CSE 421: Introduction to Algorithms

Divide and Conquer

Winter 2003
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 $a^n=a^m \bullet a^m$

Divide & Conquer Algorithm

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

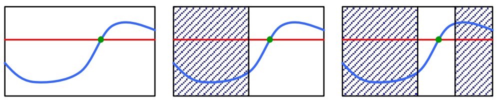
Analysis

- Worst-case recurrence
  - $T(n)=T(\lfloor n/2 \rfloor)+2$ for $n=1$
  - $T(1)=0$
- Time
  - $T(n)=T(\lfloor n/2 \rfloor)+2 \le T(\lfloor n/4 \rfloor)+2+2 \le \dots$
    $\le T(1)+\underbrace{2+\dots+2}_{\log_2 n \text{ copies}} = 2\log_2 n$
- More precise analysis:
  - $T(n)= \lceil \log_2 n \rceil$ + # of **1**'s in **n**'s binary representation

A Practical Application- RSA

- Instead of a^n want a^n **mod N**
  - a^{i+j} mod N = ((a^i mod N)•(a^j 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** multiples of 1024 bit numbers
    - relatively easy for modern machines
  - Naive algorithm: 2^{1024} multiples

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**

Bisection method

```
Bisection(a,b, e)
   if (a-b) < e  then
        return(a)
   else
        c ←(a+b)/2
        if  f(c) ≤ 0 then
             return(Bisection(c,b,e))
        else
             return(Bisection(a,c,e))
```

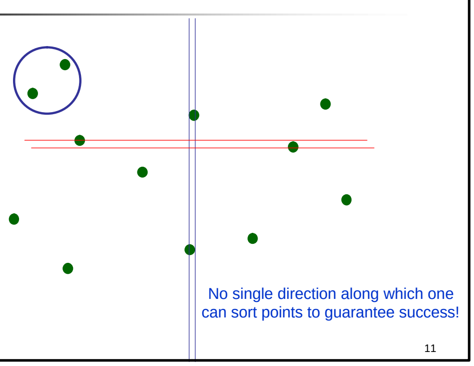

Time Analysis

- At each step we halved the size of the interval
- It started at size **b-a**
- It ended at size e
- # of calls to f is $\log_2$((**b-a**)/e)

Euclidean Closest Pair

- Given a set **P** of n points $\mathbf{p_1},\ldots,\mathbf{p_n}$ with real-valued coordinates
- Find the pair of points $\mathbf{p_i},\mathbf{p_j}$ Î **P** such that the Euclidean distance $\mathbf{d}(\mathbf{p_i},\mathbf{p_j})$ is minimized
- Q(**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?

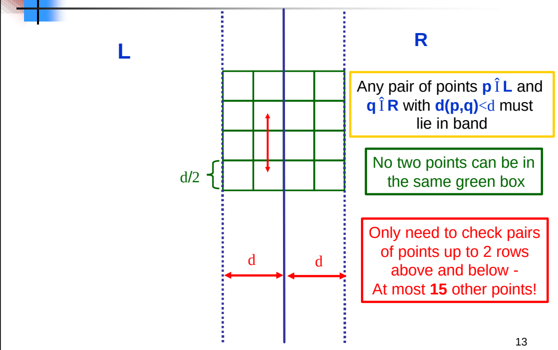
Closest Pair in the Plane



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**, $(\mathbf{p_L},\mathbf{q_L})$
  - closest pair of points in **R**, $(\mathbf{p_R},\mathbf{q_R})$
- Let d=min{$\mathbf{d}(\mathbf{p_L},\mathbf{q_L}),\mathbf{d}(\mathbf{p_R},\mathbf{q_R})$} and let (**p**,**q**) be the pair of points that has distance d
- This may not be enough!
  - Closest pair of points may involve one point from **L** and the other from **R**!

