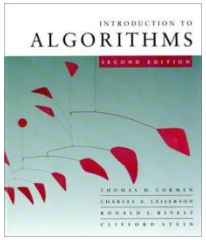
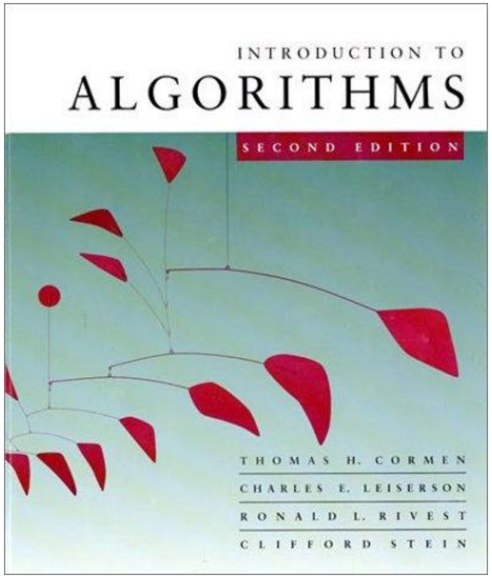


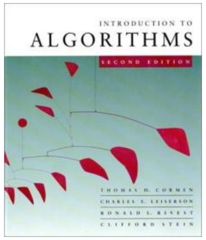
CS 5633 – Spring 2012



Randomized Algorithms & Quicksort

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk



Deterministic Algorithms

Runtime for deterministic algorithms with input size *n*:

- Best-case runtime

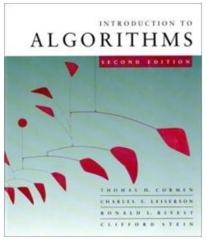
  ➔ Attained by one input of size *n*

- Worst-case runtime

  ➔ Attained by one input of size *n*

- Average runtime

  ➔ Averaged **over all possible inputs** of size *n*



Deterministic Algorithms: Insertion Sort

Best-case runtime: $O(n)$, input $[1,2,3,\ldots,n]$

➔ Attained by one input of size n

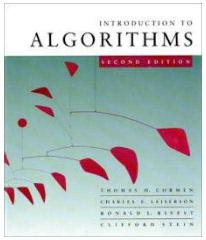
- Worst-case runtime: $O(n^2)$, input $[n, n\text{-}1, \ldots,2,1]$

➔ Attained by one input of size n

- Average runtime : $O(n^2)$; see book for analysis

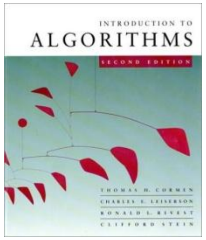
➔ Averaged **over all possible inputs** of size n

- What kind of inputs are there?
- How many inputs are there?



Average Runtime

- What kind of inputs are there?
  - Do [1,2,…,n] and [5,6,…,n+5] cause different behavior of Insertion Sort?
  - No. Therefore it suffices to only consider all permutations of [1,2,…,n] .
- How many inputs are there?
  - There are n! different permutations of [1,2,…,n]



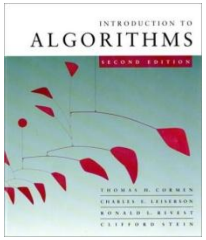
Average Runtime Insertion Sort: *n*=4

```
for j=2 to n {
    key = A[j]
    // insert A[j] into sorted sequen
    i=j-1
    while(i>0 && A[i]>key){
        A[i+1]=A[i]
        i--
    }
    A[i+1]=key
}
```

- Inputs: 4!=24

[1,2,3,4] **0**	[4,1,2,3] **3**	[4,1,3,2] **4**	[4,3,2,1] **6**
[2,1,3,4] **1**	[1,4,2,3] **2**	[1,4,3,2] **3**	[3,4,2,1] **5**
[1,3,2,4] **1**	[1,2,4,3] **1**	[1,3,4,2] **2**	[3,2,4,1] **4**
[3,1,2,4] **2**	[4,2,1,3] **4**	[4,3,1,2] **5**	[4,2,3,1] **5**
[3,2,1,4] **3**	[2,1,4,3] **2**	[3,4,1,2] **4**	[2,4,3,1] **4**
[2,3,1,4] **2**	[2,4,1,3] **3**	[3,1,4,2] **3**	[2,3,4,1] **3**

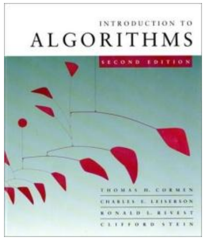
- Runtime is proportional to: 3 + **#times in while loop**
- Best: 3+**0**, Worst: 3+**6**=9, Average: 3+**72**/24 = 6



Average Runtime: Insertion Sort

- The average runtime averages runtimes over all $n!$ different input permutations
- Disadvantage of considering average runtime:
  - There are still worst-case inputs that will have the worst-case runtime
  - Are all inputs really equally likely? That depends on the application

