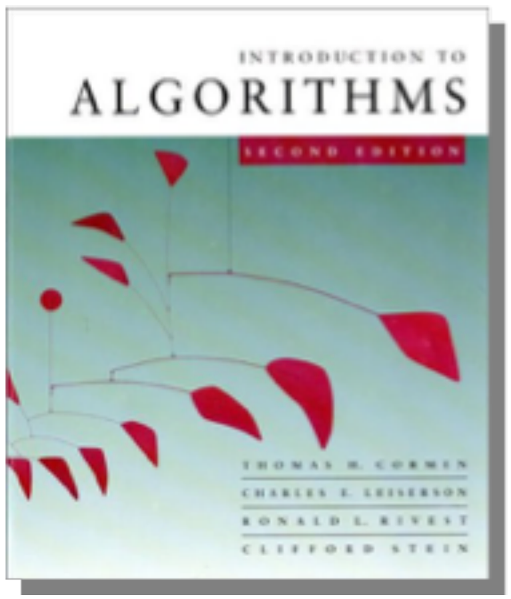
Introduction to Algorithms



- Insertion sort
- Asymptotic analysis
- Merge sort
- Recurrences

Intuitive approach for the first lectures

The problem of sorting

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Sort: 8 2 4 9 3 6

Intuition: move smaller numbers to the left …

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9 *done*

notation: $x \leftarrow y$ means assign the value y to the variable x (in most programming languages you would write x= y).

Insertion sort "pseudocode"

INSERTION-SORT (A, n) ▷ $A[1 \mathinner{.\,.} n]$

start from 2 because 1 element alone would be sorted ;-)

for $j \leftarrow 2$ to n — we will sort the first j numbers, having already sorted the first j-1

do $key \leftarrow A[j]$ — read the j-th number and put it in the auxiliary variable "key"

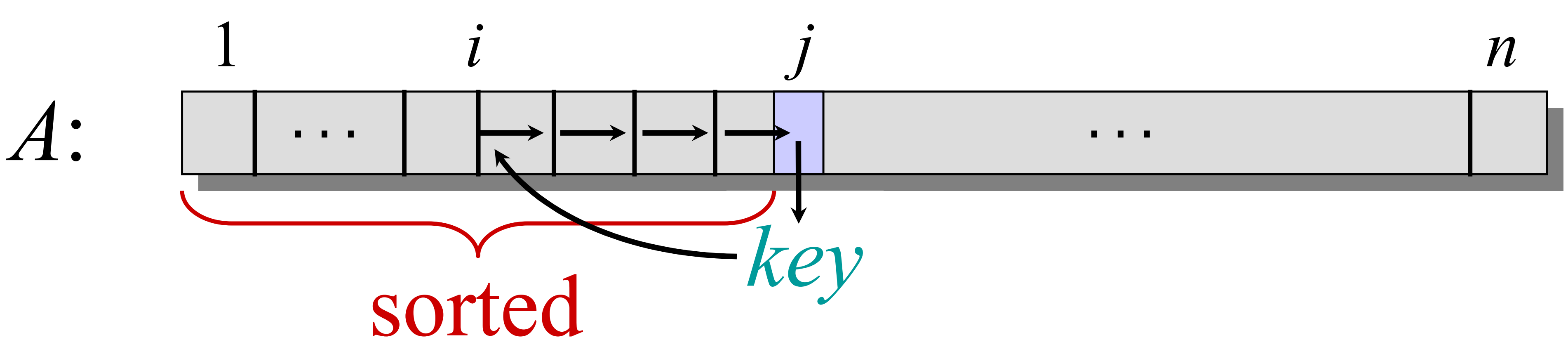
$i \leftarrow j-1$ — starting position for scanning the list backwards

while $i > 0$ and $A[i] > key$ — continue moving backwards until you find the right place for the insertion or the list ends

do $A[i+1] \leftarrow A[i]$ — while scanning backwards shift the elements to the right (create space the list)

$i \leftarrow i-1$ — move backwards

$A[i+1] = key$ — put the selected element (key <- A[j]) in the correct position

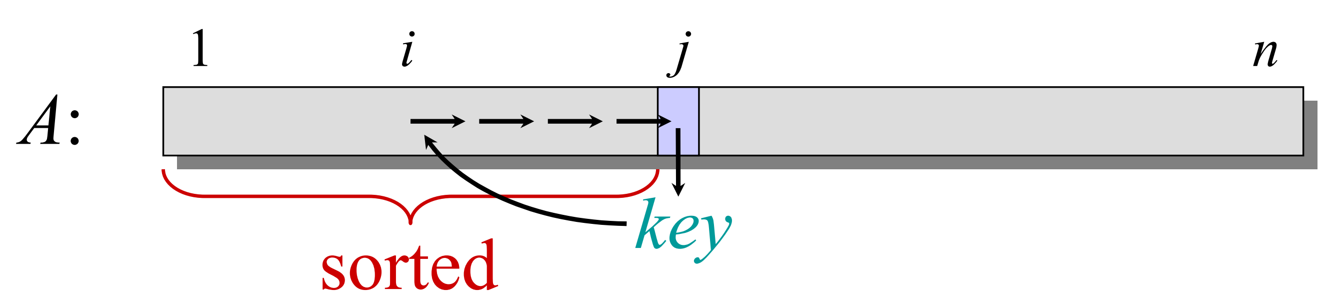


compare

INSERTION-SORT (A, n) ▷ A[1 . . n]

```
for j ← 2 to n
    do key←A[j]
        i←j–1
        while i > 0 and A[i] > key
            do A[i+1] ← A[i]
                i←i–1
        A[i+1] = key
```

8	2	4	9	3	6	
2	8	4	9	3	6	
2	4	8	9	3	6	
2	4	8	9	3	6	
2	3	4	8	9	6	
2	3	4	6	8	9	*done*



Running time

- The running time depends on the input: an already sorted sequence is easier to sort.

