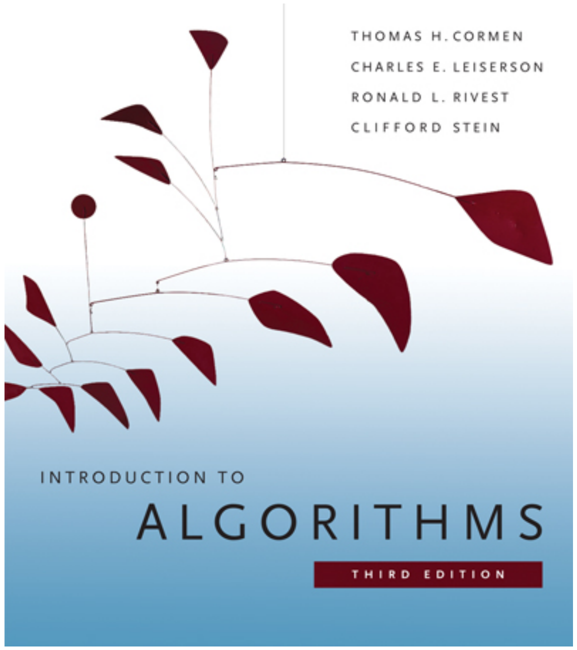
6.006- *Introduction to Algorithms*



Lecture 8

Prof. Silvio Micali

CLRS: chapter 4.

Menu

- Sorting!
  - Insertion Sort
  - Merge Sort
- Recurrences
  - Master theorem

The problem of sorting

Input: array A[1…n] of numbers.

Output: permutation B[1…n] of A such that

$$B[1] \le B[2] \le \cdots \le B[n].$$

e.g. A = [7, 2, 5, 5, 9.6] → B = [2, 5, 5, 7, 9.6]

How can we do it efficiently ?

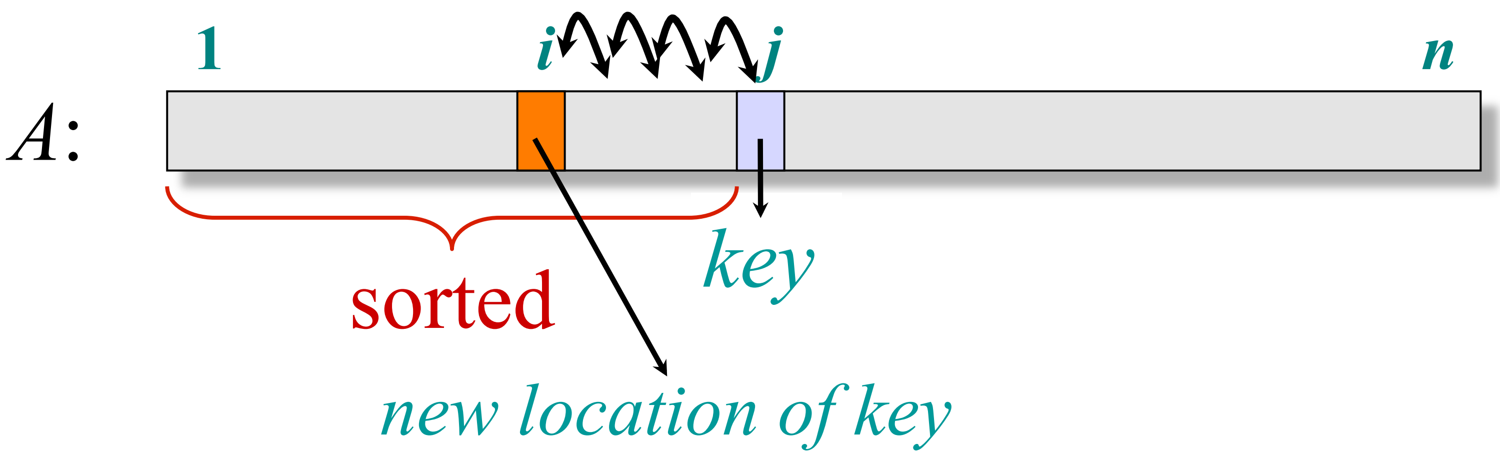
Insertion sort

INSERTION-SORT (A, n) ⊳ $A[1 \,.\,.\, n]$

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

insert key A[j] **into the (already sorted) sub-array** A[1 .. j-1] **by pairwise key-swaps down to its right position**

Illustration of iteration *j*



Example of insertion sort

8	2	4	9	3	6

Example of insertion sort

8 2 4 9 3 6 *1 swap*

Example of insertion sort

8	2	4	9	3	6
2	8	4	9	3	6

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6 *1 swap*

Example of insertion sort

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6 *0 swap*

Example of insertion sort

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6 *3 swaps*

Example of insertion sort

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

Example of insertion sort

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 swaps

Example of insertion sort

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? $O(n^2)$

e.g. when input is $A = [n, n-1, n-2, \ldots, 2, 1]$

Meet Merge Sort

divide and conquer

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

1. If $n = 1$, done (nothing to sort).
2. Otherwise, recursively sort $A[\,1 \,.\,.\, n/2\,]$ and $A[\,n/2+1 \,.\,.\, n\,]$.
3. "***Merge***" the two sorted sub-arrays.

Key subroutine: **MERGE**

Merging two sorted arrays

20	12
13	11
7	9
2	1

…

output array

Merging two sorted arrays

20 12

13 11

7 9

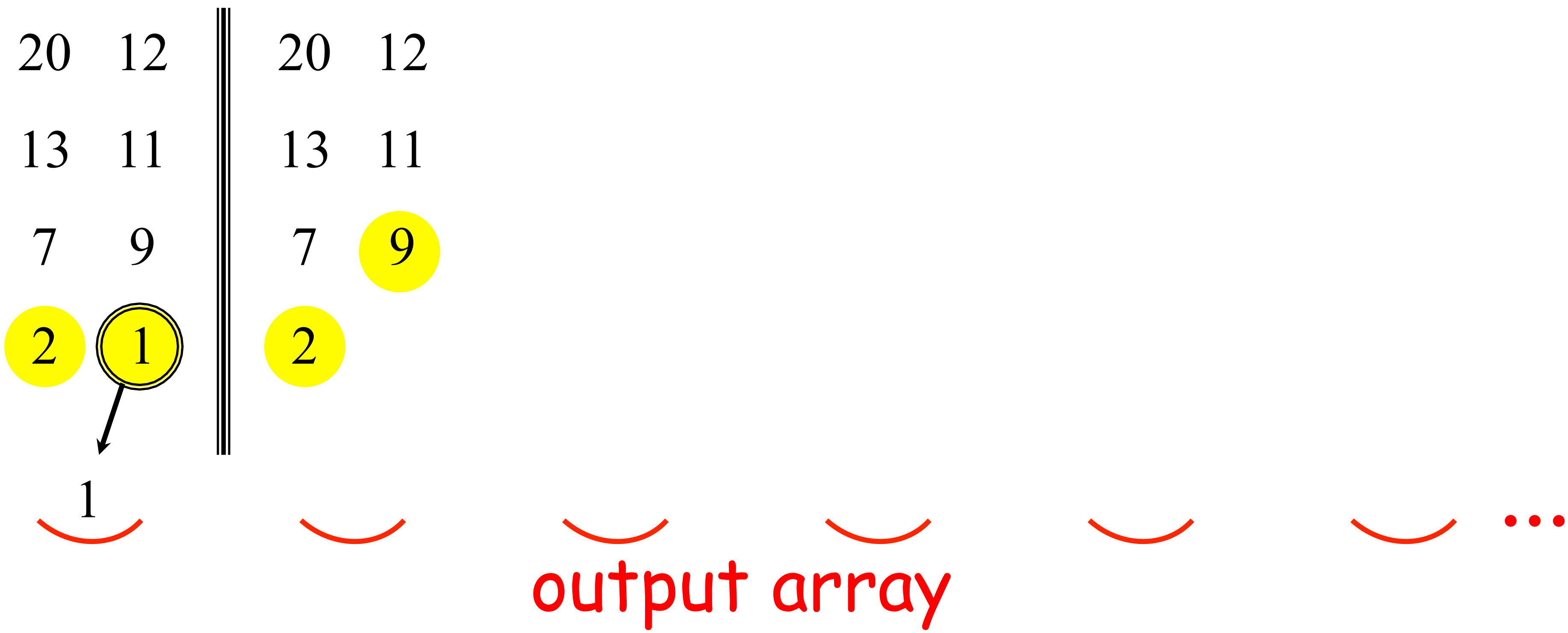
2 1

1

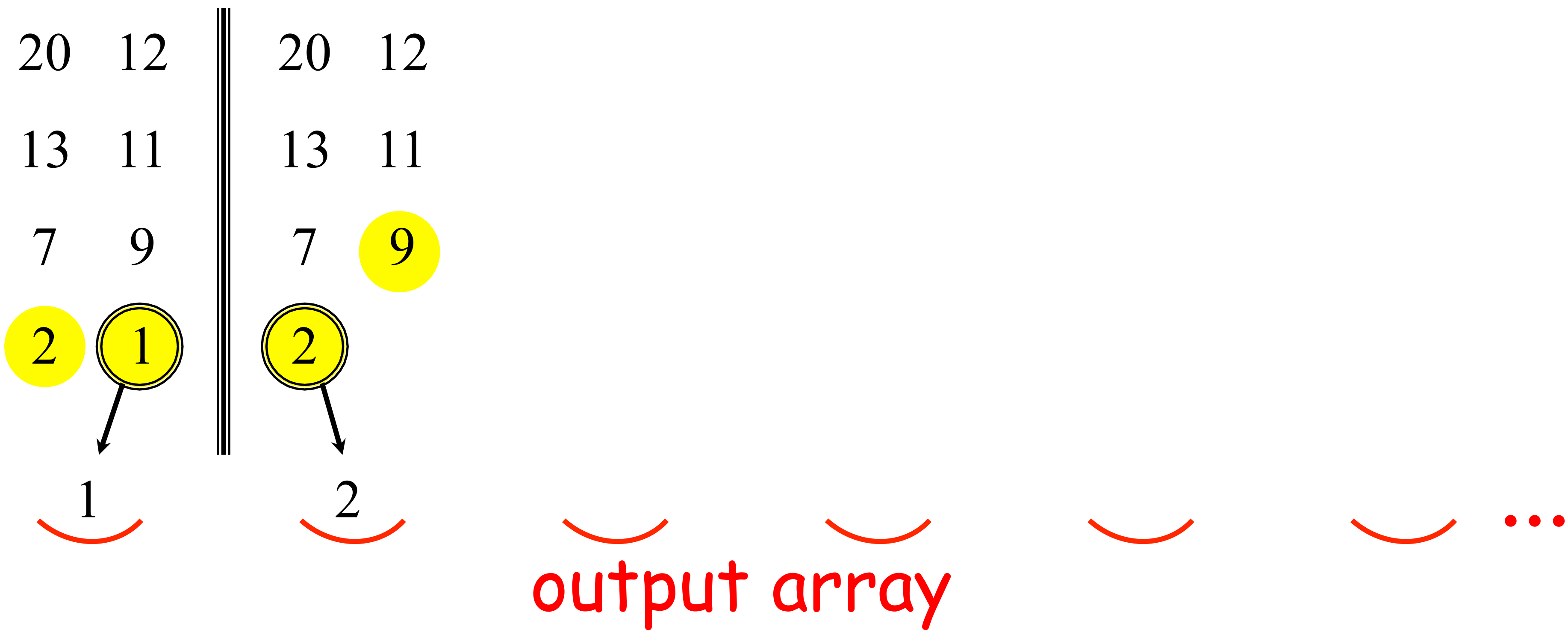
...

