Objectives

Algorithm Approach: Divide and Conquer

- Recurrence Review
- Integer Multiplication
- Matrix Multiplication

Review: Counting Inversions

Recurrence Relation:

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

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

```
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
   (rA, B) ← Sort-and-Count(A)      T(n/2)
   (rB, B) ← Sort-and-Count(B)      T(n/2)
   (r , L) ← Merge-and-Count(A, B)  O(n)

   return r = rA + rB + r and the sorted list L
```


Review: 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     O(n)
   each point and next 7 neighbors. If any of these
   distances is less than δ, update δ.

   return δ
```

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

Know Your Recurrence Relations

What algorithm has this recurrence relation?
What is that algorithm's running time?

Recurrence	Algorithm	Running Time
T(n) = T(n/2) + O(1)		
T(n) = T(n-1) + O(1)		
T(n) = 2 T(n/2) + O(1)		
T(n) = T(n-1) + O(n)		
T(n) = 2 T(n/2) + O(n)	Merge Sort	O(n log n)

Know Your Recurrence Relations

What algorithm has this recurrence relation?
What is that algorithm's running time?

Recurrence	Algorithm	Running Time
T(n) = T(n/2) + O(1)	Binary Search	O(log n)
T(n) = T(n-1) + O(1)	Sequential/ Linear Search	O(n)
T(n) = 2 T(n/2) + O(1)	Binary Tree Traversal	O(n)
T(n) = T(n-1) + O(n)	Selection Sort	$O(n^2)$
T(n) = 2 T(n/2) + O(n)	Merge Sort	O(n log n)

INTEGER 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 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

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

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 four ½ n-digit integers
- Add two ½ n-digit integers and shift to obtain result

Higher order bits Lower order bits

Shift

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

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

assumes n is a power of 2

Not an improvement over brute force

Karatsuba Multiplication

To multiply two n-digit integers:

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

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

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

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

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

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

MATRIX MULTIPLICATION

Matrix Multiplication

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

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

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

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

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

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

- Trading expensive multiplication for less expensive addition/subtraction

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

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

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

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

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

Analysis.

- Assume n is a power of 2.
- T(n) = # arithmetic operations.

Fast Matrix Multiplication in Practice

Implementation issues.

- 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

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

Remark. 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.

MIDTERM FEEDBACK

Problem 1

O is an *upperbound*

- Defn: Bounded by a constant

"at least" an upperbound doesn't make sense

Problem 2

$F_6 = \log n$

$F_7 = n^{1/2}$

$F_4 = n \log n$

$F_5 = n^3$

$F_1 = 2^n = 2 * 2 * \ldots * 2$ n times

$F_3 = n! = n * n-1 * n-2 * \ldots * 1$

$F_2 = 2^{2^\wedge n} = 2^{n+(2^\wedge n-1)} = 2 * 2 * \ldots * 2$ 2^n times

Problem 3

Creating the graph: $O(n^2)$

Need representation/ implementation, costs, runtimes

- Adjacency matrix
- For each node, keep count of number of red edges, blue edges
  - Saves time later

Removing invalid nodes (nodes w/ less than 5 red or blue edges): $O(n^2)$

- When removing node, remove its edges $O(n)$
  - Decrease the connected node's red or blue count
- A node will never become valid after invalid nodes are removed

Remaining graph's nodes represent people to invite

$O(n^2)$: *Efficient* algorithm because *polynomial* time

Problem 4

Algorithm: Shortest Job First $O(n \log n)$

- Sort jobs in order of increasing wait time
- Wait on customers in this order

Prove that algorithm is optimal

- Similar to minimizing lateness problem
- What happens if two customers are *inverted*?
  - All previous *k* customers have same wait time (W)
  - Inversion: Customer k+1 and k+2 have service times $t_{k+1} < t_{k+2}$ but k+2 is served first
  - SJF: $W + t_{k+1}$; Other: $W + t_{k+2}$ → SJF < Other
  - Inversions → increase wait time

Plan for the Week

Chapter 6: Dynamic programming

- More powerful technique

Friday: Problem set due