第9章　排序

目录

9．1 基本概念

9.1.1 排序介绍

排序（Sorting）是数据处理中一种很重要的运算，同时也是很常用的运算，一般数据处理工作25%的时间都在进行排序。简单地说，排序就是把一组记录（元素）按照某个域的值的递增（即由小到大）或递减（即由大到小）的次序重新排列的过程。

表9-1　学生档案表

学号	姓名	年龄	性别
99001	王晓佳	18	男
99002	林一鹏	19	男
99003	谢宁	17	女
99004	张丽娟	18	女
99005	周涛	20	男
99006	李小燕	16	女

例如，在表9-1中，若以每个记录的学号为关键字，按排序码年龄的递增（由小到大）排序，则所有记录的排序结果可简记为：

{（99006，16），（99003，17），（99001，18），（99004，18），（99002，19），（99005，20）}；

也可能为：

{（99006，16），（99003，17），（99004，18），（99001，18），（99002，19），（99005，20）}；

这两个结果都是表9-1按年龄的递增排序结果。若按排序码姓名来进行递增排序，则得到的排序结果为：

{（99006，李小燕），（99002，林一鹏），（99001，王晓佳），（99003，谢宁），（99004，张丽娟），（99005，周涛）}

当然，我们还可以按排序码性别来进行递增排序，在此不再作进一步的分析。

9.1.2 基本概念

1．排序码（Sort Key）

作为排序依据的记录中的一个属性。它可以是任何一种可比的有序数据类型，它可以是记录的关键字，也可以是任何非关键字。如上例中的学生年龄。在此我们认为对任何一种记录都可找到一个取得它排序码的函数Skey(一个或多个关键字的组合)。

2．有序表与无序表

一组记录按排序码的递增或递减次序排列得到的结果被称之为有序表，相应地，把排序前的状态称为无序表。

3．正序表与逆序表

若有序表是按排序码升序排列的，则称为升序表或正序表，否则称为降序表或逆序表。不失普遍性，我们一般只讨论正序表。

4．排序定义

若给定一组记录序列r_1，r_2，…，r_n，其排序码分别为s_1，s_2，…，s_n，将这些记录排成顺序为r_{k1}，r_{k2}，…，r_{kn}的一个序列R’，满足条件$s_{k1}\leq s_{k2}\leq \ldots \leq s_{kn}$，获得这些记录排成顺序为$r_{p1}$，$r_{p2}$，…，$r_{pn}$的一个序列R”，满足条件$s_{p1}\leq s_{p2}\leq \ldots \leq s_{pn}$的过程称为排序。也可以说，将一组记录按某排序码递增或递减排列的过程，称为排序。

5．稳定与不稳定

因为排序码可以不是记录的关键字，同一排序码值可能对应多个记录。对于具有同一排序码的多个记录来说，若采用的排序方法使排序后记录的相对次序不变，则称此排序方法是稳定的，否则称为不稳定的。在上例中（见表9-1，按年龄排序），如果一种排序方法使排序后的结果必为前一个结果，则称此方法是稳定的；若一种排序方法使排序后的结果可能为后一个结果，则称此方法是不稳定的。

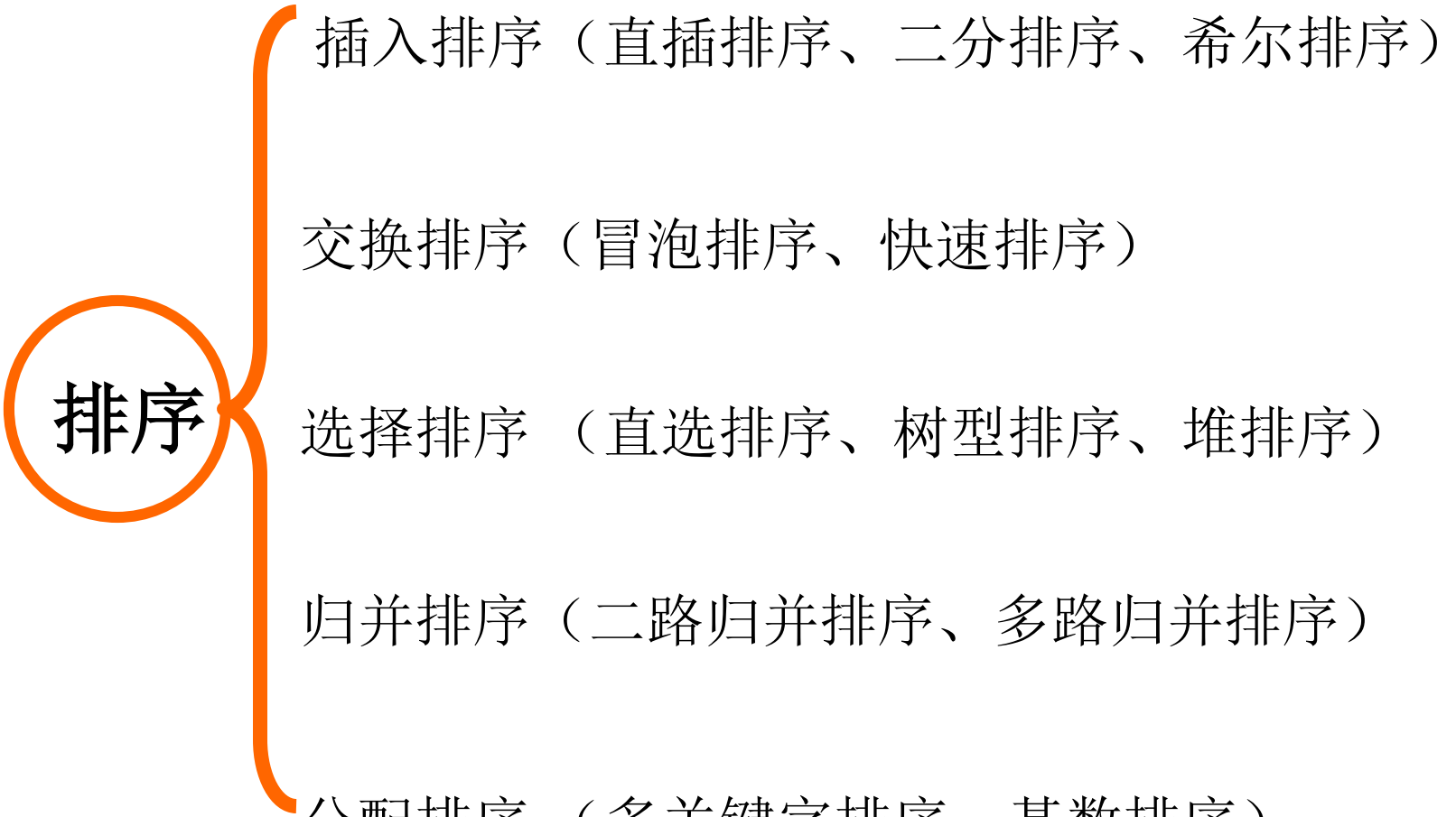
6. 内排序与外排序

按照排序过程中使用内外存的不同将排序方法分为内排序和外排序。若排序过程全部在内存中进行，则称为内排序；若排序过程需要不断地进行内存和外存之间的数据交换，则称为外排序。内排序大致可分为五类：插入排序、交换排序、选择排序、归并排序和分配排序。本章仅讨论内排序。

7. 排序的时间复杂性

排序过程主要是对记录的排序码进行比较和记录的移动过程。因此排序的时间复杂性可以算法执行中的数据比较次数及数据移动次数来衡量。当一种排序方法使排序过程在最坏或平均情况下所进行的比较和移动次数越少，则认为该方法的时间复杂性就越好，分析一种排序方法，不仅要分析它的时间复杂性，而且要分析它的空间复杂性、稳定性和简单性等。

为了以后讨论方便，我们直接将排序码写成一个一维数组的形式，并且在没有声明的情形下，所有排序都按排序码的值递增排列。



9．2 插入排序

9.2.1直接插入排序

1．直接插入排序的基本思想

直接插入排序（Straight Insertion Sorting）的基本思想是：把n个待排序的元素看成为一个有序表和一个无序表，开始时有序表中只包含一个元素，无序表中包含有n-1个元素，排序过程中每次从无序表中取出第一个元素，把它的排序码依次与有序表元素的排序码进行比较，将它插入到有序表中的适当位置，使之成为新的有序表。

