Design and Analysis of Algorithms

CSE 5311

Lecture 2 Algorithms and Growth Functions

Junzhou Huang, Ph.D.

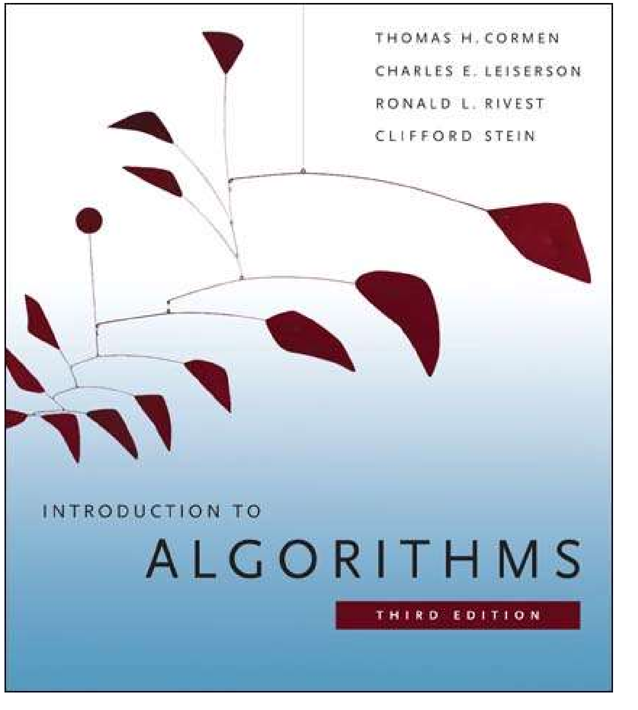
Department of Computer Science and Engineering

Administration

- **Course CSE5311**
  - What: Design and Analysis of Algorithms
  - When: Friday 1:00 ~ 3:50pm
  - Where: ERB 130
  - Who: Junzhou Huang (Office ERB 650) jzhuang@uta.edu
  - Office Hour: Friday 3:50 ~ 5:50pm and/or appointments
  - Homepage: http://ranger.uta.edu/~huang/teaching/CSE5311.htm
    (**You're required to check this page regularly**)
- **Lecturer**
  - PhD in CS from Rutgers, the State University of New Jersey
  - Research areas: machine learning, computer vision, medical image analysis and bioinformatics
- **GTA**
  - Saiyang Na (Office ERB 403), sxn3892@mavs.uta.edu
  - Office hours: Friday 10:00am ~ 12:00pm and/or appointments

Reviewing: Study Materials

- **Prerequisites**
  - Algorithms and Data Structure (CSE 2320)
  - Theoretical Computer Science (CSE 3315)
  - What this really means:
    - You have working experience s on software development.
    - You know compilation process and programming
    - Elementary knowledge of math and algorithms
- **Text book**
  - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, **Introduction to Algorithms**, third edition
  - https://mitpress.mit.edu/books/introduction-algorithms



Reviewing: What?

- **The theoretical study of design and analysis of computer algorithms**
- **Basic goals for an algorithm**
  - Always correct
  - Always terminates
- **Our class: performance**
  - Performance often draws the line between what is possible and what is impossible.
- **Design and Analysis of Algorithms**
  - **Analysis:** predict the cost of an algorithm in terms of resources and performance
  - **Design:** design algorithms which minimize the cost

Reviewing: 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*

Reviewing: Running Time

- **Running Time**
  - Depends on the input
  - An already sorted sequence is easier to sort.
- **Major Simplifying Convention**
  - Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
  - $T_A(n)$ = time of A on length n inputs. Generally, we seek upper bounds on the running time, to have a guarantee of performance.
- **Kinds of Analyses**
  - **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. Need assumption of statistical distribution of inputs.
  - **Best-case:** (Never) Cheat with a slow algorithm that works fast on some input.

Machine-independent Time

- **Question**
  - Machine-independent Time
- **Idea**
  - Ignore machine dependent constants, otherwise impossible to verify and to compare algorithms
  - Look at growth of T(n) as $n \to \infty$.

"Asymptotic Analysis"

Recall: Θ-notation

Definition:

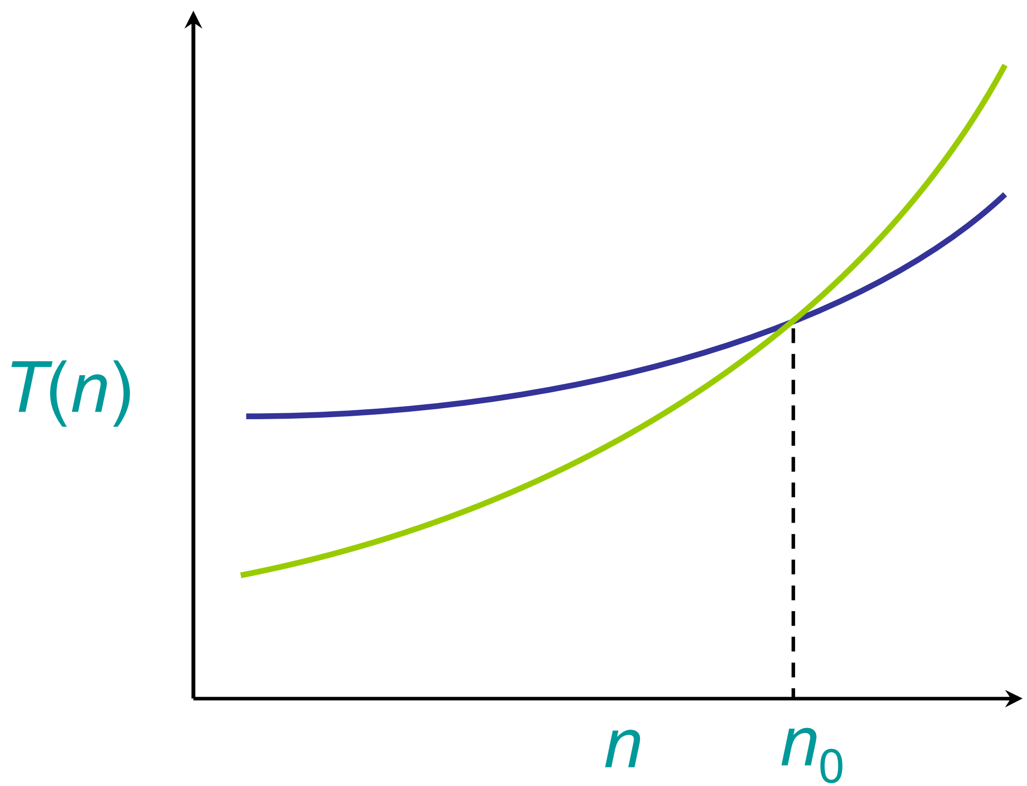
$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$ $\}$

Basic Manipulations:

- 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.



.

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

Insertion Sort Analysis

Worst case: Input reverse sorted.

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

[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.

Integer Multiplication

- **Let X = $\boxed{A \mid B}$ and Y = $\boxed{C \mid D}$ where A,B,C and D are n/2 bit integers**
- **Simple Method:**

