数据结构

第十九讲 插入排序

孙猛

http://www.math.pku.edu.cn/teachers/sunm

2017年12月14日

手机\通讯 : 手机 : Apple ›

排序： 上架时间

人气
上架时间
价格：由低到高
价格：由高到低
用户评分

客户评分: 4星及以上 | 3星及以上 | 2星及以上 | 1星及以上

查看（尺寸大小）选项

Apple iPhone 6s (64G) 4G智能手机(玫瑰金色 公开版)
Apple
￥5,768.00 货到付款
部分地区今天或明天即可送达
更多购买选择
￥5,768.00 新品 (2 卖家)
满99减10限建行IC借记卡（62开头）及 另外 2 个促销
加入购物车 加入心愿单
139

查看（尺寸大小）选项

Apple iPhone 6s (16G) 4G智能手机(玫瑰金色 公开版)
Apple
￥4,968.00 货到付款
部分地区今天或明天即可送达
更多购买选择
￥4,968.00 新品 (2 卖家)
满99减10限建行IC借记卡（62开头）及 另外 2 个促销
加入购物车 加入心愿单
139

查看（尺寸大小）选项

Apple iPhone 6s Plus (64G) 4G智能手机(玫瑰金色 公开版)
Apple
￥6,498.00 货到付款
部分地区今天或明天即可送达
更多购买选择
￥6,498.00 新品 (2 卖家)
满99减10限建行IC借记卡（62开头）及 另外 2 个促销
加入购物车 加入心愿单
72

课程内容

- 排序的基本概念

- 插入排序

排序的基本概念

- 假设给定一个有待排序的文件，它由N个记录的集合构成:$\{R_1,R_2,\ldots,R_N\}$。
- 每个记录R_i有一个排序码(不一定是关键码)，记为 K_i。
- 在排序码上确定一个全序关系<，使得对任意的三个排序码值a, b, c，下列条件成立:
  (1) $a<b, a=b, b<a$三个可能性中恰有一个为真;
  (2)如果 $a<b$ 且 $b<c$, 则 $a<c$。
- 排序的目标是确定下标为$\{1,2,\ldots,N\}$的一个排列 $p(1), \ldots, p(N)$，使得$K_{p(1)}\leq K_{p(2)}\leq \ldots \leq K_{p(N)}$。

排序可以看成字典上的操作

- 在排序的讨论中，并不关心被排序对象本身的逻辑结构，可以把排序看成是字典上的一种操作。

- 排序码可以就是关键码，也可以不是关键码。在后一种情况下，不同记录的排序码可以相同。

- 与关于索引的讨论类似，在具体排序算法中，只关心排序码的作用和表示，而把记录中的其它字段隐藏起来。

排序方法分类

- 作为排序对象的文件可以是外存的数据，也可以是内存的数据。待排序的记录在排序过程中全部存放在内存的成为**内排序**；否则称为**外排序**。
- 我们主要关注内排序算法。
- 从算法的基本思想出发大致可以将(内)排序算法分为5类:
  - 插入排序；
  - 选择排序；
  - 交换排序；
  - 分配排序；
  - 归并排序(也可用于外排序)。
- （每一类方法可能具体有多个不同算法）

评价排序算法代价的标准

- 排序算法的主要操作: 比较+记录移动(调整)
- 评价排序算法好坏的主要标准:
  (1)执行算法所需的时间(最坏或者平均时间);
  (2)执行算法所需要的附加空间;
  (3) 算法本身的复杂程度。
- 其它标准:
  - 稳定性:
    - 若某种排序方法保持相同排序码记录的原有的相对次序，则称这种排序方法是稳定的，否则是不稳定的。
  - 适应性:
    - 如果实际的待排序序列比较接近排好序的形式，算法能否更快完成工作。

排序的应用举例

- 有助于查找（例如有序顺序表的二分法查找，在等概率的情况下构造最佳二叉排序树）；
- 解决“一起性”的问题， 即把一个结构中具有相同标志（排序码）的所有项连在一起；
- 处理在两个或者多个结构中的具有相同标志的项（例如有序顺序表表示的两个集合的交运算）。

