数据结构

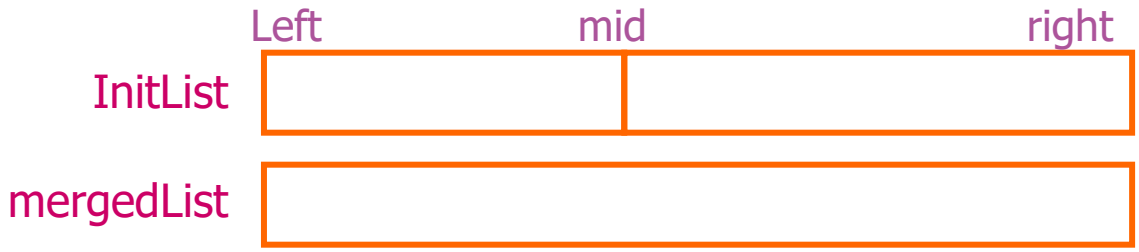
深圳技术大学
大数据与互联网学院

第十章 内部排序

10.5 归并排序

一. 归并排序

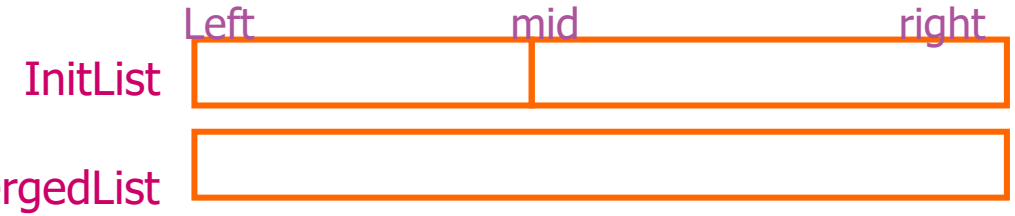
- 归并算法思路：
  - 归并是将两个或两个以上的有序表合并成一个新的有序表。
- 两路归并：
  - 将初始序列分成前后两组，通过比较，归并到目标序列中



10.5 归并排序

一. 归并排序

- 两路归并的算法实现：




```
typedef int SortData;
void merge ( SortData InitList[ ], SortData mergedList[ ],
             int left, int mid,  int right ) {
    int i = left,  j = mid+1,  k = left;
    while ( i <= mid && j <= right )      //两两比较将较小的并入
        if ( InitList[i] <= InitList[j] ) {
            mergedList [k] = InitList[i]; i++;  k++;
        } else  { mergedList [k] = InitList[j]; j++;  k++; }
    while (i <= mid)
    { mergedList[k] = InitList[i];  i++;  k++; }//将mid前剩余的并入
    while (j <= right)
    { mergedList[k] = InitList[j];  j++;  k++; }//将mid后剩余的并入
}
```

10.5 归并排序

一. 归并排序

- 两路归并性能分析：
  - 假设待归并的两个有序表长度分别为m和n，则两路归并后，新的有序表长度为m+n
  - 两路归并操作至多只需要m+n次移位和m+n次比较
  - 因此两路归并的时间复杂度为O(m+n)
  - 缺点，需要额外空间m+n

