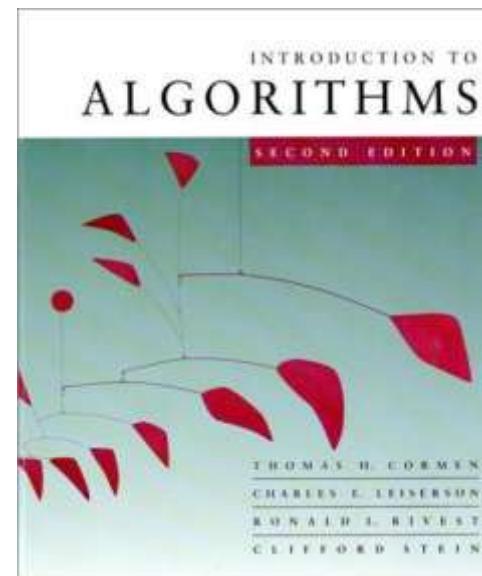


Introduction to Algorithms



Ms B N Jyothi
Assistant Professor -CSE

Welcome to *Introduction to Algorithms*, Spring 2004

Handouts

1. [Course Information](#)
2. [Course Calendar](#)
3. Problem Set 1
4. Akra-Bazzi Handout

Course information

1. Staff
2. Prerequisites
3. Lectures & Recitations
4. Handouts
5. Textbook (CLRS)
6. Website
8. Extra Help
9. Registration
10. Problem sets
11. Describing algorithms
12. Grading policy
13. Collaboration policy

What is course about?

The theoretical study of design and analysis of computer algorithms

Basic goals for an algorithm:

- always correct
- always terminates
- This 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

The problem of sorting

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

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

Example:

Input: 8 2 4 9 3 6

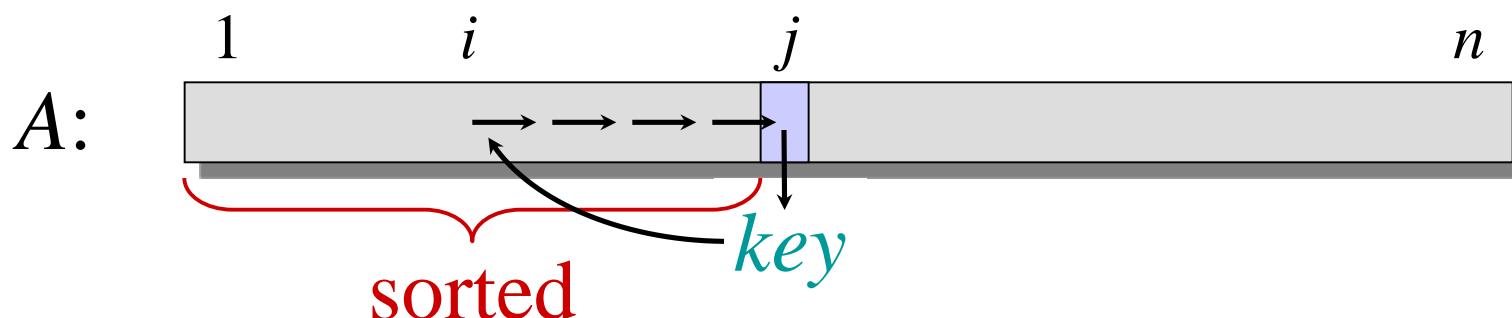
Output: 2 3 4 6 8 9

Insertion sort

“pseudocode”

INSERTION-SORT (A, n) $\triangleright A[1 \dots n]$

```
for  $j \leftarrow 2$  to  $n$ 
    do  $key \leftarrow A[j]$ 
         $i \leftarrow j - 1$ 
        while  $i > 0$  and  $A[i] > key$ 
            do  $A[i+1] \leftarrow A[i]$ 
                 $i \leftarrow i - 1$ 
         $A[i+1] = key$ 
```



