**CSE 421: Introduction to Algorithms**

**Divide and Conquer**

Winter 2005
Paul Beame

---

**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)

---

**Fast exponentiation**

- **Power(a,n)**
  - Input: integer n and number a
  - Output: a^n

- **Obvious algorithm**
  - n-1 multiplications

- **Observation**
  - if n is even, n=2m, then a^n=a^m*a^m

---

**Divide & Conquer Algorithm**

- **Power(a,n)**
  - if n=0 then return(1)
  - else if n=1 then return(a)
  - else
    - x ← Power(a, n/2)
    - if n is even then
      - return(x•x)
    - else
      - return(a•x•x)

---

**Analysis**

- **Worst-case recurrence**
  - T(n) = T(n/2) + 2 for n≥1
  - T(1) = 0

- **Time**
  - T(n) = T(n/2) + 2 ≤ T(n/4) + 2 + 2 ≤ ...
  - ≤ T(1) + 2 + ... + 2 = 2 log₂ n

- **More precise analysis**
  - T(n) = |log₂ n| + # of 1's in n's binary representation

---

**A Practical Application - RSA**

- Instead of a^n want a^n mod N
  - a^n mod N = ((a^k mod N)•(a^l mod N)) mod N
  - same algorithm applies with each x•y replaced by
    - (x mod N)•(y mod N) mod N
  - In RSA cryptosystem (widely used for security)
    - need a^n mod N where a, n, N each typically have
    - 1024 bits
  - Power: at most 2048 multiplies of 1024 bit numbers
    - relatively easy for modern machines
  - Naive algorithm: 2^1024 multiplies
Binary search for roots (bisection method)

- Given:
  - continuous function $f$ and two points $a \leq b$ with $f(a) \leq 0$ and $f(b) > 0$
- Find:
  - approximation to $c$ s.t. $f(c) = 0$ and $a < c < b$

Bisection method

$\text{Bisection}(a,b, \varepsilon)$

if $(a - b) < \varepsilon$

return(a)

else

$c \leftarrow (a + b)/2$

if $f(c) \leq 0$

return($\text{Bisection}(c,b,\varepsilon)$)

else

return($\text{Bisection}(a,c,\varepsilon)$)

Time Analysis

- At each step we halved the size of the interval
- It started at size $b - a$
- It ended at size $\varepsilon$
- $\# \text{ of calls to } f \text{ is } \log_2( \frac{b - a}{\varepsilon} )$

Euclidean Closest Pair

- Given a set $P$ of $n$ points $p_1, \ldots, p_n$ with real-valued coordinates
- Find the pair of points $p_i, p_j \in P$ such that the Euclidean distance $d(p_i, p_j)$ is minimized
- $\Theta(n^2)$ possible pairs
- In one dimension there is an easy $O(n \log n)$ algorithm
  - Sort the points
  - Compare consecutive elements in the sorted list
- What about points in the plane?

Closest Pair in the Plane

- Sort the points by their $x$ coordinates
- Split the points into two sets of $n/2$ points $L$ and $R$ by $x$ coordinate
- Recursively compute
  - closest pair of points in $L$, $(p_L, q_L)$
  - closest pair of points in $R$, $(p_R, q_R)$
- Let $\delta = \min(d(p_L, q_L), d(p_R, q_R))$ and let $(p, q)$ be the pair of points that has distance $\delta$
- This may not be enough!
  - Closest pair of points may involve one point from $L$ and the other from $R$
A clever geometric idea

Any pair of points $p \in L$ and $q \in R$ with $d(p,q) > \delta$ must lie in band

No two points can be in the same green box

Only need to check pairs of points up to 2 rows above and below
At most 15 other points!

Closest Pair Recombining

- Sort points by y coordinate ahead of time
- On recombination only compare each point in $L \cup R$ to the 12 points above it in the y sorted order
- If any of those distances is better than $\delta$ replace $(p,q)$ by the best of those pairs
- $O(n \log n)$ for x and y sorting at start
- Two recursive calls on problems on half size
- $O(n)$ recombination
- Total $O(n \log n)$

Sometimes two sub-problems aren’t enough

- More general divide and conquer
  - You’ve broken the problem into a different sub-problems
  - Each has size at most $n/b$
  - The cost of the break-up and recombining the sub-problem solutions is $O(n^k)$

- Recurrence
  - $T(n) \leq aT(n/b) + cn^k$

Master Divide and Conquer Recurrence

- If $T(n) \leq aT(n/b) + cn^k$ for $n > b$ then
  - if $a < b^k$ then $T(n)$ is $\Theta(n \log a)$
  - if $a > b^k$ then $T(n)$ is $\Theta(n^k)$
  - if $a = b^k$ then $T(n)$ is $\Theta(n^k \log n)$

  Works even if it is $\lceil n/b \rceil$ instead of $n/b$.