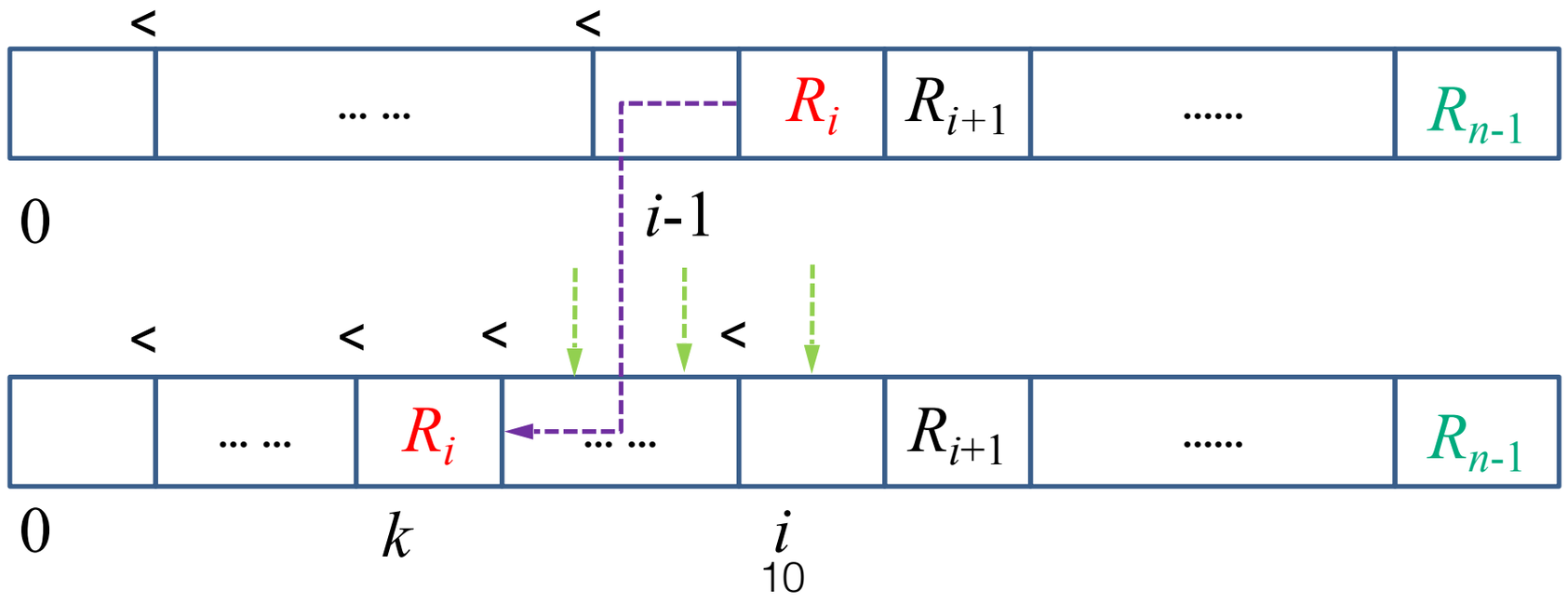
插入排序

- 插入排序的基本思想:
  - 每步将一个待排序的记录，按其排序码大小，插入到前面已经排序的文件中的适当位置，直到全部记录都被插入完为止。

- 直接插入排序
- 表插入排序
- 二分法插入排序
- Shell排序
- 多表插入排序

直接插入排序

- 基本思想：假定待排序的n个记录$\{R_0,R_1,\ldots,R_{n-1}\}$存在数组中，直接插入法在插入记录$R_i(i=1,2\ldots n-1)$时，记录序列分为两个区间$[R_0,R_{i-1}]$和$[R_i,R_{n-1}]$，前一区间里的记录已排好序，后一区间尚未排序。对 R_i 的处理就是排序码K_i依次与K_{i-1}，K_{i-2}，…，K_0比较，找出适当位置将 R_i 插入，原位置的记录向后顺移。



存储结构

```
typedef  int KeyType;
typedef int DataType;
typedef struct{
        KeyType key;      /*排序码字段*/
        DataType info;    /*记录的其它字段*/
}RecordNode;
typedef  struct{
        int n;                    /*文件中的记录个数,可以视为常量*/
        RecordNode *record;
}SortObject;
```

直接插入排序（按非递减序）算法

```
void insertSort(SortObject * pvector) {
    int i, j;
    RecordNode temp;
    RecordNode *data = pvector->record; /*引入data 使下面代码更简洁*/
    for( i = 1; i < pvector->n; ++i) {          /* 依次插入R1, R2, …,Rn-1 */
        temp = data[i];
        for ( j = i-1; temp.key < data[j].key && j >= 0; j-- )
            /* 由后向前找插入位置,排序码大于ki的记录后移 */
            data[j+1] = data[j];
        if( j != i-1 ) data[j+1] = temp;                    1时不用复制 */
    }
}
```

