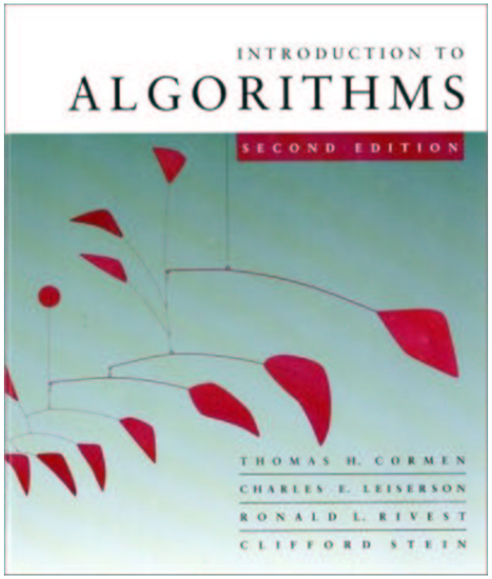
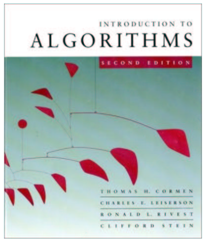
Introduction to Algorithms

6.046J/18.401J



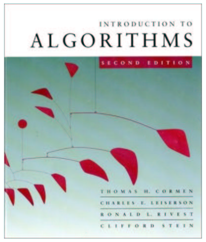
Lecture 4

Prof. Piotr Indyk



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

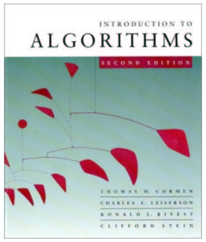
Quicksort an n-element array:

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.*



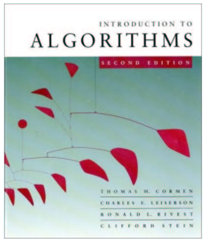
Partitioning subroutine

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

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

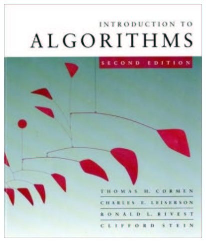
Invariant:

