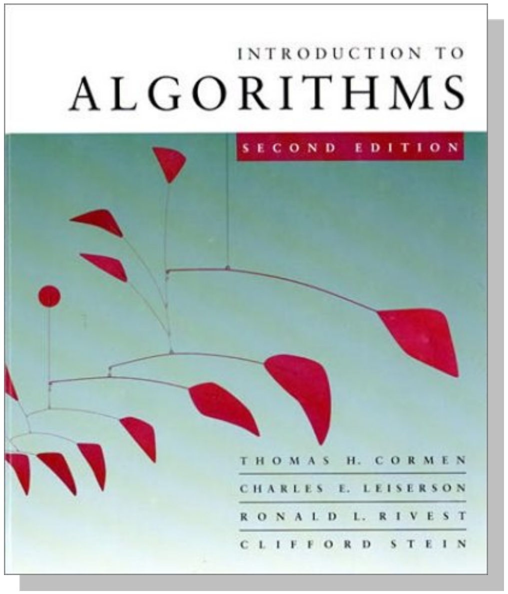
Introduction to Algorithms

6.046J/18.401J

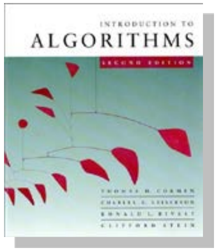


LECTURE 3

Divide and Conquer

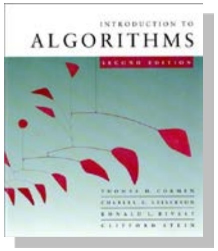
- Binary search
- Matrix multiplication
- Strassen's algorithm

Prof. Erik D. Demaine



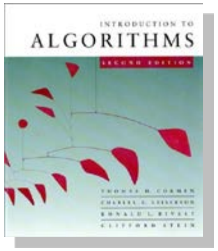
The divide-and-conquer design paradigm

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

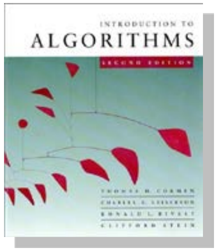
1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

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

 # subproblems

subproblem size

work dividing and combining



Master theorem (reprise)

$$T(n) = a\,T(n/b) + f(n)$$

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

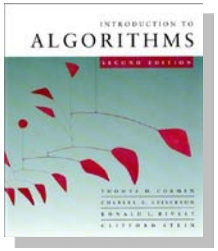
$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

CASE 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, constant $k \geq 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

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

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



Master theorem (reprise)

$$T(n) = a\,T(n/b) + f(n)$$

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

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

CASE 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, constant $k \geq 0$

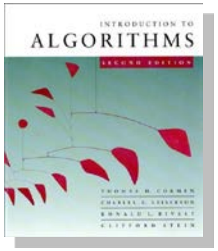
$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

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

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

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

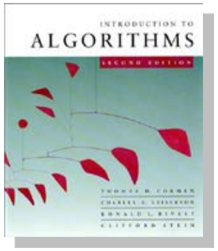
$\Rightarrow$ CASE 2 $(k = 0) \Rightarrow T(n) = \Theta(n \lg n)$.



Binary search

Find an element in a sorted array:

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



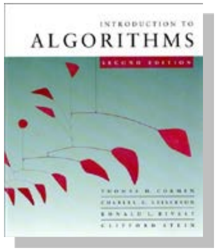
Binary search

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



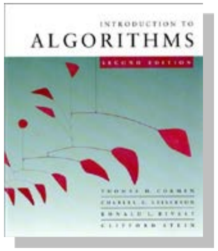
Binary search

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



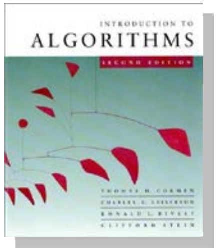
Binary search

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



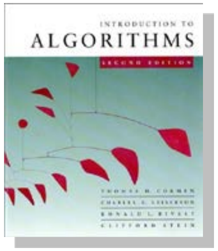
Binary search

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



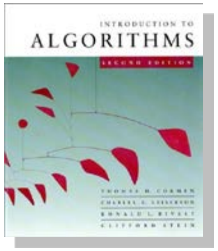
Binary search

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



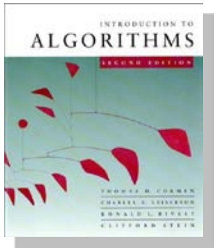
Binary search

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



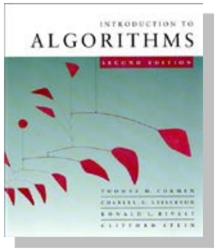
Recurrence for binary search

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

 # subproblems

subproblem size

work dividing and combining



Recurrence for binary search

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

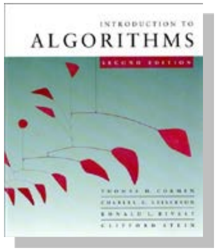
 # subproblems

subproblem size

work dividing and combining

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE } 2\ (k = 0)$$

$$\Rightarrow T(n) = \Theta(\lg n)\,.$$



Matrix multiplication

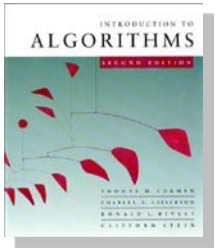
Input: $A = [a_{ij}],\ B = [b_{ij}].$

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

$$\left.\begin{array}{l} \\ \\ \end{array}\right\} \quad i, j = 1, 2, \ldots, n.$$

