

Algorithms

Kiho Choi

Fall, 2022

Department of AI·Software
Gachon University

A decorative graphic in the bottom right corner consisting of a blue curved shape filled with various-sized circles in different shades of blue, creating a bubble-like or cellular pattern.

1. Basics of Algorithm Design and Analysis (part II)

Contents

- Analyzing and Designing algorithms
 - Kinds of analyses
 - Machine-independent time
 - The running time of the algorithm
 - Asymptotic performance

Kinds of analyses

- Worst-case (usually)
 - $T(n)$ = maximum time of algorithm on any input of size n .
 - Analysis for the worst-case input(s)
- Average-case (sometimes)
 - $T(n)$ = expected time of algorithm over all inputs of size n .
(Analysis for all inputs)
 - Need assumption of statistical distribution of inputs
 - More difficult to analyze
- Best-case (bogus)
 - Analysis for the best-case input(s)

Machine-independent time

- *What is insertion sort's worst-case time?*
 - It depends on the speed of our computer:
 - relative speed (on the same machine)
 - absolute speed (on different machines)
 - BIG IDEA:
 - Ignore machine-dependent constants.
 - Look at **growth** of $T(n)$ as n .
- "Asymptotic Analysis"**

Insertion sort

INSERTION-SORT (A, n)

for $j \leftarrow 2$ **to** n

do $key \leftarrow A[j]$ $\triangleleft \triangleleft \triangleleft$ $n-1$ times

$i \leftarrow j - 1$ $\triangleleft \triangleleft \triangleleft$ $n-1$ times

while $i > 0$ and $A[i] > key$

do $A[i+1] \leftarrow A[i]$

$i \leftarrow i - 1$

$A[i+1] = key$

Θ -notation

Math:

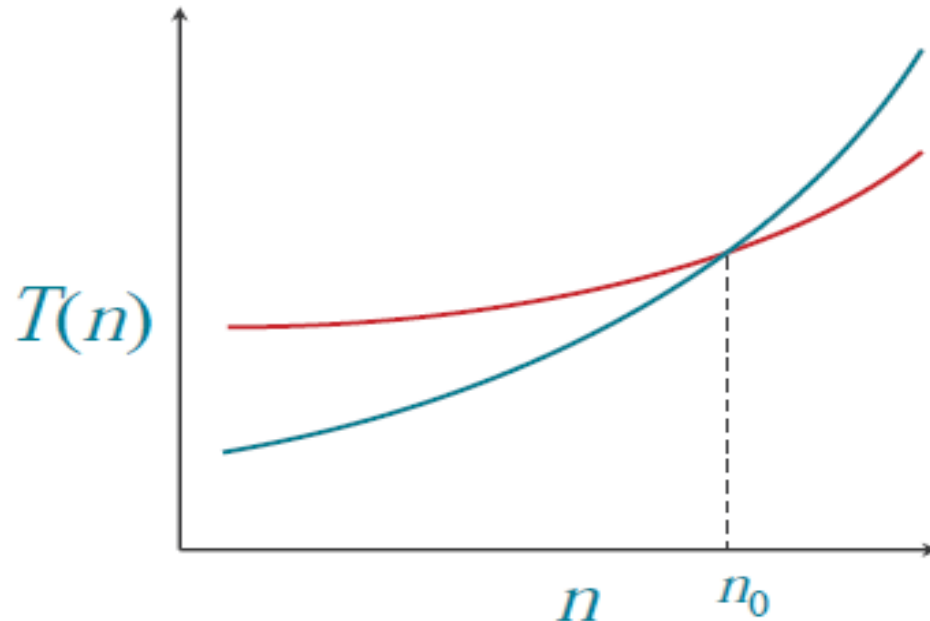
$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$

Engineering:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Asymptotic performance

When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.

The running time of the algorithm

$$\sum (cost\ of\ statement) \cdot (number\ of\ times\ statement\ is\ executed).$$

INSERTION-SORT (A, n)

for $j \leftarrow 2$ to n

do $key \leftarrow A[j]$ $\triangleleft \triangleleft \triangleleft$ $n-1$ times *of Insertion sort.*

$i \leftarrow j - 1$ $\triangleleft \triangleleft \triangleleft$ $n-1$ times $- c_3(n-1) + c_4 \sum_{j=2}^n t_j + c_5 \sum_{j=2}^n (t_j - 1)$

while $i > 0$ and $A[i] > key$

do $A[i+1] \leftarrow A[i]$

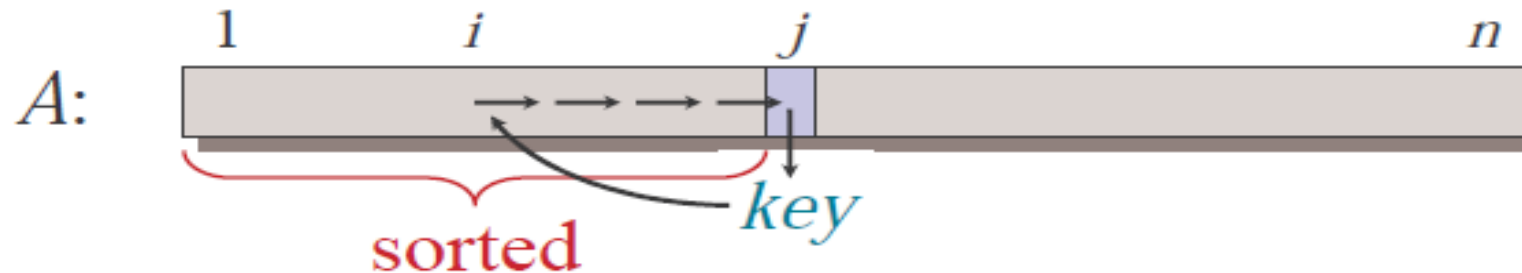
$i \leftarrow i - 1$ $+ c_7(n-1).$

$A[i+1] = key$

Insertion sort

“pseudocode”

```
INSERTION-SORT ( $A, n$ )     $\triangleleft A[1 \dots n]$   
  for  $j \leftarrow 2$  to  $n$   
    do  $key \leftarrow A[j]$   
       $i \leftarrow j - 1$   
      while  $i > 0$  and  $A[i] > key$   
        do  $A[i+1] \leftarrow A[i]$   
           $i \leftarrow i - 1$   
       $A[i+1] = key$ 
```



Insertion sort analysis

Worst case: Input reverse sorted.

$$T(n) = \sum_{j=2}^n \Theta(j) = \Theta(n^2) \quad [\text{arithmetic series}]$$

Average case: All permutations equally likely.

$$T(n) = \sum_{j=2}^n \Theta(j/2) = \Theta(n^2)$$

Is insertion sort a fast sorting algorithm?

- Moderately so, for small n .
- Not at all, for large n .

Designing algorithms

- There are many ways to design algorithms
For example, insertion sort is incremental:
having sorted $A[1 \dots j-1]$, place $A[j]$ correctly,
So that $A[1 \dots J]$ is sorted.

Another common approach

- Divide and Conquer
 - Divide: the problem into a number of subproblems that are smaller instances of the same problem.
 - Conquer: the subproblems by solving them recursively.
 - Combine: the subproblem solutions to give a solution to the original problem.

Merge sort

- MERGE-SORT $A[1 \dots n]$
 1. If $n = 1$, done.
 2. Recursively sort $A[1 \dots n/2]$ and $A[n/2 + 1 \dots n]$.
 3. “**Merge**” the 2 sorted lists.

