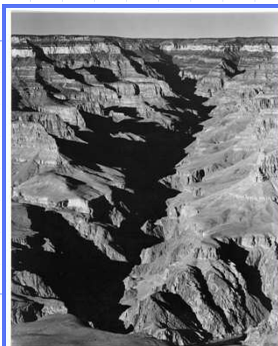
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Divide-and-Conquer

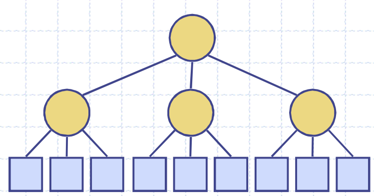


Grand Canyon from South Rim, 1941. Ansel Adams. U.S. government image. U.S. National Archives and Records Administration.

1

Divide-and-Conquer

- Divide-and conquer is a general algorithm design paradigm:
  - Divide: divide the input data S in two or more disjoint subsets S_1, S_2, …
  - Conquer: solve the subproblems recursively
  - Combine: combine the solutions for S_1, S_2, …, into a solution for S
- The base case for the recursion are subproblems of constant size
- Analysis can be done using recurrence equations

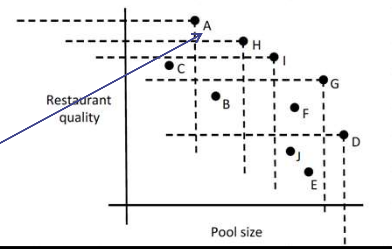


2

Maxima Set Problem

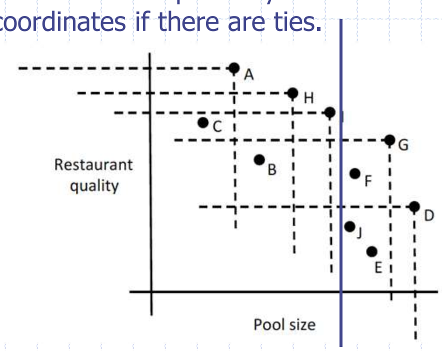
- We can visualize various trade-offs for optimizing two-dimensional data, such as points representing hotels according to their pool size and restaurant quality, by plotting each as a two-dimensional point, (x, y), where x is the pool size and y is the restaurant quality score.
- We say that such a point is a **maximum point** in a set if there is no other point, (x′, y′), in that set such that x ≤ x′ and y ≤ y′.
- The maximum points are the best potential choices based on these two dimensions and finding all of them is the **maxima set** problem.

We can efficiently find all the maxima points by divide-and-conquer.
Here the maxima set is {A,H,I,G,D}.



Solving the Maxima Set Problem

- A point (x, y) is a **maximum point** in S if there is no other point, (x′, y′), in S such that x ≤ x′ and y ≤ y′.
- To find a **maxima set** for a set, S, of n points in the plane, we may divide S into two equal parts.
- We compare two points in S using a lexicographic ordering of the points in S, that is, where we order based primarily on x-coordinates and then by y-coordinates if there are ties.