10.5 归并排序

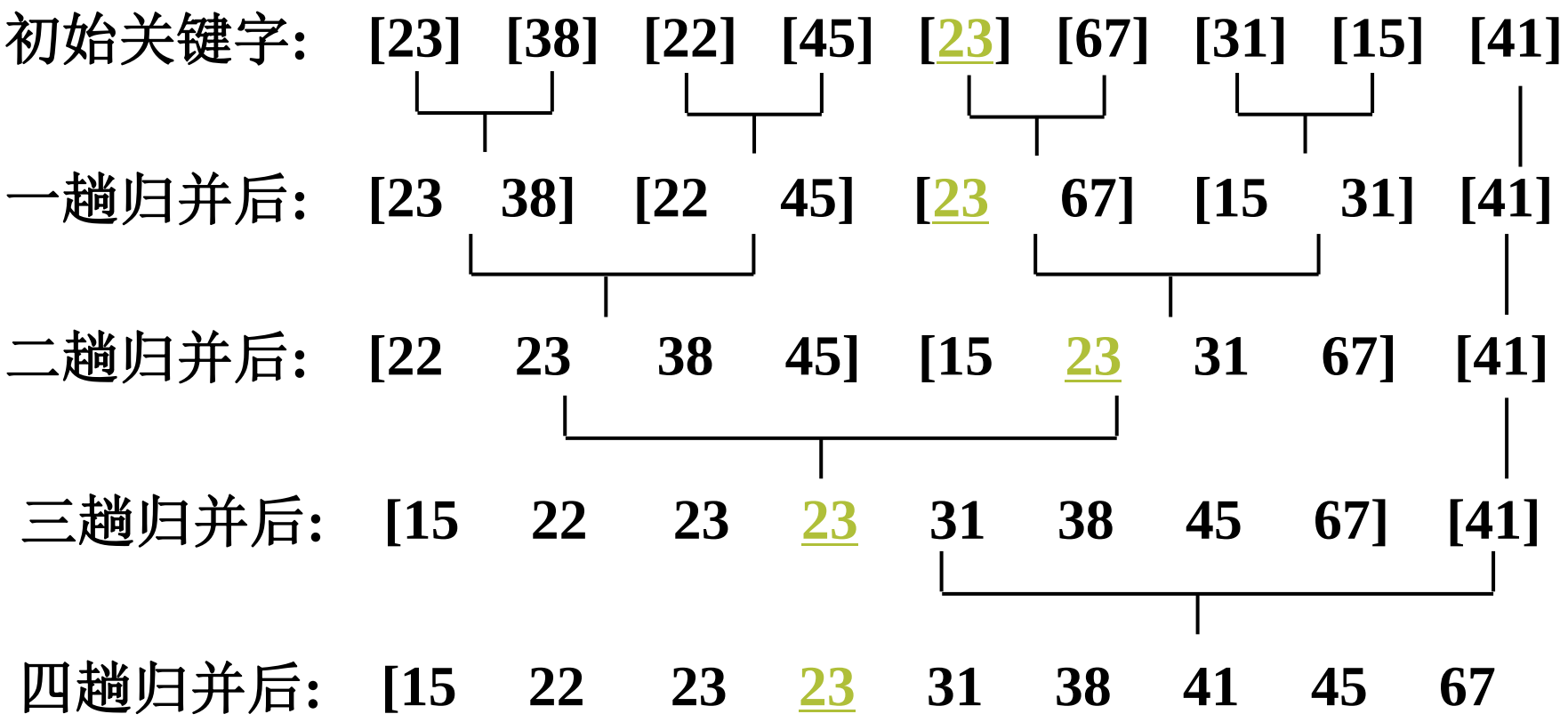
二. 2路一归并排序

- 算法思路：
  - 将n个记录看成是n个有序序列
  - 将前后相邻的两个有序序列归并为一个有序序列(两路归并)
  - 重复做两路归并操作，直到只有一个有序序列为止

10.5 归并排序

二. 2路一归并排序

- 举例：



10.5 归并排序

二. 2路一归并排序

- 程序实现：

```
void MSort(RedType SR[], RedType TR1[], int s, int t) {
   // 将SR[s..t]归并排序为TR1[s..t]。
   int m;
   RedType TR2[20];
   if (s==t) TR1[t] = SR[s];
   else {
      m=(s+t)/2;                  // 将SR[s..t]平分为SR[s..m]和SR[m+1..t]
      MSort(SR,TR2,s,m);          // 递归地将SR[s..m]归并为有序的TR2[s..m]
      MSort(SR,TR2,m+1,t);        // 将SR[m+1..t]归并为有序的TR2[m+1..t]
      Merge(TR2,TR1,s,m,t); // 将TR2[s..m]和TR2[m+1..t]归并到
TR1[s..t]
   }
} // MSort

void MergeSort(SqList &L) {   // 算法10.14
  // 对顺序表L作归并排序。
  MSort(L.r, L.r, 1, L.length);
} // MergeSort
```