- Parametrize the running time by the size of the input, since short sequences are easier to sort than long ones.

- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

Kinds of analysis

Worst-case: (usually)

- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm over all inputs of size n.
- Needs assumption of statistical distribution of inputs.

Best-case: (bogus)

- Cheat with a slow algorithm that works fast on some input.

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 (consider a conventional unit cost for any single elementary operation)
- Look at growth of $T(n)$ as $n \rightarrow \infty$.

"Asymptotic Analysis"

Θ-notation

Math:

$f(n) = \Theta(g(n))$: there exist positive constants c_1, c_2, and n

such that

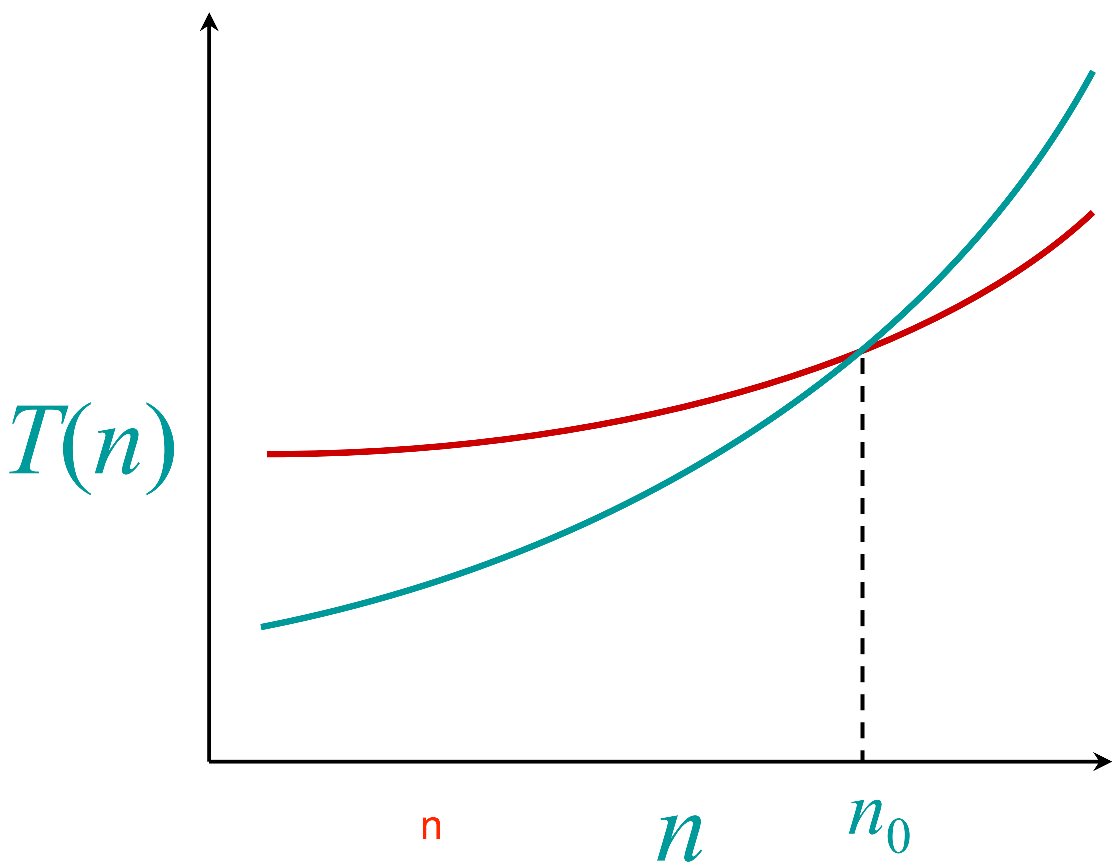
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \qquad \text{for all } n \geq n_0$$

Practictioners:

- 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.

Insertion sort analysis

Worst case: Input reverse sorted.

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

[arithmetic series (Gauss 6 years old)]

Average case: All permutations equally likely.

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

Is insertion sort a fast sorting algorithm?

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

Merge sort (recursive)

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" MERGE-SORT the 2 sorted lists.

Key subroutine: MERGE-SORT

this is an example of the approach

Divide-and-Conquer

Mergesort:

1. Divide: $n=1$, divide the given n-element array A into 2 subarrays of n/2 elements each

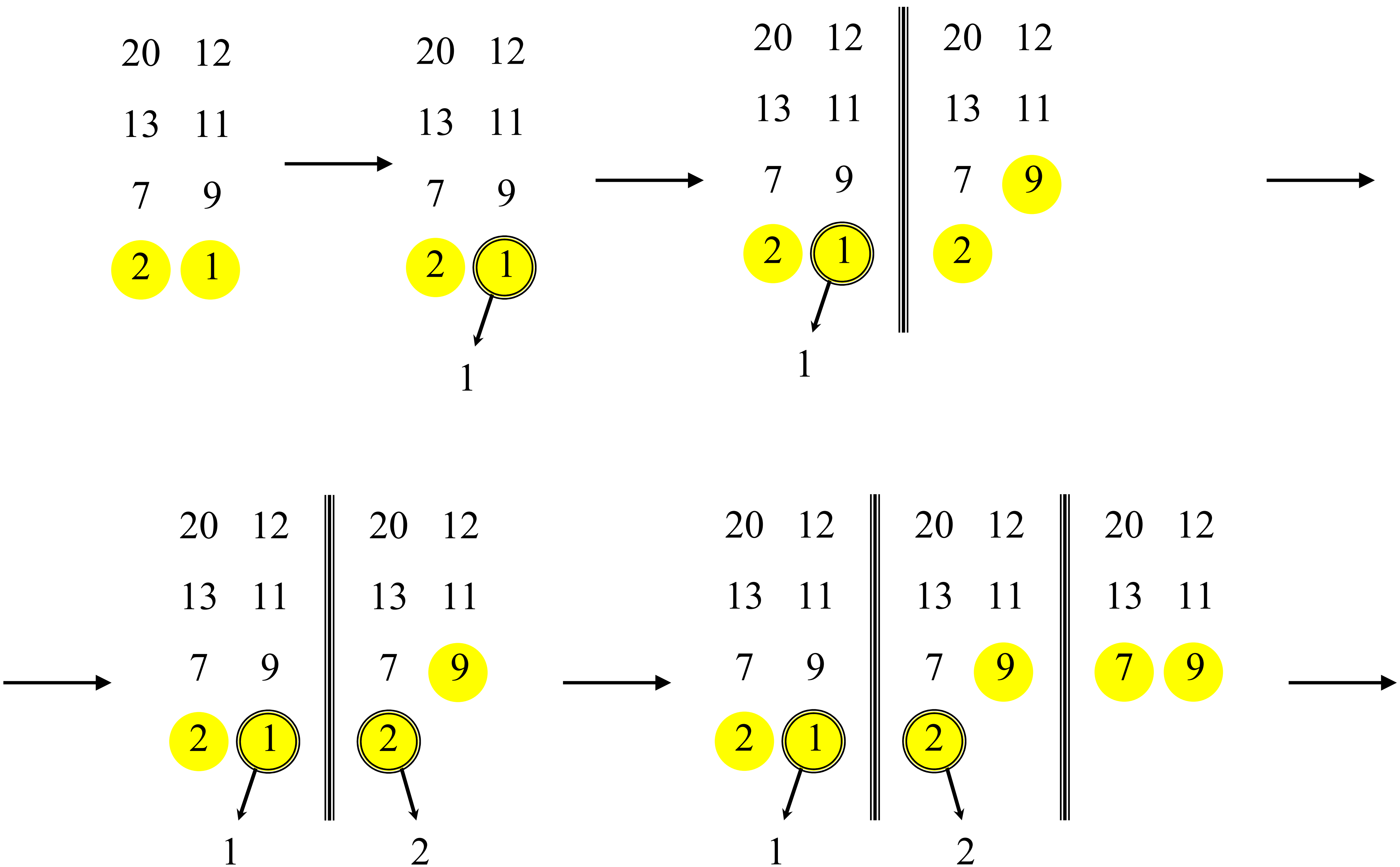
2. Conquer: recursively sort the two subarrays

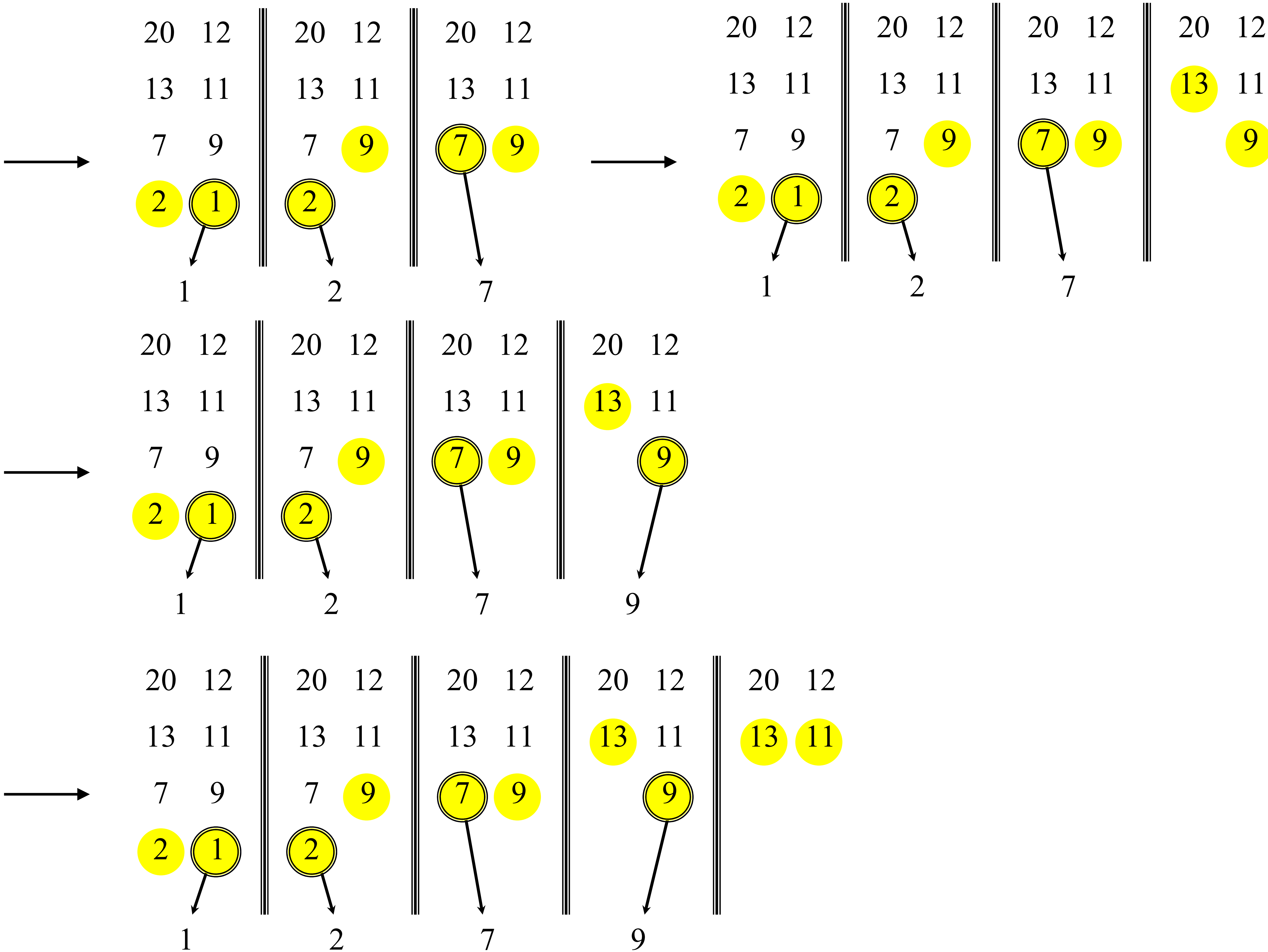
3. Combine: merge 2 sorted subarrays into 1 sorted array.

