10.4选择排序

DS/SS

简单选择排序

算法思想:

简单选择排序(Simple Selection Sort)的基本操作是通过n-i次关键字的比较，从n-i+1个元素中选出关键字最小的元素，并和第i(1<=i<=n)个元素交换之。

整个排序过程共进行n-1趟(i从1变化至n-1)选择。

```
Void SelectSort(SqList  &L){          //完成一次完整的简单选择排序
   for (i=1; i<L.length; ++i)  {      //选择第i小的元素，并交换到位
     j = SelectMinKey(L,i);           //在L.r[i .. L.length]中选择key最小的元素
     if (i != j) L.r[i] 与L.r[j]交换;  //与第i个元素交换
   }  } //SelectSort
```

简单选择排序

DS/SS


0 1 2 3 4 5 6 7 8
初始关键字 49 38 65 97 76 13 27 49*
i＝1 13 38 65 97 76 49 27 49*
i＝2 13 27 65 97 76 49 38 49*
i＝3 13 27 38 97 76 49 65 49*
i＝4 13 27 38 49 76 97 65 49*
i＝5 13 27 38 49 49* 97 65 76
i＝6 13 27 38 49 49* 65 97 76
i＝7 13 27 38 49 49* 65 76 97

10.4选择排序（续）

DS/SS

性能分析:

时间复杂度$T(n)=O(n^2)$

- 正序： 比较次数:$n(n-1)/2$;
  元素不移动
- 逆序： 比较次数:$n(n-1)/2$;
  移动次数:$3(n-1)$

空间复杂度$S(n)=O(1)$

稳定性:

简单选择排序方法是不稳定的排序方法

适用情况:

*元素初始序列基本有序;

*元素个数较少

10.4选择排序（续）

DS/SS

堆排序

- 堆(Heap)：n个元素的序列$\{k_1, k_2, \ldots, k_n\}$,当且仅当满足以下关系时，称之为堆。

  $k_i <= k_{2i}$且$k_i <= k_{2i+1}$ 或 $k_i >= k_{2i}$且$k_i >= k_{2i+1}$.

  满足前一约束的堆称之为小顶堆，满足后一约束的堆称之为大顶堆。

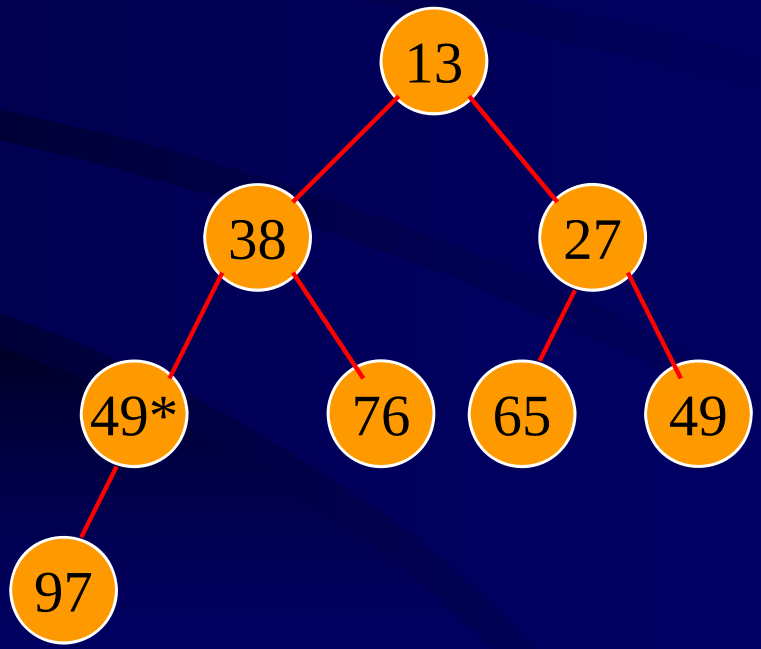
- 小顶堆的第一个元素为序列中最小者；大顶堆的第一个元素为序列中最大者。
- 堆可以用完全二叉树来描述(堆的本质仍然是一个序列)。

