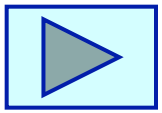
CD1: Lecture 12

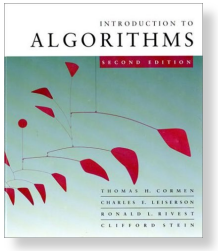
CD1
LeongHW (梁汉槐)

"Quicksort and Randomized Quicksort"

- Topics and Readings
  - Project Update, Project Presentation tomorrow
  - Quicksort and Randomized Quicksort



Quicksort is FAST
&
Randomized Quicksort is Cool.



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).

Antony Hoare (1934 –)

Invented Quicksort (at age 26)

Developed Hoare's Logic (for program correctness)

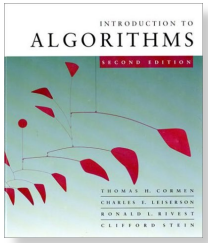
Developed CSP (including dining philisophers' problem)

Quote: (about difficulties of creating software systems)

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

Tony Hoare, Singapore 2008 Computing in the 21st Century

- **Turing Award, 1980**
- **Knighted, 2000**



Divide and conquer

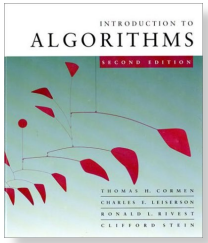
Quicksort an n-element array:

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.