x	$\le x$	$\ge x$	?
p	i	j	q



Example of partitioning

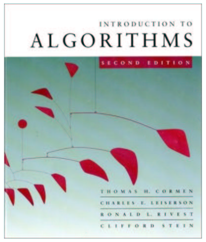
6	10	13	5	8	3	2	11
i	j						



Example of partitioning

6	10	13	5	8	3	2	11

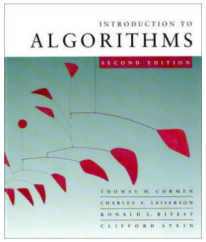
i $\longrightarrow$ j



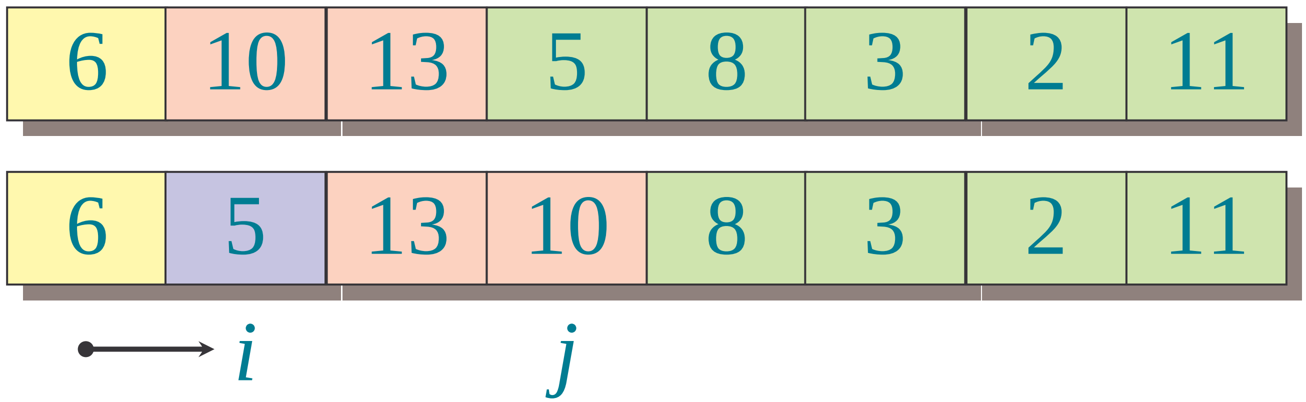
Example of partitioning

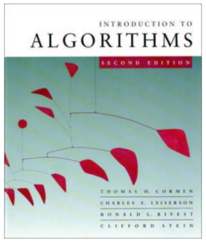
6	10	13	5	8	3	2	11

i $\quad\quad\quad \longrightarrow j$

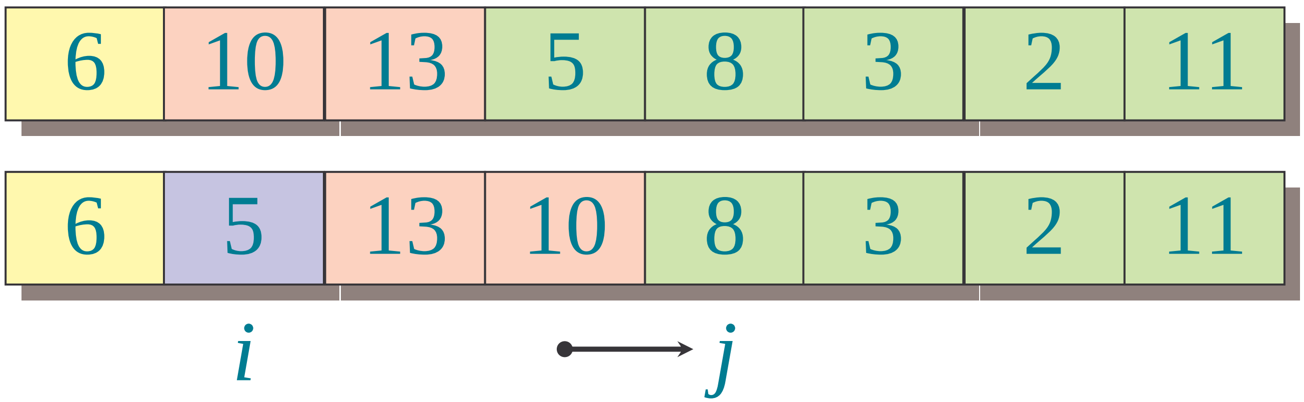


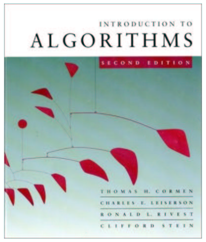
Example of partitioning





Example of partitioning





Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11
	i				j		

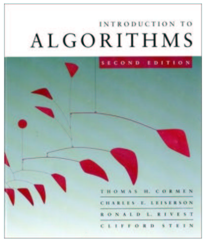
Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

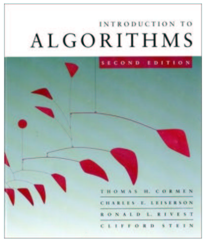
6	5	3	10	8	13	2	11

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



Example of partitioning

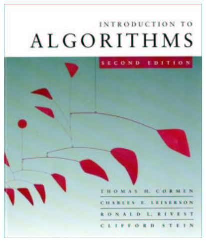
6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
		i				$\longrightarrow j$	



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

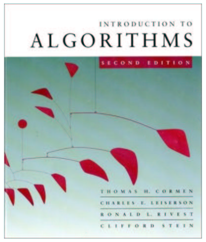
$\longrightarrow i$ $\qquad\qquad j$



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 $\longrightarrow j$



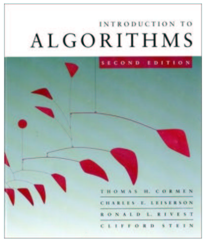
Example of partitioning

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

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				



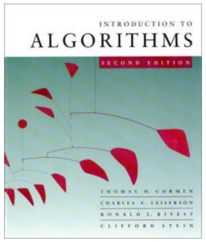
Pseudocode for quicksort

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.



