CSE 421: Introduction to Algorithms

Divide and Conquer

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 \( a^n = a^m \cdot a^m \)

Divide & Conquer Algorithm

- **Power(a, n)**
  - if \( n = 0 \) then return(1)
  - else if \( n = 1 \) then return(a)
  - else
    - \( x \leftarrow \text{Power}(a, \lfloor n/2 \rfloor) \)
    - if \( n \) is even then
      - return(\( x \cdot x \))
    - else
      - return(\( a \cdot x \cdot x \))

Analysis

- Worst-case recurrence
  - \( T(n) = T(\lfloor n/2 \rfloor) + 2 \) for \( n \geq 1 \)
  - \( T(1) = 0 \)
- Time
  - \( T(n) = T(\lfloor n/2 \rfloor) + 2 \leq T(\lfloor n/4 \rfloor) + 2 + 2 \leq \ldots \leq T(1) + 2 + \ldots + 2 = 2 \log_2 n \)
  - More precise analysis:
    - \( T(n) = \lceil \log_2 n \rceil + \# \text{ of } 1 \text{'s in } n \text{'s binary representation} \)

A Practical Application - RSA

- Instead of \( a^n \) want \( a^n \mod N \)
  - \( a^h \mod N = ((a^i \mod N) \cdot (a \mod N)) \mod N \)
  - 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)
  - 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 \( a < c < b \)

### Bisecton method

\[ \text{Bisection}(a, b, \varepsilon) \]

- if \( (a - b) < \varepsilon \) then
  - return(\( a \))
- else
  - \( c \leftarrow (a + b)/2 \)
  - if \( f(c) \leq 0 \) then
    - return(Bisection(\( c \), \( b \), \( \varepsilon \)))
  - else
    - return(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 \( \varepsilon \).
- \# of calls to \( f \) 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: Divide and Conquer

- 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 \( L \), \( (p_i, q_i) \)
  - closest pair of points in \( R \), \( (p_i, q_i) \)
- Let \( \delta = \min(d(p_1, q_1), d(p_2, q_2)) \) 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 compares 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
  - $T(n) \leq a \cdot T(n/b) + c \cdot n^k$

Master Divide and Conquer Recurrence

- If $T(n) \leq a \cdot T(n/b) + c \cdot n^k$ for $n > b$ then
  - if $a > b^k$ then $T(n)$ is $\Theta(n^{\log_b a})$
  - 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 $\left\lfloor n/b \right\rfloor$ instead of $n/b$. 
Problem size

$T(n) = a T(n/b) + c n^k$ # probs

- $n$
- $n/b$
- $n/b^2$
- $b$
- $1$

Proving Master recurrence

$T(1) = c$

Problem size

$T(n) = a T(n/b) + c n^k$ # probs

- $n$
- $n/b$
- $n/b^2$
- $b$
- $1$

Proving Master recurrence

$T(1) = c$

Problem size

$T(n) = a T(n/b) + c n^k$ # probs

- $n$
- $n/b$
- $n/b^2$
- $b$
- $1$

Proving Master recurrence

$T(1) = c$

Geometric Series

- $S = t + tr + tr^2 + \ldots + tr^{n-1}$
- $rS = tr + tr^2 + \ldots + tr^{n-1} + tr^n$
- $(r-1)S = tr^n - t$
- 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 $c a^d$
- If $a/b^k = 1$
  - all terms are equal $T(n)$ is $\Theta(n^k \log n)$
- If $a/b^k < 1$
  - first term is largest $T(n)$ is $\Theta(n^k)$
- If $a/b^k > 1$
  - last term is largest $T(n)$ is $\Theta(a^d) = \Theta(a^{\log_b n}) = \Theta(n^{\log_a a})$
  (To see this take $\log_{b}$ of both sides)

Multiplying Matrices

- $n^3$ multiplications, $n^3 - n^2$ additions
Simple Divide and Conquer

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

T(n) = 8T(n/2) + 4(n/2)^2 = 8T(n/2) + n^2

\[\Theta(\log^2 n) = \Theta(n^{\log_2 8}) = \Theta(n^3)\]

