CH08-320201
Algorithms and Data Structures

Lecture 3 — 13 Feb 2018

Prof. Dr. Michael Sedlmair

Jacobs University
Spring 2018

This & That

- Submission homework 1?
- Moodle: is online now!
  - homework submissions — deadline 23:55
  - discussion // feedback (open or anonymous?)
- Sick homework
  - include a medical excuse in your submission
  - let the TAs know by email
- Today: Part 1
  - Divide&Conquer
  - Merge Sort
- Today: Part 2
  - Recurrences
  - Substitution method / Recursion tree / Master method

1.4 First concept: Divide & Conquer

Divide & Conquer

- Divide&Conquer is a first concept that can produce faster algorithms.
- It is based on three steps:
  - **Divide** the given problem into smaller subproblems.
  - **Conquer** the subproblems by solving them recursively.
  - **Combine** the solutions of the subproblems.
- Example: Sort recursively?

Merge Sort

MERGE-SORT $A[1\,..\,n]$

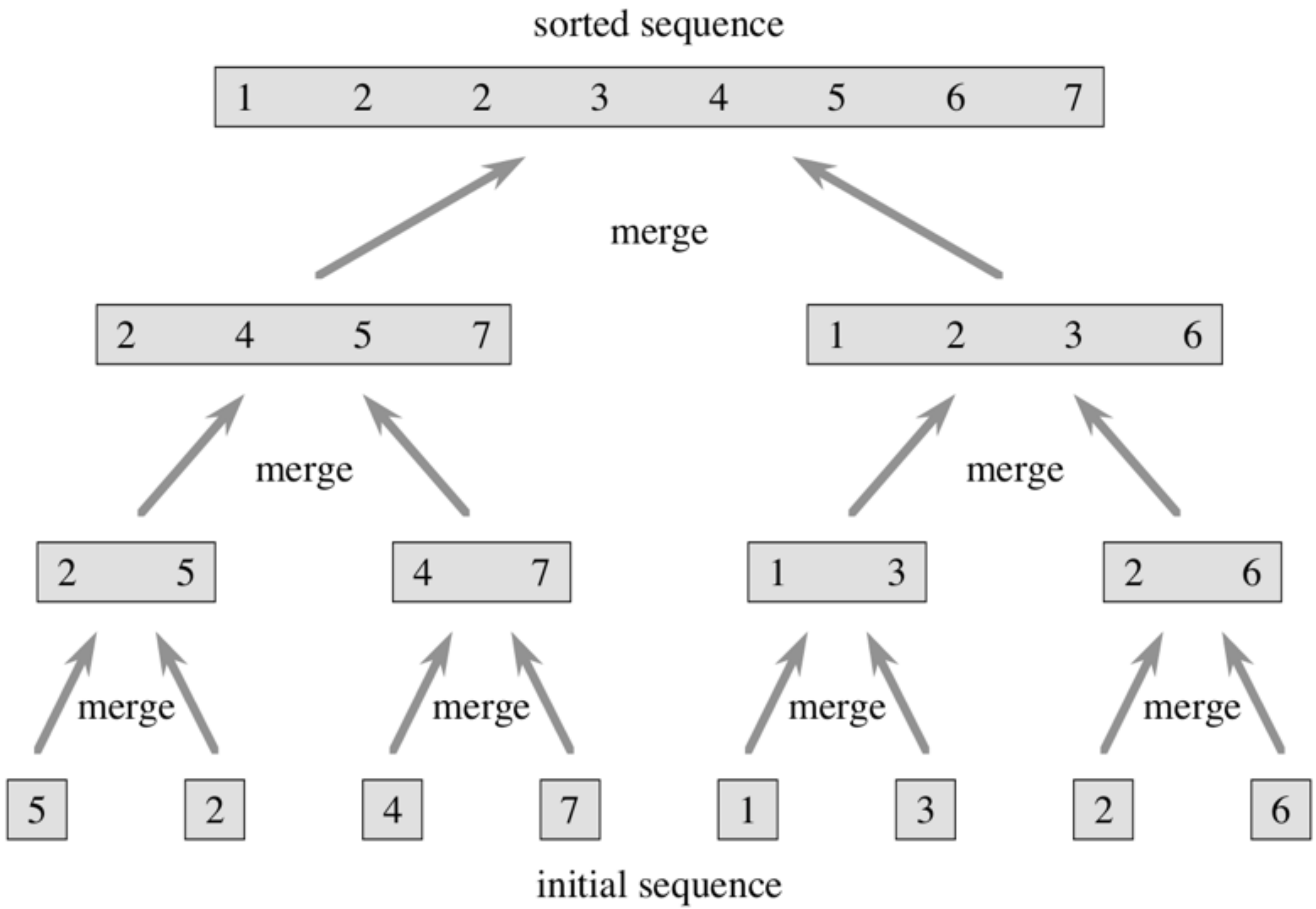
1. If $n = 1$, done.
2. Recursively sort $A[\,1\,..\lceil n/2\rceil\,]$ and $A[\,\lceil n/2\rceil{+}1\,..\,n\,]$.
3. "***Merge***" the 2 sorted lists.

