第二章 数据排序

信息获取后通常需要进行处理，处理后的信息其目的是便于人们的应用。信息处理方法有多种，通常有数据的排序，查找，插入，删除，归并等操作。读者已经接触了一些这方面的知识，本章重点介绍数据排序的几种方法。

1. 选择排序

（1） 基本思想：每一趟从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在待排序的数列的最前，直到全部待排序的数据元素排完。

（2）排序过程：

【示例】：

初 始 关键字 [49 38 65 97 76 13 27 49]
第一趟排序后 13 [38 65 97 76 49 27 49]
第二趟排序后 13 27 [65 97 76 49 38 49]
第三趟排序后 13 27 38 [97 76 49 65 49]
第四趟排序后 13 27 38 49 [76 97 65 49]
第五趟排序后 13 27 38 49 49 [97 65 76]
第六趟排序后 13 27 38 49 49 65 [97 76]
第七趟排序后 13 27 38 49 49 65 76 [97]
最后排序结果 13 27 38 49 49 65 76 97

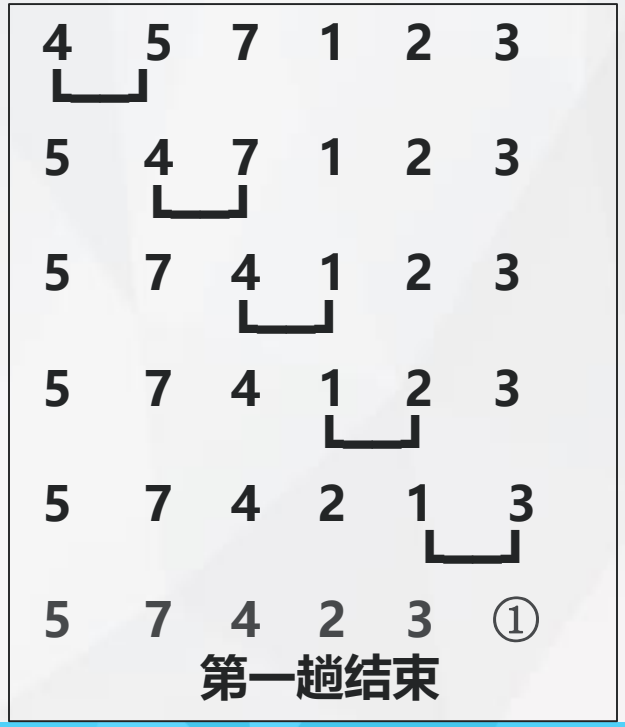
```
void SelectSort(int R[])                    //对R[1..n]进行直接选择排序
{
   for (int i=1;i<=n-1;i++)                  //做n - 1趟选择排序
      {
         k = i;
         for (int j=i+1;j<=n;j++)   //在当前无序区R[i..n]中选最小的元素R[K]
            {
               If (R[j] < R[k])  k = j;
            }
         if (k!=i)                           //交换R[i]和R[k]
           {
             int temp = R[i];
             R[i] = R[k];
             R[k] = temp;
           }
      }
}                                            //SelectSort
```

2.冒泡排序

（1）基本的冒泡排序

①基本思想

依次比较相邻的两个数，把大的放前面，小的放后面。即首先比较第1个数和第2个数，大数放前,小数放后。然后比较第2个数和第3个数……直到比较最后两个数。第一趟结束，最小的一定沉到最后。重复上过程,仍从第1个数开始,到最后第2个数，……由于在排序过程中总是大数往前，小数往后，相当气泡上升，所以叫冒泡排序。下面是6个元素的排序的过程



②算法实现

```
for (int i=1;i<=n-1;i++)
    for (int j=1;j<=n-i;j++)
        if (a[j]<a[j+1])
        {
            temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
        }
```

（2）算法改进

上例中，可以发现，第二趟结束已经排好序。但是计算机此时并不知道已经排好序。所以，还需进行一次比较，如果没有发生任何数据交换，则知道已经排好序，可以不干了。因此第三趟比较还需进行，第四趟、第五趟比较则不必要。

