第九章　排序

1、复习要点

排序是使用最频繁的一类算法。排序分内排序与外排序。内排序算法主要分 5 大类，有 12 个算法。在插入排序类中讨论了直接插入排序、二分插入排序、表插入排序和 shell 排序算法；在交换排序类中讨论了起泡排序和快速排序算法；在选择排序类中讨论了简单选择排序、锦标赛排序和堆排序算法；在归并排序类中讨论了迭代的两路归并排序和递归的表归并排序算法；在多排序码排序类中讨论了最低位优先的链表基数排序算法。其中，不稳定的排序方法有 shell 排序、简单选择排序、快速排序和堆排序；适合于待排序对象数目 n 比较大的排序方法有快速排序、堆排序、归并排序和基数排序；排序码比较次数不受对象排序码初始排列影响的排序方法有折半插入排序、简单选择排序、锦标赛排序、两路归并排序和基数排序。其中，当排序码的初始排列接近有序时，直接插入排序和起泡排序等增长很快，而快速排序则变成慢速排序。

外排序是基于外存的排序方法。由于外存以顺序存取的效率最高，以归并排序最为适合。因此，外排序以 k 路平衡归并为主。在 k 个对象排序码中选取最小排序码，采用了败者树。这是一种高效的选择算法。此外，还讨论了初始归并段生成的方法，最佳归并树等问题。

1.1 基本知识点

要求理解排序的基本概念，包括什么是排序、排序的稳定性、排序的性能分析，如时间代价（对象排序码的比较次数和对象的移动次数）和空间代价（附加对象个数）。掌握插入排序（直接插入排序、折半插入排序、链表插入排序）、交换排序（起泡排序、快速排序）、选择排序（直接选择排序、链表选择排序、锦标赛排序、堆排序）、迭代的归并排序等内排序的方法及其性能分析方法。理解基数排序方法及其性能分析方法。理解多路平衡归并等外排序方法及败者树构造方法。理解生成初始归并段及败者树构造方法。理解最佳归并树的建立方法。

1.2 算法设计

- ✧ 插入排序——直接插入排序算法、折半插入排序算法、链表插入排序算法；
- ✧ 交换排序——起泡排序算法、快速排序的递归算法和用栈实现的非递归算法；
- ✧ 选择排序——直接选择排序算法、链表选择排序算法、堆排序算法；
- ✧ 归并排序——两路归并算法、迭代的归并排序算法、递归的链表归并排序算法和用队列实现的非递归链表归并排序算法；
- ✧ 其它排序算法——计数排序算法、奇偶排序算法。

2、难点与重点

- ✧ 基本概念——排序码、初始排序码排列、排序码比较次数、数据移动次数、稳定性、附加存储、内部排序、外部排序。
- ✧ 插入排序——当待排序的排序码序列已经基本有序时，用直接插入排序最快。
- ✧ 选择排序
  - (1) 用直接选择排序在一个待排序区间中选出最小的数据时，与区间第一个数据对调，而不是顺次后移，这导致方法不稳定；
  - (2) 当在 n 个数据（n 很大）中选出最小的 5~8 个数据时，锦标赛排序最快；

(3) 锦标赛排序的算法中将待排序的数据个数 n 不足到 2 的 k 次幂 $2^{k-1}<n\leqslant 2^k$；

(4) 在堆排序中将待排序的数据组织成完全二叉树的顺序存储。

✧ 交换排序

(1) 快速排序是一个递归的排序方法；

(2) 当待排序的排序码序列已经基本有序时，快速排序显著变慢。

✧ 二路归并排序

(1) 归并排序可以递归执行；

(2) 归并排序需要较多的附加存储。可以采用一种“推拉法”实现归并排序，算法的时间复杂度为 O(n)，空间复杂度为 O(1)；

(3) 归并排序对待排序的排序码的初始排列不敏感，排序速度较稳定。

✧ 外排序

(1) 多路平衡归并排序的过程、I/O 缓冲区个数的配置；

(2) 外排序的时间分析，利用败者树进行多路平衡归并；

(3) 利用置换选择方法生成不等长的初始归并段；

(1) 最佳归并树的构造及 WPL 的计算。

3、重点概念分析

✧ **什么是内排序？什么是外排序？什么排序方法是稳定的？什么排序方法是不稳定的？**

内排序是排序过程中参与排序的数据全部在内存中所做的排序，排序过程中无需进行内外存数据传送，决定排序方法时间性能的主要是数据排序码的比较次数和数据对象的移动次数。外排序是在排序的过程中参与排序的数据太多，在内存中容纳不下，因此在排序过程中需要不断进行内外存的信息传送的排序。决定外排序时间性能的主要是读写磁盘次数和在内存中总的记录对象的归并次数。

