沈阳航空航天大学

课 程 设 计 报 告

课程设计名称：**数据结构课程设计**

课程设计题目：**排序综合**

院（系）： 计算机学院

专　　业： 网络工程

班　　级： 1602

学　　号： 163401040220

姓　　名： 左剑凯

指导教师： 高利军

沈阳航空航天大学

课程设计任务书

课程设计名称	数据结构课程设计			专业	网络工程
学生姓名	左剑凯	班级	网络1602	学号	16340104020
题目名称	排序综合				
起止日期	2017 年 12 月 25 日起至 2017 年 1 月 5 日止				

课设内容和要求：

利用随机函数产生N个随机整数（10个以上），对这些数进行多种方法进行排序。

具体包括：

1、至少采用五种方法实现上述问题求解（可采用的方法有插入排序、希尔排序、起泡排序、快速排序、选择排序、堆排序、归并排序）。并把排序后的结果保存到不同文件中；

2、统计每一种排序方法的性能（以上机运行所花费的时间为准进行比对），找出其中两种较快的方法；

3、要求采用图形界面演示；

4、完成设计任务并书写课程设计报告。

参考资料：

[1] 严蔚敏，吴伟民编著，数据结构C语言版[M]，北京：清华大学出版社，1997

[2] 谭浩强编著，C程序设计[M],北京：清华大学出版社，2010

[3] 徐金梧，杨德斌，徐科编著，Turbo C实用大全[M]，北京：机械工业出版社

系审核意见：	**系主任签字（盖章）：**
指导教师（签名）	年 月 日
学 生（签名）	年 月 日

目　　录

1 题目介绍

1.1 问题内容与要求

利用随机函数产生 N 个随机整数（10 个以上），对这些数进行多种方法进行排序。

具体包括：

1、至少采用五种方法实现上述问题求解（可采用的方法有插入排序、希尔排序、起泡排序、快速排序、选择排序、堆排序、归并排序）。并把排序后的结果保存到不同文件中；

2、统计每一种排序方法的性能（以上机运行所花费的时间为准进行比对），找出其中两种较快的方法；

3、要求采用图形界面演示；

4、完成设计任务并书写课程设计报告。

1.2 题目分析及功能设想

1. 题目分析

（1） 设计一个的菜单将在实现的功能显示出来，并有选择提示；

（2） 分别实现插入排序、希尔排序、起泡排序、快速排序、选择排序、堆排序、归并排序算法；

（3） 通过多种测试数据，对各种排序算法的时间复杂度和空间复杂度进行比较。

2. 功能设想

根据题目要求应设计一个可以用七种算法进行排序的程序，程序还能保存和读取文件，保存文件的时候应该注意不能覆盖保存随机数的文件，应该写相应代码进行保护，程序还要输出排序花费的时间，应该根据排序随机数的数量选择合适的精度，程序应该有文字上的交互。

2　系统功能模块图

2.1 总体功能模块

排序综合系统功能模块由显示菜单、输入序号、显示排序后的的数据和时间效率、显示各个排序法对同一组数据两种较快的方法和退出等 5 部分组成。

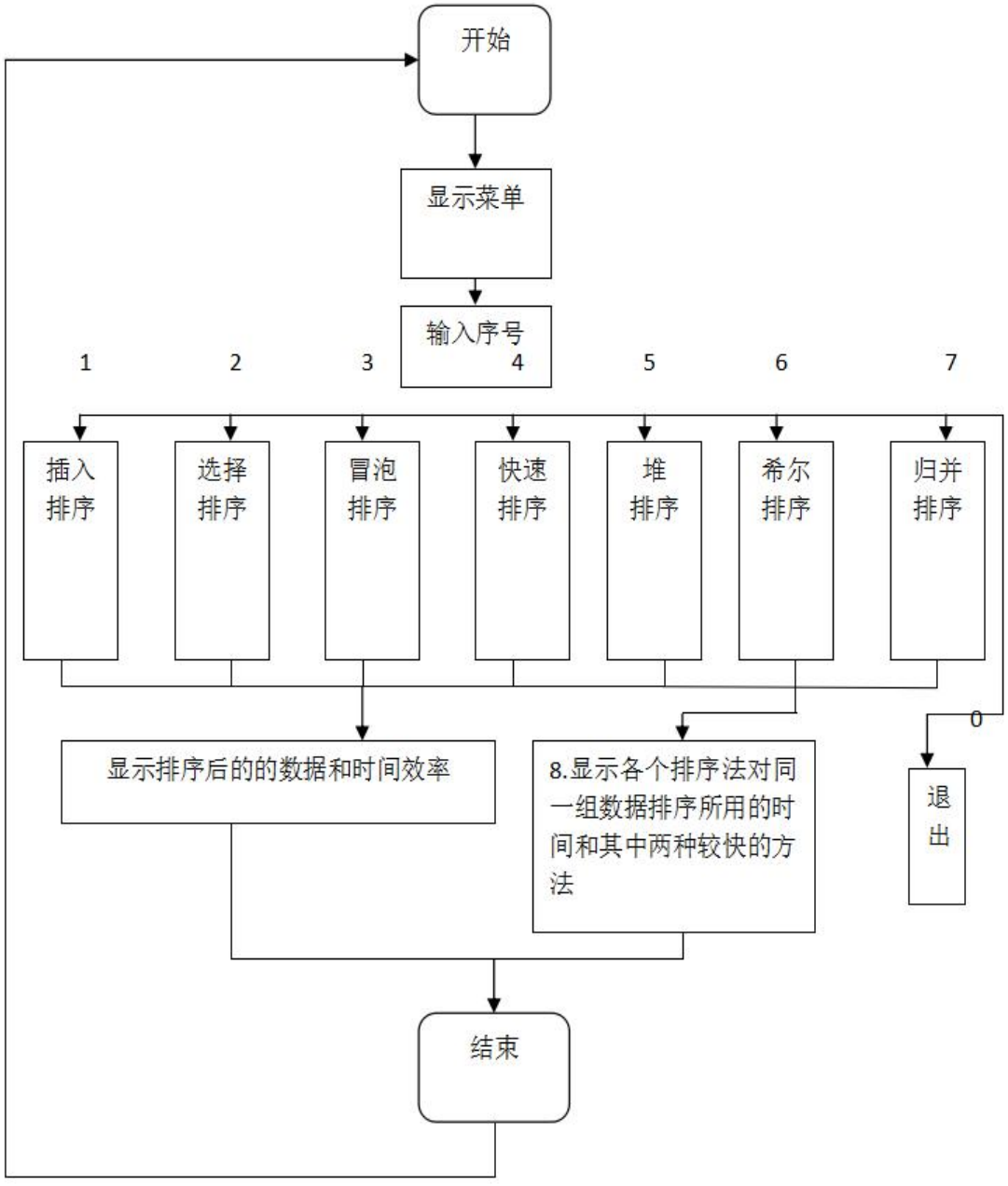

图 2.1 总功能模块图

2.2 具体排序及时间比较模块

设计 7 个高精度时间函数，分别测试各种排序的时间消耗，分别为选择排序时间计算函数（T-Select Sort）、冒泡排序时间计算函数（T-Bubble Sort）、插入排序时间计算函数（T-Insert Sort）、堆排序时间计算函数（T-Heap Sort）、快速排序时间计算函数（T-Quick Sort）、希尔时间计算函数（T-Shell Sort）和归并时间计算函数（T-Merge Sort）计算后存放在 time[]时间数组中，然后对其排序，其时间排序函数为（T-Sort），输出两种较快的排序时间及对应的名称。

