



Sorting 1

Zibin Zheng (郑子彬)

School of Data and Computer Science , SYSU

<http://www.inpluslab.com>

课程主页: <http://inpluslab.sysu.edu.cn/dsa2016/>

什么是排序

- 在信息处理过程中，最基本的操作是查找。从查找来说，效率最高的是折半查找，折半查找的前提是所有的数据元素(记录)是按关键字有序的。需要将一个无序的数据文件转变为一个有序的数据文件。
- 将任一文件中的记录通过某种方法整理成为按(记录)关键字有序排列的处理过程称为**排序**。
- 排序是**数据处理**中一种最常用的操作。

排序的基本概念

排序(Sorting)

排序是将一批(组)任意次序的记录重新排列成**按关键字有序**的记录序列的过程，其定义为：

给定一组记录序列： $\{R_1, R_2, \dots, R_n\}$ ，其相应的关键字序列是 $\{K_1, K_2, \dots, K_n\}$ 。确定 $1, 2, \dots, n$ 的一个排列 p_1, p_2, \dots, p_n ，使其相应的关键字满足如下非递减(或非递增)关系： $K_{p_1} \leq K_{p_2} \leq \dots \leq K_{p_n}$ 的序列 $\{K_{p_1}, K_{p_2}, \dots, K_{p_n}\}$ ，这种操作称为排序。

排序的基本概念

- **数据表(datalist)**：它是待排序数据对象的有限集合。
- **关键字(key)**：通常数据对象有多个**属性域**，即多个数据成员组成，其中有一个属性域可用来区分对象，作为排序依据。该域即为关键字。每个数据表用哪个属性域作为关键字，要视具体的应用需要而定。即使是同一个表，在解决不同问题的场合也可能取不同的域做关键字。

排序的基本概念

- **排序算法的稳定性 (Stability)**：如果在对象序列中有两个对象 $r_{[i]}$ 和 $r_{[j]}$ ，它们的关键字 $k_{[i]} == k_{[j]}$ ，且在排序之前，对象 $r_{[i]}$ 排在 $r_{[j]}$ 前面。如果在排序之后，对象 $r_{[i]}$ 仍在对象 $r_{[j]}$ 的前面，则称这个排序方法是稳定的，否则称这个排序方法是不稳定的。

2, 4, 9, **12, 12***, 22, 33

12, 4, 22, 9, 12*, 33, 2

2, 4, 9, **12*, 12**, 22, 33

排序的基本概念

- **内排序与外排序：** 内排序是指在排序期间数据对象全部存放在内存的排序；外排序是指在排序期间全部对象个数太多，不能同时存放在内存，必须根据排序过程的要求，不断在内、外存之间移动的排序。
- **排序的时间开销：** 排序的时间开销是衡量算法好坏的最重要的标志。排序的时间开销可用算法执行中的数据比较次数与数据移动次数来衡量。各节给出算法运行时间代价的大略估算一般都按平均情况进行估算。对于那些受对象关键字序列初始排列及对象个数影响较大的，需要按最好情况和最坏情况进行估算。

衡量排序方法的标准

- 排序时所需要的平均比较次数
- 排序时所需要的平均移动
- 排序时所需要的平均辅助存储空间
- 排序的稳定性

排序算法的存储结构

从操作角度看，排序是线性结构的一种操作，待排序记录可以用**顺序**存储结构或**链接**存储结构存储。

假定1：采用**顺序**存储结构，关键码为**整型**，且记录只有关键码一个数据项。

```
int r[n+1];
//待排序记录存储在r[1]~r[n]，r[0]留做他用
```

假定2：将待排序的记录序列排序为**升序**序列。

排序方法

- Insertion Sort (**直接插入**、希尔排序) ★
- Exchange sort (冒泡排序、快速排序)
- Selection Sort (简单选择排序、堆排序)
- Merge Sort (归并排序)
- Radix Sort (基数排序)

插入排序 (Insertion Sort)

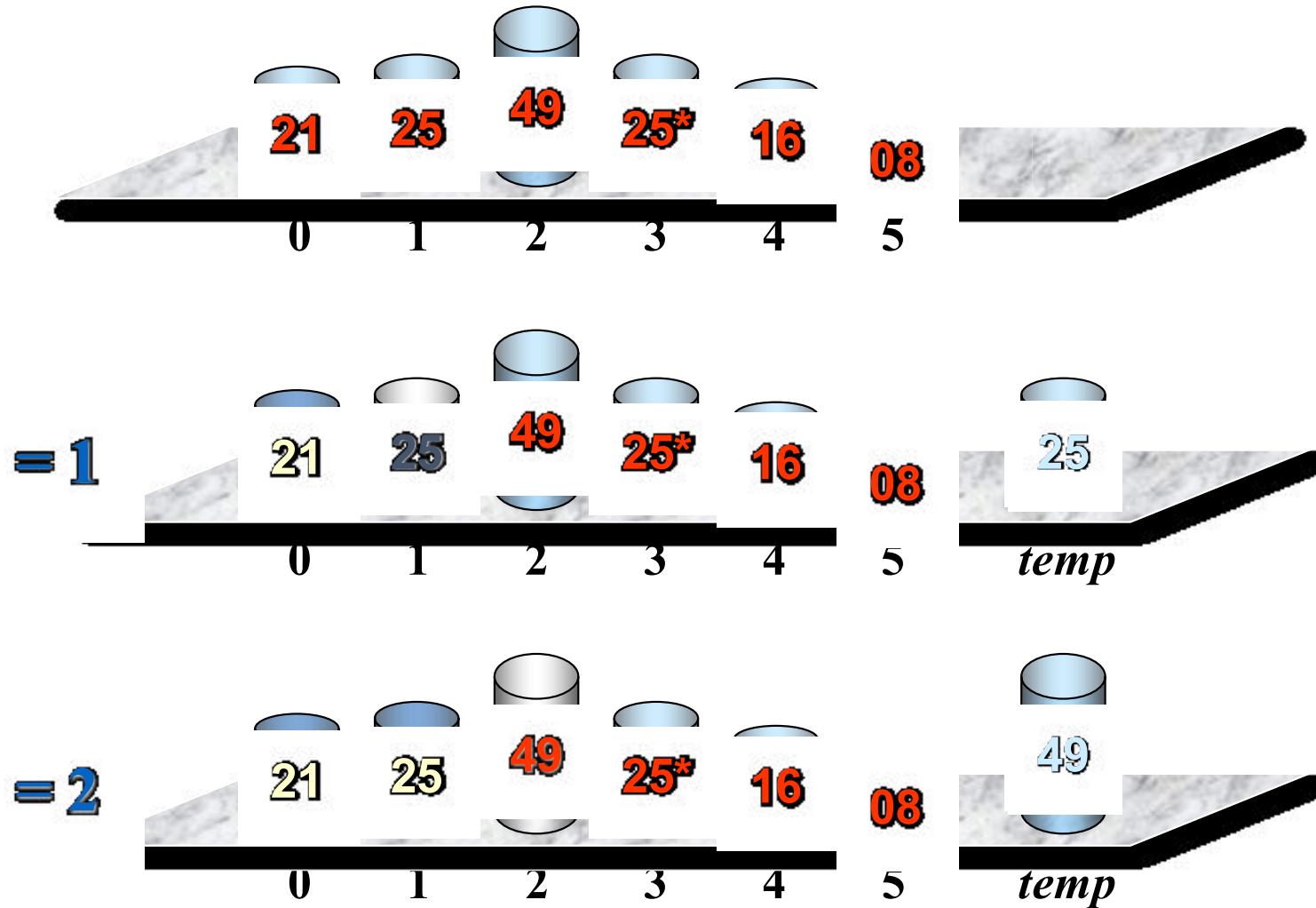
- 插入排序的基本方法是：每步将一个待排序的对象，按其关键字大小，插入到前面已经排好序的一组对象的适当位置上，直到对象全部插入为止。

直接插入排序 (Insertion Sort)

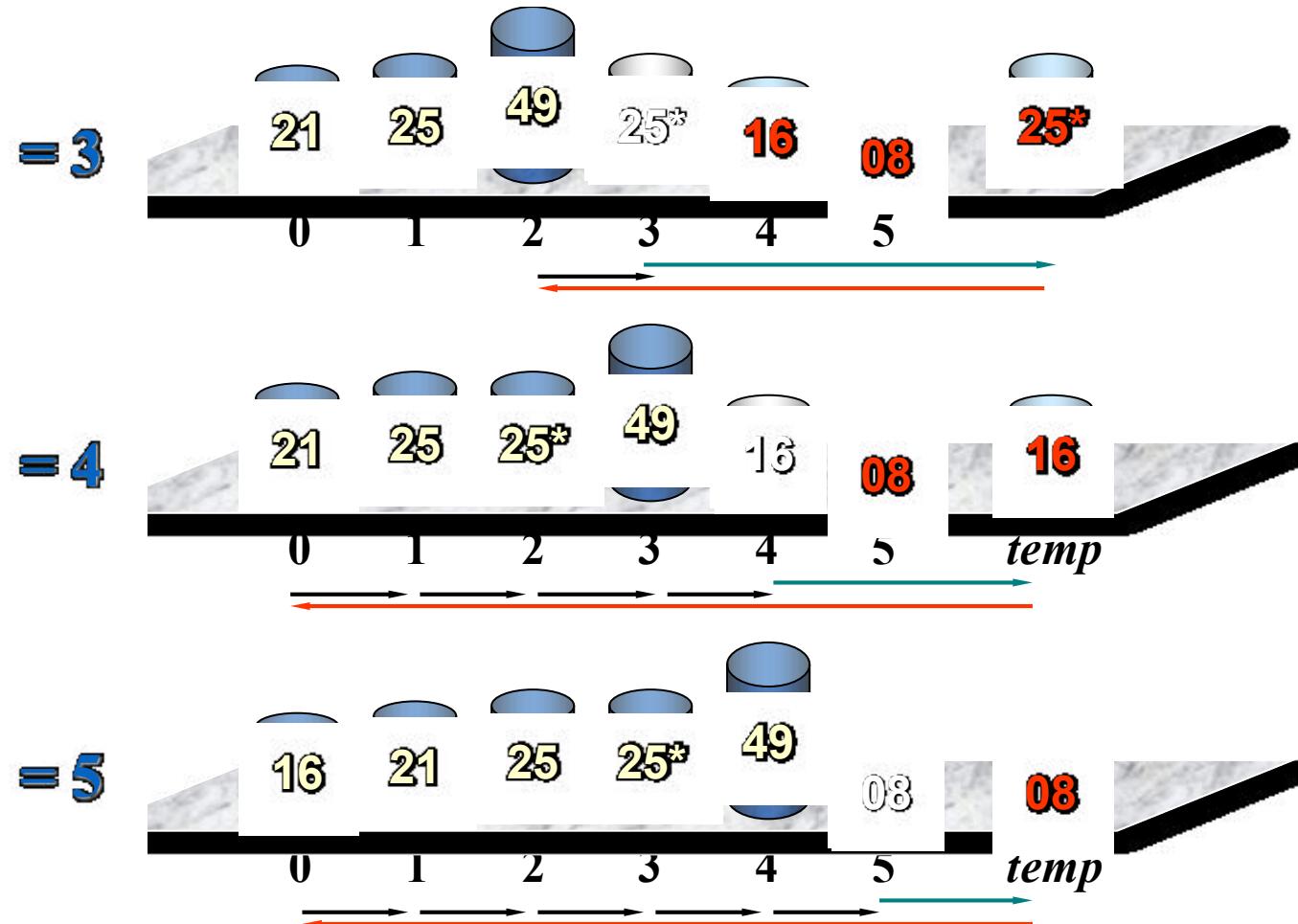
- 基本思想
- 当插入第 i ($i \geq 1$) 个对象时，前面的 $v[0], v[1], \dots, v[i-1]$ 已经排好序。这时，用 $v[i]$ 的关键字与 $v[i-1], v[i-2], \dots$ 的关键字顺序进行比较，找到插入位置即将 $v[i]$ 插入，原来位置上之后的所有对象依次向后顺移。

直接插入排序 (Insertion Sort)