A clever geometric idea



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 d 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)**

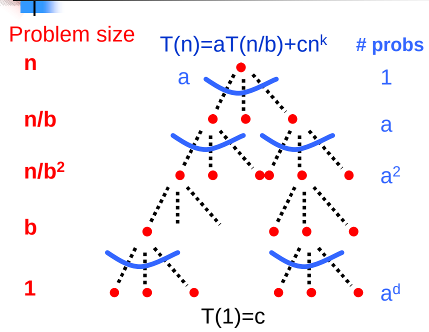
Sometimes two sub-problems aren't enough

- 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
  - $T(n) \le a \cdot T(n/b) + c \cdot n^k$

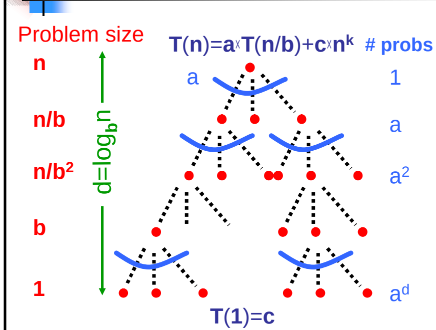
Master Divide and Conquer Recurrence

- If $T(n) \le a \cdot T(n/b) + c \cdot n^k$ 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 $\lceil n/b \rceil$ instead of **n/b**.

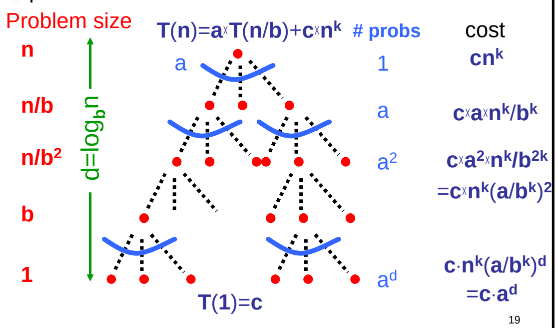
Proving Master recurrence



Proving Master recurrence



Proving Master recurrence



19

Geometric Series

- $S \quad = t + \ tr \ + tr^2 + \ldots + tr^{n-1}$
- $r{\cdot}S \quad = \qquad tr \ + tr^2 + \ldots + tr^{n-1} + tr^n$
- $(r-1)S = tr^n - t$
- 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

20

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 $T(n)$ is $Q(n^k \log n)$
- 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 $\Theta(a^d)=\Theta(a^{\log_b n})=\Theta(n^{\log_b a})$
  (To see this take $\log_b$ of both sides)

21

Multiplying Matrices

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

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

- n^3 multiplications, n^3-n^2 additions

22

Multiplying Matrices

```
for i=1 to n
    for j=1 to n
        C[i,j]←0
        for k=1 to n
            C[i,j]=C[i,j]+A[i,k]·B[k,j]
        endfor
    endfor
endfor
```

23

Multiplying Matrices

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

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

24

Multiplying Matrices

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

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

25

Multiplying Matrices

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \bullet \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \end{bmatrix}$$

26

Simple Divide and Conquer

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$= \begin{bmatrix} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \end{bmatrix}$$

- $T(n)=8T(n/2)+4(n/2)^2=8T(n/2)+n^2$
  - $8>2^2$ so $T(n)$ is $\Theta(n^{\log_b a}) = \Theta(n^{\log_2 8}) = \Theta(n^3)$

27

Strassen's Divide and Conquer Algorithm

- Strassen's algorithm
  - Multiply 2x2 matrices using 7 instead of 8 multiplications (and lots more than 4 additions)
  - $T(n)=7\,T(n/2)+cn^2$
    - $7>2^2$ 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)

28

The algorithm

$P_1 \leftarrow A_{12}(B_{11}+B_{21})$; $P_2 \leftarrow A_{21}(B_{12}+B_{22})$

$P_3 \leftarrow (A_{11} - A_{12})B_{11}$; $P_4 \leftarrow (A_{22} - A_{21})B_{22}$

