CSE 417: Algorithms and Computational Complexity

Winter 2002
Instructor: W. L. Ruzzo
Lectures 9-12
Divide and Conquer Algorithms

The Divide and Conquer Paradigm

- Outline:
  - General Idea
  - Review of Merge Sort
  - Why does it work?
    - Importance of balance
    - Importance of super-linear growth
  - Two interesting applications
    - Polynomial Multiplication
    - Matrix Multiplication

Algorithm Design Techniques

- Divide & Conquer
  - Reduce problem to one or more sub-problems of the same type
  - Typically, each sub-problem is at most a constant fraction of the size of the original problem
    - e.g. Mergesort, Binary Search, Strassen's Algorithm, Quicksort (kind of)

Mergesort (review)

Mergesort: (recursively) sort 2 half-lists, then merge results.

- \( T(n) = 2T(n/2) + cn \), \( n \geq 2 \)
- \( T(1) = 0 \)
- Solution: \( \Theta(n \log n) \)

Why Balanced Subdivision?

- Alternative "divide & conquer" algorithm:
  - Sort n-1
  - Sort last 1
  - Merge them
- \( T(n) = T(n-1) + T(1) + 3n \) for \( n \geq 2 \)
- \( T(1) = 0 \)
- Solution: \( 3n + 3(n-1) + 3(n-2) \ldots = \Theta(n^2) \)

Another D&C Approach

- Suppose we've already invented DumbSort, taking time \( n^2 \)
- Try Just One Level of divide & conquer:
  - DumbSort(first \( n/2 \) elements)
  - DumbSort(last \( n/2 \) elements)
  - Merge results
- Time: \( (n/2)^2 + (n/2)^2 + n = n^2/2 + n \)
  - Almost twice as fast!
Another D&C Approach, cont.

- **Moral 1:**
  Two problems of half size are better than one full-size problem, even given the $O(n)$ overhead of recombining, since the base algorithm has super-linear complexity.

- **Moral 2:**
  If a little’s good, then more’s better—two levels of D&C would be almost 4 times faster, 3 levels almost 8, etc., even though overhead is growing. Best is usually full recursion down to some small constant size (balancing “work” vs “overhead”).

- **Moral 3:**
  Unbalanced division less good:
  \[ (0.1n)^2 + (0.9n)^2 + n = 0.82n^2/2 + n \]
  The 18% savings compounds significantly if you carry recursion to more levels, actually giving $O(n \log n)$, but with a bigger constant. So worth doing if you can’t get 50-50 split, but balanced is better if you can.

  This is intuitively why Quicksort with random splitter is good—badly unbalanced splits are rare, and not instantly fatal.

Another D&C Example: Multiplying Faster

On the first HW you analyzed our usual algorithm for multiplying numbers

- $\Theta(n^2)$ time

We can do better!

- We’ll describe the basic ideas by multiplying polynomials rather than integers
- Advantage is we don’t get confused by worrying about carries at first

Notes on Polynomials

These are just formal sequences of coefficients so when we show something multiplied by $x^k$ it just means shifted $k$ places to the left—basically no work

**Usual Polynomial Multiplication:**

<table>
<thead>
<tr>
<th>$3x^2 + 2x + 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^2 - 3x + 1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$3x^2 + 2x + 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$9x^3 - 6x^2 - 6x$</td>
</tr>
<tr>
<td>$3x^4 + 2x^4 + 2x^2$</td>
</tr>
<tr>
<td>$3x^4 - 7x^3 - x^2 - 4x + 2$</td>
</tr>
</tbody>
</table>

Polynomial Multiplication

Given:

- Degree $m-1$ polynomials $P$ and $Q$
  \[ P = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-2} x^{m-2} + a_{m-1} x^{m-1} \]
  \[ Q = b_0 + b_1 x + b_2 x^2 + \ldots + b_{m-2} x^{m-2} + b_{m-1} x^{m-1} \]

Compute:

- Degree $2m-2$ Polynomial $PQ$
  \[ PQ = a_0 b_0 + (a_0 b_1 + a_1 b_0) x + (a_0 b_2 + a_1 b_1 + a_2 b_0) x^2 + \ldots + (a_0 b_{m-2} + a_{m-2} b_0 + a_{m-1} b_{m-2}) x^{2m-3} + a_{m-1} b_{m-1} x^{2m-2} \]

Obvious Algorithm:

- Compute all $a_i b_j$ and collect terms
- $\Theta(n^2)$ time

Naive Divide and Conquer

Assume $m=2k$

- $P = (a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-2} x^{m-2} + a_{m-1} x^{m-1}) x^k$
  \[ = P_0 + P_1 x^k \]
- $Q = Q_0 + Q_1 x^k$

$PQ = (P_0 \cdot P_1 x^k) (Q_0 \cdot Q_1 x^k)$

- $4$ sub-problems of size $k=m/2$ plus linear combining
  \[ T(m) = 4T(m/2) + \Theta(m) \]
- Solution $T(m) = O(m^2)$
Karatsuba’s Algorithm

