

# CMPS 6610/4610 – Fall 2016

## *Quicksort*

**Carola Wenk**

Slides courtesy of Charles Leiserson with additions  
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).
- 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.



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

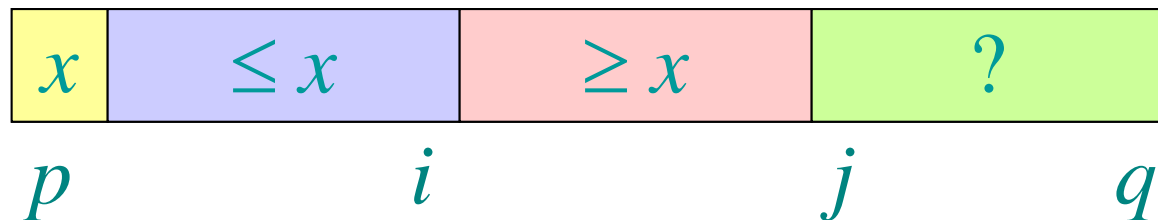
**Key:** *Linear-time partitioning subroutine.*

# Partitioning subroutine

```
PARTITION( $A, p, q$ )  $\triangleright A[p \dots q]$   
   $x \leftarrow A[p]$   $\triangleright$  pivot =  $A[p]$   
   $i \leftarrow p$   
  for  $j \leftarrow p + 1$  to  $q$   
    do if  $A[j] \leq x$   
      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.

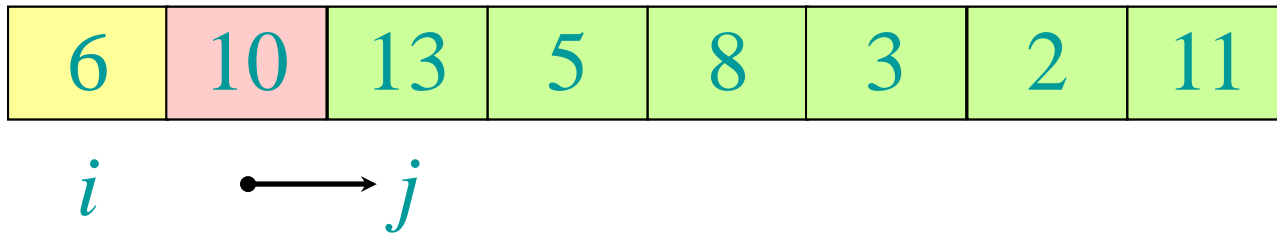