Merge Sort as Divide & Conquer

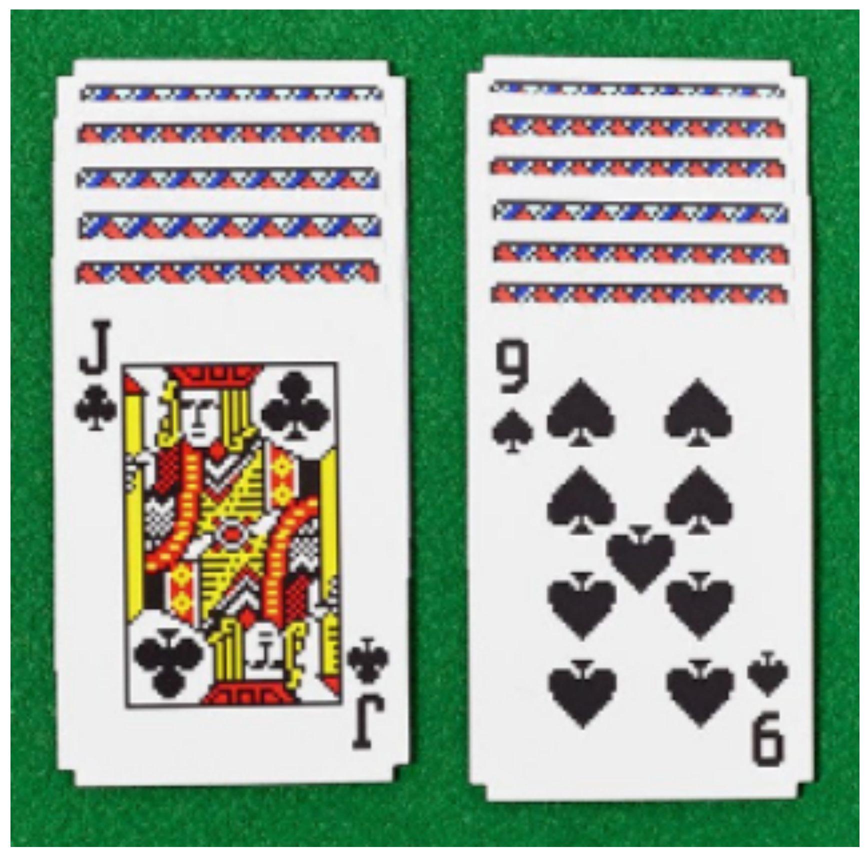
```
MERGE-SORT(A, p, r)
  if p < r                            // check for base case
      q = ⌊(p + r)/2⌋                 // divide
      MERGE-SORT(A, p, q)             // conquer
      MERGE-SORT(A, q + 1, r)         // conquer
      MERGE(A, p, q, r)               // combine
```

Initial call: Merge-Sort (*A*, 1, *A.length*)