Proving Master recurrence

Problem size

\[
\begin{align*}
T(n) &= aT(n/b) + cn^k \\
# \text{ probs} \\
\frac{n}{b} & \quad a \\
\frac{n}{b^2} & \quad a^2 \\
\frac{n}{b^3} & \quad a^3 \\
1 & \quad a^d \\
T(1) & = c
\end{align*}
\]

Proving Master recurrence

Problem size

\[
\begin{align*}
T(n) &= aT(n/b) + cn^k \\
# \text{ probs} \\
\frac{n}{b} & \quad a \\
\frac{n}{b^2} & \quad a^2 \\
\frac{n}{b^3} & \quad a^3 \\
1 & \quad a^d \\
T(1) & = c
\end{align*}
\]
Proving Master recurrence

Problem size

\[ T(n) = a \cdot T(n/b) + c \cdot n^k \] # probs

\[ \text{cost} = cn^k \]

\[ \text{d} = \log_b n \]

\[ \text{b} \]

\[ a \]

\[ n/b \]

\[ a^2 \]

\[ c \cdot a \cdot n^k/b^k \]

\[ c \cdot a^2 \cdot n^k/b^{2k} \]

\[ = c \cdot n^k(a/b)^k \]

\[ \Theta(\log n) \]

\[ \Theta(n^k) \]

\[ \Theta(n^k(a/b)^k) \]

\[ T(1) = c \]

Geometric Series

\[ S = t + tr + tr^2 + \ldots + tr^{n-1} \]

\[ rS = tr^2 + tr^3 + \ldots + tr^n \]

\[ (r-1)S = tr^n - t \]

so \[ S = t (r^n - 1)/(r-1) \] if \( r \neq 1 \).

Simple rule

If \( r \neq 1 \) then \( S \) is a constant times largest term in series

Total Cost

- Geometric series
  - ratio \( a/b^k \)
  - \( d+1 = \log_b n + 1 \) terms
  - first term \( cn^k \), last term \( ca^d \)
  - If \( a/b^k = 1 \)
    - all terms are equal \( T(n) = \Theta(n^k \log n) \)
  - If \( a/b^k < 1 \)
    - first term is largest \( T(n) = \Theta((n^k) \log n) \)
  - If \( a/b^k > 1 \)
    - last term is largest \( T(n) = \Theta((a^d) \log n) \)
    - To see this take \( \log_b \) of both sides

Multiplying Matrices

for \( i = 1 \) to \( n \)
  for \( j = 1 \) to \( n \)
    \[ C[i, j] \leftarrow 0 \]
    for \( k = 1 \) to \( n \)
    endfor
  endfor
endfor

Multiplying Matrices

for \( i = 1 \) to \( n \)
  for \( j = 1 \) to \( n \)
    \[ C[i, j] \leftarrow 0 \]
    for \( k = 1 \) to \( n \)
    endfor
  endfor
endfor
Instead of if you analyze our usual grade school advantage is we don’t get confused by worrying

We can do better!

1. We’ll describe the basic ideas by multiplying polynomials rather than integers
2. Advantage is we don’t get confused by worrying about carries at first
Notes on Polynomials

- These are just formal sequences of coefficients.
- When we show something multiplied by $x^n$ it just means shifted $k$ places to the left -- basically no work.