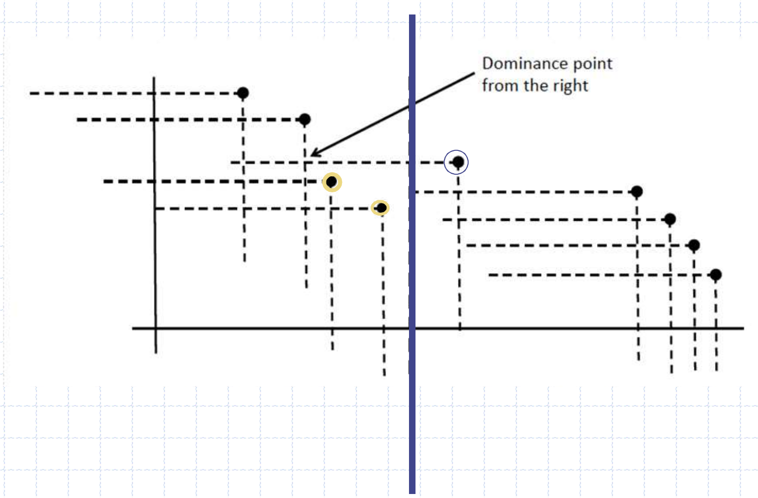


Divide-and-Conquer Solution

- Base case: If $n \le 1$, the maxima set is just S itself.
- Divide: let $p = (x_p, y_p)$ be the median point in S according to the lexicographic order. Then $x = x_p$ is a line dividing S into two halves.
- Conquer: we recursively solve the maxima-set problem for the set of points on the left of this line and also for the points on the right.
- Combine:
  - The maxima set of points on the right are also maxima points for S.
  - …

5

Example for the Combine Step



6

Divide-and-Conquer Solution

- Base case: If $n \le 1$, the maxima set is just S itself.
- Divide: let $p = (x_p, y_p)$ be the median point in S according to the lexicographic order. Then $x = x_p$ is a line dividing S into two halves.
- Conquer: we recursively solve the maxima-set problem for the set of points on the left of this line and also for the points on the right.
- Combine:
  - The maxima set of points on the right are also maxima points for S.
  - But some of the maxima points for the left set might be dominated by a point from the right, namely the point, q, that is leftmost.
  - So then we do a scan of the left set of maxima, removing any points that are dominated by q.
  - The union of remaining set of maxima from the left and the maxima set from the right is the set of maxima for S.

Pseudo-code

Algorithm MaximaSet(S):

  Input: A set, S, of n points in the plane
  Output: The set, M, of maxima points in S

  if $n \le 1$ **then**
    return S
  Let p be the median point in S, by lexicographic (x, y)-coordinates
  Let L be the set of points lexicographically less than p in S
  Let G be the set of points lexicographically greater than or equal to p in S
  $M_1 \leftarrow$ MaximaSet(L)
  $M_2 \leftarrow$ MaximaSet(G)
  Let q be the lexicographically smallest point in M_2
  for each point, r, in M_1 **do**
    if ~~$x(r) \le x(q)$ **and**~~ $y(r) \le y(q)$ **then**
      Remove r from M_1
  return $M_1 \cup M_2$

A Little Implementation Detail

- There is the issue of how to efficiently find the point, p, that is the median point in a lexicographical ordering of the points in S according to their (x, y)-coordinates.
- There are two immediate possibilities:
  - One choice is to use a linear-time median-finding algorithm, such as that given in Section 9.2. O(n) for each recursive call.
  - Another choice is to sort the points in S lexicographically by their (x, y)-coordinates as a preprocessing step, prior to calling the MaxmaSet algorithm on S. O(n log(n)) for preprocessing and O(1) for each recursive call, to find the middle of the list.

9

Analysis

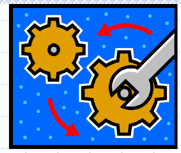
- In either case, the rest of the non-recursive steps can be performed in O(n) time, so this implies that, ignoring floor and ceiling functions, the running time for the divide-and-conquer maxima-set algorithm can be specified as follows (where b is a constant):

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

- Thus, according to the merge sort example, this algorithm runs in O(n log n) time.

10

Iterative Substitution



- In the iterative substitution, or "plug-and-chug," technique, we iteratively apply the recurrence equation to itself and see if we can find a pattern:

$$\begin{aligned} T(n) &= 2T(n/2) + bn \\ &= 2(2T(n/2^2)) + b(n/2)) + bn \\ &= 2^2T(n/2^2) + 2bn \\ &= 2^3T(n/2^3) + 3bn \\ &= 2^4T(n/2^4) + 4bn \\ &= \ldots \\ &= 2^iT(n/2^i) + ibn \end{aligned}$$

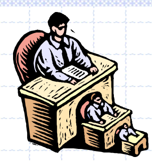
- Note that base, T(n)=b, case occurs when 2^i=n. That is, i = log n.
- So,

$$T(n) = bn + bn\log n$$

- Thus, T(n) is O(n log n).