***Invariant:***



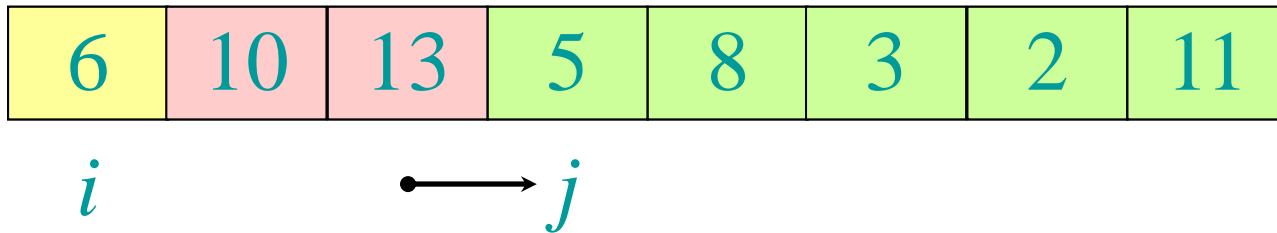
# Example of partitioning

6	10	13	5	8	3	2	11
<i>i</i>	<i>j</i>						

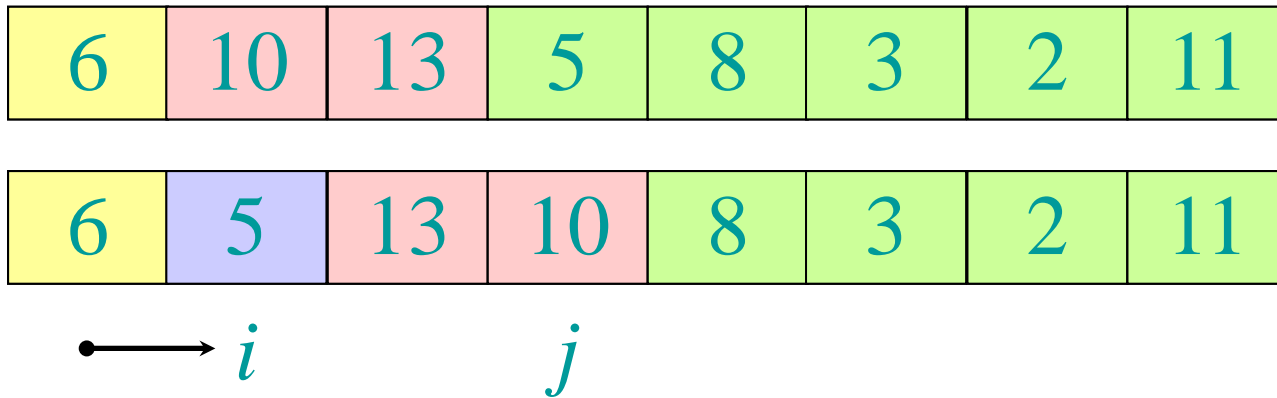
# Example of partitioning



# Example of partitioning

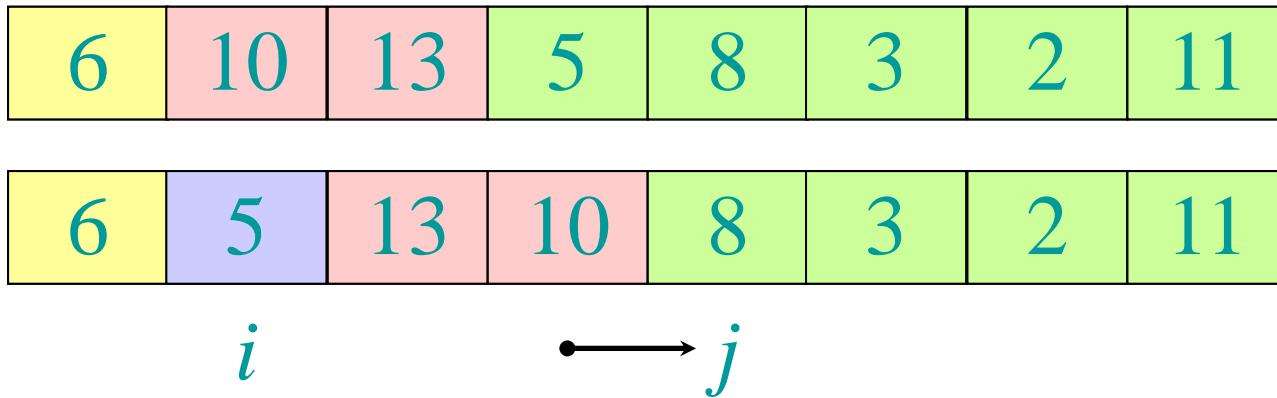


# Example of partitioning

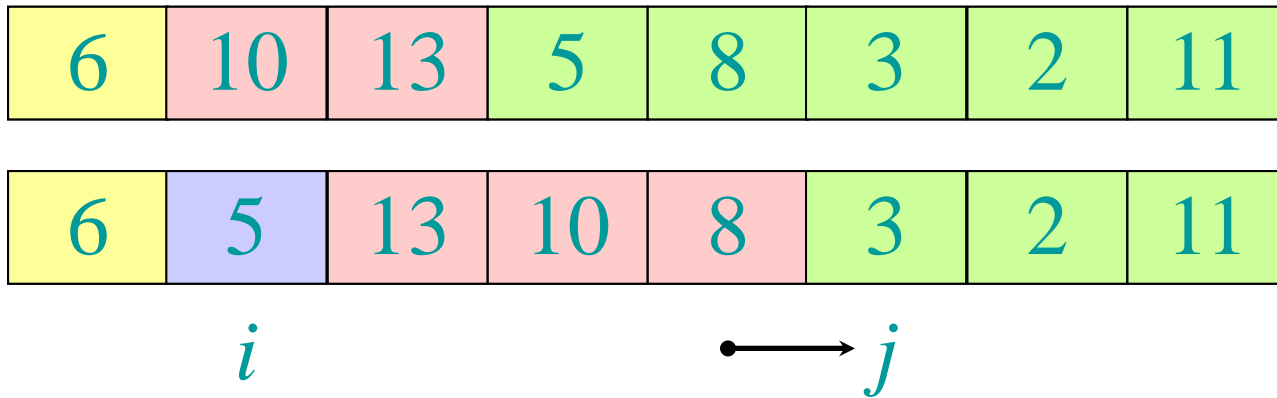




# Example of partitioning



# 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
---	---	---	----	---	----	---	----

•  $\longrightarrow$   $i$   $j$

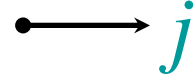
# Example of partitioning

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

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

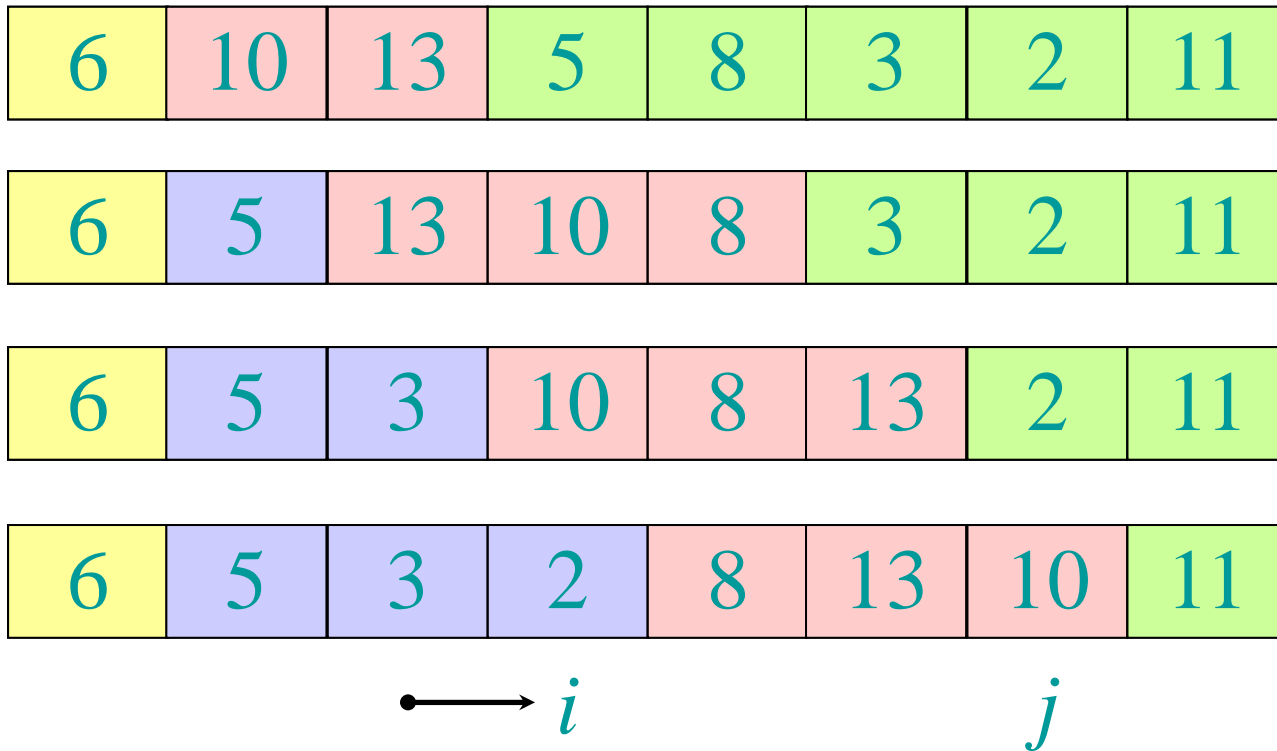
6	5	3	10	8	13	2	11