10.4选择排序（续）

小顶堆：{13, 38, 27, 49*, 76, 65, 49, 97}

顺序存储的堆

0	1	2	3	4	5	6	7	8
	13	38	27	49*	76	65	49	97



10.4选择排序（续）

DS/SS

堆排序

算法思想：

堆排序(Heap Sort)的思路是：下一趟选择排序时尽可能利用上一趟排序过程中已有的比较结果信息，从而提高排序效率。

堆排序完整过程：首先对初始序列建堆；然后，在输出堆顶的最小值之后，使得剩余n-1个元素的序列重新又建成一个堆，则得到n个元素中的次小值。如此反复进行，通过n-1趟，便能得到一个有序序列。

堆排序实现的两个问题：

*如何由初始序列建成一个堆；

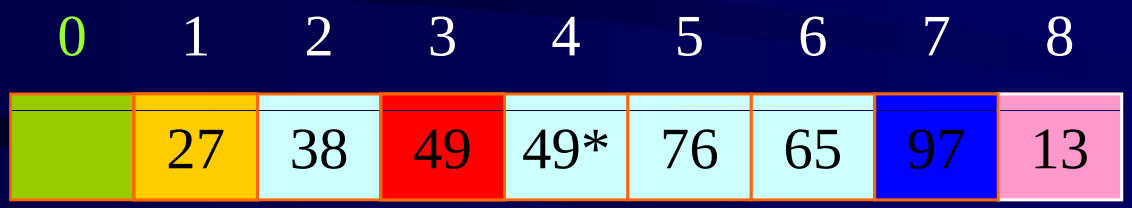
*如何在输出堆顶元素之后，调整剩余元素成为一个新的堆

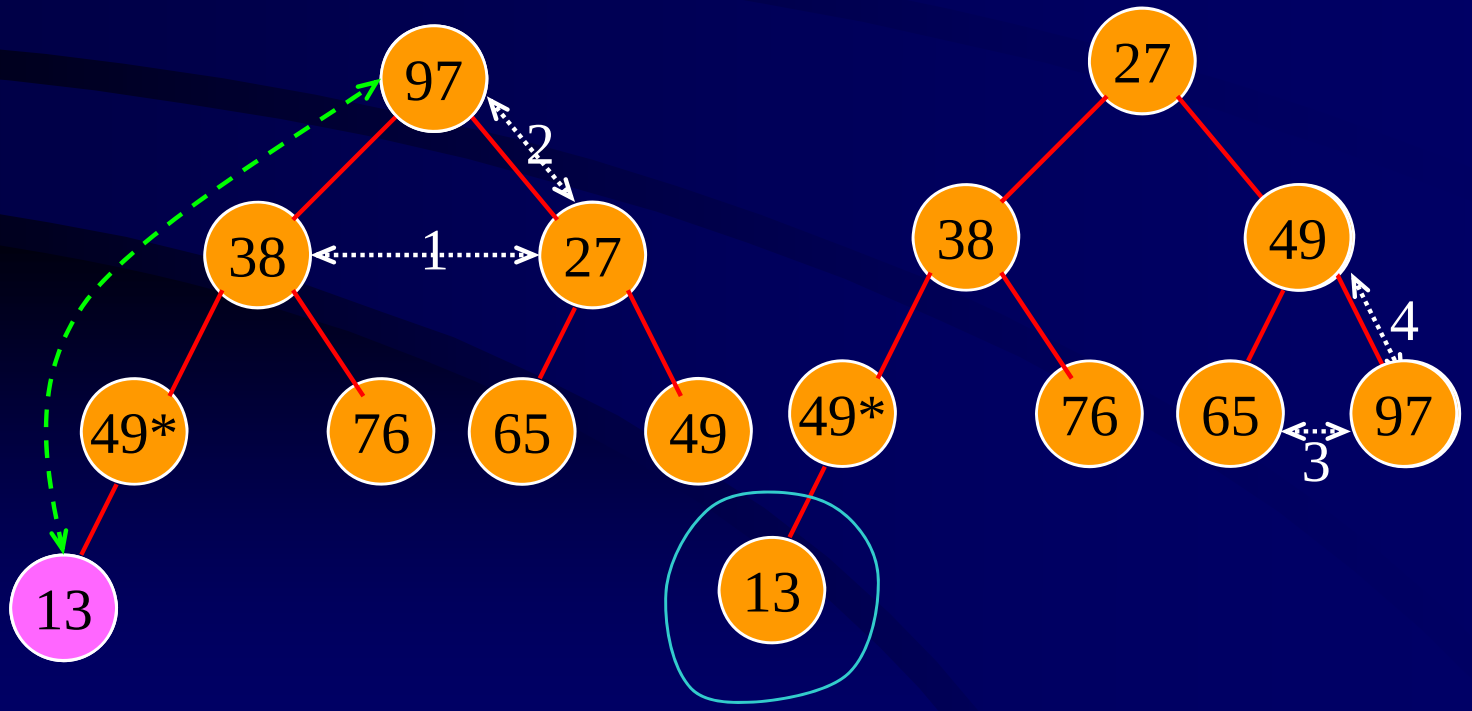
这两个问题的解决都归结到一个实现：筛选(自堆顶至叶子的调整过程)

