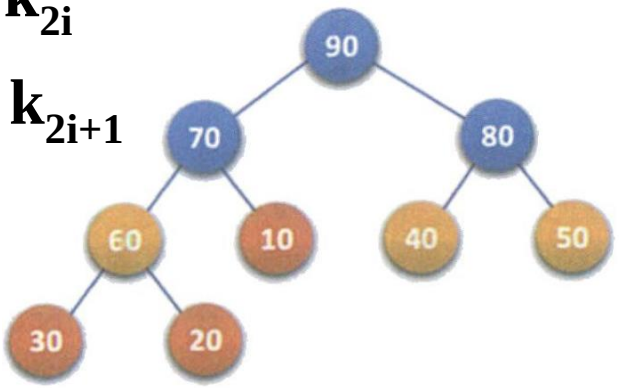
回顾

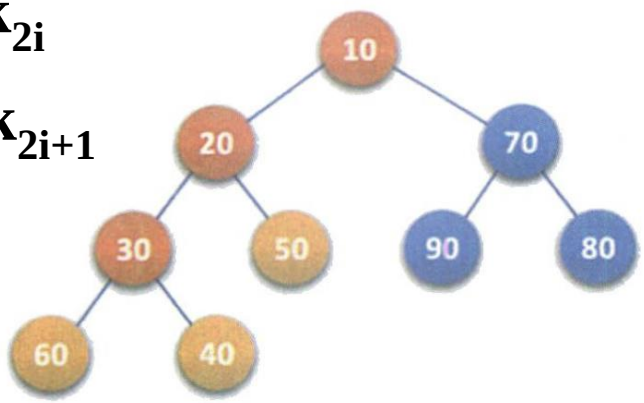
□ 堆排序

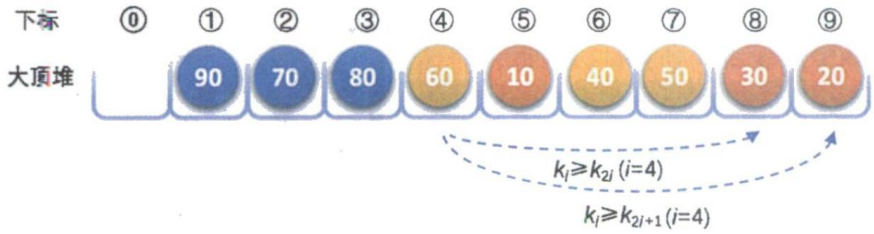
堆是具有以下性质的完全二叉树：每个结点的值都大于或等于其左右孩子结点的值，称为大顶堆；或者每个结点的值都小于或等于其左右孩子结点的值，称为小顶堆。

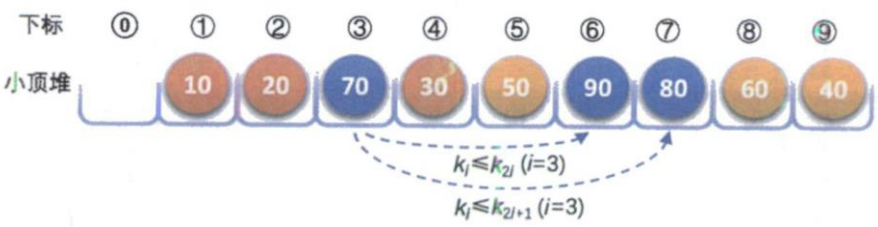
$$\begin{cases} k_i \geq k_{2i} \\ k_i \geq k_{2i+1} \end{cases}$$