$$XY = (2^{n/2}A+B)(2^{n/2}C+D) = 2^n AC + 2^{n/2}(AD+BC) + BD$$

- **Running Time Recurrence**

$$T(n) < 4T(n/2) + \Theta(n)$$

Recursive Calls **Addition and Shift**

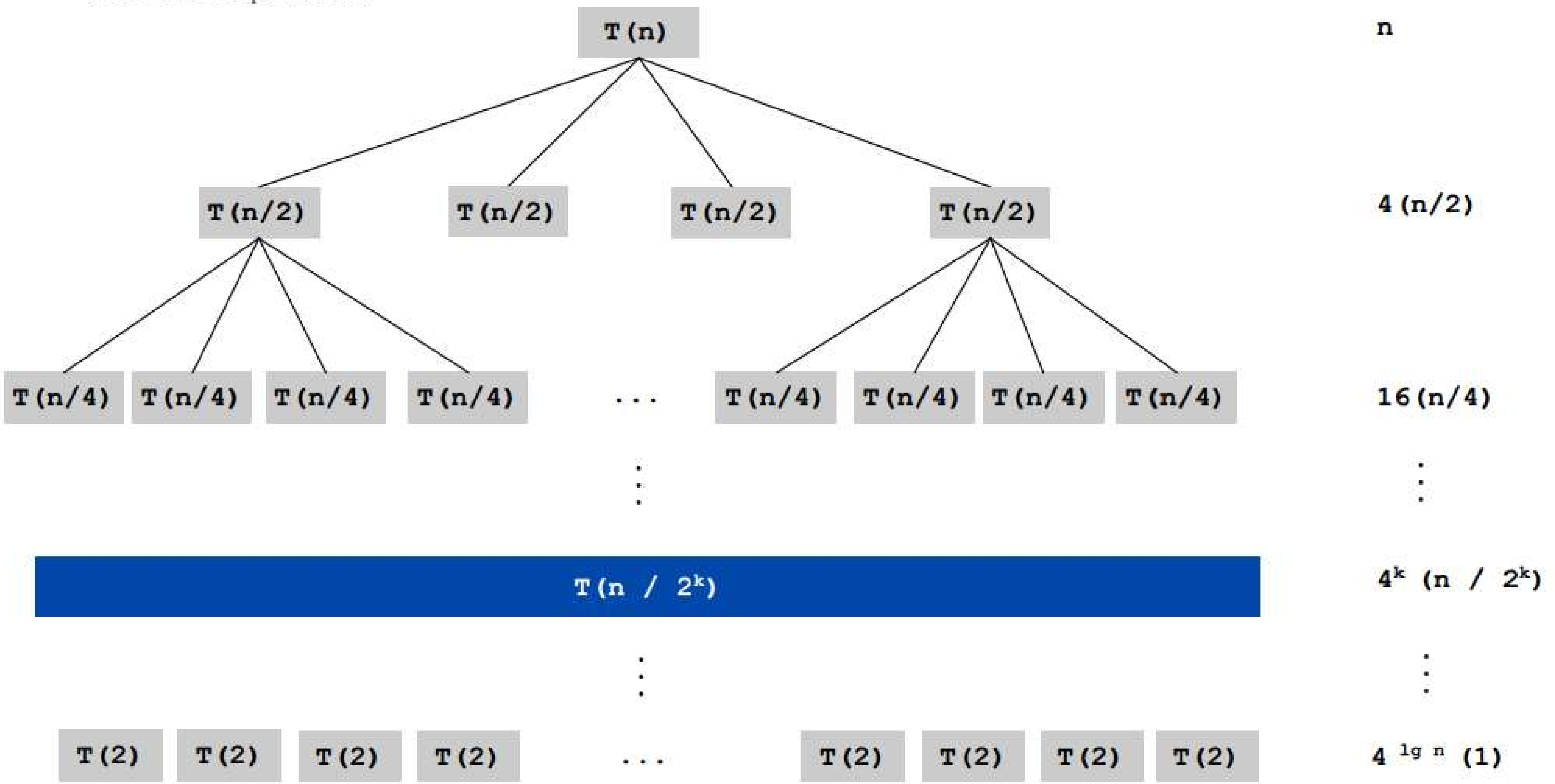
- **Solution $T(n) = \Theta(n^2)$**

Integer Multiplication

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 4T(n/2) + n & \text{otherwise} \end{cases}$$

assume n is a power of 2

$$T(n) = \sum_{k=0}^{\lg n} n\, 2^k = n\left(\frac{2^{1+\lg n} - 1}{2-1}\right) = 2n^2 - n$$



Better Integer Multiplication

- Let X = | A | B | and Y = | C | D | where A,B,C and D are n/2 bit integers
- [Karatsuba-Ofman 1962] : Can multiply two n-bit integers in $O(n^{\log 3})$ bit operations.

$$XY = (2^{n/2}A+B)(2^{n/2}C+D) = 2^{n}AC+2^{n/2}(AD+BC)+BD$$

$$= (2^{n} - 2^{n/2})AC+2^{n/2}(A+B)(C+D) + (1 - 2^{n/2})BD$$

- **Running Time Recurrence**

$$T(n) < 3T(n/2) + \Theta(n)$$

$$T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1+\lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \Rightarrow T(n) = O(n^{\lg 3}) = O(n^{1.585})$$

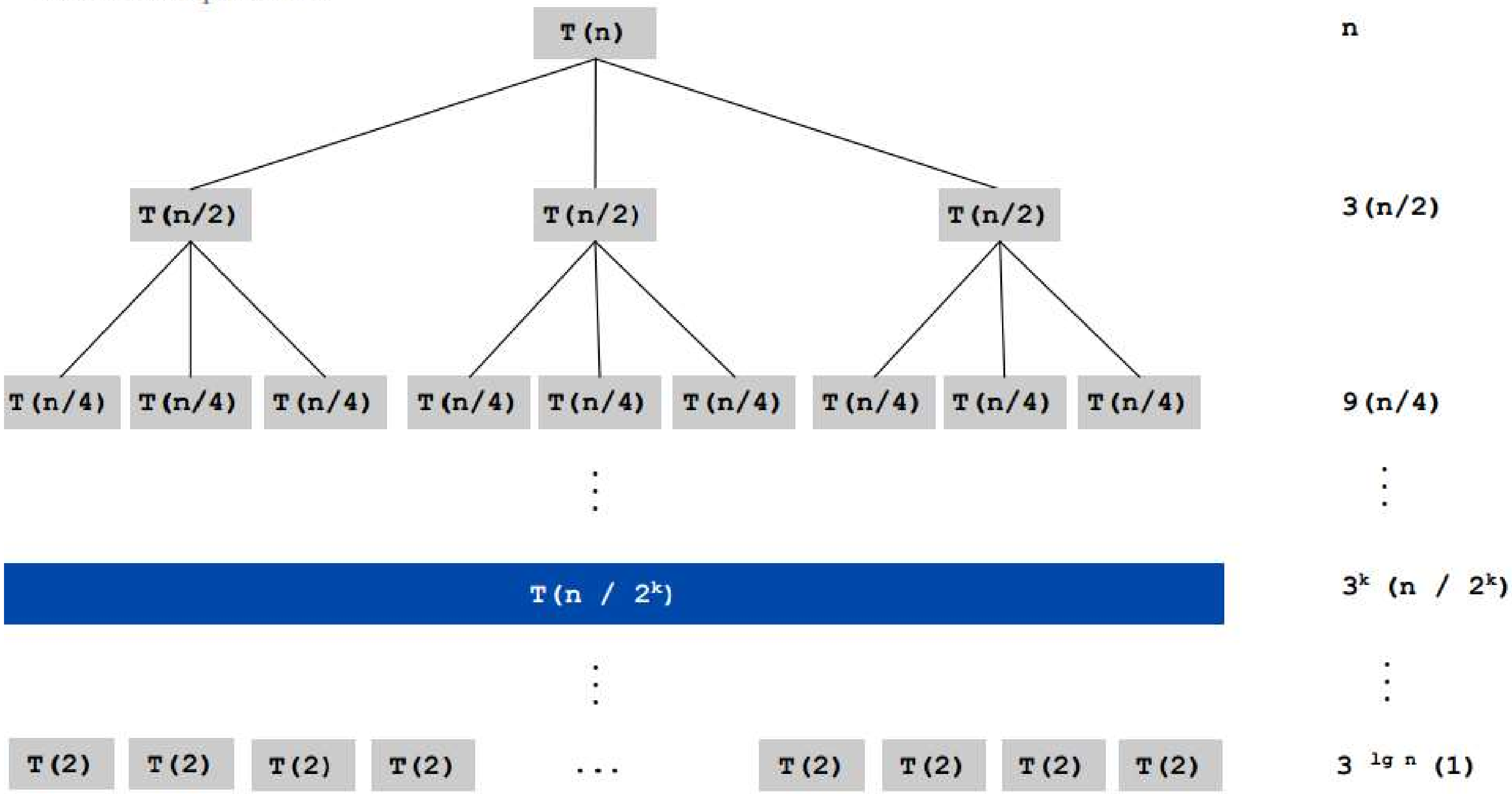
- **Solution: $T(n) = O(n^{\log 3})$**

Better Integer Multiplication

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

assume n is a power of 2

$$T(n) = \sum_{k=0}^{\lg n} n \left(\tfrac{3}{2}\right)^k = n\left(\frac{\left(\frac{3}{2}\right)^{1+\lg n} - 1}{\frac{3}{2} - 1}\right) = 3n^{\lg 3} - 2n$$