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

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



Standard algorithm

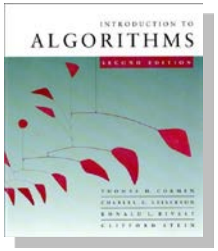
for $i \leftarrow 1$ **to** n

  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

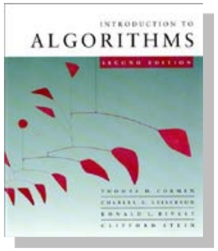
  do **for** $j \leftarrow 1$ **to** n

    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:

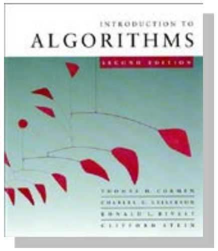
$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

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

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

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:

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

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

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

<u>*recursive*</u>

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

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

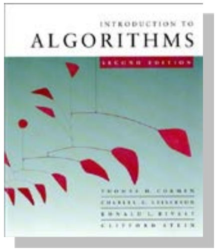
Analysis of D&C algorithm

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

 # submatrices

submatrix size

work adding submatrices



Analysis of D&C algorithm

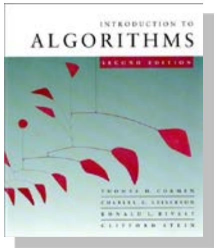
$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # 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

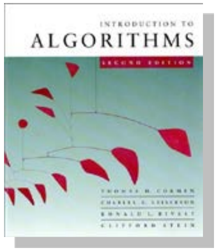
work adding submatrices

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

No better than the ordinary algorithm.

Strassen's idea

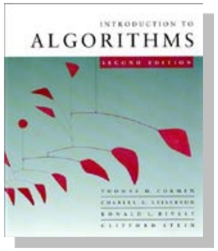
- Multiply 2×2 matrices with only 7 recursive mults.



Strassen's idea

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

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

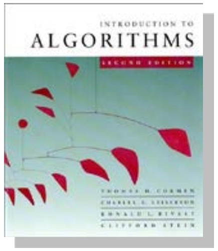


Strassen's idea

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

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

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



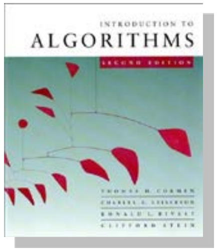
Strassen's idea

- 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!

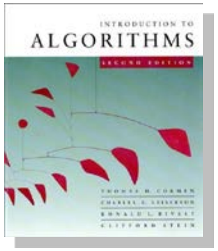


Strassen's idea

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

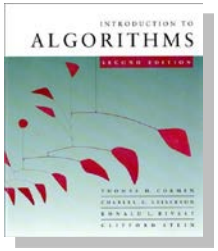
$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

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



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.

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

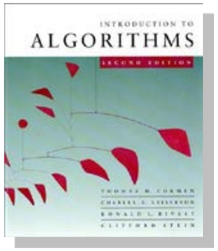
Analysis of Strassen

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

Analysis of Strassen

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

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

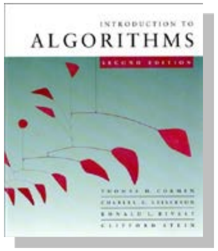


Analysis of Strassen

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

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

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.



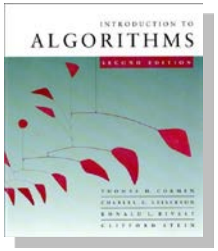
Analysis of Strassen

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

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

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): $\Theta(n^{2.376\cdots})$.

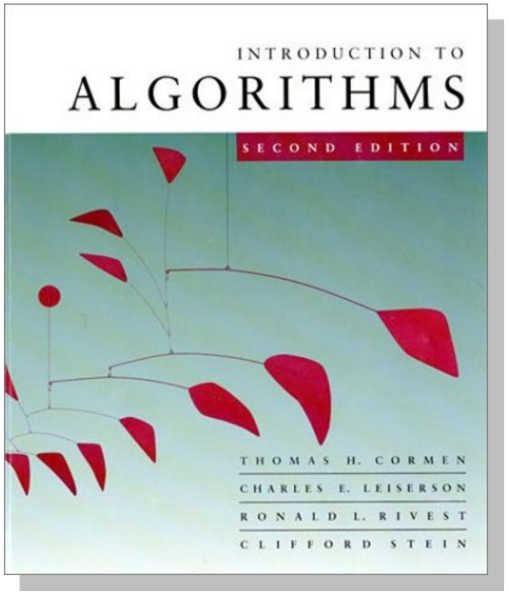


Conclusion

- 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.

