


第10章 内部排序

综合↓销量↓评论数↓新品↓价格

共159万+件商品 1/100

配送至 上海杨浦区区长白新村街道

☐ 京东物流 ☐ 百亿补贴 ☐ 211限时达 ☐ 货到付款 ☐ 仅显示有货 ☐ 京东国际 ☐ 可配送全球 ☐ 新品 ☐ PLUS专享 ☐ 拍拍二手 ☐ 品类日



品类日


¥529.00 已补贴 ¥50

自营 荣耀畅玩20A 5000mAh超大电池长续航 128GB大存储 6.5英寸高清护眼屏

100万+条评价

荣耀京东自营旗舰店

百亿补贴




品类日

¥496.51 PLUS到手价 ¥499.00

自营 小米 (MI) Redmi 12C Helio G85 性能芯 5000万高清双摄 5000mAh长续航

50万+条评价

小米京东自营旗舰店



12.12 国标军工认证 享四年质保>


¥1590.51 PLUS到手价

自营 摩托罗拉 联想moto g75 四年质保 6.8英寸LCD护眼大屏 抗水耐摔 全功能

1000+条评价

摩托罗拉手机京东自营...

新品



品类日


¥765.16 已补贴 ¥130

自营 小米 (MI) Redmi Note13 5G 1亿像素 超细四窄边OLED直屏 5000mAh大电

10万+条评价

小米移动京东自营旗...

百亿补贴



品类日

¥1043.76 已补贴 ¥250


自营 OPPO K12x 8GB+256GB 钛空灰 120Hz 旗舰级OLED直屏 80W快充

10万+条评价

OPPO京东自营官方...

百亿补贴


广告



品类日

¥795.01 PLUS到手价 ¥799.00


自营 小米 (MI) Redmi Note12 5G 手机



品类日

¥5299.00


自营 一加 13 16GB+512GB 白露晨曦 高



品类日

¥2499.00


自营 小米 (MI) REDMI K80 第三代骁龙



品类日

¥1152.00 已补贴 ¥47

自营 华为畅享 60X 7000mAh长续航



品类日

¥1146.00 已补贴 ¥53

自营 荣耀X50 1.5K超清护眼硬核曲屏

第10章 内部排序

10.1 概述

10.2 插入排序

10.3 交换排序

10.4 选择排序

10.5 归并排序

10.6 基数排序

10.1 概述

1. 什么是排序？

定义：将一组杂乱无章的数据按一定的规律顺次排列起来。

存放在数据表中

按关键字排序

- 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。
- 主关键字：唯一
- 次关键字：不唯一

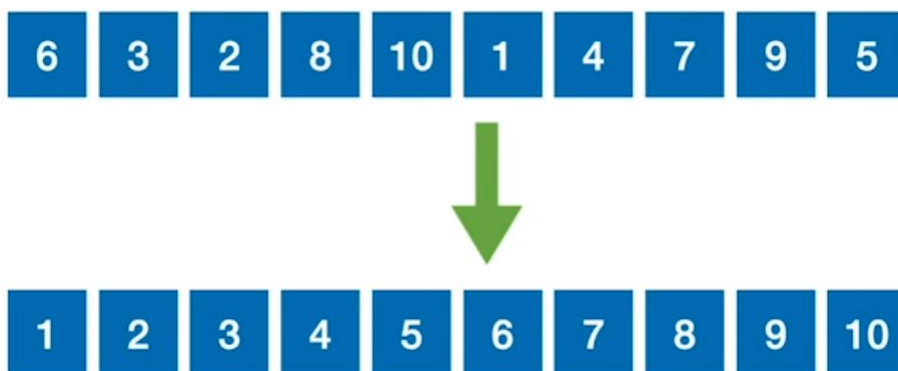
10.1 概述

1. 什么是排序?

排序 (Sort)

输入: n 个记录 R_1, R_2, \dots, R_n , 对应的关键字为 k_1, k_2, \dots, k_n 。

输出: 输入序列的一个重排 R'_1, R'_2, \dots, R'_n , 使得有 $k'_1 < k'_2 < \dots < k'_n$ (也可递减)



10.1 概述

2. 排序的目的是什么？

□ 目的：便于查找。

□ 应用场景

- 数据库查询
- 搜索引擎
- 数字信号处理
- 大数据处理
-

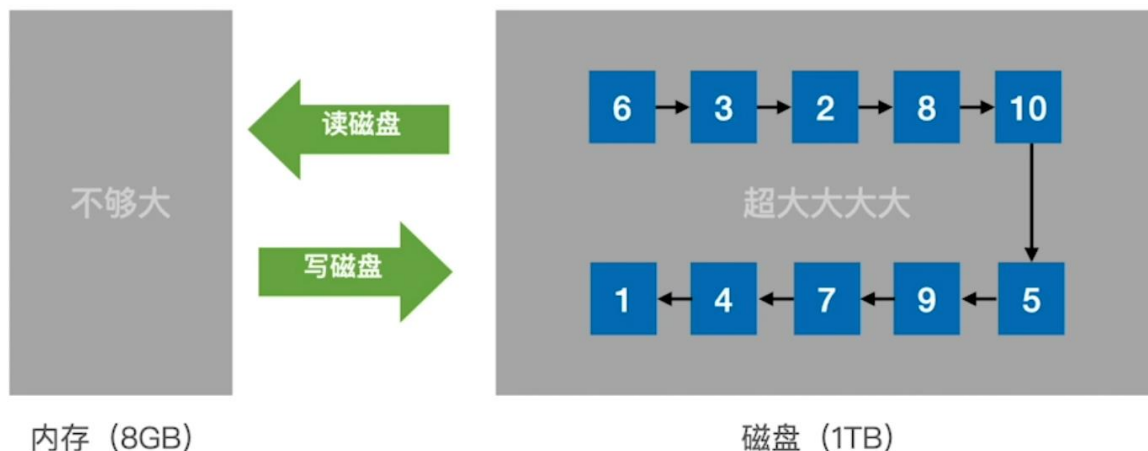
中国富豪排行榜				
1	钟睺眙	¥4,500亿	养生堂	饮料、医疗保健
2	马化腾	¥2,800亿	腾讯	互联网服务
3	黄峥	¥2,700亿	拼多多	购物网站
4	曾毓群	¥2,500亿	宁德时代	锂电池
5	张一鸣	¥2,450亿	字节跳动	社交媒体
6	丁磊	¥2,400亿	网易	互联网技术
7	李嘉诚、李泽钜	¥2,100亿	长江实业	投资
8	何享健	¥2,000亿	美的	房地产、家电制造
9	李书福	¥1,750亿	吉利	汽车制造
10	马云	¥1,700亿	阿里系	电子商务、金融科技
11	李兆基	¥1,600亿	恒基兆业	房地产
12	严昊	¥1,500亿	太平洋建设	基础建设
13	秦英林	¥1,450亿	牧原	畜牧
14	李蔡美灵	¥1,400亿	李锦记	调味品
15	吕向阳、张长虹	¥1,400亿	融捷	投资
16	王传福	¥1,350亿	比亚迪	汽车、充电电池
17	张刚	¥1,350亿	信发铝业	电解铝、氧化铝
18	刘永行、刘相宇	¥1,300亿	东方希望	氧化铝、重化工、饲料
19	郑家纯	¥1,300亿	周大福	珠宝首饰
20	陈建华、范红卫	¥1,250亿	恒力	化纤、石化、房地产