$$\begin{cases} k_i \leq k_{2i} \\ k_i \leq k_{2i+1} \end{cases}$$







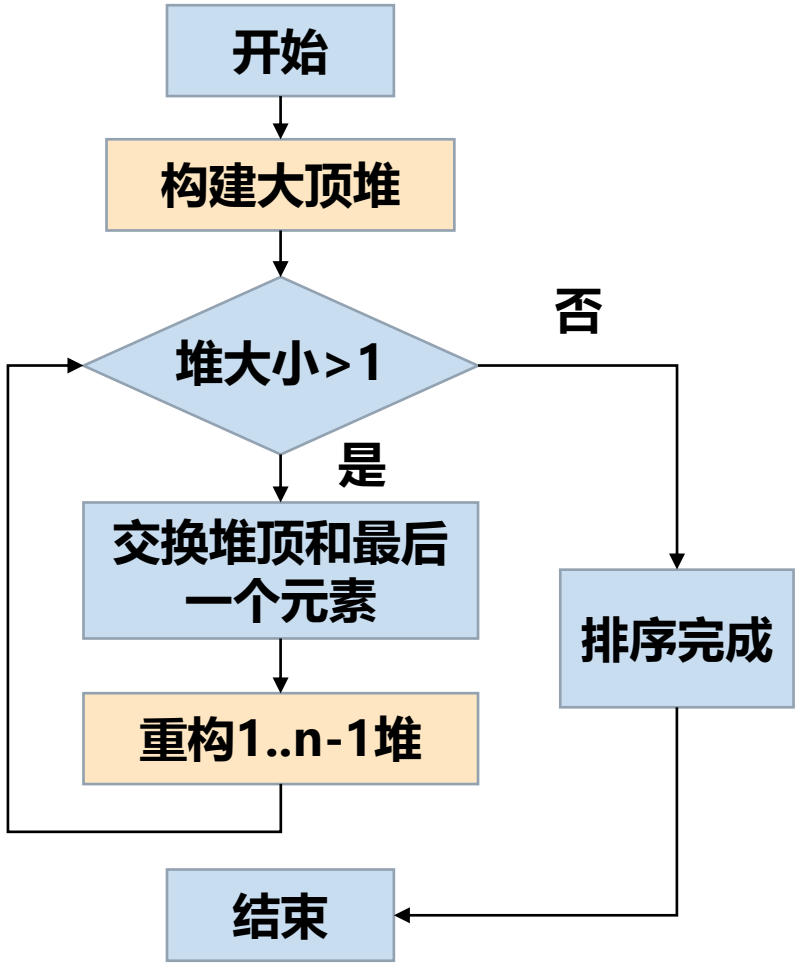
回顾

□ 堆排序

① 首先将待排序的数组构造成一个大根堆，此时，整个数组的最大值就是堆结构的顶端

② 将顶端的数与末尾的数交换，此时，末尾的数为最大值，剩余待排序数组个数为n-1

③ 将剩余的n-1个数再构造成大根堆，再将顶端数与n-1位置的数交换，如此反复执行，便能得到有序数组



回顾

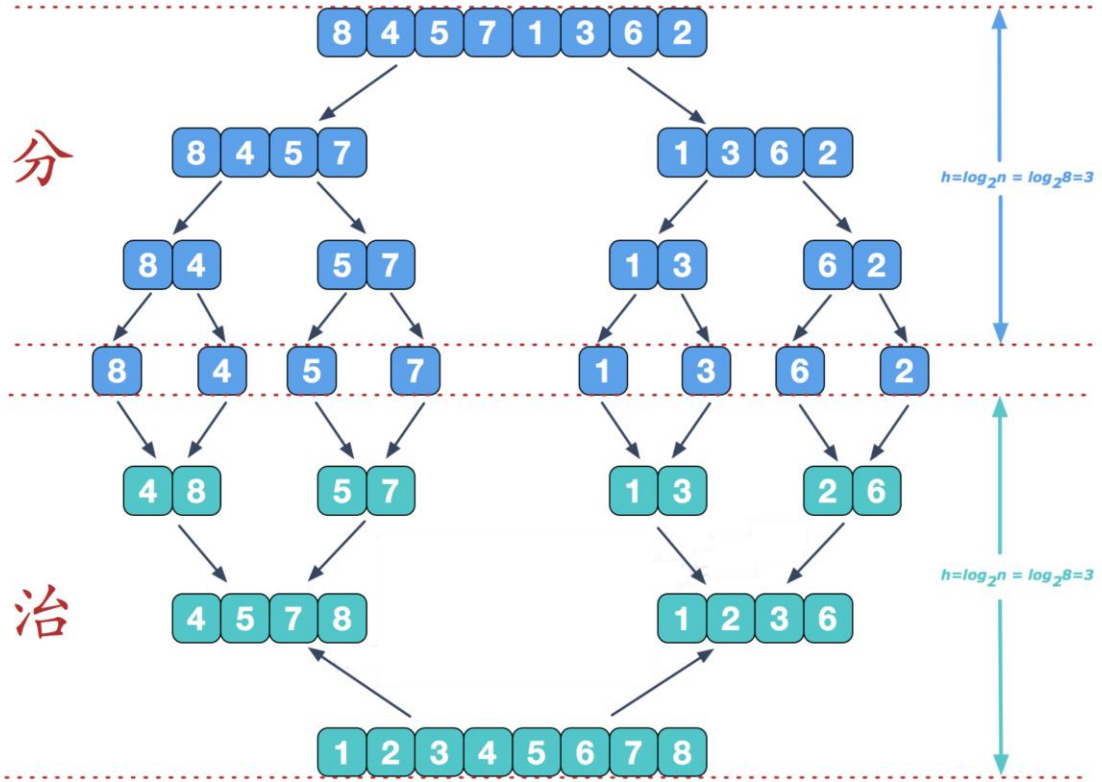
□ 归并排序

归并排序是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列

① 把长度为n的输入序列分成两个长度为n/2的子序列

② 对这两个子序列分别采用归并排序

③ 将两个排序好的子序列合并成一个最终的排序序列



第10章 内部排序

10.6 基数排序

基数排序的基本思想是:

借助多关键字排序的思想对单逻辑关键字进行排序。

即：用关键字**不同的位值**进行排序。

要讨论的问题:

1. 什么是“**多关键字**”排序？实现方法?
2. 单逻辑关键字怎样“**按位值**”排序?

10.6 基数排序

1. 什么是“多关键字”排序？实现方法？

例1：对一副扑克牌该如何排序？

若规定花色和面值的顺序关系为：

花色：♦ < ♣ < ♥ < ♠

面值：2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K < A

则可以先按花色排序，花色相同者再按面值排序；

也可以先按面值排序，面值相同者再按花色排序。

10.6 基数排序

1. 什么是“多关键字”排序？实现方法？

例2：职工分房该如何排序？

学校规定：先以总分排序（职称分＋工龄分）；

总分相同者，再按配偶总分排序，其次按配偶职称、工龄、人口……等等排序。

以上两例都是典型的多关键字排序！

10.6 基数排序

一般情况下，假定有一个 n 个对象的序列 $\{V_0, V_1, \cdots, V_{n-1}\}$，且每个对象 V_i 中含有 d 个关键字

如果对于序列中任意两个对象 V_i 和 V_j $(0 \le i < j \le n-1)$ 都满足：

$$\left(K_i^1, K_i^2, \cdots, K_i^d\right)$$

$$\left(K_i^1, K_i^2, \cdots, K_i^d\right) < \left(K_j^1, K_j^2, \cdots, K_j^d\right)$$

则称序列对关键字 $(K^1, K^2, \cdots, K^d)$ 有序。其中，K^1 称为最高位关键字，K^d 称为最低位关键字。

如果关键字是由多个数据项组成的数据项组，则依据它进行排序时就需要利用多关键字排序。

10.6 基数排序

多关键字排序的实现方法通常有两种：

- 最高位优先法**MSD (Most Significant Digit first)**
- 最低位优先法**LSD (Least Significant Digit first)**

排序后形成的有序序列叫做词典有序序列。

10.6 基数排序

- 最高位优先法通常是一个递归的过程：
  - 先根据最高位关键字K^1排序,得到若干对象组，对象组中每个对象都有相同关键字K^1
  - 再分别对每组中对象根据关键字K^2进行排序，按K^2值的不同，再分成若干个更小的子组，每个子组中的对象具有相同的K^1和K^2值。
  - 依此重复，直到对关键字K^d完成排序为止。
  - 最后，把所有子组中的对象依次连接起来，就得到一个有序的对象序列。

10.6 基数排序

- 最低位优先法
  - 首先依据最低位关键字K^d对所有对象进行一趟排序
  - 再依据次低位关键字K^{d-1}对上一趟排序的结果再排序
  - 依次重复，直到依据关键字K^1最后一趟排序完成，就可以得到一个有序的序列。

使用这种排序方法对每一个关键字进行排序时，不需要再分组，而是整个对象组都参加排序。

10.6 基数排序

例：对一副扑克牌该如何排序?

