数据结构与算法设计

3.1 插入排序

3.2 分治与归并排序

The problem of sorting

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

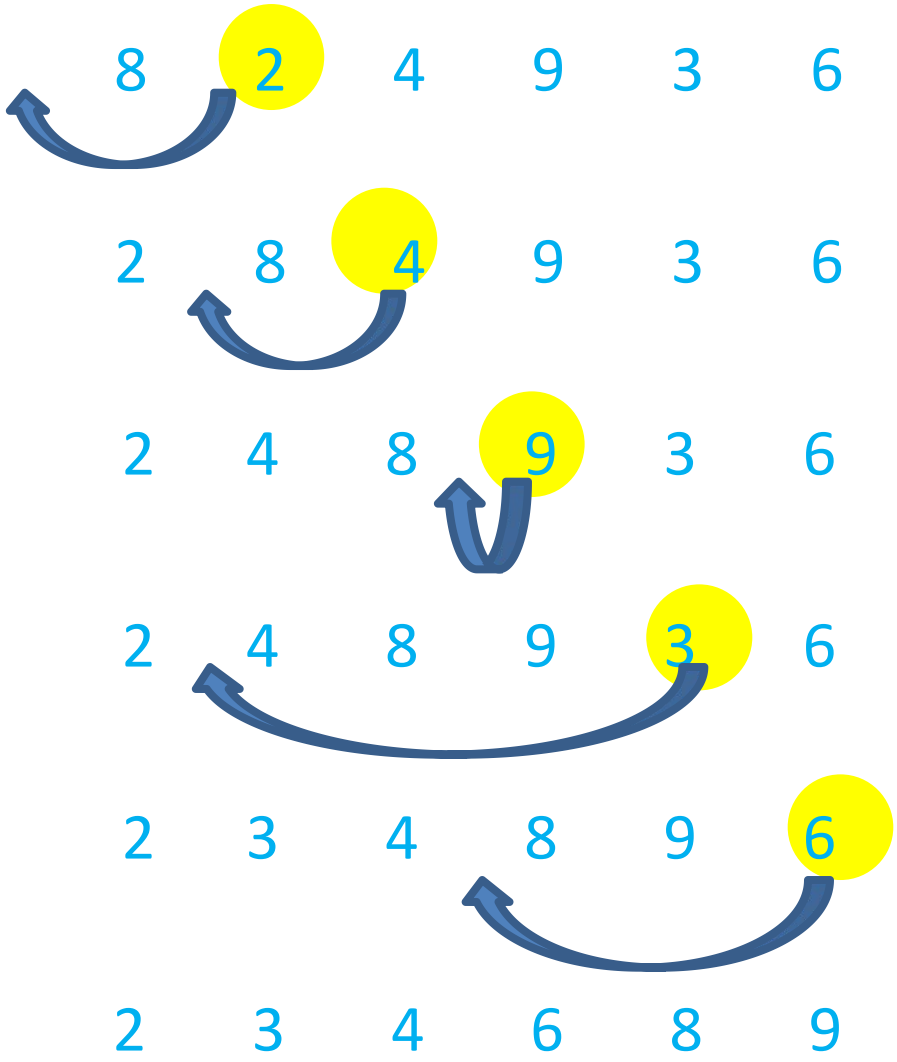
Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Example of insertion sorting

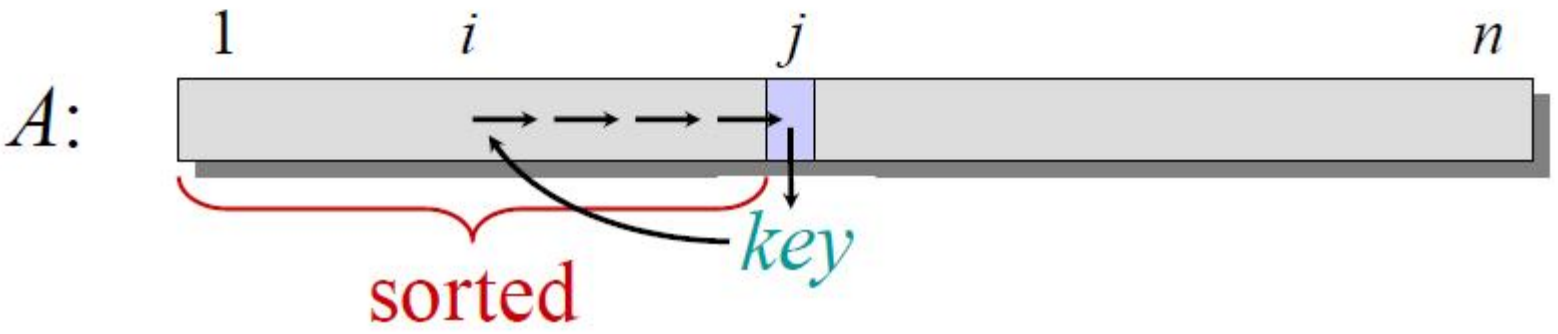


思考：箭头长短意味着什么?

