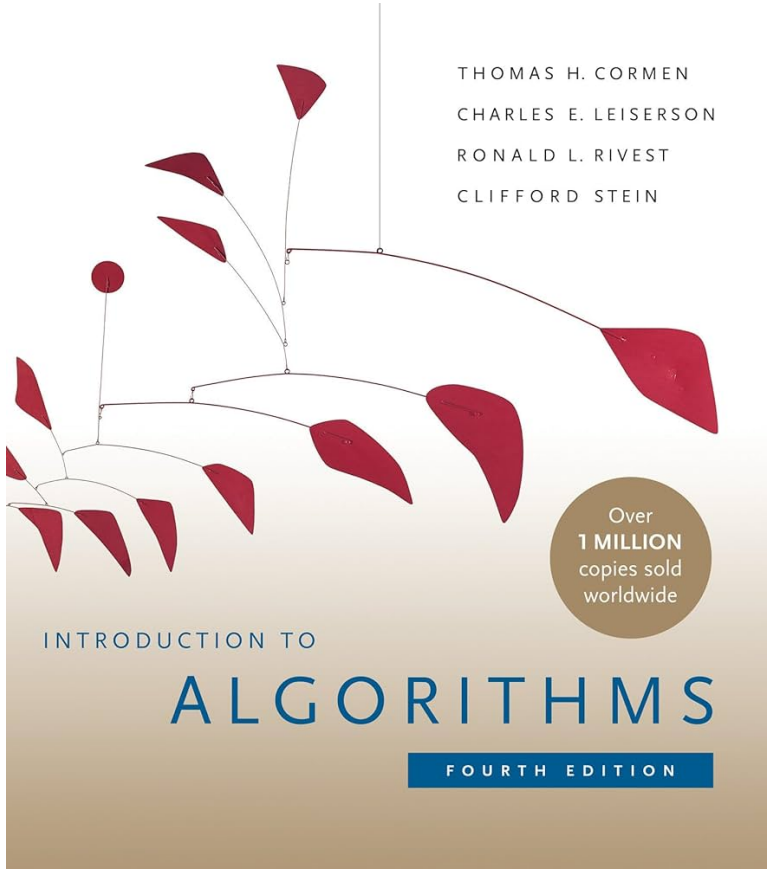
Introduction to Algorithms
Lecture 4: Quicksort

Prof. Charles E. Leiserson and Prof. Erik Demaine
Massachusetts Institute of Technology

February 18, 2025

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.
Visit `https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/`.
Original slides from *Introduction to Algorithms 6.046J/18.401J*, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at `https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/`.

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.

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

Key:

Linear-time partitioning subroutine.

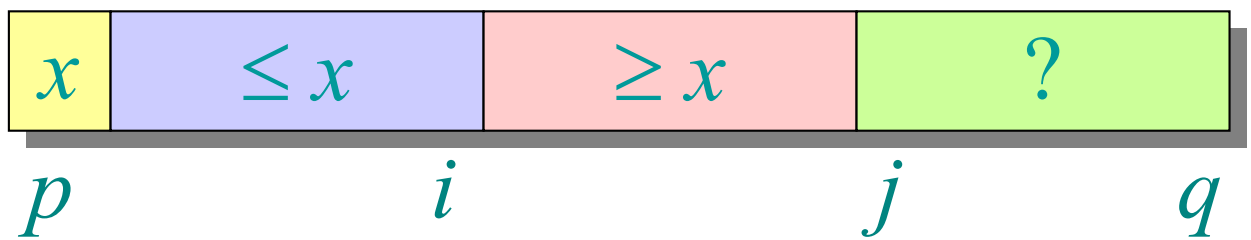
Partitioning Subroutine

```
 1: procedure PARTITION(A, p, q)
 2:     x ← A[p]
 3:     i ← p
 4:     for j ← p + 1 to q do
 5:         if A[j] ≤ x then
 6:             i ← i + 1
 7:             exchange A[i] ↔ A[j]
 8:         end if
 9:     end for
10:     exchange A[p] ↔ A[i]
11:     return i
12: end procedure
```

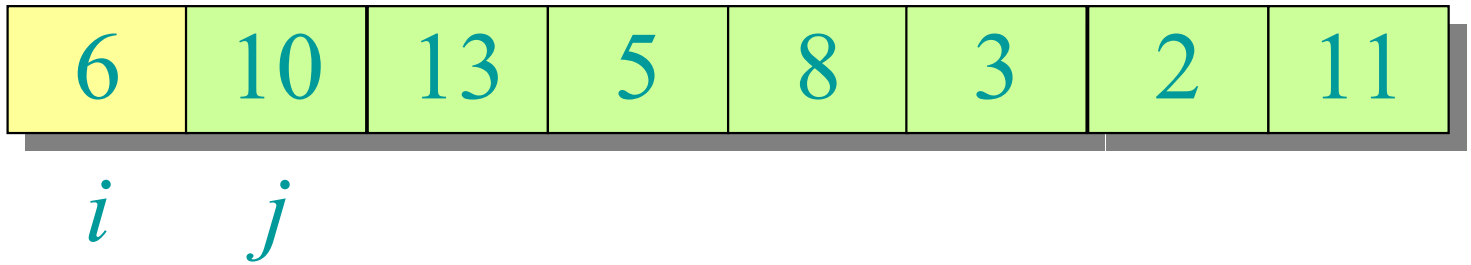
Running time:

$O(n)$ for n elements.

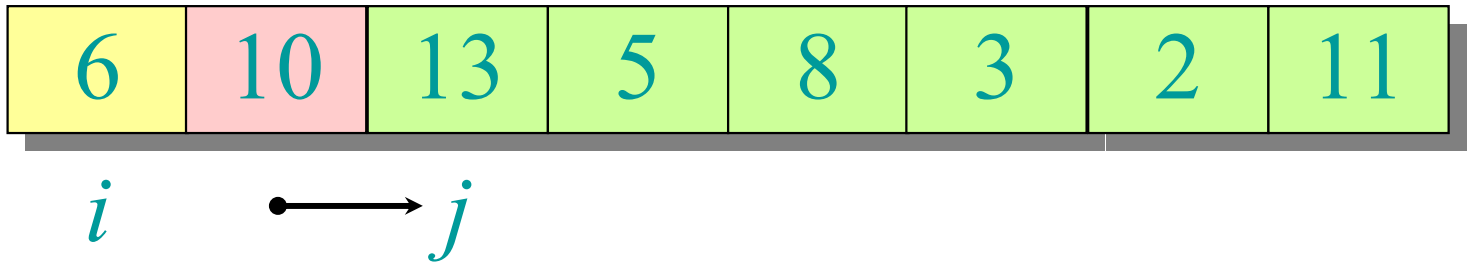
Invariant:



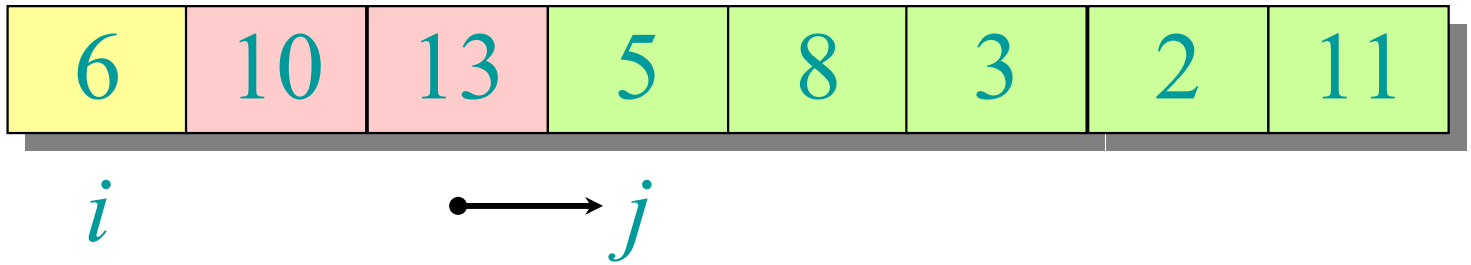
Example of partitioning



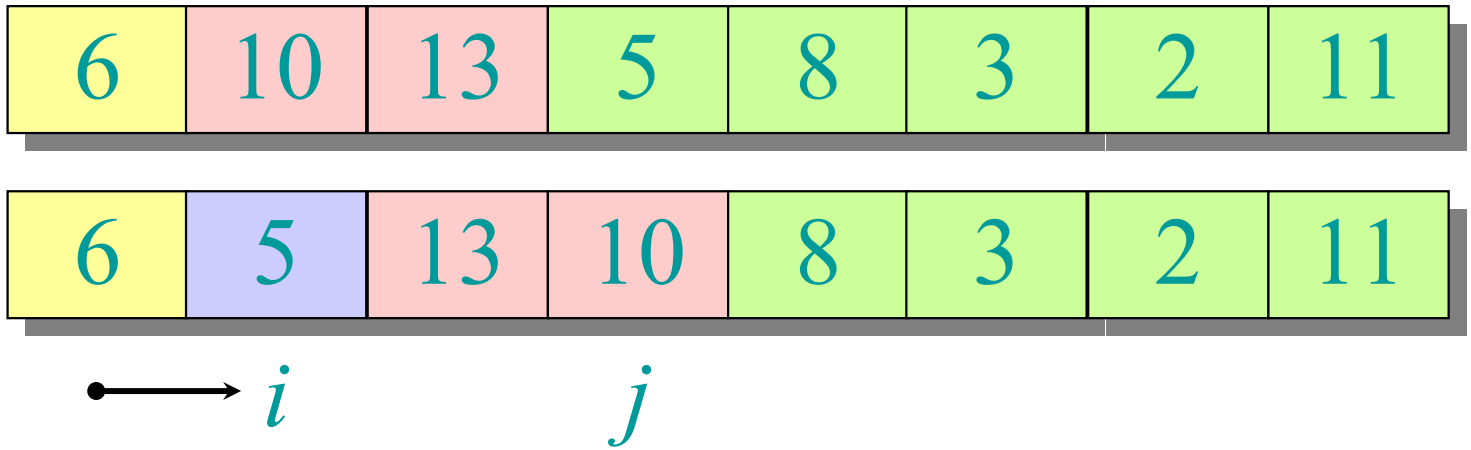
Example of partitioning



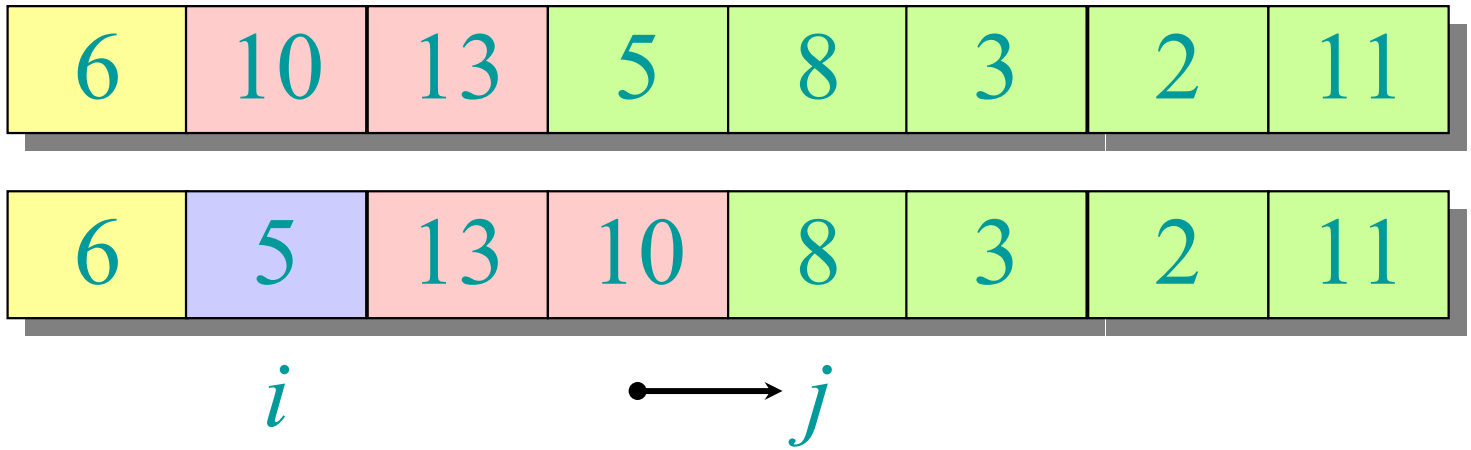
Example of partitioning



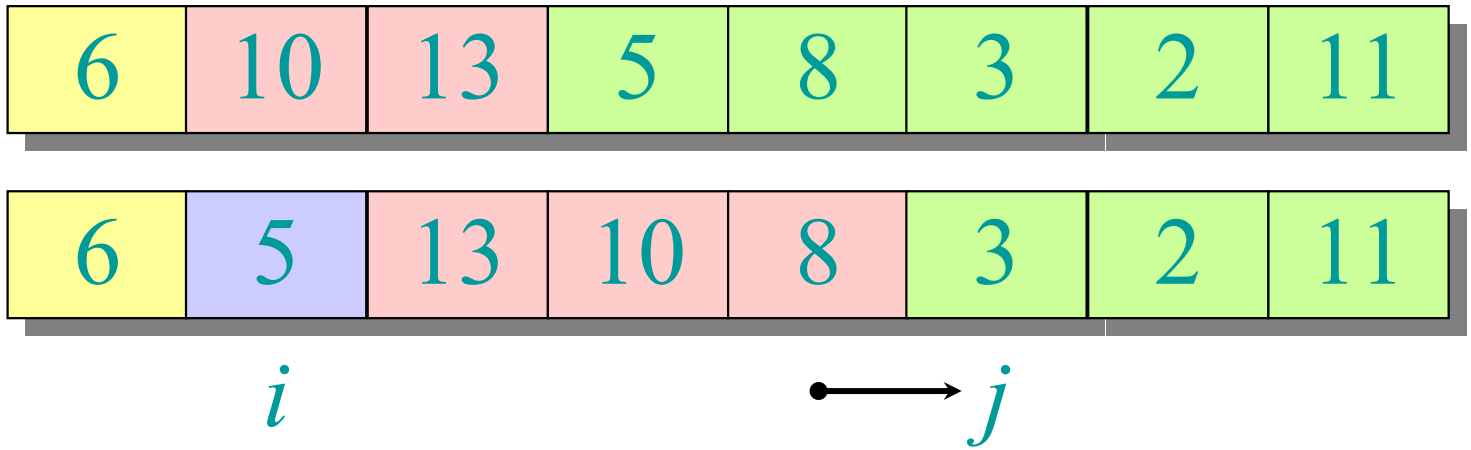
Example of partitioning



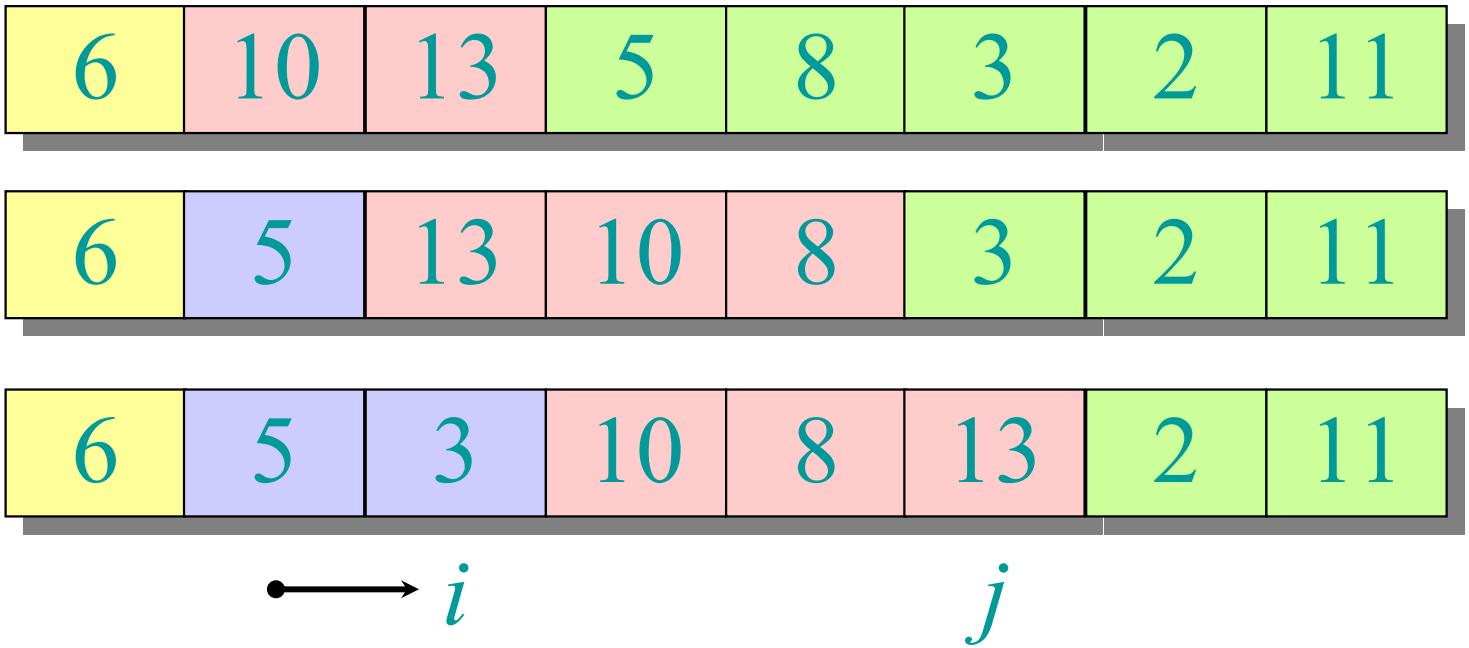