2. Conquer: Recursively sort the two subarrays.

3. Combine: Trivial.

Key: *Linear-time partitioning subroutine.*

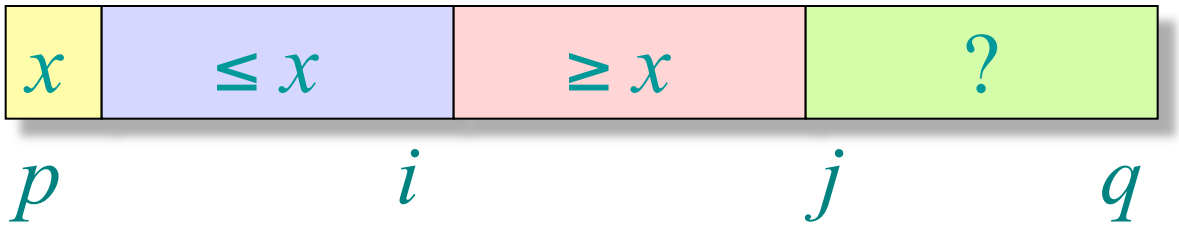


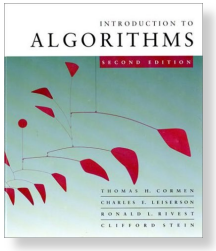
Partitioning subroutine

```
PARTITION(A, p, q)   ▷ A[p . . q]
    x ← A[p]         ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

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

Invariant:

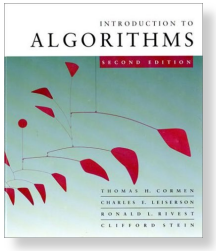




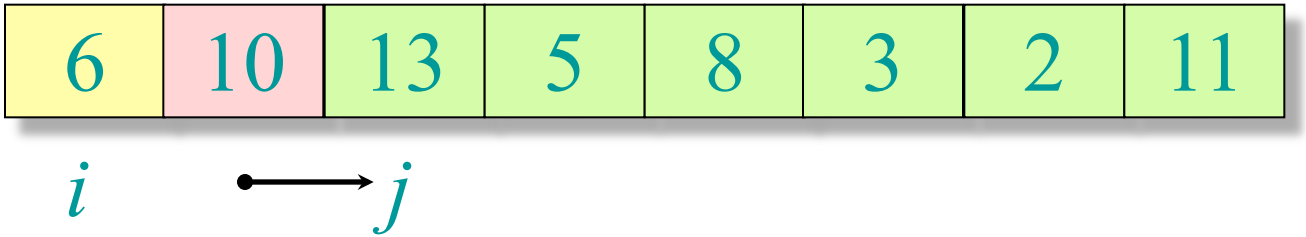
Example of partitioning

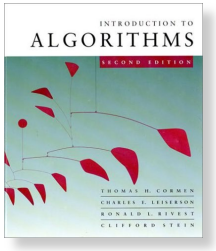
6	10	13	5	8	3	2	11

i *j*

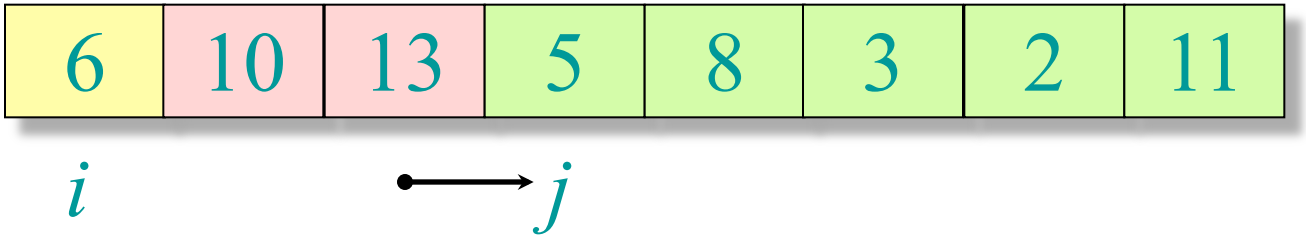


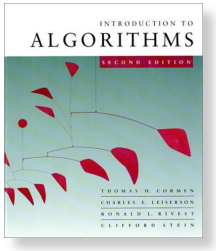
Example of partitioning



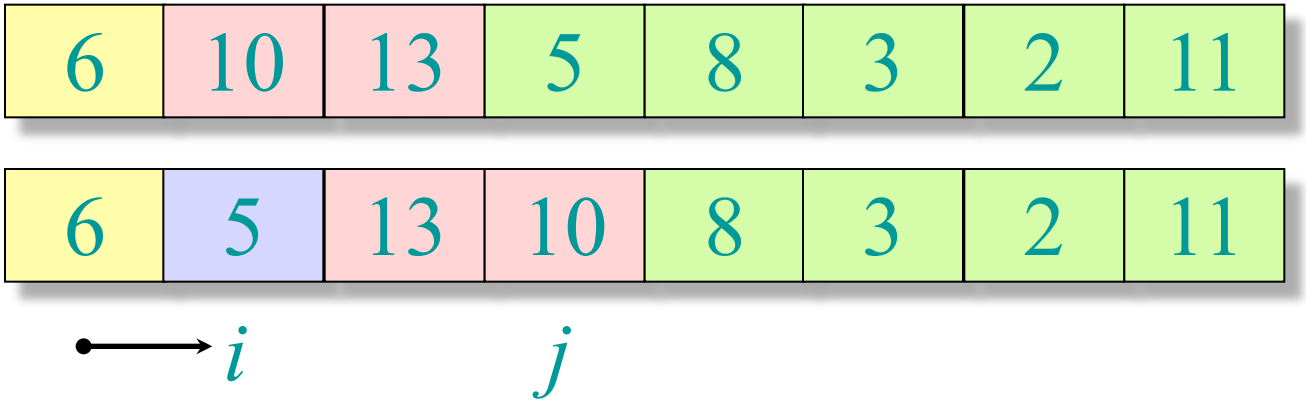


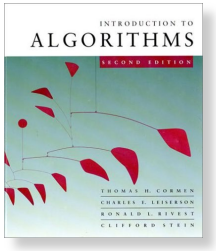
Example of partitioning



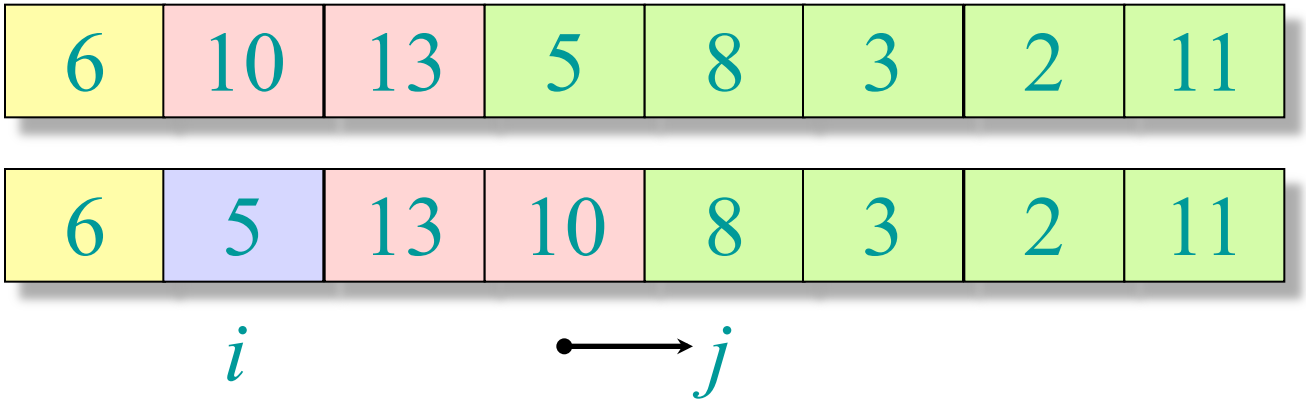


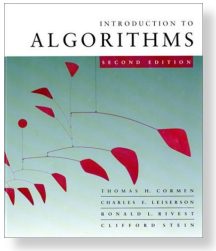
Example of partitioning



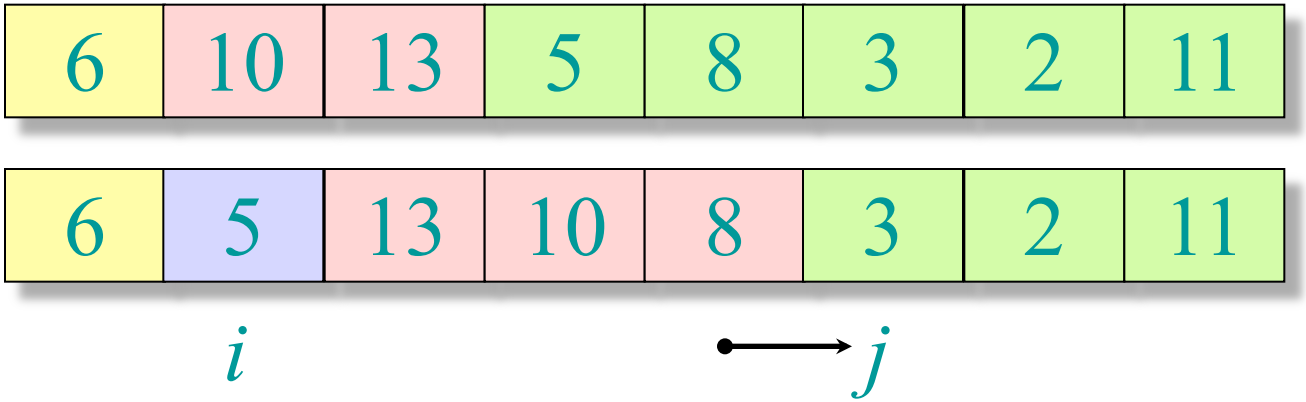


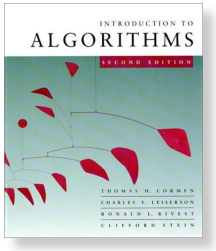
Example of partitioning



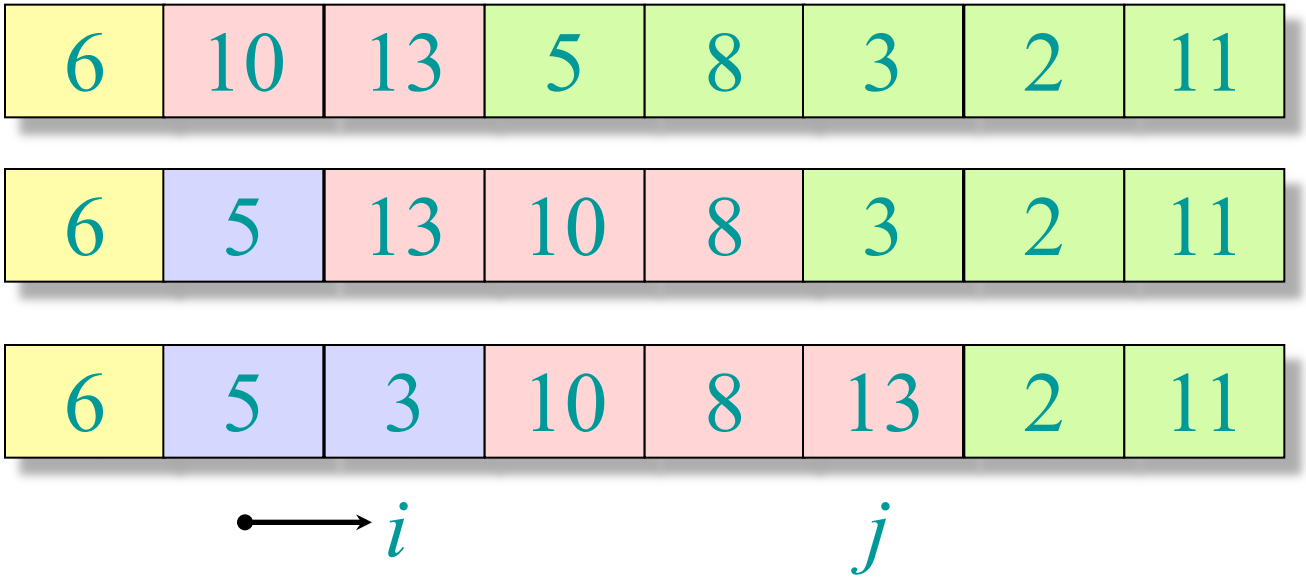


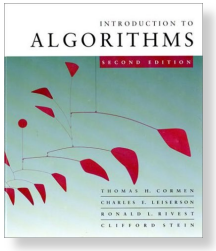
Example of partitioning



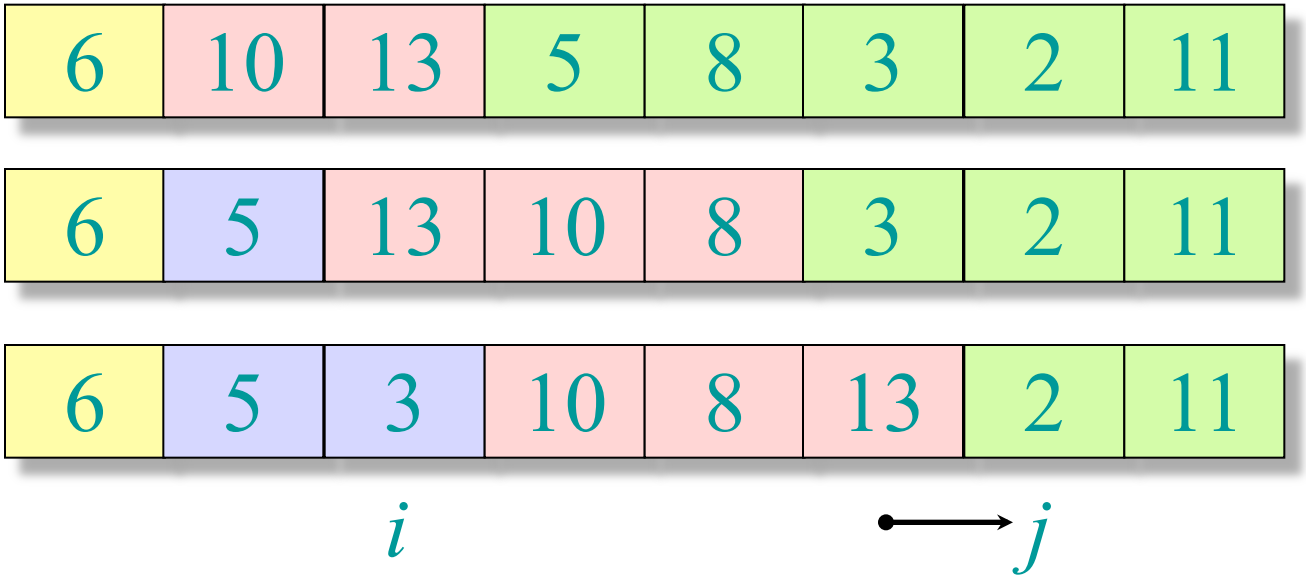


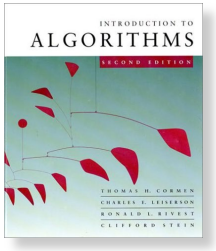
Example of partitioning



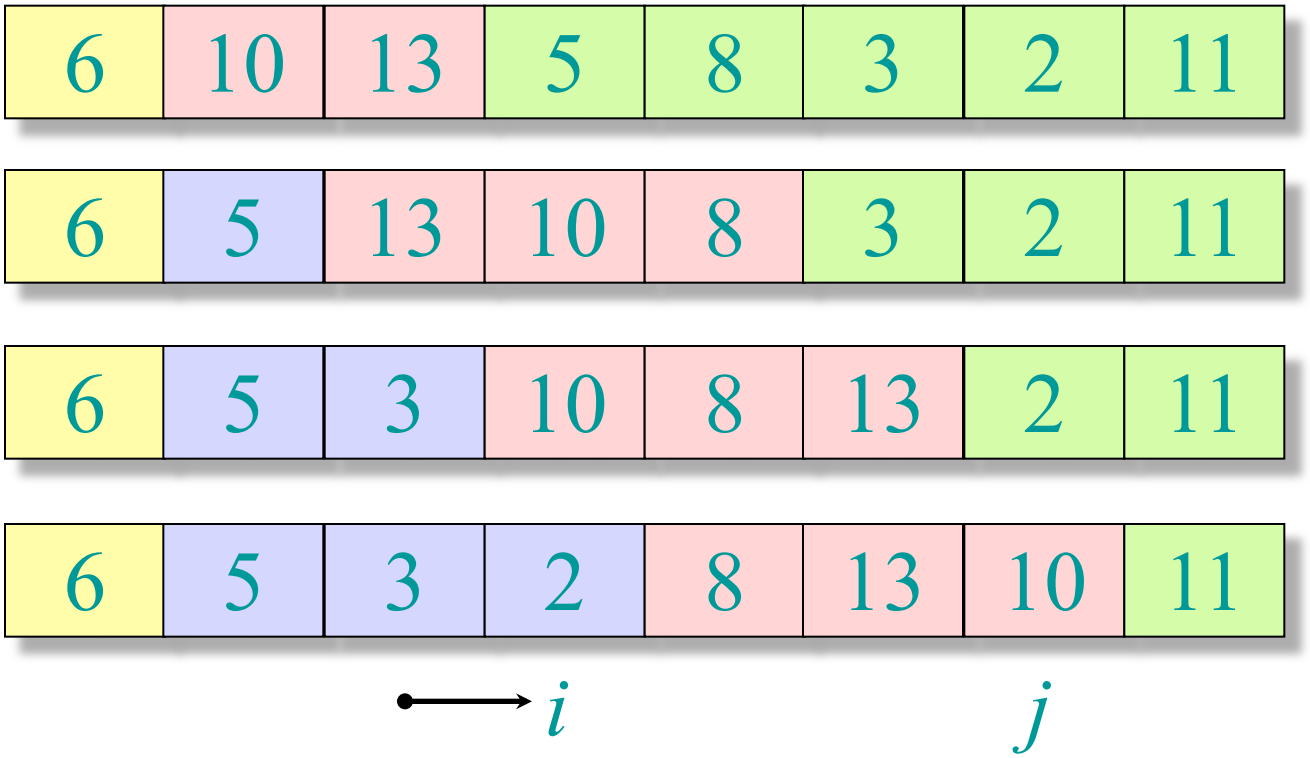


