**《算法与数据结构》**

**实 验 指 导 手 册**

# 北京邮电大学软件学院

# 2020-2021学年第1学期实验报告

**课程名称： 算法与数据结构**

**实验名称： 实验七 排序**

**实验完成人：**

**姓名：**\_\_\_王宇涵\_\_\_\_\_**学号：**\_\_\_\_2020211730\_\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_**\_\_贾红娓** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2022 年 1 月 1 日**

1. **实验目的**

本次实验旨在集中对几个专门的问题作较为深入的探讨和理解，不强调对某些特定的编程技术的训练。

1. **实验内容**

**必做内容**

1. **内部排序算法比较**

**[问题描述]**

　　各种内部排序算法的时间复杂度分析结果只给出了算法执行时间的阶，或大概执行时间。试通过随机的数据比较各算法的关键字比较次数和关键字移动次数，以取得直观感受。

**[基本要求]**

　　（1） 对以下9种常用的内部排序算法进行比较：直接插入排序；折半折入排序；二路插入排序；希尔排序；起泡排序；快速排序；简单选择排序；堆排序；归并排序。

　　（2） 待排序表的表长不少于100；其中的数据要用伪随机数产生程序产生；至少要用5组不同的输入数据作比较；比较的指标为有关键字参加的比较次数和关键字移动次数（关键字交换计为3次移动）。

**[测试数据]**

　　由随机产生器决定。

**[实现提示]**

　　主要工作是设法在程序中适当的地方插入计数操作。程序还可以包括计算几组数据得出结果波动大小的解释。注意分块调试的方法。

1. **统计成绩**

**[问题描述]**

　　给出n个学生的m门考试的成绩表，每个学生的信息由学号、姓名以及各科成绩组成。对学生的考试成绩进行有关统计，并打印统计表。

**[基本要求]**

　　（1） 按总数高低次序，打印出名次表，分数相同的为同一名次；

　　（2） 按名次打印出每个学生的学号、姓名、总分以及各科成绩。

**[测试数据]**

　　由学生依据软件工程的测试技术自己确定。注意测试边界数据。

**选做内容**

　　针对必做实验1），对不同的输入表长做试验，观察检查两个指标相关于表长的变化关系。还可以对稳定性做验证。

1. **实验环境**

VC6.0

1. **实验过程和实验结果**

**1.内部排序算法比较**

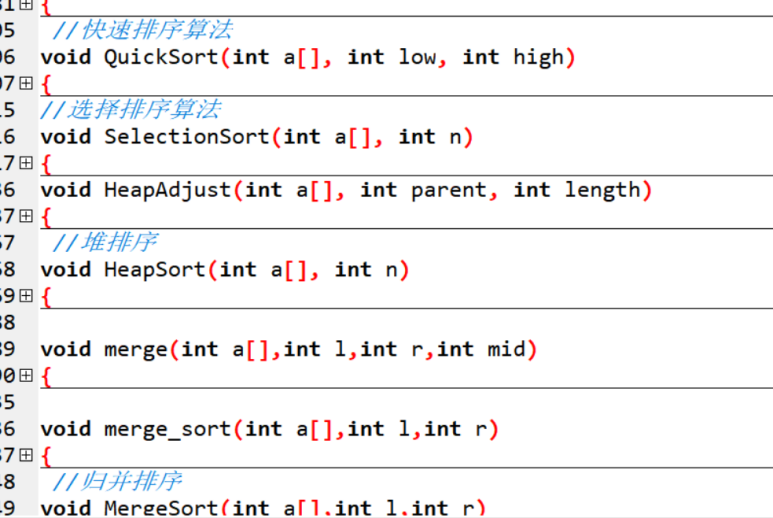
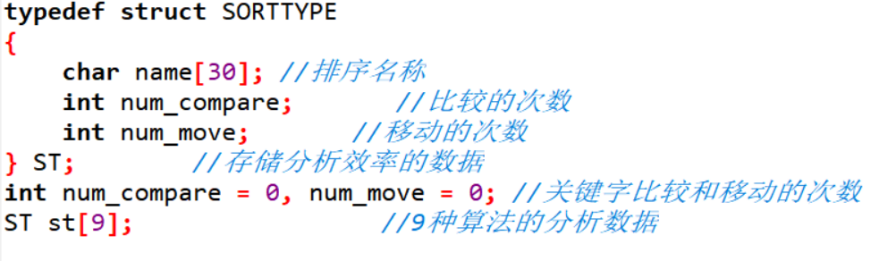
**1.1问题分析**

排序是计算机内经常进行的一种操作，其目的是将一组“无序”的记录序列调整为“有序”的记录序列。分内部排序和外部排序。若整个排序过程不需要访问外存便能完成，则称此类排序问题为内部排序。反之，若参加排序的记录数量很大，整个序列的排序过程不可能在内存中完成，则称此类排序问题为外部排序。

内部排序的过程是一个逐步扩大记录的有序序列长度的过程。主要的排序包括：直接插入排序；折半折入排序；二路插入排序；希尔排序；起泡排序；快速排序；简单选择排序；堆排序；归并排序。计算并比较这些算法的时间复杂度。

**1.2设计方案**

1. 直接插入排序
2. 希尔排序
3. 折半插入排序
4. 2路插入排序
5. 冒泡排序
6. 快速排序
7. 选择排序
8. 堆排序
9. 归并排序
10. 输出数组
11. 获取随机数函数
12. 调用各种排序算法的选项函数
13. 一个结构体，用于存储比较和排序次数

****

**1.3算法**

1. **插入排序**

插入排序算法是所有排序方法中最简单的一种算法，其主要的实现思想是将数据按照一定的顺序一个一个的插入到有序的表中，最终得到的序列就是已经排序好的数据。  
 直接插入排序是插入排序算法中的一种，采用的方法是：在添加新的记录时，使用顺序查找的方式找到其要插入的位置，然后将新记录插入。

在查找插入位置时，采用的是顺序查找的方式，而在[查找表](http://c.biancheng.net/view/3426.html" \t "http://c.biancheng.net/view/_blank)中数据本身有序的前提下，可以使用折半查找来代替顺序查找，这种排序的算法就是折半插入排序算法。

2路插入排序算法是在折半插入排序的基础上对其进行改进，减少其在排序过程中移动记录的次数从而提高效率。另外设置一个同存储记录的数组大小相同的数组 d，将无序表中第一个记录添加进 d[0] 的位置上，然后从无序表中第二个记录开始，同 d[0] 作比较：如果该值比 d[0] 大，则添加到其右侧；反之添加到其左侧。（循环数组）

//直接插入排序算法

void InsertSort(int a[], int n)

{

int i, j;

int temp;

for (i = 1; i < n; i++) //for循环内一定比较了n-1次，if判断语句

{

if (a[i] < a[i - 1]) //一旦出现了逆序的关键字，就进行插入

{

temp = a[i];

j = i - 1;

num\_compare++;

do //往后移动一个位置，腾空间给temp；

{

a[j + 1] = a[j];

num\_move++; //移动加一

j--;

num\_compare++; //比较次数加一

}

while (j >= 0 && a[j] > temp);

a[j + 1] = temp; //最后把temp放在对应的位置

num\_move += 2; //移动的temp

}

}

}

//折半插入排序法

//有序区利用折半查找来寻找的改进算法

void BinInsertSort(int a[], int n)

{

int i, j, low, high, mid;

int temp;

for (i = 1; i < n; i++) //已经比较了n-1次

{

if (a[i] < a[i - 1])

{

temp = a[i];

low = 0;

high = i - 1;

num\_compare++;

while (low <= high)

{

num\_compare++; //while进入比较

mid = (low + high) / 2;

if (temp < a[mid])

high = mid - 1;

else

low = mid + 1;

}

for (j = i - 1; j >= high + 1; j--)