11

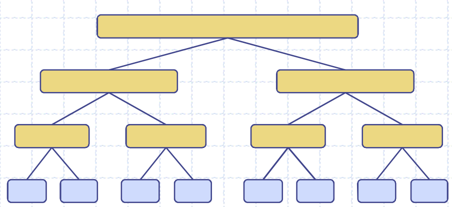
The Recursion Tree



- Draw the recursion tree for the recurrence relation and look for a pattern:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

depth	T's	size	time
0	1	n	bn
1	2	$n/2$	bn
i	2^i	$n/2^i$	bn
…	…	…	…

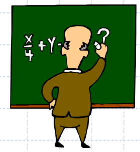


Total time = $bn + bn \log n$

(last level plus all previous levels)

12

Guess-and-Test Method



- In the guess-and-test method, we guess a closed form solution and then try to prove it is true by induction:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

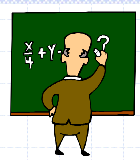
- Guess: T(n) ≤ cn log n.

$$\begin{aligned} T(n) &= 2T(n/2) + bn \\ &\leq 2(c(n/2)\log(n/2)) + bn \\ &= cn(\log n - \log 2) + bn \\ &= cn\log n - cn + bn \\ &= cn\log n - (c-b)n \end{aligned}$$

- We can conclude that T(n) ≤ cn log n if c ≥ b.

13

Guess-and-Test Method



- In the guess-and-test method, we guess a closed form solution and then try to prove it is true by induction:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn\log n & \text{if } n \geq 2 \end{cases}$$

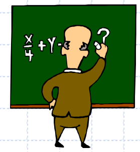
- Guess: T(n) ≤ cn log n.

$$\begin{aligned} T(n) &= 2T(n/2) + bn\log n \\ &\leq 2(c(n/2)\log(n/2)) + bn\log n \\ &= cn(\log n - \log 2) + bn\log n \\ &= cn\log n - cn + bn\log n \end{aligned}$$

- Wrong: we cannot make this last line be less than cn log n

14

Guess-and-Test Method, (cont.)



- Recall the recurrence equation:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn\log n & \text{if } n \ge 2 \end{cases}$$

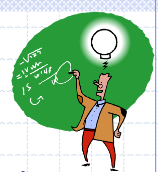
- Guess #2: T(n) ≤ cn log² n.

$$\begin{aligned} T(n) &= 2T(n/2) + bn\log n \\ &\le 2(c(n/2)\log^2(n/2)) + bn\log n \\ &= cn(\log n - \log 2)^2 + bn\log n \\ &= cn\log^2 n - 2cn\log n + cn + bn\log n \\ &\le cn\log^2 n \qquad \text{if } c \ge b. \end{aligned}$$

- So, T(n) is O(n log² n).
- In general, to use this method, you need to have a good guess and you need to be good at induction proofs.

15

Master Method



- Many divide-and-conquer recurrence equations have the form:

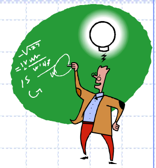
$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \ge d \end{cases}$$

- The Master Theorem:

1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
2. if $f(n)$ is $\Theta(n^{\log_b a}\log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a}\log^{k+1} n)$
3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \le \delta f(n)$ for some $\delta < 1$.

16

Master Method, Example 1



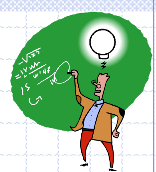
- The form: $T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \ge d \end{cases}$
- The Master Theorem:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \le \delta f(n)$ for some $\delta < 1$.
- Example:

$$T(n) = 4T(n/2) + n$$

Solution: $\log_b a=2$, so case 1 says T(n) is $O(n^2)$.

17

Master Method, Example 2



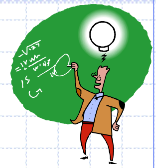
- The form: $T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \ge d \end{cases}$
- The Master Theorem:
  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b) \le \delta f(n)$ for some $\delta < 1$.
- Example:

$$T(n) = 2T(n/2) + n \log n$$

Solution: $\log_b a=1$, so case 2 says T(n) is $O(n \log^2 n)$.