Key subroutine: MERGE

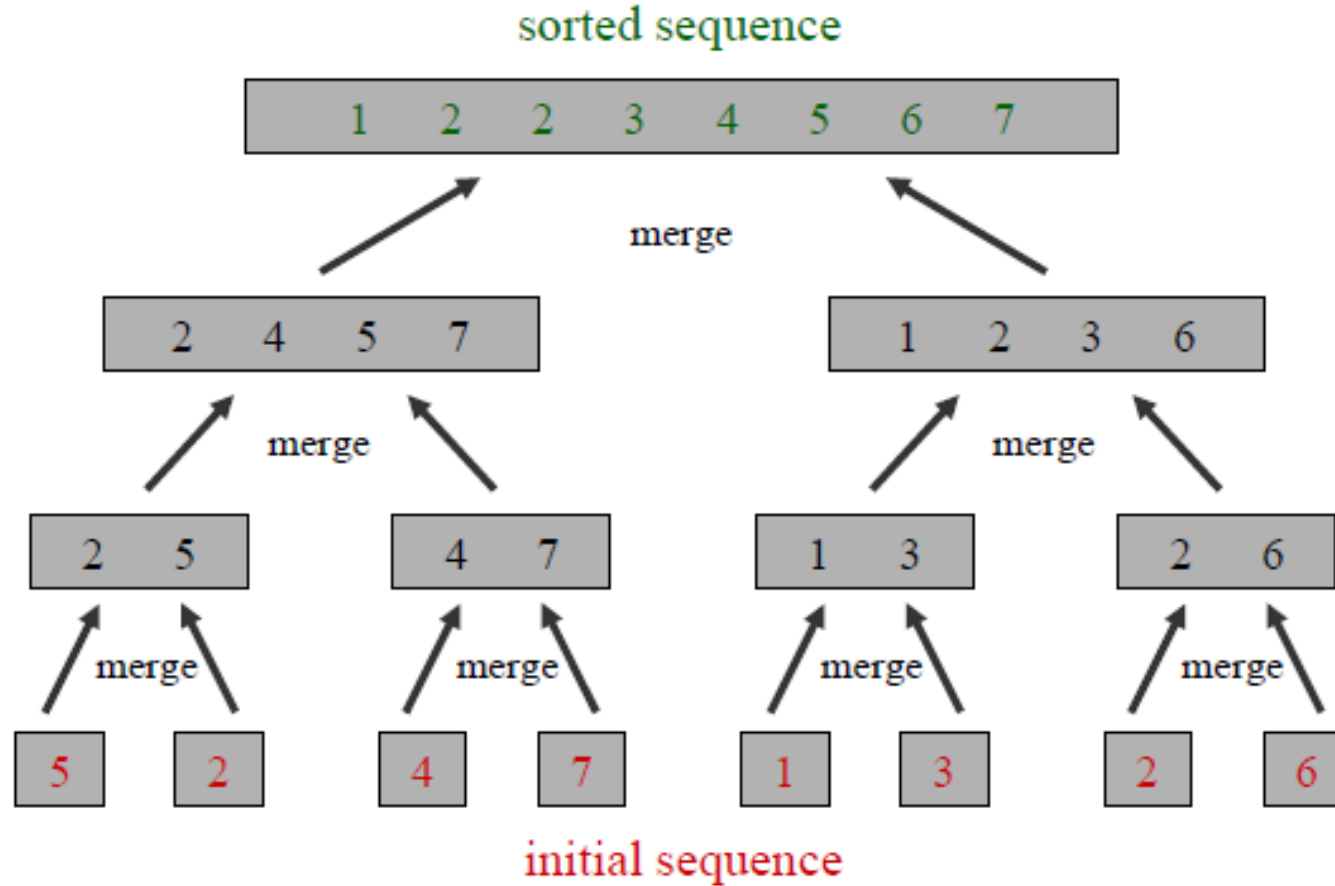
Merge sort

MERGE-SORT $A[1 \dots n]$

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

Key subroutine: **MERGE**

Operation of merge sort



<https://www.youtube.com/watch?v=JSceec-wEyw>

Merging two sorted arrays

