湖南大學
HUNAN UNIVERSITY

第四章 排序

湖南大学信息科学与工程学院

目录

第一节 比较排序

第二节 线性时间排序

排序问题

输入： 数组A[1...n]

输出： 把数组A按升序排列并用B[1...n]表示

示例：A=[7,2,5,5,9.6]　B=[2，5，5，7，9.6]

怎样能更有效率呢？

为什么要排序

- 明显的应用

  -组织MP3库维护

  -电话目录

- 一旦项目被排序，问题就变得简单

  -找到中位数，或找到最接近的对

  -二分搜索，识别统计异常值

- 不明显的应用

  -数据压缩：排序发现重复项

  -计算机图形学：从前到后渲染场景

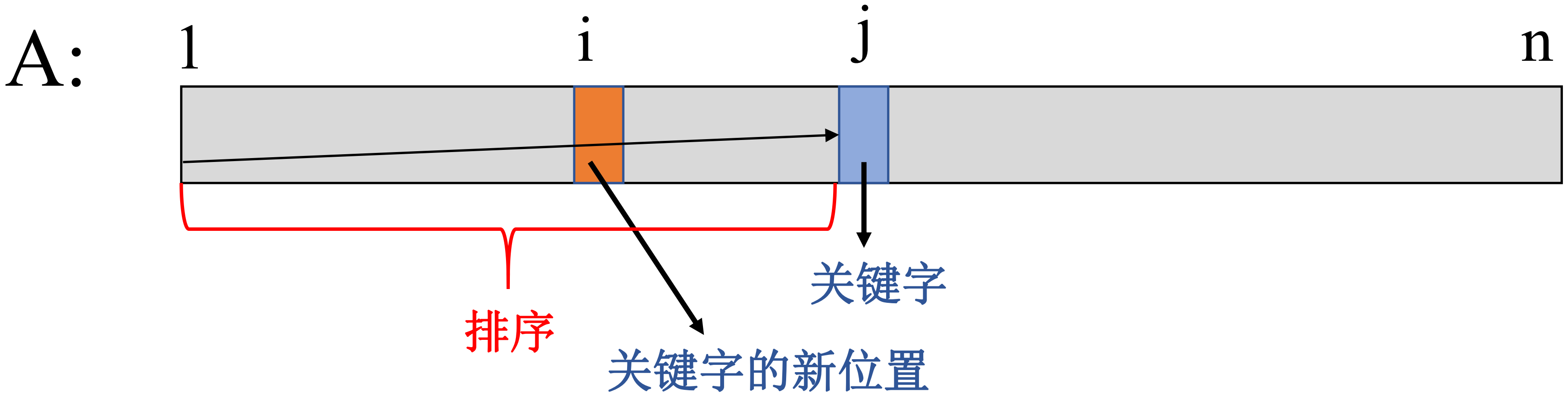
插入排序

INSERTION-SORT(A,n) 对数组A[1..n]进行操作

for j←2 to n

将关键字A[j]插入到已排序的子数组A[1..j-1]中，通过成对的键交换，使其下降到正确的位置

迭代j的说明



插入排序例子

8 2 4 9 3 6

8 2 4 9 3 6

2 8 4 9 3 6

8 2 4 9 3 6

2 8 4 9 3 6

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

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

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

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

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

	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
排序结果	2	3	4	6	8	9

插入排序例子

排序结果 2 3 4 6 8 9

运行时间？$O(n^2)$，因为需要对n^2个元素进行比较和交换。例如，当输入是A=[n, n-1, n-2, ..., 2, 1]时。

二分插入排序

BINARY-INSERTION-SORT(A,n) ▷ A[1..n]

for j ← 2 to n

将关键字A[j]插入到已排序的子数组A[1..j-1]中的正确位置，使用二分搜索找到合适的位置

二分搜索需要的时间是O(log n)。然而，插入后移动元素仍然需要O(n)的时间。

复杂性：O(n log n)次比较

$O(n^2)$次交换

归并排序

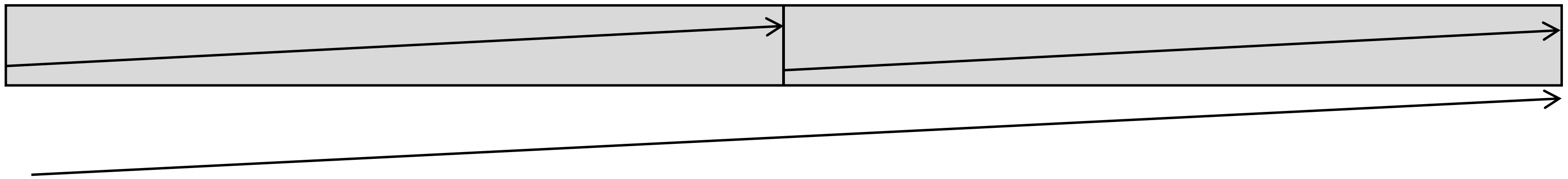
分治算法

归并排序A[1 . . n]

