Objectives

- Divide and conquer
  - Closest pair of points
  - Integer multiplication
  - Matrix multiplication

Divide-and-Conquer

Divide et impera.
Veni, vidi, vici.
- *Julius Caesar*

- Divide-and-conquer process
  - Break up problem into several parts
  - Solve each part recursively
  - Combine solutions to sub-problems into overall solution
- Most common usage
  - Break up problem of size n into two equal parts of size ½n
  - Solve two parts recursively
  - Combine two solutions into overall solution

Review: Recurrence Relations

- Use recurrences to analyze/determine the run time of divide and conquer algorithms
  - Number of sub problems
  - Size of sub problems
  - Number of times divided (number of levels)
  - Cost of merging problems
- How to solve
  - Unrolling
  - Substitution

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

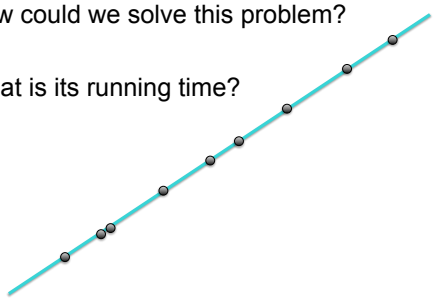
- Brute force?

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

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)

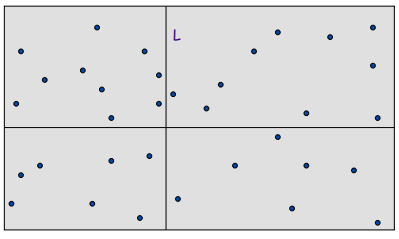
Why won't this work for 2D?

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*

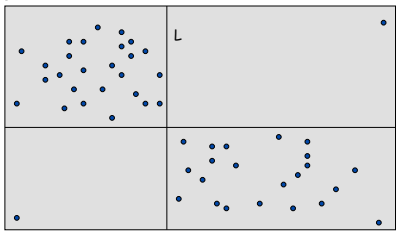
Closest Pair of Points: First Attempt

- Divide. Sub-divide region into 4 quadrants



Closest Pair of Points: First Attempt

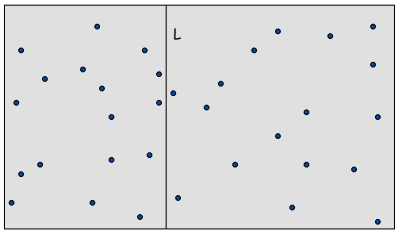
- Divide. Sub-divide region into 4 quadrants
- Obstacle. Impossible to ensure n/4 points in each piece



Closest Pair of Points

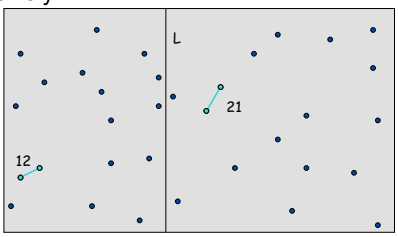
- Divide: draw vertical line L so that roughly ½n points on each side

How do we implement this?



Closest Pair of Points

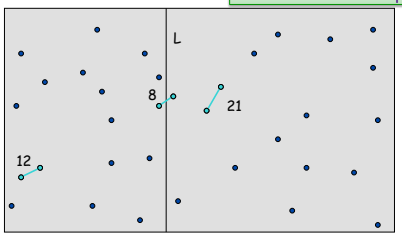
- Divide: draw vertical line L so that roughly ½n points on each side
- Conquer: find closest pair in each side recursively



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

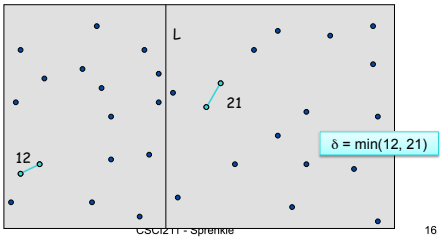
Do we need to check all pairs?