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.**

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

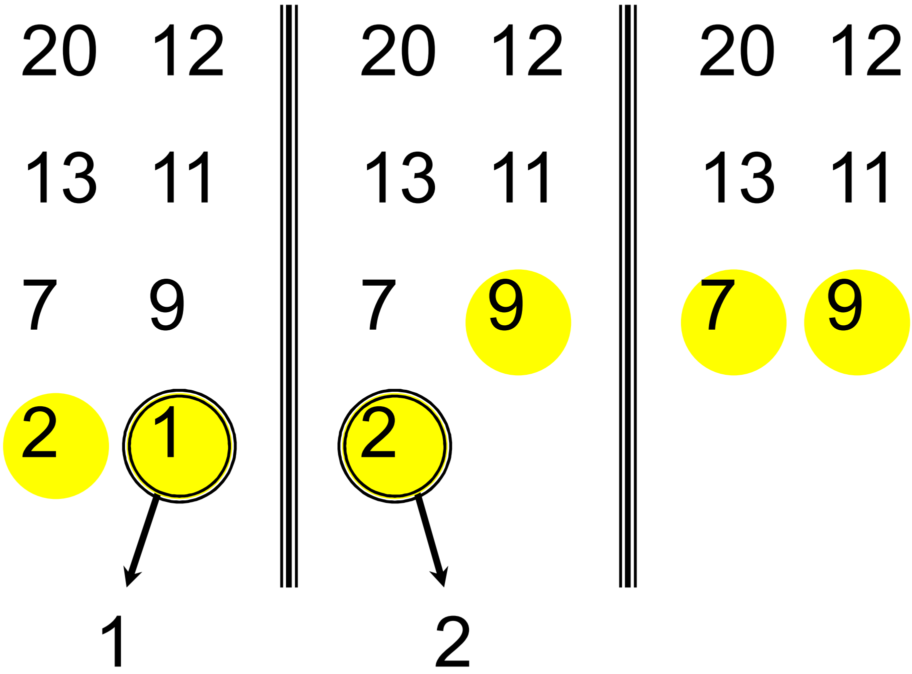
Merging Two Sorted Arrays

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

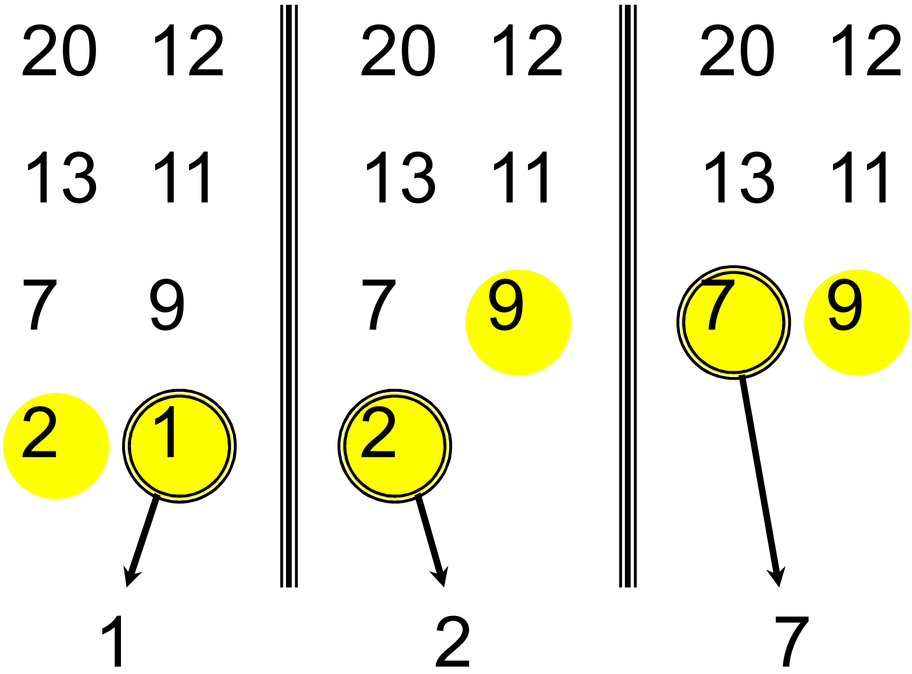
Merging Two Sorted Arrays

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

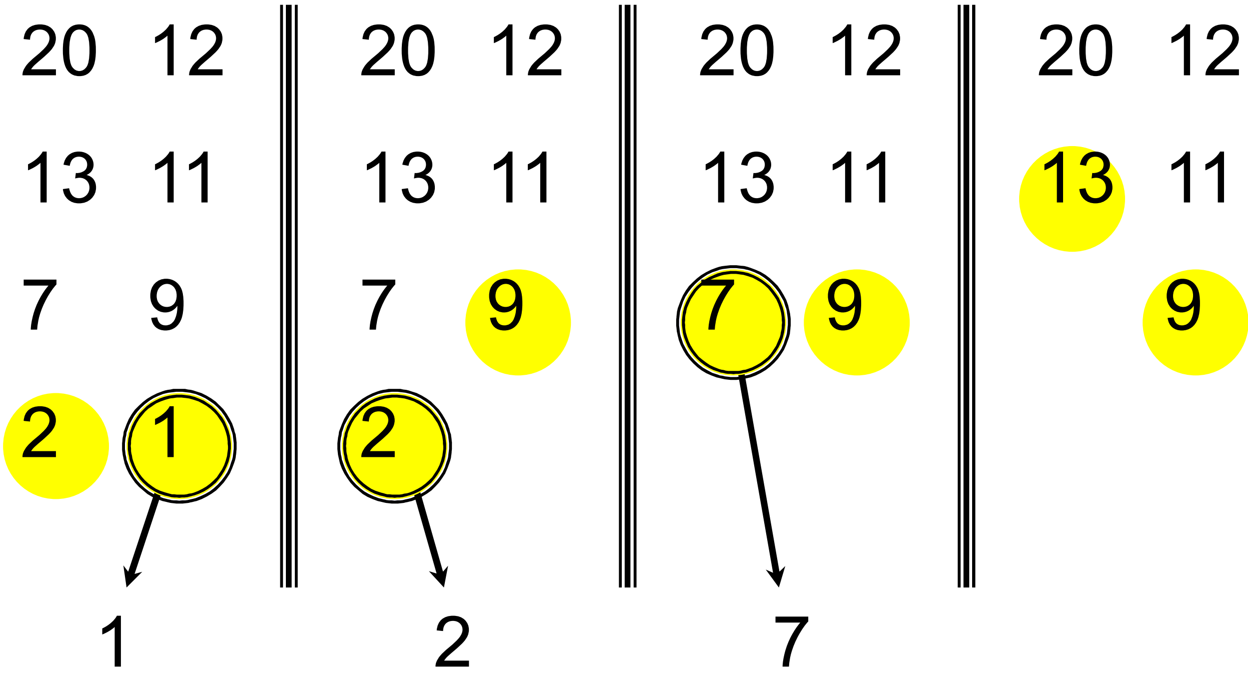