我们设置一个布尔变量bo 来记录是否有进行交换。值为false表示本趟中进行了交换，true 则没有。代码如下:

```
int i=1,temp; bool bo;
do
{
    bo=true;
    for (int j=1;j<=n-i;j++)
        if (a[j]<a[j+1])
        {
            temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
            bo=false;
        }
    i++;
}
while (!bo);
```

3. 桶排序

桶排序的思想是若待排序的记录的关键字在一个明显有限范围内(整型)时，可设计有限个有序桶，每个桶装入一个值（当然也可以装入若干个值），顺序输出各桶的值，将得到有序的序列。

例：输入n个0到100之间的不相同整数，由小到大排序输出。

```
#include<iostream>
#include <cstring>
using namespace std;
int main()
{
    int b[101],k,i,n;
    memset(b,0,sizeof(b));                    //初始化
    cin>>n;
    for( i=1;i<=n;i++)
    {
        cin>>k;   b[k]++;                  //将关键字等于k的值全部装入第k桶
    }
    for( i=0; i<=100;i++)
    while (b[i]>0)  {cout<<i<<"  " ;b[i]--;}    //输出排序结果
    cout<<endl;
}
```

4. 插入排序

插入排序是一种简单的排序方法，其算法的基本思想是：

假设待排序的数据存放在数组R[1..n]中，增加一个哨兵结点x。

(1) R[1]自成1个有序区，无序区为R[2..n];

(2) 从i=2起直至i=n为止，将R[i]放在恰当的位置，使R[1..i]数据序列有序;

① x:=R[i];

② 将x与前i-1个数比较，j:=i-1；while x<a[j] do j:=j-1;

③ 将R数组的元素从j位置开始向后移动: for k:=i downto j do a[k]:=a[k-1];

④ R[j]=x;

(3) 生成包含n个数据的有序区。.

例如:设n=8，数组R中8个元素是: 36,25,48,12,65,43,20,58，执行插入排序程序后，其数据变动情况：

第0步:[36] 25 48 12 65 43 20 58

第1步:[25 36] 48 12 65 43 20 58

第2步:[25 36 48] 12 65 43 20 58

第3步:[12 25 36 48] 65 43 20 58

第4步:[12 25 36 48 65] 43 20 58

第5步:[12 25 36 43 48 65] 20 58

第6步:[12 20 25 36 43 48 65] 58

第7步:[12 20 25 36 43 48 58 65]

该算法的程序简单，读者自己完成。其算法的时间复杂性为$O(n^2)$插入排序适用于原先数据已经排列好，插入一个新数据的情况。

```
void insertsort(int r[])                    //对r[1..n]按递增序进行插入排序，x是监视哨
{
 for (i=2;i<=n;i++)                         //依次插入r[2],...,r[n]
     {
       x=r[i];
       j= i-1;
       while (x< r[j])                      //查找r[i]的插入位置//
          {
             r[j+1] =r[j];                  //将大于r[i]的元素后移//
             j--;
          }
       r[j+1] = x;                          //插入r[I] //
     }
}
```

5.快速排序

快速排序是对冒泡排序的一种改进。它的基本思想是，通过一趟排序将待排记录分割成独立的两部分，其中一部分记录的关键字均比另一部分记录的关键字小，则可分别对这两部分记录继续进行排序，以达到整个序列有序。

假设待排序的序列为{a[L],a[L+1],a[L+2],.....,a[R]}，首先任意选取一个记录（通常可选中间一个记作为枢轴或支点），然后重新排列其余记录，将所有关键字小于它的记录都放在左子序列中，所有关键字大于它的记录都放在右子序列中。由此可以将该“支点”记录所在的位置mid作分界线，将序列分割成两个子序列和。这个过程称作一趟快速排序（或一次划分）。

一趟快速排序的具体做法是：附设两个指针i和j，它们的初值分别为L和R，设枢轴记录取mid，则首先从j所指位置起向前搜索找到第一个关键字小于的mid的记录，然后从i所指位置起向后搜索，找到第一个关键字大于mid的记录，将它们互相交换，重复这两步直至i>j为止。