10.1 概述

3. 什么叫内部排序？ 什么叫外部排序？

——若待排序记录都在内存中，称为内部排序；

——若待排序记录一部分在内存，一部分在外存，则称为外部排序。



注：外部排序时，要将数据分批调入内存来排序，中间结果还要及时放入外存，显然外部排序要复杂得多。

10.1 概述

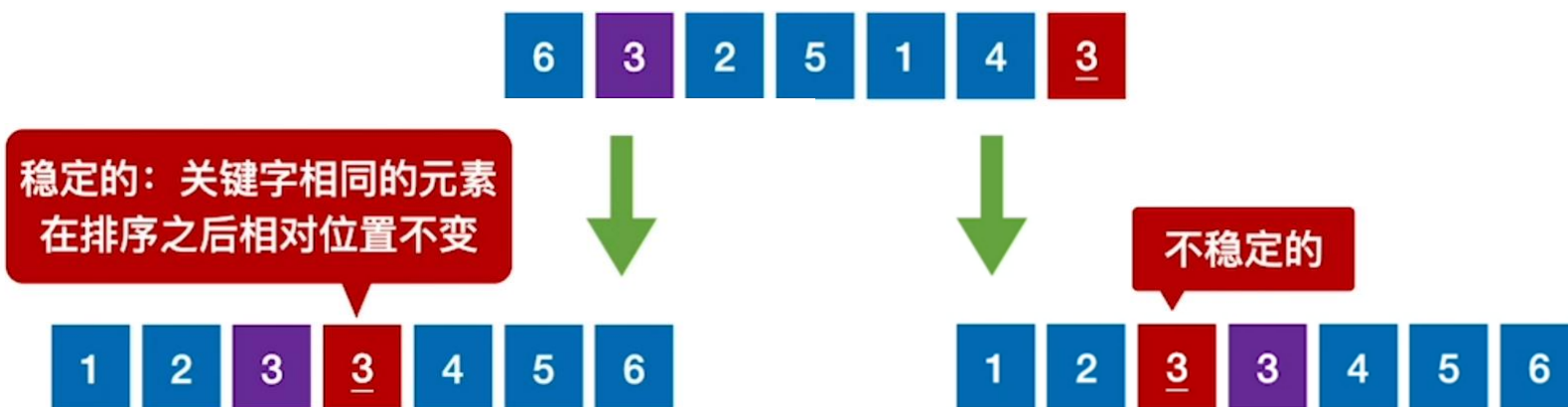
4. 排序算法的好坏如何衡量？

- 时间效率——排序速度（即排序所花费的全部比较次数）
- 空间效率——占内存辅助空间的大小

10.1 概述

4. 排序算法的好坏如何衡量？

- **稳定性**——若两个记录A和B的关键字值相等，但排序后A、B的先后次序保持不变，则称这种排序算法是稳定的。



10.1 概述

5.待排序记录在内存中怎样存储和处理?

- ① 顺序排序——排序时直接移动记录;
- ② 链表排序——排序时只移动指针;
- ③ 地址排序——排序时先移动地址, 最后再移动记录。

注: 地址排序中可以增设一维数组来专门存放记录的地址。

10.1 概述

6. 顺序存储（顺序表）的抽象数据类型如何表示？

注：大多数排序算法都是针对顺序表结构的(便于直接移动元素)

```
# define MAXSIZE 20    //设上课举例的记录数均不超过20个
```

```
typedef int KeyType;    //设关键字为整型量 (int型)
```

```
typedef struct {        //定义每个记录（数据元素）的结构
```

```
    KeyType    key;      //关键字
```

```
    InfoType    otherinfo; //其它数据项
```

```
}RecordType, node;      //例如key表示其中一个分量
```

10.1 概述

6. 顺序存储（顺序表）的抽象数据类型如何表示？

```
typedef struct { //定义顺序表L的结构
    RecordType r [ MAXSIZE +1 ]; //存储顺序表的向量
                                   //r[0]一般作哨兵或缓冲区
    int length ;    //顺序表的长度
}SqList;           //例如L.r或L.length表示其中一个分量
```

r数组是表L中的一个分量且为node类型，
r中某元素的key分量表示为：r[i].key

10.1 概述

7. 内部排序的算法有哪些？

——按排序的规则不同，可分为5类：

- ❖ 插入排序（希尔排序）
- ❖ 交换排序（重点是快速排序）
- ❖ 选择排序（堆排序）
- ❖ 归并排序
- ❖ 基数排序

10.1 概述

7. 内部排序的算法有哪些？

——按排序算法的时间复杂度不同，可分为3类：

- ❖ 简单的排序算法：时间效率低， $O(n^2)$
- ❖ 先进的排序算法：时间效率高， $O(n \log_2 n)$
- ❖ 基数排序算法：时间效率高， $O(d \times n)$

d = 关键字的位数(长度)

10.2 插入排序

插入排序的基本思想是：

每步将一个待排序的对象，按其关键码大小，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。

简言之，边插入边排序，保证子序列中随时都是排好序的。

10.2 插入排序

插入排序有多种具体实现算法：

- 1) 直接插入排序
 - 2) 折半插入排序
 - 3) 2-路插入排序
 - 4) 表插入排序
 - 5) 希尔排序（重点）
- 小改进
- 大改进

10.2 插入排序

1、直接插入排序

最简单的排序法!

新元素插入到哪里? 在已形成的有序表中线性查找, 并在适当位置插入, 把原来位置上的元素向后顺移。

例1: 关键字序列

$T = (13, 6, 3, 31, 9, 27, 5, 11)$

请写出直接插入排序的中间过程序列。

10.2 插入排序

1、直接插入排序

【13】 , 6, 3, 31, 9, 27, 5, 11 原始序列

【6, 13】 , 3, 31, 9, 27, 5, 11 第一趟排序

【3, 6, 13】 , 31, 9, 27, 5, 11

【3, 6, 13, 31】 , 9, 27, 5, 11

【3, 6, 9, 13, 31】 , 27, 5, 11

【3, 6, 9, 13, 27, 31】 , 5, 11

【3, 5, 6, 9, 13, 27, 31】 , 11

【3, 5, 6, 9, 11, 13, 27, 31】 最后一趟排序

10.2 插入排序

1、直接插入排序

```
void InsertSort (SqList &L) { //插入排序
    for(i=2;i<=L.length;++ i){ //在原始无序表L中排序
        if(L.r[i].key < L.r[i-1].key){
            //若L.r[i]较小则插入有序子表内
            L.r[0]= L.r[i];
            //先将待插入的元素放入“哨兵”位置
            for(j=i-1;L.r[0].key<L.r[j].key;--j)
                L.r[j+1]= L.r[j];
            //只要有序子表元素比哨兵大就不断后移
            L.r[j+1]= L.r[0];
            //直到子表元素小于哨兵，将哨兵值送入
            //当前要插入的位置（包括插入到表首）
        } //if
    } //for
} // InsertSort
```