2．直接插入的算法实现

```
void sort(NODE array[],int n)
//待排序元素用一个数组array[ ] 表示，数组有n个元素
{  int i, j;
    NODE x;                  // x 为中间结点变量
    for ( i=1; i<n; i++)   //i表示插入次数,共进行n-1次插入
    {  x=array[i];      //把待排序元素赋给 x
       j=i-1;
   while ((j>=0)&& ( x.key<array[j].key))
       {array[j+1]=array[j];  j--; } // 顺序比较和移动
       array[j+1]=x;  }
}
```

例如，n=6，数组R的六个排序码分别为：17，3，25，14，20，9。它的直接插入排序的执行过程如图9-1所示。

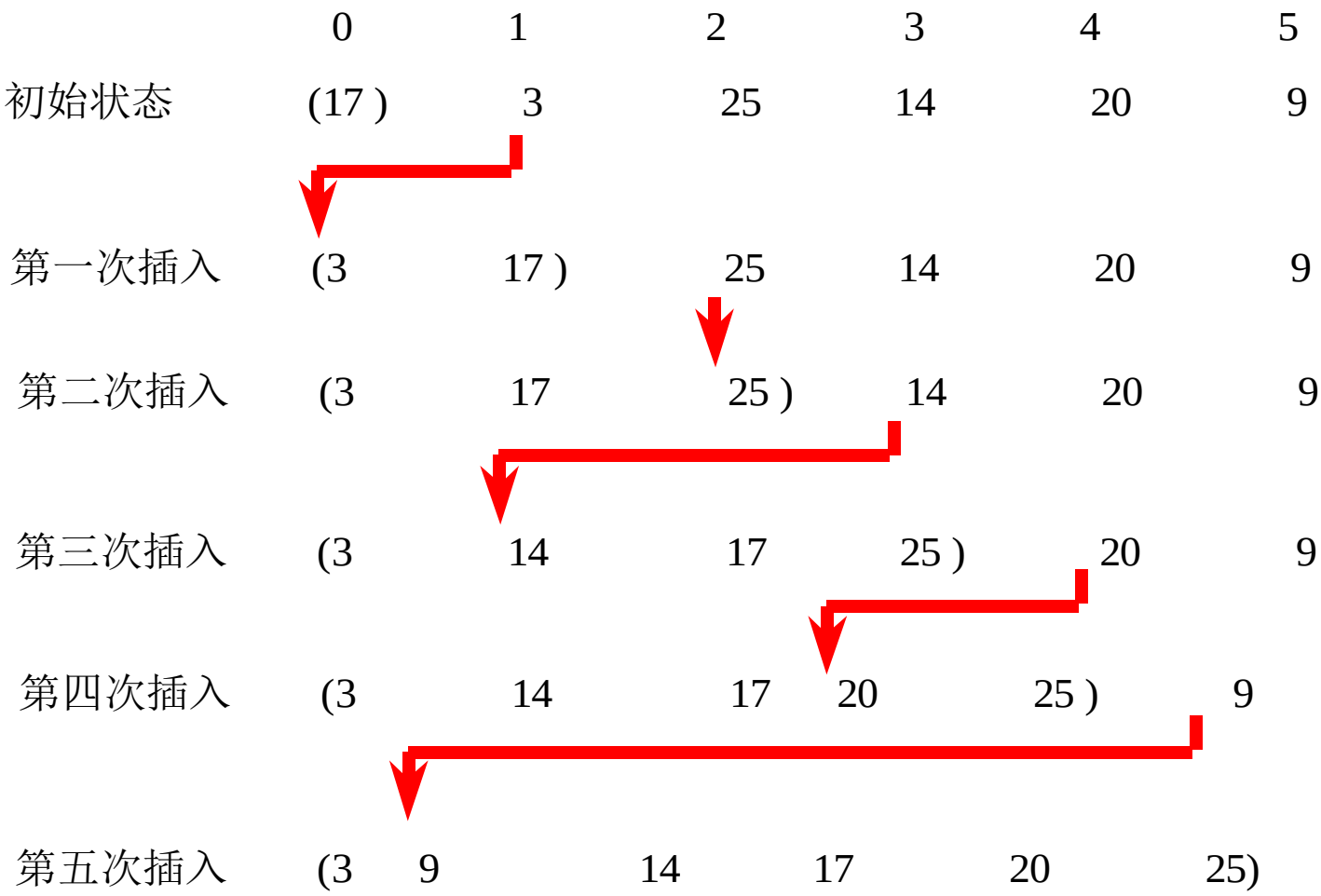


图 9-1 直接插入排序示例

3．直接插入排序的效率分析

从上面的叙述可以看出，直接插入排序算法十分简单。那么它的效率如何呢？首先从空间来看，它只需要一个元素的辅助空间，用于元素的位置交换。从时间分析，首先外层循环要进行n-1次插入，每次插入最少比较一次（正序），移动两次；最多比较i次，移动i＋2次（逆序）（i=1，2，…，n-1）。因此，直接插入排序的时间复杂度为O（n^2）。

直接插入算法的元素移动是顺序的，该方法是稳定的。

9.2.3希尔排序

1．希尔排序的基本思想

希尔排序，又称为“缩小增量排序”。是1959年由D.L.Shell提出来的。该方法的基本思想是：先将整个待排元素序列分割成若干个子序列（由相隔某个“增量”的元素组成的）分别进行直接插入排序，待整个序列中的元素基本有序（增量足够小）时，再对全体元素进行一次直接插入排序。因为直接插入排序在元素基本有序的情况下（接近最好情况），效率是很高的，因此希尔排序在时间效率上比前两种方法有较大提高。

3．希尔排序的效率分析

虽然我们给出的算法是三层循环，最外层循环为$\log_2 n$数量级，中间的for循环是n数量级的，内循环远远低于n数量级，因为当分组较多时，组内元素较少；此循环次数少；当分组较少时，组内元素增多，但已接近有序，循环次数并不增加。因此，希尔排序的时间复杂性在O（$n\log_2 n$）和O（n^2 ）之间，大致为O（$n^{1.3}$）。