快速排序的时间的复杂性是O(nlog2n)，速度快,但它是不稳定的排序方法。就平均时间而言，快速排序是目前被认为是最好的一种内部排序方法

由以上讨论可知，从时间上看，快速排序的平均性能优于前面讨论过的各种排序方法，但快速排序需一个栈空间来实现递归。若每一趟排序都将记录序列均匀地分割成长度相接近的两个子序列，则栈的最大深度为log(n+1)。

快速排序算法

```
void qsort(int l,int r)
{
    int i,j,mid,p;
    i=l;
    j=r;
    mid=a[(l+r) / 2];                    //将当前序列在中间位置的数定义为分隔数
    do
    {
        while (a[i]<mid) i++;          //在左半部分寻找比中间数大的数
        while (a[j]>mid) j--;         //在右半部分寻找比中间数小的数
        if (i<=j)
        {                              //若找到一组与排序目标不一致的数对则交换它们
            p=a[i];
            a[i]=a[j];
            a[j]=p;
            i++;
            j--;                       //继续找
        }
    }
    while (i<=j);                                //注意这里不能有等号
    if (l<j)  qsort(l,j);                //若未到两个数的边界，则递归搜索左右区间
    if (i<r)  qsort(i,r);
}
```

6.归并排序

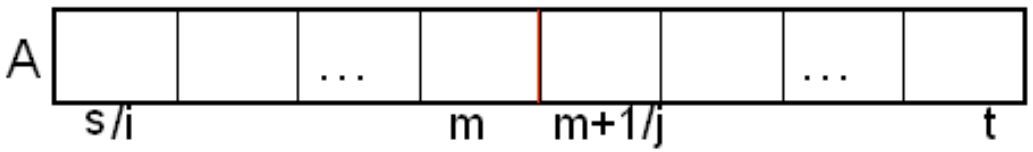
归并排序（MERGE-SORT）是建立在归并操作上的一种有效的排序算法,该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。

将两个或两个以上有序的数列(或有序表)，合并成一个仍然有序的数列(有序表)，这种操作称为归并操作。这样的方法经常用于多个有序的数据文件归并成一个有序的数据文件。若将两个有序表合并成一个有序表则称为二路归并，同理，有三路归并、四路归并等。二路归并比较简单，所以我们只讨论二路归并。例如有两个有序表: (7，10，13，15)和(4，8，19，20)，归并后得到的有序表为: (4，7，8，10，13，15，19，20)。

归并过程为:比较A[i]和A[j]的大小，若A[i]≤A[j]，则将第一个有序表中的元素A[i]复制到R[k]中，并令i和k分别加1，即使之分别指问后一单元，否则将第二个有序表中的元素A[j]复制到R[k]中，并令j和k分别加1;如此循环下去，直到其中的一个有序表取完,然后再将另一个有序表中剩余的元素复制到R中从下标k到下标t的单元.

二路归并算法描述

```
//将(A[s:t]中两个有序表A[s:m]和A[m+1:t]合并成一个有序表R[s:t]
void merge(int s, int m,int t)
     //s是第一个有序表起点位置，m+1是第二个有序表的起点
{   int i,j,k;
    i=s;
   j=m+1;
   k=s;                                  //i和j分别指向二个有序表的头部
   while (i<=m&&j<=t)
       if (A[i] <=A[j] )
           {R[k]=A[i];  i++;  k++;}      //三条语句可以写成R[k++]=A[i++]
       else {R[k]=A[j];  j++;  k++;}
    while (i<=m)   {R[k]=A[i]; i++; k++;}   //复制第一路剩余
    while ( j<=t)   {R[k]=A[j]; j++; k++; }    //复制第二路剩余
}
```



二路归并算法描述

```
void Merge(int sourceArr[],int tempArr[], int startIndex, int midIndex, int endIndex)
{
    int i = startIndex, j=midIndex+1, k = startIndex;
    while(i<=midIndex && j<=endIndex)
    {
        if(sourceArr[i] > sourceArr[j])
            tempArr[k++] = sourceArr[j++];
        else
            tempArr[k++] = sourceArr[i++];
    }
    while(i <= midIndex)
        tempArr[k++] = sourceArr[i++];
    while(j <= endIndex)
        tempArr[k++] = sourceArr[j++];
    for(i=startIndex; i<=endIndex; i++)
        sourceArr[i] = tempArr[i];
}
```