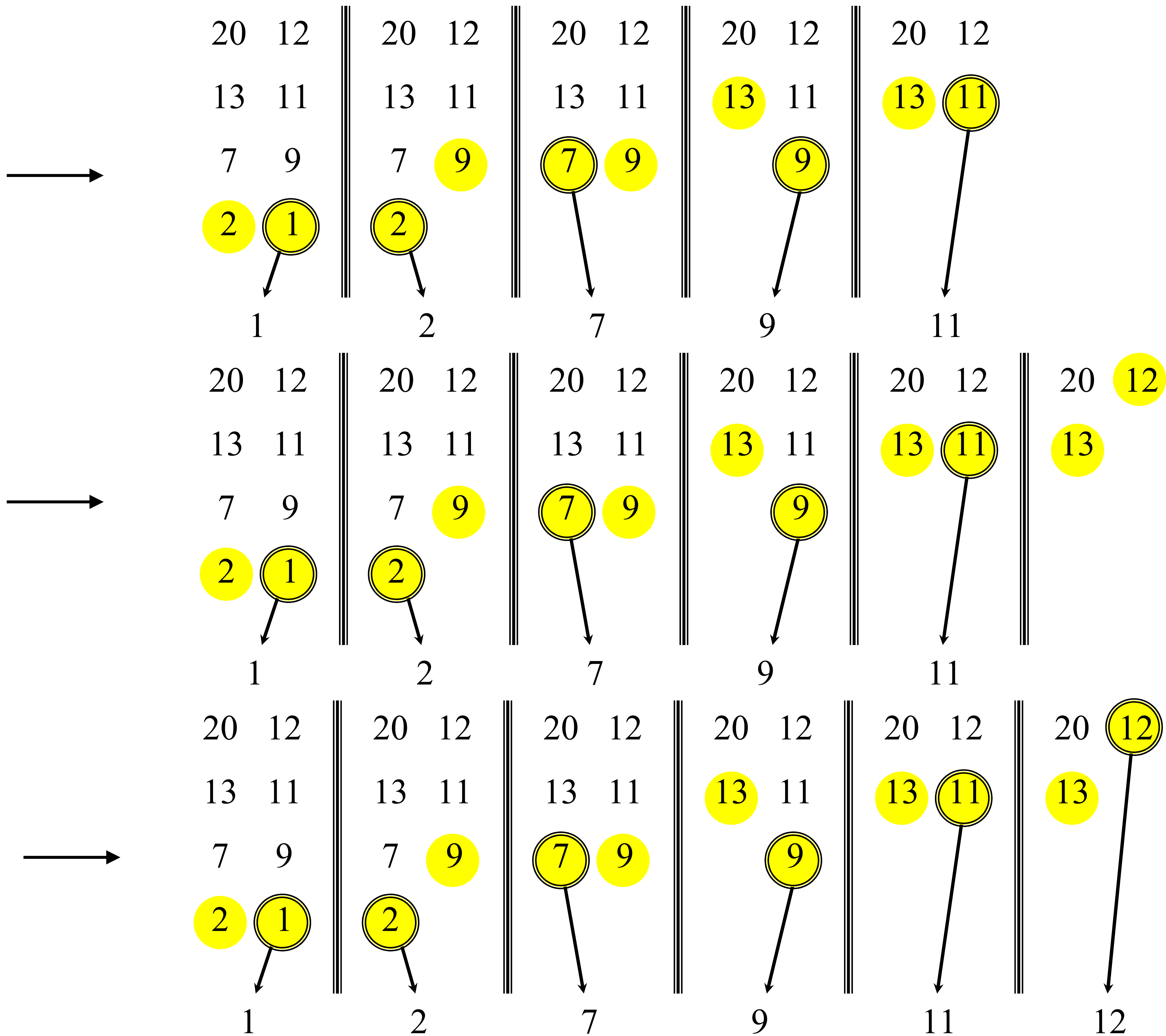
Beautiful "ultrashort" recursive pseudocode

```
MERGE-SORT (A[1 . . n])
{ if n>1
      MERGE-SORT (A[ 1 . .  ⌈n/2⌉  ])
      MERGE-SORT (A[  ⌈n/2⌉ +1 . . n ])
      MERGE(A[1 . . n])
}
```