稳定排序

直接插入排序示例

初始: [49] 38 65 97 76 13 27 49

i=1: [38 49] 65 97 76 13 27 49

i=2: [38 49 65] 97 76 13 27 49

i=3: [38 49 65 97] 76 13 27 49

i=4: [38 49 65 76 97] 13 27 49

i=5: [13 38 49 65 76 97] 27 49

i=6: [13 27 38 49 65 76 97] 49

i=7: [13 27 38 49 49 65 76 97]

算法分析

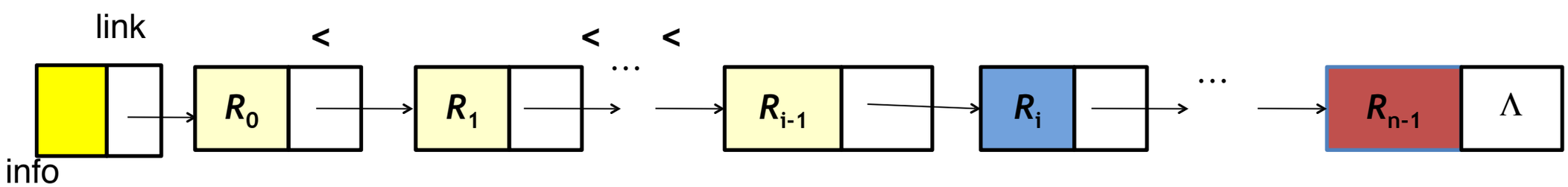
- 主要关注比较次数和移动次数：
  - 最小比较次数：$C_{min}=n-1\approx n$；
  - 最大比较次数：$C_{max}=\sum_{i=1}^{n-1} i=n(n-1)/2\approx n^2/2$；
  - 最小移动次数(包括附加记录空间x移动次数)：$M_{min}=n-1\approx n$；
  - 最大移动次数：$M_{max}=\sum_{i=1}^{n-1} (i+1) =(n+2)(n-1)/2\approx n^2/2$。
  - 插入记录R_{i-1}的平均比较次数：

    $$\sum_{j=0}^{i-1} P_jC_j = \sum_{j=0}^{i-1} (j+1)/i=(i+1)/2 \ ;$$

  - 直接插入排序的总的比较次数为：$\sum_{j=1}^{n-1} (j+1)/2=O(n^2)$。
  - 平均移动次数： $O(n^2)$。
- 直接插入排序算法的平均时间复杂度为$T(n) = O(n^2)$。
  - 具有适应性
- 空间复杂度为$S(n) = O(1)$ (附加的记录空间x)。

表插入排序

- 表插入排序的目标:
  - 在直接插入排序的基础上减少记录移动的次数。
- 基本思想:
  - 在每个记录中增加一个链接字段,指向下一个记录,整个被排序的文件表示成一个记录的单链表。
  - 插入记录R_i时，R_0至R_{i-1}已经排序，为了确定插入的位置，先将R_i脱链, 再采用顺序比较的方法找到R_i应插入的位置, 将R_i插入链表。