output array

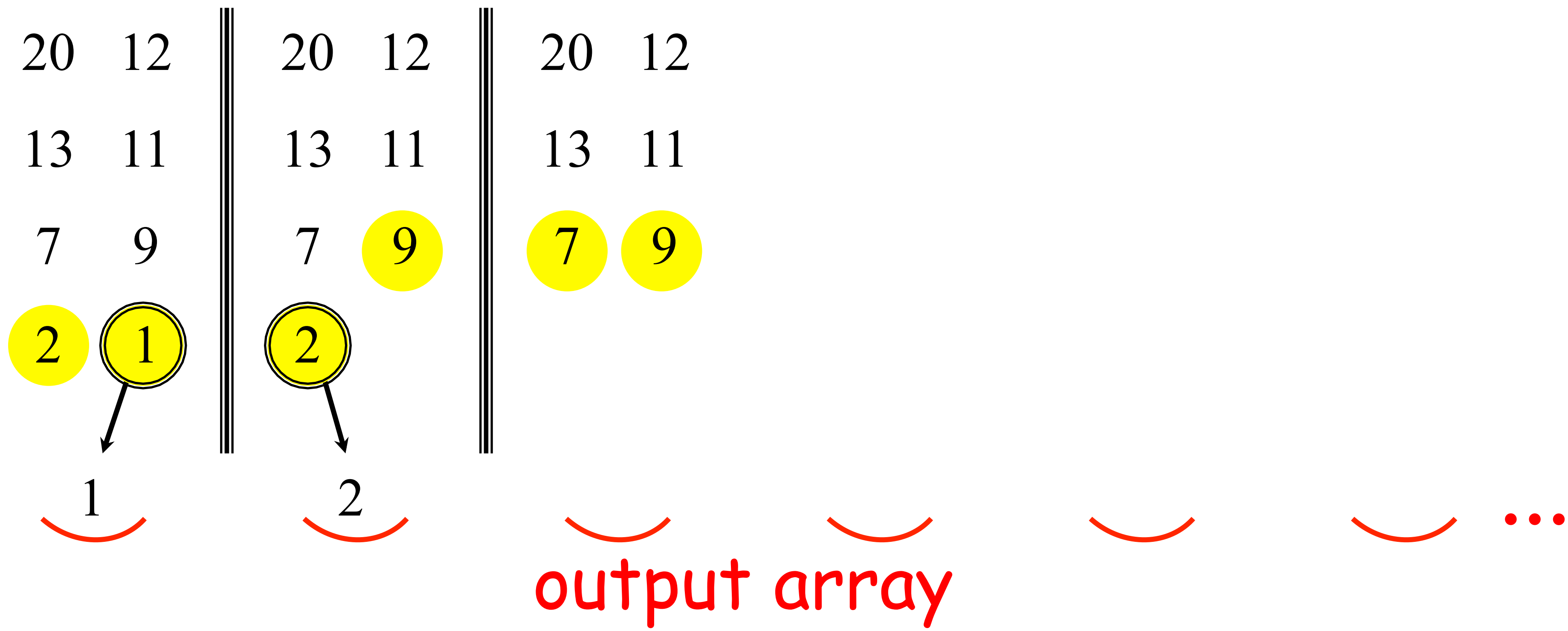
Merging two sorted arrays



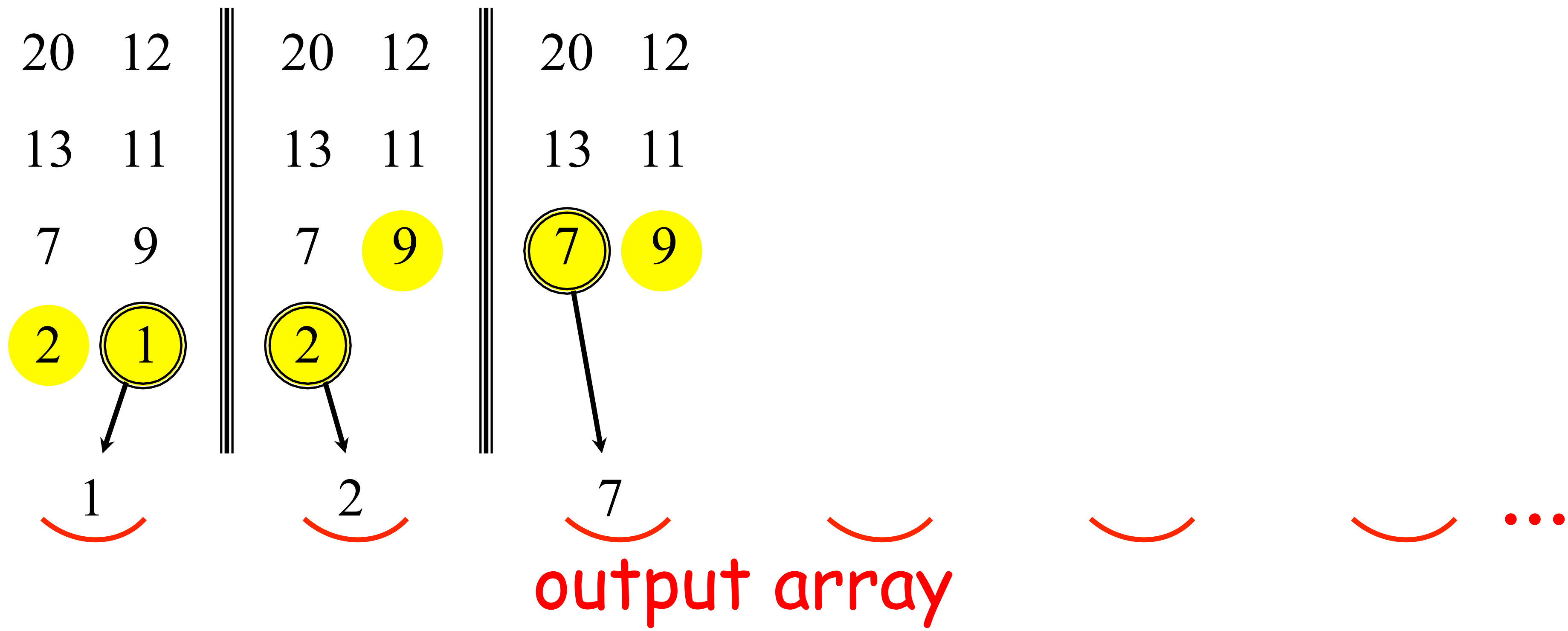
Merging two sorted arrays



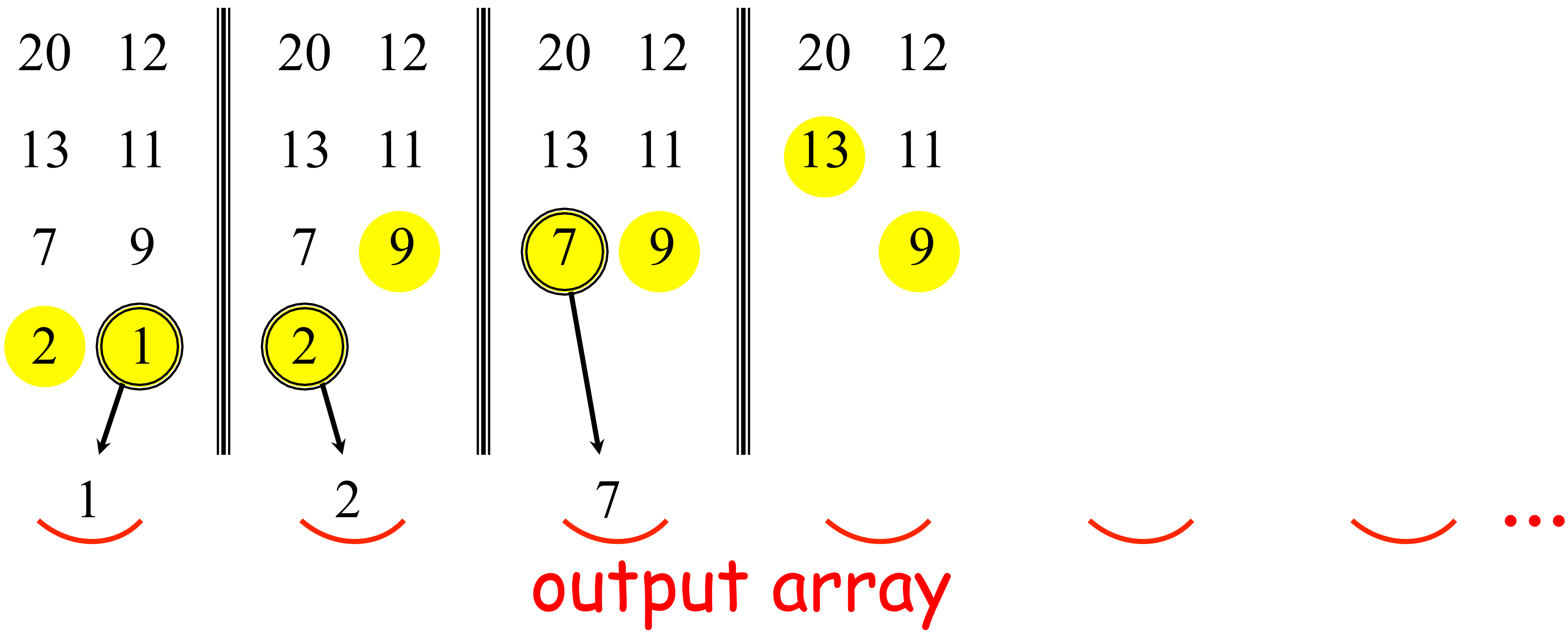
Merging two sorted arrays



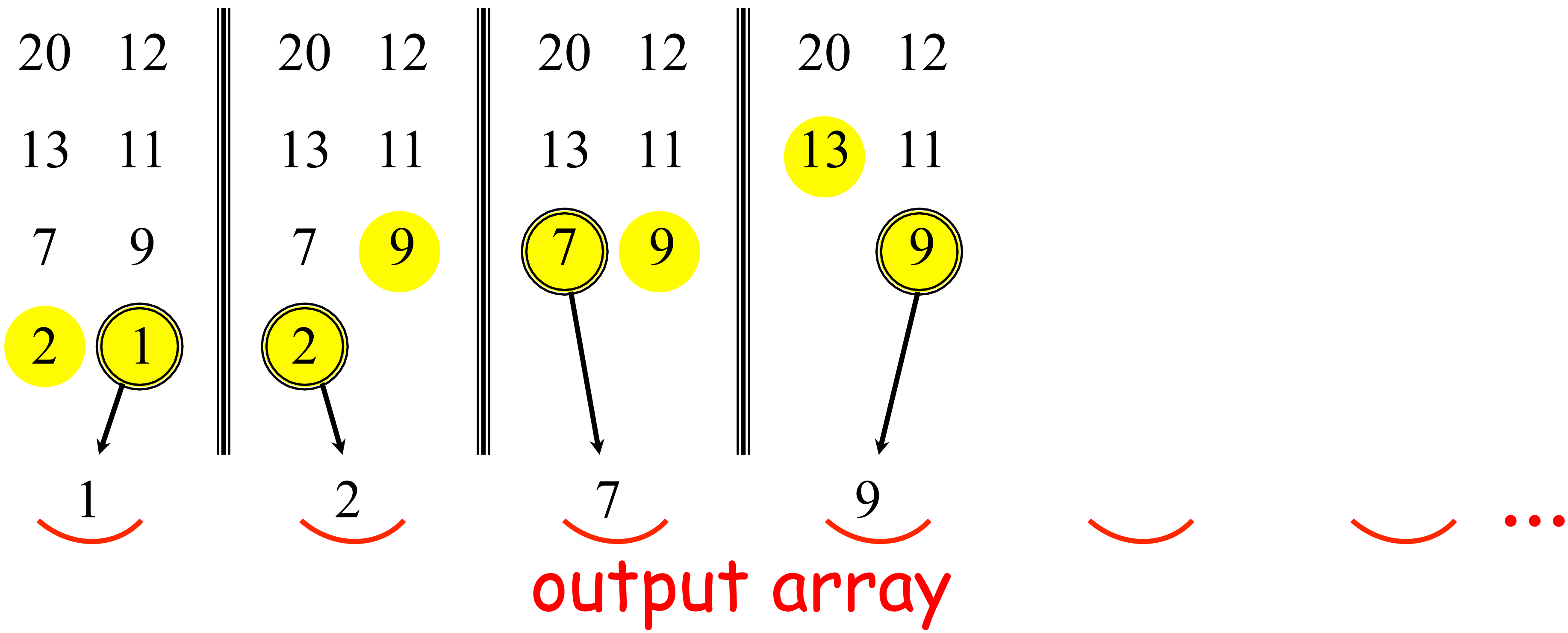
Merging two sorted arrays



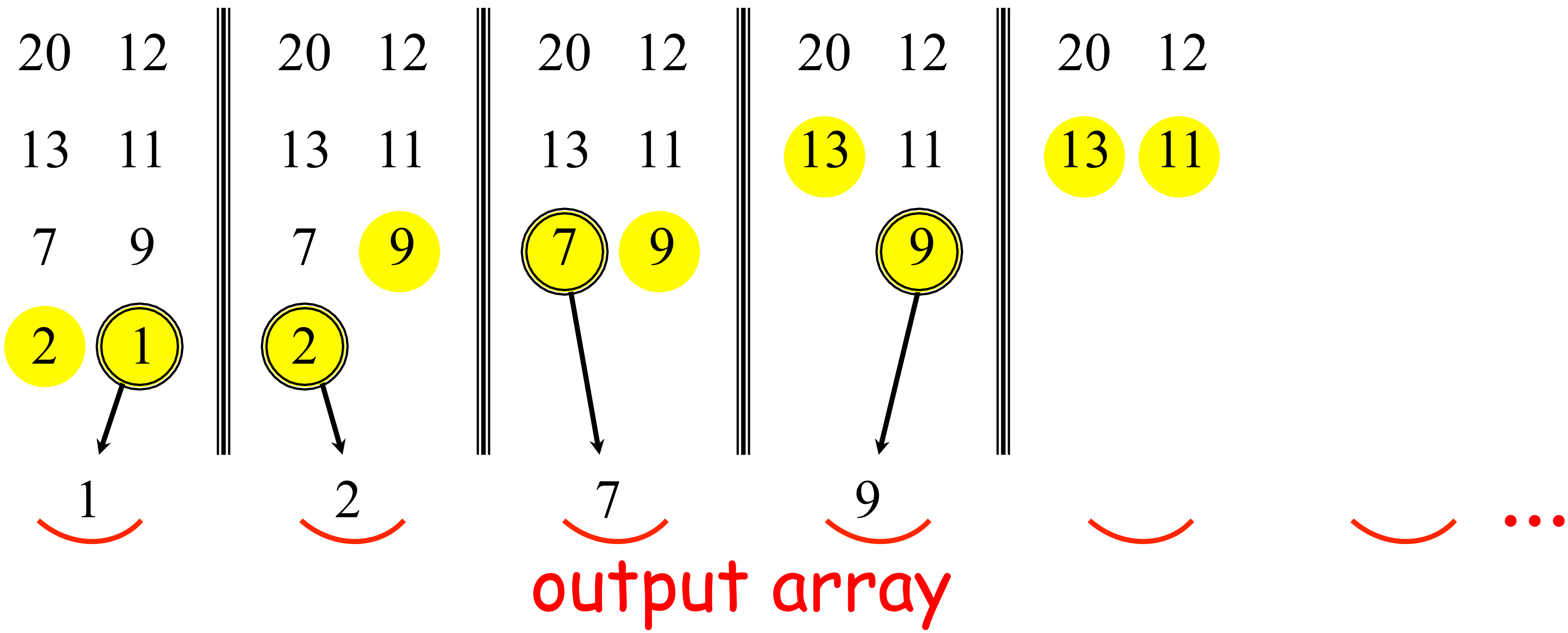