不稳定的排序方法主要有希尔排序、直接选择排序、堆排序、快速排序。不稳定的排序方法往往是按一定的间隔移动或交换记录对象的位置，从而可能导致具有相等排序码的不同对象的前后相对位置在排序前后颠倒过来。其它排序方法中如果有数据交换，只是在相邻的数据对象间比较排序码，如果发生逆序（与最终排序的顺序相反的次序）才交换，因此具有相等排序码的不同对象的前后相对位置在排序前后不会颠倒，是稳定的排序方法。但如果把算法判断逆序的比较“>（或<）”改写成“≥（或≤）”，也可能造成不稳定。

✧ **在起泡排序过程中，什么情况下排序码会朝向与排序相反的方向移动，试举例说明。在快速排序过程中有这种现象吗？**

如果在待排序序列后面的若干排序码比前面的排序码小，则在起泡排序的过程中，排序码可能向与最终它应移向的位置相反的方向移动。

✧ **如果待排序的排序码序列已经按非递减次序有序排列，试证明函数 QuickSort( )的计算时间将下降到 $O(n^2)$。**

若待排序的 n 个对象的序列已经按排序码非递减次序有序排列，且设排序的时间代价为 T(n)。当以第一个对象作为基准对象时，应用一次划分算法 Partition，通过 n-1 次排序码比较，把只能把整个序列划分为：基准对象的左子序列为空序列，右子序列为有 n-1 个对象的非递减有序序列。对于这样的递归 QuickSort( )算法，其时间代价为：

T(n)=(n-1)+T(n-1)=(n-1)+{(n-2)+T(n-2)}= (n-1)+(n-2)+{(n-3)+T(n-3)}

=…

= (n-1)+(n-2)+(n-3)+…+{2+T(1)}

= (n-1)+(n-2)+(n-3)+…+2+1

$=n(n-1)/2=O(n^2)$

✧ **在实现快速排序的非递归解法时，可根据基准对象，将待排序排序码序列划分为两个子序列。若下一趟首先对较短的子序列进行排序。试证明在此做法下，快速排序所需要的栈的深度为 $O(\log_2 n)$。**

由快速排序的算法可知，所需递归工作栈的深度取决于所需划分的最大次数。如果在排序过程中每次划分都能把整个待排序序列根据基准对象划分为左、右两个子序列。假定这两个子序列的长度相等，则所需栈的深度为：

$S(n)=1+S(n/2)=1+\{(1+S(n/4))=2+S(n/4)=2+\{1+S9n/8\}\}=3+S(n/8)$

$=\cdots$

$=\log_2 n+S(1)=\log_2 n$

（假设 1 个对象的序列所用递归栈的深度为 0）。

如果每次递归左、右子序列的长度不等，并且先将较长的子序列的左、右端点保存在递归栈中，再对较短的子序列进行排序，可用表示最坏情况的大 O 表示法表示。此时，其递归栈的深度不一定正好是 $\log_2 n$，其最坏情况为 $O(\log_2 n)$。

✧ **在使用非递归方法实现快速排序时，通常要利用一个栈记忆待排序区间的两个端点。那么能否用队列来代替这个栈？为什么？**

可以用队列来代替栈。在快速排序的过程中，通过一趟划分，可以把一个待排序区间分为两个子区间，然后分别对这两个子区间施行同样的划分。栈的作用是在处理一个子区间时，保存另一个子区间的上界和下界，待该区间处理完成后再从栈中取出另一子区间的边界，对其进行处理。这个功能利用队列也可以实现，只不过是处理子区间的顺序有所变动而已。