1、如果n=1，则完成（无需排序）

2、否则，递归地对A[1 . . n/2]和A[n/2+1 . . n]进行排序

3、“合并”两个已排序的子数组



关键子程序：归并

合并两个已排序的数组

20　12

13　11

7　9

2　1

合并两个已排序的数组

20 12

13 11

7 9

2 1

1

合并两个已排序的数组

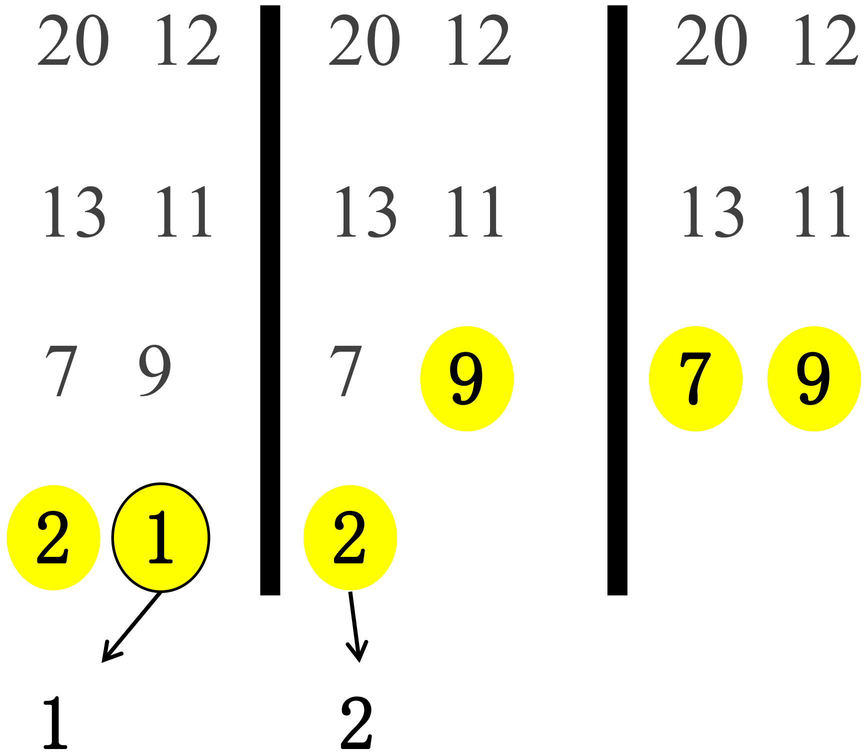
20	12	20	12
13	11	13	11
7	9	7	9
2	1	2	

1

合并两个已排序的数组

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

合并两个已排序的数组



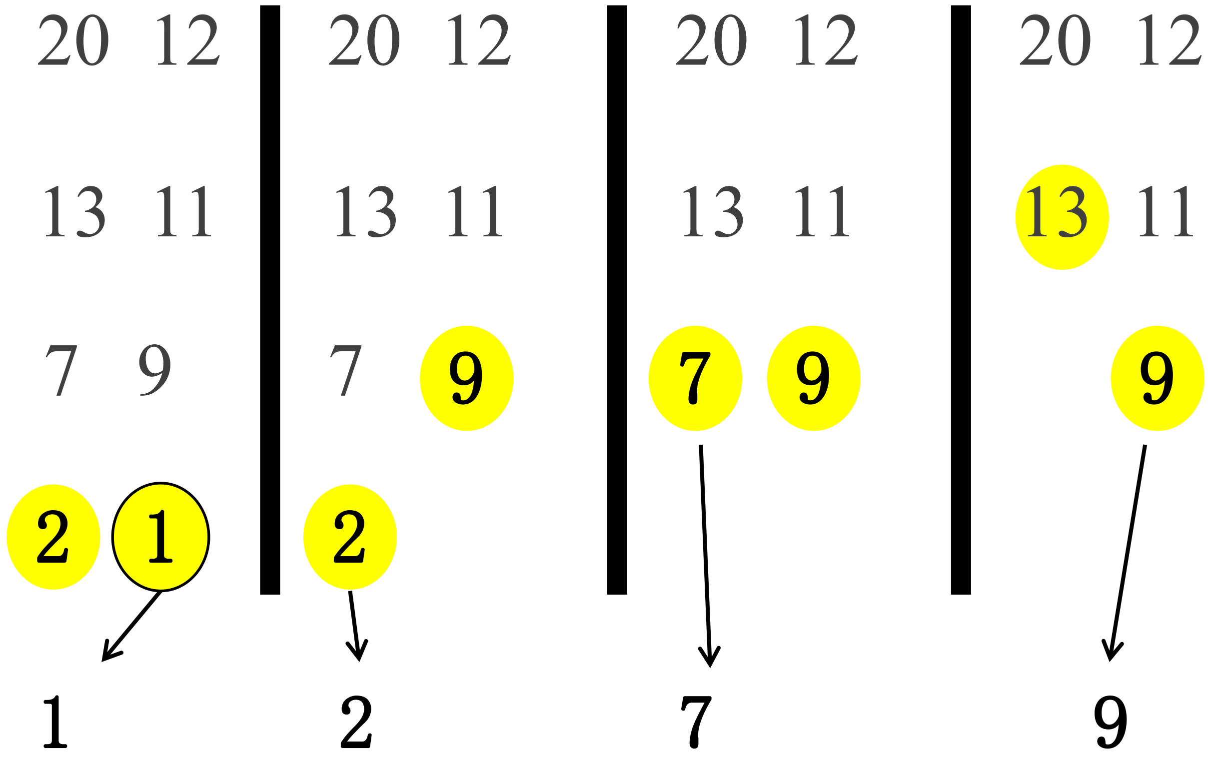
合并两个已排序的数组

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

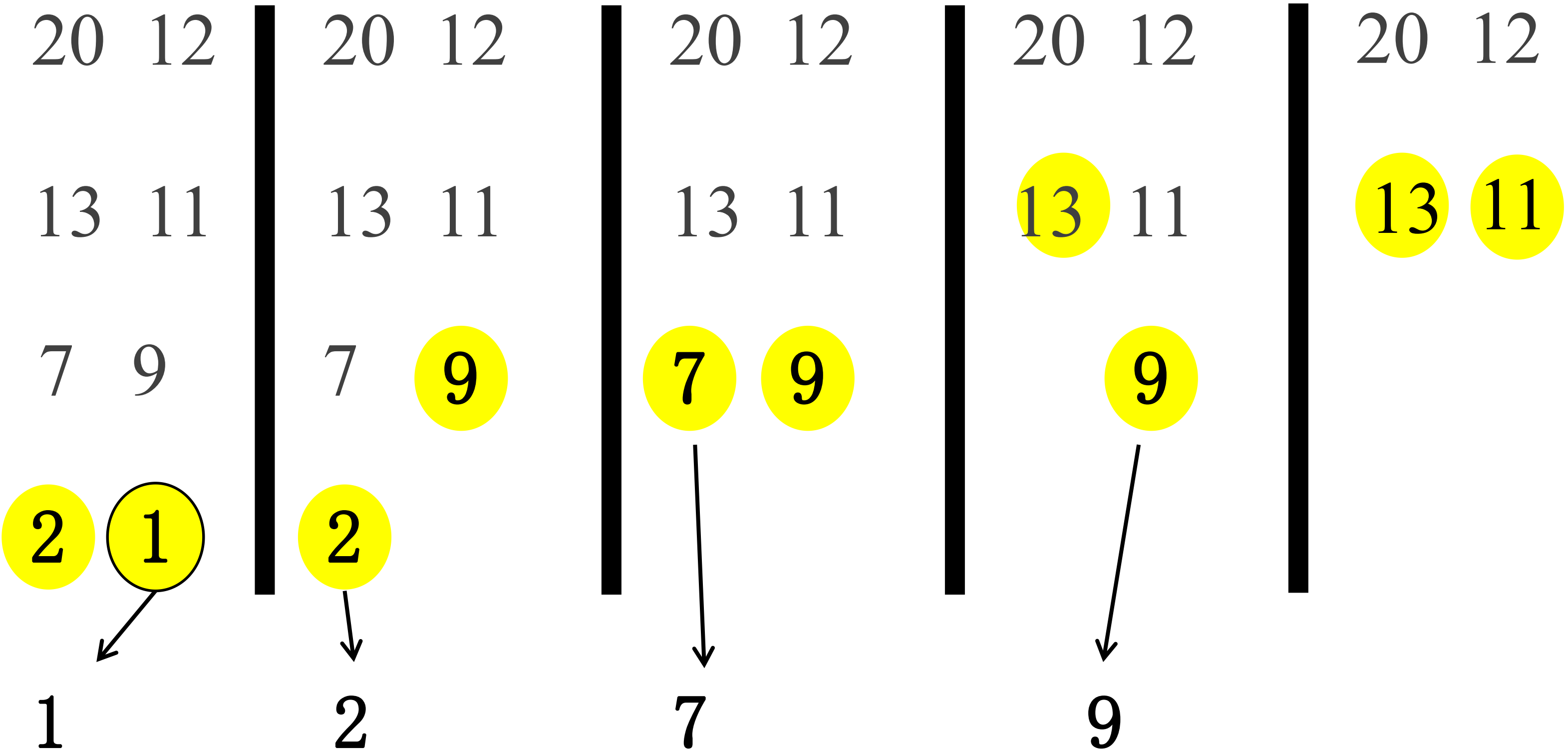
合并两个已排序的数组

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

合并两个已排序的数组

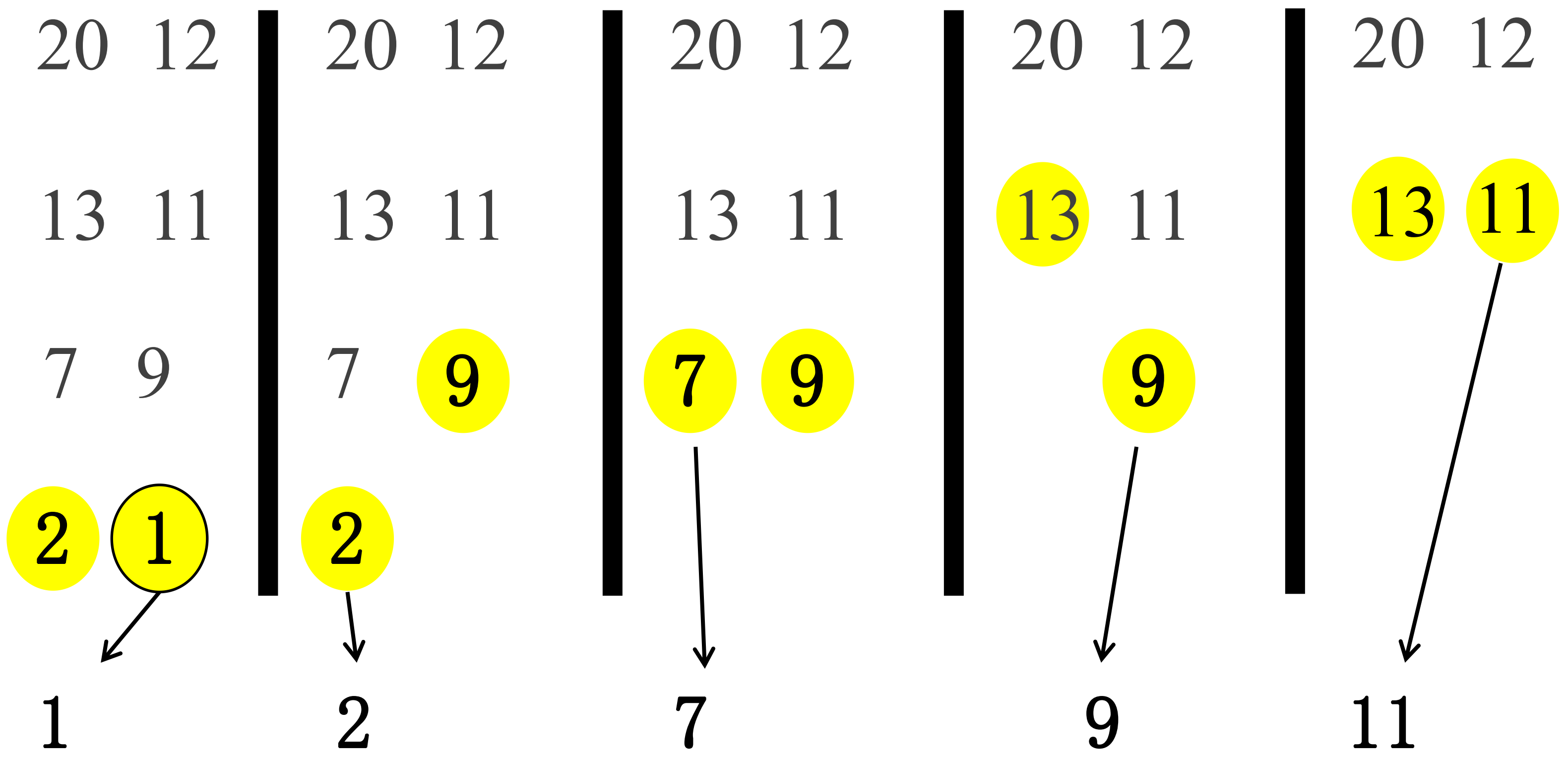


合并两个已排序的数组

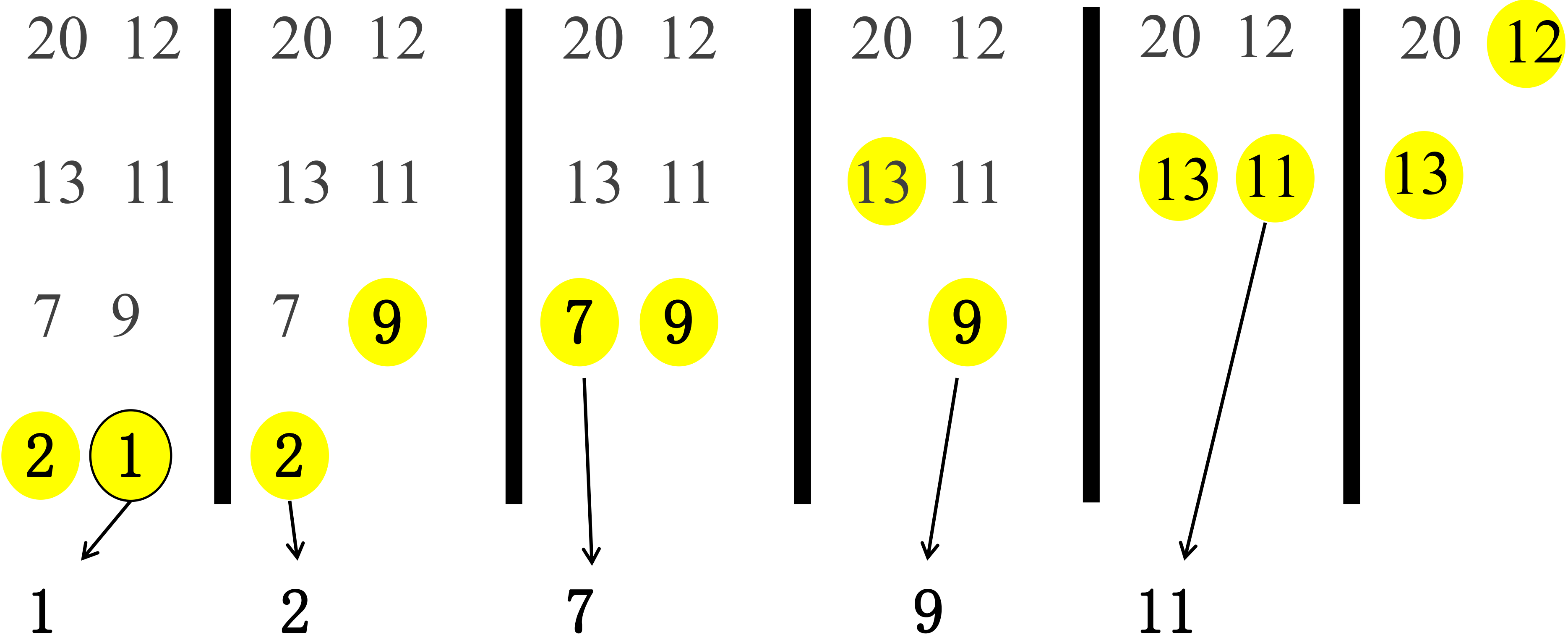


20 12
13 11
7 9
2 1
1
20 12
13 11
7 9
2
2
20 12
13 11
7 9
7
20 12
13 11
9
9
20 12
13 11