Insertion sort

"pseudocode"

```
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
```



Running time

INSERTION-SORT(A)	代价	次数
1 for j =2 to A.length	c_1	n
2 key=A[j]	c_2	n-1
3 //Insert A[j] into the sorted sequence A[1..j-1].	0	n-1
4 i=j-1	c_4	n-1
5 while i>0 and A[i]>key	c_5	$\sum_{j=2}^{n} t_j$
6 A[i+1]=A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 i=i-1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 A[i+1]=key	c_8	n-1

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

Running time

若输入数组已排好序，则出现最佳情况：

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

若输入数组已反向排序，则导致最坏情况：

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right)$$
$$+ c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$
$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n$$
$$- (c_2 + c_4 + c_5 + c_8)$$

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.

课堂练习：

① 对于一个已经排好序的序列A，采用二分查找法查找一个值v，问：能否写出二分查找的伪代码？二分查找的最坏情况运行时间是多少？

② 对于插入排序过程中第5~7行的while循环采用二分查找，问：使用二分查找后，插入排序的最坏情况总运行时间能改进吗？

Algorithm 3 BINARY-SEARCH(A, v, p, r)

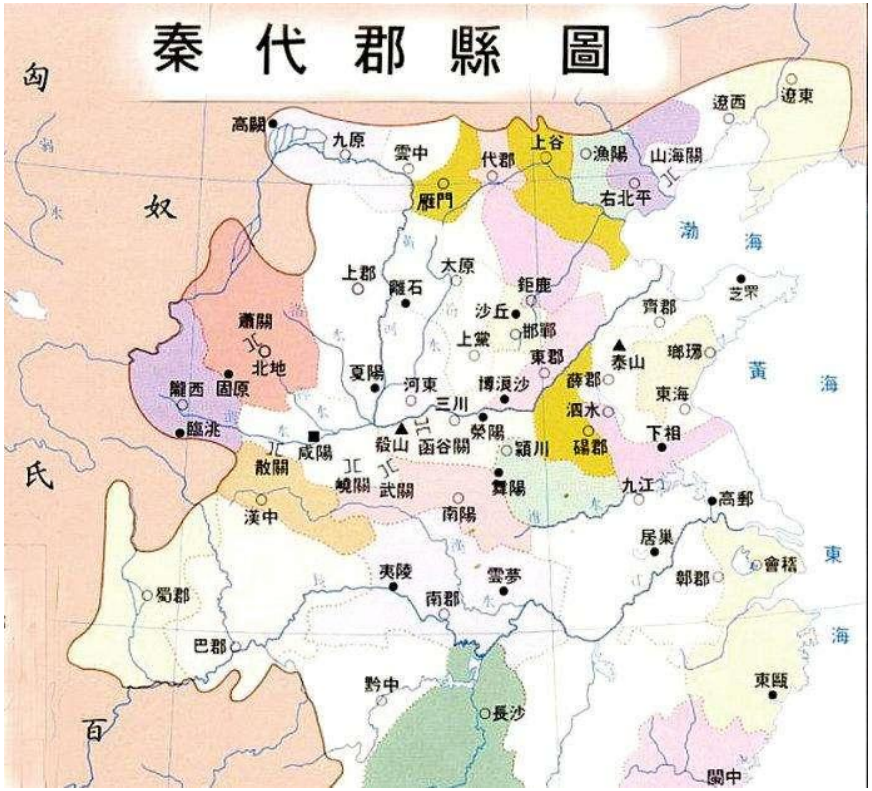
Input: A sorted array A and a value v.
Output: An index i such that $v = A[i]$ or **nil**.
if $p \geqslant r$ **and** $v \neq A[p]$ **then**
  return nil
end if
$j \leftarrow \lfloor (r + p)/2 \rfloor$
if $v = A[j]$ **then**
  return j
else
  if $v < A[j]$ **then**
    return BINARY-SEARCH(A, v, p, j)
  else
    return BINARY-SEARCH(A, v, j, r)
  end if
end if