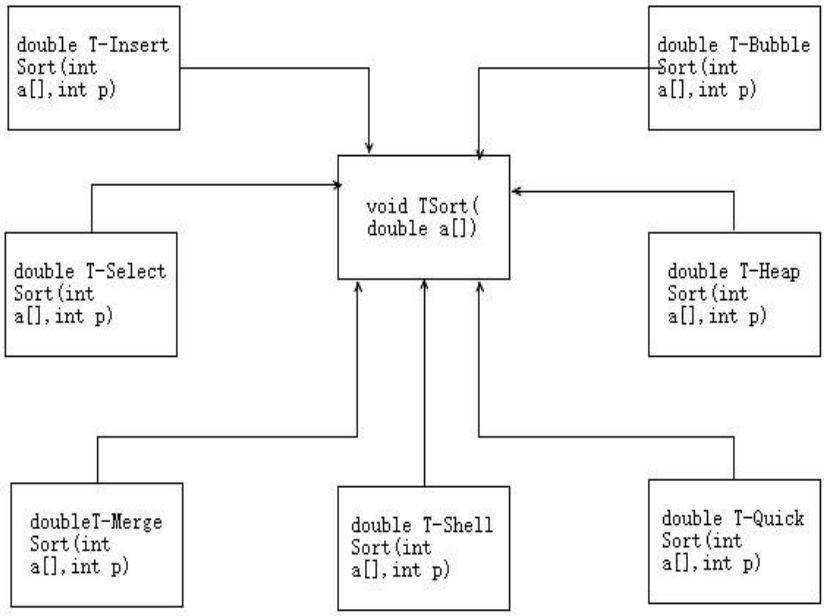

图 2.2 时间比较功能模块图

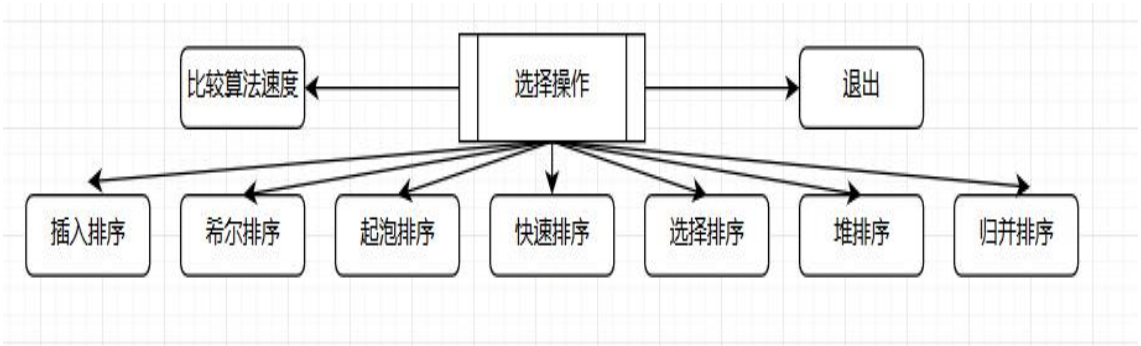

图 2.3 选择操作功能模块图示

3 数据结构设计

3.1 变量及数据类型

1.预定义常量和自定义类型：

#define N 150

2.基本函数的算法用以下形式表示：函数类型 函数名（函数参数表）//算法说明{语句序列}//函数名。

3.定义 int b,t,i,j; b 为记录交换的次数，t 为记录排序的趟数,i 为排序的数据，j 为暂存数据的临时变量。

4. 输入初始数据函数中定义 int k, j，k 为输入数据个数，j 为输入的数据。

5.快速排序中定义 static int w=0,int *low，*high。

6.在选择排序中定义 int k，临时储存 i 的值。

7.在排序时间消耗测试的函数和主函数中定义了 int p，为菜单的序号。

8.设置随机种子（1---3000） a[ ]=rand()%3000+1

9.#include<windows.h> windows 系统的高精度定时器

10.LARGE_INTEGER x={0} 表示一个 64 位有符号的整数值

11.Query Performance Frequency(&x) 检索性能计数器的频率

12. Query Performance Counter(& start) 检索性能计数器的当前值，该值是高分辨率时间戳，可用于时间间隔测量

3.2 排序函数设计

3.2.1 插入排序（Insert Sort）

将一个记录插入到已经排好序的有序表中，从而得到一个新的、记录数增 1 的有序表。设整个排序有 n 个数，则进行 n-1 趟插入，即：先将序列中的第 1 个记录看成是一个有序的子序列，然后从第 2 个记录起逐个进行插入，直至整个序列变成按关键字非递减有序序列为止。

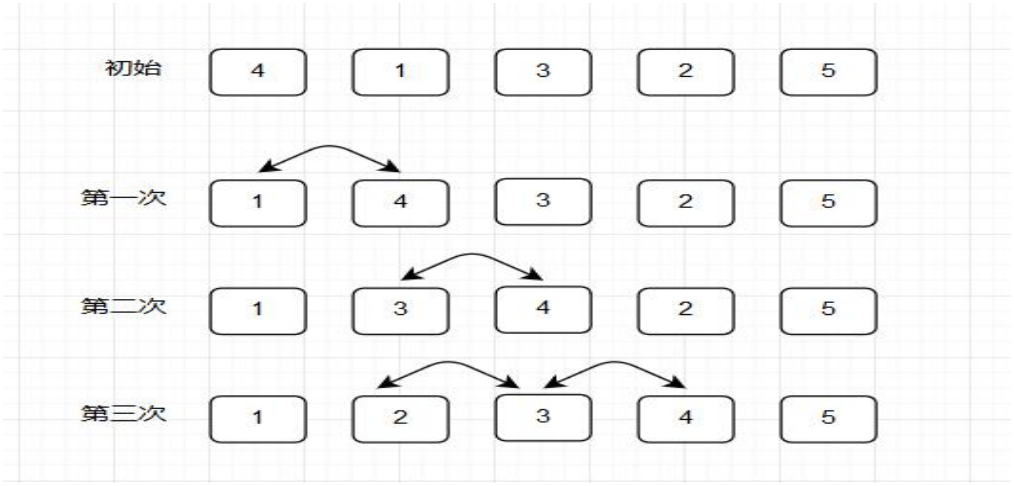

图 3.1 插入排序图解

具体代码：

```
void Insert_sort(int a[])        //1.插入排序(如同插纸牌)
{
    int i,j,temp;
    for(i=1;i<N;i++)     //从第一张牌开始循环到最后一张,第零张牌已经在手里了
    {
        temp=a[i];       //摸下一张牌
        for(j=i;j>0&&a[j-1]>temp;j--)
                a[j]=a[j-1];       //移出空位,向后错一位
        a[j]=temp;                 //新牌落位
    }
}
```

3.2.2 选择排序(Select Sort)

通过 n-1 次关键字间的比较，从 n-i+1 个记录中选出关键字最小的记录，并和第 i 个记录交换。

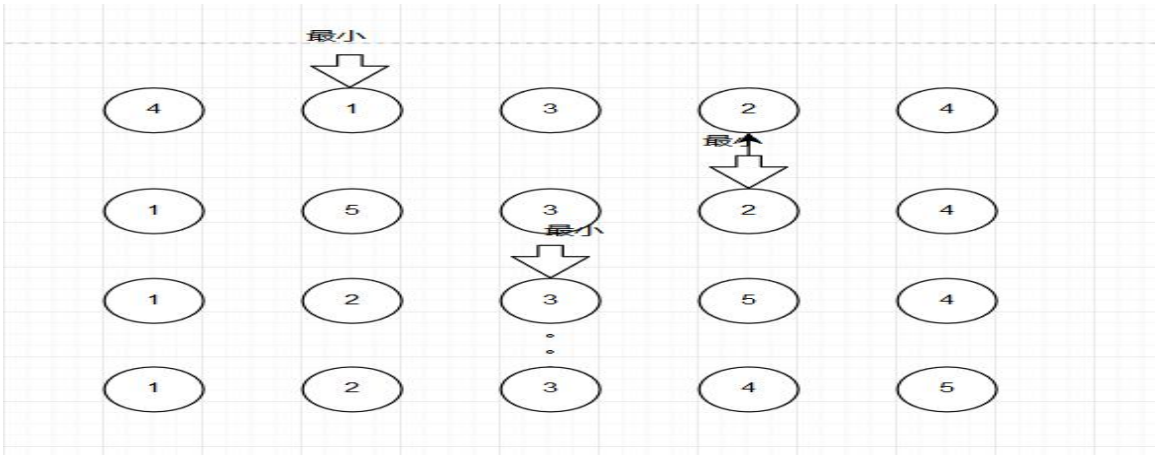

图 3.2 选择排序图解

具体代码：

```
void Select_sort(int a[])    //2.选择排序(找到最小元素的位置)
{
      int i,j,k;
      for(i=0;i<N;i++)
     {
     k=i;
           for(j=i+1;j<N;j++)
        if(a[j]<a[k])
          k=j;
                 if(k!=i)
        {
                    int temp;
                    temp=a[k];
                    a[k]=a[i];
                    a[i]=temp;
        }
     }
}
```

3.2.3 冒泡排序(Bubble Sort)

如果有 n 个数，则要进行 n-1 趟比较，在第 1 趟比较中要进行 n-1 次两次比较，在第 j 趟比较中要进行 n-j 此两两比较。

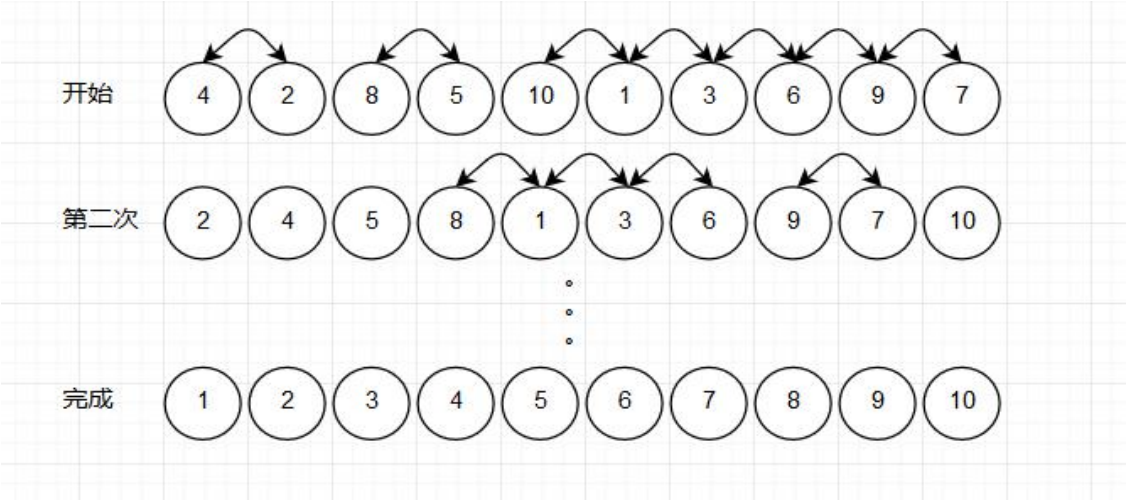

图 3.3 冒泡排序图解

具体代码：

```
void Bubble_sort(int a[]) //3.冒泡排序
{
    int i,j,temp;
for (i=0;i<N-1;i++)
```

```
{
    for(j=N-1;j>i;j--)       //比较,找出本趟最小关键字的记录,从最后一个向前找
     if(a[j]<a[j-1])         //进行交换,将最小关键字记录前移
     {
       temp=a[j];
       a[j]=a[j-1];
       a[j-1]=temp;
     }
}
}
```

3.2.4 快速排序（Quick Sort）

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要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