Example of partitioning



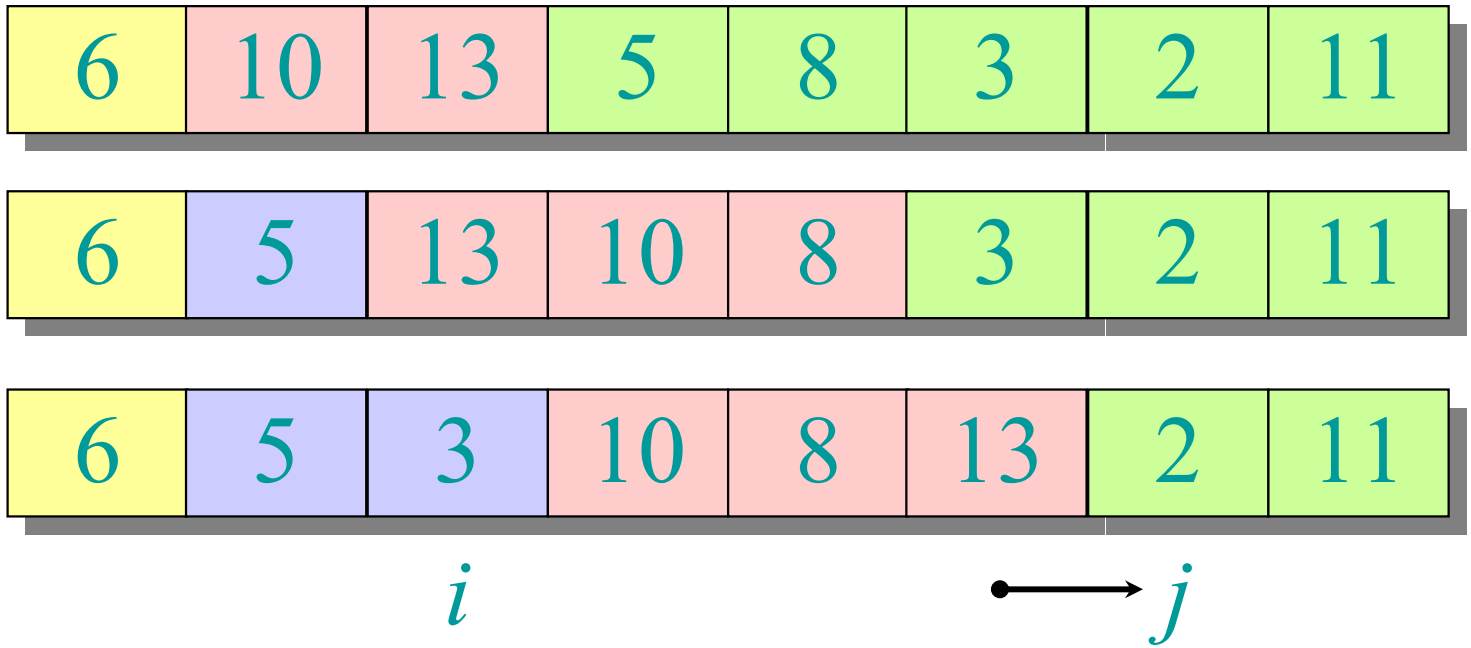
Example of partitioning



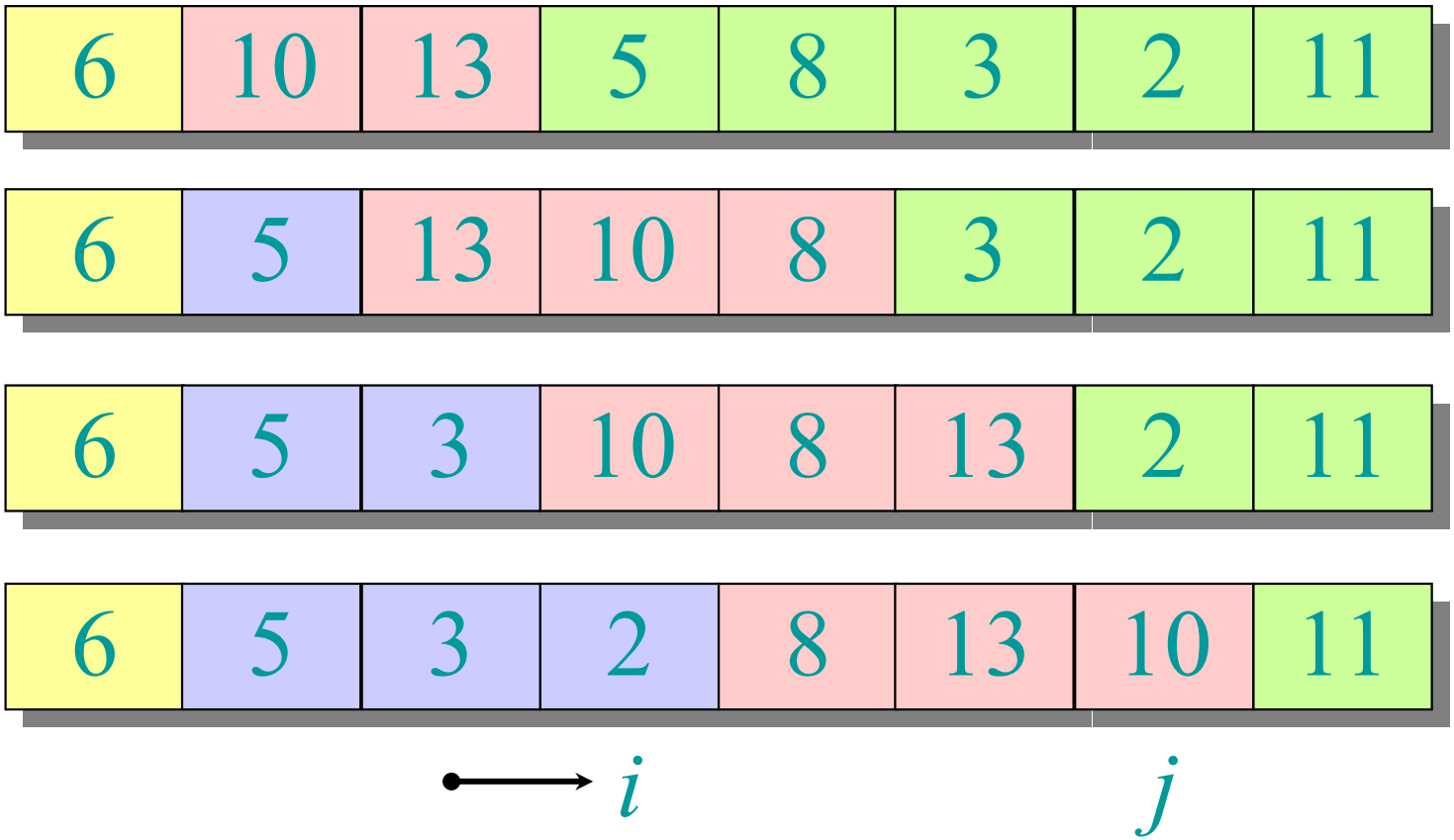
Example of partitioning



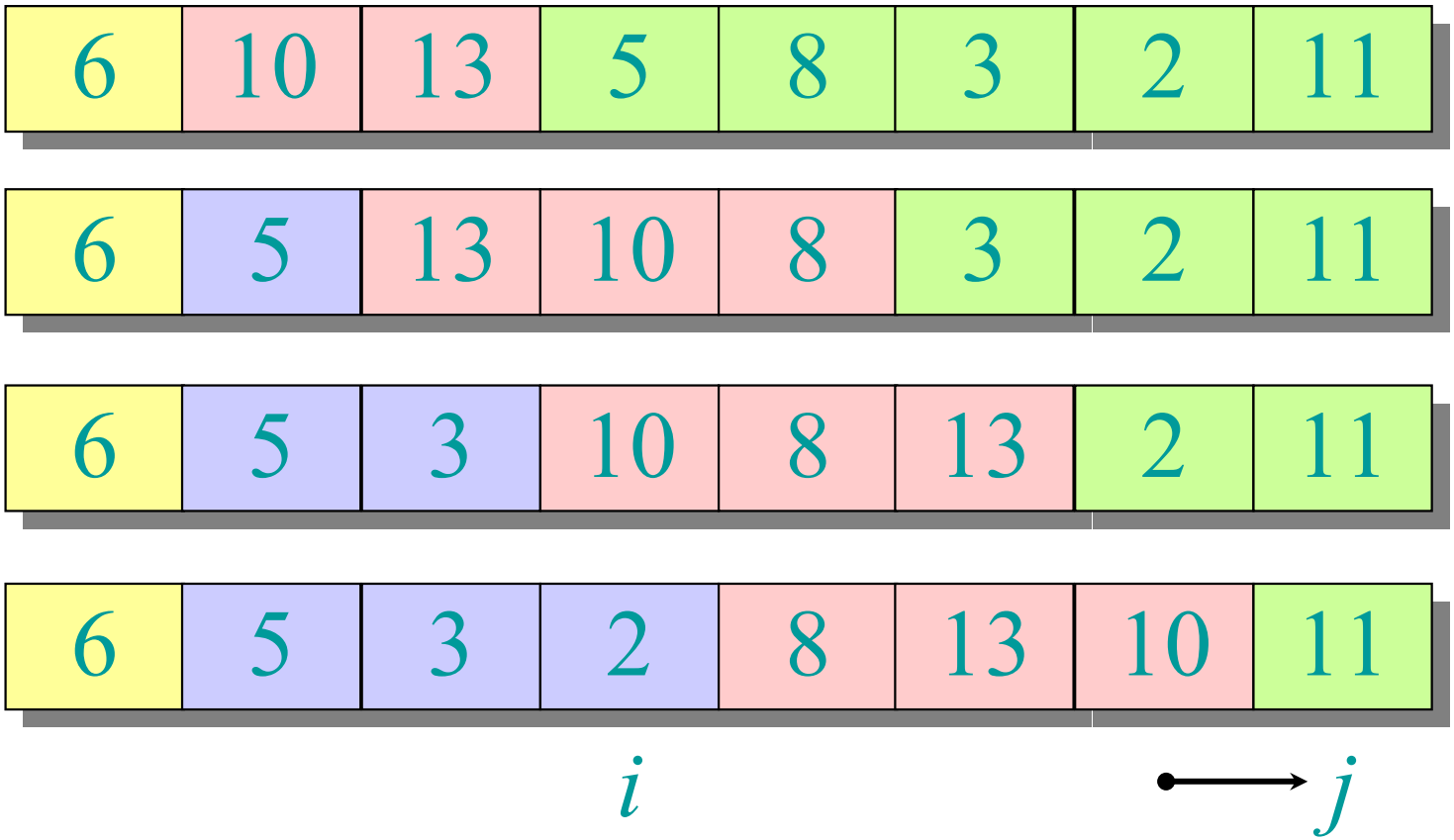
Example of partitioning



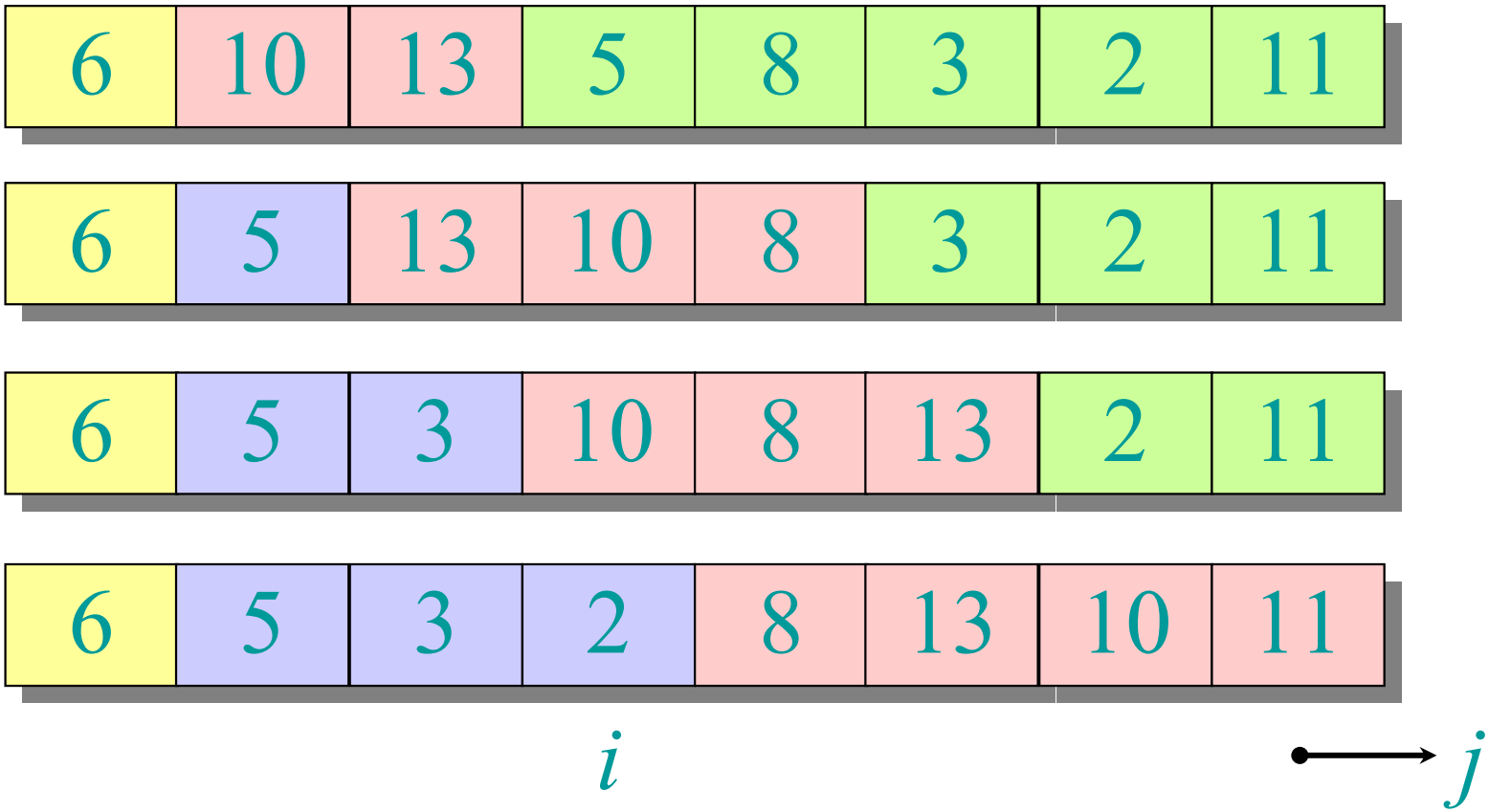
Example of partitioning