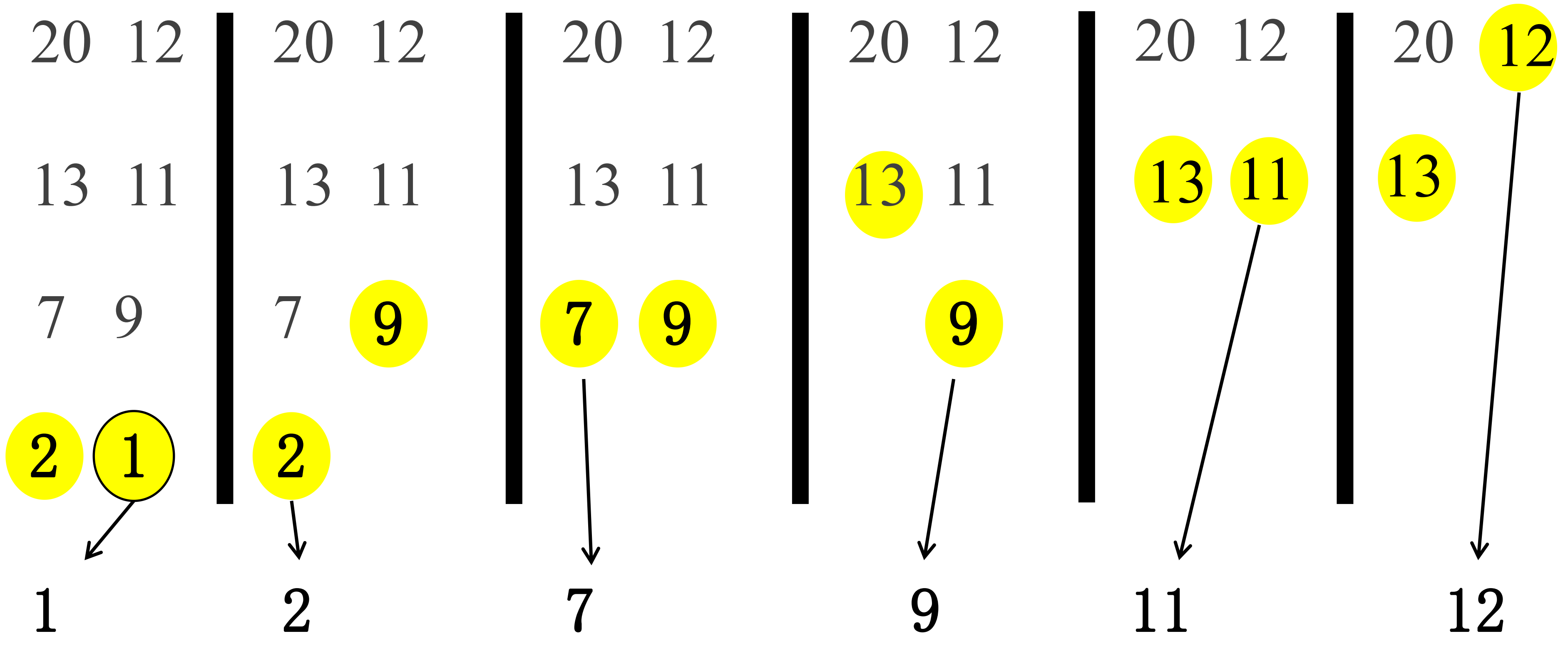
合并两个已排序的数组


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

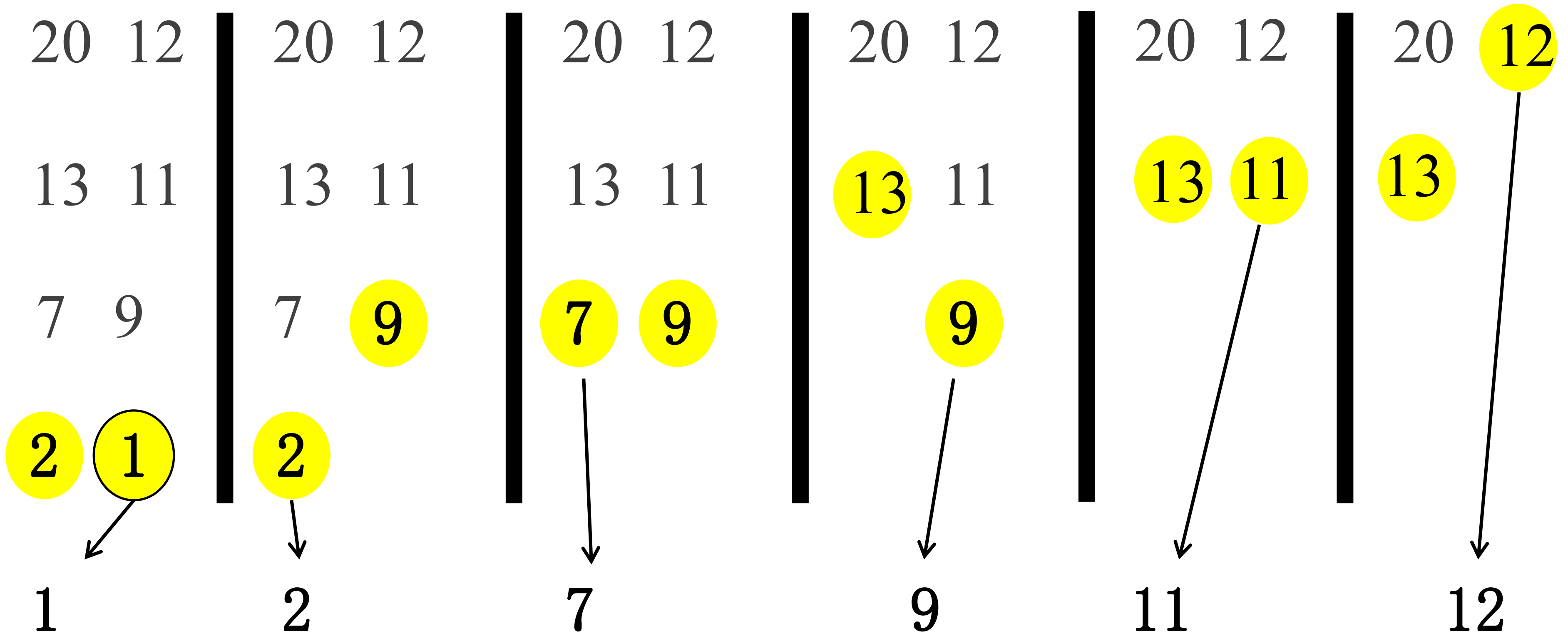
合并两个已排序的数组


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

合并两个已排序的数组

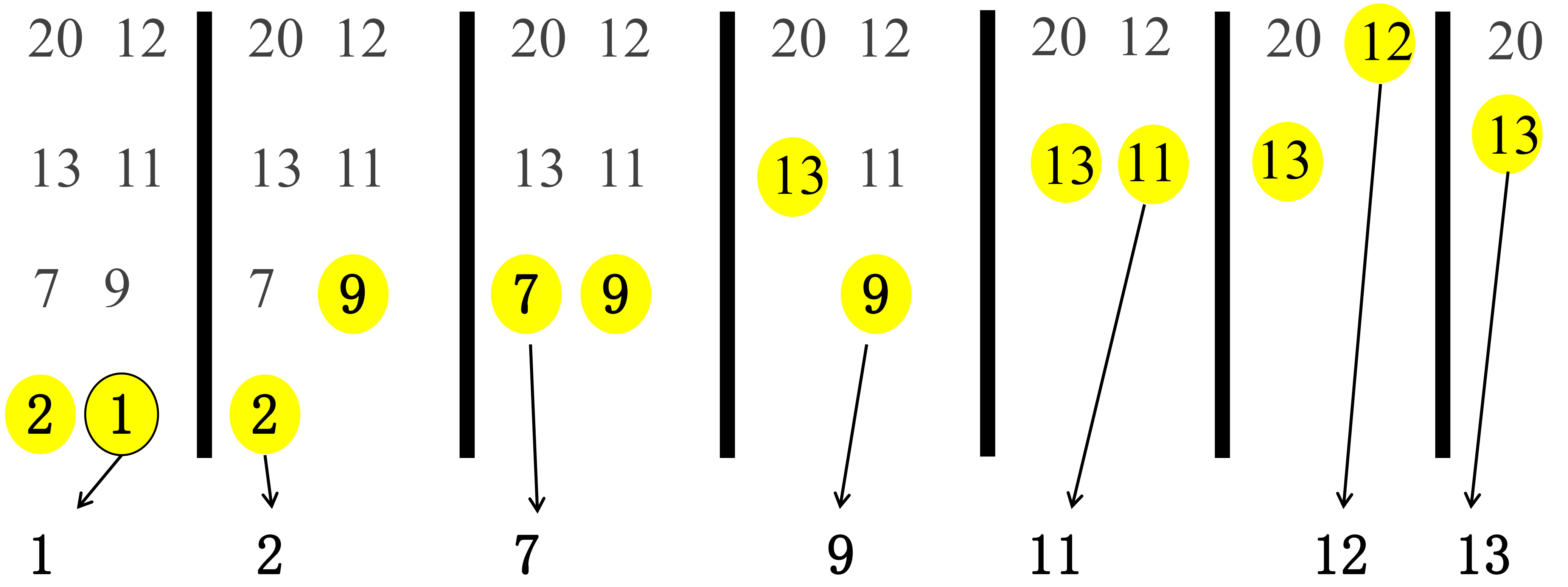


合并两个已排序的数组



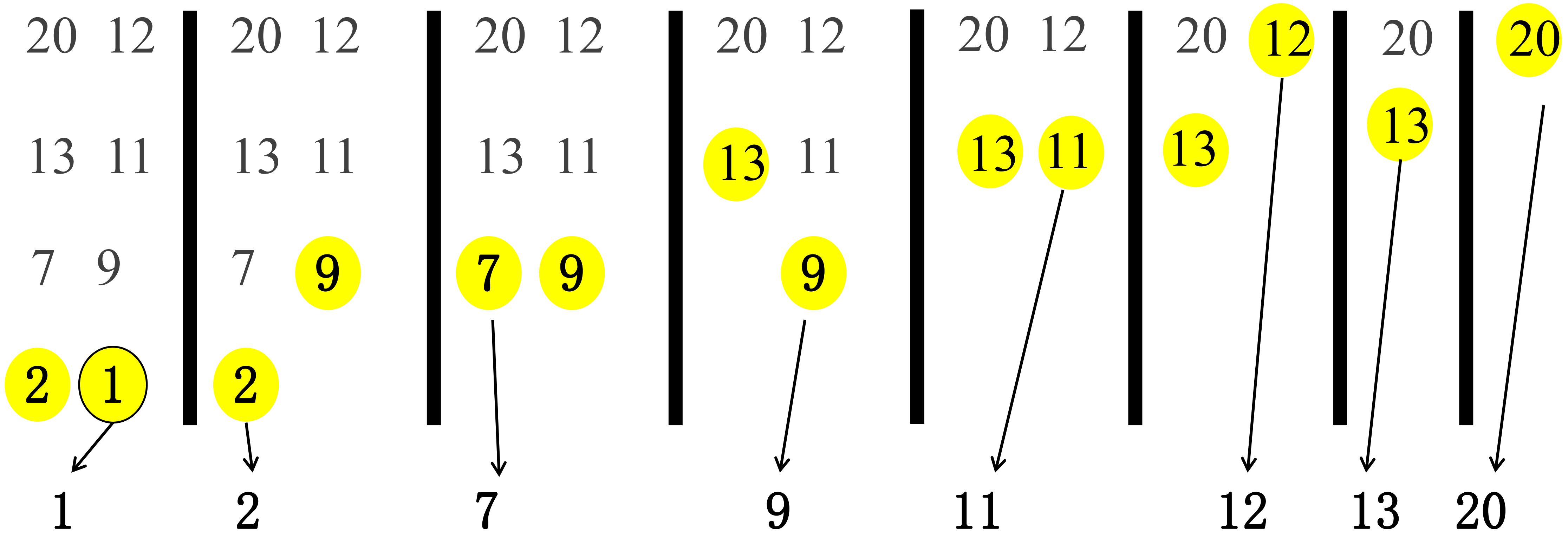
20 12
13 11
7 9
2 1
1
20 12
13 11
7 9
2
2
20 12
13 11
7 9
7
20 12
13 11
9
9
20 12
13 11
11
20 12
13
12

合并两个已排序的数组



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

合并两个已排序的数组



排序结果

这个算法需要O(n)的时间来合并两个大小为n的已排序数组（线性时间）

分析归并排序

归并排序A[1 . . n]

1.如果n=1，则完成

2.递归地对A[1..n/2]和A[n/2+1..n]进行排序

3.“合并”两个已排序的列表

T(n)

O(1)

2T(n/2)

O(n)

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

T(n) = ?