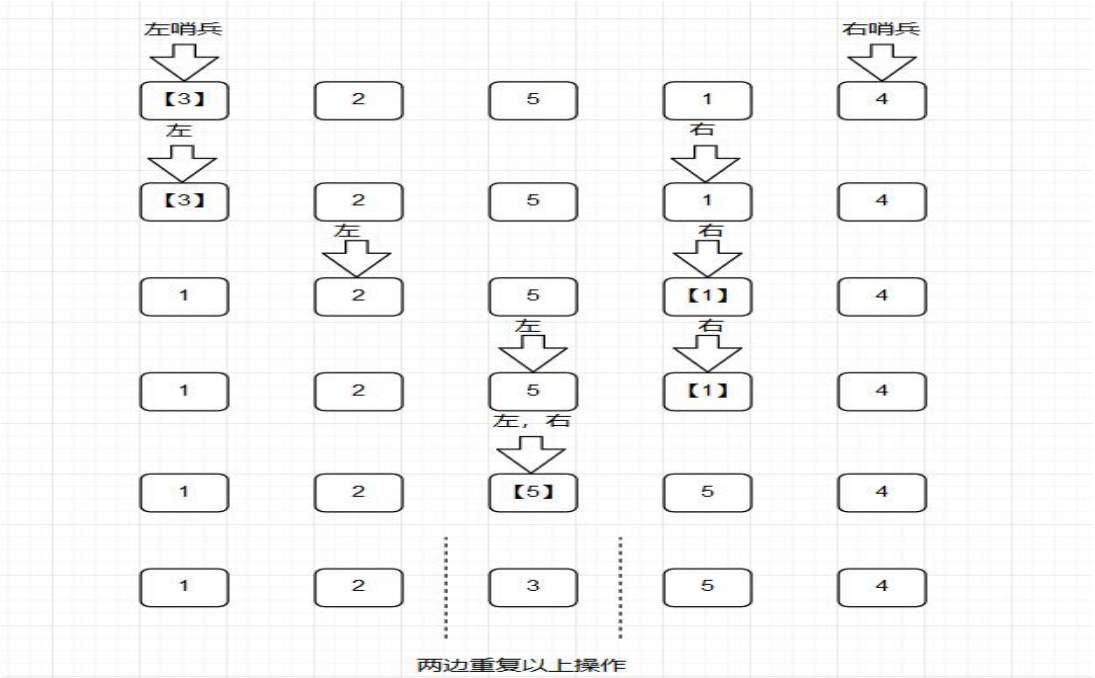

图 3.4 快速排序图解

具体代码：

```
void Quick_sort(int a[],int n)     //4.快速排序
{
        int i,j,low,high,temp,top=-1;
```

```
struct node
{
    int low,high;
 }st[N];
top++;
st[top].low=0;st[top].high=n-1;
while(top>-1)
{          low=st[top].low;high=st[top].high;
      top--;
      i=low;j=high;
      if(low<high)
      {          temp=a[low];
            while(i!=j)
            {     while(i<j&&a[j]>temp)j--;
                  if(i<j){a[i]=a[j];i++;}
                  while(i<j&&a[i]<temp)i++;
                  if(i<j){a[j]=a[i];j--;}
            }
            a[i]=temp;
            top++;st[top].low=low;st[top].high=i-1;
            top++;st[top].low=i+1;st[top].high=high;
      }
}}
```

3.2.5 堆排序（Heap Sort）

是指利用堆这种数据结构所设计的一种排序算法。堆是一个近似完全二叉树的结构，并同时满足堆性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

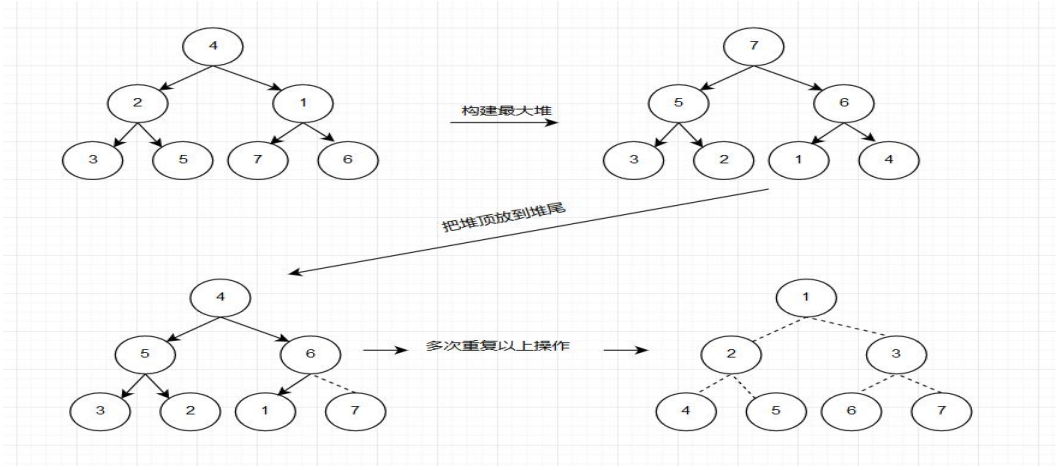

图 3.5 堆排序图解

具体代码：

```
void Heap_sort(int a[], int size)        //   5.堆排序(对选择的改进)
{
        Build Heap(a,size);                  // 建立最大堆
        for(int i=size-1;i>0;i--)
        {
                //swap(a[0],a[i]);   //a[0]根节点存的是最大元素，a[i]是当前最后一个元素的下标
                    int t;
                  t=a[0];
                  a[0]=a[i];
                  a[i]=t;
                Heap Adjust(a,0,i-1);//把剩下的元素再调整成最大堆
        }
}
```

3.2.6 希尔排序（Shell Sort）

希尔排序是一种对直接插入算法改进的算法，选取步长进行分组排序，能够减少插入排序的比较次数。对于 n 个待排序的数列，取一个小于 n 的整数 gap(gap 被称为步长)将待排序元素分成若干个组子序列，所有距离为 gap 的倍数的记录放在同一个组中；然后，对各组内的元素进行直接插入排序。 这一趟排序完成之后，每一个组的元素都是有序的。然后减小 gap 的值，并重复执行上述的分组和排序。重复这样的操作，当 gap=1 时，整个数列就是有序的。

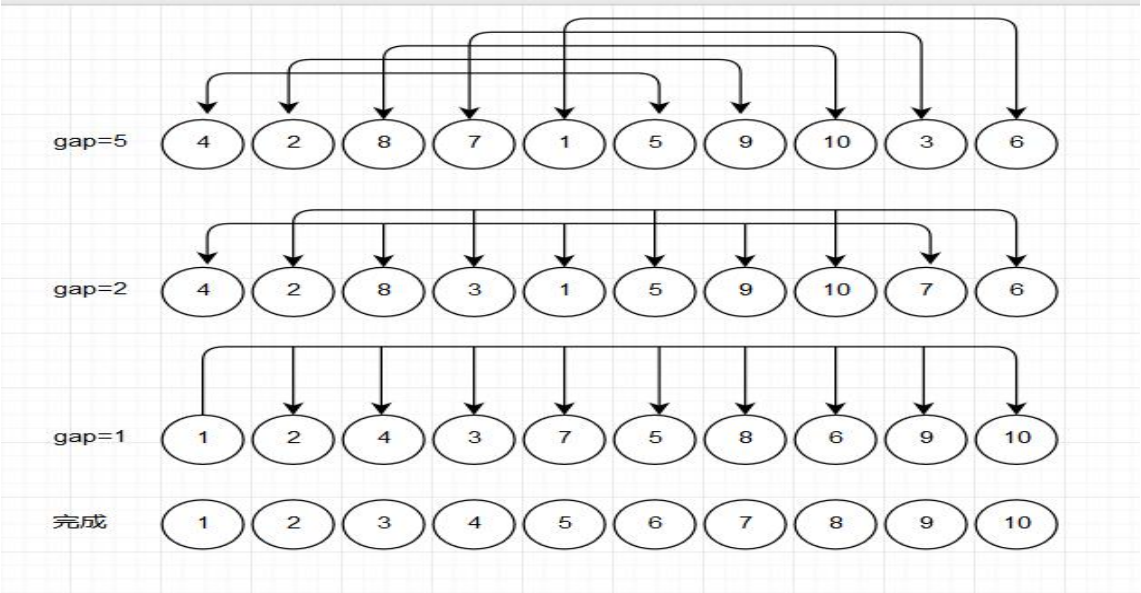

图 3.6 希尔排序图解

具体代码：

```
void Shell_sort(int a[], int n)        //6.希尔排序(对插入的改进),利用插入排序的简单，并克服插入排序每次只交换相邻元素的缺点
{
        int i,j,gap;
        for(gap=N/2;gap>0;gap/=2)//希尔增量序列
            for(i=0;i<gap;i++)
                for(j=i+gap;j<n;j+=gap)
                {
                    if(a[j]<a[j-gap])
                    {
                            int t=a[j];
                            int k=j-gap;
                            while(k>=0 && a[k]>t)
                            {
                                    a[k+gap]=a[k];
                                    k-=gap;
                            }
                            a[k+gap]=t;
                    }
                }
}
```

3.2.7 归并排序（Merge Sort）

归并排序是一种采用分治法，递归分割数列，使每个子序列有序，达到整个数列有序的算法。递归折半分解数列直到只有一个元素，这一个元素自然是有序的，然后再合并得到有序数组。