10.2 插入排序

1、直接插入排序

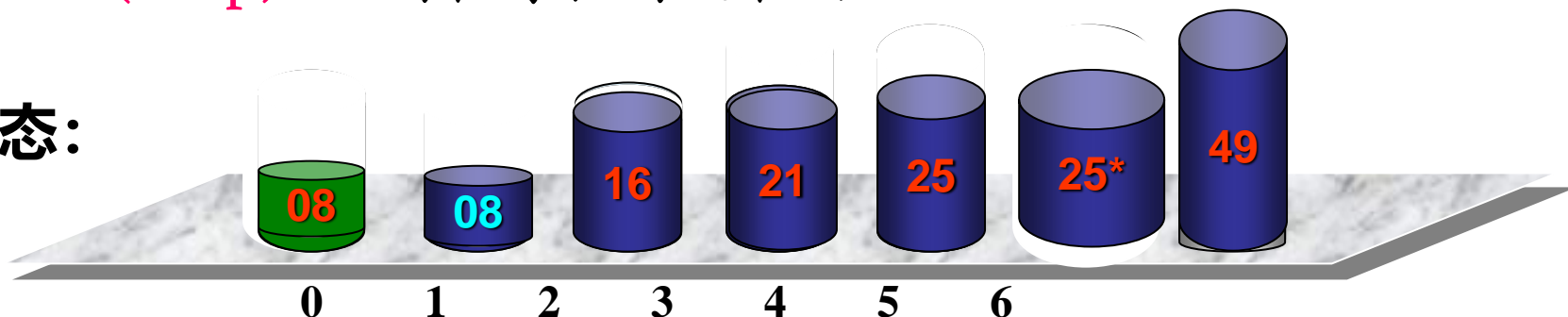
*表示后一个25

例1：关键字序列 $T = (21, 25, 49, 25^*, 16, 08)$ ，

请写出直接插入排序的具体实现过程。

解：假设该序列已存入一维数组 $r[7]$ 中，将 $r[0]$ 作为哨兵
(temp)。则程序执行过程为：

初态：



完成！

$i=1$ $i=2$ $i=3$ $i=4$ $i=5$ $i=6$

10.2 插入排序

1、直接插入排序

- ▣ 若设待排序的对象个数为 $L.length = n$ ，则该算法的主程序执行 $n-1$ 趟。
- ▣ 关键字比较次数和对象移动次数与对象关键字的初始排列有关。
- ▣ 最好情况下，排序前对象已经按关键字大小从小到大有序，每趟只需与前面的有序对象序列的最后一个对象的关键字比较 1 次，总的关键字比较次数为 $n-1$ ，不需要移动元素。

10.2 插入排序

1、直接插入排序

- ▣ 一个记录的辅助存储空间——监视哨
- ▣ 比较次数
 - 最小比较次数: $C_{\min}=n-1$
 - 最大比较次数: $C_{\max} = \sum_{i=1}^{n-1} i = n(n-1)/2$
 - 平均比较次数: $C_{\text{avg}}=(n-1)(n+4)/4$
- ▣ 移动次数
 - 最小移动次数: $M_{\min}=0$
 - 最大移动次数: $M_{\max} = \sum_{i=2}^n (i+1) = (n+4)(n-1)/2$
 - 平均移动次数: $M_{\text{avg}}=(n+8)(n-1)/4$

10.2 插入排序

1、直接插入排序

时间效率：因为在最坏情况下，所有元素的比较次数总和为 $(2+\dots+n)\rightarrow O(n^2)$ ，故时间复杂度为 $O(n^2)$

空间效率：仅占用1个缓冲单元—— $O(1)$

算法的稳定性：因为25*排序后仍然在25的后面——稳定

10.2 插入排序

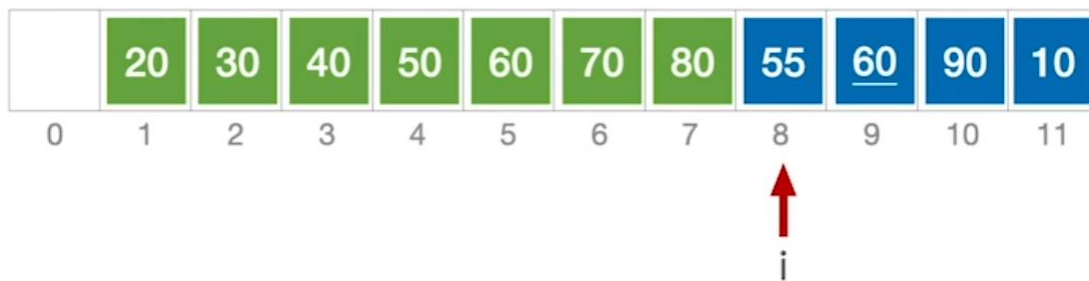
2、折半插入排序

一个想得到的改进方法：既然子表有序且为顺序存储结构，则插入时采用折半查找定可加速定位。（对应程序见教材P267）

10.2 插入排序

2、折半插入排序

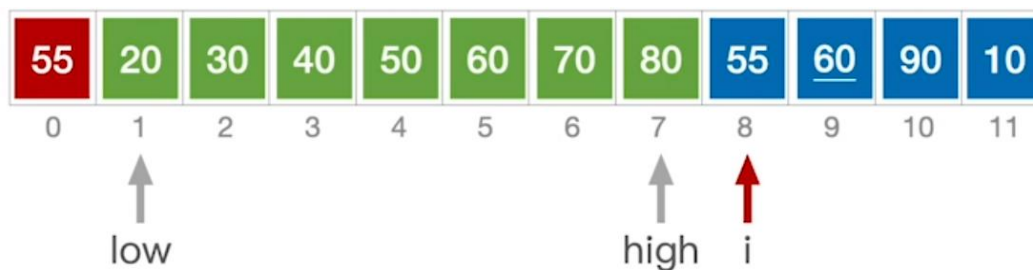
思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

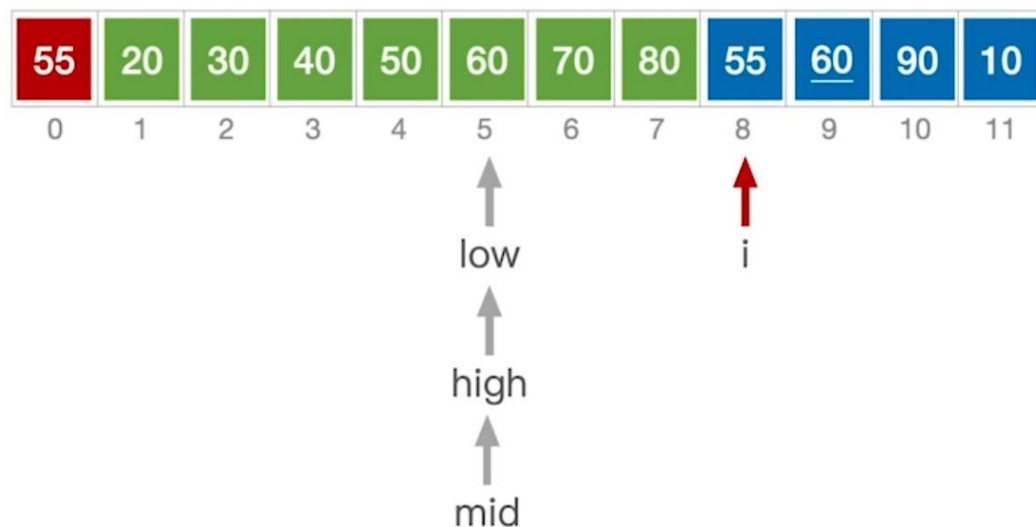
思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素