Merging Two Sorted Arrays



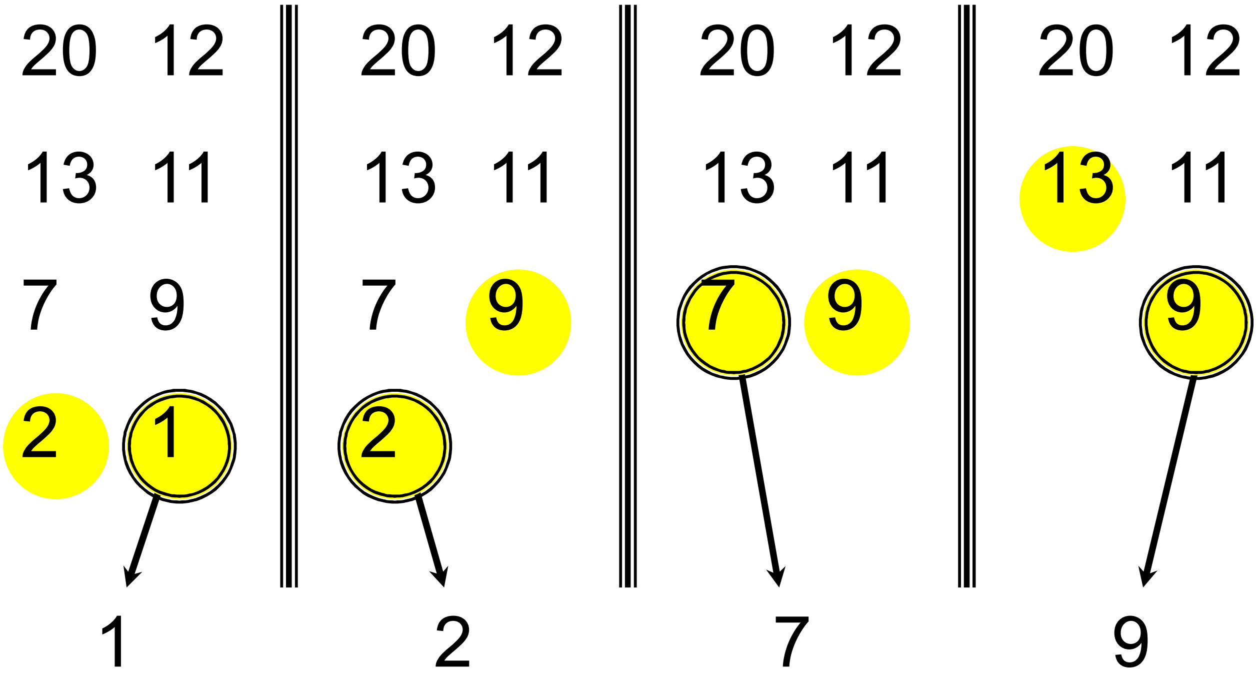
Merging Two Sorted Arrays



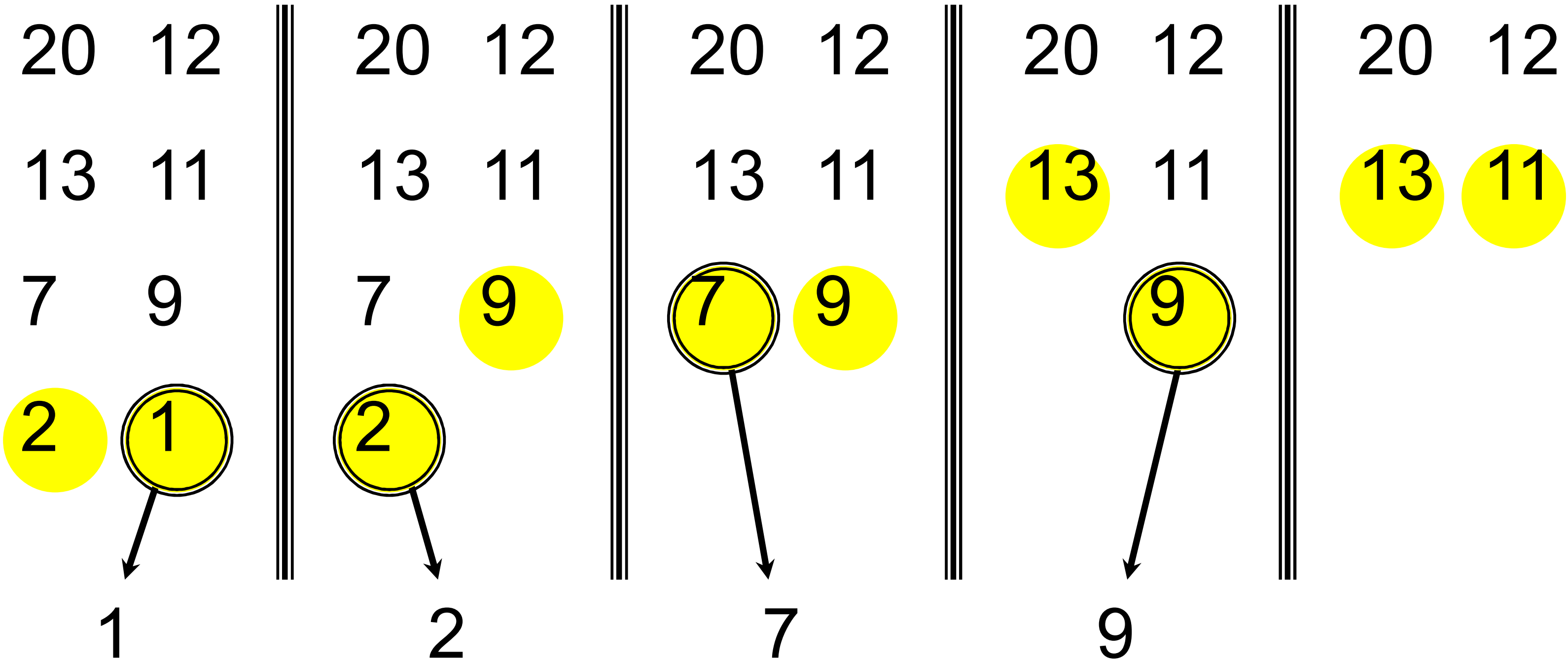
Merging Two Sorted Arrays



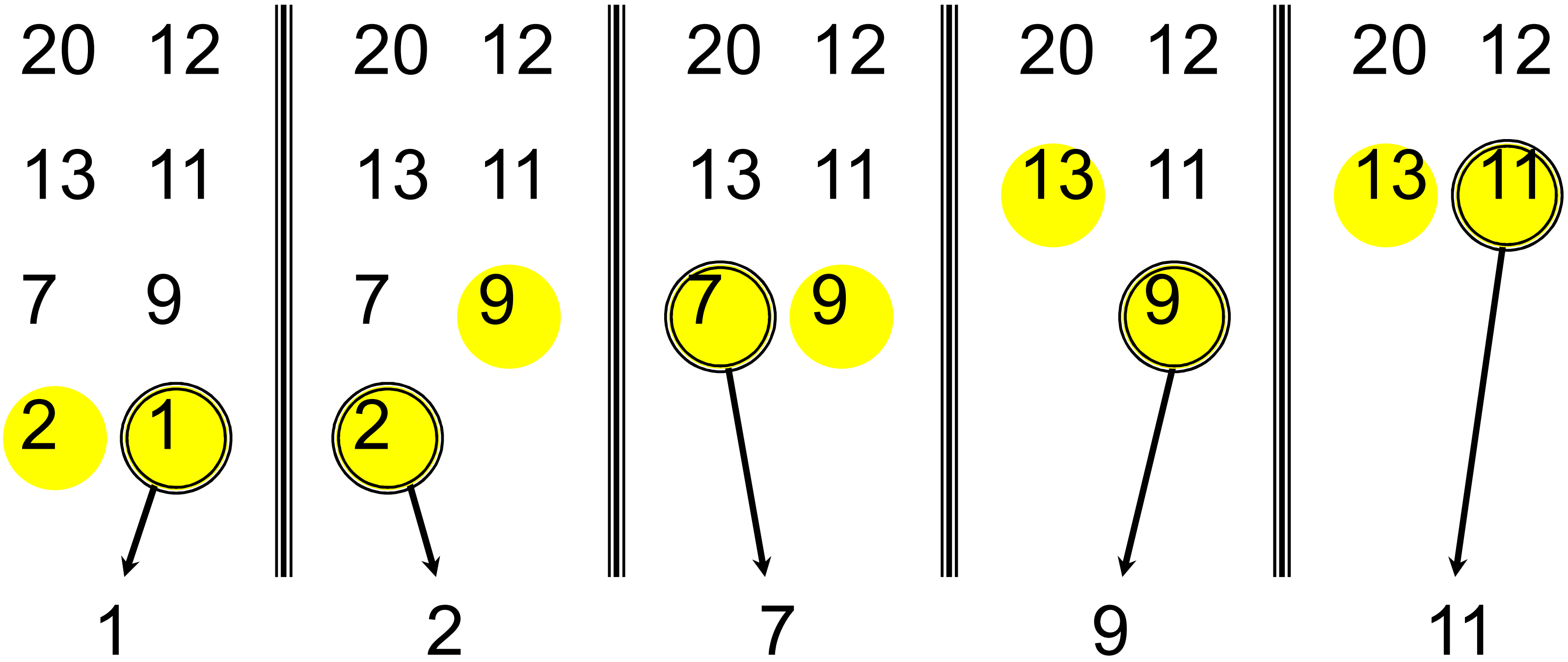
Merging Two Sorted Arrays



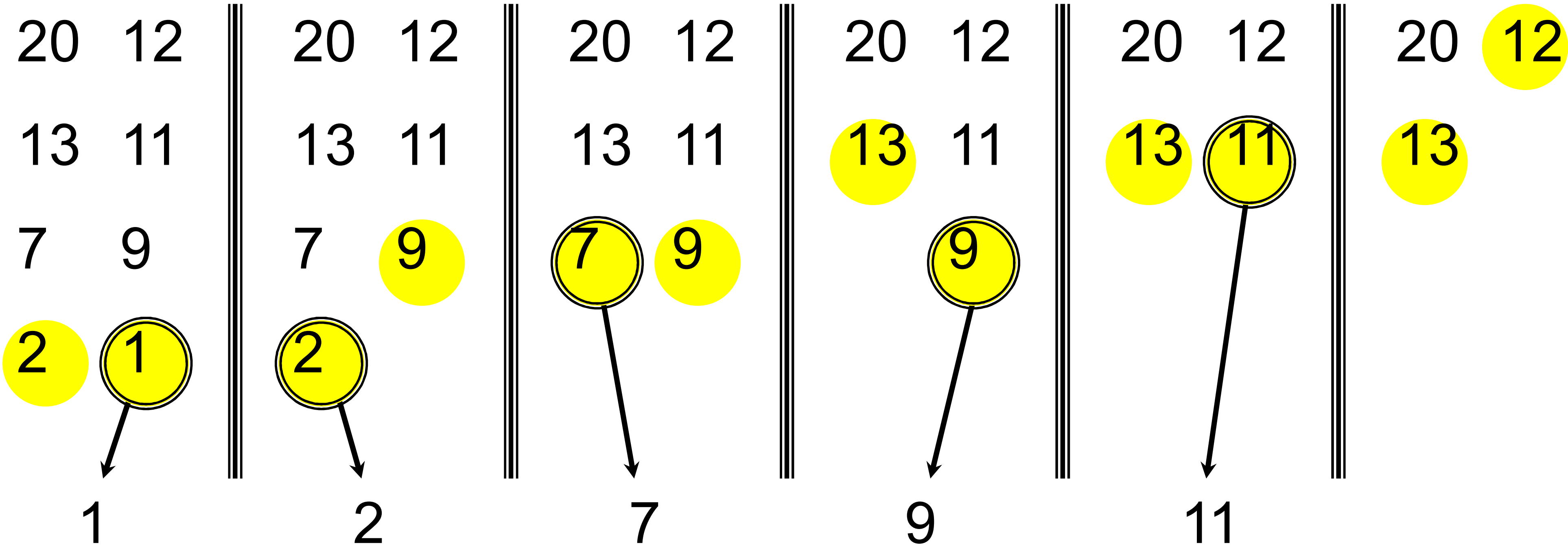
Merging Two Sorted Arrays



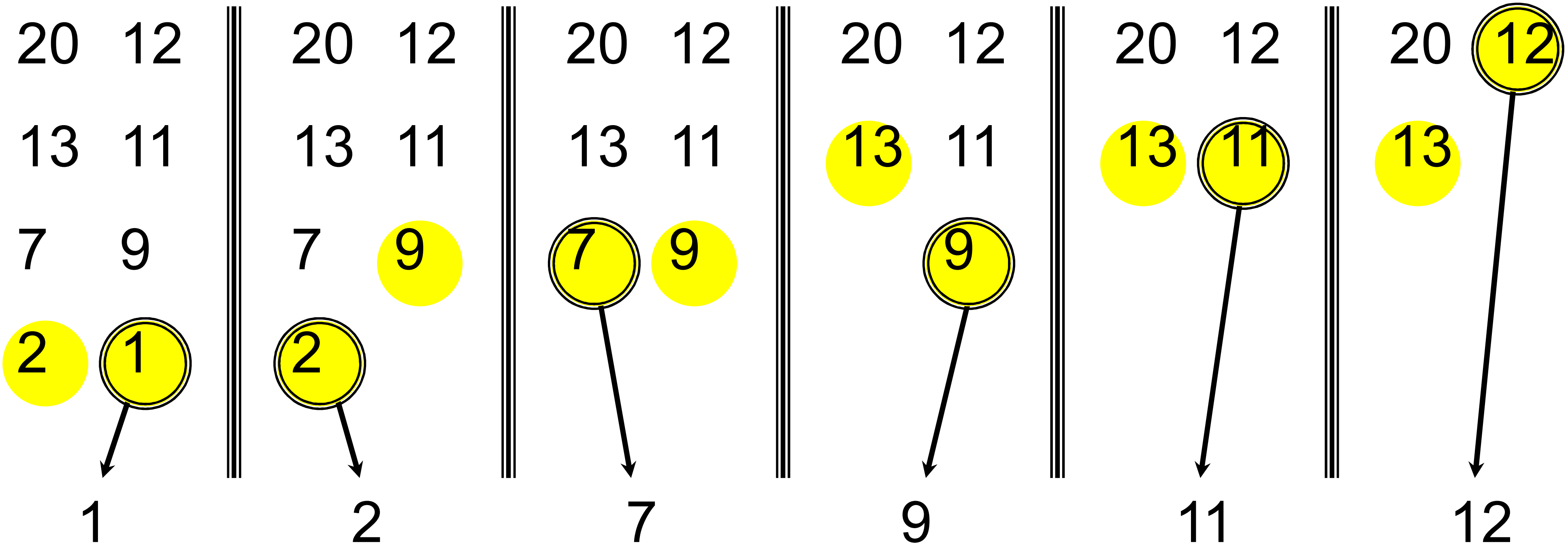
Merging Two Sorted Arrays



