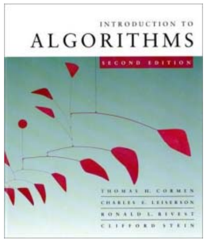
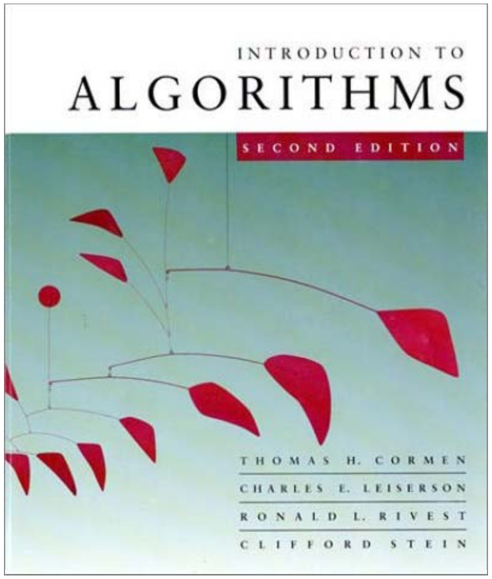


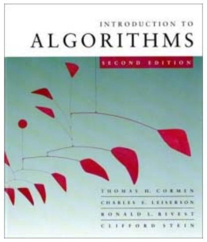
CS 3343 – Fall 2010



Quicksort

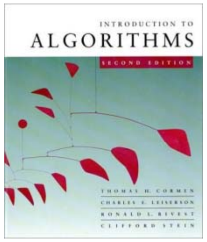
Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk



Quicksort

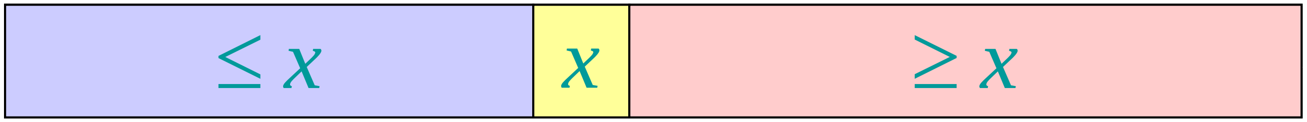
- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts "in place" (like insertion sort, but not like merge sort).
- Very practical (with tuning).



Divide and conquer

Quicksort an n-element array:

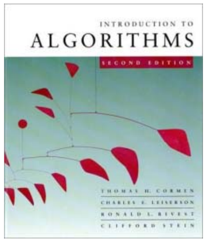
1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\le x \le$ elements in upper subarray.



$\le x$ x $\ge x$

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

Key: *Linear-time partitioning subroutine.*

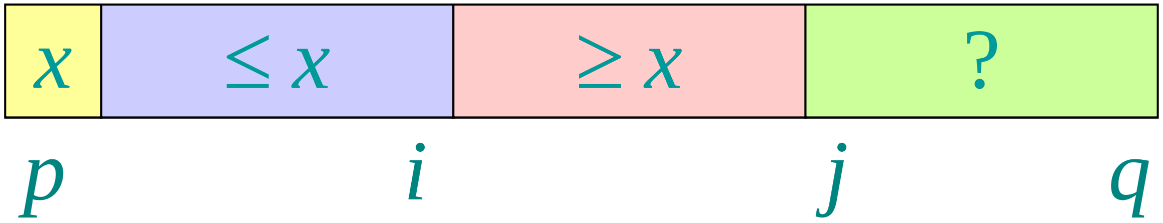


Partitioning subroutine

```
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 $i$
```

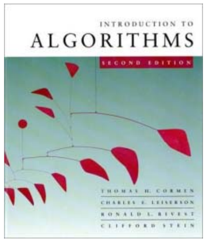
Running time $= O(n)$ for n elements.

Invariant:



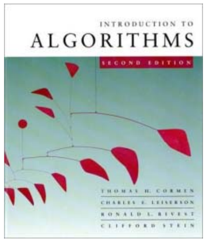
x	$\le x$	$\ge x$	?