当 $low > high$ 时折半查找停止，应将 $[low, i-1]$ 内的元素全部右移，并将 $A[0]$ 复制到 low 所指位置

10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素



当 $low > high$ 时折半查找停止，应将 $[low, i-1]$ 内的元素全部右移，并将 $A[0]$ 复制到 low 所指位置

10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置, 再移动元素

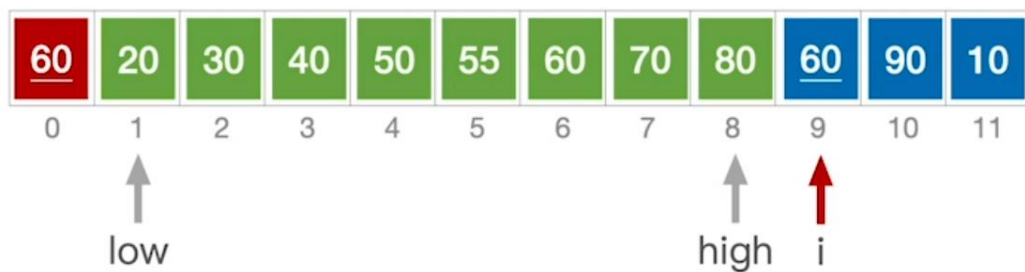


当 $low > high$ 时折半查找停止, 应将 $[low, i-1]$ 内的元素全部右移, 并将 $A[0]$ 复制到 low 所指位置

10.2 插入排序

2、折半插入排序

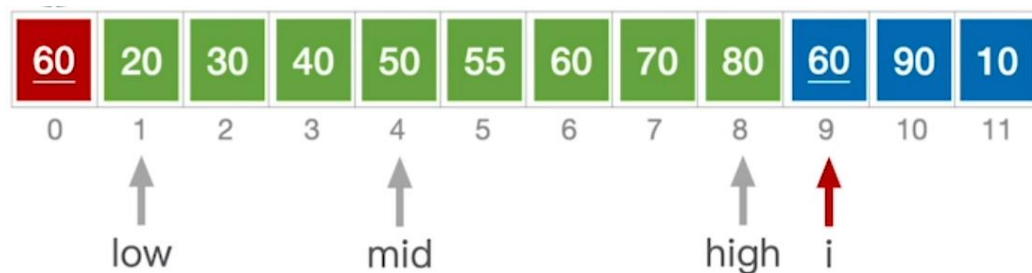
思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素

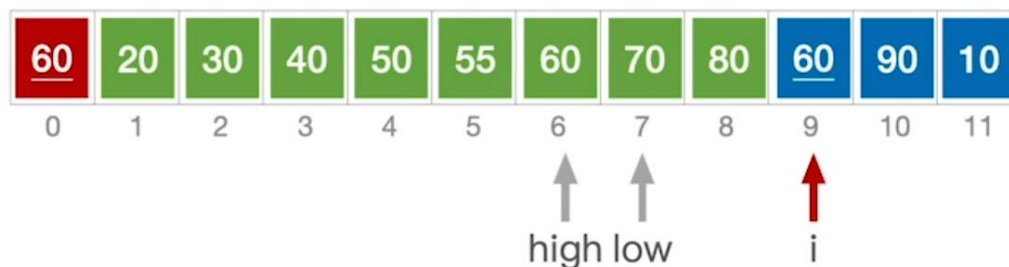


当 $A[mid] == A[0]$ 时，为了保证算法的“稳定性”，应继续在 mid 所指位置右边寻找插入位置

10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置, 再移动元素

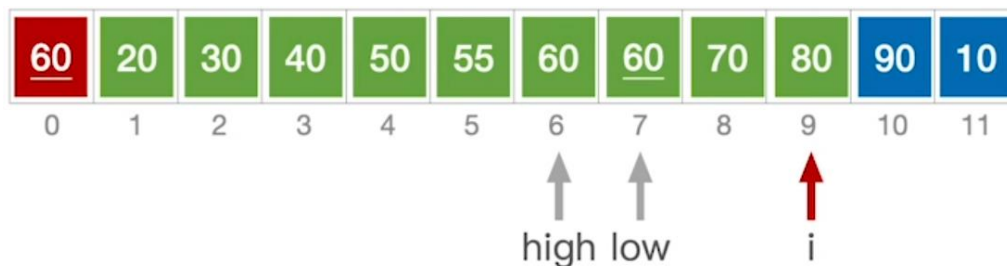


当 $low > high$ 时折半查找停止, 应将 $[low, i-1]$ 内的元素全部右移, 并将 $A[0]$ 复制到 low 所指位置当 $A[mid] = A[0]$ 时, 为了保证算法的“稳定性”, 应继续在 mid 所指位置右边寻找插入位置

10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置, 再移动元素

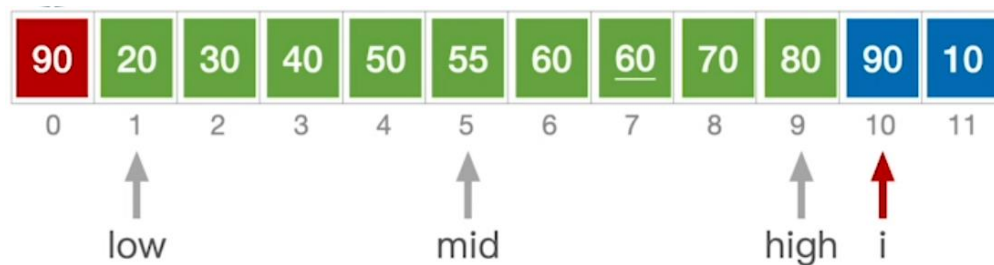


当 $low > high$ 时折半查找停止, 应将 $[low, i-1]$ 内的元素全部右移, 并将 $A[0]$ 复制到 low 所指位置当 $A[mid] = A[0]$ 时, 为了保证算法的“稳定性”, 应继续在 mid 所指位置右边寻找插入位置

10.2 插入排序

2、折半插入排序

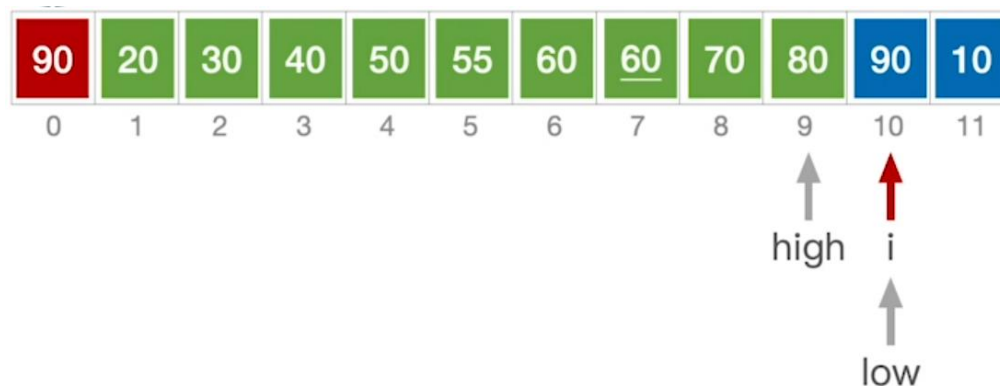
思路:先用折半查找找到应该插入的位置，再移动元素



10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置, 再移动元素

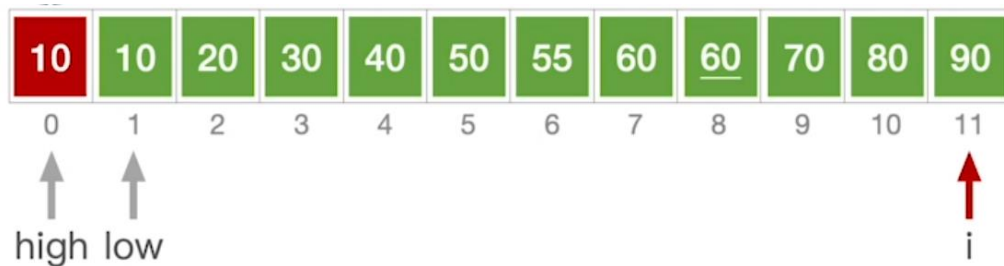


当 $low > high$ 时折半查找停止, 应将 $[low, i-1]$ 内的元素全部右移, 并将 $A[0]$ 复制到 low 所指位置

10.2 插入排序

2、折半插入排序

思路:先用折半查找找到应该插入的位置，再移动元素



当 $low > high$ 时折半查找停止，应将 $[low, i-1]$ 内的元素全部右移，并将 $A[0]$ 复制到 low 所指位置

10.2 插入排序

2、折半插入排序

例：待排序数据：2，1，6，7，4

	有序表	无序表
	2	1， 6， 7， 4
第一次比较	1， 2	6， 7， 4
第二次比较	1， 2， 6	7， 4
第三次比较	1， 2， 6， 7	4
第四次比较	1， 2， 4， 6， 7	