Introduction to Algorithms

6.046J/18.401J

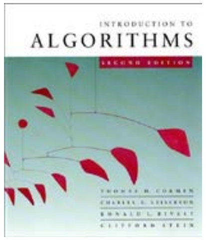


LECTURE 4

Quicksort

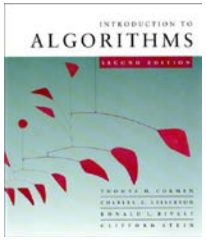
- Divide and conquer
- Partitioning
- Worst-case analysis
- Intuition
- Randomized quicksort
- Analysis

Prof. Charles E. Leiserson



Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts "in place" (like insertion sort, but not like merge sort).
- Very practical (with tuning).



Divide and conquer

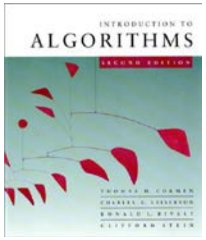
Quicksort an n-element array:

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.

$\leq x$	x	$\geq x$

2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

Key: *Linear-time partitioning subroutine.*

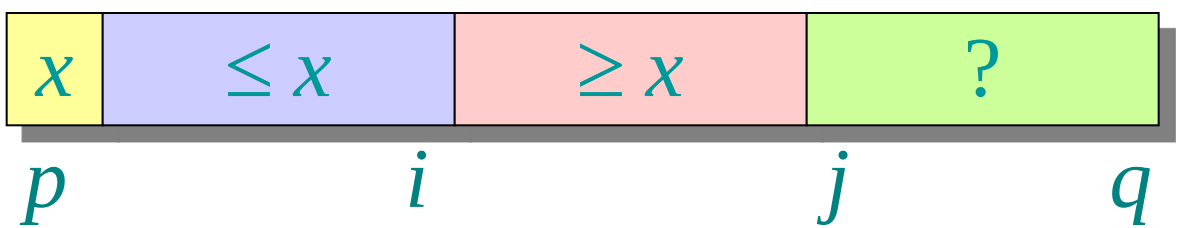


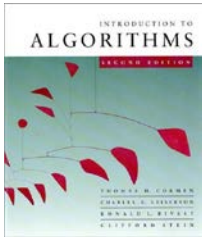
Partitioning subroutine

PARTITION(A, p, q) ⊳ $A[p \,..\, q]$
$\quad x \leftarrow A[p]$ ⊳ pivot $= A[p]$
$\quad i \leftarrow p$
$\quad$ **for** $j \leftarrow p + 1$ **to** q
$\qquad$ **do if** $A[j] \le x$
$\qquad\quad$ **then** $i \leftarrow i + 1$
$\qquad\qquad$ exchange $A[i] \leftrightarrow A[j]$
$\quad$ exchange $A[p] \leftrightarrow A[i]$
$\quad$ **return** i

Running time $= O(n)$ for n elements.

Invariant:





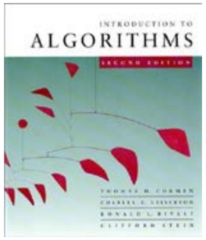
Example of partitioning

6	10	13	5	8	3	2	11
i	j						

Example of partitioning

6	10	13	5	8	3	2	11

i $\longrightarrow j$



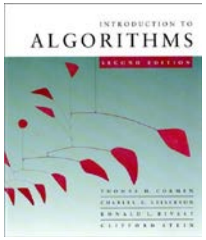
Example of partitioning

6	10	13	5	8	3	2	11
i			*j*				

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11
	i		j				

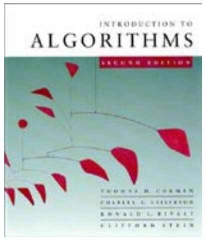


Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

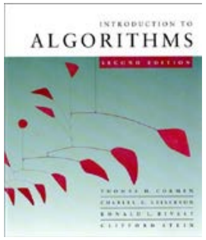
i $\longrightarrow j$



Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11
	i				j		



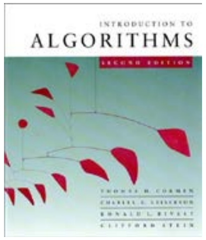
Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

i $\qquad\qquad\qquad\qquad$ j



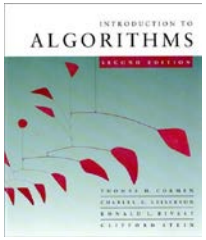
Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

i $\longrightarrow j$



Example of partitioning

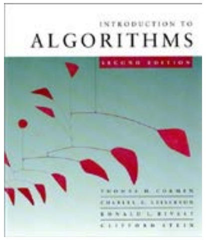
6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