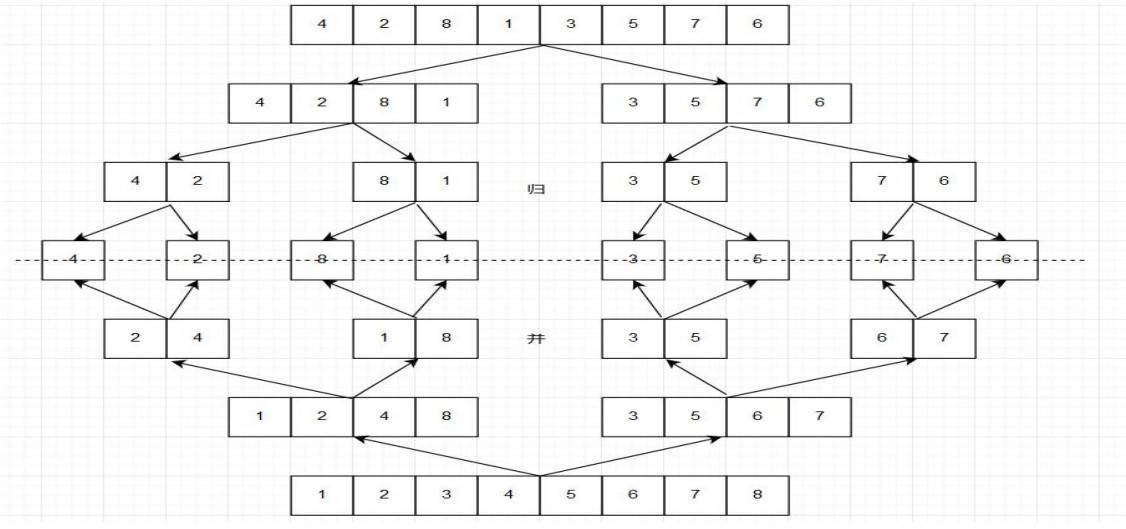

图 3.7 归并排序图解

具体代码：

```
void Merge Array(int a[], int first, int mid, int last, int temp[])
{
        int i=first, j=mid+1, m=mid, n=last, k=0;
        while(i<=m && j<=n)   // m,n 分别是左右终点
        {
                if(a[i]<=a[j])   // i 是左边，j 是右边
                    temp[k++]=a[i++];
                else
                    temp[k++]=a[j++];
        }
        while(i<=m)       //直接复制左边剩下的
              temp[k++]=a[i++];
        while(j<=n)     //直接复制右边剩下的
              temp[k++]=a[j++];
        for(i=0;i<k;i++)
             a[first+i]=temp[i];   //导回
}
void merge Sort(int a[],int first, int last, int temp[])//递归，劈开
{
        if(first<last)
        {
                int mid = (first+last)/2;
                Merge Sort(a,first,mid,temp);
                Merge Sort(a,mid+1,last,temp);
                Merge Array(a,first,mid,last,temp);
        }
}
void Merge_sort(int a[], int l)     // 7.归并排序
{
        int *p = new int[l];
        Merge Sort(a,0,len-1,p); //0，len-1  分别为最左和最右
}
```

3.3 时间函数设计

```
double T_time(int c[],int p) //时间计算函数
{
LARGE_INTEGER x={0};          Query Performance Frequency(&x);       //表示一个 64 位有符号的整数值   // 检索性能计数器的频率
LARGE_INTEGER start={0};    Query Performance Counter(& start);//检索性能计数器的当前值，该值是高分辨率时间戳，可用于时间间隔测量
 switch(p)
```

```
    {
         case 1:Insert_sort(c);break;
        case 2:Select_sort(c);break;
         case 3:Bubble_sort(c);break;
        case 4:Quick_sort(c,N);break;
         case 5:Heap_sort(c,N);break;
        case 6:Shell_sort(c,N);;break;
        case 7:Merge_sort(c,N);break;
    }
LARGE_INTEGER now={0};        Query Performance Counter(&now);
       double times=now.Quad Part - start.Quad Part;
       times/=x.Quad Part;

      return times;
}
```

4　功能模块设计

4.1 主菜单设计

为了实现排序操作功能，首先设计一个含有多个菜单项的主菜单程序，然后再为这些菜单配上相应的功能。
程序运行后，菜单的内如和输入提示，如下：

********** 欢迎来到综合排序系统! **********
菜　单

********** (1)---插入排序　　**********
********** (2)---选择排序　　**********
********** (3)---冒泡排序　　**********
********** (4)---快速排序　　**********
********** (5)---堆排序　　**********
********** (6)---希尔排序　　**********
********** (7)---归并排序　　**********
********** (8)---时间效率比较　**********
********** (9)---显示随机数　　**********
********** (0)---退出　　**********
**********设计者：左剑凯 163401040220**********
****请在上述序号中选择一个并输入:*****

4.2 选择功能函数设计

```
void Funtion ()
{
    int a[N],num;
    Create_random_num(a);
      while(1)
   {
      double t[7],tt[7];//时间数组
          scanf("%d",&num);
        switch(num)
      {
            case 0:printf("=====>谢谢使用!\n");break;
            case  1:tt[0]=t[0]=T_insert_sort(a,num);printf("\n 请 按 任 意 键 继续...");getchar();break;
            case  2:tt[1]=t[1]=T_select_sort(a,num);printf("\n 请 按 任 意 键 继续...");getchar();break;
            case  3:tt[2]=t[2]=T_bubble_sort(a,num);printf("\n 请 按 任 意 键 继
```

```
续...");getchar();break;
            case 4:tt[3]=t[3]=T_quick_sort(a,num);printf("\n 请按任意键继续...");getchar();break;
            case 5:tt[4]=t[4]=T_heap_sort(a,num);printf("\n 请按任意键继续...");getchar();break;
            case 6:tt[5]=t[5]=T_shell_sort(a,num);printf("\n 请按任意键继续...");getchar();break;
            case 7:tt[6]=t[6]=T_merge_sort(a,num);printf("\n 请按任意键继续...");getchar();break;
            case 8:TSort(t);
        printf("\n\n");
      {
        printf("排序这组数据两种较快的排序法分别是：\n");
    while(1)
    {
        if(t[0]==tt[0]){ printf("插入排序:%f 秒!\n",t[0]); break; }
        if(t[0]==tt[1]){ printf("选择排序:%f 秒!\n",t[0]); break; }
        if(t[0]==tt[2]){ printf("冒泡排序:%f 秒!\n",t[0]); break; }
        if(t[0]==tt[3]){ printf("快速排序:%f 秒!\n",t[0]); break; }
        if(t[0]==tt[4]){ printf("堆排序:%f 秒!\n",t[0]);    break; }
        if(t[0]==tt[5]){ printf("希尔排序:%f 秒!\n",t[0]);break;   }
        if(t[0]==tt[6]){ printf("归并排序:%f 秒!\n",t[0]);break;   }
      }
    while(1)
         {
             if(t[1]==tt[0]) {printf("插入排序%f 秒!\n",t[1]); break;}
             if(t[1]==tt[1]) {printf("选择排序%f 秒!\n",t[1]); break;}
             if(t[1]==tt[2]) {printf("冒泡排序%f 秒!\n",t[1]); break;}
             if(t[1]==tt[3]) {printf("快速排序%f 秒!\n",t[1]); break;}
             if(t[1]==tt[4]) {printf("堆排序%f 秒!\n",t[1]);    break;}
             if(t[1]==tt[5]) {printf("希尔排序%f 秒!\n",t[1]); break;}
             if(t[1]==tt[6]) {printf("归并排序%f 秒!\n",t[1]); break;}
         }
       }
 printf("\n 请按任意键继续...");
 srand((int)time(NULL)); for(int i=0;i<N;i++)    a[i]=rand()%5000+1;     break; // 重新又定义了一组随机值
   case 9:show(a);    FILE *fp;fp=fopen("随机数.txt","w");
for(int  i=0;i<N;i++)      {if((i-1)%10==9)      fprintf(fp,"\n");  fprintf(fp,"%-7d  ",a[i]);}
fclose(fp);
getchar();printf("\n 请按任意键继续...");getchar();break;
default:Wrong();printf("\n 请按任意键继续...");getchar();break;
    }  }      }
```