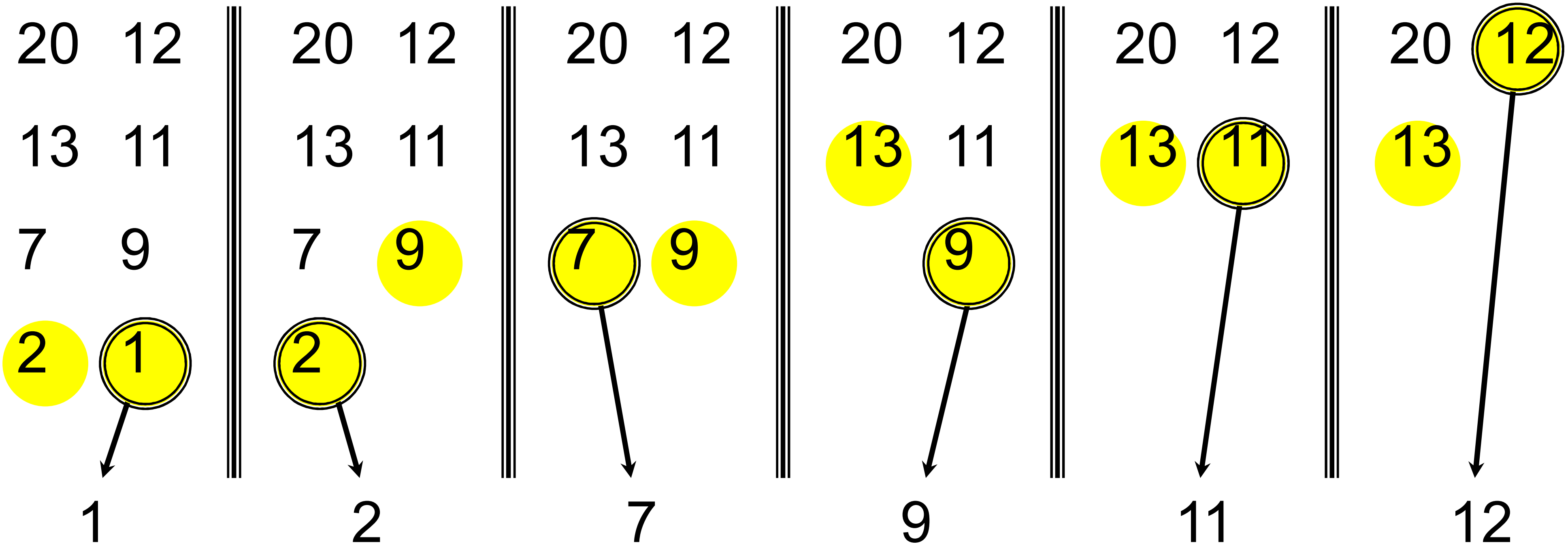
Merging Two Sorted Arrays



Merging Two Sorted Arrays



Merging Two Sorted Arrays



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

Analyzing Merge Sort

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

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.**
- **Next Lecture will provide several ways to find a good upper bound on $T(n)$.**