Merging two sorted arrays



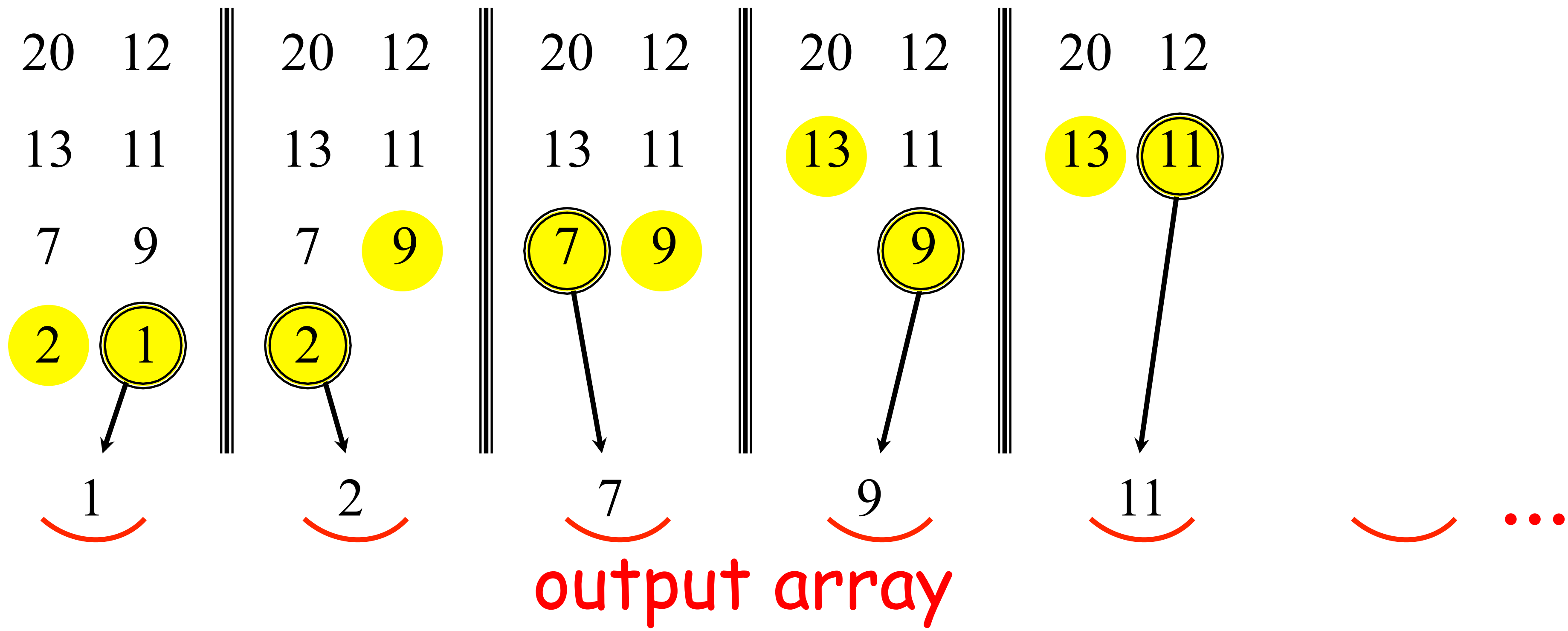
Merging two sorted arrays



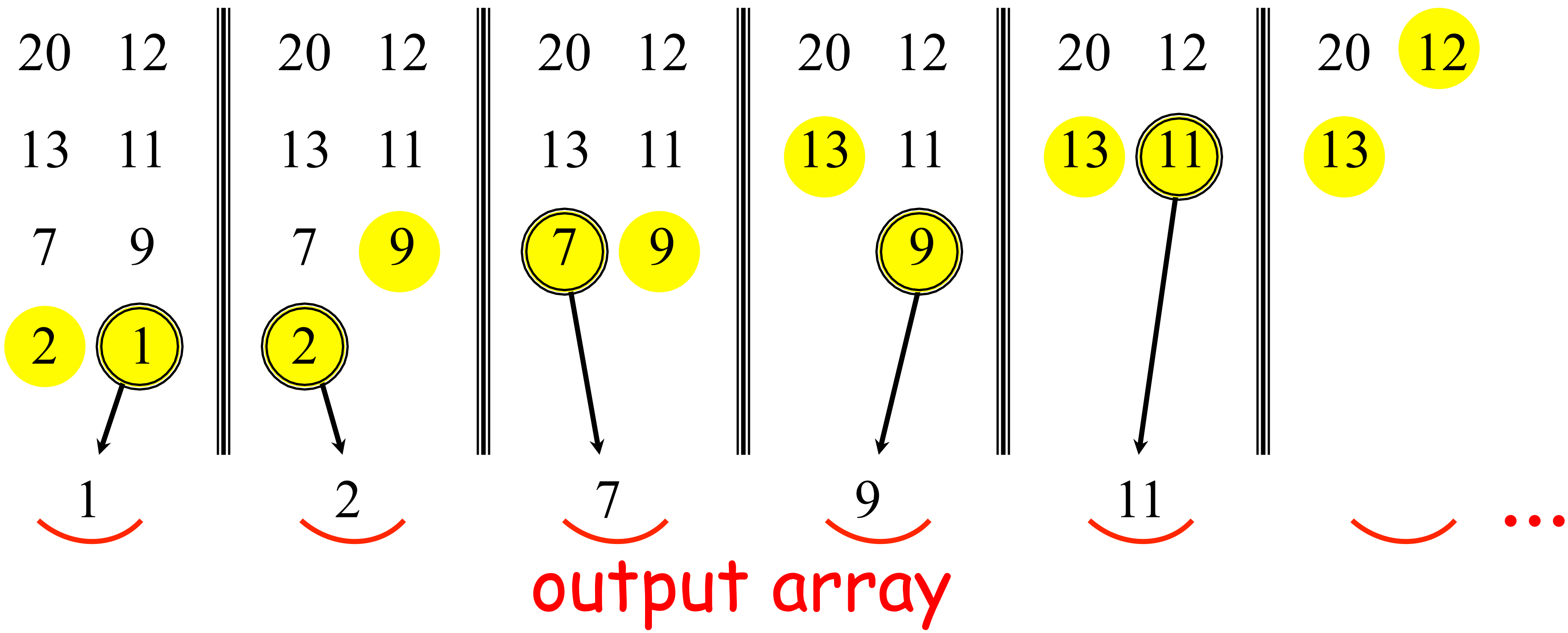
Merging two sorted arrays



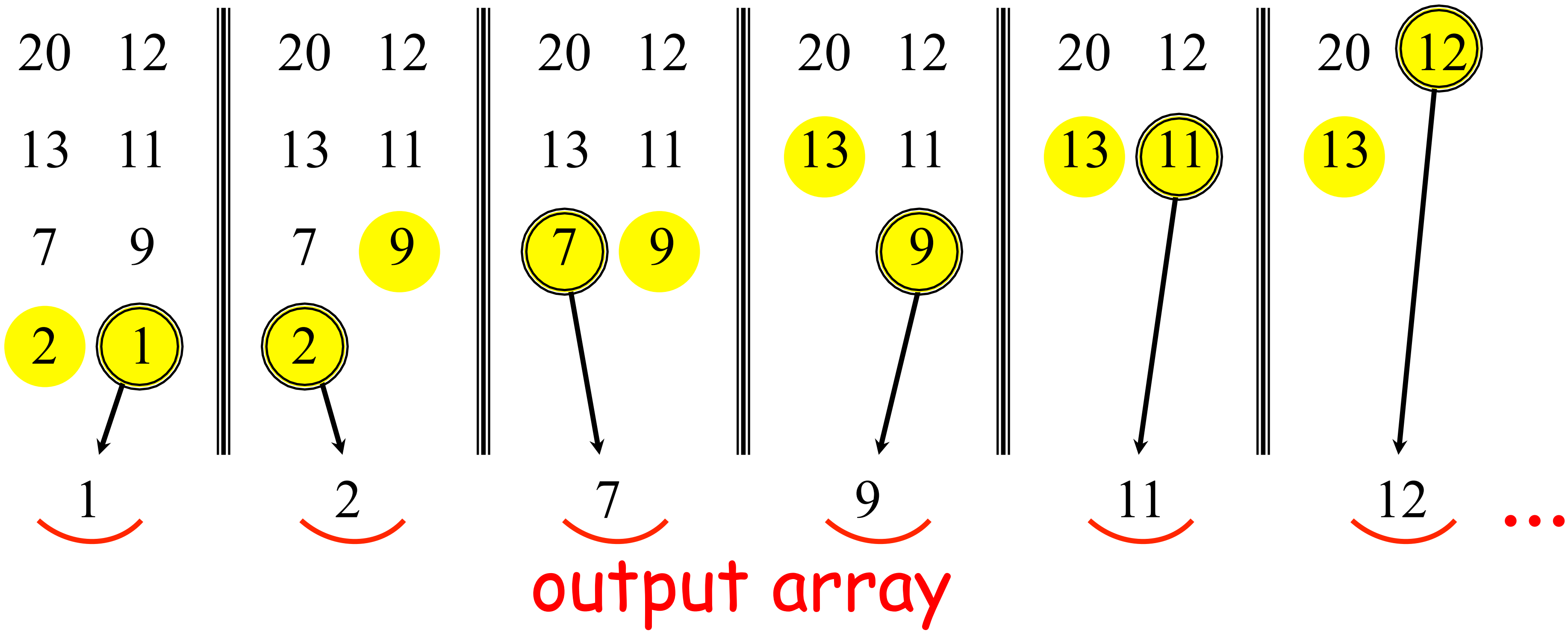
Merging two sorted arrays



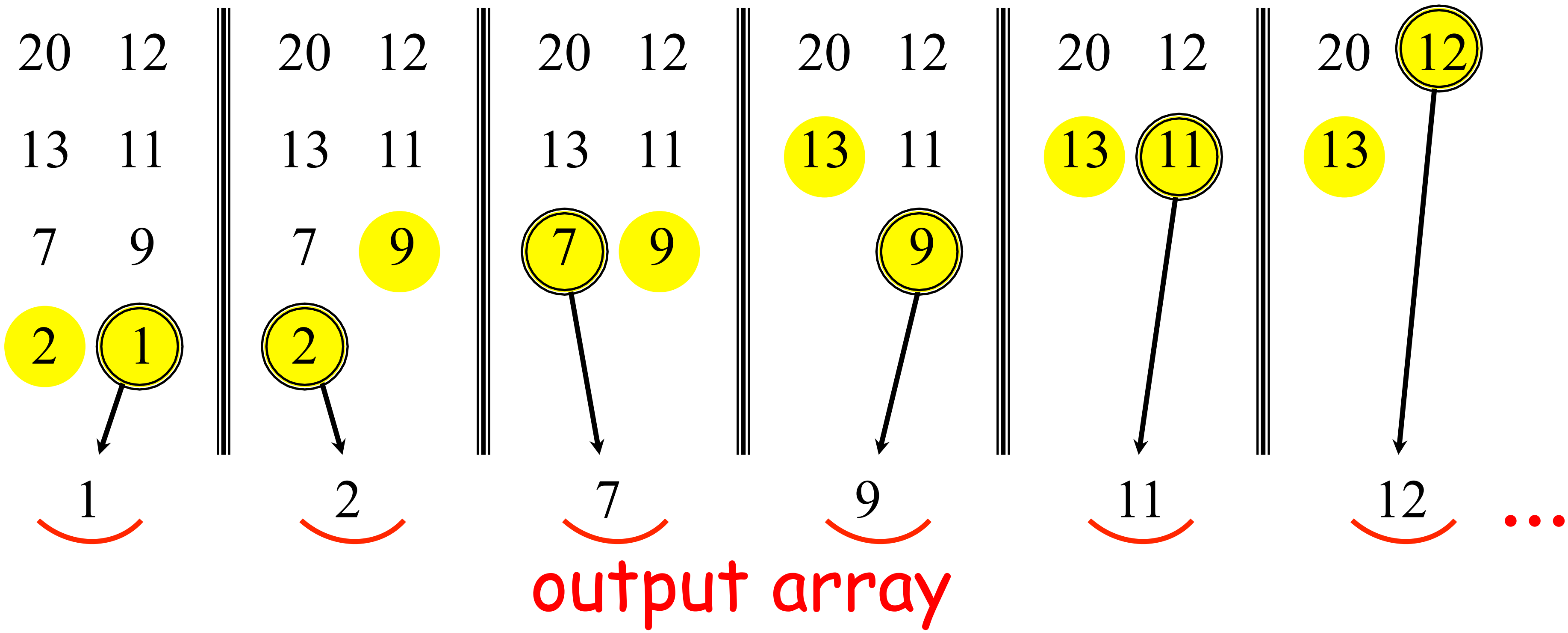
Merging two sorted arrays



Merging two sorted arrays



Merging two sorted arrays



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

Analyzing merge sort

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

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

Recurrence solving

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.