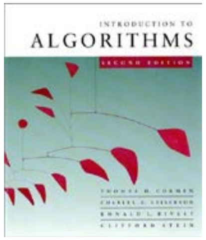
6	5	3	2	8	13	10	11

$\longrightarrow i$ $\qquad\qquad j$



Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
			i				j



Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

6	5	3	2	8	13	10	11

i $\longrightarrow j$

Example of partitioning

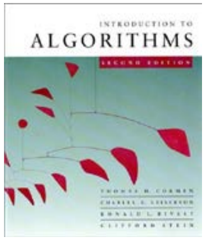
6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

6	5	3	2	8	13	10	11

2	5	3	6	8	13	10	11
			i				



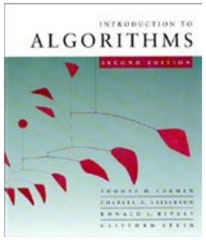
Pseudocode for quicksort

QUICKSORT(A, p, r)
    if $p < r$
        then $q \leftarrow$ PARTITION(A, p, r)
            QUICKSORT$(A, p, q–1)$
            QUICKSORT$(A, q+1, r)$

Initial call: QUICKSORT$(A, 1, n)$

Analysis of quicksort

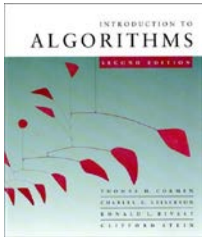
- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.



Worst-case of quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= \Theta(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2) \qquad \textbf{\textit{(arithmetic series)}}
\end{aligned}
$$



Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

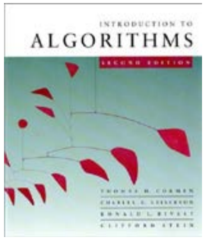
$$T(n)$$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$$cn$$

$$T(0) \qquad T(n-1)$$



Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$$
\begin{array}{ccccc}
 & cn & & & \\
T(0) & & c(n-1) & & \\
 & & T(0) & & T(n-2)
\end{array}
$$

Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

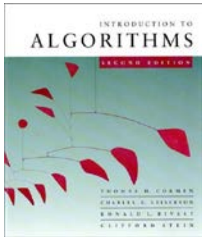
$$cn$$

$$T(0) \quad c(n–1)$$

$$T(0) \quad c(n–2)$$

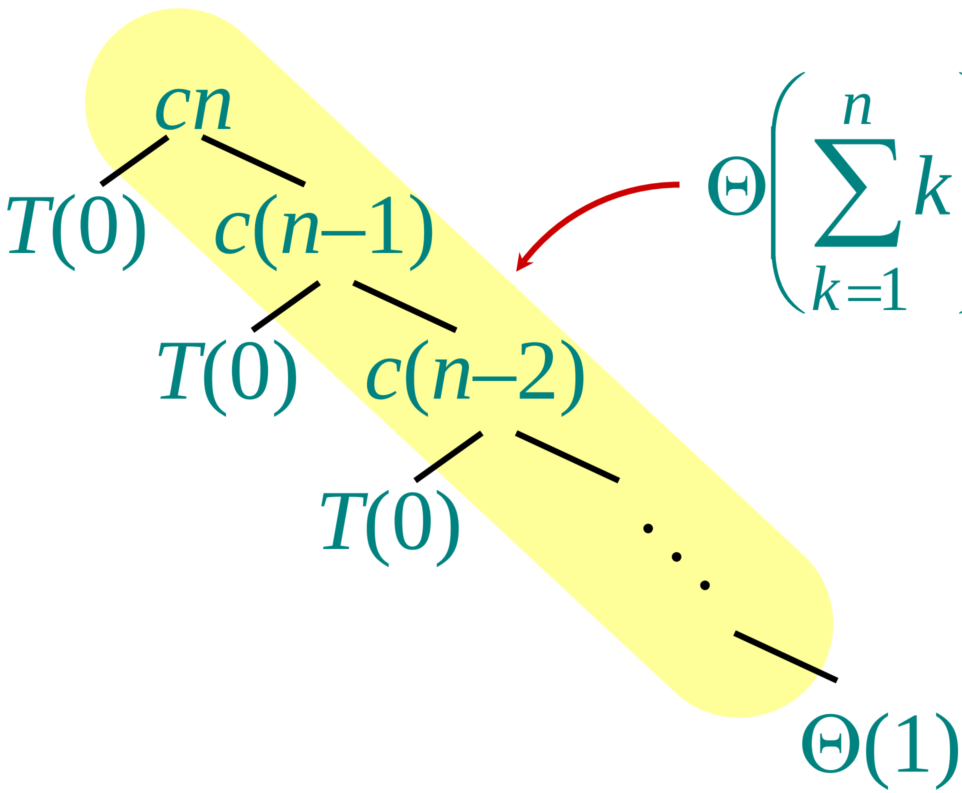
$$T(0) \quad \ddots$$

$$\Theta(1)$$

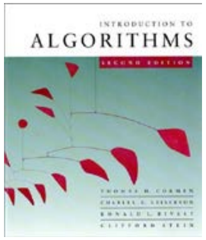


Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

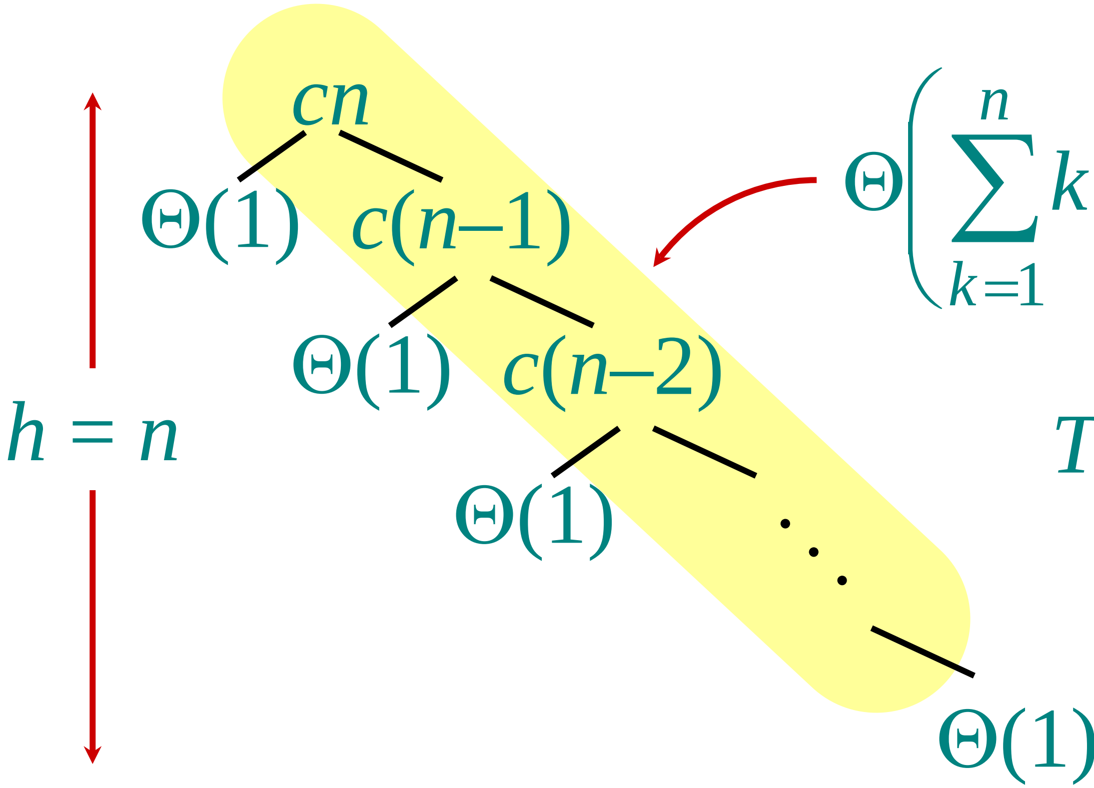


$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$



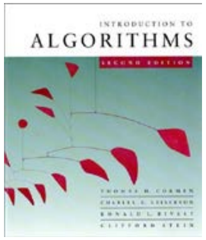
Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

$$T(n) = \Theta(n) + \Theta(n^2) = \Theta(n^2)$$



Best-case analysis

(For intuition only!)

If we're lucky, PARTITION splits the array evenly:

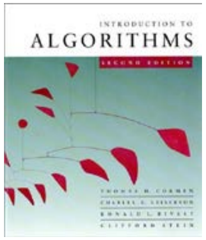
$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

(same as merge sort)

What if the split is always $\frac{1}{10} : \frac{9}{10}$?

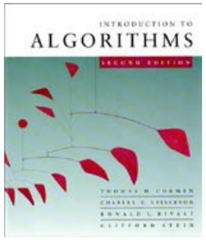
$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?



