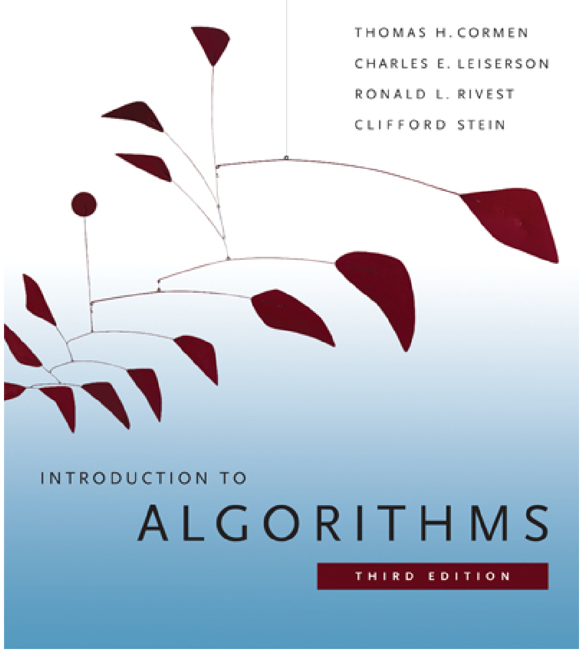
6.006- *Introduction to Algorithms*



Lecture 8

Prof. Piotr Indyk

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] £ B[2] £ ⋯ £ 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

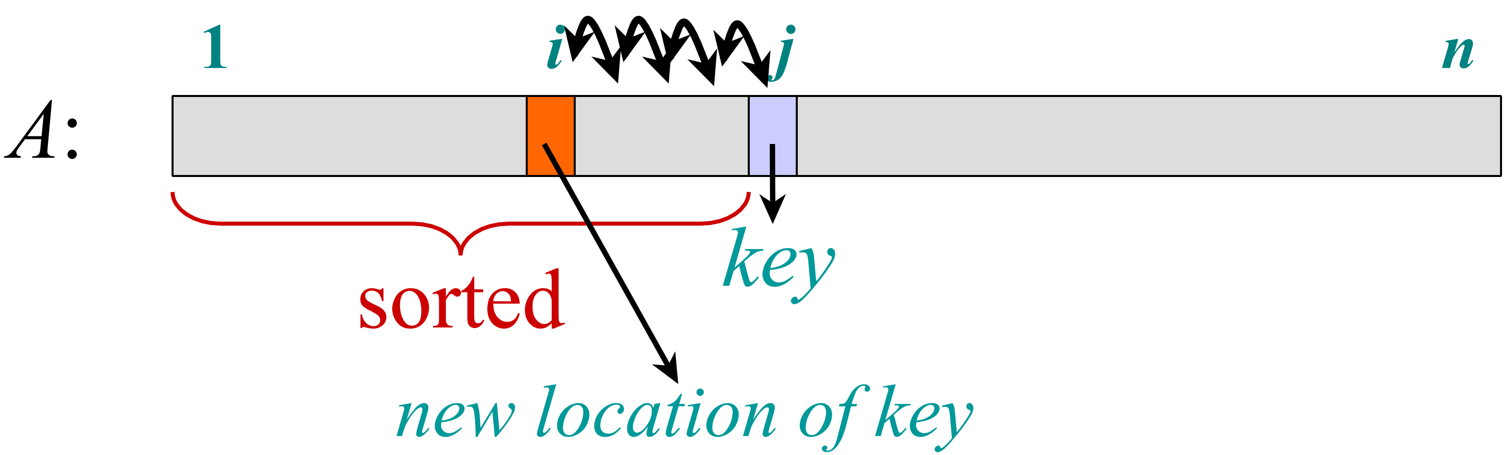
INSERTION-SORT (A, n) ⊳ $A[1 \mathinner{..} n]$

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

insert key $A[j]$ **into the (already sorted) sub-array** $A[1 \mathinner{..} j\text{-}1]$.

by pairwise key-swaps down to its right position

Illustration of iteration $\boldsymbol{j}$



Example of insertion sort

8 2 4 9 3 6

Example of insertion sort

8 2 4 9 3 6

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

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

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

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

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? $\Theta(n^2)$

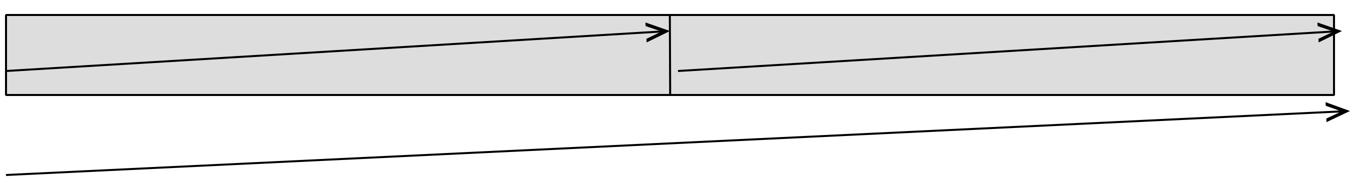
e.g. when input is $A = [n, n-1, n-2, \ldots, 2, 1]$