A better way to compute the terms

1. Compute
   - $P \cdot Q$
   - $P \cdot \overline{Q}$
   - $(P + P)(Q + Q)$ which is $P \cdot Q + P \cdot \overline{Q} + P \cdot \overline{Q} + P \cdot Q$
2. Then
   - $P \cdot \overline{Q} + P \cdot \overline{Q} = (P + P)(Q + Q) - P \cdot Q - P \cdot \overline{Q}$
3. 3 sub-problems of size $m/2$ plus $O(m)$ work
4. $T(m) = 3 T(m/2) + cm$
5. $T(m) = O(m^{\alpha})$ where $\alpha = \log_2 3 = 1.59...$

Karatsuba: Details

PolyMul(P, Q):

// P, Q are length m = 2k vectors, with P[i], Q[i] being
// the coefficient of x^i in polynomials P, Q respectively.
Let Pzero be elements 0..k-1 of P; Pone be elements k..m-1
Qzero, Qone : similar

Prod1 = PolyMul(Pzero, Qzero); // result is a (2k-1)-vector
Prod2 = PolyMul(Pone, Qone);   // ditto
Pzo = Pzero + Pone;                   // add corresponding elements
Qzo = Qzero + Qone;                  // ditto
Prod3 = polyMul(Pzo, Qzo);        // another (2k-1)-vector
Mid = Prod3
- Prod1 - Prod2;     // subtract corr. elements
R = Prod1 + Shift(Mid, m/2) +Shift (Prod2,m) // a (2m-1)-vector
Return( R);

Solve: $T(n) = 2 T(n/2) + cn$

<table>
<thead>
<tr>
<th>Level</th>
<th>Num</th>
<th>Size</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>n</td>
<td>cn</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>n/2</td>
<td>2 c n/2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>n/4</td>
<td>4 c n/4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>2^i</td>
<td>n/2^i</td>
<td>2^i c n/2^i</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>k-1</td>
<td>2^k-1</td>
<td>n/2^k-1</td>
<td>2^k-1 c n/2^k-1</td>
</tr>
<tr>
<td>k</td>
<td>2^k</td>
<td>n/2^k = 1</td>
<td>2^k T(1)</td>
</tr>
</tbody>
</table>

Solve: $T(n) = 4 T(n/2) + cn$

<table>
<thead>
<tr>
<th>Level</th>
<th>Num</th>
<th>Size</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>n</td>
<td>cn</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>n/2</td>
<td>4 c n/2</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>n/4</td>
<td>16 c n/4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>4^i</td>
<td>n/2^i</td>
<td>4^i c n/2^i</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>k-1</td>
<td>4^k-1</td>
<td>n/2^k-1</td>
<td>4^k-1 c n/2^k-1</td>
</tr>
<tr>
<td>k</td>
<td>4^k</td>
<td>n/2^k = 1</td>
<td>4^k T(1)</td>
</tr>
</tbody>
</table>

Solve: $T(1) = c$

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

$T(n) = \sum_{i=0}^{k} 3^i cn / 2^i$

$\sum_{i=0}^{k} x^i = \frac{x^{k+1} - 1}{x - 1}$

$x \neq 1$
Solve: \( T(1) = c \)
\( T(n) = 3 \ T(n/2) + cn \) (cont.)

\[
= 2cn \left( \frac{3}{2} \right)^{k+1} - 1
\]
\[< 2cn (\frac{3}{2})^{k+1}\]
\[= 3cn (\frac{3}{2})^k\]
\[= 3cn \frac{3^k}{2^k}\]

Master Divide and Conquer Recurrence

- If \( T(n) = aT(n/b) + cn^k \) for \( n > b \) then
  - if \( a > b^k \) then \( T(n) = \Theta(n^{log_b a}) \)
  - if \( a < b^k \) then \( T(n) = \Theta(n^k) \)
  - if \( a = b^k \) then \( T(n) = \Theta(n^k \log n) \)
- Works even if it is \( \lceil n/b \rceil \) instead of \( n/b \).

Multiplication – The Bottom Line

- Polynomials
  - Naïve: \( \Theta(n^2) \)
  - Karatsuba: \( \Theta(n^{1.59...}) \)
  - Best known: \( \Theta(n \log n) \)
    - “Fast Fourier Transform”
- Integers
  - Similar, but some ugly details re: carries, etc.
  - gives \( \Theta(n \log n \log \log n) \), but mostly unused in practice

Hints towards FFT: I. Interpolation

- Given set of values at 5 points

Hints towards FFT: I. Interpolation

- Given set of values at 5 points
- Find unique degree 4 polynomial going through these points
Hints towards FFT:  
II. Evaluation & Interpolation

\[ P(y_0), Q(y_0) \]
\[ P(y_1), Q(y_1) \]
\[ \ldots \]
\[ P(y_{n-1}), Q(y_{n-1}) \]