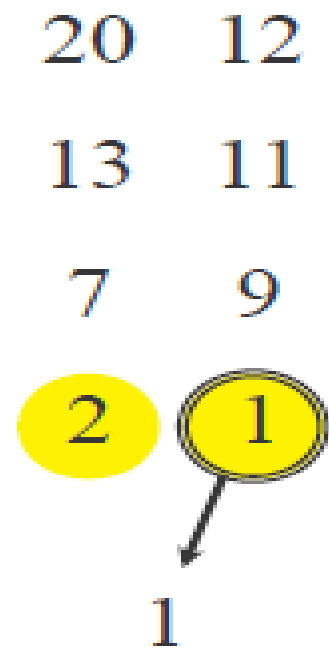
20 12

13 11

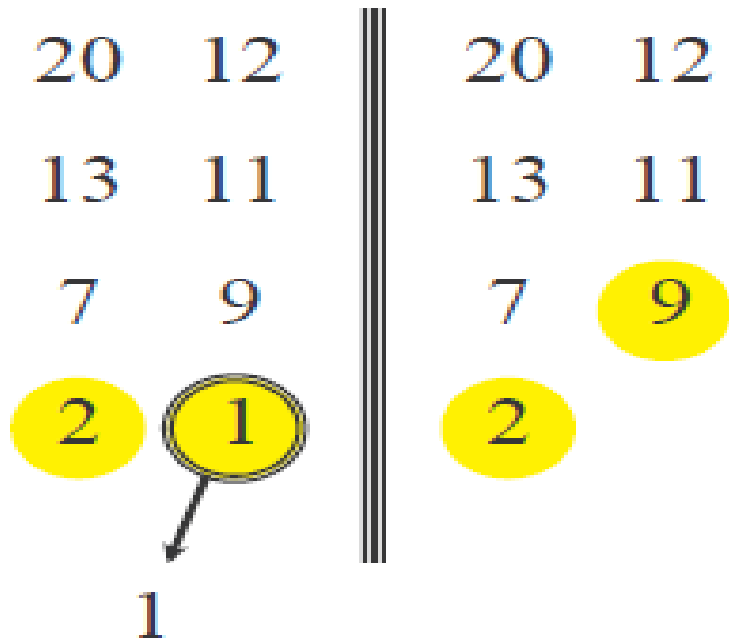
7 9

2 1

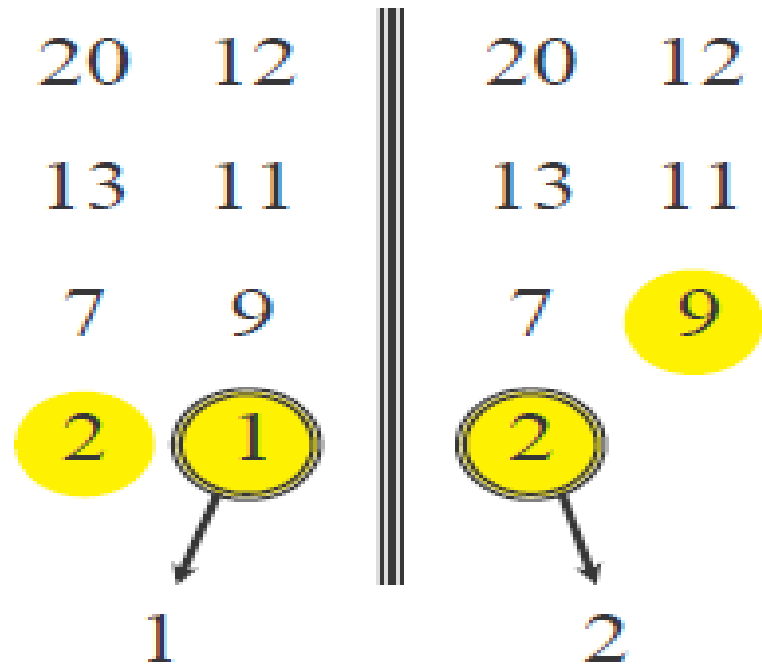
Merging two sorted arrays



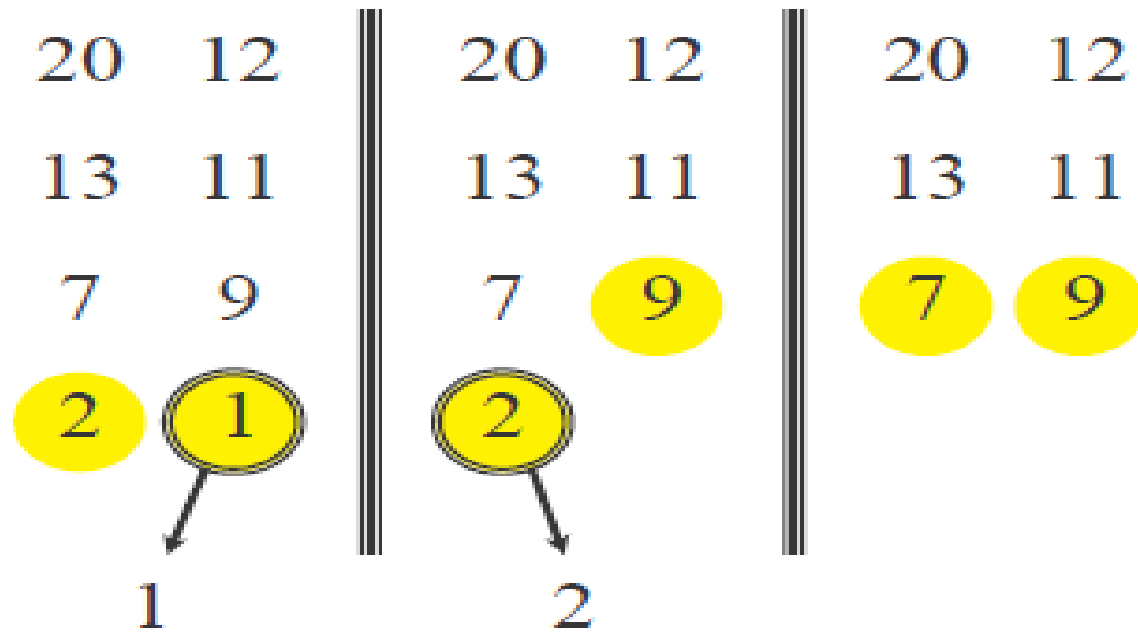
Merging two sorted arrays



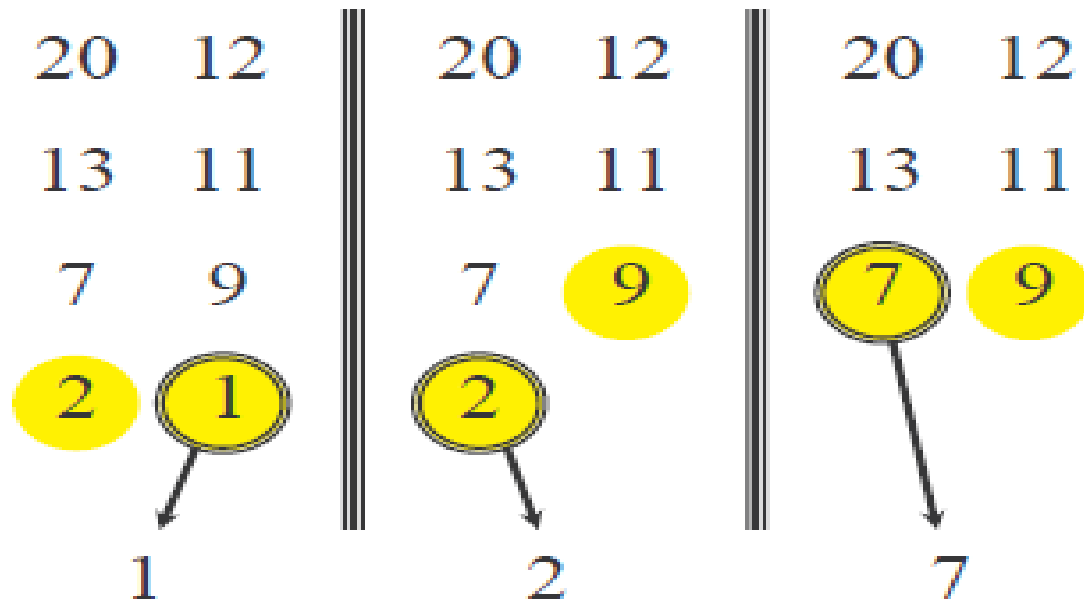
Merging two sorted arrays