答：若规定花色为第一关键字（高位），面值为第二关键字（低位），则使用MSD和LSD方法都可以达到排序目的。

10.6 基数排序

MSD方法的思路：先设立4个花色“箱”，将全部牌按花色分别归入4个箱内（每个箱中有13张牌）；然后对每个箱中的牌按面值进行插入排序（或其它稳定算法）。

LSD方法的思路：先按面值分成13堆（每堆4张牌），然后对每堆中的牌按花色进行排序（用插入排序等稳定的算法）。

想一想：用哪种方法更快些？

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

思路: 设n个记录的序列为: $\{V_0, V_1, \cdots, V_{n-1}\}$, 可以把每个记录$V_i$的单关键码$K_i$看成是一个$d$元组$(K_i^1, K_i^2, \cdots, K_i^d)$, 则其中的每一个分量$K_i^j$ $(1 \le j \le d)$也可看成是一个关键字。

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

注1： K_i^1 = 最高位，K_i^d = 最低位；K_i共有d位，可看成d元组；

注2： 每个分量K_i^j $(1 \le j \le d)$ 有$radix$种取值，则称$radix$为基数。

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

例1：关键码984可以看成是 3 元组；基数*radix*为 10 。

(9, 8, 4)

(0, 1, …, 9)

例2：关键码dian可以看成是 4 元组；基数*radix*为 26 。

(d, i, a, n)

(a, b, …, z)

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

614	738	921	485	637	101	215	530	790	306

第一趟分配（按个位排）

e[0]	e[1]	e[2]	e[3]	e[4]	e[5]	e[6]	e[7]	e[8]	e[9]
530	921			614	485	306	637	738	
790	101				215				

第一趟收集

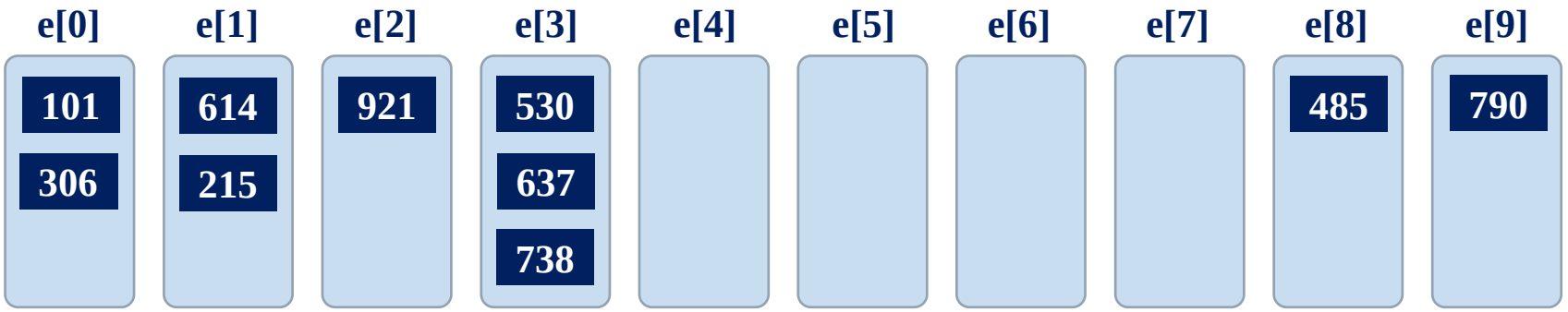
530	790	921	101	614	485	215	306	637	738

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

第一趟:

第二趟分配（按十位排）



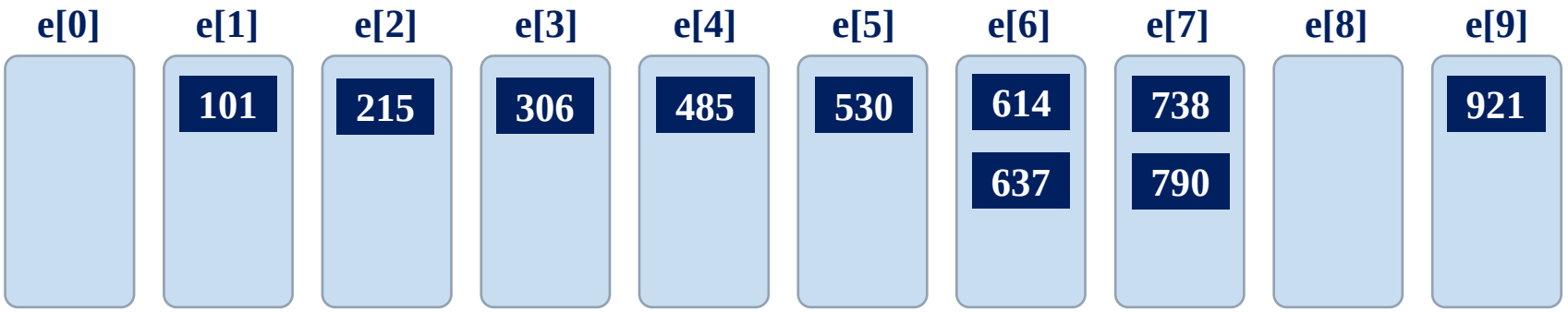
第二趟收集

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

第二趟: 101 306 614 215 921 530 637 738 485 790

第三趟分配（按百位排）



第三趟收集

101 215 306 485 530 614 637 738 790 921

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

讨论：是借用MSD方式来排序呢，还是借用LSD方式?