例如，n=8，数组array[ ]的八个元素分别为：91，67，35，62，29，72，46，57。下面用图9-2给出希尔排序算法的执行过程。

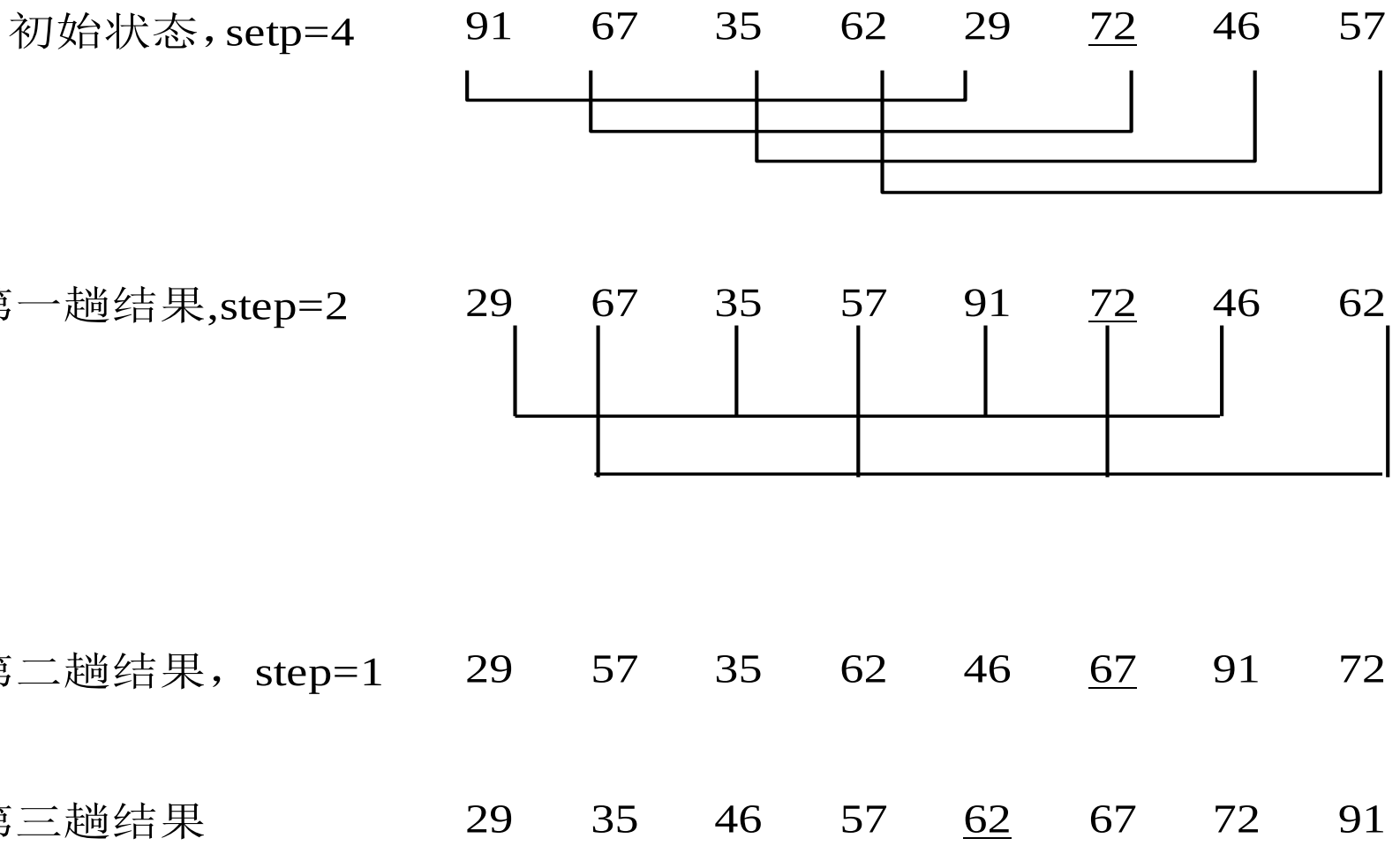


图 9-2 希尔排序算法的执行过程

由于希尔排序对每个子序列单独比较，在比较时进行元素移动，有可能改变相同排序码元素的原始顺序，因此希尔排序是不稳定的。

9．3 交换排序

9.3.1 冒泡排序

1．冒泡排序的基本思想

通过对待排序序列从后向前（从下标较大的元素开始），依次比较相邻元素的排序码，若发现逆序则交换，使排序码较小的元素逐渐从后部移向前部（从下标较大的单元移向下标较小的单元），就象水底下的气泡一样逐渐向上冒。

因为排序的过程中，各元素不断接近自己的位置，如果一趟比较下来没有进行过交换，就说明序列有序，因此要在排序过程中设置一个标志flag判断元素是否进行过交换。从而减少不必要的比较。

2. 冒泡排序的算法实现

```
void Bubblesort( NODE array[],int n)
{  int i, j, flag;    //当flag为0则停止排序
   NODE temp;
    for  ( i=1; i<n; i++) //  i  表示趟数，  最多n-1趟
    { flag=0;                  //开始时元素未交换
      for ( j=n-1; j>=i; j--)
        if (array[j].key<array[j-1].key)   //发生逆序
         temp=array[j];array[j]=array[j-1];array[j-1]=temp;
         flag=1; }                              //交换，并标记发生了交换
       if(flag==0) break;    }
  }
```

例如，n=6，数组R的六个排序码分别为：17，3，25，14，20，9。下面用图9-3给出冒泡排序算法的执行过程。

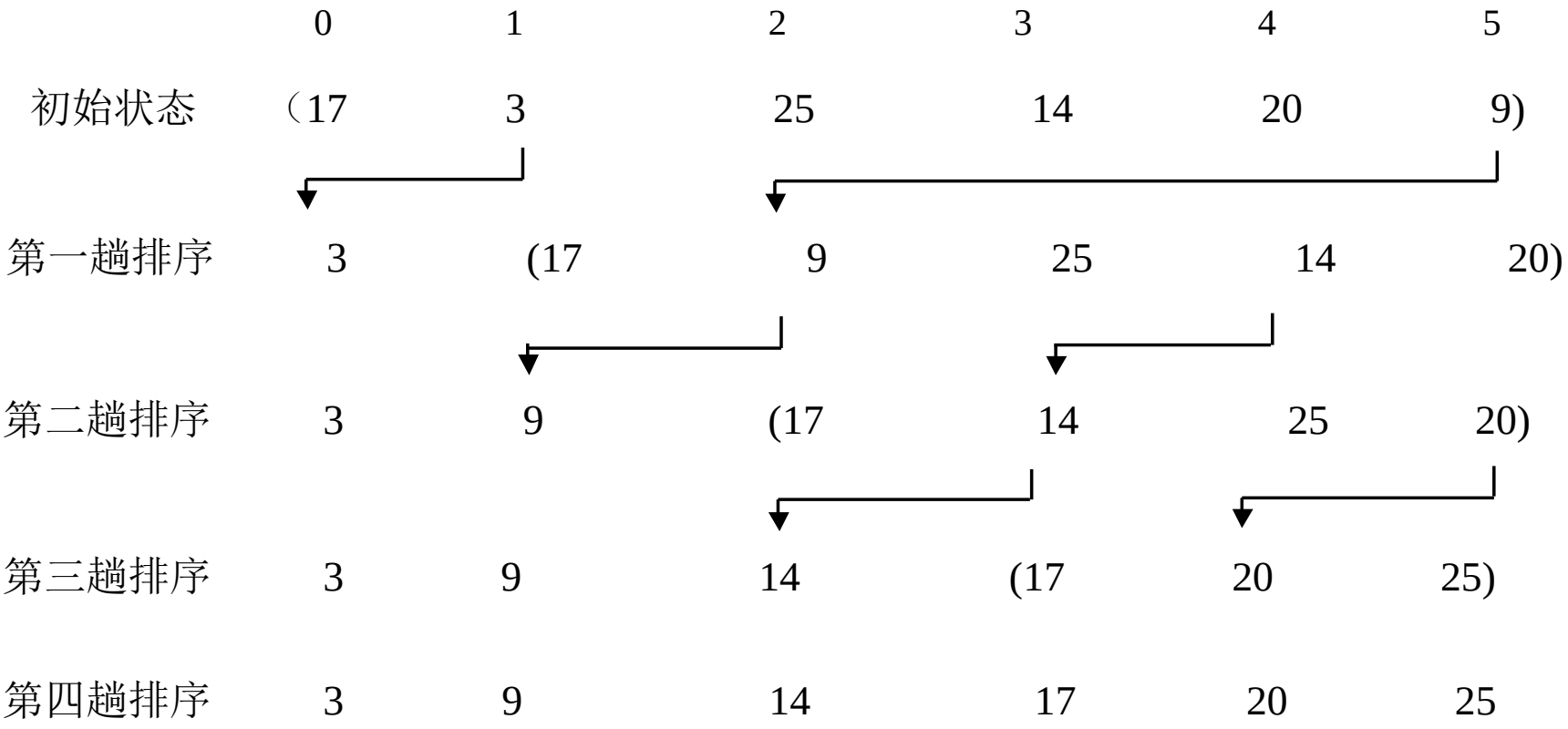


图 9-3 冒泡排序示例

3．冒泡排序的效率分析

从上面的例子可以看出，当进行完第三趟排序时，数组已经有序，所以第四趟排序的交换标志为0，即没进行交换，所以不必进行第四趟排序。

从冒泡排序的算法可以看出，若待排序的元素为正序，则只需进行一趟排序，比较次数为（n-1）次，移动元素次数为0；若待排序的元素为逆序，则需进行n-1趟排序，比较次数为$(n^2-n)/2$，移动次数为$3(n^2-n)/2$，因此冒泡排序算法的时间复杂度为$O（n^2）$。由于其中的元素移动较多，所以属于内排序中速度较慢的一种。

因为冒泡排序算法只进行元素间的顺序移动，所以是一个稳定的算法。

9.3.2 快速排序

1．快速排序的基本思想

快速排序（Quick Sorting）是迄今为止所有内排序算法中速度最快的一种。它的基本思想是：任取待排序序列中的某个元素作为基准（一般取第一个元素），通过一趟排序，将待排元素分为左右两个子序列，左子序列元素的排序码均小于或等于基准元素的排序码，右子序列的排序码则大于基准元素的排序码，然后分别对两个子序列继续进行排序，直至整个序列有序。快速排序是对冒泡排序的一种改进方法，算法中元素的比较和交换是从两端向中间进行的，排序码较大的元素一次就能够交换到后面单元，排序码较小的记录一次就能够交换到前面单元，记录每次移动的距离较远，因而总的比较和移动次数较少。

快速排序的过程为：把待排序区间按照第一个元素（即基准元素）的排序码分为左右两个子序列的过程叫做一次划分。设待排序序列为array[start]～array[end]，其中start为下限，end为上限，start＜end，mid=array[start]为该序列的基准元素，为了实现一次划分，令i,j的初值分别为start和end。在划分过程中，首先让j从它的初值开始，依次向前取值，并将每一元素 array[j] 的排序码同 mid 的排序码进行比较，直到array[j].key<mid.key时，交换array[j]与array[i]的值，使排序码相对较小的元素交换到左子序列，然后让i从i＋1开始，依次向后取值，并使每一元素array[i]的排序码同array[j]的排序码（此时array[j]为基准元素）进行比较，直到array[i]＞array[j]时，交换array[i]与array[j]的值，使排序码大的元素交换到后面子区间；再接着让j从j-1开始，依次向前取值，重复上述过程，直到i等于j,即指向同一位置为止，此位置就是基准元素最终被存放的位置。此次划分得到的前后两个待排序的子序列分别为 array[start] ～ array[i-1] 和 array[i+1] ～ array[end]。