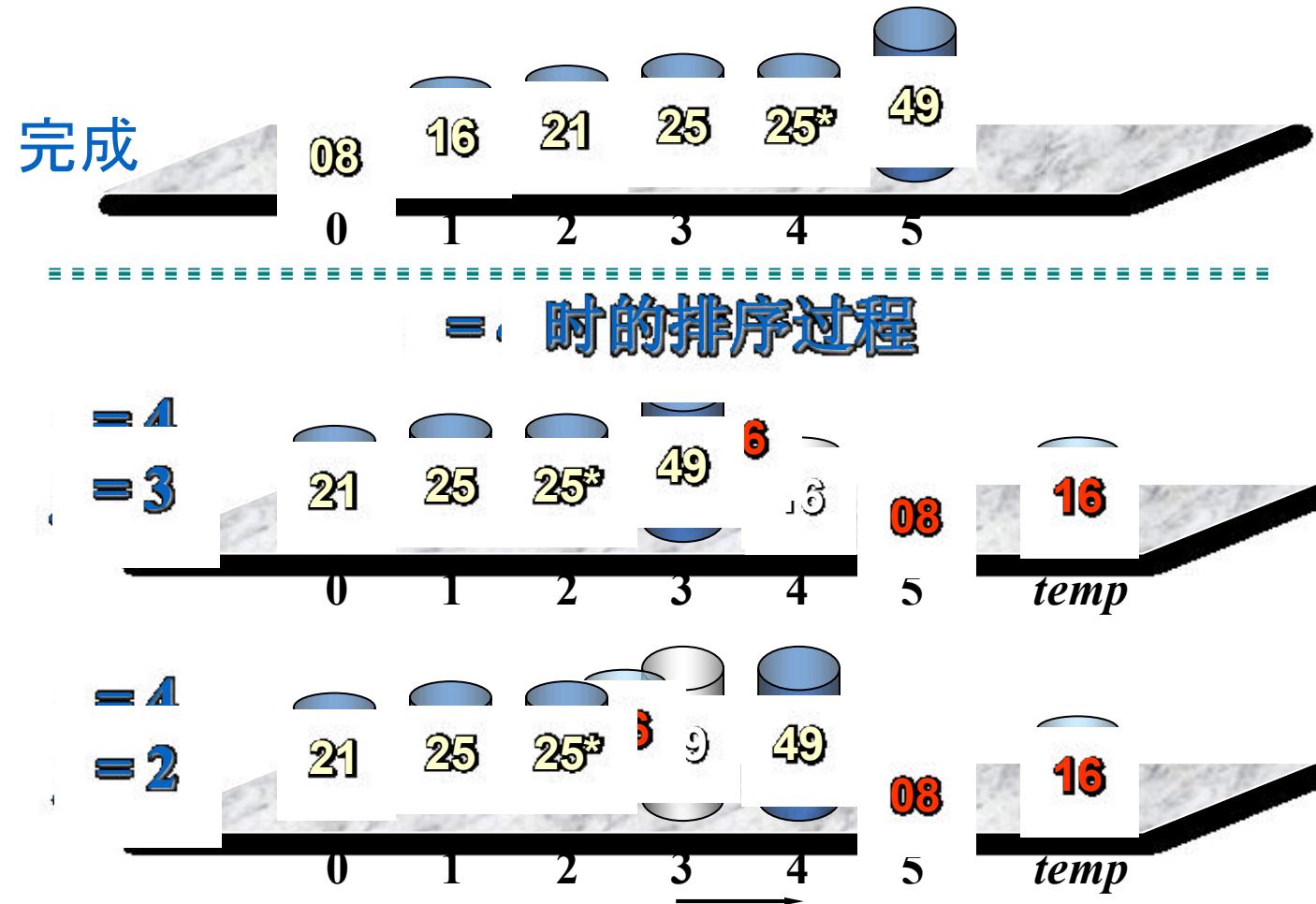
各趟排序結果



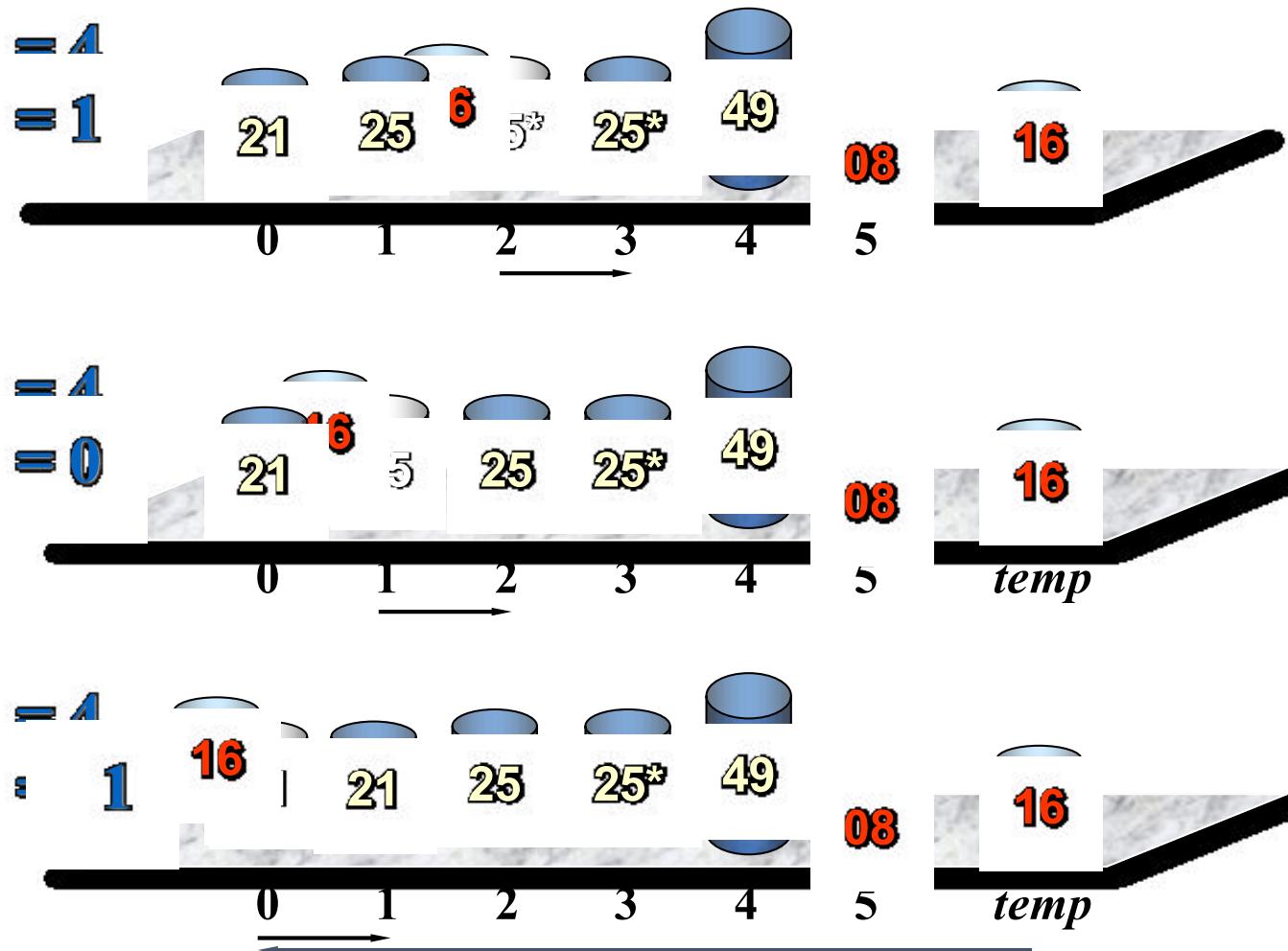
直接插入排序 (Insertion Sort)



直接插入排序 (Insertion Sort)



直接插入排序 (Insertion Sort)



```
void insertSort (int r[ ], int n)
{
    for (i=2; i<=n; i++)
    {
        r[0]=r[i]; j=i-1;
        while (r[0]<r[j])
        {
            r[j+1]=r[j];
            j=j-1;
        }
        r[j+1]=r[0];
    }
}
```



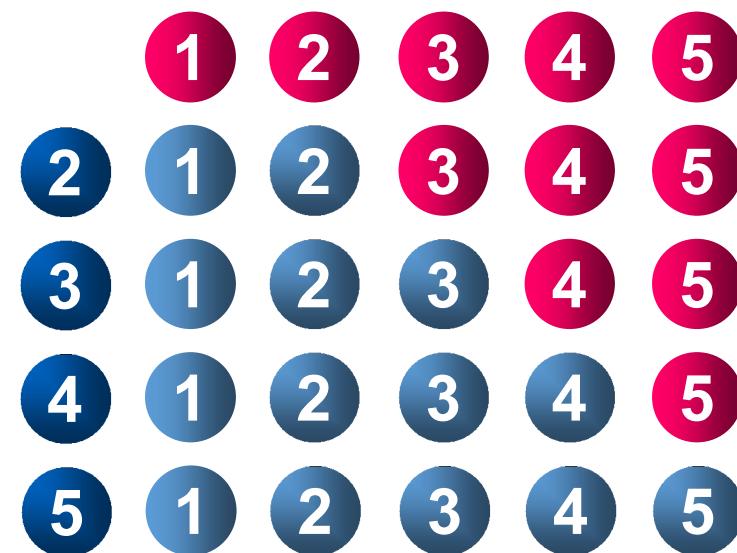
如果 $r[i] \geq r[i-1]$, 内层循环会出现什么情况?

直接插入排序算法性能分析

最好情况下（正序）：

{ 比较次数： $n-1$
移动次数： $2(n-1)$

时间复杂度为 $O(n)$ 。

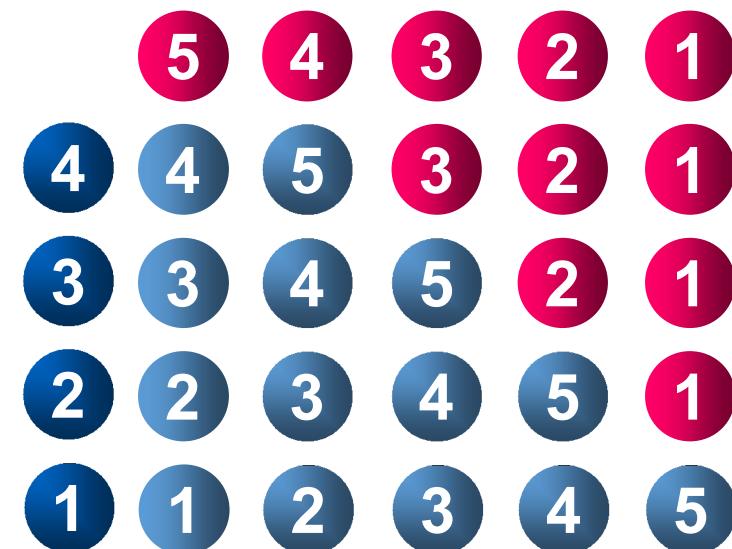


直接插入排序算法性能分析

最坏情况下（逆序或反序）：

$$\left\{ \begin{array}{l} \text{比较次数: } \sum_{i=2}^n i = \frac{(n+2)(n-1)}{2} \\ \text{移动次数: } \sum_{i=2}^n (i+1) = \frac{(n+4)(n-1)}{2} \end{array} \right.$$

时间复杂度为 $O(n^2)$ 。



直接插入排序算法性能分析

平均情况下（随机排列）：

$$\left\{ \begin{array}{l} \text{比较次数: } \sum_{i=2}^n \frac{i}{2} = \frac{(n+2)(n-1)}{4} \\ \text{移动次数: } \sum_{i=2}^n \frac{(i+1)}{2} = \frac{(n+4)(n-1)}{4} \end{array} \right.$$

时间复杂度为 $O(n^2)$ 。

直接插入排序算法性能分析

空间性能：需要一个记录的辅助空间。

直接插入排序算法是一种**稳定的**排序算法。

- 直接插入排序算法简单、容易实现，适用于待排序记录基本有序或待排序记录较小时。
- 当待排序的记录个数较多时，大量的比较和移动操作使直接插入排序算法的效率降低。

排序方法

- Insertion Sort (直接插入、**希尔排序**) ★
- Exchange sort (冒泡排序、快速排序)
- Selection Sort (简单选择排序、堆排序)
- Merge Sort (归并排序)
- Radix Sort (基数排序)

希尔排序

① 分割待排序记录的目的?

1. 减少待排序记录个数;
2. 使整个序列向基本有序发展。

基本有序: 接近正序, 例如{1, 2, 8, 4, 5, 6, 7, 3, 9};

局部有序: 部分有序, 例如{6, 7, 8, 9, 1, 2, 3, 4, 5}。

局部有序不能提高直接插入排序算法的时间性能。

② 启示?

