数据结构

深圳技术大学
大数据与互联网学院

第十章 内部排序

10.1 概述

一. 排序

- 排序（Sorting）：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

- 内部排序：在排序期间数据对象全部存放在内存的排序
- 外部排序：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

10.1 概述

一. 排序

- 排序基本操作
  - 比较：比较两个关键字的大小
  - 移动：将记录从一个位置移动至另一个位置

- 排序的时间复杂度
  - 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

10.1 概述

一. 排序

- 排序方法的稳定性
  - 如果在记录序列中有两个记录r[i]和r[j]，它们的关键字key[i] == key[j]，且在排序之前，记录r[i]排在r[j]前面。
  - 如果在排序之后，记录r[i]仍在记录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

- 排序常用的数据结构是数组、顺序表、结构体

10.2 插入排序

一. 直接插入排序

- 直接插入排序是最简单的排序方法，操作：
  - 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有序表的适当位置上，直到对象全部插入为止。

10.2 插入排序

一. 直接插入排序

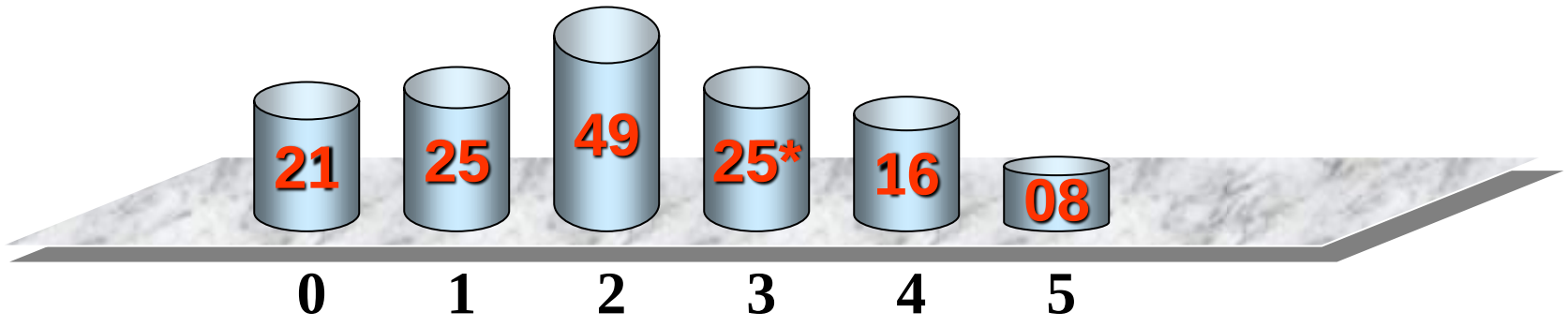
- 算法流程
  - 当插入第i(i≥1)个对象时，前面的r[0]，r[1]，...，r[i-1]已经排好序。
  - 用r[i]的关键字与r[i-1]，r[i-2]，...的关键字顺序进行比较(和顺序查找类似)，如果小于，则将r[x]向后移动(插入位置后的记录向后顺移)
  - 找到插入位置即将r[i]插入