{

a[j + 1] = a[j];

num\_move++; //移动次数加一

}

a[high + 1] = temp;

num\_move += 2; //temp交换

}

}

}

//二路插入

//类似于循环队列

void InsertSort2(int a[], int n)

{

int first=0, last=0;

int \*b = new int[n];//b循环数组

b[0] = a[0];//初始化

for (int i = 1; i < n; i++)

{

if (a[i] < b[first])

{

num\_compare++;

first = (first - 1 + n) % n;

b[first] = a[i];

num\_move++;

}

else if (a[i] >= b[last])

{

num\_compare+=2; //和b[last]及b[first]比较

last=(last + 1) % n;

b[last] = a[i];

num\_move++;

}

else

{

num\_compare+=2;

int low, high, mid, d;

low = first, high = last;

while (low != high) //折半查找

{

num\_compare++;

d = (high-low+n) % n; //元素个数

mid = (low + d / 2) % n; //中间位置

if (a[i] < b[mid])

{

high = mid;

num\_compare++;

}

else

{

low = (mid + 1) % n;

num\_move++;

}

}

for (int k = last + 1; k != low; k = (k - 1 + n) % n) //移动元素

{

b[k] = b[(k - 1 + n) % n];

num\_move++;

}

b[low] = a[i];

num\_move++;

last++;

}

}

for (int i = 0; i < n; i++)

a[i] = b[(i + first) % n];

delete[]b;

}

1. **希尔排序**

希尔排序是把记录按下标的一定增量分组，对每组使用直接插入排序算法排序；随着增量逐渐减少，每组包含的[关键词](https://baike.baidu.com/item/%E5%85%B3%E9%94%AE%E8%AF%8D/254446" \t "https://baike.baidu.com/item/%E5%B8%8C%E5%B0%94%E6%8E%92%E5%BA%8F/_blank)越来越多，当增量减至 1 时，整个文件恰被分成一组，[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/209025" \t "https://baike.baidu.com/item/%E5%B8%8C%E5%B0%94%E6%8E%92%E5%BA%8F/_blank)便终止。

//希尔排序算法

void ShellSort(int a[], int n)

{

int i, j, d;

int temp;

d = n / 2;

while (d > 0)

{

for (i = d; i < n; i++)

{

temp = a[i];

j = i - d;

while (j >= 0 && temp < a[j])

{

num\_compare++;

num\_move++;

a[j + d] = a[j];

j = j - d;

}

a[j + d] = temp;

num\_move += 2; //temp进行两次操作

}

d = d / 2;

}

}

1. **冒泡排序**

起泡排序，别名“冒泡排序”，该算法的核心思想是将无序表中的所有记录，通过两两比较关键字，得出升序序列或者降序序列。

//冒泡排序算法

void BubbleSort(int a[], int n)

{

for (int i = 0; i < n - 1; i++)

{

for (int j = n - 1; j > i; j--)

if (a[j] < a[j - 1])

{

num\_compare++;

num\_move += 3;

swap(a[j], a[j - 1]);

}

}

}

1. **快速排序**

通过一趟排序算法把所需要排序的序列的元素分割成两大块，其中，一部分的元素都要小于或等于另外一部分的序列元素，然后仍根据该种方法对划分后的这两块序列的元素分别再次实行[快速排序](https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95/369842" \t "https://baike.baidu.com/item/%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95/_blank)，排序实现的整个过程可以是递归的来进行调用，最终能够实现将所需排序的无序序列元素变为一个有序的序列。

这里使用两个函数，一个用来划分，返回pivot位置。一个调用该划分函数并递归。

//用pivot讲序列化为两个部分

int Partition(int a[], int low, int high)

{

int pivot=a[low];//将第一个元素作为pivot

while(low < high)

{

while(low < high && a[high] >= pivot)

{

high --;//找到比pivot 小的元素

num\_compare++; //进行比较

}

a[low]=a[high];//移到左端

num\_move++; //移动++

while(low < high && a[low] <= pivot)

{

low ++;//找到比pivot大的元素

num\_compare++; //进行比较

}

a[high] = a[low];//移到右

num\_move++; //移动++

}

//跳出循环表示找到pivot位置

a[low]=pivot;

num\_move +=2; //移动+2

return low;

}

//快速排序算法

void QuickSort(int a[], int low, int high)

{

if(low <high)//递归跳出条件

{

int pos=Partition(a, low, high);

QuickSort(a, low, pos-1);//划分左子表

QuickSort(a, pos+1, high);//划分右子表

}

}

1. **选择排序**

该算法的实现思想为：对于具有 n 个记录的无序表遍历 n-1 次，第 i 次从无序表中第 i 个记录开始，找出后序关键字中最小的记录，然后放置在第 i 的位置上。

//选择排序算法

void SelectionSort(int a[], int n)

{

int mid;

for (int i = 0; i < n - 1; ++i)

{

mid = i;

for (int j = i + 1; j < n; ++j)

{

if (a[j] < a[mid])

{

mid = j;

}

num\_compare++;

}

swap(a[mid], a[i]);

//cout<<num\_move<<" ";

num\_move+=3;

}

}

1. **堆排序**

通过将无序表转化为堆，可以直接找到表中最大值或者最小值，然后将其提取出来，令剩余的记录再重建一个堆，取出次大值或者次小值，如此反复执行就可以得到一个有序序列，此过程为堆排序。

void HeapAdjust(int a[], int parent, int length)

{

int temp = a[parent]; // temp保存当前父节点

int child = 2 \* parent + 1; // 先获得左孩子

while (child < length)

{

// 如果有右孩子结点，并且右孩子结点的值大于左孩子结点，则选取右孩子结点

if (child + 1 < length && a[child] < a[child + 1])

{

child++;

}

// 如果父结点的值已经大于孩子结点的值，则直接结束

if (temp >= a[child])

{

num\_compare++;

break;

}

// 把孩子结点的值赋给父结点

a[parent] = a[child];

num\_move++;

// 选取孩子结点的左孩子结点,继续向下筛选

parent = child;

child = 2 \* child + 1;

}

a[parent] = temp;

num\_move+=2;

}

//堆排序

void HeapSort(int a[], int n)

{

// 循环建立初始堆

for (int i = n / 2; i >= 0; i--)

{

HeapAdjust(a, i, n);

}

// 进行n-1次循环，完成排序

for (int i = n - 1; i > 0; i--)

{

// 最后一个元素和第一元素进行交换

swap(a[i], a[0]);

num\_move+=3;

// 筛选 R[0] 结点，得到i-1个结点的堆

HeapAdjust(a, 0, i);

}

}

1. **归并排序**

归并排序是利用归并的思想实现的排序方法，该算法采用经典的分治策略（分治法将问题分成一些小的问题然后递归求解，而治的阶段则将分的阶段得到的各答案"修补"在一起，即分而治之)。

void merge(int a[],int l,int r,int mid)

{

int aux[r-l+1],i,j,k;//aux辅助数组

for(k=l;k<=r;k++)

{

aux[k-l]=a[k];

num\_move++;

}

i=l;

j=mid+1;

for(k=l;k<=r;k++)

{

if(i>mid)

{

a[k]=aux[j-l];

num\_move++;

j++;

}

else if(j>r)

{

a[k]=aux[i-l];

num\_move++;

i++;

}

else if(aux[i-l]>aux[j-l])

{

a[k]=aux[j-l];

num\_move++;

num\_compare++;

j++;

}

else

{

a[k]=aux[i-l];

num\_move++;

num\_compare++;

i++;

}

}

}

void merge\_sort(int a[],int l,int r)