子序列的构成不能是简单地“逐段分割”, 而是将相距某个“增量”的记录组成一个子序列。

希尔插入排序过程示例

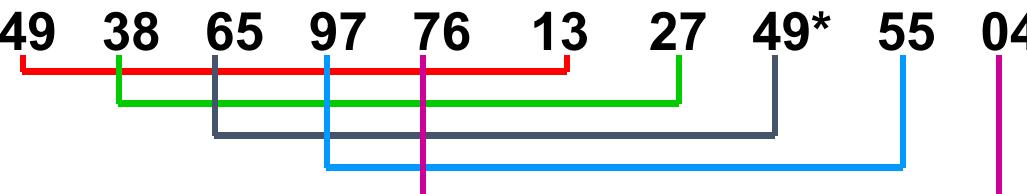
	1	2	3	4	5	6	7	8	9
初始序列	40	25	49	25*	16	21	08	30	13
$d = 4$	40	25	49	25*	16	21	08	30	13
	13	21	08	25*	16	25	49	30	40
$d = 2$	13	21	08	25*	16	25	49	30	40
	08	21	13	25*	16	25	40	30	49
$d = 1$	08	21	13	25*	16	25	40	30	49
	08	13	16	21	25*	25	30	40	49

先将整个待排记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行一次直接插入排序。

例 取增量序列为5, 3, 1

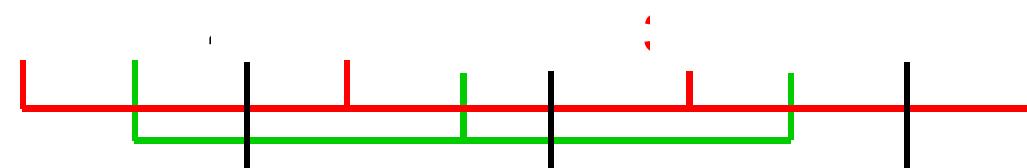
[初始关键字]: 49 38 65 97 76 13 27 49* 55 04

增量=5



一趟排序结果:

增量=3



增量=1



三趟排序结果: 04 13 27 38 49* 49 55 65 76 97

关键问题(1)应如何分割待排序记录?

解决方法:

将相隔某个“增量”的记录组成一个子序列。

增量应如何取?

希尔最早提出的方法是 $d_1=n/2$, $d_{i+1}=d_i/2$ 。

算法描述:

```
for (d=n/2; d>=1; d=d/2)
```

```
{
```

 以d为增量，进行组内直接插入排序；

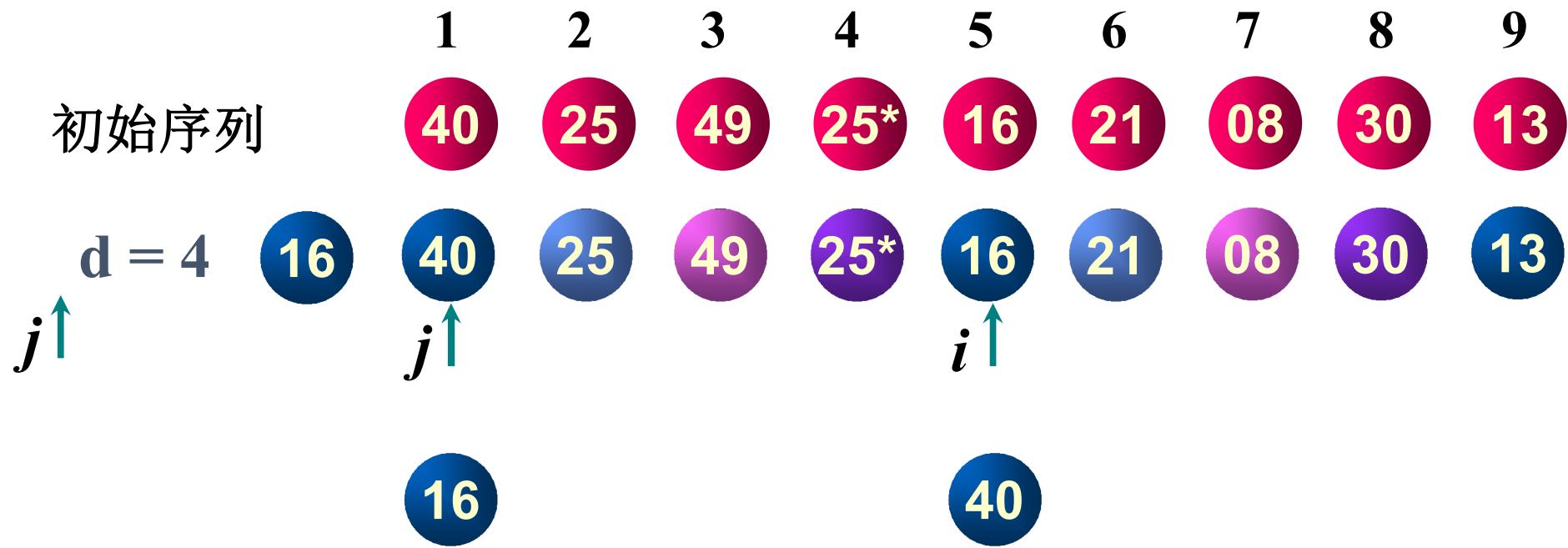
```
}
```

关键问题(2)子序列内如何进行直接插入排序?

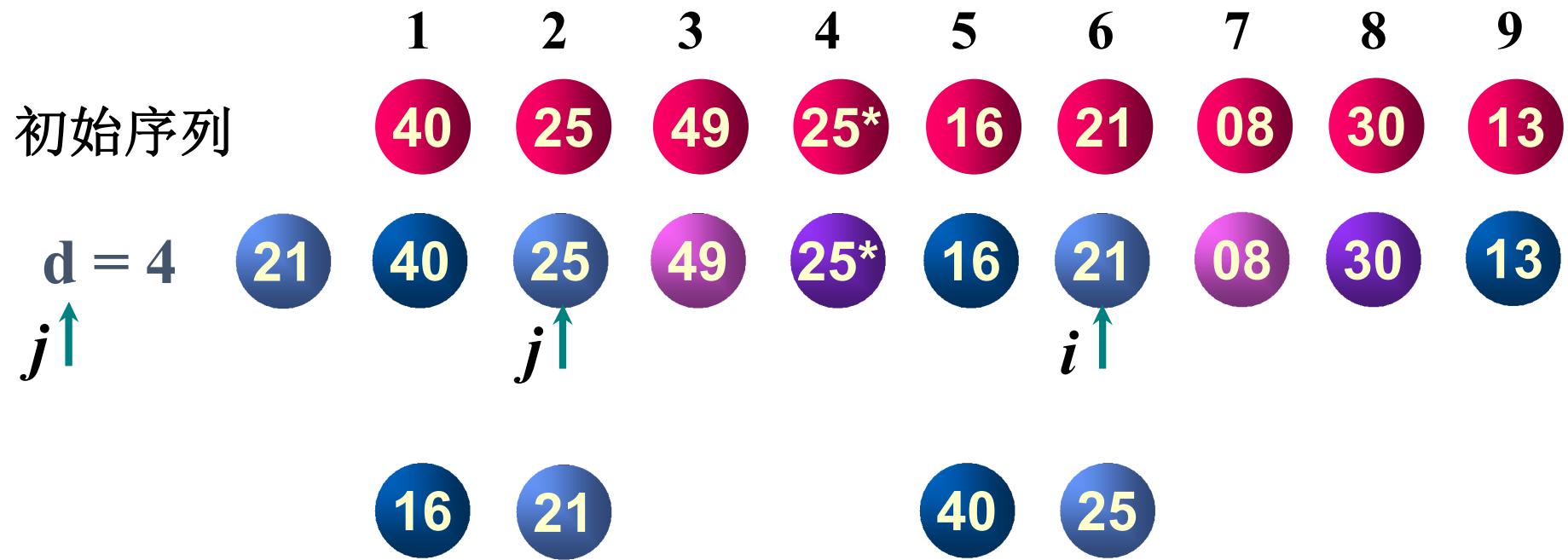
解决方法:

- 在插入记录 $r[i]$ 时，自 $r[i-d]$ 起往前跳跃式（跳跃幅度为 d ）搜索待插入位置，并且 $r[0]$ 只是暂存单元，不是哨兵。当搜索位置 <0 ，表示插入位置已找到。
- 在搜索过程中，记录后移也是跳跃 d 个位置。
- 在整个序列中，前 d 个记录分别是 d 个子序列中的第一个记录，所以从第 $d+1$ 个记录开始进行插入。