Example of insertion sort

8

2

4

9

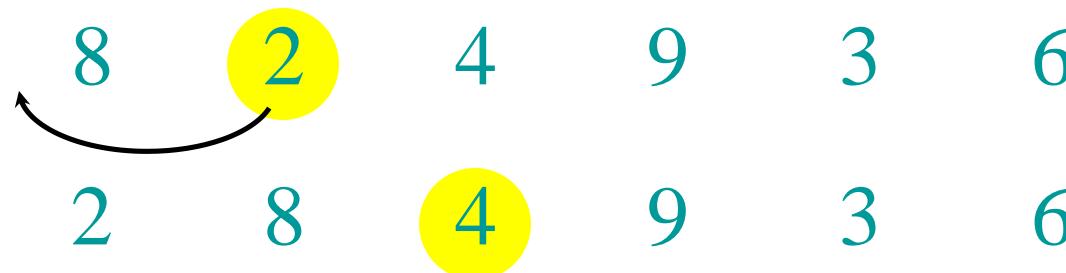
3

6

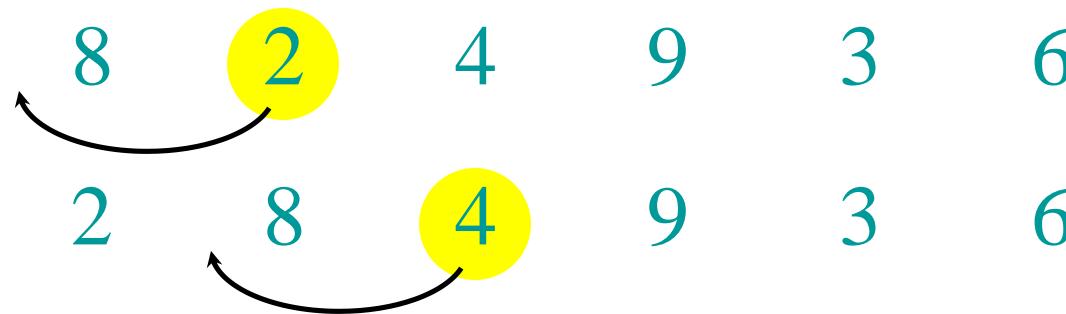
Example of insertion sort



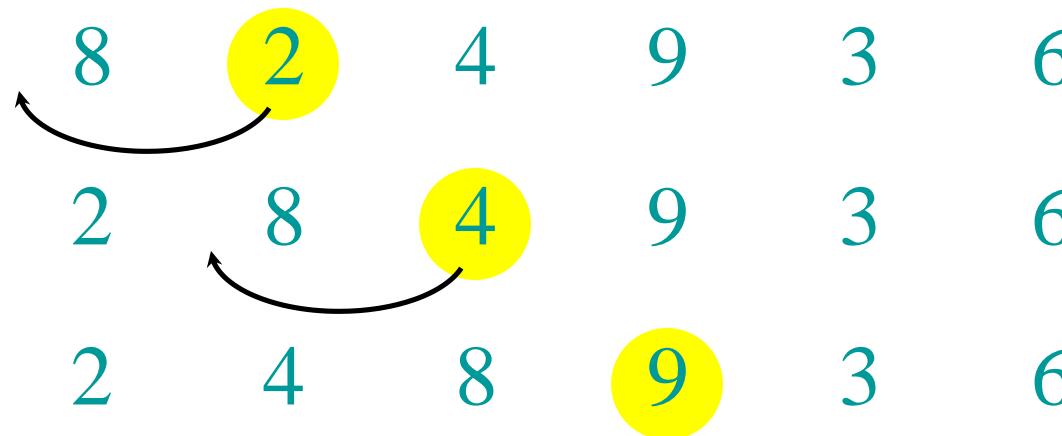
Example of insertion sort



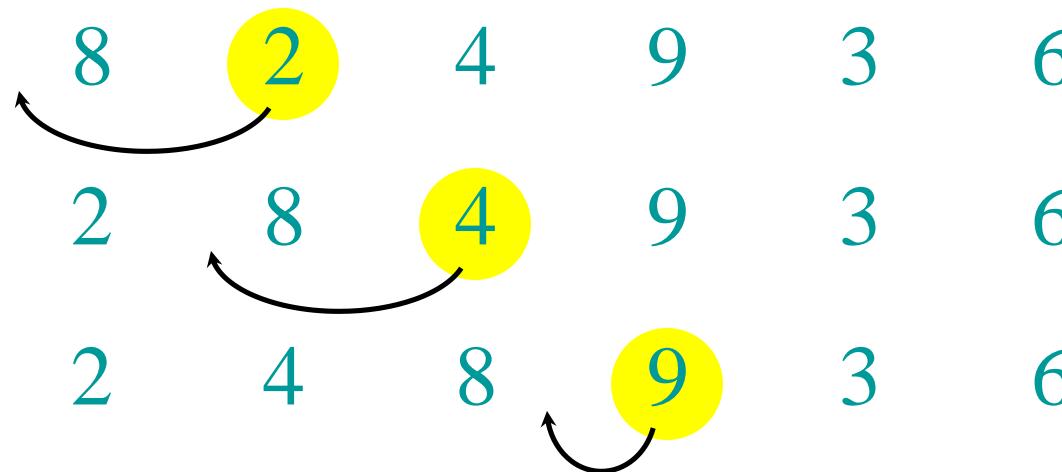
Example of insertion sort



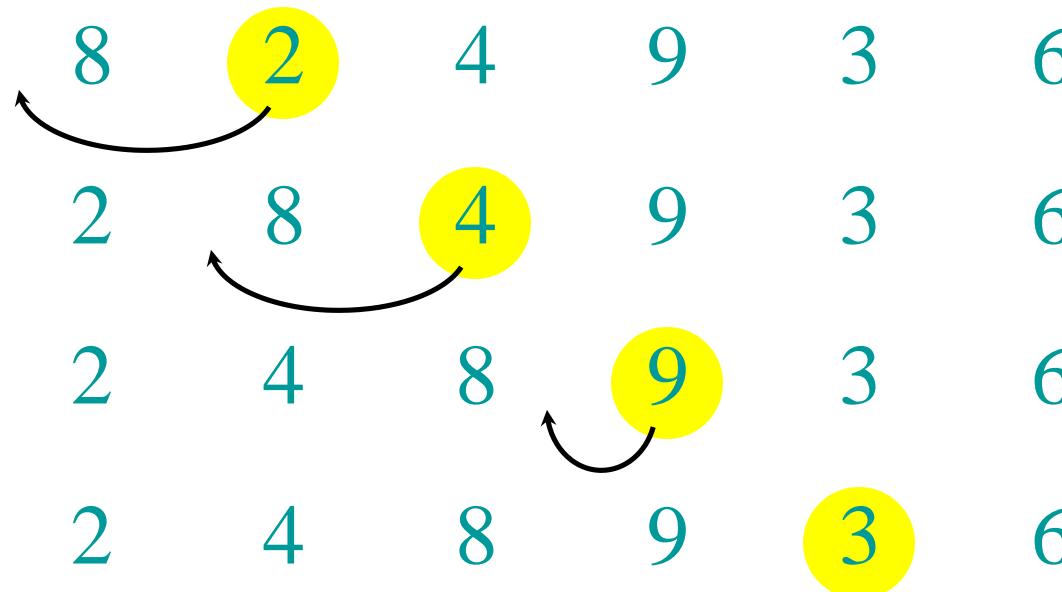
Example of insertion sort



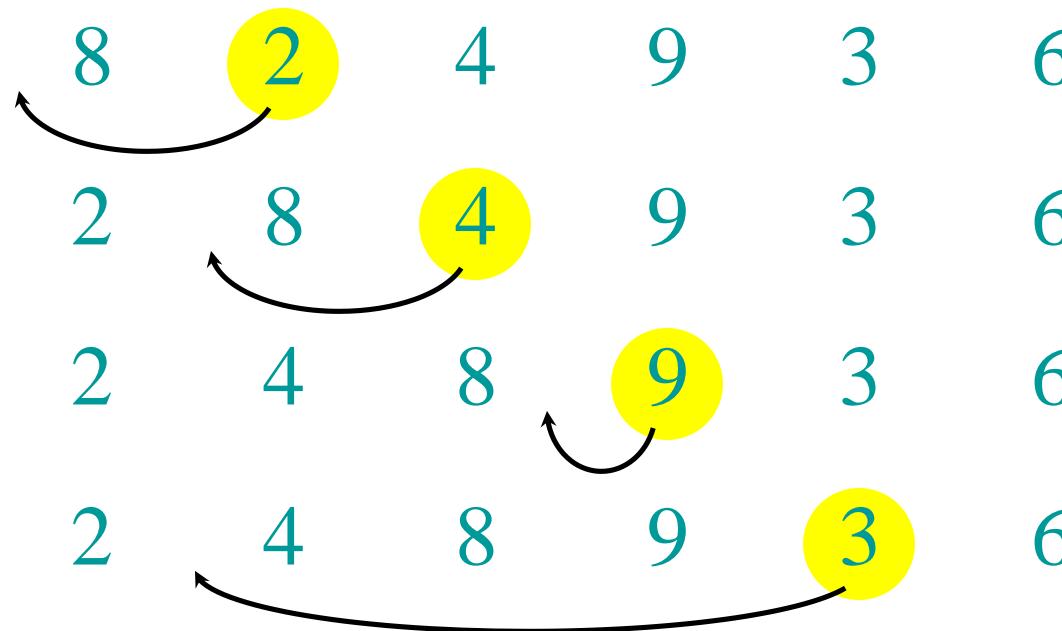
Example of insertion sort



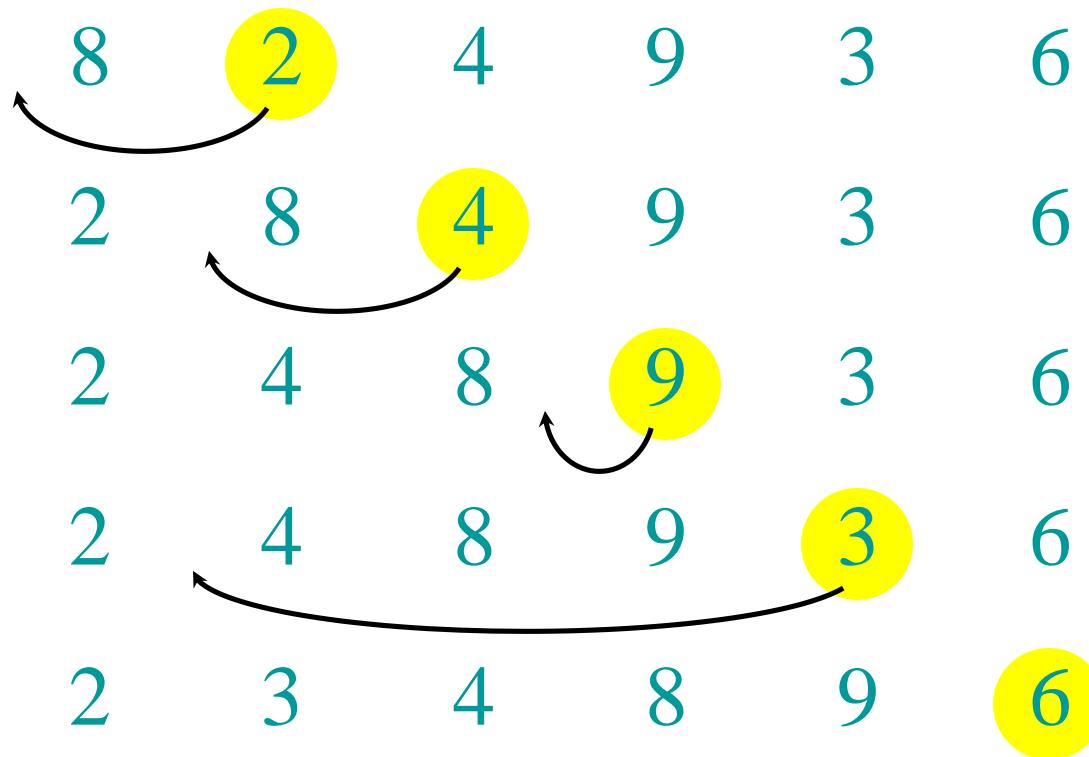
Example of insertion sort



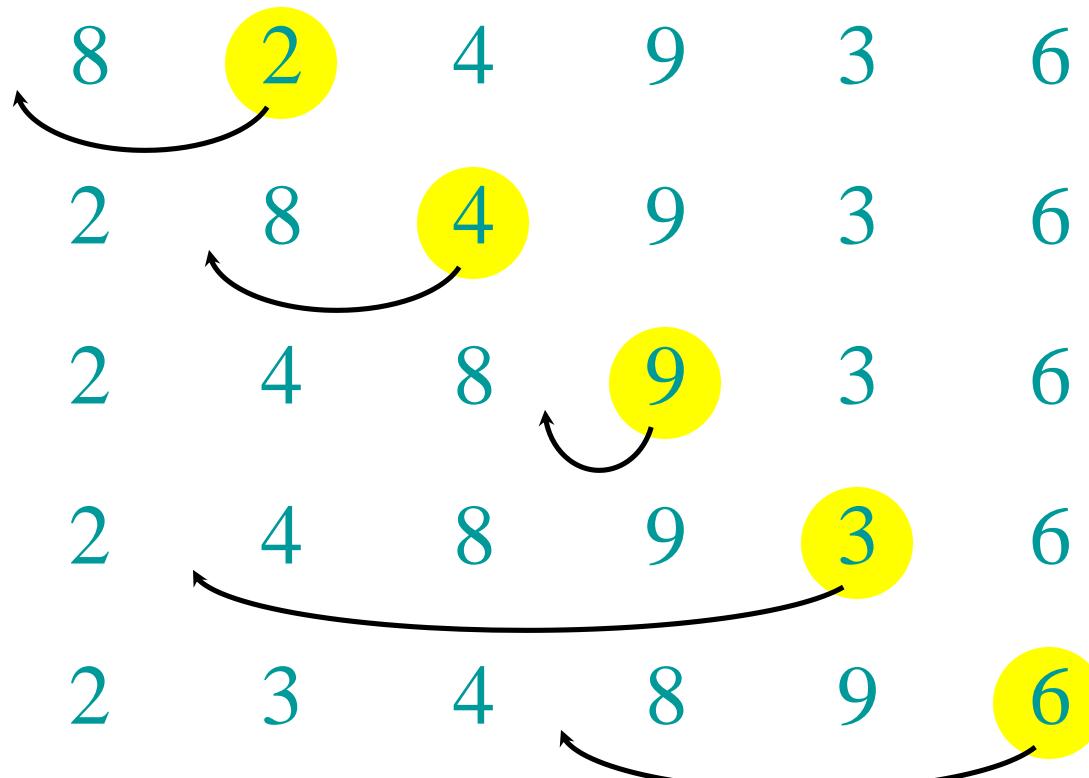
Example of insertion sort



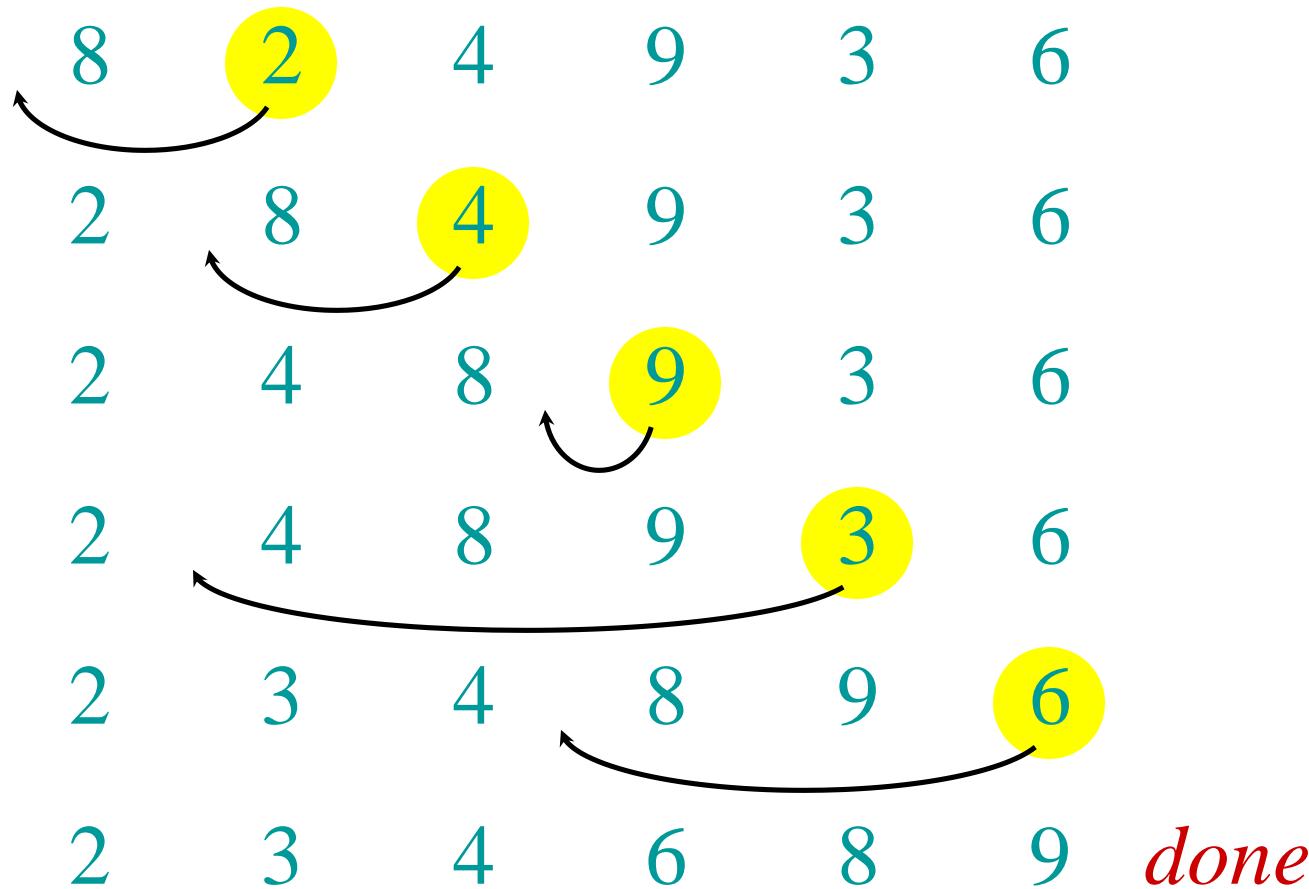
Example of insertion sort



Example of insertion sort



Example of insertion sort



Running time

- The 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

What is insertion sort's worst-case time?

BIG IDEAS:

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