递归求解

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。

递归树

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。

$$T(n)$$

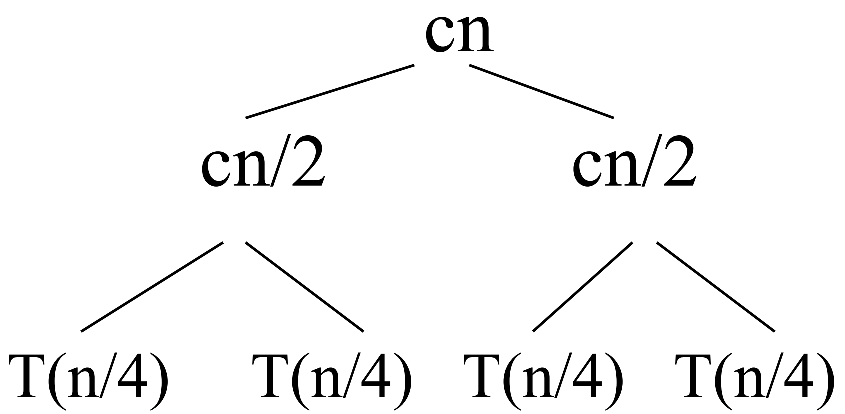
递归树

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。

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

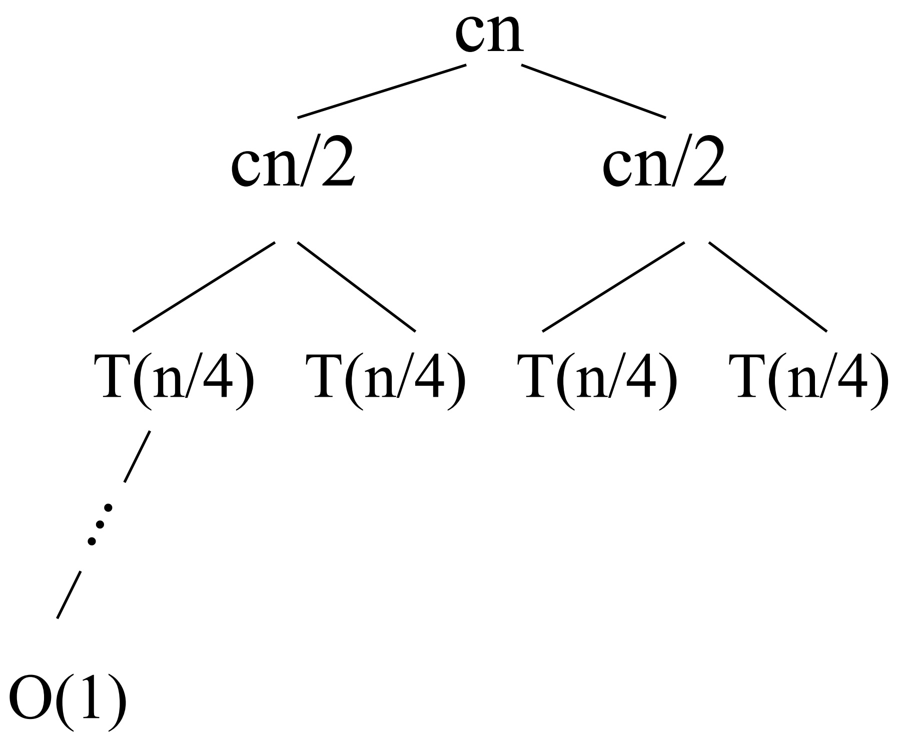
递归树

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。



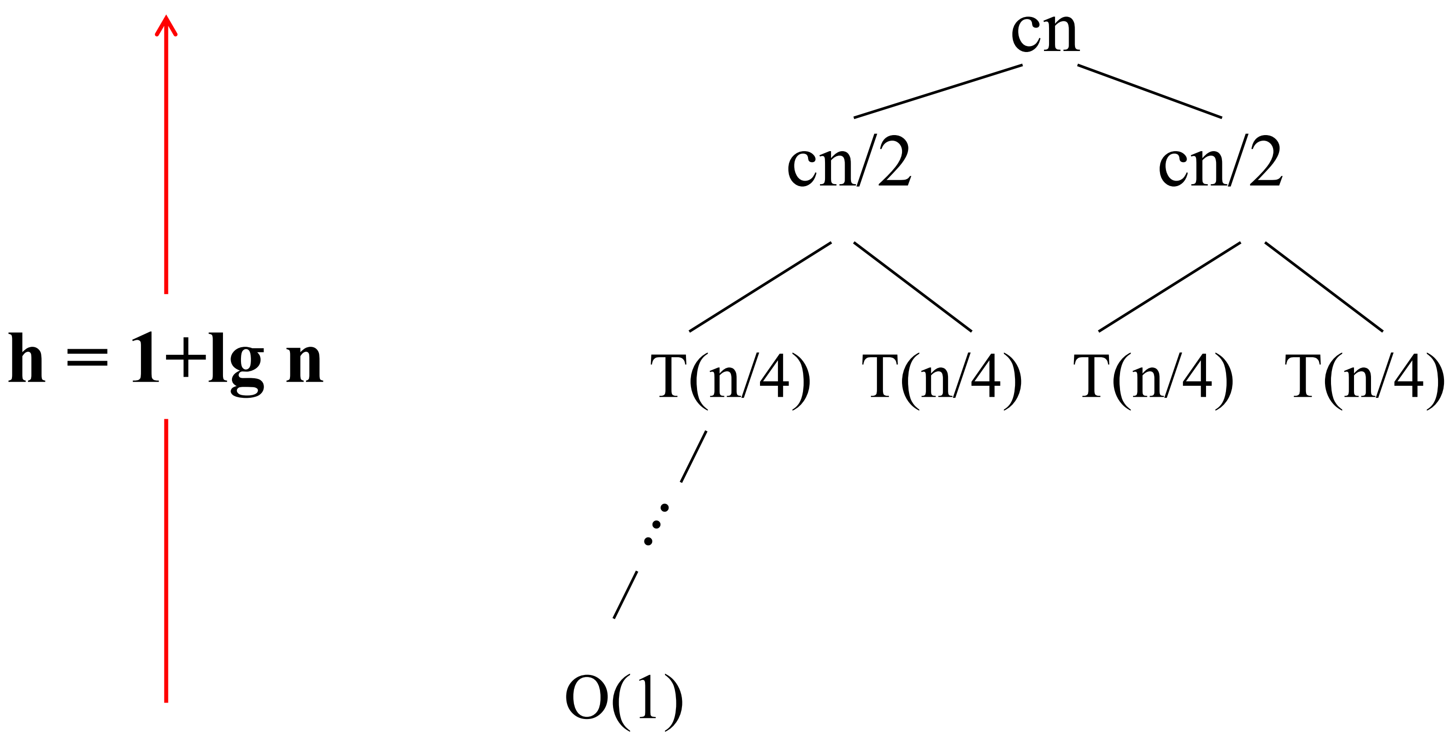
递归树

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。



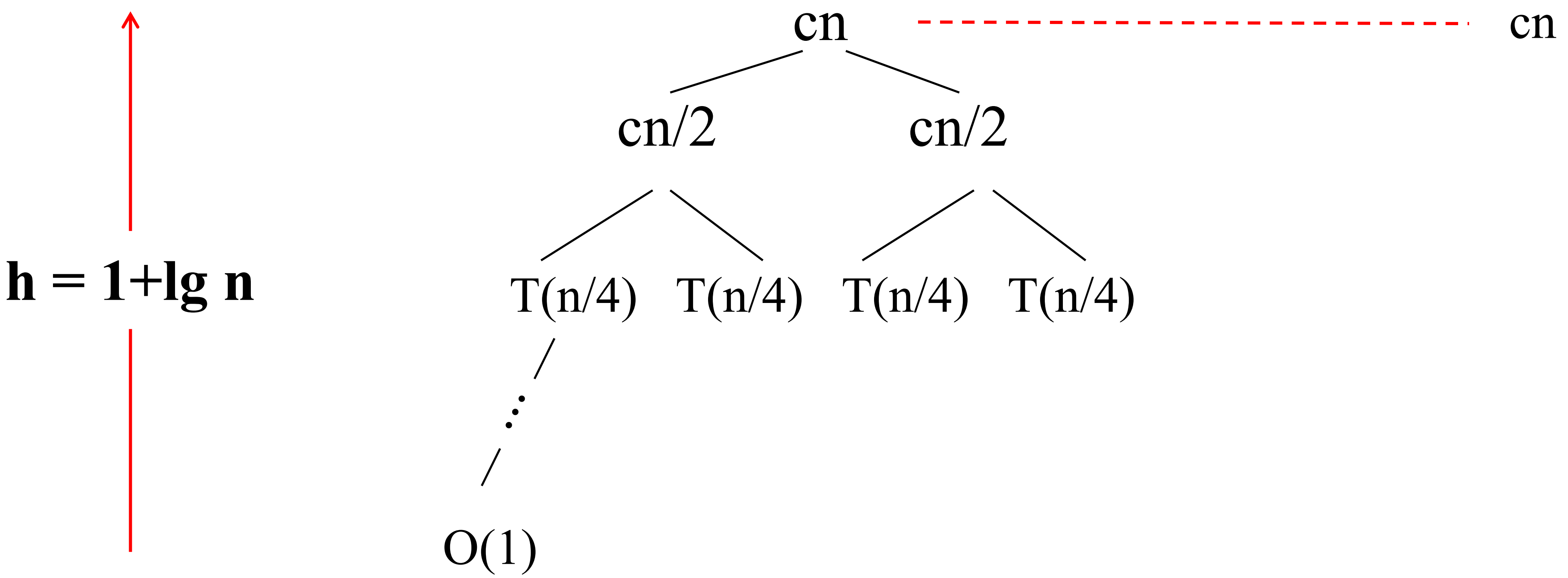
递归树

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。



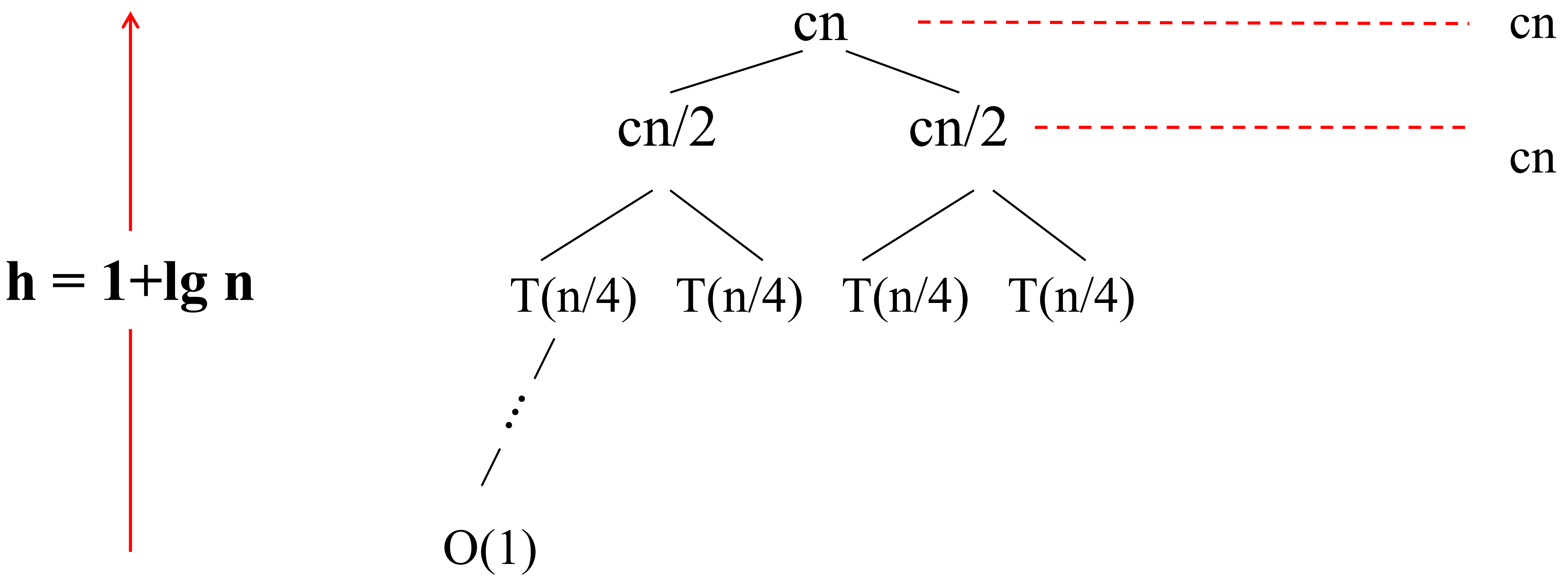
递归树

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。



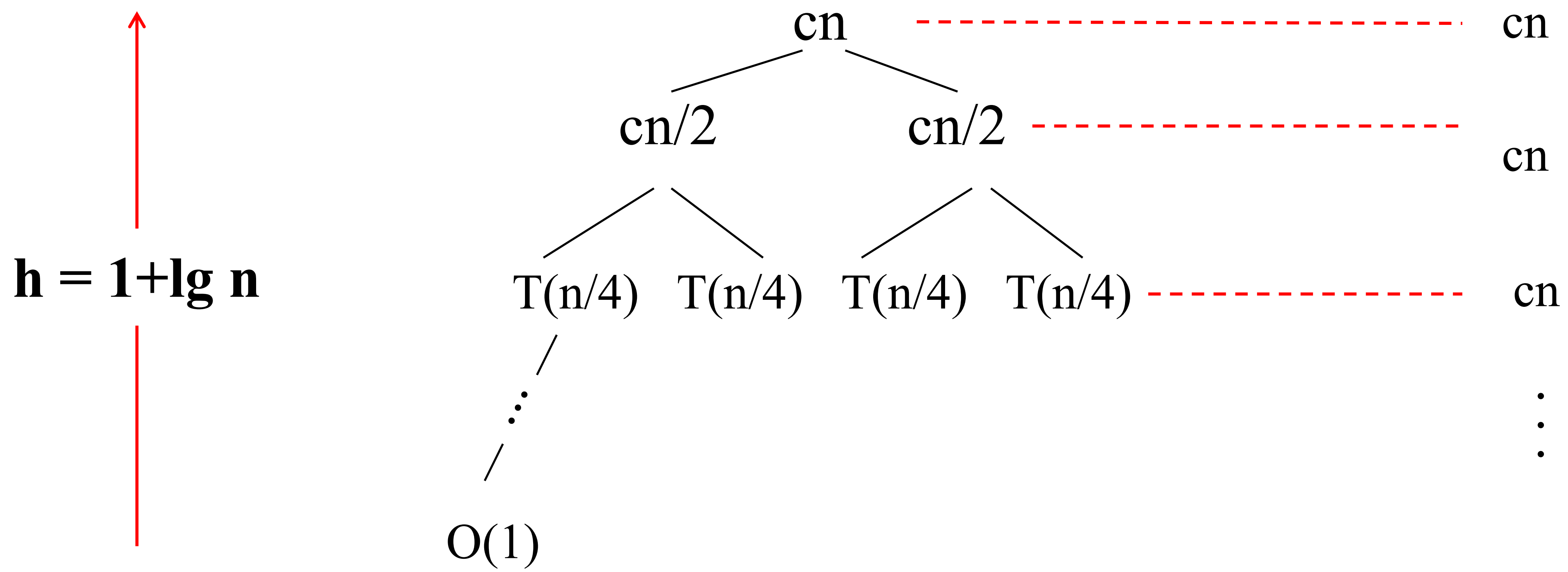
递归树

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。



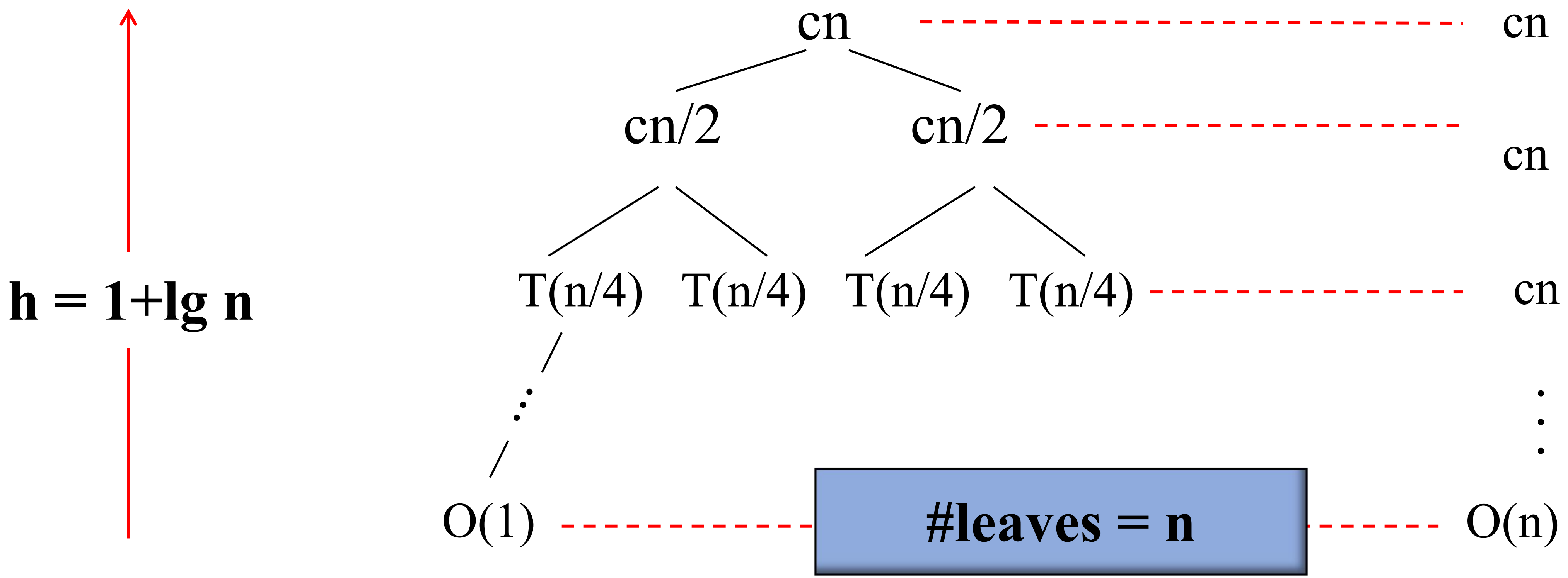
递归树

解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。

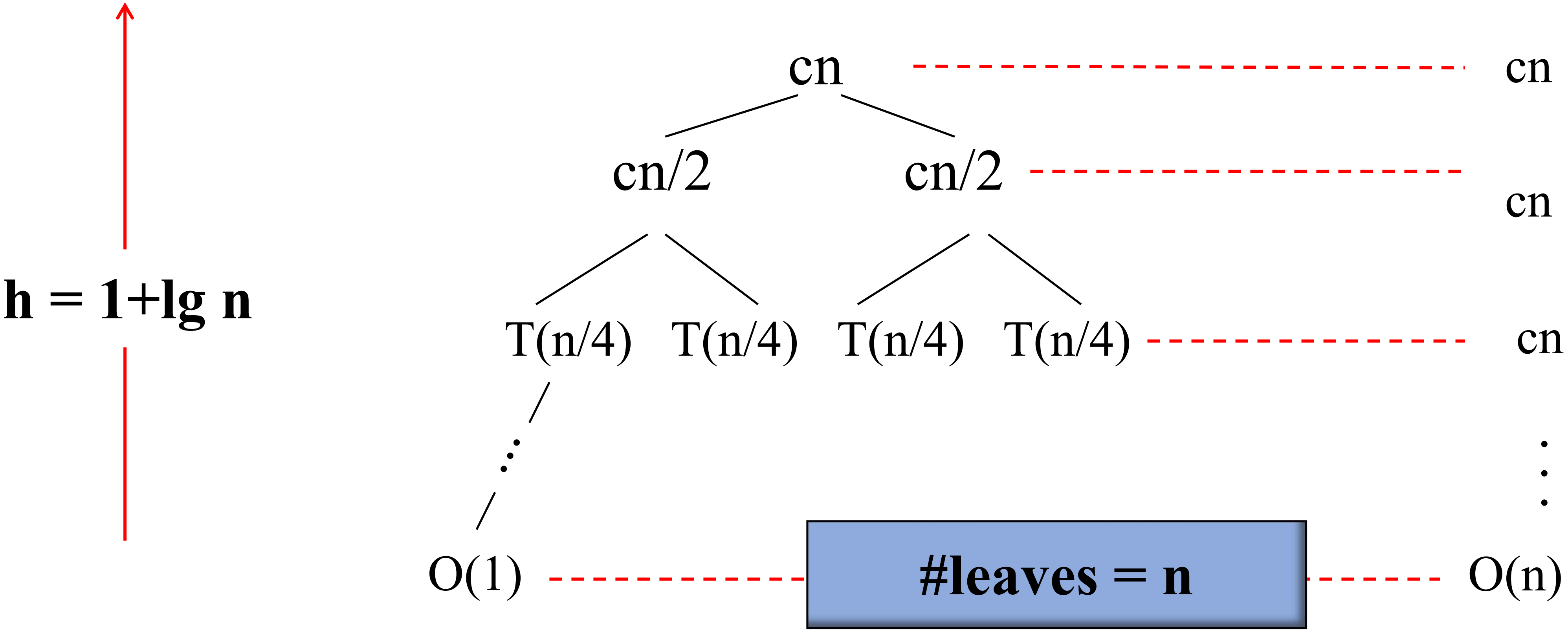


递归树

解决T(n)=2T(n/2)+cn，其中c>0是常数。

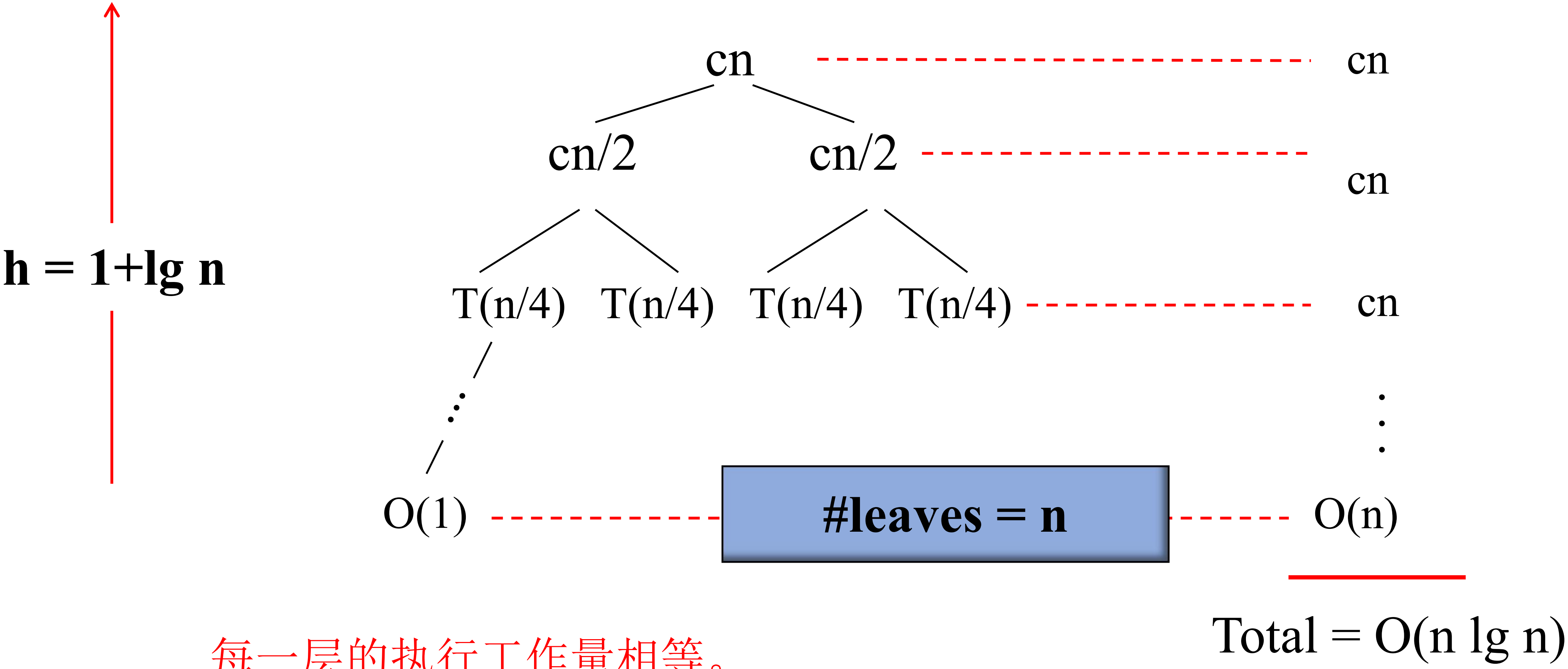


解决$T(n)=2T(n/2)+cn$，其中$c>0$是常数。



Total ?

解决T(n)=2T(n/2)+cn，其中c>0是常数。



每一层的执行工作量相等。

快速排序

- 划分：将数组围绕枢轴x划分为两个子数组

≤X	X	≥X

- 治：递归地对子数组进行排序
- 组合：合并两个子数组（这是简单的）

划分

选择A[p,…,q]中的元素p作为枢轴

所有在A[p+1, i]中的元素都小于x，所有在A[i+1, j]中的元素都大于x，所有在A[j+1, q]中的元素都是未知的

x	≤x	≥x	?

p　　　　i　　　　j　　　　q

时间复杂度为O(n)

例子

$x = 6$

6	10	13	5	8	3	2	11

i j

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

6	5	3	2	8	13	10	11

2	5	3	6	8	13	10	11

$\underbrace{2\ \ 5\ \ 3}_{\le x}\ \ \underset{x}{6}\ \ \underbrace{8\ \ 13\ \ 10\ \ 11}_{\ge x}$

排序子数组

2	5	3	6	8	13	10	11

2	5	3	6	8	13	10	11

2	3	5	6	8	10	11	13

2	3	5	6	8	10	11	13

2	3	5	6	8	10	11	13

排序完成

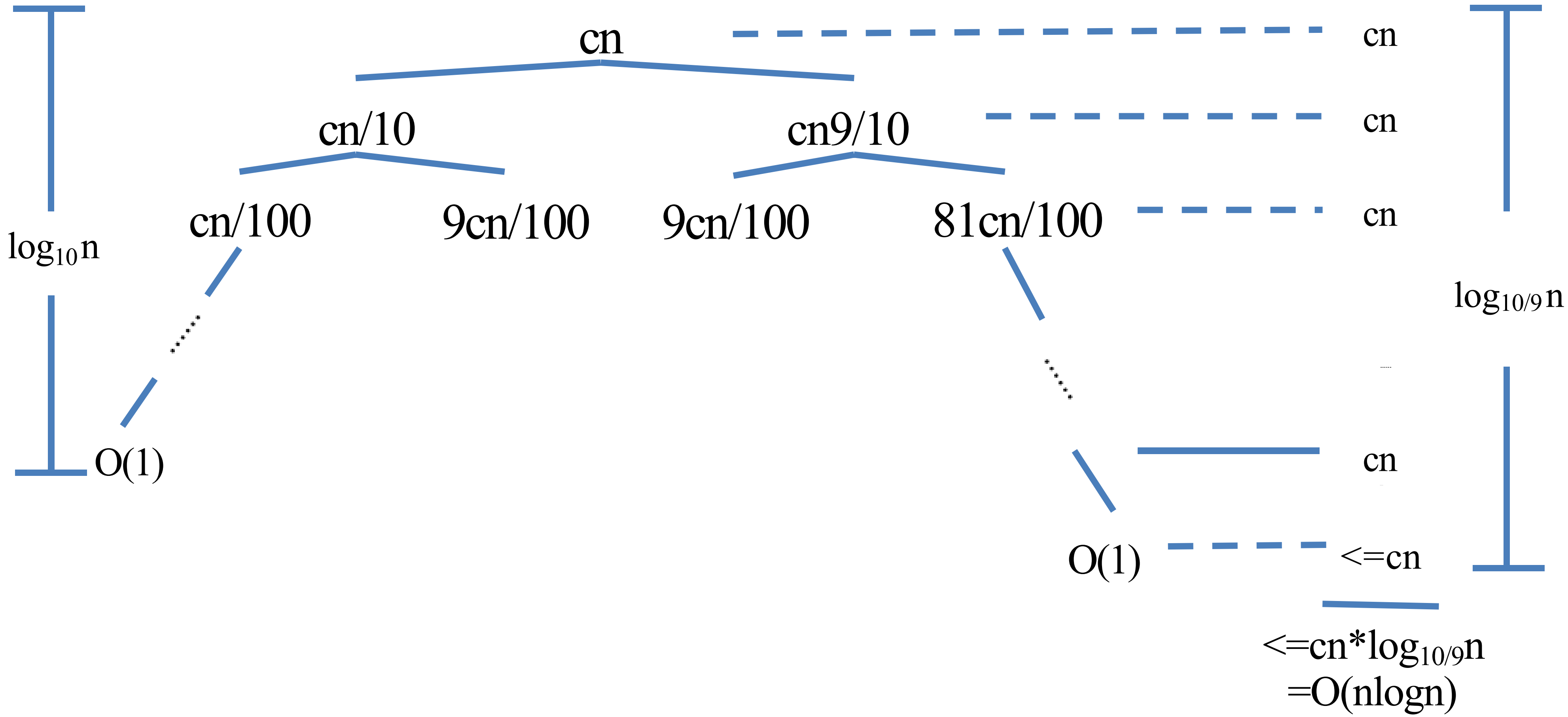
时间复杂度

- 最好情况下： $T(n)=2T(n/2)+ \Theta(n)= \Theta(n\log n)$
- 最坏情况下： $T(n)=T(0)+T(n-1)+\Theta(n) = \Theta(1)+ T(n-1)+\Theta(n) = T(n-1)+\Theta(n) = \Theta(n^2)$
- 其他情况：

第一个不平衡的情况

分割总是1:9

$$T(n)=T(n/10)+T(9n/10)+O(n)$$