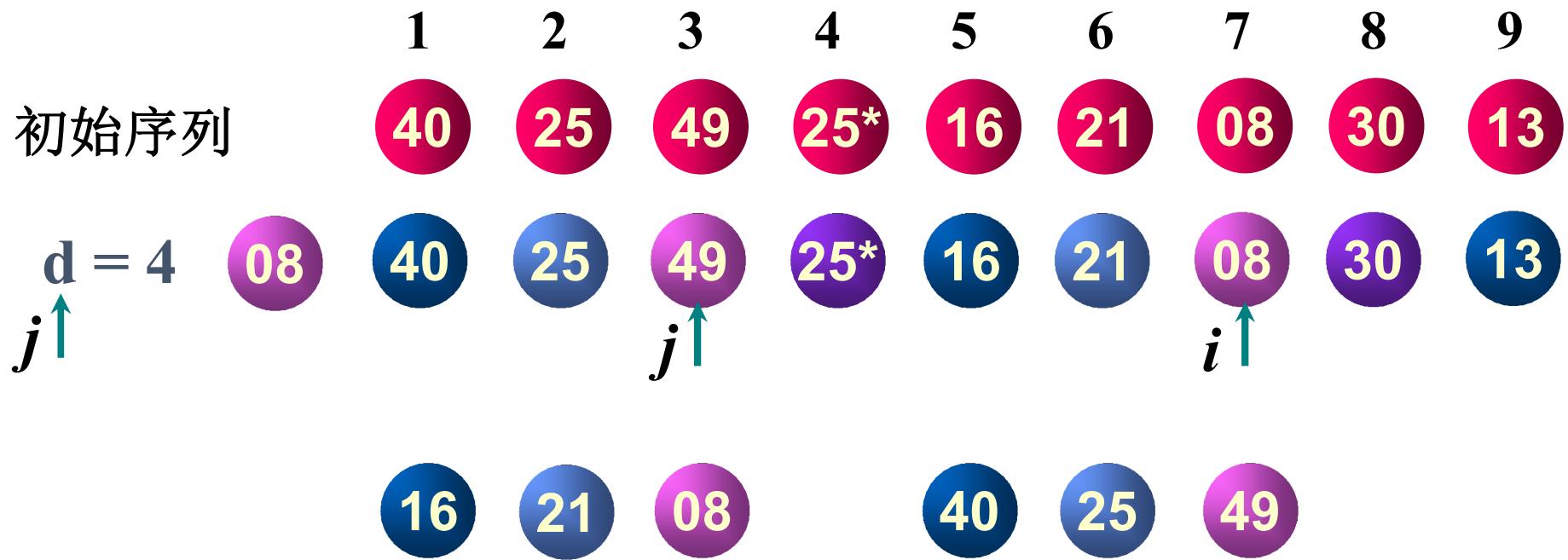
希尔插入排序过程示例



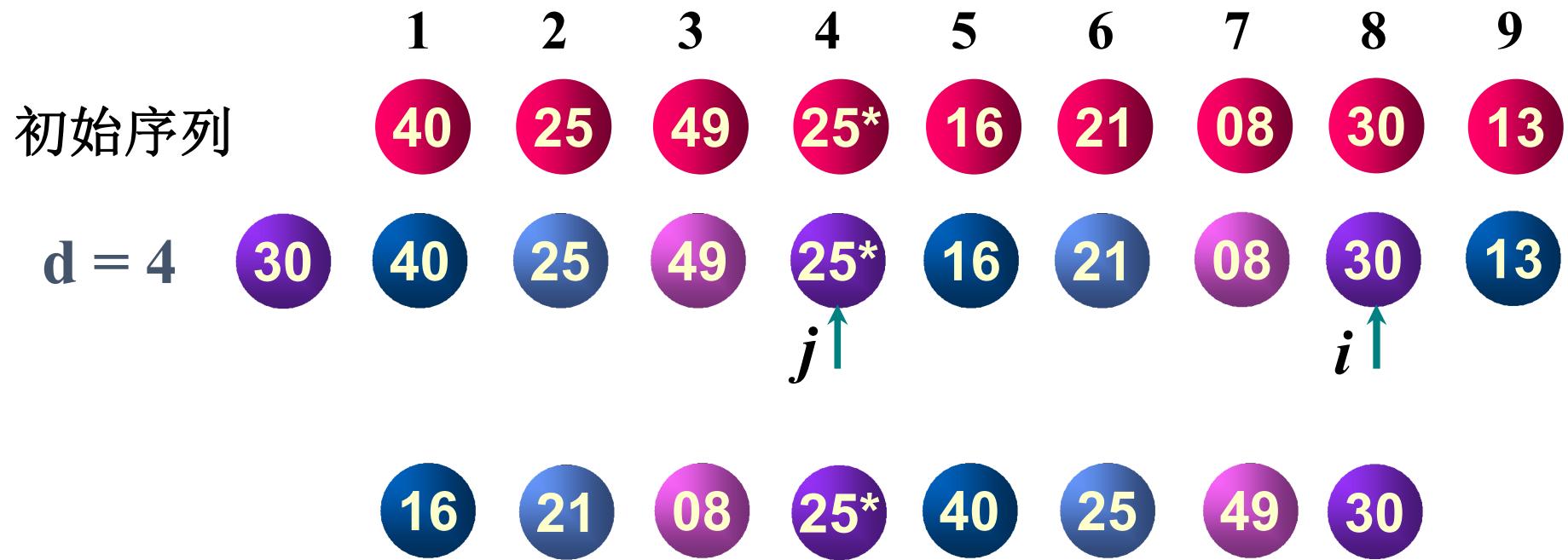
希尔插入排序过程示例



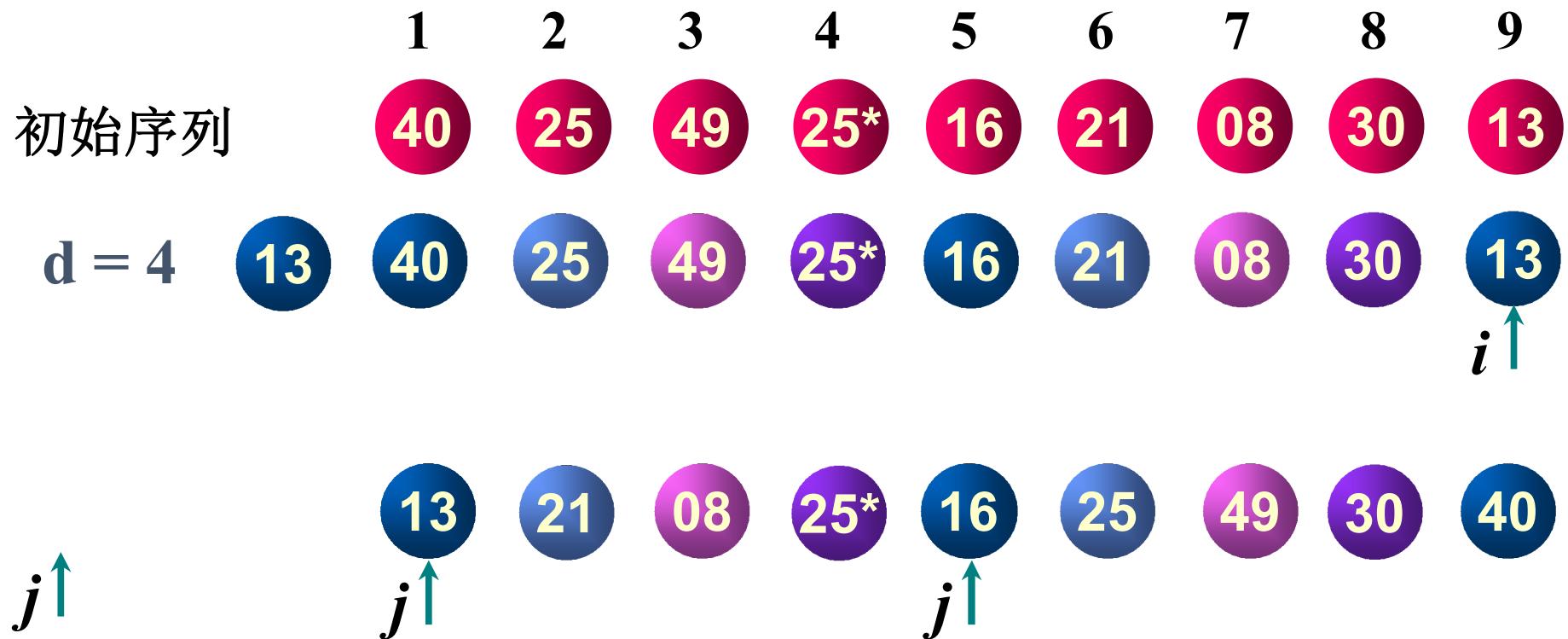
希尔插入排序过程示例



希尔插入排序过程示例



希尔插入排序过程示例



关键问题(2)子序列内如何进行直接插入排序?

算法描述:

```
for (i=d+1; i<=n; i++) //将r[i]插入到所属的子序列中
{
    r[0]=r[i];           //暂存待插入记录
    j=i-d;              //j指向所属子序列的最后一个记录
    while (j>0 && r[0]<r[j])
    {
        r[j+d]=r[j];    //记录后移d个位置
        j=j-d;           //比较同一子序列的前一个记录
    }
    r[j+d]=r[0];
}
```

希尔排序算法的时间性能

希尔排序开始时增量较大，每个子序列中的记录个数较少，从而排序速度较快；当增量较小时，虽然每个子序列中记录个数较多，但整个序列已基本有序，排序速度也较快——缩小增量。

希尔排序算法的时间性能是所取**增量**的函数，而到目前为止尚未有人求得一种最好的增量序列。

研究表明，希尔排序的时间性能在 $O(n^2)$ 和 $O(n \log_2 n)$ 之间。当 n 在某个特定范围内，希尔排序所需的比较次数和记录的移动次数约为 $O(n^{1.3})$ 。

排序方法

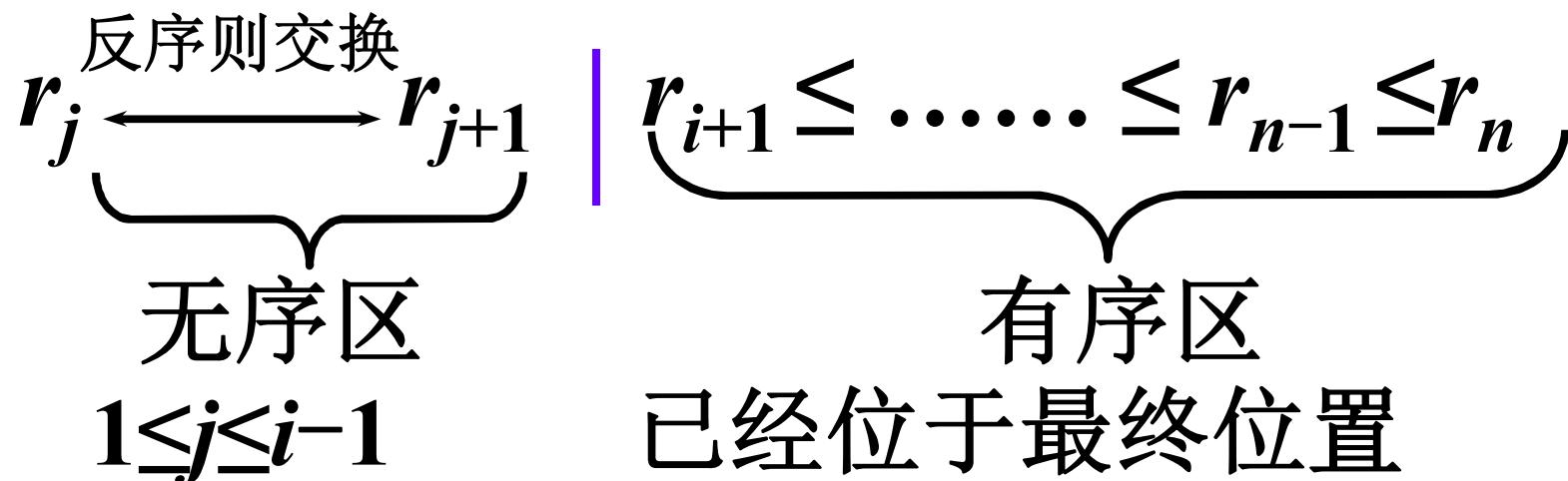
- Insertion Sort (直接插入、希尔排序)
- Exchange sort (**冒泡排序**、快速排序) 
- Selection Sort (简单选择排序、堆排序)
- Merge Sort (归并排序)
- Radix Sort (基数排序)

交换排序的主要操作是**交换**，其主要思想是：
在待排序列中选**两个**记录，将它们的关键码相
比较，如果**反序**（即排列顺序与排序后的次序
正好相反），则**交换**它们的存储位置。

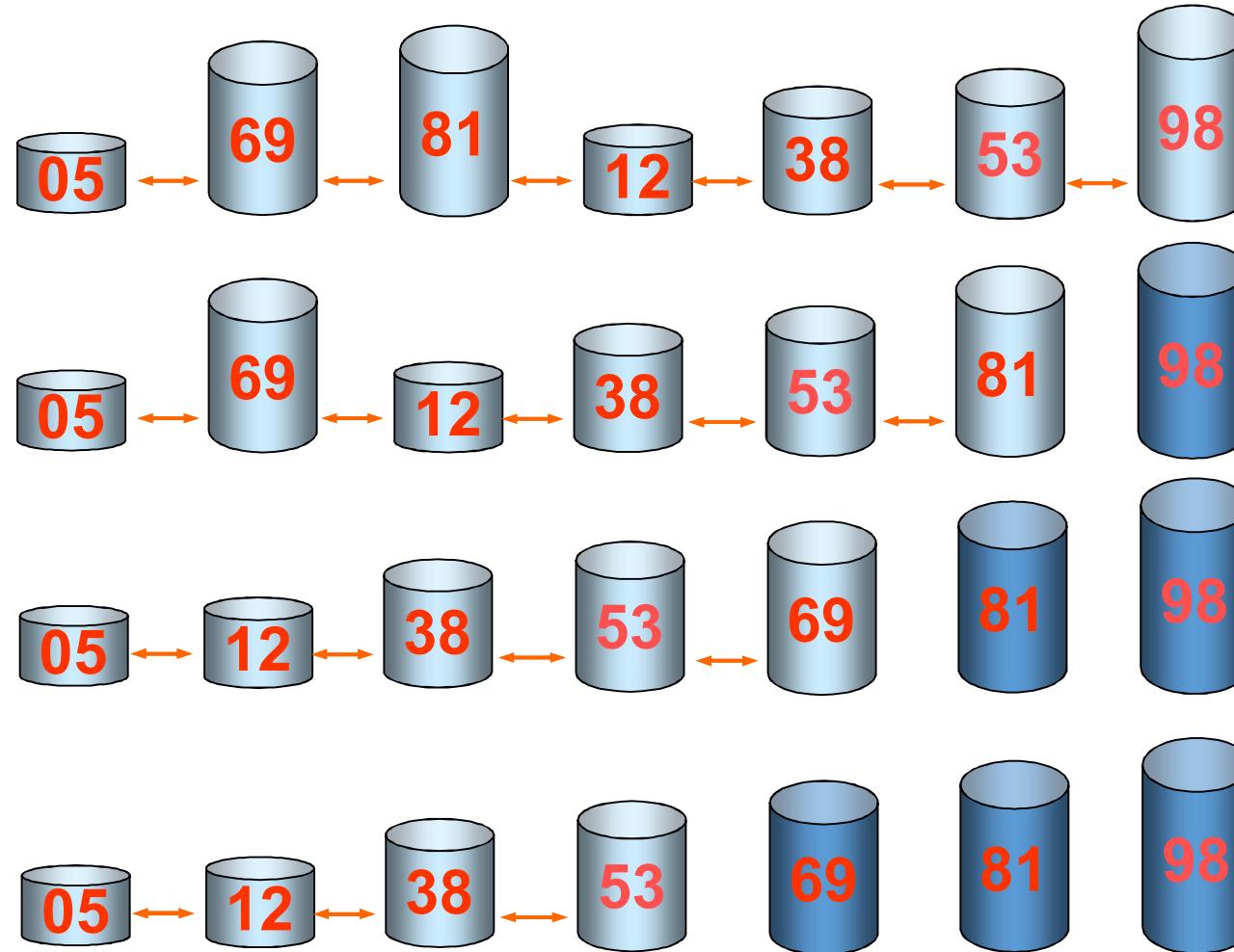


起泡排序

基本思想：两两比较相邻记录的关键码，如果反序则交换，直到没有反序的记录为止。



起泡排序过程示例



起泡排序



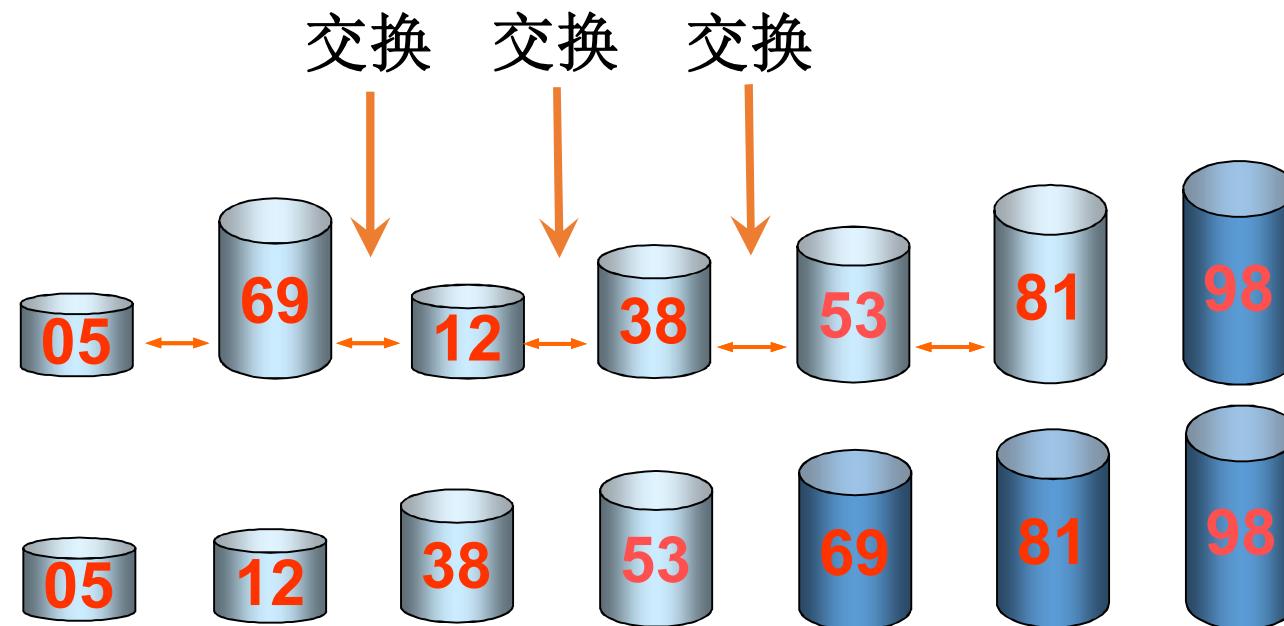
需解决的关键问题？

- (1) 在一趟起泡排序中，若有很多个记录位于最终位置，应如何记载？
- (2) 如何确定起泡排序的范围，使得已经位于最终位置的记录不参与下一趟排序？
- (3) 如何判别起泡排序的结束？

关键问题(1): 如何记载一趟排序过程中交换的多个记录?