---	---	---	----	---	----	---	----

$i$

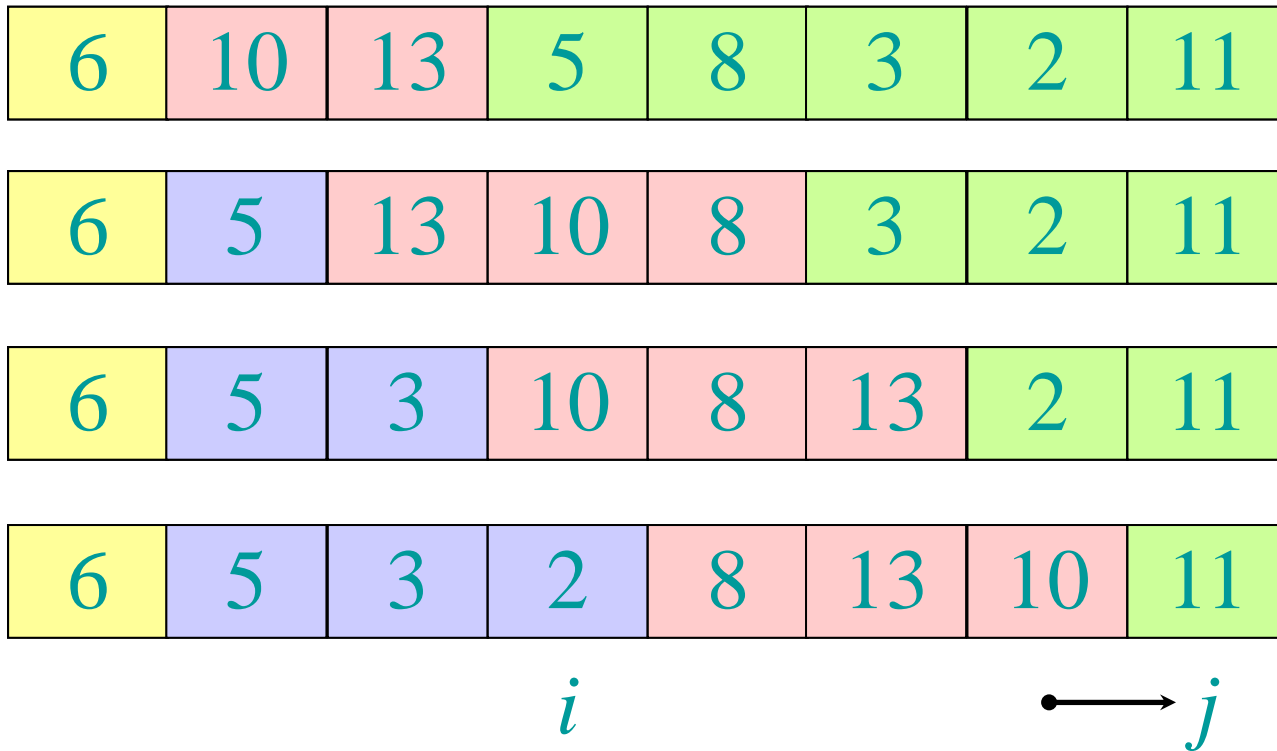


$j$

# Example of partitioning



# 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
---	---	---	---	---	----	----	----

$i$

$\longrightarrow j$

# 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.

# Deterministic Algorithms

Runtime for deterministic algorithms with input size  $n$ :

- Worst-case runtime

→ Attained by one input of size  $n$

- Best-case runtime

→ Attained by one input of size  $n$

- Average runtime

→ Averaged **over all possible inputs** of size  $n$

# Worst-case of quicksort

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

- Let  $T(n)$  = worst-case running time on an array of  $n$  elements.
- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.
- $T(n) = T(0) + T(n - 1) + \Theta(n)$

# Worst-case recursion tree

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

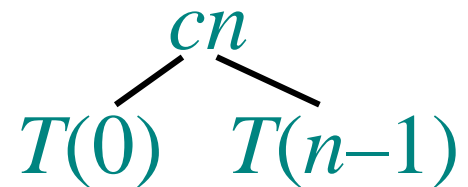
# Worst-case recursion tree

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

$T(n)$

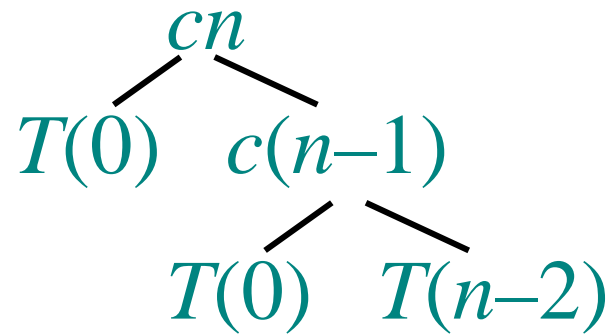
# Worst-case recursion tree

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



# Worst-case recursion tree

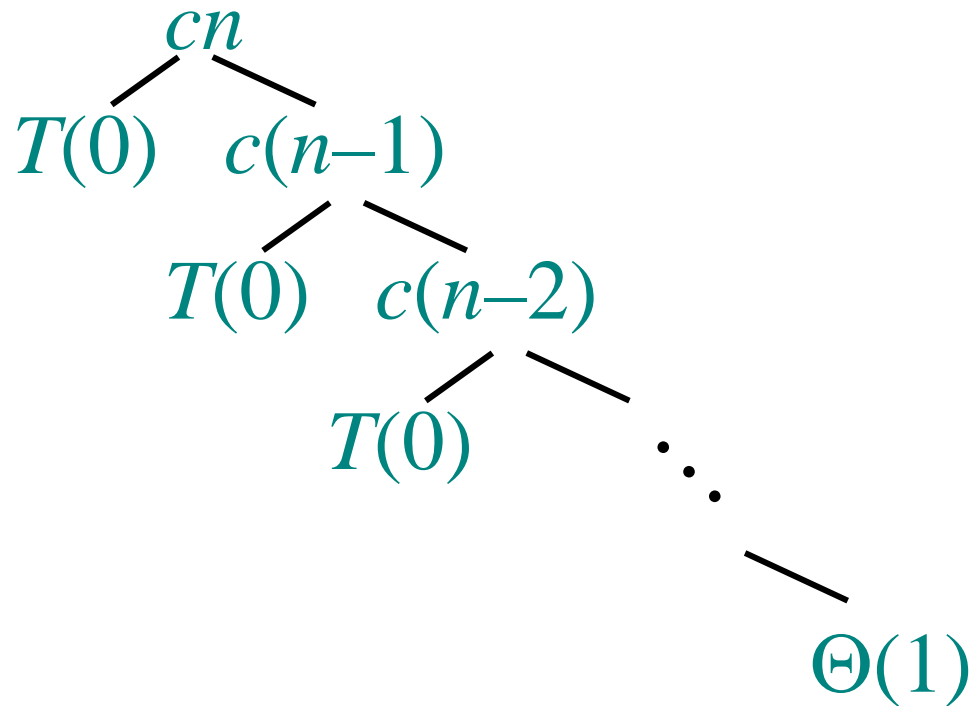
$$T(n) = T(0) + T(n-1) + cn$$





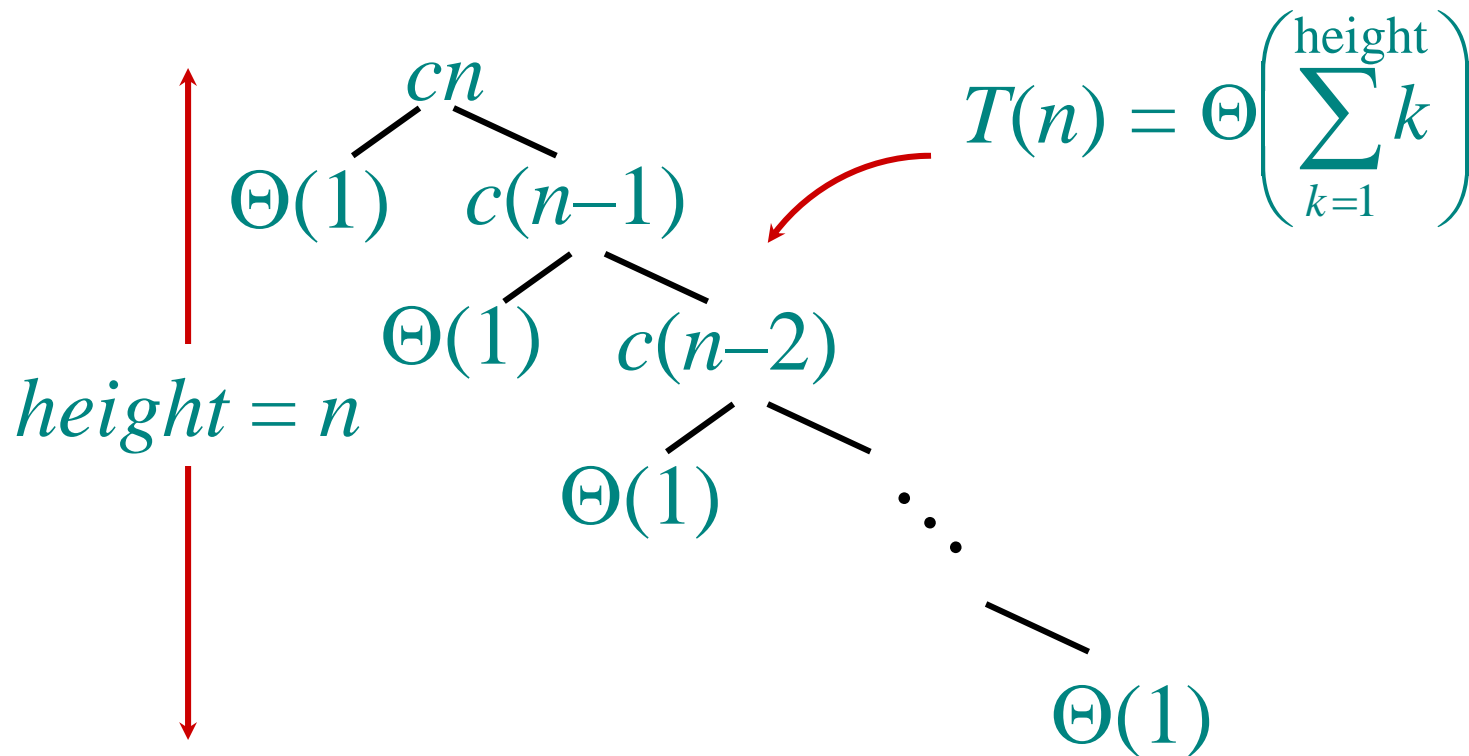
# Worst-case recursion tree

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



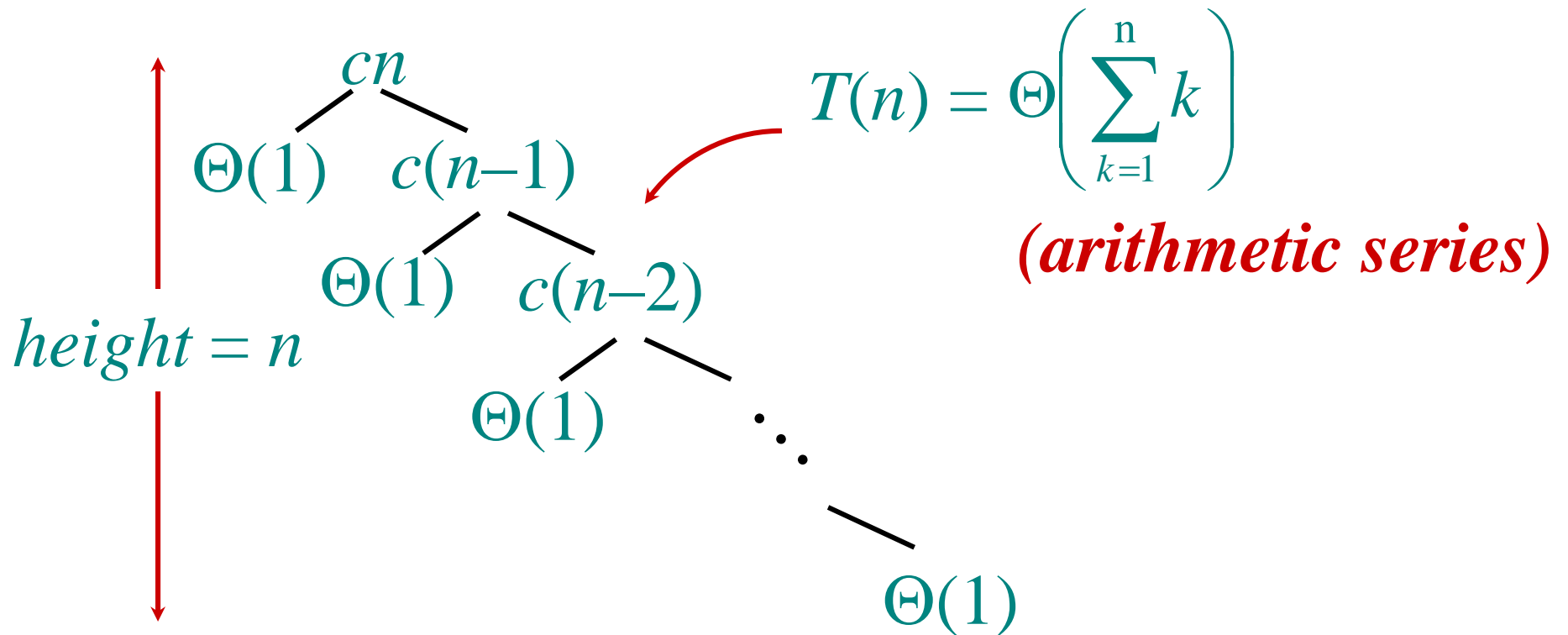