5　调试与运行结果

5.1 控制台运行结果

为了方便显示效果，调试所用随机数个数为 150 个，其数量可由操作者自己随意设定。

图 5.1 菜单界面

图 5.2 插入排序结果及时间显示

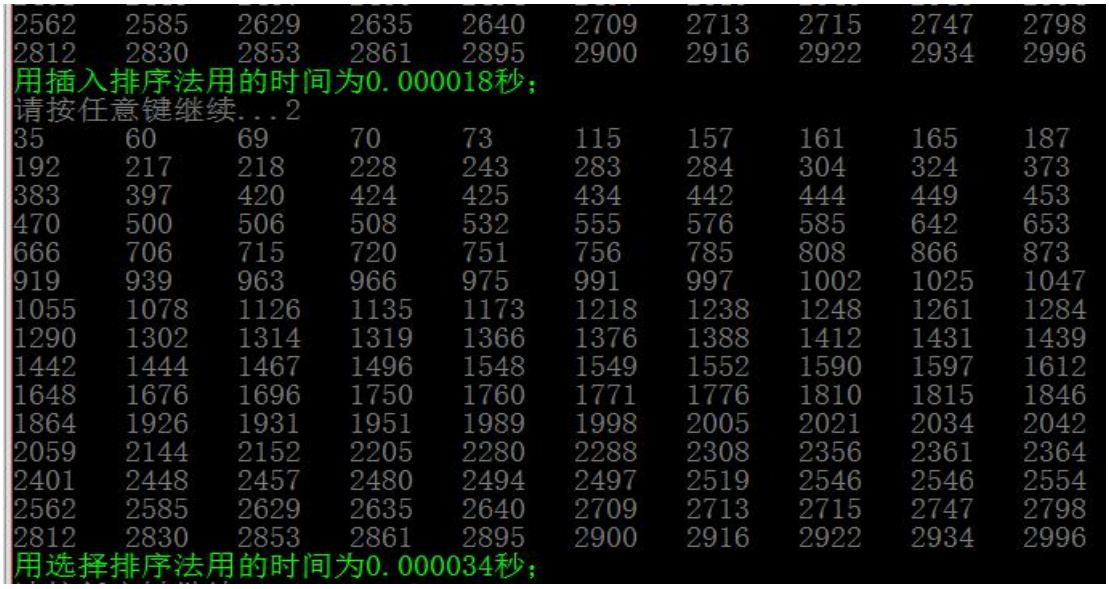

图 5.3 选择排序结果及时间显示

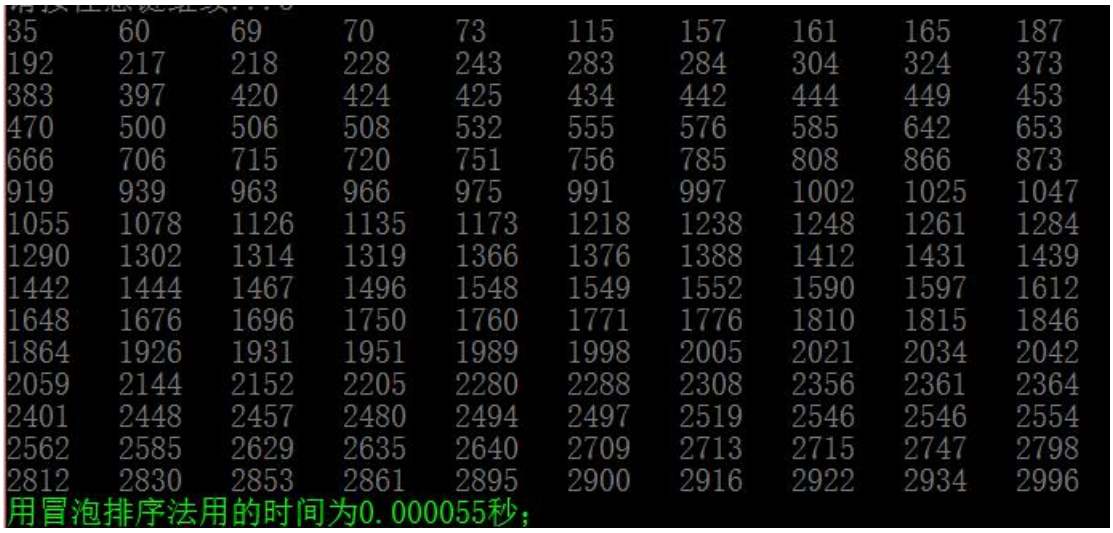

图 5.4 冒泡排序结果及时间显示

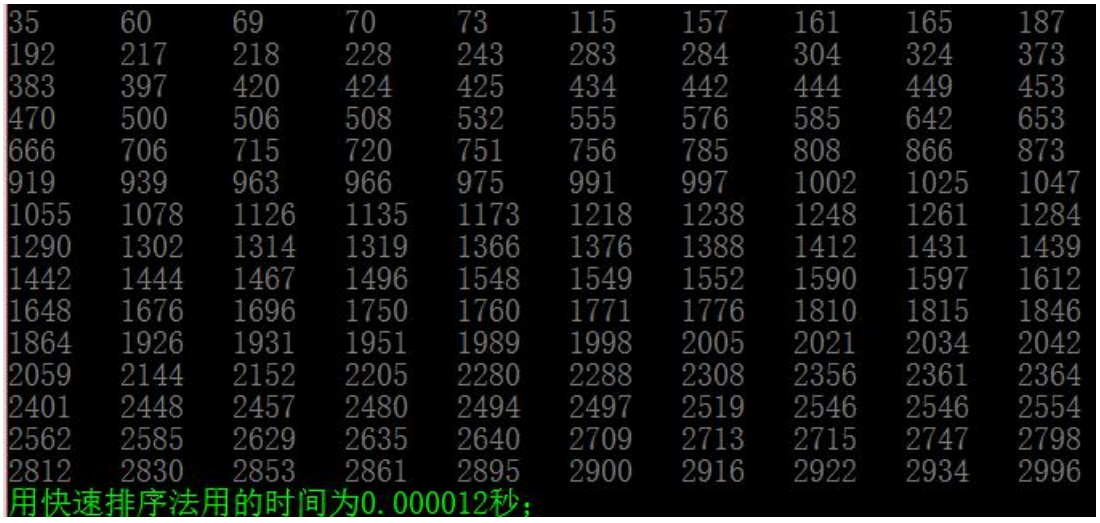

图 5.5 快速排序结果及时间显示

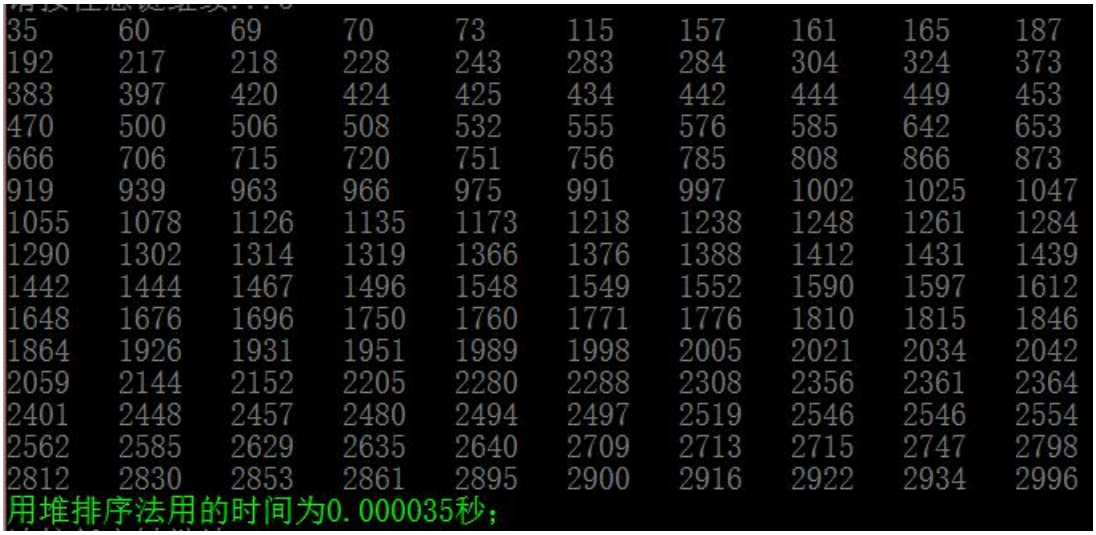

图 5.6 堆排序结果及时间显示

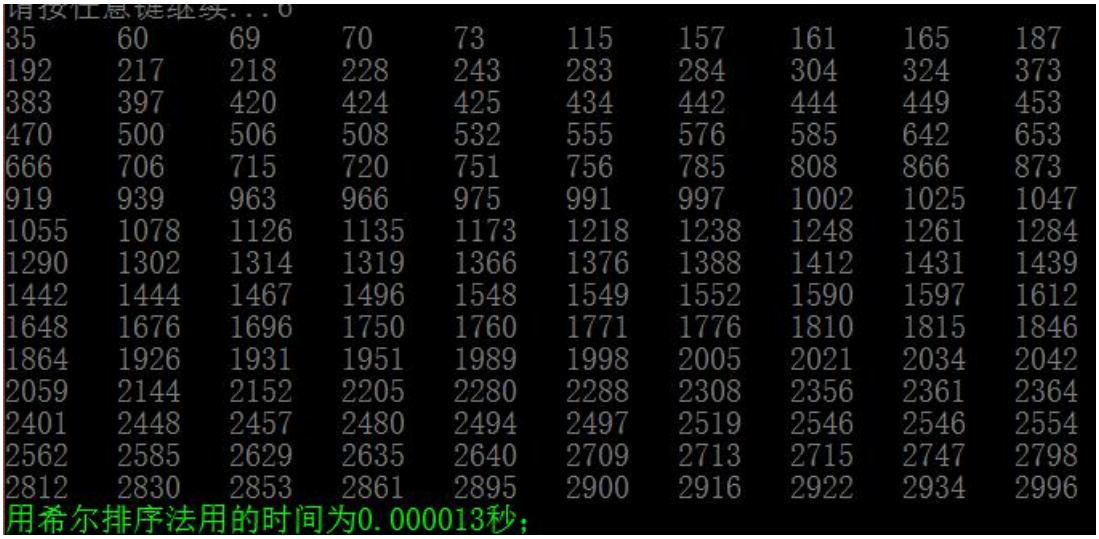

图 5.7 希尔排序结果及时间显示

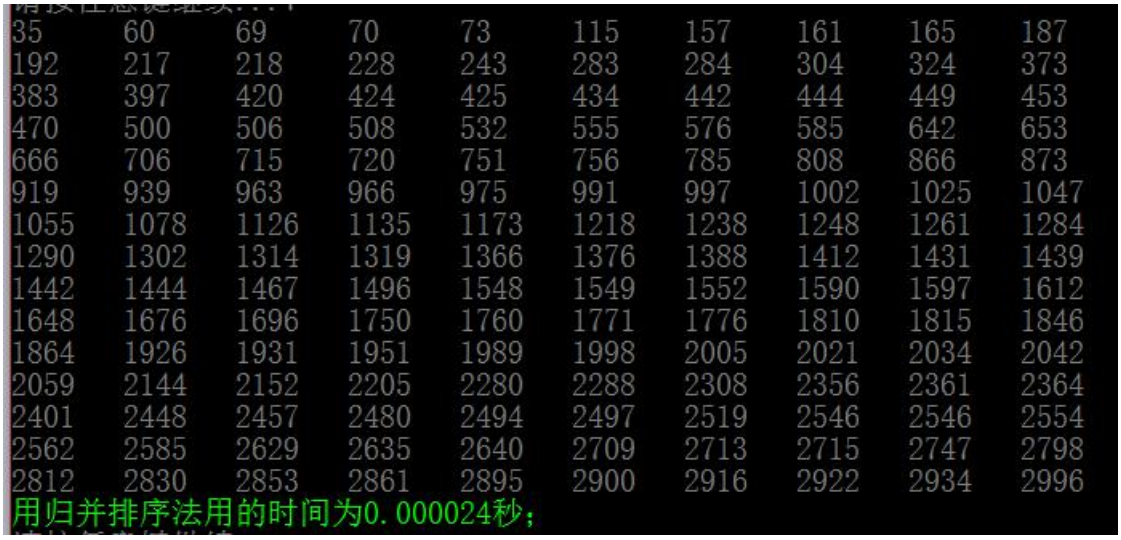

图 5.8 归并排序结果及时间显示

图 5.9 对于 150 个（可任意设定数量）随机数两种最快的排序方法

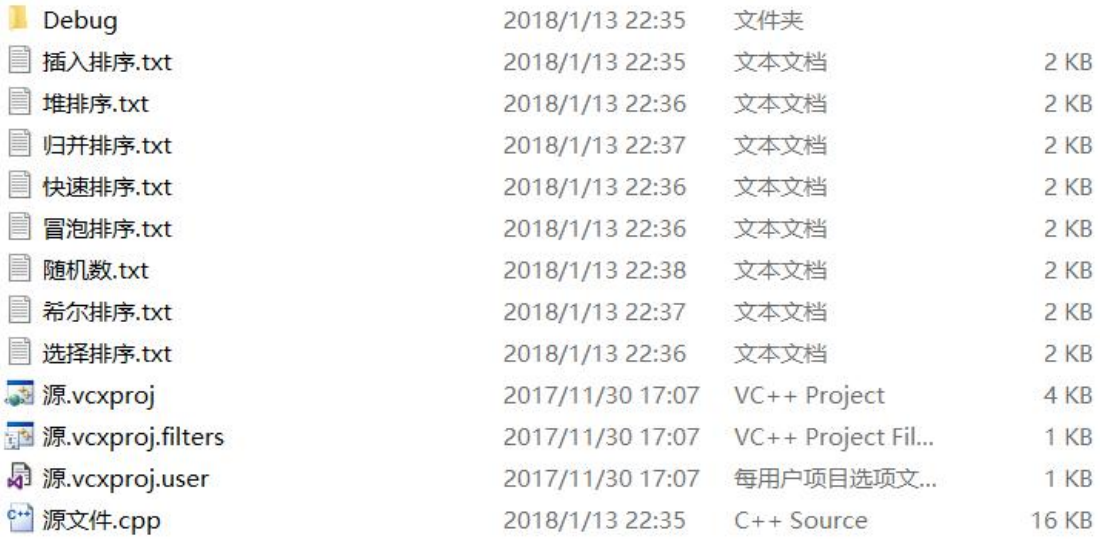

图 5.10 文本文件

5.2MFC 图形界面展示

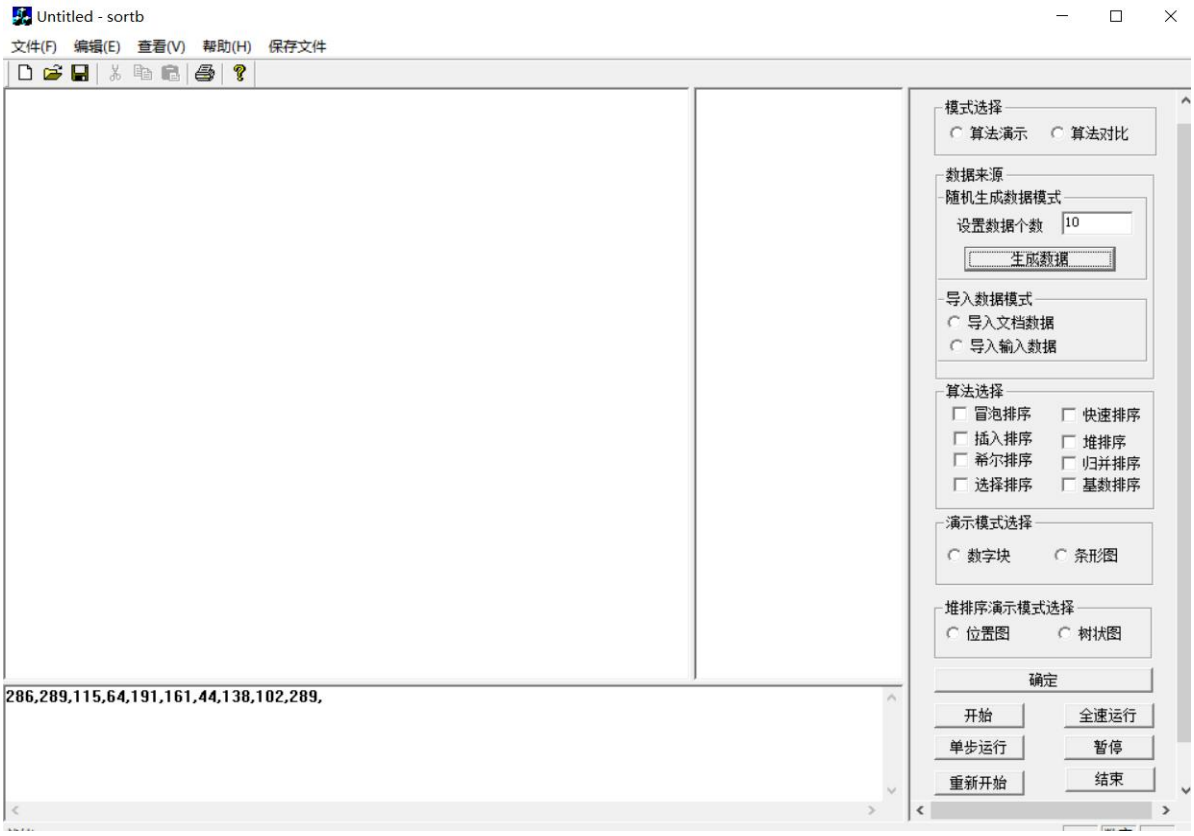

图 5.11 图形界面