✧ **在什么条件下，MSD 基数排序比 LSD 基数排序效率更高？**

由于高位优先的 MSD 方法是递归的方法，就一般情况来说，不像低位优先的 LSD 方法那样直观自然，而且实现的效率较低。但如果待排序的排序码的大小只取决于高位的少数几位而与大多数低位无关时，采用 MSD 方法比 LSD 方法的效率要高。

✧ **试证明对一个有 n 个对象的序列进行基于比较的排序，最少需要执行 $n\log_2 n$ 次排序码比较。**

在基于比较的排序方法中，采用分治法进行排序是平均性能最好的方法。方法描述如下：

```
Sort(List)
{
    if (List 的长度大于 1)
    {
        将序列 List 划分为两个子序列 LeftList 和 RightList;
        Sort(LeftList); Sort(RightList); //分别对两个子序列施行排序
        将两个子序列 LeftList 和 RightList 合并为一个序列 List;
    }
}
```

典型的例子就是快速排序和归并排序。若设 T(n)是对 n 个对象的序列进行排序所需的时间，而且把序列划分为长度相等的两个子序列后，对每个子序列进行排序所需的时间为 T(n/2)，最后合并两个已排好序的子序列所需时间为 cn（c 是一个常数）。此时，总的计算时间为：

$T(n)\le cn+2T(n/2)$ //c 是一个常数

≤cn+2(cn/2+2T(n/4))=2cn+4T(n/4)

≤cn+4(cn/4+2T(n/8))=3cn+8T(n/8)

…..

≤cnlog$_2$n+nT(1)=O(nlog$_2$n)

4、重点习题解析

9-4 试修改起泡排序算法，在正反两个方向交替进行扫描，即第一趟把排序码最大的对象放到序列的最后，第二趟把排序码最小的对象放到序列的最前面，如此反复进行。

【解答】

静态数据表类定义：

```
#include <iostream.h>
const int DefaultSize=100;
template <class Type> class dataList; //数据表的前视声明
template <class Type> class Element
{ //数据表元素类的定义
    friend class dataList <Type>;
    private:
        Type key; //排序码
        field otherdata; //其它数据成员
    public:
        Type getKey( ) { return key; } //取当前结点的排序码
        void setKey(const Type x) { key=x; } //将当前结点的排序码修改为 x
        Element <Type> & operator = (Element <Type> x) //结点 x 的值赋给 this
        { key=x.key; otherdata=x.otherdata; }
        int operator = = (Element <Type> x) { return key= =x.key; } //判 this 与 x 相等
        int operator <= (Element <Type> x) { return key<=x.key; } //判 this 小于或等于 x
        int operator > (Element <Type> x) { return key>x.key; } //判 this 大于 x
        int operator < (Element <Type> x) { return key<x.key; } //判 this 小于 x
};
template <class Type> class dataList
{ //用顺序表来存储待排序的元素，这些元素的类型是 Type
    private:
        Element <Type> *Vector; //存储待排序元素的向量
        int MaxSize, CurrentSize; //最大元素个数与当前元素个数
        int Partition(const int low, const int high); //用于快速排序的一次划分算法
    public:
        dataList(int MaxSz=DefaultSize) : MaxSize(Maxsz), CurrentSize(0)
        { Vector=new Element <Type> [MaxSize]; } //构造函数
        int length( ) { return CurrentSize; }
        Element <Type> & operator [ ] (int i) { return Vector[i]; }
        void swap(Element <Type> &x, Element <Type> &y)
        { Element <Type> temp=x; x=y; y=temp; }
        void Sort( ) ; //排序
};
```

```
template <class Type> void dataList <Type> :: shaker_Sort( )
{ //奇数趟对表 Vector 从前向后，比较相邻的排序码，遇到逆序即交换
  //直到把参加比较排序码序列中最大的排序码移到序列尾部
  //偶数趟从后向前，比较相邻的排序码，遇到逆序即交换
  //直到把参加比较排序码序列中最小的排序码移到序列前端
    int i=1, j; int exchange;
    while (i<CurrentSize)
    { //起泡排序趟数不超过 n-1 次
        exchange=0; //假定元素未交换
        for (j=i; j<=CurrentSize-i-1; j++)
        { //正向起泡
            if (Vector[j]>Vector[j+1])
            { //发生逆序
                Swap(Vector[j], Vector[j+1]); //交换，最大排序码放在 Vector[n-i]处
                exchange=1; //做“发生了交换”标志
            }
        }
        if (exchange= =0) break; //当 exchange 为 0 则停止排序
        for (j=CurrentSize-i; j>=i; j--)
        { //逆向起泡
            if (Vector[j-1]>Vector[j])
            { //发生逆序
                Swap(Vector[j-1]>Vector[j]); //交换，最小排序码放在 Vector[i-1]处
                exchange=1; //做“发生了交换”标志
            }
        }
        if (exchange= =0) break; //当 exchange 为 0 则停止排序
        i++;
    }
}
```