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

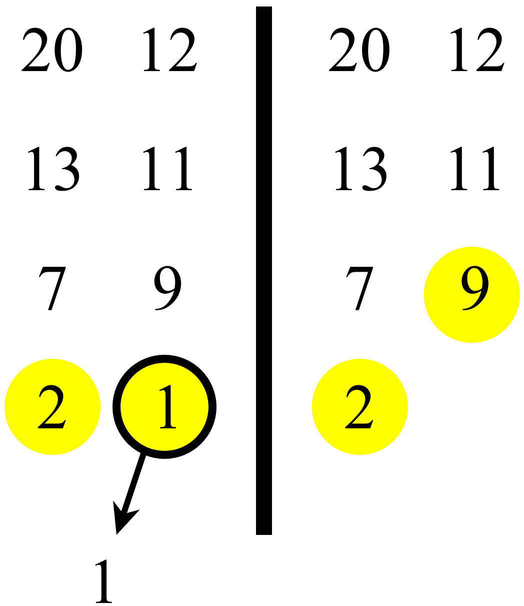
20	12
13	11
7	9
2	1

Merging two sorted arrays

20	12
13	11
7	9
2	1

1

Merging two sorted arrays



Merging two sorted arrays

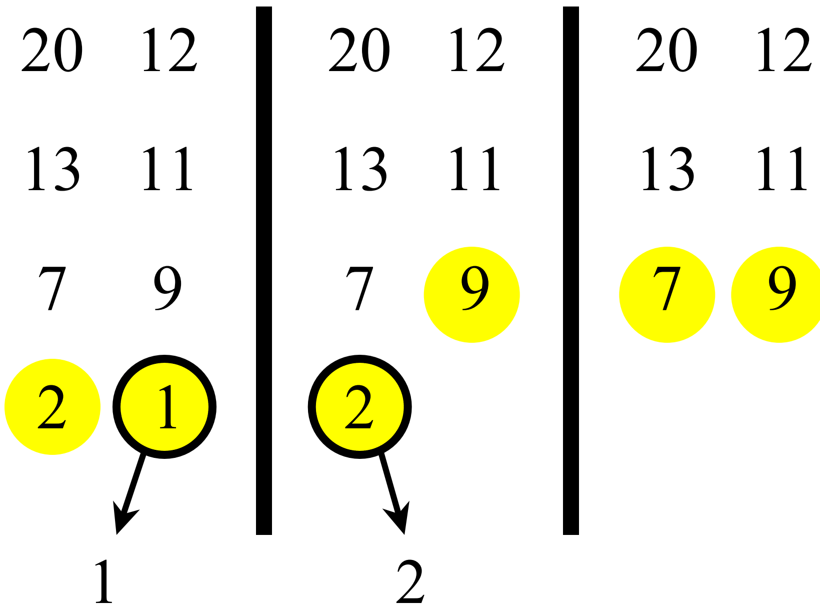
20	12
13	11
7	9
2	1

1

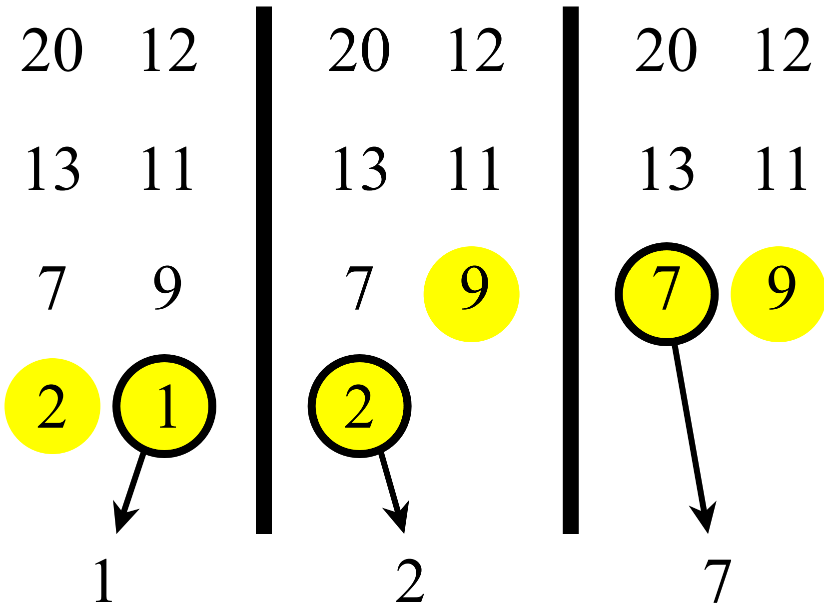