# Worst-case recursion tree

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



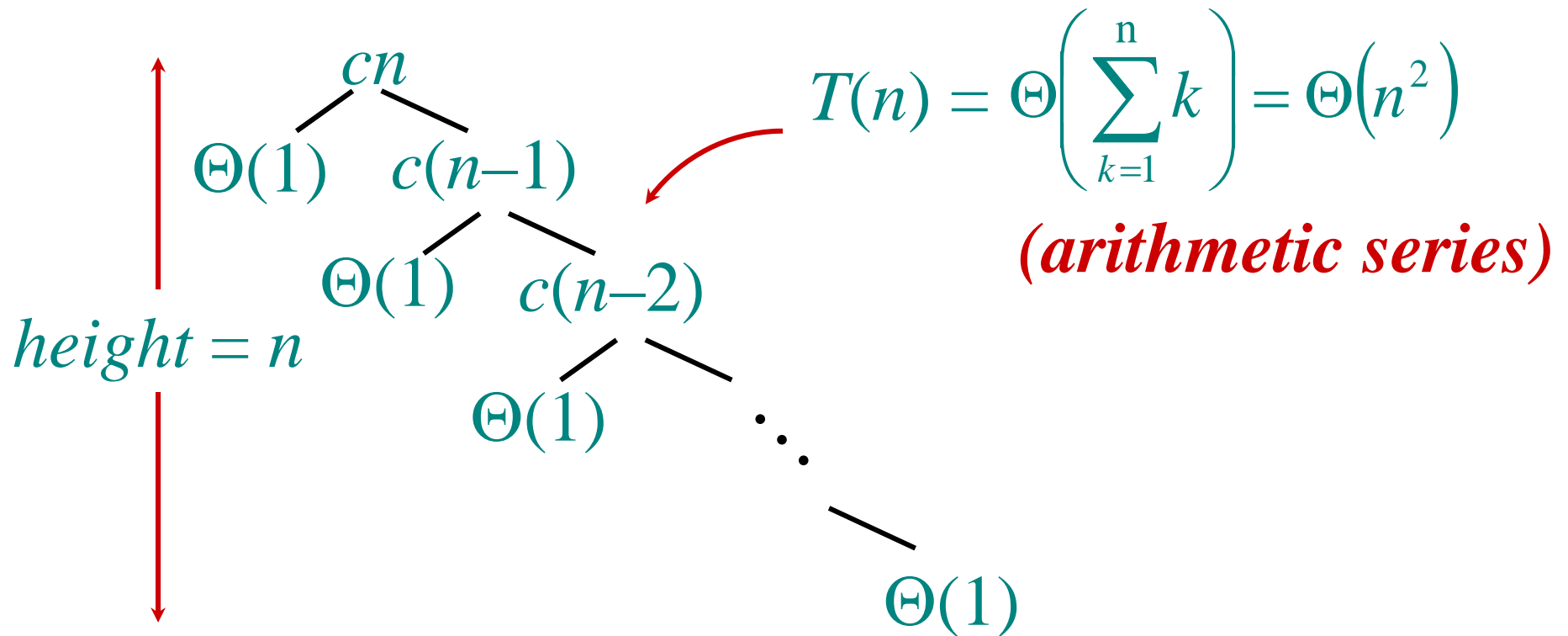
# Worst-case recursion tree

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



# Worst-case recursion tree

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



# Deterministic Algorithms

Runtime for deterministic algorithms with input size  $n$ :

- Worst-case runtime:  $O(n^2)$ 
  - Attained by input:  $[1, 2, 3, \dots, n]$  or  $[n, n-1, \dots, 2, 1]$
- Best-case runtime
  - Attained by one input of size  $n$
- Average runtime
  - Averaged **over all possible inputs** of size  $n$

# Best-case analysis

*(For intuition only!)*

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