或者：

```
template <class Type> void dataList <Type> :: shaker_Sort( )
{
    int low=1, high=CurrentSize-1, i, j; int exchange;
    while (low<high)
    { //当比较范围多于一个对象时排序
        j=low; //记忆元素交换位置
        for (i=low; i<high; i++)
        { //正向起泡
            if (Vector[i]>Vector[i+1])
            { //发生逆序
                Swap(Vector[i], Vector[i+1]); //交换
                j=i; //记忆右边最后发生交换的位置 j
            }
```

```
            }
            high=j; //比较范围上界缩小到 j
            for (i=high; i>low; i--)
            { //反向起泡
                if (Vector[i-1]>Vector[i])
                { //发生逆序
                    Swap(Vector[i-1], Vector[i]); //交换
                    j=i; //记忆左边最后发生交换的位置 j
                }
            }
            low=j; //比较范围下界缩小到 j
        }
    }
```

9-9 试设计一个算法，使得在 O(n)的时间内重排数组，将所有取负值的排序码排在所有取正值（非负值）的排序码之前。

【解答】

```
template <class Type> void rearrange(dataList <Type> &L)
{ //数组元素类型 Type 只可能取 int 或 float
    int i=1, j=L.length( )-1;
    while (i!=j)
    {
        while (L[i].getKey( )<0) i++;
        while (L[j].getKey( )>=0) j--;
        swap(L[i], L[j]);
        i++; j--;
    }
}
```

9-10 奇偶交换排序是另一种交换排序。它的第 1 趟对序列中所有奇数项 i 扫描，第 2 趟对序列中的所有偶数项 i 扫描。若 A[i]>A[i+1]，则交换它们。第 3 趟对所有的奇数项，第 4 趟对所有的偶数项，……，如此反复，直到整个序列全部排好序为止。

(1) 这种排序方法结束的条件是什么？

(2) 写出奇偶交换排序的算法。

(3) 当待排序排序码序列的初始排列是从小到大有序，或从大到小有序时，在奇偶交换排序过程中的排序码比较次数是多少？

【解答】

(1) 设有一个布尔变量 exchange，判断在每一次做过一趟奇数项扫描和一趟偶数项扫描后是否有过交换。若 exchange=1，表示刚才有过交换，还需继续做下一趟奇数项扫描和一趟偶数项扫描；若 exchange=0，表示刚才没有交换，可以结束排序。

(2) 奇偶排序的算法：

```
template <Type> void dataList <Type> :: odd-evenSort( )
{
    int i, exchange;
    do {
```

```
            exchange=0;
            for (i=1; i<CurrentSize; i+=2)
            { //扫描所有奇数项
                if (Vector[i]>Vector[i+1])
                { //相邻两项比较，发生逆序
                    exchange=1; //作交换标记
                    swap(Vector[i], Vector[i+1]); //交换
                }
            }
            for (i=0; i<CurrentSize; i+=2)
            { //扫描所有偶数项
                if (Vector[i]>Vector[i+1])
                { //相邻两项比较，发生逆序
                    exchange=1; //作交换标记
                    swap(Vector[i], Vector[i+1]); //交换
                }
            }
        } while (exchange!=0)
    }
```

(3) 设待排序对象序列中总共有 n 个对象。序列中各个对象的序号从 0 开始。则当所有待排序对象序列中的对象按排序码从大到小初始排列时，执行 $m=\lfloor (n+1)/2 \rfloor$ 趟奇偶排序。当所有待排序对象序列中的对象按排序码从小到大初始排列时，执行 1 趟奇偶排序。