20	12
13	11
7	9
2	

2

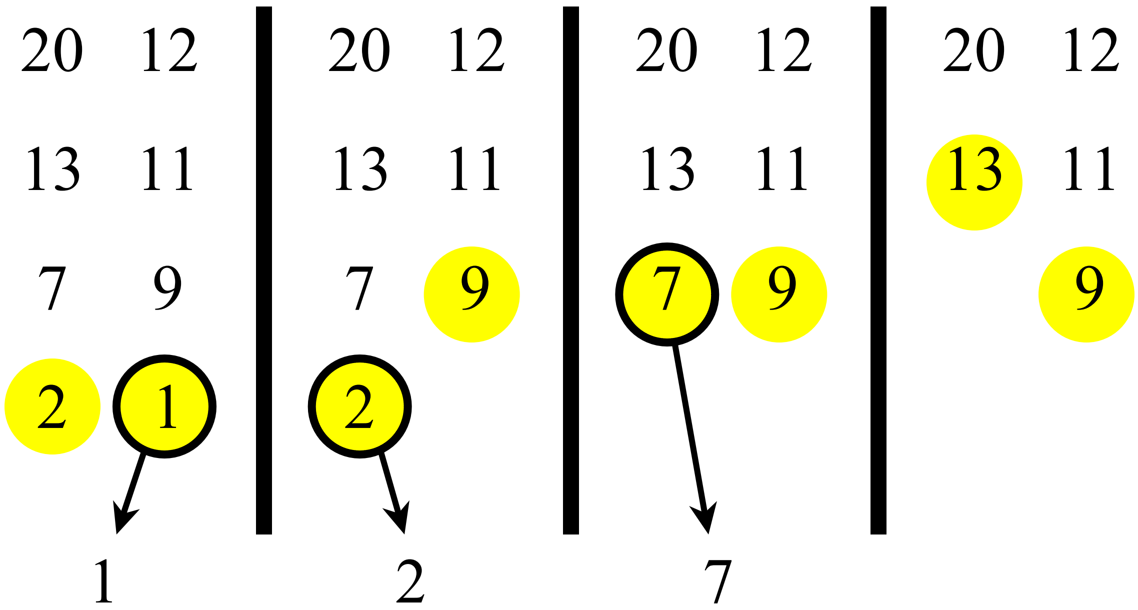
Merging two sorted arrays



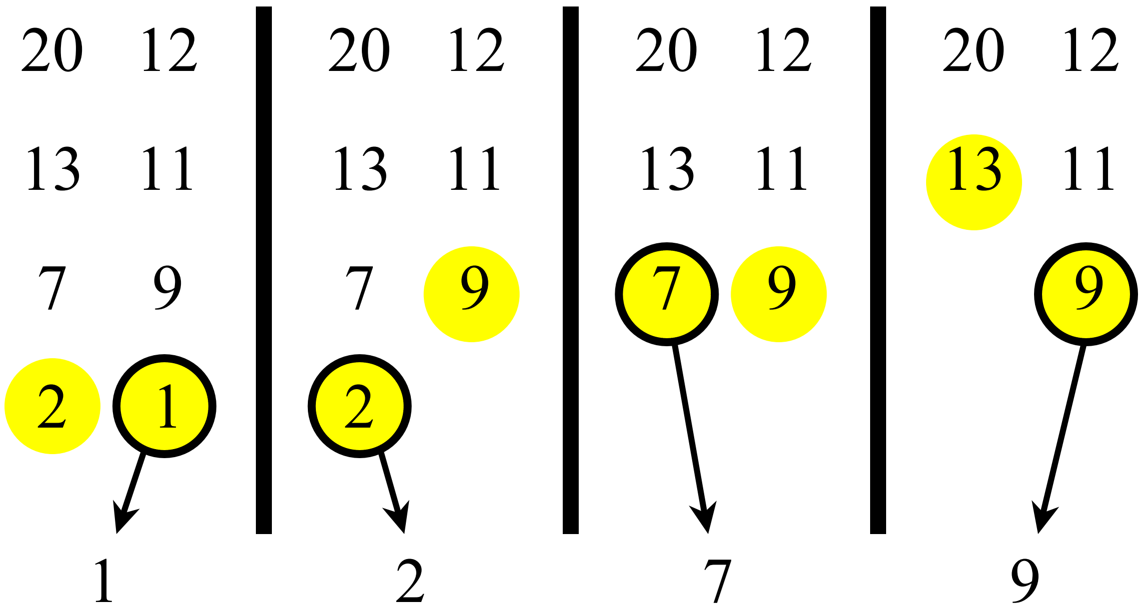
Merging two sorted arrays



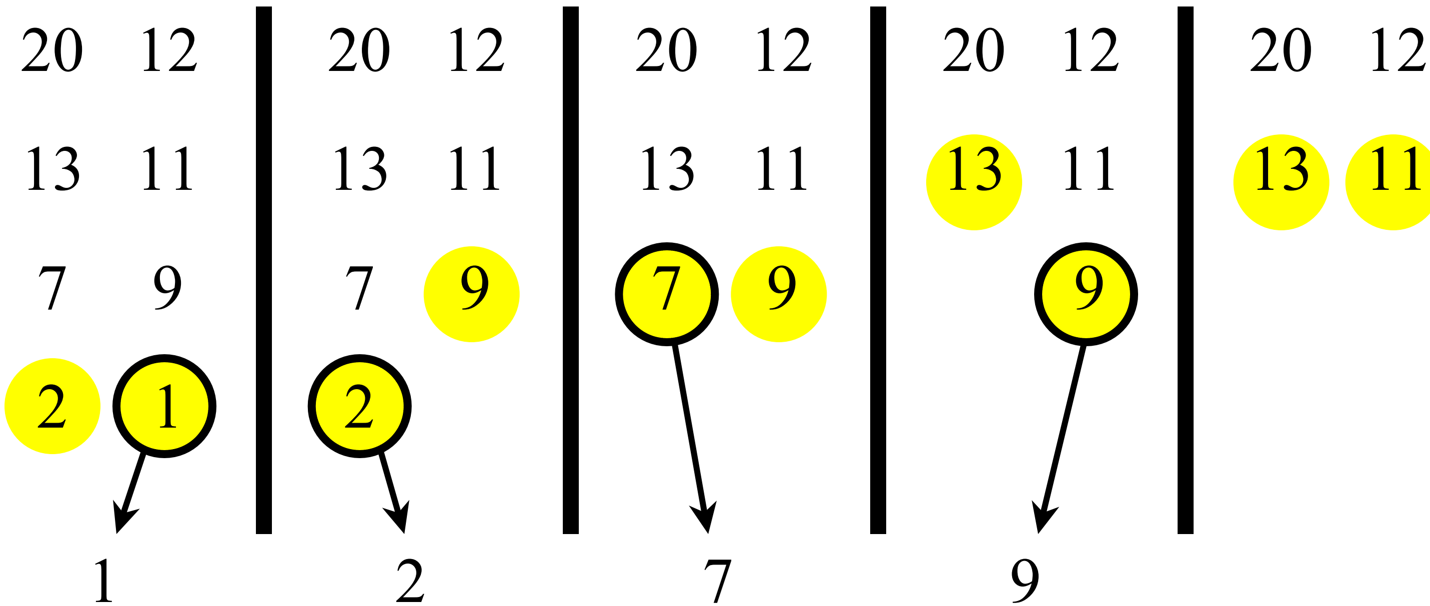
Merging two sorted arrays



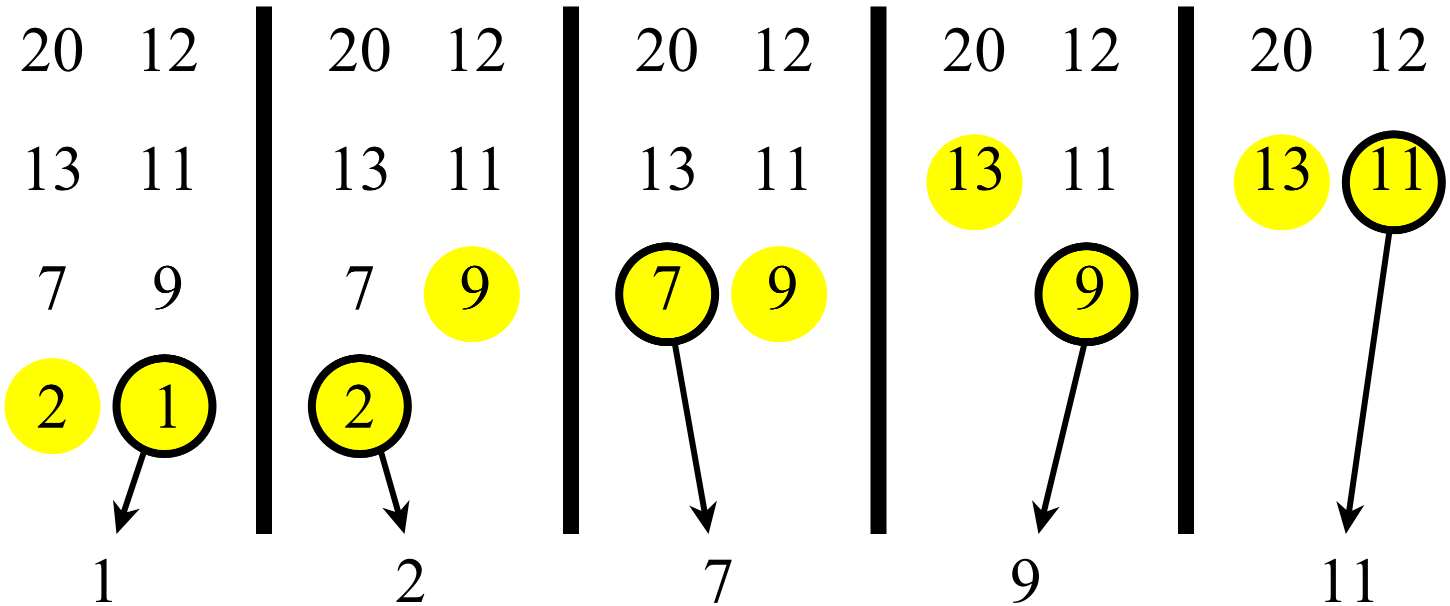
Merging two sorted arrays



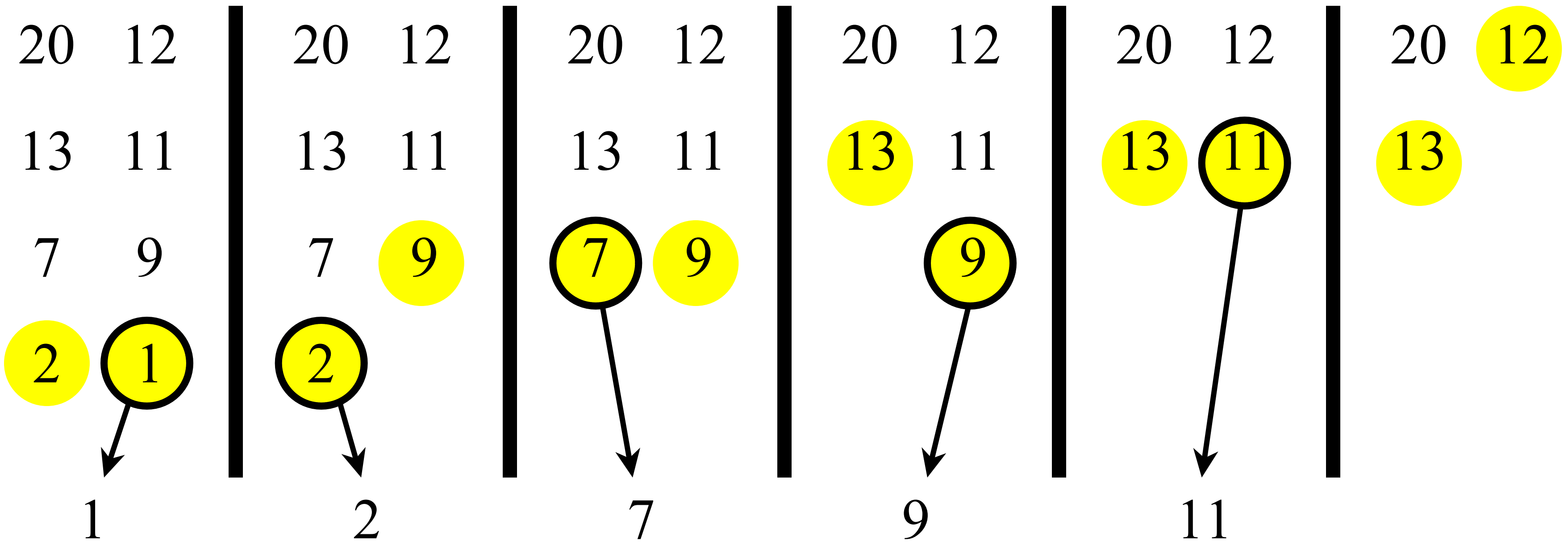
