计算思维与实践

哈尔滨工业大学（深圳）
计算机科学与技术学院
大数据技术中心
张保权

课程内容安排

程序设计思想与数据操作方法融会贯通，内容由浅入深

第十三讲 排序算法——学习内容

13.1 概述

13.2 插入排序

13.3 交换排序

13.4 选择排序

13.5 内部排序方法的比较

13.1 概述

- 排序是计算机程序设计中的一种重要操作，它的功能是将一个数据元素(或记录)的任意序列，重新排列成一个按关键字有序的序列。
- 排序的目的之一是方便数据查找。

13.1 概述

内部排序和外部排序

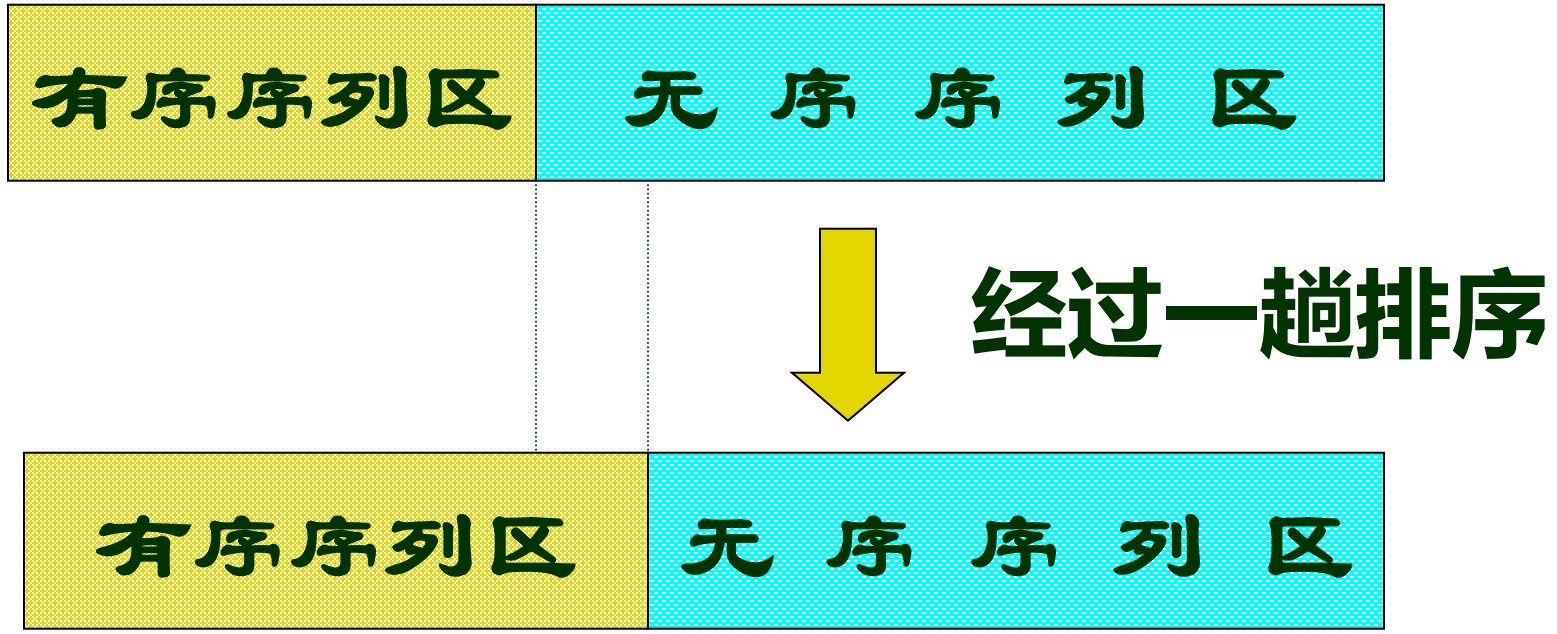
在排序过程中，只使用计算机的内存存放待排序记录，称这种排序为内部排序。

排序期间文件的全部记录不能同时存放在计算机的内存中，要借助计算机的外存才能完成排序，称之为“外部排序”。

内外存之间的数据交换次数就成为影响外部排序速度的主要因素。

13.1.1 概述——内部排序

内部排序的过程是一个逐步扩大记录的有序序列长度的过程。



13.1.1 概述——内部排序

基于不同的“扩大” 有序序列长度的方法，内部排序方法大致可分下列几种类型：

插入类　　交换类　　选择类

归并类　　其它方法

13.1.1 概述——内部排序

排序算法的性能：

- **基本操作。内排序在排序过程中的基本操作：**
  - **比较：关键码之间的比较；**
  - **移动：记录从一个位置移动到另一个位置。**
- **辅助存储空间。**
  - **辅助存储空间是指在数据规模一定的条件下，除了存放待排序记录占用的存储空间之外，执行算法所需要 的其他额外存储空间。**
- **算法本身的复杂度。**