Recursion Tree

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

$$cn$$

$$T(n/2) \qquad\qquad T(n/2)$$

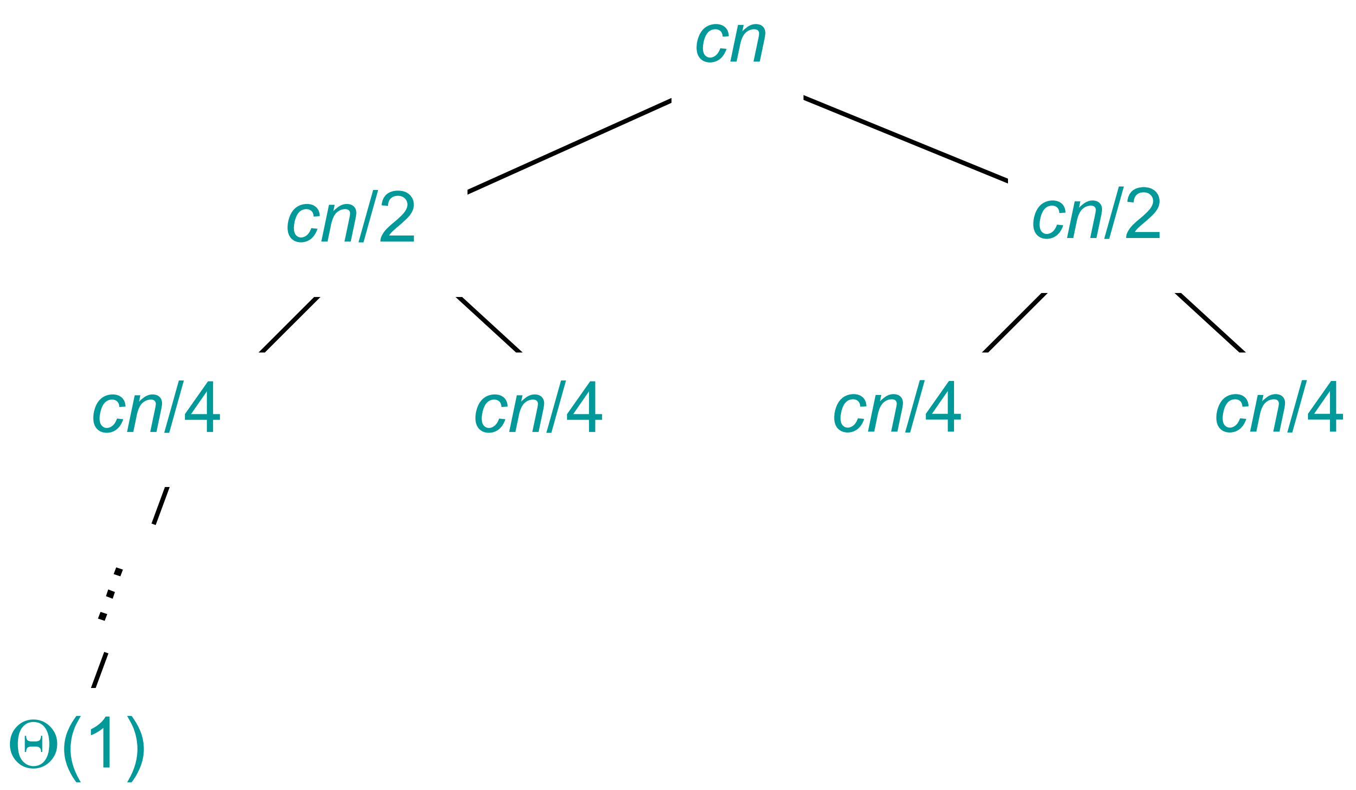
Recursion Tree

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

$$
\begin{array}{ccccccc}
 & & & cn & & & \\
 & cn/2 & & & & cn/2 & \\
T(n/4) & & T(n/4) & & T(n/4) & & T(n/4)
\end{array}
$$