18

Master Method, Example 3



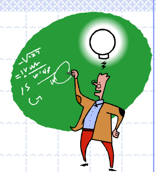
- The form: $T(n)=\begin{cases} c & \text{if } n<d \\ aT(n/b)+f(n) & \text{if } n\ge d \end{cases}$
- The Master Theorem:
  1. if $f(n)$ is $O(n^{\log_b a-\varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a}\log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a}\log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a+\varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b)\le \delta f(n)$ for some $\delta<1$.
- Example:

$$T(n)=T(n/3)+n\log n$$

Solution: $\log_b a=0$, so case 3 says T(n) is O(n log n).

19

Master Method, Example 4



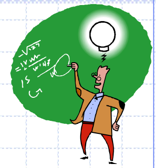
- The form: $T(n)=\begin{cases} c & \text{if } n<d \\ aT(n/b)+f(n) & \text{if } n\ge d \end{cases}$
- The Master Theorem:
  1. if $f(n)$ is $O(n^{\log_b a-\varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a}\log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a}\log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a+\varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b)\le \delta f(n)$ for some $\delta<1$.
- Example:

$$T(n)=9T(n/3)+n^3$$

Solution: $\log_b a=2$, so case 3 says T(n) is $O(n^3)$.

20

Master Method, Example 5



- The form: $T(n)=\begin{cases} c & \text{if } n<d \\ aT(n/b)+f(n) & \text{if } n\ge d \end{cases}$
- The Master Theorem:
  1. if $f(n)$ is $O(n^{\log_b a-\varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
  2. if $f(n)$ is $\Theta(n^{\log_b a}\log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a}\log^{k+1} n)$
  3. if $f(n)$ is $\Omega(n^{\log_b a+\varepsilon})$, then $T(n)$ is $\Theta(f(n))$, provided $af(n/b)\le \delta f(n)$ for some $\delta<1$.
- Example:

$$T(n)=T(n/2)+1$$ (binary search)

Solution: $\log_b a=0$, so case 2 says T(n) is O(log n).

21

Possible Quiz Questions

Using Master Theorem, find solutions for the following recurrence relations:

$$T(n)=2T(n/2)+\log n$$

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

22

Possible Quiz Question

- Please design an efficient algorithm (as fast as you can) which will merge n sorted lists of the same length m, into a single sorted list. You may use available function merge(A, B), which returns a sorted list consisting of elements from two sorted lists A and B, with cost O(|A| + |B|), where |X| is the length of X, i.e., the number of elements in X. Please analyze the complexity of your algorithm in terms of n and m.

23

Integer Addition

- Addition. Given two n-bit integers a and b, compute $a + b$.
- Grade-school. $\Theta(n)$ bit operations.

1	1	1	1	1	1	0	1		*carry*
	1	1	0	1	0	1	0	1	*a*
+	0	1	1	1	1	1	0	1	*b*
1	0	1	0	1	0	0	1	0	

Remark: Grade-school addition algorithm is optimal.

Integer Multiplication

- Multiplication. Given two n-bit integers a and b, compute $a \times b$.
- Grade-school. $\Theta(n^2)$ bit operations.

```
                1 1 0 1 0 1 0 1
              × 0 1 1 1 1 1 0 1
                1 1 0 1 0 1 0 1
              0 0 0 0 0 0 0 0 0
            1 1 0 1 0 1 0 1 0
          1 1 0 1 0 1 0 1 0
        1 1 0 1 0 1 0 1 0
      1 1 0 1 0 1 0 1 0
    1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1
```

- Q. Is grade-school multiplication algorithm optimal?

25

Divide-and-Conquer Multiplication: Warmup

- To multiply two n-bit integers a and b:
  - Multiply four ½n-bit integers, recursively.
  - Add and shift to obtain result.