解决方法:

设变量exchange记载记录交换的位置，则一趟排序后，exchange记载的一定是这一趟排序中记录的最后一次交换的位置，且从此位置以后的所有记录均已经有序。



关键问题(1): 如何记载一趟排序过程中交换的多个记录?

解决方法:

设变量**exchange**记载记录交换的位置，则一趟排序后，**exchange**记载的一定是这一趟排序中记录的最后一次交换的位置，且从此位置以后的所有记录均已经有序。

算法描述:

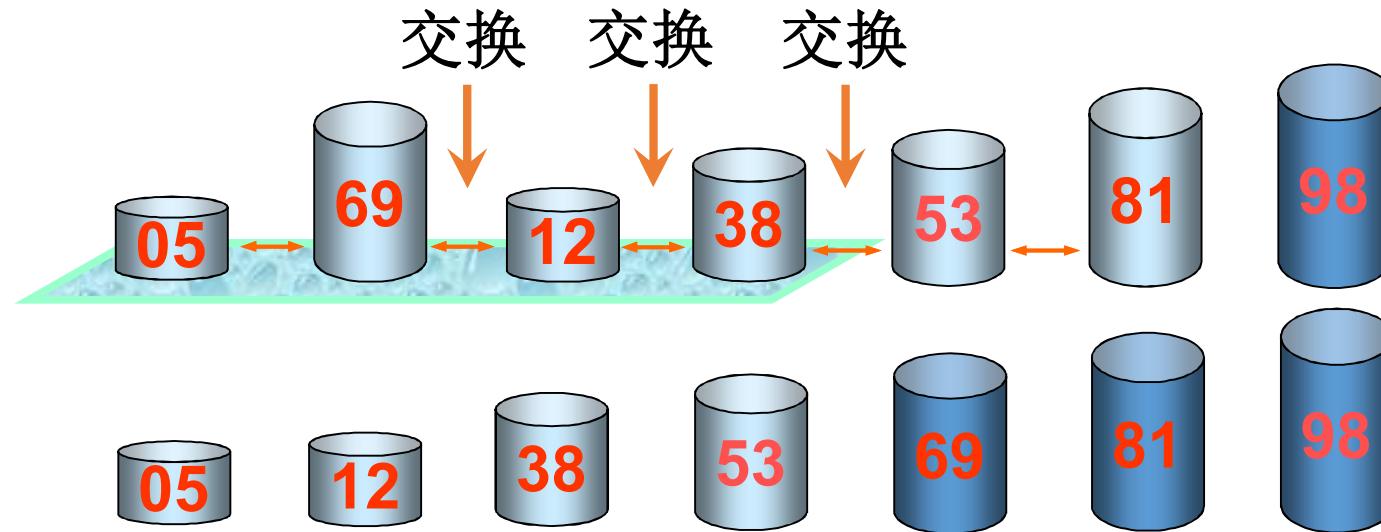
```
if (r[j]>r[j+1]) {  
    r[j]←→r[j+1];  
    exchange=j;  
}
```

关键问题(2): 如何确定起泡排序的范围?

解决方法:

设**bound**位置的记录是无序区的最后一个记录，则每趟起泡排序的范围是 $r[1] \sim r[bound]$ 。

在一趟排序后，从`exchange`位置之后的记录一定是有序的，所以 $bound=exchange$ 。



关键问题(2): 如何确定起泡排序的范围?

解决方法:

设**bound**位置的记录是无序区的最后一个记录，则每趟起泡排序的范围是r[1] ~ r[**bound**]。

在一趟排序后，从exchange位置之后的记录一定是有序的，所以**bound=exchange**。

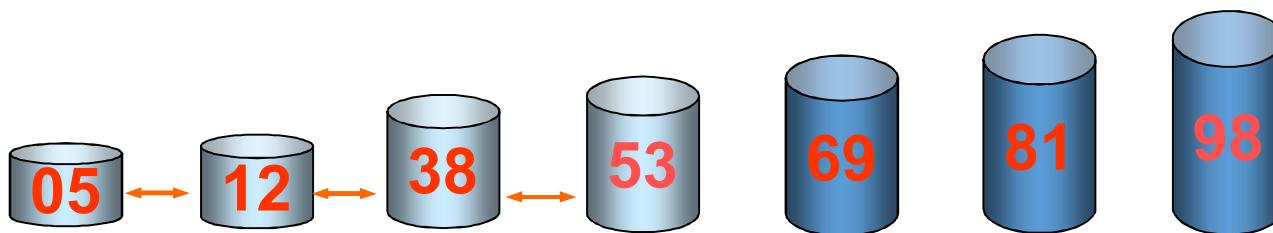
算法描述:

```
bound=exchange;  
for (j=1; j<bound; j++)  
    if (r[j]>r[j+1]) {  
        r[j]<==>r[j+1];  
        exchange=j;  
    }
```

关键问题(3): 如何判别起泡排序的结束?

解决方法:

在每一趟起泡排序之前，令**exchange**的初值为0，在以后的排序过程中，只要有记录交换，**exchange**的值就会大于0。这样，在一趟比较完毕，就可以通过**exchange**的值是否为0来判别是否有记录交换，从而判别整个起泡排序的结束。



关键问题(3): 如何判别起泡排序的结束?

解决方法:

在每一趟起泡排序之前，令**exchange**的初值为0，在以后的排序过程中，只要有记录交换，**exchange**的值就会大于0。这样，在一趟比较完毕，就可以通过**exchange**的值是否为0来判别是否有记录交换，从而判别整个起泡排序的结束。

算法描述:

```
while (exchange)
```

```
{
```

 执行一趟起泡排序；

```
}
```

起泡排序算法

```
void BubbleSort(int r[ ], int n)
{
    exchange=n;
    while (exchange)
    {
        bound=exchange;
        exchange=0;
        for (j=1; j<bound; j++)
            if (r[j]>r[j+1]) {
                r[j]↔r[j+1];
                exchange=j;
            }
    }
}
```

冒泡排序 (Bubble Sort)

- 在对象的初始排列已经按关键字从小到大排好序时，此算法只执行一趟冒泡，做 $n-1$ 次关键字比较，不移动对象。这是最好的情形。
- 最坏的情形是算法执行了 $n-1$ 趟冒泡，第 i 趟 ($1 \leq i < n$) 做了 $n-i$ 次关键字比较，执行了 $n-i$ 次对象交换。这样在最坏情形下总的**关键字比较次数** KCN 和**对象移动次数** RMN 为：

$$KCN = \sum_{i=1}^{n-1} (n - i) = \frac{1}{2} n(n - 1)$$

$$RMN = 3 \sum_{i=1}^{n-1} (n - i) = \frac{3}{2} n(n - 1)$$

平均情况：时间复杂度为 $O(n^2)$ 。

排序方法

- Insertion Sort (直接插入、希尔排序)
- Exchange sort (冒泡排序、**快速排序**) ★
- Selection Sort (简单选择排序、堆排序)
- Merge Sort (归并排序)
- Radix Sort (基数排序)

快速排序

改进的着眼点：在起泡排序中，记录的比较和移动是在**相邻**单元中进行的，记录每次交换只能上移或下移一个单元，因而总的比较次数和移动次数较多。

减少总的比较次数和移动次数



增大记录的比较和移动距离



较大记录从前面直接移动到后面
较小记录从后面直接移动到前面

快速排序的基本思想

首先选一个**轴值**（即比较的基准），通过一趟排序将待排序记录**分割**成独立的两部分，前一部分记录的关键码均**小于或等于**轴值，后一部分记录的关键码均**大于或等于**轴值，然后分别对这两部分重复上述方法，直到整个序列有序。



需解决的关键问题？

- (1) 如何选择轴值？
- (2) 如何实现分割（称一次划分）？
- (3) 如何处理分割得到的两个待排序子序列？
- (4) 如何判别快速排序的结束？

关键问题(1): 如何选择轴值?

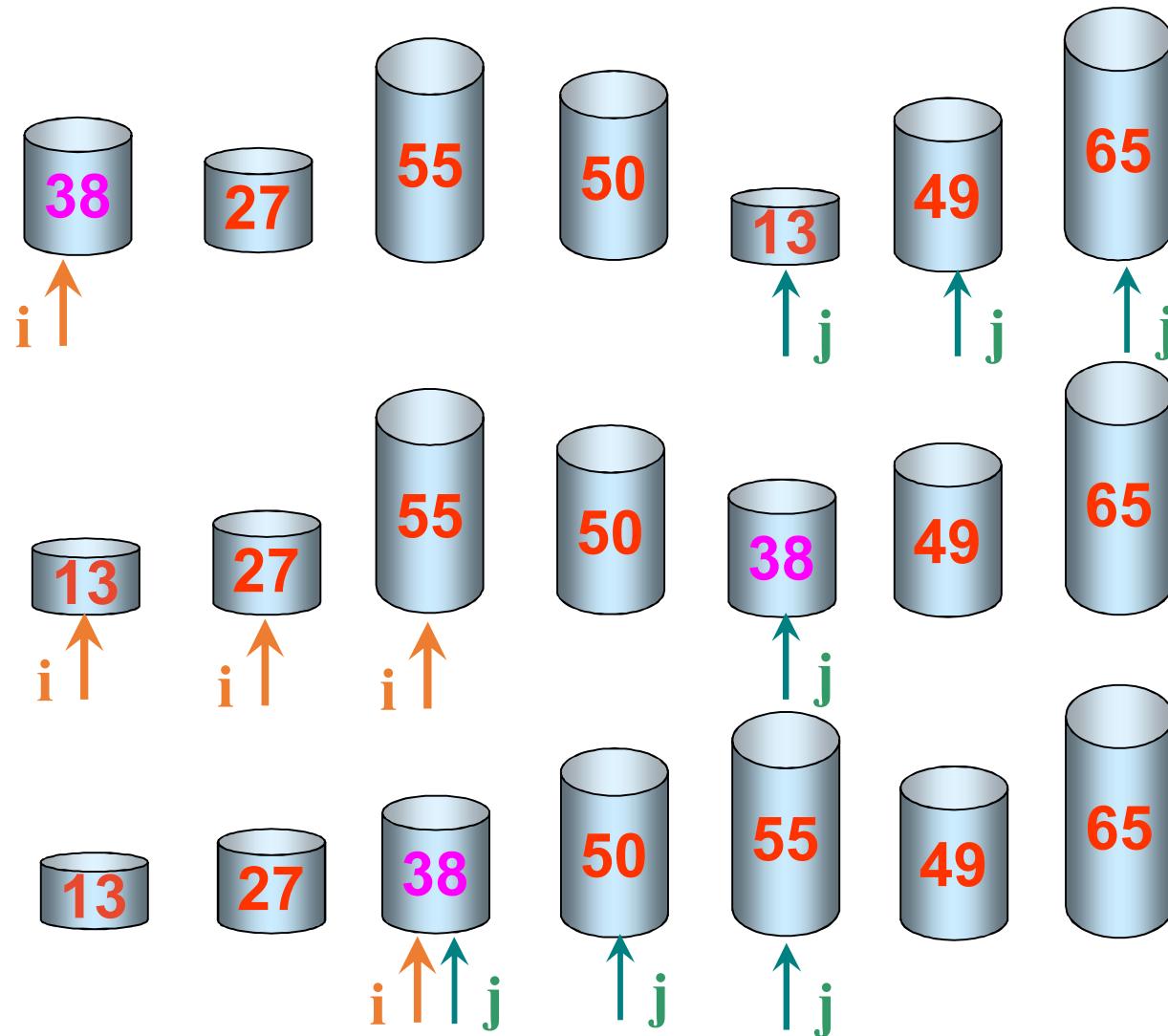
选择轴值的方法:

1. 使用第一个记录的关键码;
2. 选取序列中间记录的关键码;
3. 比较序列中第一个记录、最后一个记录和中间记录的关键码，取关键码居中的作为轴值并调换到第一个记录的位置;
4. 随机选取轴值。

选取不同轴值的后果:

决定两个子序列的长度，子序列的长度最好相等。

关键问题(2): 如何实现一次划分?



