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

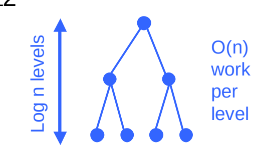
Algorithm Design Techniques

- 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").

7

Another D&C Approach, cont.

- Moral 3: unbalanced division less good:
  - $(.1n)^2 + (.9n)^2 + n = .82n^2/2 + n$
    - The 18% savings compounds significantly if you carry recursion to more levels, actually giving O(nlogn), 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.
  - $(1)^2 + (n-1)^2 + n = n^2 - 2n + 2 + n$
    - Little improvement here.

8

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

9

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:

$$
\begin{array}{r}
3x^2 + 2x + 2 \\
x^2 - 3x + 1 \\
\hline
3x^2 + 2x + 2 \\
-9x^3 - 6x^2 - 6x \quad \\
3x^4 + 2x^3 + 2x^2 \qquad\qquad \\
\hline
3x^4 - 7x^3 - x^2 - 4x + 2
\end{array}
$$

10

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
  - $PQ = a_0b_0 + (a_0b_1+a_1b_0)\,x + (a_0b_2+a_1b_1+a_2b_0)\,x^2 + \ldots + (a_{m-2}b_{m-1}+a_{m-1}b_{m-2})\,x^{2m-3} + a_{m-1}b_{m-1}\,x^{2m-2}$
- Obvious Algorithm:
  - Compute all a_ib_j and collect terms
  - Θ **(n^2)** time

11

Naive Divide and Conquer

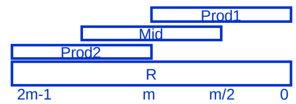
- Assume $m=2k$
  - $P = (a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-2} x^{k-2} + a_{k-1} x^{k-1}) + (a_k + a_{k+1} x + \ldots + a_{m-2}x^{k-2} + a_{m-1}x^{k-1})\,x^k = P_0 + P_1 x^k$
  - $Q = Q_0 + Q_1 x^k$
- $PQ = (P_0+P_1x^k)(Q_0+Q_1x^k) = P_0Q_0 + (P_1Q_0+P_0Q_1)x^k + P_1Q_1x^{2k}$
- 4 sub-problems of size $k=m/2$ plus linear combining
  - $T(m)=4T(m/2)+cm$
  - Solution $T(m) = O(m^2)$

12

Karatsuba's Algorithm

- A better way to compute the terms
  - Compute
    - P_0Q_0
    - P_1Q_1
    - $(P_0+P_1)(Q_0+Q_1)$ which is $P_0Q_0+P_1Q_0+P_0Q_1+P_1Q_1$
  - Then
    - $P_0Q_1+P_1Q_0 = (P_0+P_1)(Q_0+Q_1) - P_0Q_0 - P_1Q_1$
  - 3 sub-problems of size m/2 plus O(m) work
    - $T(m) = 3\,T(m/2) + cm$
    - $T(m) = O(m^{\alpha})$ where $\alpha = \log_2 3 = 1.59...$

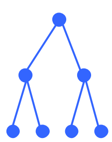
Karatsuba: Details



PolyMul(P, Q):