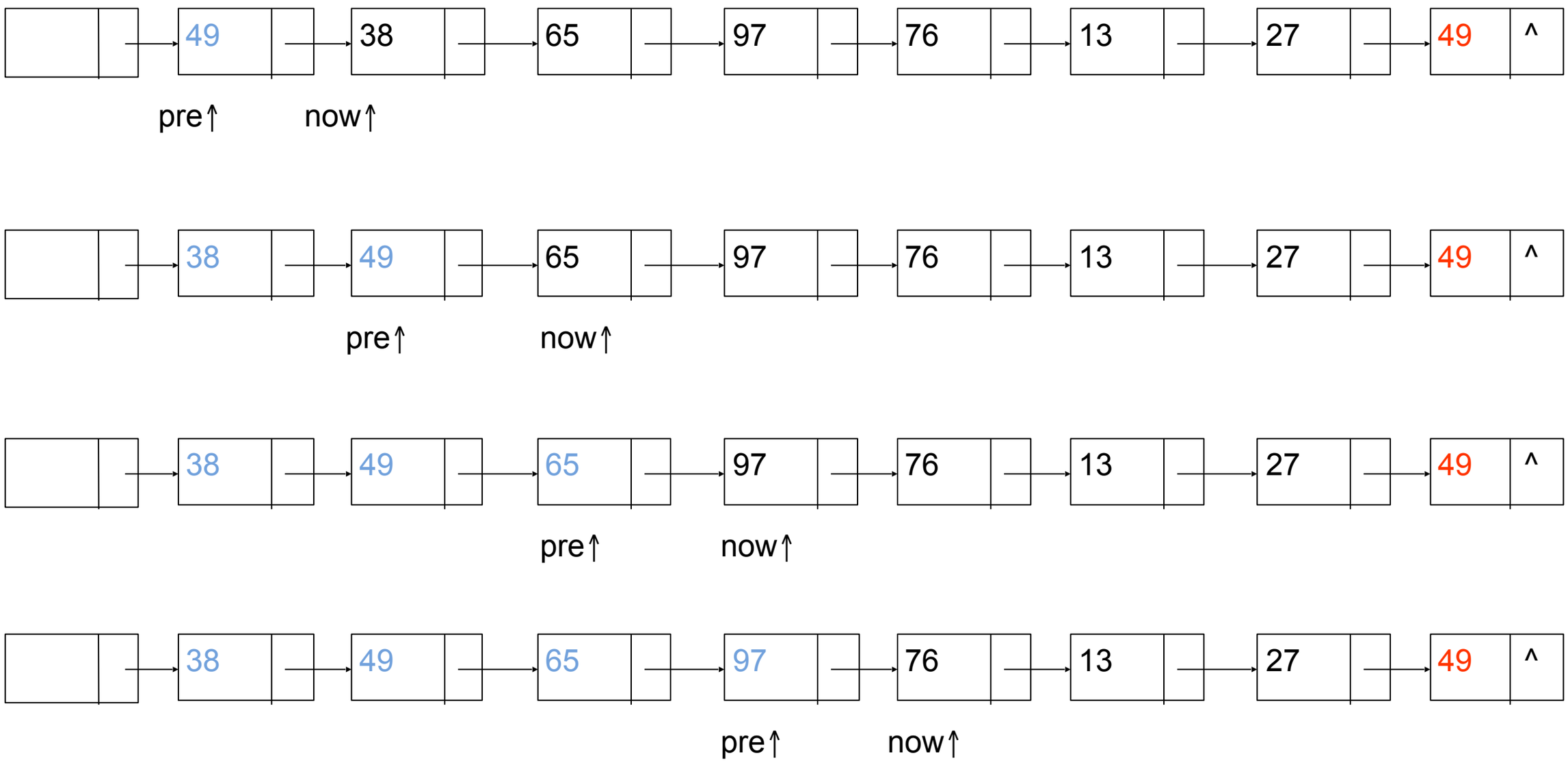
存储结构

```
struct Node;
typedef struct  Node  ListNode;
struct  Node  {
    KeyType   key;          /*记录的排序码字段*/
    DataType   info;        /*记录的其他字段*/
    ListList   *next;       /*记录的指针字段*/
};
typedef  ListNode *LinkList;
```

表插入排序算法

```
void listSort (LinkList * plist) {                          /* 表插入排序,有头结点 */
  ListNode *now, *pre, *p, *q, *head;
  head=*plist; pre=head->next; if(pre==NULL) return;
  now=pre->next; if(now==NULL) return;
  while( now!=NULL){
    q=head;    p=head->next;
    while(p!=now && p->key<=now->key)
     {q=p;p=p->next;}                                       /* 循环结束找到插入位置 */
    if(p==now) {pre=pre->next;now=pre->next; continue ;}  /* now应放在原位置 */
    pre->next=now->next;                                    /* 使now记录脱链 */
    now->next=p; q->next=now;                               /* 将now记录插入链表中 */
    now=pre->next;
  }
}
```