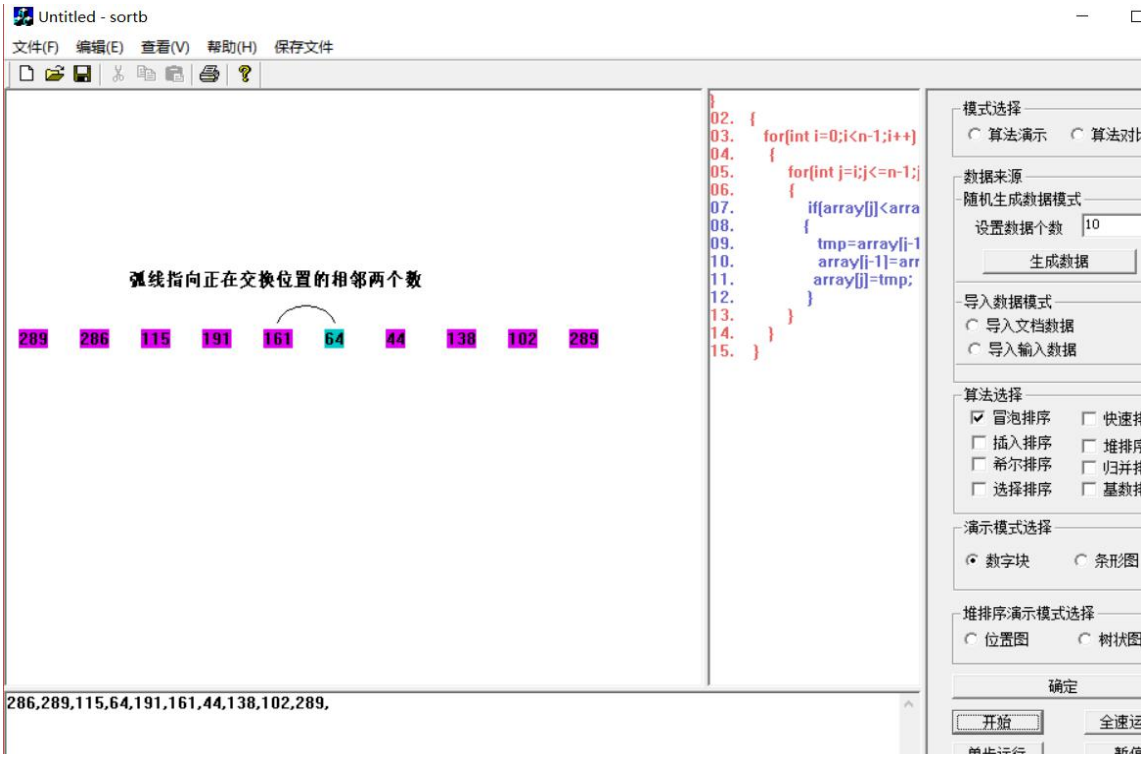

图 5.12 图形界面

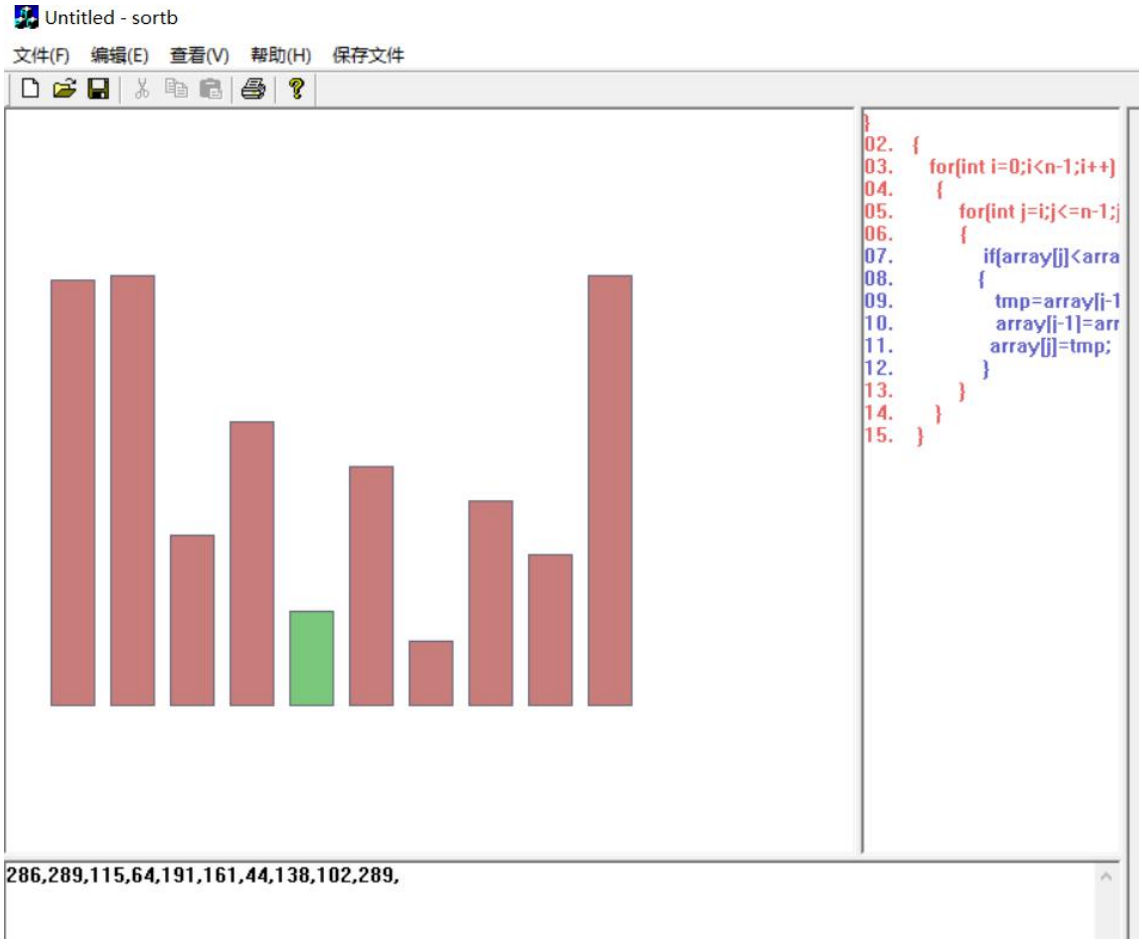

图 5.13 图形界面

图 5.14 图形界面


模式选择
算法演示
算法对比
数据来源
随机生成数据模式
设置数据个数
10
生成数据
导入数据模式
导入文档数据
导入输入数据
算法选择
冒泡排序
快速排序
插入排序
堆排序
希尔排序
归并排序
选择排序
基数排序
演示模式选择
数字块
条形图
堆排序演示模式选择
位置图
树状图
确定
开始
全速运行
单步运行
暂停
重新开始
结束

图 5.15 图形界面

6 总结

经过这次课程设计的锻炼，使我更加理解数据结构这门课程。在设计综合排序算法时，我不仅懂得了许多算法设计思想，而且更加了解了循环的使用。在课程设计中也遇到不少困难，如怎么编写堆排序算法，如何设计一个高精度时间算法函数。但是通过查阅资料，询问同学，在应用课本的前提下，拓展课外知识，完成了此次课程设计，虽然还有许多不足之处，有的算法有点复杂，不够简洁，但是从中我学到了许多东西。另外通过查找各种资料，学会了拿 MFC 展示图形界面，我们不能拘泥于所学知识，应对其灵活运用。如并不是把书上的所有的排序算法照搬到课设中就可以了，还应自行定义数据类型和函数，不断的改进算法，达到正确性、可读性、健壮性的目的。

在本次课设中还要感谢高利军老师细心指导和严格要求!