p i j q

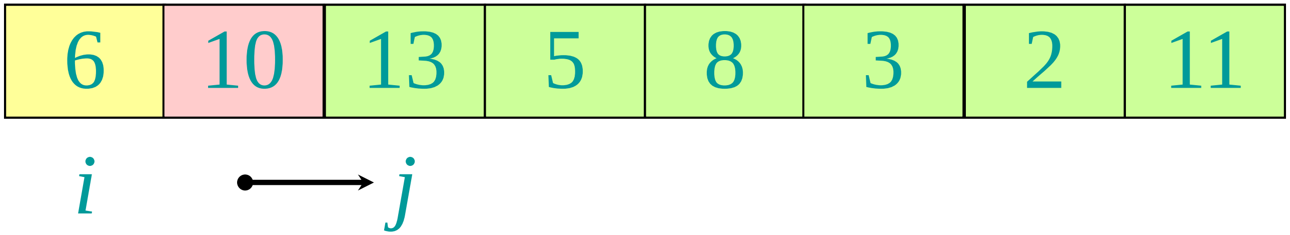


Example of partitioning

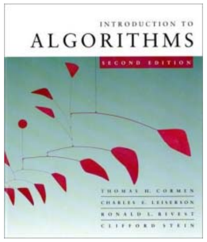
6	10	13	5	8	3	2	11
i	*j*						



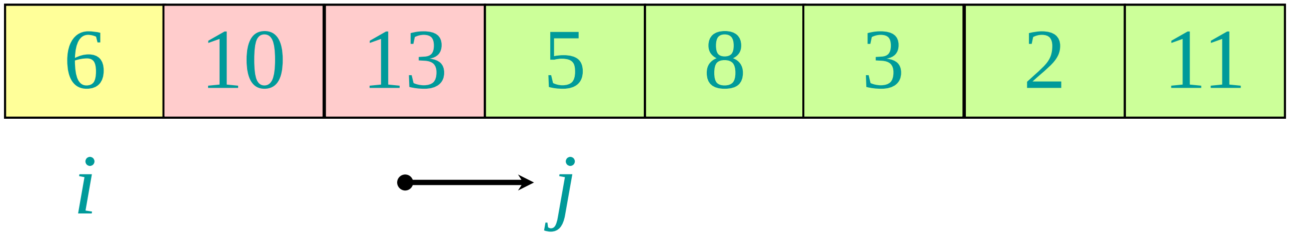
Example of partitioning



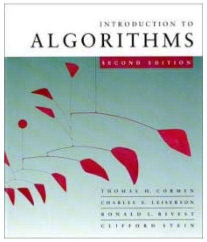
6	10	13	5	8	3	2	11



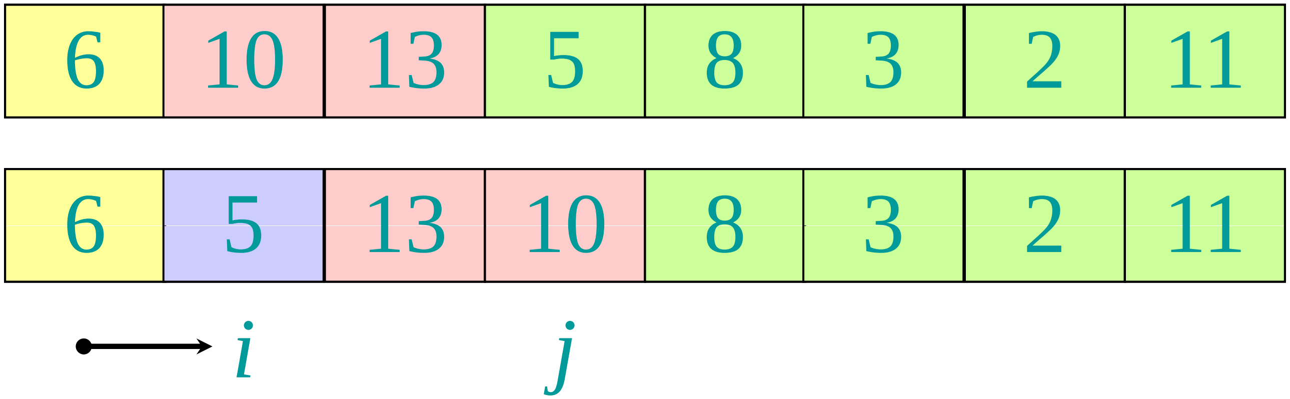
Example of partitioning



6	10	13	5	8	3	2	11

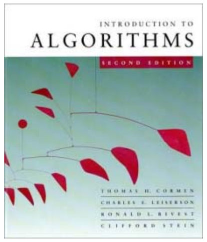


Example of partitioning

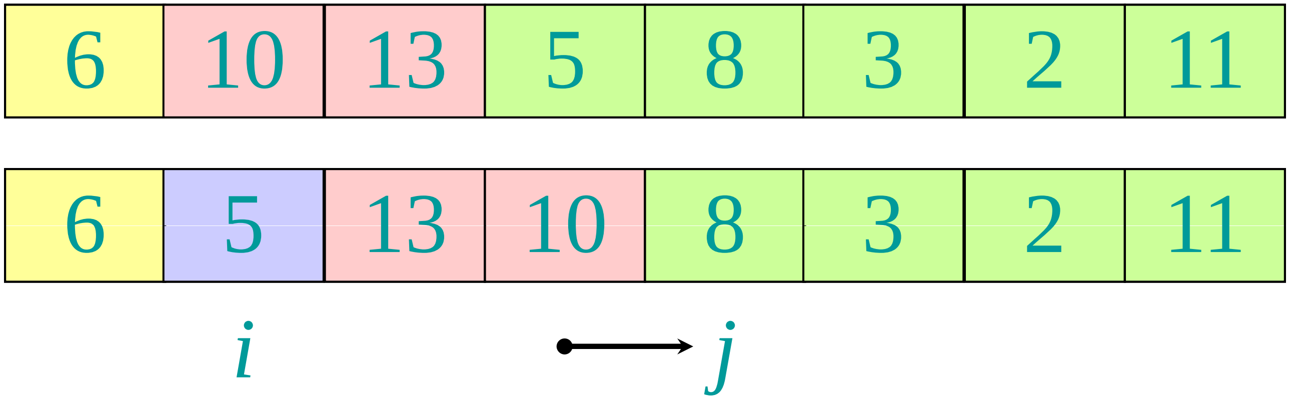


6	10	13	5	8	3	2	11

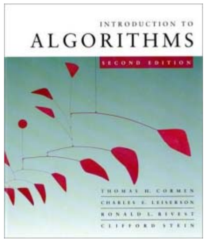
6	5	13	10	8	3	2	11



Example of partitioning



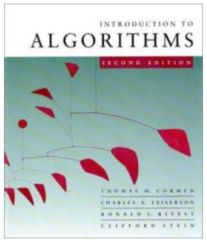
6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11



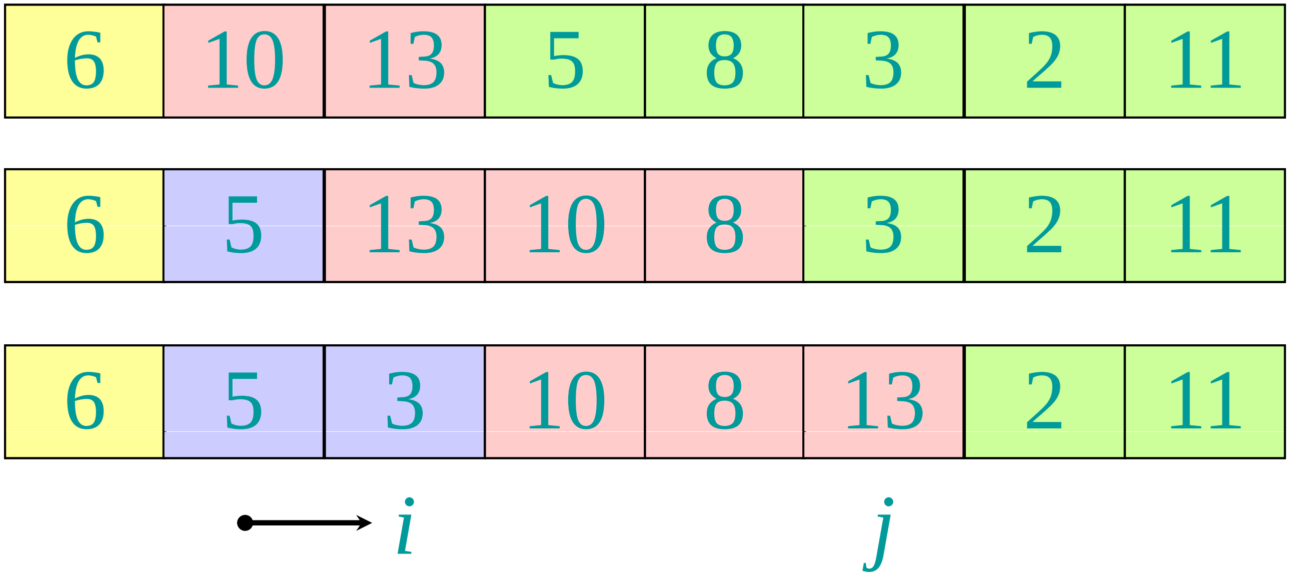
Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11
	i			⟶	*j*		



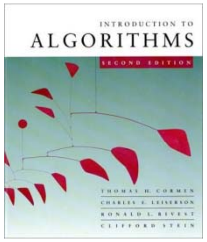
Example of partitioning



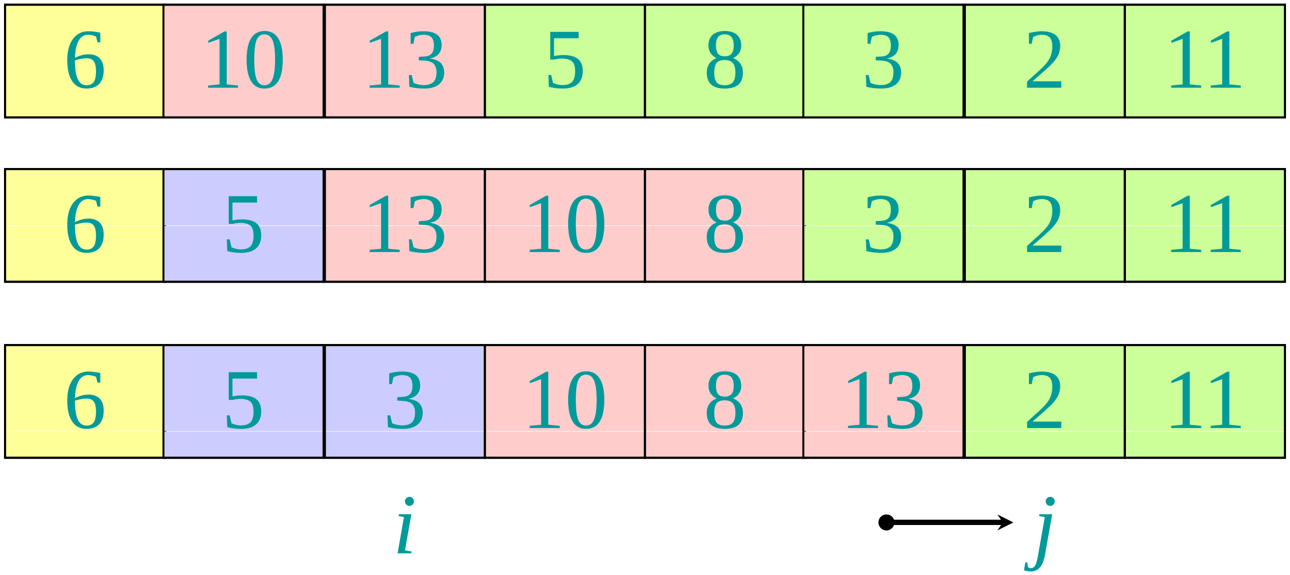
6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11
	→	*i*			*j*		



