第6章 排序

一、单元概述

排序(sorting)又称分类,是计算机程序设计中的一个重要操作,即把一批任意序列的数据元素(或记录),重新排列成一个按关键字有序的序列。从操作的角度看,排序是对线性结构的一种操作。在当今的计算机系统中,花费在排序上的时间占系统CPU运行时间高达20%~60%。如何进行排序,特别是如何高效率地排序是计算机应用中的一个重要课题。本章主要介绍各种排序方法的思想、算法实现和算法分析。

二、知识要点及掌握程度

6.1 基本概念和术语:理解
6.2 插入排序:分析;
6.3 交换排序:分析;
6.4 选择排序:分析;
6.5 归并排序:分析;
6.7 基数排序:分析;
6.8 各种排序方法的比较:分析。

三、能力要点及重要程度

(1)计算机基础知识:掌握各种排序算法的思想和实现。(重要)

(2)分析问题:针对具体的排序问题,要能够运用各种排序方法去进行分析,找到最优的算法。(重要)

(3)解决方法和建议:在具体工程应用中,发现了关于排序的问题,要能够解决问题,并提出合理的建议。(中等)

(4)设计过程的分段与方法:采取不同的阶段去设计(概念设计、详细设计)一个具体的排序算法。(中等)

(5)软件实现过程:了解系统各个模块中的关于排序的问题;讨论算法(数据结构、控制流程、数据流程);使用编程语言实施底层设计(编程)。(中等)

四、教学重点与难点

重点:掌握各种排序算法的思想。
难点:复杂排序的具体实现。

五、教学设计与实施方法

本单元主要采用项目驱动法、讲授教学法、实践教学法和练习教学法。通过案例驱动,引出

东软电子出版社

课程内容；通过讲授教学法，教师课堂讲授理论知识；通过课堂和课后实践，完成项目实施；练习教学法通过学生课后作业完成，主要是习题环节。

六、实践环节设计

本单元针对每一种排序算法都提供了 C 语言的源程序，供读者参考。在项目实践中设计了 5 个项目，每个项目给出了设计的思想和实现的方法，读者可以根据给出的思路自行编写程序，从而进一步加深理解。

七、目标达成度检验（教学效果测评）

本单元将通过习题进行效果测评。

6.1　项目导学

你有没有在杂乱无章的电话簿中花很长时间寻找一个电话号码的经历？试想一下，若想在图书馆中找到一本没有编号的图书，是不是要花费很长时间？如果把电话簿中好友的姓名按照拼音或笔画排序，在图书馆中，把每本书按类别分类排序，查找起来就方便得多了。这些都是在日常生活中排序的例子。

本章，我们将学习排序的基本知识。按照涉及的存储器不同，对记录进行分类所采用的方法也不同，可将排序分为两类：一类是内部排序（internal sorting），指排序的过程中，记录全部存放在计算机内存中，并在内存中调整记录的位置进行排序；另一类是外部排序（external sorting），指待排序的记录数量很大，以至内存一次不能容纳全部记录时，将记录的主要部分存在外存储器中，借助计算机的内存储器，调整外存储器上的记录的位置进行排序。

6.2　基本概念

1.关键字（key）

记录（排序中经常把数据元素称为记录）中的某一个可以用来标识一个数据元素（记录）的数据项，被称为关键字项，该数据项的值称为关键字。

关键字可以作为排序运算的依据，选取哪一个数据项作为关键字，应根据具体情况而定。例如考试成绩统计中，一个考生的记录包括：考号、姓名、英语成绩、数学成绩、语文成绩、政治成绩、历史成绩和总分等数据项。若要快速查找某一个考生的成绩，应该选取“考号”作为关键字进行排序，因为考号可以唯一标识一个考生的记录。如果想按考生的总分排名次则应把“总分”作为主关键字对考试成绩表进行排序。

2.排序（Sorting）

是将一组记录按照记录中某个关键字进行有序（递增或递减）排列的过程。

设文件中有 n 个记录$\{r_1, r_2, \cdots, r_n\}$，其关键字分别为$\{k_1, k_2, \cdots, k_n\}$，通过排序可以重新构造一种排列$\{r_{j1}, r_{j2}, \cdots, r_{jn}\}$，使其关键字呈现如下关系：$k_{j1} \leq k_{j2} \leq \cdots \leq k_{jn}$

其中 $j_1, j_2, \cdots, j_n$ 属于集合$\{1, 2, \cdots, n\}$。也就是说，排序是把一组记录按关键字值递增（或递减）的次序重新排列，使它变成一个按关键字值大小有序的序列。需要注意的是“≤”号可以理解为一种关系符号（或大于号，或小于号），但在一个算法中只使用一种含义。

东软电子出版社

3.稳定性

如果待排序的文件中,存在多个关键字相同的记录,例如:在(r_1,r_2,…,r_n)中,有 $r_i = r_j$。排序前(…,r_i,…,r_j,…),而排序后,这些记录的相对次序保持不变,即(…,r_i,…,r_j,…),则称这种排序方法是稳定的,否则(…,r_j,…,r_i,…)为不稳定的。

4.排序的性能指标

排序是数据处理中经常遇到的一种重要运算,而要在多种排序算法中选择合适的排序算法,就要分析算法的时间复杂度。由于排序的过程就是对记录值进行比较和记录移动的过程,因此在分析时间复杂度时通常只考虑记录值的比较次数和记录的移动次数,即以记录值的比较和记录移动为标准进行操作。一种排序方法的排序过程在最坏情况下所进行的比较和移动次数越少,则说明该排序方法的时间复杂度就越好,否则就越坏。除了分析排序算法的时间复杂度外,有时还必须分析算法的空间复杂度以及结构的复杂性等,因为这些也是衡量一个排序算法好坏的重要指标。

6.3　插入排序

插入排序类似于玩纸牌时整理手中纸牌的过程:按照牌面数字的大小,将纸牌插入到前面已经排序的序列的适当位置,直到全部的纸牌插入完毕。

根据查找插入位置的方法不同,插入排序的方法也有多种,本章介绍两种最简单的插入排序:直接插入排序和希尔排序。

6.3.1　直接插入排序

直接插入排序(straight insertion sort)是一种最简单的排序方法。该排序的基本思想是:逐个处理待排序序列中的每一个记录,将它和前面已经排好序的子序列中的记录进行比较,确定它插入的位置,并将它插入到子序列中。直到整个记录序列有序为止。

1.具体算法

(1)首先,把第 1 个记录看成是已经排好序的子序,这个子序中只有一个记录。

(2)从第 2 个记录到最后一个记录,依次将记录和前面子序中的记录比较,确定记录插入的位置。

(3)将记录插入,并将子序中记录个数加 1,直到所有的记录都插入完为止。

2.提出问题

假设一组待排序记录的关键字为:9　10　38　25　13　2　,给出直接插入排序执行过程。

3.分析问题

一个记录的序列总是有序的,因此,对含有 n 个记录的序列,可从第二个记录开始直到第 n 个记录,逐个向有序列中进行插入操作,从而得到 n 个记录按关键字有序的序列。一个有 6 个记录的无序序列需要经过 5 趟排序。每进行一趟排序就是取待排序的记录,依次和已排序的序列中的关键字值作比较,找到合适的插入位置插入。如图 6.1 所示。

i=1 时,	初始关键字为:	[9]	10	38	25	13	2
i=2 时,	第一趟排序后:	[9	10]	38	25	13	2
i=3 时,	第二趟排序后:	[9	10	38]	25	13	2
i=4 时,	第三趟排序后:	[9	10	25	38]	13	2
i=5 时,	第四趟排序后:	[9	10	13	25	38]	2
i=6 时,	第五趟排序后:	[2	9	10	13	25	38]

图 6.1　直接插入排序过程

东软电子出版社

4.解决问题

```
/*  ================Program Description================  */
/* 程序名称:InsertSort.cpp                                */
/* 程序目的:直接插入排序                                  */
/*  Written By Cheng Zhuo                                 */
/*  ================================================  */
/*  # include <iostream.h>  */
# include <conio.h>
# define MAXSIZE 20
# define MAX_LENGTH 100
typedef int RedType;

typedef struct                          /* 定义 SqList */
{   RedType    r[MAXSIZE+1];
    int length;
}SqList;

void InsertSort(SqList *L)              /* InsertSort()子函数 */
{  int i,j;
   for(i=2;i<=L->length;++i)
       if(L->r[i]<L->r[i-1])
       {    L->r[0]=L->r[i];            /* 复制为哨兵 */
            for(j=i-1;L->r[0]<L->r[j];--j)
                L->r[j+1]=L->r[j];      /* 记录后移 */
            L->r[j+1]=L->r[0];          /* 插入到正确的位置 */
       }
}                                       /* InsertSort()结束 */

void main()                             /* main()函数 */
{  int i;
   SqList L;
 printf("InsertSort.cpp \n");
   printf("=================== \n");
   printf("Please input the length of SqList (eg, 5): ");
 scanf("%d",&L.length);                 /* 输入线性表长 */

   for(i=1;i<=L.length;++i)
   {  printf("Please input the %d th integer (eg,58): ",i);
      scanf("%d",&L.r[i]);              /* 输入线性表元素,其中 L.r[0]是哨兵 */
   }
   printf("The disordered : ");        /* 输出未排序之前的线性表 */
   for(i=1;i<=L.length;i++)
       printf("%d  ", L.r[i]);
   InsertSort(&L);                      /* 调用 InsertSort()    */
   printf("\n");
   printf("The ordered    : ");        /* 输出排序之后的线性表 */
   for(i=1;i<=L.length;i++)
    printf("%d  ",L.r[i]);
```

东软电子出版社

```
48      getch();
49  }                                          /*main() end*/
```

运行结果:

```
InsertSort.cpp
================
Please input the length of SqList (eg, 5): 6
Please input the 1 th integer (eg,58): 9
Please input the 2 th integer (eg,58): 10
Please input the 3 th integer (eg,58): 38
Please input the 4 th integer (eg,58): 25
Please input the 5 th integer (eg,58): 13
Please input the 6 th integer (eg,58): 2
The disordered : 9  10  38  25  13  2
The ordered    : 2  9  10  13  25  38
```

5.算法分析

(1)空间效率:仅用了一个辅助单元。

(2)时间效率:向有序序列中逐个插入记录的操作,进行了 n-1 趟,每趟操作分为比较关键字值和移动记录,而比较的次数和移动记录的次数取决于待排序列的初始排列。

(3)最好情况(关键字在记录序列中顺序有序)下:即待排序列已按关键字有序,每趟操作只需 1 次比较和 0 次移动。

- 总比较次数=n-1 次
- 总移动次数=0 次

(4)最坏情况(关键字在记录序列中逆序有序)下:即第 j 趟操作,插入记录需要同前面的 j 个记录进行 j 次关键字比较,移动记录的次数为 j 次。

- 总比较次数= $\sum_{j=1}^{n-1} j = \frac{1}{2}n(n-1)$
- 总移动次数= $\sum_{j=1}^{n-1} j = \frac{1}{2}n(n-1)$

(5)平均情况下:即第 j 趟操作,插入记录大约同前面的 j/2 个记录进行关键字比较,移动记录的次数为 j/2 次。

- 总比较次数= $\sum_{j=1}^{n-1} \frac{j}{2} = \frac{1}{4}n(n-1) \approx \frac{1}{4}n^2$
- 总移动次数= $\sum_{j=1}^{n-1} \frac{j}{2} = \frac{1}{4}n(n-1) + 2(n-1) \approx \frac{1}{4}n^2$

由此,直接插入排序的时间复杂度为 $O(n^2)$,是一个稳定的排序方法。

6.3.2 希尔排序

希尔(Shell)排序又称缩小增量法,是对直接插入排序的一种改进。

1.具体算法

先取定一个正整数 $d_1 < n$,把全部记录分成 d_1 个组,所有距离为 d_1 的倍数的记录放在一组中,在各组内进行直接插入排序;然后取 $d_2 < d_1$,重复上述分组和排序工作,直至取 $d_i = 1$,即所有记录放在一个组中排序为止。希尔排序中增量 d_1 有多种取法,我们选取 $d_1 = n/2\ (8)_{10} = (1000)2$,$d_{i+1} = d_i/2$。

