第八章 排序（一）

学习目标

- 掌握排序的基本概念和常用术语
- 掌握插入排序、希尔排序；起泡排序、快速排序；选择排序；归并排序的基本思想、算法原理、排序过程和算法实现。
- 掌握各种排序算法的性能及其分析方法，以及各种排序方法的比较和选择等。

8.1 基本概念

- 排序（sorting）：
  - 假设给定一组n个记录序列{$r_1, r_2, \ldots\ldots, r_n$}，其相应的关键字序列为{$k_1, k_2, \ldots\ldots, k_n$}。排序是将这些记录排列成顺序为{$r_{s1}, r_{s2}, \ldots\ldots, r_{sn}$}的一个序列，使得相应的关键字的值满足$k_{s1} \le k_{s2} \le \ldots\ldots \le k_{sn}$（称为升序）或$k_{s1} \ge k_{s2} \ge \ldots\ldots \ge k_{sn}$（称为降序）。
- 排序的目的：方便查询和处理。
- 排序算法的稳定性：
  - 假定在待排序的记录集中，存在多个具有相同关键字值的记录，若经过排序，这些记录的相对次序仍然保持不变，即在原序列中，$k_i=k_j$且r_i在r_j之前，而在排序后的序列中，r_i仍在r_j之前，则称这种排序算法是稳定的；否则称为不稳定的。

8.1 基本概念（cont.）

- 排序算法及其存储结构：
  - 从操作角度看，排序是线性结构的一种操作。假定待排序记录采用顺序存储结构：

```
struct   records {
        keytype   key ;
        fields   other ;
};
typedef   records  LIST[maxsize] ;
```

  - **Sort（LIST &A, int n ）：**
    - 对n个记录的数组按照关键字不减的顺序进行排序
    - n是正整数
    - 只讨论基于比较的排序（> = < 有定义）
    - 只讨论内部排序 （排序过程中数据对象全部存放在内存的排序）
    - 稳定性：任意两个相等的数据， 排序前后的相对位置不发生改变

8.1 基本概念（cont.）

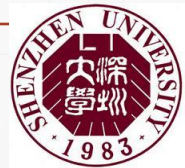
- 排序算法的性能：
  - 基本操作：
    - 比较：关键字之间的比较；
    - 移动：记录从一个位置移动到另一个位置。
  - 辅助存储空间。
    - 辅助存储空间是指在数据规模一定的条件下，除了存放待排序记录占用的存储空间之外，执行算法所需要的其他额外存储空间。
  - 算法复杂度。
    - 排序的时间复杂度可用算法执行中的记录关键字比较次数与记录移动次数来衡量