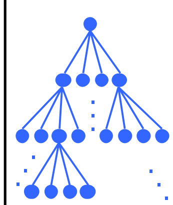
```
// P, Q are length m =2k vectors, with P[i], Q[i] being
// the coefficient of x^i in polynomials P, Q respectively.
Let Pzero be elements 0..k-1 of P; Pone be elements k..m-1
Qzero, Qone : similar
Prod1 = PolyMul(Pzero, Qzero); // result is a (2k-1)-vector
Prod2 = PolyMul(Pone, Qone);   // ditto
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



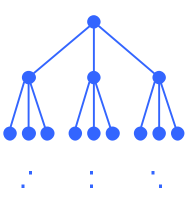
Level	Num	Size	Work
0	$1=2^0$	n	cn
1	$2=2^1$	n/2	2 c n/2
2	$4=2^2$	n/4	4 c n/4
…	…	…	…
i	2^i	$n/2^i$	2^i c $n/2^i$
…	…	…	…
k-1	2^{k-1}	$n/2^{k-1}$	2^{k-1} c $n/2^{k-1}$
k	2^k	$n/2^k=1$	2^k T(1)

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



Level	Num	Size	Work
0	$1=4^0$	n	cn
1	$4=4^1$	n/2	4 c n/2
2	$16=4^2$	n/4	16 c n/4
…	…	…	…
i	4^i	$n/2^i$	4^i c $n/2^i$
…	…	…	…
k-1	4^{k-1}	$n/2^{k-1}$	4^{k-1} c $n/2^{k-1}$
k	4^k	$n/2^k=1$	4^k T(1)

Solve: T(1) = c
T(n) = 3 T(n/2) + cn



Level	Num	Size	Work
0	$1=3^0$	n	cn
1	$3=3^1$	n/2	3 c n/2
2	$9=3^2$	n/4	9 c n/4
…	…	…	…
i	3^i	$n/2^i$	3^i c $n/2^i$
…	…	…	…
k-1	3^{k-1}	$n/2^{k-1}$	3^{k-1} c $n/2^{k-1}$
k	3^k	$n/2^k=1$	3^k T(1)

$n = 2^k$; $k = \log_2 n$

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

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

$$T(n) = \sum_{i=0}^{k} 3^i cn/2^i$$
$$= cn\sum_{i=0}^{k} 3^i/2^i$$
$$= cn\sum_{i=0}^{k} \left(\frac{3}{2}\right)^i$$
$$= cn\frac{\left(\frac{3}{2}\right)^{k+1}-1}{\left(\frac{3}{2}\right)-1}$$

$$\sum_{i=0}^{k} x^i = \frac{x^{k+1}-1}{x-1} \quad (x \neq 1)$$

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

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

$$< 2cn\left(\frac{3}{2}\right)^{k+1}$$

$$= 3cn\left(\frac{3}{2}\right)^{k}$$

$$= 3cn\frac{3^k}{2^k}$$

19

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

$$= 3cn\frac{3^{\log_2 n}}{2^{\log_2 n}}$$

$$= 3cn\frac{3^{\log_2 n}}{n}$$

$$= 3c3^{\log_2 n}$$

$$= 3c\left(n^{\log_2 3}\right)$$

$$= O\left(n^{1.59...}\right)$$

$$a^{\log_b n} = \left(b^{\log_b a}\right)^{\log_b n} = \left(b^{\log_b n}\right)^{\log_b a} = n^{\log_b a}$$

20

Master Divide and Conquer Recurrence

- If $T(n)=aT(n/b)+cn^k$ 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.

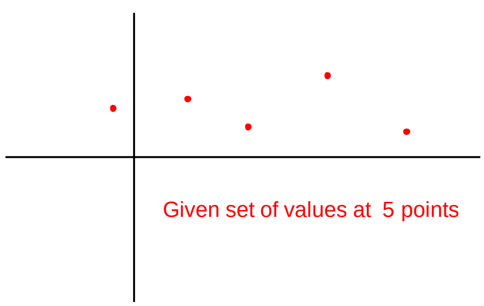
21

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 $\Theta(n \log n \operatorname{loglog} n)$,
    - but mostly unused in practice

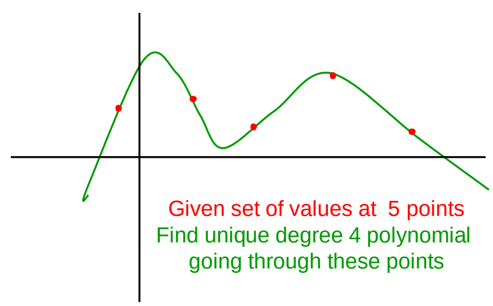
22

Hints towards FFT: I. Interpolation



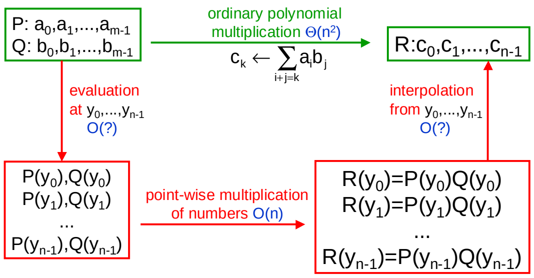
23

Hints towards FFT: I. Interpolation



24

Hints towards FFT: II. Evaluation & Interpolation



Hints towards FFT: III. Evaluation at Special Points

- Evaluation of polynomial at 1 point takes O(m), so m points (naively) takes O(m²)—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

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

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

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

Multiplying Matrices

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

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

Multiplying Matrices

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

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

Multiplying Matrices

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

Multiplying Matrices

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

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

- $T(n)=8T(n/2)+4(n/2)^2=8T(n/2)+n^2$
  - $8>2^2$ so T(n) is

$$\Theta(n^{\log_b a}) = \Theta(n^{\log_2 8}) = \Theta(n^3)$$

31

Strassen's 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)

32

The algorithm

- $P_1=A_{12}(B_{11}+B_{21})$ $\quad P_2=A_{21}(B_{12}+B_{22})$
- $P_3=(A_{11}-A_{12})B_{11}$ $\quad P_4=(A_{22}-A_{21})B_{22}$
- $P_5=(A_{22}-A_{12})(B_{21}-B_{22})$
- $P_6=(A_{11}-A_{21})(B_{12}-B_{11})$
- $P_7=(A_{21}-A_{12})(B_{11}+B_{22})$
- $C_{11}=P_1+P_3$ $\quad C_{12}=P_2+P_3+P_6-P_7$
- $C_{21}=P_1+P_4+P_5+P_7$ $\quad C_{22}=P_2+P_4$

33

Another D&C Example: 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 $a^n=a^m \cdot a^m$

34

Divide & Conquer Algorithm

- Power(a,n)

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

35

Analysis

- Worst-case recurrence
  - $T(n)=T(\lfloor n/2\rfloor)+2$
- By master theorem
  - $T(n)=O(\log n)$
- More precise analysis:
  - $T(n)=\lceil \log_2 n\rceil$ + # of 1's in n's binary representation

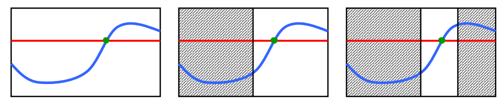
36

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 multiplies of 1024 bit numbers
    - relatively easy for modern machines
  - Naive algorithm: 2^{1024} multiplies

37

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**

38

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, …

39