归并排序(Merge sort)

- 归并排序(Merge sort)就是利用归并操作把一个无序表排列成一个有序表的过程。二路归并排序的过程是首先把待排序区间（即无序表）中的每一个元素都看作为一个有序表，则n个元素构成n个有序表，接着两两归并（即第一个表同第二个表归并，第三个表同第四个表归并，...），得到[n/2]个长度为2的有序表（最后一个表的长度可能小于2），称此为一趟归并，然后再两两有序表归并，得到[[n/2]/2]个长度为4的有序表（最后一个表的长度可能小于4），如此进行下去，直到归并第[log2n]趟后得到一个长度为n的有序表为止。
- 归并排序算法我们用递归实现，先把待排序区间[s,t]以中点二分，接着把左边子区间排序，再把右边子区间排序，最后把左区间和右区间用一次归并操作合并成有序的区间[s,t]。对左右子区间的排序与原问题一样，所以我们可以调用同样的子程序，只是区间大小不一样。

归并排序算法

```
void MergeSort(int sourceArr[], int tempArr[], int startIndex, int endIndex)
{
    int midIndex;
    if(startIndex < endIndex)
    {
        midIndex = startIndex + (endIndex-startIndex) / 2;
                                                    //避免溢出int
        MergeSort(sourceArr, tempArr, startIndex, midIndex);
        MergeSort(sourceArr, tempArr, midIndex+1, endIndex);
        Merge(sourceArr, tempArr, startIndex, midIndex, endIndex);
    }
}
```

主程序调用

```
#include<iostream>
using namespace std;

int main( )
{
   int a[10001],b[10001],n,i;
   cin>>n;
   for(i=0;i<n ;i++)                    //读入n个待排序数据
   cin>>a[i];


   MergeSort(a, b, 0, n-1);
   for(i=0; i<n; i++)
      printf("%d ", a[i]);
   printf("\n");
   return 0;
}
```

7.各种排序算法的比较

1.稳定性比较

插入排序、冒泡排序、二叉树排序、二路归并排序及其他线形排序是稳定的。

选择排序、希尔排序、快速排序、堆排序是不稳定的。

2.时间复杂性比较

插入排序、冒泡排序、选择排序的时间复杂性为O(n2)；快速排序、堆排序、归并排序的时间复杂性为O(nlog2n)；桶排序的时间复杂性为O(n);

若从最好情况考虑，则直接插入排序和冒泡排序的时间复杂度最好，为O(n)，其它算法的最好情况同平均情况相同；若从最坏情况考虑，则快速排序的时间复杂度为O(n2)，直接插入排序和冒泡排序虽然平均情况相同，但系数大约增加一倍，所以运行速度将降低一半，最坏情况对直接选择排序、堆排序和归并排序影响不大。

由此可知，在最好情况下，直接插入排序和冒泡排序最快；在平均情况下，快速排序最快；在最坏情况下，堆排序和归并排序最快。

3.辅助空间的比较

桶排序、二路归并排序的辅助空间为O(n)，快速排序的辅助空间为O(log2n)，最坏情况为O(n)，其它排序的辅助空间为O(1);

4.其它比较

插入、冒泡排序的速度较慢，但参加排序的序列局部或整体有序时，这种排序能达到较快的速度。反而在这种情况下，快速排序反而慢了。

当n较小时，对稳定性不作要求时宜用选择排序，对稳定性有要求时宜用插入或冒泡排序。

若待排序的记录的关键字在一个明显有限范围内时,且空间允许是用桶排序。

当n较大时，关键字元素比较随机，对稳定性没要求宜用快速排序。

当n较大时，关键字元素可能出现本身是有序的，对稳定性没有要求时宜用堆排序

快速排序是目前基于比较的内部排序中被认为是最好的方法，当待排序的关键字是随机分布时，快速排序的平均时间最短;