Merging two sorted arrays






20 12
13 11
7 9
2 1
1
2
7
9
11
12

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

Analyzing merge sort

$T(n)$	**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

Abuse

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

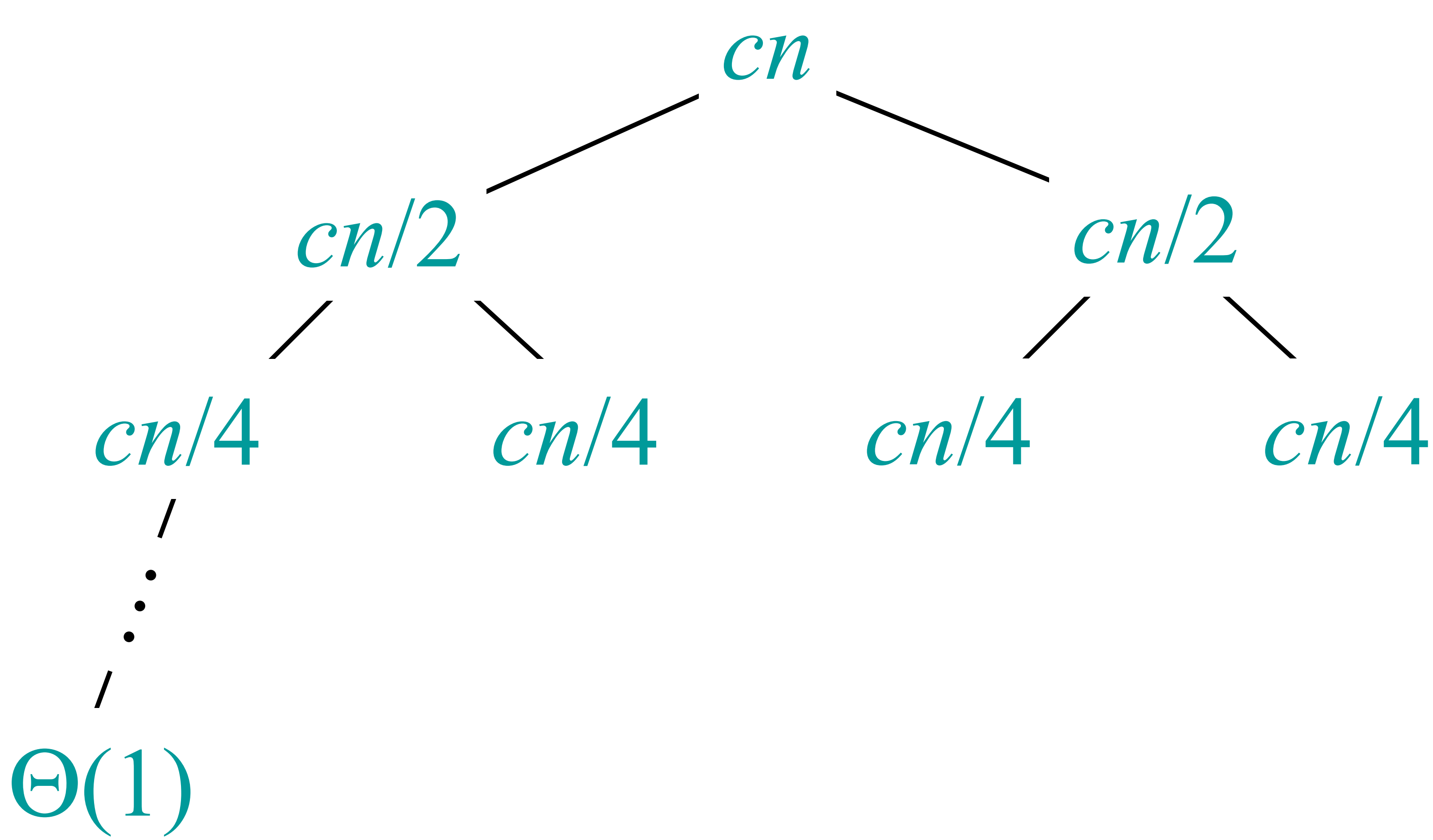