关键问题(2): 如何实现一次划分?

解决方法:

设待划分的序列是 $r[s] \sim r[t]$, 设参数*i*, *j*分别指向子序列左、右两端的下标*s*和*t*, 令 $r[s]$ 为轴值,

- (1) *j*从后向前扫描, 直到 $r[j] < r[i]$, 将 $r[j]$ 移动到 $r[i]$ 的位置, 使关键码小 (同轴值相比) 的记录移动到前面去;
- (2) *i*从前向后扫描, 直到 $r[i] > r[j]$, 将 $r[i]$ 移动到 $r[j]$ 的位置, 使关键码大 (同轴值比较) 的记录移动到后面去;
- (3) 重复上述过程, 直到*i=j*。

关键问题(2): 如何实现一次划分?

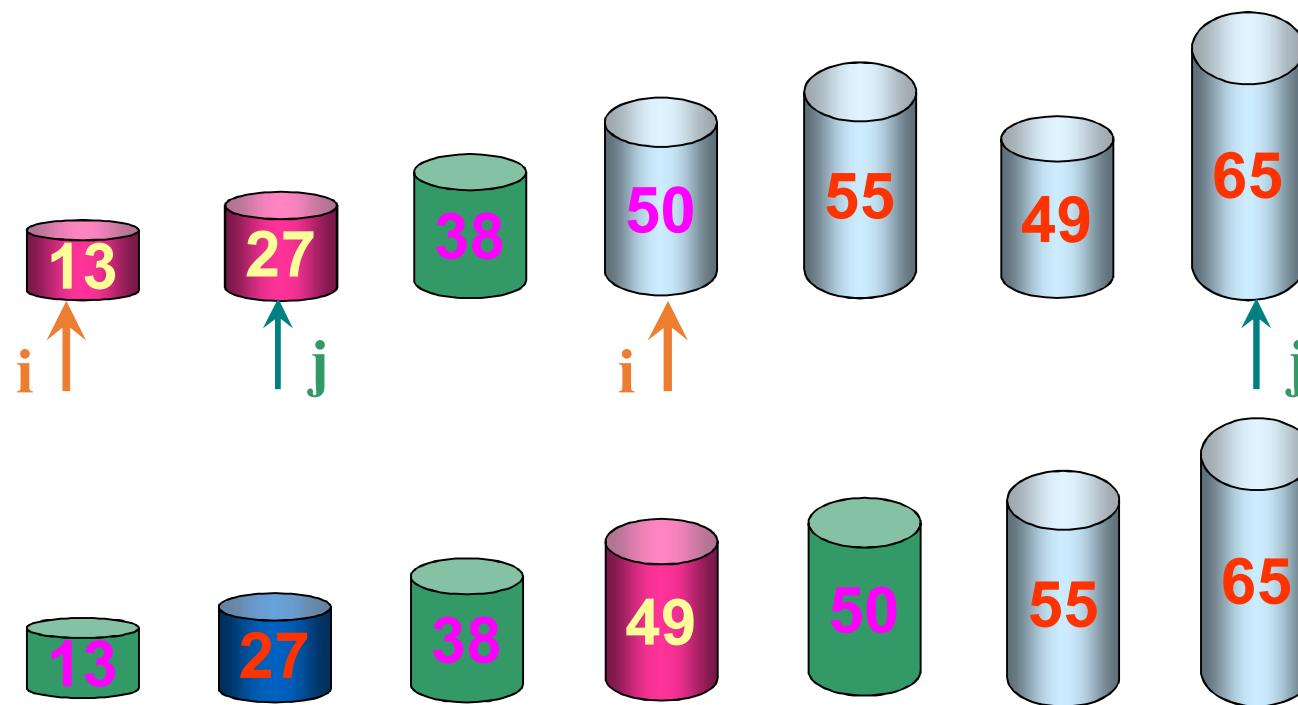
算法描述:

```
int Partition(int r[ ], int first, int end)
{
    i=first; j=end;          //初始化
    while (i<j)
    {
        while (i<j && r[i]<= r[j]) j--; //右侧扫描
        if (i<j) {
            r[i]→r[j]; i++; //将较小记录交换到前面
        }
        while (i<j && r[i]>= r[j]) i++; //左侧扫描
        if (i<j) {
            r[j]→r[i]; j--; //将较大记录交换到后面
        }
    }
    return i; //i为轴值记录的最终位置
}
```

关键问题(3): 如何处理分割得到的两个待排序子序列?

解决方法:

对分割得到的两个子序列递归地执行快速排序。



关键问题(3): 如何处理分割得到的两个待排序子序列?

算法描述:

```
void QuickSort (int r[ ], int first, int end )
{
    pivotpos = Partition (r, first, end ); //一次划分
    QuickSort (r, first, pivotpos-1);
                    //对前一个子序列进行快速排序
    QuickSort (r, pivotpos+1, end );
                    //对后一个子序列进行快速排序
}
```

关键问题(4): 如何判别快速排序的结束?

解决方法:

若待排序列中只有一个记录，显然已有序，否则进行一次划分后，再分别对分割所得的两个子序列进行快速排序（即递归处理）。

关键问题(4): 如何判别快速排序的结束?

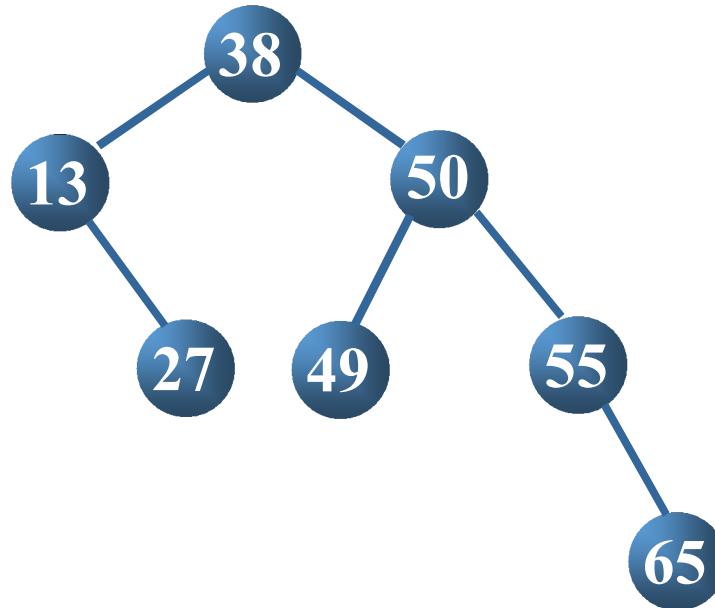
算法描述:

```
void QuickSort (int r[ ], int first, int end )  
{//在序列 first~end中递归地进行快速排序  
    if (first < end) {  
        pivotpos = Partition (r, first, end );  
        QuickSort (r, first, pivotpos-1);  
        QuickSort (r, pivotpos+1, end );  
    }  
}
```

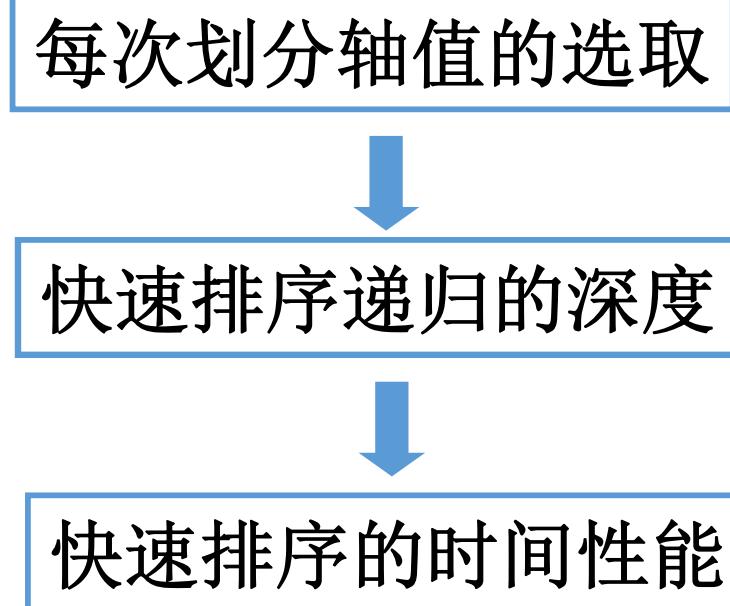
快速排序的时间性能分析

快速排序的递归执行过程可以用递归树描述。

例：{38, 27, 55, 50, 13, 49, 65}的快速排序递归树如下：



快速排序的时间性能分析



快速排序的时间性能分析

最好情况：

每一次划分对一个记录定位后，该记录的左侧子表与右侧子表的长度相同，为 $O(n \log_2 n)$ 。

$$\begin{aligned} T(n) &\leq 2T(n/2) + n \\ &\leq 2(2T(n/4) + n/2) + n = 4T(n/4) + 2n \\ &\leq 4(2T(n/8) + n/4) + 2n = 8T(n/8) + 3n \\ &\dots \dots \dots \\ &\leq nT(1) + n\log_2 n = O(n \log_2 n) \end{aligned}$$