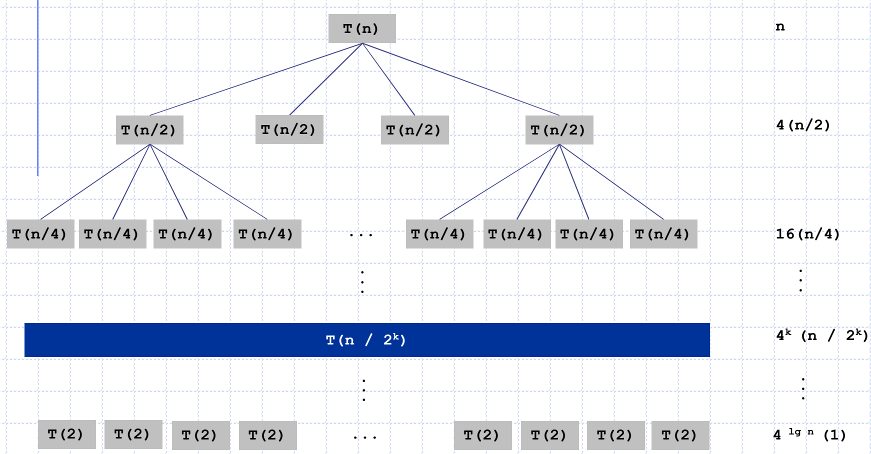
$$a = 2^{n/2} \cdot a_1 + a_0$$
$$b = 2^{n/2} \cdot b_1 + b_0$$
$$ab = \left(2^{n/2} \cdot a_1 + a_0\right)\left(2^{n/2} \cdot b_1 + b_0\right) = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left(a_1 b_0 + a_0 b_1\right) + a_0 b_0$$

- Ex. $a = \underbrace{1000}_{a_1}\underbrace{1101}_{a_0}$ $\quad b = \underbrace{1110}_{b_1}\underbrace{0001}_{b_0}$

26

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 4T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\lg n} n\,2^k = n\left(\frac{2^{1+\lg n} - 1}{2 - 1}\right) = 2n^2 - n$$



27

Karatsuba Multiplication

- To multiply two n-bit integers a and b:
  - Add two ½n bit integers.
  - Multiply three ½n-bit integers, recursively.
  - Add, subtract, and shift to obtain result.

$$\begin{aligned} a &= 2^{n/2} \cdot a_1 + a_0 \\ b &= 2^{n/2} \cdot b_1 + b_0 \\ ab &= 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left(a_1 b_0 + a_0 b_1\right) + a_0 b_0 \\ &= 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0\right) + a_0 b_0 \end{aligned}$$

1 2 1 3 3

28

Karatsuba Multiplication

- To multiply two n-bit integers a and b:
  - Add two ½n bit integers.
  - Multiply three ½n-bit integers, recursively.
  - Add, subtract, and shift to obtain result.

$$\begin{aligned} a &= 2^{n/2}\cdot a_1 + a_0 \\ b &= 2^{n/2}\cdot b_1 + b_0 \\ ab &= 2^{n}\cdot a_1 b_1 + 2^{n/2}\cdot (a_1 b_0 + a_0 b_1) + a_0 b_0 \\ &= 2^{n}\cdot \underset{①}{a_1 b_1} + 2^{n/2}\cdot \big(\underset{②}{(a_1+a_0)(b_1+b_0)} - \underset{①}{a_1 b_1} - \underset{③}{a_0 b_0}\big) + \underset{③}{a_0 b_0} \end{aligned}$$

$$T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1+\lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \Rightarrow T(n) = O(n^{\lg 3}) = O(n^{1.585})$$

Dot Product

Dot product. Given two length n vectors a and b, compute $c = a \cdot b$.

Grade-school. $\Theta(n)$ arithmetic operations.

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

$$\begin{aligned} a &= [.70 \quad .20 \quad .10] \\ b &= [.30 \quad .40 \quad .30] \\ a \cdot b &= (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32 \end{aligned}$$

Remark. Grade-school dot product algorithm is optimal.

30

Matrix Multiplication

Matrix multiplication. Given two n-by-n matrices A and B, compute $C = AB$.
Grade-school. $\Theta(n^3)$ arithmetic operations.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Q. Is grade-school matrix multiplication algorithm optimal?