### Usual polynomial multiplication

$$4x^2 + 2x + 2 \quad x^2 - 3x + 1 \quad 4x^2 + 2x + 2 \quad -12x^3 - 6x^2 - 6x \quad 4x^4 + 2x^2 + 2x^2 \quad 4x^4 - 10x^2 + 0x^2 - 4x + 2$$

### Polynomial Multiplication

- Given:
  - Degree n-1 polynomials $P$ and $Q$:
    $$P = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$$
    $$Q = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-2} x^{n-2} + b_{n-1} x^{n-1}$$
  - Compute:
    - Degree 2n-2 Polynomial $PQ$:
      $$PQ = (a_0 b_0 + (a_0 b_1 + a_1 b_0)) x + (a_0 b_2 + a_1 b_1 + a_2 b_0) x^2 + \ldots + (a_0 b_n + a_n b_0) x^{2n-2} + a_n b_n x^{2n-1}$$
  - Obvious Algorithm:
    - Compute all $a_i b_j$ and collect terms
    - $O(n^2)$ time

### Naive Divide and Conquer

- Assume $n=2k$.
  - $P = (a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}) + (a_k + a_{k+1} x + a_{k+2} x^{k+2} + \ldots + a_{2k-1} x^{2k-1}) = P_0 + P_1 x^k$ where $P_0$ and $P_1$ are degree $k$ polynomials.
  - Similarly $Q = Q_0 + Q_1 x^k$.
  - $PQ = (P_0 + P_1 x^k)(Q_0 + Q_1 x^k) = P_0 Q_0 + (P_1 Q_0 + P_0 Q_1)x^k + P_1 Q_1 x^{2k}$.
  - 4 sub-problems of size $k=n/2$ plus linear combining.

- $T(n) = 4T(n/2) + cn$ Solution $T(n) = \Theta(n^2)$

### Karatsuba's Algorithm

- A better way to compute the terms.
  - Compute:
    - $A \leftarrow P_0 Q_0$
    - $B \leftarrow P_1 Q_1$
    - $C \leftarrow (P_0 + P_1)(Q_0 + Q_1) = P_0 Q_0 + P_1 Q_0 + P_0 Q_1 + P_1 Q_1$
  - Then:
    - $P_0 Q_1 + P_1 Q_0 = C - A - B$
    - So $PQ = A + C - B + Bx^{2k}$
  - 3 sub-problems of size $n/2$ plus $O(n)$ work
  - $T(n) = 3T(n/2) + cn$
  - $T(n) = O(n^\alpha)$ where $\alpha = \log_2 3 = 1.59...$

### Karatsuba: Details

- PolyMul($P, Q$):
  - $P$ and $Q$ are length $n = 2k$ vectors, with $P[0], Q[0]$ being the coefficient of $x^0$ in polynomials $P, Q$ respectively.
  - Let $Pzero$ be elements $0..k-1$ of $P$. $Pone$ be elements $k..n-1$.
  - If $n=1$ then Return($P[0], Q[0]$)
  - $A \leftarrow$ PolyMul($Pzero, Qzero$); /* result is a ($2k$)-vector */
  - $B \leftarrow$ PolyMul($Pone, Qone$); /* ditto */
  - $Psam \leftarrow$ $Pzero + Pone$; /* add corresponding elements */
  - $Qsum \leftarrow$ $Qzero + Qone$; /* ditto */
  - $C \leftarrow$ PolyMul($Psam, Qsum$); /* another ($2k$)-vector */
  - $Mid \leftarrow C - A - B$; /* subtract correspond elements */
  - $R \leftarrow A - Shift(Mid, n/2) + Shift(B, n)$; /* a ($2n$)-vector */
  - Return($R$)

### Multiplication

- Polynomials
  - Naive: $\Theta(n^2)$
  - Karatsuba: $\Theta(n^{1.59})$.
  - Best known: $\Theta(n \log n)$
  - "Fast Fourier Transform" $\rightarrow$ FFT widely used for signal processing.

- Integers
  - Similar, but some ugly details re: carries, etc. gives $\Theta(n \log n \log \log n)$,
  - mostly unused in practice except for symbolic manipulation systems like Maple.
Hints towards FFT: Interpolation