$P_5 \leftarrow (A_{22} - A_{12})(B_{21} - B_{22})$

$P_6 \leftarrow (A_{11} - A_{21})(B_{12} - B_{11})$

$P_7 \leftarrow (A_{21} - A_{12})(B_{11}+B_{22})$

$C_{11} \leftarrow P_1+P_3$; $C_{12} \leftarrow P_2+P_3+P_6 - P_7$

$C_{21} \leftarrow P_1+P_4+P_5+P_7$; $C_{22} \leftarrow P_2+P_4$

29

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

30

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

$$
\begin{array}{r}
4x^2 + 2x + 2 \\
x^2 - 3x + 1 \\
\hline
4x^2 + 2x + 2 \\
-12x^3 - 6x^2 - 6x \quad \\
4x^4 + 2x^3 + 2x^2 \qquad\qquad \\
\hline
4x^4 - 10x^3 + 0x^2 - 4x + 2
\end{array}
$$

Polynomial Multiplication

- **Given:**
  - Degree $n-1$ polynomials P and Q
    - $P = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1}$
    - $Q = b_0 + b_1 x + b_2 x^2 + \dots + b_{n-2}x^{n-2} + b_{n-1}x^{n-1}$
- **Compute:**
  - Degree $2n-2$ Polynomial $P\,Q$
  - $P\,Q = a_0b_0 + (a_0b_1 + a_1b_0)\,x + (a_0b_2 + a_1b_1 + a_2b_0)\,x^2 + \dots + (a_{n-2}b_{n-1} + a_{n-1}b_{n-2})\,x^{2n-3} + a_{n-1}b_{n-1}\,x^{2n-2}$
- **Obvious Algorithm:**
  - Compute all a_ib_j and collect terms
  - Q (n^2) time

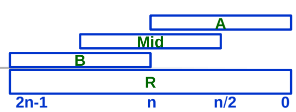
Naive Divide and Conquer

- Assume $n=2k$
  - $P = (a_0 + a_1 x + a_2 x^2 + \dots + a_{k-2} x^{k-2} + a_{k-1} x^{k-1}) + (a_k + a_{k+1} x + \dots + a_{n-2}x^{k-2} + a_{n-1}x^{k-1})\,x^k$
    $= P_0 + P_1 x^k$ where P_0 and P_1 are degree $k-1$ polynomials
  - Similarly $Q = Q_0 + Q_1 x^k$
- $P\,Q = (P_0+P_1x^k)(Q_0+Q_1x^k)$
  $= P_0Q_0 + (P_1Q_0+P_0Q_1)x^k + P_1Q_1x^{2k}$
- 4 sub-problems of size $k=n/2$ plus linear combining
  - $T(n)=4 \cdot T(n/2)+cn$ Solution $T(n) =$ Q(n^2)

Karatsuba's Algorithm

- A better way to compute the terms
  - Compute
    - $A \leftarrow P_0Q_0$
    - $B \leftarrow P_1Q_1$
    - $C \leftarrow (P_0+P_1)(Q_0+Q_1) = P_0Q_0+P_1Q_0+P_0Q_1+P_1Q_1$
  - Then
    - $P_0Q_1+P_1Q_0 = C-A-B$
    - So $PQ=A+(C-A-B)x^k+Bx^{2k}$
  - 3 sub-problems of size $n/2$ plus $O(n)$ work
    - $T(n) = 3\,T(n/2) + cn$
    - $T(n) = O(n^a)$ where $a = \log_2 3 = 1.59\dots$