例如，给定排序码为：（46，55，13，42，94，05，17，70），具体划分如图9-4所示。

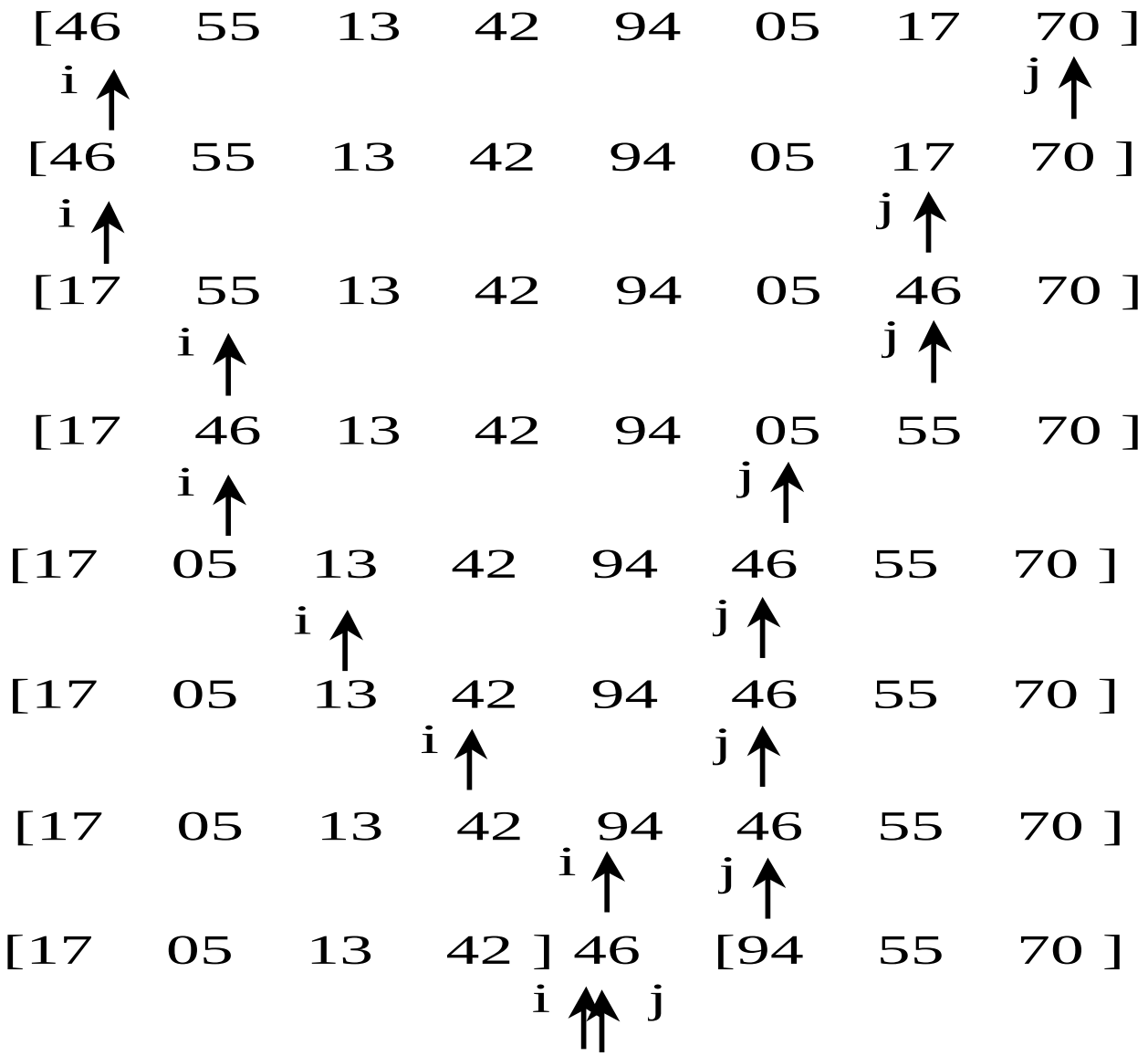


图 9-4　快速排序的一次划分

从图9-4可知，通过一次划分，将一个区间以基准值分成两个子区间，左子区间的值小于等于基准值，右子区间的值大于基准值。对剩下的子区间重复此划分步骤，则可以得到快速排序的结果。

2．快速排序的算法实现

下面给出快速排序算法的递归算法如下：

```
void quicksort(NODE array[],int start , int end)
{  int i , j; NODE mid;
   if (start>=end) return;
   i=start;j=end;mid=array[i];
  while (i<j)
  {    while (i<j && array[j].key>mid.key)  j--;
    if (i<j) {array[i]=array[j]; i++;    }
    while (i<j && array[i].key<=mid.key)   i++;
    if (i<j)  {array[j]=array[i]; j--;    }  } //一次划分得到基准值的正确位置
      array[i]=mid;
       quicksort(array,start,i-1);       //递归调用左子区间
       quicksort(array,i+1,end); }//递归调用右子区间
```

3．快速排序的效率分析

若快速排序出现最好的情形（左、右子区间的长度大致相等），则结点数n与二叉树深度h应满足$\log_2 n<h<\log_2 n+1$，所以总的比较次数不会超过$(n+1)\log_2 n$。因此，快速排序的最好时间复杂度应为O（$n\log_2 n$）。而且在理论上已经证明，快速排序的平均时间复杂度也为O（$n\log_2 n$）。

若快速排序出现最坏的情形（每次能划分成两个子区间，但其中一个是空），则这时得到的二叉树是一棵单分枝树，得到的非空子区间包含有n-i个(i代表二叉树的层数（$1\le i\le n$）元素，每层划分需要比较n-i+2次，所以总的比较次数为（n^2+3n-4）/2。因此，快速排序的最坏时间复杂度为$O(n^2)$。

快速排序所占用的辅助空间为栈的深度，故最好的空间复杂度为O（$\log_2 n$），最坏的空间复杂度为O(n)。

快速排序是一种不稳定的排序方法。

9．4 选择排序

9.4.1 直接选择排序

1．直接选择排序的基本思想

直接选择排序也是一种简单的排序方法。它的基本思想是：第一次从array[0]~array[n-1]中选取最小值，与array[0]交换，第二次从array[1]~array[n-1]中选取最小值，与array[1]交换，第三次从array[2]~array[n-1]中选取最小值，与array[2]交换，…，第i次从array[i-1]~array[n-1]中选取最小值，与array[i-1]交换，…，第n-1次从array[n-2]~array[n-1]中选取最小值，与array[n-2]交换，总共通过n-1次，得到一个按排序码从小到大排列的有序序列。