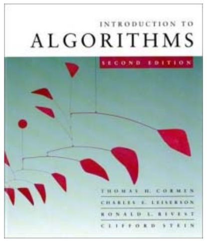
Example of partitioning



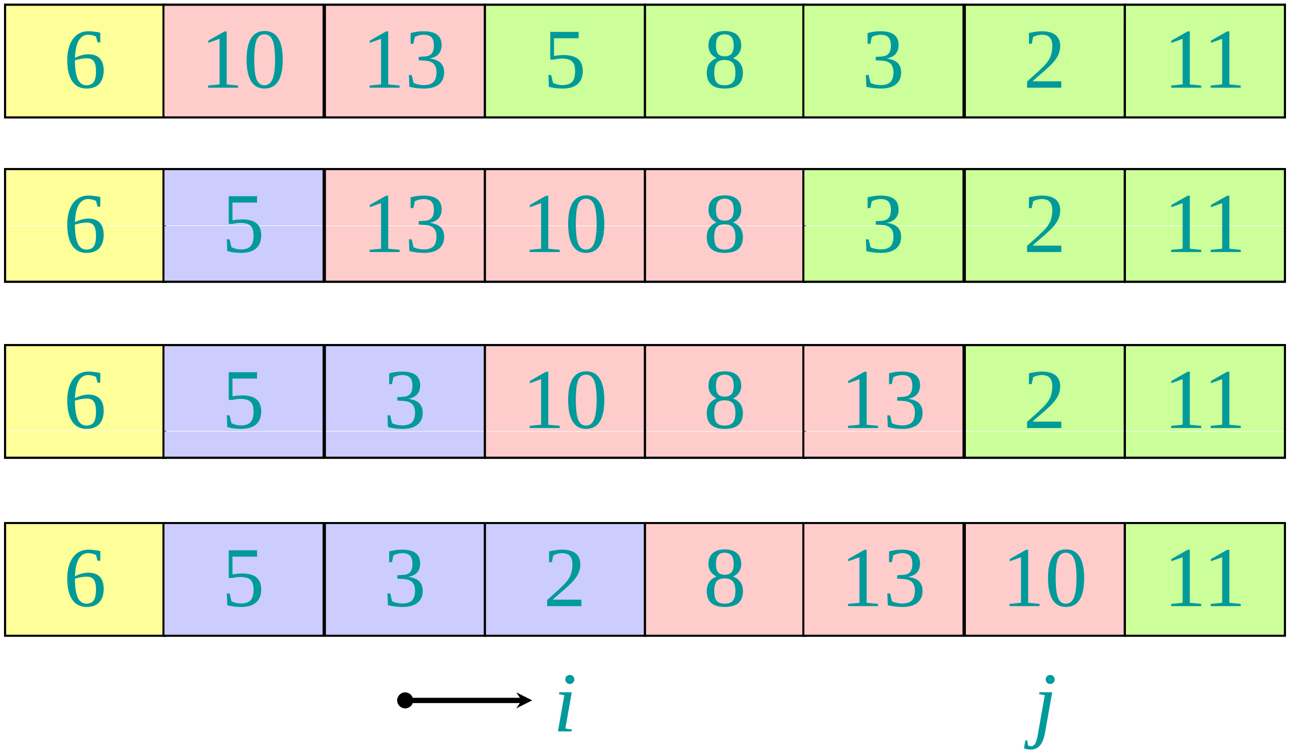
6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11



Example of partitioning

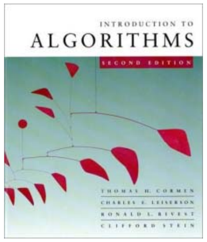


6	10	13	5	8	3	2	11

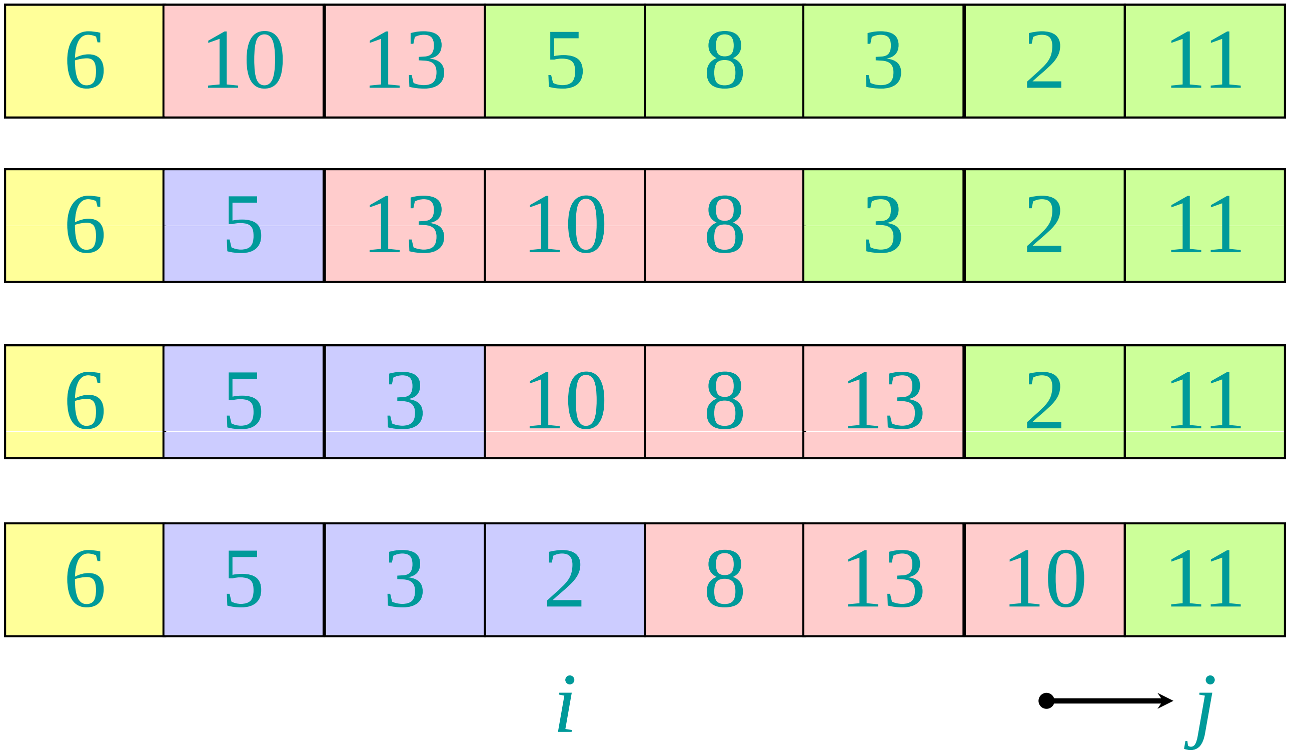
6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