起泡排序

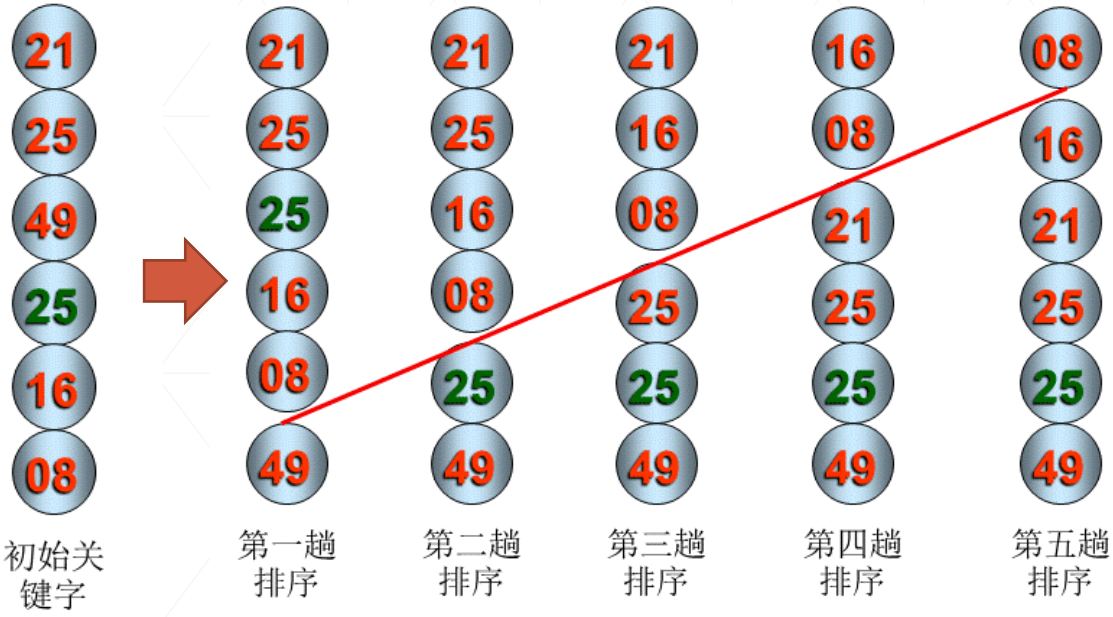
- 将待排序的记录看作是竖着排列的“气泡”，关键字较小的记录比较轻，从而要往上浮。
- 对这个“气泡”序列进行*n*-1趟处理。所谓一趟处理，就是自上而下检查一遍这个序列，依次比较相邻两个记录的关键字，如果发生逆序，即“轻”的记录在下面，就交换它们的位置。

```
void  BubbleSort ( int  n , LIST  &A )
{   for ( int i =n; i >=1; i-- ){
       flag=0;
       for ( int j =1; j <i; j++ ) { /*一趟起泡*/
          if ( A[j].key > A[j+1].key ){
              swap (A[j], A[j+1])
              flag=1;  /*标识发生了变化*/
           }
       }
       if ( flag==0)    break;   /*全称无交换*/
   }
}
```

稳定



起泡排序(cont.)



第一趟排序之后，关键字最大的记录就被交换到最后一个位置；
第二趟排序之后，关键字次大的记录就被交换到最后第二个位置；
…
关键字小的记录不断上浮（起泡），关键字大的记录不断下沉。
在作第二趟排序时，由于最后一个位置上的记录已是最大的，所以不必检查；
因此第i趟起泡排序一般是从1到n-i+1。

起泡排序(cont.)

- 算法（时间）性能分析：
  - 最好情况（正序）：
    - 比较次数：$n-1$
    - 移动次数：0
    - 时间复杂度：$O(n)$

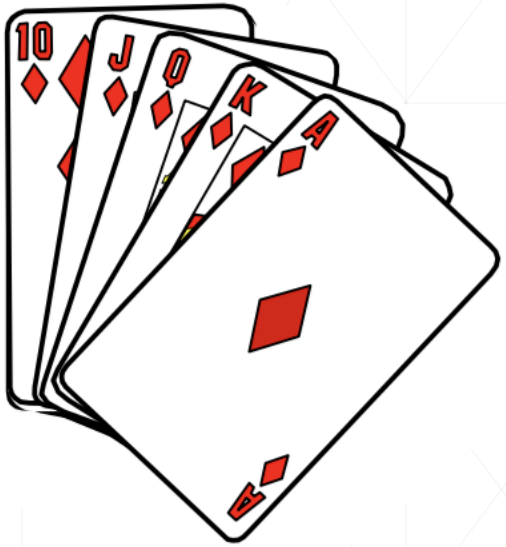
☺ 只执行一次起泡，做n-1次关键字比较，不移动记录

  - 最坏情况（反序）：
    - 比较次数：$\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}$
    - 移动次数：$3\sum_{i=1}^{n-1}(n-i)=\frac{3n(n-1)}{2}$
    - 时间复杂度：$O(n^2)$

☹ 执行$n-1$次起泡，第i趟做$n-i$次关键字比较，执行$n-i$次记录交换

```
void swap(records &x, records &y)
{   records w ;
    w=x;      x=y;      y=w;
}
```