10.2 插入排序

2、折半插入排序

```
void BInsertSort(SqList &L) {  
    int low,high,mid;  
    for (int 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  
            mid = (low+high)/2; // 折半  
            if (LT(L.r[0].key,L.r[mid].key)) high=mid-1;//插入点在前半  
            else low=mid+1; // 插入点在后半区  
        }  
        for (int 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  
    }  
}
```

10.2 插入排序

2、折半插入排序

优点：比较次数大大减少，全部元素比较次数仅为 $O(n\log_2 n)$ 。

时间效率：虽然比较次数大大减少，可惜移动次数并未减少，所以排序效率仍为 $O(n^2)$ 。

空间效率：仍为 $O(1)$

稳定性：稳定

思考：折半插入排序还可以改进吗？

能否减少移动次数？

10.2 插入排序

3、2-路插入排序

思路：增开辅助数组b, 大小与a相同。

首先将a[1]赋值给b[1], 然后以b[1]内容为中值, 将a[i]元素逐个插入到b[1]值之前或之后的有序序列中。

计算机处理时, 可把d设为循环向量(队列), 再设头尾两个指针。

10.2 插入排序

3、2-路插入排序

这是对直接插入排序的一种改进，其目的是减少排序过程中的移动次数。

代价：需要增加 n 个记录的辅助空间。

10.2 插入排序

3、2-路插入排序

例：使用 2-路插入排序对无序表{49,38,65,97,76,13,27,49}排序

```
int n = 8;  
int *b = (int*)malloc(n * sizeof(int));
```

a[]	0	1	2	3	4	5	6	7
	49	38	65	97	76	13	27	49

```
int first,final;  
first = final = 0;  
b[0]=a[0];
```

b[]	0	1	2	3	4	5	6	7
	49							

```
for (int i = 1; i < n; i++)
```

10.2 插入排序

3、2-路插入排序

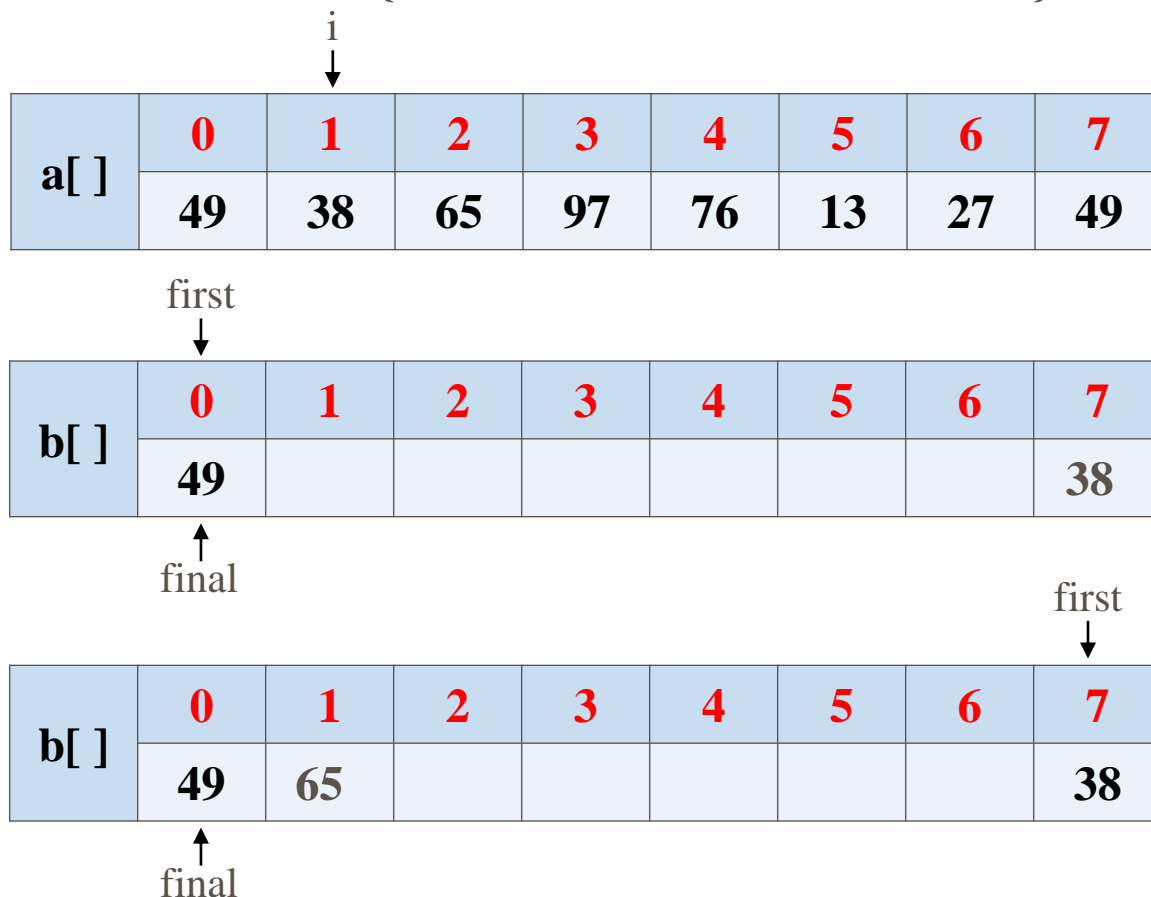
例：使用 2-路插入排序对无序表{49,38,65,97,76,13,27,49}排序

$a[i]=38 < b[first]=49$

```
if (a[i] < b[first]) {  
    first = (first-1+n)%n;  
    b[first] = a[i];  
}
```

$a[i]=65 > b[final]=49$

```
if (a[i] > b[final]) {  
    final++;  
    b[final] = a[i];  
}
```