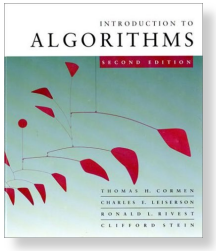
Example of partitioning



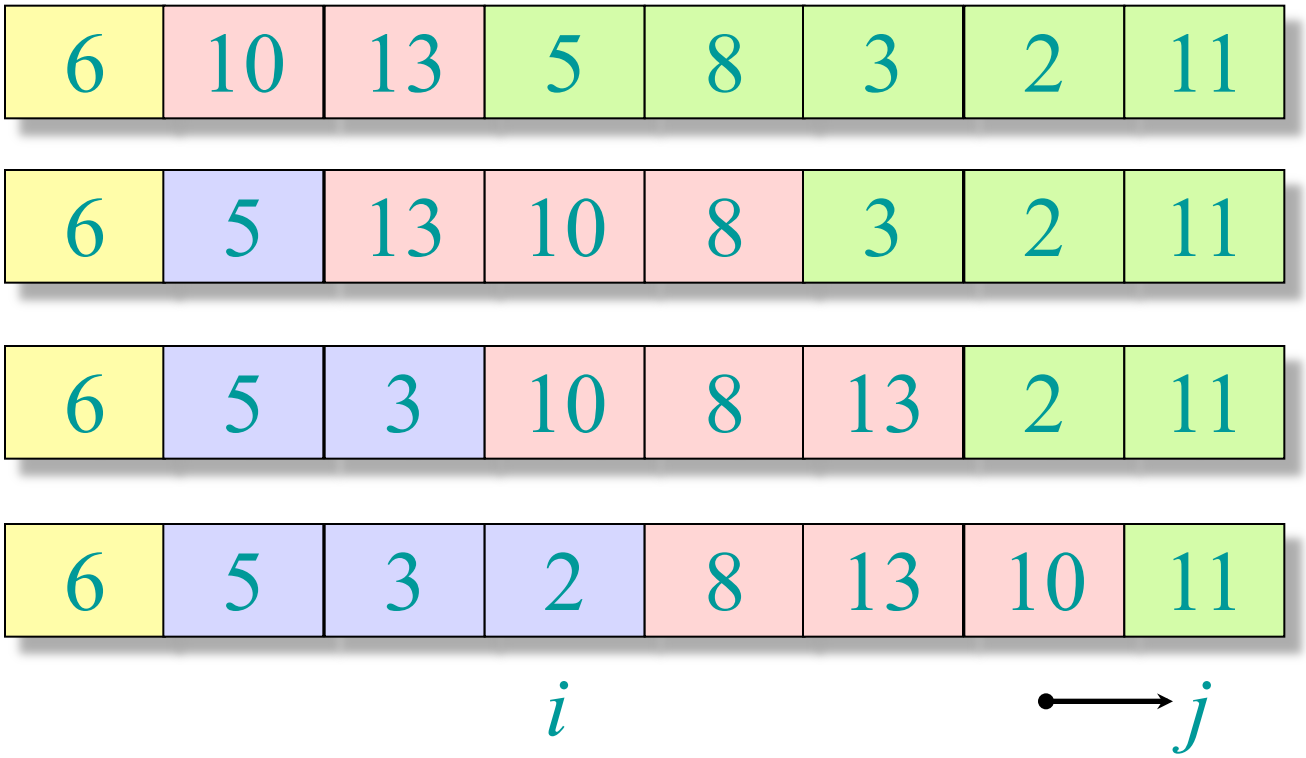


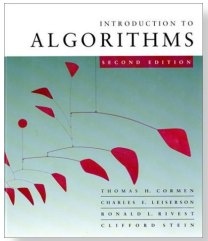
Example of partitioning



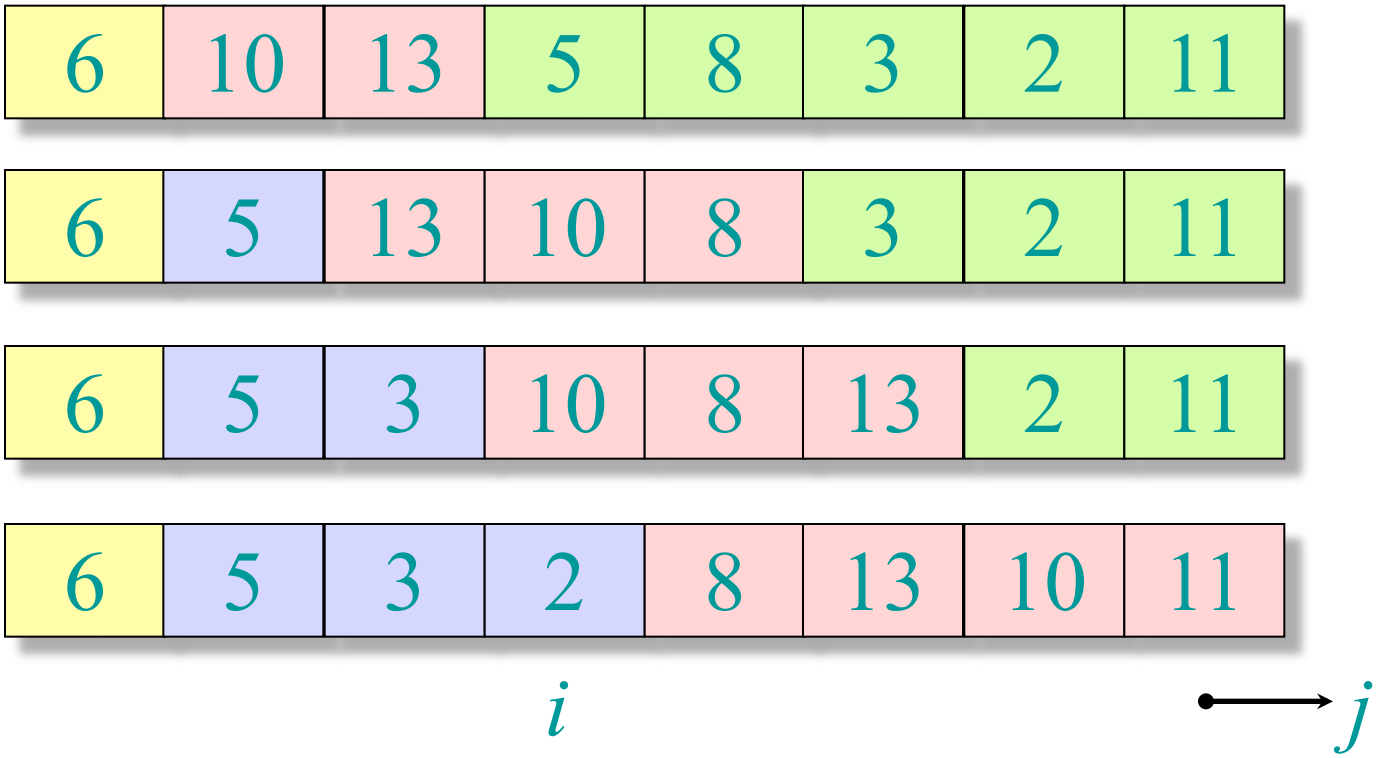


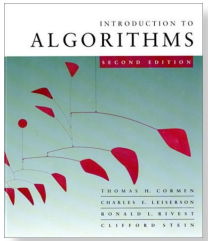
Example of partitioning





Example of partitioning

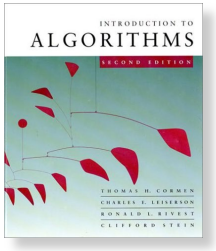




Example of partitioning

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

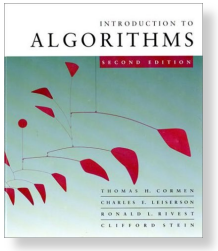
i



Pseudocode for quicksort

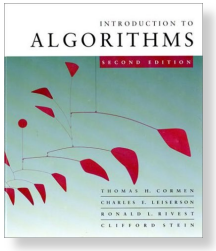
```
QUICKSORT(A, p, r)
    if p < r
        then q ← PARTITION(A, p, r)
             QUICKSORT(A, p, q–1)
             QUICKSORT(A, q+1, r)
```

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



Analysis of quicksort

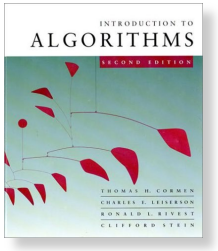
- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.



Worst-case of quicksort

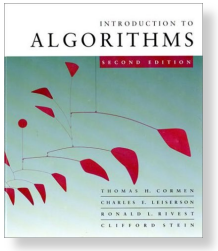
- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

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



Worst-case recursion tree

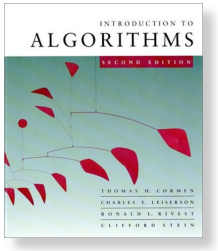
$$T(n) = T(0) + T(n-1) + cn$$



Worst-case recursion tree

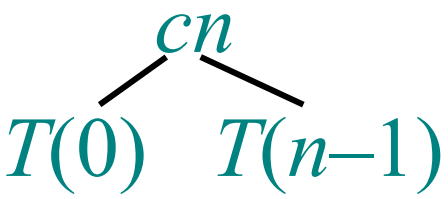
$$T(n) = T(0) + T(n–1) + cn$$

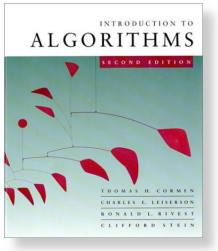