Recurrence for merge sort

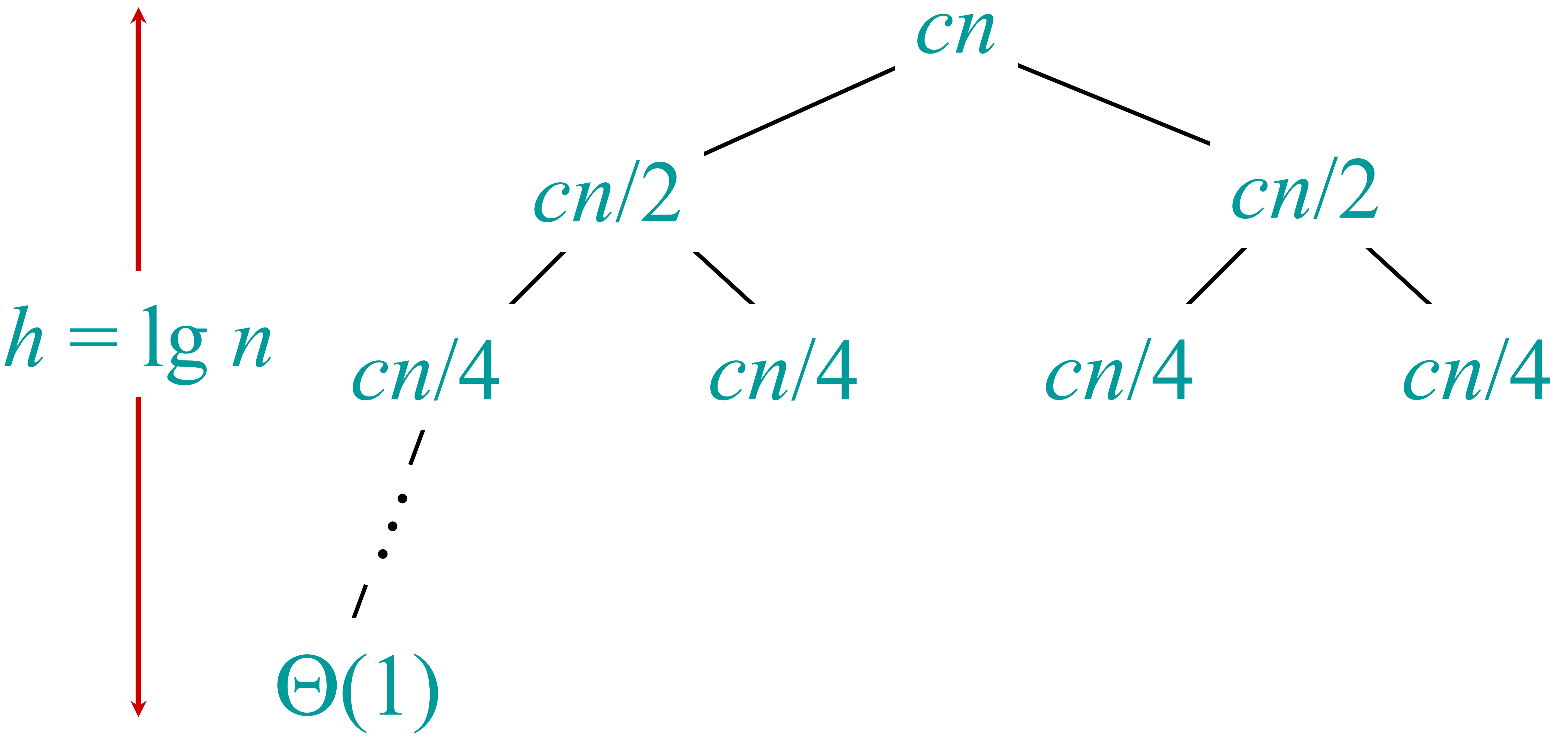
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

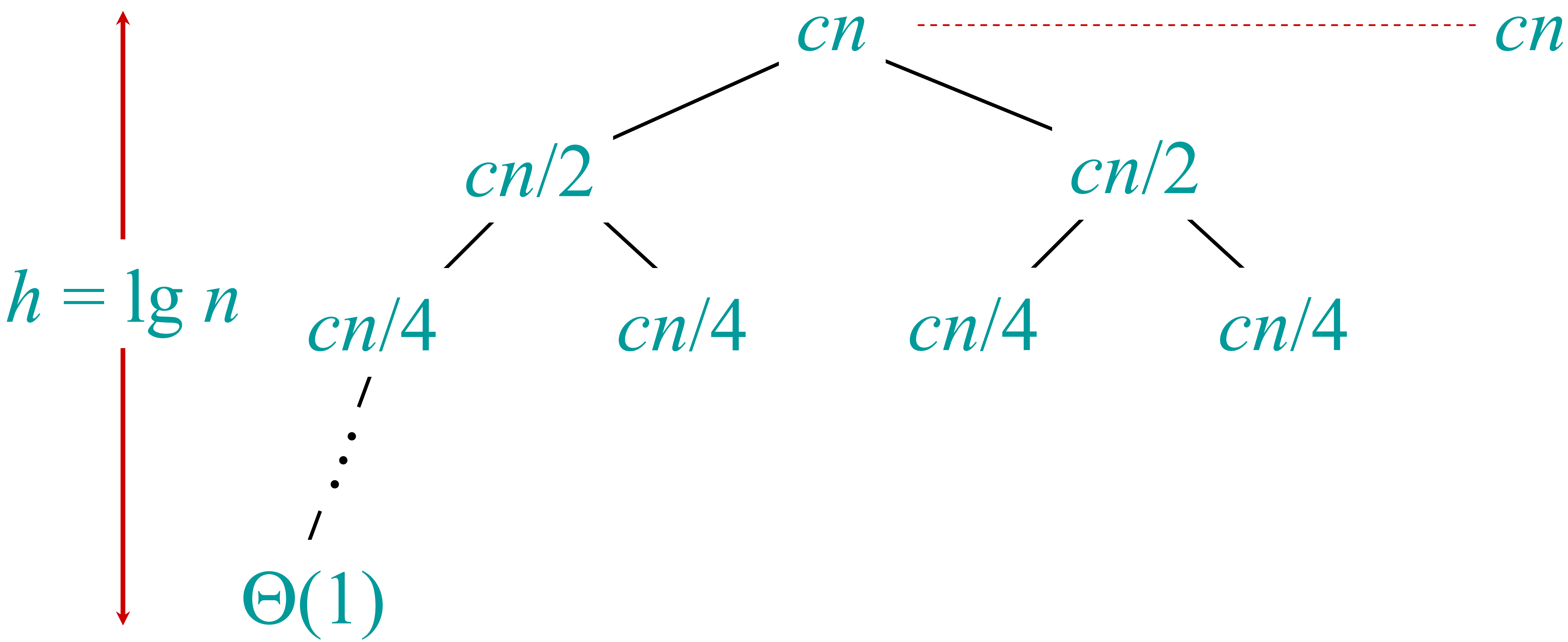
- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.

- CLRS (Textbook) provide several ways to find a good upper bound on $T(n)$.