例如，给定n=8，数组R中的8个元素的排序码为：（8，3，2，1，7，4，6，5），则直接选择排序过程如图9-5所示。

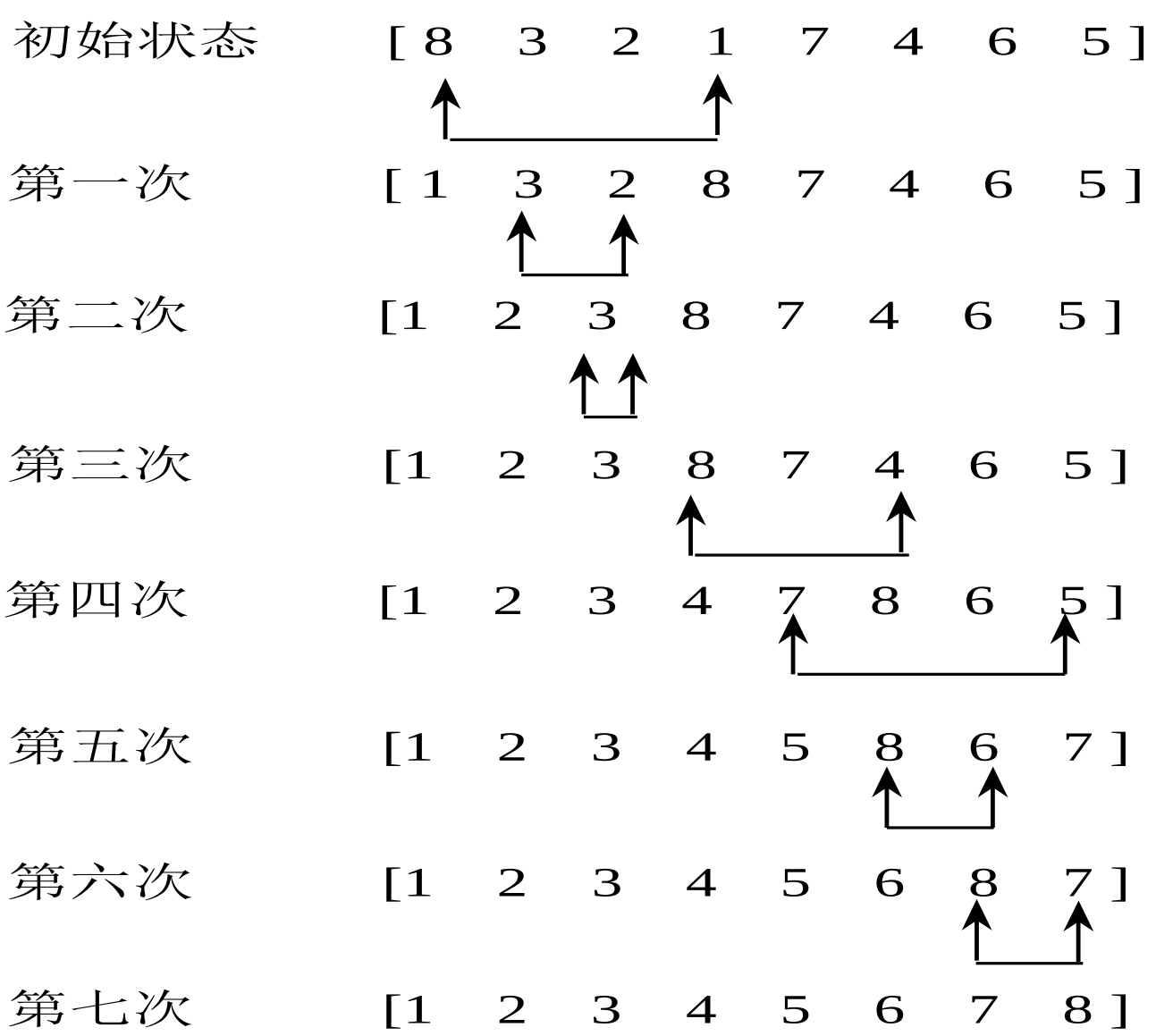


图 9-5 直接选择排序的过程示例

3．直接选择排序的效率分析

在直接选择排序中，共需要进行n-1次选择和交换，每次选择需要进行n-i次比较（1≤i≤n-1），而每次交换最

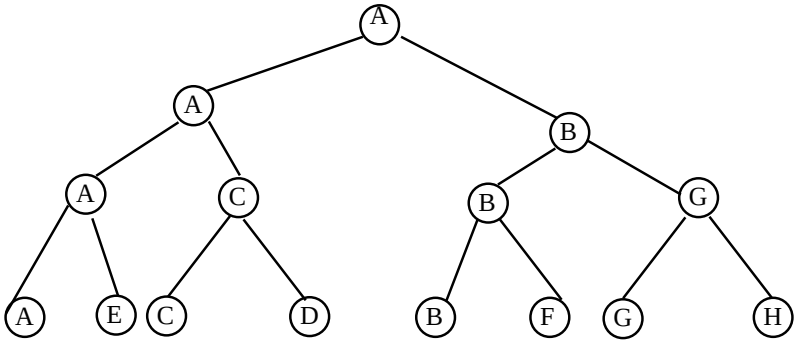
多需3次移动，因此，总的比较次数C=$\sum_{i=1}^{n-1}(n-i)$=(n^2-n)/2，

总的移动次数M=$\sum_{i=1}^{n-1}3$ =3(n-1)。由此可知，直接选择排序的时间复杂度为$O(n^2)$数量级，所以当记录占用的字节数较多时，通常比直接插入排序的执行速度要快一些。

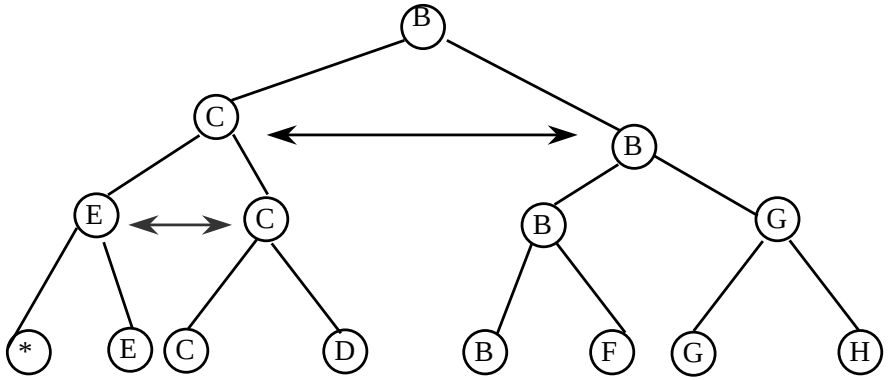
由于在直接选择排序中存在着不相邻元素之间的互换，因此，直接选择排序是一种不稳定的排序方法。例如，给定排序码为3，7，3，2，1，排序后的结果为1，2，3，3，7。

9.4.2 树形选择排序

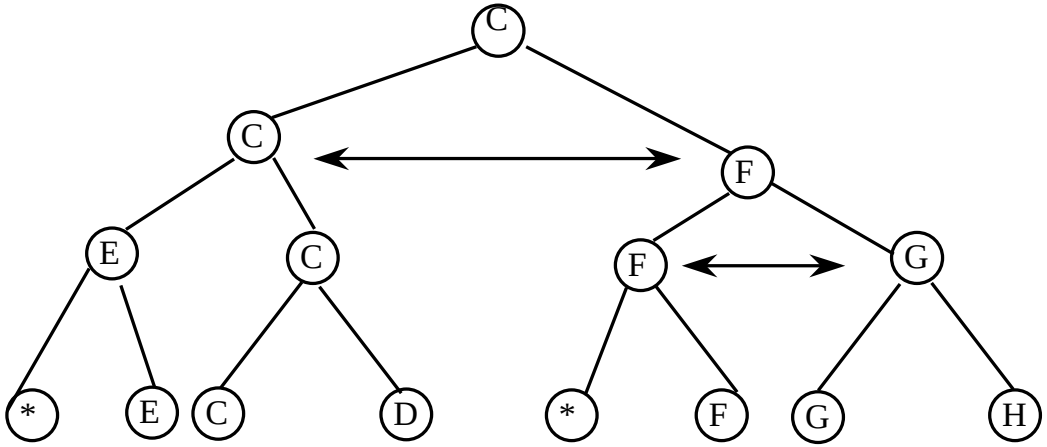
从直接选择排序可知，在n个排序码中，找出最小值需n-1次比较，找出第二小值需n-2次比较，找出第三小值需n-3次比较，其余依此类推。所以，总的比较次数为：(n-1)+(n-2)+…+3+2+1= $(n^2-n)/2$，那么，能否对直接选择排序算法加以改进，使总的比较次数比 $(n^2-n)/2$小呢？显然，在n个排序码中选出最小值，至少进行n-1次比较，但是，继续在剩下的n-1个关键字中选第二小的值，就并非一定要进行n-2次比较，若能利用前面n-1次比较所得信息，则可以减少以后各趟选择排序中所用的比较次数，比如8个运动员中决出前三名，不需要7+6+5=18场比赛（前提是，若甲胜乙，而乙胜丙，则认为甲胜丙），最多需要11场比赛即可（通过7场比赛产生冠军后，第二名只能在输给冠军的三个对中产生，需 2场比赛，而第三名只能在输给亚军的三个对中产生，也需2场比赛，总共11场比赛）。具体见图9-6所示。