在一趟奇偶排序过程中，对所有奇数项扫描一遍，排序码比较 $\lfloor (n-1)/2 \rfloor$ 次；对所有偶数项扫描一遍，排序码比较 $\lfloor n/2 \rfloor$ 次。所以每趟奇偶排序两遍扫描的结果，排序码总比较次数为 $\lfloor (n-1)/2 \rfloor + \lfloor n/2 \rfloor = n-1$。

9-11 请编写一个算法，在基于单链表表示的待排序排序码序列上进行简单选择排序。

【解答】

采用静态链表作为存储表示。用 Vector[0]作为表头结点，各待排序数据对象从 Vector[1]开始存放。算法的思想是每趟在原始链表中摘下排序码最大的结点（几个排序码相等时为最前面的结点），把它插入到结果链表的最前端。由于在原始链表中摘下的排序码越来越小，在结果链表前端插入的排序码也越来越小，最后形成的结果链表中的结点将按排序码非递减的顺序有序链接。

静态链表类定义：

```
template <class Type> class staticlinkList; //静态链表类的前视声明
template <class Type> class Element
{ //静态链表元素类的定义
    friend class staticlinkList <Type>;
    private:
        Type key; //排序码，其它信息略
        int link; //结点的链接指针
```

```
        public:
            Type getKey( ) { return key; } //取当前结点的排序码
            void setKey(Type x) { key=x; } //将当前结点的排序码修改为 x
            int getLink( ) { return link; } //取当前结点的链接指针
            void setLink(int ptr) { link=ptr; } //将当前结点的链接指针为 ptr
    };
    template <class Type> class staticlinkList
    { //静态链表的类定义
        private:
            Element <Type> *Vector; //存储待排序元素的向量
            int MaxSize, CurrentSize; //向量中最大元素个数和当前元素个数
        public:
            staticlinkList(int Maxsz=DefaultSize) : MaxSize(Maxsz), CurrentSize(0)
            { Vector=new Element <Type> [Maxsz]; }
            void Sort( );
    };
    template <class Type> void staticlinkList <Type> :: selectSort( )
    {
        int h=Vector[0].link, p, q, r, s;
        Vector[0].link=0;
        while (h!=0)
        { //原始链表未扫描完
            p=s=h; q=r=0;
            while (p!=0)
            { //扫描原始链表，寻找排序码最大的结点
                if (Vector[p].data>Vector[s].data) //记忆当前找到的排序码最大结点
                { s=p; r=q; }
                q=p; p=Vector[p].link;
            }
            if (s= =h) h=Vector[h]; //排序码最大的结点是原始链表前端结点，摘下
            else Vector[r].link=Vector[s].link;
            //排序码最大的结点是原始链表表中结点，摘下
            Vector[s].link=Vector[0].link; //结点 s 插入到结果链表的前端
            Vector[0].link=s;
        }
    }
```

9-12 若参加锦标赛排序的排序码有 11 个，为了完成排序，至少需要多少次排序码比较？

【解答】

对于有 n(n>0)个数据的序列，锦标赛排序选最小数据需进行 n-1 次数据比较，以后每选一个数据，进行数据比较的次数，均需$\lfloor \log_2 n \rfloor - 1$次（在外结点层无比较）。对于有 11 个排序码的序列，第一次选具最小排序码的数据，需进行 10 次排序码比较，以后在剩下的序列中每选一个具最小排序码的数据，都需进行$\lfloor \log_2 11 \rfloor - 1 = 2$次排序码比较。因此，为了

完成排序，需要 10+2*10=30 次排序码比较。

9-15 如果只想在一个有 n 个元素的任意序列中得到其中最小的第 k(k<n)个元素之前的部分排序序列，那么最好采用什么排序方法？为什么？例如，有这样一个序列：{503, 017, 512, 908, 170, 897, 275, 653, 612, 154, 509, 612*, 677, 765, 094}，要得到其第 4 个元素之前的部分有序序列：{017, 094, 154, 170}，用所选择的算法实现时，要执行多少次比较？

【解答】

一般来讲，当 n 比较大且要选的数据 k 远小于 n 时，采用堆排序方法中的调整算法 FilterDown( )最好。但当 n 比较小时，采用锦标赛排序方法更好。