6	5	3	2	8	13	10	11



Example of partitioning

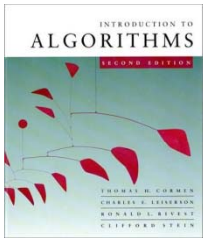


6	10	13	5	8	3	2	11

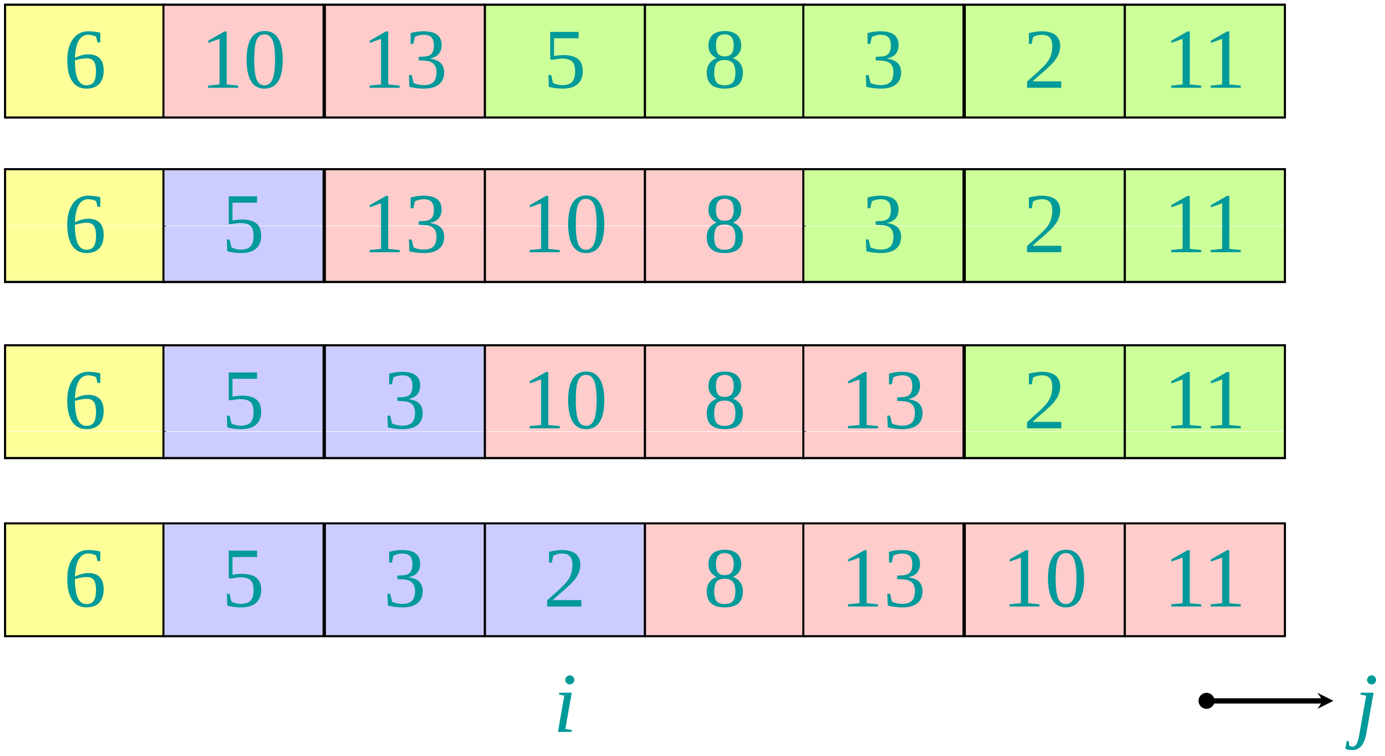
6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

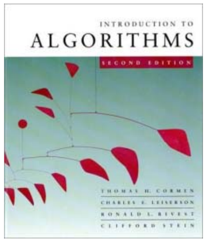
6	5	3	2	8	13	10	11



Example of partitioning

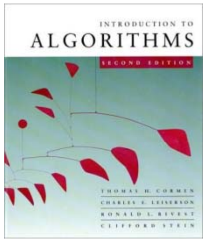


6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11



Example of partitioning

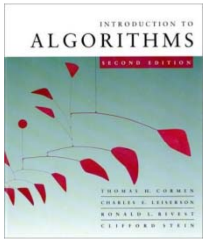
6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
2	5	3	6	8	13	10	11
			i				



Pseudocode for quicksort

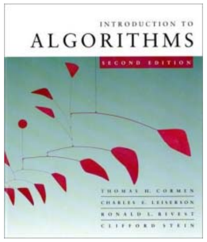
```
QUICKSORT($A, p, r$)
    if $p < r$
        then $q \leftarrow$ PARTITION($A, p, r$)
            QUICKSORT($A, p, q-1$)
            QUICKSORT($A, q+1, r$)
```

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.

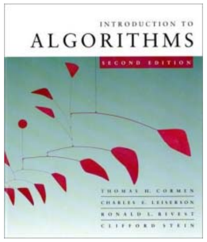


Worst-case of quicksort

```
T(n)  QUICKSORT($A$, $p$, $r$)
          if $p < r$
c·n         then $q \leftarrow$ PARTITION($A$, $p$, $r$)
T(?)             QUICKSORT($A$, $p$, $q-1$)
T(?)             QUICKSORT($A$, $q+1$, $r$)
```

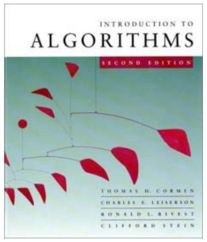
- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

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



Worst-case recursion tree

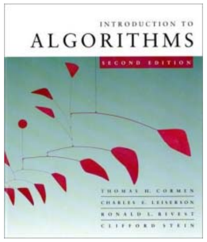
$$T(n) = T(0) + T(n-1) + cn$$



Worst-case recursion tree

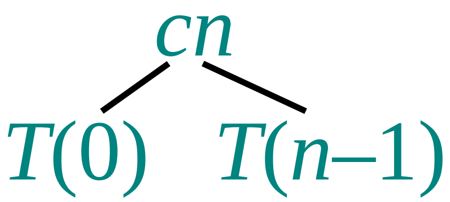
$$T(n) = T(0) + T(n\text{–}1) + cn$$

$$T(n)$$



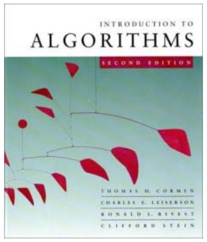
Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$