Merging two sorted arrays



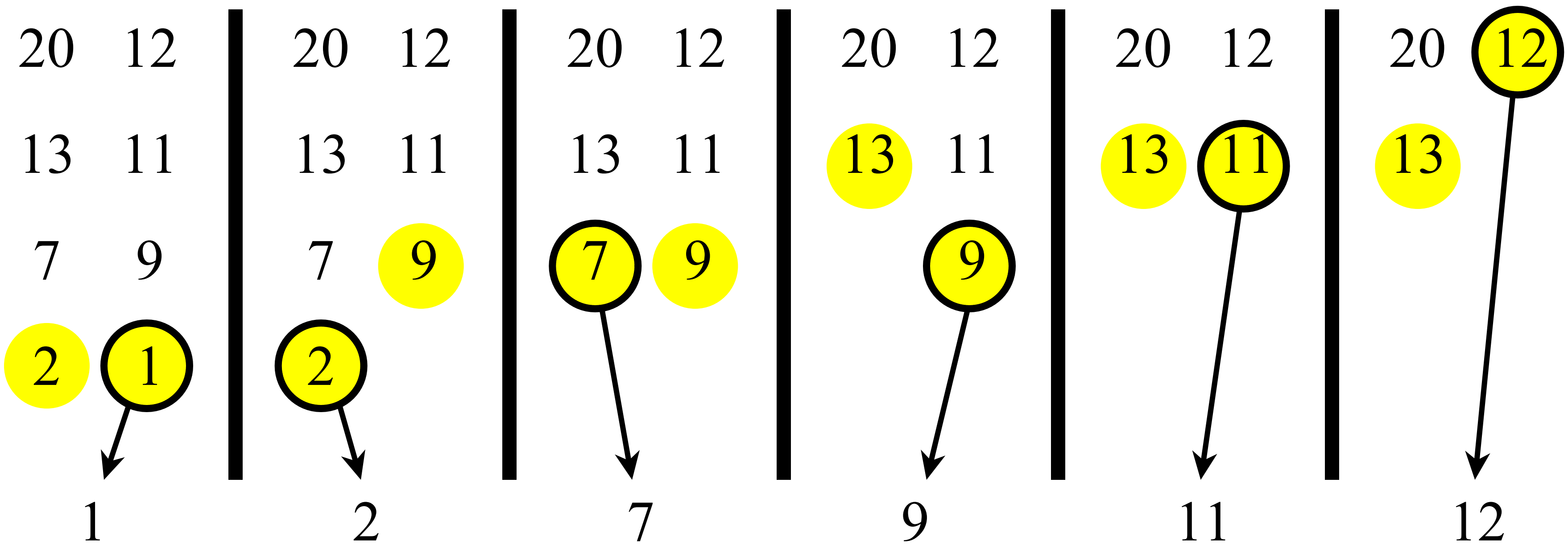
Merging two sorted arrays



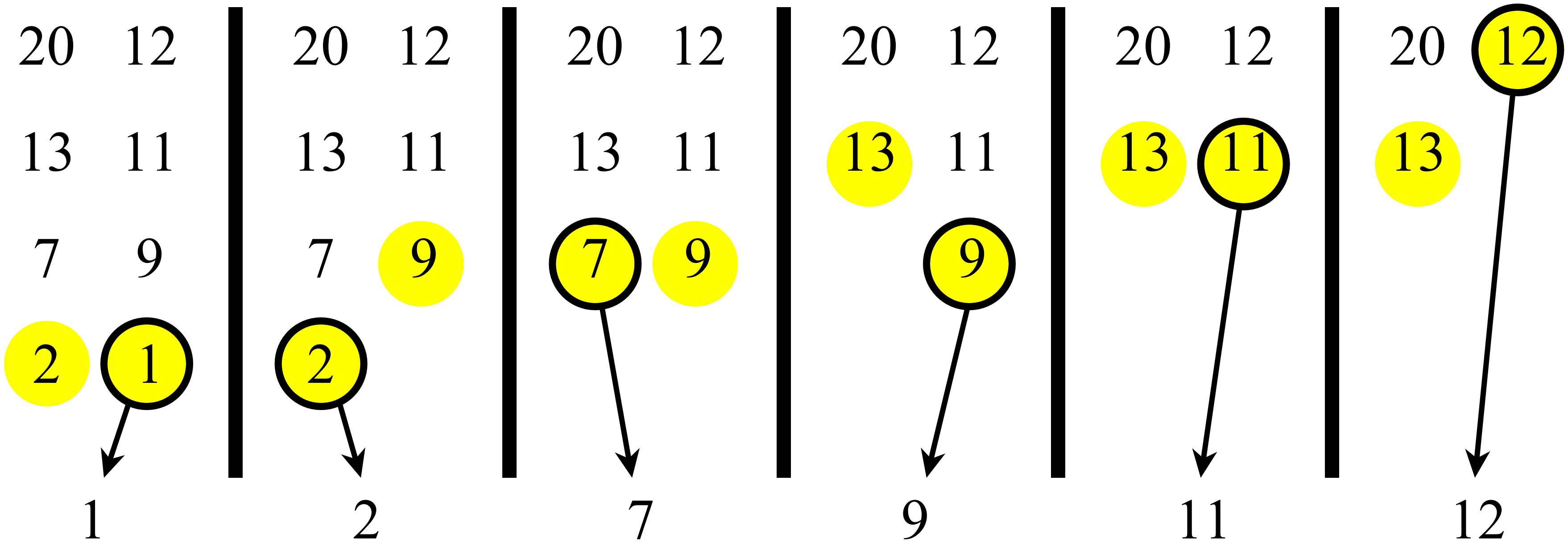
Merging two sorted arrays



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.	$\mathrm{Q}(1)$
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	$\mathrm{Q}(n)$