{

if(l>=r)

return;

int mid=(l+r)/2;//找中间

merge\_sort(a,l,mid);//归并左边

merge\_sort(a,mid+1,r);//归并右边

merge(a,l,r,mid);//合并

}

//归并排序

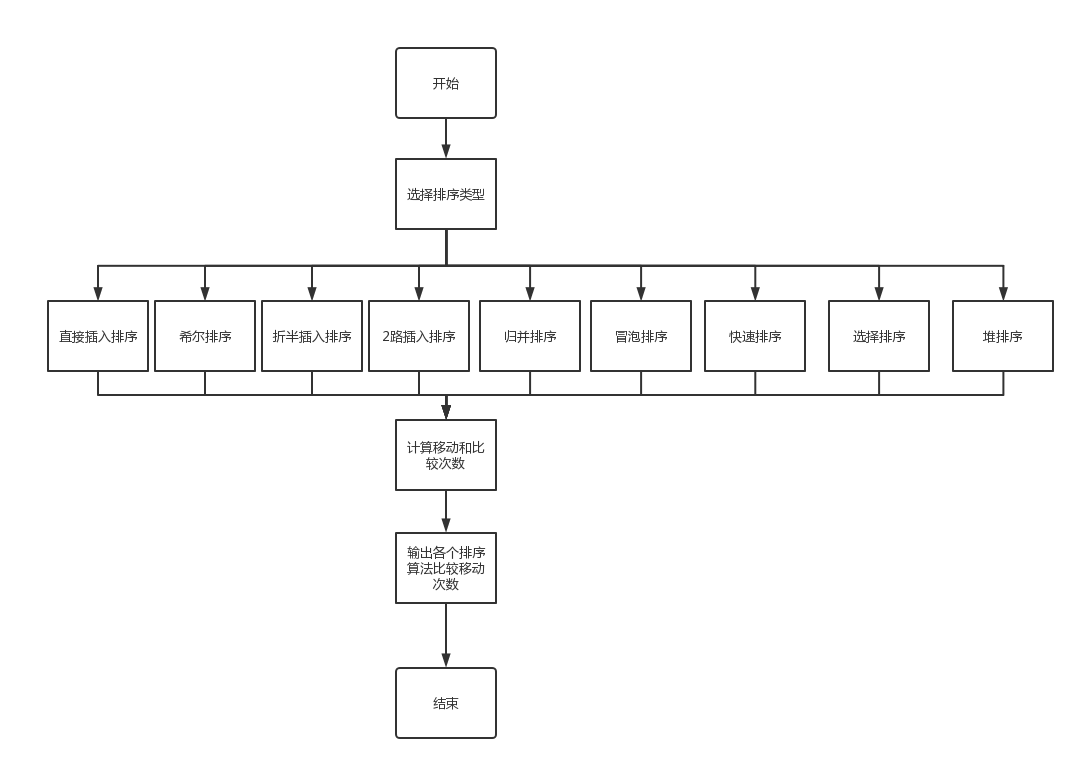
void MergeSort(int a[],int l,int r)

{

merge\_sort(a,l,r-1);

}

**1.4设计图**



**1.5程序**

#include <cstdio>

#include <cstdlib>

#include <string.h>

#include <ctime>

#include <iostream>

#define MAX 150

using namespace std;

typedef struct SORTTYPE

{

char name[30]; //排序名称

int num\_compare; //比较的次数

int num\_move; //移动的次数

} ST; //存储分析效率的数据

int num\_compare = 0, num\_move = 0; //关键字比较和移动的次数

ST st[9]; //9种算法的分析数据

//直接插入排序算法

void InsertSort(int a[], int n)

{

int i, j;

int temp;

for (i = 1; i < n; i++) //for循环内一定比较了n-1次，if判断语句

{

if (a[i] < a[i - 1]) //一旦出现了逆序的关键字，就进行插入

{

temp = a[i];

j = i - 1;

num\_compare++;

do //往后移动一个位置，腾空间给temp；

{

a[j + 1] = a[j];

num\_move++; //移动加一

j--;

num\_compare++; //比较次数加一

}

while (j >= 0 && a[j] > temp);

a[j + 1] = temp; //最后把temp放在对应的位置

num\_move += 2; //移动的temp

}

}

}

//折半插入排序法

//有序区利用折半查找来寻找的改进算法

void BinInsertSort(int a[], int n)

{

int i, j, low, high, mid;

int temp;

for (i = 1; i < n; i++) //已经比较了n-1次

{

if (a[i] < a[i - 1])

{

temp = a[i];

low = 0;

high = i - 1;

num\_compare++;

while (low <= high)

{

num\_compare++; //while进入比较

mid = (low + high) / 2;

if (temp < a[mid])

high = mid - 1;

else

low = mid + 1;

}

for (j = i - 1; j >= high + 1; j--)

{

a[j + 1] = a[j];

num\_move++; //移动次数加一

}

a[high + 1] = temp;

num\_move += 2; //temp交换

}

}

}

//二路插入

//类似于循环队列

void InsertSort2(int a[], int n)

{

int first=0, last=0;

int \*b = new int[n];//b循环数组

b[0] = a[0];//初始化

for (int i = 1; i < n; i++)

{

if (a[i] < b[first])

{

num\_compare++;

first = (first - 1 + n) % n;

b[first] = a[i];

num\_move++;

}

else if (a[i] >= b[last])

{

num\_compare+=2; //和b[last]及b[first]比较

last=(last + 1) % n;

b[last] = a[i];

num\_move++;

}

else

{

num\_compare+=2;

int low, high, mid, d;

low = first, high = last;

while (low != high) //折半查找

{

num\_compare++;

d = (high-low+n) % n; //元素个数

mid = (low + d / 2) % n; //中间位置

if (a[i] < b[mid])

{

high = mid;

num\_compare++;

}

else

{

low = (mid + 1) % n;

num\_move++;

}

}

for (int k = last + 1; k != low; k = (k - 1 + n) % n) //移动元素

{

b[k] = b[(k - 1 + n) % n];

num\_move++;

}

b[low] = a[i];

num\_move++;

last++;

}

}

for (int i = 0; i < n; i++)

a[i] = b[(i + first) % n];

delete[]b;

}

//希尔排序算法

void ShellSort(int a[], int n)

{

int i, j, d;

int temp;

d = n / 2;

while (d > 0)

{

for (i = d; i < n; i++)

{

temp = a[i];

j = i - d;

while (j >= 0 && temp < a[j])

{

num\_compare++;

num\_move++;

a[j + d] = a[j];

j = j - d;

}

a[j + d] = temp;

num\_move += 2; //temp进行两次操作

}

d = d / 2;

}

}

//冒泡排序算法

void BubbleSort(int a[], int n)

{

for (int i = 0; i < n - 1; i++)

{

for (int j = n - 1; j > i; j--)

if (a[j] < a[j - 1])

{

num\_compare++;

num\_move += 3;

swap(a[j], a[j - 1]);

}

}

}

//用pivot讲序列化为两个部分

int Partition(int a[], int low, int high)

{

int pivot=a[low];//将第一个元素作为pivot

while(low < high)

{

while(low < high && a[high] >= pivot)

{

high --;//找到比pivot 小的元素

num\_compare++; //进行比较

}

a[low]=a[high];//移到左端

num\_move++; //移动++

while(low < high && a[low] <= pivot)

{

low ++;//找到比pivot大的元素

num\_compare++; //进行比较

}

a[high] = a[low];//移到右

num\_move++; //移动++

}

//跳出循环表示找到pivot位置

a[low]=pivot;

num\_move +=2; //移动+2

return low;

}

//快速排序算法

void QuickSort(int a[], int low, int high)

{

if(low <high)//递归跳出条件

{

int pos=Partition(a, low, high);

QuickSort(a, low, pos-1);//划分左子表

QuickSort(a, pos+1, high);//划分右子表

}

}

//选择排序算法

void SelectionSort(int a[], int n)

{

int mid;

for (int i = 0; i < n - 1; ++i)

{

mid = i;

for (int j = i + 1; j < n; ++j)

{

if (a[j] < a[mid])

{

mid = j;

}

num\_compare++;

}

swap(a[mid], a[i]);