2.提出问题

已知一组记录的关键字初值如下:70 81 59 21 92 34 13 76 给出希尔排序的执

东软电子出版社

行过程。

3.分析问题

这是一个有八个记录的序列，n＝8。第一趟分组取 $d_1 = n/2 = 4$，所以把第一个记录和第五个记录比较；记录 2 和 6 比较，记录，记录 3 和 7 比较，记录 4 和 8 比较。正序不变。如果是逆序，则交换两个记录。第二趟 $d_2 = d_1/2 = 2$，以此类推。如图 6.2 所示。

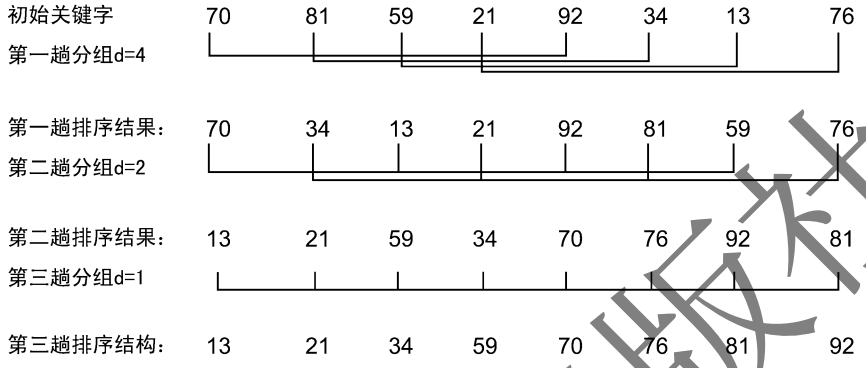


图 6.2　希尔排序过程

4.解决问题

```
/*  ============Program Description=============================  */
/* 程序名称:Shellinert.cpp                                          */
/* 程序目的:希尔排序                                                */
/*  Written By Cheng Zhuo                                          */
/*  ============================================================  */

# include <conio.h>
# define MAXSIZE 20
# define OK 1
# define ERROR 0
typedef int RedType;

typedef struct                          /* 定义 SqList */
{    RedType    r[MAXSIZE+1];
     int length;
}SqList;

void Shellinsert(SqList  *L,int dk)      /* Shellinsert()子函数 */
/* 对顺序表 L 作一趟希尔插入排序.排序中,记录增量为 dk,L.r[0]是暂存元素 */
{  int i,j;
   for(i=dk+1;i<=L->length;++i)
       if(L->r[i]<L->r[i-dk])          /* 需要将 L.r[i]插入到有序增量子表 */
       {    L->r[0]=L->r[i];              /* 暂存在 L.r[0] */
            for(j=i-dk;j>0&&(L->r[0]<L->r[j]);j-=dk)
                L->r[j+dk]=L->r[j];     /* 记录后移  */
            L->r[j+dk]=L->r[0];         /* 插入到正确的位置 */
       }
}

```

东软电子出版社

```
void main()                                  /*main()函数*/
{  int i,dk=5;
   SqList L;/*={{0,49,38,65,97,76,13,27,49,55,4},10};*/
  printf("Shellinsert.cpp \n");
   printf("================ \n");
   printf("Please input the length of SqList (eg, 5):  ");
   scanf("%d",  &L.length);                          /*输入线性表长*/
   printf("\n");
   for(i=1;i<=L.length;++i)
   {   printf("Please input the  %d th integer (eg,58): ", i);
       scanf("%d", &L.r[i]);              /*输入线性表元素,其中L.r[0]是哨兵*/
   }
   printf("The disordered : ");
   for(i=1;i<=L.length;i++)
       printf("%d ",L.r[i]);
   Shellinsert(&L,dk);                        /*调用 Shellinsert()*/
   printf("\n");
   printf("The once ShellSorted sorted: ");
   for(i=1;i<=L.length;i++)
     printf("%d ",L.r[i]);
   printf("\n");
   getch();
} /*main() end */运算结果:
Shellinsert.cpp
================
Please input the length of SqList (eg, 5):  8

Please input the 1 th integer (eg,58): 70
Please input the 2 th integer (eg,58): 81
Please input the 3 th integer (eg,58): 59
Please input the 4 th integer (eg,58): 21
Please input the 5 th integer (eg,58): 92
Please input the 6 th integer (eg,58): 34
Please input the 7 th integer (eg,58): 13
Please input the 8 th integer (eg,58): 76
The disordered : 70 81 59 21 92 34 13 76
The once ShellSorted sorted: 34 13 59 21 92 70 81 76
```

5.算法分析

希尔排序的性能分析是一个比较复杂的问题,因为它的运算时间取决于“增量”序列的函数,目前对增量的选取无一定论。希尔排序的速度一般要比直接插入排序快,但它是不稳定的。

帮助理解

希尔排序是在直接插入排序的基础上进行的改进,改进的关键点是:如何尽快地找到插入位置。选择一种合适的方法找到插入的位置,当记录的个数很多时,可大大减少为寻找插入点而进行的比较次数。

东软电子出版社

6.4　交换排序

所谓交换排序，就是根据记录集中两个记录的关键字值的比较结果来对换这两个记录在序列中的位置。交换排序的基本思想是：将键值较大的记录向记录集的一端移动，键值较小的记录向记录集的另一端移动。

6.4.1　冒泡排序

冒泡排序法(bubble sort)，是最简单的交换排序方法。它和气泡从水中往上冒的情况有些类似。

冒泡排序的基本思想是：将当前待排序的记录，从头到尾依次对相邻的两个记录进行比较，若为“逆序”则将两个记录交换。将序列照此方法从头到尾处理一遍称作一趟冒泡排序，这样是将键值最大的记录交换到最后的位置。然后，对前面未排好序的记录重复上述冒泡排序过程。若某一趟排序过程中没有任何记录发生交换，则说明记录排序次序已经符合要求，排序过程结束。对含 n 个记录的线性表进行排序最多执行 n－1 趟冒泡排序。

1.具体算法

(1)将第 n 个记录的关键字值和第 n－1 个记录的关键字值进行比较，若为逆序则将两个记录进行交换，若为正序则保持原序；

(2)将第 n－1 个记录的关键字和第 n－2 个记录的关键字进行比较，重复上面过程；

(3)上述(1)和(2)的排序过程称做第一趟冒泡排序，其结果是将关键字值最小的记录排在第一个位置上。

(4)以同样的方法进行第二趟排序，从第 n 个记录开始至第 2 个记录进行比较和交换，其结果是将次小的记录放在第二个位置上。

(5)依此类推，设有 n 个关键字，需要经过 n－1 趟比较和交换，使得 n 个记录的关键字从小到大排好序。

2.提出问题

已知一组记录的关键字值初始排列如下：39　17　14　10　25　11　20　12 给出冒泡排序的执行过程。

3.分析问题

利用冒泡算法，首先将第一个记录 39 和第 2 个记录 17 比较，如为逆序，则交换，反之，不变。然后将第 2 个记录和第 3 个记录比较，若为逆序，则交换。一直比较到第 n－1 个记录和第 n 个记录。这样完成第一趟排序。这时第 n 个记录就是整个序列中关键字最大的记录。第二趟排序的方法和前面的一样，从第 1 个记录比较到第 n－1 个记录。取出第 2 大的记录排在第 n－1 的位置上。依此类推，最多进行 n－1 趟排序。

东软电子出版社

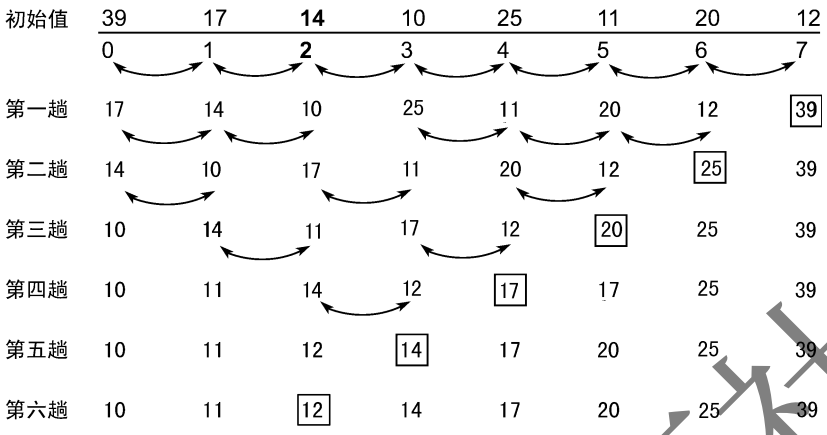


图 6.3　冒泡排序过程

冒泡排序提前结束的情况是:当在某趟排序过程中,没有记录需要交换时,表示该序列已经为有序序列,则排序过程停止。

4.解决问题

```
/*  ================Program Description=============  */
/*程序名称:BubbleSort.cpp                              */
/*程序目的:冒泡排序                                    */
/*  Written By Cheng Zhuo                              */
/*  ================================================  */
#  include <conio.h>
#  include <stdio.h>
#  define MAXSIZE 20
#  define MAX_LENGTH 100
typedef int RedType;
typedef struct          /*定义 SqList */
{   RedType    r[MAXSIZE+1];
    int length;
}SqList;

void BubbleSort(SqList *L)
{int i,j,temp,flag;
for(i=0;i<L->length;++i)
  {flag=0;
      for(j=0;j<L->length-i;j++)
          {if(L->r[j+1]<L->r[j])
          {    temp=L->r[j+1];          /*交换 L.r[j+1]和 L.r[j]*/
               L->r[j+1]=L->r[j];
               L->r[j]=temp;
               flag=1;
          }
           }
      if(flag==0)  break;
    }
```

东软电子出版社

```
}
void main()                                  /*main()函数*/
{  int i;
   SqList L;
   printf("BubbleSort.cpp \n");
   printf("==========\n");
   printf("Please input the length of SqList (eg,6): ");
   scanf("%d" ,&L.length);
   for(i=1;i<=L.length;++i)
   {   printf("Please input the %d th element of SqList (eg,23): ",i);
       scanf("%d",&L.r[i]);
   }
   printf("\n");
   printf("The disordered : ");          /*未排序的线性表*/
   for(i=1;i<=L.length;i++)
       printf("%d  ", L.r[i]);

   BubbleSort(&L);                          /*调用 BubbleSort()*/
   printf("\n");
   printf("The ordered    : ");
   for(i=1;i<=L.length;i++)                 /*排序后的线性表*/
          printf("%d  ",L.r[i]);
   getch();
}                                        /*main() end*/运行结果:
BubbleSort.cpp
==========
Please input the length of SqList (eg,6): 8
Please input the 1 th element of SqList (eg,23): 39
Please input the 2 th element of SqList (eg,23): 17
Please input the 3 th element of SqList (eg,23): 14
Please input the 4 th element of SqList (eg,23): 10
Please input the 5 th element of SqList (eg,23): 25
Please input the 6 th element of SqList (eg,23): 11
Please input the 7 th element of SqList (eg,23): 20
Please input the 8 th element of SqList (eg,23): 12

The disordered : 39  17  14  10  25  11  20  12
The ordered    : 10  11  12  14  17  20  25  39
```

5.算法分析

冒泡排序的效率与排序前记录的次序有关,若排序前记录为正序,则冒泡排序只需要一趟排序,在排序过程中进行 n－1 次关键字的比较,并且不需要移动记录;但是,若排序前记录为逆序,则冒泡排序需要进行 n－1 趟排序,并且需要进行 n(n－1)/2 次关键字的比较,且需要等量级的记录移动。因此,总的时间复杂度为 $O(n^2)$。冒泡排序是稳定的。适用于记录基本有序的场合。