10.2 插入排序

3、2-路插入排序

例：使用 2-路插入排序对无序表{49,38,65,97,76,13,27,49}排序

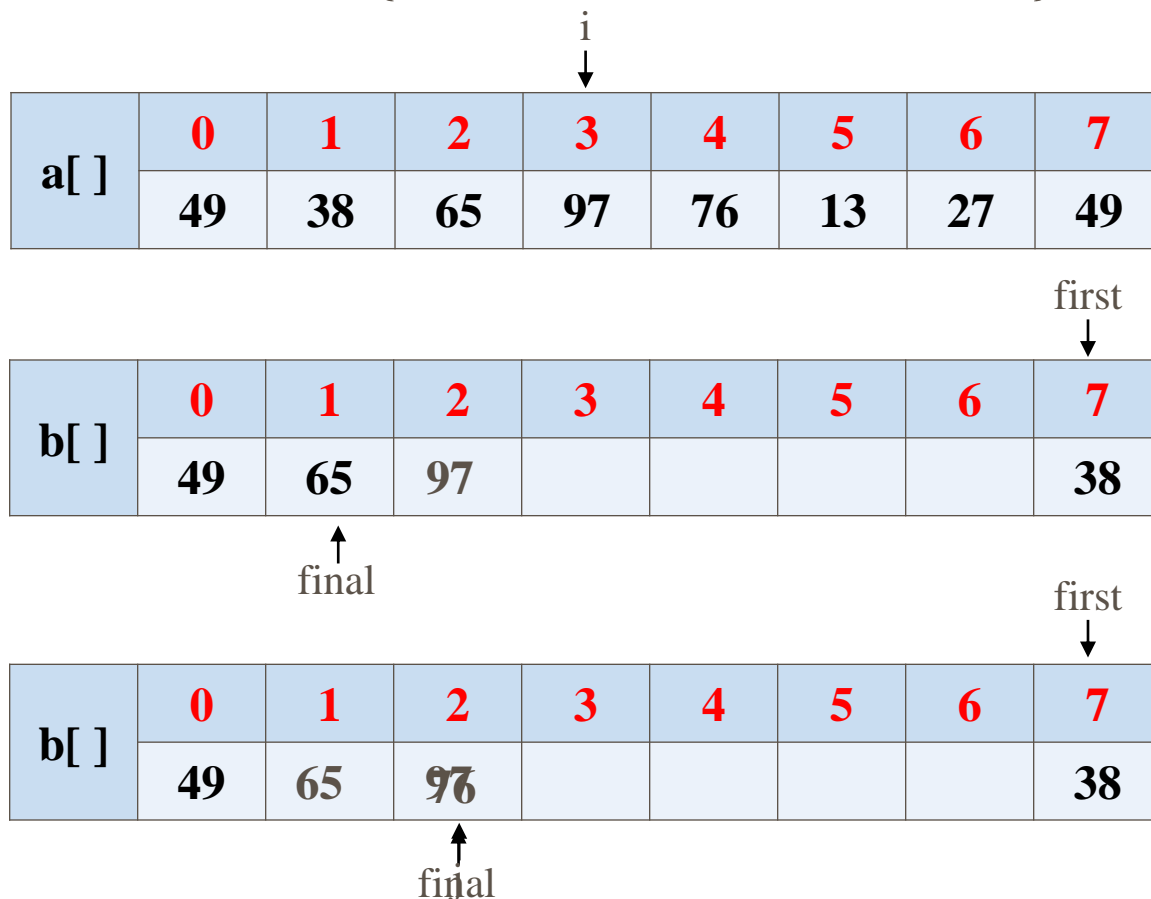
$a[i]=97 > b[final]=65$

```
if (a[i] > b[final]) {  
    final++;  
    b[final] = a[i];  
}
```

$a[i]=76$ 不小于 $b[first]$ 也不大于 $b[final]$ ，从 $a[final]$ 处按直接插入排序进行插入

```
final++; b[final] = b[final - 1];  
for (j = final - 1; b[j-1] > a[i]; j--) {  
    b[j] = b[j - 1];  
}  
b[j] = a[i];
```

$j=(j-1+n)\%n;$



10.2 插入排序

3、2-路插入排序

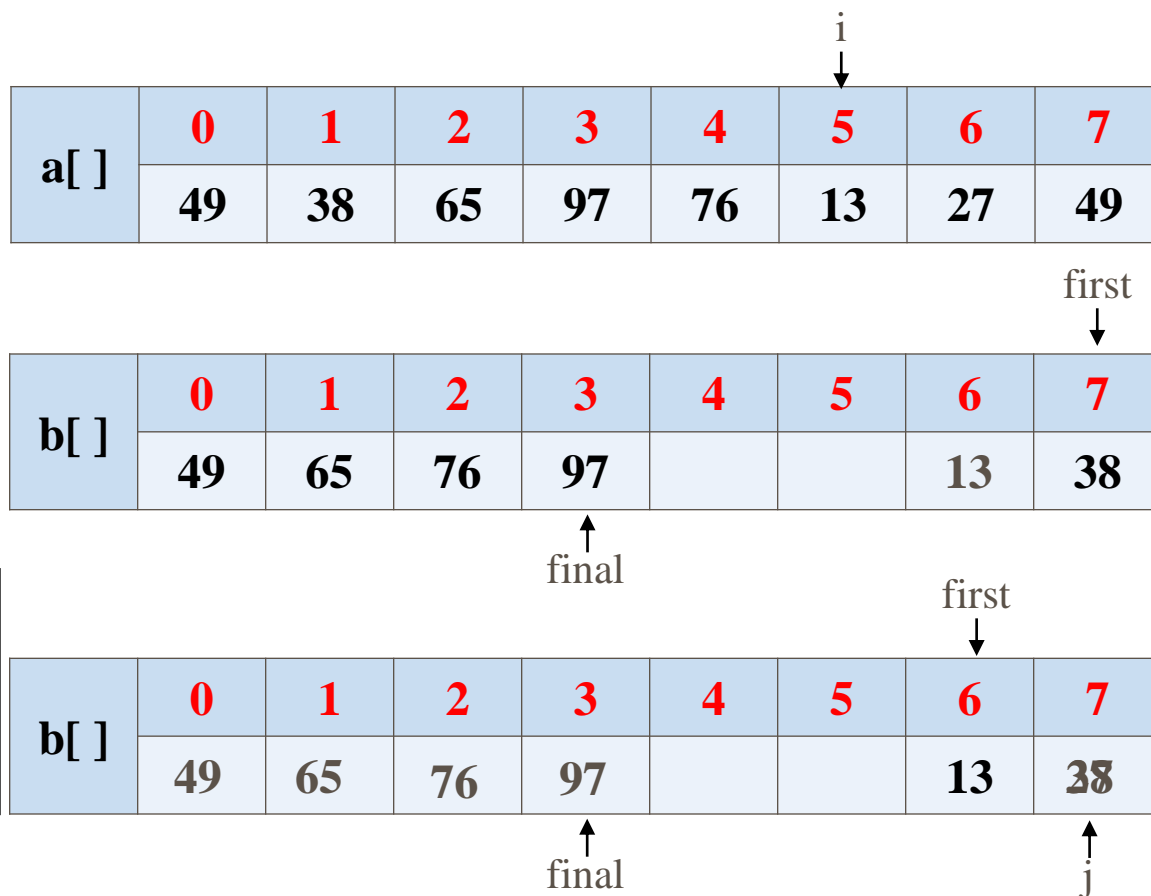
例：使用 2-路插入排序对无序表{49,38,65,97,76,13,27,49}排序

$a[i]=13$, $first--$, $b[first]=a[i]$

```
if (a[i] < b[first]) {  
    first = (first-1+n)%n;  
    b[first] = a[i];  
}
```

$a[i]=27$, $final++$, 97后移, j 指向 $final-1$

```
final++; b[final] = b[final - 1];  
for (j=final-1; b[(j-1+n)%n] > a[i]; j--) {  
    b[j] = b[(j-1+n)%n];  
}  
b[j] = a[i];
```



10.2 插入排序

3、2-路插入排序

例：使用 2-路插入排序对无序表{49,38,65,97,76,13,27,49}排序

$a[i]=49 < b[first]=49$

```
final++;b[final] = b[final - 1];  
for (j=final-1; $b[(j-1+n)\%n] > a[i]$ ;j--) {  
    b[j] = b[ $(j-1+n)\%n$ ];  
}  
b[j] = a[i];
```

