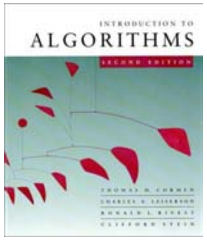
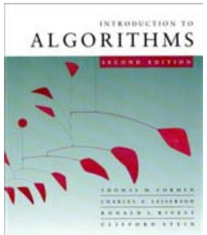


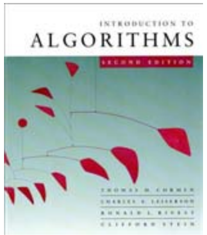
The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.



Merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.



Merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = 2\,T(n/2) + \Theta(n)$$

 # subproblems

subproblem size

work dividing and combining

Master theorem (reprise)

$$T(n) = aT(n/b) + f(n)$$

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

CASE 2: $f(n) = \Theta(n^{\log_b a})$

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$.

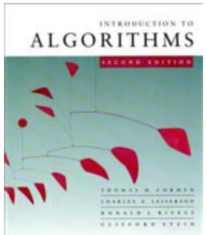
CASE 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$,

and regularity condition

$\Rightarrow T(n) = \Theta(f(n))$.

Merge sort: $a = 2$, $b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$

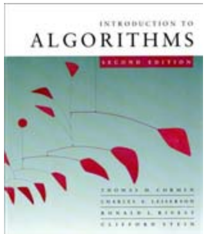
$\Rightarrow$ CASE 2 $T(n) = \Theta(n \lg n)$.



Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.



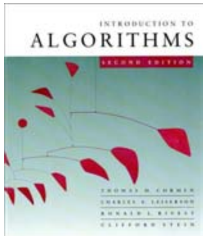
Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3	5	7	8	9	12	15



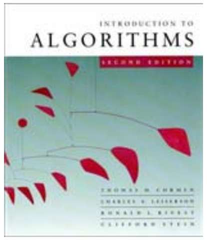
Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3	5	7	8	9	12	15



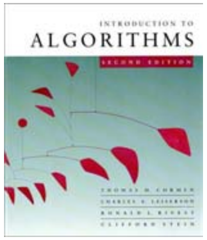
Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15



Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15

Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3	5	7	8	9	12	15

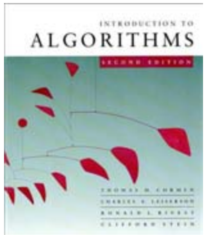
Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15



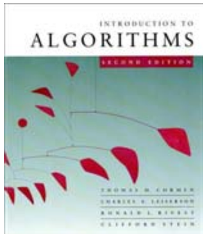
Recurrence for binary search

$$T(n) = 1\,T(n/2) + \Theta(1)$$

 # subproblems

subproblem size

work dividing and combining



Recurrence for binary search

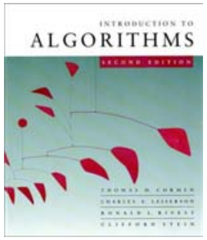
$$T(n) = 1\,T(n/2) + \Theta(1)$$

 # subproblems

subproblem size

work dividing and combining

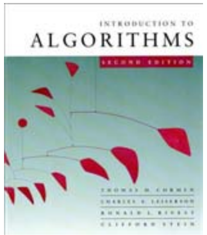
$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$
$$\Rightarrow T(n) = \Theta(\lg n)\,.$$



Powering a number

Problem: Compute a^n, where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.



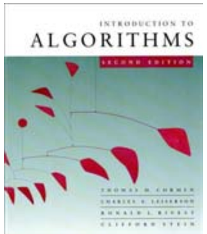
Powering a number

Problem: Compute a^n, where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$



Powering a number

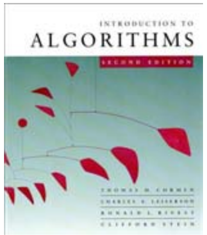
Problem: Compute a^n, where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \;\Rightarrow\; T(n) = \Theta(\lg n)\,.$$



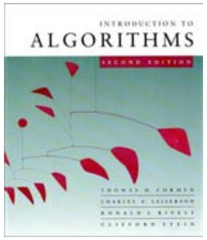
Matrix multiplication

Input: $A = [a_{ij}]$, $B = [b_{ij}]$.
Output: $C = [c_{ij}] = A \cdot B$.

$\left.\right\}$ $i, j = 1, 2, \ldots, n$.