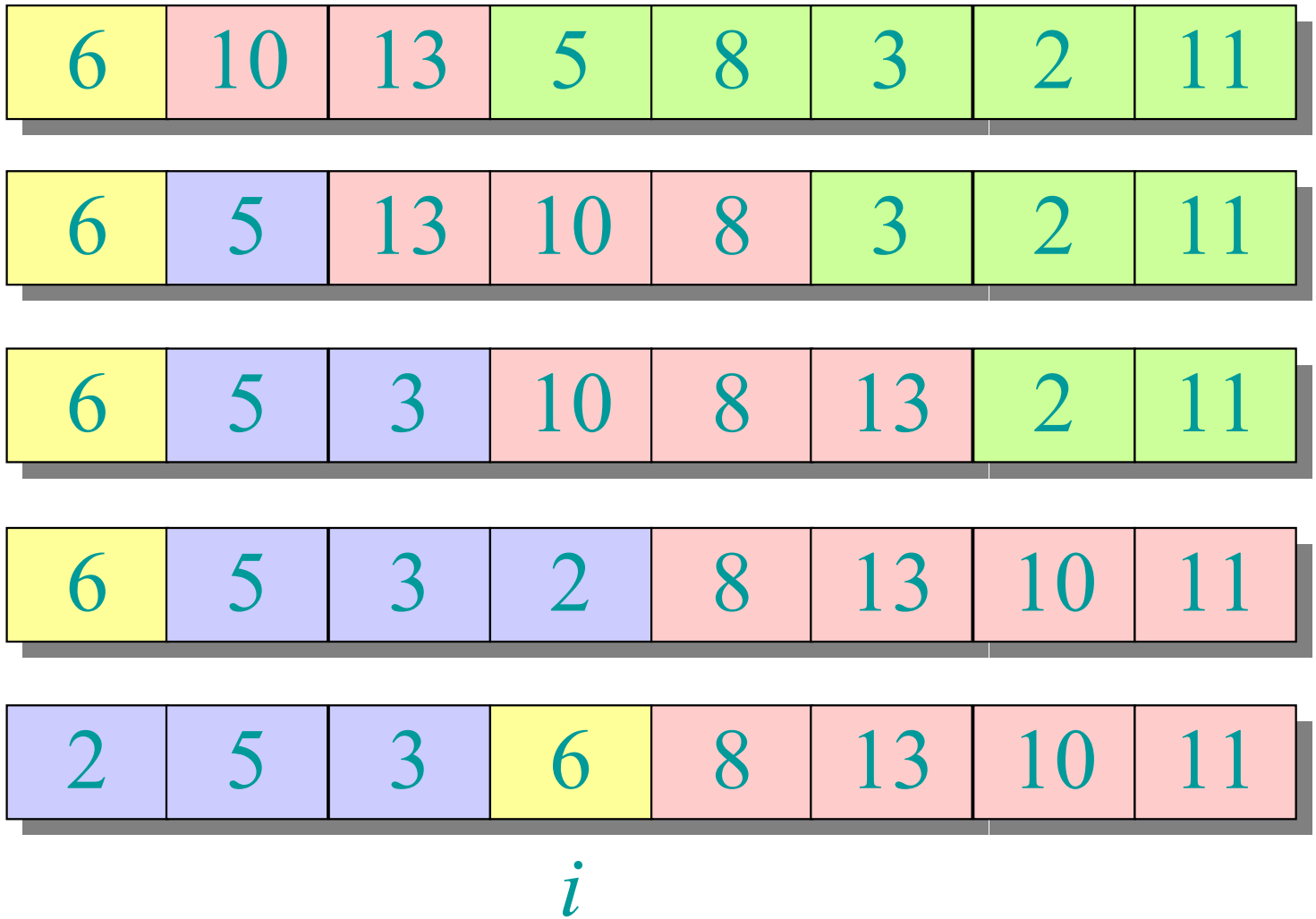
Example of partitioning



Example of partitioning



Example of partitioning



Pseudocode for Quicksort

```
1: procedure QUICKSORT(A, p, r)
2:     if p < r then
3:         q ← PARTITION(A,p, r)
4:         QUICKSORT(A, p, q − 1)
5:         QUICKSORT(A, q + 1, r)
6:     end if
7: end procedure
```

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.

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

Worst-case of Quicksort

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

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

Arithmetic Series!