(a) 8 个队决出冠军的情形（共 7 场比赛）



(b)决出亚军的情形（共 2 场比赛，少于 6 场）



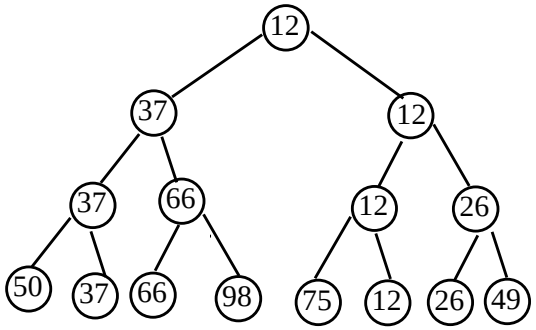
(c) 决出第三名的情形（共 2 场比赛，少于 6 场）

图 9-6 决出比赛前三名的过程示意图

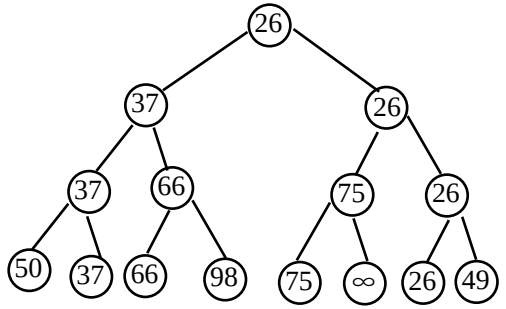
从图9-6（a）可知，8个队经过4场比赛，获胜的4个队进入半决赛，再经过2场半决赛和1场决赛即可知道冠军属谁（共7场比赛）按照锦标赛的传递关系，亚军只能产生于分别在决赛，半决赛和第一轮比赛中输给冠军的选取手中，于是亚军只能在b、c、e这3个队中产生（见图9-6（b）），即进行2场比赛（e 与c一场，e与c的胜队与b一场）后，即可知道亚军属谁。同理，第三名只需在c、f、g这3个队产生（见图9-6（c））即进2场比赛（f与g一场，f与g的胜队与c一场）即可知道第三名属谁。

树形选择排序，又称锦标赛排序，是一种按照锦标赛的思想进行选择排序的方法。首先对n个记录的排序码进行两两比较，然后在其中⌈n/2⌉ 个较小者之间再进行两两比较，如此重复，直到选出最小排序码为止。

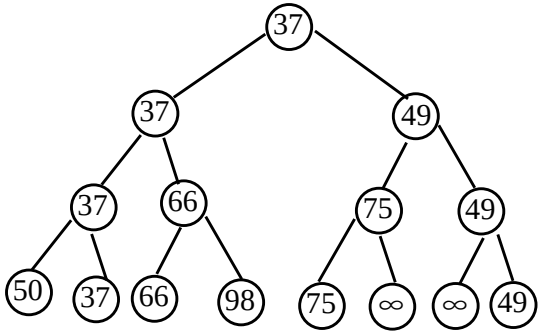
例如，给定排序码头 50，37，66，98，75，12，26，49，树形选择排序过程见图9-7。



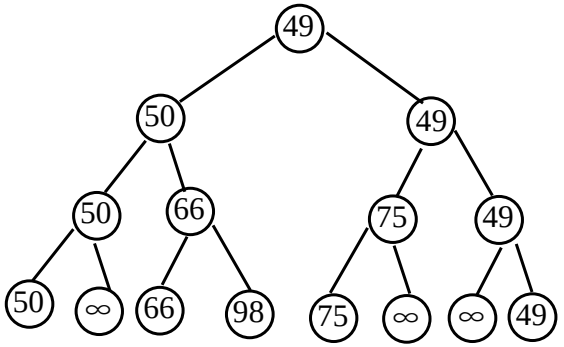
(a) 经过 7 次比较得到最小值 12



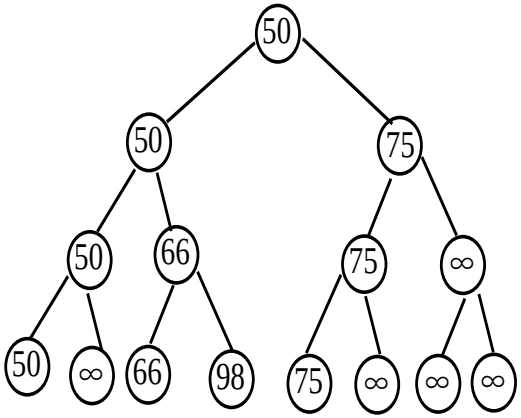
(b) 输出 12 后，经过 2 次比较得到第二小值 26



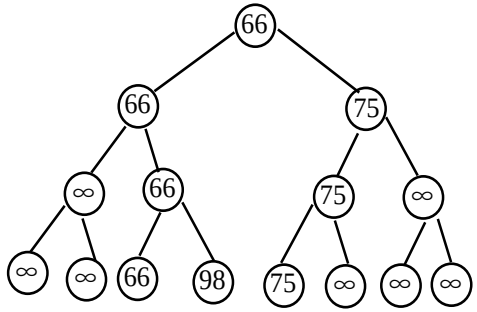
(c) 输出 12，26 后，经过 2 次比较得到第三小值 37



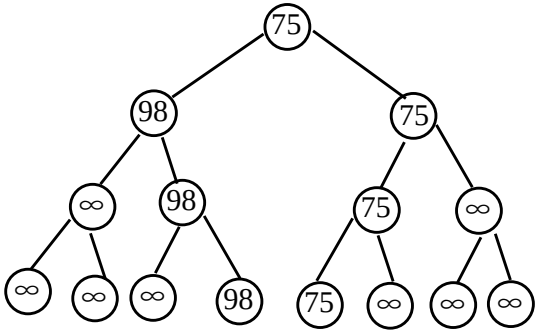
（d）输出 12,26,37 后，经过 2 次比较得到第四小值 49



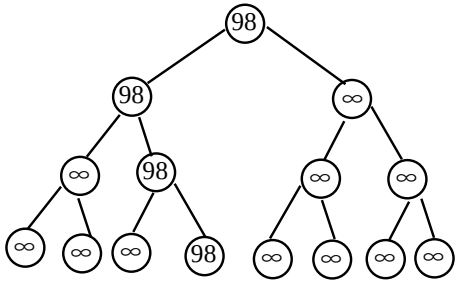
(e) 输出 12,26,37,49 后，经过 1 次比较得到第五小值 50



(f) 输出 12,26,37,49,50 后，经过 1 次比较得到第六小值 66



(g) 输出 12,26,37,49,50,66 后，经过 1 次比较得到第七小值 75



(h) 输出 12,26,37,49,50,66,75 后，经过 1 次比较得到第八小值 98

图 9-7 树形选择排序示意过程

9.4.3 堆排序

1. 堆的定义

若有n个元素的排序码k_1，k_2，k_3，…，k_n，当满足如下条件：

（1） $\begin{cases} k_i \le k_{2i} \\ k_i \le k_{2i+1} \end{cases}$ 或 (2) $\begin{cases} k_i \ge k_{2i} \\ k_i \ge k_{2i+1} \end{cases}$

其中$i=1,2,\ldots,\lfloor n/2 \rfloor$则称此n个元素的排序码$k_1$，$k_2$，$k_3$，…，$k_n$为一个堆。

若将此排序码按顺序组成一棵完全二叉树，则（1）称为小根堆（二叉树的所有根结点值小于或等于左右孩子的值），（2）称为大根堆（二叉树的所有根结点值大于或等于左右孩子的值）。

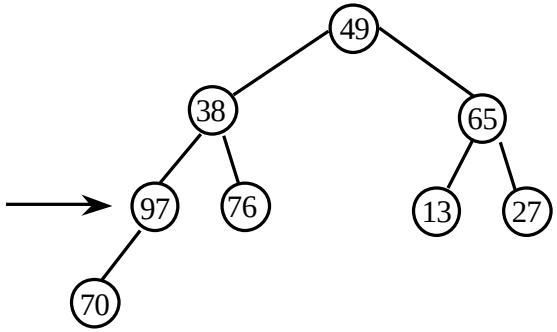
若n个元素的排序码k_1，k_2，k_3，…，k_n满足堆，且让结点按1、2、3、…、n顺序编号，根据完全二叉树的性质（若i为根结点，则左孩子为2i，右孩子为2i+1）可知，堆排序实际与一棵完全二叉树有关。若将排序码初始序列组成一棵完全二叉树，则堆排序可以包含建立初始堆（使排序码变成能符合堆的定义的完全二叉树）和利用堆进行排序两个阶段。

2．堆排序的基本思想

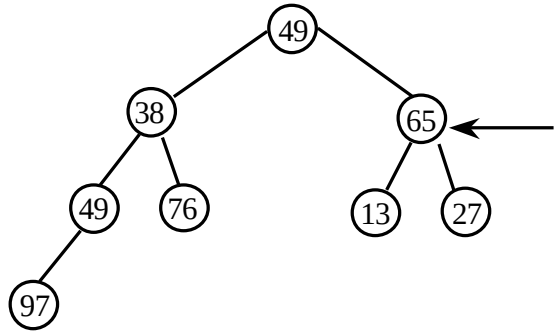
将排序码k_1，k_2，k_3，…，k_n表示成一棵完全二叉树，然后从第$\lfloor n/2 \rfloor$个排序码(即树的最后一个非终端结点)开始筛选，使由该结点作根结点组成的子二叉树符合堆的定义，然后从第$\lfloor n/2 \rfloor$-1个排序码重复刚才操作，直到第一个排序码止。这时候，该二叉树符合堆的定义，初始堆已经建立。