表插入排序示例

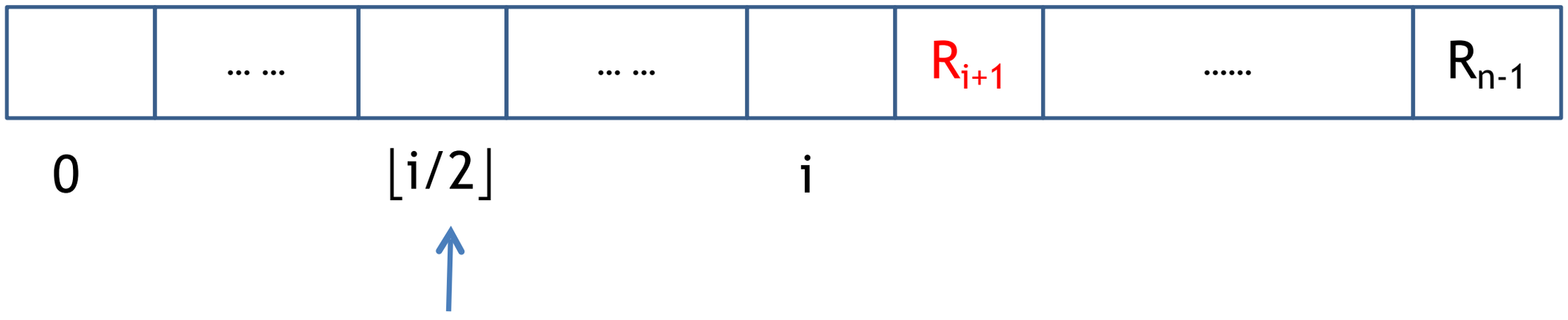

49
38
65
97
76
13
27
49
^
pre↑
now↑
38
49
65
97
76
13
27
49
^
pre↑
now↑
38
49
65
97
76
13
27
49
^
pre↑
now↑
38
49
65
97
76
13
27
49
^
pre↑
now↑

表插入排序算法分析

- **空间效率:**
  - **临时空间是常量的** $O(1)$;
  - **如果考虑链表用的额外空间(除保存数据之外的空间)，每个记录增加了一个链接字段，这部分空间** $S(n) = O(n)$。
- **时间效率:**
  - **用链表方式不需要记录移动，摘除结点和链入结点的次数为**$O(n)$，**但关键码比较次数仍是**$O(n^2)$，**因此算法的时间复杂度仍为**$O(n^2)$。
- **稳定性:**
  - **从头往后找插入位置的循环里，通过关键码之间的小于等于关系进行比较，可以保证表插入排序的稳定性。**

二分法插入排序

- 基本思想：在直接插入法的基础上，为了减少插入R_{i+1}记录的比较次数，改用二分法在前面已经排序的$[R_0,R_i]$找插入位置。
- 二分法插入排序只能用于顺序存储的数据。每步的工作：
  - 用二分法在已排序的前一段找出当前记录应插入的位置；
  - 取出当前记录,将该位置之后的已排序记录顺序后移,腾出空位后将当前记录插入。



二分法插入排序算法

```
void  binSort(SortObject * pvector)  {
    int i, j, left, mid, right; RecordNode temp;
    RecordNode *data = pvector->record;
    for( i = 1; i < pvector->n; i++ ) {
      temp = data[i];  left = 0;  right = i – 1;   /* 置已排序区间的下、上界初值 */
      while (left <= right) {
        mid=(left+right)/2;                        /* mid指向已排序区间的中间位置 */
        if  (temp.key<data[mid].key) right=mid-1;   /* 插入点在左子区间 */
        else left=mid+1;                            /* 插入点在右子区间 */
        }
     for(j=i-1; j>=left; j--) data[j+1] = data[j];          /* 将排序码大的记录后移 */
     if(left!=i) data[left] = temp;
    }
}
```

稳定排序

二分法插入排序示例

初始: [49] 38 65 97 76 13 27 49

i=6: [13 27 38 49 65 76 97] 49

(1): [13 27 38 49 65 76 97] 49

left=0 mid=3 right=6

(2): 13 27 38 49 [65 76 97] 49

left=4 mid=5 right=6

(3): 13 27 38 49 [65] 76 97 49

left=4 right=4
mid=4

49<65 right=3<left=4，查找结束

(4): 13 27 38 49 49 65 76 97

二分法插入排序算法分析

- 二分法插入排序的比较次数与待排序记录的初始状态无关，仅依赖于记录的个数。
- 插入第 i 个记录时，
  - 如果$i=2^j$，$(0 \le j \le \lfloor log_2 n \rfloor)$，则要经过$j=log_2 i$ 次比较才能确定插入位置；
  - 如果$2^j<i\le 2^{j+1}$，则比较次数为$j+1$。
- 将n个记录$(n=2^k)$排序的总比较次数为

$$\sum_{i=1}^{n} \lceil log_2 i \rceil \approx n*log_2 n。$$

- 当n比较大时，比直接插入排序的最大比较次数$O(n^2)$小，但比最小比较次数$O(n)$大。
- 移动次数为$O(n^2)$。
- 因此平均时间复杂度$T(n)=O(n^2)$。
- 空间复杂度为$S(n)=O(1)$ (附加的记录空间x)。