10.2 插入排序

一. 直接插入排序

- 举例

  已知待序的一组记录的初始排列为：21，25，49，25*，16，08

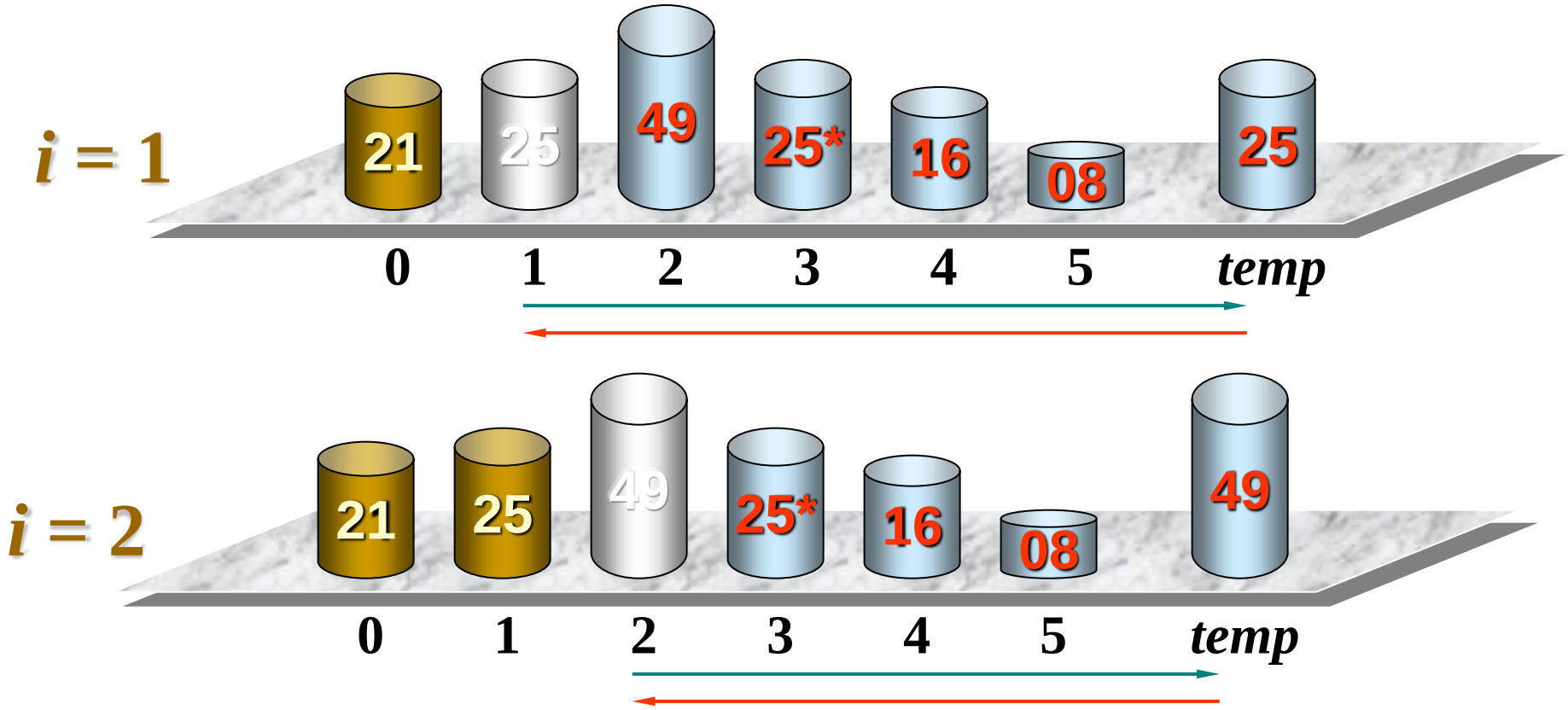


$i = 0$

10.2 插入排序

一. 直接插入排序

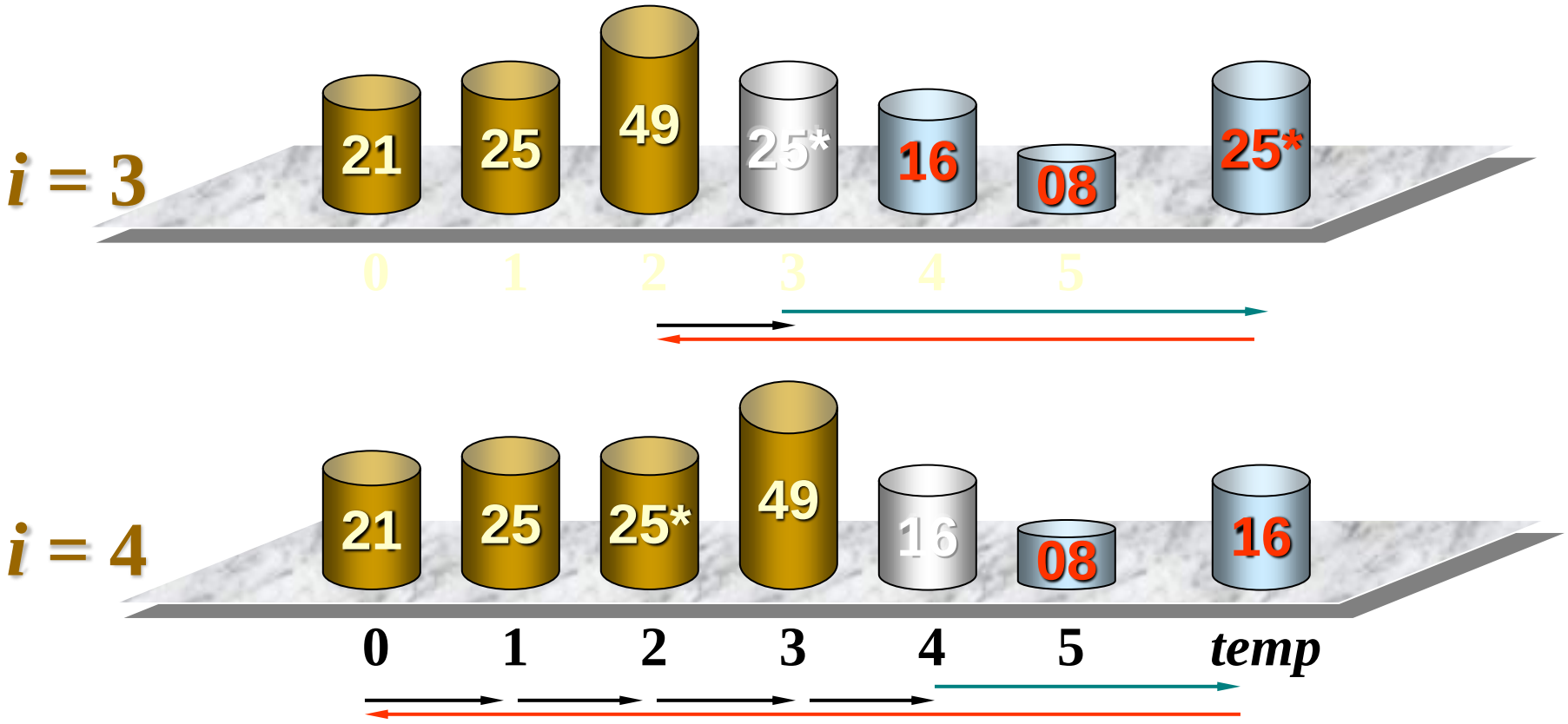
- 插入过程



10.2 插入排序

一. 直接插入排序

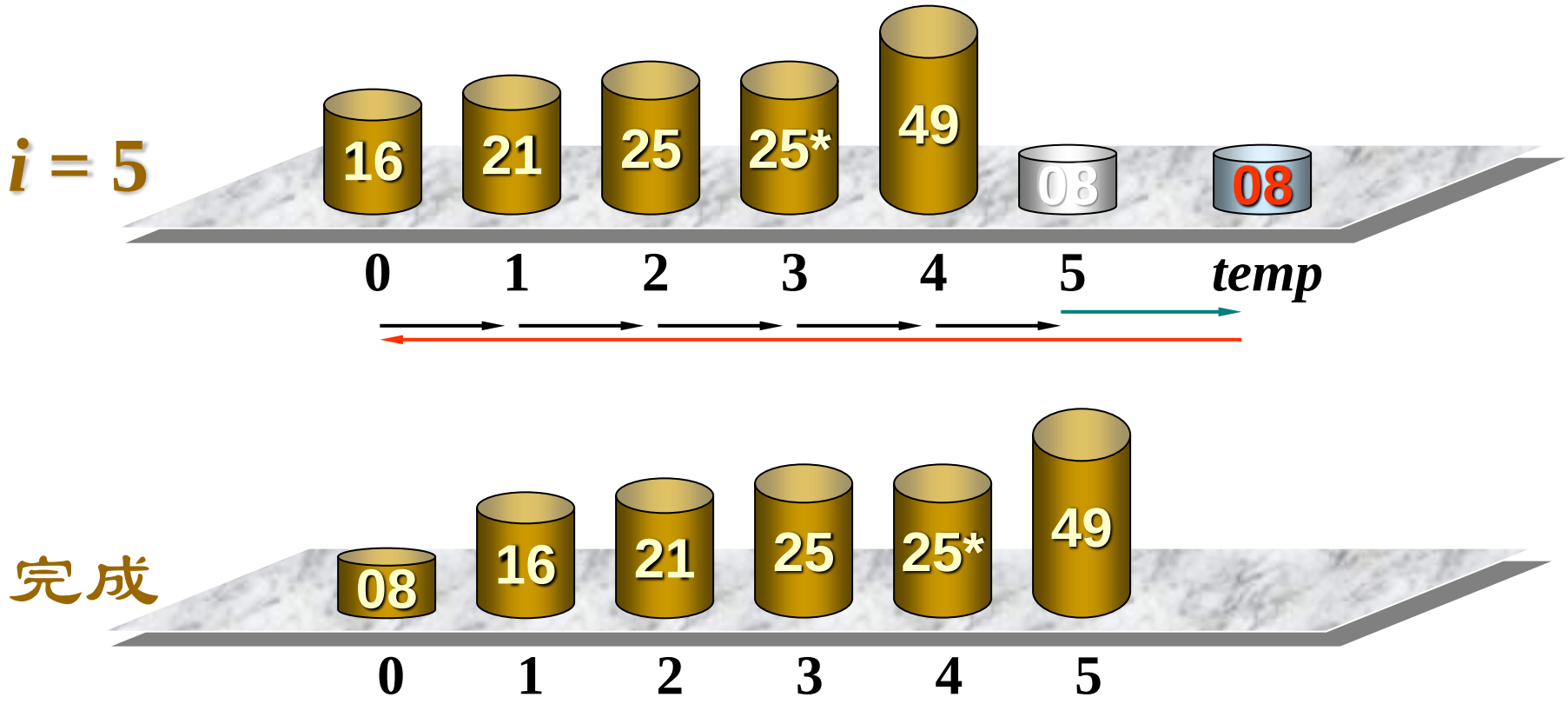
- 插入过程



10.2 插入排序

一. 直接插入排序

- 插入过程



10.2 插入排序

一. 直接插入排序

- 算法实现

```
void InsertSort (int r[ ], int n ) {
    // 假设关键字为整型，放在向量r[]中
    int i, j, temp;