3.1 插入排序

3.2 分治与归并排序

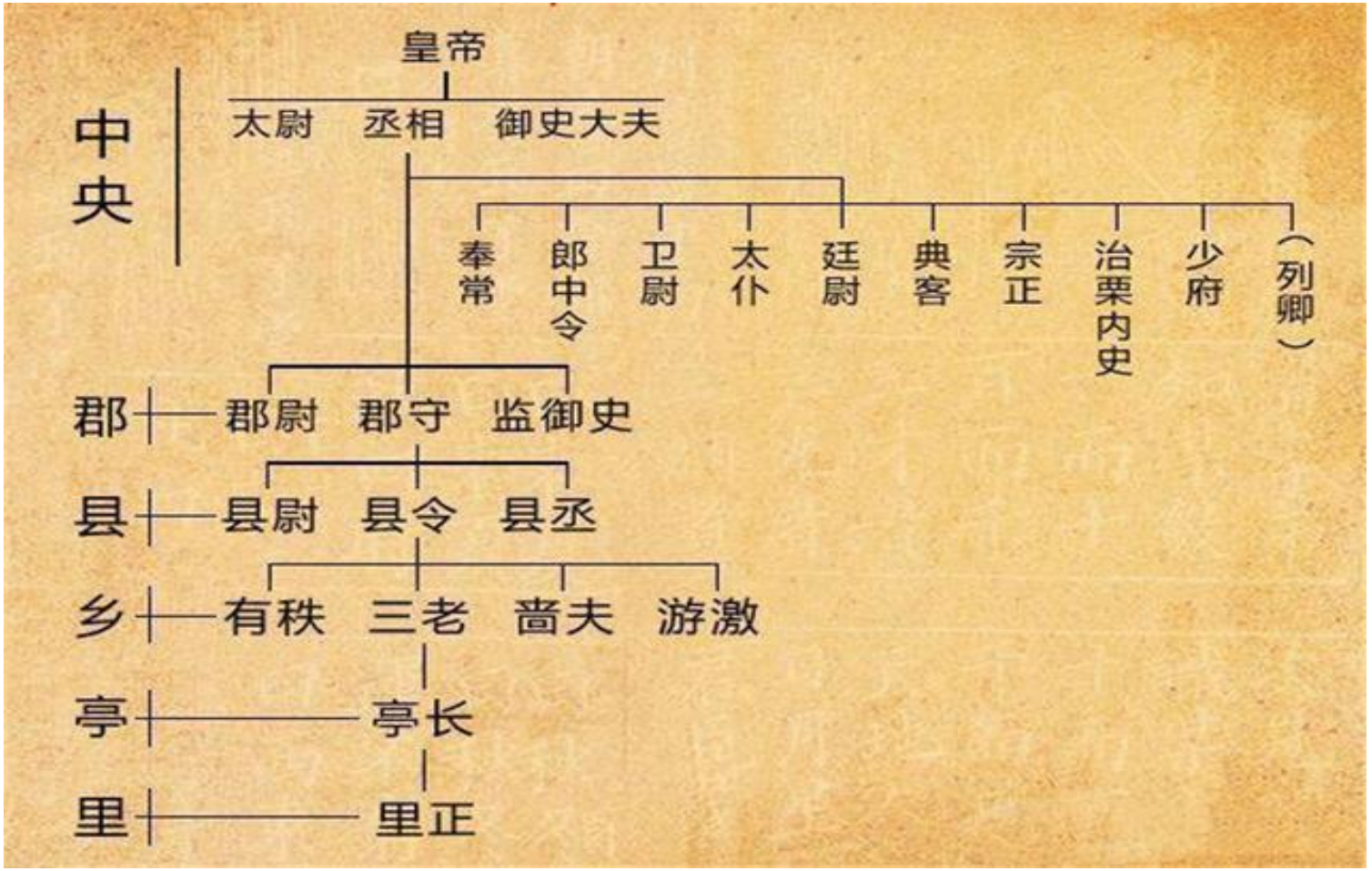
Divide-and-Conquer 分治

◘ Ancient wisdom of Divide-and-Conquer



- System of prefectures and counties (郡县制)

A system of local administration during the Spring and Autumn Period and the Qin Dynasty, which symbols finally accomplish of the change of Chinese early country to mature country.