Shell排序

又称缩小增量的排序(Diminishing Increment Sort)。

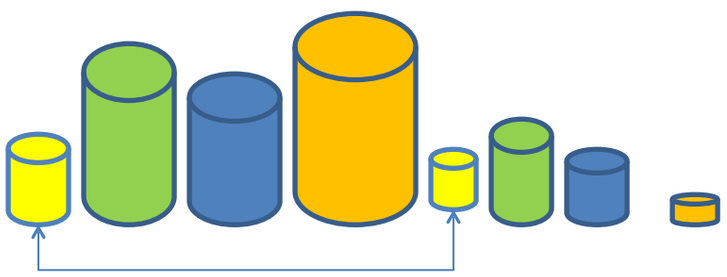
- Shell (1959) 认为:
  - 排序较慢的原因是元素移动太慢， 可能经过许多步才能到达最终位置(如插入排序中位于前面的大关键码记录);
  - 若能加大元素的移动步伐, 就可能加快排序。
- Shell提出的想法： 把一个序列看成由间距d的元素构成的d个“子序列”。反复做:

  (1)在每个子序列内部分别排序;

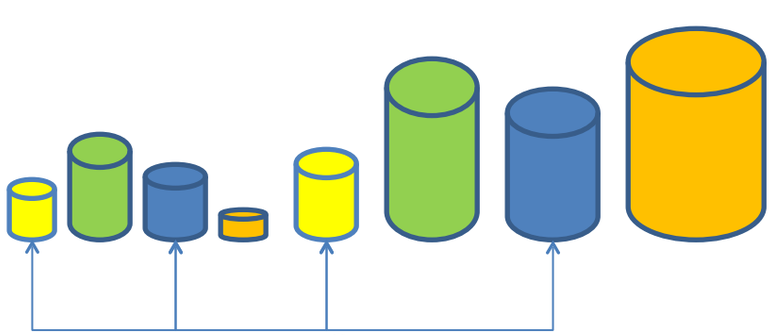
  (2)缩小d之后重复第1步,直到d缩小为1时再做一次排序(这也就是整个序列排序)。

Shell排序

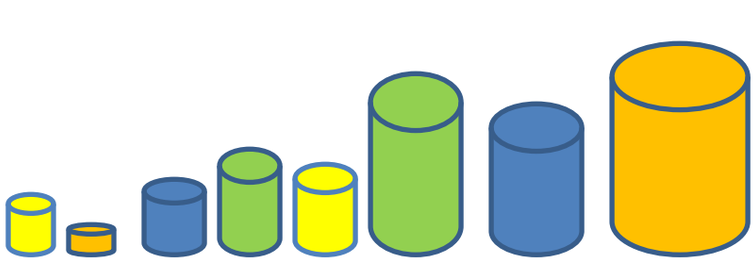
- **Shell　排序希望：前面各遍排序可以使距离目标位置很远的元素大步迁移到目标位置的附近，后续排序主要是局部调整，希望局部调整可以用较少时间完成。**



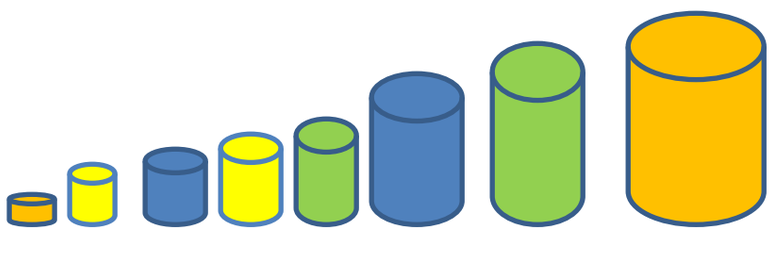
d_1=4



d_2=2



d_3=1



Shell排序

- **Shell排序通过许多次排序实现整个序列的排序。**
- **这一策略保证成功,因为只有最后一次排序就足够了。**

非形式算法：

inc = d　　#　d < n

while (inc > 0)：

对相距 inc 的各组记录做直接插入排序

inc = decrease (inc)