“**Asymptotic Analysis**”

Θ -notation

DEF:

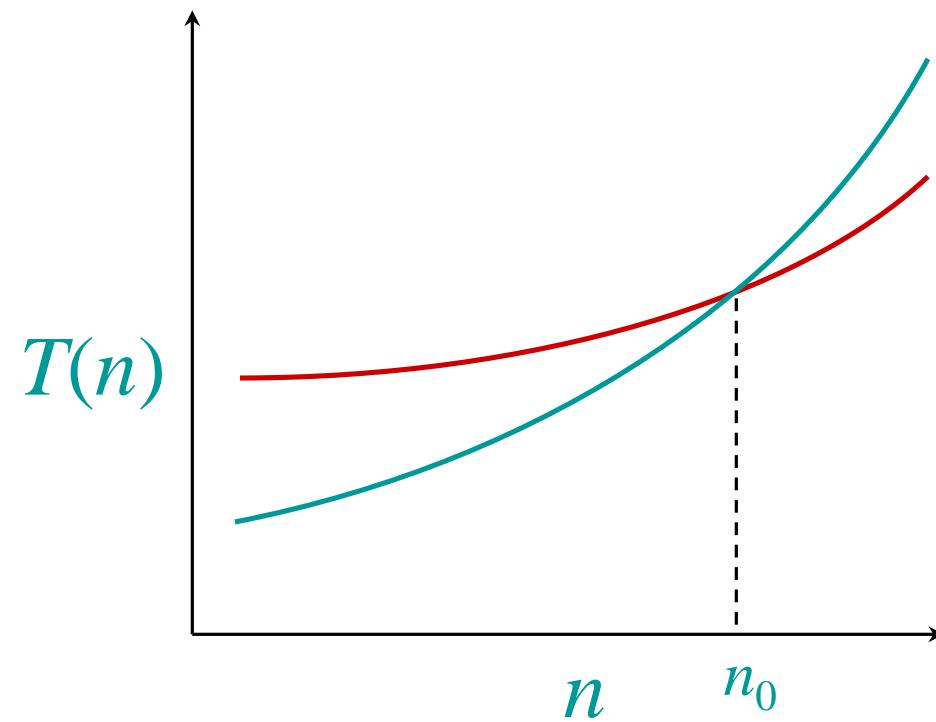
$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$

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.

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

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 .

Example 2: Integer Multiplication

- Let $X = \boxed{A} \boxed{B}$ and $Y = \boxed{C} \boxed{D}$ where A,B,C and D are $n/2$ bit integers
- Simple Method: $XY = (2^{n/2}A+B)(2^{n/2}C+D)$
- Running Time Recurrence

$$T(n) < 4T(n/2) + 100n$$

- Solution $T(n) = \theta(n^2)$

Better Integer Multiplication

- Let $X = \boxed{A} \boxed{B}$ and $Y = \boxed{C} \boxed{D}$ where A,B,C and D are $n/2$ bit integers
- **Karatsuba:**
$$XY = (2^{n/2}+2^n)AC + 2^{n/2}(A-B)(C-D) + (2^{n/2}+1) BD$$
- **Running Time Recurrence**
$$T(n) < 3T(n/2) + 100n$$
- **Solution:** $\theta(n) = O(n^{\log 3})$