$T(n)$



Worst-case recursion tree

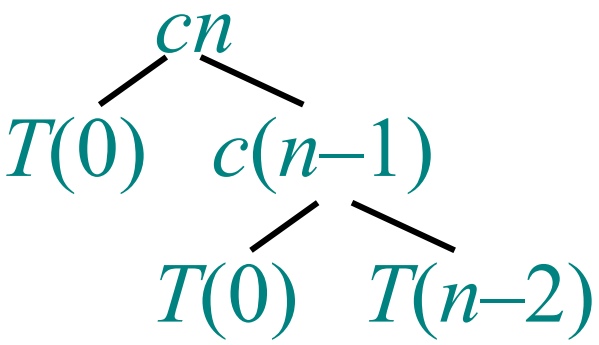
$$T(n) = T(0) + T(n-1) + cn$$

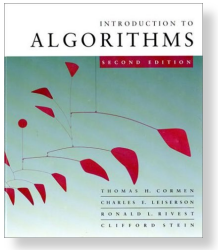




Worst-case recursion tree

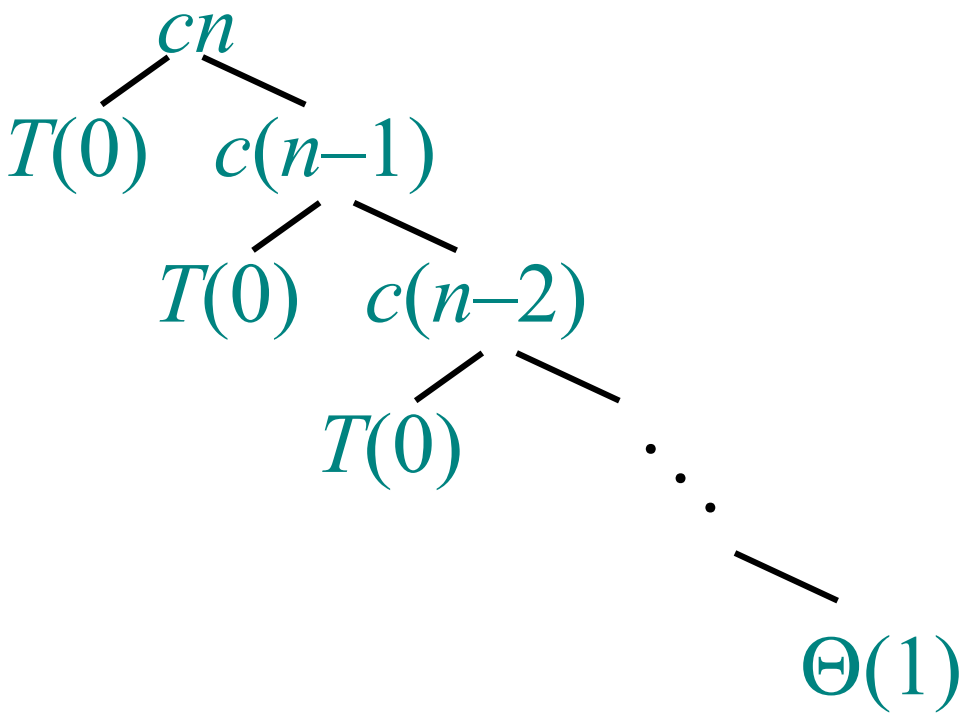
$$T(n) = T(0) + T(n–1) + cn$$

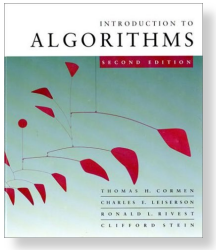




Worst-case recursion tree

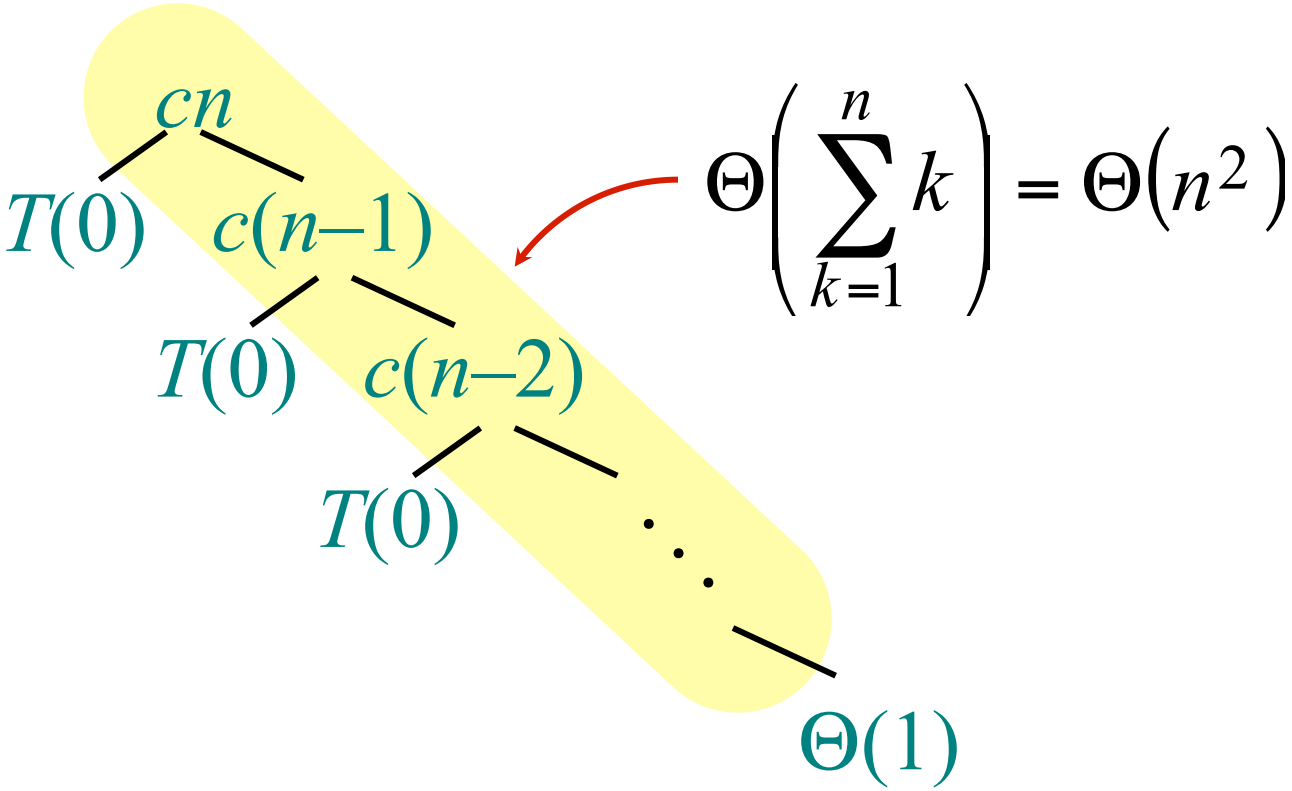
$$T(n) = T(0) + T(n–1) + cn$$

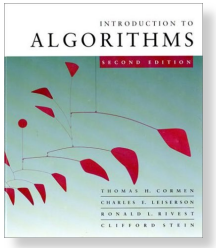




Worst-case recursion tree

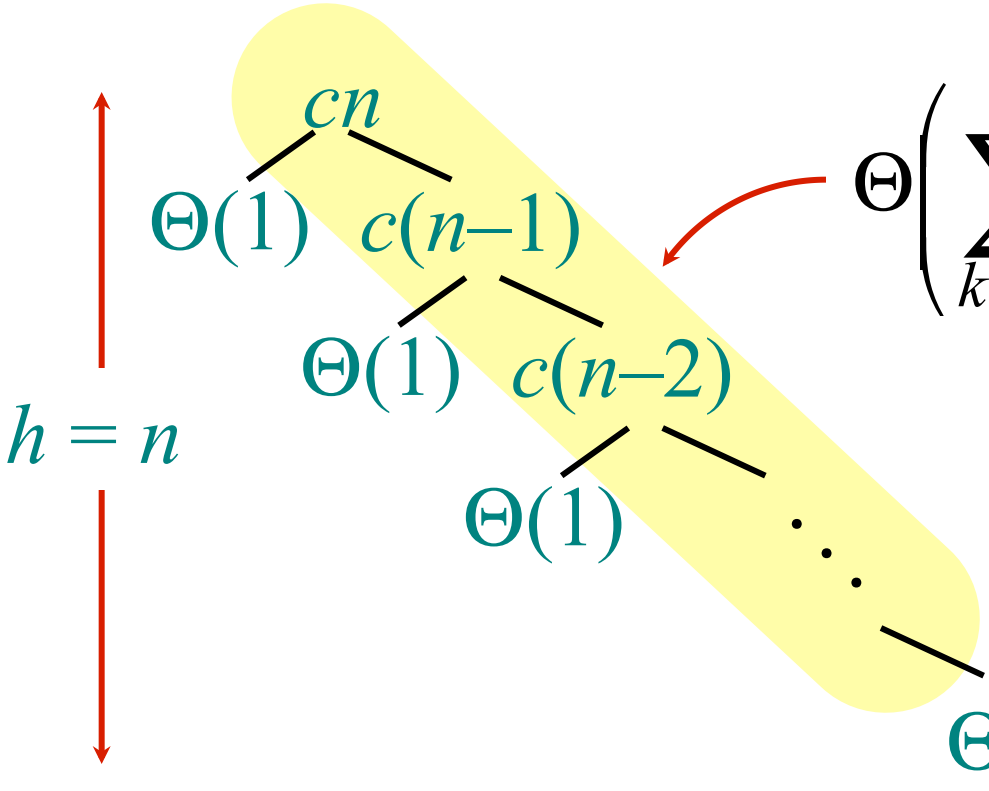
$$T(n) = T(0) + T(n-1) + cn$$





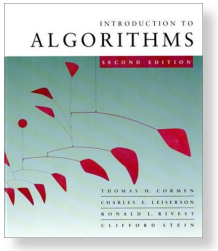
Worst-case recursion tree

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



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

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



Best-case analysis

(For intuition only!)

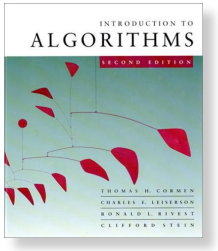
If we're lucky, PARTITION splits the array evenly:

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

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

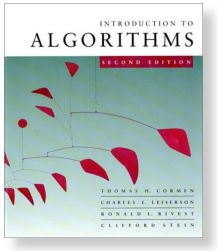
$$T(n) = T\left(\tfrac{1}{10} n\right) + T\left(\tfrac{9}{10} n\right) + \Theta(n)$$

What is the solution to this recurrence?

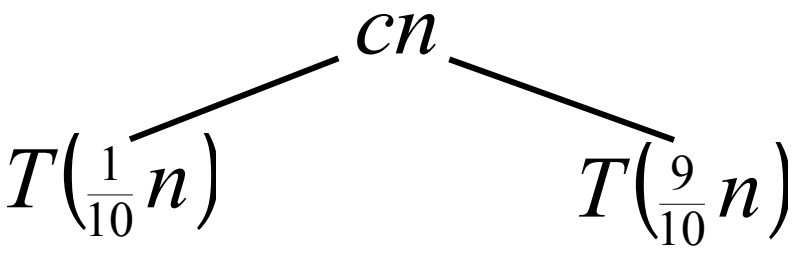


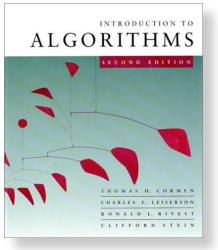