第二个不平衡的情况

- 交替的最佳和最差情况
- 最好情况下： $B(n) = 2W(n/2)+\Theta(n)$
- 最坏情况下： $W(n) = B(n-1)+\Theta(n)$
- 其他情况：$B(n) = 2(B(n/2- 1) + \Theta(n/2))+ \Theta(n)2$

$$= 2B(n/2 - 1) + \Theta(n)2$$

$$= O(n\log n)$$

随机化快速排序

- 随机选择枢轴元素
- 运行时间不依赖于输入的运行顺序

分析

- 我们首先定义一个指示随机变量x_k
- 如果生成的是k:n-k-1的分割，那么$x_k=1$,否则$x_k=0$
- 然后，x_k的期望值是

$$E[x_k]=0*pr\{x_k=0\}+1*pr\{x_k=1\}=1/n$$

分析

- T(n) = T(0)+T(n-1)+Θ(n) if 0:n-1 split
- T(n) = T(1)+T(n-2)+Θ(n) if 1:n-2 split
- …
- T(n) = T(n-1)+T(0)+Θ(n) if n-1:0 split
- Then, T(n) = $\sum_{k=0}^{n-1} x_k \times)T)k(+ T)n-k-1(+ O)n(($

分析

$$\mathrm{E(T(n))} = \mathrm{E}(\textstyle\sum_{k=0}^{n-1} x_k \ \times)T)k(\ + \ T)n - k - 1(\ + \ O)n(((}$$

$$= \textstyle\sum_{k=0}^{n-1} \mathrm{E}(x_k \ \times)T)k(\ + \ T)n - k - 1(\ + O)n((($$

$$= \frac{1}{n} \textstyle\sum_{k=0}^{n-1} \mathrm{E}((T(k) \ + \ T(n - k - 1) \ + O(n)))$$

$$= \frac{1}{n}) \textstyle\sum_{k=0}^{n-1} \mathrm{E}(T(k)) \ + \ \sum_{k=0}^{n-1} E) \, T)n - k - 1((\ + \sum_{k=0}^{n-1} O)n(($$

分析

$$\mathrm{E(T(n))} = \frac{1}{n}\sum_{k=0}^{n-1}\mathrm{E}(T(k)) + \frac{1}{n}\sum_{k=0}^{n-1} E)\,T)n - k - 1((+ \frac{1}{n}\sum_{k=0}^{n-1} O)n($$

$$= \frac{2}{n}\sum_{k=0}^{n-1}\mathrm{E}((T(k)) + \frac{1}{n}\sum_{k=0}^{n-1} O)n($$

$$= \frac{2}{n}\sum_{k=0}^{n-1}\mathrm{E}((T(k)) + O(n)$$

吸收k=0和k=1，我们可以得到一个更简单的公式：

$$\mathrm{E(T(n))} = \frac{2}{n}\sum_{k=2}^{n-1}\mathrm{E}((T(k)) + O(n)$$

分析

为证明E(T(n))=O(nlogn)，下考虑下述引理

$$\sum_{k=2}^{n-1} klogk \le \frac{1}{2}n^2 \times logn$$

分析

定理： 对于 E(T(n)) $= \frac{2}{n}\sum_{k=2}^{n-1} \mathrm{E}(T(k)) + O(n)$，我们可以通过归纳法证明E(T(n))=O(nlogn)

证明： 数学归纳，基础步：n=1的时候，显然成立；

归纳步：$n \leq m$的时候都成立，然后证明n=m的时候成立

$$\mathrm{E(T(n))} = \frac{2}{n}\sum_{k=2}^{n-1} \mathrm{E}((T(k)) + O(n) = \frac{2}{n}\sum_{k=2}^{n-1} O)nlogn(+ O)n($$

$$\leq c_1 \frac{2}{n}\sum_{k=2}^{n-1} klogk + c_2 n \leq c_1 \times \frac{2}{n} \times \frac{1}{2} n^2\, logn + c_2 n = c_1 nlogn + c_2 n$$

=O(nlogn)