$$T(n)$$

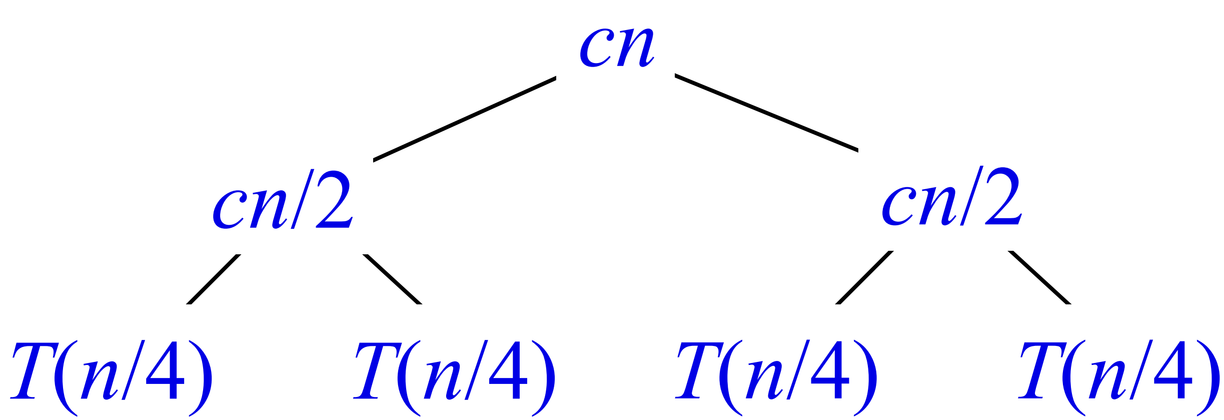
Recursion tree

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

$$
\begin{array}{ccc}
 & cn & \\
\swarrow & & \searrow \\
T(n/2) & & T(n/2)
\end{array}
$$