Example 3: 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**

Merging two sorted arrays

20 12

13 11

7 9

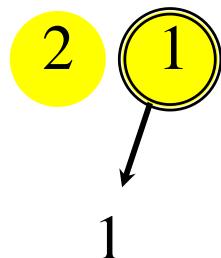
2 1

Merging two sorted arrays

20 12

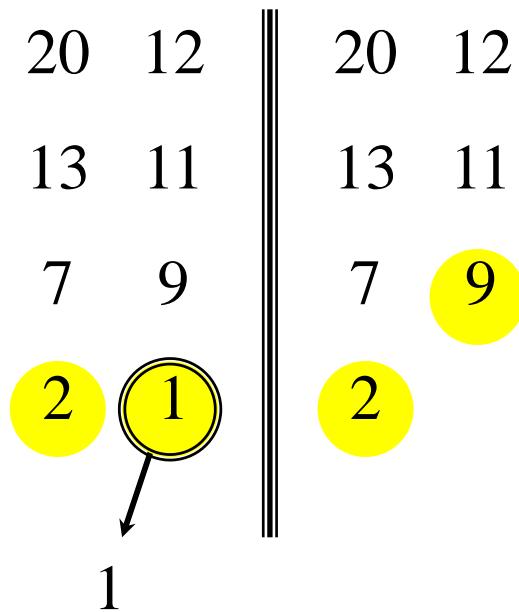
13 11

7 9



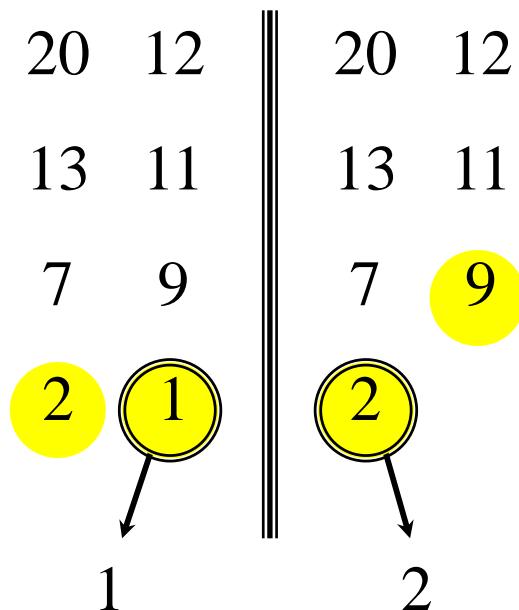


Merging two sorted arrays



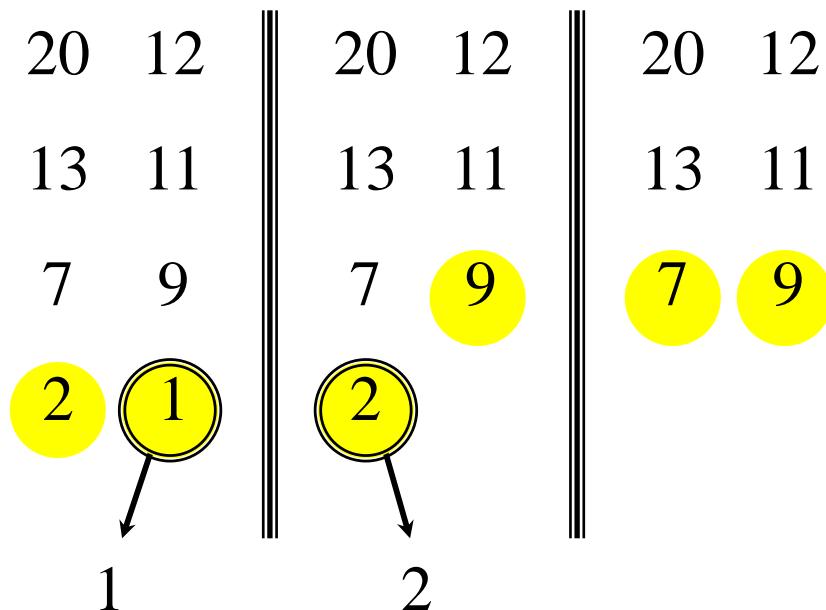


Merging two sorted arrays