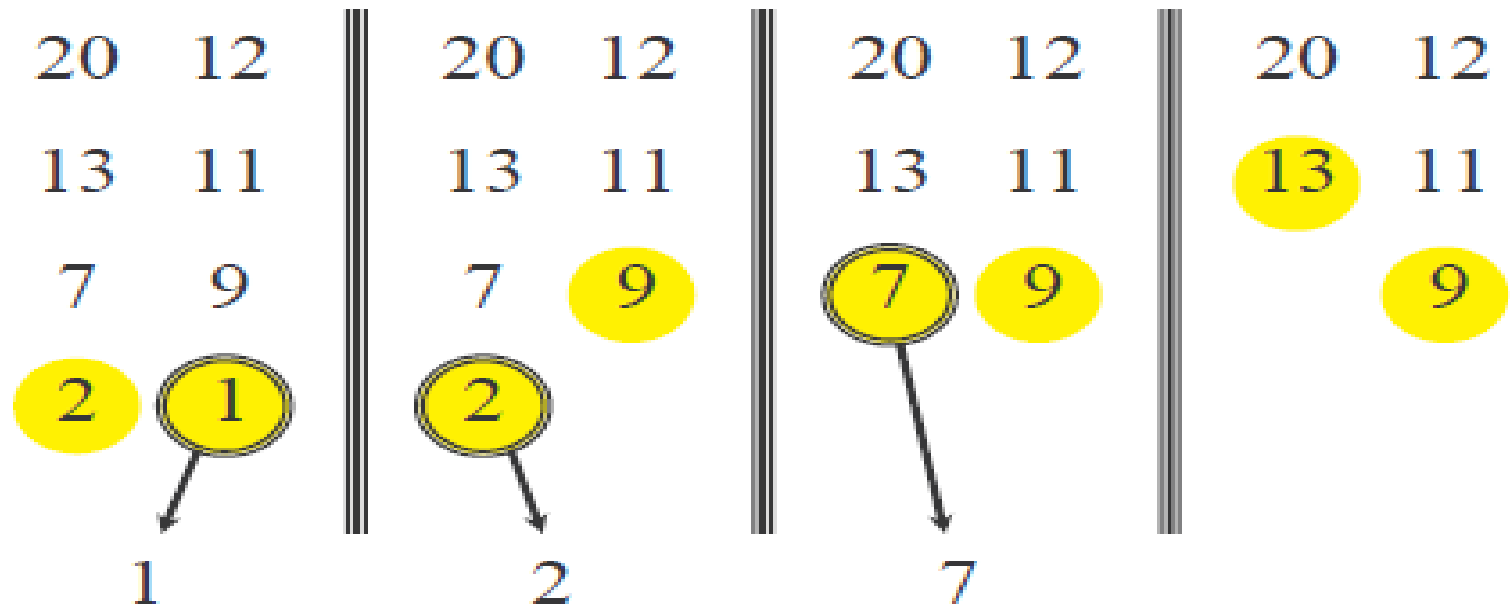
Merging two sorted arrays



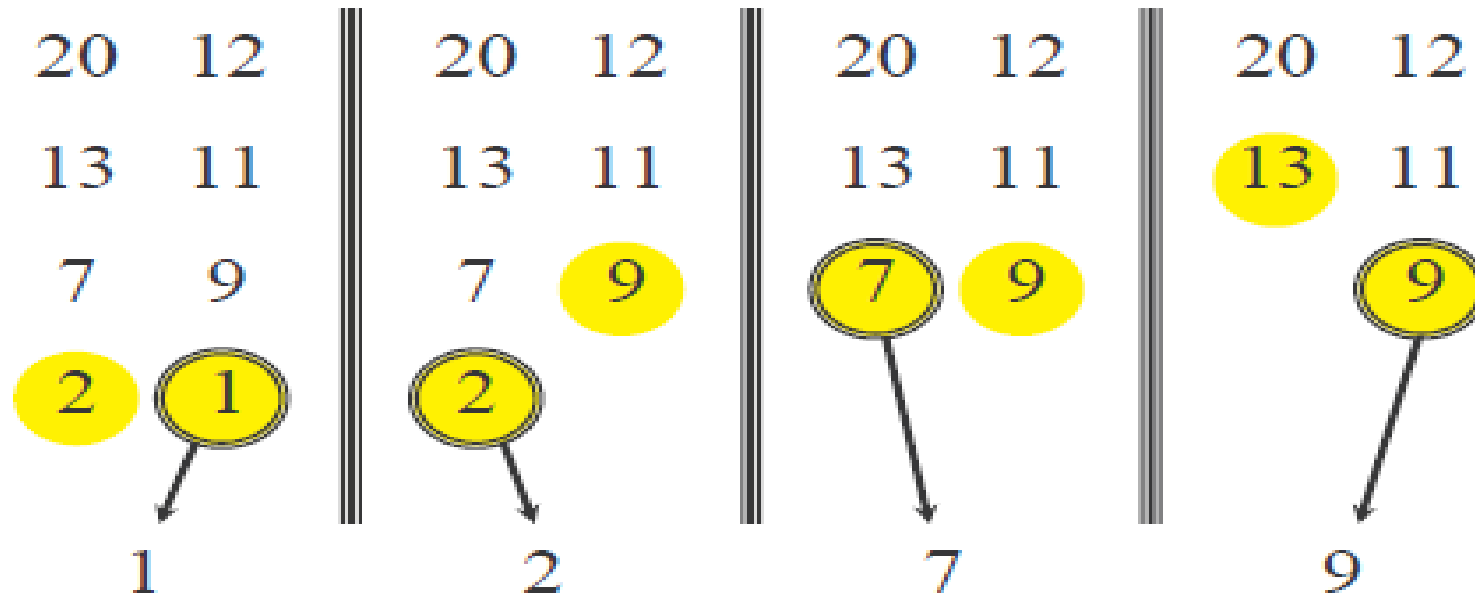
Merging two sorted arrays



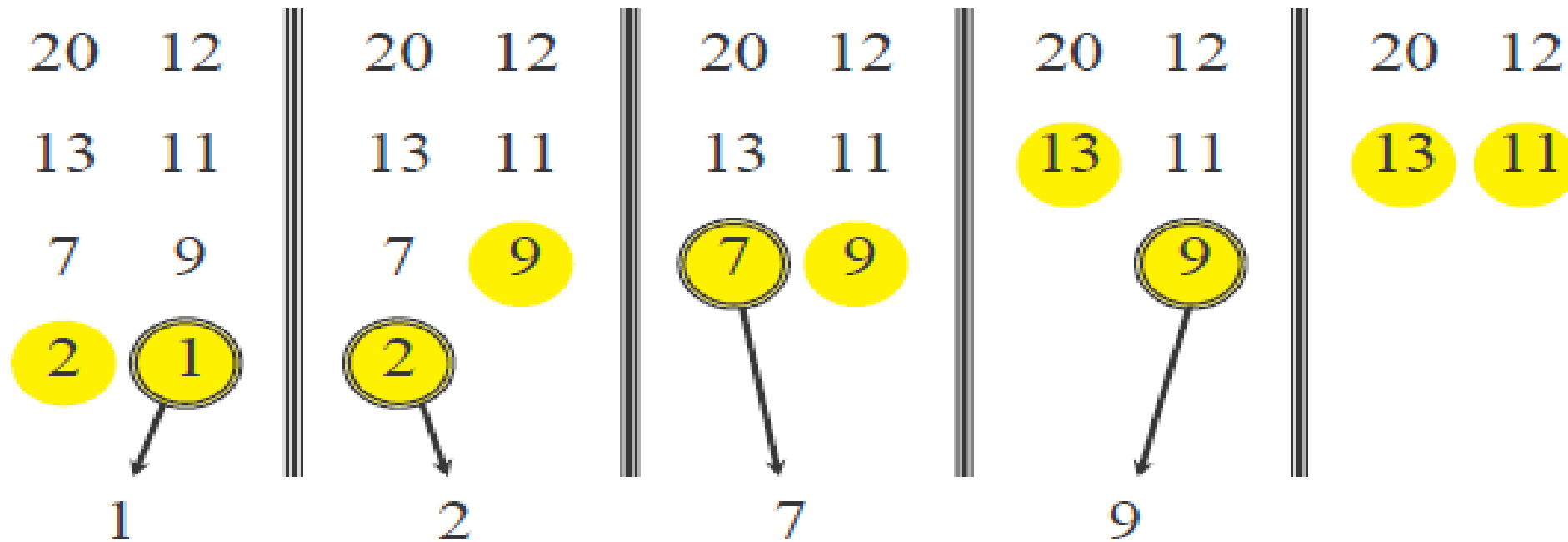
Merging two sorted arrays



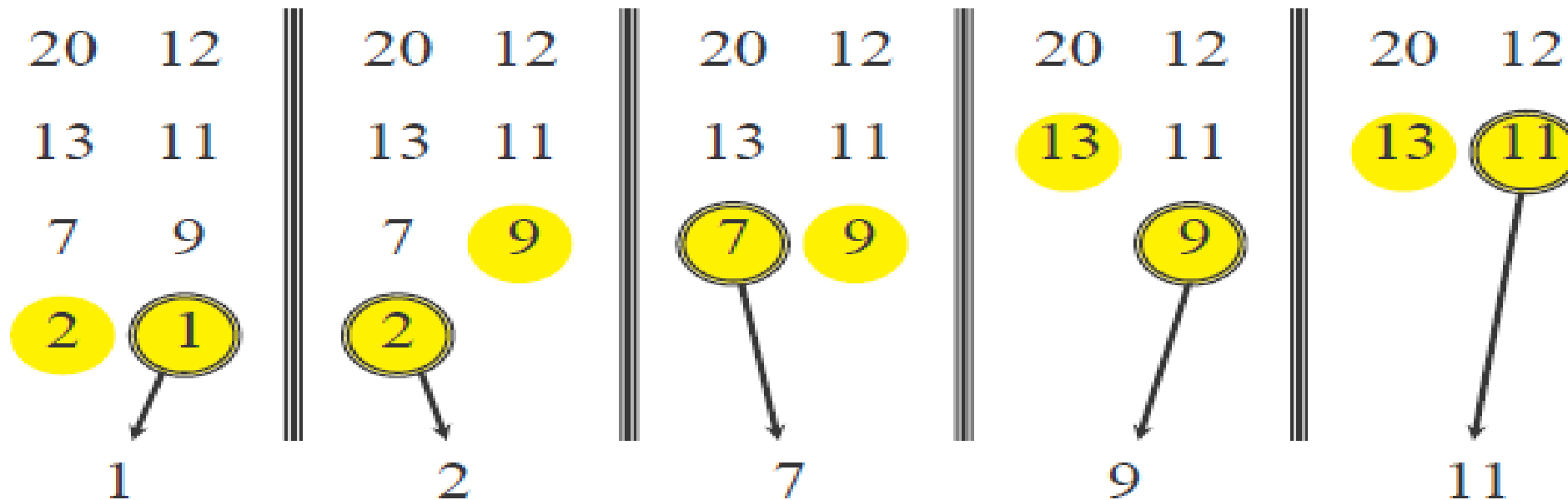
Merging two sorted arrays