Karatsuba: Details

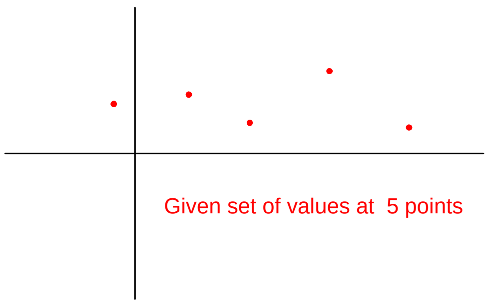



```
PolyMul(P, Q):
  // P, Q are length n =2k vectors, with P[i], Q[i] being
  // the coefficient of xi in polynomials P, Q respectively.
  // Let Pzero be elements 0..k-1 of P; Pone be elements k..n-1
  // Qzero, Qone : similar
  If n=1 then Return(P[0]*Q[0]) else
    A ← PolyMul(Pzero, Qzero);    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


Multiplication

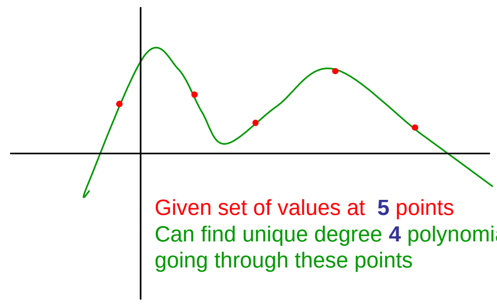
- Polynomials
  - Naïve: Q(n^2)
  - Karatsuba: Q$(n^{1.59\dots})$
  - Best known: Q$(n \log n)$
    - "Fast Fourier Transform"
    - FFT widely used for signal processing
- Integers
  - Similar, but some ugly details re: carries, etc. gives Q$(n \log n \log\log n)$,
    - mostly unused in practice except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

Hints towards FFT: Interpolation



37

Hints towards FFT: Interpolation



38

Interpolation

- Given values of degree **n-1** polynomial **R** at **n** distinct points $y_1,\dots,y_n$
  - $R(y_1),\dots,R(y_n)$
- Compute coefficients $c_0,\dots,c_{n-1}$ such that
  - $R(x)=c_0+c_1x+c_2x^2+\dots+c_{n-1}x^{n-1}$
- System of linear equations in $c_0,\dots,c_{n-1}$

$$c_0+c_1y_1+c_2y_1^2+\dots+c_{n-1}y_1^{n-1}=R(y_1)$$
$$c_0+c_1y_2+c_2y_2^2+\dots+c_{n-1}y_2^{n-1}=R(y_2)$$
$$\dots$$
$$c_0+c_1y_n+c_2y_n^2+\dots+c_{n-1}y_n^{n-1}=R(y_n)$$

known

unknown

39

Interpolation: n equations in n unknowns

- Matrix form of the linear system

$$\begin{pmatrix} 1 & y_1 & y_1^2 & \dots & y_1^{n-1} \\ 1 & y_2 & y_2^2 & \dots & y_2^{n-1} \\ & \dots & & & \\ & \dots & & & \\ 1 & y_n & y_n^2 & \dots & y_n^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \cdot \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} R(y_1) \\ R(y_2) \\ \cdot \\ \cdot \\ R(y_n) \end{pmatrix}$$

- Fact: Determinant of the matrix is $\prod_{i<j}(y_i-y_j)$ which is not **0** since points are distinct
  - System has a unique solution $c_0,\dots,c_{n-1}$

40

Hints towards FFT: Evaluation & Interpolation

P: $a_0,a_1,\dots,a_{n-1}$
Q: $b_0,b_1,\dots,b_{n-1}$

ordinary polynomial multiplication $O(n^2)$

$c_k \leftarrow \sum_{i+j=k} a_i b_j$

R: $c_0,c_1,\dots,c_{2n-1}$

evaluation at $y_0,\dots,y_{2n-1}$ O(?)

interpolation from $y_0,\dots,y_{2n-1}$ O(?)

$P(y_0),Q(y_0)$
$P(y_1),Q(y_1)$
...
$P(y_{2n-1}),Q(y_{2n-1})$

point-wise multiplication of numbers O(n)

$R(y_0)\leftarrow P(y_0)\cdot Q(y_0)$
$R(y_1)\leftarrow P(y_1)\cdot Q(y_1)$
...
$R(y_{2n-1})\leftarrow P(y_{2n-1})\cdot Q(y_{2n-1})$

41

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
  - $PQ=(P_0+P_1z)(Q_0+Q_1z)$
    $=P_0Q_0+(P_1Q_0+P_0Q_1)z+P_1Q_1z^2$
  - Evaluate at **0,1,-1** (Could also use other points)
    - $A=P(0)Q(0)=P_0Q_0$
    - $C=P(1)Q(1)=(P_0+P_1)(Q_0+Q_1)$
    - $D=P(-1)Q(-1)=(P_0-P_1)(Q_0-Q_1)$
  - Interpolating, Karatsuba's **Mid=(C-D)/2** and **B=(C+D)/2-A**

42