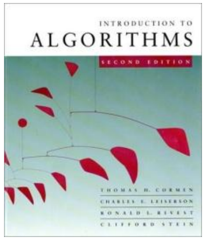
$\Rightarrow$ **Better:** Use a randomized algorithm



Randomized Algorithm: Insertion Sort

- **Randomize the order of the input array:**
  - Either prior to calling insertion sort,
  - or during insertion sort (insert random element)
- This makes the runtime depend on a probabilistic experiment (sequence of numbers obtained from random number generator)

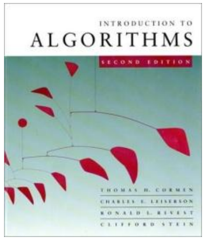
  $\Rightarrow$Runtime is a random variable (maps sequence of random numbers to runtimes)
- **Expected runtime** = expected value of runtime random variable



Randomized Algorithm: Insertion Sort

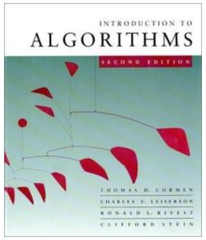
- Runtime is independent of input order ([1,2,3,4] may have good or bad runtime, depending on sequence of random numbers)
- No assumptions need to be made about input distribution
- No one specific input elicits worst-case behavior
- The worst case is determined only by the output of a random-number generator.

$\Rightarrow$ When possible use expected runtimes of randomized algorithms instead of average case analysis of deterministic algorithms



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).
- We are going to perform an expected runtime analysis on randomized quicksort



Quicksort: Divide and conquer

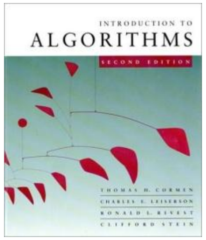
Quicksort an n-element array:

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.*



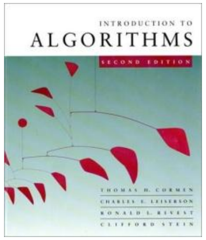
Partitioning subroutine

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.

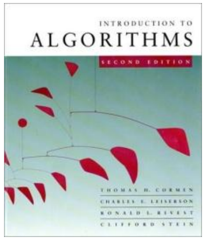
Invariant:

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



Example of partitioning

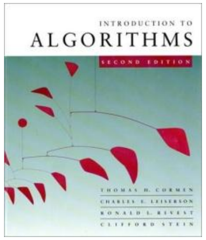
6	10	13	5	8	3	2	11
i	*j*						



Example of partitioning

6	10	13	5	8	3	2	11

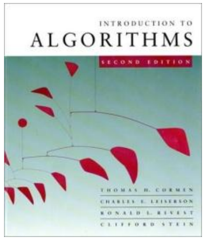
i $\longrightarrow j$



Example of partitioning

6	10	13	5	8	3	2	11

i $\longrightarrow$ j

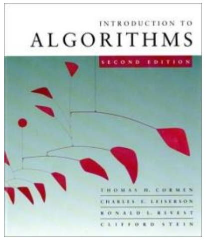


Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

i j

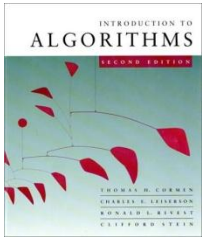


Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

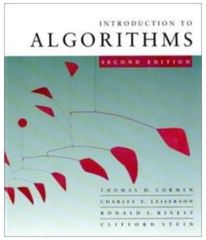
i $\longrightarrow j$



Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11
	i				j		



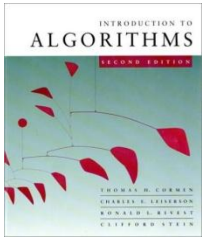
Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

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



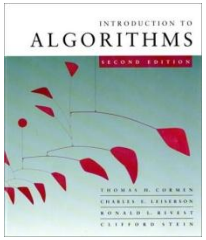
Example of partitioning

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

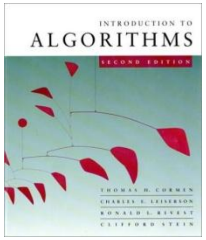
6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

6	5	3	2	8	13	10	11

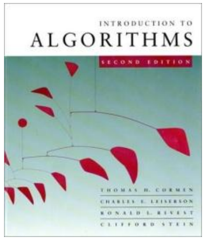
$\longrightarrow i$ $\qquad\qquad j$



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

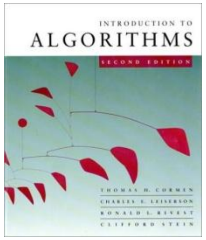
i *j*



Example of partitioning

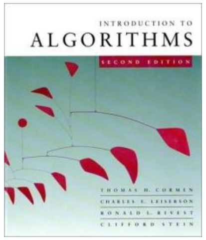
6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11

i $\longrightarrow j$



Example of partitioning

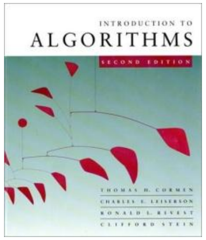
6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
2	5	3	6	8	13	10	11
			i				



Pseudocode for quicksort

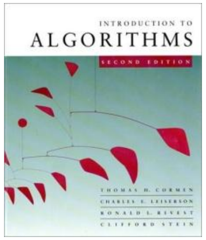
QUICKSORT(A, p, r)
    if $p < r$
        then $q \leftarrow$ PARTITION(A, p, r)
            QUICKSORT$(A, p, q-1)$
            QUICKSORT$(A, q+1, r)$

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.