    for ( i = 1; i < n; i++ ) {
        temp = r[i];
        for ( j = i;  j > 0; j-- ) {
        //从后向前顺序比较，并依次后移
          if ( temp < r[j-1] ) r[j] = r[j-1];
          else break;
        }
        r[j] = temp;
    }
}
```

10.2 插入排序

一. 直接插入排序

- 算法分析：
  - 关键字比较次数和记录移动次数与记录关键字的初始排列有关
  - 最好情况下，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1次，移动0次记录，总的关键字比较次数为 n-1，记录移动次数为0
  - 最坏情况下，排序前记录按逆序排列，第i趟时第i个记录必须与前面i个记录都做关键字比较，并且每做1次比较就要做1次数据移动。则总关键字比较次数KCN和记录移动次数RMN分别为

$$KCN = \sum_{i=1}^{n-1} i = n(n-1)/2 \approx n^2/2,$$

$$RMN = \sum_{i=1}^{n-1} (i+2) = (n+4)(n-1)/2 \approx n^2/2$$

10.2 插入排序

一. 直接插入排序

- 直接插入排序的时间复杂度为$O(n^2)$
- 直接插入排序是一种稳定的排序方法
- 直接插入排序最大的优点是简单，在记录数较少时，是比较好的办法

10.2 插入排序

二. 折半插入排序

- 折半插入排序是指在查找记录插入位置时，采用折半查找算法
- 折半查找比顺序查找快，所以折半插入排序在查找上性能比直接插入排序好，但需要移动的记录数目与直接插入排序相同(为$O(n^2)$)
- 折半插入排序的时间复杂度为$O(n^2)$
- 折半插入排序是一种稳定的排序方法