//cout<<num\_move<<" ";

num\_move+=3;

}

}

void HeapAdjust(int a[], int parent, int length)

{

int temp = a[parent]; // temp保存当前父节点

int child = 2 \* parent + 1; // 先获得左孩子

while (child < length)

{

// 如果有右孩子结点，并且右孩子结点的值大于左孩子结点，则选取右孩子结点

if (child + 1 < length && a[child] < a[child + 1])

{

child++;

}

// 如果父结点的值已经大于孩子结点的值，则直接结束

if (temp >= a[child])

{

num\_compare++;

break;

}

// 把孩子结点的值赋给父结点

a[parent] = a[child];

num\_move++;

// 选取孩子结点的左孩子结点,继续向下筛选

parent = child;

child = 2 \* child + 1;

}

a[parent] = temp;

num\_move+=2;

}

//堆排序

void HeapSort(int a[], int n)

{

// 循环建立初始堆

for (int i = n / 2; i >= 0; i--)

{

HeapAdjust(a, i, n);

}

// 进行n-1次循环，完成排序

for (int i = n - 1; i > 0; i--)

{

// 最后一个元素和第一元素进行交换

swap(a[i], a[0]);

num\_move+=3;

// 筛选 R[0] 结点，得到i-1个结点的堆

HeapAdjust(a, 0, i);

//System.out.format("第 %d 趟: \t", n - i);

//printPart(a, 0, a.length - 1);

}

}

void merge(int a[],int l,int r,int mid)

{

int aux[r-l+1],i,j,k;//aux辅助数组

for(k=l;k<=r;k++)

{

aux[k-l]=a[k];

num\_move++;

}

i=l;

j=mid+1;

for(k=l;k<=r;k++)

{

if(i>mid)

{

a[k]=aux[j-l];

num\_move++;

j++;

}

else if(j>r)

{

a[k]=aux[i-l];

num\_move++;

i++;

}

else if(aux[i-l]>aux[j-l])

{

a[k]=aux[j-l];

num\_move++;

num\_compare++;

j++;

}

else

{

a[k]=aux[i-l];

num\_move++;

num\_compare++;

i++;

}

}

}

void merge\_sort(int a[],int l,int r)

{

if(l>=r)

return;

int mid=(l+r)/2;//找中间

merge\_sort(a,l,mid);//归并左边

merge\_sort(a,mid+1,r);//归并右边

merge(a,l,r,mid);//合并

}

//归并排序

void MergeSort(int a[],int l,int r)

{

merge\_sort(a,l,r-1);

}

//主菜单

void menu()

{

printf("\t\t1.直接插入排序法\n");

printf("\t\t2.折半插入排序法\n");

printf("\t\t3.希尔排序法\n");

printf("\t\t4.冒泡排序法\n");

printf("\t\t5.快速排序法\n");

printf("\t\t6.二路插入\n");

printf("\t\t7.简单选择排序\n");

printf("\t\t8.堆排序\n");

printf("\t\t9.归并排序\n");

printf("\t\t10.效率比较\n");

printf("\t\t11.退出\n");

printf("请选择操作：");

}

//打印数组数据

void printArray(int a[])

{

int i;

for (i = 0; i < MAX; i++)

printf("%2d%c", a[i], (i+1)%40 ? ' ' : '\n');

putchar(10);

}

//为数组获得随机数

void GetRandArray(int a[])

{

int i;

for (i = 0; i < MAX; i++)

a[i] = rand() % 100;

}

//调用各个排序的实现函数

void op1(int a[])

{

GetRandArray(a);

printf("伪随机数已经生成的150个新的随机数\n");

//打印排序前的数组

printArray(a);

InsertSort(a, MAX);

cout<<"利用直接插入排序后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n直接插入排序法：\n一共比较%d，移动%d\n", num\_compare, num\_move);

st[0].num\_compare = num\_compare;

st[0].num\_move = num\_move;

strcpy(st[0].name, "直接插入排序");

}

void op2(int a[])

{

GetRandArray(a);

printf("已经生成150个新的随机数\n");

//打印排序前的数组

printArray(a);

num\_compare = 0;

num\_move = 0;

BinInsertSort(a, MAX);

cout<<"利用折半插入排序后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n折半插入排序：\n一共比较了%d次，移动了%d次\n", num\_compare, num\_move);

st[1].num\_compare = num\_compare;

st[1].num\_move = num\_move;

strcpy(st[1].name, "折半插入排序");

}

void op3(int a[])

{

GetRandArray(a);

printf("已经生成150个新的随机数\n");

//打印排序前的数组

printArray(a);

num\_compare = 0;

num\_move = 0;

ShellSort(a, MAX);

cout<<"\n利用希尔排序算法后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n希尔排序算法：\n一共比较了%d次，移动了%d次\n", num\_compare, num\_move);

st[2].num\_compare = num\_compare;

st[2].num\_move = num\_move;

strcpy(st[2].name, "希尔排序算法");

}

void op4(int a[])

{

GetRandArray(a);

printf("已经生成150个新的随机数\n");

//打印排序前的数组

printArray(a);

num\_compare = 0;

num\_move = 0;

BubbleSort(a, MAX);

cout<<"利用冒泡排序法后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n冒泡排序算法：\n一共比较了%d次，移动了%d次\n", num\_compare, num\_move);

st[3].num\_compare = num\_compare;

st[3].num\_move = num\_move;

strcpy(st[3].name, "冒泡排序算法");

}

void op5(int a[])

{

GetRandArray(a);

printf("已经生成150个新的随机数\n");

//打印排序前的数组

printArray(a);

num\_compare = 0;

num\_move = 0;

QuickSort(a, 0, MAX);

cout<<"\n利用快速排序算法后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n快速排序算法：\n一共比较了%d次，移动了%d次\n", num\_compare, num\_move);

st[4].num\_compare = num\_compare;

st[4].num\_move = num\_move;

strcpy(st[4].name, "快速排序算法");

}

void op6(int a[])

{

GetRandArray(a);

printf("已经生成150个新的随机数\n");

//打印排序前的数组

printArray(a);

num\_compare = 0;

num\_move = 0;

InsertSort2(a, MAX);

cout<<"利用二路插入算法后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n二路插入算法：\n一共比较了%d次，移动了%d次\n", num\_compare, num\_move);

st[5].num\_compare = num\_compare;

st[5].num\_move = num\_move;

strcpy(st[5].name, "二路插入算法");

}

void op7(int a[])

{

GetRandArray(a);

printf("已经生成150个新的随机数\n");

//打印排序前的数组

printArray(a);

num\_compare = 0;

num\_move = 0;

SelectionSort(a, MAX);

cout<<"利用选择排序算法后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n选择排序算法：\n一共比较了%d次，移动了%d次\n", num\_compare, num\_move);

st[6].num\_compare = num\_compare;

st[6].num\_move = num\_move;

strcpy(st[6].name, "选择排序算法");

}

void op8(int a[])

{

GetRandArray(a);

printf("已经生成150个新的随机数\n");

//打印排序前的数组

printArray(a);

num\_compare = 0;

num\_move = 0;

HeapSort(a, MAX);

cout<<"利用堆排序算法后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n堆排序算法：\n一共比较了%d次，移动了%d次\n", num\_compare, num\_move);

st[7].num\_compare = num\_compare;

st[7].num\_move = num\_move;

strcpy(st[7].name, "堆排序算法");

}

void op9(int a[])

{

GetRandArray(a);

printf("已经生成150个新的随机数\n");

//打印排序前的数组

printArray(a);

num\_compare = 0;

num\_move = 0;

MergeSort(a, 0, MAX);

cout<<"利用归并排序算法后的数列如下："<<endl;

//打印排序后的数组

printArray(a);

printf("\n\n归并排序算法：\n一共比较了%d次，移动了%d次\n", num\_compare, num\_move);

st[8].num\_compare = num\_compare;

st[8].num\_move = num\_move;

strcpy(st[8].name, "归并排序算法");