Worst-case Recursion Tree

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

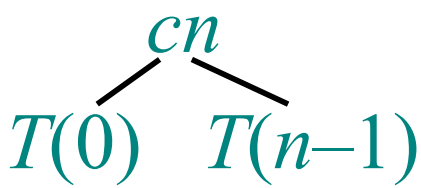
Worst-case Recursion Tree

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

$T(n)$

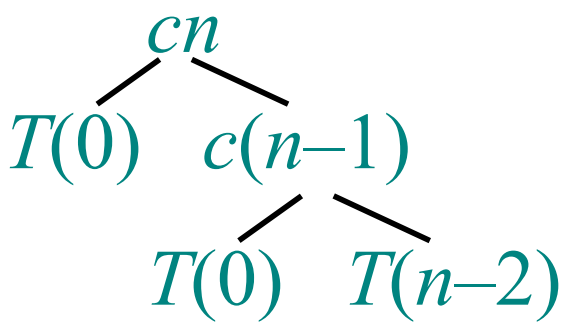
Worst-case Recursion Tree

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



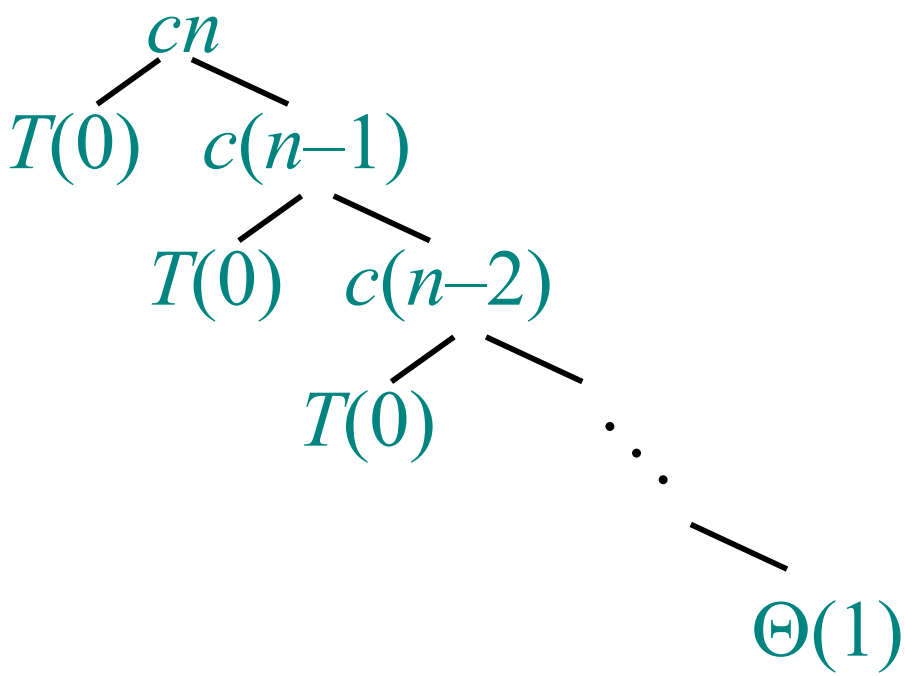
Worst-case Recursion Tree

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



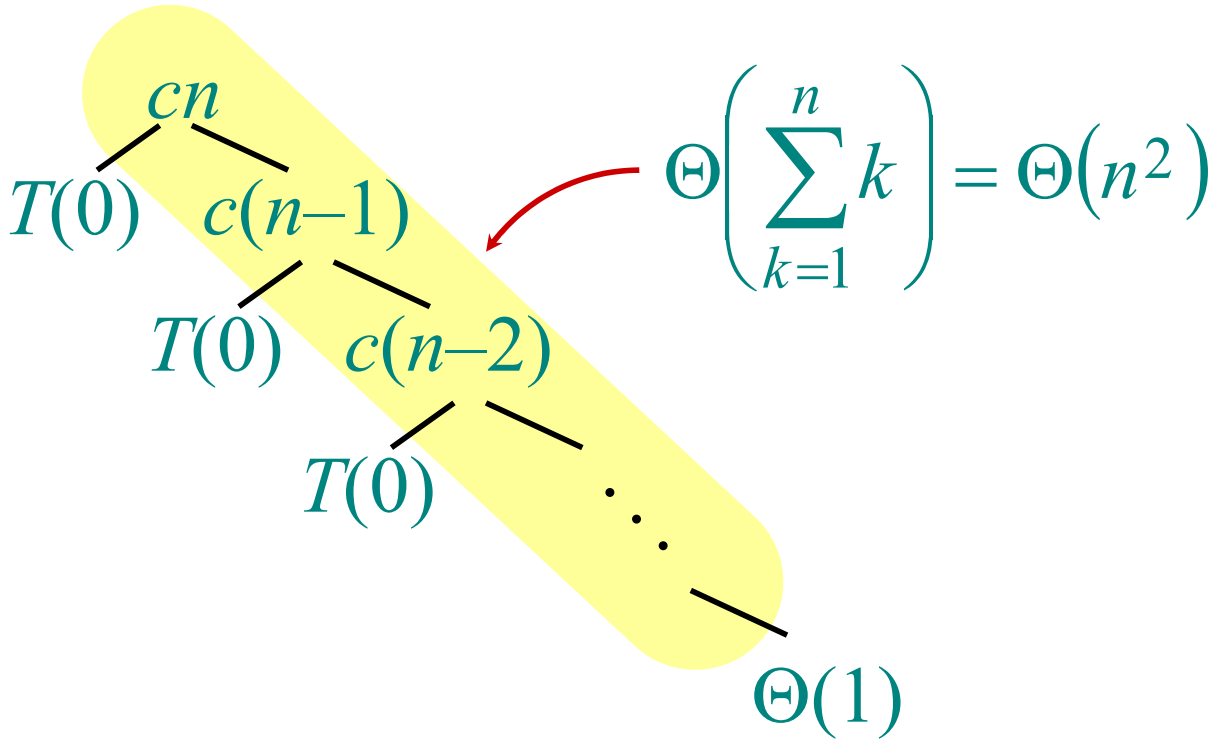
Worst-case Recursion Tree

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



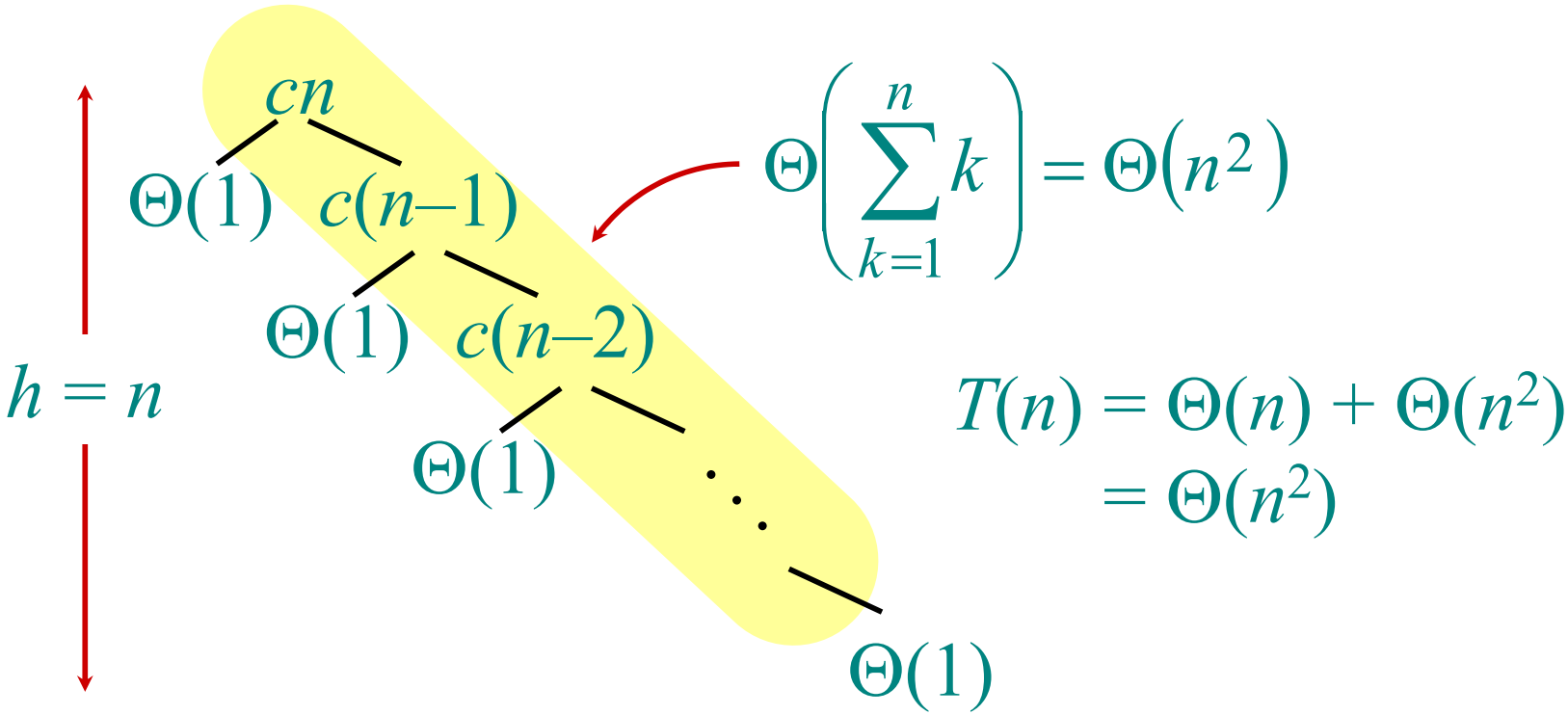
Worst-case Recursion Tree

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



Worst-case Recursion Tree

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



Best-case Analysis

For intuition only!

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

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + \Theta(n) \\ &= \Theta(n \ln n) \qquad \text{(same as merge-sort)} \end{aligned}$$

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

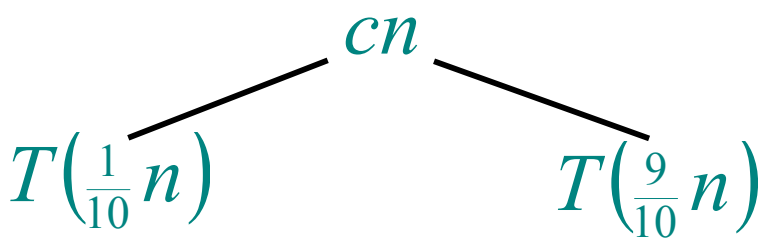
$$T(n) = T\left(\frac{1}{10}n\right) + T\left(\frac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