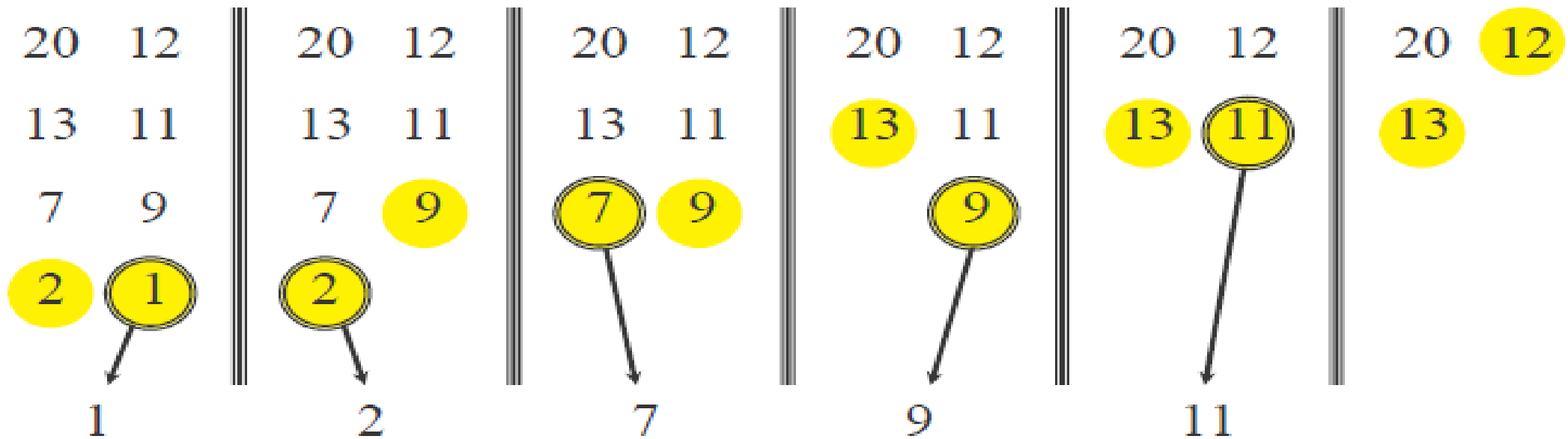
Merging two sorted arrays



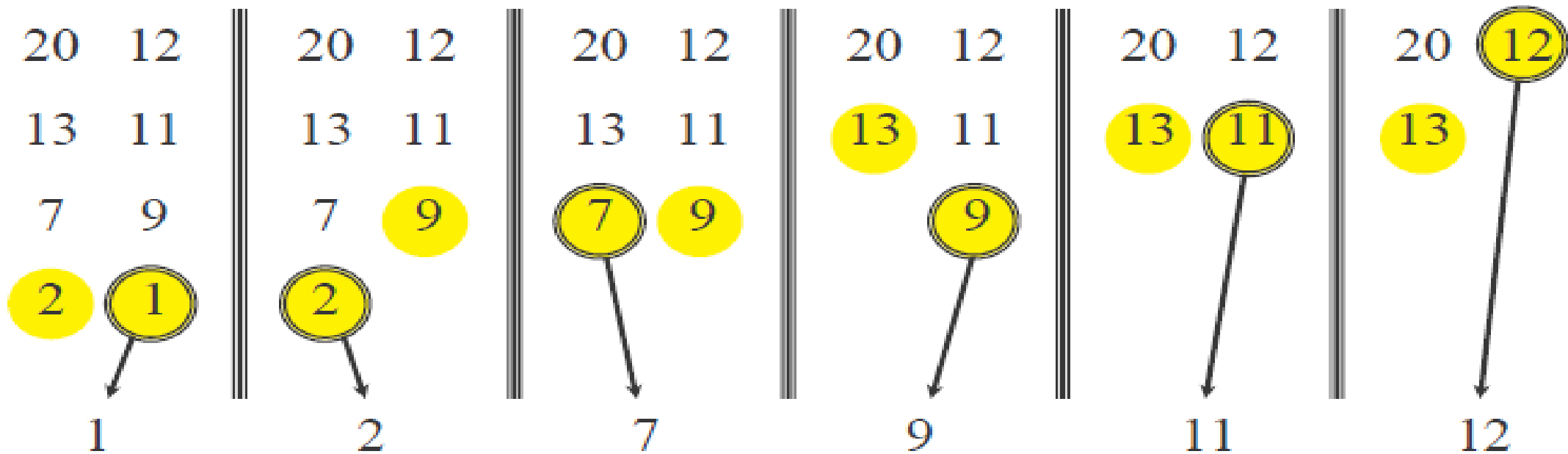
Merging two sorted arrays



Merging two sorted arrays

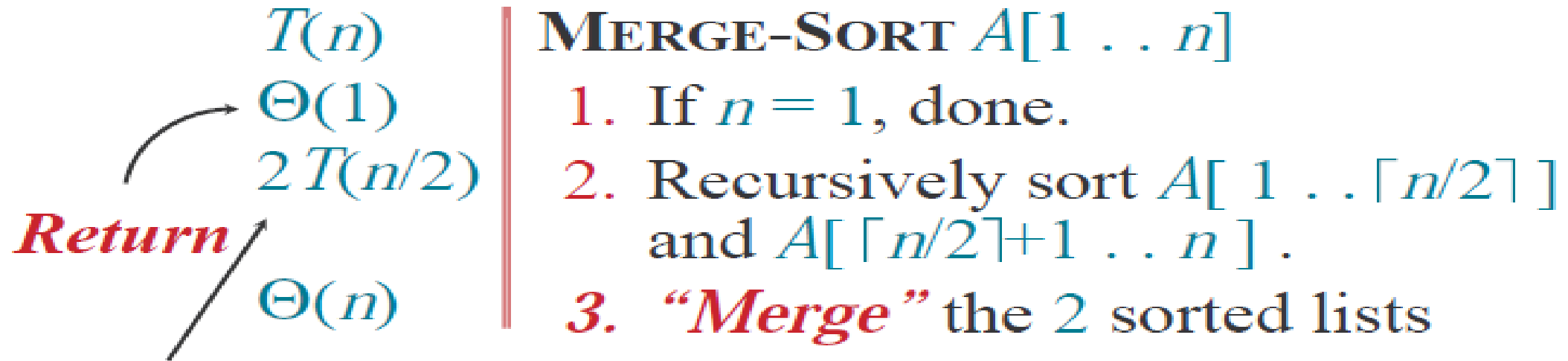


Merging two sorted arrays



Time = $\Theta(n)$ to merge a total of n elements (linear time).