筛选(自堆顶至叶子的调整过程):

输出堆顶元素后,以堆中最后一个元素替代。这时破坏了整个树描述的堆,但其左、右子树仍然是堆，则仅需自上而下进行调整。

首先以堆顶元素和其左、右子树根结点值小者交换，若破坏了子树的堆，则进行类似的调整，直至叶子结点。此时堆顶为n-1个元素的最小者。这就是一次调整过程。





10.4选择排序（续）

DS/SS

堆排序

建堆：

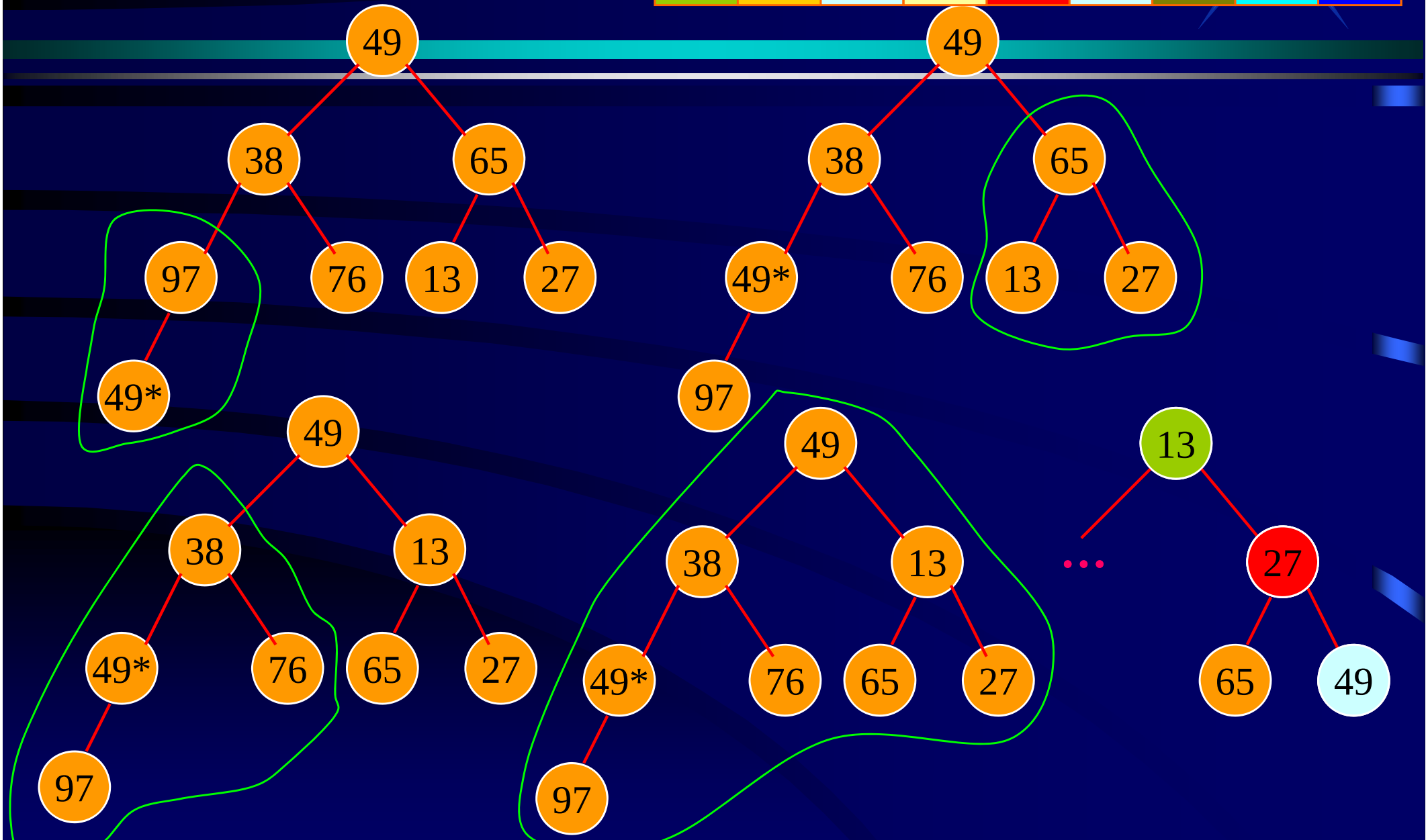
从一个无序序列建堆的过程就是一个反复“筛选”的过程。

将此序列用完全二叉树描述，则最后一个非终端结点是“第(n/2)取下整”个元素，由此，筛选只需从“第(n/2)取下整”个元素开始。

共进行“第(n/2)取下整”次筛选就完成初始建堆过程。

{49, 38, 65, 97, 76, 13, 27, 49*}

0	1	2	3	4	5	6	7	8
	13	38	27	49*	76	65	49	97



```
typedef SqList HeapType; //堆采用顺序表存储表示
void HeapAdjust(HeapType  &H ,int s, int  m){    //一次调整
    rc = H.r[s];
    for (j =2*s; j<=m; j *=2) {          //沿key较大的孩子结点向下筛选
        if (j<m &&LT(H.r[j].key, H.r[j+1].key))  ++j;    //j为key较大的元素的下标
        if ( !LT(rc.key, H.r[j].key)) break;              //rc应插入在位置s上
        H.r[s] = H.r[j];          s = j;
    }
    H.r[s] = rc;                  //插入      } // HeapAdjust