堆排序所需的辅助空间少于快速排序，并且不会出现快速排序可能出现的最坏情况。这两种排序都是不稳定的。

【上机练习】

1、明明的随机数（Noip2006）

【问题描述】明明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性，他先用计算机生成了N个1到1000之间的随机整数（N≤100），对于其中重复的数字，只保留一个，把其余相同的数去掉，不同的数对应着不同的学生的学号。然后再把这些数从小到大排序，按照排好的顺序去找同学做调查。请你协助明明完成“去重”与“排序”的工作。

【输入文件】输入文件random.in 有2行，

第1行为1个正整数，表示所生成的随机数的个数：N

第2行有N个用空格隔开的正整数，为所产生的随机数。

【输出文件】输出文件random.out 也是2行，第1行为1个正整数M，表示不相同的随机数的个数。第2行为M个用空格隔开的正整数，为从小到大排好序的不相同的随机数。

【输入样例】

10

20 40 32 67 40 20 89 300 400 15

【输出样例】

8

15 20 32 40 67 89 300 400

2、车厢重组（carry.pas）

【问题描述】

在一个旧式的火车站旁边有一座桥，其桥面可以绕河中心的桥墩水平旋转。一个车站的职工发现桥的长度最多能容纳两节车厢，如果将桥旋转180度，则可以把相邻两节车厢的位置交换，用这种方法可以重新排列车厢的顺序。于是他就负责用这座桥将进站的车厢按车厢号从小到大排列。他退休后，火车站决定将这一工作自动化，其中一项重要的工作是编一个程序，输入初始的车厢顺序，计算最少用多少步就能将车厢排序。

【输入文件】

输入文件有两行数据，第一行是车厢总数N（不大于10000），第二行是N个不同的数表示初始的车厢顺序。

【输出文件】

一个数据，是最少的旋转次数。

【输入样例】carry .in

4

4 3 2 1

【输出样例】carry .out

6

3、众数(masses.pas)

【问题描述】

由文件给出N个1到30000间无序数正整数，其中1≤N≤10000，同一个正整数可能会出现多次，出现次数最多的整数称为众数。求出它的众数及它出现的次数。

【输入格式】

输入文件第一行是正整数的个数N，第二行开始为N个正整数。

【输出格式】

输出文件有若干行，每行两个数，第1个是众数，第2个是众数出现的次数。

【输入样例】masses.in

12

2 4 2 3 2 5 3 7 2 3 4 3

【输出样例】masses.out

2 4

3 4

5、军事机密(Secret.pas)

【问题描述】

军方截获的信息由n（n<=30000）个数字组成，因为是敌国的高端秘密，所以一时不能破获。最原始的想法就是对这n个数进行小到大排序，每个数都对应一个序号，然后对第i个是什么数感兴趣，现在要求编程完成。

【输入格式】

第一行n，接着是n个截获的数字，接着一行是数字k，接着是k行要输出数的序号。

【输出格式】

k行序号对应的数字。

【输入样例】Secret.in

```
5
121  1  126  123   7
3
2
4
3
```

【输出样例】Secret.out

```
7
123
121
```

6、奖学金(Noip2007)

【问题描述】

某小学最近得到了一笔赞助，打算拿出其中一部分为学习成绩优秀的前5名学生发奖学金。期末，每个学生都有3门课的成绩：语文、数学、英语。先按总分从高到低排序，如果两个同学总分相同，再按语文成绩从高到低排序，如果两个同学总分和语文成绩都相同，那么规定学号小的同学排在前面，这样，每个学生的排序是唯一确定的。

任务：先根据输入的3门课的成绩计算总分，然后按上述规则排序，最后按排名顺序输出前5名学生的学号和总分。注意，在前5名同学中，每个人的奖学金都不相同，因此，你必须严格按上述规则排序。例如，在某个正确答案中，如果前两行的输出数据（每行输出两个数：学号、总分）是：

7 279

5 279

这两行数据的含义是：总分最高的两个同学的学号依次是7号、5号。这两名同学的总分都是279（总分等于输入的语文、数学、英语三科成绩之和），但学号为7的学生语文成绩更高一些。如果你的前两名的输出数据是：