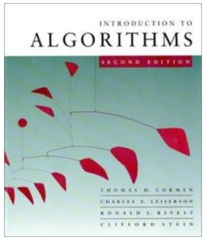


Worst-case of quicksort

```
QUICKSORT(A, p, r)
    if p < r
        then q ← PARTITION(A, p, r)
            QUICKSORT(A, p, q–1)
            QUICKSORT(A, q+1, r)
```

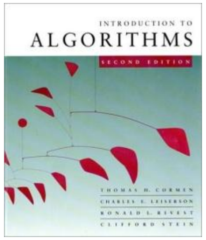
- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

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



Worst-case recursion tree

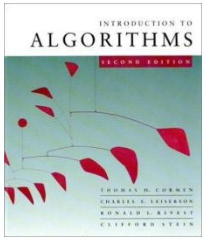
$$T(n) = T(0) + T(n-1) + cn$$



Worst-case recursion tree

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

$T(n)$

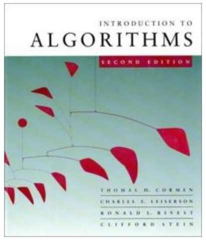


Worst-case recursion tree

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

$$cn$$

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



Worst-case recursion tree

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

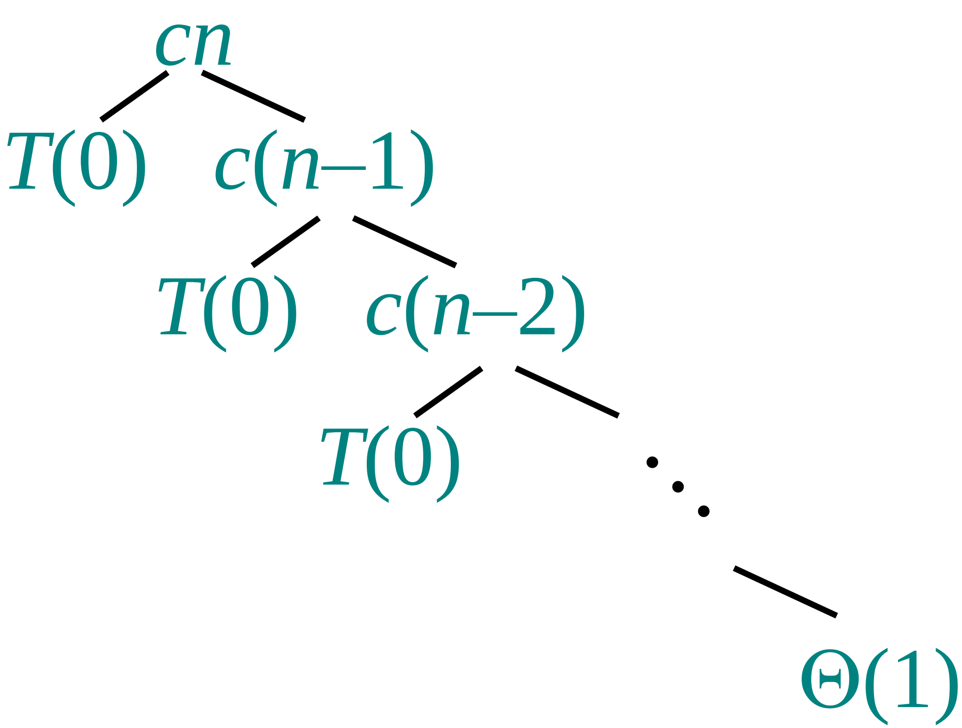
$$cn$$

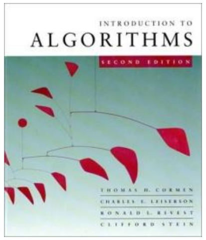
$$T(0) \quad c(n-1)$$

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

Worst-case recursion tree

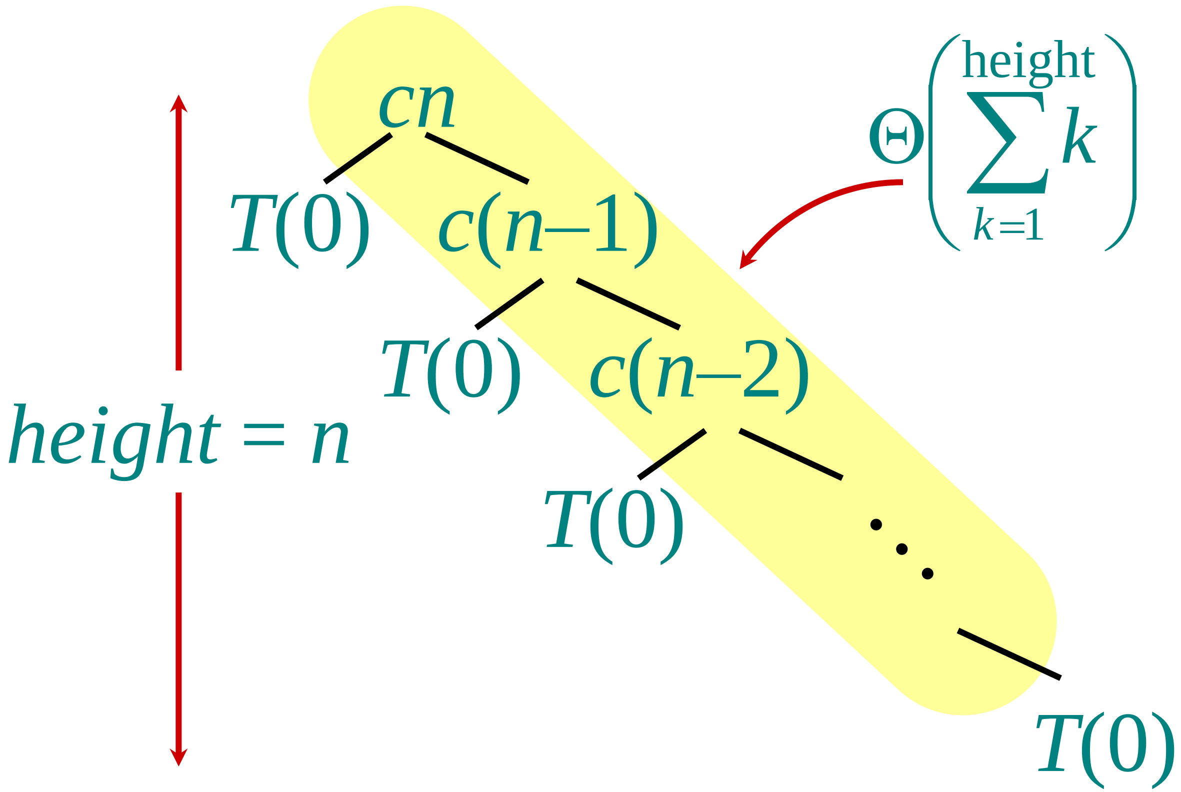
$$T(n) = T(0) + T(n-1) + cn$$

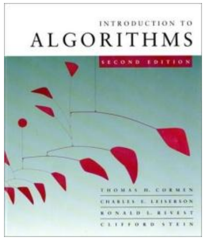




Worst-case recursion tree

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

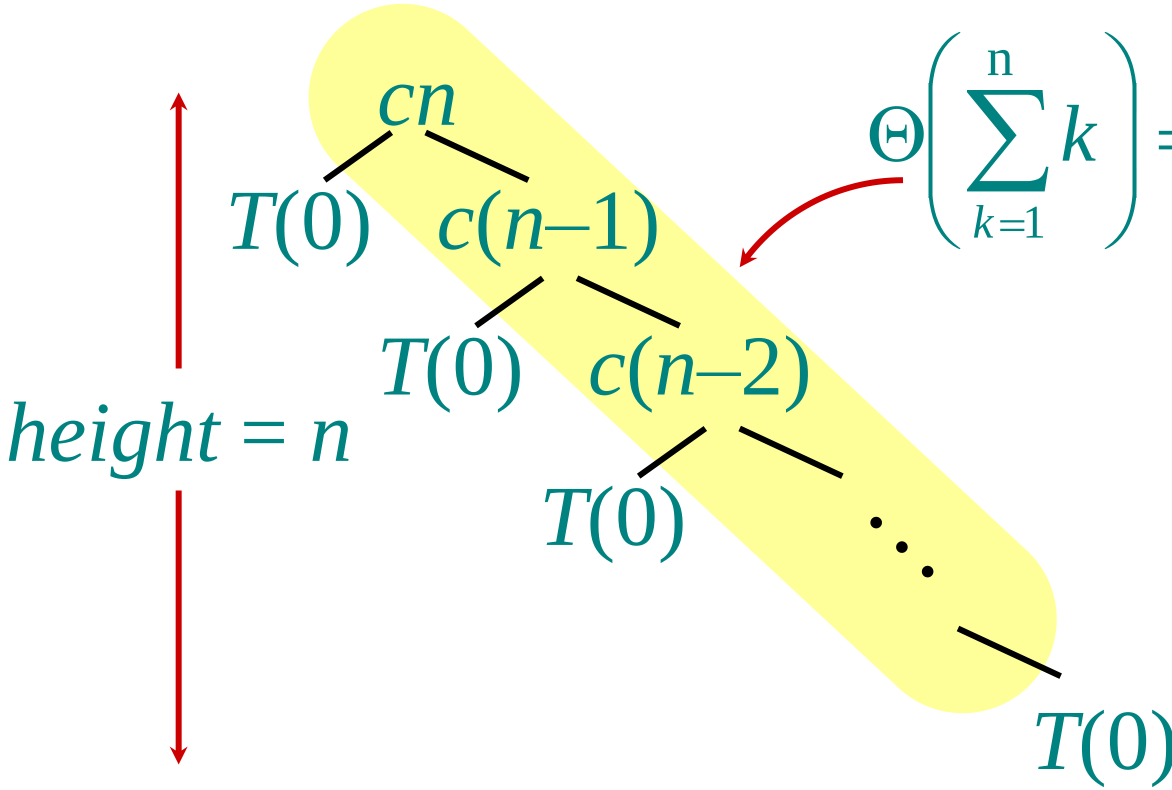


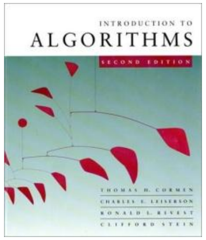


Worst-case recursion tree

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

