Divide and Conquer

- Algorithm paradigm
  - Break problems into subproblems until easy to solve
  - Work is split between ‘divide’, ‘combine’, and ‘base’ components
- Standard examples
  - MergeSort and QuickSort
- Analysis tool: Recurrences

Matrix Multiplication

- N x N Matrix, A B = C

```java
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

Recursive Matrix Multiplication

- Multiply 2 x 2 Matrices:
  - | r | a | b |
  - | s | c | d |
  - r = ae + bf
  - s = ag + bh
  - t = ce + df
  - u = cg + dh

A N x N matrix can be viewed as a 2 x 2 matrix with entries that are (N/2) x (N/2) matrices.

The recursive matrix multiplication algorithm recursively multiplies the (N/2) x (N/2) matrices and combines them using the equations for multiplying 2 x 2 matrices.

- How many recursive calls are made at each level?
- How much work in combining the results?
- What is the recurrence?
What is the runtime for the recursive Matrix Multiplication Algorithm?

- Recurrence:

\[ T(n) = 7T(n/2) + cn^2 \]

What is the runtime?

\[ \log_7 7 = 2.8073549221 \]

Recurrence for Strassen's Algorithm

- \( T(n) = 7T(n/2) + cn^2 \)
- What is the runtime?

Strassen's Algorithm

Multiply 2 x 2 Matrices:

\[
\begin{array}{c|c|c|c}
| r & s | & | a & b | & | e & g | \\
| t & u | & | c & d | & | f & h | \\
\end{array}
\]

Where:

\[
\begin{align*}
p_1 &= (b - d)(f + h) \\
p_2 &= (a + d)(e + h) \\
p_3 &= (a - c)(e + g) \\
p_4 &= (a + b)h \\
p_5 &= a(g - h) \\
p_6 &= d(f - e) \\
p_7 &= (c + d)e
\end{align*}
\]

Recurrence for Strassen's Algorithms

- \( T(n) = 7T(n/2) + cn^2 \)
- What is the runtime?

Inversion Problem

- Let \( a_1, \ldots, a_n \) be a permutation of \( 1 \ldots n \)
- \((a_i, a_j)\) is an inversion if \( i < j \) and \( a_i > a_j \)

\( 4, 6, 1, 7, 3, 2, 5 \)

- Problem: given a permutation, count the number of inversions
- This can be done easily in \( O(n^2) \) time
  - Can we do better?

Application

- Counting inversions can be used to measure how close ranked preferences are
  - People rank 20 movies, based on their rankings you cluster people who like that same type of movie
Counting Inversions

Count inversions on lower half
Count inversions on upper half
Count the inversions between the halves

Problem – how do we count inversions between sub problems in $O(n)$ time?
• Solution – Count inversions while merging

Use the merge algorithm to count inversions

Computing the Median

• Given $n$ numbers, find the number of rank $n/2$
• One approach is sorting
  — Sort the elements, and choose the middle one
  — Can you do better?
• Selection, given $n$ numbers and an integer $k$, find the $k$-th largest
Select(A, k)

Select(A, k){
  Choose element x from A
  \( S_1 = \{ y \in A \mid y < x \} \)
  \( S_2 = \{ y \in A \mid y > x \} \)
  \( S_3 = \{ y \in A \mid y = x \} \)
  if (\( |S_2| \geq k \))
    return Select(S_2, k)
  else if (\( |S_2| + |S_3| \geq k \))
    return x
  else
    return Select(S_1, k - |S_2| - |S_3|)
}

Deterministic Selection

- What is the run time of select if we can guarantee that choose finds an x such that \( |S_1| < 3n/4 \) and \( |S_2| < 3n/4 \) in O(n) time

- What is the run time of select if we can guarantee that choose finds an x such that \( |S_1| < 3n/4 \) and \( |S_2| < 3n/4 \) in O(n) time

BFPRT Algorithm

- A very clever choose algorithm . . .

- Deterministic algorithm that guarantees that \( |S_1| < 3n/4 \) and \( |S_2| < 3n/4 \)

- Actual recurrence is:
  \[ T(n) \leq T(3n/4) + T(n/5) + c \cdot n \]

Recursive Multiplication Algorithm (First attempt)

\[ x = x_1 2^{n/2} + x_0 \]
\[ y = y_1 2^{n/2} + y_0 \]
\[ xy = (x_1 2^{n/2} + x_0) (y_1 2^{n/2} + y_0) \]
\[ = x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0 \]

Recurrence:
Run time:

Simple algebra

\[ x = x_1 2^{n/2} + x_0 \]
\[ y = y_1 2^{n/2} + y_0 \]
\[ xy = x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0 \]
\[ p = (x_1 + x_0)(y_1 + y_0) = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0 \]
Karatsuba’s Algorithm

Multiply n-digit integers $x$ and $y$

Let $x = x_1 2^{n/2} + x_0$ and $y = y_1 2^{n/2} + y_0$

Recursively compute

$a = x_1 y_1$
$b = x_0 y_0$
$p = (x_1 + x_0)(y_1 + y_0)$

Return $a2^n + (p - a - b)2^{n/2} + b$

Recurrence: $T(n) = 3T(n/2) + cn$