10.2 插入排序

二. 折半插入排序

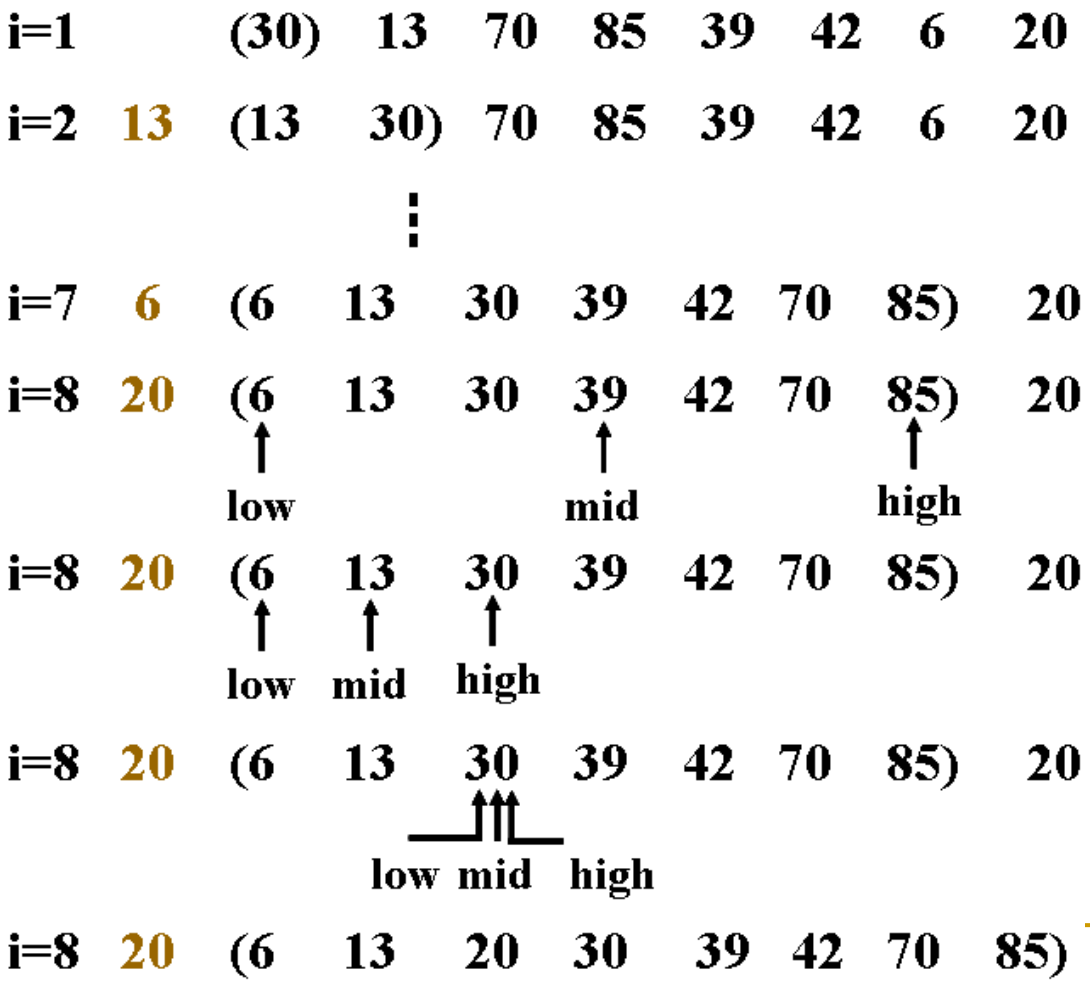
- 折半插入排序算法实现

```
void BInsertSort(SqList &L) { // 对顺序表L作折半插入排序。
    int i,j,high,low,m;
    for (i=2; i<=L.length; ++i) {
        L.r[0] = L.r[i];          // 将L.r[i]暂存到L.r[0]
        low = 1;   high = i-1;
        while (low<=high) {
        // 在r[low..high]中折半查找有序插入的位置
            m = (low+high)/2;              // 折半
            if (LT(L.r[0].key, L.r[m].key)) high = m-1;
            // 插入点在低半区
            else  low = m+1;               // 插入点在高半区
        }//end while
        for (j=i-1; j>=high+1; --j) L.r[j+1] = L.r[j];
// 记录后移
        L.r[high+1] = L.r[0];     // 插入
    }//end for
} // BInsertSort
```