```

已知H.r[s..m]中元素的关键字除H.r[s].key之外均满足堆的定义，本函数调整H.r[s]的关键字，使H.r[s..m]成为一个大顶堆

```
Void HeapSort(HeapType  &H){  //完成一次完整的堆排序
   for (i=H.length/2; i>0; --i)    HeapAdjust (H, i, H.length);
   for (i=H.length; i>0; --i)  {
      H.r[1] 与H.r[i]交换;       //将堆顶元素和H.r[1..i]中最后一个元素相互交换
      HeapAdjust (H, 1, i-1);    //将H.r[1..i-1]重新调整成大顶堆} } //HeapSort
```

10.4选择排序（续）

DS/SS

性能分析:

时间复杂度T(n)= O($nlog_2n$) (最坏情况下)

*建立n个元素、深度为h的堆，关键字比较总次数不超过4n;

*调整新堆时调用HeapAdjust共n-1次，总共进行的比较次数不超过$2nlog_2n$次。

空间复杂度S(n)=O(1) (元素交换时用)

稳定性:

堆排序方法是不稳定的排序方法

适用情况:

*元素个数较多 (时间主要耗费在初始建堆和调整新堆时进行的反复筛选上)

10.6基数排序

- **基数排序**(Radix Sorting)：借助多关键字排序的思想对单逻辑关键字进行排序的方法。
- **多关键字排序**：n个元素的序列 $\{R_1,R_2,\ldots, R_n\}$,每个元素R_i有d个关键字$(K^0_i, K^1_i,\ldots, K^{d-1}_i)$,则称序列对关键字$(K^0, K^1,\ldots, K^{d-1})$有序：对于任意两个元素$R_i$和$R_j$都满足下列有序关系：