Block Matrix Multiplication

C_{11}, A_{11}, A_{12}, B_{11}, B_{21}

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Matrix Multiplication: Warmup

To multiply two n-by-n matrices A and B:
- Divide: partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply 8 pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

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

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

33

Fast Matrix Multiplication

Key idea. multiply 2-by-2 blocks with only 7 multiplications.

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

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

- 7 multiplications.
- $18 = 8 + 10$ additions and subtractions.

34

Fast Matrix Multiplication

To multiply two n-by-n matrices A and B: [Strassen 1969]
- Divide: partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Compute: 14 $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices via 10 matrix additions.
- Conquer: multiply 7 pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.
- Assume n is a power of 2.
- $T(n)$ = # arithmetic operations.

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

35

Fast Matrix Multiplication

To multiply two n-by-n matrices A and B: [Strassen 1969]

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

	Multiplications	Additions	Complexity
Traditional alg.	n^3	$n^3 - n^2$	$\Theta(n^3)$
Recursive version	n^3	$n^3 - n^2$	$\Theta(n^3)$
Strassen's alg.	$n^{\log 7}$	$6n^{\log 7} - 6n^2$	$\Theta(n^{\log 7})$

Table 6.2 The number of arithmetic operations done by the three algorithms.

	n	Multiplications	Additions
Traditional alg.	100	1,000,000	990,000
Strassen's alg.	100	411,822	2,470,334
Traditional alg.	1000	1,000,000,000	999,000,000
Strassen's alg.	1000	264,280,285	1,579,681,709
Traditional alg.	10,000	10^{12}	9.99×10^{12}
Strassen's alg.	10,000	0.169×10^{12}	10^{12}

Table 6.3 Comparison between Strassen's algorithm and the traditional algorithm.

36

Fast Matrix Multiplication: Practice

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n = 128$.

Common misperception. *"Strassen is only a theoretical curiosity."*
- Apple reports 8x speedup on G4 Velocity Engine when $n \approx 2{,}500$.
- Range of instances where it's useful is a subject of controversy.

Remark. Can "Strassenize" $Ax = b$, determinant, eigenvalues, ...

37

Fast Matrix Multiplication: Theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?
A. Yes! [Strassen 1969] $\Theta(n^{\log_2 7}) = O(n^{2.807})$

Q. Multiply two 2-by-2 matrices with 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

Q. Two 3-by-3 matrices with 21 scalar multiplications?
A. Also impossible. $\Theta(n^{\log_3 21}) = O(n^{2.77})$

Begun, the decimal wars have. [Pan, Bini et al, Schönhage, ...]
- Two 20-by-20 matrices with $4{,}460$ scalar multiplications. $O(n^{2.805})$
- Two 48-by-48 matrices with $47{,}217$ scalar multiplications. $O(n^{2.7801})$
- A year later. $O(n^{2.7799})$
- December, 1979. $O(n^{2.521813})$
- January, 1980. $O(n^{2.521801})$

38

Fast Matrix Multiplication: Theory

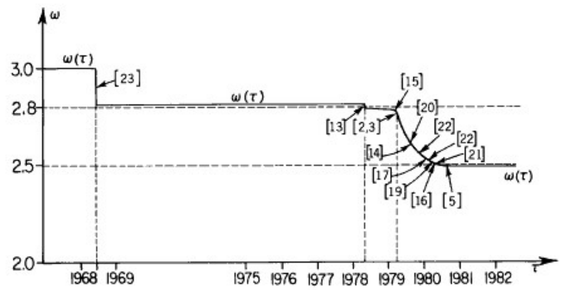


FIG. 1. *ω(t) is the best exponent announced by time τ.*

Best known. $O(n^{2.376})$ [Coppersmith-Winograd, 1987]

Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

Caveat. Theoretical improvements to Strassen are progressively less practical.

39

Possible Quiz Quetion

Given a m×m matrix M and a positive integer n, how to compute M^n efficiently?

40