Objectives

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

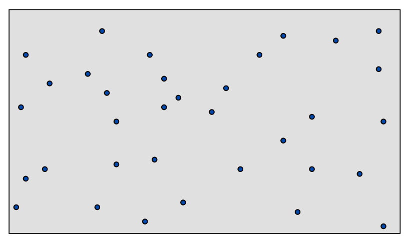
Reviewing Closest Pair of Points

Closest Pair of Points

- Closest pair. Given *n* points in the plane, find a pair with smallest Euclidean distance between them.
  - 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*

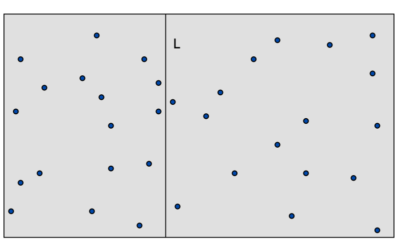
Closest Pair of Points

- Recall the approach?



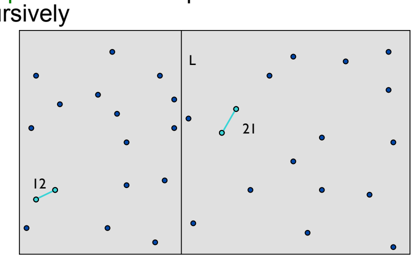
Closest Pair of Points

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



Closest Pair of Points

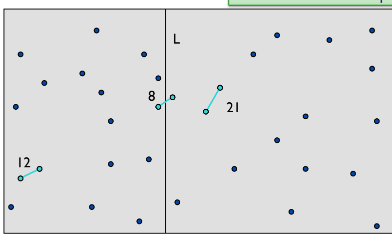
- Divide: draw vertical line L so that roughly ½n points on each side
- Conquer: find closest pair in each side recursively



Closest Pair of Points

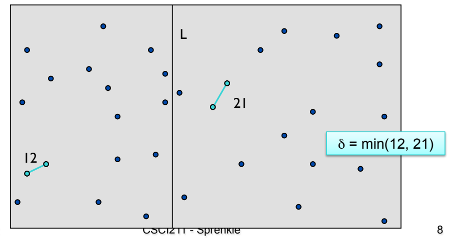
- Divide: draw vertical line L so that roughly ½n points on each side
- Conquer: find closest pair in each side recursively
- Combine: find closest pair with one point in each side *seems like $\Theta(n^2)$*
- Return best of 3 solutions

Do we need to check all pairs?



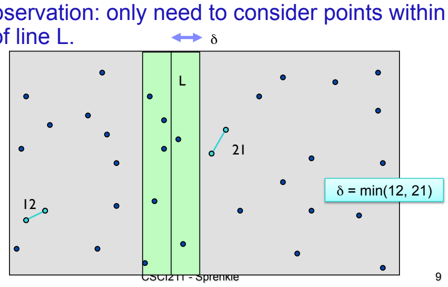
Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < δ
  - where δ = min(left_min_dist, right_min_dist)



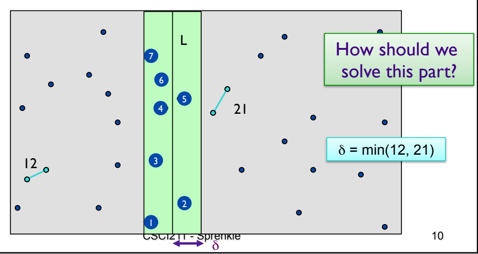
Closest Pair of Points

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



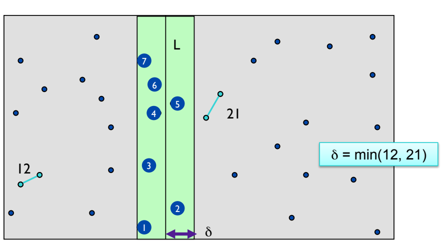
Closest Pair of Points

- 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



Closest Pair of Points

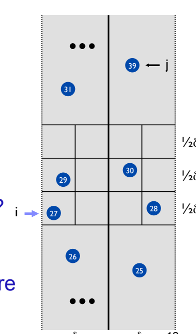
- Find closest pair w/ 1 point in each side, assuming distance < δ.
  - Observation: only consider points within δ of line L
  - Sort points in 2δ-strip by their y coordinate
    - **Only checks distances of those within 11 positions in sorted list!**



Analyzing Cost of Combining

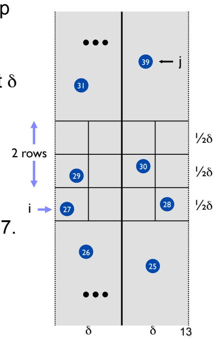
Prepare minds to be blown…

- 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 $\geq 2(½δ)$. ▪
- 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
```

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

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

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
  + 0 1 1 1 1 1 0 1
  -----------------
  1 0 1 0 1 0 0 1 0
```