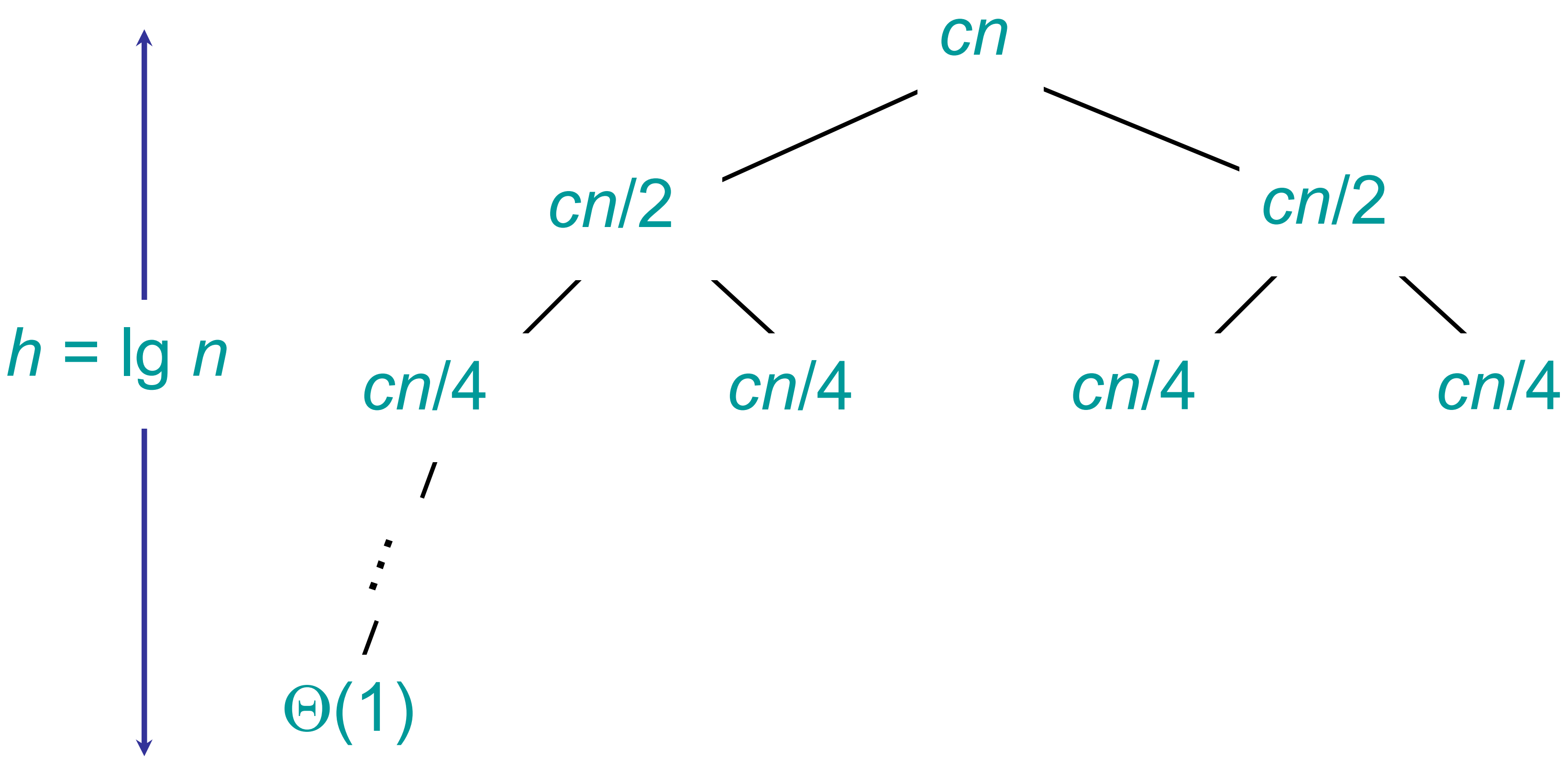
Recursion Tree

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



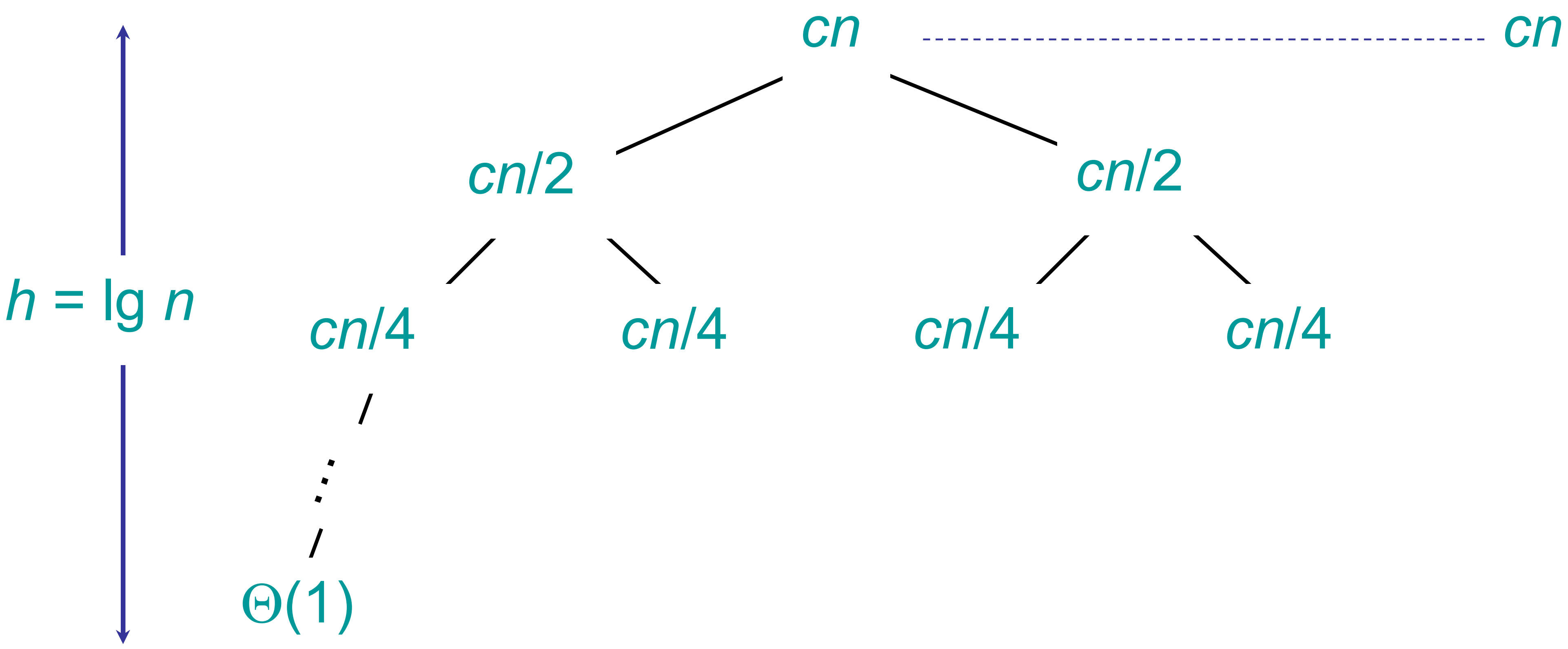
Recursion Tree

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



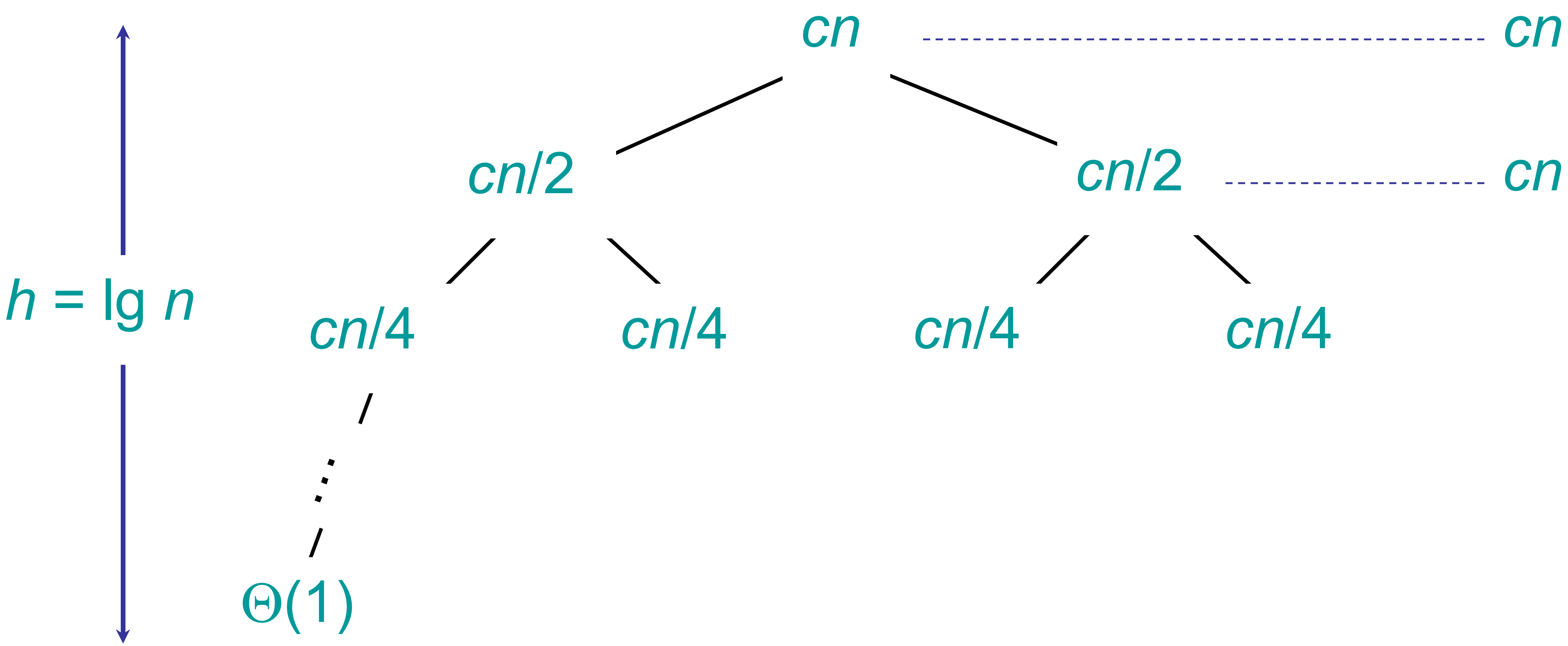
Recursion Tree

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



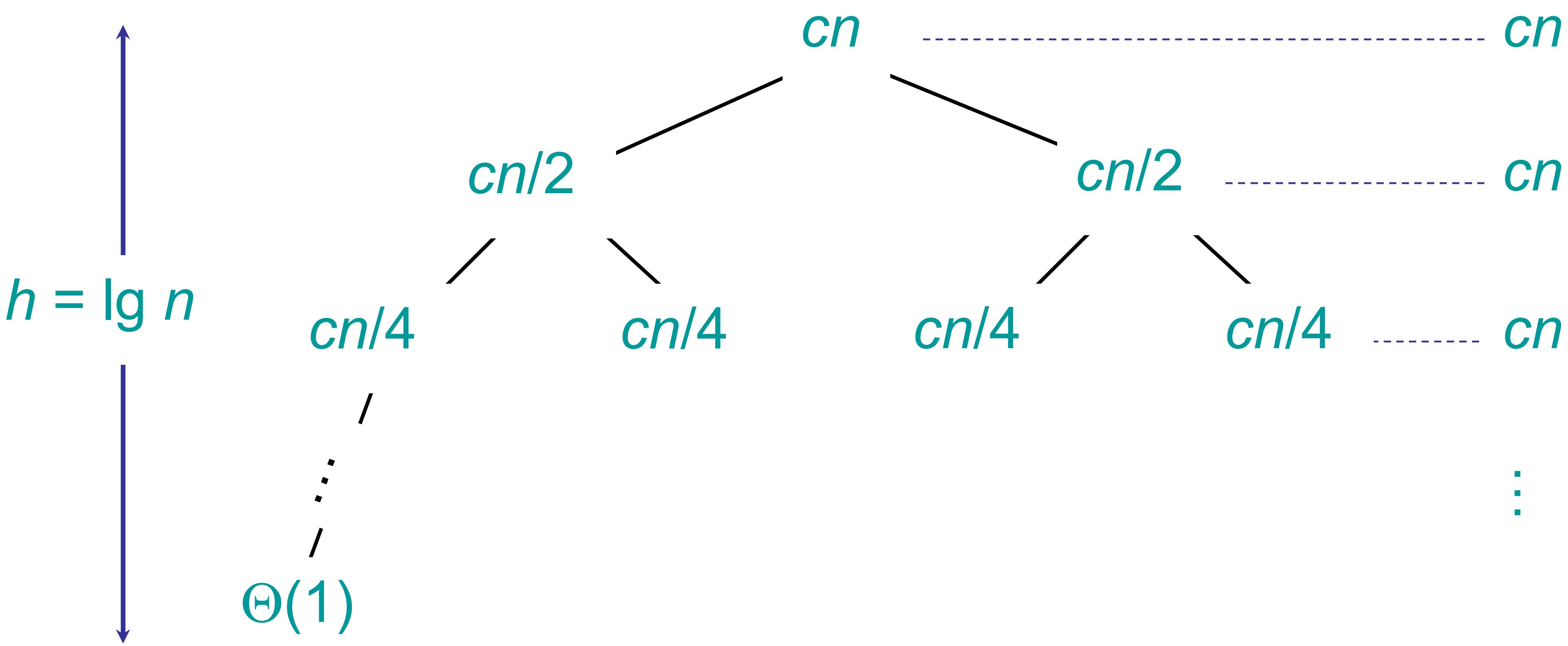