13.1.2 概述——排序算法的稳定性

排序方法的稳定和不稳定

在排序过程中，有若干记录的**关键字相等**，即$K_i=K_j(1\le i\le n，1\ 1\le j\le n，i\neq j)$，在排序前后，含相等关键字的记录的**相对位置保持不变**，即排序前R_i在R_j之前，排序后R_i仍在R_j之前，称这种排序方法是**稳定的**；

反之，若可能使排序后的序列中R_i在R_j之后，称所用排序方法是**不稳定的**。

13.1.3 概述——排序对象：顺序表


```
#define MAXSIZE  1000 // 待排顺序表最大长度
typedef  int  KeyType;  // 关键字类型为整数类型
    typedef  struct {
       KeyType   key;                    // 关键字项
        InfoType  otherinfo;             // 其它数据项
    } RcdType;                           // 记录类型
typedef  struct {
   RcdType    r[MAXSIZE+1];             // r[0]闲置
   int            length;               // 顺序表长度
} SqList;                                // 顺序表类型
```

13.2 插入排序

13.2 插入排序——算法要点

将无序子序列中的一个或几个记录“插入”到有序序列中，从而增加记录的有序子序列的长度。

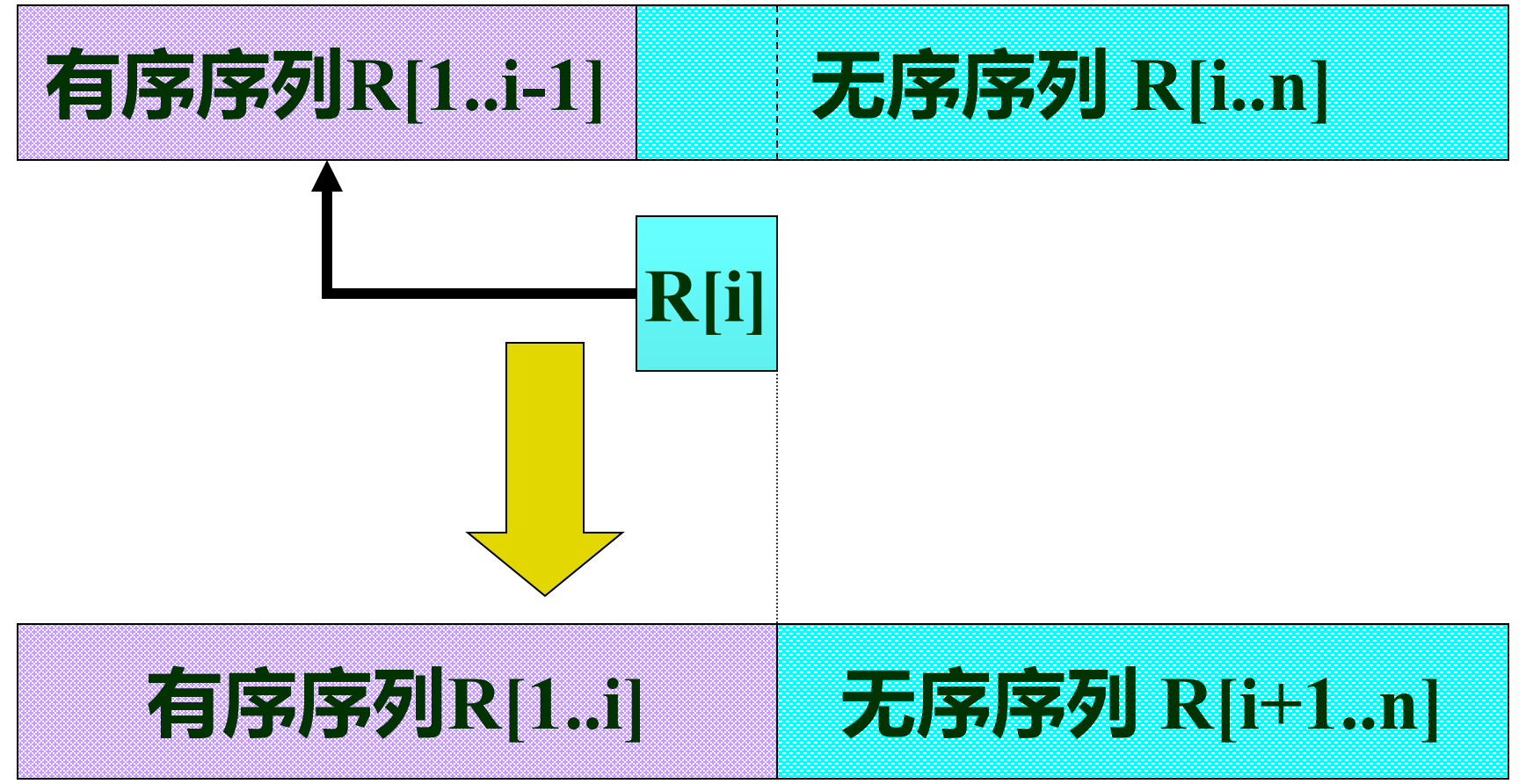
13.2 插入排序——算法类型

13.2.1 直接插入排序

13.2.2 折半插入排序

13.2.1 直接插入排序——基本思想

一趟直接插入排序的基本思想：



13.2.1 直接插入排序——算法概述

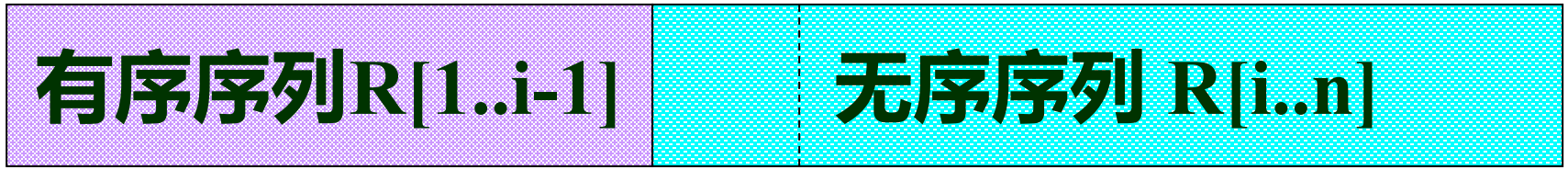
直接插入排序算法概述

(1)将序列中的第1个记录看成是一个有序的子序列;

(2)从第2个记录起逐个进行插入，直至整个序列变成按关键字有序序列为止;

每一趟排序需要进行比较、后移记录、插入适当位置。从第二个记录到第n个记录共需n-1趟。

13.2.1 插入排序——基本步骤



实现“一趟插入排序”可分三步进行：

1．在R[1..i-1]中查找R[i]的插入位置，

$R[1..j].key \le R[i].key < R[j+1..i-1].key$；

2．将R[j+1..i-1]中的所有记录均后移一个位置；

3．将R[i] 插入(复制)到R[j+1]的位置上。

13.2.1 直接插入排序——如何查找插入位置

算法的实现要点：

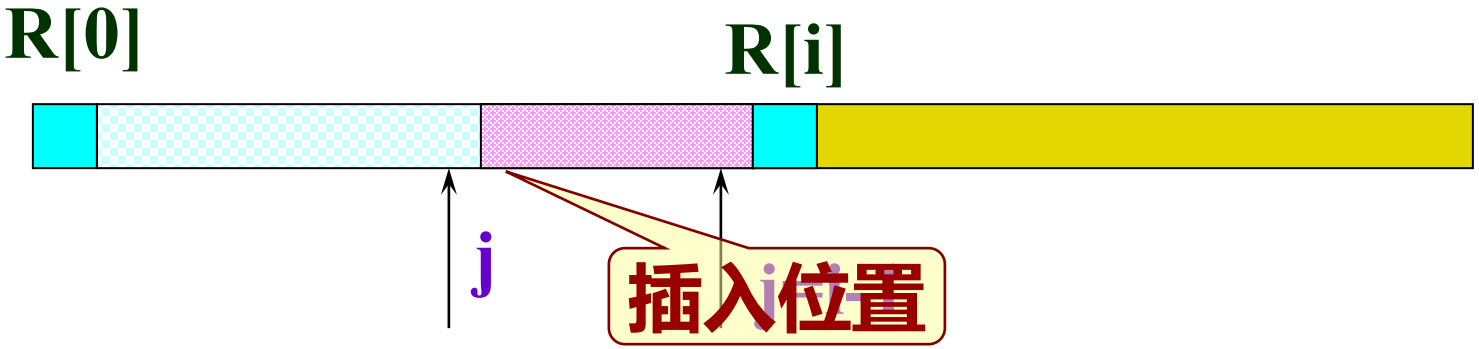
利用 “顺序查找” 实现

“在R[1..i-1]中查找R[i]的插入位置”