Analysis of "almost-best" case

$$T(n)$$

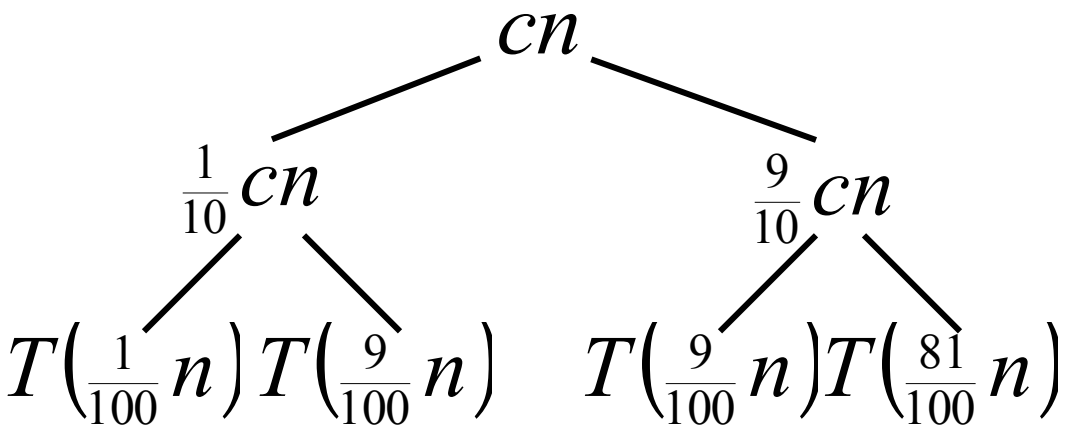


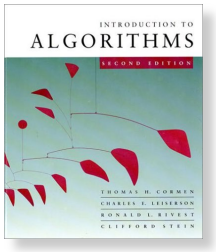
Analysis of "almost-best" case



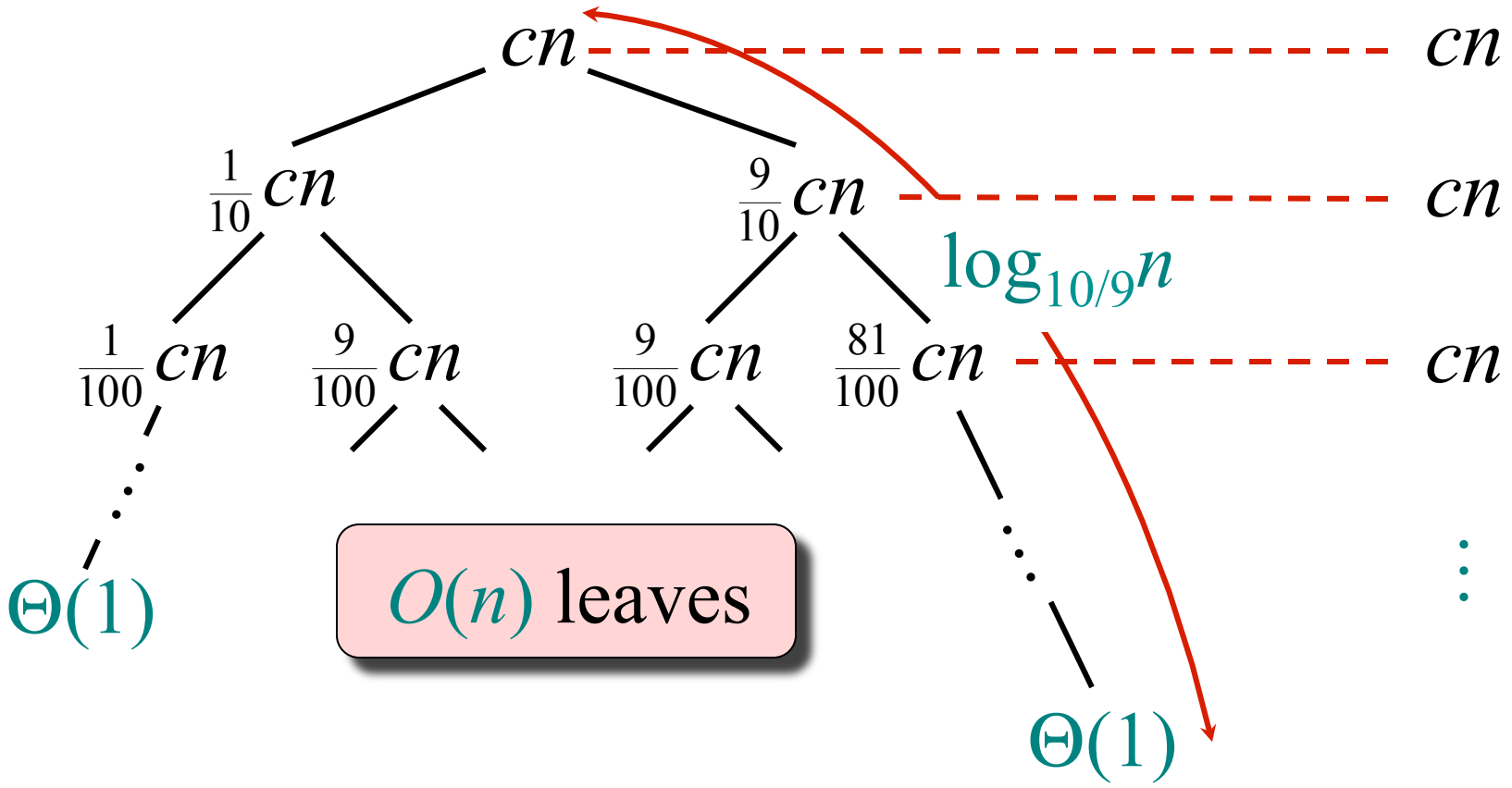


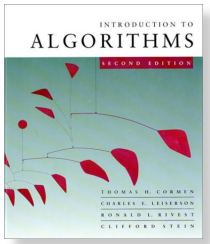
Analysis of "almost-best" case



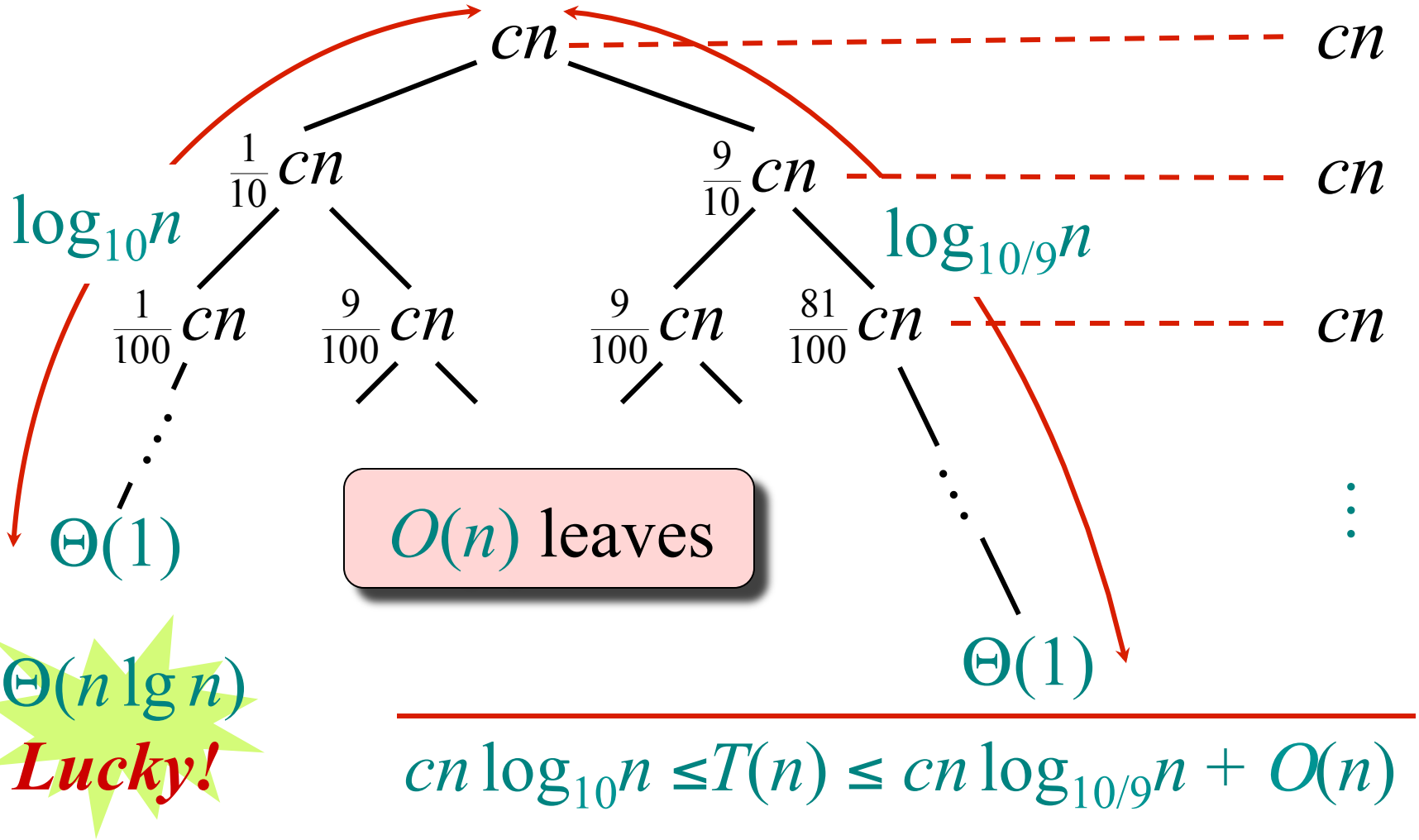


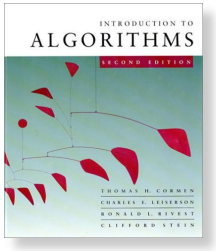
Analysis of "almost-best" case





Analysis of "almost-best" case





More intuition

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

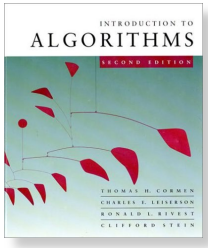
$$L(n) = 2U(n/2) + \Theta(n) \quad \textbf{\textit{lucky}}$$
$$U(n) = L(n-1) + \Theta(n) \quad \textbf{\textit{unlucky}}$$

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

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

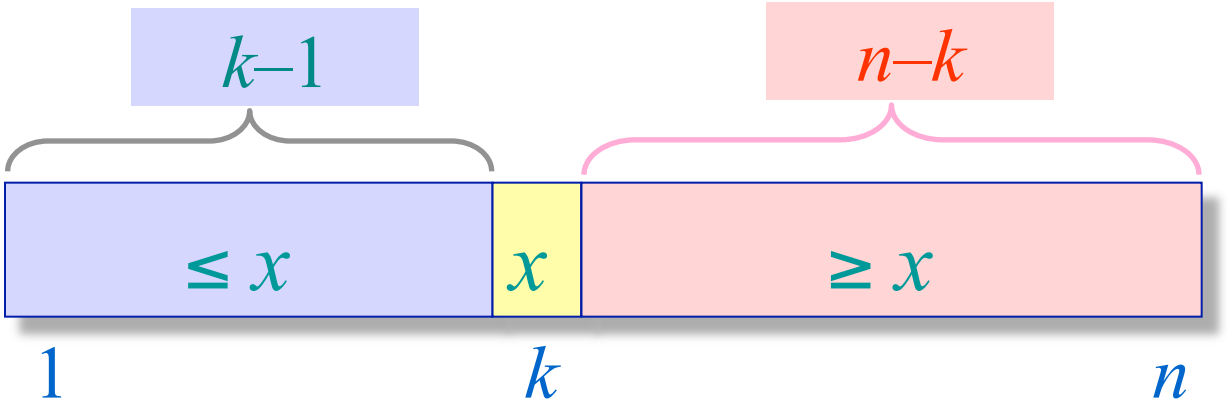
- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

Analysis of Randomized Quicksort

Let $T(n)$ = the *average* time taken to sort an array of size n using Quicksort

If pivot x ends up in position k,

then $T(n) = T(k-1) + T(n-k) + (n+1)$



Prob(pivot is at pos k) = $1/n$ *for all* k

Analysis of Randomized Quicksort

Then, we have