例：初始关键字序列T=（32, 13, 27, 32*, 19，33），请分别用MSD和LSD进行排序，并讨论其优缺点。

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

法1（MSD）：原始序列：32，13，　27，　32*，19，　33

先按高位K_i^1排序：（13，19），27，（32，32*，33）

再按低位K_i^2排序：　13，19，27，　32，32*，33

因为有分组，故此算法需递归实现。

10.6 基数排序

2. 单逻辑关键字怎样“按位值”排序?

法2（LSD）： 原始序列： 32, 13, 27, 32*, 19 , 33

先按低位K_i^2排序： 32, 32*, 13, 33, 27, 19

再按高位K_i^1排序： 13, 19 , 27, 32, 32*, 33

无需分组，易编程实现!

10.6 基数排序

计算机怎样实现LSD算法?

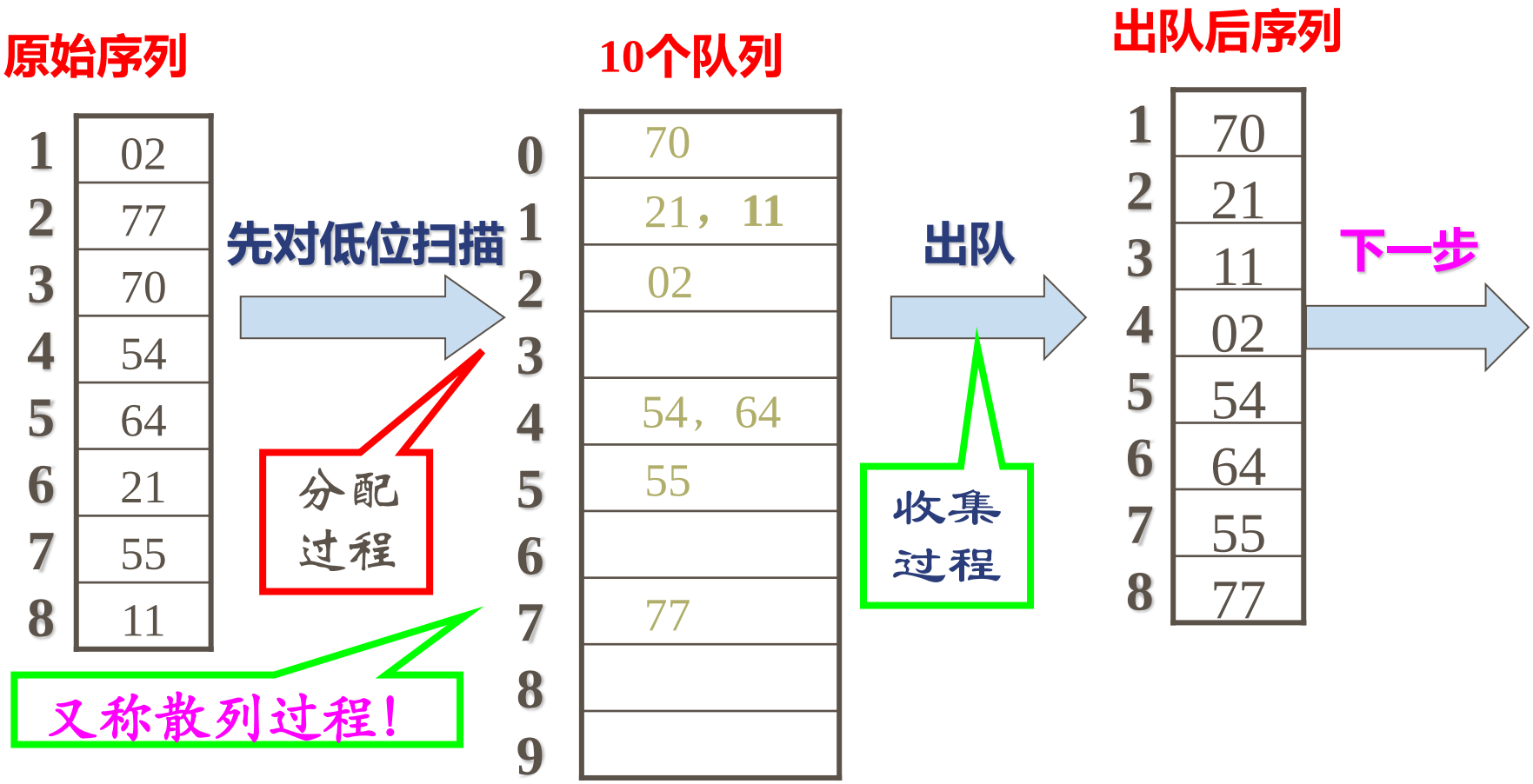
例：T=（02，77，70，54，64，21，55，11），用LSD排序。

分析:

①各关键字可视为2元组；②每位的取值范围是：0-9；即基数*radix* ＝10 。因此，特设置10个队列，并编号为0-9。

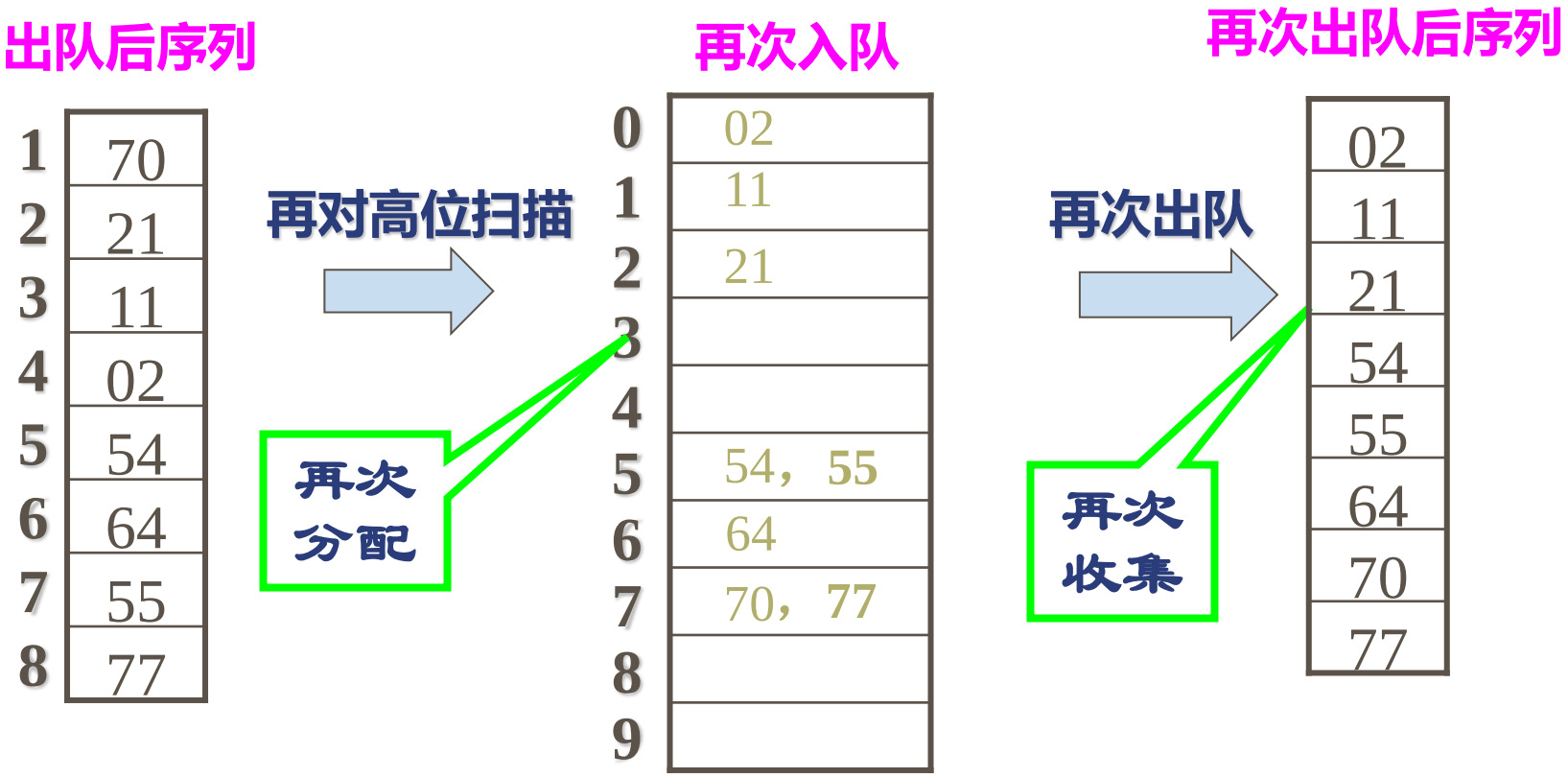
10.6 基数排序

计算机怎样实现LSD算法?



10.6 基数排序

计算机怎样实现LSD算法?



10.6 基数排序

计算机怎样实现LSD算法?

小结：排序时经过了反复的“分配”和“收集”过程。当对关键字所有的位进行扫描排序后，整个序列便从无序变为有序了。

10.6 基数排序

效率分析

时间效率：O(d*(n+rd))

(d:关键字个数，rd:关键字取值范围为rd个值)

空间效率：O(d+rd)

稳定性：　稳定

10.6 基数排序

例如：10000个人按照生日排序

年　月　日　3个关键字

假设取值范围分别是：1934~2022年 1~12月 1~31日

$O(n^2) \approx 10^8$

$O(n\log n) \approx 10^5$

$O(k*(n+m)) \approx (10000+89) +(10000+12) +(10000 +31)$

$\approx 10^4$

10.6 基数排序

讨论： 所用队列是顺序结构，浪费空间，能否改用链式结构？

用链队列来实现基数排序——

链式基数排序

10.6 基数排序

链式基数排序实现思路：

针对 *d* 元组中的每一位分量，把原始链表中的所有记录，按 K_i^j 的取值，让 j = d, d-1, ···, 2, 1，

① 先“分配”到*radix*个链队列中去；

② 然后再按各链队列的顺序，依次把记录从链队列中“收集”起来；

③ 分别用这种“分配”、“收集”的运算逐趟进行排序；

④ 在最后一趟“分配”、“收集” 完成后，所有记录就按其关键码的值从小到大排好序了。

10.6 基数排序

链式基数排序

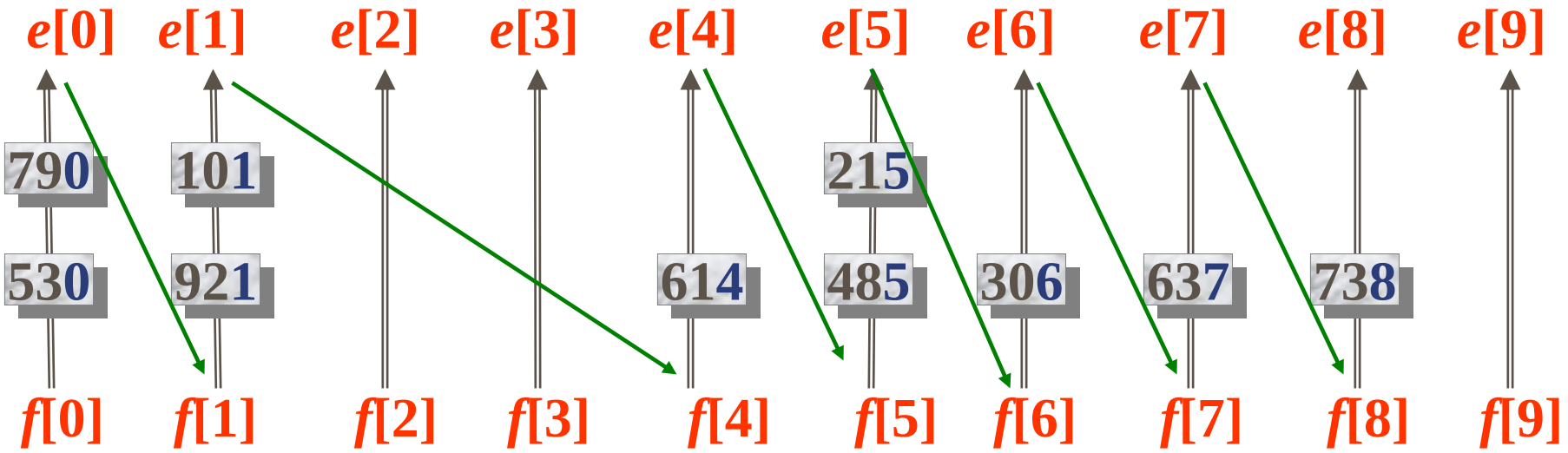
例: 请实现以下关键字序列的链式基数排序:

T=（614，738，921，485，637， 101，215，530，790，306）

10.6 基数排序

原始序列静态链表：

第一趟分配 （从最低位 $i=3$ 开始排序，f[] 是队首指针，e[] 为队尾指针）

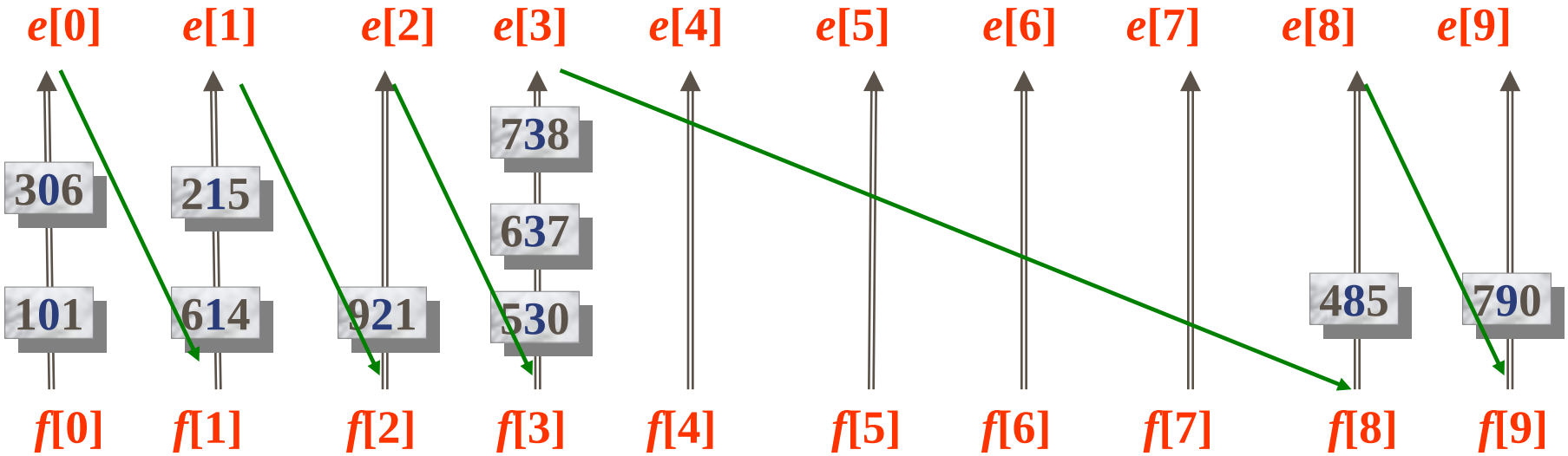


第一趟收集（让队尾指针e[i] 链接到下一非空队首指针f[i+1] 即可）

10.6 基数排序

第一趟收集的结果:

第二趟分配（按次低位 $i=2$）

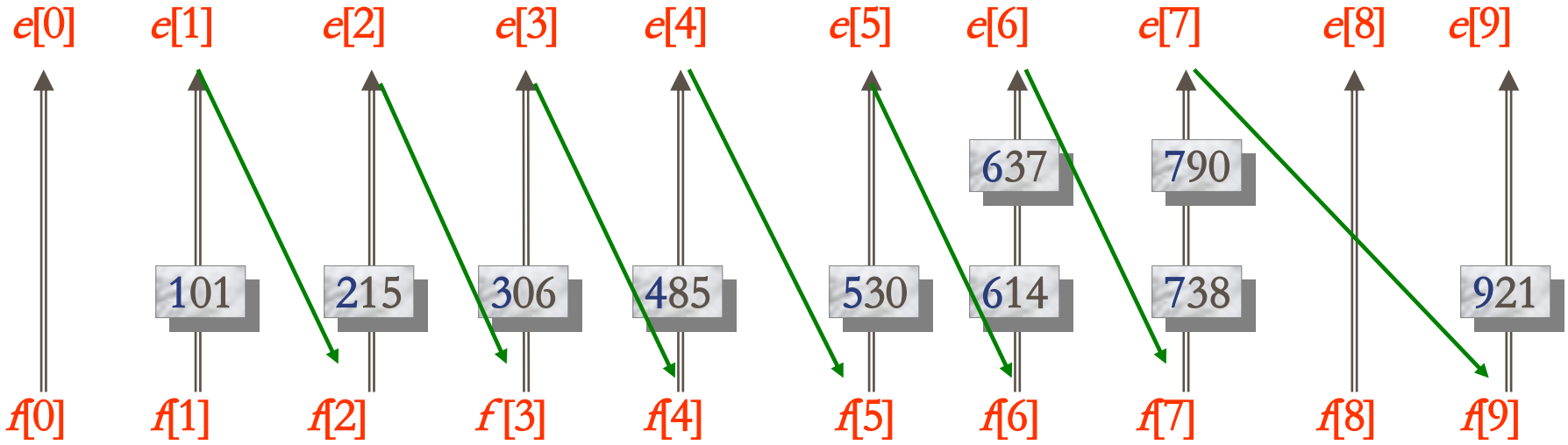


第二趟收集（让队尾指针e[i] 链接到下一非空队首指针f[i+1]）

10.6 基数排序

第二趟收集的结果:

第三趟分配（按最高位 $i=1$）



第三趟收集（让队尾指针e[i] 链接到下一非空队首指针f[i+1]）

排序结束!