5 279

7 279

则按输出错误处理，不能得分。0

【输入格式】

输入文件scholar.in包含*n*+1行：
第1行为一个正整数*n*，表示该校参加评选的学生人数。
第2到*n*+1行，每行有3个用空格隔开的数字，每个数字都在0到100之间。第*j*行的3个数字依次表示学号为*j*-1的学生的语文、数学、英语的成绩。每个学生的学号按照输入顺序编号为1~*n*（恰好是输入数据的行号减1）。 所给的数据都是正确的，不必检验。

【输出格式】输出文件scholar.out共有5行，每行是两个用空格隔开的正整数，依次表示前5名学生的学号和总分。

【输入输出样例1】

scholar.in	scholar.out
6 90 67 80 87 66 91 78 89 91 88 99 77 67 89 64 78 89 98	6 265 4 264 3 258 2 244 1 237

【输入输出样例2】

scholar.in	scholar.out
8 80 89 89 88 98 78 90 67 80 87 66 91 78 89 91 88 99 77 67 89 64 78 89 98	8 265 2 264 6 264 1 258 5 258

【限制】

50%的数据满足：各学生的总成绩各不相同

100%的数据满足：
6<=n<=300

7、统计数字(Noip2007)

【问题描述】某次科研调查时得到了*n*个自然数，每个数均不超过1500000000（$1.5*10^9$）。已知不相同的数不超过10000个，现在需要统计这些自然数各自出现的次数，并按照自然数从小到大的顺序输出统计结果。

【输入格式】输入文件count.in包含*n*+1行：

第1行是整数*n*，表示自然数的个数。

第2~*n*+1行每行一个自然数。

【输出格式】输出文件count.out包含*m*行（*m*为*n*个自然数中不相同数的个数），按照自然数从小到大的顺序输出。每行输出两个整数，分别是自然数和该数出现的次数，其间用一个空格隔开。

【输入输出样例】

count.in	count.out
8 2 4 2 4 5 100 2 100	2 3 4 2 5 1 100 2

【限制】

40%的数据满足：1<=*n*<=1000

80%的数据满足：1<=*n*<=50000

100%的数据满足：

1<=*n*<=200000，每个数均不超过1 500 000 000（1.5*109）

8、输油管道问题

【问题描述】某石油公司计划建造一条由东向西的主输油管道。该管道要穿过一个有n 口油井的油田。从每口油井都要有一条输油管道沿最短路经(或南或北)与主管道相连。如果给定n口油井的位置，即它们的x 坐标（东西向）和y 坐标（南北向），应如何确定主管道的最优位置，即使各油井到主管道之间的输油管道长度总和最小的位置？证明可在线性时间内确定主管道的最优位置。

【编程任务】给定n 口油井的位置,编程计算各油井到主管道之间的输油管道最小长度总和。

【输入格式】由文件pipe.in提供输入数据。文件的第1 行是油井数n，1≤n≤10000。接下来n 行是油井的位置，每行2个整数x和y，-10000≤x,y≤10000。

【输出格式】程序运行结束时，将计算结果输出到文件pipe.out 中。文件的第1 行中的数是油井到主管道之间的输油管道最小长度总和。

【输入样例】

```
5
1  2
2  2
1  3
3  -2
3  3
```

【输出样例】

```
6
```

9、士兵站队问题

【问题描述】在一个划分成网格的操场上，n个士兵散乱地站在网格点上。网格点由整数坐标(x,y)表示。士兵们可以沿网格边上、下、左、右移动一步，但在同一时刻任一网格点上只能有一名士兵。按照军官的命令，士兵们要整齐地列成一个水平队列，即排列成(x,y),(x+1,y),...,(x+n-1,y)。如何选择x和y的值才能使士兵们以最少的总移动步数排成一列。

【编程任务】 计算使所有士兵排成一行需要的最少移动步数。

【输入格式】由文件sol.in提供输入数据。文件的第1 行是士兵数n，1≤n≤10000。接下来n 行是士兵的初始位置，每行2 个整数x 和y，-10000≤x，y≤10000。

【输出格式】 程序运行结束时，将计算结果输出到文件sol.out中。文件的第1 行中的数是士兵排成一行需要的最少移动步数。

【输入样例】

```
5
1 2
2 2
1 3
3 -2
3 3
```

【输出样例】

```
8
```