东软电子出版社

6.4.2　快速排序

快速排序又称分区交换排序,它是由冒泡排序改进而得来的。快速排序的基本思想是:在待排序的 n 个记录中任取一个基准记录(一般取第一个记录为基准记录),采用从两头向中间扫描的办法,利用比较和交换,将基准记录移动到最终位置上。即以基准记录为基准,将待排序序列划分成左右两部分,所有比该记录关键字值小的记录放到左边,所有比它大的放到右边,并把该记录排在这两部分的中间,这个过程称为一趟快速排序。然后分别对所划分的两部分重复上述过程,直到每一部分的记录个数是 1 为止。

1.具体算法

一次快速排序的算法过程如下:

设 1≤p<q≤n,r[p],r[p+1],…,r[q]为待排序列。

(1)low=p;high=q;//设置两个搜索指针,low 是向后搜索指针,high 是向前搜索指针。

r[0]=r[low];　　//取第一个记录为支点记录,low 位置暂设为支点空位

(2)若 low=high,支点空位确定,即为 low。

r[low]=r[0];　　//填入支点记录,一次划分结束

否则,low<high,搜索需要交换的记录,并交换之。

(3)若 low<high 且 r[high].key≥r[0].key,//从 high 所指位置向前搜索,至多到 low+1 位置。

high=high−1;转(3)　　//寻找 r[high].key<r[0].key

r[low]=r[high];　　//找到 r[high].key<r[0].key,设置 high 为新支点位置,

//小于支点记录关键码的记录前移。

(4)若 low<high 且 r[low].key<r[0].key　//从 low 所指位置向后搜索,至多到 high−1 位置。

low=low+1;转(4)　　//寻找 r[low].key≥r[0].key

r[high]=r[low];　　//找到 r[low].key≥r[0].key,设置 low 为新支点位置,大于等于支点记录关键码的记录后移。

转(2)　　//继续寻找支点空位

2.提出问题

假设一组待排序记录的关键字序列为:35　87　66　17　75　58　44　22,写出快速排序的执行过程。

3.分析问题

对其进行一趟快速排序的过程如图 6.4 所示。

东软电子出版社

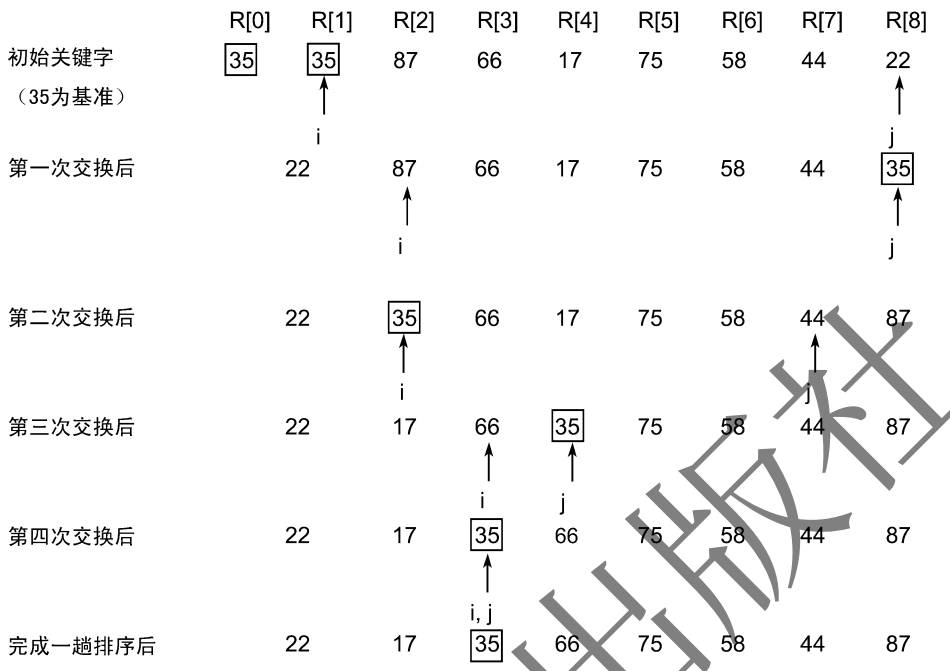


图 6.4　快速排序

上面只是一趟快速排序过程，一趟快速排序后，整个记录序列被关键字值为位 35 的基准记录分成两部分，左边部分的关键字值都小于 35，右边部分的关键字值都大于 35。要使整个记录序列完全排好，还应对其左右两边再进行快速排序。这实际上是一个递归过程。

4.解决问题

```
/*  ==========Program Description==========  */
/*程序名称:QuickSort.cpp                      */
/*程序目的:快速排序                           */
/*  Written By Cheng Zhuo                     */
/*  ========================================*/

#  include <conio.h>
#  define MAXSIZE 20
typedef int RedType;
typedef struct                    /*定义 structure */
{    RedType    r[MAXSIZE+1];
     int length;
}SqList;

int  Partition(SqList *L,int low,int high)     /*Partition()子函数 */
{  int pivotkey;
   L->r[0]=L->r[low];              /*用子表的第一个记录作"枢纽"记录*/
   pivotkey=L->r[low];          /*pivotkey 是枢纽的记录关键字*/
   while(low<high)              /*从表的两端交替地向中间扫描*/
      {  while(low<high&&L->r[high]>=pivotkey)
         --high;
```

东软电子出版社

```
22        L->r[low]=L->r[high];     /* 比"枢纽"小的记录移到底端 */
23        while(low<high&&L->r[low]<=pivotkey)
24           ++low;
25        L->r[high]=L->r[low];     /* 比"枢纽"大的记录移到高端 */
26         }
27      L->r[low]=L->r[0];              /* "枢纽"记录到位 */
28      return (low);             /* 返回"枢纽"位置 */
29  } /* Partition() end */
30
31  void Qsort(SqList *L,int low,int high)     /* Qsort() sub-function */
32  {  int pivotloc;
33     if(low<high)                      /* 长度大于 1 */
34     {  pivotloc=Partition(L,low,high);
35        Qsort(L,low,pivotloc-1);
36        Qsort(L,pivotloc+1,high);
37     }
38  }
39
40  void QuickSort(SqList *L)         /* QuickSort()子函数 */
41  {  Qsort(L,1,L->length);             /* 调用 Qsort() */
42  }
43
44  void main()                       /* main()函数 */
45  {  int i;
46     SqList L;
47    printf("QuickSort.cpp\n");
48    printf("=============\n");
49    printf("Please input the length of SqList (eg,5): ");
50     scanf("%d",&L.length);
51     for(i=1;i<=L.length;++i)
52     {   printf("Please input the %d th element of SqList (eg,58):",i );
53         scanf("%d",&L.r[i]);
54     }
55     printf("\n");
56     printf("The disordered : ");          /* 排序之前的线性表 */
57     for(i=1;i<=L.length;i++)
58        printf("%d  ", L.r[i]);
59     QuickSort(&L);                 /* 调用 QuickSort() */
60     printf("\n");
61     printf("The sorted     : ");         /* 排序之后的线性表 */
62     for(i=1;i<=L.length;i++)
63       printf("%d  ",L.r[i]);
64     getch();
65  }                           /* main() end */
```

运行结果:

```
QuickSort.cpp
=============
Please input the length of SqList (eg,5): 8
```

东软电子出版社

```
Please input the 1 th element of SqList (eg,58):35
Please input the 2 th element of SqList (eg,58):87
Please input the 3 th element of SqList (eg,58):66
Please input the 4 th element of SqList (eg,58):17
Please input the 5 th element of SqList (eg,58):75
Please input the 6 th element of SqList (eg,58):58
Please input the 7 th element of SqList (eg,58):44
Please input the 8 th element of SqList (eg,58):22

The disordered : 35  87  66  17  75  58  44  22
The sorted     : 17  22  35  44  58  66  75  87
```

5.算法分析

(1)空间效率:快速排序是递归的,每层递归调用时的指针和参数均要用栈来存放,递归调用层次数与上述二叉树的深度一致。因而,存储开销在理想情况下为 $O(\log_2 n)$,即树的高度;在最坏情况下,即二叉树是一个单链,为 $O(n)$。

(2)时间效率:在 n 个记录的待排序列中,一次划分需要约 n 次关键码比较,时效为 $O(n)$,若设 $T(n)$为对 n 个记录的待排序列进行快速排序所需时间,理想情况下,每次划分,正好将待排序列分成两个等长的子序列,则:

$$
\begin{aligned}
T(n) &\leqslant cn+2T(n/2) && c\text{ 是一个常数}\\
&\leqslant cn+2(cn/2+2T(n/4))=2cn+4T(n/4)\\
&\leqslant 2cn+4(cn/4+T(n/8))=3cn+8T(n/8)\\
&\cdots\cdots\\
&\leqslant cn\log_2 n+nT(1)=O(n\log_2 n)
\end{aligned}
$$

在最坏情况下:即每次划分只得到一个子序列,时间复杂度为 $O(n^2)$。

快速排序是通常被认为在同数量级($O(n\log_2 n)$)的排序方法中平均性能最好的。但若初始序列按关键码有序或基本有序时,快速排序反而蜕化为冒泡排序。为改进之,通常以“三者取中法”来选取支点记录,即将排序区间的两个端点与中点三个记录关键码居中的调整为支点记录。快速排序是一个不稳定的排序方法。

6.5　选择排序

选择排序也是一种常用的、简单的、最为广大读者熟悉的一种排序方法。

选择排序思想是:不断地从待排序的记录中选择关键字值最小(或最大)的记录,依次排放到已排好序的序列的后面。这样,由选取记录的顺序,便得到按关键字有序的序列。

6.5.1　直接选择排序

1.基本思想

设 n 个待排序的记录存放在 r[1..n]中,对 n 个待排序记录进行 n－1 趟排序:第一趟扫描选出 n 个记录中关键字最小的记录,并与 r[1]记录交换位置。第二趟扫描选出余下的 n－1 个记录中关键字值最小的记录,并与 r[2]记录交换位置。依此类推,直到第 n－1 趟扫描结束,所有记录有序为止。

2.提出问题

已知一组记录的关键字值初始排列如下:23　15　36　3　48　85　8　53,给出利用直接选择排序的执行过程。

东软电子出版社

3.分析问题

如图 6.5 所示。

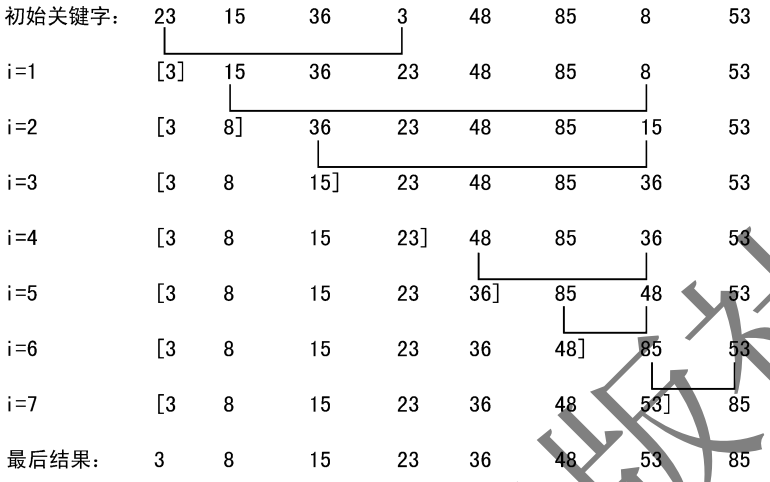


图 6.5 直接选择排序过程

4.解决问题

```
/* ================Program Description================ */
/* 程序名称:SelectSort.cpp                              */
/* 程序目的:直接选择排序                                */
/*  Written By Cheng Zhuo                               */
/* ================================================== */
# include <conio.h>
# define MAXSIZE 20
typedef int RedType;

typedef struct              /* 定义 SqList */
{    RedType    r[MAXSIZE+1];
     int length;
}SqList;

void SelectSort(SqList  *L)          /* SelectSort()子函数 */
{    int i,j,k,temp;
     for(i=1;i<L->length;++i)     /* 选择第 i 个小的记录,并交换到位 */
     {   k=i;
         for(j=i+1;j<=L->length;++j) /* 在 L.r[i..length]中选择值最小的记录 */
              if(L->r[j]<L->r[k])
                  k=j;
          if(i! =k)
          {   temp=L->r[i];          /* 与第 i 个记录交换 */
              L->r[i]=L->r[k];
              L->r[k]=temp;
          }
     }
}/* SelectSort() end */
```