10.5 归并排序

二. 2路一归并排序

- 性能分析：
  - 如果待排序的记录为n个，则需要做$\log_2 n$趟两路归并排序
  - 每趟两路归并排序的时间复杂度为O(n)
  - 因此2路一归并排序的时间复杂度为O($n\log_2 n$)
  - 归并排序是一种稳定的排序方法

练习

一. 设关键字序列为（314，617，253，335，19，237，464，121，46，231，176，344）的一组记录，请给出两路一归并排序的每一躺结果

10.6 基数排序

一. 多关键字的排序

- 有时候排序不仅只有1个关键字，可能包含多个

例：对52张扑克牌按以下次序排序：

♣2<♣3<……<♣A<♦2<♦3<……<♦A<

♥2<♥3<……<♥A<♠2<♠3<……<♠A

两个关键字：花色（♣<♦<♥<♠）

面值（2<3<…<A）

并且“花色”地位高于“面值”

10.6 基数排序

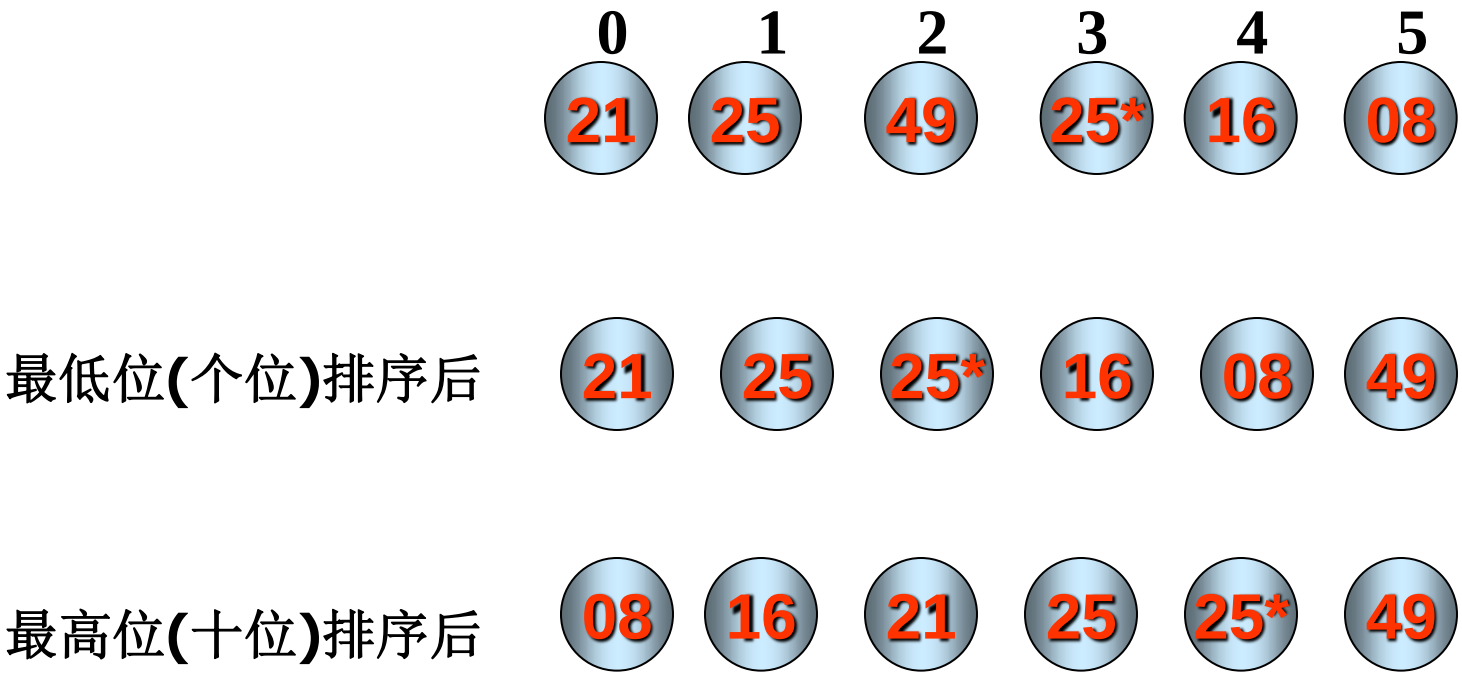
一. 多关键字的排序

- 最低位优先法LSD
  - 从最低位关键字kd起进行排序，
  - 然后再对高一位的关键字排序，……
  - 依次重复，直至对最高位关键字k1排序后，便成为一个有序序列