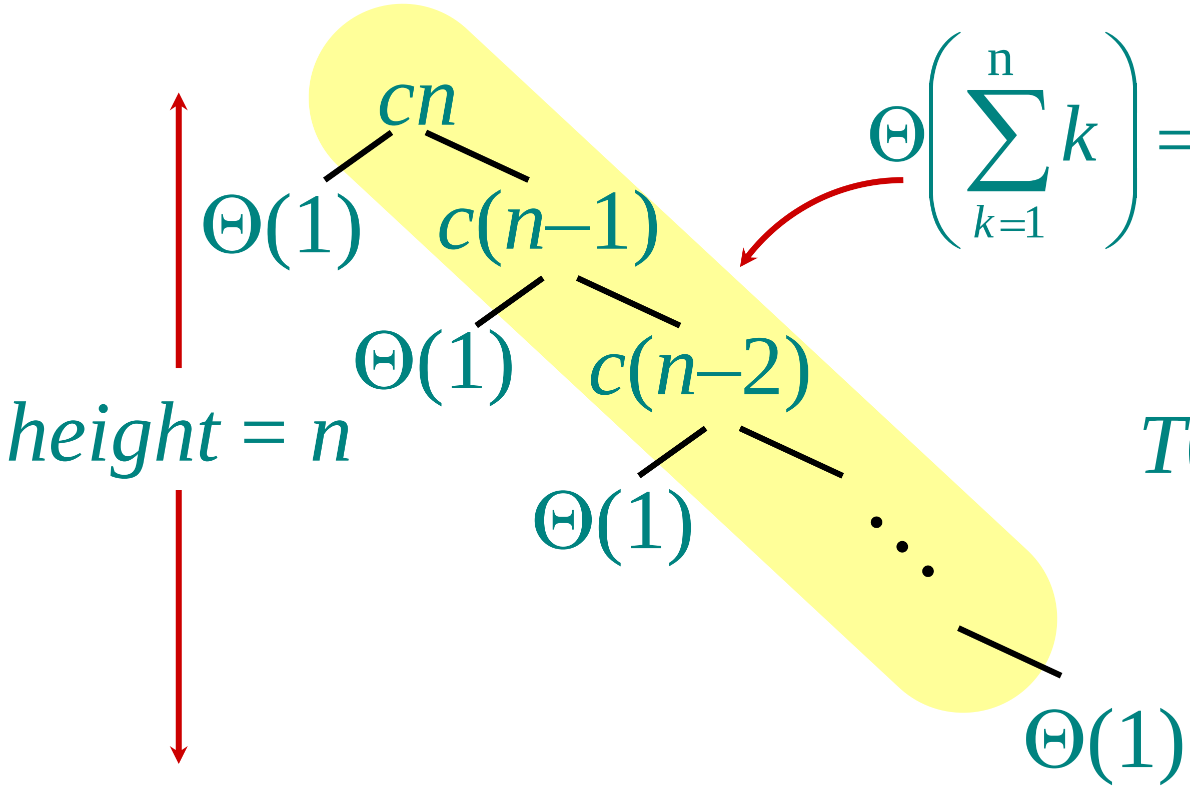
$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$





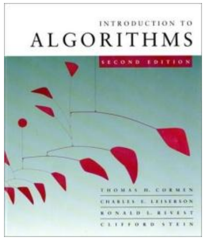
Worst-case recursion tree

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



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

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



Best-case analysis

(For intuition only!)

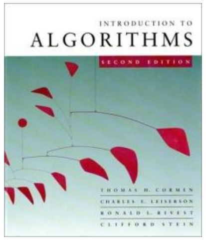
If we're lucky, PARTITION splits the array evenly:

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

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

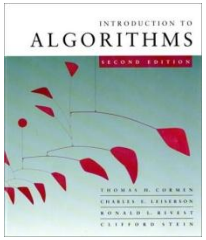
$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?