}

void op10(int a[])

{

int i;

printf("各种排序算法的比较于移动次数的对比：\n\n");

printf(" 名称 比较次数 移动次数\n");

for (i = 0; i < 9; i++)

{

printf("%-18s%-18d %d\n", st[i].name, st[i].num\_compare, st[i].num\_move);

}

}

int main()

{

int a[MAX]; //列表数组

int op;

srand((unsigned)time(NULL)); //随机种子

do

{

system("cls");

menu();

scanf("%d", &op);

switch (op)

{

case 1:

op1(a);

break;

case 2:

op2(a);

break;

case 3:

op3(a);

break;

case 4:

op4(a);

break;

case 5:

op5(a);

break;

case 6:

op6(a);

break;

case 7:

op7(a);

break;

case 8:

op8(a);

break;

case 9:

op9(a);

break;

case 10:

op10(a);

break;

default:

break;

}

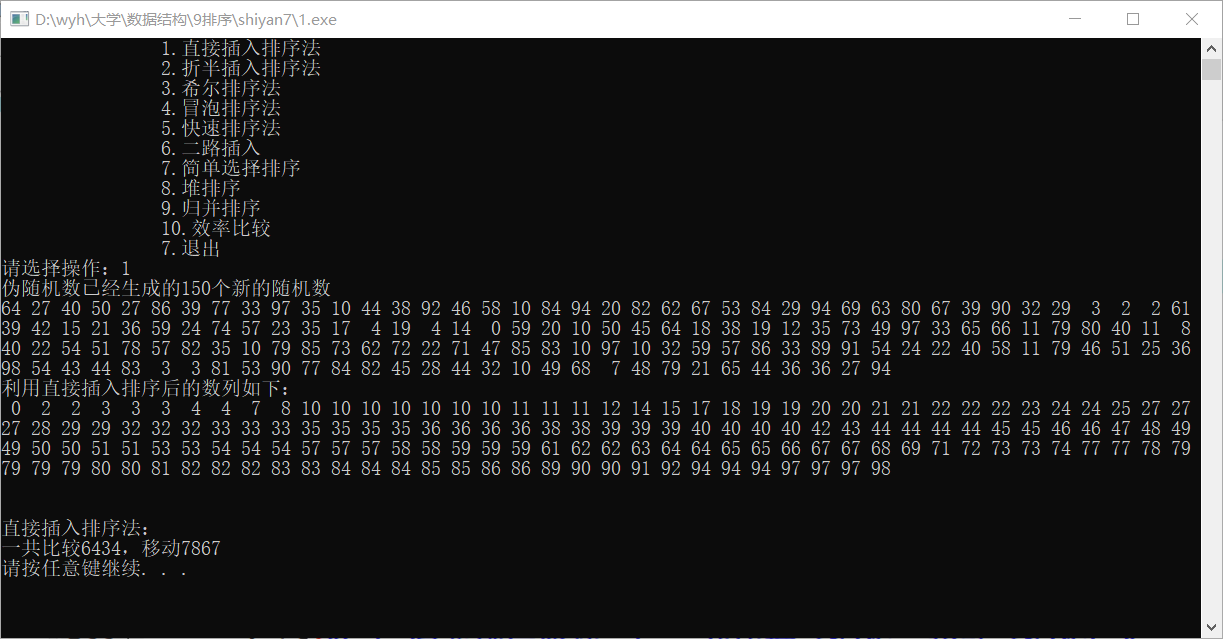
system("pause");

}while(op!=11);

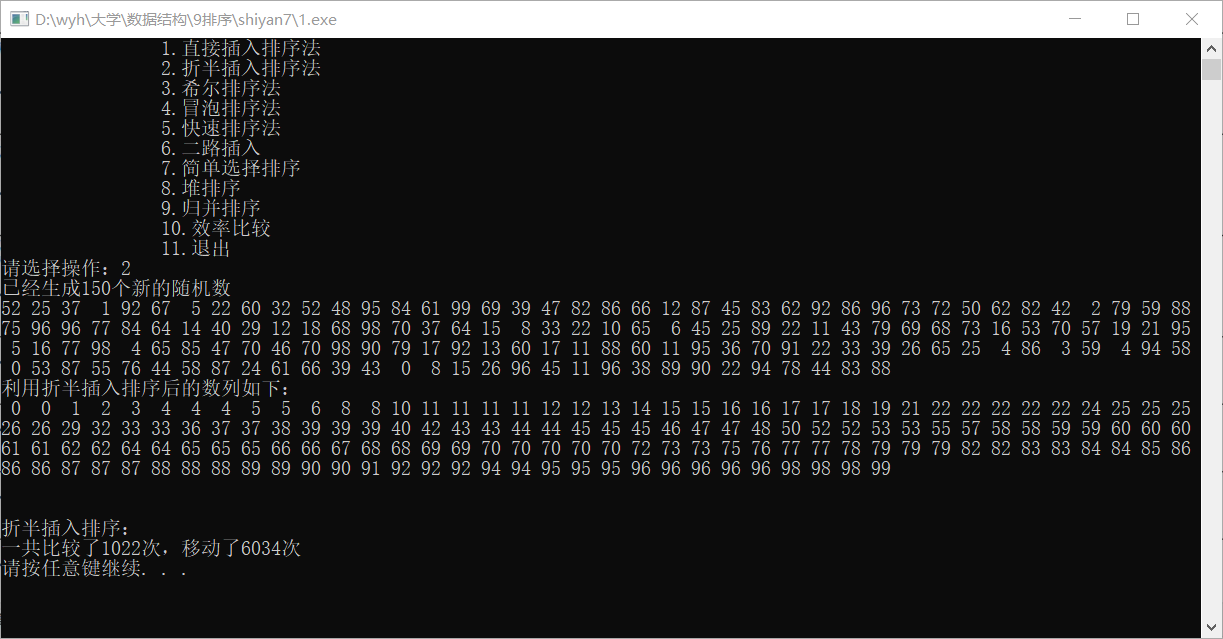
return 0;