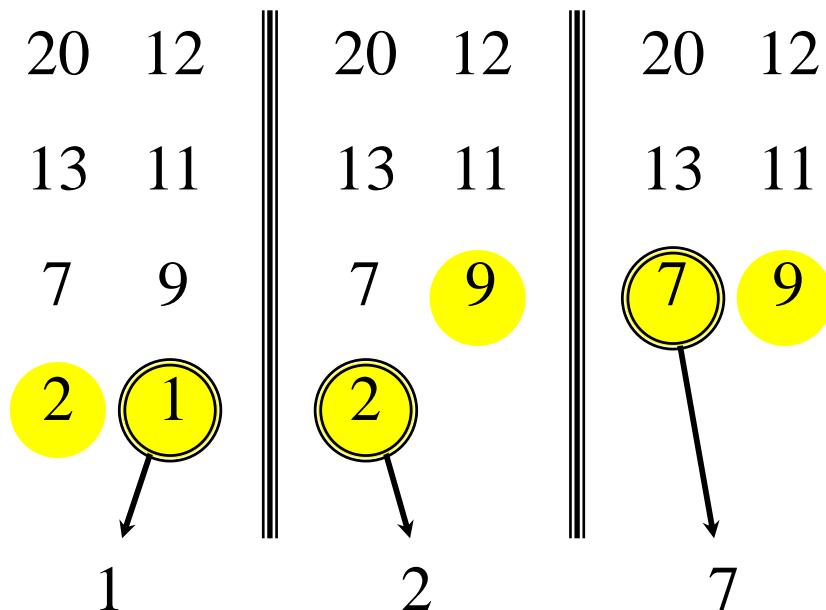


Merging two sorted arrays



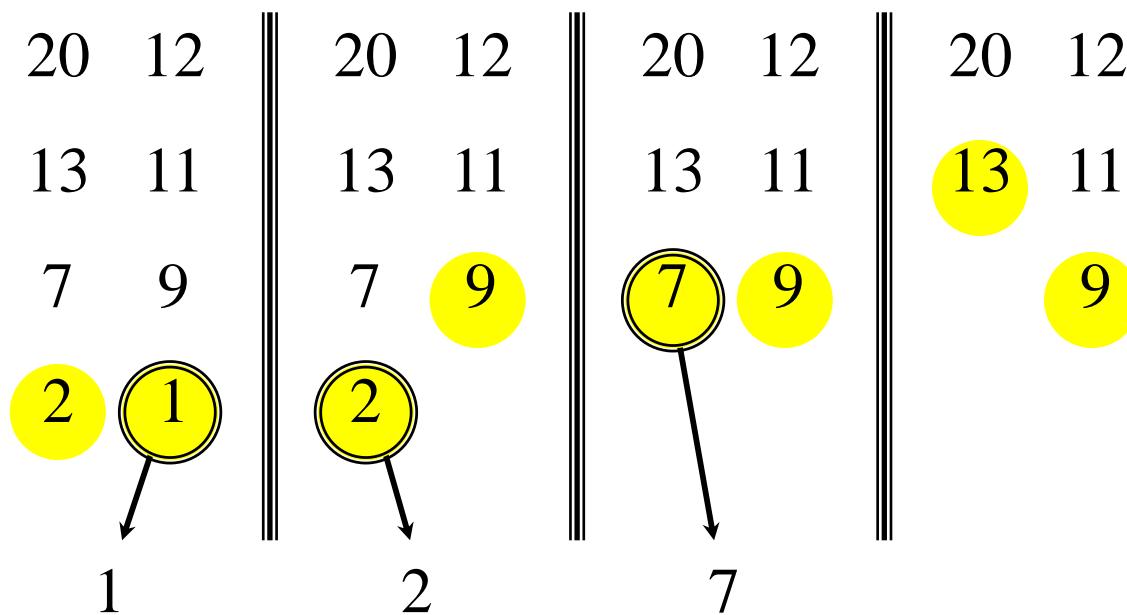


Merging two sorted arrays



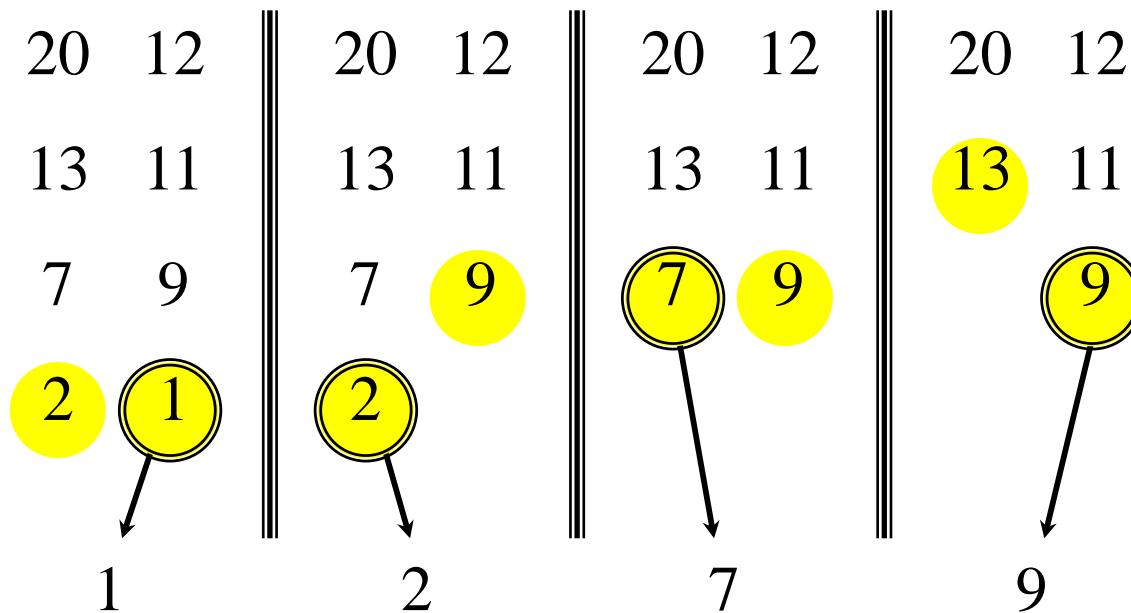


Merging two sorted arrays



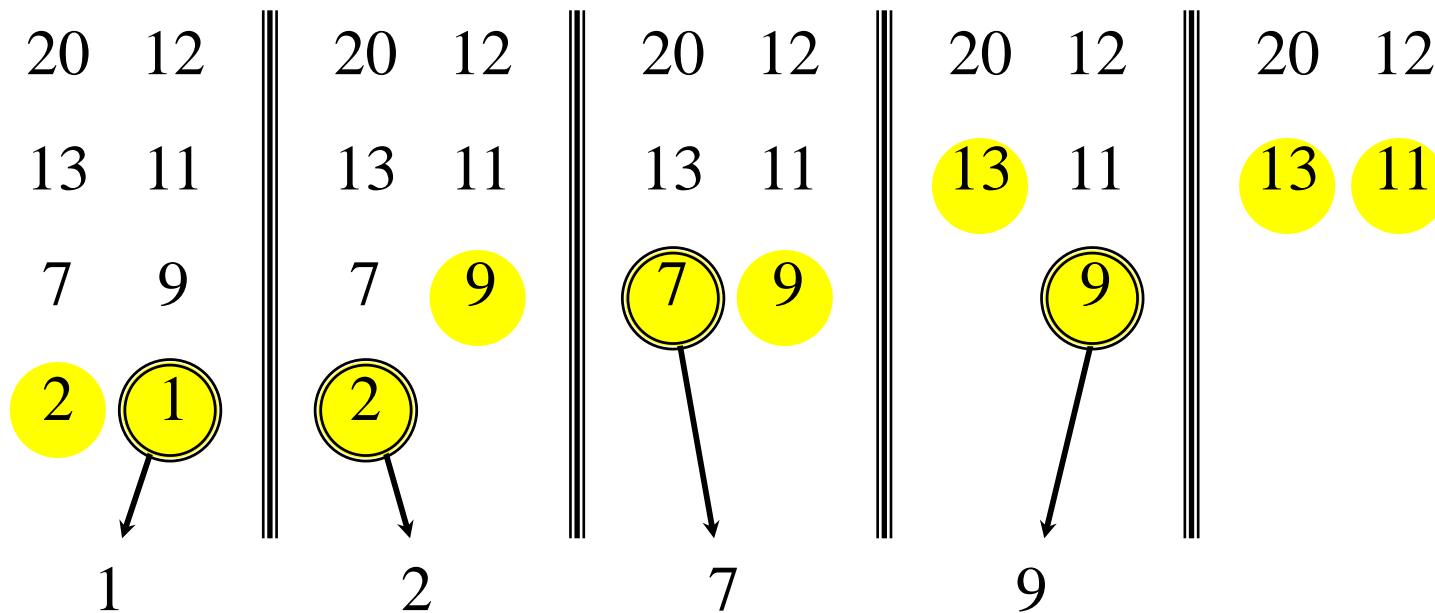


Merging two sorted arrays



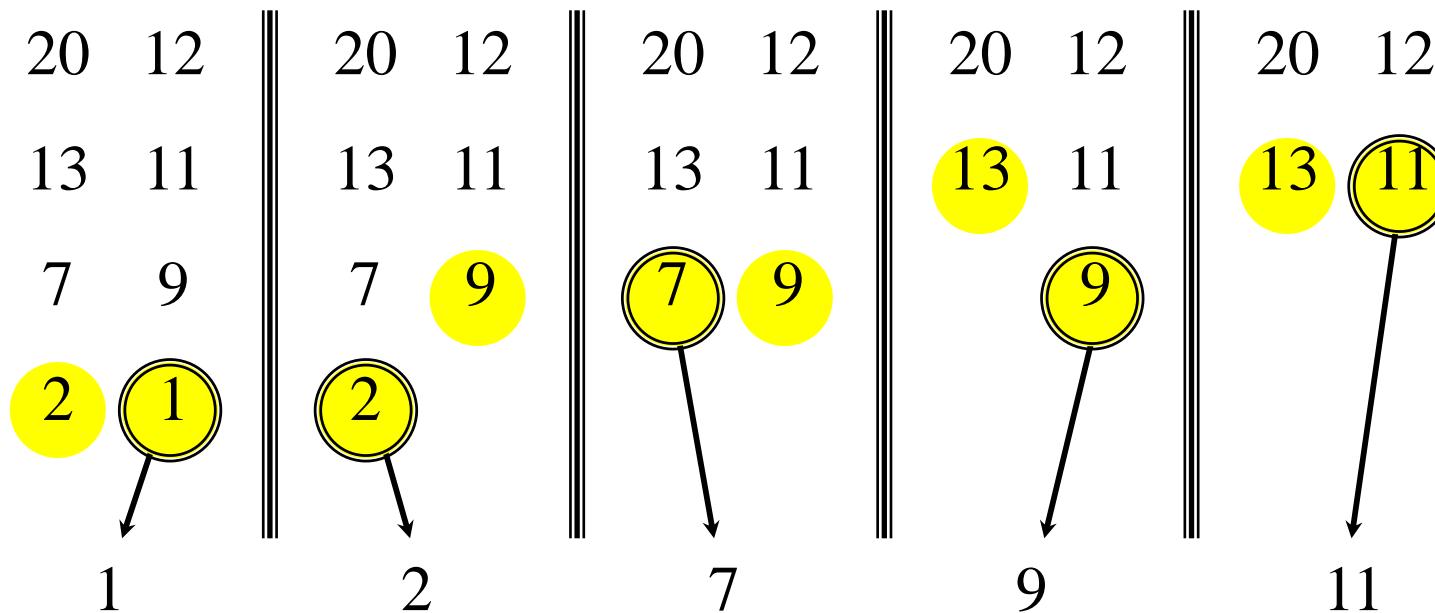


Merging two sorted arrays



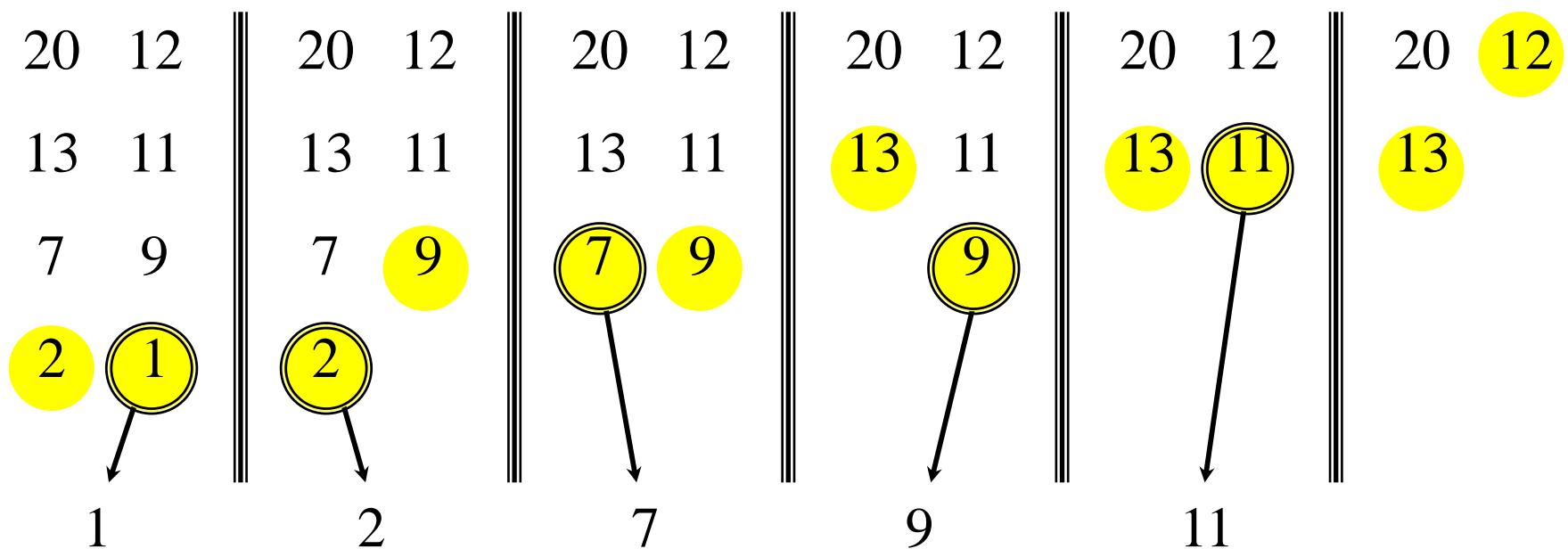