Analysis of “almost-best” case

$$T(n)$$



Analysis of “almost-best” case

$$cn$$

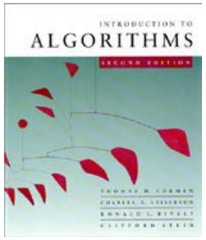
$$T\left(\tfrac{1}{10}n\right) \qquad\qquad T\left(\tfrac{9}{10}n\right)$$

Analysis of "almost-best" case

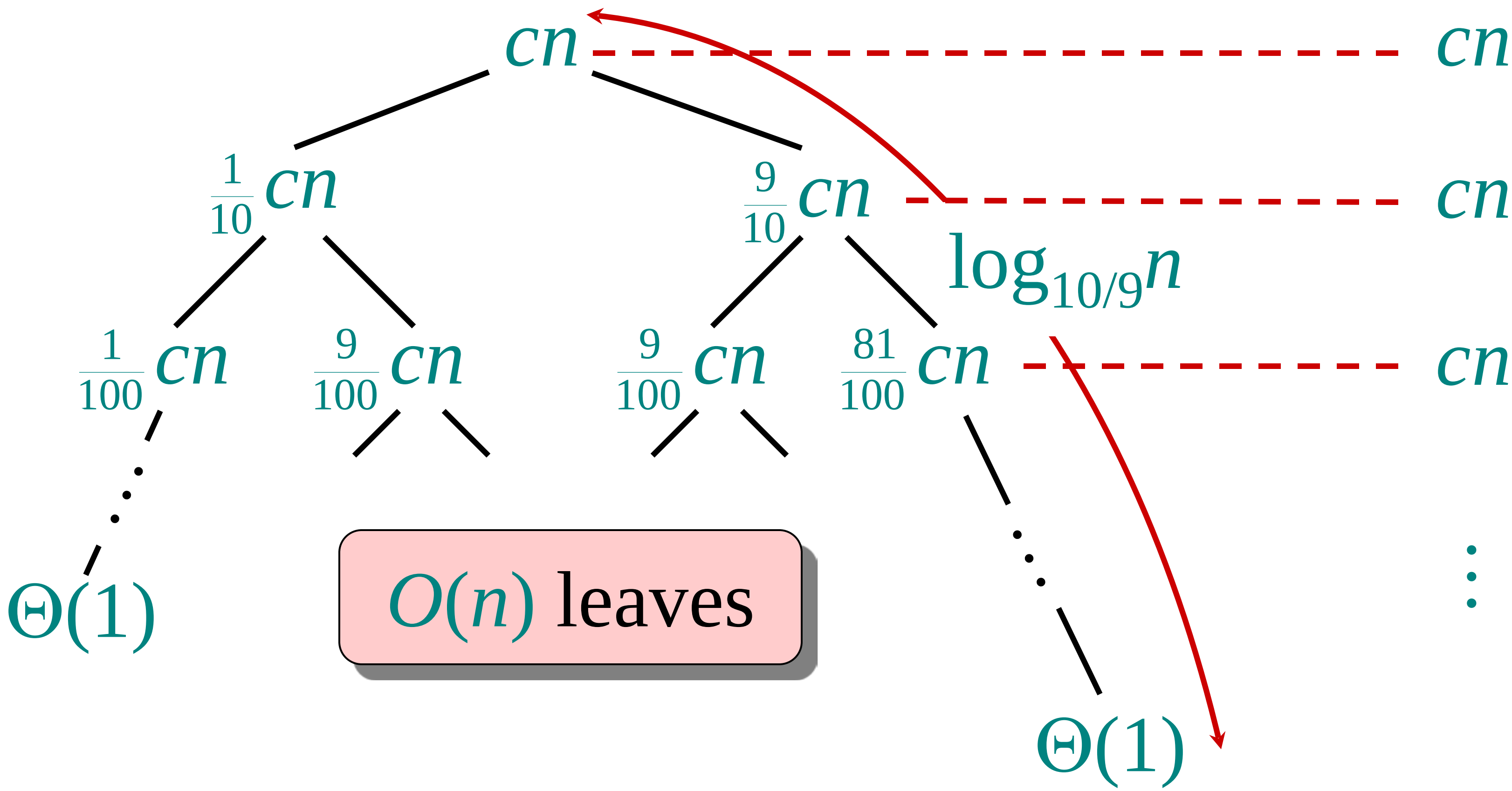
$$cn$$

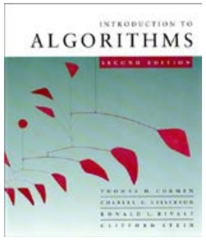
$$\tfrac{1}{10}cn \qquad \tfrac{9}{10}cn$$

$$T\left(\tfrac{1}{100}n\right)\; T\left(\tfrac{9}{100}n\right) \qquad T\left(\tfrac{9}{100}n\right)\; T\left(\tfrac{81}{100}n\right)$$