  $(K^0_i, K^1_i,\ldots, K^{d-1}_i) < (K^0_j, K^1_j,\ldots, K^{d-1}_j)$。其中$K^0$称为最主位关键字，$K^{d-1}$称为最次位关键字。
- **最高位优先**(Most Significant Digit first)：先对最主位关键字K^0进行排序，将序列分成若干子序列，每个子序列中的元素都具有相同的K^0值，然后分别就每个子序列对关键字K^1进行排序，按K^1值不同再分成若干更小的子序列，依次重复，直至对K^{d-2}进行排序之后得到的每一子序列中的元素都具有相同的关键字$(K^0, K^1,\ldots, K^{d-2})$，而后分别每个子序列对$K^{d-1}$进行排序，最后将所有子序列依次联接在一起称为一个有序序列。
- **最低位优先**(Least Significant Digit first)：从最次位关键字K^{d-1}起进行排序。然后再对高一位的关键字K^{d-2}进行排序，依次重复，直至对K^0进行排序后便成为一个有序序列。

10.6基数排序（续）

DS/SS

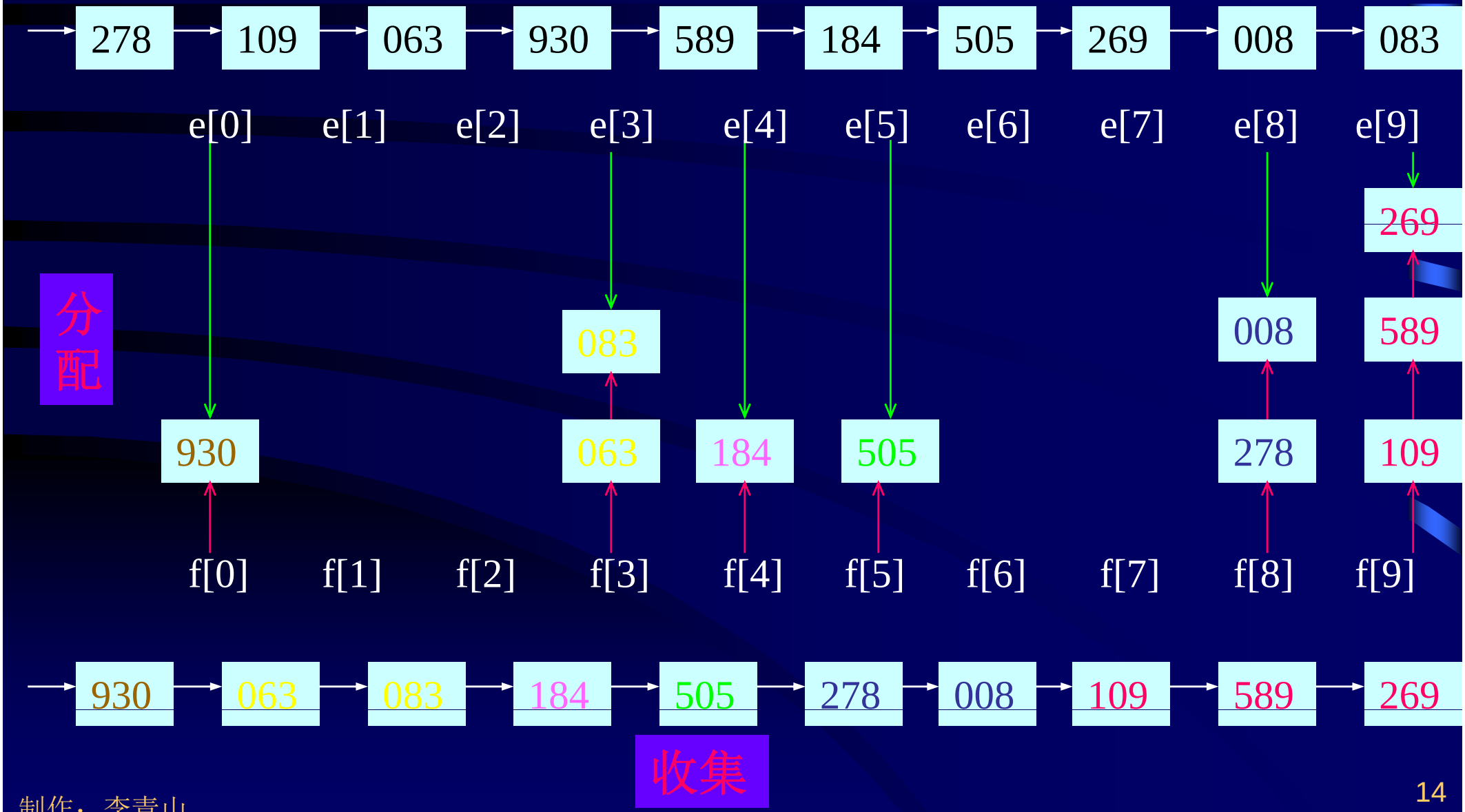
算法思想：

借助“分配”和“收集”两种操作对单逻辑关键字进行排序。

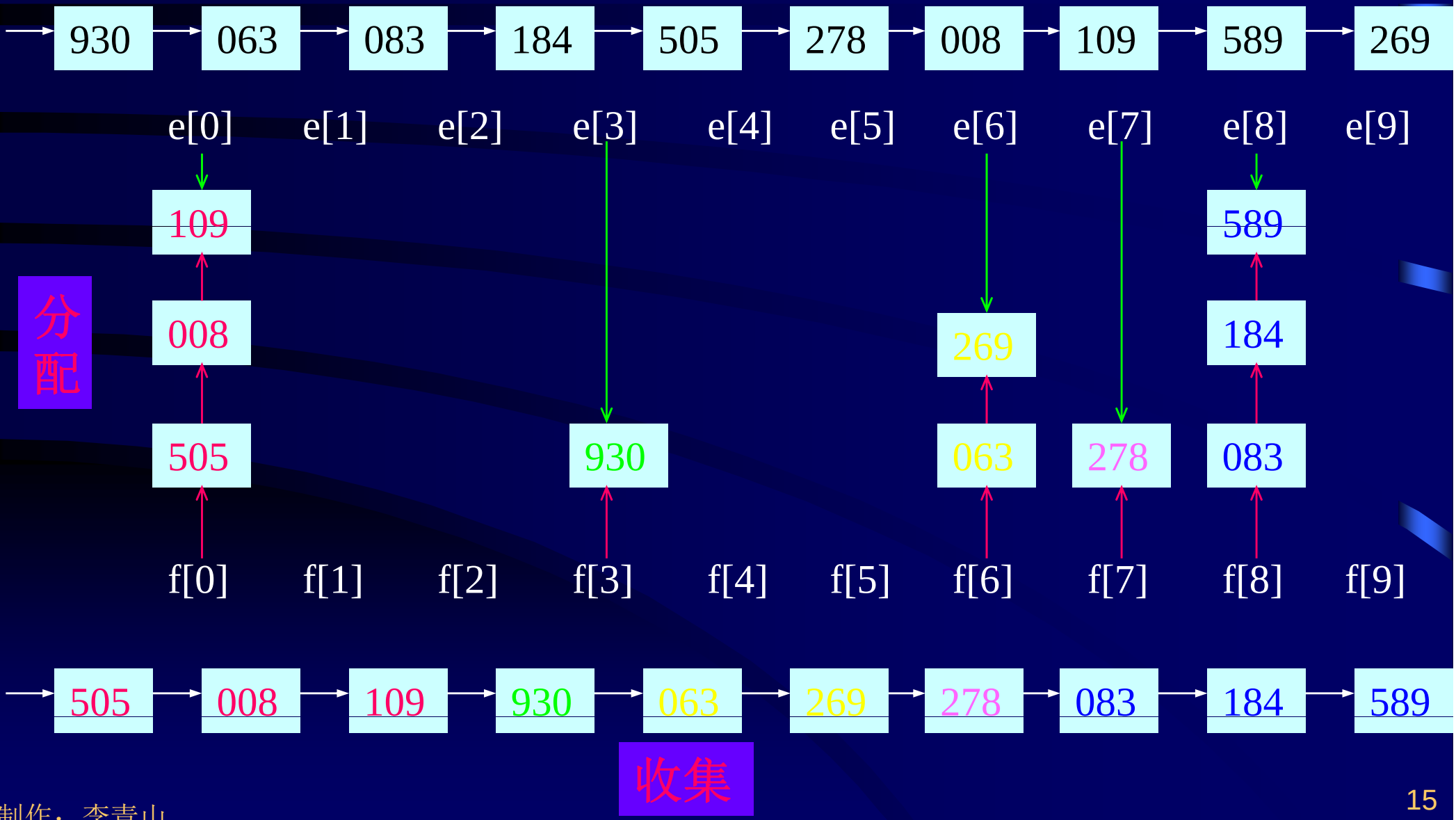
有的单逻辑关键字可以看成由若干个关键字复合而成。比如数值，设0<=K<=999，则K可以看作三个关键字(K^0, K^1, K^2)组成(d=3)，其中是K^0是百位数, K^1是十位数, K^2是个位数，且0<= K^i<=9(RADIX=10)。由于每个关键字范围都相同，则按LSD进行排序更方便，即从最低位关键字字起，按关键字的不同值将序列中元素“分配”到RADIX个队列中后再“收集”之，如此重复d次，即可实现基数排序。