皇帝
中央
太尉 丞相 御史大夫
奉常 郎中令 卫尉 太仆 廷尉 典客 宗正 治粟内史 少府 （列卿）
郡 郡尉 郡守 监御史
县 县尉 县令 县丞
乡 有秩 三老 啬夫 游激
亭 亭长
里 里正


前221年秦边界
政权部族界
今国界
长城
九原郡
河南地
阴山
渔阳郡
辽东郡
朝鲜
代郡
勃海
上郡
沙丘
临淄郡
陇西郡
北地郡
邯郸郡
琅邪郡
临洮
咸阳
颍川郡
大泽乡
东海
汉中郡
陈郡
会稽郡
羌
蜀郡
南郡
巴郡
黔中郡
长沙郡
庐江郡
夜郎
闽中郡
滇
灵渠
五岭
越

◘ Ancient wisdom of Divide-and-Conquer

“分而治之，逐个击破”

“治众如治寡，分数是也”

Divide-and-Conquer approach:

- **break** （分解）the problem into several subproblems that are similar to the original problem but smaller in size, **solve** （治理）the subproblems recursively, and then **combine** （合并）these solutions to create a solution to the original problem.
- ***recursive*** : to solve a given problem, they call themselves recursively one or more times to deal with closely related subproblems.

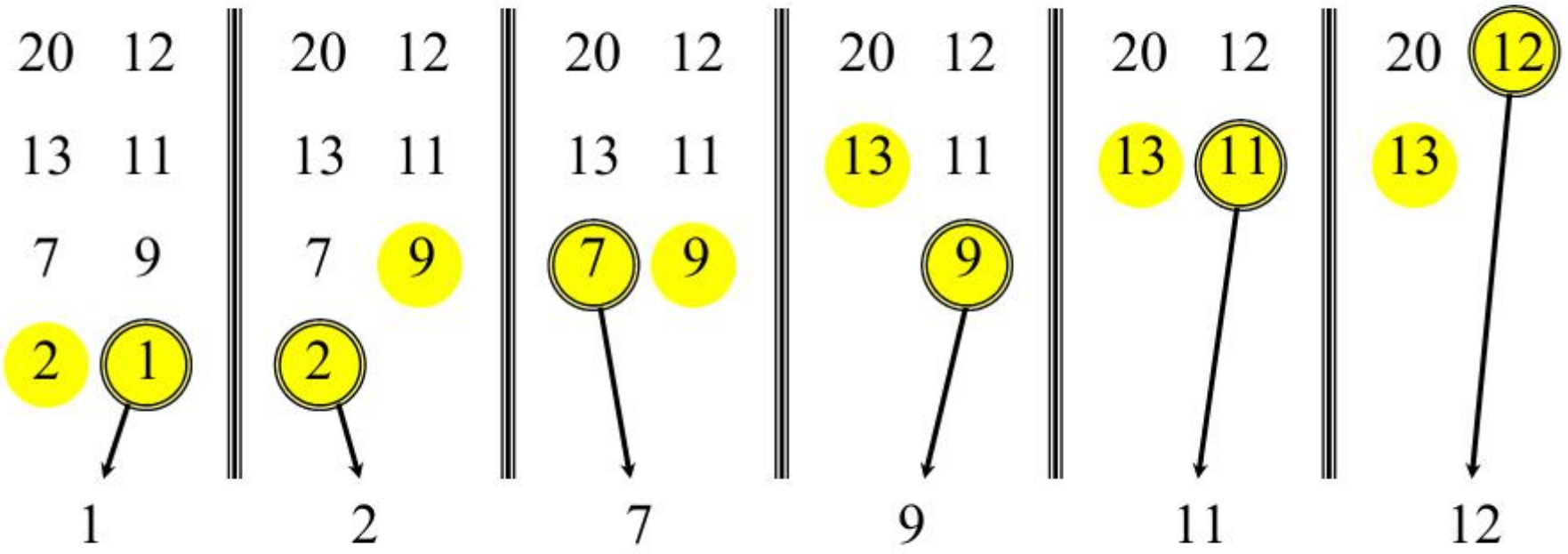
Sort wisdom of Divide-and-Conquer

MERGE-SORT $A[1 \ldots n]$

1. If $n = 1$, done.
2. Recursively sort $A[1 \ldots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \ldots n]$.
3. "***Merge***" the 2 sorted lists.