- Given set of values at 5 points

Interpolation

- Given values of degree n-1 polynomial R at n distinct points \( y_1, \ldots, y_n \)
  - \( R(y_1), \ldots, R(y_n) \)
- Compute coefficients \( c_0, \ldots, c_{n-1} \) such that
  - \( R(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{n-1} x^{n-1} \)
- System of linear equations in \( c_0, \ldots, c_{n-1} \)
  - \( c_0 + c_1 y_1 + c_2 y_1^2 + \ldots + c_{n-1} y_1^{n-1} = R(y_1) \) known
  - \( c_0 + c_1 y_2 + c_2 y_2^2 + \ldots + c_{n-1} y_2^{n-1} = R(y_2) \) known
  - \( \ldots \)
  - \( c_0 + c_1 y_n + c_2 y_n^2 + \ldots + c_{n-1} y_n^{n-1} = R(y_n) \) unknown
- Matrix form of the linear system
  - \[
  \begin{pmatrix}
  1 & y_1 & y_1^2 & \ldots & y_1^{n-1} \\
  1 & y_2 & y_2^2 & \ldots & y_2^{n-1} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & y_n & y_n^2 & \ldots & y_n^{n-1}
  \end{pmatrix}
  \begin{pmatrix}
  c_0 \\
  c_1 \\
  \vdots \\
  c_{n-1}
  \end{pmatrix} =
  \begin{pmatrix}
  R(y_1) \\
  R(y_2) \\
  \vdots \\
  R(y_n)
  \end{pmatrix}
  \]
- Fact: Determinant of the matrix is \( \Pi_{i=0}^{n-1} (y_i - y_j) \)
  - which is not 0 since points are distinct
  - System has a unique solution \( c_0, \ldots, c_{n-1} \)

Hints towards FFT: Evaluation & Interpolation

- Ordinary polynomial multiplication \( \Theta(n^2) \)
- Evaluation at \( y_0, \ldots, y_{2n-1} \)
- Interpolation from \( y_0, \ldots, y_{2n-1} \)
- Point-wise multiplication of numbers \( 0 \leq i < n \)
- \[
  \begin{align*}
  P(y_0), Q(y_0) & \rightarrow R(y_0) = \frac{1}{2} \left( P(y_0) + Q(y_0) \right) \\
  P(y_1), Q(y_1) & \rightarrow R(y_1) = \frac{1}{2} \left( P(y_1) + Q(y_1) \right) - \frac{1}{2} \left( P(y_0) + Q(y_0) \right) \\
  \vdots \\
  P(y_{2n-1}), Q(y_{2n-1}) & \rightarrow R(y_{2n-1}) = \frac{1}{2} \left( P(y_{2n-1}) + Q(y_{2n-1}) \right) - \frac{1}{2} \left( P(y_{2n-2}) + Q(y_{2n-2}) \right)
  \end{align*}
  \]

Karatsuba’s algorithm and evaluation and interpolation

- Strassen gave a way of doing 2x2 matrix multiplies with fewer multiplications
- Karatsuba’s algorithm can be thought of as a way of multiplying degree 1 polynomials (which have 2 coefficients) using fewer multiplications
- \[
  P_0 = P_2 = (0, 0) \qquad P_3 = (1, 1) \qquad P_4 = (1, 1)
  \]
- Evaluate at \( 0, 1, -1 \) (Could also use other points)
- \[
  \begin{align*}
  A = P_0 + P_0 & = P_0 \\
  B = P_1 + P_1 & = P_1 \\
  C = P_0 + P_1 & = P_0 \\
  D = P_0 + P_1 & = P_0 \\
  \text{Interpolating, Karatsuba’s Mid} & = (C - D) / 2 \text{ and } B + (C + D) / 2 - A
  \end{align*}
  \]
Evaluation at Special Points

- Evaluation of polynomial at 1 point takes $O(n)$
  - So $2n$ points (naively) takes $O(n^2)$—no savings
- Key trick:
  - Use carefully chosen points where there’s some sharing of work for several points, namely various powers of $\omega = e^{2\pi i/n}$, $i = \sqrt{-1}$

- Plus more Divide & Conquer.

- Result:
  - Both evaluation and interpolation in $O(n \log n)$ time

Hints towards FFT: Evaluation at Special Points

- $\omega^n = 1$
- $\omega^{n/2} = -1$
- $\omega^{n/2k} = -\omega^k$ for all values of $k$
- $\omega^2 = e^{2\pi i/m}$ where $m=n/2$
- $\omega^k = \cos(2k\pi n/n) + i \sin(2k\pi n/n)$ so can compute with powers of $\omega$

The key idea for $n$ even

- $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$
  - $P(\omega) = a_0 + a_1\omega + a_2\omega^2 + \ldots + a_n\omega^n$
- $P(-\omega) = a_0 - a_1\omega + a_2\omega^2 - \ldots + a_n\omega^n$