接着，可以按如下方法进行堆排序：将堆中第一个结点（二叉树根结点）和最后一个结点的数据进行交换（k_1与k_n），再将k_1~k_{n-1}重新建堆，然后k_1和k_{n-1}交换，再将k_1~k_{n-2}重新建堆，然后k_1和k_{n-2}交换，如此重复下去，每次重新建堆的元素个数不断减1，直到重新建堆的元素个数仅剩一个为止。这时堆排序已经完成，则排序码k_1，k_2，k_3，…，k_n已排成一个有序序列。

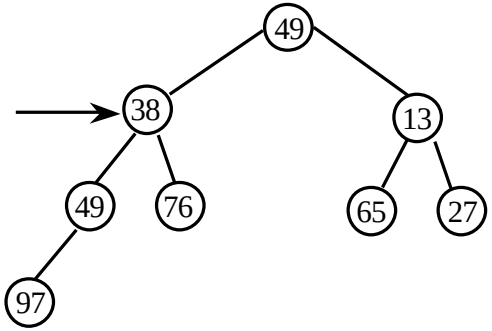
例如，给定排序码49，38，65，97，76，13，27，49，建立初始堆的过程如图9-8所示。



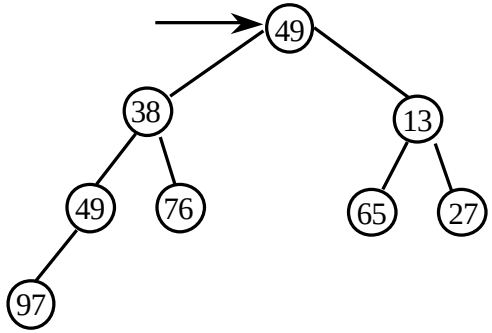
（a）初始无序的结点，从 97 开始调整



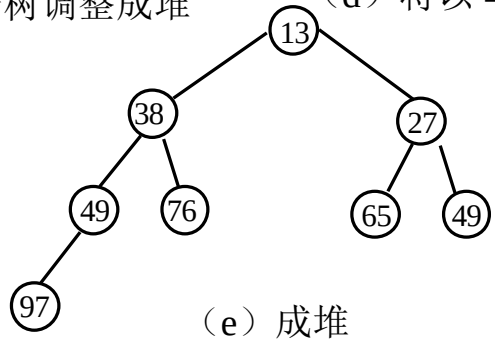
（b）将以 65 为根的子树调整成堆



（c）将以 38 为根的子树调整成堆



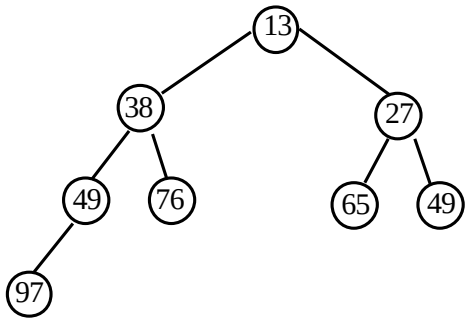
（d）将以 49 为根的子树调整成堆



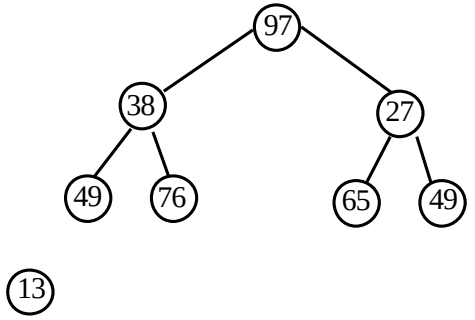
（e）成堆

图 9-8　建立堆的过程示意图

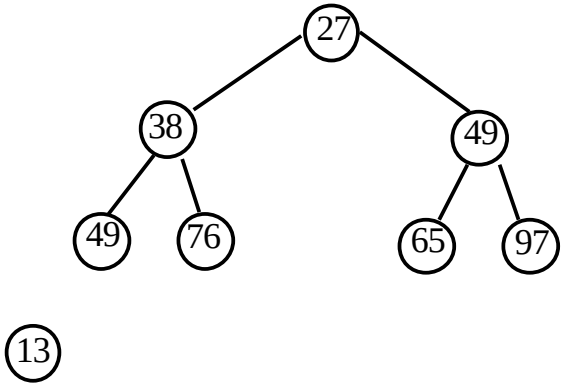
建成如图9-8(e)所示的堆后，堆排序过程如图9-9所示。



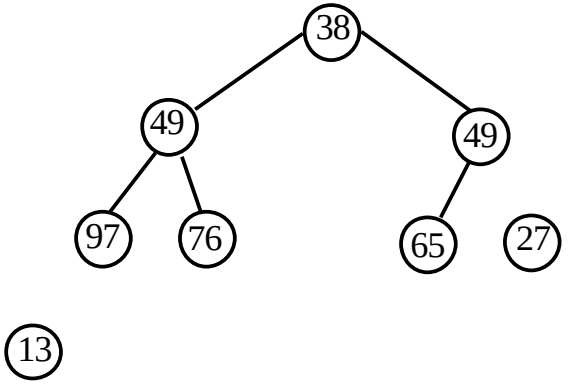
(a)　初始堆



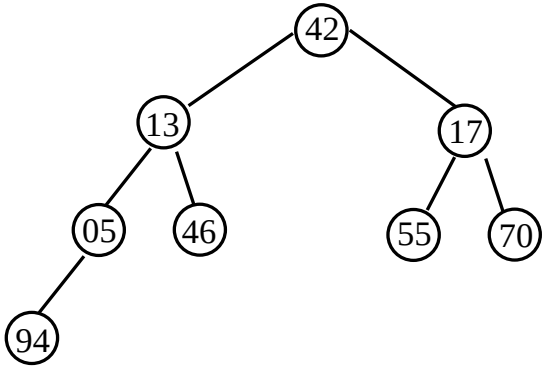
(b)　13 与 97 交换



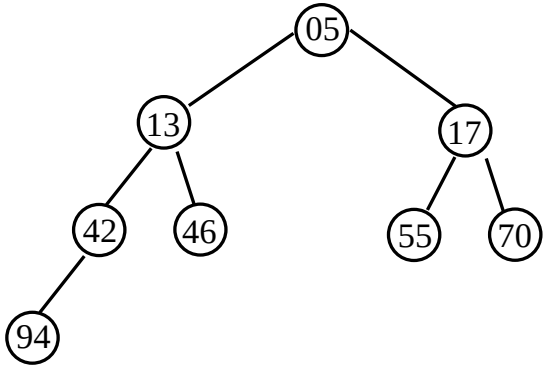
(c)重新建成堆



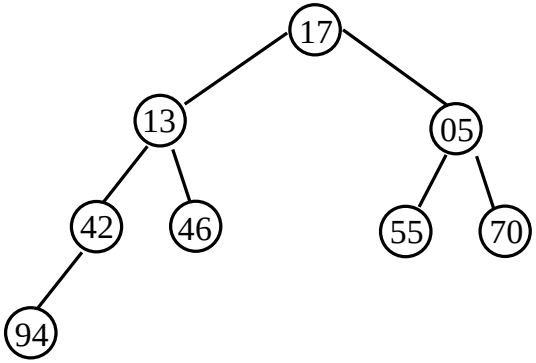
(d) 27 和 97 交换后再调整



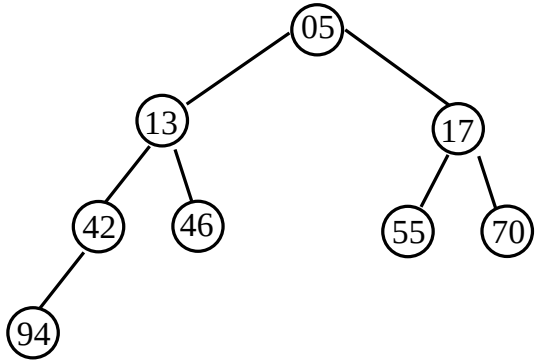
(i)　前 4 个排序码重新建成堆



(j)　42 和 05 交换



(k)　前 3 个排序码重新建成堆



(l)　17 和 05 交换

4．堆排序的效率分析

在整个堆排序中，共需要进行$n+\lfloor n/2 \rfloor -1$次筛选运算，每次筛选运算进行双亲和孩子或兄弟结点的排序码的比较和移动次数都不会超过完全二叉树的深度，所以，每次筛选运算的时间复杂度为$O(\log_2 n)$，故整个堆排序过程的时间复杂度为$O(n\log_2 n)$。