Merging two sorted arrays



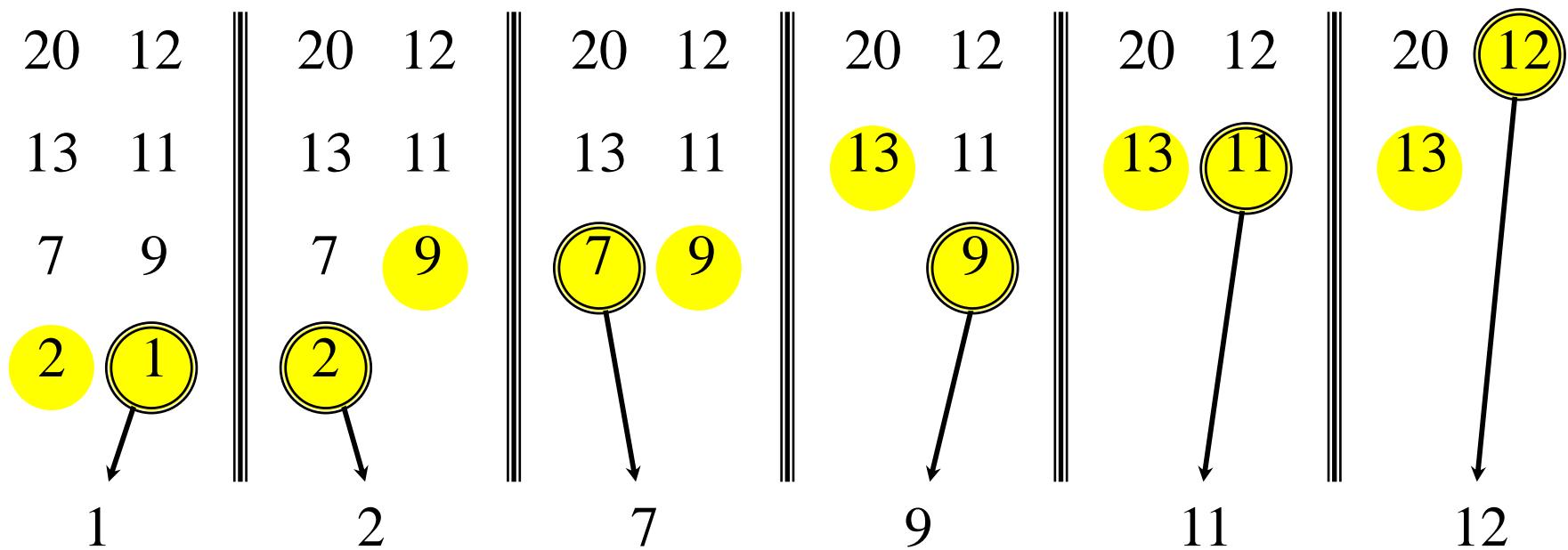


Merging two sorted arrays

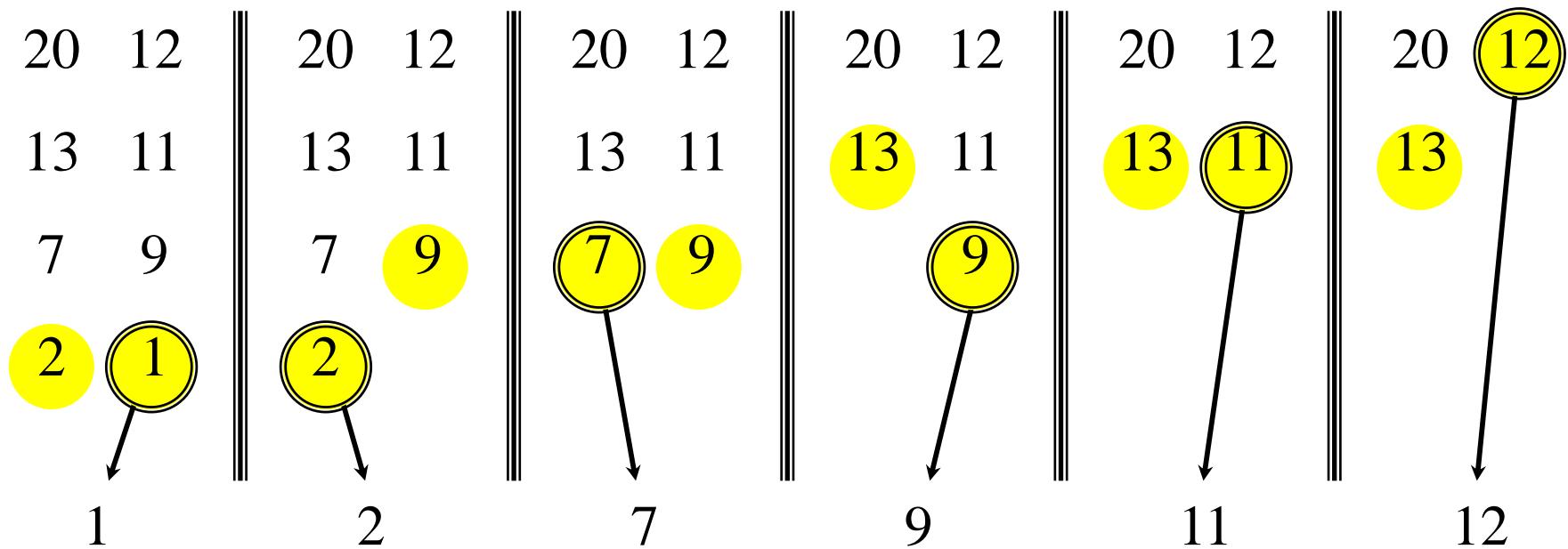




Merging two sorted arrays

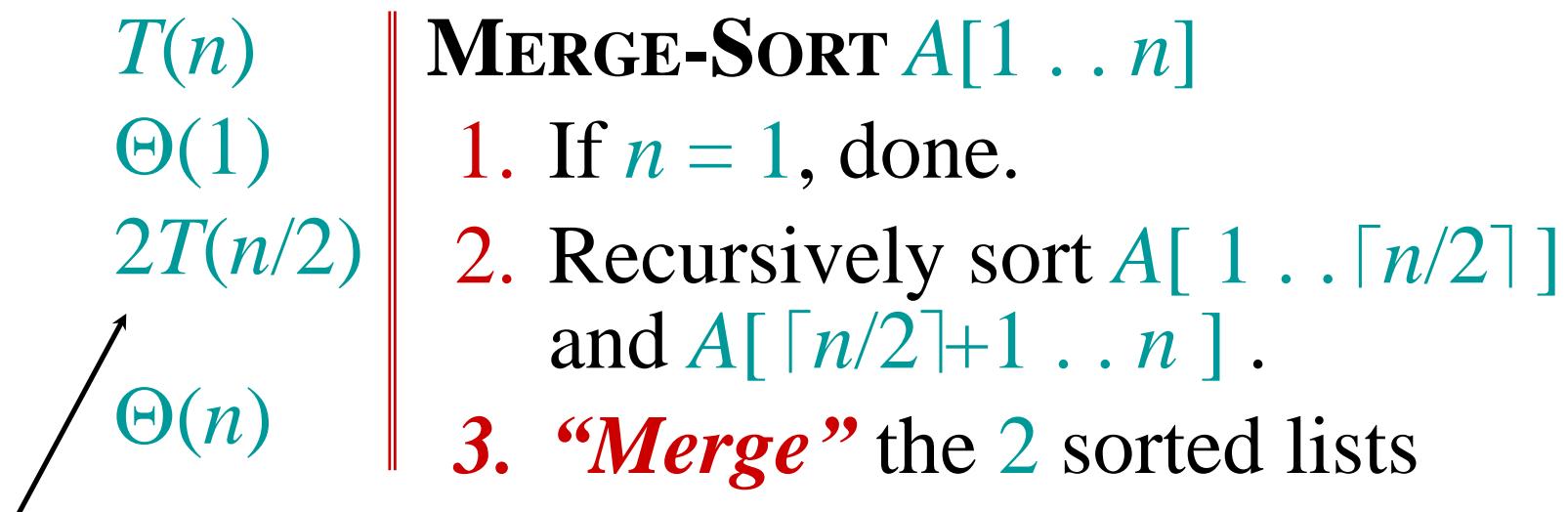


Merging two sorted arrays



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.
- Lecture 2 provides 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.