Closest Pair of Points

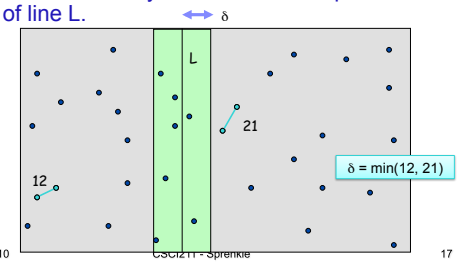
- Find closest pair with one point in each side, assuming that distance < δ

where δ = min(left_min_dist, right_min_dist)



Closest Pair of Points

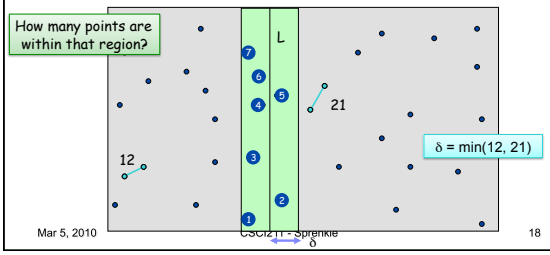
- Find closest pair with one point in each side, assuming that distance < δ.
  - Observation: only need to consider points within δ of line L.



Closest Pair of Points

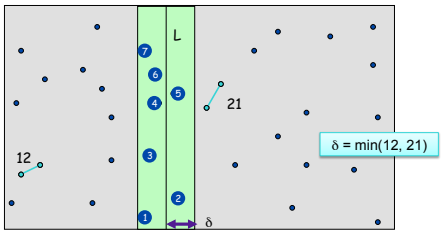
- Find closest pair w/ 1 point in each side, assuming that distance < δ.
  - Observation: only consider points within δ of line L
  - Sort points in 2δ-strip by their y coordinate

How many points are within that region?



Closest Pair of Points

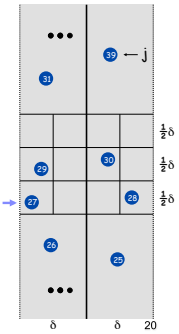
- Find closest pair w/ 1 point in each side, assuming that distance < δ.
  - Observation: only consider points within δ of line L
  - Sort points in 2δ-strip by their y coordinate
    - Only checks distances of those within 11 positions in sorted list!



Analyzing Cost of Combining

Prepare minds to be blown…

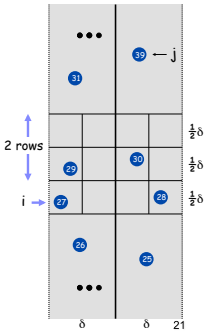
- Def. Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate
- Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ
  - What is the distance of the box?
  - How many points can be in a box?
  - When do we know that points are > δ apart?



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 ½δ-by-½δ box
  - Two points at least 2 rows apart have distance ≥ 2(½δ). ▪
- Fact. Still true if we replace 12 with 7.

Cost of combining is therefore…?



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

   return δ
```


Closest Pair Algorithm

```
Closest-Pair(p1, …, pn)
   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
   each point and next 7 neighbors. If any of these      O(n)
   distances is less than δ, update δ.

   return δ
```

T(n) = 2 T(n/2) + O(n log n)

Closest Pair of Points: Analysis

- Running time. Solved in 5.2

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

- Can we achieve O(n log n)?

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

- Yes. Don't sort points in strip from scratch each time.
  - Each recursive 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 two 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
+   0 1 1 1 1 1 0 1
-----------------
1 0 1 0 1 0 0 1 0
```

O(n) operations

Integer Arithmetic

- Multiply. Given two n-digit integers a and b, compute a × b

Algorithm?
Runtime?

```
  1 1 0 1 0 1 0 1
* 0 1 1 1 1 1 0 1
```


Integer Arithmetic

- Multiply. Given two 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 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 1 0
```


Divide-and-Conquer Multiplication: Warmup

- To multiply two n-digit integers:
  - Multiply four ½ n-digit integers
  - Add two ½ n-digit integers and shift to obtain result

Higher order bits Lower order bits

Shift

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

A B C D

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

Divide-and-Conquer Multiplication: Warmup

- To multiply two n-digit integers:
  - Multiply four ½ n-digit integers
  - Add two ½ n-digit integers and shift to obtain result

Higher order bits Lower order bits

Shift

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

A B C D

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

assumes n is a power of 2

Not an improvement over brute force

PS4 Feedback

- Whenever you develop an algorithm, **analyze** its running time (e.g., Prob 3)
- Be explicit
  - Explicitly state the metric, who is the pronoun (say which algorithm), etc.
  - Creative in solution, not in explanation

Assignments

- Continue reading Chapter 5, start Chapter 6
- PS6 due next Friday
  - May want to try to implement problems 2 and 3 (to some extent) to help ensure that your algorithm is correct