- $P(\omega^k) = a_0 + a_1\omega^k + a_2\omega^{2k} + \ldots + a_n\omega^{nk}$
- $P(-\omega^k) = a_0 - a_1\omega^k + a_2\omega^{2k} - \ldots + a_n\omega^{nk}$

- $P_{\text{even}}(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^{n/2+1}$
- $P_{\text{odd}}(x) = a_1 + a_2x + a_3x^2 + \ldots + a_nx^{n-1}$

The recursive idea for $n$ a power of 2

- Also $P_{\text{even}}$ and $P_{\text{odd}}$ have degree $n/2$ where
  - $P_{\text{even}}(x) = P_{\text{even}}(\omega x^2) + \omega P_{\text{odd}}(x)$
  - $P_{\text{odd}}(x) = P_{\text{even}}(\omega^2 x) - \omega P_{\text{odd}}(x)$

- Recursitve Algorithm
  - Evaluate $P_{\text{even}}$ at $1, \omega, \omega^2, \ldots, \omega^{n/2-1}$
  - Evaluate $P_{\text{odd}}$ at $1, \omega^2, \omega^4, \ldots, \omega^{n-1}$
  - Combine to compute $P$ at $1, \omega, \omega^2, \ldots, \omega^{n-1}$
  - (i.e., at $\omega^0, \omega^{n/2}, \ldots, \omega^{2n-1}$)

Analysis and more

- Run-time:
  - $T(n) = 2T(n/2) + cn$ so $T(n) = O(n \log n)$

- So much for evaluation ... what about interpolation?
  - Given
    - $r_0 = R(1), r_1 = R(\omega), r_2 = R(\omega^2), \ldots, r_{n-1} = R(\omega^{n-1})$
  - Compute
    - $c_0, c_1, \ldots, c_{n-1}$ s.t. $R(x) = c_0 + c_1x + \ldots + c_{n-1}x^{n-1}$

- Interpolation = Evaluation: strange but true

- Weird fact:
  - If we define a new polynomial $S(x) = r_0 + r_1x + r_2x^2 + \ldots + r_{n-1}x^{n-1}$ where $r_0, r_1, \ldots, r_{n-1}$ are the evaluations of $R$ at $1, \omega, \ldots, \omega^{n-1}$
  - Then $c_k = S(\omega^k)/n$ for $k=0, \ldots, n-1$

- So...
  - Evaluate $S$ at $1, \omega, \omega^2, \ldots, \omega^{n-1}$ then divide each answer by $n$ to get $c_0, c_1, \ldots, c_{n-1}$

- $\omega^1$ behaves just like $\omega$ did so thesame $O(n \log n)$ evaluation algorithm applies!
Divide and Conquer Summary

- Powerful technique, when applicable
- Divide large problem into a few smaller problems of the same type
- Choosing sub-problems 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, ...

Why this is called the discrete Fourier transform

- Real Fourier series
  - Given a real valued function \( f \) defined on \([0,2\pi]\) the Fourier series for \( f \) is given by
    \[
    f(x) = a_0 + a_1 \cos(x) + a_2 \cos(2x) + ... + a_m \cos(mx) + ...
    \]
    where
    \[
    a_m = \frac{1}{2\pi} \int_0^{2\pi} f(x) \cos(mx) \, dx
    \]
  - is the component of \( f \) of frequency \( m \)
  - In signal processing and data compression one ignores all but the components with large \( a_m \) and there aren’t many since

- Complex Fourier series
  - Given a function \( f \) defined on \([0,2\pi]\) the complex Fourier series for \( f \) is given by
    \[
    f(z) = b_0 + b_1 e^{iz} + b_2 e^{2iz} + ... + b_m e^{miz} + ...
    \]
    where
    \[
    b_m = \frac{1}{2\pi} \int_0^{2\pi} f(z) e^{-imz} \, dz
    \]
  - is the component of \( f \) of frequency \( m \)
  - If we discretize this integral using values at \( \frac{2\pi}{n} \) equally spaced points between 0 and \( 2\pi \) we get
    \[
    b_m = \frac{1}{n} \sum_{k=0}^{n-1} f\left(2\pi \frac{k}{n}\right) e^{-im\frac{2\pi k}{n}} = \frac{1}{n} \sum_{k=0}^{n-1} f_k e^{-im\frac{2\pi k}{n}} \quad \text{where } f_k = f(2\pi k/n)
    \]
    just like interpolation!