CSE 417
Algorithms and Complexity

Autumn 2020
Lecture 16
Divide and Conquer Algorithms
Announcements

• Homework 5, Due Friday
Matrix Multiplication

- N X N Matrix, \( A \times B = C \)

```c
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++) {
        int t = 0;
        for (int k = 0; k < n; k++)
            t = t + A[i][k] * B[k][j];
        C[i][j] = t;
    }
```
Recursive Matrix Multiplication

Multiply 2 x 2 Matrices:
\[
\begin{bmatrix}
  r & s \\
  t & u
\end{bmatrix}
= 
\begin{bmatrix}
  a & b \\
  c & d
\end{bmatrix}
\begin{bmatrix}
  e & g \\
  f & h
\end{bmatrix}
\]

- \[ r = ae + bf \]
- \[ s = ag + bh \]
- \[ t = ce + df \]
- \[ u = cg + dh \]

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.
Recursive Matrix Multiplication

• How many recursive calls are made at each level?

• How much work in combining the results?

• What is the recurrence?
What is the run time for the recursive Matrix Multiplication Algorithm?

• Recurrence:
Strassen’s Algorithm

Multiply 2 x 2 Matrices:

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

- \(r = p_1 + p_2 - p_4 + p_6\)
- \(s = p_4 + p_5\)
- \(t = p_6 + p_7\)
- \(u = p_2 - p_3 + p_5 - p_7\)

Where:

- \(p_1 = (b - d)(f + h)\)
- \(p_2 = (a + d)(e + h)\)
- \(p_3 = (a - c)(e + g)\)
- \(p_4 = (a + b)h\)
- \(p_5 = a(g - h)\)
- \(p_6 = d(f - e)\)
- \(p_7 = (c + d)e\)

From Aho, Hopcroft, Ullman 1974
Recurrence for Strassen’s Algorithms

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

$log_2 7 = 2.8073549221$
Strassen’s Algorithms

• Treat $n \times n$ matrices as $2 \times 2$ matrices of $n/2 \times n/2$ submatrices.
• Use Strassen’s trick to multiply $2 \times 2$ matrices with 7 multiplies.
• Base case standard multiplication for single entries.
• Recurrence: $T(n) = 7 \ T(n/2) + cn^2$.
• Solution is $O(7^{\log n}) = O(n^{\log 7})$ which is about $O(n^{2.807})$. 
Strassen’s Algorithms

• Treat $n \times n$ matrices as $2 \times 2$ matrices of $n/2 \times n/2$ submatrices
• Use Strassen’s trick to multiply $2 \times 2$ matrices with 7 multiplies
• Base case standard multiplication for single entries
• Recurrence: $T(n) = 7 \, T(n/2) + cn^2$
• Solution is $O(7^{\log n}) = O(n^{\log 7})$ which is about $O(n^{2.807})$
Inversion Problem

• Let \( a_1, \ldots, a_n \) be a permutation of 1 . . 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

| 11 | 12 | 4 | 1 | 7 | 2 | 3 | 15 | 9 | 5 | 16 | 8 | 6 | 13 | 10 | 14 |

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

11 12 4 1 7 2 3 15

9 5 16 8 5 1 7 2 3 15

9 5 16 8 6 13 10 14

11 12 4 1 7 2 3 15

9 5 16 8 6 13 10 14

11 12 4 1 7 2 3 15

9 5 16 8 6 13 10 14

11 12 4 1 7 2 3 15 9 5 16 8 6 13 10 14

44

19

10

6

1

3

2

5

8

15

11

12

4

1
Problem – how do we count inversions between sub problems in $O(n)$ time?

• Solution – Count inversions while merging

Standard merge algorithm – add to inversion count when an element is moved from the upper array to the solution
Use the merge algorithm to count inversions

Indicate the number of inversions for each element detected when merging
Inversions

• Counting inversions between two sorted lists
  – $O(1)$ per element to count inversions

\[
\begin{array}{cccccccc}
  x & x & x & x & x & x & x & x \\
\end{array}
\quad
\begin{array}{cccccccc}
  y & y & y & y & y & y & y & y \\
\end{array}
\quad
\begin{array}{cccccccc}
  z & z & z & z & z & z & z & z \\
\end{array}
\quad
\begin{array}{cccccccc}
  z & z & z & z & z & z & z & z \\
\end{array}
\quad
\begin{array}{cccccccc}
  z & z & z & z & z & z & z & z \\
\end{array}
\quad
\begin{array}{cccccccc}
  z & z & z & z & z & z & z & z \\
\end{array}
\quad
\begin{array}{cccccccc}
  z & z & z & z & z & z & z & z \\
\end{array}
\quad
\begin{array}{cccccccc}
  z & z & z & z & z & z & z & z \\
\end{array}
\cdot
\]

• Algorithm summary
  – Satisfies the “Standard recurrence”
  – $T(n) = 2 \cdot T(n/2) + cn$
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?
Problem generalization

• *Selection*, given n numbers and an integer k, find the k-th largest
Select(A, k)

Select(A, k) {
    Choose element x from A
    $S_1 = \{ y \in A | y < x \}$
    $S_2 = \{ y \in A | y > x \}$
    $S_3 = \{ y \in A | y = x \}$
    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 - $|S_2| - |S_3|$)
}
Randomized Selection

• Choose the element at random
• Analysis can show that the algorithm has expected run time $O(n)$
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
BFPRT Algorithm

• A very clever choose algorithm . . .

Split into $n/5$ sets of size 5
M be the set of medians of these sets
Let $x$ be the median of M
BFPRT runtime

$|S_1| < \frac{3n}{4}, |S_2| < \frac{3n}{4}$

Split into $\frac{n}{5}$ sets of size 5
M be the set of medians of these sets
x be the median of M
Construct $S_1$ and $S_2$
Recursive call in $S_1$ or $S_2$
BFPRT Recurrence

• $T(n) \leq T(3n/4) + T(n/5) + cn$

Prove that $T(n) \leq 20 \cdot c \cdot n$