$$T(n) = \begin{cases} T(0) + T(n-1) + (n+1) & \text{if } 0 : n-1 \text{ split} \\ T(1) + T(n-2) + (n+1) & \text{if } 1 : n-2 \text{ split} \\ T(2) + T(n-3) + (n+1) & \text{if } 2 : n-3 \text{ split} \\ \vdots & \vdots \\ T(n-1) + T(0) + (n+1) & \text{if } n-1 : 0 \text{ split} \end{cases}$$

Prob(pivot is at pos k) = $1/n$ *for all* k

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

Hon Wai Leong, NUS

Analysis of Randomized Quicksort

Then, we have the following recurrence:

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

→ ***Expand the summations***

Analysis of Randomized Quicksort

Then, we have the following recurrence:

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

$$nT(n) = 2\sum_{k=0}^{n-1} T(k) + n(n+1)$$

$$nT(n) = 2\big(T(0) + T(1) + \ldots + T(n-1)\big) + n(n+1)$$

→Get rid of dependence on "full history"

Analysis of Randomized Quicksort

Then, we get rid of "full history":

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

$$nT(n) = 2\left[T(0) + T(1) + \ldots + T(n-2) + T(n-1) \right] + n(n+1)$$

$$(n-1)T(n-1) = 2\left[T(0) + T(1) + \ldots + T(n-2) \right] + (n-1)n$$

$$nT(n) = (n+1)T(n-1) + 2n$$

→ ***Divide by n(n+1)…*** (make it telescopic)

Hon Wai Leong, NUS

Analysis of Randomized Quicksort

Divide by $n(n+1)$… (make it telescopic)

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

$$nT(n) = (n+1)T(n-1) + 2n$$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2}{n+1}$$