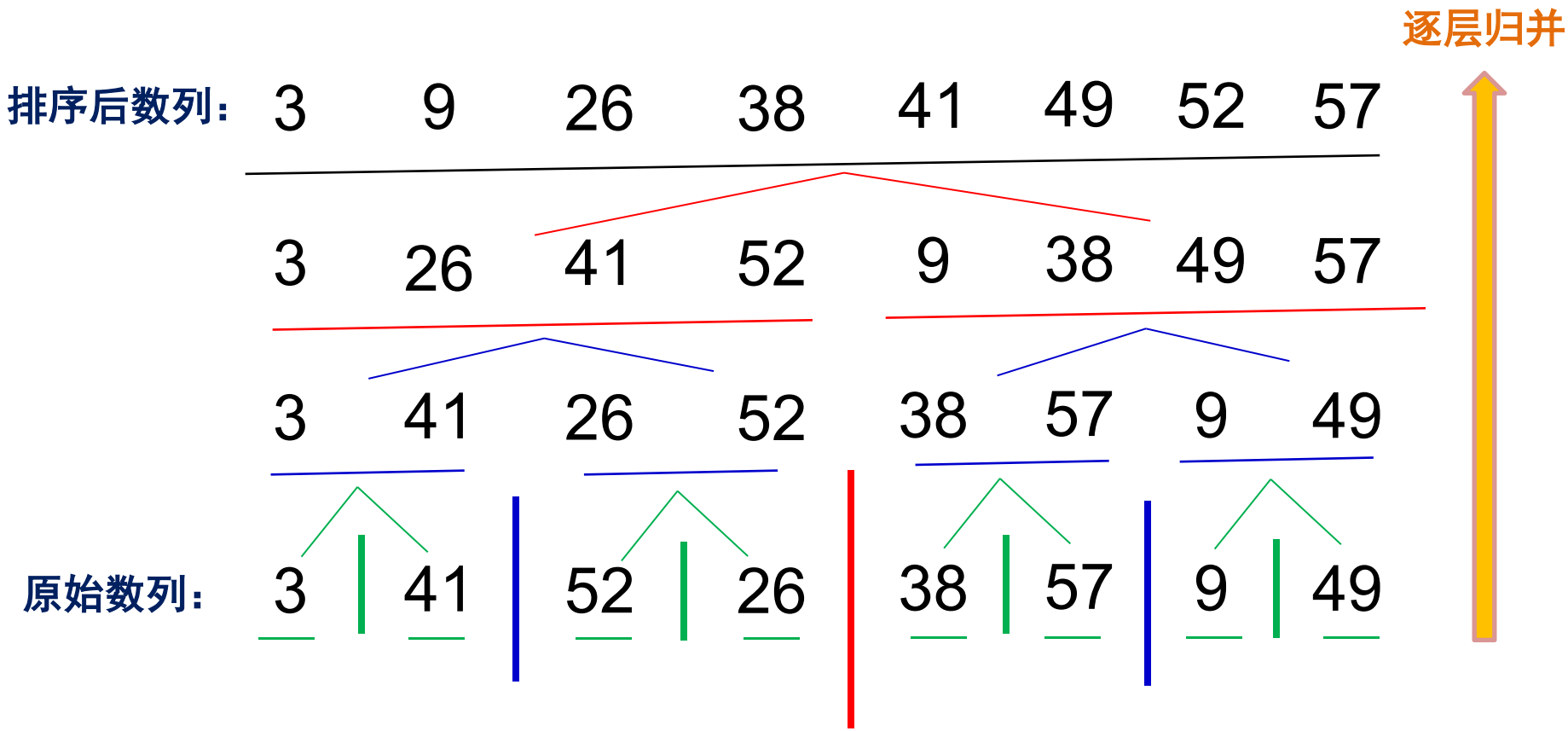
Key subroutine: **MERGE**

Merging two sorted arrays



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

归并排序数列A=（3,41,52,26,38,57,9,49）。



Analyzing merge sort

$T(n)$	MERGE-SORT $A[1 \ldots n]$
$\Theta(1)$	1. If $n = 1$, done.
$2T(n/2)$	2. Recursively sort $A[1 \ldots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \ldots n]$.
$\Theta(n)$	***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.