Analyzing merge sort



Sloppiness: Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.

Recurrence for merge sort

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

We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n , but only when it has no effect on the asymptotic solution to the recurrence.

Recursion tree

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

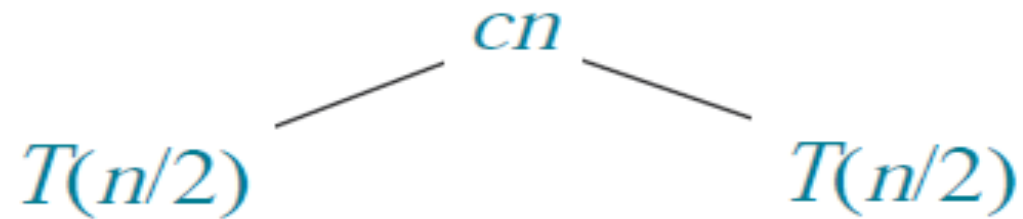
Recursion tree

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

$$T(n)$$

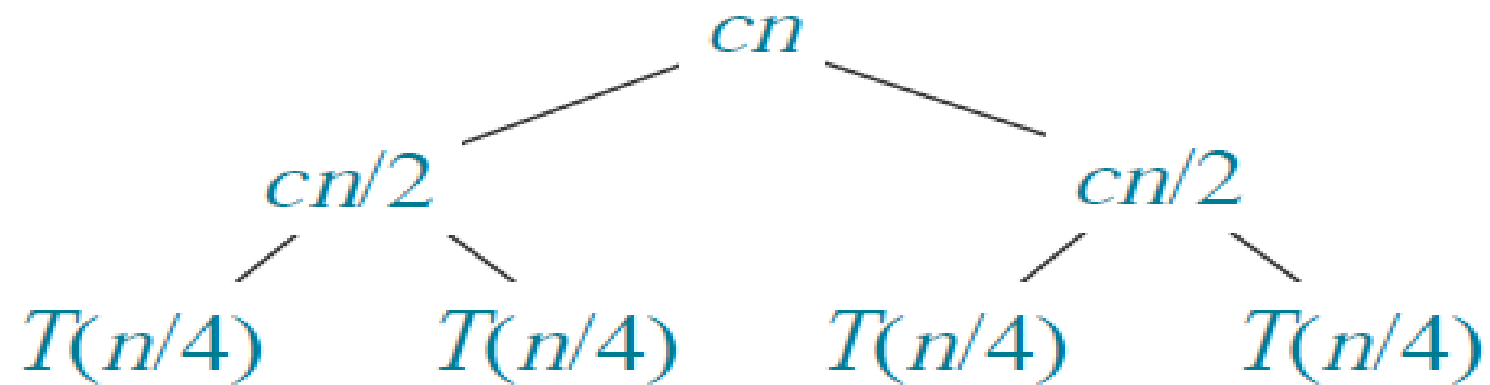
Recursion tree

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



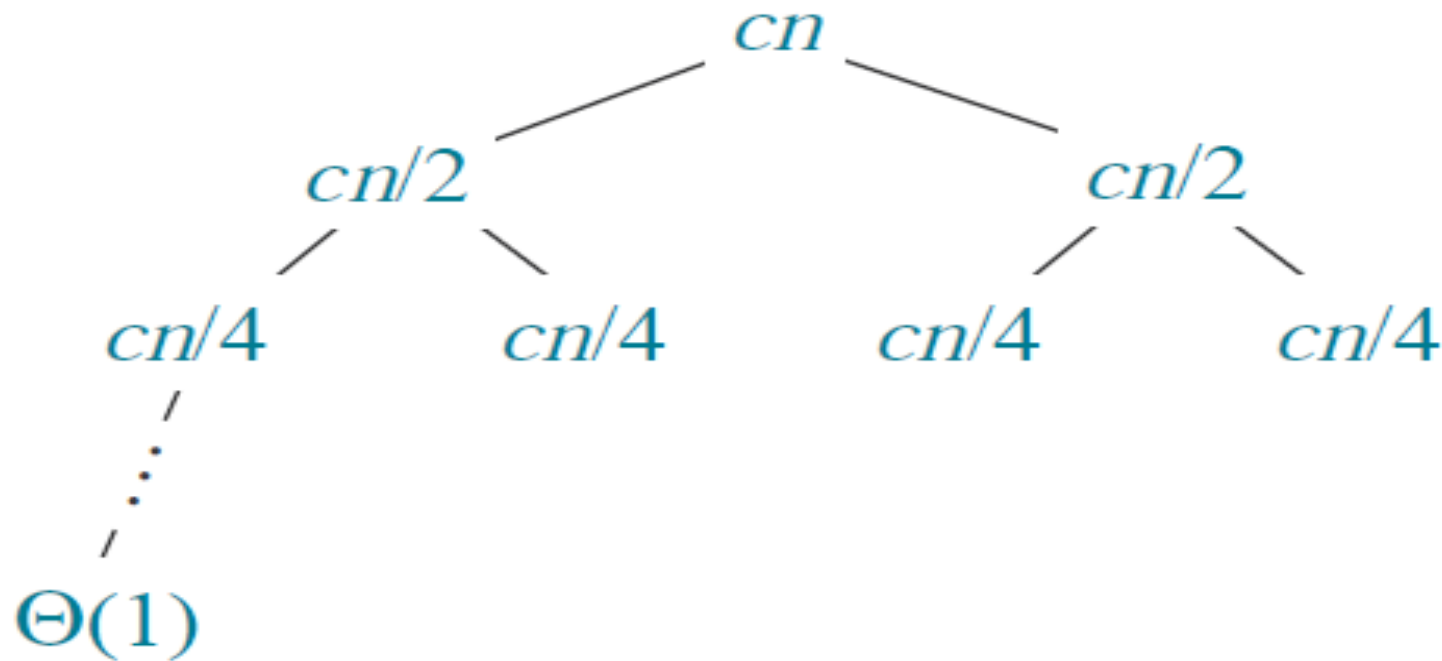
Recursion tree

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



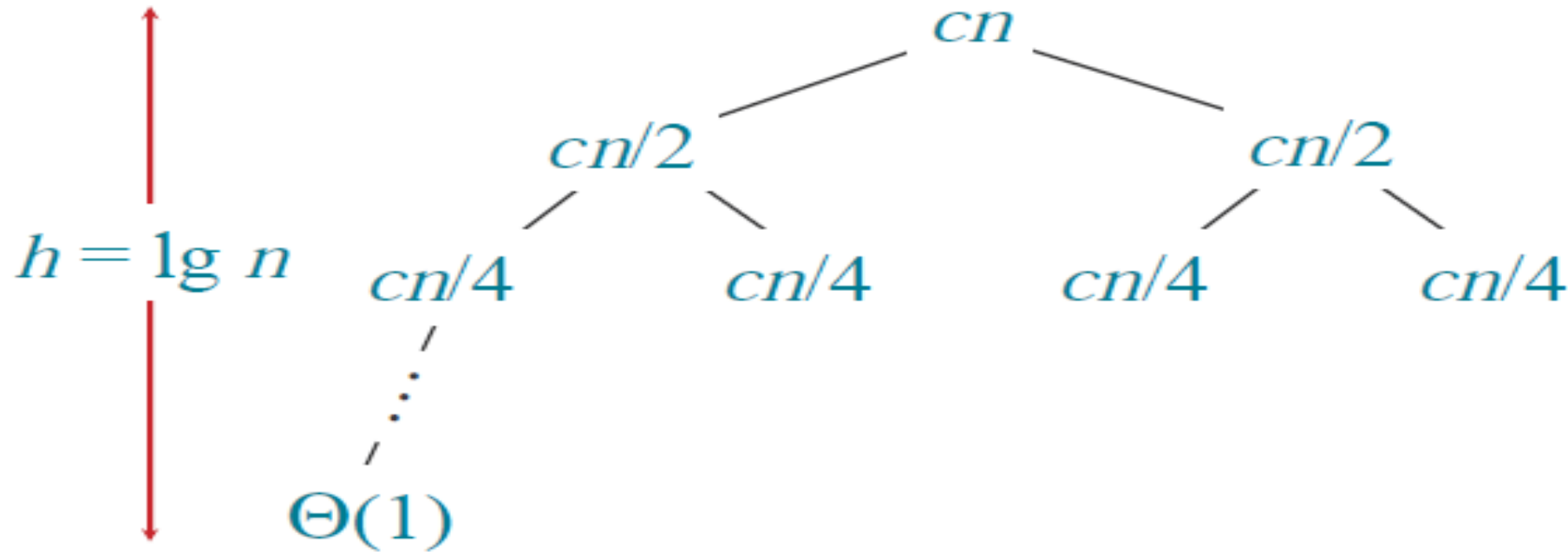
Recursion tree

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



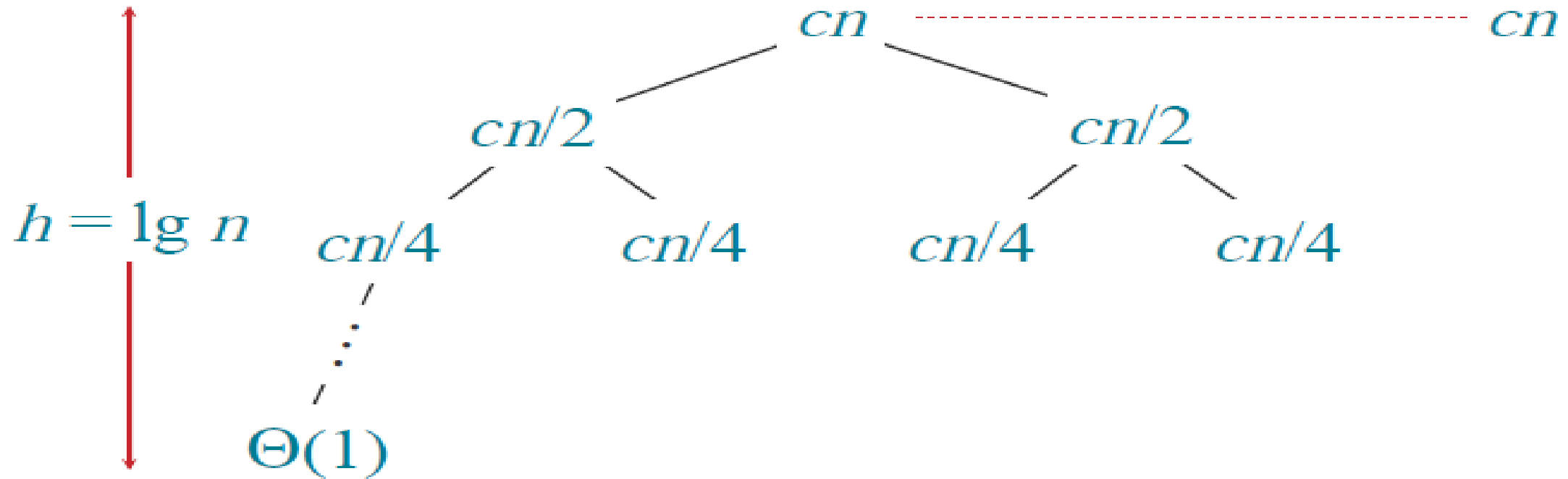
Recursion tree

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



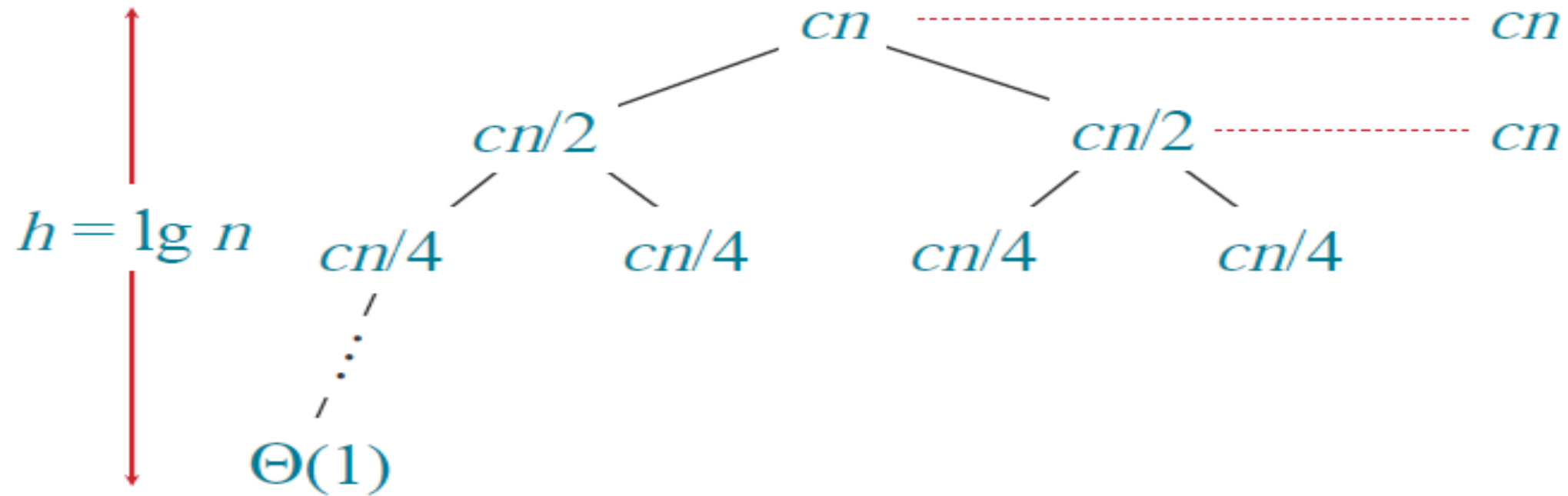
Recursion tree

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



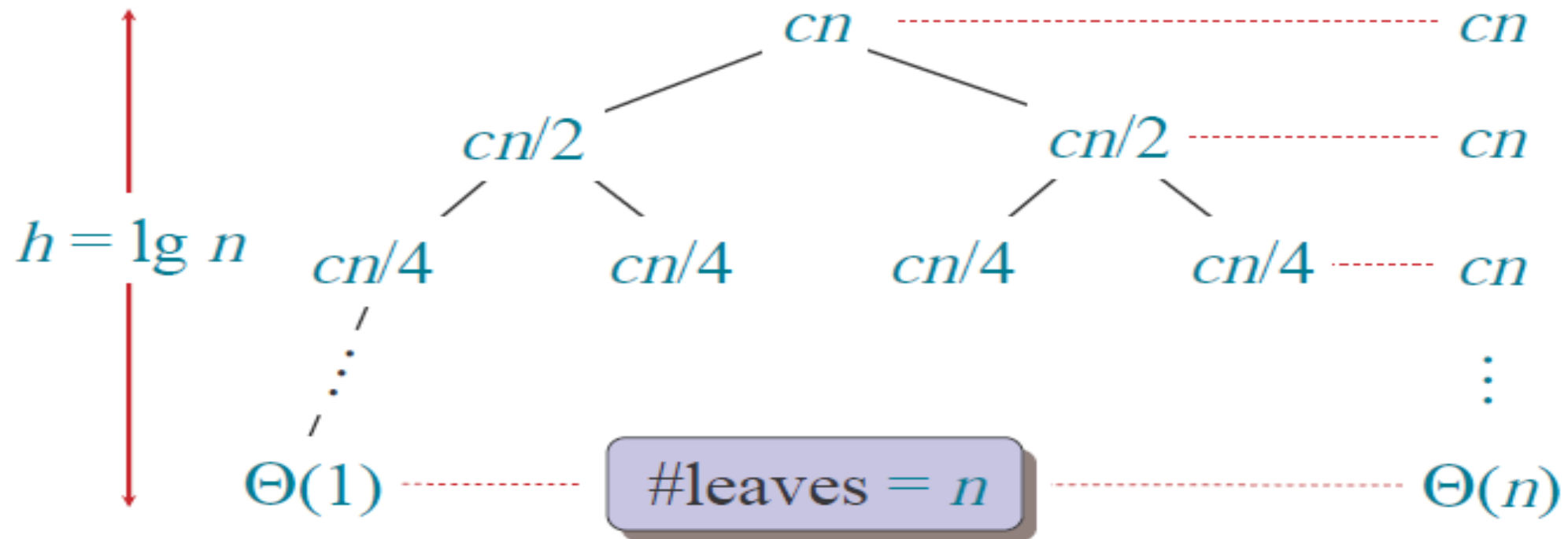