10.2 插入排序

二. 折半插入排序

- 示例，设有一组关键字30, 13, 70, 85, 39, 42, 6, 20



10.2 插入排序

三. 希尔排序

- 从直接插入排序可以看出，当待排序列为正序时，时间复杂度为0(n)
- 若待排序列基本有序时，插入排序效率会提高
- 希尔排序方法是先将待排序列分成若干子序列分别进行插入排序，待整个序列基本有序时，再对全体记录进行一次直接插入排序
- 希尔排序又称为缩小增量排序

10.2 插入排序

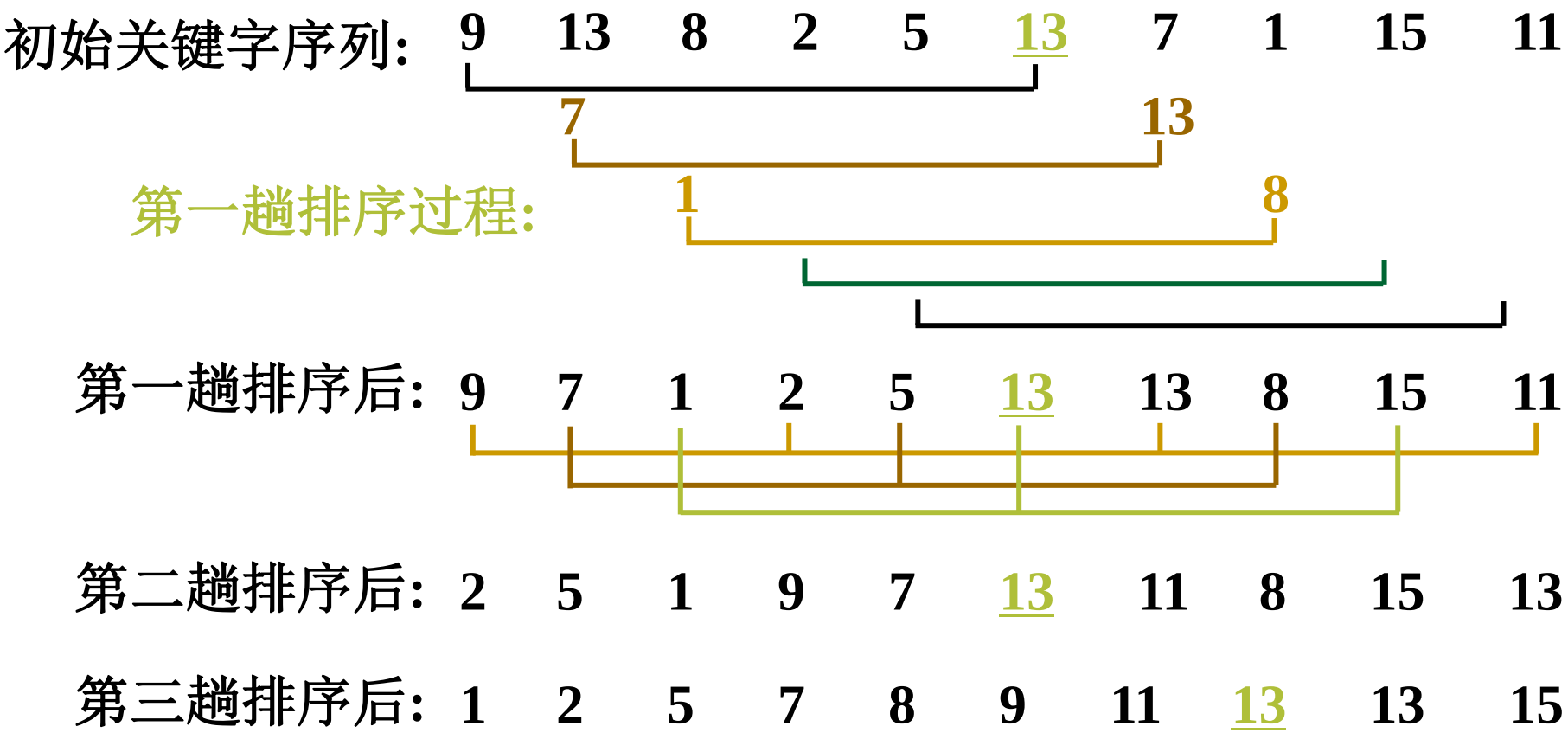
三. 希尔排序

- 算法过程：
  - 首先取一个整数 gap < n(待排序记录数) 作为间隔，将全部记录分为 gap 个子序列，所有距离为 gap 的记录放在同一个子序列中
  - 在每一个子序列中分别施行直接插入排序。
  - 然后缩小间隔 gap，例如取 gap = gap/2
  - 重复上述的子序列划分和排序工作，直到最后取gap = 1，将所有记录放在同一个序列中排序为止。