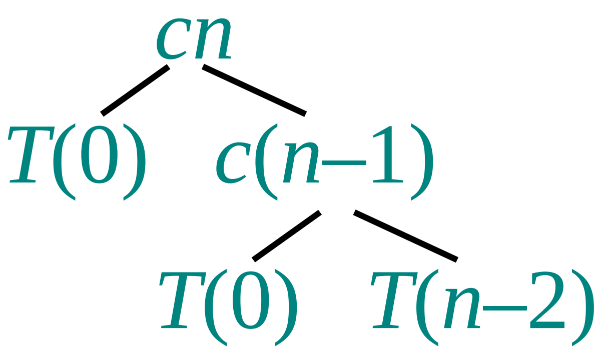
cn

$T(0)$ $T(n–1)$



Worst-case recursion tree

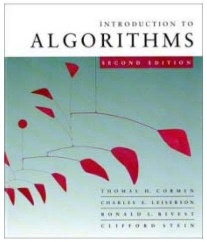
$$T(n) = T(0) + T(n-1) + cn$$



cn

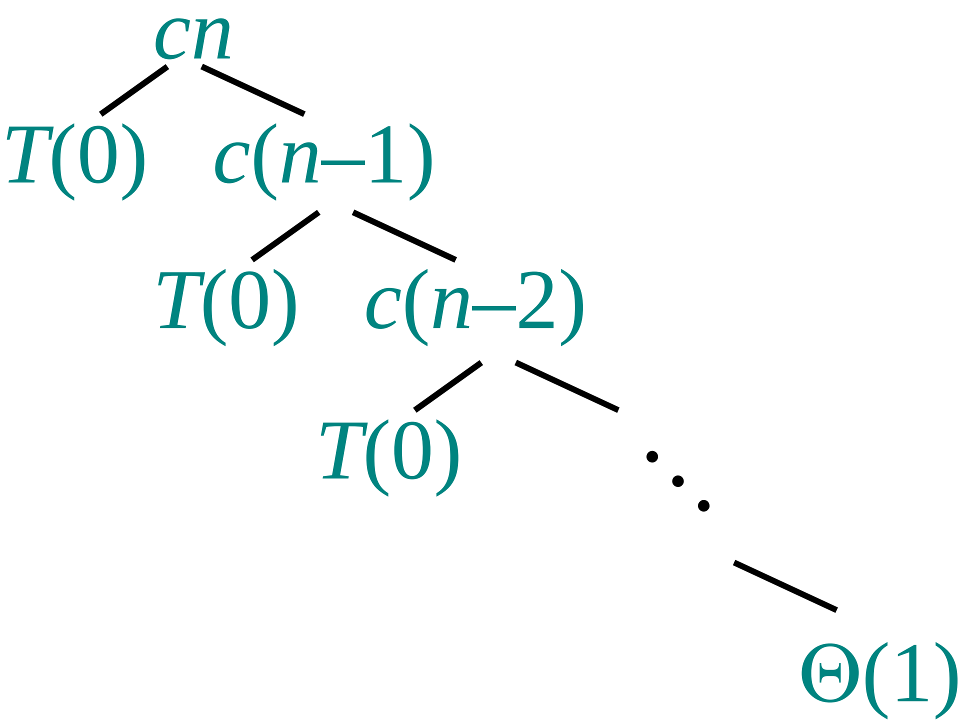
$T(0)$ $c(n-1)$

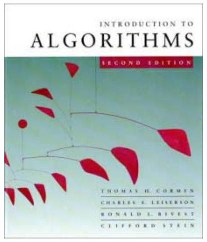
$T(0)$ $T(n-2)$



Worst-case recursion tree

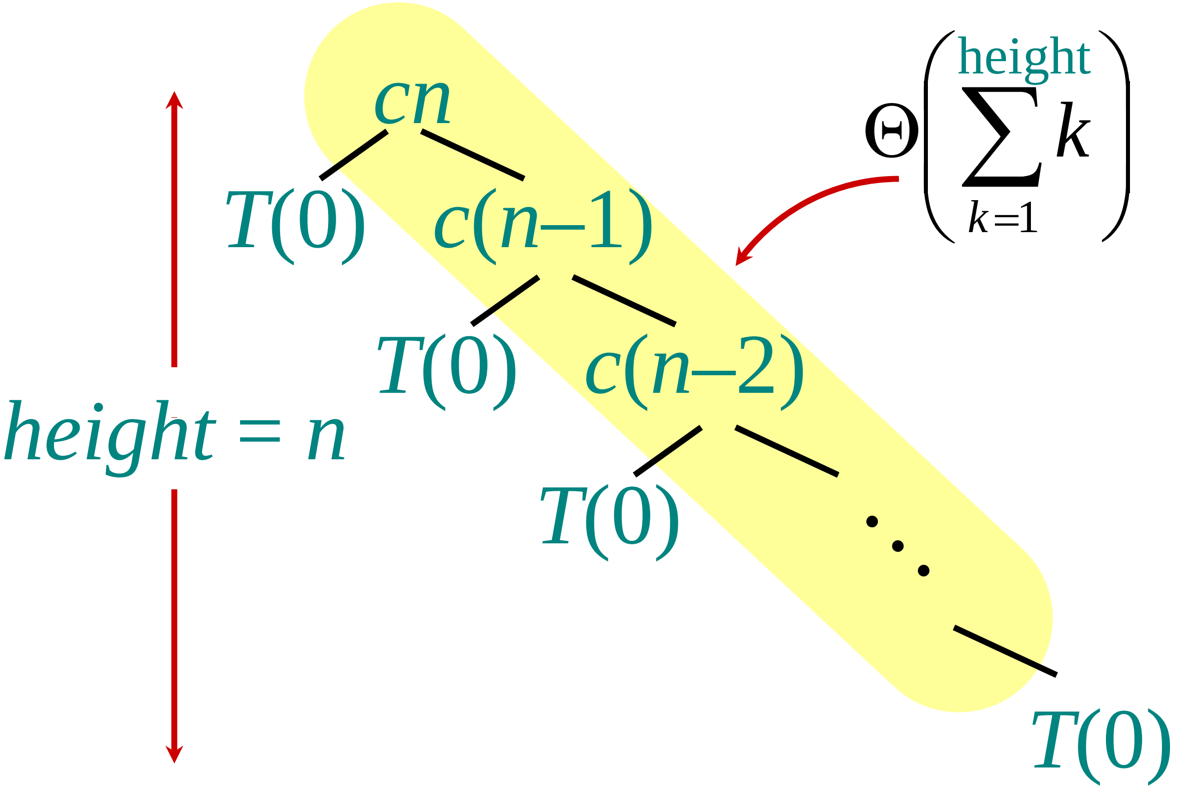
$$T(n) = T(0) + T(n-1) + cn$$



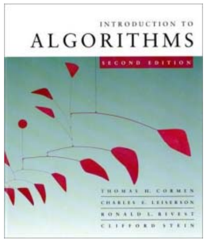


Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

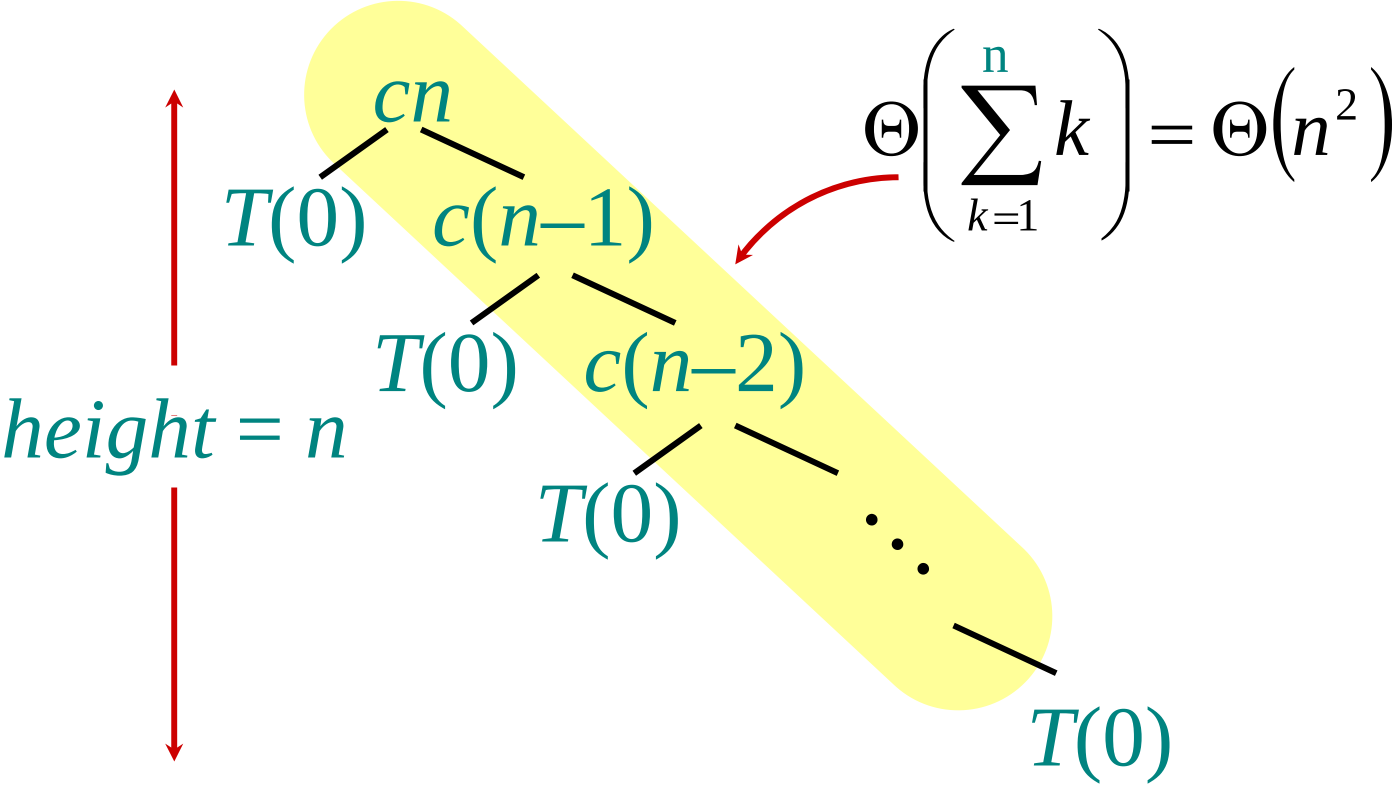


$$\Theta\left(\sum_{k=1}^{\text{height}} k\right)$$

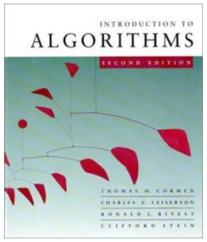


Worst-case recursion tree

$$T(n) = T(0) + T(n–1) + cn$$

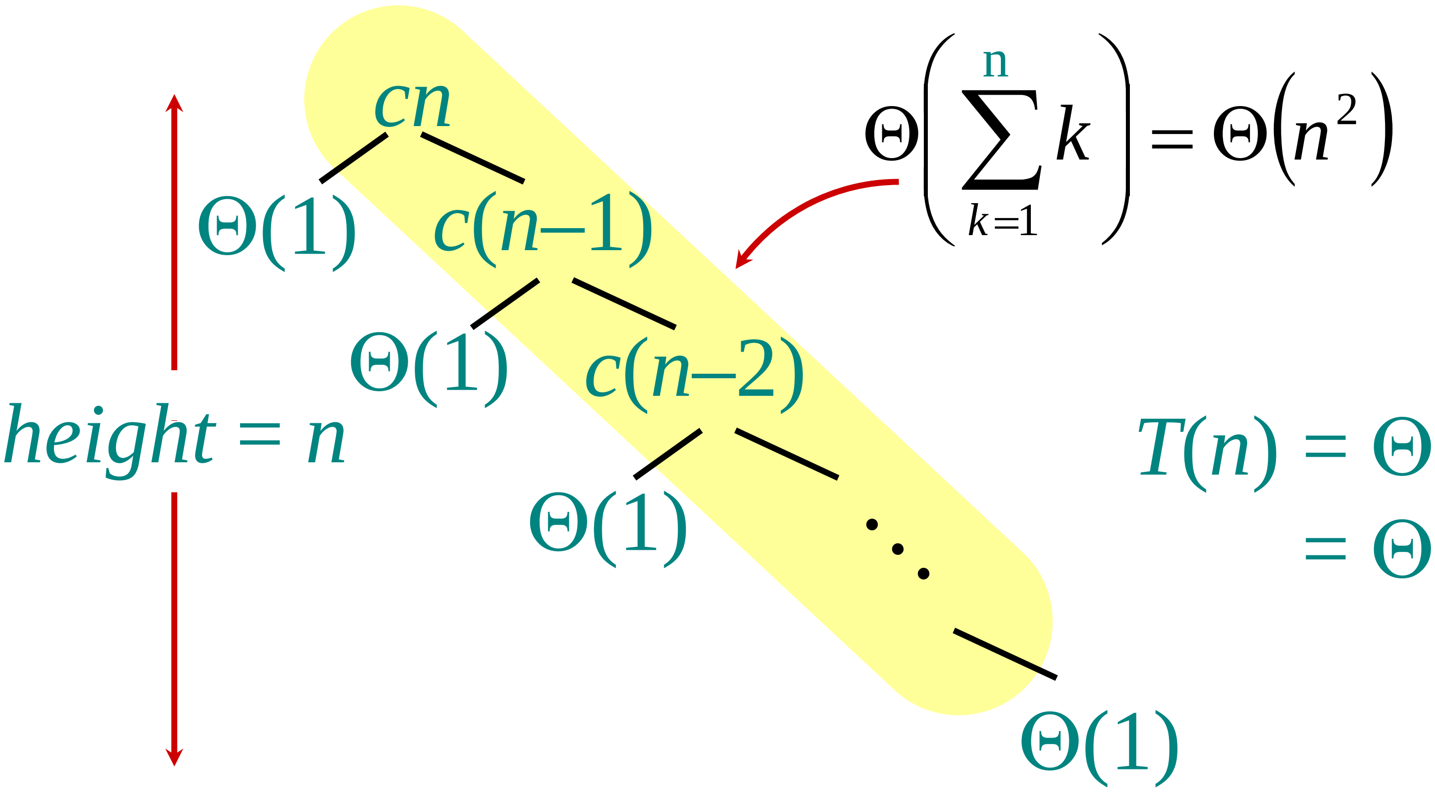


$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$



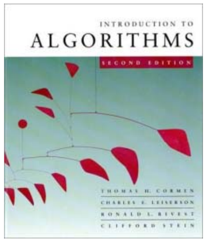
Worst-case recursion tree

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



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

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



Best-case analysis

(For intuition only!)

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

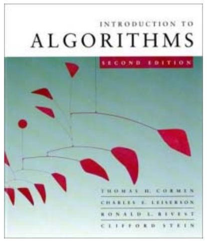
$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$

(same as merge sort)

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

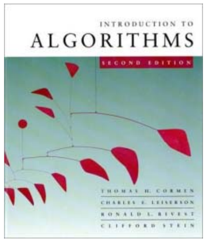
$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \cancel{\Theta(n)} \quad c \cdot n$$

What is the solution to this recurrence?