Recursion Tree

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



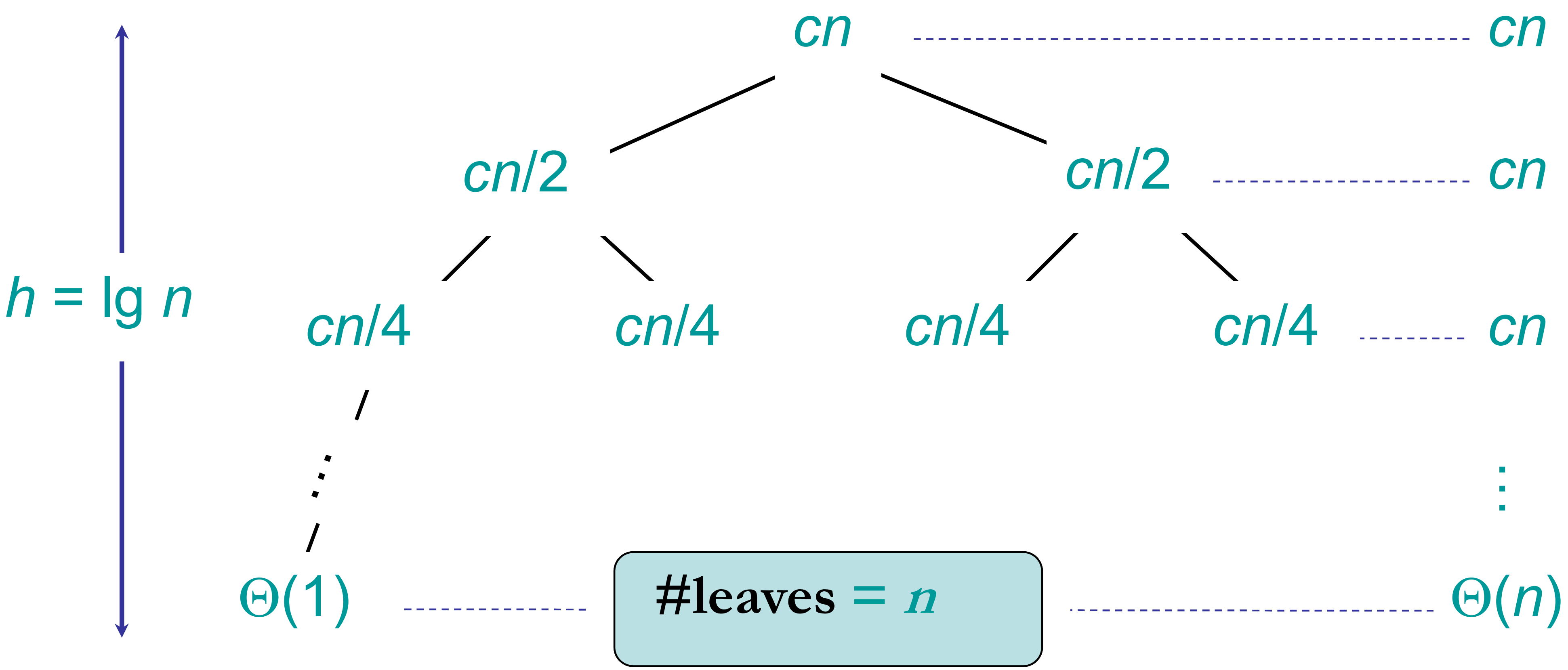
Recursion Tree

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



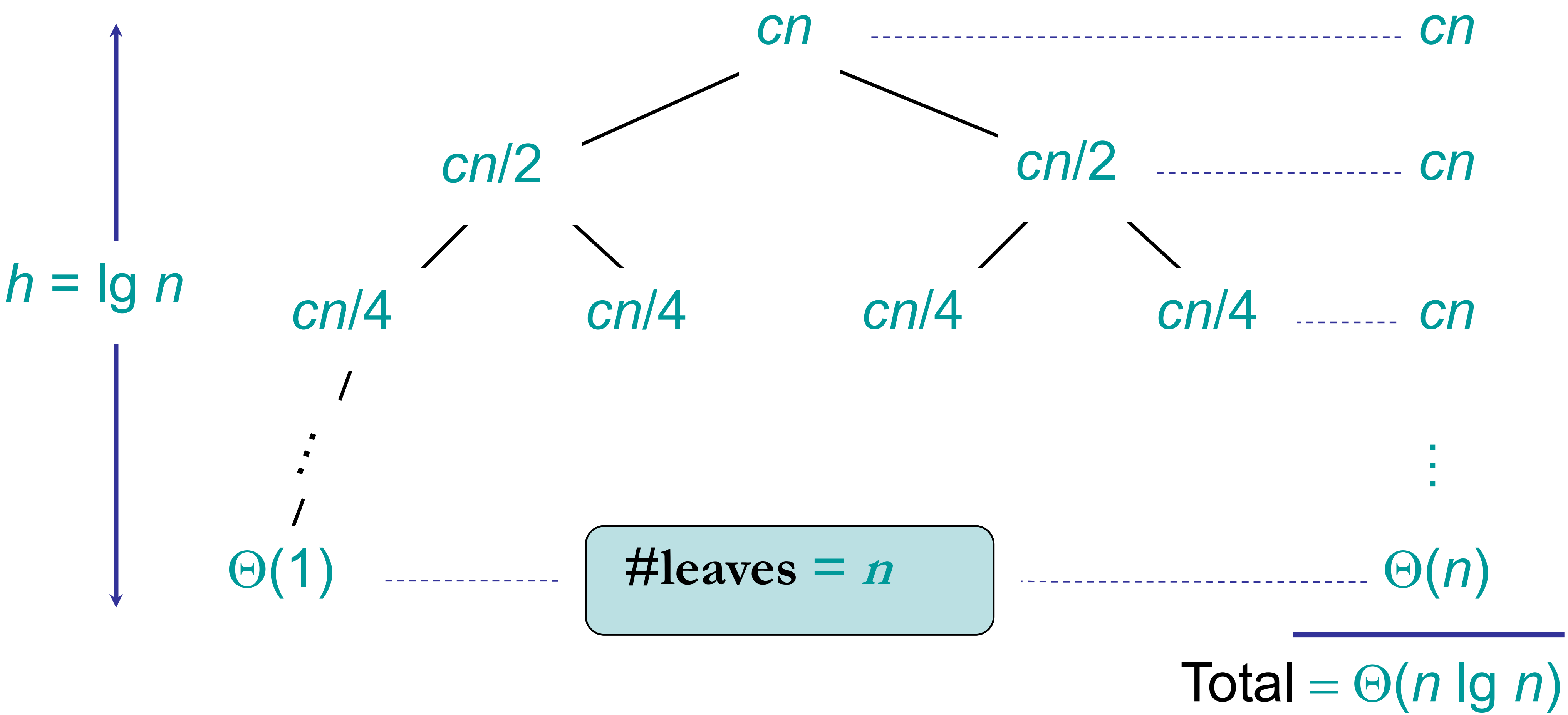
Recursion Tree

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.



Summary

- $\Theta(n \lg n)$ 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.