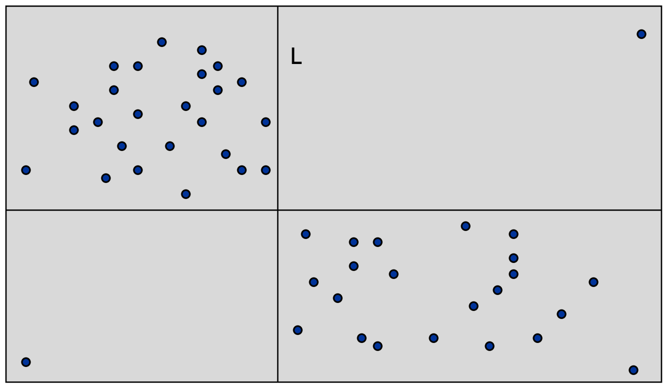
- Divide. Sub-divide region into 4 quadrants



Why does this seem to be a natural first step?
Any problems with implementing this approach?

Closest Pair of Points: First Attempt

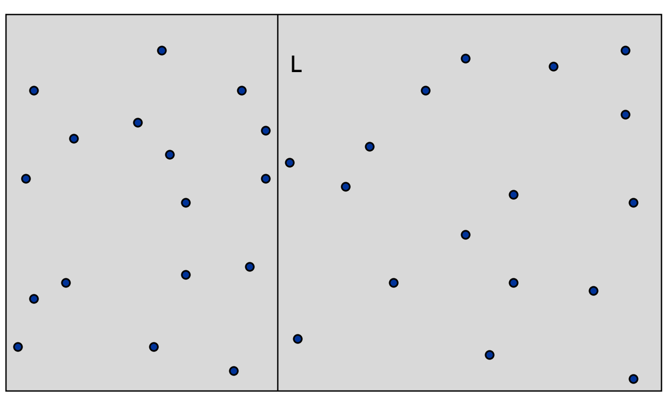
- Divide. Sub-divide region into 4 quadrants
- Obstacle. Impossible to ensure n/4 points in each piece



Closest Pair of Points

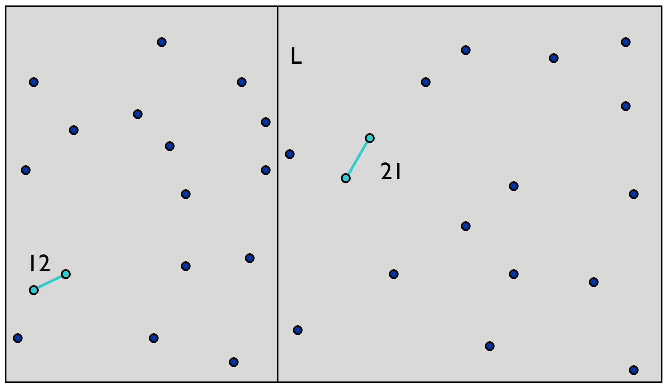
- Divide: draw vertical line L so that roughly ½n points on each side

How do we implement this?



Closest Pair of Points

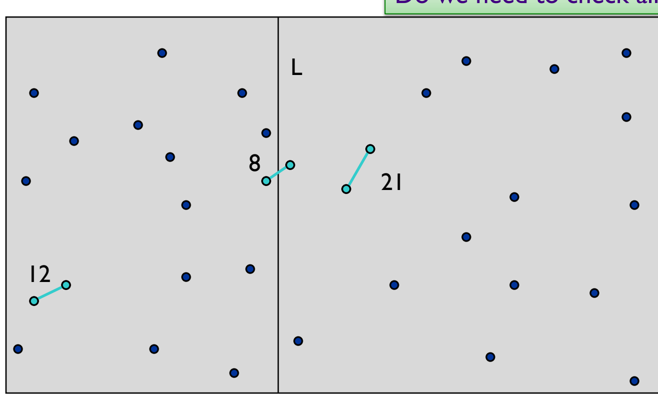
- Divide: draw vertical line L so that roughly ½n points on each side
- Conquer: find closest pair in each side recursively