堆排序占用的辅助空间为1（供交换元素用），故它的空间复杂度为O（1）。

堆排序是一种不稳定的排序方法，例如，给定排序码：2，1，2，它的排序结果为：1，2，2。

9．5 归并排序

9.5.1 二路归并排序

二路归并排序的基本思想

二路归并排序的基本思想是：将两个有序子区间（有序表）合并成一个有序子区间，一次合并完成后，有序子区间的数目减少一半，而区间的长度增加一倍，当区间长度从1增加到n（元素个数）时，整个区间变为一个，则该区间中的有序序列即为我们所需的排序结果。

例如，给定排序码46，55，13，42，94，05，17，70，二路归并排序过程如图9-10所示。

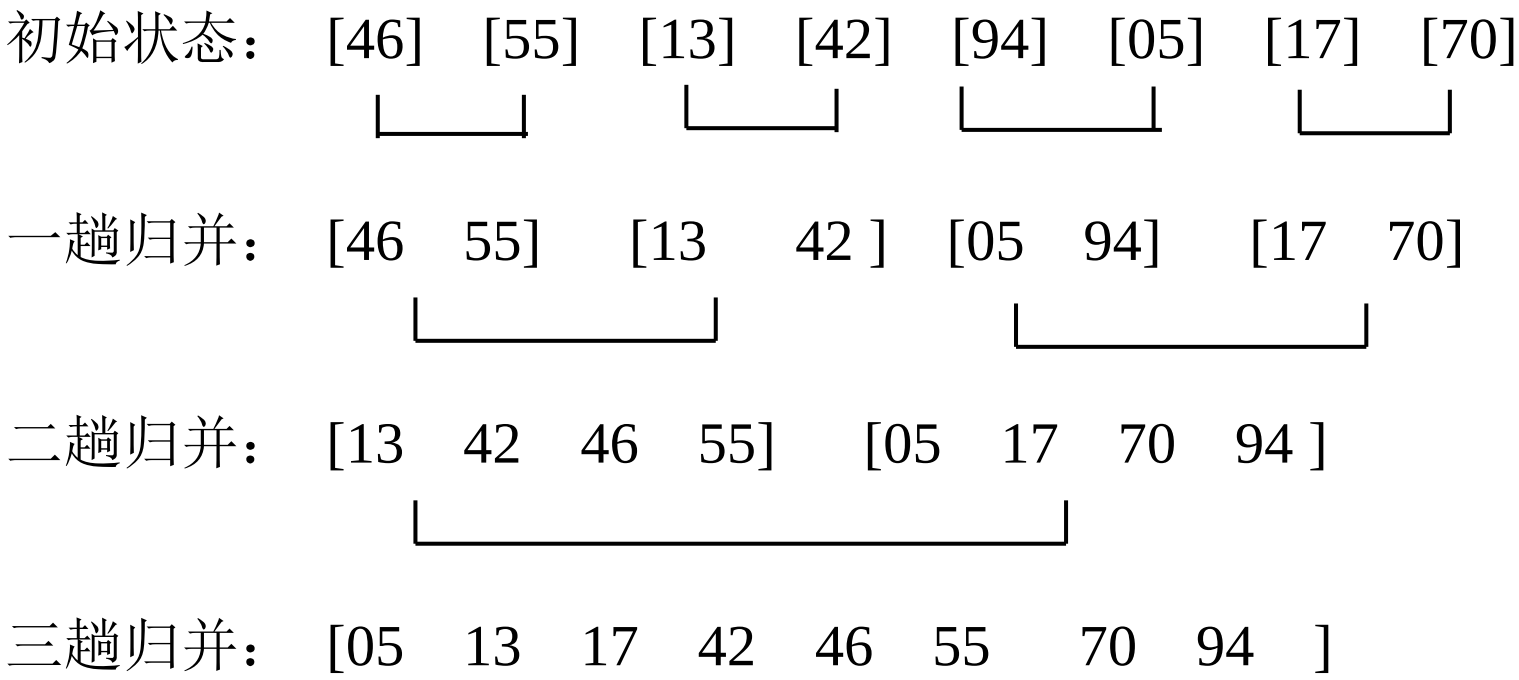


图 9-10 二路归并排序过程示意图

3．二路归并排序的效率分析

二路归并排序的时间复杂度等于归并趟数与每一趟时间复杂度的乘积。而归并趟数为$\lceil \log_2 n \rceil$（当$\lceil \log_2 n \rceil$为奇数时，则为$\lceil \log_2 n \rceil$ +1)。因为每一趟归并就是将两两有序子区间合并成一个有序子区间，而每一对有序子区间归并时，记录的比较次数均小于等于记录的移动次数（即由一个数组复制到另一个数组中的记录个数），而记录的移动次数等于这一对有序表的长度之和，所以，每一趟归并的移动次数均等于数组中记录的个数n，即每一趟归并的时间复杂度为O(n)。因此，二路归并排序的时间复杂度为$O(n\log_2 n)$。

利用二路归并排序时，需要利用与待排序数组相同的辅助数组作临时单元，故该排序方法的空间复杂度为O(n)，比前面介绍的其它排序方法占用的空间大。

由于二路归并排序中，每两个有序表合并成一个有序表时，若分别在两个有序表中出现有相同排序码，则会使前一个有序表中相同排序码先复制，后一有序表中相同排序码后复制，从而保持它们的相对次序不会改变。所以，二路归并排序是一种稳定的排序方法。

9．7 各种内排序方法的比较和选择

9.7.1 各种内排序方法的比较

1．从时间复杂度比较

从平均时间复杂度来考虑，直接插入排序、冒泡排序、直接选择排序是三种简单的排序方法，时间复杂度都为$O(n^2)$，而快速排序、堆排序、二路归并排序的时间复杂度都为$O(n\log_2 n)$，希尔排序的复杂度介于这两者之间。若从最好的时间复杂度考虑，则直接插入排序和冒泡排序的时间复杂度最好，为$O(n)$，其它的最好情形同平均情形相同。若从最坏的时间复杂度考虑，则快速排序的为$O(n^2)$，直接插入排序、冒泡排序、希尔排序同平均情形相同，但系数大约增加一倍，所以运行速度将降低一半，最坏情形对直接选择排序、堆排序和归并排序影响不大。

2．从空间复杂度比较

归并排序的空间复杂度最大，为O(n)，快速排序的空间复杂度为$O(\log_2 n)$，其它排序的空间复杂度为O（1）。

3.从稳定性比较

直接插入排序、冒泡排序、归并排序是稳定的排序方法，而直接选择排序、希尔排序、快速排序、堆排序是不稳定的排序方法。

4．从算法简单性比较

直接插入排序、冒泡排序、直接选择排序都是简单的排序方法，算法简单，易于理解，而希尔排序、快速排序、堆排序、归并排序都是改进型的排序方法，算法比简单排序要复杂得多，也难于理解。

9.7.2 各种内排序方法的选择

1．从时间复杂度选择

对元素个数较多的排序，可以选快速排序、堆排序、归并排序，元素个数较少时，可以选简单的排序方法。

2．从空间复杂度选择

尽量选空间复杂度为O（1）的排序方法，其次选空间复杂度为O($\log_2 n$)的快速排序方法，最后才选空间复杂度为O（n）二路归并排序的排序方法。

3．一般选择规则

（1） 当待排序元素的个数n较大，排序码分布是随机，而对稳定性不做要求时，则采用快速排序为宜。

（2)当待排序元素的个数n大，内存空间允许，且要求排序稳定时，则采用二路归并排序为宜。

（3）当待排序元素的个数n大，排序码分布可能会出现正序或逆序的情形，且对稳定性不做要求时，则采用堆排序或二路归并排序为宜。

（4)当待排序元素的个数n小，元素基本有序或分布较随机，且要求稳定时，则采用直接插入排序为宜。

(5)当待排序元素的个数n小，对稳定性不做要求时，则采用直接选择排序为宜，若排序码不接近逆序，也可以采用直接插入排序。冒泡排序一般很少采用。

谢谢使用

- 地　　址：南京气象学院计算机科学与技术系
- 邮　　编：210044
- 电　　话：025-8731482
- E-mail：cst@njim.edu.cn

欢迎提出宝贵意见！