Analysis of “almost-best” case

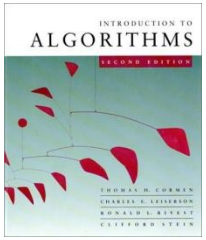
$$T(n)$$



Analysis of “almost-best” case

$$cn$$

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

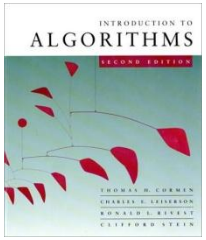


Analysis of "almost-best" case

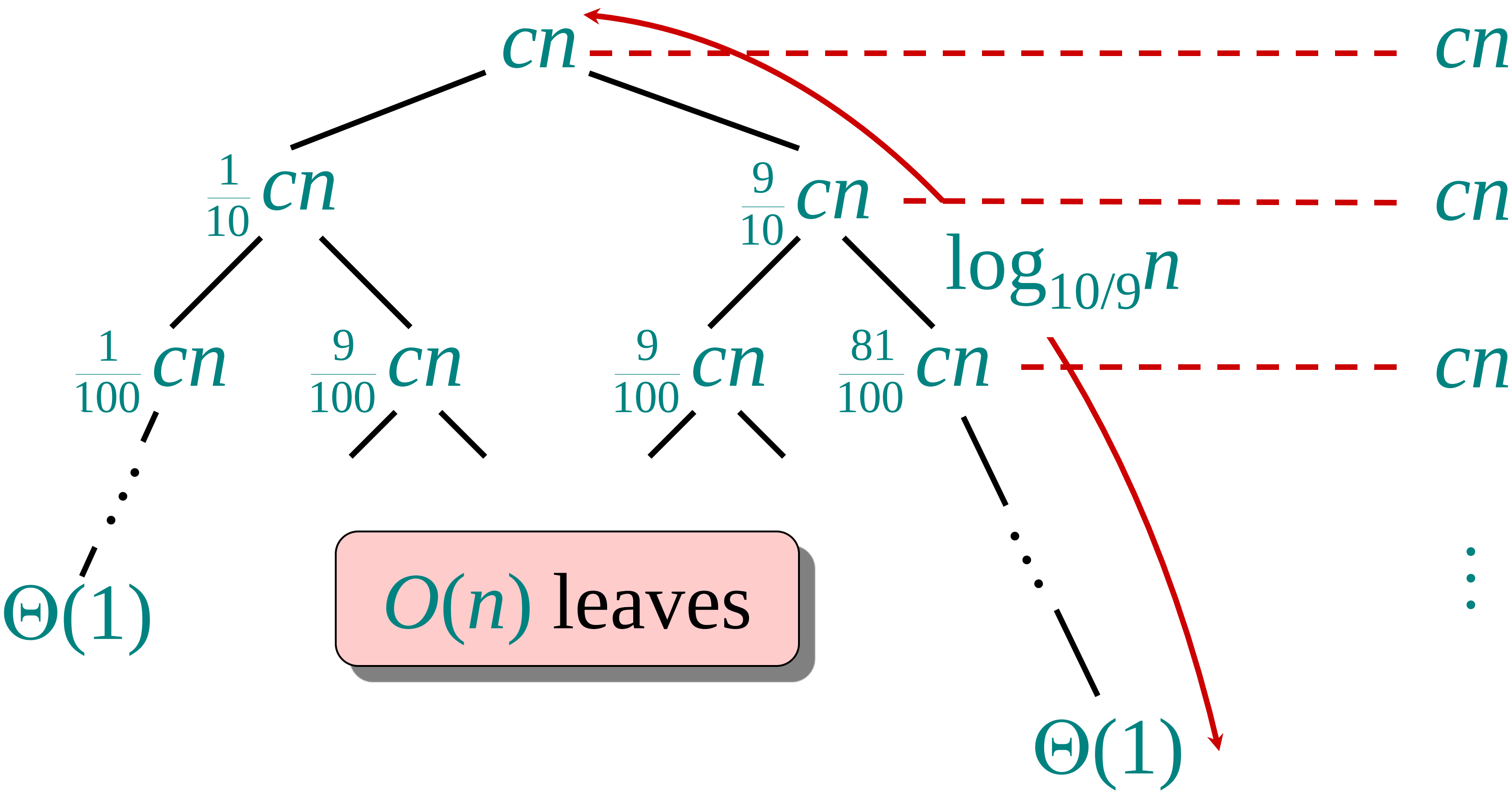
$$cn$$

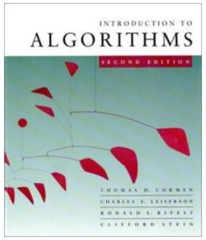
$$\frac{1}{10}cn \qquad \frac{9}{10}cn$$