10.6 基数排序

一. 多关键字的排序

- 最低位优先法LSD举例



10.6 基数排序

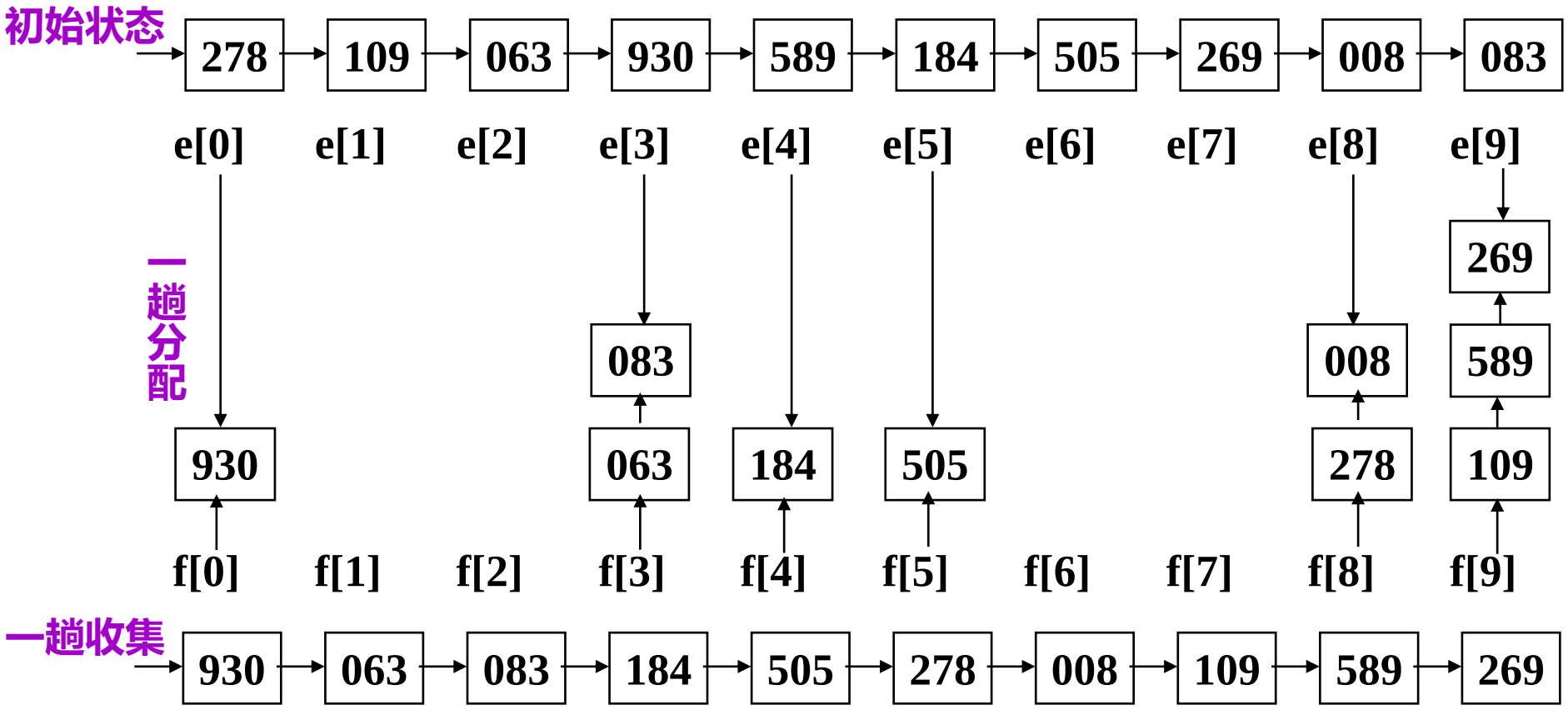
二. 链式基数排序

- 算法思路
  - 基数排序：借助“分配”和“收集”对单逻辑关键字进行排序的一种方法
  - 链式基数排序方法：用链表作存储结构的基数排序
  - 设置10个队列，f[i]和e[i]分别为第i个队列的头指针和尾指针
  - 第i趟分配：根据第i位关键字的值，改变记录的指针，将链表中记录按次序分配至10个链队列中（采用队尾插入法）；每个队列中记录关键字的第i位关键字相同
  - 第i趟收集：改变所有非空队列的队尾记录的指针域，令其指向下一个非空队列的队头记录，重新将10个队列链成一个链表