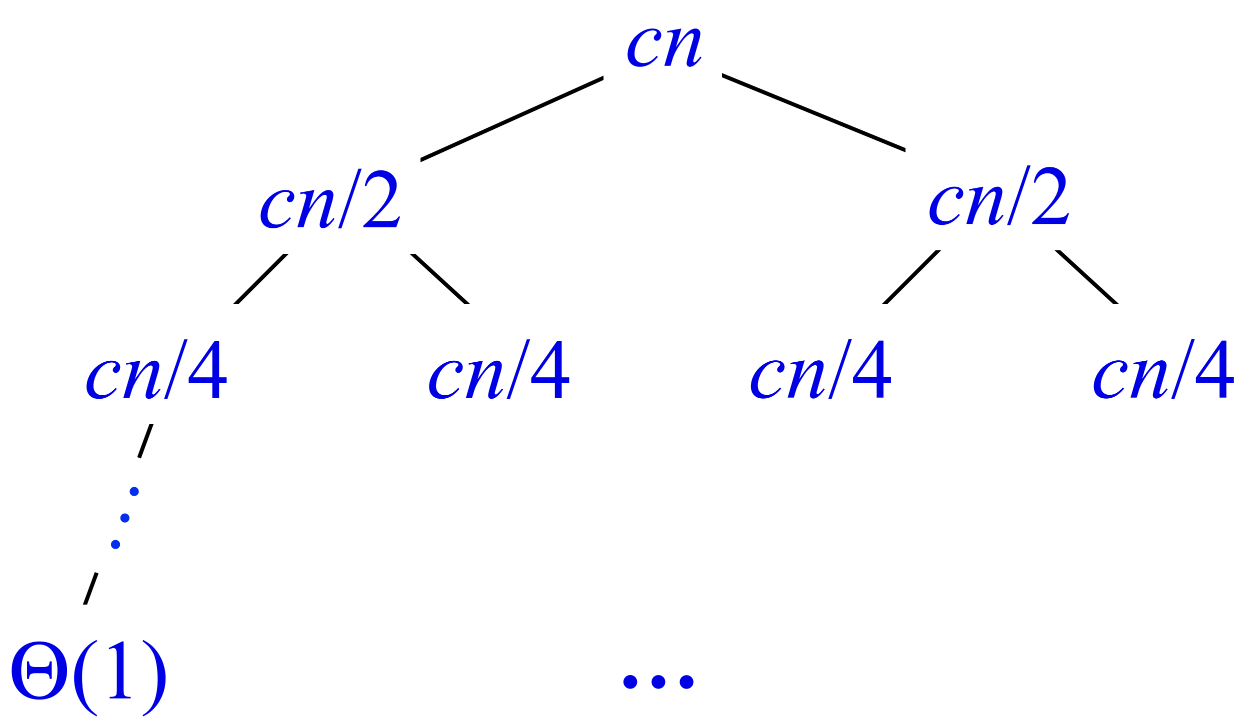
Recursion tree

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



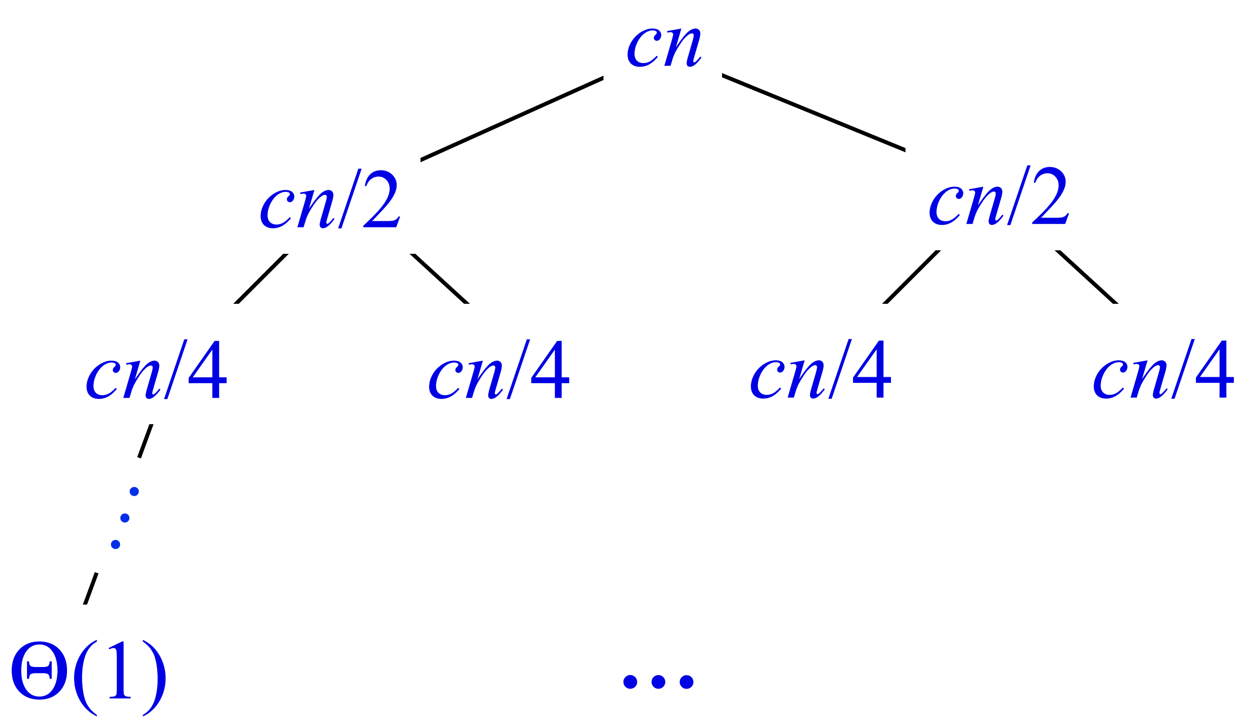
Recursion tree

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



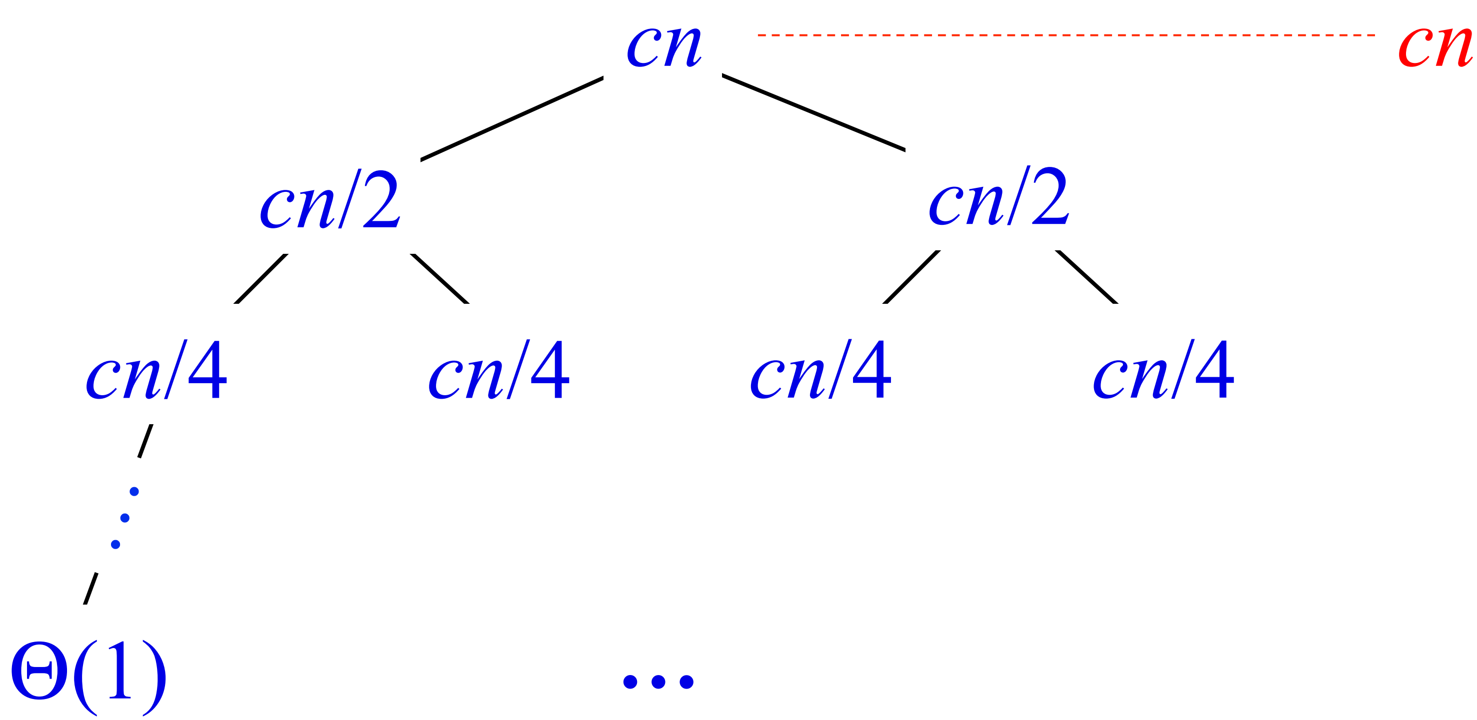
Recursion tree

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



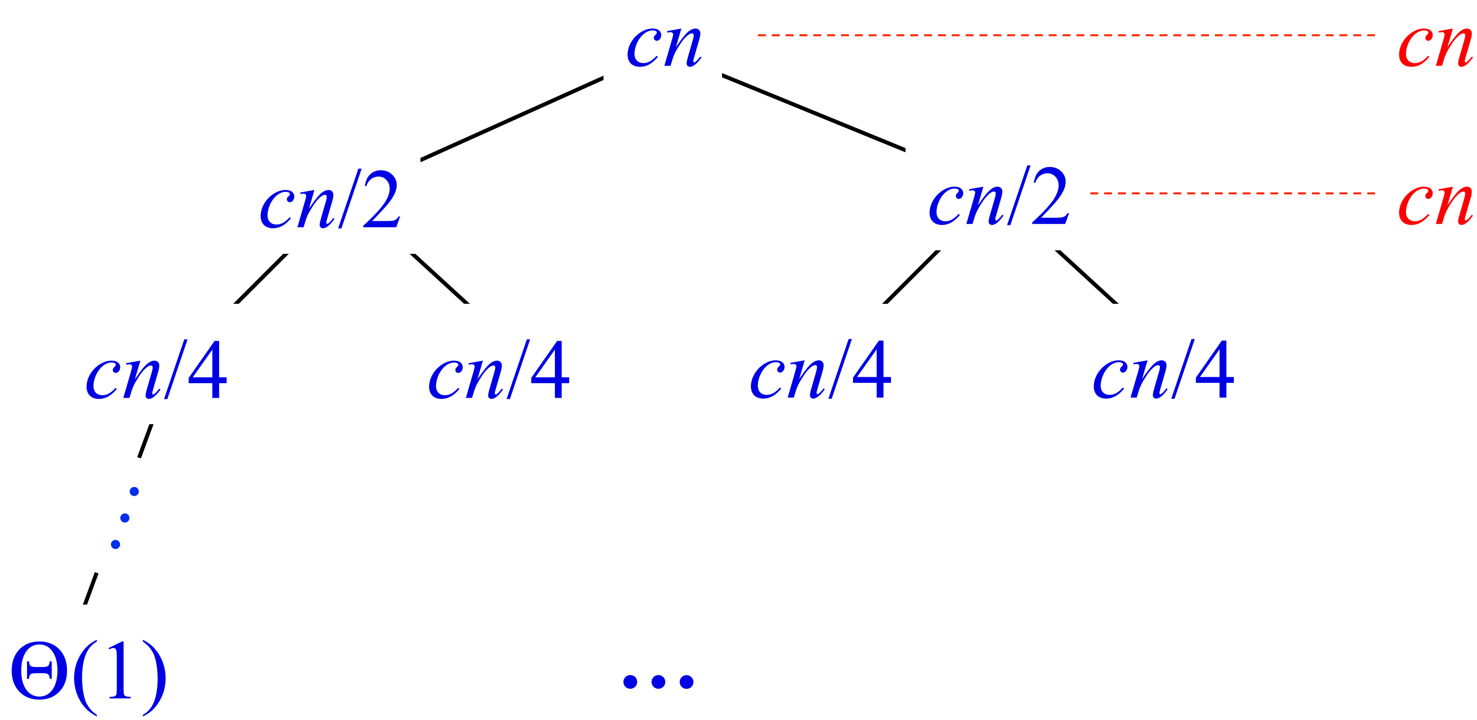
Recursion tree

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



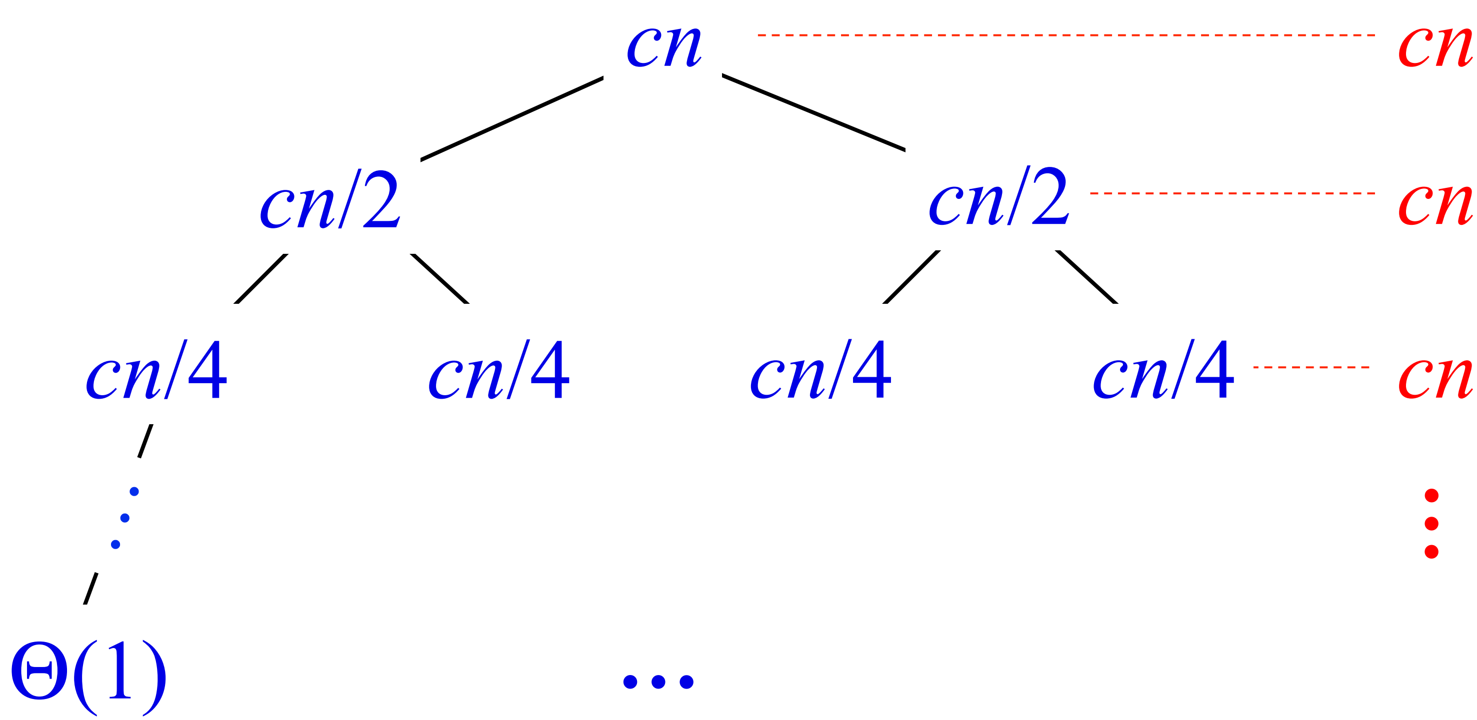
Recursion tree

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



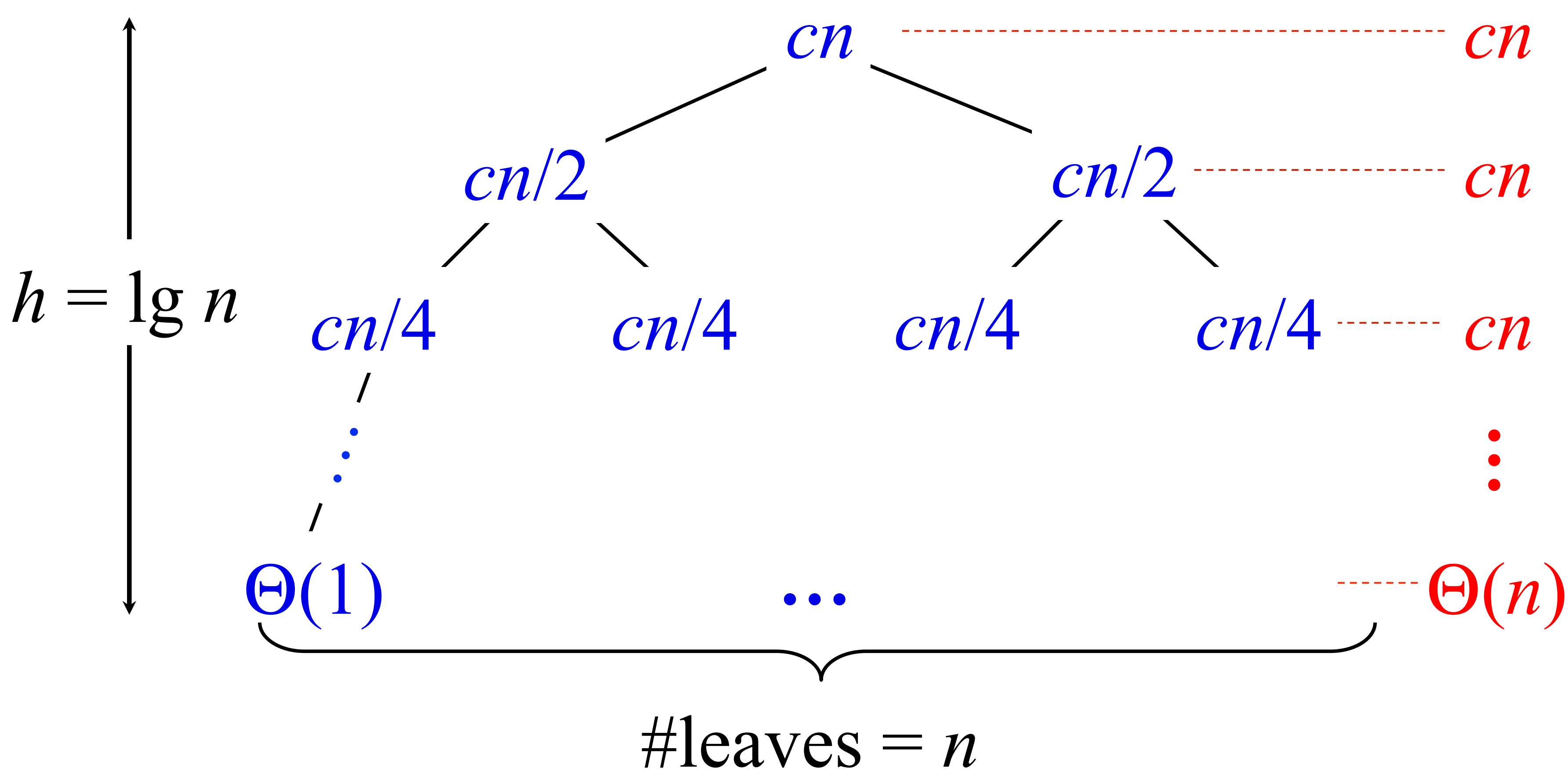
Recursion tree

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



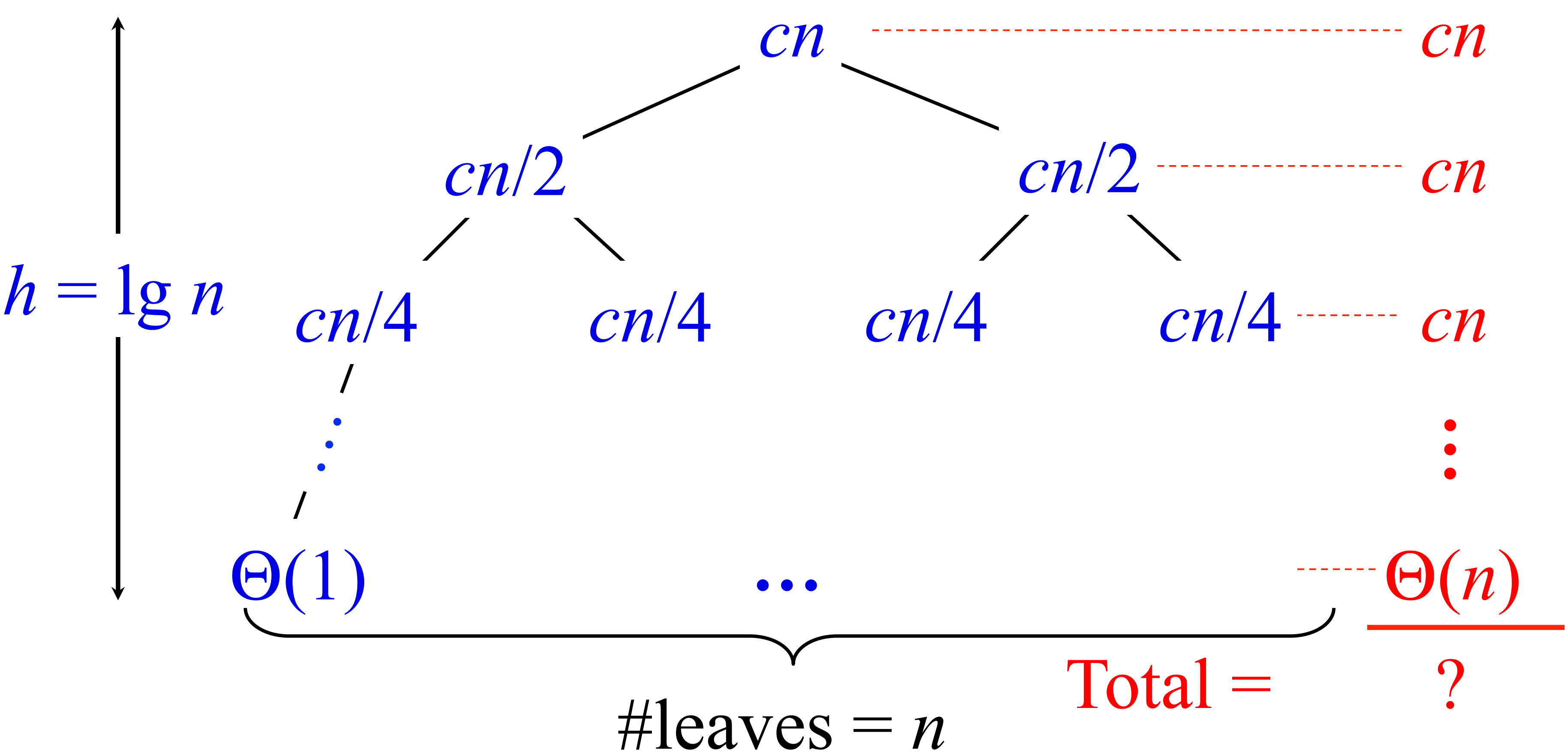