Shell排序

- **Shell排序可以看作一种排序模式，其中需要：**
  - **选择一种基本排序算法做分组的排序。这一排序算法必须有适应性，因前面排序已把大元素移向右，小元素移向左，后面遇到的是接近排序的序列。例如采用直接插入排序。**
  - **确定一种减小间距值 d 的模式。常用每次间距 d 减小一半，即 $d_1=\lfloor n/2 \rfloor$； $d_{i+1}=\lfloor d_i/2 \rfloor$ (Shell)**

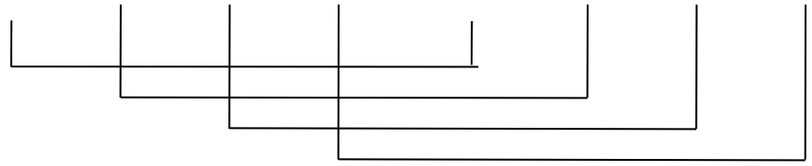
Shell排序算法

```
void  shellSort(SortObject * pvector, int d) {
    int i, j, inc;
    RecordNode temp, *data = pvector->record;
    for (inc = d; inc > 0; inc /= 2) {                    /* inc 为当时增量 */
        for (i = inc; i < pvector->n; i++) {              /*同时做inc个序列排序*/
            temp = data[i];                               /* 保存待插入记录Ri*/
                                                          /* 按间隔 inc 寻找插入点 */
            for (j = i - inc; j >= 0 && temp.key < data[j].key; j -= inc)
                data[j+inc] = data[j];                    /* 大记录后移 */
            data[j+inc] = temp;                           /* 插入记录Ri */
        }
    }
}
```

Shell排序示例

初始: 49 38 65 97 13 76 27 49

d_1=4



d_2=2 13 38 27 49 49 76 65 97

d_3=1 13 38 27 49 49 76 65 97

排序结果: 13 27 38 49 49 65 76 97

算法分析

- 采用上述常见模式(插入排序和间距减半)，Shell排序的平均比较次数和平均移动次数都为$O(n^{1.3})$左右；

  《The Art of Computer Programming》第3卷：排序与检索

- Shell排序算法只需常量的辅助空间；
- 一般说，Shell排序不能实现稳定排序。

d_i的不同取法

- **Shell最早提出**

$$d_1=\lfloor n/2 \rfloor,\ \ d_{i+1}=\lfloor d_i/2 \rfloor,$$

- **一般认为d_i都取成奇数、d_i之间互素为好，究竟如何选取d_i最好?**
- **进一步的相关问题可以参考 Knuth 的《The Art of Computer Programming》第3卷。**

多表插入排序


Refinement Calculus
A Systematic Introduction
Modeling Time in Computing
ALGEBRAIC TOPOLOGY
Formal Methods and Software Engineering
FME 2003: Formal Methods
Algebraic Methodology and Software Technology
Formal Methods for Components and Objects
Theoretical Aspects of Computing – ICTAC 2005
Formal Methods for Components and Objects
Programming Languages and Systems
Fundamental Approaches to Software Engineering
Unifying Theories of Programming
Component-Based Software Engineering
Fundamentals of Software Engineering
Formal Methods for Components and Objects
International Symposium on Fundamentals of Software Engineering
Formal Methods for Open Object-Based Distributed Systems
Logic Based Program Synthesis and Transformation
Formal Methods for Components and Objects
Unifying Theories of Programming
Formal Aspects of Component Software
Formal Aspects of Component Software
Formal Aspects of Component Software
Formal Aspects of Component Software
Formal Methods and Software Engineering
Dependable Software Engineering
Theory and Practice of Concurrency
Model Checking
Baier and Katoen
Software Engineering
Object-Oriented Software Engineering
UML 2 Semantics and Applica
MATHEMATICAL FRAMEWORKS FOR COMPONENT SOFTWARE
The LaTeX Companion
算法导论
计算机程序设计艺术 卷1：基本算法
计算机程序设计艺术 卷2：半数值算法
计算机程序设计艺术 卷3：排序与查找
Steve Jobs by Walter Isaacson
WOLFRAM A NEW KIND OF SCIEN
Python编程
Python编程
Handbook of Theoretical Computer Science
Handbook of Theoretical Computer Science
计算理论导引
自动机理论、语言和计算导论
面向对象软件工程
软件化软件

插入排序

- 直接插入排序
- 二分法插入排序
- 表插入排序
- 希尔排序
- *多表插入排序