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

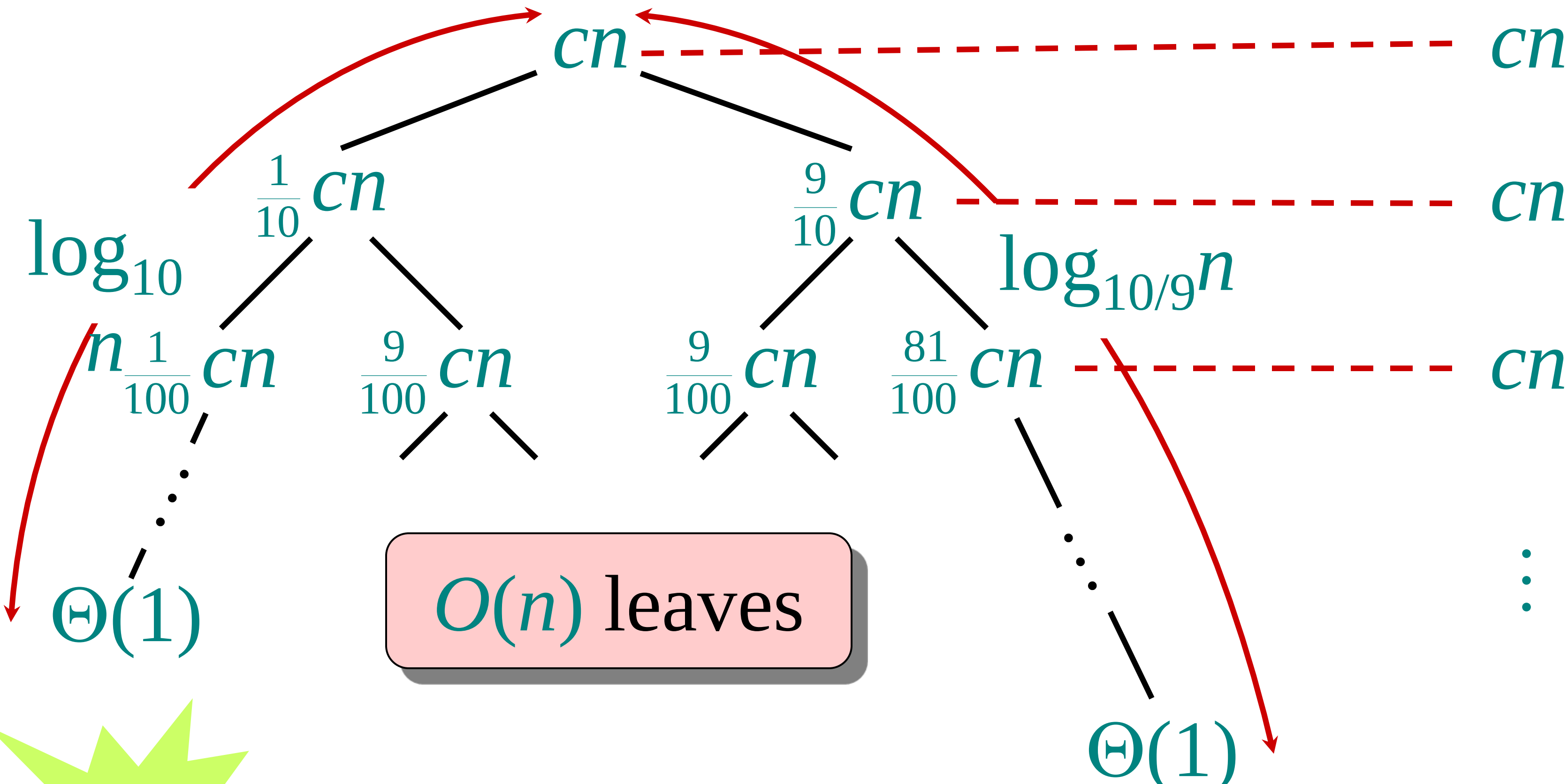


Analysis of “almost-best” case



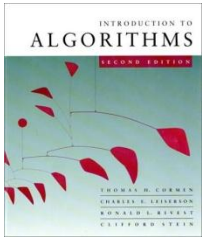


Analysis of "almost-best" case



$\Theta(n \log n)$

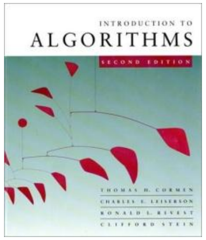
$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n + O(n)$$



Quicksort Runtimes

- Best case runtime $T_{\text{best}}(n) \in O(n \log n)$
- Worst case runtime $T_{\text{worst}}(n) \in O(n^2)$

- Worse than mergesort? Why is it called quicksort then?
- Its average runtime $T_{\text{avg}}(n) \in O(n \log n)$
- Better even, the expected runtime of **randomized quicksort** is $O(n \log n)$

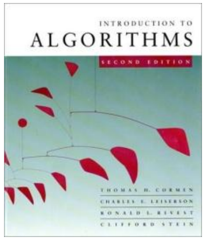


Average Runtime

The **average runtime** $T_{avg}(n)$ for Quicksort is the average runtime over **all possible inputs** of length n.

- $T_{avg}(n)$ has to average the runtimes over all n! different input permutations.
- There are still worst-case inputs that will have a $O(n^2)$ runtime

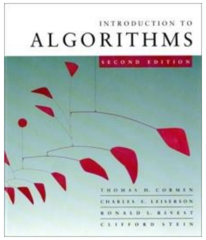
⇒ **Better:** Use randomized quicksort



Randomized quicksort

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

- Running time is independent of the input order. It depends only on the sequence s of random numbers.
- 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 sequence s of random numbers.