O(n) operations

Integer Arithmetic

- Multiply. Given 2 *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 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 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 0
```


Divide-and-Conquer Multiplication: Warmup

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

Higher order bits

Lower order bits

Shift

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

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 4 ½ n-digit integers
  - Add 2 ½ n-digit integers and shift to obtain result

Higher order bits

Lower order bits

Shift

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

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

assumes n is a power of 2

Not an improvement over brute force

Karatsuba Multiplication

- 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

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

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

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

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

$$
\Rightarrow \mathrm{T}(n) = O(n^{\log_2 3}) = O(n^{1.585})
$$

MATRIX MULTIPLICATION

Matrix Multiplication

- Given 2 n-by-n matrices A and B, compute C = AB

$$
c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}
$$

$$
\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}
$$

  - Ex: $c_{12} = a_{11} b_{12} + a_{12} b_{22} + a_{13} b_{32} + \ldots + a_{1n} b_{n2}$

Solve using brute force …

Matrix Multiplication

- Given 2 n-by-n matrices A and B, compute C = AB

$$
c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}
$$

$$
\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}
$$

  - Ex: $c_{12} = a_{11} b_{12} + a_{12} b_{22} + a_{13} b_{32} + \ldots + a_{1n} b_{n2}$

- Brute force. $\Theta(n^3)$ arithmetic operations
- Fundamental question: Can we improve upon brute force?

Matrix Multiplication: Warmup

- Divide: partition A and B into ½n-by-½n blocks
- Conquer: multiply 8 ½n-by-½n recursively
- Combine: add appropriate products using 4 matrix additions

$$
\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}
$$

$$
\begin{aligned}
C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\
C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\
C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\
C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\end{aligned}
$$

Recurrence relation? Runtime?

Matrix Multiplication: Warmup

- Divide: partition A and B into ½n-by-½n blocks
- Conquer: multiply 8 ½n-by-½n recursively
- Combine: add appropriate products using 4 matrix additions

$$
\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}
$$

$$
\begin{aligned}
C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\
C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\
C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\
C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\end{aligned}
$$

$$
\mathrm{T}(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow \mathrm{T}(n) = \Theta(n^3)
$$

Matrix Multiplication: Key Idea

Trading expensive multiplication for less expensive addition/subtraction

- Multiply 2-by-2 block matrices with only 7 multiplications and 15 additions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

Fast Matrix Multiplication

[Strassen, 1969]

- Divide: partition A and B into ½n-by-½n blocks
- Compute: 14 ½n-by-½n matrices via 10 matrix additions
- Conquer: multiply 7 ½n-by-½n matrices recursively
- Combine: 7 products into 4 terms using 8 matrix additions
- Analysis.
  - Assume n is a power of 2.
  - T(n) = # arithmetic operations.

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Fast Matrix Multiplication in Practice

- Implementation issues: problems with putting theory into practice
  - Sparsity
  - Caching effects
  - Numerical stability
    - Theoretically correct but possible problems with round off errors, etc
  - Odd matrix dimensions
  - Crossover to classical algorithm around n = 128

Fast Matrix Multiplication in Practice

- Common misperception: "Strassen is only a theoretical curiosity."
  - Advanced Computation Group at Apple Computer reports **8x** speedup on G4 Velocity Engine when n ~ 2,500
  - Range of instances where it's useful is a subject of controversy
- Can "Strassenize" Ax=b, determinant, eigenvalues, and other matrix ops

Fast Matrix Multiplication in Theory

- Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
- A. Yes! [Strassen, 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$
- Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
- A. Impossible [Hopcroft and Kerr, 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$
- Q. Two 3-by-3 matrices with only 21 scalar multiplications?
- A. Also impossible $\Theta(n^{\log_3 21}) = O(n^{2.77})$
- Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
- A. Yes! [Pan, 1980] $\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$

- Decimal wars.
  - December, 1979: $O(n^{2.521813})$
  - January, 1980: $O(n^{2.521801})$

Fast Matrix Multiplication in Theory

- Best known. $O(n^{2.376})$ [Coppersmith-Winograd, 1987]
  - But *really* large constant
- Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

- Caveat. Theoretical improvements to Strassen are progressively less practical.

Problem Set 5 Feedback

- Don't forget to analyze the runtime of every algorithm you write
- How do you prove optimality of Greedy algorithms?

Greedy Stays Ahead Proofs

1. Define your solutions
   - Describe the form of your greedy solution and of some other solution (possibly the optimal solution)
     - Example: Let A be the solution constructed by the greedy algorithm and O be a solution
2. Find a measure
   - Find a measure by which greedy stays ahead of the optimal solution
     - Ex: Let $a_1, \ldots, a_k$ be the first k measures of greedy algorithm and $o_1, \ldots, o_m$ be the first m measures of other solution (sometimes m = k)
3. Prove greedy stays ahead
   - Show that the partial solutions constructed by greedy are always just as good as the optimal solution's initial segments based on the measure
     - Ex: for all indices $r \le \min(k,m)$, prove by **induction** that $a_r \ge o_r$ or $a_r \le o_r$
   - Use the greedy algorithm to help you argue the inductive step
4. Prove optimality
   - Prove that since greedy stays ahead of the other solution with respect to the measure, then the greedy solution is optimal

Greedy Exchange Proofs

1. Label your algorithm's solution and a general solution.
   - Example: let $A = \{a_1, a_2, \ldots, a_k\}$ be the solution generated by your algorithm, and let $O = \{o_1, o_2, \ldots, o_m\}$ be an arbitrary (or optimal) feasible solution.
2. Compare greedy with other solution.
   - Assume that your arbitrary/optimal solution is not the same as your greedy solution (since otherwise, you are done).
   - Typically, can isolate a simple example of this difference, such as:
     1. There is an element $e \in O$ that $\notin A$ and an element $f \in A$ that $\notin O$
     2. 2 consecutive elements in O are in a different order than in A (i.e., there is an *inversion*).
3. Exchange.
   - Swap the elements in question in O (either ① swap one element out and another in or ② swap the order of the elements) and argue that solution is no worse than before.
   - Argue that if you continue swapping, you eliminate all differences between O and A in a *finite* # of steps without worsening the solution's quality.
   - Thus, the greedy solution produced is just as good as any optimal solution, and hence is optimal itself.

Assignments

- Wiki for 5.2-5.5 due Tuesday
- Chapter 6 starts Monday
- PS7 due Friday
  - May want to try to implement solutions (to some extent) to help ensure that your algorithm is correct