Closest Pair of Points

- Divide: draw vertical line L so that roughly ½n 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

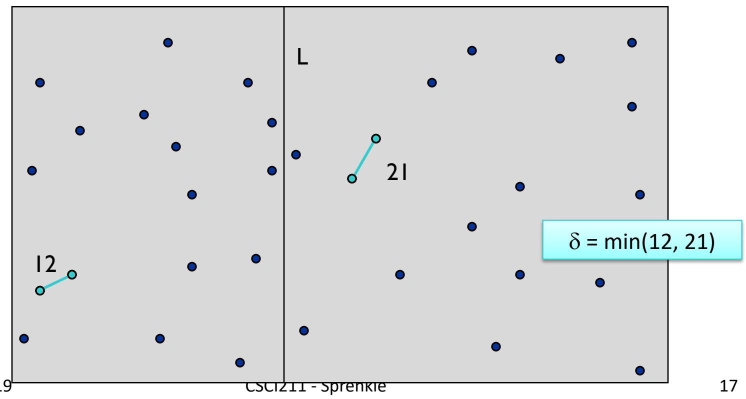
Do we need to check all pairs?



Closest Pair of Points

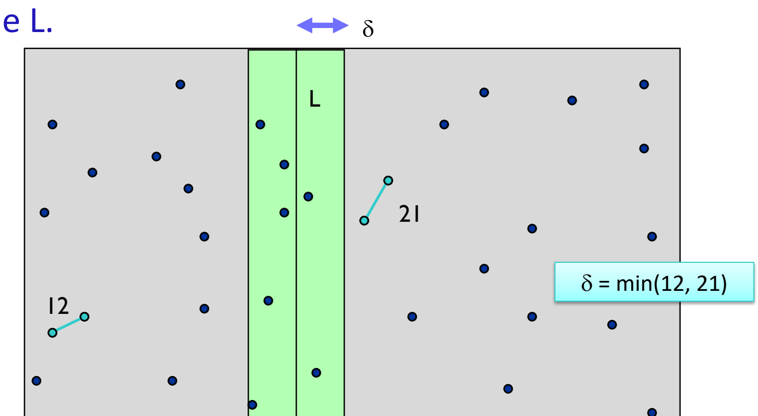
- Find closest pair with one point in each side, assuming that distance < δ

  where δ = min(left_min_dist, right_min_dist)



Closest Pair of Points

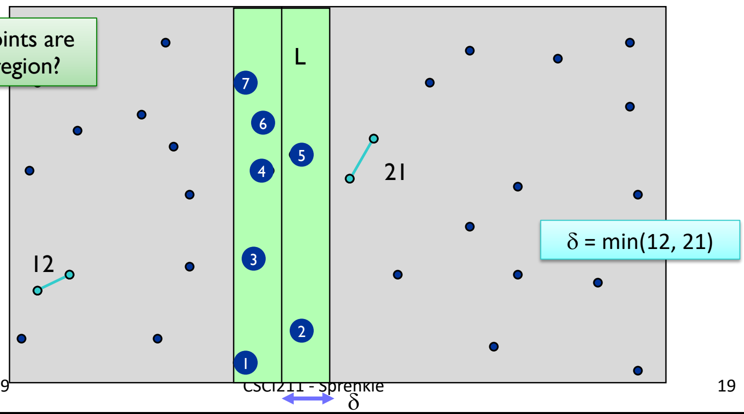
- Find closest pair with one point in each side, assuming that distance < δ.
  - Observation: only need to consider points within δ of line L.