}

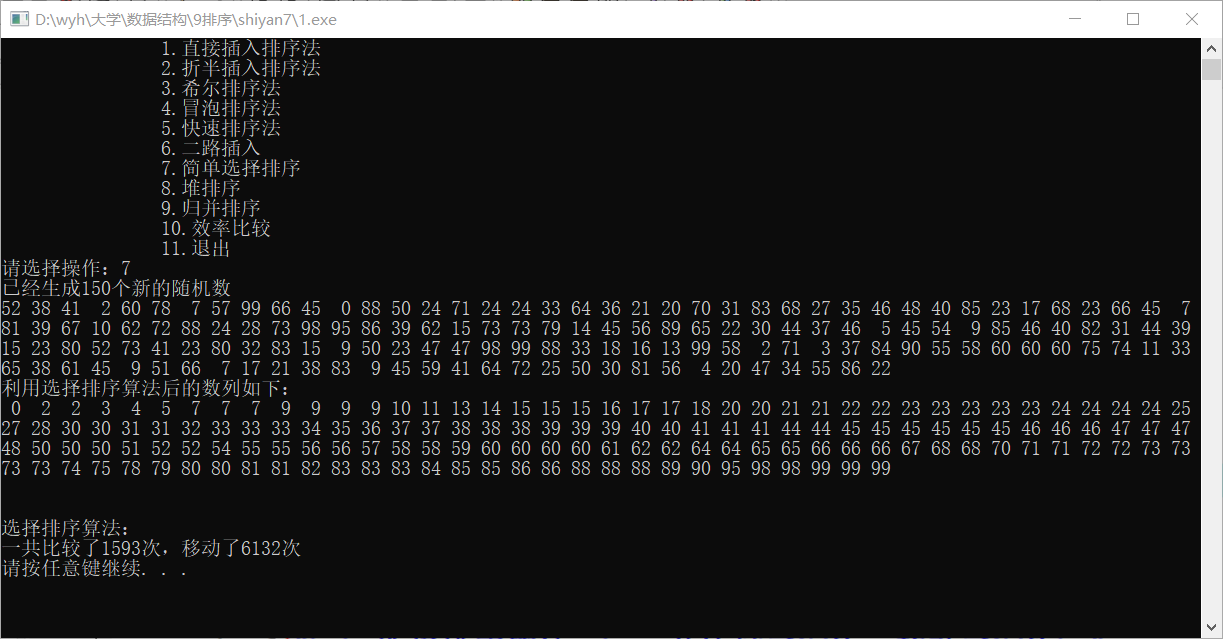
**1.6调试过程截图**

****

**调试第一个界面**

****

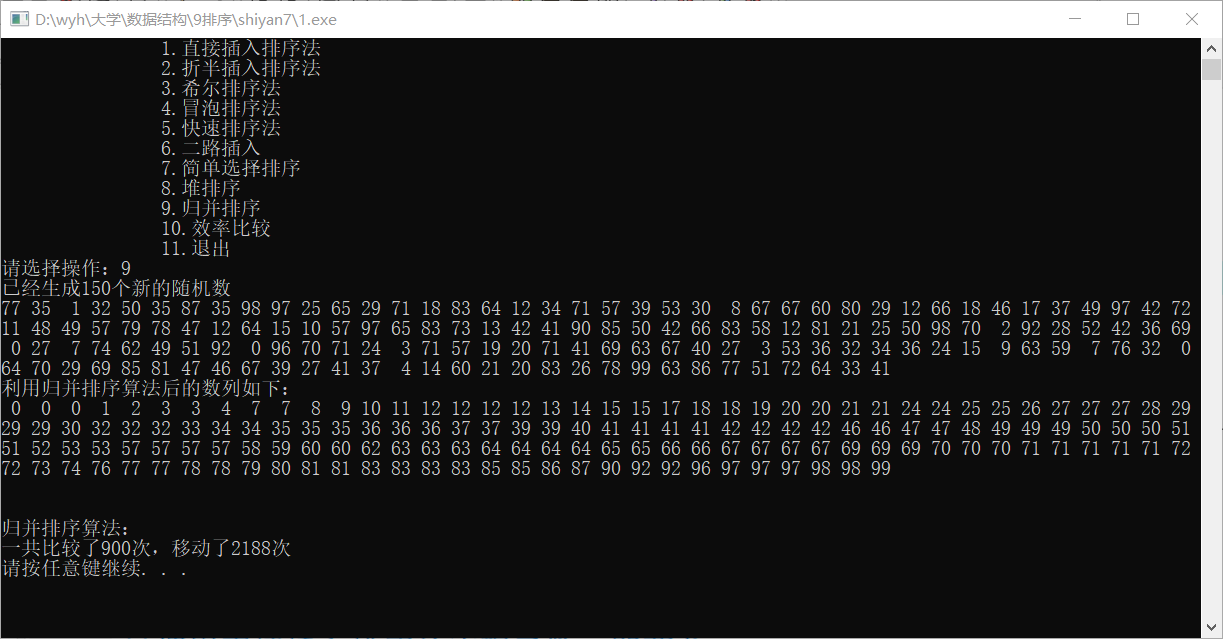
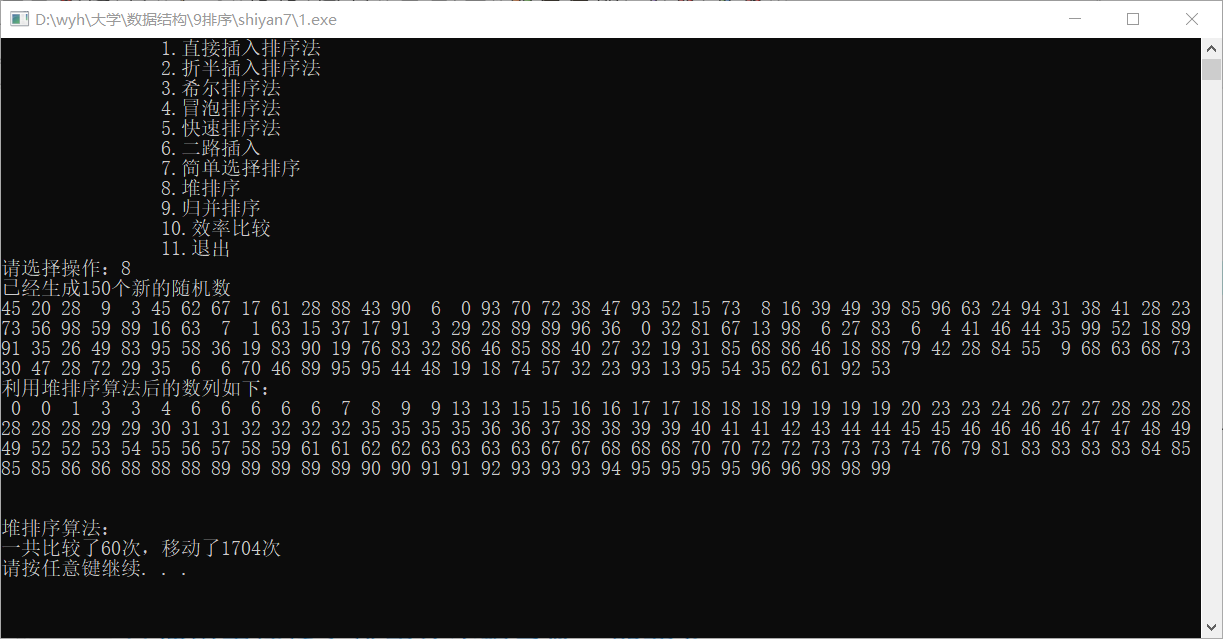