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

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



Standard algorithm

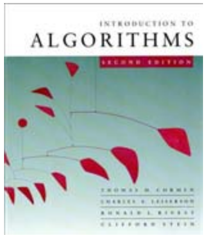
for $i \leftarrow 1$ **to** n

  do for $j \leftarrow 1$ **to** n

    do $c_{ij} \leftarrow 0$

     for $k \leftarrow 1$ **to** n

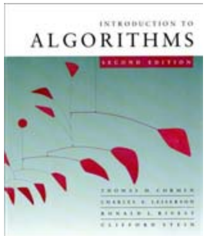
      do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$



Standard algorithm

for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ **to** n
        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time $= \Theta(n^3)$



Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

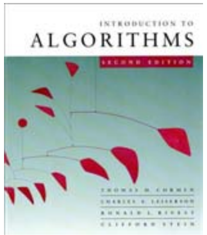
$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices



Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dh \\ u &= cf + dg \end{aligned}\right\}$$

recursive

8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices

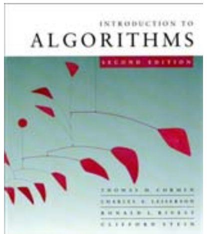
Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices



Analysis of D&C algorithm

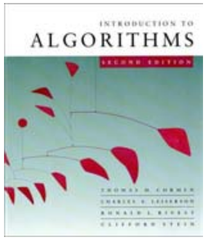
$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices

$$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\; \text{Case 1} \;\Rightarrow\; T(n) = \Theta(n^3).$$



Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

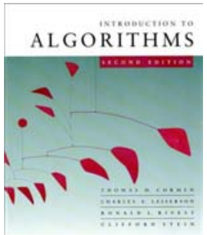
work adding submatrices

$$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^3).$$

No better than the ordinary algorithm.

Strassen's idea

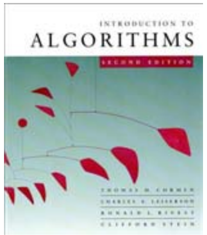
- Multiply 2×2 matrices with only 7 recursive mults.



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

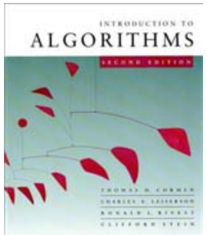


Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$



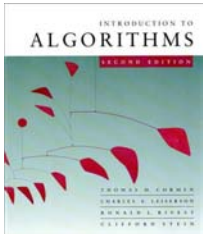
Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!

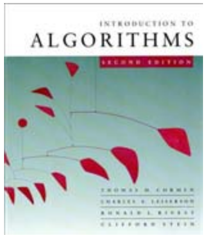


Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

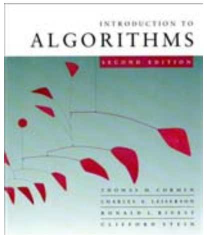
$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$



Strassen's algorithm

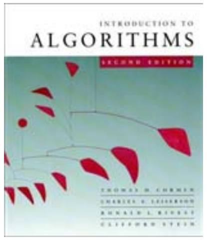
1. ***Divide:*** Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
2. ***Conquer:*** Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. ***Combine:*** Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.



Strassen's algorithm

1. ***Divide:*** Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
2. ***Conquer:*** Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. ***Combine:*** Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$



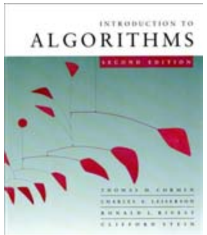
Analysis of Strassen

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

Analysis of Strassen

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \;\Rightarrow\; \textsc{Case 1} \;\Rightarrow\; T(n) = \Theta(n^{\lg 7}).$$

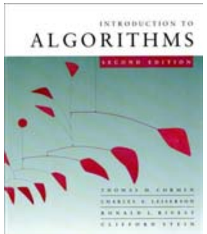


Analysis of Strassen

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^{\lg 7}).$$

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.



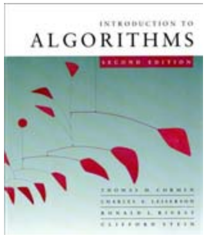
Analysis of Strassen

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^{\lg 7}).$$

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

Best to date (of theoretical interest only): $\Theta(n^{2.376\cdots})$.



Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method (so practice this math).
- The divide-and-conquer strategy often leads to efficient algorithms.