参考文献

[1]严蔚敏，吴伟民编著，数据结构 C 语言版[M]，北京：清华大学出版社，1997

[2]谭浩强编著，C 程序设计[M],北京：清华大学出版社，2010

[3]徐金梧，杨德斌，徐科编著，Turbo C 实用大全[M]，北京：机械工业出版社

附　录

```
#include<stdio.h>
#include<stdlib.h>
#include<windows.h>   //windows 系统的高精度定时器
#include<time.h>
HANDLE handle1;
#define N 150
void Wrong()
{SetConsoleTextAttribute(handle1,FOREGROUND_INTENSITY| FOREGROUND_RED);
printf("\n---------->按键错误！\n");
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY); }
void show(int a[])
{ int i;
for(i=0;i<N;i++)
{
if((i-1)%10==9)        //每行 10 个输出，方便浏览
printf("\n");
printf("%-7d",a[i]);
}}
void Insert_sort(int a[])        //1.插入排序(如同插纸牌)
{
int i,j,temp;
for(i=1;i<N;i++)     //从第一张牌开始循环到最后一张,第零张牌已经在手里了
{
temp=a[i];       //摸下一张牌
for(j=i;j>0&&a[j-1]>temp;j--)
a[j]=a[j-1];       //移出空位,向后错一位
a[j]=temp;                    //新牌落位
}
}
void Select_sort(int a[])    //2.选择排序(找到最小元素的位置)
{
int i,j,k;
for(i=0;i<N;i++)
{
k=i;
for(j=i+1;j<N;j++)
if(a[j]<a[k])
k=j;
if(k!=i)
```

```
{
int temp;
temp=a[k];
a[k]=a[i];
a[i]=temp;
}
}
}
void Bubble_sort(int a[]) //3.冒泡排序
{
int i,j,temp;
for (i=0;i<N-1;i++)
{
for(j=N-1;j>i;j--)       //比较,找出本趟最小关键字的记录,从最后一个向前找
if(a[j]<a[j-1])          //进行交换,将最小关键字记录前移
{
temp=a[j];
a[j]=a[j-1];
a[j-1]=temp;
}
}
}
void Quick_sort(int a[],int n)      //4.快速排序
{
int i,j,low,high,temp,top=-1;
struct node
{
int low,high;
}st[N];
top++;
st[top].low=0;st[top].high=n-1;
while(top>-1)
{           low=st[top].low;high=st[top].high;
top--;
i=low;j=high;
if(low<high)
{           temp=a[low];
while(i!=j)
{     while(i<j&&a[j]>temp)j--;
if(i<j){a[i]=a[j];i++;}
while(i<j&&a[i]<temp)i++;
if(i<j){a[j]=a[i];j--;}
}
```

```
a[i]=temp;
top++;st[top].low=low;st[top].high=i-1;
top++;st[top].low=i+1;st[top].high=high;
}
}
}
void HeapAdjust(int a[], int i, int size)       //调整堆
{
int Lchild = 2*i+1;
int Rchild = 2*i+2;
int max = i;
if(i<=size/2)
{
if(Lchild<=size && a[Lchild]>a[max])
max=Lchild;
if(Rchild<=size && a[Rchild]>a[max])
max=Rchild;
if(max!=i)
{
// swap(a[i],a[max]);     //  作用是交换
int t;
t=a[i];
a[i]=a[max];
a[max]=t;
HeapAdjust(a,max,size);}}}
void BuildHeap(int a[], int size)//  构建堆
{    for(int i=size/2;i>=0;i--)
       HeapAdjust(a,i,size); //i 是根节点所在的位置，size 是堆里一共有多少元素
}
void Heap_sort(int a[], int size)         //    5.堆排序(对选择的改进)
{
BuildHeap(a,size);                  //  建立最大堆
for(int i=size-1;i>0;i--)
{//swap(a[0],a[i]);   //a[0]根节点存的是最大元素，a[i]是当前最后一个元素的下标
int t;
t=a[0];
a[0]=a[i];
a[i]=t;
HeapAdjust(a,0,i-1);//把剩下的元素再调整成最大堆
}
}
void Shell_sort(int a[], int n)         //6.希尔排序(对插入的改进),利用插入排序的简单，并克服插入排序每次只交换相邻元素的缺点
```

```
{
int i,j,gap;
for(gap=N/2;gap>0;gap/=2)//希尔增量序列
for(i=0;i<gap;i++)
for(j=i+gap;j<n;j+=gap)
{
if(a[j]<a[j-gap])
{
int tmp=a[j];
int k=j-gap;
while(k>=0 && a[k]>tmp)
{
a[k+gap]=a[k];
k-=gap;
}
a[k+gap]=tmp;
}}}
void MergeArray(int a[], int first, int mid, int last, int temp[])
{int i=first, j=mid+1, m=mid, n=last, k=0;
while(i<=m && j<=n)    // m,n 分别是左右终点
{
if(a[i]<=a[j])    // i 是左边，j 是右边
temp[k++]=a[i++];
else
temp[k++]=a[j++];
}
while(i<=m)       //直接复制左边剩下的
temp[k++]=a[i++];
while(j<=n)     //直接复制右边剩下的
temp[k++]=a[j++];
for(i=0;i<k;i++)
a[first+i]=temp[i];    //导回
}
void mergeSort(int a[],int first, int last, int temp[])//递归，劈开
{
if(first<last)
{
int mid = (first+last)/2;
mergeSort(a,first,mid,temp);
mergeSort(a,mid+1,last,temp);
MergeArray(a,first,mid,last,temp);
}
}
```

```
void Merge_sort(int a[], int len)     // 7.归并排序
{
int *p = new int[len];
mergeSort(a,0,len-1,p); //0，len-1 分别为最左和最右
}
double T_time(int c[],int p) //时间计算函数
{   LARGE_INTEGER x={0};          QueryPerformanceFrequency(&x);       //表示一个
64 位有符号的整数值   // 检索性能计数器的频率
LARGE_INTEGER start={0};   QueryPerformanceCounter(& start);//检索性能计数器
的当前值，该值是高分辨率时间戳，可用于时间间隔测量
switch(p)
{
case 1:Insert_sort(c);break;
case 2:Select_sort(c);break;
case 3:Bubble_sort(c);break;
case 4:Quick_sort(c,N);break;
case 5:Heap_sort(c,N);break;
case 6:Shell_sort(c,N);;break;
case 7:Merge_sort(c,N);break;
}
LARGE_INTEGER now={0};       QueryPerformanceCounter(&now);
double times=now.QuadPart - start.QuadPart;
times/=x.QuadPart;
return times;
}
double T_insert_sort(int a[],int p)     //1.插入排序---时间计算
{
int i, b[N],c[N];
for(i=0;i<N;i++)     b[i]=a[i];
for(i=0;i<N;i++)     c[i]=a[i];
Insert_sort(b);     if(p!=8)     show(b);
double time=T_time(c,p);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY |
FOREGROUND_GREEN);
printf("\n 用插入排序法用的时间为%f 秒；",time);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
FILE *fp;   fp=fopen("插入排序.txt","w");
for(i=0;i<N;i++)     {   if((i-1)%10==9)   fprintf(fp,"\n"); fprintf(fp,"%-7d ",b[i]);}
fclose(fp);
return   time;
}
double T_select_sort(int a[],int p)     //2.选择排序---时间计算
{
```

```
int i, b[N],c[N];
for(i=0;i<N;i++)      b[i]=a[i];
for(i=0;i<N;i++)      c[i]=a[i];
Select_sort(b);      if(p!=8)      show(b);
double time=T_time(c,p);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY |
FOREGROUND_GREEN);
printf("\n 用选择排序法用的时间为%f 秒；",time);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
FILE *fp;    fp=fopen("选择排序.txt","w");
for(i=0;i<N;i++)    {    if((i-1)%10==9)    fprintf(fp,"\n"); fprintf(fp,"%-7d ",b[i]);}
fclose(fp);
return    time;
}
double T_bubble_sort(int a[],int p)    //3.冒泡排序---时间计算
{
int i, b[N],c[N];
for(i=0;i<N;i++)      b[i]=a[i];
for(i=0;i<N;i++)      c[i]=a[i];
Bubble_sort(b);      if(p!=8)      show(b);

double time=T_time(c,p);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY |
FOREGROUND_GREEN);
printf("\n 用冒泡排序法用的时间为%f 秒；",time);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
FILE *fp;    fp=fopen("冒泡排序.txt","w");
for(i=0;i<N;i++)      {    if((i-1)%10==9)    fprintf(fp,"\n"); fprintf(fp,"%-7d ",b[i]);}
fclose(fp);
return    time;
}
double T_quick_sort(int a[],int p)    //4.快速排序---时间计算
{
int i, b[N],c[N];
for(i=0;i<N;i++)      b[i]=a[i];
for(i=0;i<N;i++)      c[i]=a[i];