复制b到a

```
for (int k = 0; k < n; k++) {  
    a[k] = b[first];  
    first=(first+1)%n;  
} //可能越界的加入模除
```

a[]	0	1	2	3	4	5	6	7
	49	38	65	97	76	13	27	49

i
↓

b[]	0	1	2	3	4	5	6	7
	38	49	49	76	97		13	27

first
↓

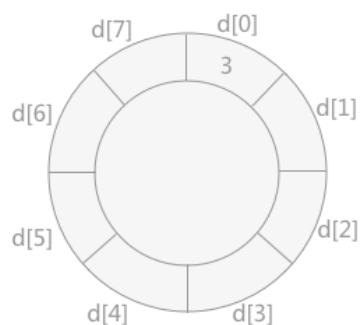
j final
↑ ↑

a[]	0	1	2	3	4	5	6	7
	13	27	38	49	49	65	76	97

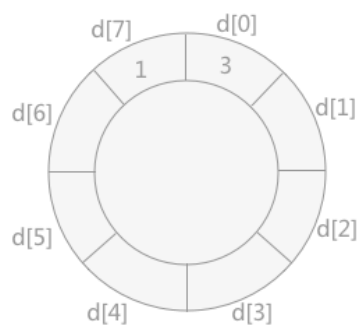
10.2 插入排序

3、2-路插入排序

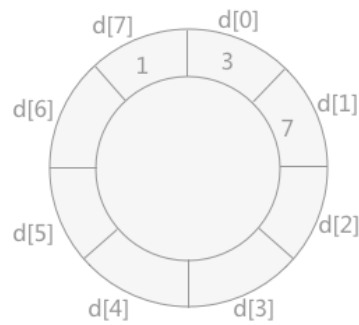
例：使用 2-路插入排序算法对无序表{3,1,7,5,2,4,9,6}排序



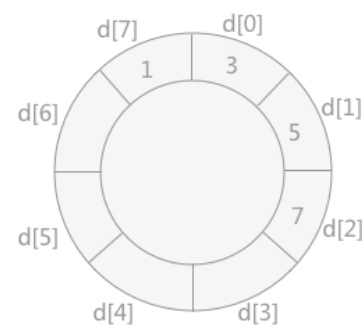
(1)



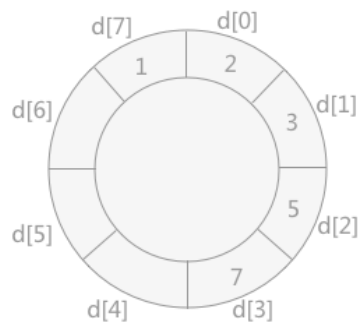
(2)



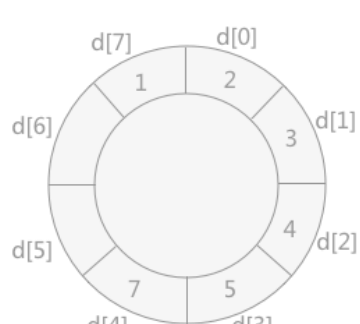
(3)



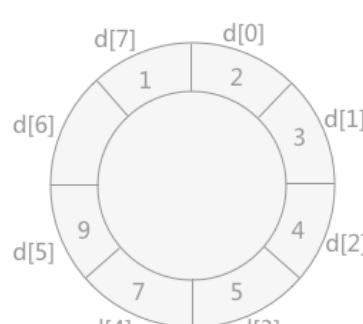
(3)



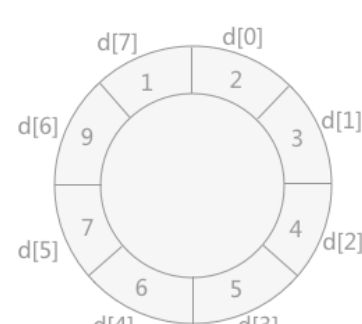
(4)



(5)



(6)



(7)

10.2 插入排序

4、表插入排序

基本思想：在顺序存储结构中，给每个记录增开一个指针分量，在排序过程中将指针内容逐个修改为已经整理（排序）过的后继记录地址。

优点：在排序过程中不移动元素，只修改指针。

10.2 插入排序

4、表插入排序

```
#define SIZE 100
typedef struct{
    RcdType rc;
    int next;
}SLNode;
typedef struct{
    SLNode r[SIZE];
    int length;
}SLinkListType;
```

初始状态	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	1	0	-	-	-	-	-	-	-

i=2	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	2	0	1	-	-	-	-	-	-

i=3	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	2	3	1	0	-	-	-	-	-

i=4	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	2	3	1	4	0	-	-	-	-

10.2 插入排序

4、表插入排序

```
#define SIZE 100
typedef struct{
    RcdType rc;
    int next;
}SLNode;
typedef struct{
    SLNode r[SIZE];
    int length;
}SLinkListType;
```

i=5	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	2	3	1	5	0	4	-	-	-

i=6	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	6	3	1	5	0	4	2	-	-

i=7	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	6	3	1	5	0	4	7	2	-

i=8	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	6	8	1	5	0	4	7	2	3

10.2 插入排序

4、表插入排序

□ 重排过程 ($p > i$, 直接互换; $p < i$, 继续顺链查找直到 $p \geq i$ 为止)

初始状态	0	1	2	3	4	5	6	7	8
	MAXINT	49	38	65	97	76	13	27	49^
	6	8	1	5	0	4	7	2	3

i=1 p=6	0	1	2	3	4	5	6	7	8
	MAXINT	13	38	65	97	76	49	27	49^
	6	(6)	1	5	0	4	8	2	3

i=2 p=7	0	1	2	3	4	5	6	7	8
	MAXINT	13	27	65	97	76	49	38	49^
	6	(6)	(7)	5	0	4	7	1	3

i=3 p=(2) 7	0	1	2	3	4	5	6	7	8
	MAXINT	13	27	38	97	76	49	65	49^
	6	(6)	(7)	(7)	0	4	7	5	3

```
void Arrange (SLinkListType &SL){
    p = SL.r[0].next;
    for (i = 1; i < SL.length; ++ i){
        while (p < i) p = SL.r[p].next;
        q = SL.r[p].next;
        if (p != i) {
            SL.r[p] <----> SL.r[i];
            L.r[i].next = p;
        }
        p = q;
    }
}
```

10.2 插入排序

4、表插入排序

□ 重排过程 ($p > i$, 直接互换; $p < i$, 继续顺链查找直到 $p \geq i$ 为止)

i=4 p=(1), 6	0	1	2	3	4	5	6	7	8
	MAXINT	13	27	38	97	76	49	65	49^
	6	(6)	(7)	(7)	(6)	4	7	5	3

i=5 p=8	0	1	2	3	4	5	6	7	8
	MAXINT	13	27	38	49	76	97	65	49^
	6	(6)	(7)	(7)	(6)	(8)	0	5	4

i=6 p=(3), 7	0	1	2	3	4	5	6	7	8
	MAXINT	13	27	38	49	49^	65	97	76
	6	(6)	(7)	(7)	(6)	(8)	(7)	5	4

i=7 p=(5), 8	0	1	2	3	4	5	6	7	8
	MAXINT	13	27	38	49	49^	65	76	97
	6	(6)	(7)	(7)	(6)	(8)	(7)	(8)	0

```
void Arrange (SLinkListType &SL){  
    p = SL.r[0].next;  
    for (i = 1; i<SL.length; ++ i){  
        while (p<i) p = SL.r[p].next;  
        q = SL.r[p].next;  
        if (p!= i) {  
            SL.r[p]<---->SL.r[i];  
            L.r[i].next = p;  
        }  
        p = q;  
    }  
}
```


10.2 插入排序

4、表插入排序

- ① 无需移动记录，只需修改 $2n$ 次指针值。但由于比较次数没有减少，故时间效率仍为 $O(n^2)$ 。
- ② 空间效率肯定低，因为增开了指针分量（但在运算过程中没有用到更多的辅助单元）。
- ③ 稳定性：49和49*排序前后次序未变，稳定。

注：此算法得到的只是一个有序链表，查找记录时只能满足顺序查找方式。

10.2 插入排序

5、希尔排序

基本思想：先将整个待排记录序列分割成若干子序列，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。

又称缩小增量排序

10.2 插入排序

5、希尔排序

技巧：子序列的构成不是简单地“逐段分割”，而是将相隔某个增量 dk 的记录组成一个子序列，让增量 dk 逐趟缩短（例如依次取5,3,1），直到 $dk=1$ 为止。