Recursion tree

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



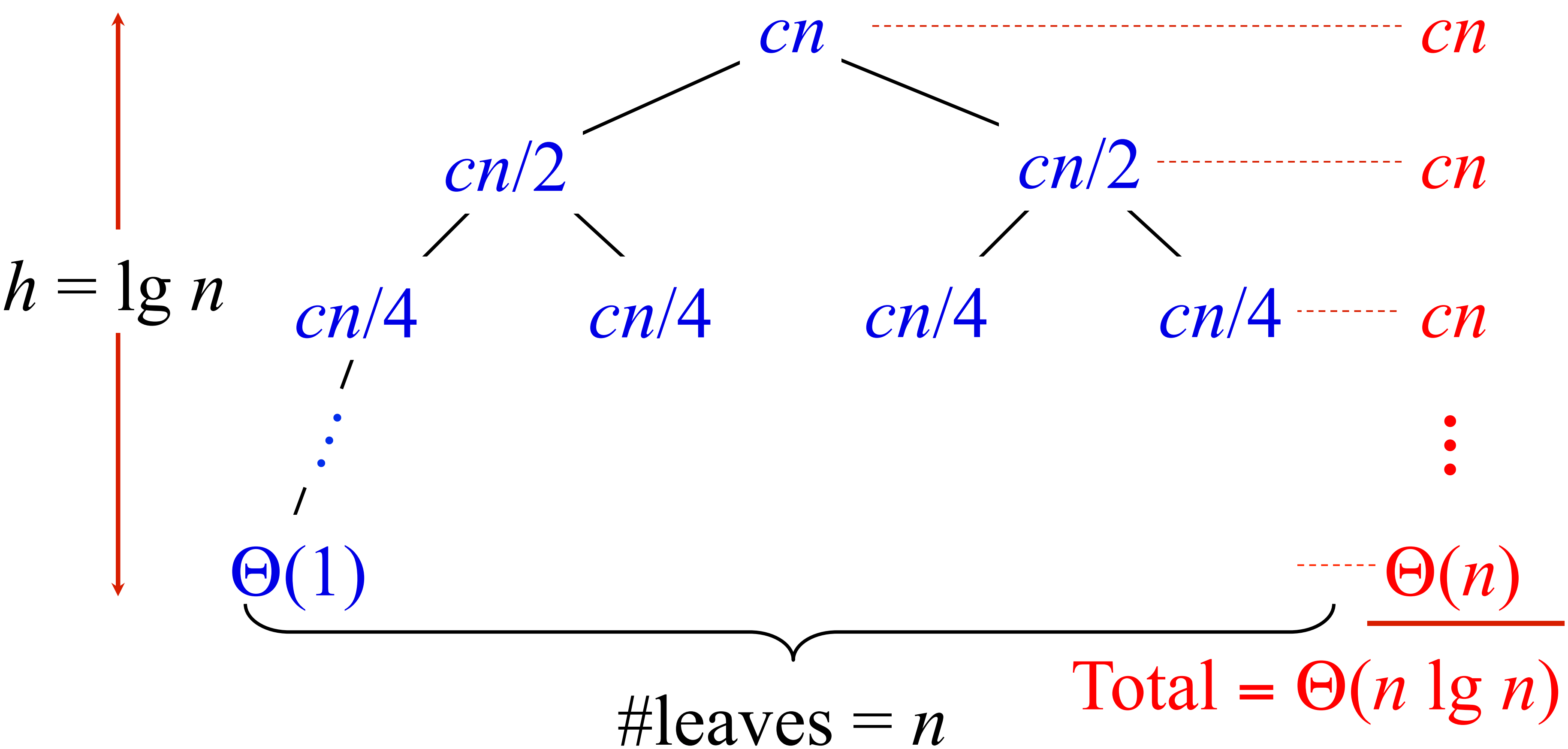
Recursion tree

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



Recursion tree

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



Total = $\Theta(n \lg n)$

The master method

"One theorem for all recurrences" (sort of)

It applies to recurrences of the form

#subproblems

size of each subproblem

time to split into subproblems and combine results

$$T(n) = \underbrace{a}_{\text{\#subproblems}}\, T(n/\underbrace{b}_{\text{size of each subproblem}}) + \underbrace{f(n)}_{\text{time to split into subproblems and combine results}},$$

where $a \geq 1$, $b > 1$, and f is positive.