Recursion tree

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



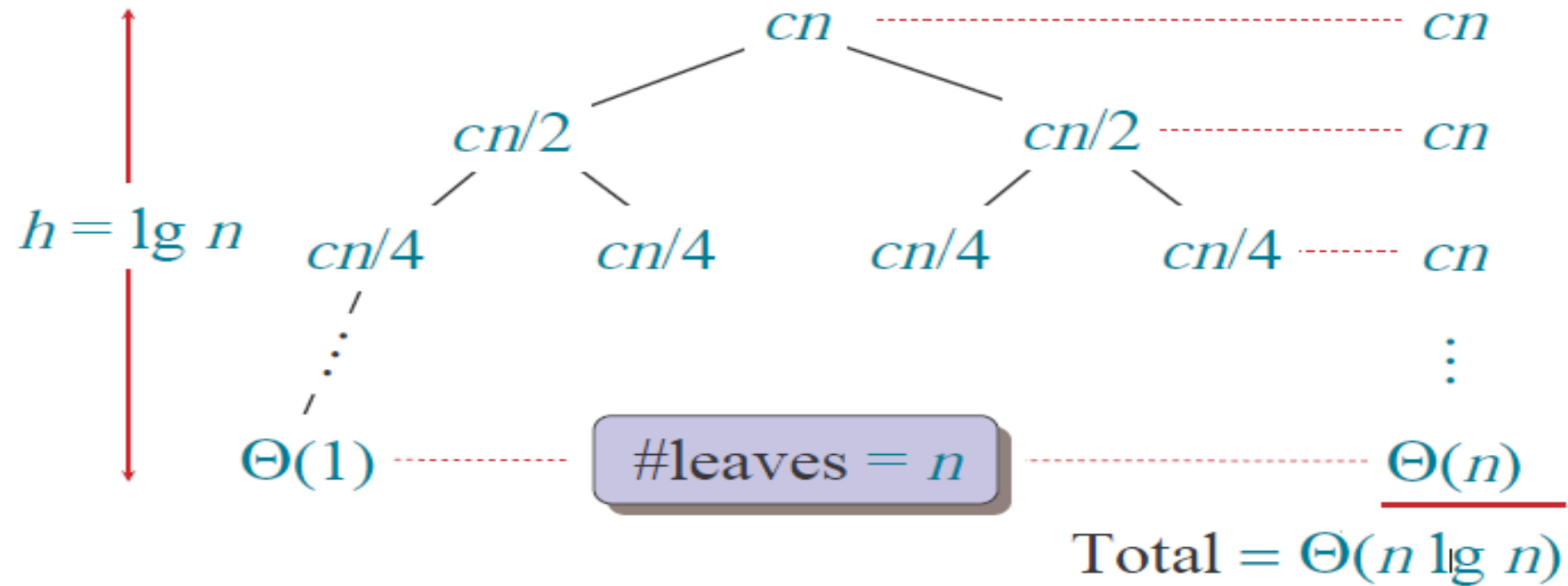
Recursion tree

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



Recursion tree

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



Conclusions

$\Theta(n \lg n)$ grows more slowly than $\Theta(n^2)$.

Therefore, merge sort asymptotically beats insertion sort in the worst case.

In practice, merge sort beats insertion sort for $n > 30$ or so.

Implementation

Implementation of merge sort

MERGE-SORT $A[1 \dots n]$

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

Key subroutine: **MERGE**

Implementation of merge sort

MERGE-SORT $A[1 \dots n]$

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

Key subroutine: MERGE

```
[1] # Definition of merge sort
def mergeSort(A, p:int, r:int):
    if p < r:
        q = (p+r) // 2
        mergeSort(A, p, q)
        mergeSort(A, q+1, r)
        merge(A, p, q, r)

def merge(A, p:int, q:int, r:int):
    i = p; j = q+1; t = 0
    tmp = [0 for i in range(len(A))]
    while i <= q and j <= r:
        if A[i] <= A[j]:
            tmp[t] = A[i]; t += 1; i += 1
        else:
            tmp[t] = A[j]; t += 1; j += 1
    while i <= q:
        tmp[t] = A[i]; t += 1; i += 1
    while j <= r:
        tmp[t] = A[j]; t += 1; j += 1
    i = p; t = 0
    while i <= r:
        A[i] = tmp[t]; t += 1; i += 1
```

```
[2] # List
input_list1 = [8, 2, 4, 9, 3, 6]
print(input_list1)

[8, 2, 4, 9, 3, 6]
```

```
[3] # Sorting
mergeSort(input_list1, 0, len(input_list1)-1)
print(input_list1)

[2, 3, 4, 6, 8, 9]
```

```
[4] # random list
import random
input_list2 = random.sample(range(100),10)
print(input_list2)

[42, 72, 31, 30, 96, 35, 49, 3, 20, 12]
```

```
[5] # Sorting
mergeSort(input_list2, 0, len(input_list2) -1 )
print(input_list2)

[3, 12, 20, 30, 31, 35, 42, 49, 72, 96]
```

Example code test

- Code test: <https://www.acmicpc.net/problem/2751>
- Solving the problem using merge sort
- Example result of submission

2751	aikiho	모든 언어 ▼	모든 결과 ▼	검색
------	--------	---------	---------	----

제출 번호	아이디	문제	결과	메모리	시간	언어	코드 길이	제출한 시간
48335930	aikiho	2751	맞았습니다!!	139560 KB	1444 ms	PyPy3 / 수정	806 B	2분 전

THANK YOU