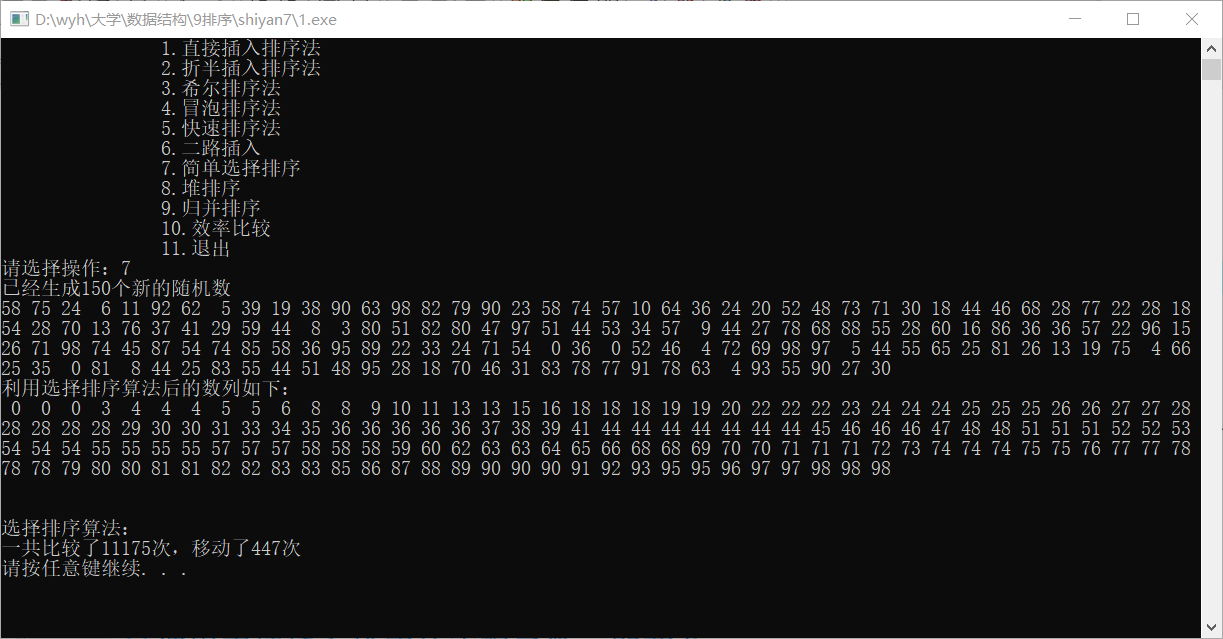
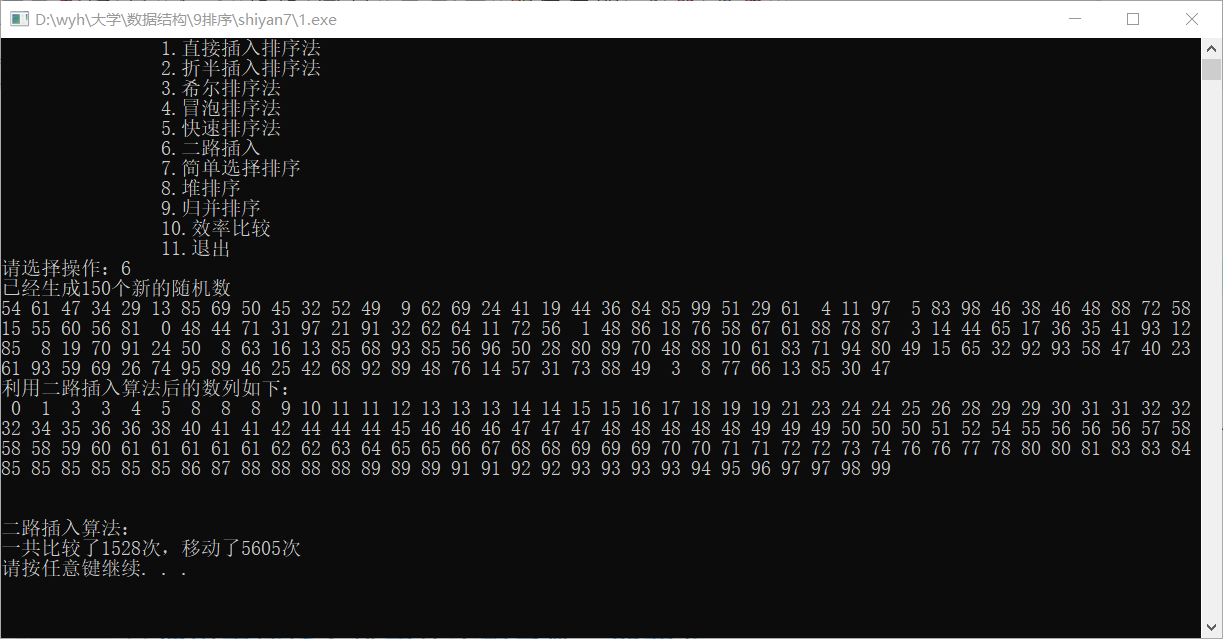
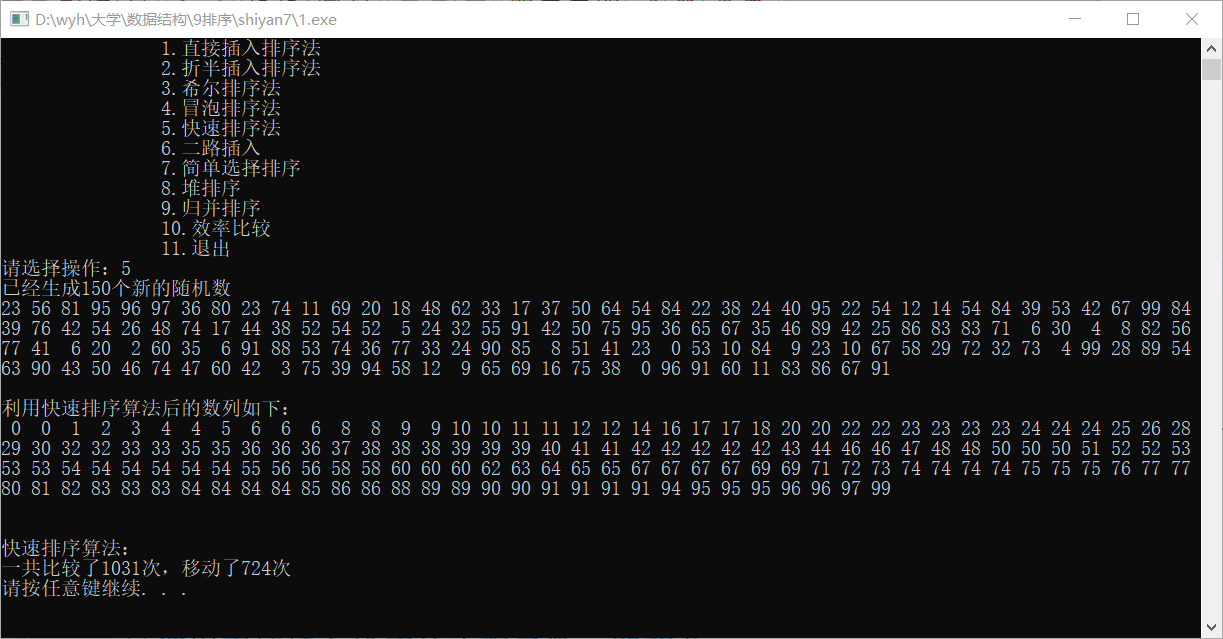
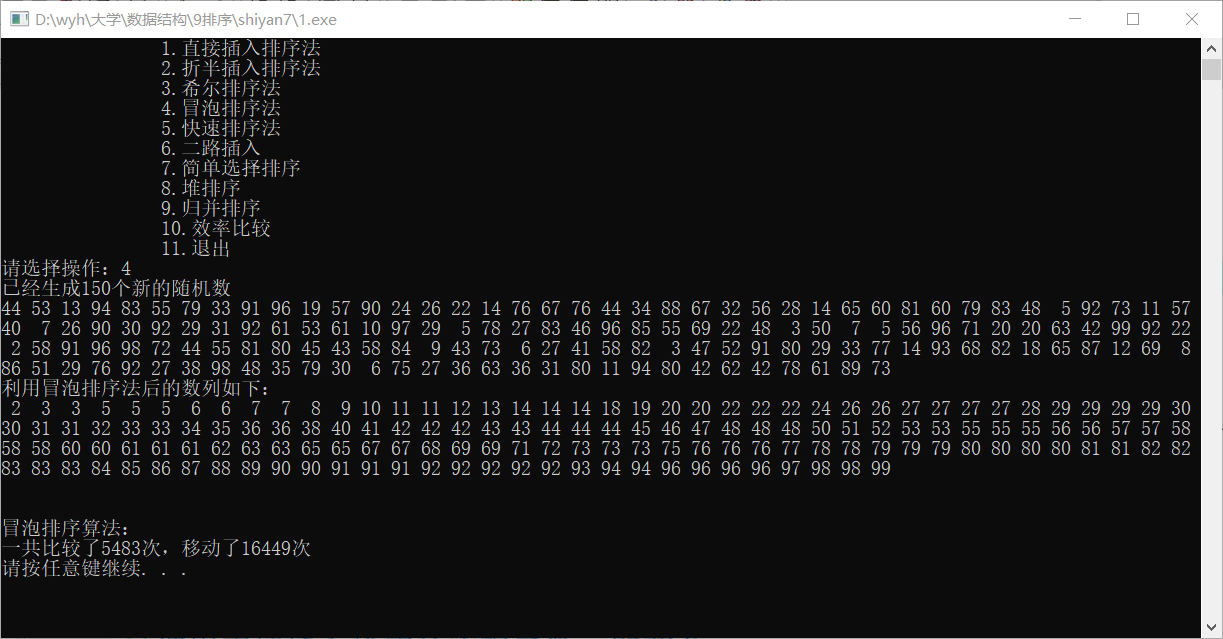