比较排序

我们迄今为止看到的所有排序算法都是比较排序：它们只使用比较来确定元素的相对顺序。

我们看到过的最好的运行时间是O(nlogn)。

O(nlogn)是我们能做到的最好吗？

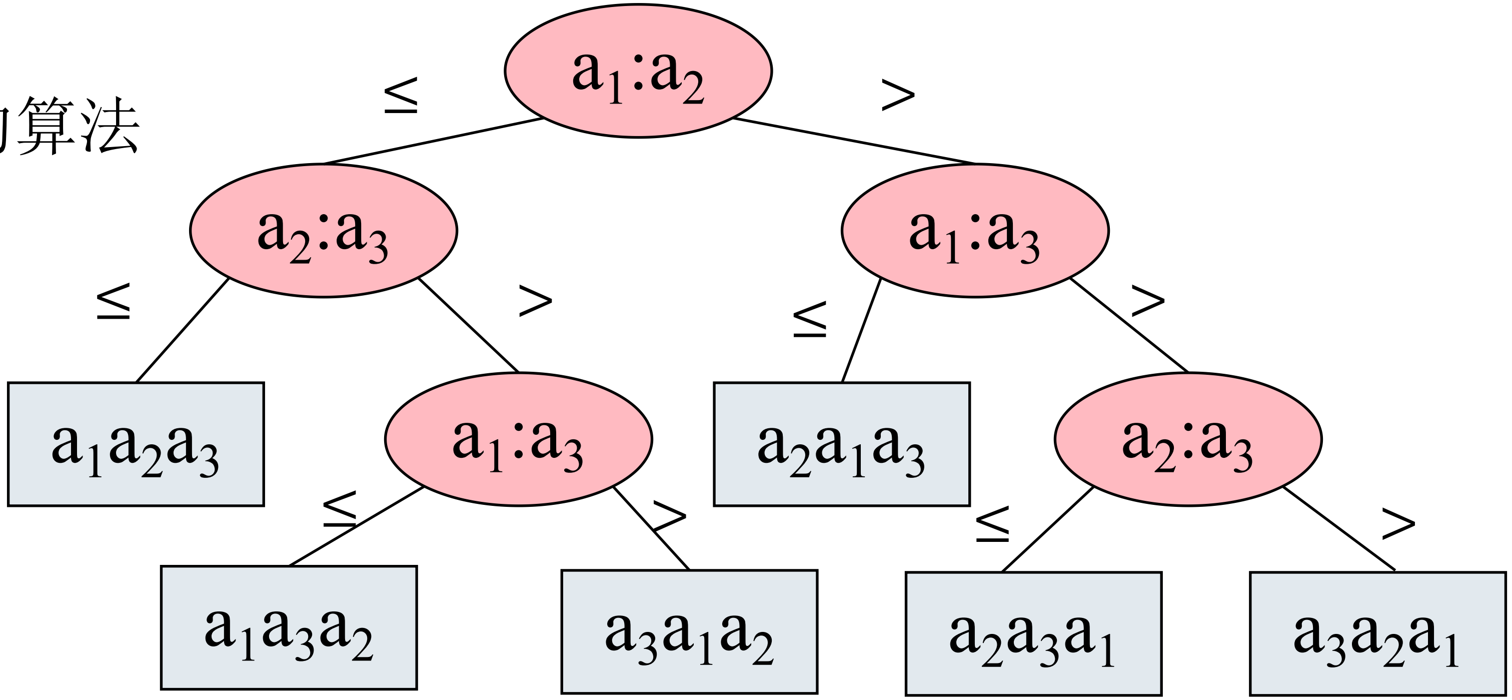
决策树

对n个字符<$a_1,a_2,\ldots,a_n$>排序的算法

节点是两个元素的比较，

例$a_i:a_j$

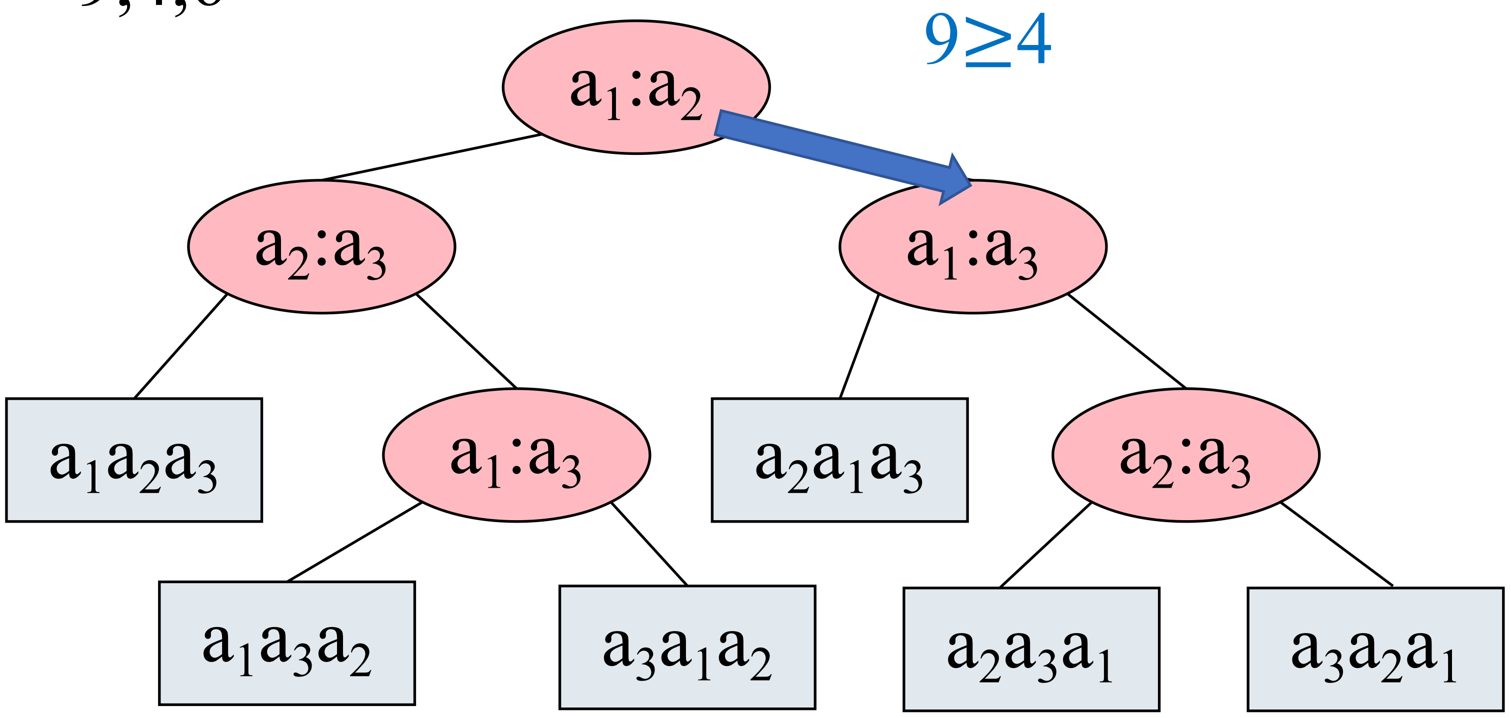
分支走向由比对结果决定



叶节点是标记好的排序，对应的是排序的结果

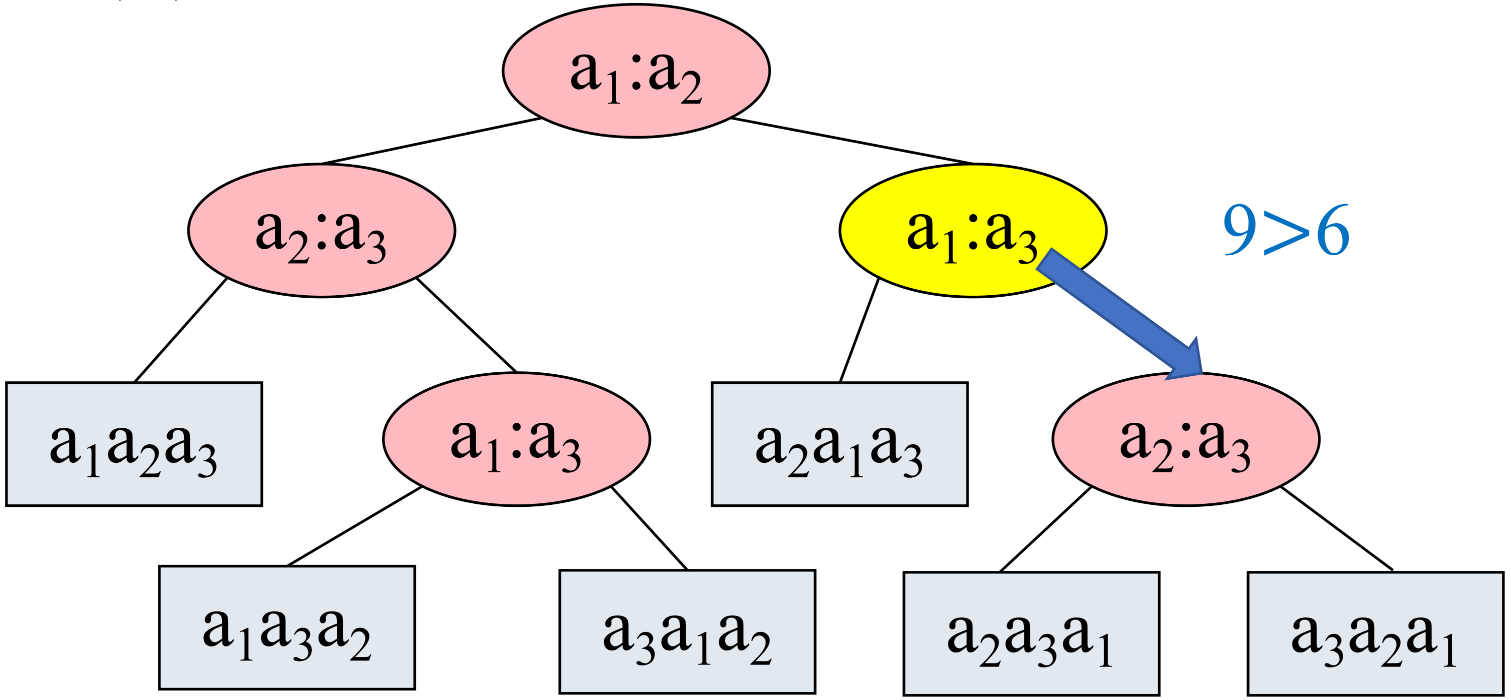
决策树例子

Sort<a_1, a_2, a_3> = <9,4,6>



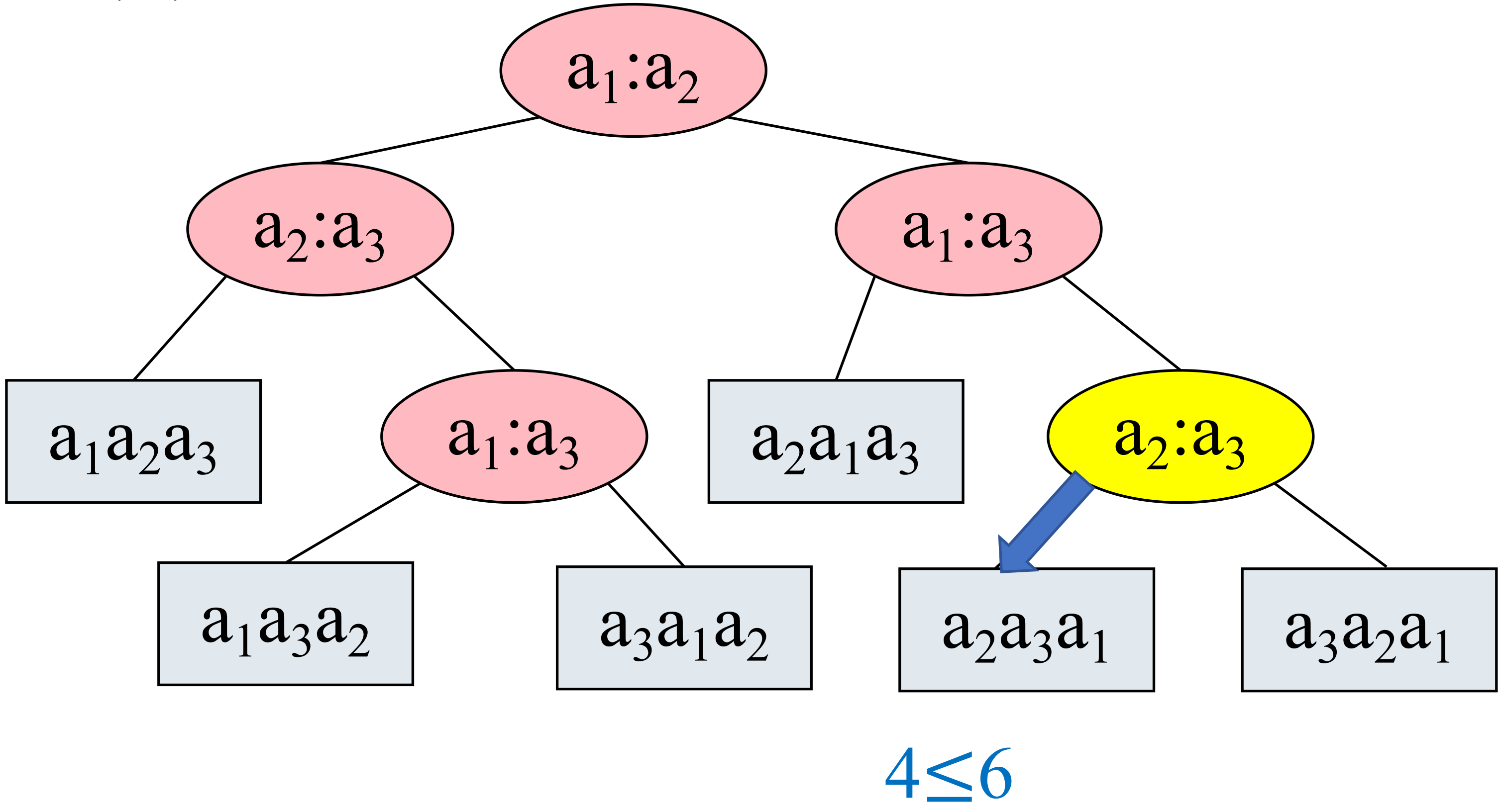
决策树例子

Sort<a_1, a_2, a_3> = <9,4,6>



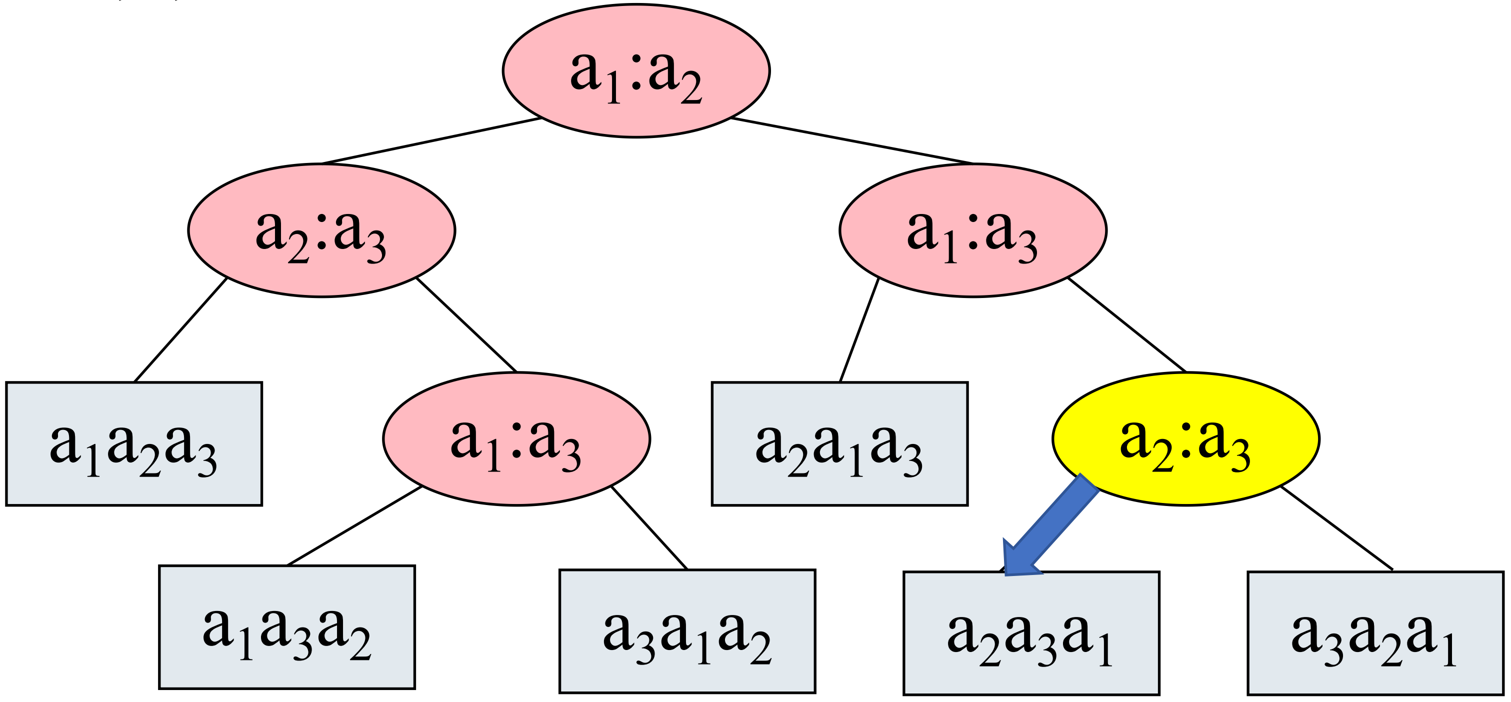
决策树例子

Sort<a_1, a_2, a_3> = <9,4,6>



决策树例子

Sort$\langle a_1, a_2, a_3 \rangle = \langle 9,4,6 \rangle$



$4 \leq 6 \leq 9$

决策树模型

一个决策树可以模拟任何比对排序的执行

- 输入的大小为n
- 一个从跟节点到叶节点的路径表示这个算法可能执行的一系列比对的跟踪
- 算法的运行时间=路径的长度
- 算法的最差情况下的运行时间=树的高度