Strassen’s Divide and Conquer Algorithm

- Strassen’s algorithm
  - Multiply 2x2 matrices using 7 instead of 8 multiplications (and lots more than 4 additions)
  - 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)
The algorithm

\[ P_1 \leftarrow A_{12}(B_{11}+B_{21}) ; \quad P_2 \leftarrow A_{21}(B_{12}+B_{22}) \]
\[ P_3 \leftarrow (A_{11} - A_{12})B_{11} ; \quad P_4 \leftarrow (A_{22} - A_{21})B_{22} \]
\[ P_5 \leftarrow (A_{22} - A_{12})(B_{21} - B_{22}) \]
\[ P_6 \leftarrow (A_{11} - A_{21})(B_{12} - B_{11}) \]
\[ P_7 \leftarrow (A_{21} - A_{12})(B_{11} - B_{22}) \]
\[ C_{11} \leftarrow P_1 + P_3 \quad ; \quad C_{12} \leftarrow P_2 + P_3 + P_6 - P_7 \]
\[ C_{21} \leftarrow P_1 + P_4 + P_5 + P_7 \quad ; \quad C_{22} \leftarrow P_2 + P_4 \]

Another Divide & Conquer Example: Multiplying Faster

- If you analyze our usual grade school algorithm for multiplying numbers
  \[ \Theta(n^2) \] time
- On real machines each “digit” is, e.g., 32 bits long but still get \( \Theta(n^2) \) running time with this algorithm when run on \( n \)-bit multiplication
- 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
  - when we show something multiplied by \( x^k \) it just means shifted \( k \) places to the left – basically no work

Usual polynomial multiplication

\[
\begin{align*}
4x^2 + 2x + 2 & \quad x^2 - 3x + 1 \\
4x^2 + 2x + 2 & -12x^3 + 6x^2 + 6x \\
4x^4 + 2x^3 & -20x^2 + 4x + 2
\end{align*}
\]

Polynomial Multiplication

- Given:
  - Degree \( n-1 \) polynomials \( P \) and \( Q \)
    \[ P = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1} \]
    \[ Q = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-2} x^{n-2} + b_{n-1} x^{n-1} \]
- Compute:
  - Degree \( 2n-2 \) Polynomial \( PQ \)
  - \[ PQ = (a_0b_0 + (a_0b_1 + a_1b_0)) x + (a_0b_2 + a_1b_1 + a_2b_0) x^2 + \ldots + (a_{n-2}b_{n-1} + a_{n-1}b_{n-2}) x^{n-3} + a_{n-1}b_{n-1} x^{n-2} \]
- Obvious Algorithm:
  - Compute all \( a_i b_j \) and collect terms
  - \( \Theta(n^2) \) time

Naive Divide and Conquer

- Assume \( n = 2k \)
  \[ P = (a_0 + a_1 x + a_2 x^2 + \ldots + a_{2k-2} x^{2k-2} + a_{2k-1} x^{2k-1}) + (a_0 + a_1 x + \ldots + a_{2k-2} x^{2k-2} + a_{2k-1} x^{2k}) x^k \]
  where \( P_0 \) and \( P_1 \) are degree \( k-1 \) polynomials
- Similarly \( Q = Q_0 + Q_1 x^k \)
- \[ PQ = (P_0 + P_1 x^k)(Q_0 + Q_1 x^k) \]
  \[ = P_0Q_0 + (P_0Q_1 + P_1Q_0) x^k + P_1Q_1 x^{2k} \]
- 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
    \[ A \leftarrow P_0 Q_0 \]
    \[ B \leftarrow P_1 Q_1 \]
    \[ C \leftarrow (P_0 + P_1)(Q_0 + Q_1) = P_0Q_0 + P_1Q_1 + P_0Q_1 + P_1Q_0 \]
  - Then
    \[ P_0Q_1 + P_1Q_0 = C - A - B \]
    \[ = S_0P_0Q_1 + S_1P_1Q_0 = (C - A - B)x^k \]
  - 3 sub-problems of size \( n/2 \) plus \( O(n) \) work
  - \( T(n) = 3T(n/2) + cn \)
  - \( T(n) = O(n^\log_23) \approx 1.59... \)
Karatsuba:
Details