时间复杂度

Recurrence for merge sort

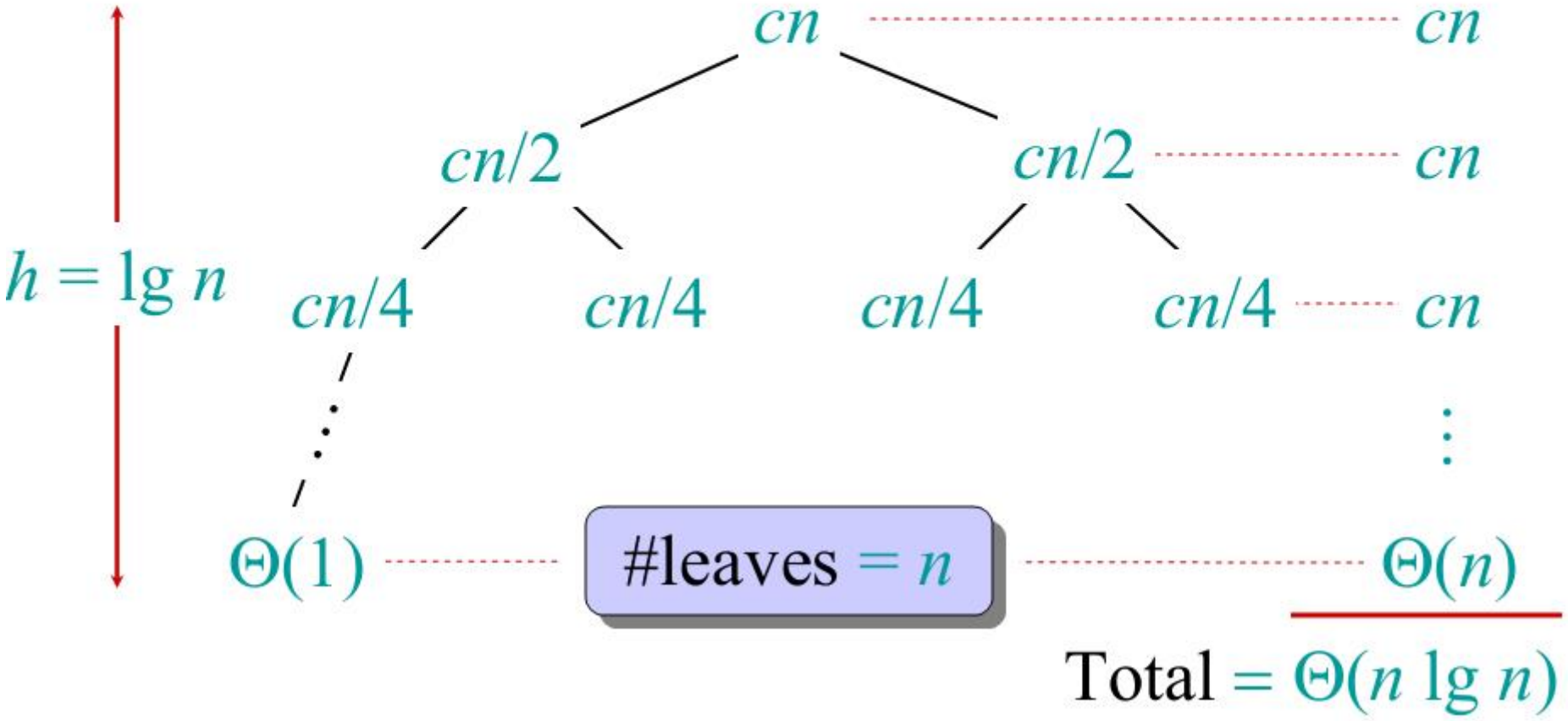
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

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.

note: in the worst case

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 . . n1 + 1] and R[1 . . n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

- The key operation of the merge sort algorithm is the merging of two sorted sequences in the "combine" step.
- We merge by calling **an auxiliary procedure** MERGE(A, p, q, r), where A is an array and p, q, and r are indices into the array such that $p \le q < r$.
- MERGE procedure takes time $\Theta(n)$, where $n = r - p + 1$ is the total number of elements being merged, and it works as follows.

Homework

- 1.The divide-and-conquer method consists three basic steps:_____, _____, _____.

- 2.In textbook, you saw how to solve the recurrence relation for MergeSort by drawing its recursion tree and adding up the total running time. Use the same method(Page 21) to solve the following recurrences — that is, get a tight bound of the form $T(n) = \Theta(f(n))$ for the appropriate function f. Show your work. If you wish, you may assume that n initially has the form $n = a^i$, for an appropriate constant a.
- 1). $T(n) = 3T(n/3) + 2n$
- 2). $T(n) = 4T(n/2) + \Theta(n)$

- 3. In order to determine whether a number is in the array, we can check the number of arrays one by one. But if it is a sorted array, there is a better way. Use the divide-and-conquer method to solve this problem, and we prefer you describe your algorithm by the pseudo code.

Thank You!

Q&A