决策树排序的下界

定理：任何用于n个元素的决策树的高度必须满足Ω(nlogn)(≥)。

证明：首先，由于存在n!种可能的排列，因此树必须包含(≥)n!个叶子节点。一个高度为h的二叉树最多有≤2^h个叶子节点。为了能够对其进行排序，必须满足以下条件：

$$2^h \geq n!$$

$$h \geq \log(n!)$$

$$\geq \log((n/e)^n)$$

$$= n \log n - n \log e$$

$$= \Omega(n \log n)$$

目录

线性时间排序

计数排序：元素之间没有比对

- 输入数组的值在1-k之间
- 输出数组B是对数组A的排序
- 辅助存储空间，长度为k数组C
- 运行时间为O(n+k)

计数排序

n=5,k=4

	1	2	3	4	5
A:	4	1	3	4	3

B:					

	1	2	3	4
C:				

数组C的索引对应的数组A

中每一个可能的键值

循环1：初始化

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	0	0	0	0

初始化数组C的每一个元素值为0

for i $\leftarrow$ 1 to k

do C[i] $\leftarrow$ 0

循环2：计算频率

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	0	0	0	1

C[i]=数组A中键值为i的元素个数,i=4

```
for j ⟵ 1 to n
   do C[ A[j] ] ⟵ C[ A[j] ] + 1      ▶   C[ i ] = |{key = i}|
```

循环2：计算频率

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	0	1

C[i]=数组A中键值为i的元素个数,i=1

for j ⟵ 1 to n

do C[A[j]] ⟵ C[A[j]] + 1 ▶ **C[i] = |{key = i}|**

循环2：计算频率

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	1	1

C[i]=数组A中键值为i的元素个数,i=3

for j ⟵ 1 to n

do C[A[j]] ⟵ C[A[j]] + 1 ▶ **C[i] = |{key = i}|**

循环2：计算频率

A:

1	2	3	4	5
4	1	3	4	3

B:

1	2	3	4	5

C:

1	2	3	4
1	0	1	2

C[i]=数组A中键值为i的元素个数,i=4

for j ← 1 to n

do C[A[j]] ← C[A[j]] + 1 ▶ **C[i] = |{key = i}|**

循环2：计算频率

A:

1	2	3	4	5
4	1	3	4	3

B:

C:

1	2	3	4
1	0	2	2

C[i]=数组A中键值为i的元素个数,i=3

for j ⟵ 1 to n

do C[A[j]] ⟵ C[A[j]] + 1 ▶ **C[i] = |{key = i}|**

A parenthesis:快速完成

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	2	2

B:

对频率数组C从头到尾走一遍，我们可以排序好的键值放到输出数组B中来。

for j ⟵ 1 to n

do C[A[j]] ⟵ C[A[j]] + 1

▶ **C[i] = |{key = i}|**

A parenthesis:快速完成

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	2	2

B:

1				

对于C[i]，数组A中键值为i的元素数目为C[i]!对于C[1]，数组A中键值为1的元素数目为1!

A parenthesis:快速完成

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	2	2

B:

1				

对于C[2]，数组A中键值为2的元素数目为0!

A parenthesis:快速完成

A:

1	2	3	4	5
4	1	3	4	3

B:

1	3	3		

C:

1	2	3	4
1	0	2	2

对于C[3]，数组A中键值为3的元素数目为2!

A parenthesis:快速完成

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:	1	3	3	4	4

对于C[4]，数组A中键值为4的元素数目为2!

B是排好了序，但只对键值排了序，没有对A中的元素排序，因为A中元素除了键值还有其他的数据，这里B只有键值的排序而没有对A中元素的交换！

循环3：从频率到累积频率……

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C′:	1	1	2	2

C[i]=数组A中键值≤i的元素个数,i=2

for i ⟵ 2 to k

do C[i] ⟵ C[i] + C[i - 1] ▶ **C[i] = |{key ≤ i}|**

循环3：从频率到累积频率……

A:

1	2	3	4	5
4	1	3	4	3

B:

C:

1	2	3	4
1	0	2	2

C′:

1	1	3	2

C[i]=数组A中键值≤i的元素个数,i=3

for i ⟵ 2 to k

do C[i] ⟵ C[i] + C[i - 1] ▶ **C[i] = |{key ≤ i}|**

循环3：从频率到累积频率……

A:

1	2	3	4	5
4	1	3	4	3

B:

C:

1	2	3	4
1	0	2	2

C′:

1	1	3	5

C[i]=数组A中键值≤i的元素个数,i=4

for i $\longleftarrow$ 2 to k

do C[i] $\longleftarrow$ C[i] + C[i - 1] ▶ **C[i] = |{key ≤ i}|**

循环 4：交换 A 中的元素

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C′:	1	1	3	5

for j ⟵ n downto 1

do B[C[A[j]]] ⟵ A[j]

C[A[j]] ⟵ C[A[j]] - 1

对数组A从尾到头进行元素置换

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

C′:

1	2	3	4
1	1	3	5

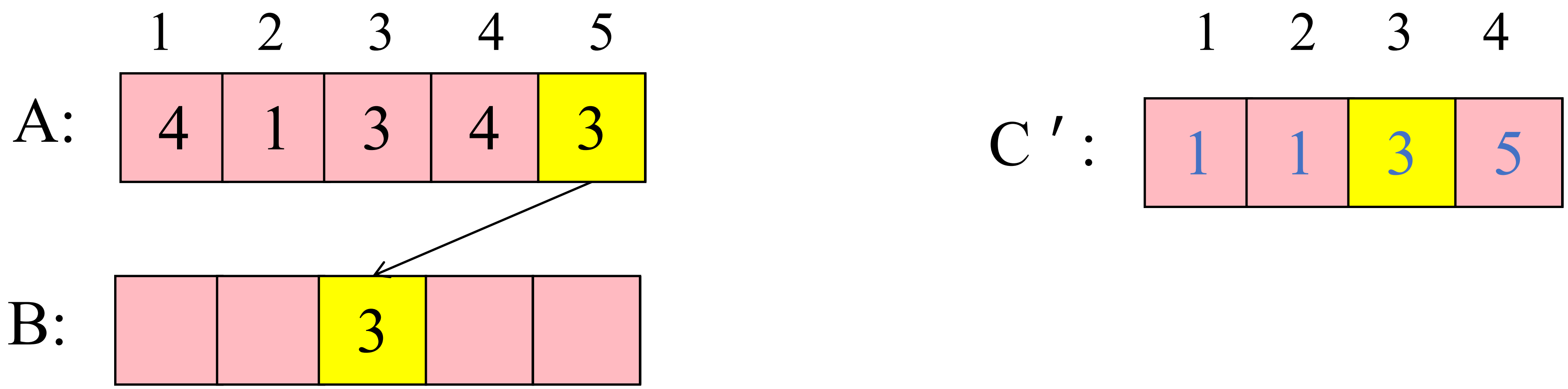
for j ← n downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1

正好有3个元素≤ A [5],那么应该把A[5]元素放到哪个位置去？

循环 4：交换 A 中的元素



for j ⟵ n downto 1

do B[C[A[j]]] ⟵ A[j]

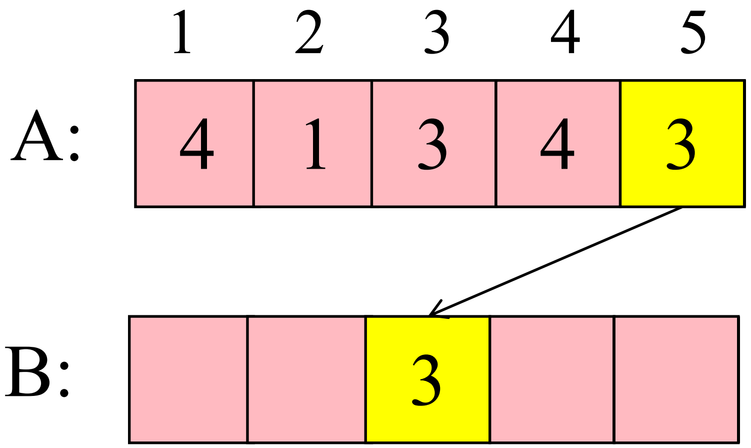
C[A[j]] ⟵ C[A[j]] - 1

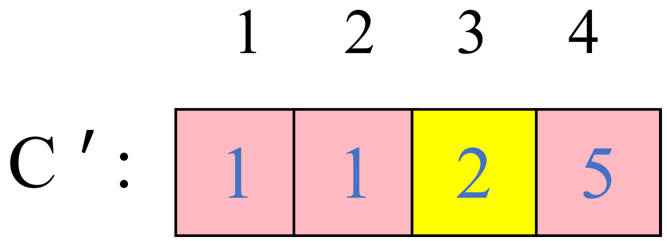
把A[5]放到B[C[A[5]]]中

用了C[A[5]]的最上面一个元素位置

更新C[A[5]]来给A数组中其他键值为A[5]元素来安排位置！

循环 4：交换 A 中的元素





for j ⟵ n downto 1

do B[C[A[j]]] ⟵ A[j]

C[A[j]] ⟵ C[A[j]] - 1

把A[5]放到B[C[A[5]]]中

用了C[A[5]]的最上面一个元素位置

更新C[A[5]]来给A数组中其他键值为A[5]

元素来安排位置！

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

		3		

C′:

1	2	3	4
1	1	2	5

for j ← n downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1

把A[4]放到B[C[A[4]]]中

用了C[A[4]]的最上面一个元素位置

更新C[A[4]]来给A数组中其他键值为A[4]元素来安排位置！

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

		3		

C′:

1	2	3	4
1	1	2	5

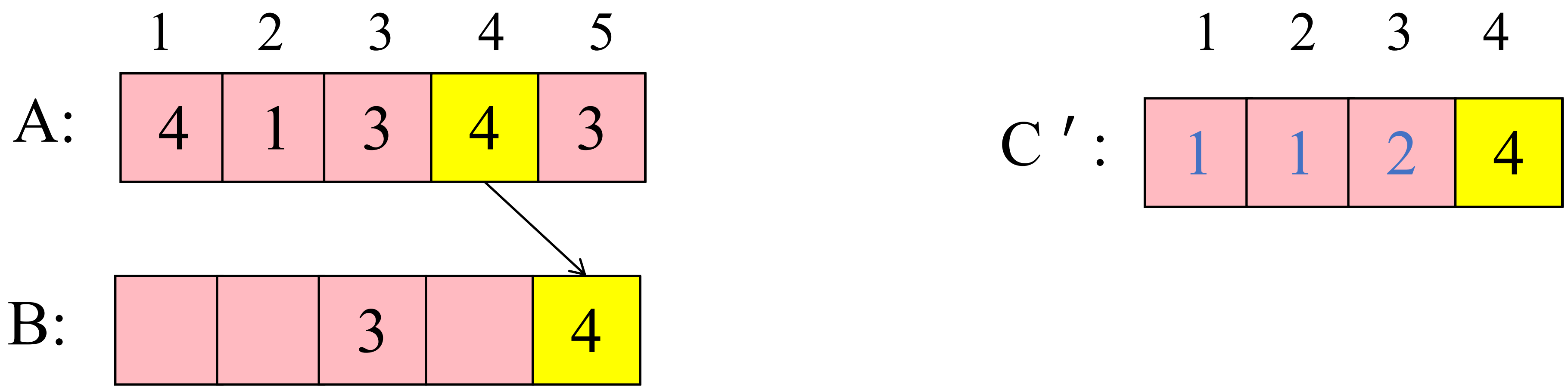
```
for j ← n downto 1
  do B[ C[ A[ j ] ]] ← A[ j]
     C[ A[ j ] ] ← C[ A[ j ] ] - 1
```

把A[4]放到B[C[A[4]]]中

用了C[A[4]]的最上面一个元素位置

更新C[A[4]]来给A数组中其他键值为A[4]元素来安排位置！

循环 4：交换 A 中的元素



for j ← n downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1

把A[4]放到B[C[A[4]]]中

用了C[A[4]]的最上面一个元素位置

更新C[A[4]]来给A数组中其他键值为A[4]元素来安排位置！

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

1	2	3	4	5
		3		4

C′:

1	2	3	4
1	1	2	4

for j ⟵ n downto 1

do B[C[A[j]]] ⟵ A[j]

C[A[j]] ⟵ C[A[j]] - 1

把A[3]放到B[C[A[3]]]中

用了C[A[3]]的最上面一个元素位置

更新C[A[3]]来给A数组中其他键值为A[4]元素来安排位置！

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

		3		4

C′:

1	2	3	4
1	1	2	4

for j ← n downto 1

do B[C[A[j]]] ← A[j]