PolyMul($P, Q$):

$\begin{align*}
// &P, Q\text{ are length } n \rightarrow 2k \text{ vectors, with } P[i], Q[i] \text{ being } \\
// &the coefficient of } x^i \text{ in polynomials } P, Q \text{ respectively.} \\
// &Let } Pzero \text{ be elements } 0..k-1 \text{ of } P; \text{ Pone be elements } k..n-1 \\
// &Qzero, Qone: similar \\
// &If } n = 1 \text{ then Return(} P[0] * Q[0]) \text{ else} \\
A &← PolyMul(Pzero, Qzero); \text{ // result is a } (2k-1)\text{-vector} \\
B &← PolyMul(Pone, Qone); \text{ // ditto} \\
Psum &← Pzero + Pone; \text{ // add corresponding elements} \\
Qsum &← Qzero + Qone; \text{ // ditto} \\
C &← polyMul(Psum, Qsum); \text{ // another } (2k-1)\text{-vector} \\
Mid &← C - A - B; \text{ // subtract correspond elements} \\
R &← A + Shift(Mid, n/2) + Shift(B, n) \text{ // a } (2n-1)\text{-vector} \\
\end{align*}$

Return( $R$ );

Multiplication

- Polynomials
  - Naive: $\Theta(n^2)$
  - Karatsuba: $\Theta(n^{1.59...})$
  - Best known: $\Theta(n \log n)$
- "Fast Fourier Transform"
- FFT widely used for signal processing
- Integers
  - Similar, but some ugly details re: carries, etc.
    - mostly unused in practice except for symbolic manipulation systems like Maple

Hints towards FFT:
Interpolation

- Given set of values at 5 points

Interpolation: n equations in n unknowns

- Matrix form of the linear system
  $\begin{bmatrix}
  1 & y_1 & y_1^2 & \ldots & y_1^{n-1} \\
  1 & y_2 & y_2^2 & \ldots & y_2^{n-1} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & y_n & y_n^2 & \ldots & y_n^{n-1}
  \end{bmatrix} \begin{bmatrix}
  c_0 \\
  c_1 \\
  \vdots \\
  c_{n-1}
  \end{bmatrix} = \begin{bmatrix}
  R(y_1) \\
  R(y_2) \\
  \vdots \\
  R(y_n)
  \end{bmatrix}$
- Fact: Determinant of the matrix is $\prod_{i \neq j} (y_i - y_j)$
  which is not 0 since points are distinct
- System has a unique solution $c_0, \ldots, c_{n-1}$
Hints towards FFT: Evaluation & Interpolation

- 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

The key idea for $n$ even

- $P(\alpha) = a_0 + a_1 \omega + a_2 \omega^2 + a_3 \omega^3 + \cdots + a_n \omega^{n-1}$
  - $a_0 + a_1 \omega^2 + a_2 \omega^4 + \cdots + a_{n/2} \omega^{n/2}$
  - $a_n \omega + a_{n-1} \omega^2 + a_{n-2} \omega^4 + \cdots + a_1 \omega^{n/2}$
  - $P_{\text{even}}(\omega^2) + \omega P_{\text{odd}}(\omega^2)$
- $P(\alpha^m) = a_0 + a_1 \omega^m + a_2 \omega^{2m} + a_3 \omega^{3m} + \cdots + a_n \omega^{nm}$
  - $(a_0 + a_1 \omega^2 + a_2 \omega^4 + \cdots + a_{n/2} \omega^{n/2})$
  - $- (a_n \omega + a_{n-1} \omega^2 + a_{n-2} \omega^4 + \cdots + a_1 \omega^{n/2})$
  - $P_{\text{even}}(\omega^m) - \omega P_{\text{odd}}(\omega^m)$
- $P_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n/2} x^{n/2-1}$
- $P_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$