10.6 基数排序

二. 链式基数排序

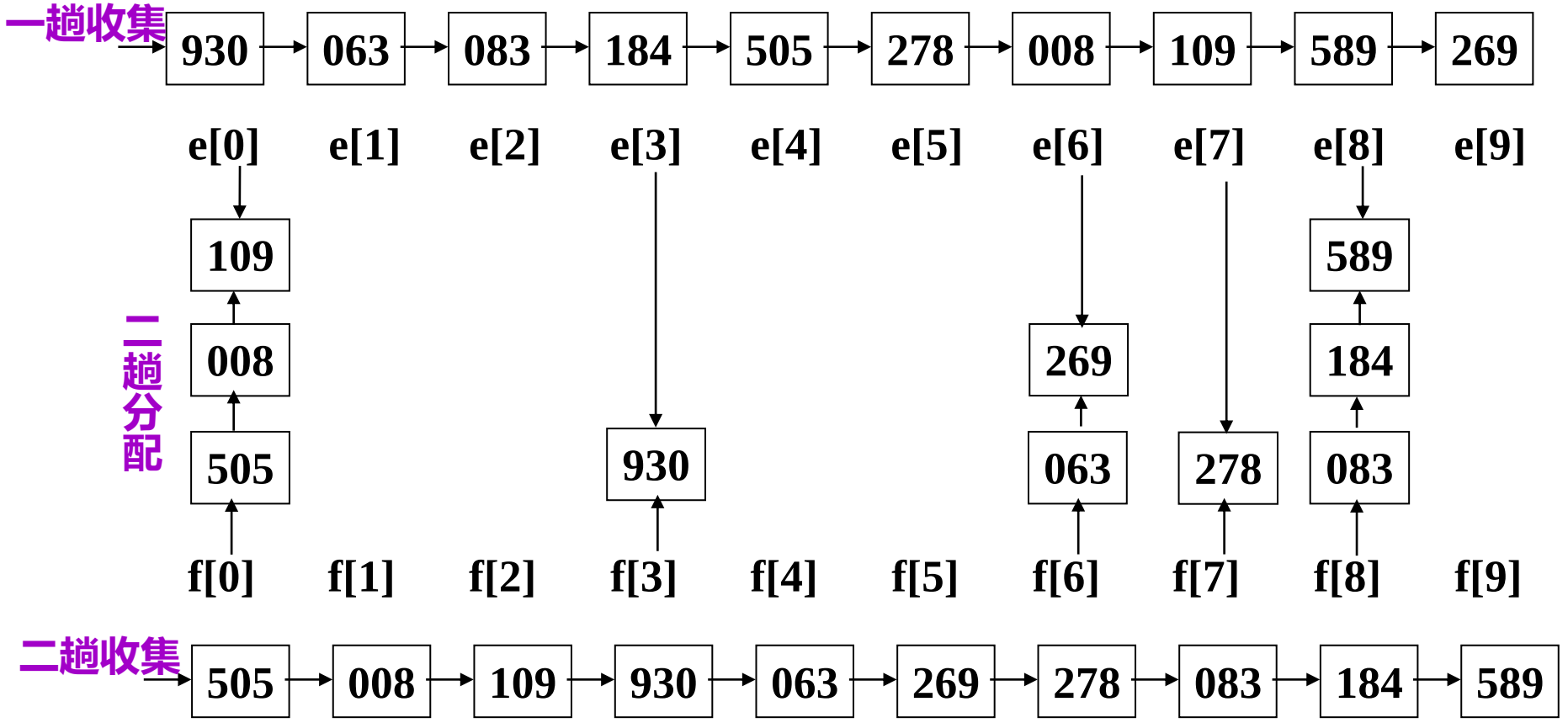
- 举例，假设关键字都是3位整数



10.6 基数排序

二. 链式基数排序

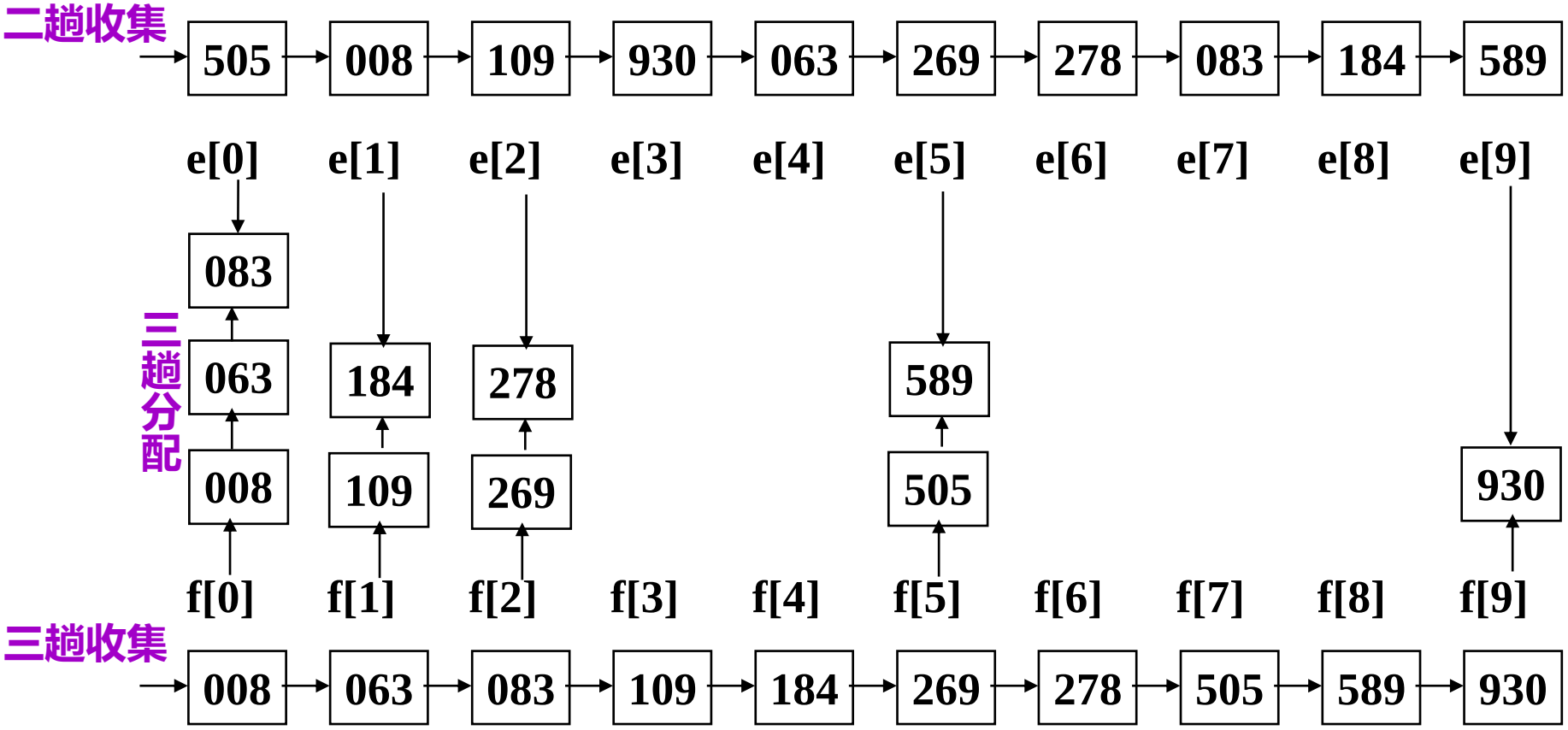
- 举例



10.6 基数排序

二. 链式基数排序

- 举例



10.6 基数排序

二. 链式基数排序

- 程序实现

```
void RadixSort(SLList &L) {   // 算法10.17
    // L是采用静态链表表示的顺序表。
    // 对L作基数排序，使得L成为按关键字自小到大的有序静态链表，L.r[0]为头结点
    int i;
    ArrType f, e;
    for (i=1; i<L.recnum; ++i) L.r[i-1].next = i;
    L.r[L.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, i, f, e);     // 第i趟分配
        Collect(L, i, f, e);        // 第i趟收集
        print_SLList2(L, i);
    }
} // RadixSort
```

10.6 基数排序

二. 链式基数排序

- 程序实现

```