→ ***Now "telescope"…***

Hon Wai Leong, NUS

Analysis of Randomized Quicksort

Now, telescope…

$$\frac{T(n)}{(n+1)} = \frac{2}{(n+1)} + \frac{T(n-1)}{(n)}$$

$$= \frac{2}{(n+1)} + \frac{2}{(n)} + \frac{T(n-2)}{(n-1)}$$

$$= \frac{2}{(n+1)} + \frac{2}{(n)} + \frac{2}{(n-1)} + \frac{T(n-3)}{(n-2)}$$

$$= \frac{2}{(n+1)} + \frac{2}{(n)} + \frac{2}{(n-1)} + \ldots + \frac{2}{(3)} + \frac{T(1)}{(2)}$$

$$\frac{T(n)}{(n+1)} = \frac{T(1)}{(2)} + 2\left[\frac{1}{(n+1)} + \frac{1}{(n)} + \frac{1}{(n-1)} + \ldots + \frac{1}{(3)}\right]$$

Analysis of Randomized Quicksort

Then, we have the following recurrence:

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

$$\frac{T(n)}{(n+1)} = \frac{T(1)}{(2)} + 2\left[\frac{1}{(n+1)} + \frac{1}{(n)} + \frac{1}{(n-1)} + \ldots + \frac{1}{(3)} \right]$$

$$T(n) = 2(n+1)H(n+1) + O(n)$$

$$H(n) = \sum_{k=1}^{n} \frac{1}{k} \text{ is the Harmonic series}$$

Analysis of Randomized Quicksort

Avg running time of Randomized Quicksort:

$$T(n) = 2(n+1)H(n+1) + O(n)$$

$$H(n) = \ln n + O(1) \quad \text{[CLRS] - App.A}$$

$$T(n) = 2(n+1)\ln n + O(n)$$

$$T(n) = 1.386 n \lg n + O(n)$$

Randomized Quicksort is ***only* 38.6% *from optimal***.

Optimal sorting is $T^*(n) = (n \lg n)$ [See L.B. for Sorting]