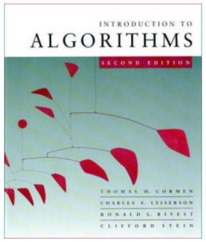
Worst-case of quicksort

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

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

Worst-case recursion tree

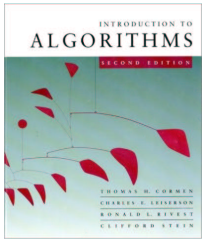
$$T(n) = T(0) + T(n–1) + cn$$



Worst-case recursion tree

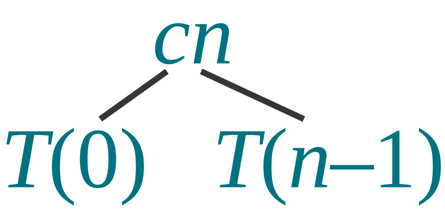
$$T(n) = T(0) + T(n–1) + cn$$

$T(n)$



Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



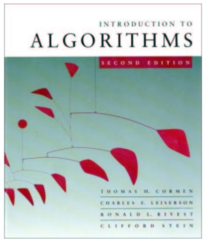
Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

$$cn$$

$$T(0) \quad c(n–1)$$

$$T(0) \quad T(n–2)$$



Worst-case recursion tree

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

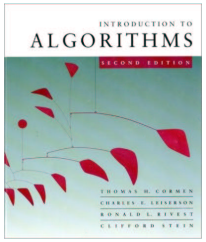
$$cn$$

$$T(0) \quad c(n-1)$$

$$T(0) \quad c(n-2)$$

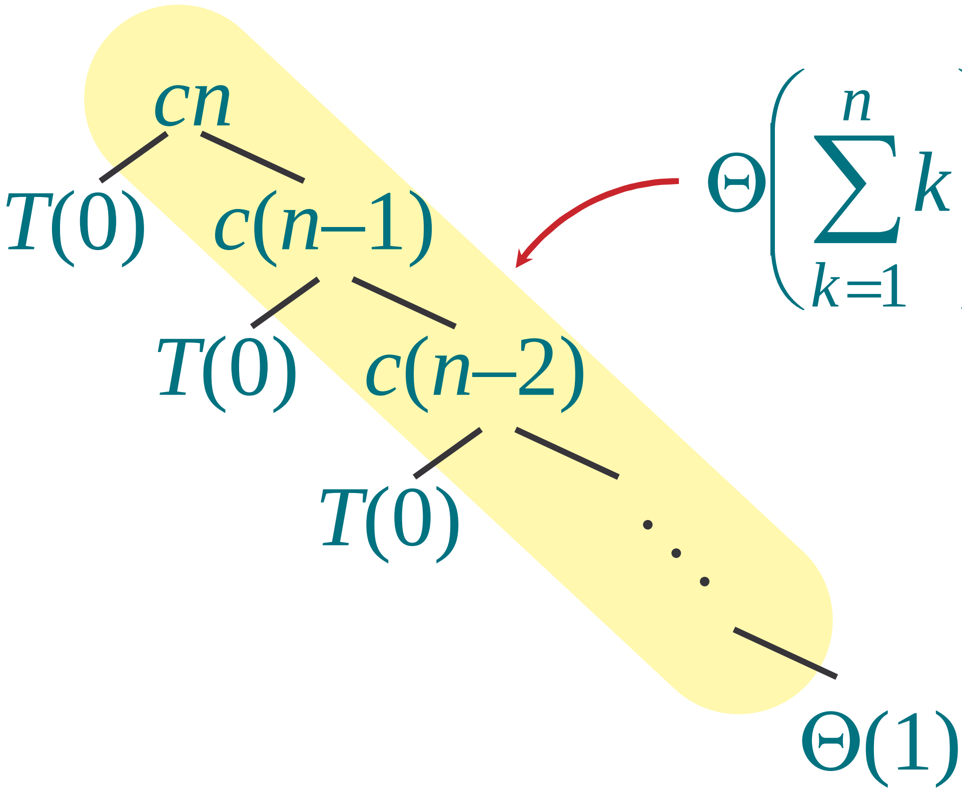
$$T(0) \quad \ddots$$

$$\Theta(1)$$

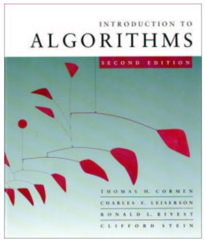


Worst-case recursion tree

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

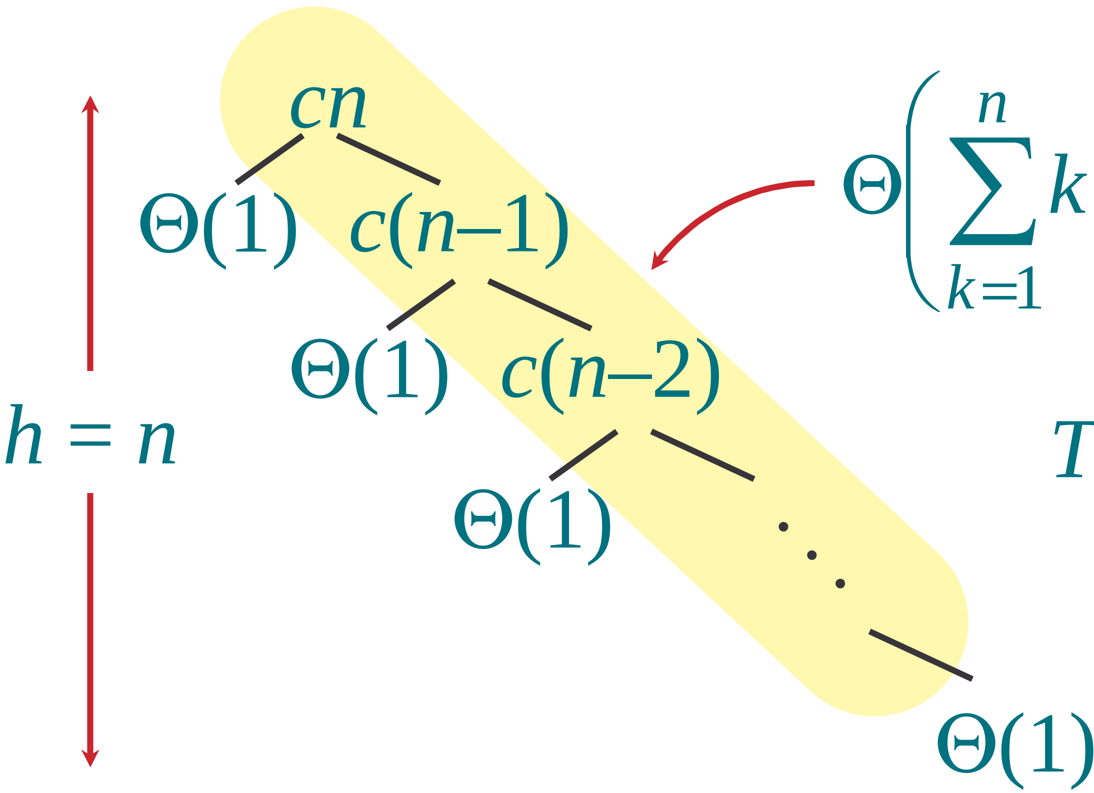


$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$



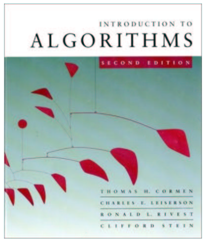
Worst-case recursion tree

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



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

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



Best-case analysis

(For intuition only!)