Quick_sort(b,N);      if(p!=8)      show(b);

double time=T_time(c,p);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY |
FOREGROUND_GREEN);
printf("\n 用快速排序法用的时间为%f 秒；",time);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
```

```
FILE *fp;    fp=fopen("快速排序.txt","w");
for(i=0;i<N;i++)       {    if((i-1)%10==9)    fprintf(fp,"\n"); fprintf(fp,"%-7d ",b[i]);}
fclose(fp);
return    time;
}
double T_heap_sort(int a[],int p) //5.堆排序---时间计算
{
int i, b[N],c[N];
for(i=0;i<N;i++)      b[i]=a[i];
for(i=0;i<N;i++)      c[i]=a[i];
Heap_sort(b,N);      if(p!=8)      show(b);

double time=T_time(c,p);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY |
FOREGROUND_GREEN);
printf("\n 用堆排序法用的时间为%f 秒；",time);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
FILE *fp;    fp=fopen("堆排序.txt","w");
for(i=0;i<N;i++)      {    if((i-1)%10==9)    fprintf(fp,"\n"); fprintf(fp,"%-7d ",b[i]);}
fclose(fp);
return    time;
}
double T_shell_sort(int a[],int p)      //6.希尔排序---时间计算
{
int i, b[N],c[N];
for(i=0;i<N;i++)      b[i]=a[i];
for(i=0;i<N;i++)      c[i]=a[i];
Shell_sort(b,N);      if(p!=8)      show(b);

double time=T_time(c,p);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY |
FOREGROUND_GREEN);
printf("\n 用希尔排序法用的时间为%f 秒；",time);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
FILE *fp;    fp=fopen("希尔排序.txt","w");
for(i=0;i<N;i++)      {    if((i-1)%10==9)    fprintf(fp,"\n"); fprintf(fp,"%-7d ",b[i]);}
fclose(fp);
return    time;
}
double T_merge_sort(int a[],int p)    //7.归并排序---时间计算
{
int i, b[N],c[N];
for(i=0;i<N;i++)      b[i]=a[i];
```

```
for(i=0;i<N;i++)     c[i]=a[i];
Merge_sort(b,N);     if(p!=8)     show(b);

double time=T_time(c,p);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY |
FOREGROUND_GREEN);
printf("\n 用归并排序法用的时间为%f 秒；",time);
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
FILE *fp;    fp=fopen("归并排序.txt","w");
for(i=0;i<N;i++)     {    if((i-1)%10==9)    fprintf(fp,"\n"); fprintf(fp,"%-7d ",b[i]);}
fclose(fp);
return    time;
}
/*****************************************************************
************************************************************/

void TSort(double a[])     //时间数据的选择排序
{
int i,j;
double temp;
for(i=0;i<7;i++)
for(j=i+1;j<7;j++)
if(a[i]>a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
void Create_random_num(int a[])
{srand((int)time(NULL));
for(int i=0;i<N;i++)    a[i]=rand()%3000+1;     //随机种子:1---3000
}
void Funtion()
{
int a[N],num;
Create_random_num(a);
while(1)
{double t[7],tt[7];//时间数组
scanf("%d",&num);
switch(num)
{ case 0:printf("=====>谢谢使用!\n");break;
case 1:tt[0]=t[0]=T_insert_sort(a,num);printf("\n 请按任意键继
```

```
续...");getchar();break;
case 2:tt[1]=t[1]=T_select_sort(a,num);printf("\n 请按任意键继
续...");getchar();break;
case 3:tt[2]=t[2]=T_bubble_sort(a,num);printf("\n 请按任意键继
续...");getchar();break;
case 4:tt[3]=t[3]=T_quick_sort(a,num);printf("\n 请按任意键继续...");getchar();break;
case 5:tt[4]=t[4]=T_heap_sort(a,num);printf("\n 请按任意键继续...");getchar();break;
case 6:tt[5]=t[5]=T_shell_sort(a,num);printf("\n 请按任意键继续...");getchar();break;
case 7:tt[6]=t[6]=T_merge_sort(a,num);printf("\n 请按任意键继
续...");getchar();break;
case 8:TSort(t);
printf("\n\n");
{
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY | FOREGROUND_RED);
printf("排序这组数据两种较快的排序法分别是：\n");
while(1)
{
if(t[0]==tt[0]){ printf("插入排序:%f 秒!\n",t[0]); break; }
if(t[0]==tt[1]){ printf("选择排序:%f 秒!\n",t[0]); break; }
if(t[0]==tt[2]){ printf("冒泡排序:%f 秒!\n",t[0]); break; }
if(t[0]==tt[3]){ printf("快速排序:%f 秒!\n",t[0]); break; }
if(t[0]==tt[4]){ printf("堆排序:%f 秒!\n",t[0]);     break; }
if(t[0]==tt[5]){ printf("希尔排序:%f 秒!\n",t[0]);break;   }
if(t[0]==tt[6]){ printf("归并排序:%f 秒!\n",t[0]);break;   }
}
while(1)
{
if(t[1]==tt[0]) {printf("插入排序%f 秒!\n",t[1]); break;}
if(t[1]==tt[1]) {printf("选择排序%f 秒!\n",t[1]); break;}
if(t[1]==tt[2]) {printf("冒泡排序%f 秒!\n",t[1]); break;}
if(t[1]==tt[3]) {printf("快速排序%f 秒!\n",t[1]); break;}
if(t[1]==tt[4]) {printf("堆排序%f 秒!\n",t[1]);     break;}
if(t[1]==tt[5]) {printf("希尔排序%f 秒!\n",t[1]); break;}
if(t[1]==tt[6]) {printf("归并排序%f 秒!\n",t[1]); break;}
}
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
}
printf("\n 请按任意键继续...");
srand((int)time(NULL)); for(int i=0;i<N;i++)    a[i]=rand()%5000+1;     break; // 重新又
定义了一组随机值
case 9:show(a);    FILE *fp;fp=fopen("随机数.txt","w");
for(int i=0;i<N;i++)    {if((i-1)%10==9)    fprintf(fp,"\n"); fprintf(fp,"%-7d ",a[i]);}
fclose(fp);
```

```
getchar();printf("\n 请按任意键继续...");getchar();break;
default:Wrong();printf("\n 请按任意键继续...");getchar();break;
}
}
}
void menu()
{
printf("              ********** 欢迎来到综合排序系统!
**********\n");
printf("                                    ※※※※※※※
\n");
printf("                                    ※  菜  单  ※
\n");
printf("                                    ※※※※※※※
\n");
printf("
\n");
printf("          ********** (1)---插入排序          **********\n");
printf("          ********** (2)---选择排序          **********\n");
printf("          ********** (3)---冒泡排序          **********\n");
printf("          ********** (4)---快速排序          **********\n");
printf("          ********** (5)---堆排序            **********\n");
printf("          ********** (6)---希尔排序            **********\n");
printf("          ********** (7)---归并排序            **********\n");
printf("          ********** (8)---时间效率比较      **********\n");
printf("          ********** (9)---显示随机数        **********\n");
printf("          ********** (0)---退出              **********\n");
printf("
\n");
printf("************************************************************\n
");
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY |
FOREGROUND_GREEN);
printf("\n====>请在上述序号中选择一个并输入:");
SetConsoleTextAttribute(handle1, FOREGROUND_INTENSITY);
}
void main()
{    handle1 = GetStdHandle(STD_OUTPUT_HANDLE);
       menu();
   Funtion();
}
```