10.2 插入排序

三. 希尔排序

- 举例



希尔排序过程

10.2 插入排序

三. 希尔排序

- 算法实现

```
void ShellInsert(SqList &L, int dk) {//作一趟希尔插入排序
  // 1. 前后记录位置的增量是dk，而不是1;
  // 2. r[0]只是暂存单元，当j<=0时，插入位置已找到。
  int i,j;
  for (i=dk+1; i<=L.length; ++i)
    if (LT(L.r[i].key, L.r[i-dk].key)) { // 需要做交换
      L.r[0] = L.r[i];                   // 暂存在L.r[0]
      for (j=i-dk; j>0 && LT(L.r[0].key, L.r[j].key); j-=dk)
        L.r[j+dk] = L.r[j];              // 记录后移，查找插入位置
      L.r[j+dk] = L.r[0];                // 插入
    }
} // ShellInsert

void ShellSort(SqList &L, int dlta[], int t) {  // 算法10.5
  // 按增量序列dlta[0..t-1]对顺序表L作希尔排序。
  for (int k=0; k<t; ++k)
    ShellInsert(L, dlta[k]);  // 一趟增量为dlta[k]的插入排序
} // ShellSort
```

10.2 插入排序

三. 希尔排序

- 算法分析：
  - 开始时 gap 的值较大，子序列中的记录较少，排序速度较快
  - 随着排序进展，gap 值逐渐变小，子序列中记录个数逐渐变多，由于前面大多数记录已基本有序，所以排序速度仍然很快。
  - Gap的取法有多种。 shell 提出取 gap = ⌊n/2⌋，gap = ⌊gap/2⌋，直到gap = 1

10.2 插入排序

三. 希尔排序

- 算法分析：
  - 对特定的待排序记录序列，可以准确地估算关键字的比较次数和记录移动次数。
  - 希尔排序所需的比较次数和移动次数约为$n^{1.3}$
  - 当n趋于无穷时可减少到$n \times (\log_2 n)^2$
  - 希尔排序的时间复杂度约为$O(n \times (\log_2 n)^2)$
  - 希尔排序是一种不稳定的排序方法

10.3 快速排序

一. 起泡排序

- 算法设计
  - 设待排序记录序列中的记录个数为n
  - 一般地，第i趟起泡排序从1到n−i+1
  - 依次比较相邻两个记录的关键字，如果发生逆序，则交换之
  - 其结果是这n−i+1个记录中，关键字最大的记录被交换到第n−i+1的位置上，最多作n−1趟。