东软电子出版社

```
void main()              /*main()函数*/
{  int i;
   SqList L;
   printf("SelectSort.cpp \n");
   printf("==============\n");
   printf("Please input the length of SqList (eg,5): ");
   scanf("%d",&L.length);
   for(i=1;i<=L.length;++i)
   {   printf("Please input the %d th element of SqList (eg,58): ",i);
      scanf("%d",&L.r[i]);
   }
   printf("\n");
   printf("The disordered : ");        /*未排序的线性表*/
   for(i=1;i<=L.length;i++)
       printf("%d  ", L.r[i]);
   SelectSort(&L);                      /*调用 SelectSort()*/
    printf("\n");
   printf("The sorted     : ");        /*排序后的线性表*/
   for(i=1;i<=L.length;i++)
     printf("%d  ",L.r[i]);
   getch();
} /*main() end*/运行结果:

SelectSort.cpp
==============
Please input the length of SqList (eg,5): 8
Please input the 1 th element of SqList (eg,58): 23
Please input the 2 th element of SqList (eg,58): 15
Please input the 3 th element of SqList (eg,58): 30
Please input the 4 th element of SqList (eg,58): 3
Please input the 5 th element of SqList (eg,58): 48
Please input the 6 th element of SqList (eg,58): 85
Please input the 7 th element of SqList (eg,58): 8
Please input the 8 th element of SqList (eg,58): 53

The disordered : 23  15  30  3  48  85  8  53
The sorted     : 3  8  15  23  30  48  53  85
```

5.算法分析

从运算时间上看,在第一趟排序中关键字的比较次数为 n−1 次,在第二趟排序中关键字的比较次数为 n−2 次。依此类推,在第 i 趟排序中关键字的比较次数为 n−i 次,所以总比较次数为:

$$\text{比较次数} = \sum_{i=1}^{n-1}(n-i) = n(n-1)/2$$

在一趟排序中,记录的移动次数最好为 0 次,最坏为 n*(n−1)/2 次。所以,该排序的时间复杂度为 $O(n^2)$。直接选择排序算法是不稳定的。

6.5.2　堆排序

为了避免在直接选择排序中进行重复比较的缺点而引入了树形选择排序方法。它的基本

东软电子出版社

思想是:首先对 n 个记录的键值进行两两比较,然后在其中的[n/2]个较小者之间再进行两两比较,如此重复,直至选出最小键值的记录为止。

采用一棵树来表示这一排序过程,其方法和选择排序类似,所不同的是,以前的比较结果,通过树记录下来了,使得后面的选择可以在它们的基础上进行。但是,如果专门设立树,则造成存储浪费。堆排序是一种巧妙的树形选择排序,它不需要专门设立树。

先来看几个概念:

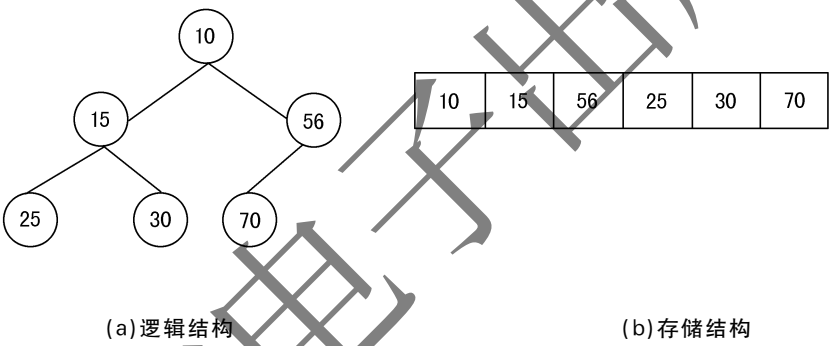
(1)堆排序定义:n 个关键字序列 $K1,K2,\cdots,Kn$ 称为堆,当且仅当该序列满足如下性质(简称为堆性质):

$ki \leqslant K2i$ 且 $ki \leqslant K2i+1$ 或 $Ki \geqslant K2i$ 且 $ki \geqslant K2i+1(1 \leqslant i \leqslant [n/2])$

若将此序列所存储的向量 $R[1..n]$ 看做是一棵完全二叉树的存储结构,则堆实质上是满足如下性质的完全二叉树:树中任一非叶结点的关键字均不大于(或不小于)其左右孩子(若存在)结点的关键字。

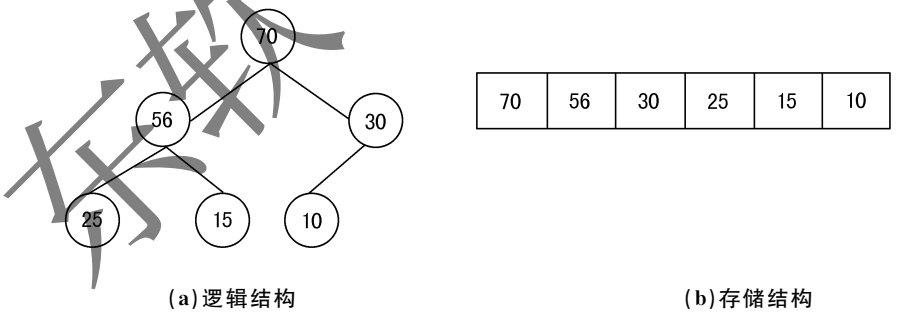
利用数组存储一个堆,那么堆对应着这样的一棵完全二叉树,它所有非叶结点的值均不大于(或不小于)其子结点的值,根结点的值是最小(或最大)的。

【例】关键字序列(10,15,56,25,30,70)和(70,56,30,25,15,10)分别满足堆性质(1)和(2),故它们均是堆,其对应的完全二叉树分别如小根堆示例和大根堆示例所示。



(a)逻辑结构　　(b)存储结构

图 6.6　小根堆示例



(a)逻辑结构　　(b)存储结构

图 6.7　大根堆示例

(2)大根堆和小根堆:根结点(亦称为堆顶)的关键字是堆里所有结点关键字中最小者的堆称为小根堆;根结点(亦称为堆顶)的关键字是堆里所有结点关键字中最大者的堆,称为大根堆。

1.基本思想

设有 n 个元素,将其按关键字排序。首先将这 n 个元素按关键字建成堆,将堆顶元素输出,得到 n 个元素中关键码最小(或最大)的元素。然后,对剩下的 $n-1$ 个元素建成堆,输出堆顶元素,得到 n 个元素中关键码次小(或次大)的元素。如此反复,便得到一个按关键码有序的序列。称这个过程为堆排序。

东软电子出版社

因此，实现堆排序需解决两个问题：①如何将 n 个元素的序列按关键字建成堆——构造堆；②输出堆顶元素后，怎样调整剩余 $n-1$ 个元素，使其按关键字成为一个新堆——调整堆。

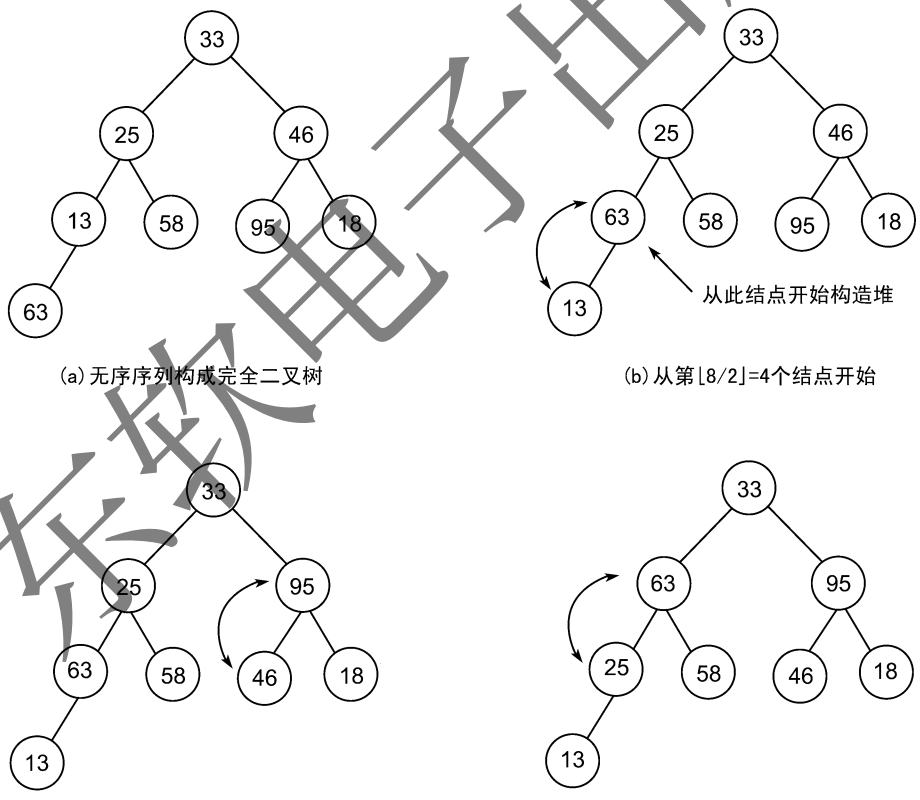
首先考虑第二个问题，调整堆；然后再考虑第一个问题，构造堆。

(1)调整堆：设有 m 个元素的堆，输出堆顶元素后，剩下 m－1 个元素。将堆底元素送入堆顶，堆被破坏，其原因仅是根结点不满足堆的性质。将根结点与左、右子结点中较小(或小大)的进行交换。若与左子结点交换，则左子树堆被破坏，且仅左子树的根结点不满足堆的性质；若与右子结点交换，则右子树堆被破坏，且仅右子树的根结点不满足堆的性质。继续对不满足堆性质的子树重复上述交换操作，直到叶子结点，则堆被建成。称这个自根结点到叶子结点的调整过程为筛选。

(2)构造堆：从一个无序序列构造成堆的过程就是一个反复“筛选”的过程。若将此序列看成是一个完全二叉树，则最后一个非终端结点是第 $\lfloor n/2 \rfloor$ 个元素，由此“筛选”只需从第 $\lfloor n/2 \rfloor$ 个元素开始，依次将第 $\lfloor n/2 \rfloor$ 个结点，第 $\lfloor n/2 \rfloor-1$ 个结点，…，第 1 个结点按照堆的定义逐一加到他们的子结点上，直到建成一个完全的堆。

2.提出问题

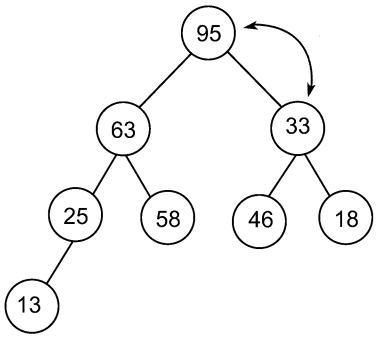
有 8 个待排序记录的关键字序列 33　25　46　13　58　95　18　63，构造堆的过程如图 6.8 所示。



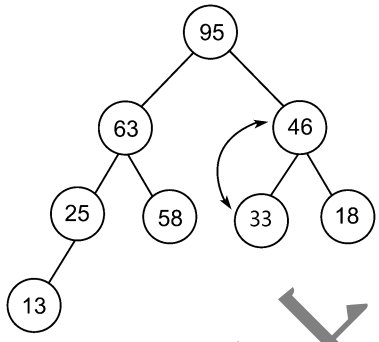
(a)无序序列构成完全二叉树　(b)从第⌊8/2⌋=4个结点开始

(c)46被筛选之后的状态　(d)25被筛选之后的状态

东软电子出版社



(e)33被筛选的中间状态



(f)33被筛选之后建成堆

图 6.8 构造堆过程

脚下留心

在程序中要注意数组下标和树结点编号间的转换。树结点编号是从1起的连续自然数,而数组下标为从0起的连续自然数,故需要换算。

帮助理解

堆排序的基本方法可描述为:

(1)建堆:将原始数据调整为堆。

(2)输出:输出堆顶元素。

(3)堆调整:将堆结构中剩余元素重新调整为堆。

(4)判断:若全部元素均已输出,则结束,输出次序为排序次序;否则转(2)。

(5)利用大根堆实现升序排序,小根堆实现降序排序。

3.解决问题

```
/*  ================Program Description================  */
/* 程序名称:HeapSort.cpp                                  */
/* 程序目的:堆排序                                        */
/*  Written By Cheng Zhuo                                 */
/* ======================================================*/
# include <conio.h>
# define MAXSIZE 20
typedef int RedType;

typedef struct                          /* 定义 SqList */
{    RedType    r[MAXSIZE+1];
     int length;
}SqList;
typedef SqList HeapType;

void HeapAdjust(HeapType *H,int s,int m) /* HeapAdjust()子函数 */
{  int temp,j;
   temp=H->r[s];
   for(j=2*s;j<=m;j*=2)
```

东软电子出版社

```
20      {  if(j<m && (H->r[j]<H->r[j+1]))
21           ++j;
22         if(! (temp<H->r[j]))
23           break;
24         H->r[s]=H->r[j];
25         s=j;
26      }
27      H->r[s]=temp;
28  } /* HeapAdjust() end */
29
30  void HeapSort(HeapType  *H)            /* HeapSort()子函数  */
31  {  int i,temp;
32      for(i=H->length/2;i>=1;--i)
33          HeapAdjust(H,i,H->length);         /* 把 H.r[1..H.length]构造成堆 */
34      for(i=H->length;i>=1;--i)
35      {  temp=H->r[1];                       /* 把堆顶记录和未经排序的子序列 */
36         H->r[1]=H->r[i];                     /* H.r[1..i]中的最后一个记录相交换 */
37         H->r[i]=temp;
38         HeapAdjust(H,1,i-1);              /* 将 H.r[1,i-1]重新调整为堆 */
39      }
40  }/* HeapSort() end  */
41
42  void main()                              /* main()函数*/
43  {  int i;
44      HeapType H;
45      printf("HeapSort.cpp\n");
46      printf("=============\n");
47      printf("Please input the length of SqList (eg,5): ");
48      scanf("%d",&H.length);
49      for(i=1;i<=H.length;++i)
50      {   printf("Please input the  %d th element of SqList (eg,58): ",i);
51          scanf("%d",&H.r[i]);
52      }
53      printf("\n");
54      printf("The disordered : ");     /* 未排序之前  */
55      for(i=1;i<=H.length;i++)
56          printf("%d  ",H.r[i]);
57      HeapSort(&H);                            /* 调用 HeapSort() */
58     printf("\n");
59     printf("The sorted     : ");     /* 排序之后 */
60      for(i=1;i<=H.length;i++)
61          printf("%d  ",H.r[i]);
62      getch();
63  } /* main() end */运行结果：
HeapSort.cpp
=============
Please input the length of SqList (eg,5): 8
Please input the 1 th element of SqList (eg,58): 33
```

东软电子出版社

```
Please input the 2 th element of SqList (eg,58): 25
Please input the 3 th element of SqList (eg,58): 46
Please input the 4 th element of SqList (eg,58): 13
Please input the 5 th element of SqList (eg,58): 58
Please input the 6 th element of SqList (eg,58): 95
Please input the 7 th element of SqList (eg,58): 18
Please input the 8 th element of SqList (eg,58): 63

The disordered : 33  25  46  13  58  95  18  63
The sorted     : 13  18  25  33  46  58  63  95
```

4.算法分析

堆排序算法的运行时间主要耗费在构造初始堆时进行的反复“筛选”上,对深度为 k 的堆,关键字比较次数至多为 2(k－1)次。所以,在建好堆后,排序过程中的筛选次数不超过$2n\log_2 n$:

$2(\lfloor\log_2(n-1)\rfloor+\lfloor\log_2(n-2)\rfloor+\cdots+\log_2 2\rfloor)<2n\log_2 n$

而建堆时的比较次数不超过 4n 次,因此堆排序最坏情况下,时间复杂度也为 $O(n\log_2 n)$。相对于快速排序来说,这是堆排序最大的优点。另外,堆排序仅需要一个记录大小的辅助存储空间,供交换元素使用。堆排序是不稳定的,不适合用于记录较少的情况。

6.6 归并排序

归并排序是又一类不同的排序方法。它是一种基于合并有序段的排序方法,也就是将若干有序段逐步合并,最后合并为一个有序段的过程。将初始的 n 个待排序记录看成 n 个子序列长度为 1 的有序序列,从第 1 个子序列开始进行两两归并,如此反复,直到得到一个长度为 n 的有序序列为止。

将两个有序序列归并为一个新的有序序列,称为 2-路归并;将三个有序序列归并为一个新的有序序列,称为 3-路归并;将多个有序序列归并为一个新的有序序列,称为多路归并。

1.基本思想

对于有 n 个记录的无序序列进行归并排序:

(1)将 n 个待排序的记录分成只含有一个记录的 n 个有序子序列。

(2)将这 n 个有序子序列依次两两归并,得到$\lfloor n/2\rfloor$个长度为 2 的有序子序列(当 n 为奇数时,有一个长度为 1 的子序列)。

(3)再对它们做两两合并,如此重复,直到得到一个长度为 n 的有序序列为止,归并排序完成。

2.提出问题

对于一个有 n 个待排序记录的关键字序列 23　15　36　3　48　85　8　63 ,请利用 2—路归并排序方法进行排序。

3.分析问题

归并算法的核心问题是将一维数组中前后相邻的两个有序序列归并为一个有序序列。

东软电子出版社

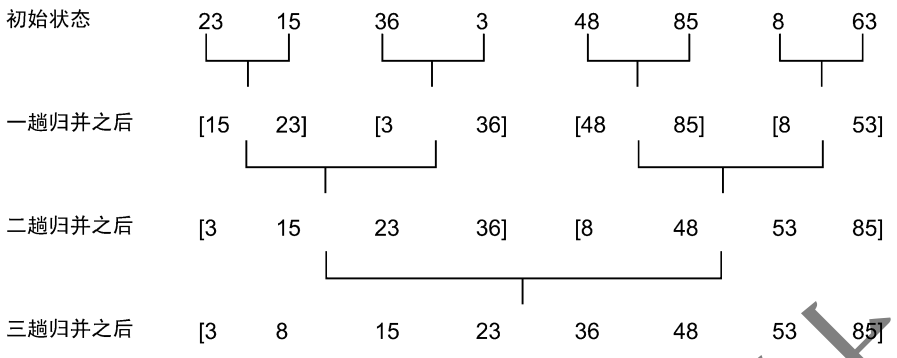


图 6.9　归并排序过程

4.解决问题

```
/*  ==================Program Description==========  */
/* 程序名称:MergeSort.cpp                            */
/* 程序目的:归并排序                                 */
/*  Written By Cheng Zhuo                            */
/*  ================================================*/


#include <conio.h>

#define MAXSIZE 20
#define LENGTH 7
typedef int RedType;

typedef struct                  /*定义 SqList*/
{    RedType    r[MAXSIZE+1];
     int length;
}SqList;
typedef SqList RcdType;

void Merge(RcdType *SR,RcdType *TR,int i,int m,int n)/*Merge()子函数*/
/*将有序的 SR[i..m]和有序的 SR[m+1..n]归并为有序的 TR[i..n]*/
{   int j,k;
    for(j=m+1,k=i;i<=m&&j<=n;++k)     /*将 SR 中的记录由小到大地并入 TR*/
    {   if(SR->r[i]<=SR->r[j])
            TR->r[k]=SR->r[i++];
        else
            TR->r[k]=SR->r[j++];
    }
    while(i<=m)                 /*将剩余的 SR[i..m]复制到 TR*/
       TR->r[k++]=SR->r[i++];
    while(j<=n)                 /*将剩余的 SR[j..n]复制到 TR*/
       TR->r[k++]=SR->r[j++];
}/*end of Merge() function*/

void MSort(RcdType *SR,RcdType *TR1,int s, int t)/*MSort()子函数*/
```

东软电子出版社

```
/*将 SR[s..t]归并排序为 TR1[s..t]*/
{  int m;
   RcdType TR2;
   if(s==t)
     TR1->r[s]=SR->r[t];
   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);    /*递归地将 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]*/
   }/*end of else */
}/*end of MSort() function*/

void MergeSort(SqList *L)    /*MergeSort()函数*/
/*对顺序表 L 作归并排序*/
{
   MSort(L,L,1,L->length);
}/*end of MergeSort()函数*/

void main()                  /*main 函数 */
{  int i;
   SqList L;
   printf("MergeSort.cpp \n");
   printf("=============\n");
   printf("Please input the length of SqList L: <eg,8>:");
   scanf("%d",&L.length);
  /*  printf("Please input the disordered array L.r: ");*/
   for(i=1;i<=L.length;++i)
        {   printf("Please input the %d th element of SqList (eg,58):",i);
            scanf("%d",&L.r[i]);
        }
   printf("\n");
   printf("The disordered : ");        /*未排序的线性表*/
   for(i=1;i<=L.length;i++)
      printf("%d ",L.r[i]);
   printf("\n");
   MergeSort(&L);                        /*调用 MergeSort()*/
   printf("The sorted     : ");        /*排序后的*/
   for(i=1;i<=L.length;i++)
       printf("%d ",L.r[i]);
   getch();
}                                  /*main() end */
```

运行结果:

```
MergeSort.cpp
=============
Please input the length of SqList L: <eg,8>:8
Please input the 1 th element of SqList (eg,58):23
Please input the 2 th element of SqList (eg,58):15
```

东软电子出版社

```
Please input the 3 th element of SqList (eg,58):36
Please input the 4 th element of SqList (eg,58):3
Please input the 5 th element of SqList (eg,58):48
Please input the 6 th element of SqList (eg,58):85
Please input the 7 th element of SqList (eg,58):8
Please input the 8 th element of SqList (eg,58):53

The disordered : 23 15 36 3 48 85 8 53
The sorted     : 3 8 15 23 36 48 53 85
```

5.算法分析

需要一个与表等长的辅助元素数组空间，所以空间复杂度为 O(n)。

对 n 个元素的表，将这 n 个元素看作叶结点，若将两两归并生成的子表看作它们的父结点，则归并过程对应由叶向根生成一棵二叉树的过程。所以归并趟数约等于二叉树的高度－1，即 $\log_2 n$，每趟归并需移动记录 n 次，故时间复杂度为 $O(n\log_2 n)$。

6.7　基数排序

基数排序(radix sorting)是和前面所述的各类排序方法完全不同的一种排序方法。前面几种排序方法主要是通过对关键字的比较和移动来实现排序的，而基数排序不需要进行关键字的比较和记录的移动，它是一种基于多关键字排序的思路而对单逻辑关键字进行排序的一种内部排序方法。

1.多关键字排序

下面通过一个例子来说明多关键字的排序思想：假设扑克牌 52 张牌的次序为：2♣＜3♣＜…＜A♣＜2◆＜3◆＜…＜A◆＜2♠＜3♠＜…＜A♠＜2♥＜3♥＜…＜A♥。每一张牌有两个“关键字”：花色(♣＜◆＜♠＜♥)和面值(2＜3＜…＜A)，而且花色优先于面值，即在比较任意两张牌的大小时，先比较花色，花色相同时再比较面值。

先将扑克牌整理成上面规定的次序，通常采用两种方法：最高位优先(MSD)法和最低位优先(LSD)法。

MSD 法：先对花色排序，将其分为 4 个组，即梅花组、方块组、红心组、黑心组。再对每个组分别按面值进行排序，最后，将 4 个组连接起来即可。

LSD 法：先按 13 个面值给出 13 个编号组(2 号，3 号，…，A 号)，将牌按面值依次放入对应的编号组，分成 13 堆。再按花色给出 4 个编号组(梅花、方块、红心、黑心)，将 2 号组中牌取出分别放入对应花色组，再将 3 号组中牌取出分别放入对应花色组，…，这样，4 个花色组中均按面值有序，然后，将 4 个花色组依次连接起来即可。