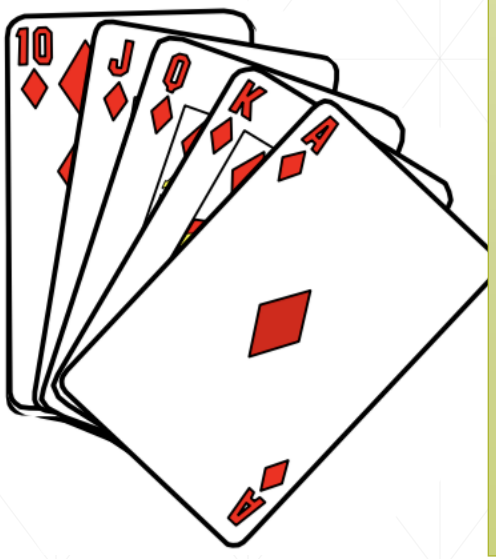
直接插入排序



❖ 插入排序的主要操作是插入，其基本思想是：每次将一个待排序的记录按其关键字的大小插入到一个已经排好序的有序序列中，直到全部记录排好序为止

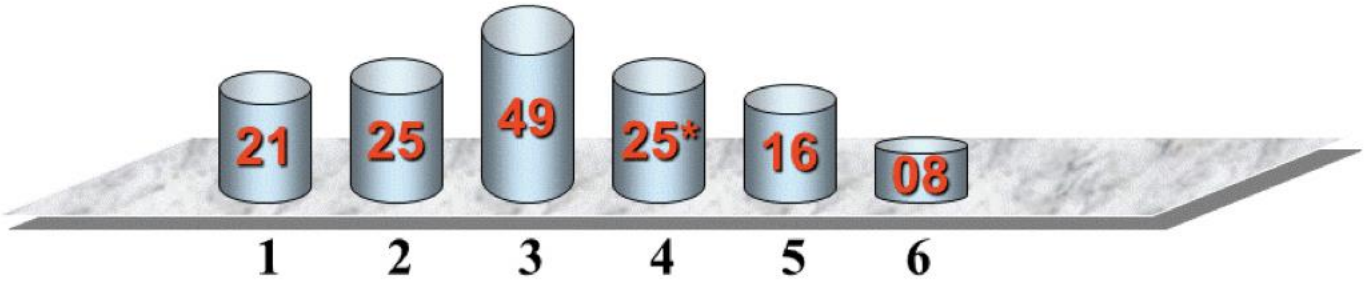
- 经过*i-1*遍处理后，A[1…*i*-1]已排好序。第*i*遍处理仅将A[*i*]插入A[1…*i*-1, *i*]的适当位置，使得A[1…*i*]又是排好序的序列
- 首先比较A[*i*]和A[*i*-1]的关键字，如果A[*i*-1].key≤A[*i*].key，由于A[1…*i*]已排好序，第*i*遍处理就结束了；
- 否则交换A[*i*]与A[*i*-1]的位置，继续比较A[*i-1*]和A[*i*-2]的关键字，直到找到某一个位置*j(1≤j≤i-1)*，使得A[*j*].*key*≤ A[*j*+1].*key*时为止。