具体而言，第一趟分配对最低位关键字(个位数)进行，将初始序列中元素分配到RADIX个队列中去，每个队列中的元素关键字的个位数相等。f[i]和e[i]分别为第i个队列的头指针和尾指针；第一趟收集是改变所有非空队列的队尾元素的指针域，令其指向下一个非空队列的队头元素，重新将RADIX个队列中的元素链接成一个链表。第二趟分配，第二趟收集及第三趟分配，第三趟收集分别对十位数和百位数进行的，过程相同。

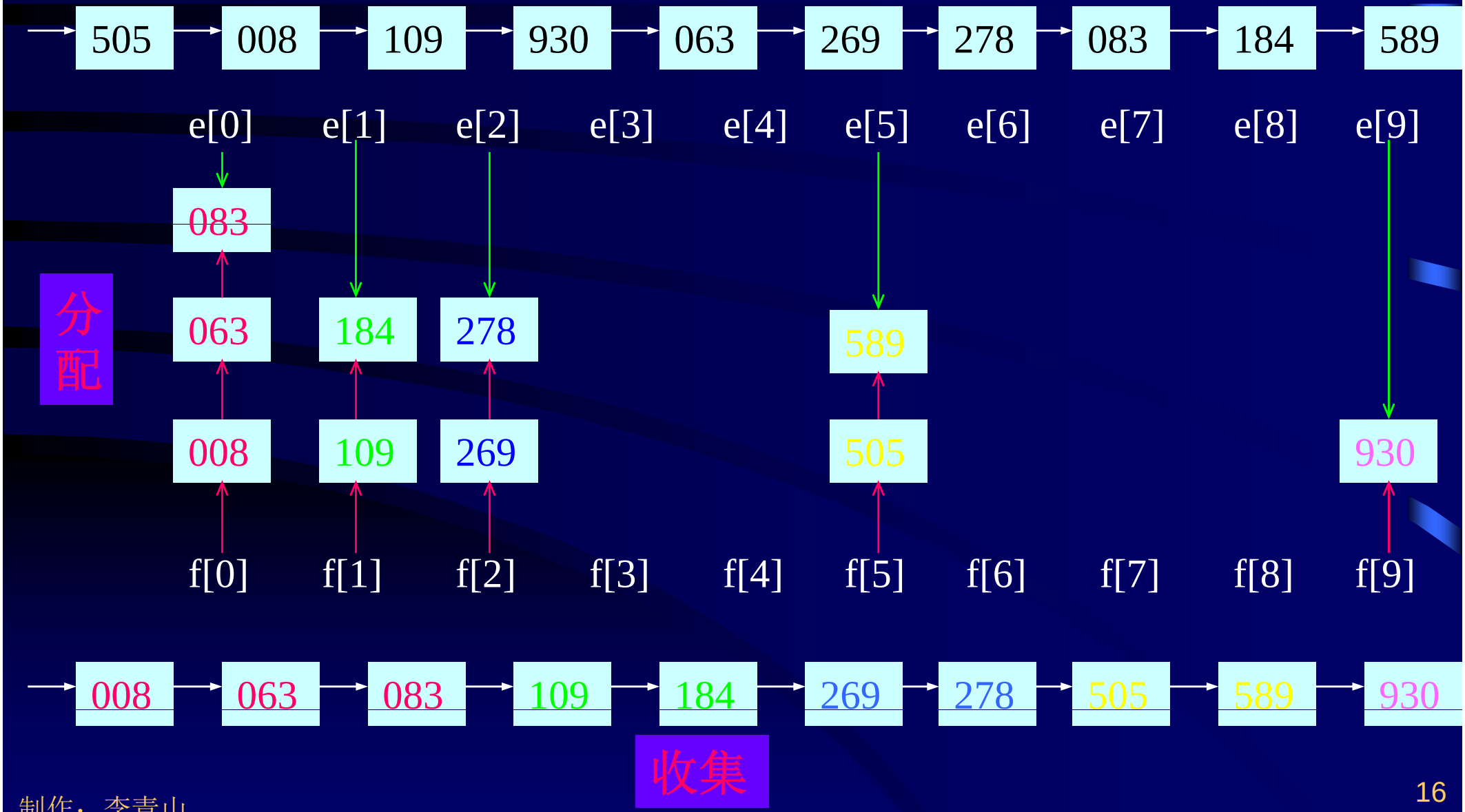
10.6基数排序（续）


278
109
063
930
589
184
505
269
008
083
e[0]
e[1]
e[2]
e[3]
e[4]
e[5]
e[6]
e[7]
e[8]
e[9]
分配
269
083
008
589
930
063
184
505
278
109
f[0]
f[1]
f[2]
f[3]
f[4]
f[5]
f[6]
f[7]
f[8]
f[9]
930
063
083
184
505
278
008
109
589
269
收集

10.6基数排序（续）


930
063
083
184
505
278
008
109
589
269
e[0]
e[1]
e[2]
e[3]
e[4]
e[5]
e[6]
e[7]
e[8]
e[9]
109
008
505
930
269
063
278
589
184
083
分配
f[0]
f[1]
f[2]
f[3]
f[4]
f[5]
f[6]
f[7]
f[8]
f[9]
505
008
109
930
063
269
278
083
184
589
收集

10.6基数排序（续）


505
008
109
930
063