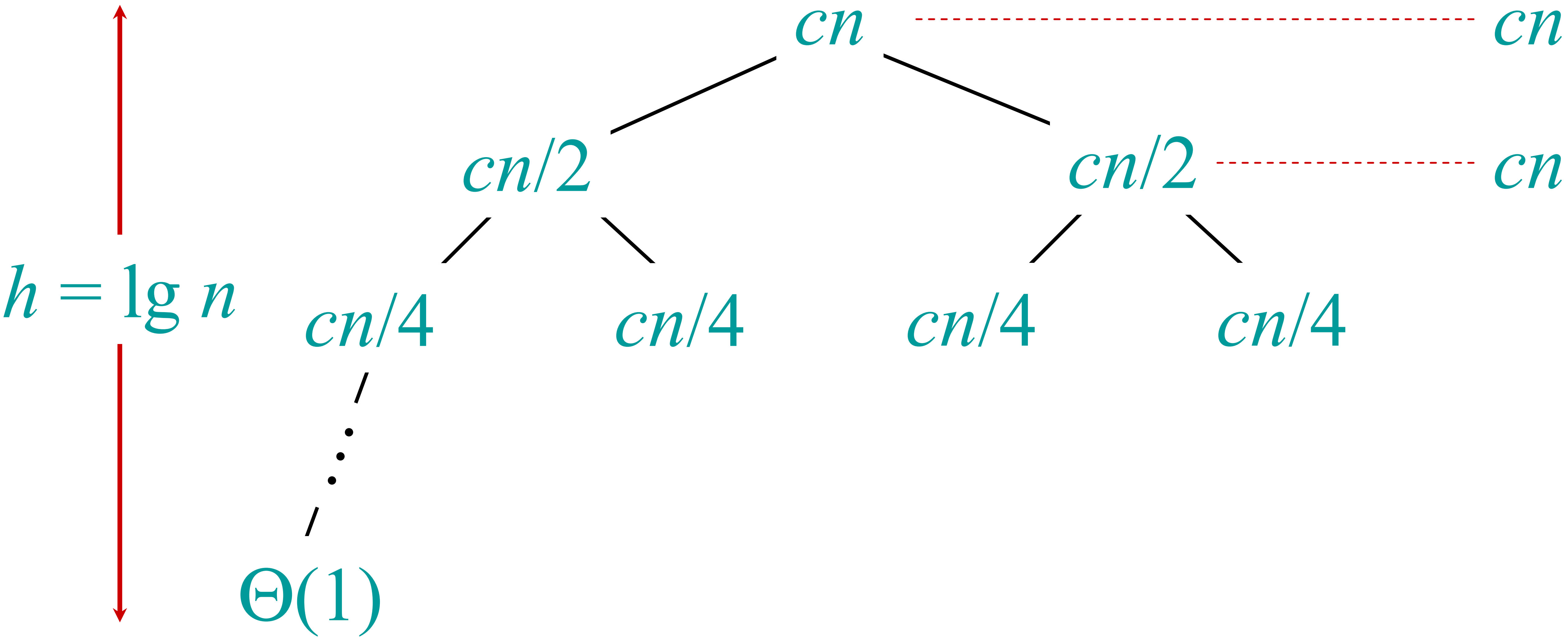
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