Fun facts about $\omega = e^{2\pi i/n}$ for even $n$

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

Hints towards FFT: 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

The recursive idea for $n$ a power of 2

- Also
  - $P_{\text{even}}$ and $P_{\text{odd}}$ have degree $n/2$ where
    - $P(\alpha^m) = P_{\text{even}}(\alpha^{2m}) + \omega^m P_{\text{odd}}(\alpha^{2m})$
    - $P(\alpha^m) = P_{\text{even}}(\alpha^{2m}) - \omega^m P_{\text{odd}}(\alpha^{2m})$
- Recursive Algorithm
  - Evaluate $P_{\text{even}}$ at $1, \omega^2, \omega^4, \ldots, \omega^{n/2}$
  - Evaluate $P_{\text{odd}}$ at $1, \omega^2, \omega^4, \ldots, \omega^{n/2-1}$
  - Combine to compute $P$ at $1, \omega^2, \omega^4, \ldots, \omega^{n/2-1}$
  - (i.e. at $\omega^2, \omega^4, \omega^6, \ldots, \omega^{n-1}$)

Karatsuba’s algorithm and evaluation and interpolation

- Strassen gave a way of doing $2 \times 2$ 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
  - $\bar{P} = P_0 + (P_1 \cdot Q_0 - Q_1 \cdot Q_0)$
  - $P_0 \cdot Q_0 + P_1 \cdot Q_1$ + $P_0 \cdot Q_2$
- Evaluate at 0, 1, $-1$ (Could also use other points)
  - $A = P_0(0) Q_0(0)$
  - $B = P_0(1) Q_1(0)$
  - $C = P_1(1) Q_0(1)$
  - $D = P_1(1) Q_1(0)$
- Interpolating, Karatsuba’s Mid=(C-D)/2 and Be(C+D)/2-A

The recursive idea for $n$ a power of 2

- Also
  - $P_{\text{even}}$ and $P_{\text{odd}}$ have degree $n/2$ where
    - $P(\alpha^m) = P_{\text{even}}(\alpha^{2m}) + \omega^m P_{\text{odd}}(\alpha^{2m})$
    - $P(\alpha^m) = P_{\text{even}}(\alpha^{2m}) - \omega^m P_{\text{odd}}(\alpha^{2m})$
- Recursive Algorithm
  - Evaluate $P_{\text{even}}$ at $1, \omega^2, \omega^4, \ldots, \omega^{n/2}$
  - Evaluate $P_{\text{odd}}$ at $1, \omega^2, \omega^4, \ldots, \omega^{n/2-1}$
  - Combine to compute $P$ at $1, \omega^2, \omega^4, \ldots, \omega^{n/2-1}$
  - (i.e. at $\omega^2, \omega^4, \omega^6, \ldots, \omega^{n-1}$)
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
    \[ r_0 = R(1), r_1 = R(\omega), r_2 = R(\omega^2), ..., r_{n-1} = R(\omega^{n-1}) \]
  - Compute
    \[ c_0, c_1, ..., c_{n-1}, \text{s.t.} \quad R(x) = c_0 + c_1 x + ... + c_{n-1} x^{n-1} \]

Interpolation = Evaluation: strange but true

- Weird fact:
  - If we define a new polynomial \( S(x) = r_0 + r_1 x + r_2 x^2 + ... + r_{n-1} x^{n-1} \)
    where \( r_0, r_1, ..., r_{n-1} \)
  - Then \( c_k = S(\omega^k)/n \) for \( k = 0, ..., n-1 \)
- So...
  - evaluate \( S \) at \( 1, \omega, \omega^2, ..., \omega^{n-1} \) then divide each answer by \( n \) to get the \( c_0, ..., c_{n-1} \)
  - \( \omega \) behaves just like \( \omega \) did so the same \( O(n \log n) \)
    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
  \[ b_m = \frac{1}{2\pi} \int_0^{2\pi} f(z) e^{-miz} \, dz \]
  is the component of \( f \) of frequency \( m \)
- If we discretize this integral using values at \( n \)
  equally spaced points between \( 0 \) and \( 2\pi \) we get
  \[ b_m = \frac{1}{n} \sum_{k=0}^{n-1} f(2k\pi/n) e^{-2m\pi i/n} = \frac{1}{n} \sum_{k=0}^{n-1} f(2k\pi/n) \omega^{-mk} \]
  where \( \omega = i(2\pi/n) \)
  just like interpolation!