269
278
083
184
589
e[0]
e[1]
e[2]
e[3]
e[4]
e[5]
e[6]
e[7]
e[8]
e[9]
分配
083
063
184
278
589
008
109
269
505
930
f[0]
f[1]
f[2]
f[3]
f[4]
f[5]
f[6]
f[7]
f[8]
f[9]
008
063
083
109
184
269
278
505
589
930
收集

10.6基数排序（续）

DS/SS

性能分析:

时间复杂度T(n)= O(d(n+rd))　(rd为每个关键字基数)

每一趟：分配时间复杂度O(n)

每一趟：收集时间复杂度O(rd)

共d趟：　d(n+rd)

空间复杂度S(n)=O(rd) (2rd个队列指针)

稳定性：

基数排序方法是稳定的排序方法

适用情况：

*元素个数n很大且关键字较小

10.7内部排序方法比较

DS/SS

排序方法	时间性能	空间性能	稳定性	适用情况
直接插入排序	$O(n^2)$	$O(1)$	稳定	n小;初始序列基本有序
希尔排序	$O(n^{1.3})$	$O(1)$	不稳定	
冒泡排序	$O(n^2)$	$O(1)$	稳定	n小;初始序列基本有序
快速排序	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定	初始序列无序
简单选择排序	$O(n^2)$	$O(1)$	不稳定	n小
堆排序	$O(n\log_2 n)$	$O(1)$	不稳定	n大;只求前几位
2-路归并排序	$O(n\log_2 n)$	$O(n)$	稳定	n很大
链式基数排序	$O(d(n+rd))$	$O(rd)$	稳定	n大;关键字值小