Recap…

Beautiful analysis of
Randomized Quicksort to get…

$$T(n) = 1.386 n \lg n + O(n)$$

Not that difficult, *right?*

Where are the key steps?

- Get rid of full history
- Telescope

Recap: The Key Steps

This recurrence depends on *full history*

$$n \cdot T(n) = 2\sum_{k=0}^{n} T(k) + n(n+1)$$

Step 1: *Get rid of full history*… to get

$$n \cdot T(n) = (n+1)T(n-1) + 2n$$

Step 2: Get to a form that can *telescope*…

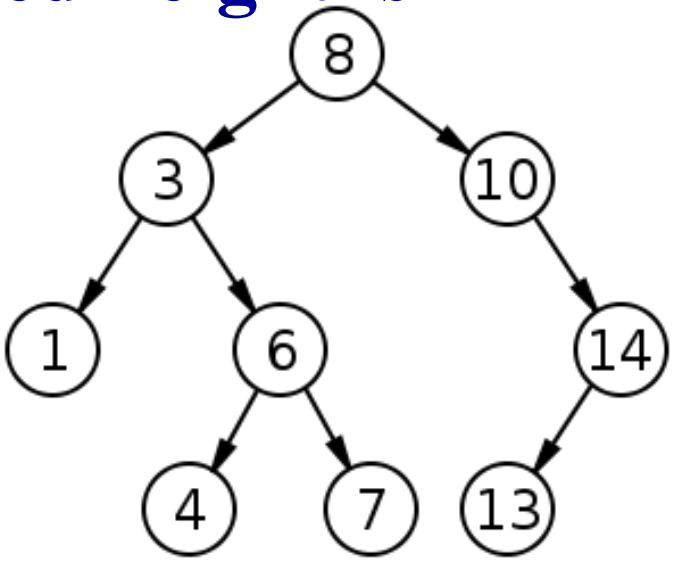
$$\frac{T(n)}{(n+1)} = \frac{T(n-1)}{(n)} + \frac{2}{(n+1)}$$

Hon Wai Leong, NUS

Using the result…

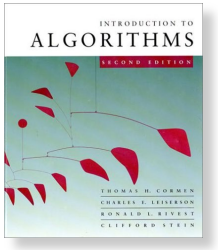
Using a similar analysis, we can show…

- **For a randomly built *n*-node BST (binary search tree), the expected height is**
  - **1.386 lg *n***



- ***Try it out yourself…***

  Or read [CLRS]-C12.4

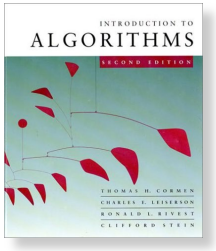


Randomized quicksort analysis [by CLRS]

[CLRS] uses a slightly different 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.

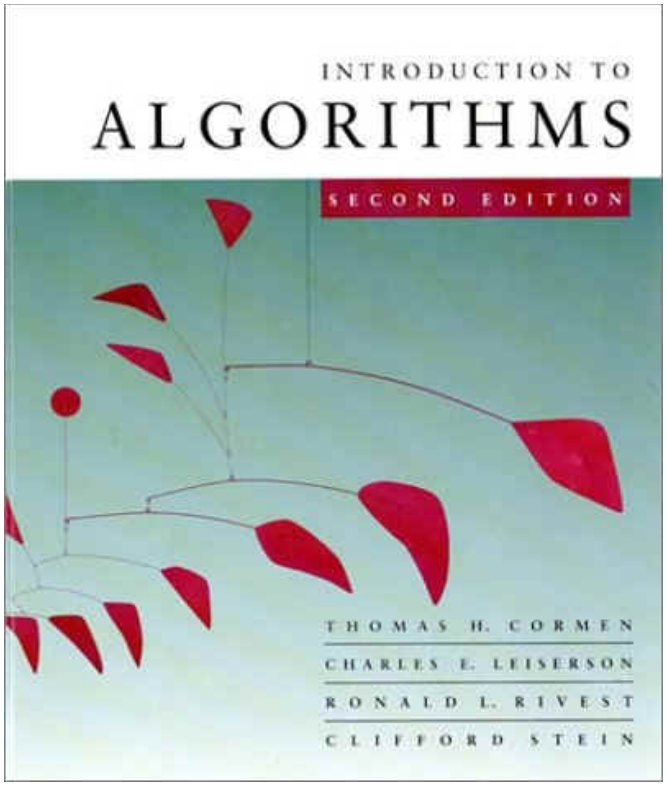
For $k = 0, 1, \ldots, n-1$, define the ***indicator random variable***



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.

[CLRS]…



Sabbatical leave at NUS
Computer Science Dept 1995/96

[CLRS] & Charles Leiserson.

Thank you.

Q & A

School of Computing