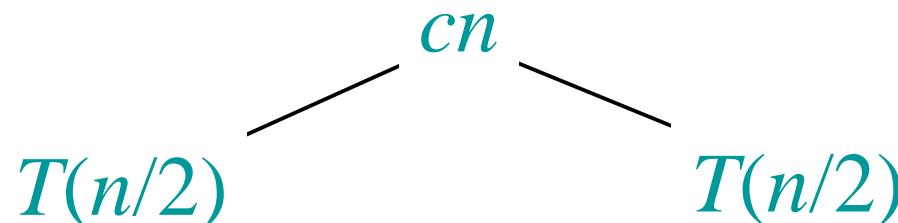
Recursion tree

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

$$T(n)$$

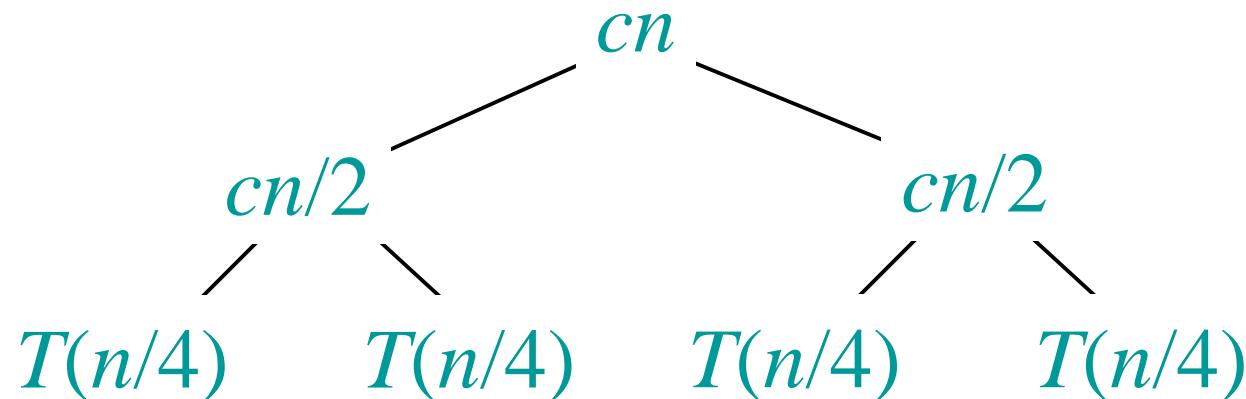
Recursion tree

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



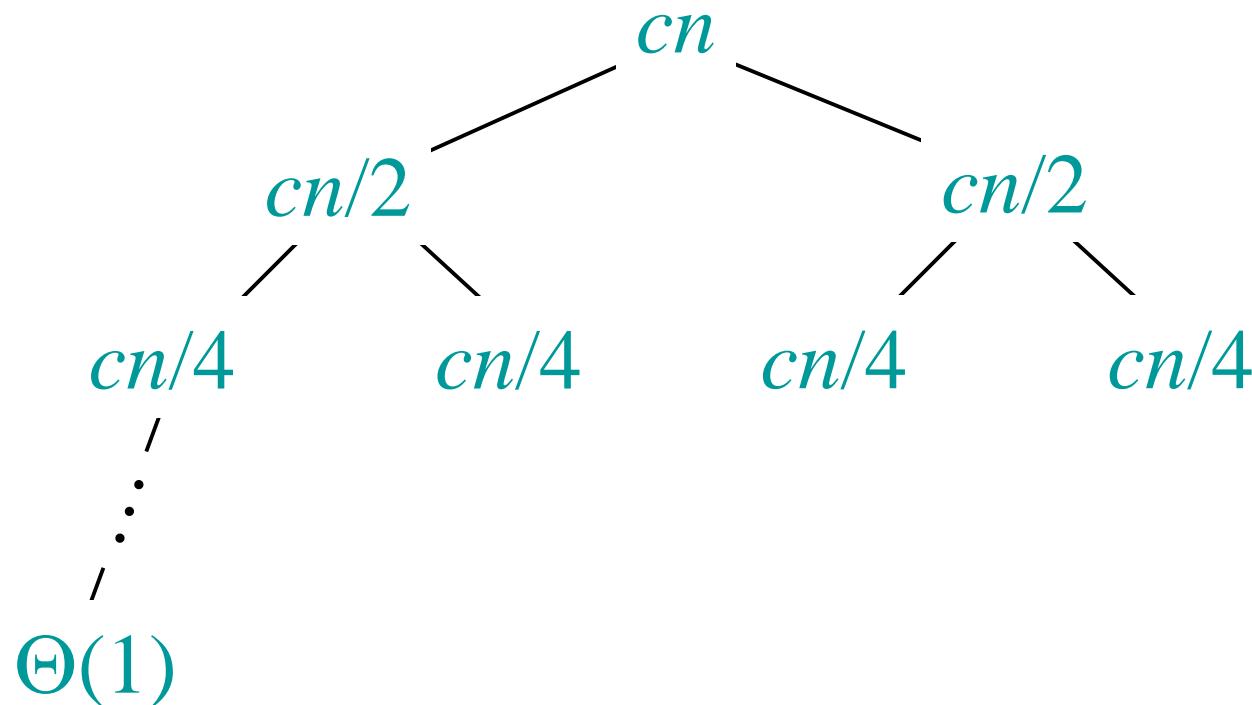
Recursion tree

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



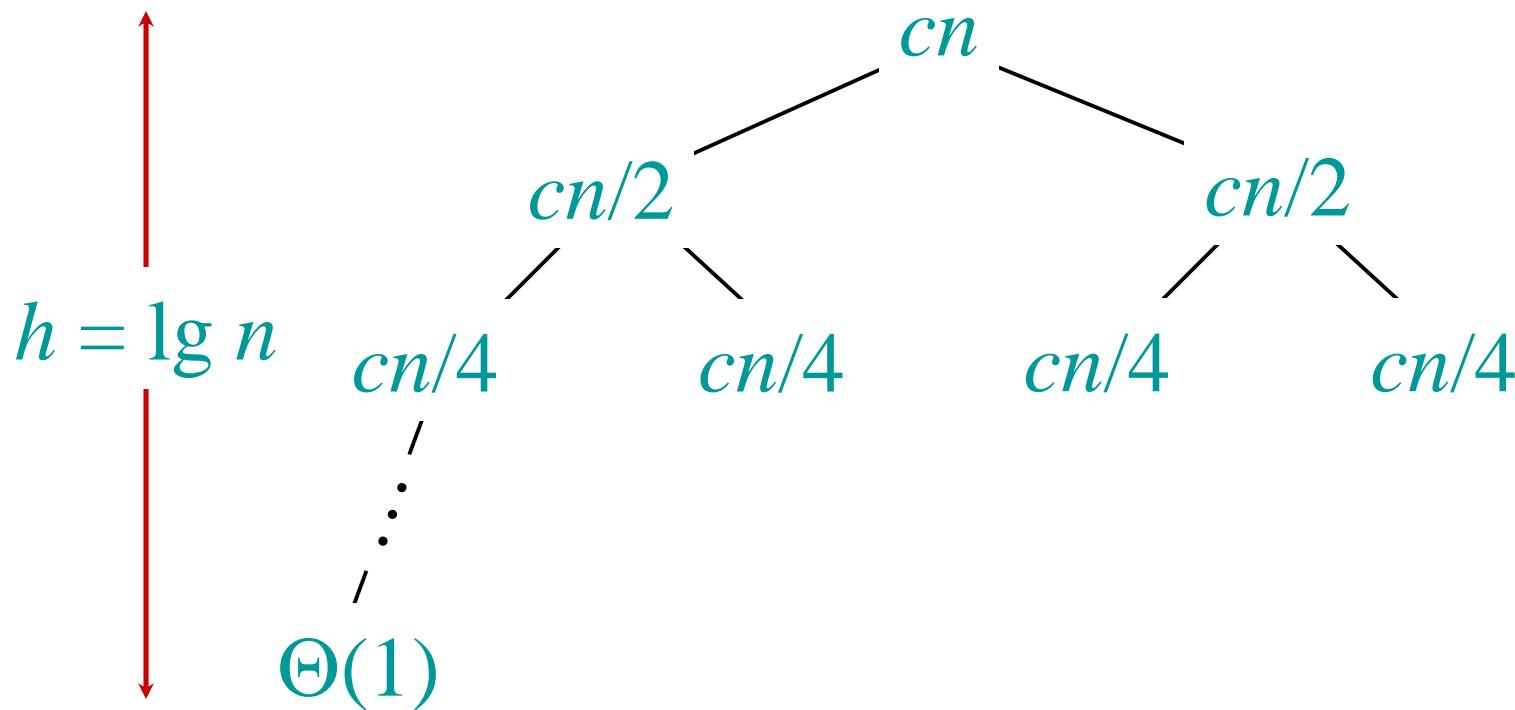
Recursion tree

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



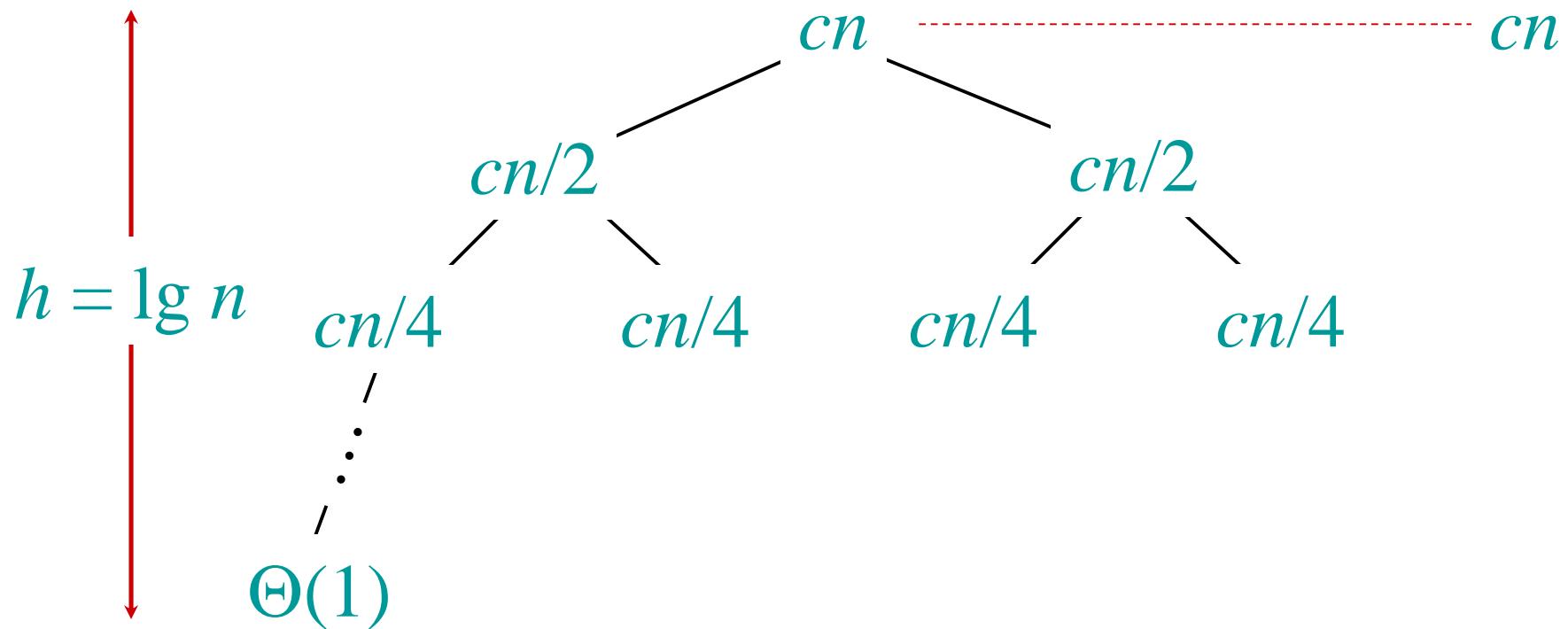
Recursion tree