13.2.1 直接插入排序——如何查找插入位置

从R[i-1]起向前进行顺序查找，

监视哨设置在R[0]；



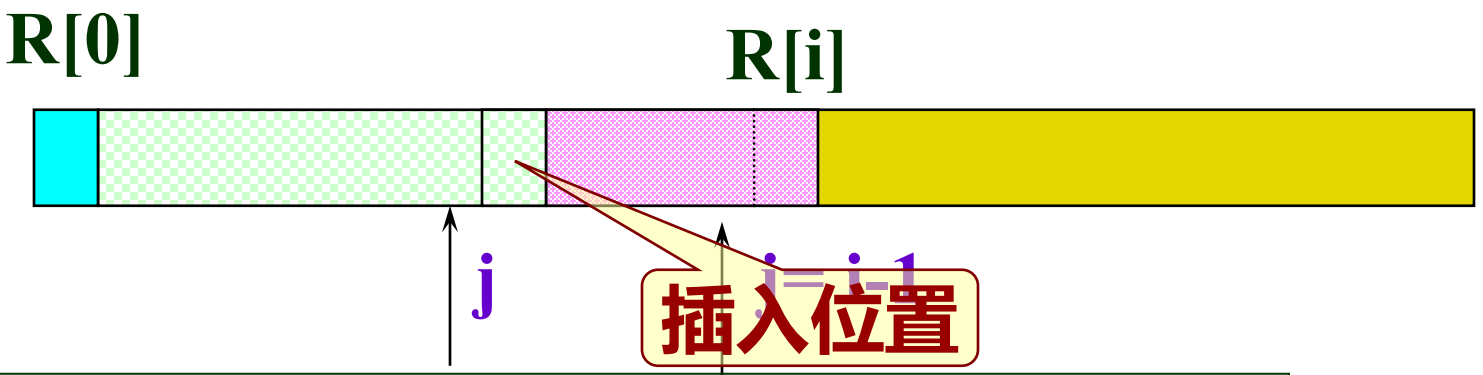

```
R[0] = R[i];            // 设置“哨兵”
for (j=i-1; R[0].key<R[j].key; --j);
                        // 从后往前找
```

循环结束表明R[i]的插入位置为 *j +1*

13.2.1 直接插入排序——如何查找插入位置

对于在查找过程中找到的那些关键字不小于R[i].key的记录，并在查找的同时实现记录向后移动；

```
for (j=i-1; R[0].key<R[j].key; --j);
    R[j+1] = R[j]
```



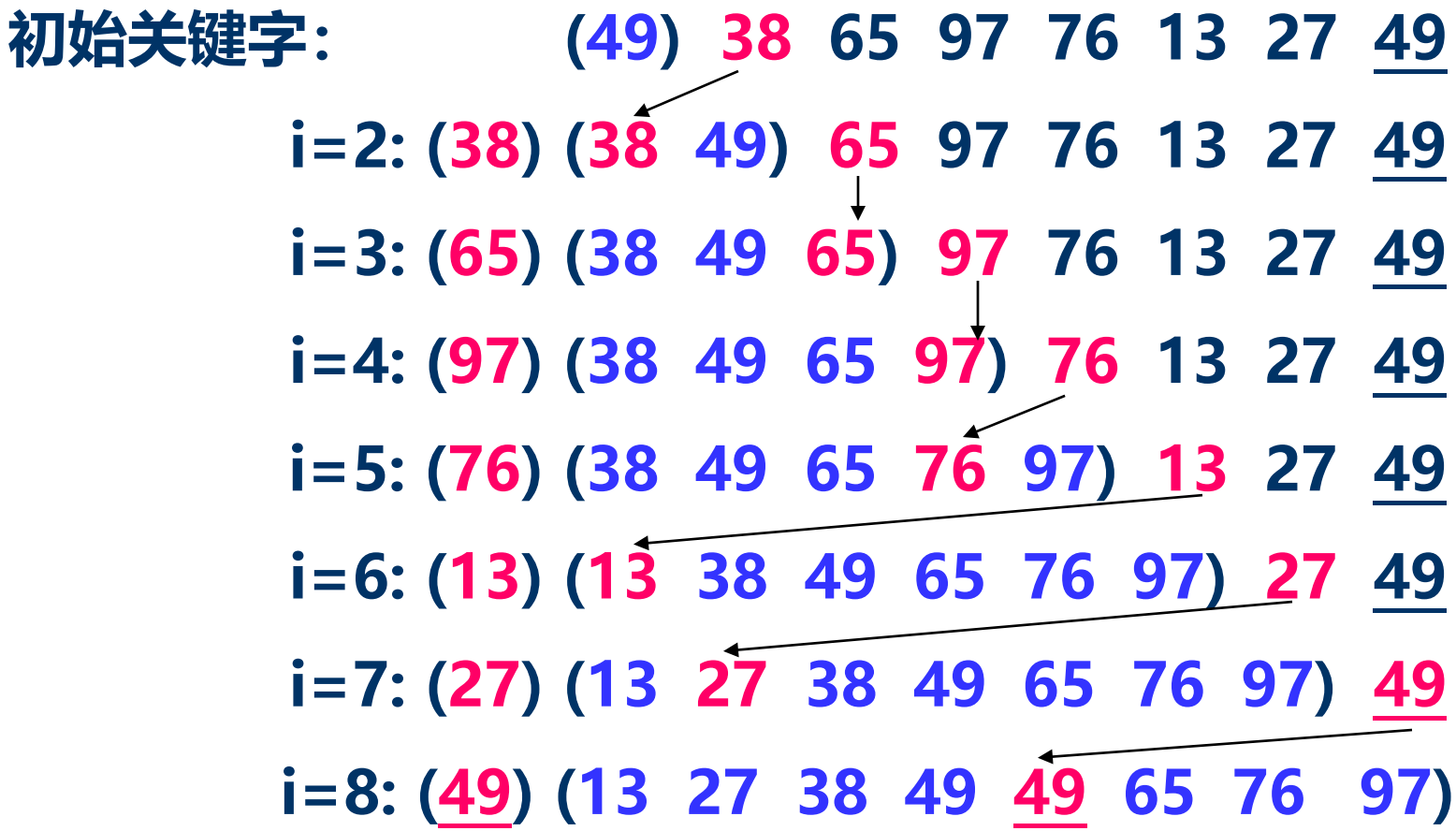
上述循环结束后可以直接进行“插入”