直接插入排序（cont.）

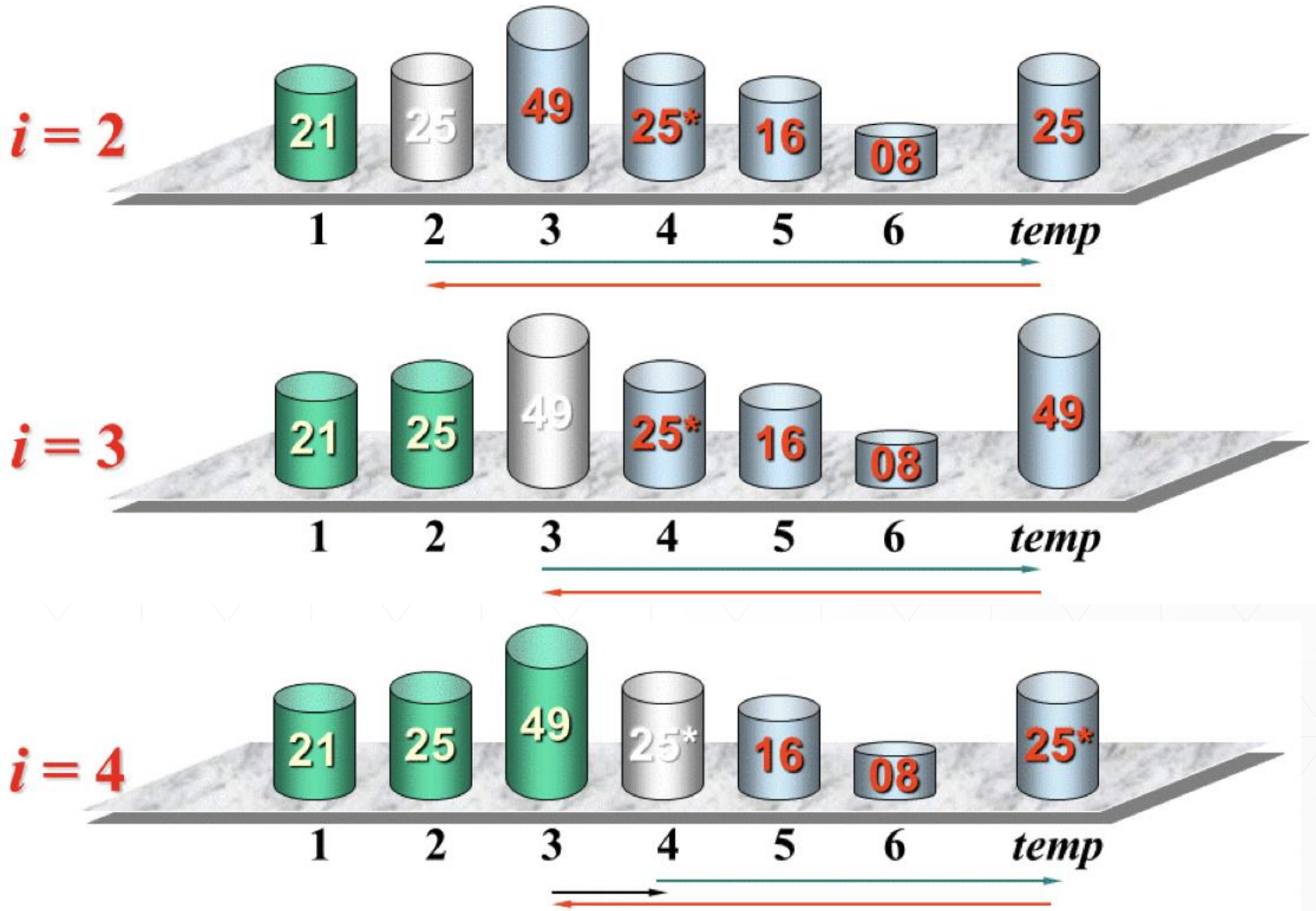



```
void  Insertion_Sort ( int  n , LIST  &A )
{   for ( int i =2; i <=N; i++ ){
       tmp=A[i];  /*摸下一张牌*/
       for ( int j =i; j>1&&A[j-1].key>tmp.key; j-- )
          A[j]=A[j-1]; /*移出空位*/
       A[j]=tmp; /*新牌落位*/
    }
}
```