优点：让关键字值小的元素能很快前移，且序列若基本有序时，再用直接插入排序处理，时间效率会高很多。

10.2 插入排序

5、希尔排序

关键字序列 $T=(49, 38, 65, 97, 76, 13, 27, 49^*, 55, 04)$,

希尔排序的具体实现过程。

$r[i]$	0	1	2	3	4	5	6	7	8	9	10
初态:		49	38	65	97	76	13	27	49*	55	04
第1趟 (dk=5)		13	27	49*	55	04	49	38	65	97	76
第2趟 (dk=3)		13	04	49*	38	27	49	55	65	97	76
第3趟 (dk=1)		04	13	27	38	49*	49	55	65	76	97

10.2 插入排序

5、希尔排序

算法分析：开始时 dk 的值较大，子序列中的对象较少，排序速度较快；随着排序进展， dk 值逐渐变小，子序列中对象个数逐渐变多，由于前面工作的基础，大多数对象已基本有序，所以排序速度仍然很快。

10.2 插入排序

5、希尔排序

```
void ShellSort(SqList &L, int dlta[ ], int t){
```

```
    //按增量序列dlta[0...t-1]对顺序表L作Shell排序
```

```
    for(k=0; k<t; ++k)
```

```
        ShellInsert(L, dlta[k]); //增量为dlta[k]的一趟插入排序
```

```
} // ShellSort
```

dk值依次装在dlta[t]中

10.2 插入排序

5、希尔排序

```
void ShellInsert(SqList &L, int dk) {  
    //对顺序表L进行一趟增量为dk的Shell排序, dk为步长因子  
    for(i=dk+1; i<=L.length; ++i)  
        if(r[i].key < r[i-dk].key) { //开始将r[i] 插入有序增量子表  
            r[0]=r[i]; //暂存在r[0], 此处r[0]仍是哨兵  
            for(j=i-dk; j>0&&(r[0].key<r[j].key); j-=dk) r[j+dk]=r[j];  
            r[j+dk]=r[0];  
        } //if  
} //ShellInsert
```

//关键字较大的记录在子表中不断后移
//在本趟结束时将r[i]插入到正确位置

理解难点：整理动作是二合一的，r[0] 仍是每个dk子集的哨兵，用于子集的彻底排序！

10.2 插入排序

5、希尔排序

时间效率分析

$O(n^{1.25}) \sim O(1.6n^{1.25})$ —— 由经验公式得到

空间效率： $O(1)$ —— 因为仅占用1个缓冲单元

算法的稳定性： **不稳定**——

因为49*排序后却到了49的前面

10.2 插入排序

5、希尔排序

算法2

数据变化

下标	①	②	③	④	⑤	⑥	⑦	⑧	⑨
	9	1	5	8	3	7	4	6	2

```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

dk=4, i=5

L->r[i]=3,
L->r[i-dk]=9

10.2 插入排序

5、希尔排序

算法2

数据变化

下标	①	②	③	④	⑤	⑥	⑦	⑧	⑨	
	3	9	1	5	8	3	7	4	6	2

```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

dk=4, i=5

L->r[i]=3,
L->r[0]=3

10.2 插入排序

5、希尔排序

算法2

数据变化



```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

dk=4,i=5,j=1

L->r[5]=L->r[1],
L->r[1]=L->r[0]

10.2 插入排序

5、希尔排序

算法2

数据变化



```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

dk=4,i=6

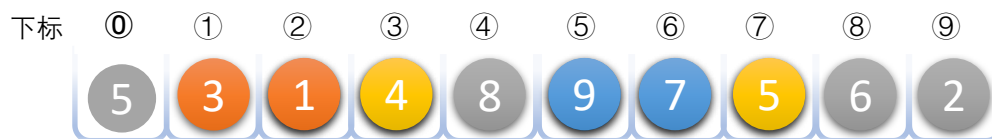
L->r[6]=7,
L->r[2]=1

10.2 插入排序

5、希尔排序

算法2

数据变化



```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

dk=4,i=7

L->r[7]=4,
L->r[3]=5

10.2 插入排序

5、希尔排序

算法2

数据变化



```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

dk=4,i=8

L->r[8]=8,
L->r[4]=6

10.2 插入排序

5、希尔排序

算法2

数据变化



```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

dk=4,i=9

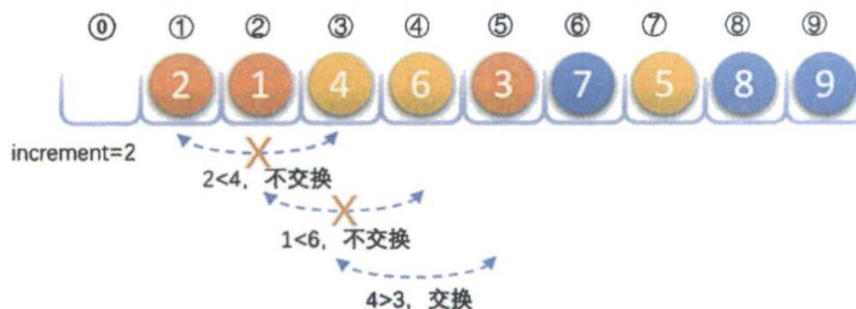
L->r[9]=9,
L->r[5]=3,
L->r[1]=2

10.2 插入排序

5、希尔排序

算法2

数据变化



```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++){  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

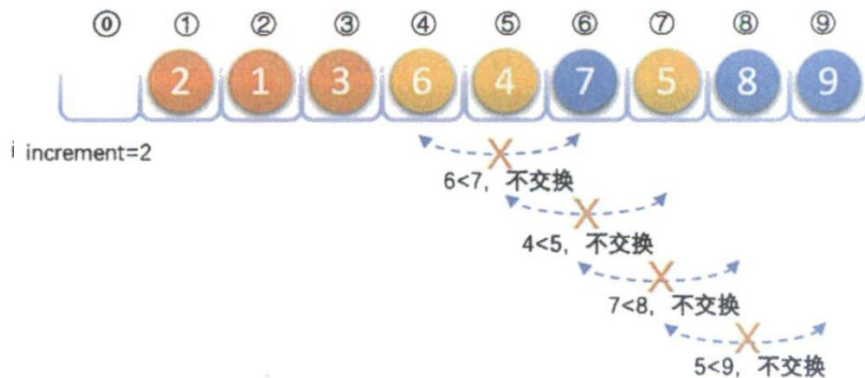
$$dk=4/3+1=2$$

10.2 插入排序

5、希尔排序

算法2

```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```



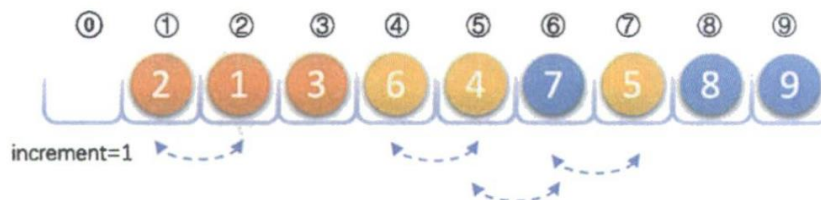
$$dk = 4/3 + 1 = 2$$

10.2 插入排序

5、希尔排序

算法2

数据变化



```
void ShellSort(SqList *L) {  
    int i,j,k=0;  
    int dk=L->length;  
    do{  
        dk=dk/3+1;  
        for(i=dk+1;i<=L->length;i++) {  
            if (L->r[i]<L->r[i-dk]) {  
                L->r[0]=L->r[i];  
                for(j=i-dk;j>0 && L->r[0]<L->r[j];j-=dk)  
                    L->r[j+dk]=L->r[j];  
                L->r[j+dk]=L->r[0];  
            }  
        }  
    }  
    while(dk>1);  
}
```

$$dk=2/3+1=1$$

正在答疑