10.6 基数排序

链式基数排序

如何编程实现?

1. 用静态链表形式存储排序前后的n个记录（在r[ ]中增开n个指针分量）；这样在记录重排时不必移动数据，只需修改各记录的链接指针即可。

10.6 基数排序

链式基数排序

如何编程实现?

2.在radix个静态队列中，每个队列都要设置两个指针:

int f[radix]指示队头(f[j]初始为空）；

int e[radix] 指向队尾（e[j]可省略初始化,队尾元素随时赋值);

分配到同一队列的关键码要用链接指针链接起来。

（注：以上一共增开了n+2 radix个附加指针分量）

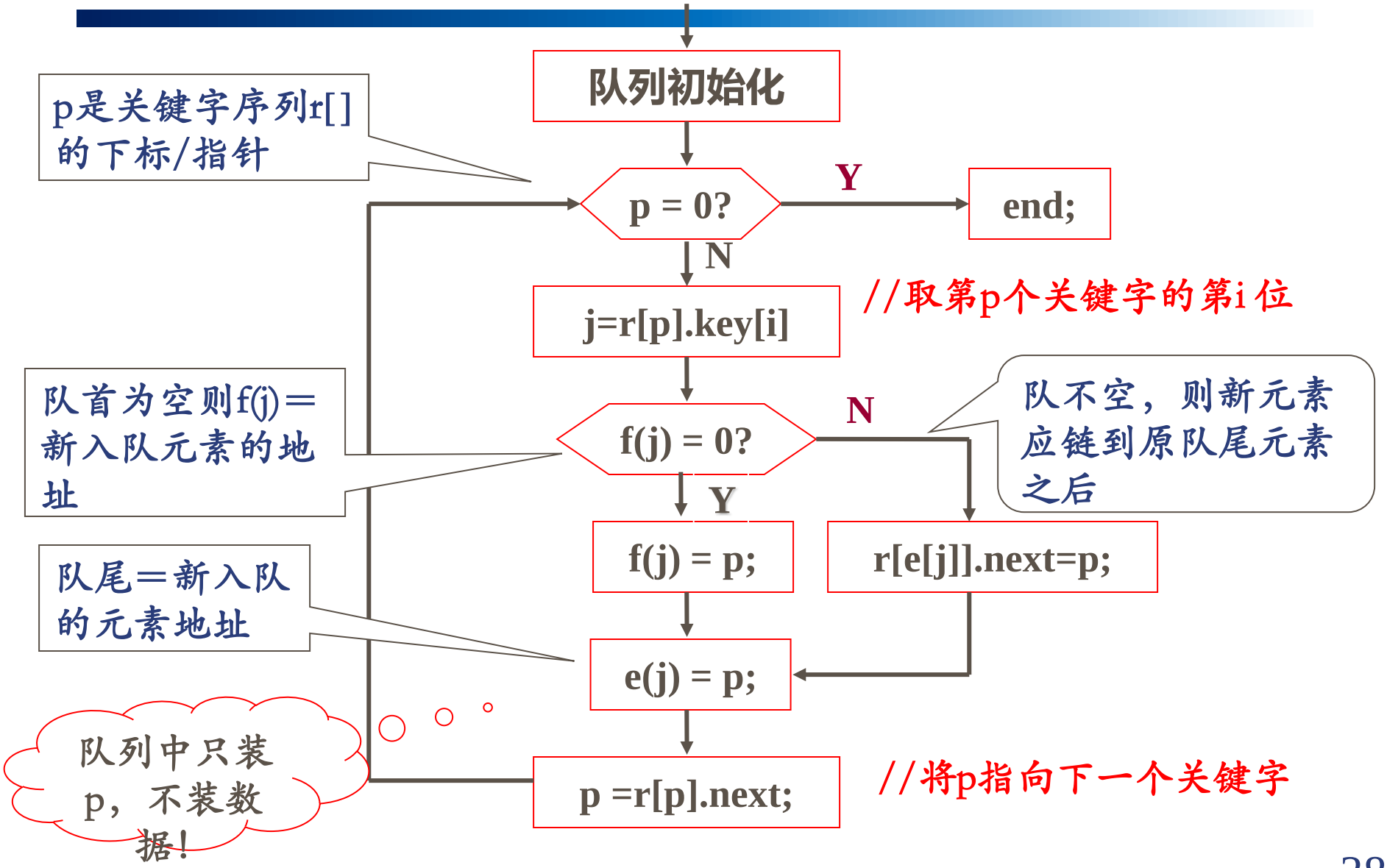
10.6 基数排序

链式基数排序

如何编程实现?

3. 存入r[ ]中的待排记录可分为3部分:

一趟“分配”过程的算法流程



10.6 基数排序

```
void RadixSort(SLList &L){ // L是采用静态链表表示的顺序表
// 基数排序,使得L成为按关键字自小到大有序,L.r[0]为头结点
    for (i = 0; i < L.recnum; ++i)
        L.r[i].next = i + l;
    L.r[recnum].next = 0; // 将L改造为静态链表
    for (i = 0; i L.keynum; ++i)
    {   // 按最低位优先依次对各关键字进行分配和收集
        Distribute(L.r, i, f, e); // 第i趟分配
        Collect(L.r, i, f, e);    // 第 i 趟收集
    }
} // RadixSort
```

10.6 基数排序

```
void Distribute (SLCell &r,int i, ArrType &f,ArrType &e){
    //静态链表L的r域中记录已按(keys[0], ..keys[i- 1])有序。
    //本算法按第i个关键字keys[i]建立 RADIX个子表使同一子表中记录的 keys[i]相同
    // f[0...RADIX-1]和e[0...RADIX-1]分别指向各子表中第一个和最后一个记录
    for(j=0;j<Radix; ++j) f[j] = 0; //各子表初始化为空表
    for (p=r[0].next; p; p=r[p].next) {
        j = ord(r[p].keys[i]); // ord将记录中第i个关键字映射到[0...RADIX-1]
        if(!f[j]) f[j]=p;
        else r[e[j]].next = p;
        e[j]=p; // 将p所指的结点插入第 j个子表中
    }
}
```