13.2.1 直接插入排序——代码实现


```
void InsertionSort ( SqList *L ) { //升序
  // 对顺序表 L 作直接插入排序。
   for ( i=2; i<=L->length; ++i )
      if (L->r[i].key < L->r[i-1].key) {
          L->r[0] = L->r[i];                  // 复制为监视哨
          for ( j=i-1; L->r[0].key < L->r[j].key;  -- j )
              L->r[j+1] = L->r[j];          // 记录后移
          L->r[j+1] = L->r[0];             // 插入到正确位置
      }
} // InsertSort
```

13.2.1 直接插入排序——示例演示

例1 直接插入排序的过程。



13.2.1 直接插入排序——基本操作

实现直接插入排序的基本操作有两个：

(1) “比较”序列中两个关键字的大小；

(2) “移动”记录。

13.2.1 直接插入排序——时间复杂度

最好的情况（关键字在记录序列中顺序有序）：

“比较”的次数： $\sum_{i=2}^{n} 1 = n - 1$

“移动”的次数： **0**

时间复杂度

比较O(n),移动O(1);

13.2.1 直接插入排序——时间复杂度

最坏的情况（关键字在记录序列中逆序有序）：

“比较”的次数：

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

“移动”的次数：

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

时间复杂度

比较$O(n^2)$,移动$O(n^2)$;

13.2.1 直接插入排序——算法分析

(1)稳定性

直接插入排序是稳定的排序方法。

R[0].key<R[j].key

(2)算法效率

a.时间复杂度

最好情况O(n):比较O(n),移动O(1);

最坏情况$O(n^2)$:比较$O(n^2)$,移动$O(n^2)$;

平均$O(n^2)$

b.空间复杂度

O(1)。

13.2.1 直接插入排序——算法分析

算法的性能分析

- 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。

改进的直接插入排序-----折半插入排序

直接插入排序，在插入第 i（$i>1$）个记录时，前面的 i-1 个记录已经排好序，则在寻找插入位置时，可以用**折半查找来代替顺序查找**，从而较少比较次数。

13.2.1 折半插入排序——算法思想

(1)在直接插入排序中，r[1..i-1] 是一个按关键字有序的有序序列;