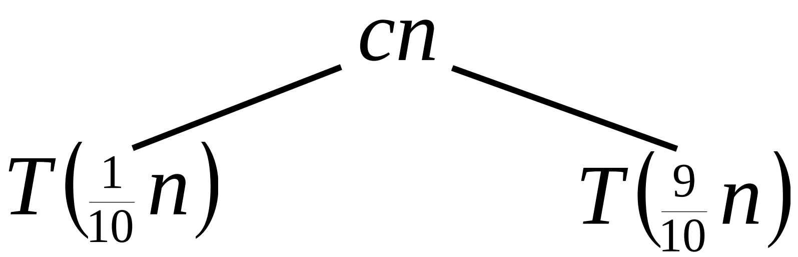


Analysis of “almost-best” case

$$T(n)$$

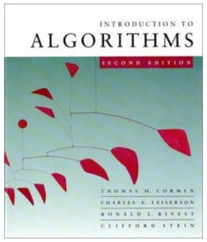


Analysis of “almost-best” case

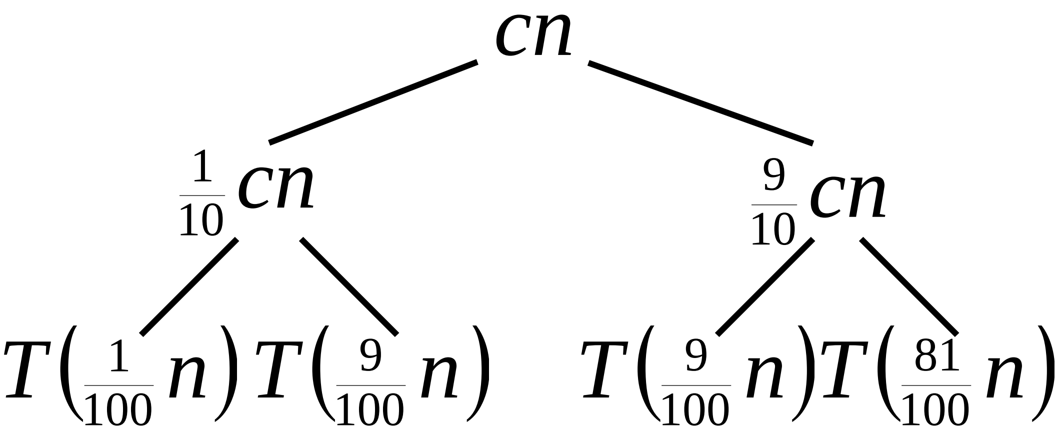


cn

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



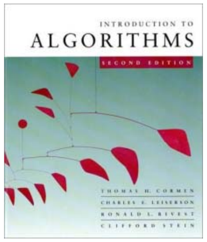
Analysis of “almost-best” case



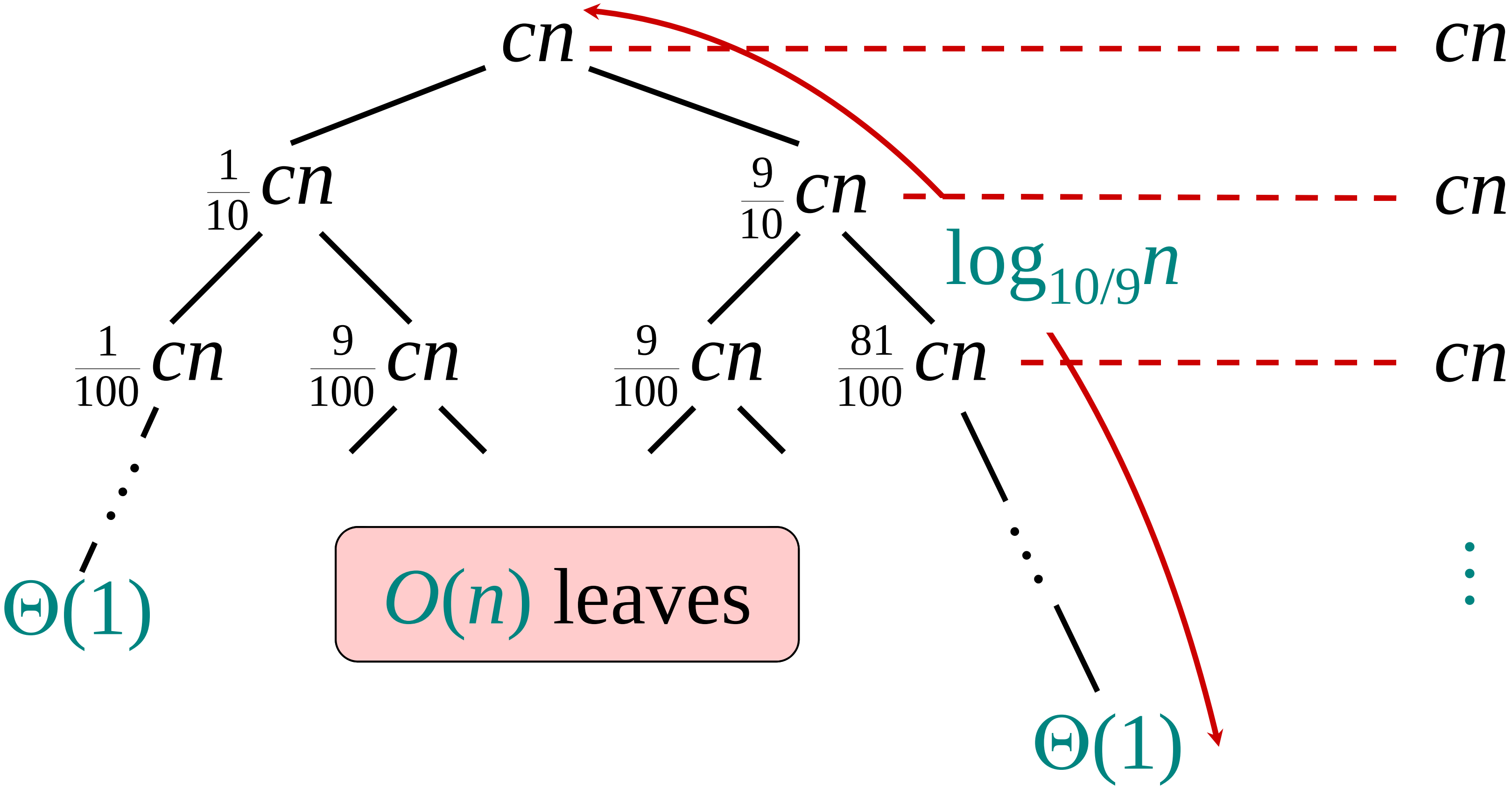
cn

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

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



Analysis of “almost-best” case



cn cn

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

$\log_{10/9} n$

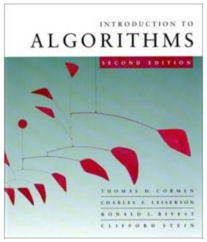
$\frac{1}{100}cn$ $\frac{9}{100}cn$ $\frac{9}{100}cn$ $\frac{81}{100}cn$ cn