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

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

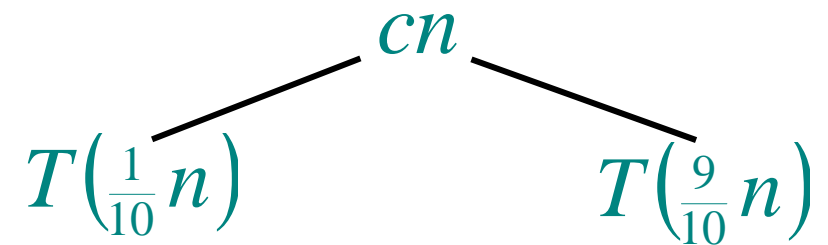
$$T(n) = T\left(\frac{1}{10}n\right) + T\left(\frac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

# Analysis of “almost-best” case

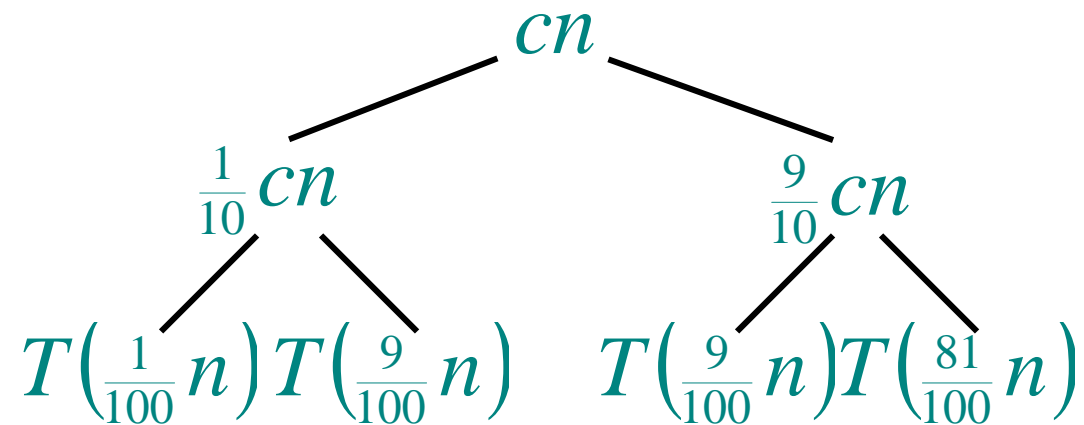
$$T(n)$$

# Analysis of “almost-best” case

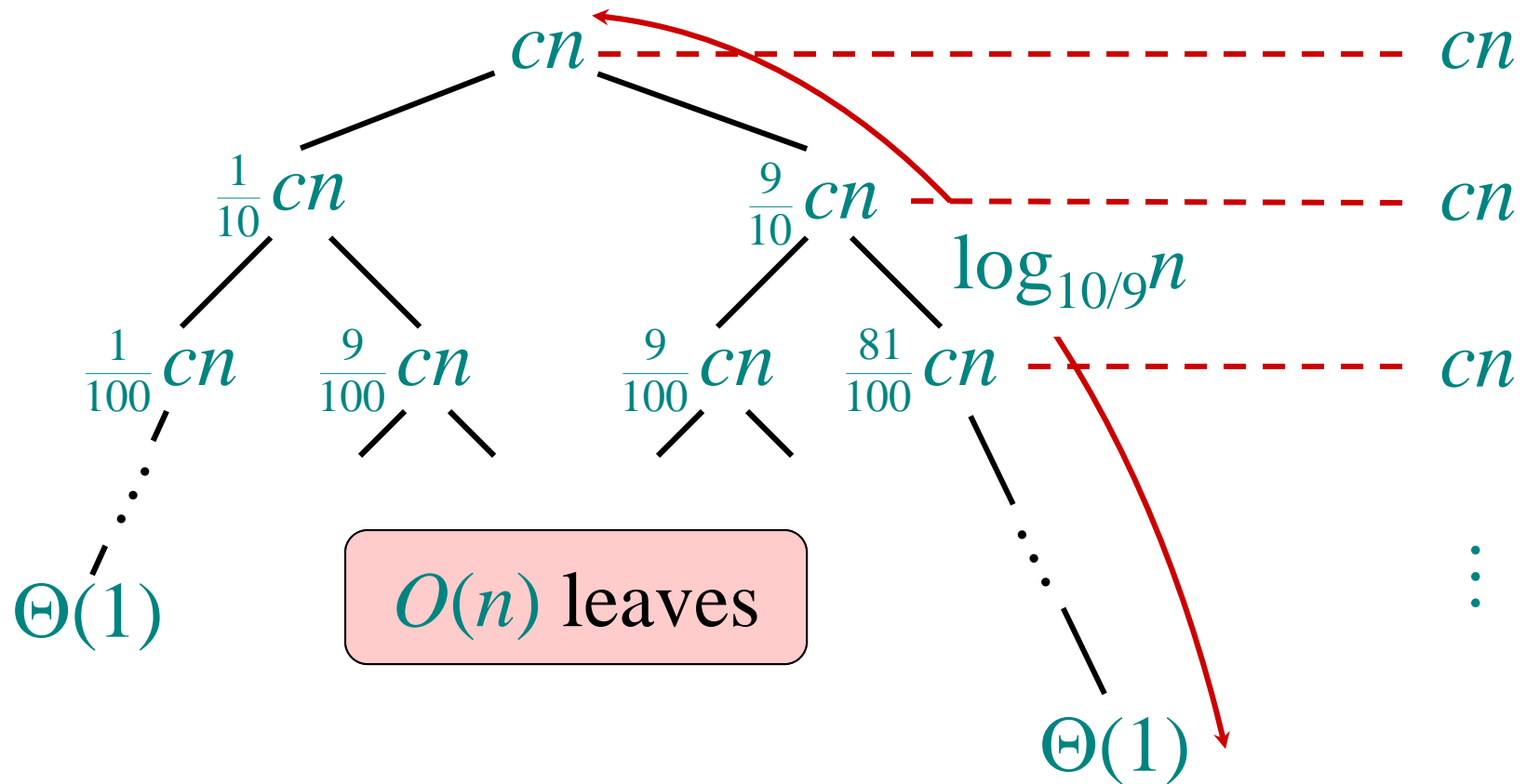




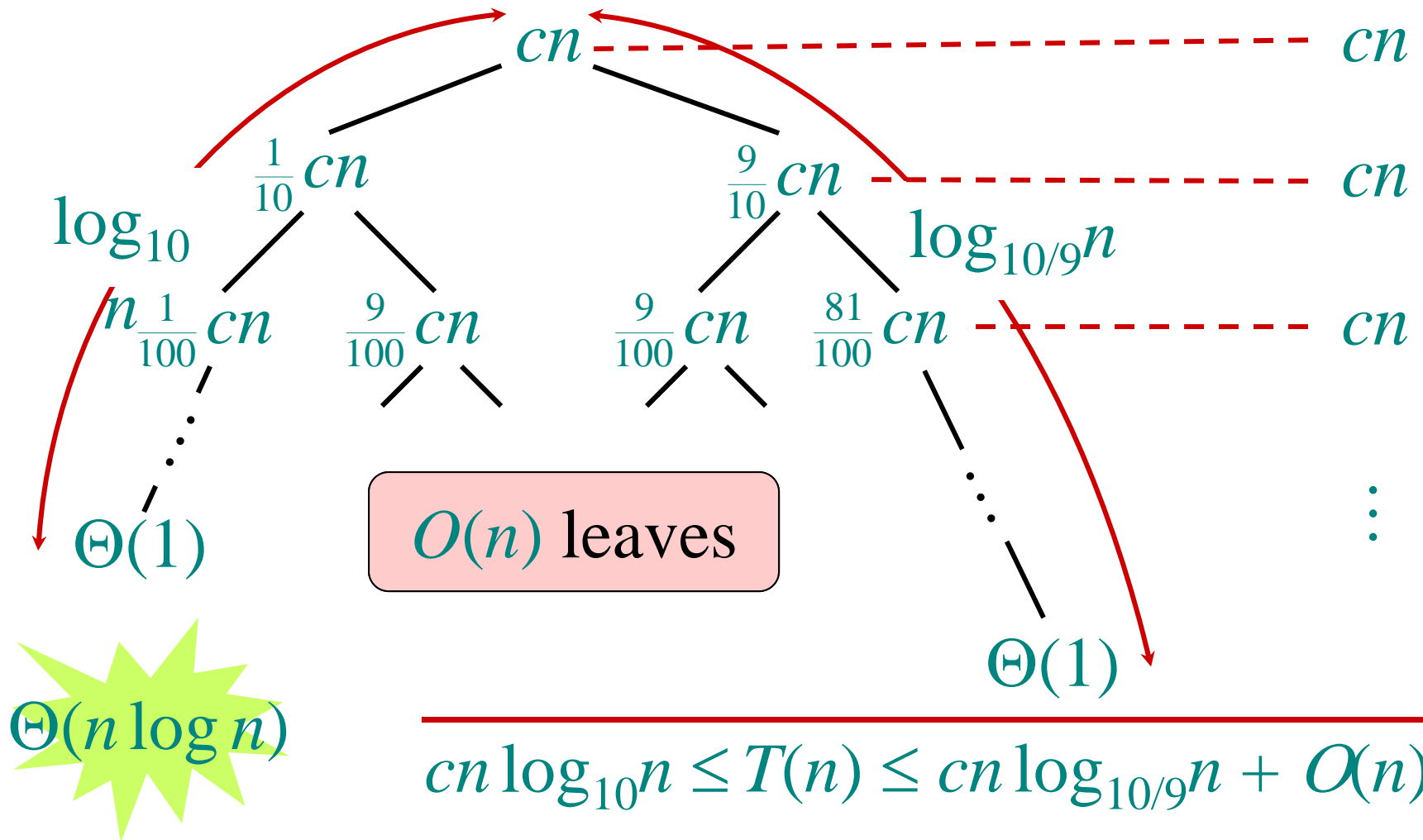
# Analysis of “almost-best” case



# Analysis of “almost-best” case



# Analysis of “almost-best” case



# Deterministic Algorithms

Runtime for deterministic algorithms with input size  $n$ :

- Worst-case runtime:  $O(n^2)$ 
  - ➔ Attained by input:  $[1, 2, 3, \dots, n]$  or  $[n, n-1, \dots, 2, 1]$
- Best-case runtime:  $O(n \log n)$ 
  - ➔ Attained by input of size  $n$  that splits evenly or  $\frac{1}{10} : \frac{9}{10}$  at every recursive level
- Average runtime
  - ➔ Averaged **over all possible inputs** of size  $n$

# Average Runtime

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

⇒ Average over all  $n!$  input permutations.

# Average Runtime: Quicksort

- The average runtime averages runtimes over all  $n!$  different input permutations
  - One can show that the average runtime for Quicksort is  $O(n \log n)$
  - Disadvantage of considering average runtime:
    - There are still worst-case inputs that will have the worst-case runtime of  $O(n^2)$
    - Are all inputs really equally likely? That depends on the application
- ⇒ **Better:** Use a randomized algorithm

# Randomized quicksort

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

- Running time is independent of the input order. It depends on a probabilistic experiment (sequence  $s$  of numbers obtained from random number generator)
  - ⇒ Runtime is a random variable (maps sequence of random numbers to runtimes)
- **Expected runtime** = expected value of runtime random variable
- 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.

# Quicksort Runtimes

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

# 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.