(2)可以利用折半查找实现“在r[1..i-1]中查找r[i]的插入位置”；

(3)称这种排序为折半插入排序。

13.3 交换排序

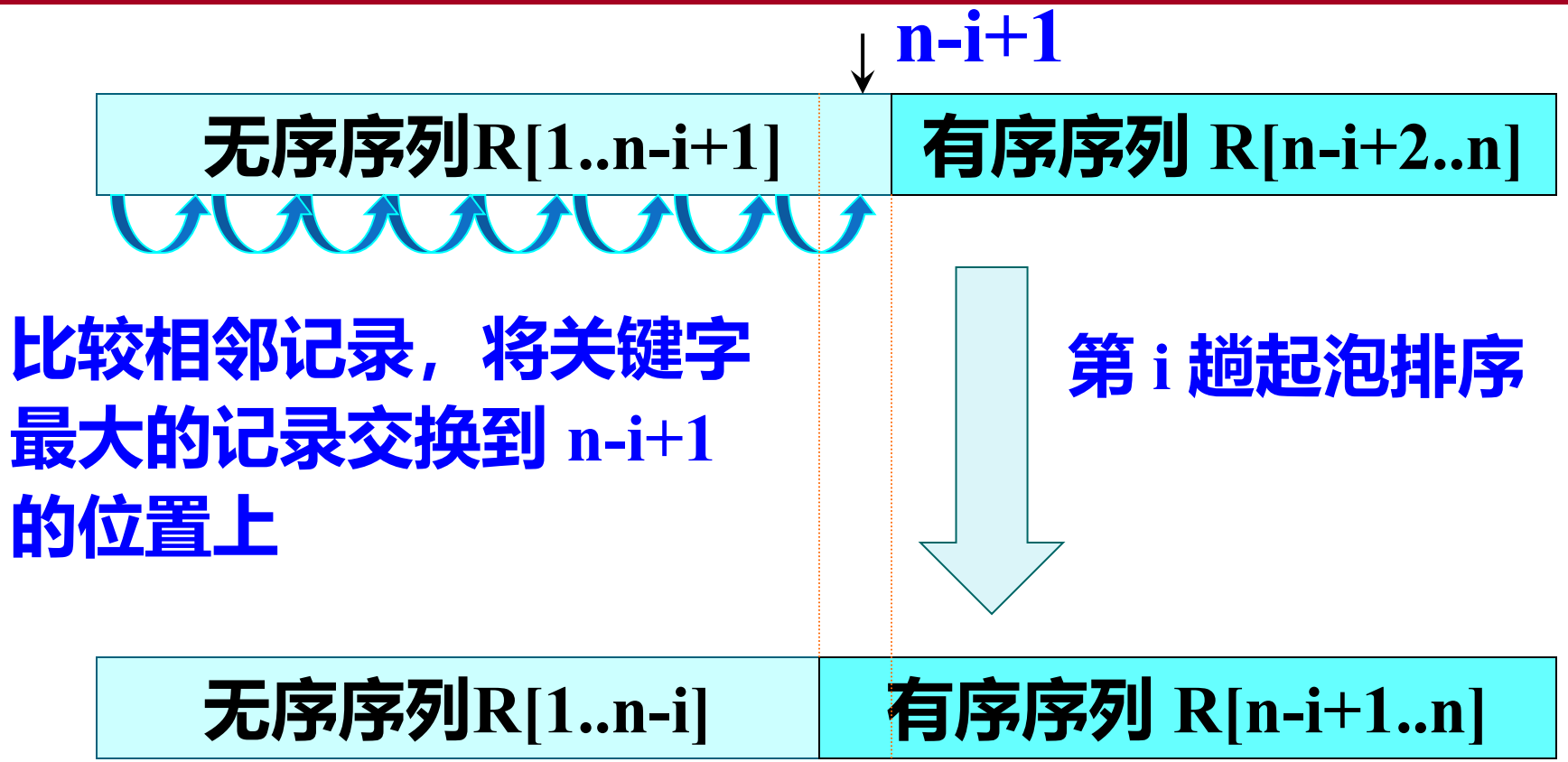
13.3 交换排序——算法要点

通过“交换”无序序列中的记录从而得到其中关键字最小或最大的记录，并将它加入到有序子序列中，以此方法增加记录的有序子序列的长度。

13.3 交换排序——算法类型

13.3.1 起泡排序（冒泡排序）

13.3.1 起泡排序——基本思想



起泡排序的特点：

有序序列中的关键字值都比无序序列中的大

13.3.1 起泡排序——算法概述

(1) 从第一个记录开始，两两记录比较，若F[1].key>F[2].key，则将两个记录交换；

(2)第1趟比较结果将序列中关键字最大的记录放置到最后一个位置，称为“沉底”；

(3)n个记录最多比较n-1遍(趟)。

(4) 若一趟排序中没有交换，即可结束程序

13.3.1 起泡排序——代码实现


```
void BubbleSort(SqList *L )
{   int i,j,noswap; RcdType  temp;
    for(i=1;i<=n-1;i++)
  {    noswap=TRUE;
       for(j=1;j<=n-i;j++)
          if (L->r[j].key>L->r[j+1].key)
             {temp=L->r[j]; L->r[j]=L->r[j+1]; L->r[j+1]=temp;
              noswap=FALSE; }
       if (noswap)  break;
  }
}
```

13.3.1 起泡排序——示例演示

例：关键字序列 T=(21，25，49，25*，16，08），请写出冒泡排序的具体实现过程。

初态：	21，25，49，25*，16，08
第1趟	21，25，25*，16，08，49
第2趟	21，25，16，08，25*，49
第3趟	21，16，08，25，25*，49
第4趟	16，08，21，25，25*，49
第5趟	08，16，21，25，25*，49

稳定

13.3.1 起泡排序——时间复杂度

最好的情况（关键字在记录序列中顺序有序）：
只需进行一趟起泡

“比较”的次数：

n-1

“移动”的次数：

0

最坏的情况（关键字在记录序列中逆序有序）：
需进行n-1趟起泡，每一次比较交换都移动3次

“比较”的次数：

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

“移动”的次数：

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

13.3.1 起泡排序——算法分析

(1)稳定性

起泡排序是稳定的排序方法。

(2)时间复杂度

最好情况：比较O(n),移动O(1)

最坏情况：比较$O(n^2)$,移动$O(n^2)$

平均情况：$O(n^2)$

(3)空间复杂度

O(1)