$\vdots$

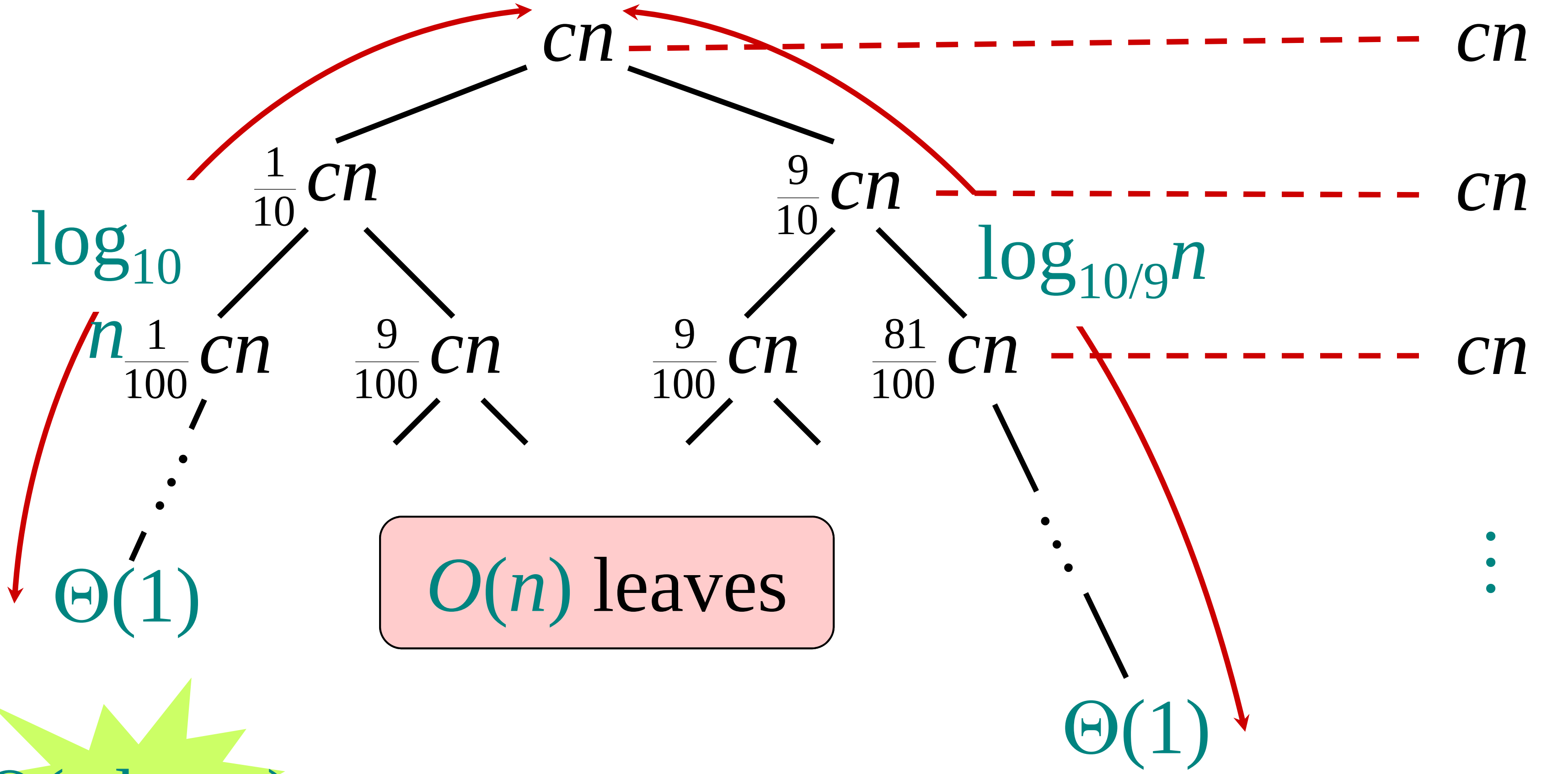
$\Theta(1)$

$O(n)$ leaves

$\Theta(1)$

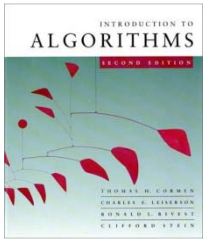


Analysis of “almost-best” case



$\Theta(n \log n)$

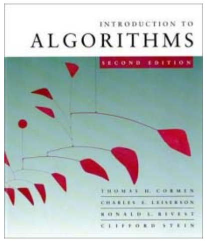
$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n + O(n)$$



Quicksort Runtimes

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

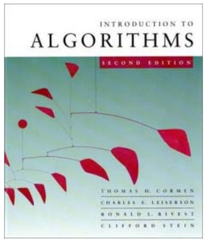
- Worse than mergesort? Why is it called quicksort then?
- Its average runtime $T_{avg}(n) \in O(n \log n)$
- Better even, the expected runtime of **randomized quicksort** is $O(n \log n)$



Average Runtime

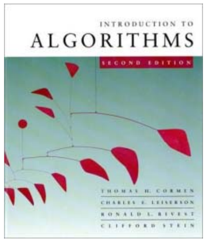
The **average runtime** $T_{avg}(n)$ for Quicksort is the average runtime over **all possible inputs** of length n.

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



Average Runtime

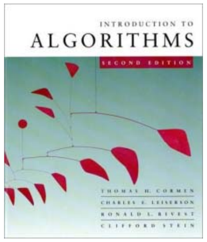
- What kind of inputs are there?
  - Do [1,2,…,n] and [5,6,…,n+5] cause different runtimes of Quicksort?
  - No. Therefore only consider all permutations of [1,2,…,n] .
- How many inputs are there?
  - There are n! different permutations of [1,2,…,n]



Average Runtime

- Therefore, $T_{avg}(n)$ has to average the runtimes over all $n!$ different input permutations
- Disadvantage of considering average runtime:
    - There are still worst-case inputs that will have a $O(n^2)$ runtime
    - Are all inputs really equally likely? That depends on the application

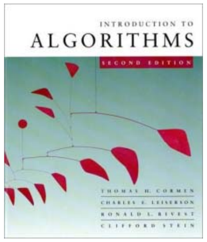
⇒ **Better:** Use randomized quicksort



Randomized quicksort

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

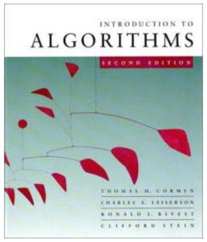
- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.



Randomized quicksort analysis

- $T(n)^{(S)}$ ← permutations $=$ random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent.
- $E(T(n)) =$ expected value of $T(n)$, the "expected runtime" of randomized quicksort.

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

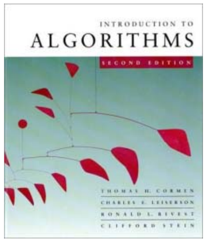


Randomized quicksort analysis

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

$$X_k \overset{(5)}{=} \begin{cases} 1 & \text{if } \textsc{Partition} \text{ generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

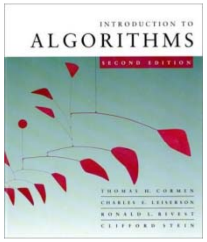
$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.



Analysis (continued)

$$T(n)^{(5)} = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \quad\cdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

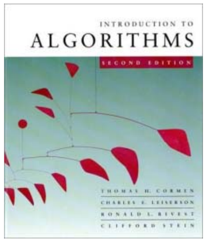
$$= \sum_{k=0}^{n-1} X_k^{(5)} \left(T(k)^{(5)} + T(n-k-1)^{(5)} + \Theta(n) \right).$$



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.

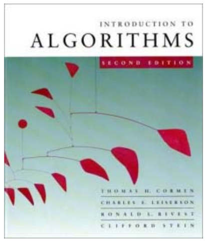


Calculating expectation

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

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

Linearity of expectation.



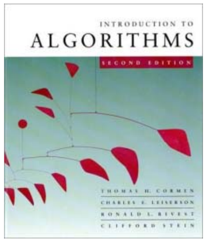
Calculating expectation

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

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

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

Independence of X_k from other random choices.



Calculating expectation

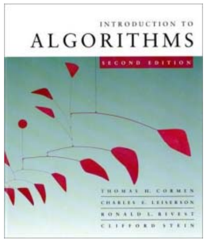
$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

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

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

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

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



Calculating expectation

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

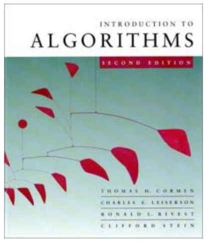
$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

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

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

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

Summations have identical terms.



Hairy recurrence

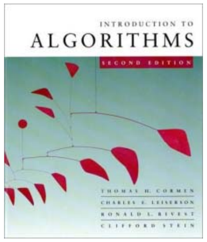
$$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(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 $n \ge 2$.

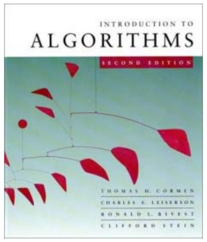
Use fact: $\sum_{k=2}^{n-1} k \log k \le \frac{1}{2} n^2 \log n - \frac{1}{8} n^2$ (exercise).



Substitution method

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

Substitute inductive hypothesis.

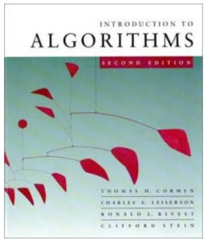


Substitution method

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

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

Use fact.



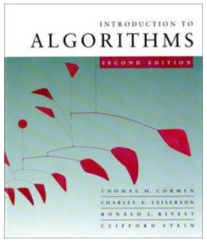
Substitution method

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

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

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

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



Substitution method

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

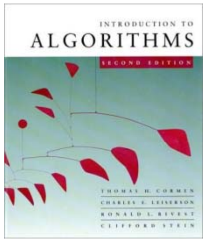
$$= \frac{2a}{n}\left(\frac{1}{2}n^2\log n - \frac{1}{8}n^2\right) + \Theta(n)$$

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

$$\le an\log n$$

,

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.