例如，对于序列{57, 40, 38, 11, 13, 34, 48, 75, 6, 19, 9, 7}，选最小的数据 6，需形成初始堆，进行 18 次数据比较；选次小数据 7 时，需进行 4 次数据比较；再选数据 9 时，需进行 6 次数据比较；选数据 11 时，需进行 4 次数据比较。

但如果选用锦标赛排序，对于有 n(n>0)个数据的序列，选最小数据需进行 n-1 次数据比较，以后每选一个数据，进行数据比较的次数，均需$\lfloor \log_2 n \rfloor - 1$次。例如，同样 12 个数据，第一次选最小的数据 6，需进行 11 次数据比较，以后选 7, 9, 11 时，都是$\lfloor \log_2 12 \rfloor - 1 = 2$次数据比较。

9-17 设有 n 个待排序元素存放在一个不带表头结点的单链表中，每个链表结点只存放一个元素，头指针为 r。试设计一个算法，对其进行二路归并排序，要求不移动结点中的元素，只改各链结点中的指针，排序后 r 仍指示结果链表的第一个结点。

【解答】

(1) 两路归并算法：

```
template <Type> void staticlinkList <Type> :: merge(int ha, int hb, int &hc)
{ //合并两个以 ha 和 hb 为表头指针的有序链表，结果链表的表头由 hc 返回
    int pa, pb, pc;
    if (Vector[ha].data<=Vector[hb].data) //确定结果链的表头
    { hc=ha; pa=Vector[ha].link; pb=hb; }
    else { hc=hb; pb=Vector[hb].link; pa=ha; }
    pc=hc; //结果链的链尾指针
    while ((pa!=0) && (pb!=0))
    { //两两比较，小者进结果链
        if (Vector[pa].data<=Vector[pb].data)
        { Vector[pc].link=pa; pc=pa; pa=Vector[pa].link; }
        else { Vector[pc].link=pb; pc=pb; pb=Vector[pb].link; }
    }
    if (pa!=0) Vector[pc].link=pa; //pb 链处理完，pa 链链入结果链
    else Vector[pc].link=pb; //pb 链处理完，pb 链链入结果链
}
```

(2) 归并排序主程序

```
template <class Type> void staticlinkList <Type> :: merge_sort( )
{
    int r, s, t; Queue <int> Q;
    if (Vector[0].link= =0) return;
```

```
        s=Vector[0].link; Q.EnQueue(s); //链表第一个结点进队列
        while (1)
        {
            t=Vector[s].link; //结点 t 是结点 s 的下一个链中结点
            while ((t!=0) && (Vector[s].data<=Vector[t].data))
            { s=t; t=Vector[t].link; } //在链表中寻找一段有序链表
            Vector[s].link=0; s=t;
            if (t!=0) Q.EnQueue(s); //存在一段有序链表，截取下来进队列
            else break; //到链尾
        }
        while (!Q.IsEmpty( ))
        {
            r=Q.getFront( ); Q.DlQueue( ); //从队列退出一个有序链表的表头 r
            if (Q.IsEmpty( )) break; //队列空，表示排序处理完成，退出
            s=Q.getFront( ); Q.DlQueue( ); //从队列再退出一个有序链表的表头 s
            merge(r, s, t); Q.EnQueue(t); //归并两个有序链表后结果链表进队列
        }
        Vector[0].link=r;
    }
```

9-19 试编写一个算法，将对象序列($x_1, x_2, \ldots, x_n$)循环右移 p 个位置，$0 \le p \le n$。要求该算法的时间复杂度为 O(n)，而空间复杂度为 O(1)。

【解答】

```
    void siftk(int X[ ], int n, int p)
    {
        int i, j, m=GCD(n, p); //寻找 n 与 p 的最大公约数
        for (int k=0; k<m; k++)
        {
            int temp=X[k]; i=k; j=(n+i-p)%n;
            while (j!=k)
            { X[i]=X[j]; i=j; j=(n+j-p)%n; }
            X[i]=temp;
        }
    }
    int GCD(int dividend, int divisor)
    { //函数返回 dividend 与 divisor 的最大公约数
      //若 dividend 与 divisor 互质，则函数返回 1
        if (dividend<divisor) { int temp=dividend; dividend=divisor; divisor=temp; }
        int remainder=dividend%divisor;
        while (remainder!=0)
        {
            dividend=divisor; divisor=remainder;
            remainder=dividend%divisor;
        }
        return divisor;
```

```
}
```