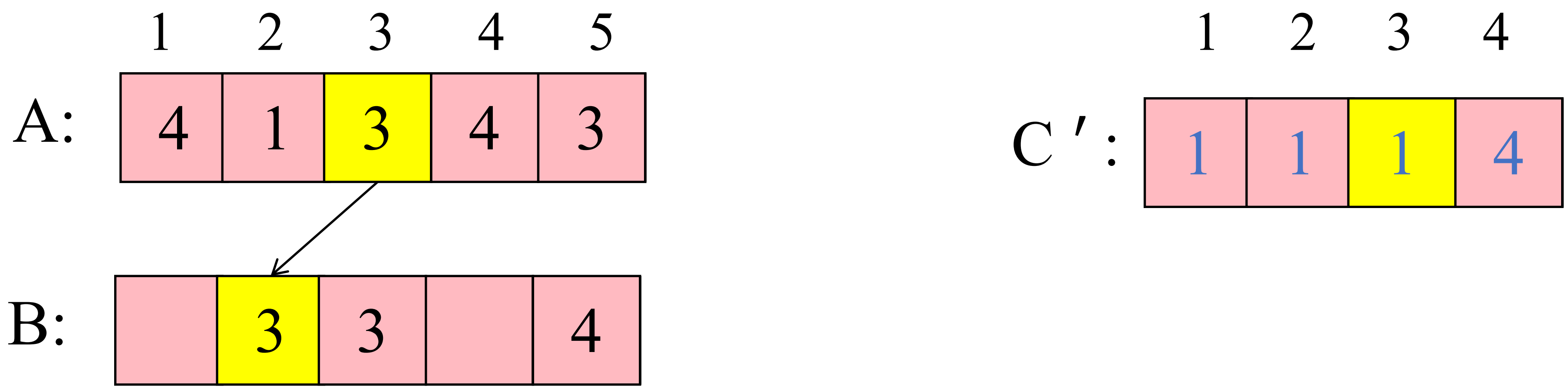
C[A[j]] ← C[A[j]] - 1

把A[3]放到B[C[A[3]]]中

用了C[A[3]]的最上面一个元素位置

更新C[A[3]]来给A数组中其他键值为A[4]元素来安排位置！

循环 4：交换 A 中的元素




```
for j ← n downto 1
 do B[ C[ A[ j ] ]] ← A[ j]
     C[ A[ j ] ] ← C[ A[ j ] ] - 1
```

把A[3]放到B[C[A[3]]]中

用了C[A[3]]的最上面一个元素位置

更新C[A[3]]来给A数组中其他键值为A[4]元素来安排位置！

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

	3	3		4

C′:

1	2	3	4
1	1	1	4

for j ← n downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1

把A[2]放到B[C[A[2]]]中

用了C[A[2]]的最上面一个元素位置

更新C[A[2]]来给A数组中其他键值为A[2]元素来安排位置！

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

	3	3		4

C′:

1	2	3	4
1	1	1	4

for j ⟵ n downto 1

do B[C[A[j]]] ⟵ A[j]

C[A[j]] ⟵ C[A[j]] - 1

把A[2]放到B[C[A[2]]]中

用了C[A[2]]的最上面一个元素位置

更新C[A[2]]来给A数组中其他键值为A[2]元素来安排位置！

循环 4：交换 A 中的元素

	1	2	3	4	5
A:	4	1	3	4	3
B:	1	3	3		4

	1	2	3	4
C ′ :	0	1	1	4

for j ⟵ n downto 1

do B[C[A[j]]] ⟵ A[j]

C[A[j]] ⟵ C[A[j]] - 1

把A[2]放到B[C[A[2]]]中

用了C[A[2]]的最上面一个元素位置

更新C[A[2]]来给A数组中其他键值为A[2]元素来安排位置！

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

1	3	3		4

C′:

1	2	3	4
0	1	1	4

for j ← n downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1

把A[1]放到B[C[A[1]]]中

用了C[A[1]]的最上面一个元素位置

更新C[A[1]]来给A数组中其他键值为A[1]元素来安排位置！

循环 4：交换 A 中的元素

A:

1	2	3	4	5
4	1	3	4	3

B:

1	3	3		4

C′:

1	2	3	4
0	1	1	4

```
for j ⟵ n downto 1
  do B[ C[ A[ j ] ]] ⟵ A[ j]
     C[ A[ j ] ] ⟵ C[ A[ j ] ] - 1
```

把A[1]放到B[C[A[1]]]中

用了C[A[1]]的最上面一个元素位置

更新C[A[1]]来给A数组中其他键值为A[1]元素来安排位置！

循环 4：交换 A 中的元素

	1	2	3	4	5
A:	4	1	3	4	3
B:	1	3	3	4	4

	1	2	3	4
C′:	0	1	1	3

for j $\leftarrow$ n downto 1

do B[C[A[j]]] $\leftarrow$ A[j]

C[A[j]] $\leftarrow$ C[A[j]] - 1

把A[1]放到B[C[A[1]]]中

用了C[A[1]]的最上面一个元素位置

更新C[A[1]]来给A数组中其他键值为A[1]

元素来安排位置！

计数排序

for i ⟵ 1 to k

　do C[i] ⟵ 0

for j ⟵ 1 to n

　do C[A[j]] ⟵ C[A[j]] + 1

将数组A中不同键的频率存储在C中 。C[i] = |{key = i}|

for i ⟵ 2 to k

　do C′[i] ⟵ C[i] + C[i - 1]

将数组A中不同键的频率存储在C中 。C[i] = |{key ≤ i}|

for j ⟵ n downto 1

　do B[C ′[A[j]]] ⟵ A[j]

　C ′[A[j]] ⟵ C ′[A[j]] - 1

使用累积频率构建有序排列

O(k)

O(n)

O(k)

O(n)

O(n+k)

运行时间

加入$k=O(n)$,那么计数排序需要

如果$k=O(n)$，那么计数排序需要的时间是$O(n)$。

但是，排序需要的时间是$\Omega(n\lg n)$！

哪里出了问题？

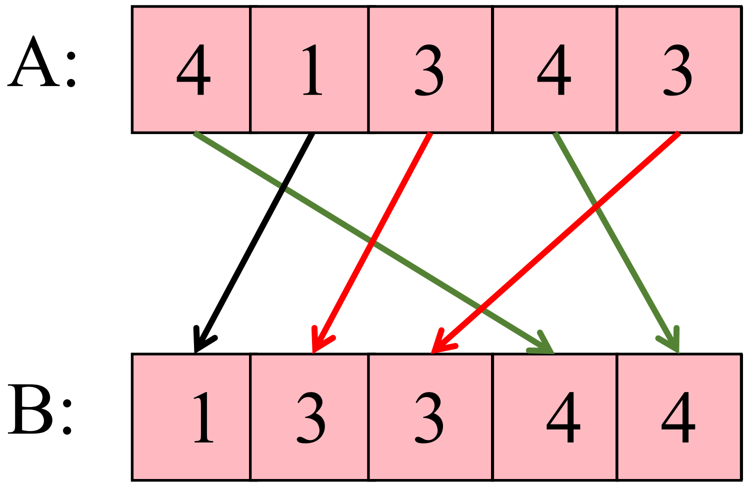
答案：比较排序需要的时间是$\Omega(n\lg n)$。

计数排序不是一种比较排序。

实际上，没有进行过任何元素之间的比较！

稳定排序

计数排序是一种稳定排序：它给同样键值的元素们保留了原先排序！



基数排序

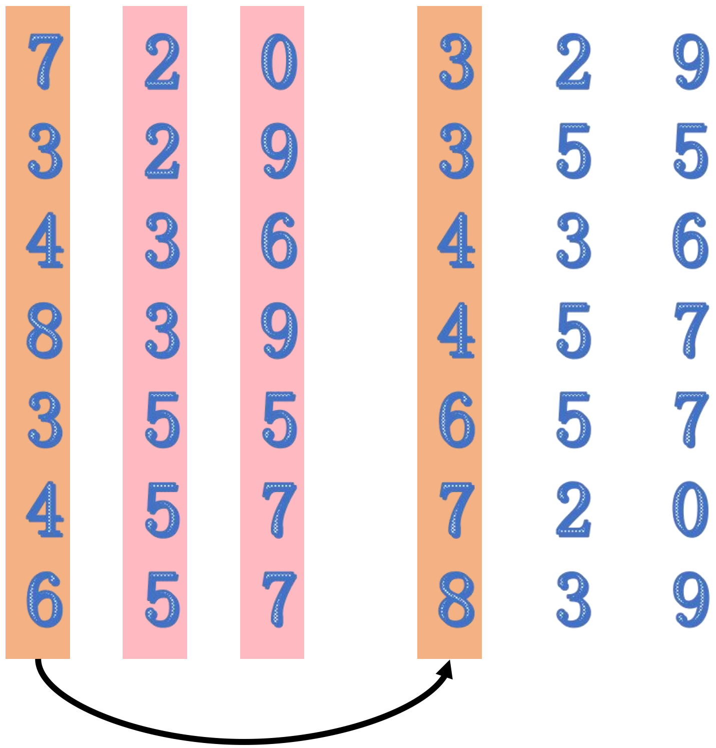
- 1890年用来解决美国的人口普查！
- 一个位数接一个位数的排序！
- 原来的（差的）想法：高位优先
- 好想法：低位优先

基数排序的操作过程

	个位排序	十位排序	百位排序
329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

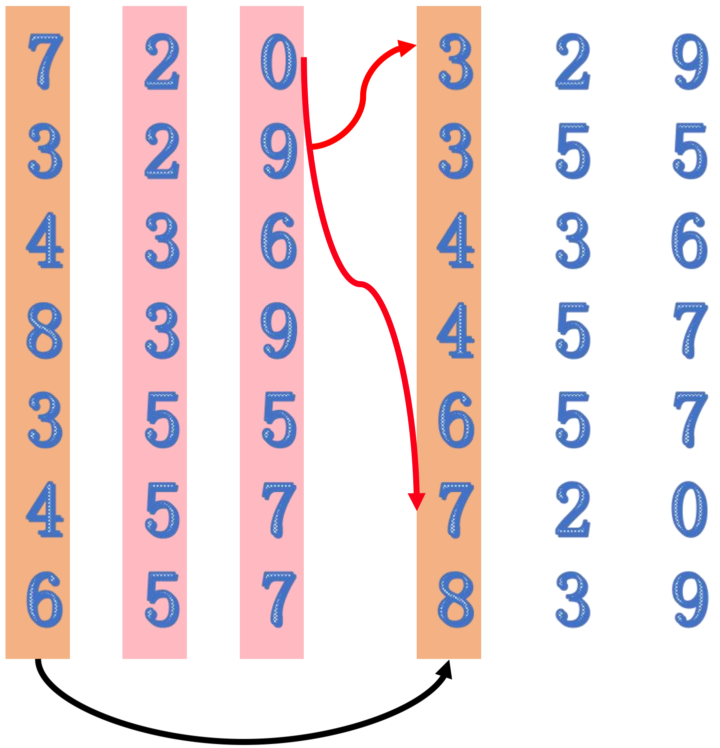
基数排序的正确性

- 对位数上的数学归纳
- 假设t-1位数们，都是排好序的
- 对t位数的排序操作能保证键值只有t位的数组的排序是正确的。



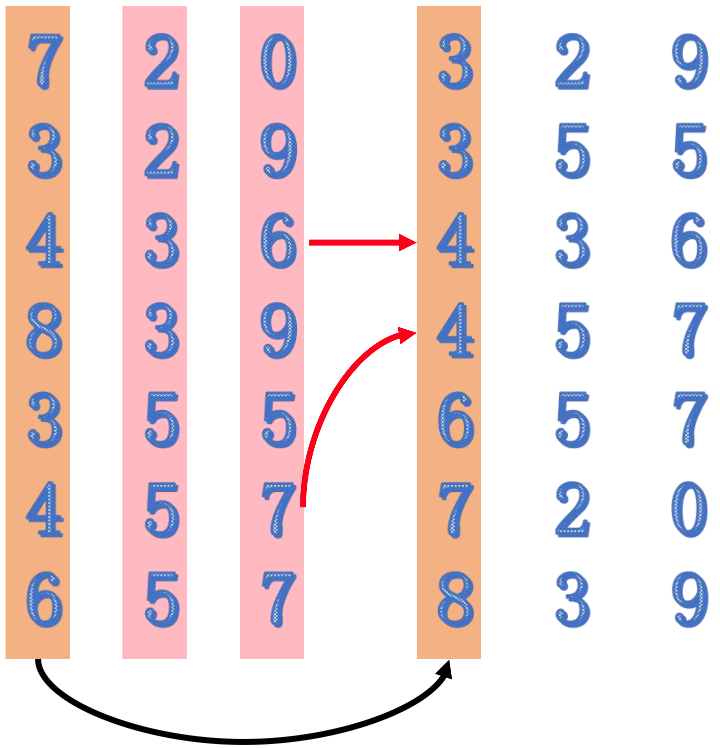
基数排序的正确性

- 对位数上的数学归纳
- 假设t-1位数们，都是排好序的
- 任意两个在t位数上键值不一样　的元素，在t位上比对交换大小　排序后，与这两个元素在数位1- t位上的大小顺序是一致的。



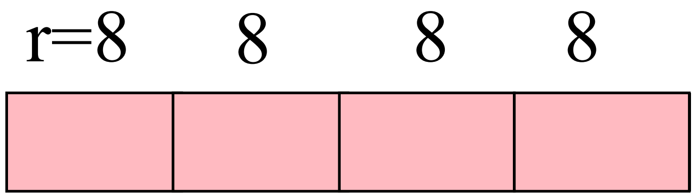
基数排序的正确性

- 对位数上的数学归纳
- 假设**t-1**位数们，都是排好序的
- 任意两个在t位数上键值一样的元素，与这两个元素在数位1-t位上的大小顺序依然是一致的（得证！）



基数排序的运行时间分析

- 计数排序作为辅助的稳定排序方法。
- 每一个键值由b个二进制组成
- 基数为2^r个比特， 一个键值可以化成b/r个位数



- 例如，对于一个32位的字来说，如果我们将每个b位的字拆分成r位的片段，那么每次计数排序的过程需要$Q(n+2^r)$的时间。因此，总体时间为$(b/r)(n+2^r)$。如果我们设定$r=\log n$，那么在每次传递过程中可以得到$O(n)$的时间复杂度，或者总的时间复杂度为$O(n * b / \log n)$。

湖南大学
HUNAN UNIVERSITY

谢谢观赏

实 事 求 是　敢 为 人 先