Closest Pair of Points

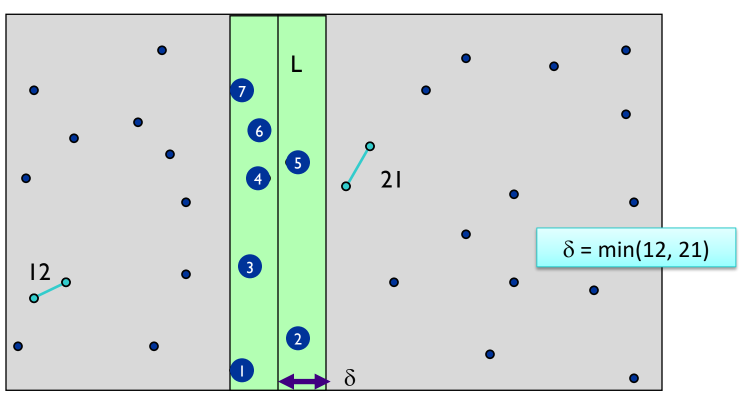
- Find closest pair w/ 1 point in each side, assuming that distance < δ.
  - Observation: only consider points within δ of line L
  - Sort points in 2δ-strip by their y coordinate

How many points are within that region?



Closest Pair of Points

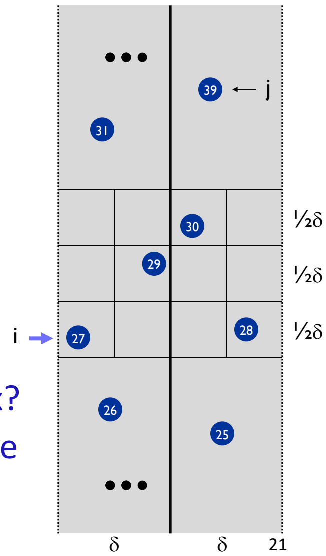
- Find closest pair w/ 1 point in each side, assuming that distance < δ
  - Observation: only consider points within δ of line L
  - Sort points in 2δ-strip by their y coordinate
    - Only checks distances of those within 11 positions in sorted list!



Analyzing Cost of Combining

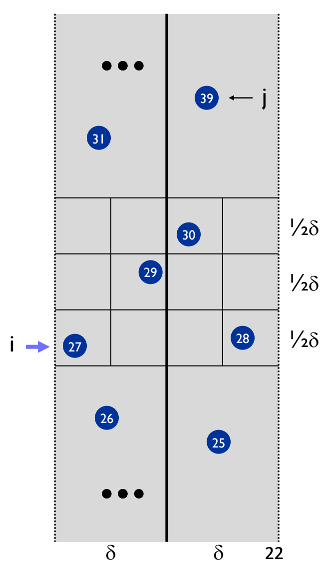
Prepare minds to be blown…

- 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
  - What is the distance of the box?
  - How many points can be in a box?
  - When do we know that points are $> \delta$ apart?



Analyzing Cost of Combining

- 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.



Cost of combining is therefore…?

Closest Pair Algorithm

```
Closest-Pair(p₁, …, pₙ)
   if n <= 3:
       return distance of closest pair by brute force

   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 7 neighbors. If any of these
   distances is less than δ, update δ.

   return δ
```

23

Closest Pair Algorithm

```
Closest-Pair(p₁, …, pₙ)
   if n <= 3:
       return distance of closest pair by brute force

   Compute separation line L such that half the points      O(n log n)
   are on one side and half on the other side.

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

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

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

   Scan points in y-order and compare distance between       O(n)
   each point and next 7 neighbors. If any of these
   distances is less than δ, update δ.

   return δ
```

Putting the recurrence relation together…

$T(n) = 2\,T(n/2) + O(n \log n)$

Closest Pair of Points: Analysis

- Running time. Solved in 5.2

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

- Can we achieve O(n log n)?

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

- Yes. Don't sort points in strip from scratch each time.
  - Each recursive call returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate
  - Sort by merging two pre-sorted lists

INTEGER AND MATRIX MULTIPLICATION

Integer Arithmetic

- Add. Given 2 *n*-digit integers a and b, compute a + b.
  - Algorithm?
  - Runtime?