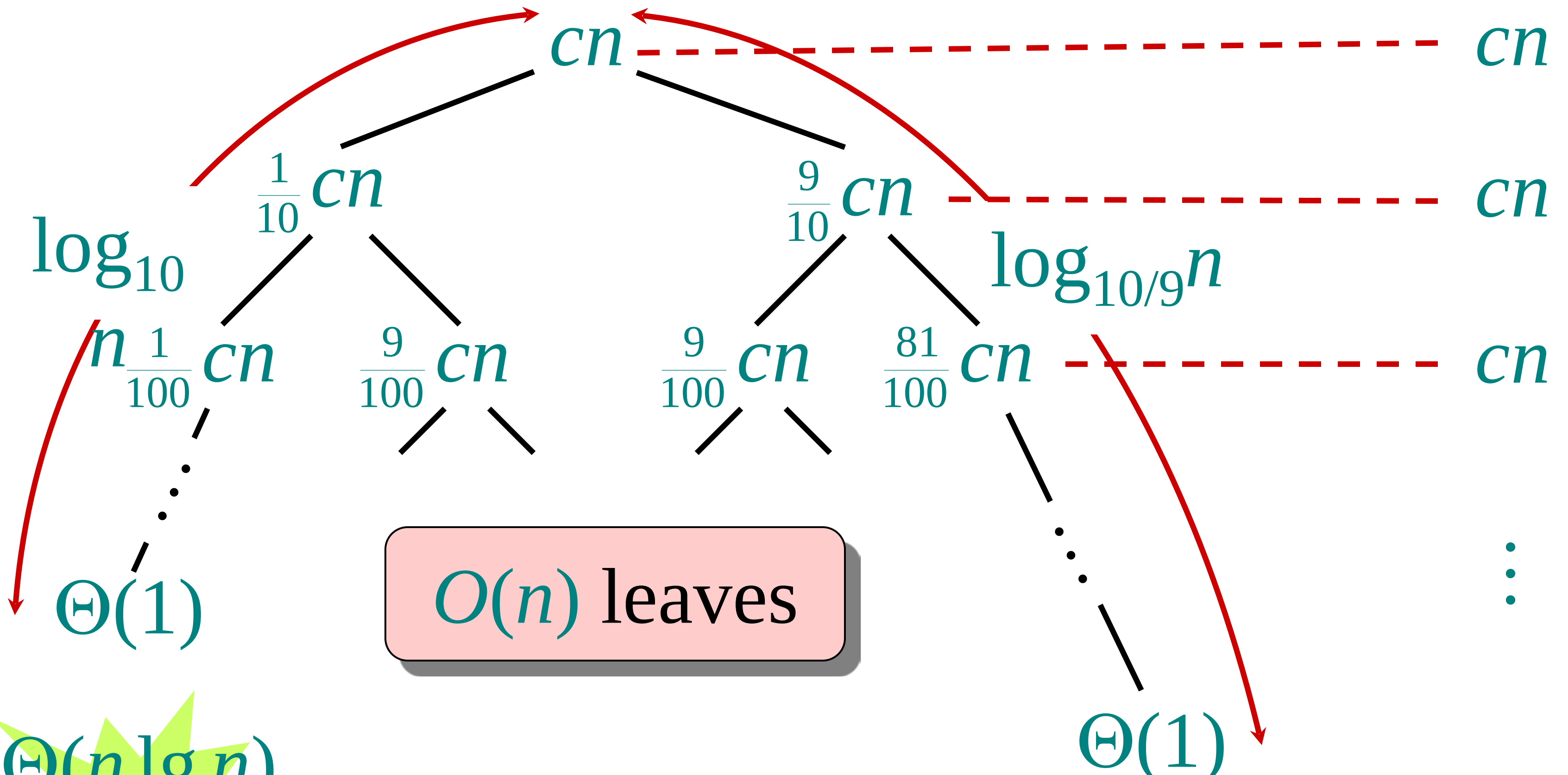


Analysis of "almost-best" case





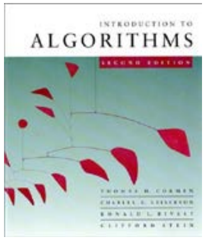
Analysis of "almost-best" case



$\Theta(n \lg n)$

Lucky!

$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n + O(n)$$



More intuition

Suppose we alternate lucky, unlucky, lucky, unlucky, lucky, ….

$L(n) = 2U(n/2) + \Theta(n)$ ***lucky***

$U(n) = L(n-1) + \Theta(n)$ ***unlucky***

Solving:

$$\begin{aligned} L(n) &= 2(L(n/2-1) + \Theta(n/2)) + \Theta(n) \\ &= 2L(n/2-1) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

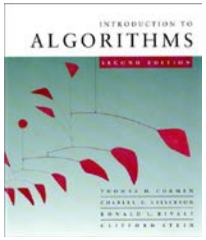
Lucky!