13.4 选择排序

13.4 选择排序——算法要点

从记录的无序子序列中“选择”关键字最小或最大的记录，并将它加入到有序子序列中，以此方法增加记录的有序子序列的长度。

13.4 选择排序——算法类型

13.4.1 简单选择排序

13.4.1 简单选择排序——算法概述

(1)第一次从n个关键字中选择一个最小值,确定第一个;

(2)第二次再从剩余元素中选择一个最小值,确定第二个;

(3)共需n-1次选择。

13.4.1 简单选择排序——算法步骤

设需要排序的表是A[n+1]:

(1)第一趟排序是在无序区A[1]到A[n]中选出最小的记录，将它与A[1]交换，确定最小值;

(2)第二趟排序是在A[2]到A[n]中选关键字最小的记录，将它与A[2]交换，确定次小值;

(3)第i趟排序是在A[i]到A[n]中选关键字最小的记录，将它与A[i]交换;

(4)共n-1趟排序。

13.4.1 简单选择排序——算法区别

简单选择排序与起泡排序的区别在：

起泡排序每次比较后 ，如果发现顺序不对立即进行交换，而选择排序不立即进行交换而是找出最小关键字记录后再进行交换。

和直接插入排序的区别：

直接插入排序是从无序区随意选择一个，插入有序区相应位置

简单选择排序是从无序区中选择最小（大），直接放入有序区最小或最大位置

13.4.1 简单选择排序——代码实现


```
void  SelectSort(SqList *L)
{int i,j,low;
  for(i=1;i<L->length;i++)
   {low=i;
    for(j=i+1;j<=L->length;j++)
       if(L->r[j].key<L->r[low].key)
           low=j;
     if(i!=low)
       {L->r[0]=L->r[i];L->r[i]=L->r[low];L->r[low]=L->r[0];
       }
     }
}
```

时间复杂度为$O(n^2)$。

13.4.1 简单选择排序——算法分析

(1)稳定性

简单选择排序方法是不稳定的。例（2,2',1）

(2)时间复杂度

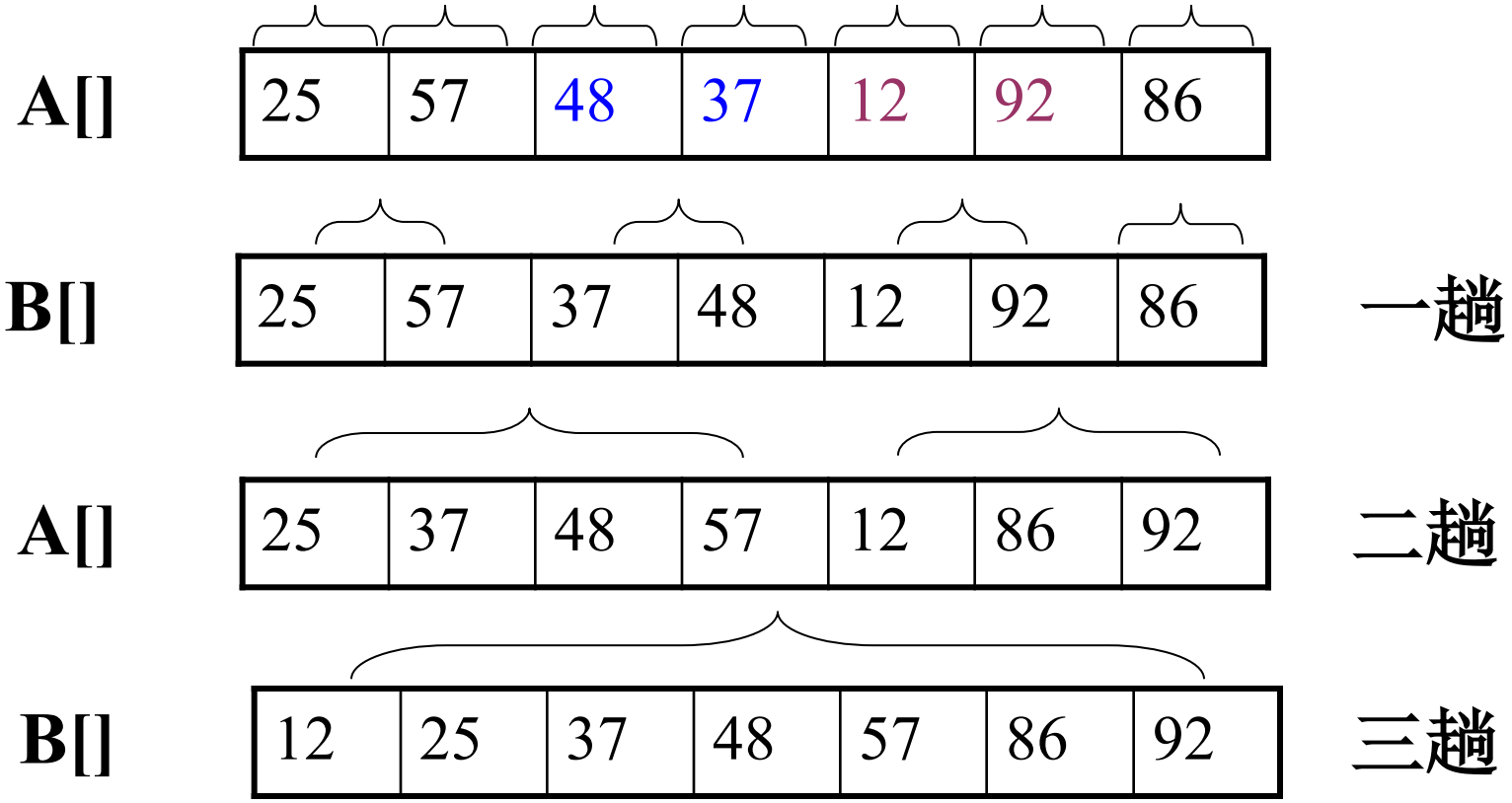
选择排序的每一趟排序都需要遍历整个无序序列，因此无论序列是否已经有序，比较次数始终是$O(n^2)$,