快速排序的时间性能分析

最好情况：

每一次划分对一个记录定位后，该记录的左侧子表与右侧子表的长度相同，为 $O(n \log_2 n)$ 。

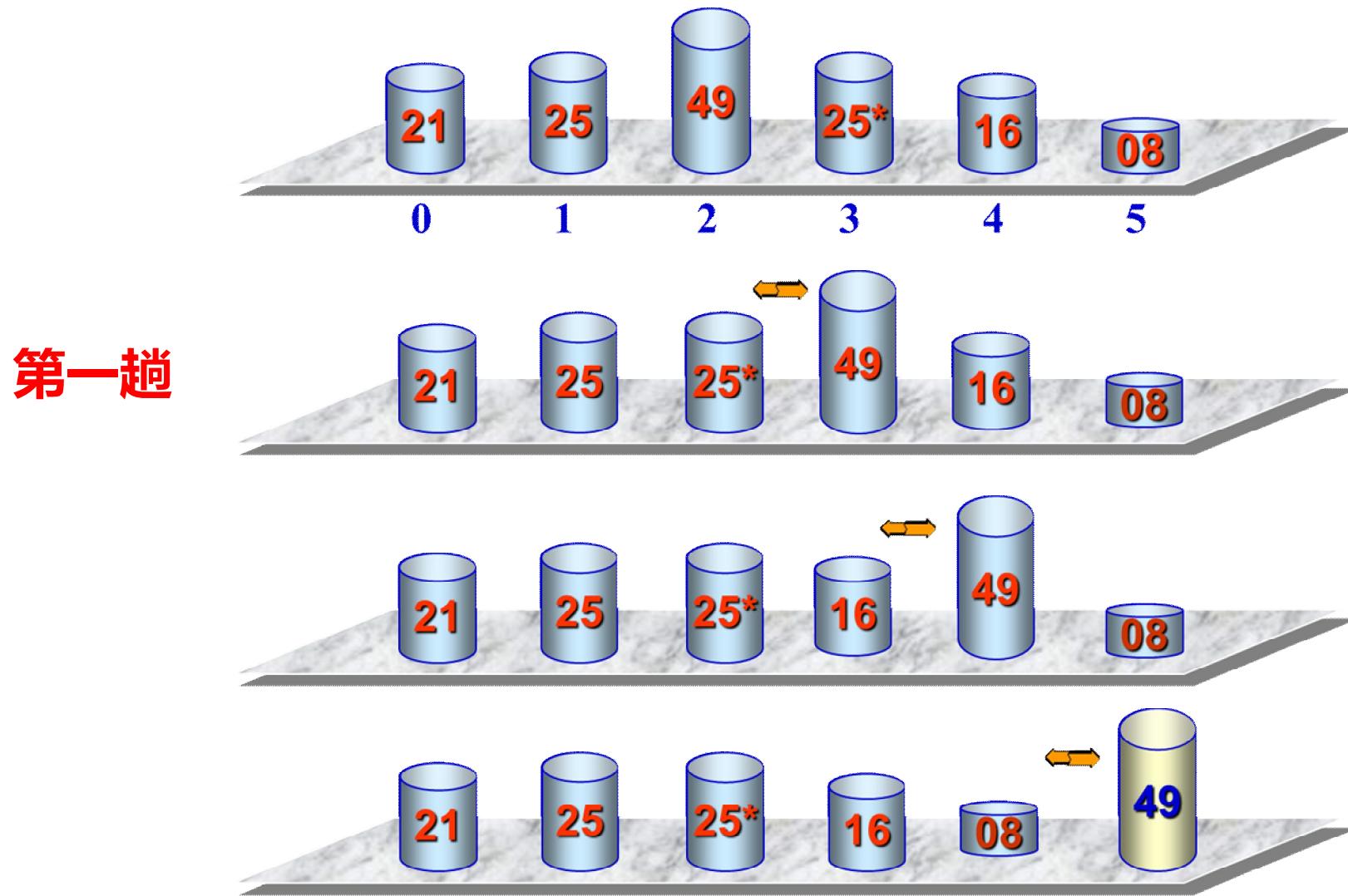
最坏情况：

每次划分只得到一个比上一次划分少一个记录的子序列（另一个子序列为空），为 $O(n^2)$ 。

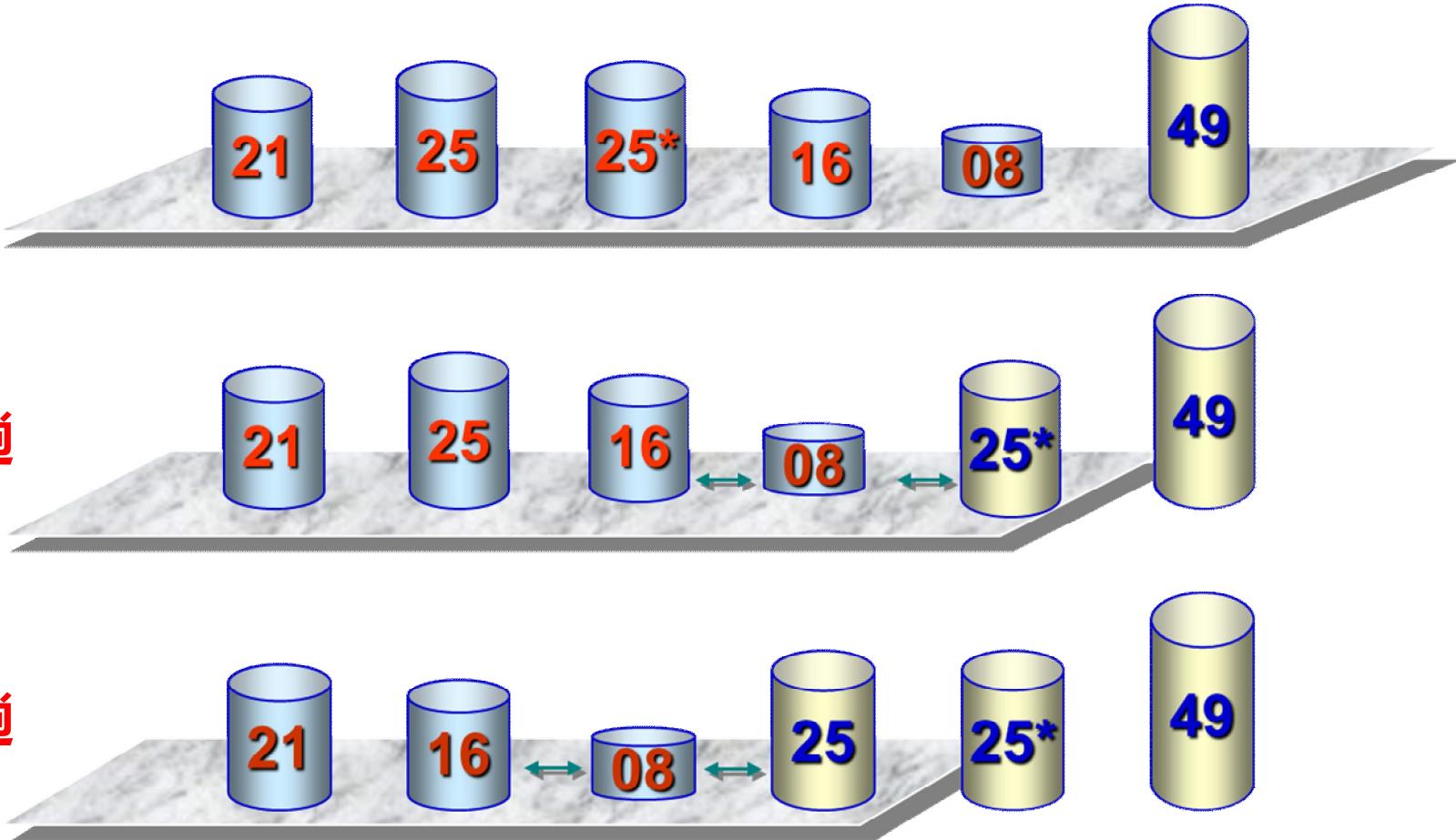
$$\sum_{i=1}^{n-1} (n - i) = \frac{1}{2} n(n - 1) = O(n^2)$$

平均情况：为 $O(n \log_2 n)$ 。

冒泡排序 (Bubble Sort)

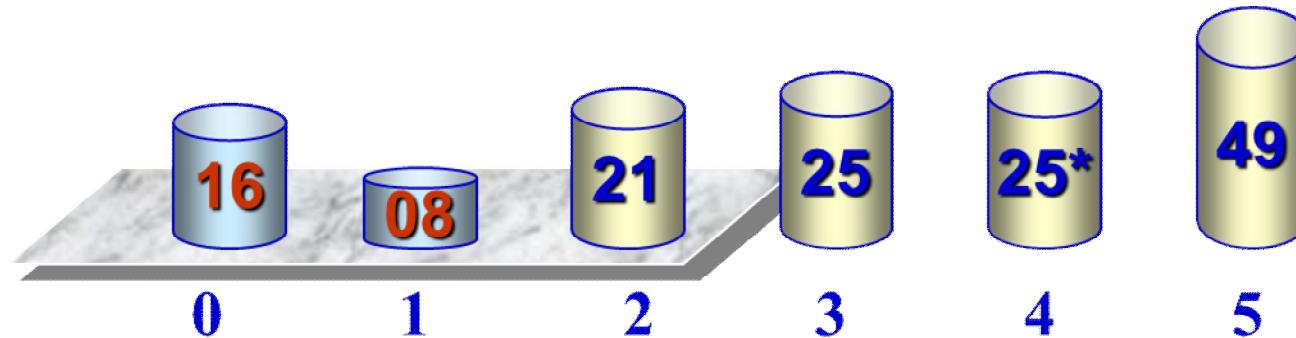


冒泡排序 (Bubble Sort)

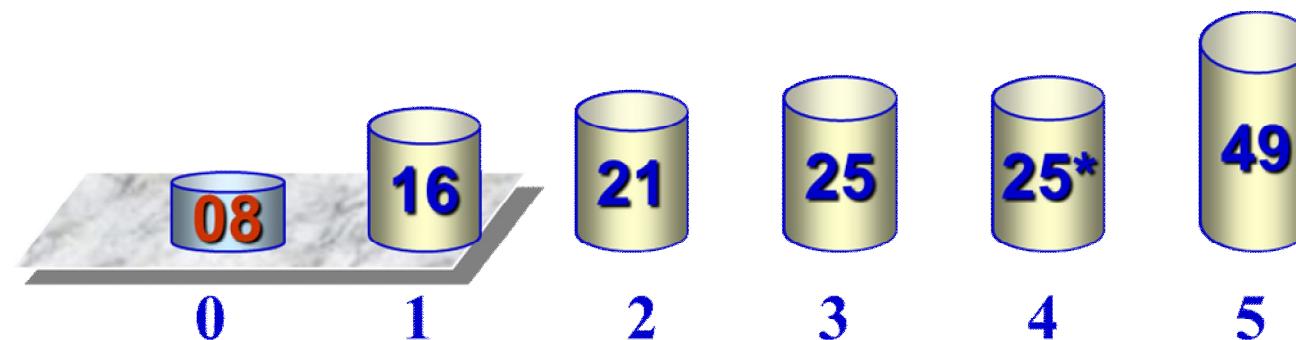


冒泡排序 (Bubble Sort)