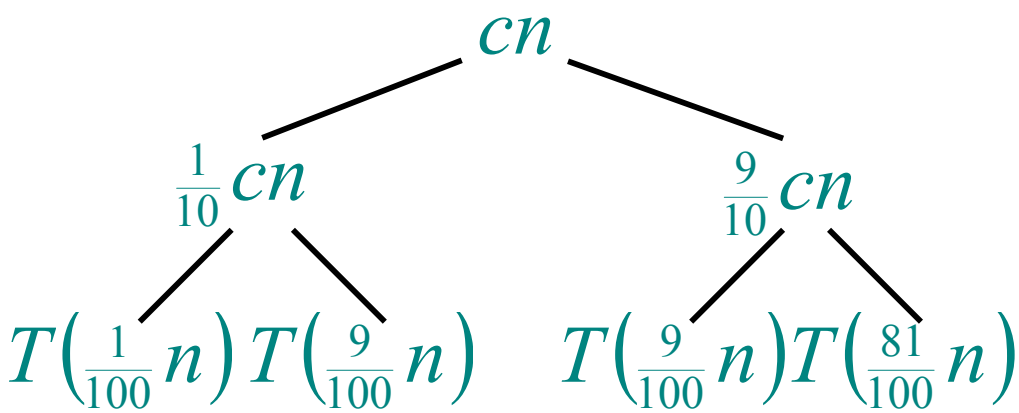
Analysis of "Almost-best" Case

$$T(n)$$

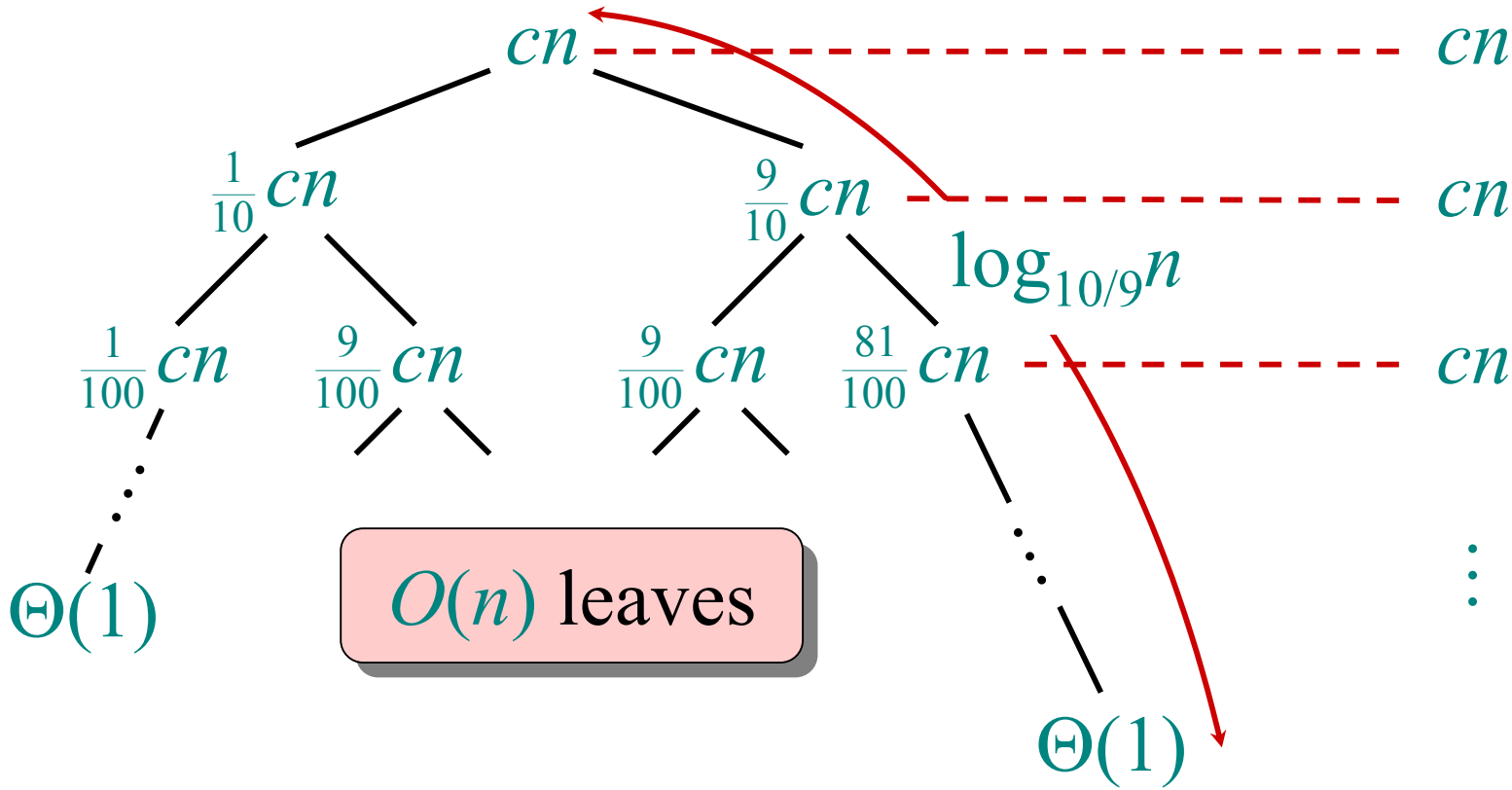
Analysis of "Almost-best" Case



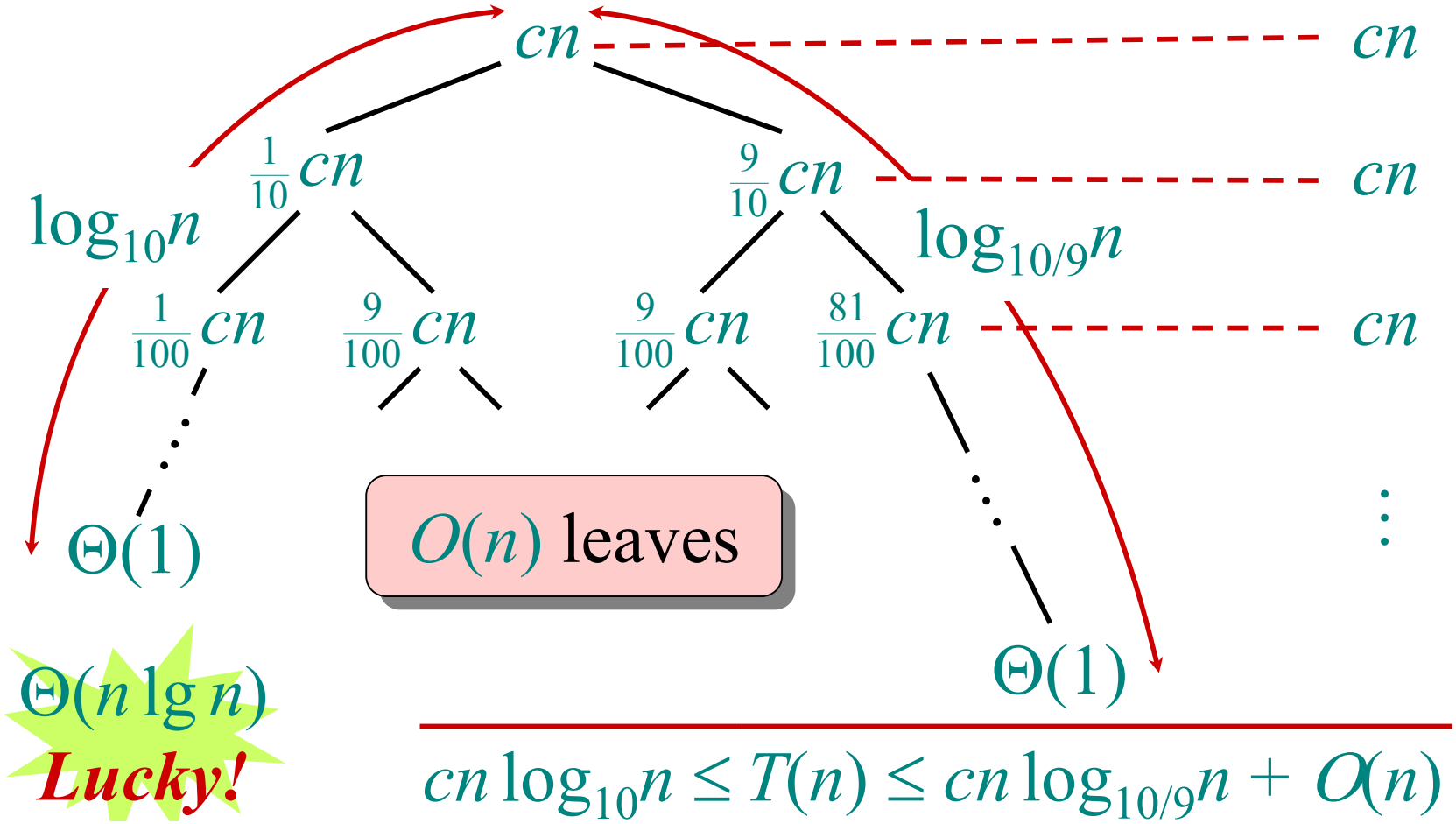
Analysis of "Almost-best" Case



Analysis of "Almost-best" Case



Analysis of "Almost-best" Case



More Intuition

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

$$\begin{aligned} L(n) &= 2U\left(\frac{n}{2}\right) + \Theta(n) && \text{luckcy} \\ U(n) &= L\left(n-1\right) + \Theta(n) && \text{unluckcy} \end{aligned}$$