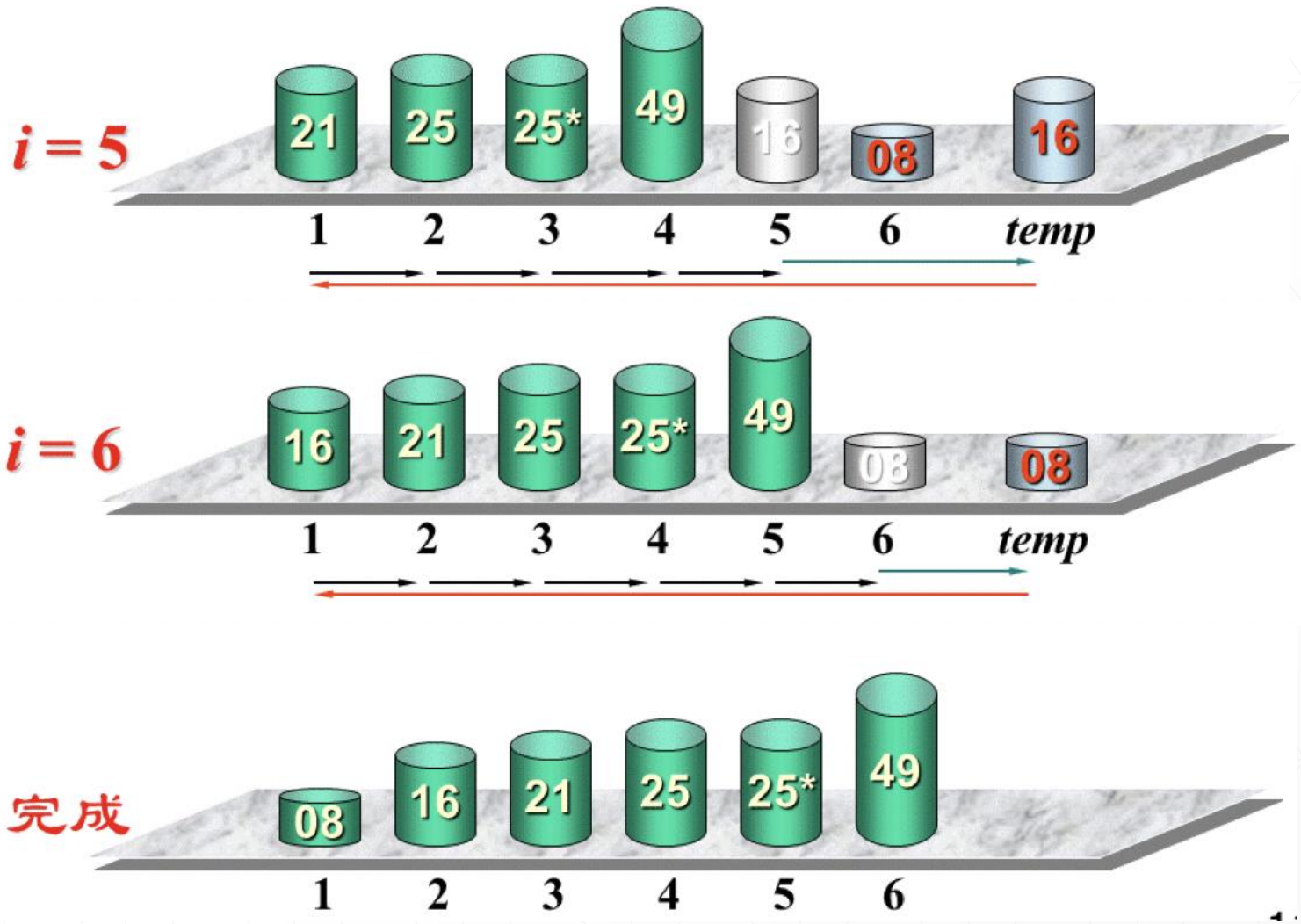
- 已知待序的一组记录的初始排列为21，25，49，25*，16，08



直接插入排序（cont.）


i = 2
21
25
49
25*
16
08
25
1
2
3
4
5
6
temp
i = 3
21
25
49
25*
16
08
49
1
2
3
4
5
6
temp
i = 4
21
25
49
25*
16
08
25*
1
2
3
4
5
6
temp

直接插入排序（cont.）


i = 5
21
25
25*
49
16
08
16
1
2
3
4
5
6
temp
i = 6
16
21
25
25*
49
08
08
1
2
3
4
5
6
temp
完成
08
16
21
25
25*
49
1
2
3
4
5
6

直接插入排序（cont.）

- 算法的性能分析
  - 最好情况下（正序）：
    - 比较次数：n-1
    - 时间复杂度为$O(n)$。

☺ 排序前记录已按关键字从小到大有序，每趟只需与前面有序记录序列的最后一个记录比较一次，总的关键字比较次数为$n-1$

第i趟时需第i个记录需要与前面$i-1$个记录都做关键字比较，并且每一次比较都要做一次数据移动

  - 最坏情况下（反序）：
    - 比较次数：$\sum_{i=2}^{n} i-1 = \frac{n(n-1)}{2}$
    - 移动次数：$\sum_{i=2}^{n} (i+1) = \frac{(n+4)(n-1)}{2}$
    - 时间复杂度为$O(n^2)$。

直接插入排序（cont.）

- 算法的性能分析
  - 在平均情况下关键字的比较次数和记录移动次数约为$n^2/4$
  - 直接插入排序是一种稳定的排序方法
  - 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。
- 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。