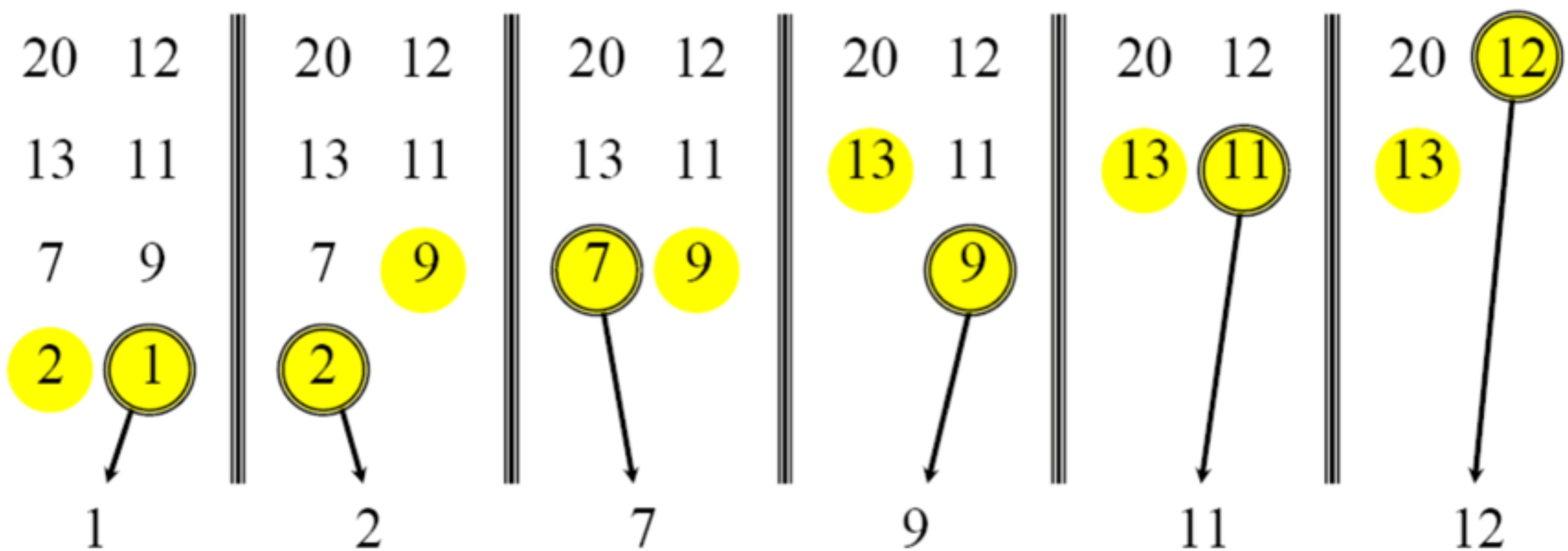
Merge Sort: Example



How is merging done?



How is merging done?



```
MERGE(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i − 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```

Correctness (Merging)

- LoopInvariant:
  - At the start of each iteration of the for k loop, the subarray **A[p..k-1] contains the k - p smallest elements** of L and R in sorted order.
  - Moreover, **L[i] and R[j] are the smallest elements** of their arrays which have not been copied back into A.

MERGE(A, p, q, r)

$n_1 = q - p + 1$
$n_2 = r - q$
let $L[1 \,..\, n_1 + 1]$ and $R[1 \,..\, n_2 +$
for $i = 1$ **to** n_1
    $L[i] = A[p + i - 1]$
for $j = 1$ **to** n_2
    $R[j] = A[q + j]$
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
$i = 1$
$j = 1$
for $k = p$ **to** r
    if $L[i] \le R[j]$
        $A[k] = L[i]$
        $i = i + 1$
    else $A[k] = R[j]$
        $j = j + 1$

Asymptotic analysis

- Merging step?

MERGE(A, p, q, r)

```
n1 = q − p + 1
n2 = r − q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

Asymptotic analysis

- Merging step?
  Computation time is $\Theta(n)$
- What is the over all computation time?

Asymptotic analysis

$T(n)$	**MERGE-SORT** $A[1\,..\,n]$
$\Theta(1)$	1. If $n = 1$, done.
$2T(n/2)$	2. Recursively sort $A[\,1\,..\lceil n/2\rceil\,]$ and $A[\,\lceil n/2\rceil+1\,..\,n\,]$.
$\Theta(n)$	**3. *"Merge"*** the 2 sorted lists