9-21 在已排好序的序列中，一个对象所处的位置取决于具有更小排序码的对象的个数。基于这个思想，可得计数排序方法。该方法在声明对象时为每个对象增加一个计数域 count，用于存放在已排好序的序列中该对象前面的对象数目，最后依 count 域的值，将序列重新排列，就可完成排序。试编写一个算法，实现计数排序。并说明对于一个有 n 个对象的序列，为确定所有对象的 count 值，最多需要做 n(n-1)/2 次排序码比较。

【解答】

```
template <class Type> void datalist <Type> :: count_sort( )
{ //initList 是待排序表，resultList 是结果表
    int i, j;
    int *c=new datalist <Type>; //c 是存放计数排序结果的临时表
    for (i=0; i<CurrentSize; i++) Vector[i].count=0; //初始化，计数值都为 0
    for (i=0; i<CurrentSize-1; i++)
        for (j=i+1; j<CurrentSize; j++)
            if (Vector[j].key<Vector[i].key) Vector[i].count++;
            else Vector[j].count++; //统计
    for (i=0; i<CurrentSize; i++) //在 c->Vector[ ]中各就各位
        c->Vector[Vector[i].count]=Vector[i];
    for (i=0; i<CurrentSize; i++) Vector[i]=c->Vector[i]; //结果复制会当前表对象中
    delete c;
}
```

9-23 如果某个文件经内排序得到 80 个初始归并段，试问：

(1) 若使用多路归并并执行 3 趟完成排序，那么应取的归并路数至少应为多少？

(2) 如果操作系统要求一个程序同时可用的输入/输出文件的总数不超过 15 个，则按多路归并至少需要几趟可以完成排序？如果限定这个趟数，可取的最低路数是多少？

【解答】

(1) 设归并路数为 k 的初始归并段个数 m=80，根据归并趟数计算公式 $S=\lceil \log_k m \rceil = \lceil \log_k 80 \rceil = 3$ 得：$k^3 \geq 80$。由此，解得 k≥5，即应取的归并路数至少为 5。

(2) 设多路归并的归并路数为 k，需要 k 个输入缓冲区和 1 输出缓冲区。1 个缓冲区对应 1 个文件，有 k+1=15，因此 k=14，可做 14 路归并。由 $S=\lceil \log_k m \rceil = \lceil \log_{14} 80 \rceil = 2$。即至少需 2 趟归并可完成排序。

若限定这个趟数，由 $S=\lceil \log_k 80 \rceil = 2$，有 $80 \leq k^2$，可取的最低路数为 9。即要在 2 趟内完成排序，进行 9 路排序即可。

9-24 假设文件有 4500 个记录，在磁盘上每个页块可放 75 个记录。计算机中用于排序的内存区可容纳 450 个记录。试问：

(1) 可建立多少个初始归并段？每个初始归并段有多少记录？存放于多少个页块中？

(2) 应采用几路归并？请写出归并过程及每趟需要读写磁盘的页块数。

【解答】

(1) 文件有 4500 个记录，计算机中用于排序的内存区可容纳 450 个记录，可建立的初始归

并段有 4500/450=10 个。每个初始归并段中有 450 个记录，存于 450/75=6 个页块中。

(2) 内存区可容纳 6 个页块，可建立 6 个缓冲区，其中 5 个缓冲区用于输入，1 个缓冲区用于输出，因此可采用 5 路归并。

9-27 给出 12 个初始归并段，其长度分别为 30, 44, 8, 6, 3, 20, 60, 18, 9, 62, 68, 85。现要做 4 路外归并排序。试画出表示归并过程的最佳归并树，并计算该归并树的带权路径长度 WPL。

【解答】

设初始归并段个数 n=12，外归并路数 k=4，计算(n-1)%(k-1)=11%3=2≠0，必须补 k-2-1=1 个长度为 0 的空归并段，才能构造 k 路归并树。此时，归并树的内结点应有(n-1+1)/k-1=12/3=4 个。

WPL=(3+6+8)*3+(9+18+20+30+44+60+62)*2+(68+85)*1

=51+486+153=690