1	1	1	1	1	1	0	1		
	1	1	0	1	0	1	0	1	a
+	0	1	1	1	1	1	0	1	b
1	0	1	0	1	0	0	1	0	a + b

March 13, 2019 CSCI211 - Sprenkle 27

Integer Arithmetic

- Add. Given 2 *n*-digit integers a and b, compute a + b.
  - Algorithm?
  - Runtime?

1	1	1	1	1	1	0	1		
	1	1	0	1	0	1	0	1	a
+	0	1	1	1	1	1	0	1	b
1	0	1	0	1	0	0	1	0	a + b

O(n) operations

March 13, 2019 CSCI211 - Sprenkle 28

Integer Arithmetic

- Multiply. Given 2 *n*-digit integers a and b, compute a × b.
  - Algorithm?
  - Runtime?

```
   1 1 0 1 0 1 0 1    a
 * 0 1 1 1 1 1 0 1    b
                      a × b
```


Integer Arithmetic

- Multiply. Given 2 *n*-digit integers a and b, compute a × b.
  - Brute force solution: $\Theta(n^2)$ bit operations

Goal: Faster algorithm

```
              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
          1 1 0 1 0 1 0 1
        1 1 0 1 0 1 0 1
      1 1 0 1 0 1 0 1
    1 1 0 1 0 1 0 1
  1 1 0 1 0 1 0 1
0 0 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0 0 0 0 0 0 1
```

Divide-and-Conquer Multiplication: Warmup

- To multiply 2 n-digit integers:
  - Multiply 4 ½n-digit integers
  - Add 2 ½n-digit integers and shift to obtain result

Higher order bits Lower order bits x=10001101

Shift x_1=1000 x_0=1101

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot \underset{A}{x_1 y_1} + 2^{n/2} \cdot (\underset{B}{x_1 y_0} + \underset{C}{x_0 y_1}) + \underset{D}{x_0 y_0}$$

What is the recurrence relation?
- How many subproblems?
- What is merge cost?
- What is its runtime?

Divide-and-Conquer Multiplication: Warmup

- To multiply 2 n-digit integers:
  - Multiply 4 ½n-digit integers
  - Add 2 ½n-digit integers and shift to obtain result

Higher order bits Lower order bits

Shift

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot \underset{A}{x_1 y_1} + 2^{n/2} \cdot (\underset{B}{x_1 y_0} + \underset{C}{x_0 y_1}) + \underset{D}{x_0 y_0}$$

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

assumes n is a power of 2

Not an improvement over brute force

Karatsuba Multiplication

Anatolii Alexeevich Karatsuba

- To multiply two n-digit integers:
  - Add 2 ½n digit integers
  - Multiply 3 ½n-digit integers
  - Add, subtract, and shift ½n-digit integers to obtain result

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^{n} \cdot \underbrace{x_1 y_1}_{A} + 2^{n/2} \cdot \big(\underbrace{(x_1 + x_0)(y_1 + y_0)}_{B} - \underbrace{x_1 y_1}_{A} - \underbrace{x_0 y_0}_{C} \big) + \underbrace{x_0 y_0}_{C}
\end{aligned}
$$

What is the recurrence relation? Runtime?

Karatsuba Multiplication

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

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^{n} \cdot \underbrace{x_1 y_1}_{A} + 2^{n/2} \cdot \big(\underbrace{(x_1 + x_0)(y_1 + y_0)}_{B} - \underbrace{x_1 y_1}_{A} - \underbrace{x_0 y_0}_{C} \big) + \underbrace{x_0 y_0}_{C}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{T}(n) &\le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \\
\Rightarrow \mathrm{T}(n) &= O(n^{\log_2 3}) = O(n^{1.585})
\end{aligned}
$$

Looking Ahead

- PS7 due Friday
- Exam 2 handed out Friday
- Moving to Dynamic Programming on Friday!