void Distribute(SLList &L, 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。
  int j, p;
  for (j=0; j<RADIX; ++j) f[j] = 0;      // 各子表初始化为空表
  for (p=L.r[0].next;  p;  p=L.r[p].next) {
    j = L.r[p].keys[i]-'0';  // 将记录中第i个关键字映射到
[0..RADIX-1],
    if (!f[j]) f[j] = p;
    else L.r[e[j]].next = p;
    e[j] = p;                          // 将p所指的结点插入第j个子表中
  }
} // Distribute
```

10.6 基数排序

二. 链式基数排序

- 程序实现

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

10.6 基数排序

二. 链式基数排序

- 性能分析
  - 若每个关键字有 d 位,关键字的基数为radix
  - 需要重复执行d 趟“分配”与“收集”
  - 每趟对 n 个对象进行“分配”，对radix个队列进行“收集”
  - 总时间复杂度为O(d(n+radix))。

10.6 基数排序

二. 链式基数排序

- 性能分析
  - 若基数radix相同，对于对象个数较多而关键字位数较少的情况，使用链式基数排序较好。
  - 基数排序需要增加n+2radix个附加链接指针。
  - 基数排序是稳定的排序方法。

练习

一. 已知数列为125、45、388、272、165、39、 428、64，要对数列进行从小到大的排序，

- 若采用堆排序，要求写出每次调整形成的最大堆
- 若2路归并一路排序，要求写出每趟排序结果
- 若采用链式基数排序，要求写出每趟排序结果

本章总结

- 排序的概念：内部排序、外部排序、稳定排序、KCN和RMN
- 直接插入排序、折半插入排序
  - 时间复杂度为$O(n^2)$，是一种稳定排序方法
- 希尔排序：划分子序列进行插入排序，通过增量gap进行子序列划分
  - 时间复杂度约为$O(n \times (\log_2 n)^2)$，是一种不稳定的排序方法
- 起泡排序：依次比较相邻两个记录，每趟排序找出最大对象放在末尾
  - 时间复杂度为$O(n^2)$，是一种稳定排序方法
- 快速排序：选定枢轴记录来回扫描，将小于枢轴的数据放在枢轴左边，大于枢轴的数据放在右边，然后对左右子序列递归排序
  - 时间复杂度为$O(n\log_2 n)$，是一种不稳定排序方法
- 简单选择排序：每一趟选出i…n的最小记录，交换到前面
  - 时间复杂度为$O(n^2)$，是一种不稳定排序方法
- 堆排序：将初始序列建成最大堆，筛选根结点后，重复调整最大堆
  - 时间复杂度为$O(n\log_2 n)$，是一种不稳定排序方法
- 归并排序：两路归并排序、2路归并一路排序
- 基数排序：最低位优先法LSD、链式基数排序