10.3 快速排序

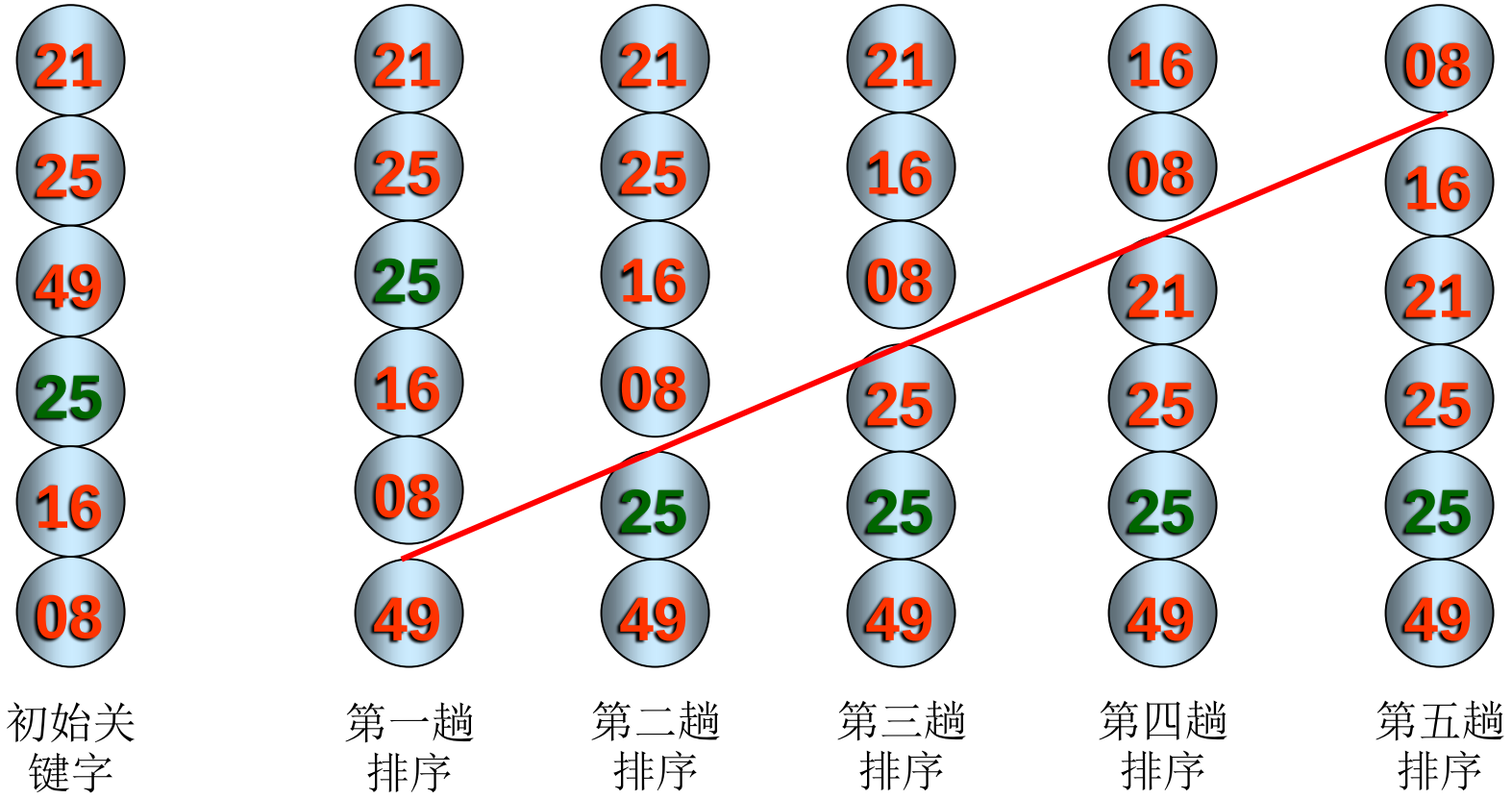
一. 起泡排序

- 算法过程
  - i=1，为第一趟排序，关键字最大的记录将被交换到最后一个位置
  - i=2，为第二趟排序，关键字次大的记录将被交换到最后第二个位置
  - 依此类推……
  - 关键字小的记录不断上浮(起泡)，关键字大的记录不断下沉(每趟排序最大的一直沉到底)

10.3 快速排序

一. 起泡排序

- 举例



10.3 快速排序

一. 冒泡排序

- 算法实现

```
void Bubble_Sort(Sqlist *L){
    int j ,k , flag ;
    for (j=0; j<L->length; j++) {   //共有n-1趟排序
          flag=TRUE ;
          for (k=1; k<=L->length-j; k++)    //一趟排序
              if (LT(L->R[k+1].key, L->R[k].key ) ) {
                 flag=FALSE ;
                 L->R[0]=L->R[k] ;
                 L->R[k]=L->R[k+1] ;
                 L->R[k+1]=L->R[0] ;
              }
          if   (flag==TRUE)  break ;
      }
}
```

10.3 快速排序

一. 起泡排序

- 性能分析
  - 最好情况：在记录的初始排列已经按关键字从小到大排好序时，此算法只执行一趟起泡，做n-1次关键字比较，不移动记录
  - 最坏情况：执行n-1趟起泡，第i趟做n-i次关键字比较，执行n-i次记录交换，比较次数KCN和交换次数RCN共计：

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

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

- 起泡排序的时间复杂度为$O(n^2)$
- 起泡排序是一种稳定的排序方法