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

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \qquad \text{(same as merge sort)} \end{aligned}$$

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

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

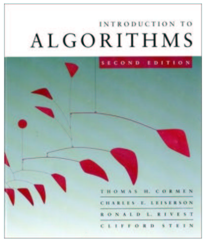
What is the solution to this recurrence?

Analysis of "almost-best" case

$$T(n)$$

Analysis of “almost-best” case

$$
\begin{array}{ccc}
 & cn & \\
\swarrow & & \searrow \\
T\left(\frac{1}{10}n\right) & & T\left(\frac{9}{10}n\right)
\end{array}
$$

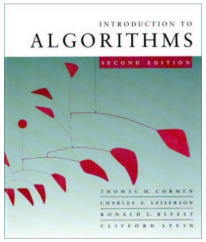


Analysis of “almost-best” case

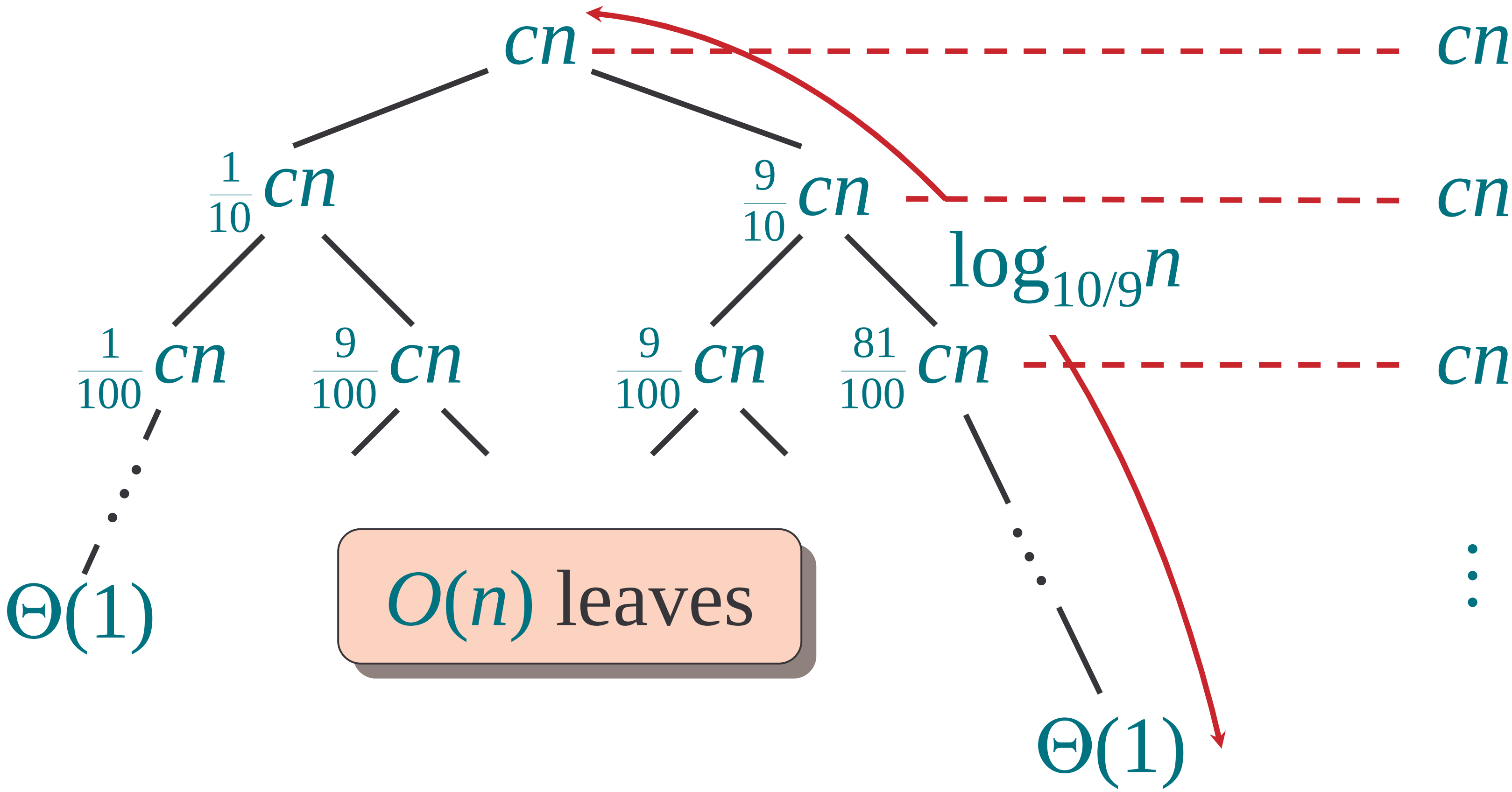
$$cn$$

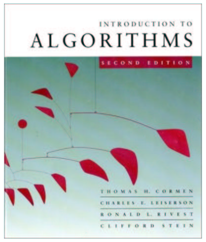
$$\frac{1}{10}cn \qquad \frac{9}{10}cn$$

$$T\left(\frac{1}{100}n\right) \quad T\left(\frac{9}{100}n\right) \qquad T\left(\frac{9}{100}n\right) \quad T\left(\frac{81}{100}n\right)$$