Hints towards FFT: Evaluation at Special Points

- Evaluation of polynomial at **1** point takes **O(n)**
  - So **2n** points (naively) takes **O(n²)**—no savings
- Key trick:
  - use carefully chosen points where there's some sharing of work for several points, namely various powers of $w = e^{2pi/n}$, $i = \sqrt{-1}$
- Plus more Divide & Conquer.
- Result:
  - both evaluation and interpolation in **O(n log n)** time

43

Fun facts about $w=e^{2pi/n}$ for even n

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

44

The key idea for n even

- $P(w) = a_0+a_1w+a_2w^2+a_3w^3+a_4w^4+\ldots+a_{n-1}w^{n-1}$
  $= a_0 +a_2w^2 +a_4w^4 +\ldots+ a_{n-2}w^{n-2}$
  $+ a_1w+a_3w^3 +a_5w^5 +\ldots+a_{n-1}w^{n-1}$
  $= P_{even}(w^2) + w\,P_{odd}(w^2)$
- $P(-w)=a_0 -a_1w+a_2w^2 -a_3w^3+a_4w^4-\ldots -a_{n-1}w^{n-1}$
  $= a_0 +a_2w^2 +a_4w^4 +\ldots+ a_{n-2}w^{n-2}$
  $- (a_1w+a_3w^3 +a_5w^5 +\ldots+a_{n-1}w^{n-1})$
  $= P_{even}(w^2) - w\,P_{odd}(w^2)$

where $P_{even}(x) = a_0 +a_2x +a_4x^2 +\ldots+ a_{n-2}x^{n/2-1}$

and $P_{odd}(x) = a_1+a_3x +a_5x^2 +\ldots+a_{n-1}x^{n/2-1}$

45

The recursive idea for n a power of 2

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

w^2 is $e^{2pi/m}$ where $m=n/2$ so problems are of same type but smaller size

46

Analysis and more

- Run-time
  - $T(n)=2\times T(n/2)+cn$ so $T(n)=O(n \log n)$
- So much for evaluation ... what about interpolation?
  - Given
    - $r_0=R(1)$, $r_1=R(w)$, $r_2=R(w^2)$,..., $r_{n-1}=R(w^{n-1})$
  - Compute
    - $c_0, c_1,\ldots,c_{n-1}$ s.t. $R(x)=c_0+c_1x+\ldots+c_{n-1}x^{n-1}$

47

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 $1, w, \ldots, w^{n-1}$
  - Then $c_k=S(w^{-k})/n$ for $k=0,\ldots,n-1$
- So...
  - evaluate S at $1,w^{-1},w^{-2},\ldots,w^{-(n-1)}$ then divide each answer by n to get the $c_0,\ldots,c_{n-1}$
  - w^{-1} behaves just like w did so the same **O(n log n)** evaluation algorithm applies !

48

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, …

49

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 $f(x)=a_0+a_1 \cos(x) + a_2 \cos(2x) +\ldots+ a_m \cos(mx) +\ldots$ where

$$\boxed{a_m=\frac{1}{2\pi}\int_0^{2\pi} f(x)\cos(mx)\,dx}$$

  - 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

50

Why this is called the discrete Fourier transform

- Complex Fourier series
  - Given a function f defined on $[0,2\pi]$ the complex Fourier series for f is given by $f(z)=b_0+b_1 e^{iz} + b_2 e^{2iz} +\ldots+ b_m e^{miz} +\ldots$ where

$$\boxed{b_m= \frac{1}{2\pi}\int_0^{2\pi} f(z)\, e^{-miz}\, dz}$$

    is the component of f of frequency m
  - If we **discretize** this integral using values at n equally spaced points between 0 and 2π we get

2p/n apart

$$\boxed{\bar{b}_m = \frac{1}{n}\sum_{k=0}^{n-1} f_k\, e^{-2kmi\pi/n} = \frac{1}{n}\sum_{k=0}^{n-1} f_k\, \omega^{-km} \text{ where } f_k=f(2k\pi/n)}$$

just like interpolation!

51