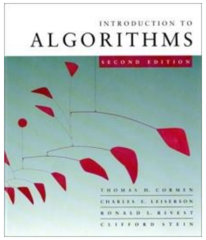


Randomized quicksort analysis

- $T(n,s)$ = random variable for the running time of randomized quicksort on an input of size n, with sequence s of random numbers which are assumed to be independent.

- $E(T(n))$ = expected value of $T(n,s)$, the "expected runtime" of randomized quicksort.

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

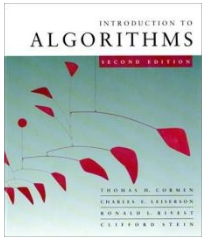


Randomized quicksort analysis

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

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

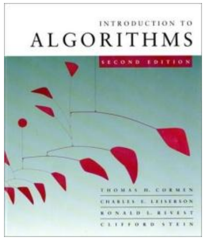
$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.



Analysis (continued)

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

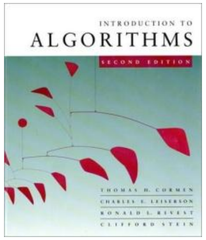
$$= \sum_{k=0}^{n-1} X_k(s)\big(T(k,s) + T(n-k-1,s) + \Theta(n)\big)$$



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.

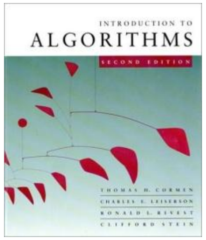


Calculating expectation

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

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

Linearity of expectation.



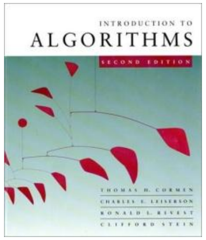
Calculating expectation

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

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

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

Independence of X_k from other random choices.



Calculating expectation

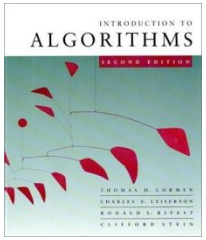
$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\big(T(k)+T(n-k-1)+\Theta(n)\big)\right]$$

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

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

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

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



Calculating expectation

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

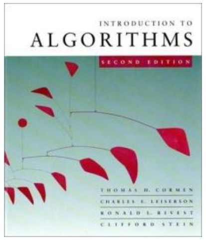
$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

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

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

$$= \frac{2}{n}\sum_{k=0}^{n-1} E[T(k)] + \Theta(n)$$

Summations have identical terms.



Hairy recurrence

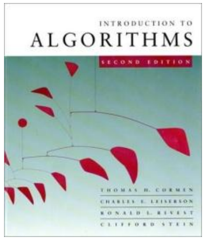
$$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

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

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

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

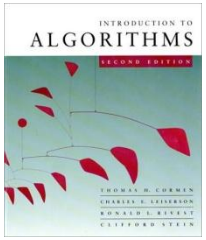
Use fact: $\sum_{k=2}^{n-1} k \log k \le \frac{1}{2} n^2 \log n - \frac{1}{8} n^2$ (exercise).



Substitution method

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

Substitute inductive hypothesis.

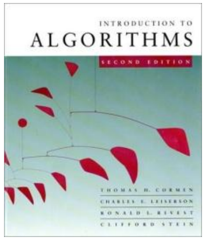


Substitution method

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

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

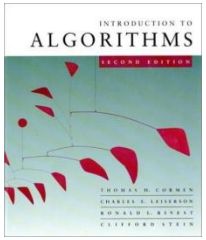
Use fact.



Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=2}^{n-1} ak\log k + \Theta(n) \\
&\le \frac{2a}{n}\left(\frac{1}{2}n^2\log n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an\log n - \left(\frac{an}{4} - \Theta(n)\right)
\end{aligned}
$$

Express as ***desired – residual***.

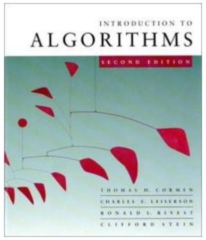


Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=2}^{n-1} ak\log k + \Theta(n) \\
&= \frac{2a}{n}\left(\frac{1}{2}n^2\log n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an\log n - \left(\frac{an}{4} - \Theta(n)\right) \\
&\le an\log n
\end{aligned}
$$

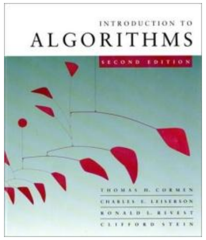
,

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.



Average Runtime vs. Expected Runtime

- Average runtime is averaged over all inputs of a deterministic algorithm.
- Expected runtime is the expected value of the runtime random variable of a randomized algorithm. It effectively "averages" over all sequences of random numbers.
- De facto both analyses are very similar. However in practice the randomized algorithm ensures that not one single input elicits worst case behavior.