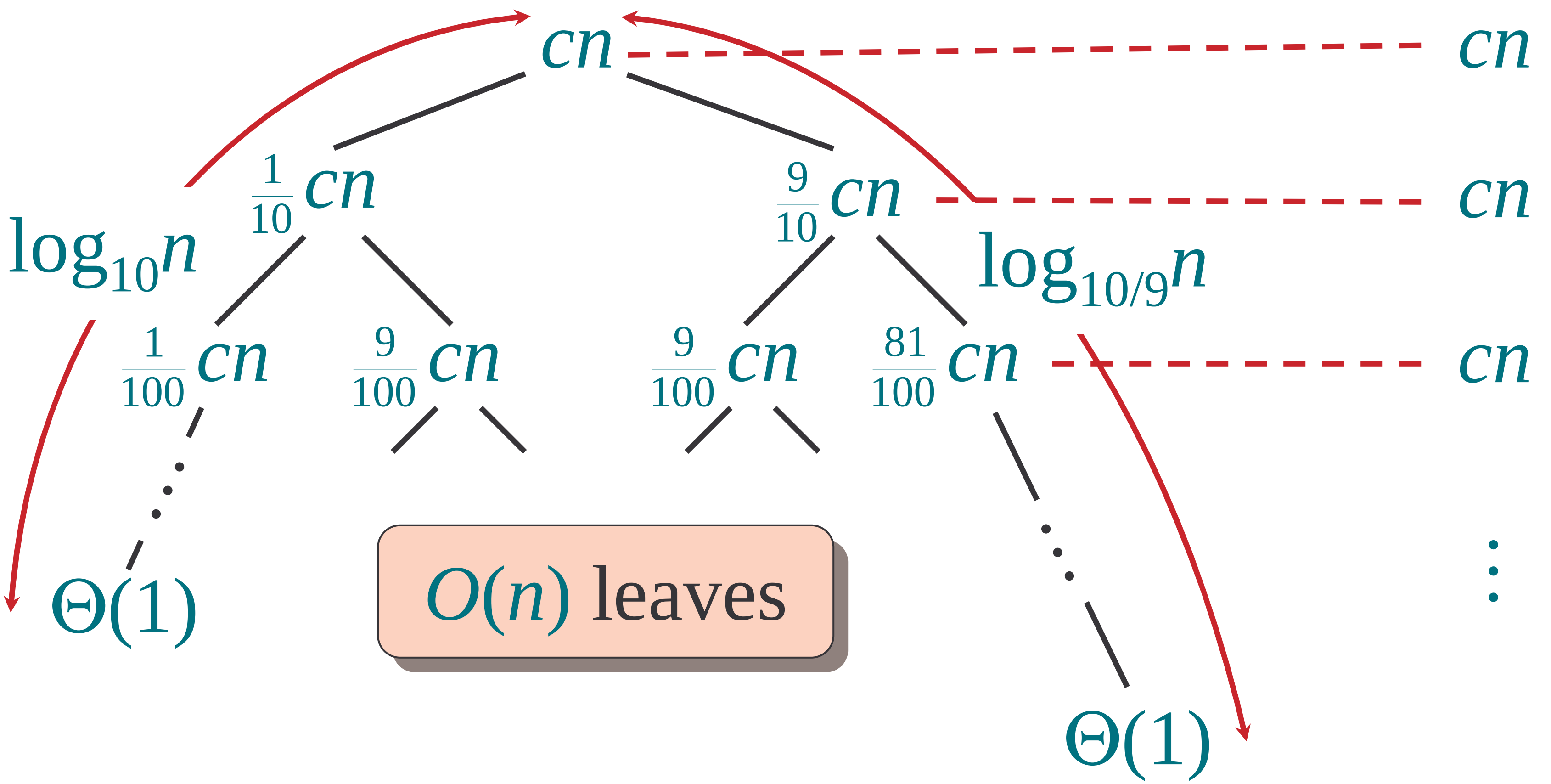


Analysis of "almost-best" case

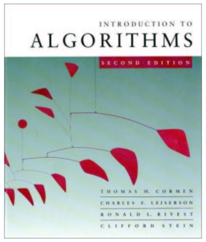




Analysis of "almost-best" case

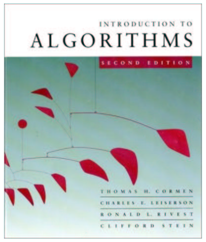


$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n + O(n)$$



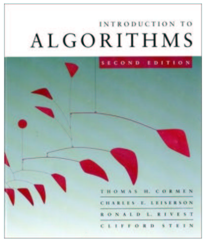
Randomized quicksort

IDEA: Partition around a ***random*** element.
I.e., around *A[t]*, where *t* chosen uniformly at random from *{p…r}*



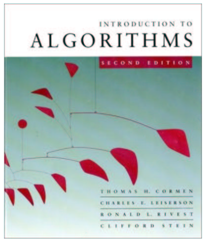
Randomized Algorithms

- Algorithms that make decisions based on random coin flips.
- Can "fool" the adversary.
- The running time (or even correctness) is a random variable; we measure the ***expected*** running time.
- We assume all random choices are ***independent*** .
- This is ***not*** the average case !



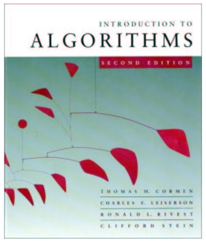
“Paranoid” quicksort

- Will modify the algorithm to make it easier to analyze:
  - Repeat:
    - Choose the pivot at random
    - Perform PARTITION
  - Until the resulting split is lucky, i.e., not worse than 1/10: 9/10
  - Recurse on both subarrays



Analysis

- Let $T(n)$ be an upper bound on the ***expected*** running time on any array of n elements
- Consider any input of size n
- The time needed to sort the input is bounded from the above by a sum of
  - The time needed to sort the left subarray
  - The time needed to sort the right subarray
  - The number of iterations until we get a lucky split, times cn