10.6 基数排序

```
void Collect (SLCell &.r, in七i, ArrType f, ArrType e) {
//本算法按keys[i]自小至大地将f[O...RADIX-1]所指各子表依次链接成一个链表
//e[O...RADIX- 1]为各子表的尾指针
    for(j=O; !f[j]; j=succ(j)); //找第一个非空子表， succ 为求后继函数
    r[0].next = f[j]; // r[0].next 指向第一个非空子表中第一个结点
    t=e[j];
    while(j<RADIX){
        for (j=succ(j); j<RADIX-1 && !f[j]; j= succ(j));// 找下一个非空子表
        if(f[j]) {r[t].next = f[j]; t=e[j];}//链接两个非空子表
    }
    r[t].next = 0; // t 指向最后一个非空子表中的最后一个结点
}
```

10.6 基数排序

链式基数排序

基数排序算法分析

- 假设有n 个记录，每个记录的关键字有d 位，每个关键字的取值有$radix$个，则每趟分配需要的时间为O(n)，每趟收集需要的时间为O($radix$)，合计每趟总时间为O(n+$radix$)

  全部排序需要重复进行d趟“分配”与“收集”。因此时间复杂度为：O(d(n+$radix$))。

10.6 基数排序

链式基数排序

基数排序算法分析

- 基数排序需要增加$n+2radix$个附加链接指针，空间效率低，

  空间复杂度：O(n+*radix*) .

- 稳定性：稳定。(一直前后有序)。

小结

排序

一、时间性能

1. 按平均的时间性能来分，有三类排序方法

- 时间复杂度为O(nlogn)的方法有
  → 快速排序、堆排序和归并排序，其中以快速排序为最好

- 时间复杂度为O(n^2)的有
  → 直接插入排序、冒泡排序和简单选择排序，其中以直接插入为最好，特别是对那些对关键字近似有序的记录尤为如此

- 时间复杂度为0(n)的排序方法只有:
  → 基数排序

排序

一、时间性能

2. 当待排序记录序列按关键字顺序有序时，直接插入排序和冒泡排序能达到O(n)的时间复杂度。而对于快速排序而言，这是最不好的情况，此时的时间性能退化为$O(n^2)$，因此应该避免这种情况

3. 简单选择排序、堆排序和归并排序的时间性能不随记录序列中关键字的分布而该改变

排序

二、空间性能（指的是排序过程中所需的辅助空间大小）

1. 所有的简单排序方法(包括: 直接插入，冒泡和简单选择)和堆排序的空间复杂度为O(1)
2. 快速排序为O(logn)，为栈所需的辅助空间
3. 归并排序所需辅助空间最多，其空间复杂度为O(n)
4. 链式基数排序需附设队列首尾指针，则空间复杂度为O(n+radix)

排序

三、排序算法稳定性

- 稳定的排序方法是指，对于两个关键字相等的记录，它们在序列中的相对位置，在排序之前和经过排序之后，没有改变。
- 当对多关键字的记录序列进行LSD方法排序时，必须采用稳定的排序方法
- 对于不稳定的排序方法，只要能举出一个实例说明即可
- 快速排序和堆排序是不稳定的排序方法

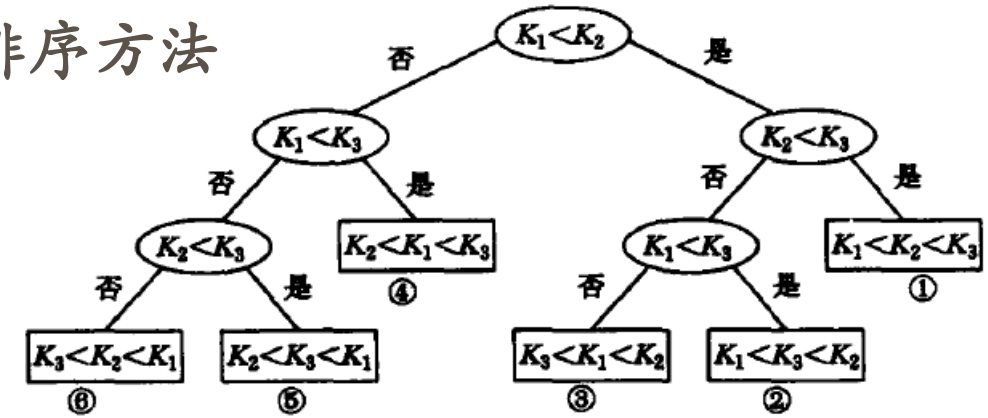
排序

四、关于“排序方法的时间复杂度的下限”

- 本章各种排序方法，除基数排序外，其他方法都是基于“比较关键字”进行排序的排序方法。可以证明，这类排序法可能达到的最快的时间复杂度为O(nlogn)

  (基数排序不是基于“比较关键字”的排序方法，所以它不受这个限制)

- 可以用一棵判定树来描述这类基于“比较关键字”进行排序的排序方法



排序

排序方法	时间复杂度	时间复杂度		稳定性	附加存储	
		最好	最差		最好	最差
直接插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	√	$O(1)$	
折半插入排序	$O(n\log_2 n)$	$n\log_2 n$		√	$O(1)$	
2-路插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	√	$O(1)$	
表插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	√	$O(n)$	
希尔排序	$O(n^{1.25})$	$O(n)$	$O(1.6n^{1.25})$	X	$O(1)$	
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	√	$O(1)$	
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	X	$O(\log_2 n)$	$O(n^2)$
简单选择排序	$O(n^2)$	n^2		X	$O(1)$	
树形选择排序	$O(n\log_2 n)$	$O(n\log_2 n)$		√	$O(n)$	
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$		X	$O(1)$	
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$		√	$O(n)$	
基数排序	$O(d(n+radix))$	$O(d\ (n+radix))$		√	$O(n+radix)$	