Point-wise multiplication of numbers \( O(n) \)

\[ \sum_{k=0}^{n-1} a_k b_k \]

\[ n^3 \text{ multiplications, } n^3 - n^2 \text{ additions} \]

Multiplying Matrices

\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  a_{31} & a_{32} & a_{33} & a_{34} \\
  a_{41} & a_{42} & a_{43} & a_{44}
\end{bmatrix}
\begin{bmatrix}
  b_{11} & b_{12} & b_{13} & b_{14} \\
  b_{21} & b_{22} & b_{23} & b_{24} \\
  b_{31} & b_{32} & b_{33} & b_{34} \\
  b_{41} & b_{42} & b_{43} & b_{44}
\end{bmatrix}
\]

\[
\begin{bmatrix}
  c_{11} & c_{12} & c_{13} & c_{14} \\
  c_{21} & c_{22} & c_{23} & c_{24} \\
  c_{31} & c_{32} & c_{33} & c_{34} \\
  c_{41} & c_{42} & c_{43} & c_{44}
\end{bmatrix}
\]

Interpolation from \( y_{0}, \ldots, y_{n-1} \), evaluation

III. Evaluation at Special Points

Hints towards FFT:

- Evaluation of polynomial at 1 point takes \( O(m) \).
- So \( m \) points (naively) takes \( O(m^2) \)—no savings
- Key trick: use carefully chosen points where there's some sharing of work for several points, namely various powers of \( \omega = e^{2\pi i / m}, i = \sqrt{-1} \)
- Plus more Divide & Conquer.
- Result: both eval and interpolation in \( O(n \log n) \)
Multiplying Matrices

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}
= 
\begin{pmatrix}
A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\
A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22}
\end{pmatrix}
\]

\[T(n)=8T(n/2)+4(n/2)^2=8T(n/2)+n^2\]

\[8>2^2\] so \(T(n)\) is \(\Theta(n^{\log_2 8}) = \Theta(n^3)\)

Strassen’s algorithm

\[
\begin{align*}
\text{Strassen’s algorithm} \\
\text{Multiply 2x2 matrices using 7 instead of 8 multiplications (and lots more than 4 additions)}
\end{align*}
\]

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

\[7>2^2\] so \(T(n)\) is \(\Theta(n^{\log_2 7})\) which is \(O(n^{2.81})\)

Fastest algorithms theoretically use \(O(n^{2.376})\) time

not practical but Strassen’s is practical provided calculations are exact and we stop recursion when matrix has size about 100 (maybe 10)

Another D&C Example:

Fast exponentiation

\[
\text{Power}(a,n)
\]

\[
\text{Input: integer } n \text{ and number } a
\]

\[
\text{Output: } a^n
\]

\[
\text{Obvious algorithm}
\]

\[n-1\] multiplications

\[
\text{Observation:}
\]

if \(n\) is even, then \(n=2m\), then \(a^n=a^m \cdot a^m\)

Divide & Conquer Algorithm

\[
\text{Power}(a,n)
\]

\[
\text{if } n=0 \text{ then return(1)}
\]

\[
\text{else}
\]

\[
\text{x } \leftarrow \text{Power}(a, \lfloor n/2 \rfloor)
\]

\[
\text{if } n \text{ is even then return(x*x)}
\]

\[
\text{else return(a*x*x)}
\]

Analysis

\[
\text{Worst-case recurrence}
\]

\[T(n)=T(\lfloor n/2 \rfloor)+2\]

\[
\text{By master theorem}
\]

\[T(n)=O(\log n)\]

\[
\text{More precise analysis:}
\]

\[T(n)= \lfloor \log_2 n \rfloor + \# \text{ of 1’s in } n \text{’s binary representation} \]
A Practical Application- RSA

Instead of $a^n$ want $a^n \mod N$

1. $a^{i+j} \mod N = ((a^i \mod N) \cdot (a^j \mod N)) \mod N$
2. same algorithm applies with each $x \cdot y$ replaced by $((x \mod N) \cdot (y \mod N)) \mod N$

In RSA cryptosystem (widely used for security)

1. need $a^n \mod N$, $a$, $n$, $N$ each typically have 1024 bits
2. power: at most 2048 multiplies of 1024 bit numbers relatively easy for modern machines
3. Naive algorithm: $2^{2048}$ multiples

Another Example: Binary search for roots (bisection method)

Given:
- continuous function $f$ and two points $a < b$ with $f(a) < 0$ and $f(b) > 0$

Find:
- approximation to $c$ s.t. $f(c) = 0$ and $a < c < b$

Divide and Conquer Summary

- Powerful technique, when applicable
- Divide large problem into a few smaller problems of the same type
- Choosing subproblems of roughly equal size is usually critical
- Examples:
  - Merge sort, quicksort (sort of), polynomial multiplication, FFT, Strassen's matrix multiplication algorithm, powering, binary search, root finding by bisection, ...