Expectations

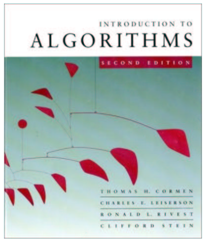
- By linearity of expectation:

$$T(n) \leq \max T(i) + T(n-i) + E[\#\ partitions] \bullet cn$$

where maximum is taken over $i \in [n/10, 9n/10]$

- We will show that $E[\#partitions]$ is less than 2
- Therefore:

$$T(n) \leq \max T(i) + T(n-i) + 2cn, i \in [n/10, 9n/10]$$



Final bound

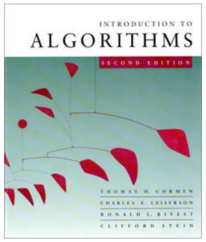
- Can use the recursion tree argument:
  - Tree depth is $\Theta(\log n)$
  - Total work at each level is at most $2cn$
  - The total expected time is $O(n \log n)$

Lucky partitions

- The probability that a random pivot induces lucky partition is at least *8/10*

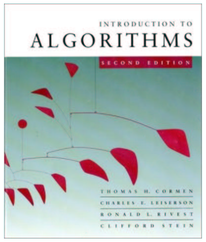
  (we are ***not*** lucky if the pivot happens to be among the smallest/largest *n/10* elements)

- If we flip a coin, with heads prob. $p=8/10$, the expected waiting time for the first head is $1/p = 10/8 < 2$



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.



More intuition

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

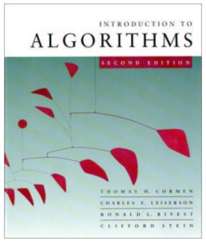
$$L(n) = 2U(n/2) + \Theta(n) \quad \textit{lucky}$$
$$U(n) = L(n-1) + \Theta(n) \quad \textit{unlucky}$$

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?



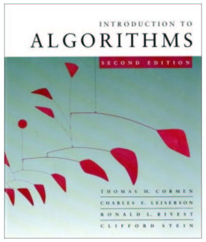
Randomized quicksort analysis

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***

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

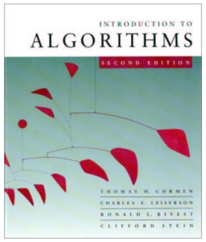
$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.



Analysis (continued)

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

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



Calculating expectation

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

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.

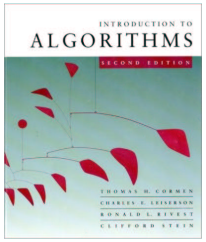
Calculating expectation

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

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

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

Independence of X_k from other random choices.



Calculating expectation

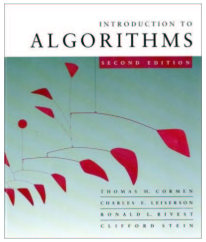
$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \big(T(k) + T(n-k-1) + \Theta(n)\big)\right]$$

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

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

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

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

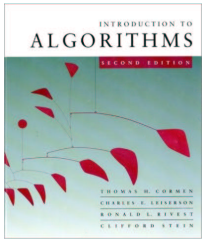
$$E[T(n)] = \frac{2}{n} \sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(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 $n \ge 2$.

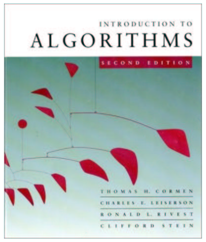
Use fact: $\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$ (exercise).



Substitution method

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

Substitute inductive hypothesis.

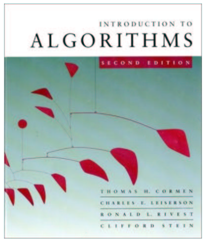


Substitution method

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

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

Use fact.



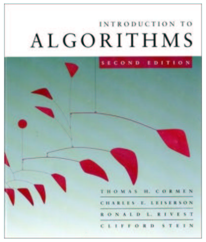
Substitution method

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

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

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

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 $\Theta(n)$.

- Assume