基数排序(Radix Sort)：根据组成关键字的每个位的有效值，用“分配”和“收集”的方法进行排序。

2.排序算法

```
1  /* ==================Program Description========== */
2  /* 程序名称:Radixsort.cpp                          */
3  /* 程序目的:基数排序                               */
4  /*  Written By Cheng Zhuo                          */
5  /* ================================================ */
6
7  # include <stdio.h>
```

东软电子出版社

```
 8  # include <conio.h>
 9  # define MAX_NUM_OF_KEY 8
10  # define RD 10
11  # define MAX_SPACE 500
12  # define ERROR -1
13
14  typedef int KeyType;
15  typedef int InfoType;
16  typedef int ArrType[RD];
17
18  typedef struct SLCell
19  {  KeyType keys[MAX_NUM_OF_KEY];
20     InfoType otheritems;
21     int next;
22  }SLCell;
23
24  typedef struct SLList
25  {  SLCell r[MAX_SPACE];
26     int keynum;
27     int recnum;
28  }SLList;
29
30  int Succ(int j)                          /* Succ() function */
31  {/* To get the next function */
32     j++;
33     return (j);
34  }/* end of Succ() function */
35
36  int Ord(int KeyBit)                      /* Ord() function  */
37  {
38      int j;
39      for(j=0;j<=RD-1&&j! =KeyBit;j++);
40      if(j! =KeyBit) return(ERROR);          /* KeyBit OVERFLOW THEN ERROR */
41         else return(j);
42  } /* end of Ord() function */
43
44  void OutExample(SLList *L,int i)          /* OutExample() function  */
45  {
46
47       int temp,k;
48       printf("\nThe %d Collect result is:  ",i);
49  /*      temp=L.r[0].otheritems;  */
50  /*      printf("%d -> ",temp);*/
51       temp=L->r[0].next;
52       printf("%d -> ",L->r[temp].otheritems);
53       for(k=0;k<L->recnum-2;k++)
54       { temp=L->r[temp].next;
55         printf("%d -> ",L->r[temp].otheritems);
```

东软电子出版社

```
56        }
57        printf(" %d",L->r[L->r[temp].next].otheritems);
58        printf("\n");
59
60  }/* end of OutExample() function */
61
62  void Distribute(SLList *L,int i,ArrType f,ArrType e)     /* Distribute() function */
63  {  int j,p;
64     for(j=0;j<RD;j++)
65        f[j]=0;
66     for(p=L->r[0].next;p;p=L->r[p].next)
67     {  j=Ord(L->r[p].keys[i]);                              /* call Ord() */
68        if(! f[j])
69           f[j]=p;
70        else
71           L->r[e[j]].next=p;
72        e[j]=p;
73     }/* end of for */
74  }/* end of Distrubute() function */
75
76  void Collect(SLList *L,int i,ArrType f,ArrType e)        /* Collect() function */
77  {  int j,t;
78     for(j=0;! f[j];j=Succ(j));                              /* Succ() */
79     L->r[0].next=f[j];
80     t=e[j];
81     while(j<RD-1)
82     {  for(j=Succ(j);j<RD-1&&! f[j];j=Succ(j));
83        if(f[j])
84        {  L->r[t].next=f[j];
85           t=e[j];
86        }/* end of if */
87     }/* end of while */
88     L->r[t].next=0;
89     OutExample(L,i);                              /* Add Output Example function here  */
90  }/* end of Collect() function */
91
92  void RadixSort(SLList *L)
93  {  int i;
94     ArrType f,e;
95
96     for(i=0;i<L->recnum;i++)
97     L->r[i].next=i+1;
98     L->r[L->recnum].next=0;
99
100     for(i=0;i<L->keynum;i++)
101     {   Distribute(L,i,f,e);
102         Collect(L,i,f,e);
103     }/* end of for */
```

东软电子出版社

```
104 }/*end of RadixSort() function*/
105
106 void InitExample(SLList *L)
107 {                                   /*Take SLList L for example*/
108     L->keynum=3;                    /*total key number is 3 */
109     L->recnum=7;                    /*total current node is 10 */
110     L->r[1].otheritems=278;
111     L->r[2].otheritems=109;
112     L->r[3].otheritems=163;
113     L->r[4].otheritems=930;
114     L->r[5].otheritems=580;
115     L->r[6].otheritems=184;
116     L->r[7].otheritems=505;
117     printf("The InitExample SLList L is:278->109->163->930->580->184->505 ");
118     printf("\n");
119
120     L->r[1].keys[0]=8;
121     L->r[1].keys[1]=7;
122     L->r[1].keys[2]=2;
123
124     L->r[2].keys[0]=9;
125     L->r[2].keys[1]=0;
126     L->r[2].keys[2]=1;
127
128     L->r[3].keys[0]=3;
129     L->r[3].keys[1]=6;
130     L->r[3].keys[2]=1;
131
132     L->r[4].keys[0]=0;
133     L->r[4].keys[1]=3;
134     L->r[4].keys[2]=9;
135
136     L->r[5].keys[0]=0;
137     L->r[5].keys[1]=8;
138     L->r[5].keys[2]=5;
139
140     L->r[6].keys[0]=4;
141     L->r[6].keys[1]=8;
142     L->r[6].keys[2]=1;
143
144     L->r[7].keys[0]=5;
145     L->r[7].keys[1]=0;
146     L->r[7].keys[2]=5;
147 }
148 void main()                         /*main function*/
149 {
150     SLList L;
151     printf("RadixSort.cpp \n");
```

东软电子出版社

```
152       printf("============== \n");
153       InitExample(&L);                /* For example */
154       RadixSort(&L);                  /* RadixSort  */
155       /* cout<<endl; */
156       getch();
157   }
```

运行结果：

```
RadixSort.cpp
==============
The InitExample SLList L is:278->109->163->930->580->184->505

The 0 Collect result is:  930 -> 580 -> 163 -> 184 -> 505 -> 278 -> 109

The 1 Collect result is:  505 -> 109 -> 930 -> 163 -> 278 -> 580 -> 184

The 2 Collect result is:  109 -> 163 -> 184 -> 278 -> 505 -> 580 -> 930
```

3.算法分析

对于记录个数为 n、各记录的排序码位数为 d，每一位可以有 rd 种取值可能的待排序序列，r 称为基数。基数排序的时间复杂度为 $o(d(rd+n))$。

基数排序是稳定的。

6.8　各种内部排序方法的比较和选择

排序在计算机程序设计中非常重要，前面所述的各种排序方法各有其优缺点，适用场合也不同。

1．选择排序方法的依据

在选择排序方法时需要考虑的因素有以下四点：

(1)待排序的记录数目 n 的大小；

(2)记录本身数据量的大小，也就是记录中除关键字外的其他信息量的大小；

(3)关键字的结构及其分布情况；

(4)对排序稳定性的要求。

2.选择排序方法的结论

依据上述在选择排序方法时需要考虑的因素，可以得出如下五点结论：

(1)如果待排序记录的初始状态已经按关键字基本有序，则选择直接插入排序法或冒泡排序法为宜。

(2)如果待排序记录的个数 n 较小，则可以采用直接插入排序法或直接选择排序法。

(3)如果待排序记录的个数 n 较大，则应该采用时间复杂度为 $o(n\log_2 n)$ 的排序方法，比如，快速排序法、堆排序法或归并排序法。

①快速排序法被认为是目前基于比较记录关键字的内部排序中最好的排序方法，当待排序序列的关键字是随机分布时，快速排序的平均时间复杂度最优，但是在待排序序列基本有序时，将蜕化为冒泡排序，其时间性能不如堆排序和归并排序；

②堆排序所需要的辅助空间少于快速排序，并且在最坏的情况下时间复杂度不会发生变化；

③归并排序所需要的时间比堆排序省，但是它所需要的辅助存储空间最多。

东软电子出版社

(4)快速排序法、堆排序法、希尔排序法都是不稳定的排序法;直接插入排序法、冒泡排序法、直接选择排序法、归并排序法和基数排序法都是稳定的排序法。

(5)基数排序可以在 $o(d(n+rd))$ 时间内完成对 n 个记录的排序,其中,d 是指单逻辑关键字的个数,rd 是单关键字的取值范围,即基数。基数排序只适用于字符串和整数这类有明显结构特征的关键字;如果 n 很大,d 较小时,用基数排序较好。

表 6.1 各种排序方法的性能比较

排序方法	平均时间	最坏时间	辅助空间	稳定性
直接插入排序	$o(n^2)$	$o(n^2)$	$o(1)$	稳定
冒泡排序	$o(n^2)$	$o(n^2)$	$o(1)$	稳定
快速排序	$o(n\log_2 n)$	$o(n^2)$	$o(\log_2 n)$	不稳定
直接选择排序	$o(n^2)$	$o(n^2)$	$o(1)$	不稳定
归并排序	$o(n\log_2 n)$	$o(n\log_2 n)$	$o(n)$	稳定

本章讨论的排序算法,除了基数排序外,都是在向量存储结构上实现的。当记录本身信息量很大时,为了避免大量时间用在移动数据上,可以利用链表作为存储结构。插入排序和归并排序都容易在链表上实现,但是有的排序方法,比如快速排序和堆排序在链表上却很难实现。

本章小结

排序是程序设计和数据处理中常用的一种运算。本章首先介绍了排序的基本概念,接下来详细介绍了插入排序、交换排序、选择排序、归并排序和基数排序等五类内部排序算法,每类排序中又介绍了最常用的几种算法,包括排序方法的基本思想、排序过程及算法实现,并简要分析了算法的时间复杂度和空间复杂度。

一般来说,比较简单的排序算法,如直接插入、直接选择、冒泡排序等,所需要的时间代价大,为 $O(n^2)$,但在某些特殊情况下可能取得很好的效果。对一些复杂的排序算法,如快速排序、堆排序、归并排序等,平均情况下的时间复杂度为 $O(n\log_2 n)$,堆排序和归并排序在各种情况下的时间复杂度差别不大。快速排序在最坏情况下退为 $O(n^2)$,发生在初始记录已排好序的时候。事实上,快速排序对初始记录排列很“乱”时更有效。

在稳定性方面,快速排序、堆排序、直接选择排序等是不稳定的排序,其他排序是稳定的。

6.9 排序项目实践

读者需要掌握直接插入排序、希尔排序、冒泡排序、快速排序、选择排序、堆排序、归并排序以及基数排序等常用排序算法的思想。

通过本章项目的练习,可进一步掌握各种排序算法的思想,提高排序算法的实现能力,同时,通过大量数据的测试分析,可比较各种排序算法的时间效率。(详见本书光盘)

东南电子出版社

6.10　习题

6.10.1　知识点:直接插入排序

一、选择题

1.② 用直接插入排序方法对下面四个序列进行排序(由小到大),元素比较次数最少的是(　　)。

A. 94,32,40,90,80,46,21,69　　B. 32,40,21,46,69,94,90,80

C. 21,32,46,40,80,69,90,94　　D. 90,69,80,46,21,32,94,40

2.② 直接插入排序在最坏情况下的时间复杂度为(　　)。

A. $O(\log_2 n)$　　B. $O(n)$　　C. $O(n\log_2 n)$　　D. $O(n^2)$

3.② 若对 n 个元素进行直接插入排序,则进行第 I 趟排序过程前,有序表中的元素个数为(　　)。

A. I　　B. I+1　　C. I−1　　D. 1

二、填空题②

直接插入排序用监视哨的作用是_____。【南京理工大学 2001】

三、判断题②

直接插入排序算法在最好情况下的时间复杂度为 O(n)(　　)。【合肥工业大学 2001】

四、简答题③

算法模拟:设待排序的记录共 7 个,排序码分别为 8,3,2,5,9,1,6。用直接插入排序以排序码序列的变化描述形式说明排序全过程(动态过程)要求按递减顺序排序。【山东工业大学 1997】