e.g. Mergesort: a=

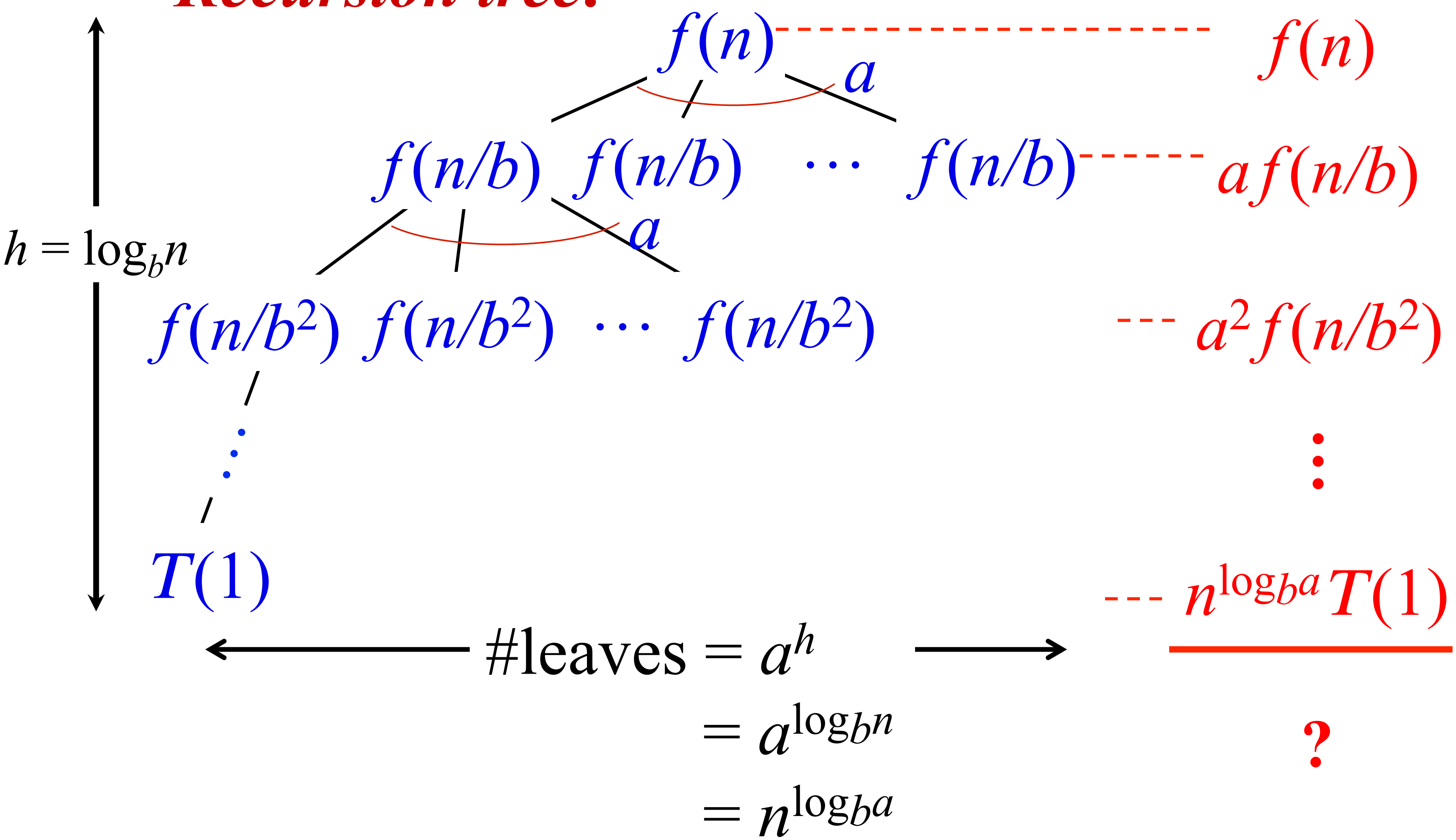
e.g.2 Binary Search: a=

Basic Idea: Compare $f(n)$ with $n^{\log_b a}$.

Idea of master theorem

$$T(n) = a\,T(n/b) + f(n)$$

Recursion tree:



Idea of master theorem

$$T(n) = a\,T(n/b) + f(n)$$

Recursion tree:

$h = \log_b n$

$f(n)$ ------------------------------ $f(n)$

a

$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$ ------ $a\,f(n/b)$

a

$f(n/b^2)$ $f(n/b^2)$ $\cdots$ $f(n/b^2)$ ------------- $a^2 f(n/b^2)$

$\vdots$

$T(1)$

CASE 1: The weight increases geometrically from root to leaves.

$\Rightarrow$ **Leaves hold a constant fraction of total weight!**

$n^{\log_b a}\,T(1)$

$\Theta(n^{\log_b a})$

Idea of master theorem

$$T(n) = a\,T(n/b) + f(n)$$

Recursion tree:

$f(n)$ ---------- $f(n)$

a

$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$ ---------- $a\,f(n/b)$

a

$f(n/b^2)$ $f(n/b^2)$ $\cdots$ $f(n/b^2)$ ---------- $a^2 f(n/b^2)$

$\vdots$

$h = \log_b n$

$T(1)$

$n^{\log_b a}\,T(1)$

CASE 2: Weight approximately the same on each level.

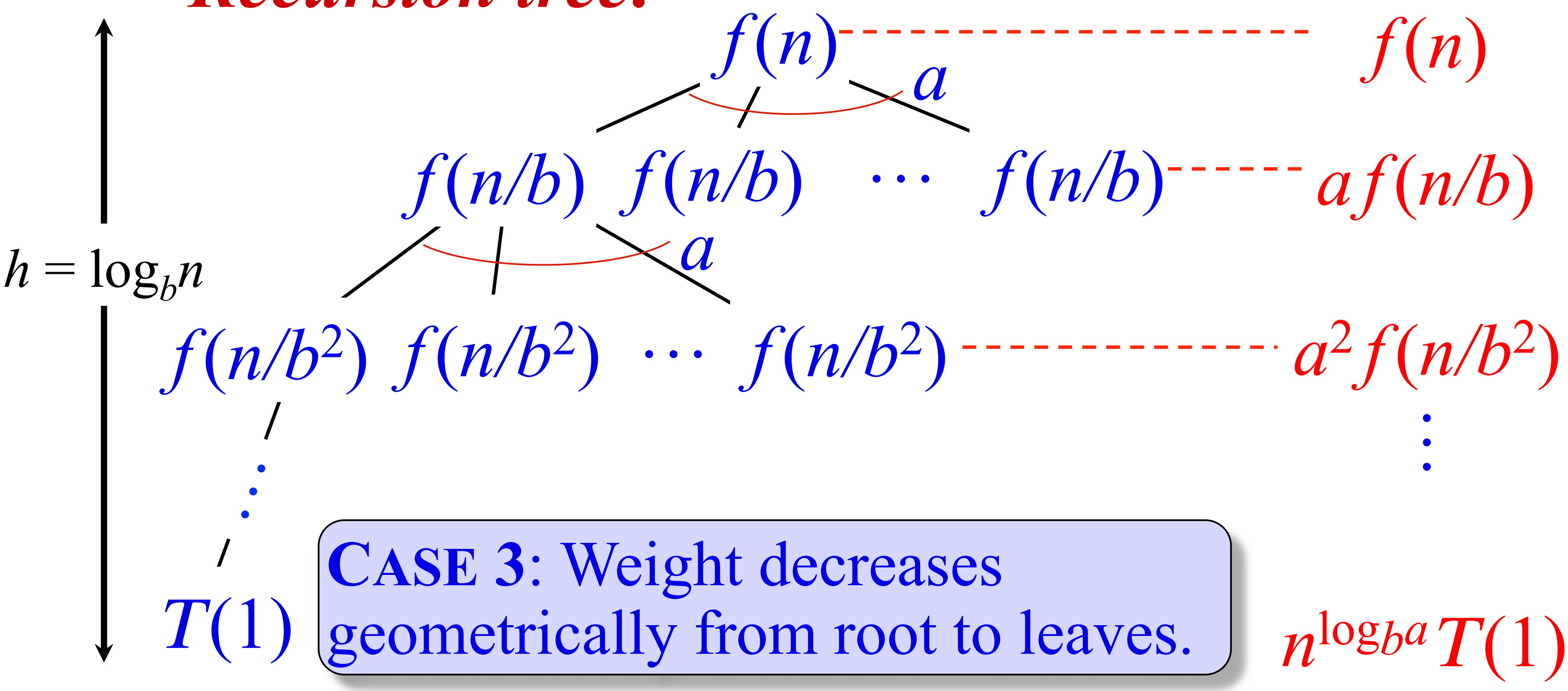
$\Rightarrow$ **Total weight $\approx$ #levels $\times$ leaves' weight!**

$\Theta(n^{\log_b a} ? \log_b a)$

Idea of master theorem

$$T(n) = a\,T(n/b) + f(n)$$

Recursion tree:



CASE 3: Weight decreases geometrically from root to leaves.

$\Rightarrow$ **Root holds a constant fraction of total weight!**

$\Theta(f(n))$

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = \Theta(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

I.e., $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

cost of level $i = a^{i} f(n/b^{i}) = \Theta(n^{\log_b a - \varepsilon} \cdot (b^{\varepsilon})^{i})$

so geometric increase of cost as we go deeper in the tree

hence, leaf level cost dominates!

Solution: $T(n) = \Theta(n^{\log_b a})$.

Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

2. $f(n) = \Theta(n^{\log_b a} \log_b^k n)$ for some constant $k \geq 0$.

I.e., $f(n)$ and $n^{\log_b a}$ grow at similar rates.

(cost of level i) = $a^i f(n/b^i) = \Theta(n^{\log_b a} \cdot \log\_b^k(n/b^i))$

so all levels have about the same cost

Solution: $T(n) = \Theta(n^{\log_b a} \log_b^{k+1} n)$

Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Theta(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.

I.e., $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor).

(cost of level i) = $a^i f(n/b^i) = \Theta(n^{\log_b a+\varepsilon} \cdot b^{-i\varepsilon})$

so geometric decrease of cost as we go deeper in the tree

hence, root cost dominates!

Solution: $T(n) = \Theta(f(n))$.

Example1

$$T(n) = 2T(n/2) + 1$$

Please don't!

Use Master Theorem:

1. Compute a and b
2. Compute $n^{\log_b a}$ and $f(n)$
3. Compare

$a = 2,\ b = 2 \Rightarrow n^{\log_b a} = n$, $f(n) = 1$

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon = 1$

$\therefore\ T(n) = \Theta(n)$.

Example 2

$$T(n) = 2T(n/2) + n$$

$a = 2, b = 2 \quad \Rightarrow \quad n^{\log_b a} = n \quad f(n) = n$

Case 2: $f(n) = \Theta(n \lg^0 n)$, that is, $k = 0$

$\therefore\ T(n) = \Theta(n \lg n)$.

Example 3

$$T(n) = 4T(n/2) + n^3$$

$a = 4, b = 2 \quad \Rightarrow \quad n^{\log_b a} = n^2 \quad f(n) = n^3$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

$\therefore\ T(n) = \Theta(n^3)$.

Let's go master the rest of the day!