第四趟

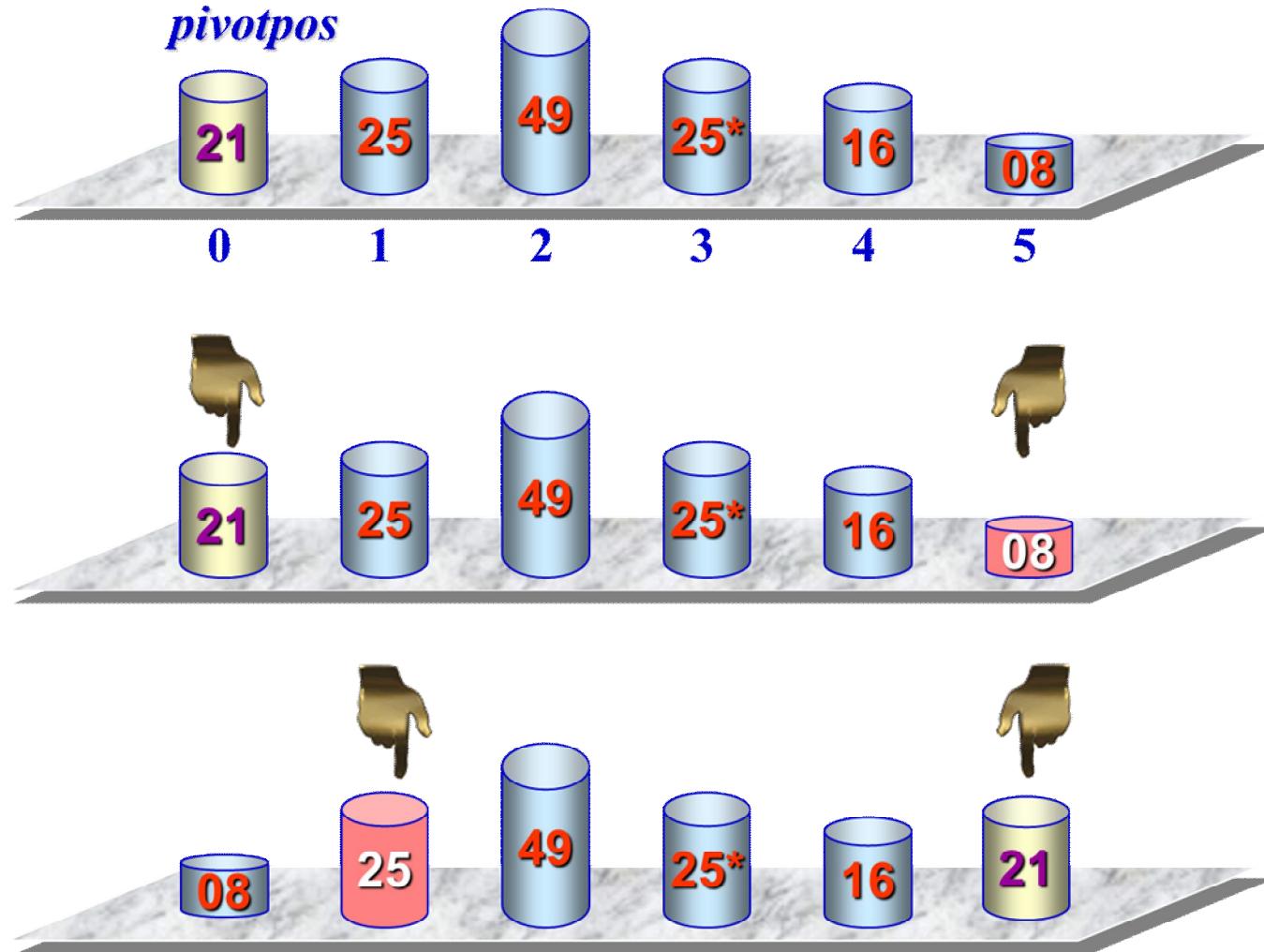


第五趟

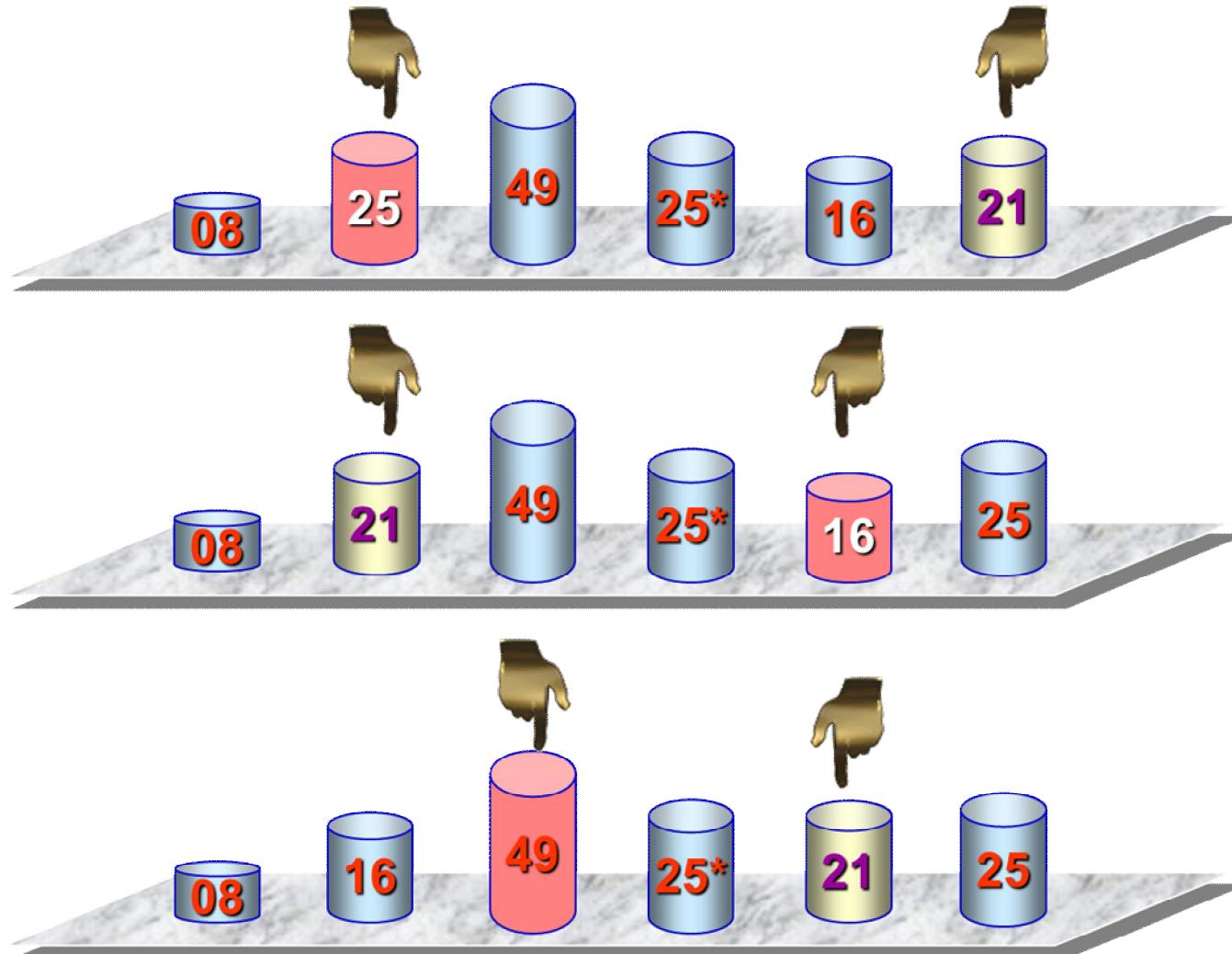


快速排序 (Quick Sort)

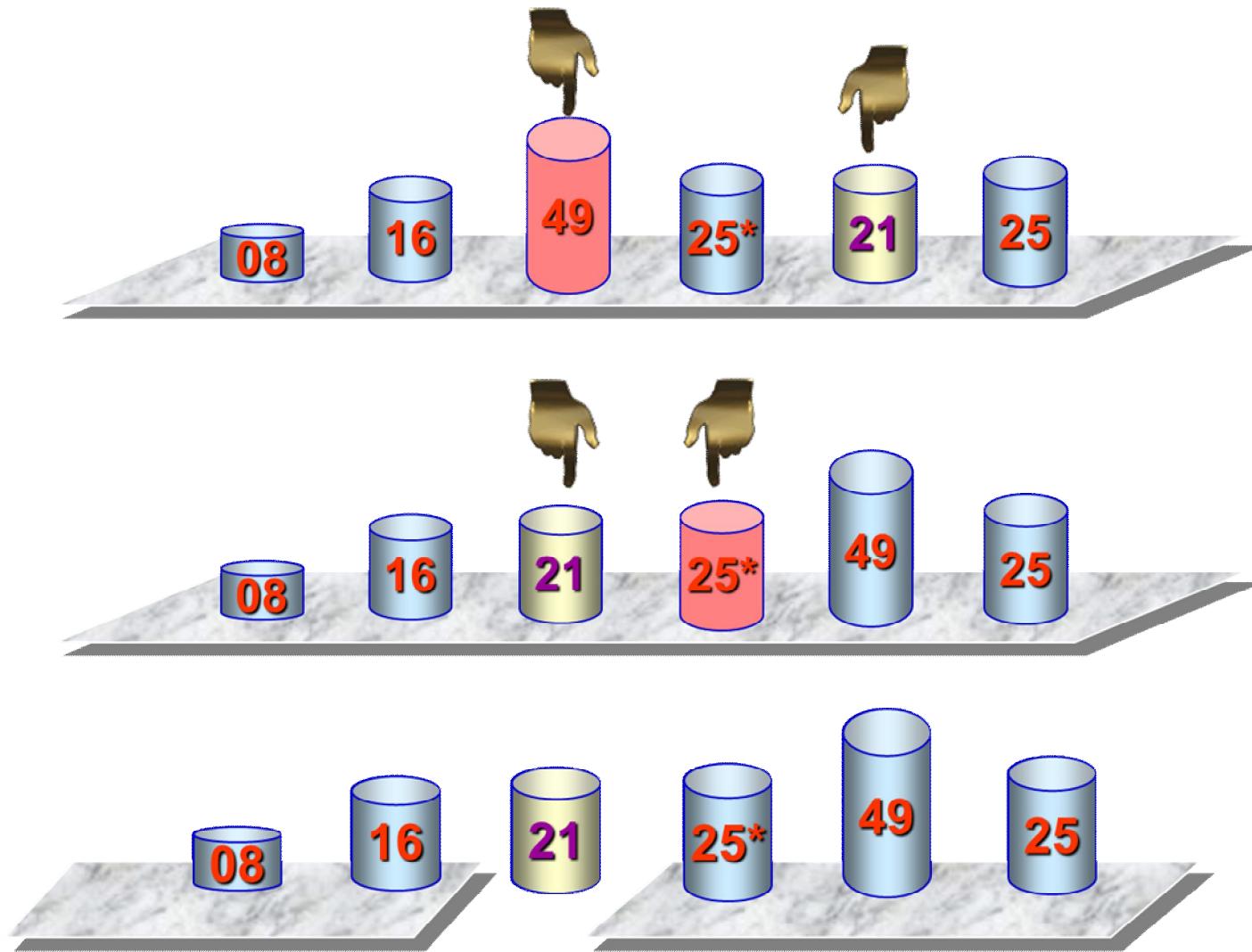
$i = 1$
划分



快速排序 (Quick Sort)

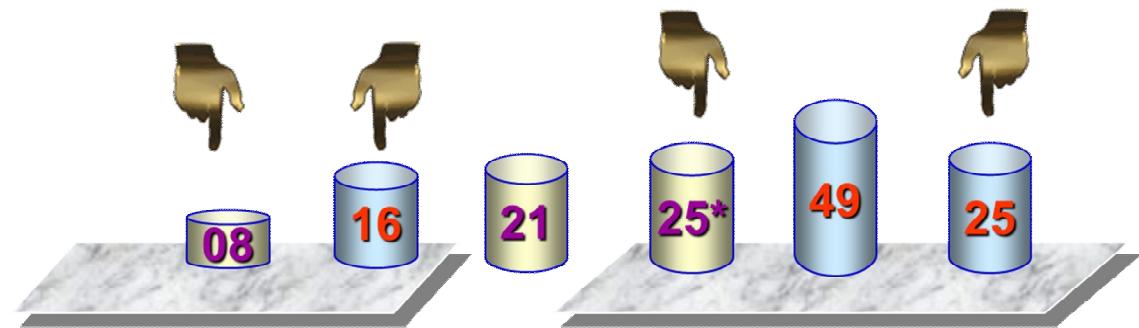


快速排序 (Quick Sort)

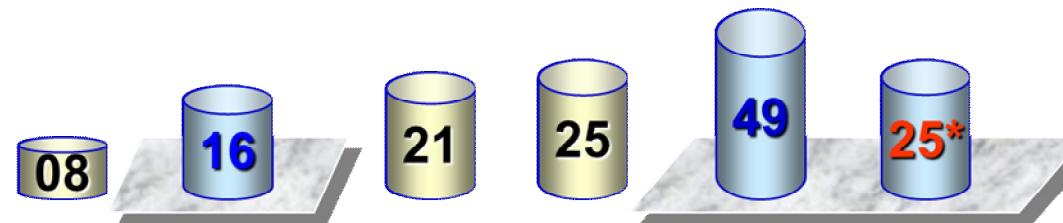


快速排序 (Quick Sort)

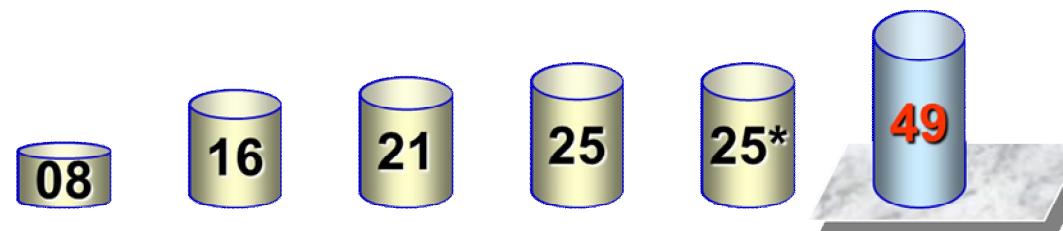
第二趟



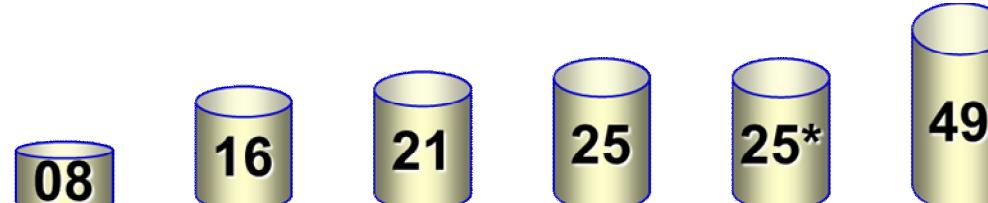
第三趟



第四趟



第五趟



Exercises

- 已知待排序记录 (70, 83, 100, 65, 10, 32, 7, 9) , 分别用插入排序 , 希尔排序 , 冒泡排序和快速排序算法进行排序 , 写出各趟排序结果。

插入排序	初始	(70), 83, 100, 65, 10, 32, 7, 9	冒泡排序	初始	70, 83, 100, 65, 10, 32, 7, 9
	1	(70, 83), 100, 65, 10, 32, 7, 9		1	70, 83, 65, 10, 32, 7, 9, 100
	2	(70, 83, 100), 65, 10, 32, 7, 9		2	70, 65, 10, 32, 7, 9, 83, 100
	3	(65, 70, 83, 100), 10, 32, 7, 9		3	65, 10, 32, 7, 9, 70, 83, 100
	4	(10, 65, 70, 83, 100), 32, 7, 9		4	10, 32, 7, 9, 65, 70, 83, 100
	5	(10, 32, 65, 70, 83, 100), 7, 9		5	10, 7, 9, 32, 65, 70, 83, 100
	6	(7, 10, 32, 65, 70, 83, 100), 9		6	7, 9, 10, 32, 65, 70, 83, 100
	7	(7, 9, 10, 32, 65, 70, 83, 100)			
希尔排序	初始	70, 83, 100, 65, 10, 32, 7, 9	快速排序	初始	(70), 83, 100, 65, 10, 32, 7, 9
	1 d=4	10, 32, 7, 9, 70, 83, 100, 65		1	(9, 7, 32, 65, 10), 70, (100, 83)
	2 d=2	7, 9, 10, 32, 70, 65, 100, 83		2	(7), 9, (32, 65, 10), 70, 83, 100
	3 d=1	7, 9, 10, 32, 65, 70, 83, 100		3	7, 9, 10, 32, 65, 70, 83, 100