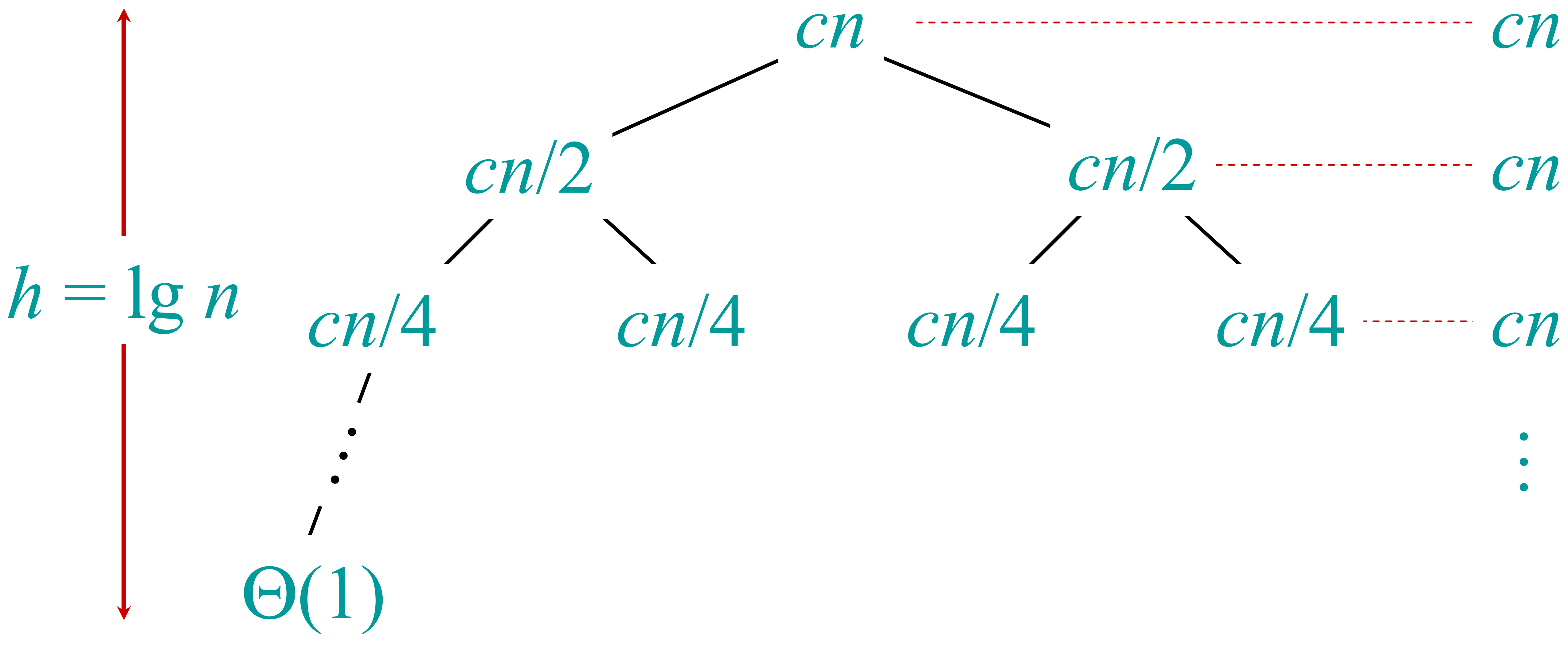

cn
$cn/2$
$cn/2$
$cn/4$
$cn/4$
$cn/4$
$cn/4$
$\Theta(1)$
$h = \lg n$



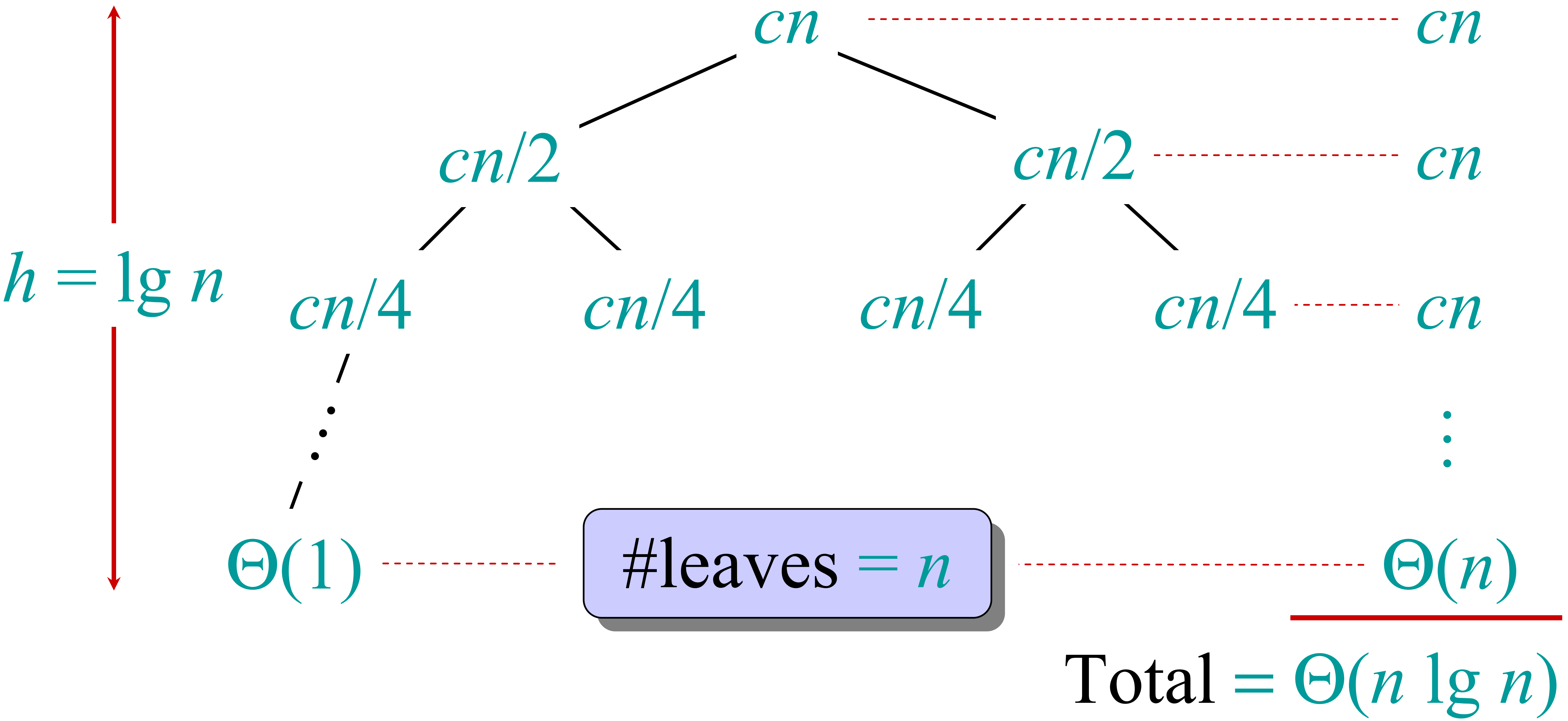
cn
cn
cn/2
cn/2
h = lg n
cn/4
cn/4
cn/4
cn/4
Θ(1)



cn
cn/2
cn/2
cn
cn
h = lg n
cn/4
cn/4
cn/4
cn/4
Θ(1)



cn
cn
$cn/2$
$cn/2$
cn
$h = \lg n$
$cn/4$
$cn/4$
$cn/4$
$cn/4$
cn
$\Theta(1)$



cn
cn/2
cn/2
cn/4
cn/4
cn/4
cn/4
h = lg n
Θ(1)
#leaves = n
cn
cn
cn
Θ(n)
Total = Θ(n lg n)

Conclusions

- $\Theta(n \log n)$ 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.
- Go test it out for yourself!