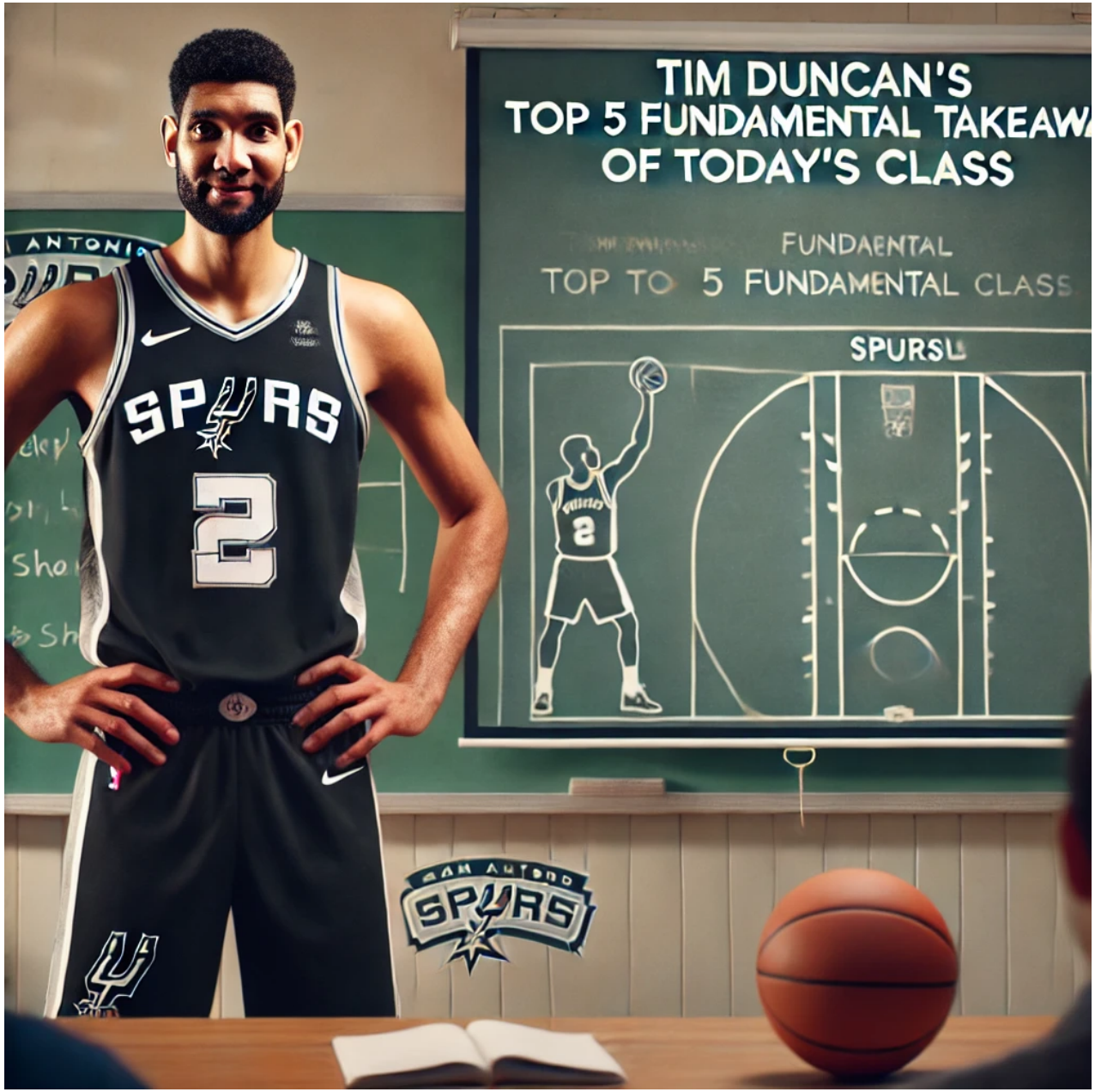
Solving:

$$\begin{aligned} L(n) &= 2\left(L\left(\frac{n}{2}-1\right) + \Theta(\frac{n}{2})\right) + \Theta(n) \\ &= 2L\left(\frac{n}{2}-1\right) + \Theta(n) \\ &= \Theta(n \ln n) && \textbf{lucky!} \end{aligned}$$

How can we make sure we are usually lucky?

End of Lecture 4.

TDT5FTOTTC


TIM DUNCAN'S
TOP 5 FUNDAMENTAL TAKEAW
OF TODAY'S CLASS
FUNDAENTAL
TOP TO 5 FUNDAMENTAL CLASS
SPURS
SPURS
SPURS

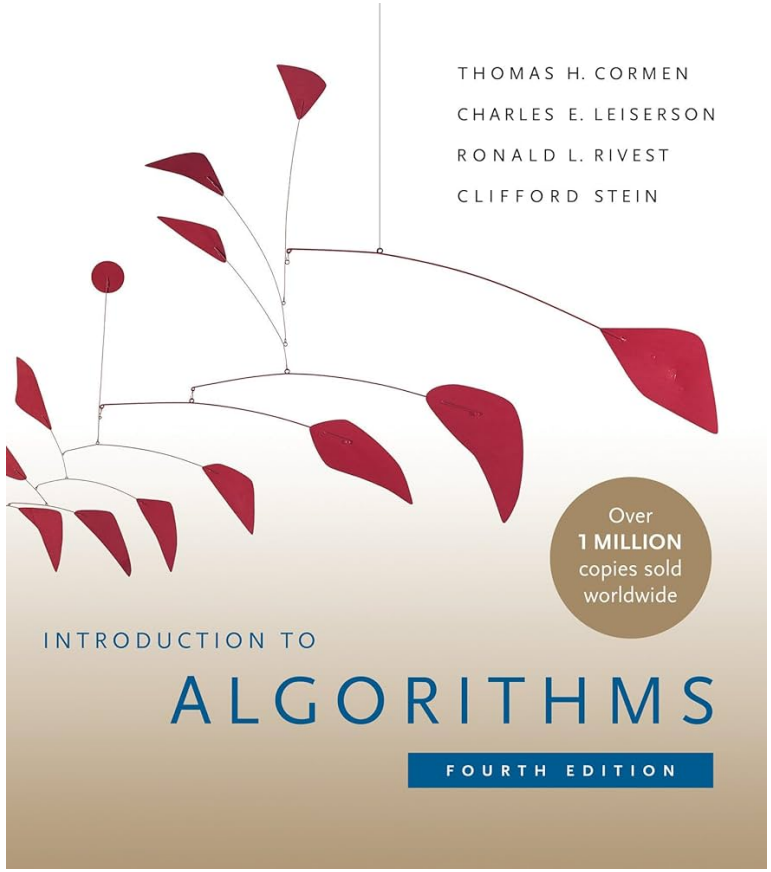
Top 5 Fundamental Takeaways

Top 5 Fundamental Takeaways

Top 5 Fundamental Takeaways

5

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.
Visit `https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/`.
Original slides from *Introduction to Algorithms 6.046J/18.401J*, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at `https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/`.