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

$$T(n) = ?$$

Recurrence solving

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

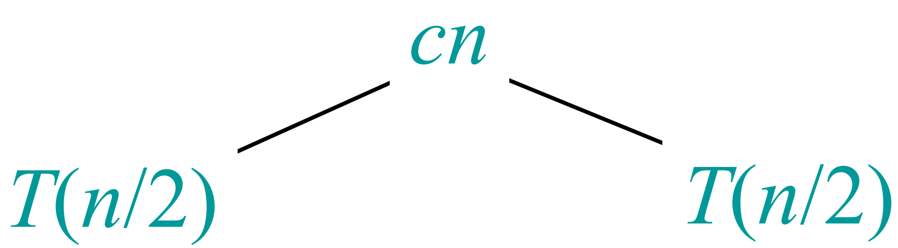
Recursion tree

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

$$T(n)$$

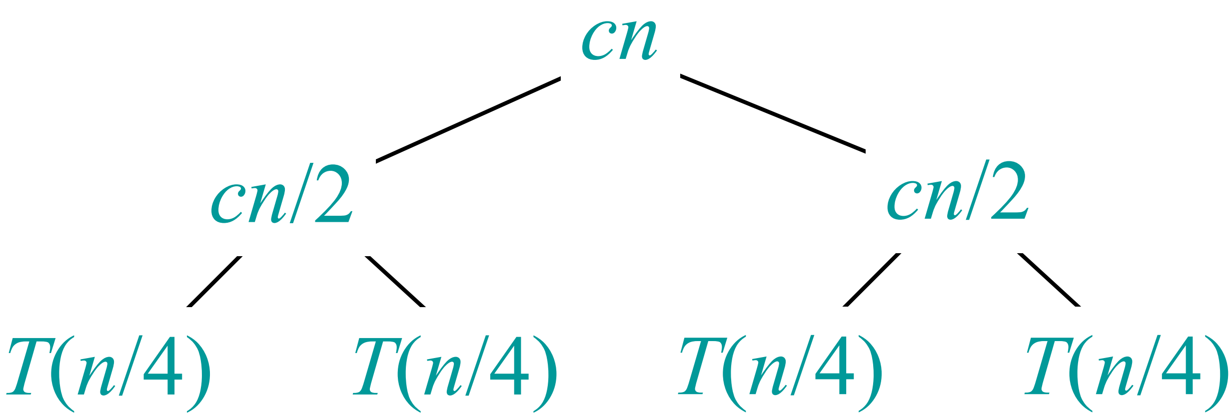
Recursion tree

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



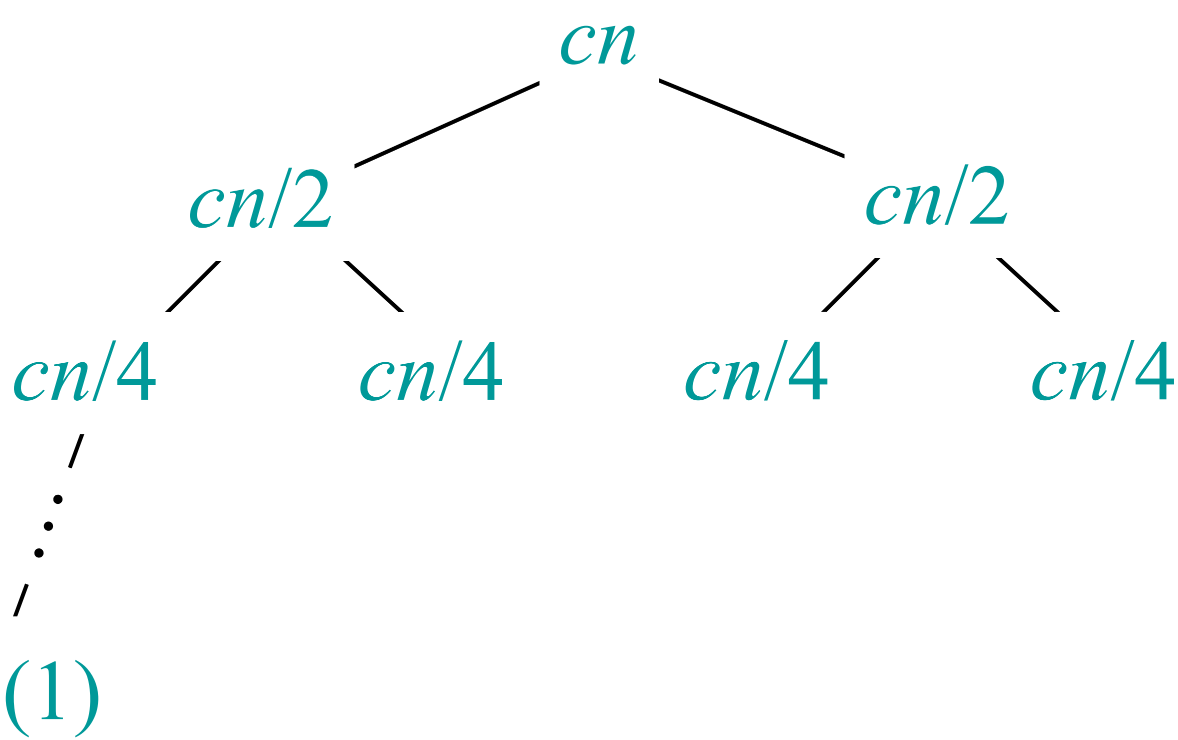
Recursion tree

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



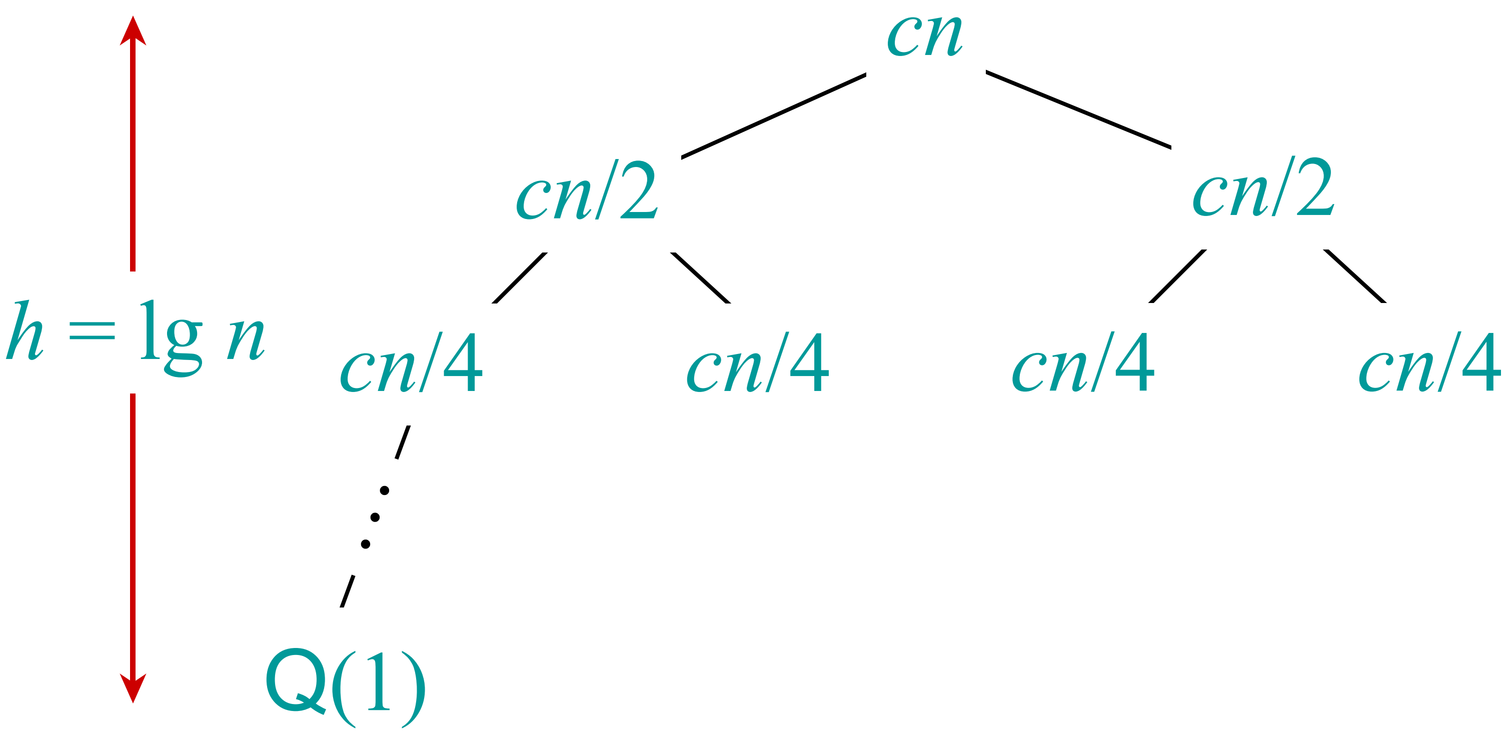
Recursion tree

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



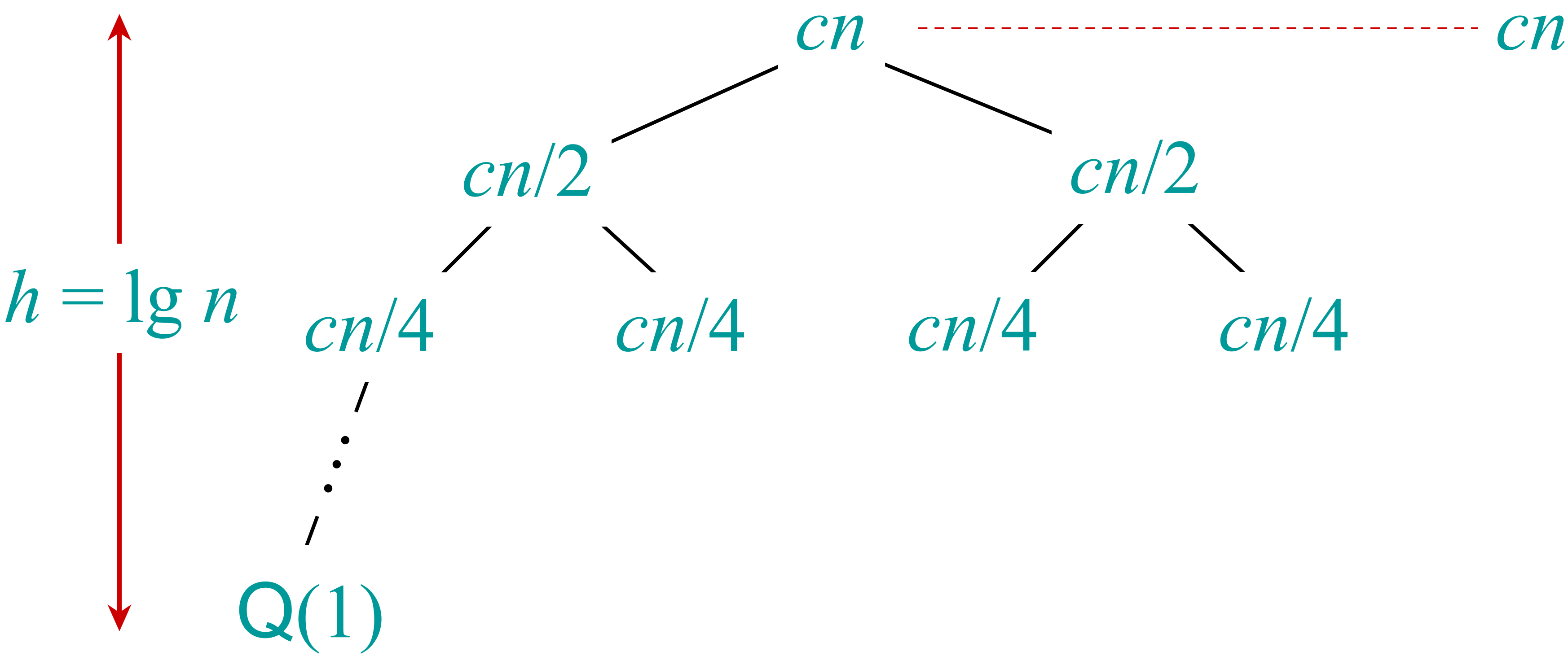
Recursion tree

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



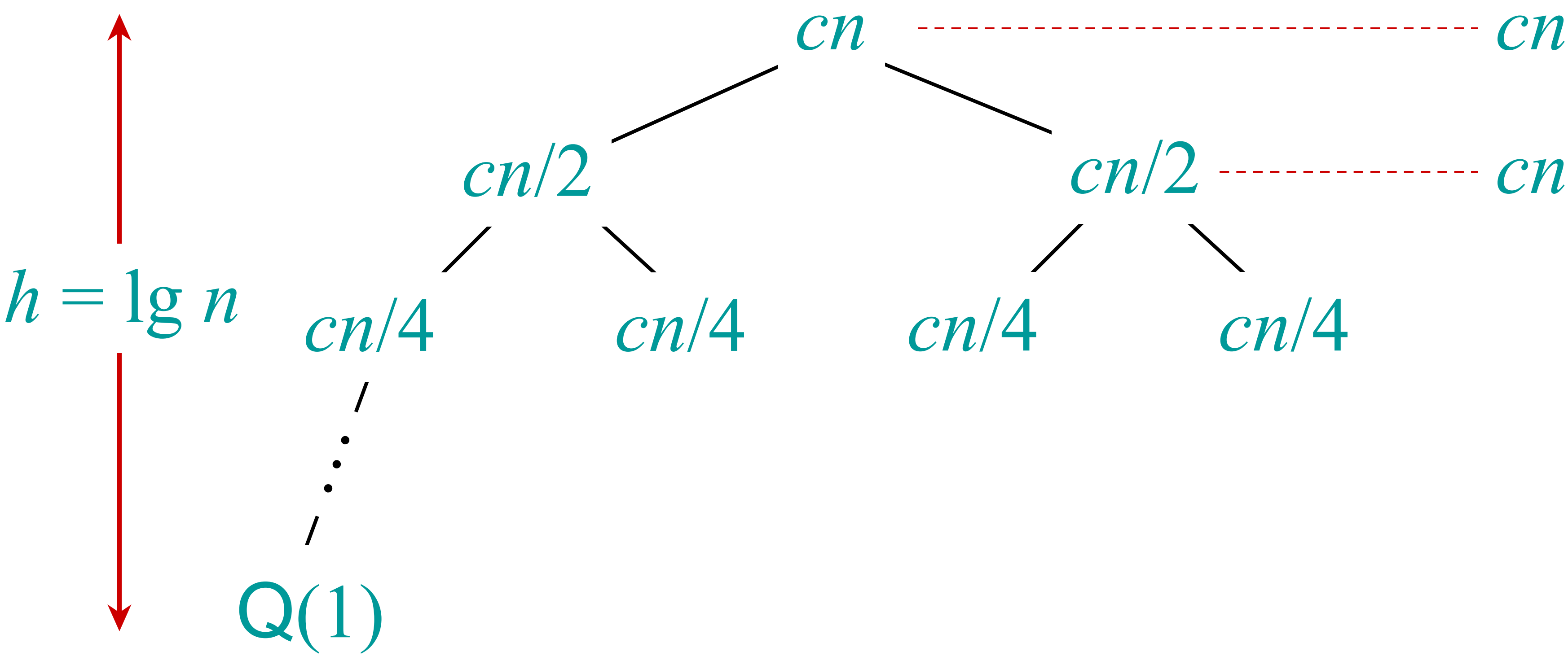
Recursion tree

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



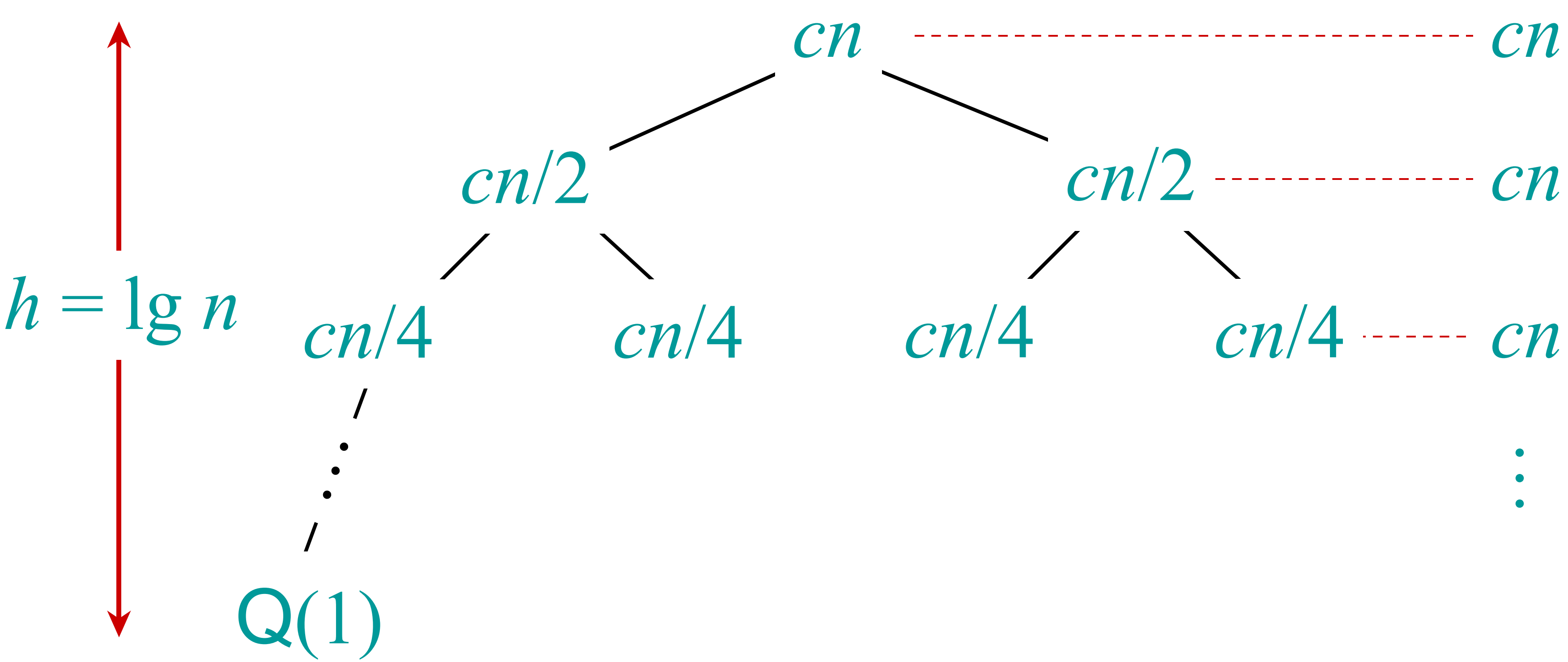
Recursion tree

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



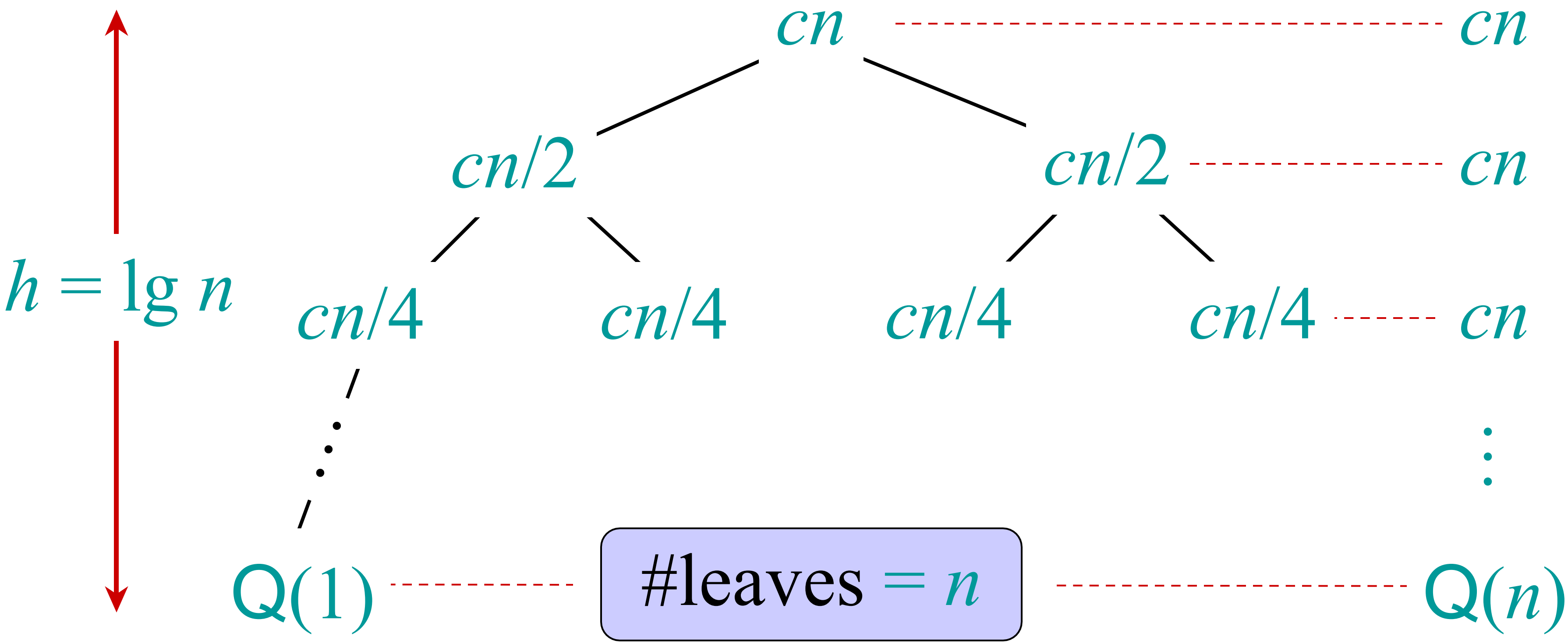
Recursion tree

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



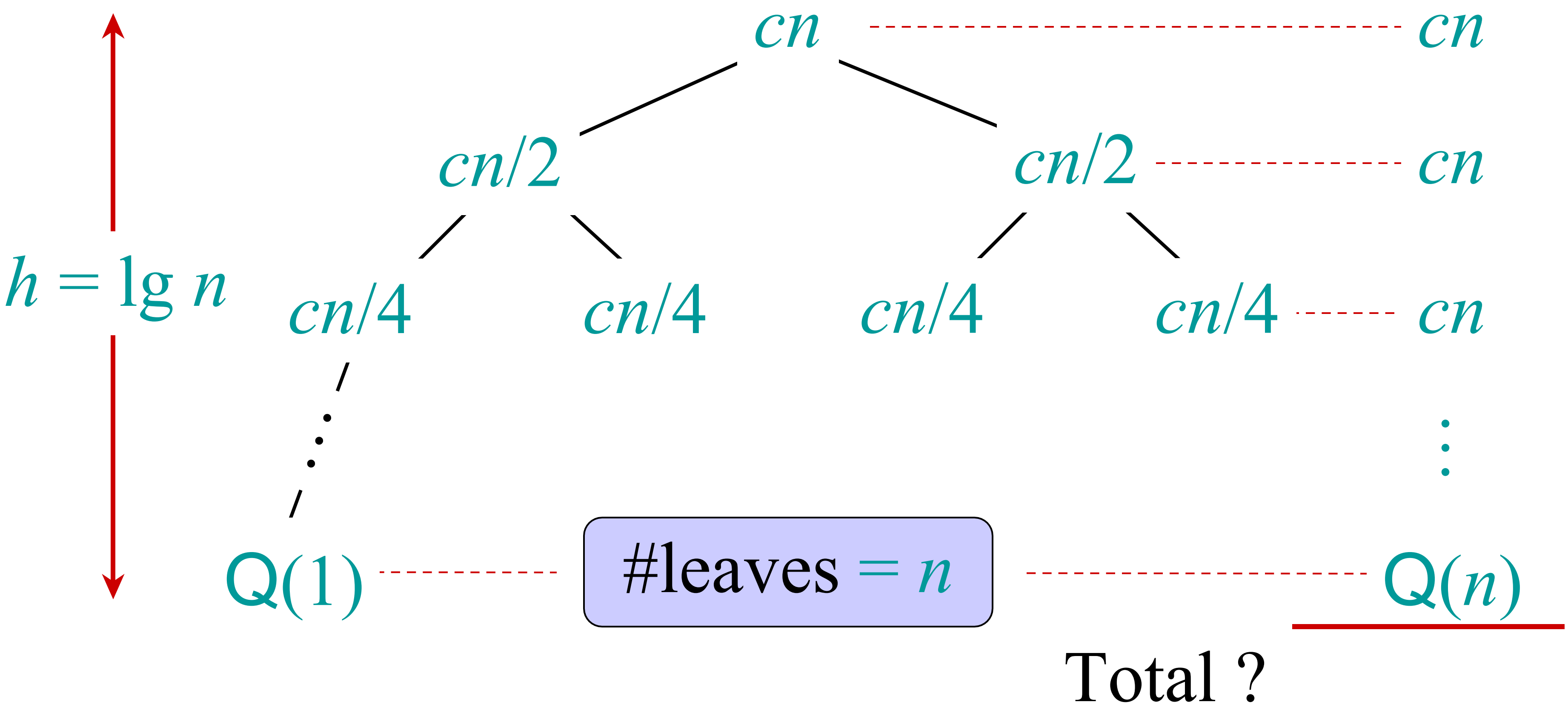