(3)空间复杂度

O(1)

13.5 归并排序——示例演示

例 给定排序码25，57，48，37，12，92，86，写出二路归并排序过程。



13.5 归并排序——算法分析

(1) 稳定性

归并排序是稳定的排序方法。

(2) 时间复杂度

每趟归并所花时间比较移动都是O(n)；

归并趟数为$\log_2 n$;

时间复杂度为$O(n\log_2 n)$。

(3) 空间复杂度

O(n)

13.4 选择排序

13.5 内部排序方法的比较——考虑因素

- 对排序算法应该从以下几个方面综合考虑：
  - (1)时间复杂度；
  - (2)空间复杂度；
  - (3)稳定性；
  - (4)算法简单性；
  - (5)待排序记录个数n的大小；
  - (6)记录本身信息量的大小；
  - (7)关键字值的分布情况。

13.5 内部排序方法的比较——时间复杂度

- ⑴时间复杂度比较：

排序方法	平均情况	最好情况	最坏情况
直接插入排序	$O(n^2)$	$O(n)$	$O(n^2)$
起泡排序	$O(n^2)$	$O(n)$	$O(n^2)$
直接选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$

13.5 内部排序方法的比较——空间复杂度和稳定性

- (2)空间复杂度比较和稳定性比较：

排序方法	辅助空间	稳定性/不稳定举例
直接插入排序	$O(1)$	是
起泡排序	$O(1)$	是
直接选择排序	$O(1)$	否/2,2',1

13.5 内部排序方法的比较——简单性和比较次数

- **(4)算法简单性比较：从算法简单性看，**
  - **一类是简单算法，包括直接插入排序、直接选择排序和起泡排序；**
  - **另一类是改进后的算法，包括希尔排序、堆排序、快速排序和归并排序，这些算法都很复杂。**

- **(5)待排序的记录个数比较：从待排序的记录个数n的大小看，**
  - **n越小，采用简单排序方法越合适；**

13.5 内部排序方法的比较——记录规模

- (1)若 n 较小(如 $n \leq 50$)，可采用直接插入或直接选择排序。
  当记录规模较小时，直接插入排序较好；否则因为直接选择移动的记录数少于直接插人，应选直接选择排序为宜。

- (2)若文件初始状态基本有序(指正序)，则应选用直接插入、冒泡或随机的快速排序为宜；且以直接插入排序最佳。

- (3)若 n 较大，则应采用时间复杂度为O($n\lg n$)的排序方法：
  快速排序、归并排序。

13.5 内部排序方法的比较——适用场景

- 快速排序是目前基于比较的内部排序中被认为是最好的方法，当待排序的关键字是随机分布时，快速排序的平均时间最短；
- 若要求排序稳定，则可选用归并排序，基数排序稳定性最佳。

13.5 内部排序方法的比较——统计结果

对100万个数据排序统计结果(单位：毫秒)

序号	排序方法	平均情况	最坏情况（逆序）	最好情况（正序）
1	冒泡排序	549432.000	1534035.000	366936.000
2	选择排序	478694.000	587240.000	367658.000
3	插入排序	253115.000	515621.000	0.897
4	归并排序	70.000	140.000	61.000
5	快速排序	315.000	93.000	30.000

本章小结

- ✓ 熟练掌握：
  - 直接插入排序、冒泡排序、选择排序的思想和算法。充分了解各种排序算法的应用背景和优缺点。
- ✓ 重点学习：
  - 加强各种排序算法在实际应用中的训练，提高实际应用水平。