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



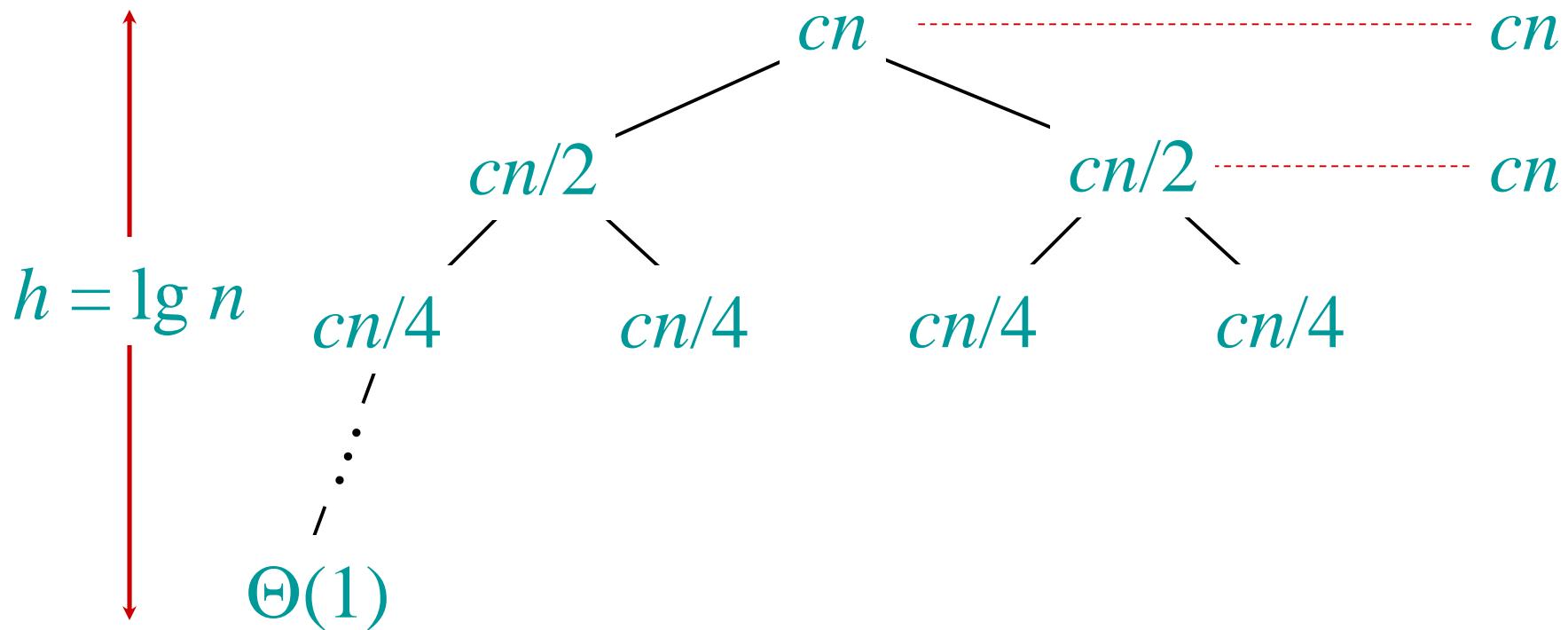
Recursion tree

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



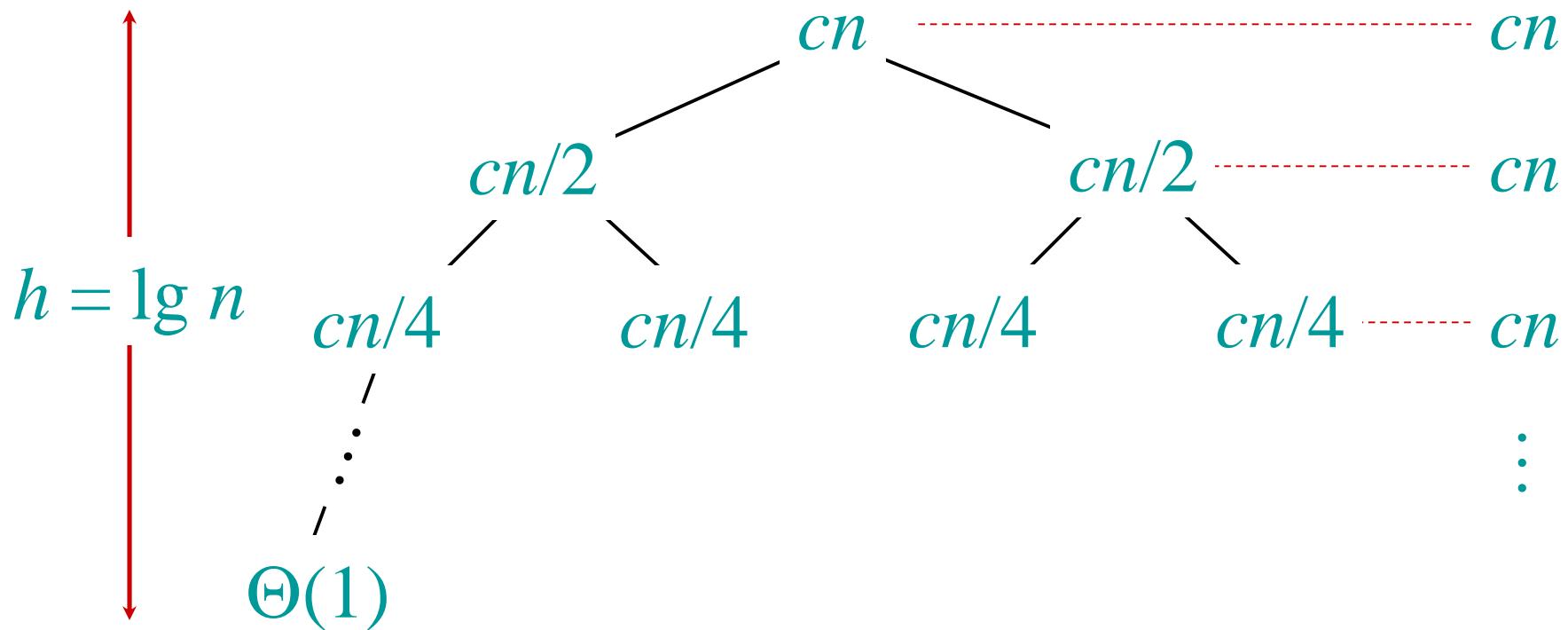
Recursion tree

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



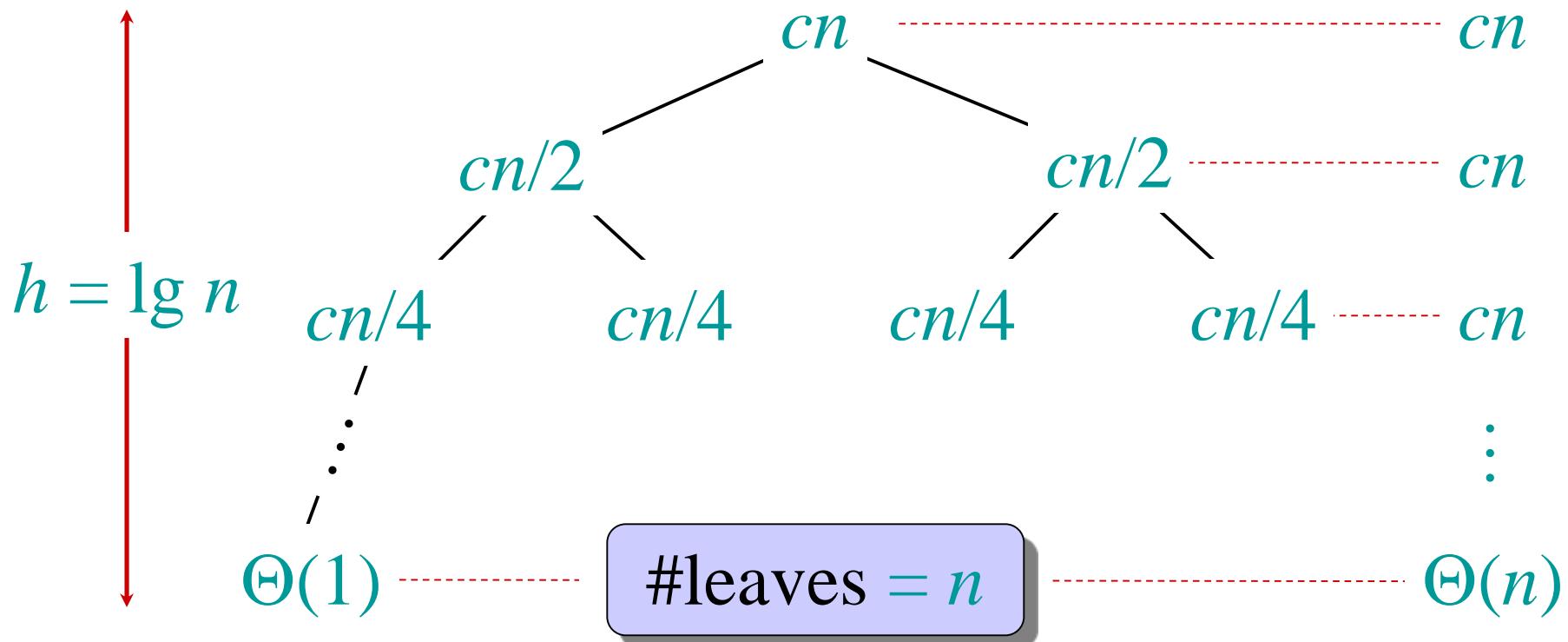
Recursion tree

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



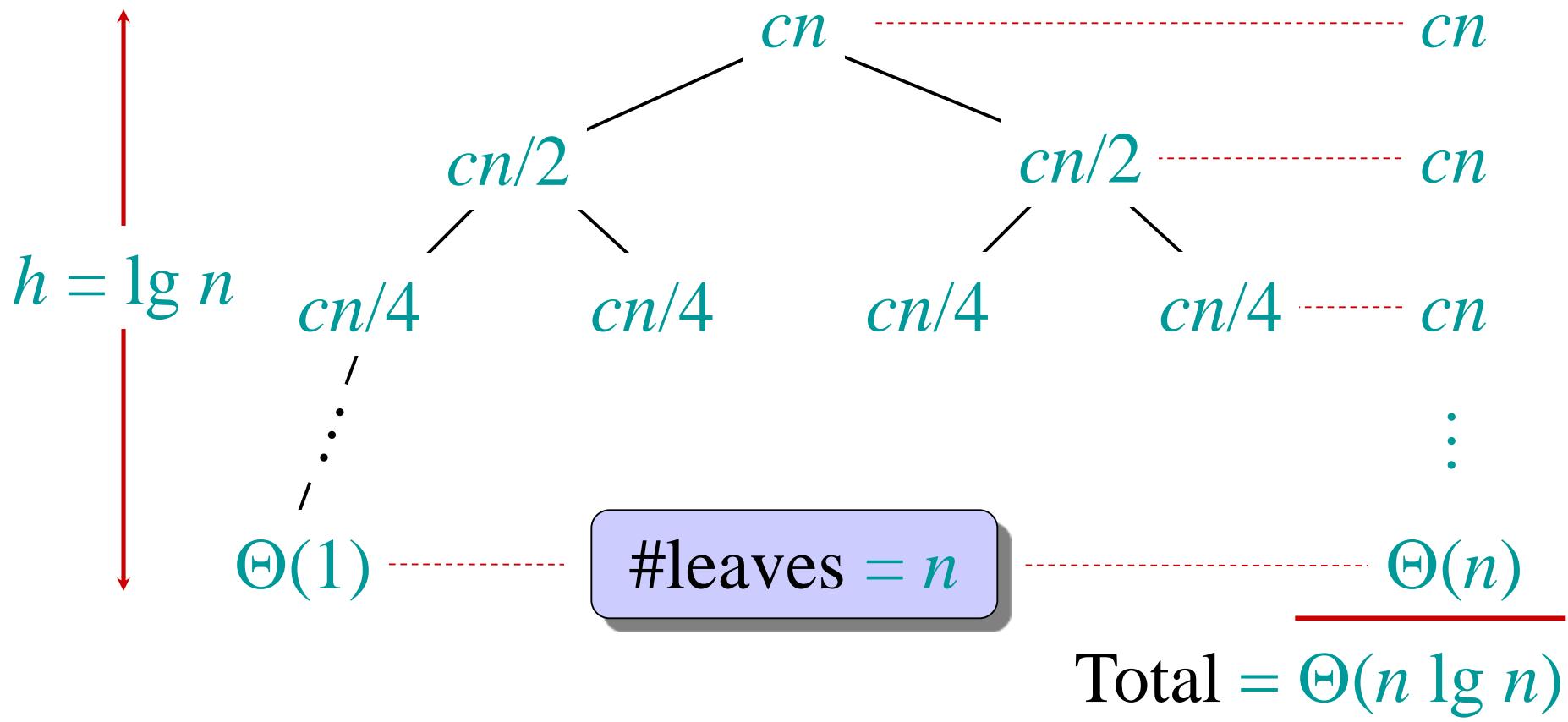
Recursion tree

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.



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.