How can we make sure we are usually lucky?

Randomized quicksort

IDEA: Partition around a ***random*** element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.



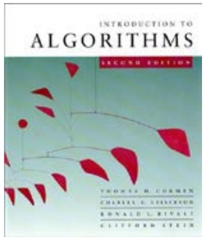
Randomized quicksort analysis

Let $T(n)$ = the random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent.

For $k = 0, 1, \ldots, n-1$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

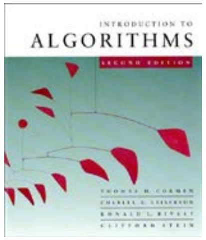
$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.



Analysis (continued)

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

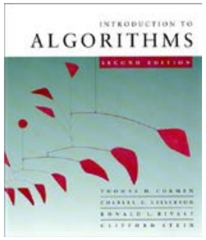
$$= \sum_{k=0}^{n-1} X_k \big(T(k) + T(n-k-1) + \Theta(n) \big)$$



Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Take expectations of both sides.

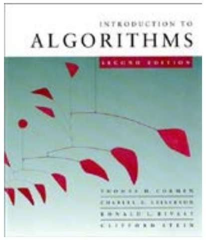


Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\big(T(k)+T(n-k-1)+\Theta(n)\big)\right]$$

$$= \sum_{k=0}^{n-1} E\big[X_k\big(T(k)+T(n-k-1)+\Theta(n)\big)\big]$$

Linearity of expectation.



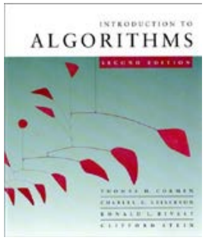
Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

Independence of X_k from other random choices.



Calculating expectation

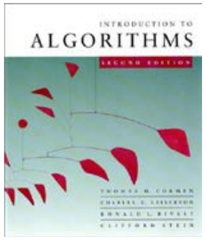
$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \big(T(k) + T(n-k-1) + \Theta(n)\big)\right]$$

$$= \sum_{k=0}^{n-1} E\big[X_k \big(T(k) + T(n-k-1) + \Theta(n)\big)\big]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)$$

Linearity of expectation; $E[X_k] = 1/n$.



Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\left(T(k)+T(n-k-1)+\Theta(n)\right)\right]$$

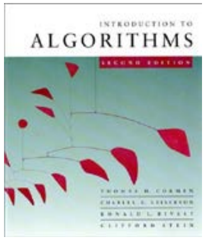
$$= \sum_{k=0}^{n-1} E\left[X_k\left(T(k)+T(n-k-1)+\Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k]\cdot E[T(k)+T(n-k-1)+\Theta(n)]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1}\Theta(n)$$

$$= \frac{2}{n}\sum_{k=1}^{n-1} E[T(k)] + \Theta(n)$$

Summations have identical terms.



Hairy recurrence

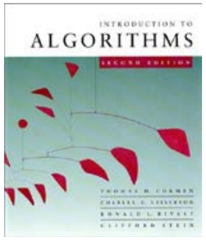
$$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(The $k = 0, 1$ terms can be absorbed in the $\Theta(n)$.)

Prove: $E[T(n)] \le a n \lg n$ for constant $a > 0$.

- Choose a large enough so that $a n \lg n$ dominates $E[T(n)]$ for sufficiently small $n \ge 2$.

Use fact: $\displaystyle\sum_{k=2}^{n-1} k \lg k \le \tfrac{1}{2} n^2 \lg n - \tfrac{1}{8} n^2$ (exercise).



Substitution method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n)$$

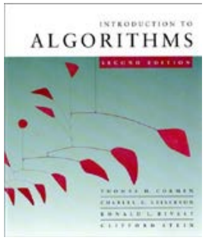
Substitute inductive hypothesis.

Substitution method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n)$$

$$\le \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

Use fact.



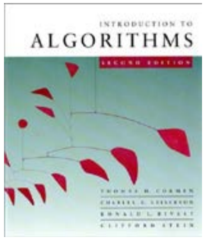
Substitution method

$$E[T(n)] \le \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$\le \frac{2a}{n} \left(\frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \Theta(n)$$

$$= an \lg n - \left(\frac{an}{4} - \Theta(n) \right)$$

Express as ***desired*** **–** ***residual***.



Substitution method

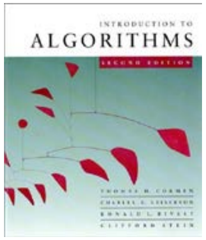
$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

$$= \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

$$= an \lg n - \left(\frac{an}{4} - \Theta(n)\right)$$

$$\le an \lg n \,,$$

if a is chosen large enough so that $an/4$ dominates the $\Theta(n)$.



Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from ***code tuning***.
- Quicksort behaves well even with caching and virtual memory.