五、算法题③

请编写直接插入排序算法。(用 C 语言写)

```
Struct  rcdtype{
 Int key;
Element otheritem;
} ARRAY[0..n];
```

【北京轻工业学院 1998】

6.10.2　知识点:希尔排序

一、选择题③

对序列{15,9,7,8,20,−1,4},用希尔排序方法排序,经一趟后序列变为{15,−1,4,8,20,9,7},则该次采用的增量是(　　)。

A. l　　B. 4　　C. 3　　D. 2

二、填空题

1.③ 设用希尔排序对数组{98,36,−9,0,47,23,1,8,10,7}进行排序,给出的步长(也称增量序列)依次是 4,2,1 则排序需_____趟,写出第一趟结束后,数组中数据的排列次序_____。

2.③ 关键码序列{Q,H,C,Y,Q,A,M,S,R,D,F,X},要按照关键码值递增的次序进行排序,若采用初始步长为 4 的 Shell 排序法,则一趟扫描的结果是_____。【北京大学 1997】

三、简答题③

对下面数据表,写出采用 SHELL 排序算法排序的每一趟的结果,并标出数据移动情况。

东软电子出版社

(125,11,22, 34,15,44,76,66,100,8,14,20,2,5,1)。【合肥工业大学 1999】

6.10.3 知识点:冒泡排序

一、选择题

1.③若用冒泡排序方法对序列{10,14,26,29,41,52}从大到小排序,需进行 ( )次比较。

A. 3 B. 10 C. 15 D. 25

2.②对 n 个不同的排序码进行冒泡排序,在下列哪种情况下比较的次数最多?( )

A. 从小到大排列好的 B. 元素逆序 C. 元素无序 D. 元素基本有序

3②对 n 个不同的排序码进行冒泡排序,在元素逆序的情况下比较的次数为( )

A. n+1 B. n C. n-1 D. n(n-1)/2

二、简答题

1.②设要求从大到小排序,问在什么情况下冒泡排序算法关键字交换的次数为最多。

2.③设与记录 R1,R2,…,Rn 对应的关键词分别是 K1,K2,…,Kn。如果存在 Ri 和 Rj 使得 j<i 且 Ki<Kj 成立,试证明经过一趟冒泡后,一定有记录与 Ri 进行交换。【吉林大学 1996】

三、算法题④

冒泡排序算法是把大的元素向上移(气泡的上浮),也可以把小的元素向下移(气泡的下沉)请给出上浮和下沉过程交替的冒泡排序算法。【吉林大学 2001】

6.10.4 知识点:快速排序

一、选择题

1.③有一组数据{15,9,7,8,20,-1,7,4},用快速排序的划分方法进行一趟划分后数据的排序为 ( )(按递增序)。

A. 下面的 B,C,D 都不对。 B. {9,7,8,4,-1,7,15,20}

C. {20,15,8,9,7,-1,4,7} D. {9,4,7,8,7,-1,15,20}

2.③一组记录的关键码为{46,79,56,38,40,84},则利用快速排序的方法,以第一个记录为基准得到的一次划分结果为( )。

A. {38,40,46,56,79,84} B. {40,38,46,79,56,84}

C. {40,38,46,56,79,84} D. {40,38,46,84,56,79}

3.③对下列关键字序列用快速排序法进行排序时,速度最快的情形是( )。

A. {21,25,5,17,9,23,30} B. {25,23,30,17,21,5,9}

C. {21,9,17,30,25,23,5} D. {5,9,17,21,23,25,30}

4.③对关键码序列{28,16,32,12,60,2,5,72}快速排序,从小到大一次划分结果为( )。

A. {2,5,12,16} 26 {60,32,72} B. {5,16,2,12} 28 {60,32,72}

C. {2,16,12,5} 28 {60,32,72} D. {5,16,2,12} 28 {32,60,72}

5.③. 当 n 个整型数据是有序时,对这 n 个数据用快速排序算法排序,则时间复杂度是( ),当用递归算法求 n! 时,算法的时间复杂度是 ( )。

A. O(n) B. O($n\log_2 n$) C. O(n^2) D. O($\log_2 n$)

6.②快速排序在最坏情况下的时间复杂度是( ),比( )的性能差。

A. O($n\log_2 n$) B. O(n^2) C. O(n^3)

D. 堆排序 E. 冒泡排序 F. 选择排序

7.①快速排序方法在( )情况下最不利于发挥其长处。

A. 要排序的数据量太大 B. 要排序的数据中含有多个相同值

C. 要排序的数据个数为奇数 D. 要排序的数据已基本有序

东软电子出版社

二、填空题

1.② 对于 7 个元素的集合{1,2,3,4,5,6,7}进行快速排序，具有最小比较和交换次数的初始排列次序为_____。

2.② 在数据表有序时，快速排序算法的时间复杂度是_____。【合肥工业大学 2001】

三、判断题

1.② 快速排序的速度在所有排序方法中为最快，而且所需附加空间也最少。　（　）

2.② 在初始数据表已经有序时，快速排序算法的时间复杂度为 $O(n\log_2 n)$。　（　）

3.② 在待排数据基本有序的情况下，快速排序效果最好。【南京理工大学 1997】　（　）

四、算法题

1.④ 写出一趟快速排序算法。【山东师范大学 2000】

2.③ 借助于快速排序的算法思想，在一组无序的记录中查找给定关键字值等于 key 的记录。设此组记录存放于数组 r[l..h]中。若查找成功，则输出该记录在 r 数组中的位置及其值，否则显示“not find”信息。请编写出算法并简要说明算法思想。【北京邮电大学 1998】

6.10.5　知识点：直接选择排序

一、选择题③

1. 采用简单选择排序，比较次数与移动次数分别为（　）。

A. $O(n)$，$O(\log_2 n)$　　B. $O(\log_2 n)$，$O(n^2)$

C. $O(n^2)$，$O(n)$　　D. $O(n\log_2 n)$，$O(n)$

2. 在对 n 个元素进行直接选择排序过程中，第 I 趟需从（　　）个元素中选择出最小值元素。

A. n－I＋1　　B. n－I　　C. I　　D. I＋1

二、填空题

1.③ 对 n 个记录的表 r[1..n]进行简单选择排序，所需进行的关键字间的比较次数为_____。

2.③ 用链表表示的数据的简单选择排序，结点的域为数据域 data，指针域 next；链表首指针为 head，链表无头结点。

```
selectsort(head)
    p=head;
while(p(1)_____)
{
q=p; r=(2)_____;
while((3)_____)
{if((4)_____)
q=r;
 r=(5)_____ ;
}
  tmp=q->data; q->data=p->data; p->data=tmp; p=(6)_____;
    }
```

【南京理工大学 2000】

3.③ 下面的 c 函数实现对链表 head 进行选择排序的算法，排序完毕，链表中的结点按结点值从小到大链接。请在空框处填上适当内容，每个空框只填一个语句或一个表达式：

```
#include <stdio.h>
typedef struct node {char data; struct node *link; }node;
```

东软电子出版社

```
node *select(node *head)
{ node *p,*q,*r,*s;
p=(node *)malloc(sizeof(node));
p->link=head; head=p;
while(p->link! =null)
     {q=p->link; r=p;
      while(      (1)      )
         { if(q->link->data<r->link->data) r=q;
           q=q->link;
    }
      if(      (2)      )
{s=r->link; r->link=s->link; s->link=      (3)      ;      (4)      ;
      (5)      ;
     }
p=head; head=head->link; free(p); return(head);
}
```

【复旦大学 1999】

4.④下面的排序算法的思想是:第一趟比较将最小的元素放在 r[1]中,最大的元素放在 r[n]中,第二趟比较将次小的放在 r[2]中,将次大的放在 r[n-1]中,…,依次下去,直到待排序列为递增序。(注:<--->代表两个变量的数据交换)

```
void  sort(SqList &r,int n)  {
i=1;
while(      (1)      )   {
min=max=1;
for(j=i+1;      (2)      ;++j){
if(      (3)      ) min=j; else if(r[j].key>r[max].key)  max=j; }
if(      (4)      ) r[min] <---> r[j];
if(max! =n-i+1){
if(      (5)      ) r[min] <---> r[n-i+1];
else      (6)      ;
}
i++;
}
}
```

【南京理工大学 2001】

三、简答题②

算法模拟:设待排序的记录共 7 个,排序码分别为 8,3,2,5,9,1,6。用直接选择排序以排序码序列的变化描述形式说明排序全过程(动态过程)要求按递减顺序排序。【山东工业大学 1997】

四、算法题④

输入 50 个学生的记录(每个学生的记录包括学号和成绩),组成记录数组,然后按成绩由高到低的次序输出(每行 10 个记录)。排序方法采用选择排序。【北京师范大学 1999】

6.10.6 知识点:堆排序

一、选择题

1.③在含有 n 个关键字的小根堆(堆顶元素最小)中,关键字最大的记录有可能存储在(　　)位置上。

东软电子出版社

A. ⌊n/2⌋　　B. ⌊n/2⌋ −1　　C. 1　　D. ⌊n/2⌋ +2

2.② 以下序列不是堆的是(　　)。

A. {100,85,98,77,80,60,82,40,20,10,66}

B. {100,98,85,82,80,77,66,60,40,20,10}

C. {10,20,40,60,66,77,80,82,85,98,100}

D. {100,85,40,77,80,60,66,98,82,10,20}

3.② 下列四个序列中,哪一个是堆(　　)。

A. 75,65,30,15,25,45,20,10　　B. 75,65,45,10,30,25,20,15

C. 75,45,65,30,15,25,20,10　　D. 75,45,65,10,25,30,20,15

4.② 堆排序是(　　)类排序,堆排序平均执行的时间复杂度是(　　),需要附加的存储空间复杂度是(　　)。

A. 插入　　B. 交换　　C. 归并

D. 基数　　E. 选择

F. O(n^2)和 O(1)　　G. O($n\log_2 n$)和 O(1)

H. O($n\log_2 n$)和 O(n)　　I. O(n^2)和 O(n)

5.③ 有一组数据{15,9,7,8,20,−1,7,4},用堆排序的筛选方法建立的初始堆为(　　)。

A. −1,4,8,9,20,7,15,7　　B. −1,7,15,7,4,8,20,9

C. −1,4,7,8,20,15,7,9　　D. A,B,C 均不对。

二、填空题

1.③ 堆是一种有用的数据结构,堆排序是一种_____排序,堆实质上是一棵_____结点的层次序列。对含有 N 个元素的序列进行排序时,堆排序的时间复杂度是_____,所需的附加存储结点是_____。关键码序列{05,23,16,68,94,72,71,73}是否满足堆的性质_____。【山东工业大学 1996】

2.② 已知一关键码序列为:3,87,12,61,70,97,26,45。试根据堆排序原理,填写完整下列各步骤结果。

建立堆结构:_____

交换与调整:

(1)87 70 26 61 45 12 3 97;(2)_____;

(3)61 45 26 3 12 70 87 97;(4)_____;

(5)26 12 3 45 61 70 87 97;(6)_____;

(7)3 12 26 45 61 70 87 97;【首都经贸大学 1998】

三、判断题

1.② 堆肯定是一棵平衡二叉树。　(　　)

2.② 堆是满二叉树。　(　　)

3.③ {101,88,46,70,34,39,45,58,66,10}是堆。【北京邮电大学 1999】　(　　)

4.② 在用堆排序算法排序时,如果要进行增序排序,则需要采用“大根堆”。【合肥工业大学 2000】　(　　)

5.② 堆排序是稳定的排序方法。【上海交通大学 1998】　(　　)

四、简答题

1.③ 关于堆排序方法,完成如下工作:

(1) 简述该方法的基本思想。

(2) 分析该算法的时间复杂度。【西南财经大学 1999】

2③ 根据给定的关键字集合{20,15,40,35,45,25,50,30,10}。

东软电子出版社

(1)构造一棵完全二叉树；

(2)画出整理好的一棵堆树；

(3)画出一棵输出一个排序记录后的二叉树；

(4)画出重新调整好的堆树。【大连海事大学 2001】

3.③一最小最大堆(min max heap)是一种特定的堆,其最小层和最大层交替出现,根总是处于最小层。最小最大堆中的任一结点的关键字值总是在以它为根的子树中的所有元素中最小(或最大)。如图 6.1 所示为一最小最大堆。

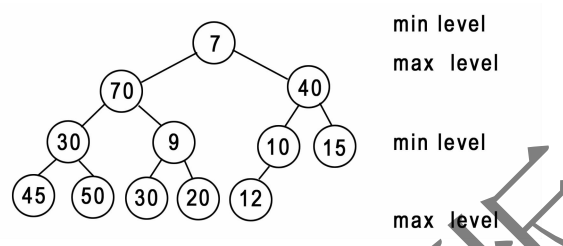


图 6.29　4 题图

(1) 画出在上图中插入关键字为 5 的结点后的最小最大堆。

(2) 画出在上图中插入关键字为 80 的结点后的最小最大堆。【浙江大学 1996】

6.10.7　知识点:归并排序

一、选择题

1.②归并排序中,归并的趟数是(　　)。

A. $O(n)$　　B. $O(\log_2 n)$　　C. $O(n\log_2 n)$　　D. $O(n^2)$

2.②归并排序的时间复杂性是(　　)。

A. $O(n^2)$　　B. $O(n)$　　C. $O(n\log_2 n)$　　D. $O(\log_2 n)$

二、填空题

1.③设有字母序列{Q,D,F,X,A,P,N,B,Y,M,C,W},请写出按 2 路归并排序方法对该序列进行一趟扫描后的结果_____。

2.①外部排序的基本方法是归并排序,但在之前必须先生成_____。【北京邮电大学 2001】

三、判断题②

归并排序辅助存储为 O(1)。　　(　　)

6.10.8　知识点:基数排序

一、简答题③

对整数序列{179,208,93,306,55,859,984,9,271,33}图示其基数排序的全过程。

6.10.9　综合习题

一、选择题

1.②某内排序方法的稳定性是指(　　)。

A. 该排序算法不允许有相同的关键字记录　B. 该排序算法允许有相同的关键字记录

C. 平均时间为 $O(n\log_2 n)$的排序方法　　D. 以上都不对

2.①下面给出的四种排序法中,(　　)排序法是不稳定性排序法。

A. 插入　　B. 冒泡　　C. 二路归并　　D. 堆

东软电子出版社

3.① 下列排序算法中，其中(　　)是稳定的。

A. 堆排序，冒泡排序　　B. 快速排序，堆排序

C. 直接选择排序，归并排序　　D. 归并排序，冒泡排序

4.② 若要求尽可能快地对序列进行稳定的排序，则应选(　　)。

A. 快速排序　　B. 归并排序　　C. 冒泡排序

5.② 若需在 O($n\log_2 n$)的时间内完成对数组的排序，且要求排序是稳定的，则可选择的排序方法是(　　)。

A. 快速排序　　B. 堆排序　　C. 归并排序　　D. 直接插入排序

6.② 下列内部排序算法中：

(1)其比较次数与序列初态无关的算法是(　　)。

(2)不稳定的排序算法是(　　)。

(3)在初始序列已基本有序(除去 n 个元素中的某 k 个元素后即呈有序，k<<n)的情况下，排序效率最高的算法是(　　)。

(4)排序的平均时间复杂度为 O($n\log_2 n$)的算法是(　　)。

(5)为 O(n^2)的算法是(　　)。

A. 快速排序　　B. 直接插入排序　　C. 归并排序　　D. 直接选择排序

E. 冒泡排序　　F. 堆排序

7.② 排序趟数与序列的原始状态有关的排序方法是(　　)排序法。

A. 插入　　B. 选择　　C. 冒泡　　D. 快速

8.③ 数据序列{8,9,10,4,5,6,20,1,2}只能是下列排序算法中的(　　)的两趟排序后的结果。

A. 选择排序　　B. 冒泡排序　　C. 插入排序　　D. 堆排序

9.③ 数据序列{2,1,4,9,8,10,6,20}只能是下列排序算法中的(　　)的两趟排序后的结果。

A. 快速排序　　B. 冒泡排序　　C. 选择排序　　D. 插入排序

10.③ 对一组数据{84,47,25,15,21}排序，数据的排列次序在排序的过程中的变化为：

(1) 84 47 25 15 21　(2) 15 47 25 84 21

(3) 15 21 25 84 47　(4) 15 21 25 47 84

则采用的排序是(　　)。

A. 选择　　B. 冒泡　　C. 快速　　D. 插入

11.③ 对序列{15,9,7,8,20,－1,4}进行排序，进行一趟后数据的排列变为{4,9,－1,8,20,7,15}；则采用的是(　　)排序。

A. 选择　　B. 快速　　C. 希尔　　D. 冒泡

12.③ 若对序列{15,9,7,8,20,－1,4}进行排序经一趟排序后的排列为{9,15,7,8,20,－1,4}，则采用的是(　　)排序。

A. 选择　　B. 堆　　C. 直接插入　　D. 冒泡

13.③ 下列排序算法中，(　　)不能保证每趟排序至少能将一个元素放到其最终的位置上。

A. 快速排序　　B. 希尔排序　　C. 堆排序　　D. 冒泡排序

14.③ 下列排序算法中，(　　)排序在一趟结束后不一定能选出一个元素放在其最终位置上。

A. 选择　　B. 冒泡　　C. 归并　　D. 堆

15.② 在下面的排序方法中，辅助空间为 O(n)的是(　　)。

A. 希尔排序　　B. 堆排序　　C. 选择排序　　D. 归并排序

16.② 下列排序算法中，在待排序数据已有序时，花费时间反而最多的是(　　)排序。

东软电子出版社

A. 冒泡　　B. 希尔　　C. 快速　　D. 堆

17.② 下列排序算法中,在每一趟都能选出一个元素放到其最终位置上,并且其时间性能受数据初始特性影响的是(　　)。

A. 直接插入排序　B. 快速排序　　C. 直接选择排序　D. 堆排序

18.② 对初始状态为递增序列的表按递增顺序排序,最省时间的是(　　)算法,最费时间的是(　　)算法。

A. 堆排序　　B. 快速排序　　C. 插入排序　　D. 归并排序

19.② 就平均性能而言,目前最好的内排序方法是(　　)排序法。

A. 冒泡　　B.　希尔排序　　C. 直接插入　　D. 快速

20.③ 如果只想得到1000个元素组成的序列中第5个最小元素之前的部分排序的序列,用(　　)方法最快。

A. 冒泡排序　　B. 快速排序　　C. 希尔排序　　D. 堆排序

E. 直接选择排序

21.② 在文件"局部有序"或文件长度较小的情况下,最佳内部排序的方法是(　　)。

A. 直接插入排序　B. 冒泡排序　　C. 直接选择排序

22.③ 下列排序算法中,(　　)算法可能会出现下面情况:在最后一趟开始之前,所有元素都不在其最终的位置上。

A. 堆排序　　B. 冒泡排序　　C. 快速排序　　D. 插入排序

23.③ 下列排序算法中,占用辅助空间最多的是(　　)。

A. 归并排序　　B. 快速排序　　C. 希尔排序　　D. 堆排序

24.② 从未排序序列中依次取出一个元素与已排序序列中的元素依次进行比较,然后将其放在已排序序列的合适位置,该排序方法称为(　　)排序法。

A. 插入　　B. 选择　　C. 希尔　　D. 归并

25.② 在排序算法中,每次从未排序的记录中挑出最小(或最大)关键码字的记录,加入到已排序记录的末尾,该排序方法是(　　)。

A. 选择　　B. 冒泡　　C. 插入　　D. 堆

26.② 在排序算法中每一项都与其他各项进行比较,计算出小于该项的项的个数,以确定该项的位置叫(　　)。

A. 插入排序　　B. 枚举排序　　C. 选择排序　　D. 交换排序

27.② 就排序算法所用的辅助空间而言,堆排序、快速排序、归并排序的关系是(　　)。

A. 堆排序 < 快速排序 < 归并排序　　B. 堆排序 < 归并排序 < 快速排序

C. 堆排序 > 归并排序 > 快速排序　　D. 堆排序 > 快速排序 > 归并排序

E. 以上答案都不对

28.③ 设要将序列{q,h,c,y,p,a,m,s,r,d,f,x}中的关键码按字母升序重新排序,(　　)是初始步长为4的shell排序一趟扫描的结果;(　　)是对排序初始建堆的结果;(　　)是以第一个元素为分界元素的快速一趟扫描的结果。从下面供选择的答案中选出正确答案填入括号内。

A.　f,h,c,d,p,a,m,q,r,s,y,x

B.　p,a,c,s,q,d,f,x,r,h,m,y

C.　a,d,c,r,f,q,m,s,y,p,h,x

D.　h,c,q,p,a,m,s,r,d,f,x,y

E.　h,q,c,y,a,p,m,s,d,r,f,x

29.③ 将两个各有N个元素的有序表归并成一个有序表,其最少的比较次数是(　　)。

A. N　　B. 2N−1　　C. 2N　　D. N−1

华软电子出版社

二、填空题

1.① 若不考虑基数排序，则在排序过程中，主要进行的两种基本操作是关键字的_____和记录的_____。【北京邮电大学 2001】

2.② 分别采用堆排序、快速排序、冒泡排序和归并排序，对初态为有序的表，则最省时间的是_____算法，最费时间的是_____算法。【福州大学 1998】

3.① 不受待排序初始序列的影响，时间复杂度为 $O(n^2)$ 的排序算法是_____，在排序算法的最后一趟开始之前，所有元素都可能不在其最终位置上的排序算法是_____。【中国人民大学 2001】

三、判断题

1.② 当待排序的元素很大时，为了交换元素的位置，移动元素要占用较多的时间，这是影响时间复杂度的主要因素。【长沙铁道学院 1998】（　　）

2.② 内排序要求数据一定要以顺序方式存储。【南京理工大学 1997】（　　）

3.② 排序的稳定性是指排序算法中的比较次数保持不变，且算法能够终止。【南京航空航天大学 1996】（　　）

4.② 在执行某个排序算法过程中，出现了排序码朝着最终排序序列位置相反方向移动，则该算法是不稳定的。【上海交通大学 1998】（　　）

5.② 快速排序的速度在所有排序方法中为最快，而且所需附加空间也最少。（　　）

6.② 冒泡排序和快速排序都是基于交换两个逆序元素的排序方法，冒泡排序算法的最坏时间复杂性是 $O(n^2)$，而快速排序算法的最坏时间复杂性是 $O(n\log_2 n)$，所以快速排序比冒泡排序算法效率更高。【上海海运学院 1997】（　　）

7.② 在任何情况下，归并排序都比简单插入排序快。【北京邮电大学 2000】（　　）

8.② 快速排序总比简单排序快。【东南大学 2001】（　　）

四、简答题

1.① 内部排序(名词解释)。【燕山大学 1999】

2.② 简述直接插入排序，简单选择排序，2-路归并排序的基本思想以及在时间复杂度和排序稳定性上的差别。【西北工业大学 1999】

3.③ 给出一组关键字 T={12,2,16,30,8,28,4,10,20,6,18}，写出用下列算法从小到大排序时第一趟结束时的序列：

(1) 希尔排序(第一趟排序的增量为 5)。

(2) 快速排序(选第一个记录为枢轴(分隔))。

(3) 链式基数排序(基数为 10)。【上海交通大学 1999】

4.③ 给出一组关键字：29,18,25,47,58,12,51,10，分别写出按下列各种排序方法进行排序时的变化过程：

(1) 归并排序：每归并一次书写一个次序。

(2) 快速排序：每划分一次书写一个次序。

(3) 堆排序：先建成一个堆，然后每从堆顶取下一个元素后，将堆调整一次。【南开大学 1998】

东软电子出版社