Asymptotic analysis

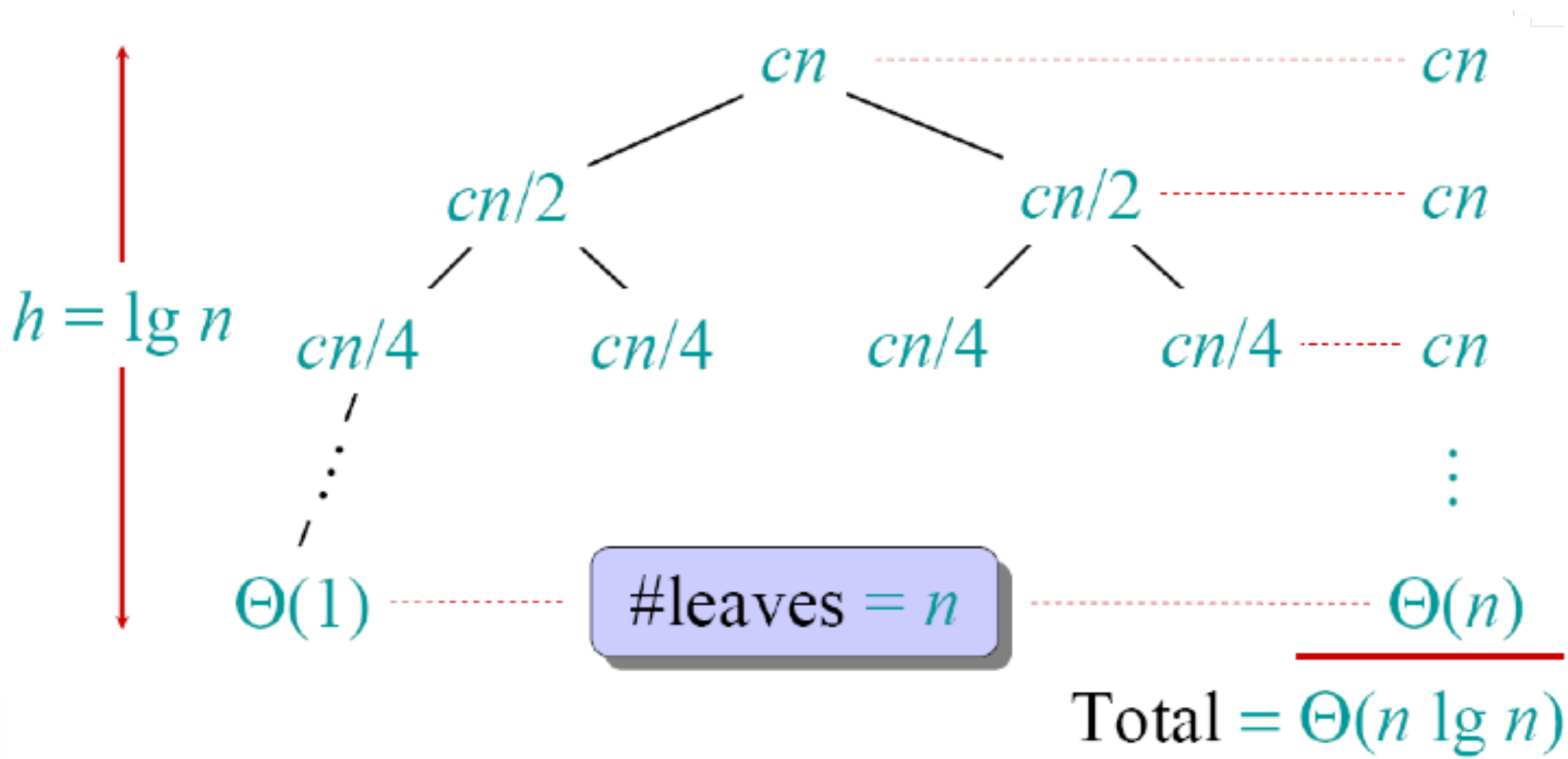
- The overall running time for Merge Sort is given by the **recurrence**

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- The base case may be omitted, if it is obvious that it can be neglected in the asymptotic analysis.

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where c > 0 is constant.



Merge Sort vs. Insertion Sort

- $\Theta(n\lg n)$ grows more slowly than $\Theta(n^2)$.
- Merge Sort asymptotically beats Insertion Sort (in the average case and in the worst case).
- In practice, Merge Sort beats Insertion Sort for $n > 30$ or so.