Recursion tree

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



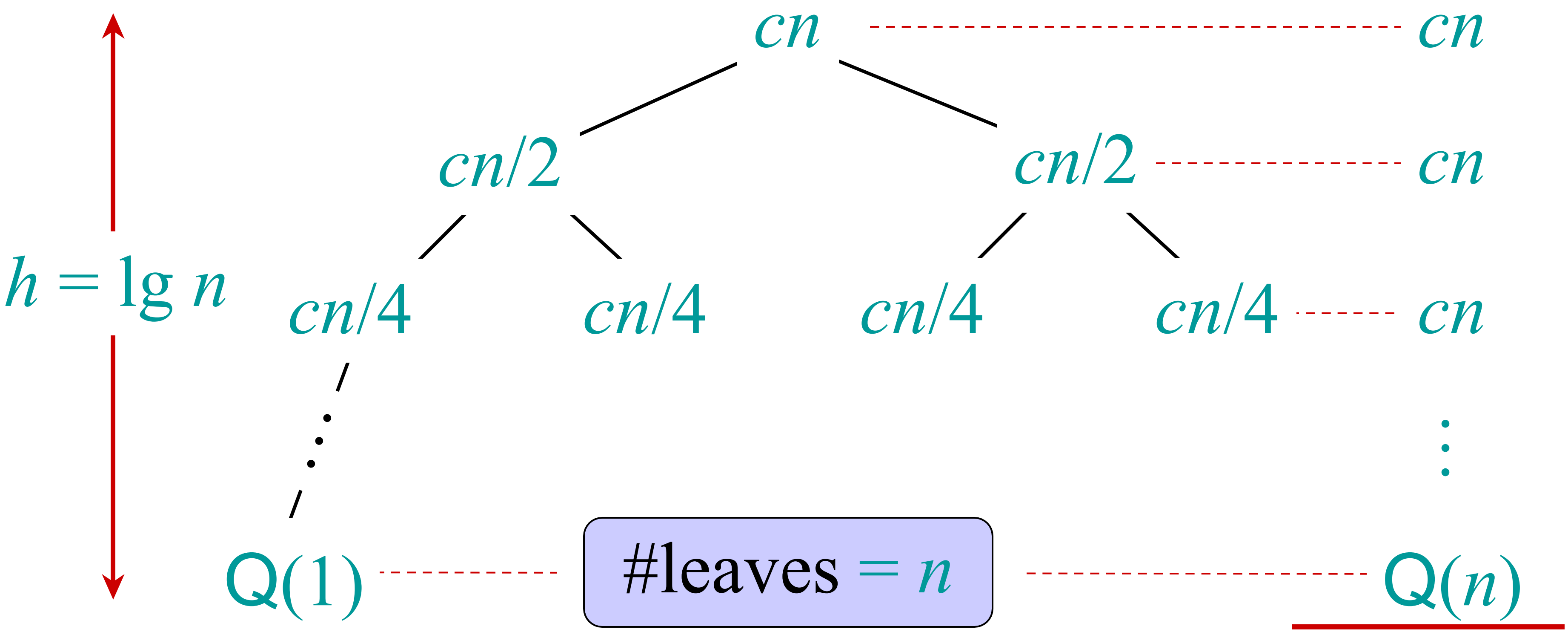
Recursion tree

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 = Q(n lg n)

The master method

"One theorem to rule them all" (sort of)

The master method applies to recurrences of the form

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

#subproblems

size of each subproblem

time to split into subproblems and combine results

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

e.g. Mergesort: a=2, b=2, $f(n)$=O(n)

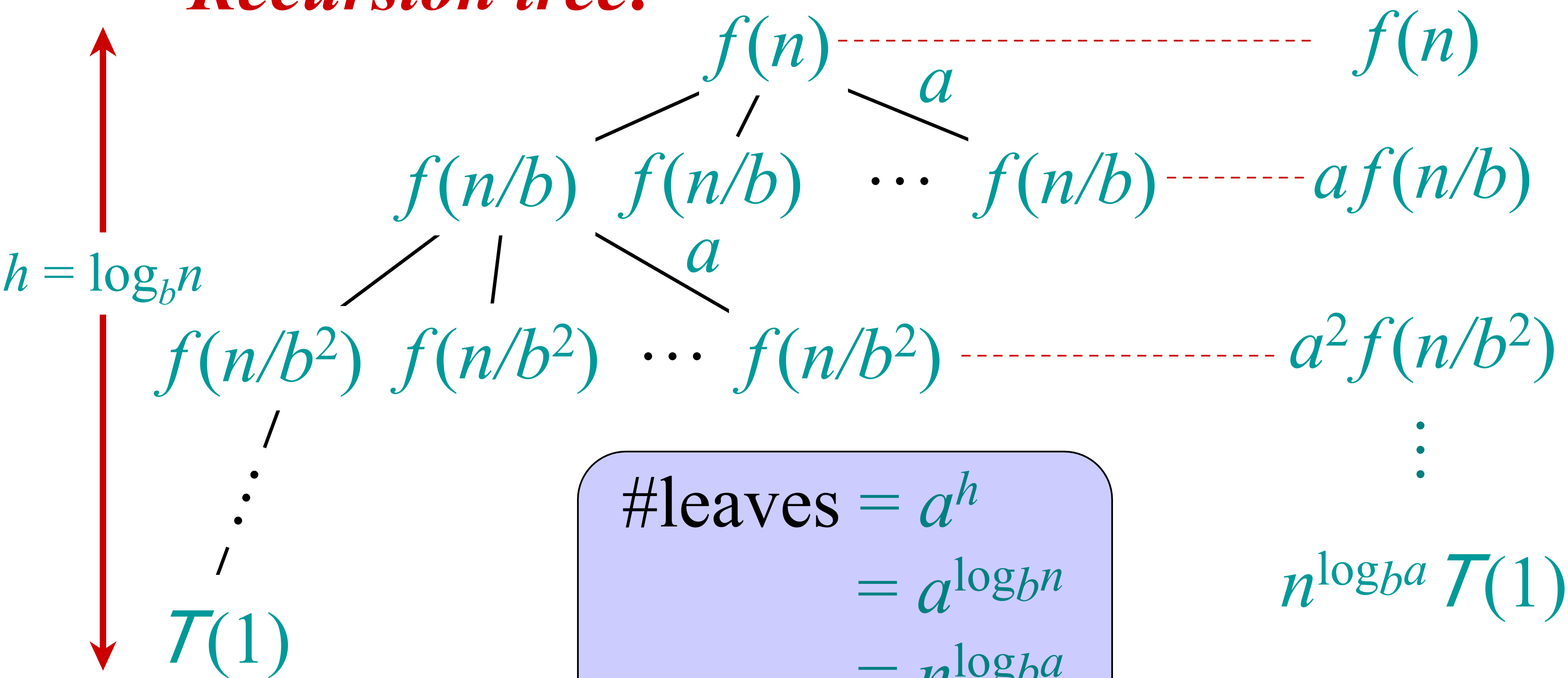
e.g.2 Binary Search: a=1, b=2, $f(n)$=O(1)

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:

$$f(n) \quad \cdots\cdots \quad f(n)$$

$$\overbrace{f(n/b) \quad f(n/b) \quad \cdots \quad f(n/b)}^{a} \quad \cdots\cdots \quad a\,f(n/b)$$

$$\overbrace{f(n/b^2) \quad f(n/b^2) \quad \cdots \quad f(n/b^2)}^{a} \quad \cdots\cdots \quad a^2 f(n/b^2)$$

$$\vdots$$

$$h = \log_b n$$

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

CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

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

Idea of master theorem

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

Recursion tree:

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

a

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

a

$h = \log_b n$

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

$\vdots$

$T(1)$

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

CASE 2: ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

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

Idea of master theorem

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

Recursion tree:

$f(n)$ — a — $f(n)$

$f(n/b) \quad f(n/b) \quad \cdots \quad f(n/b)$ — $a\,f(n/b)$

a

$h = \log_b n$

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

$\vdots$

$T(1)$ — $n^{\log_b a}\,T(1)$

CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the 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 $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

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

   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. $f(n) = \Theta(n^{\log_b a} \log^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

(cost of level i) = $a^i f(n/b^i) = \Theta(n^{\log_b a} \cdot \log^k (n/b^i))$
so all levels have about the same cost

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

Three common cases (cont.)

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

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

Examples

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

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

Case 1: $f(n) = O(n^{1-\varepsilon})$ for $\varepsilon = 1$.

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

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

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

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

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

Examples

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

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

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

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