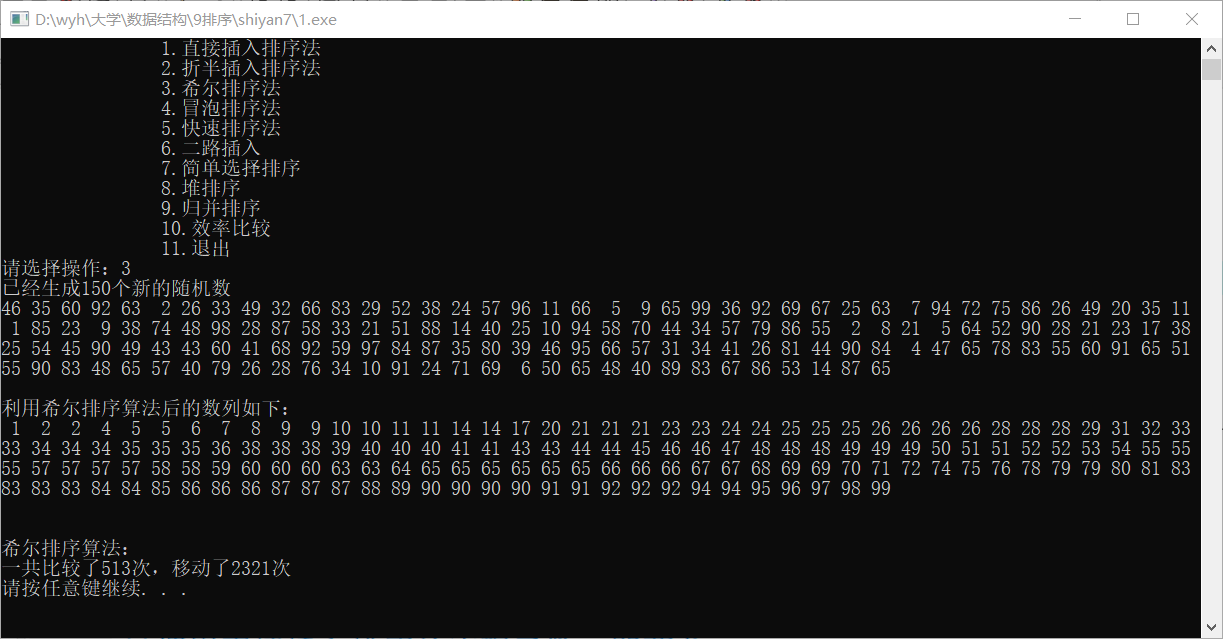
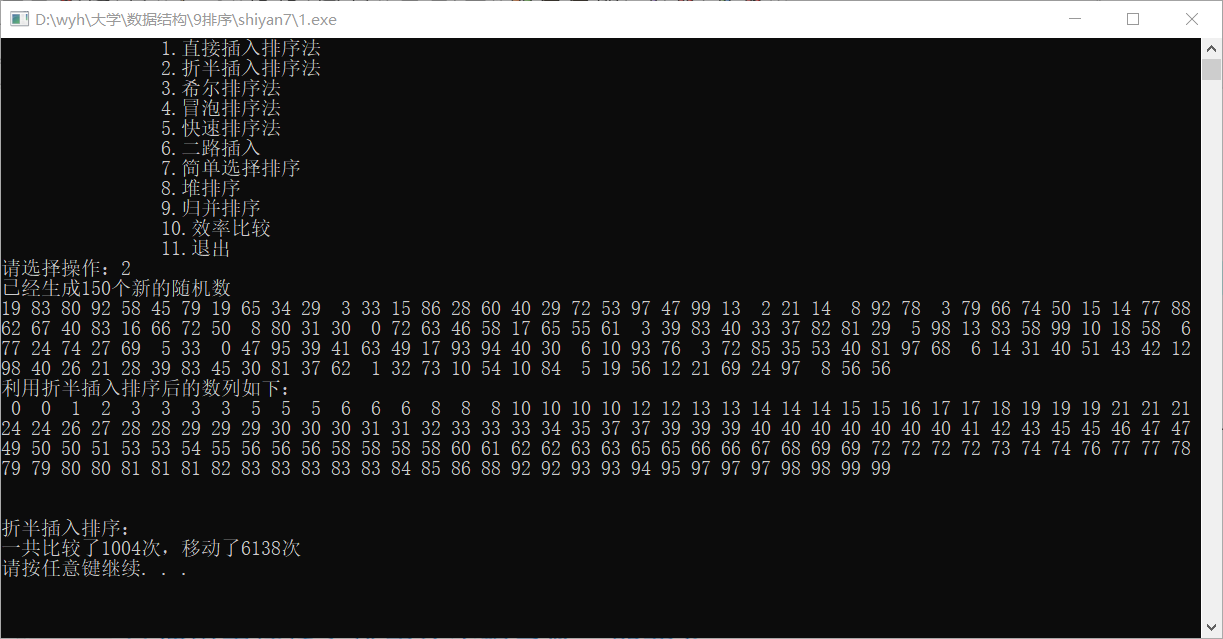
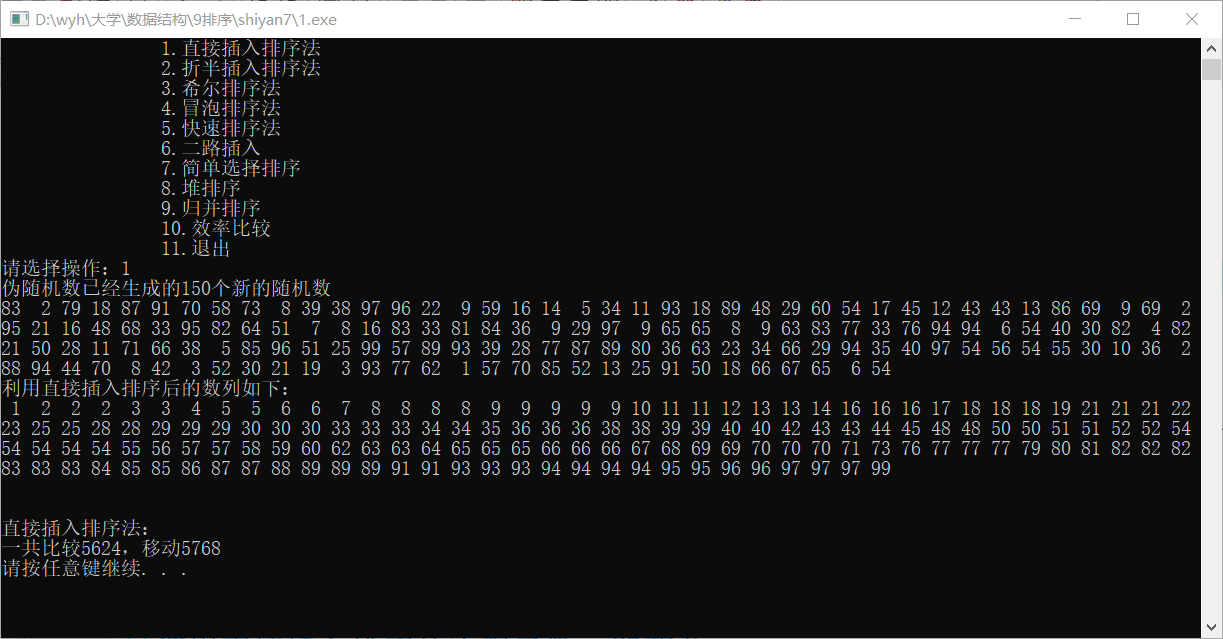
**调试第二个界面**

****

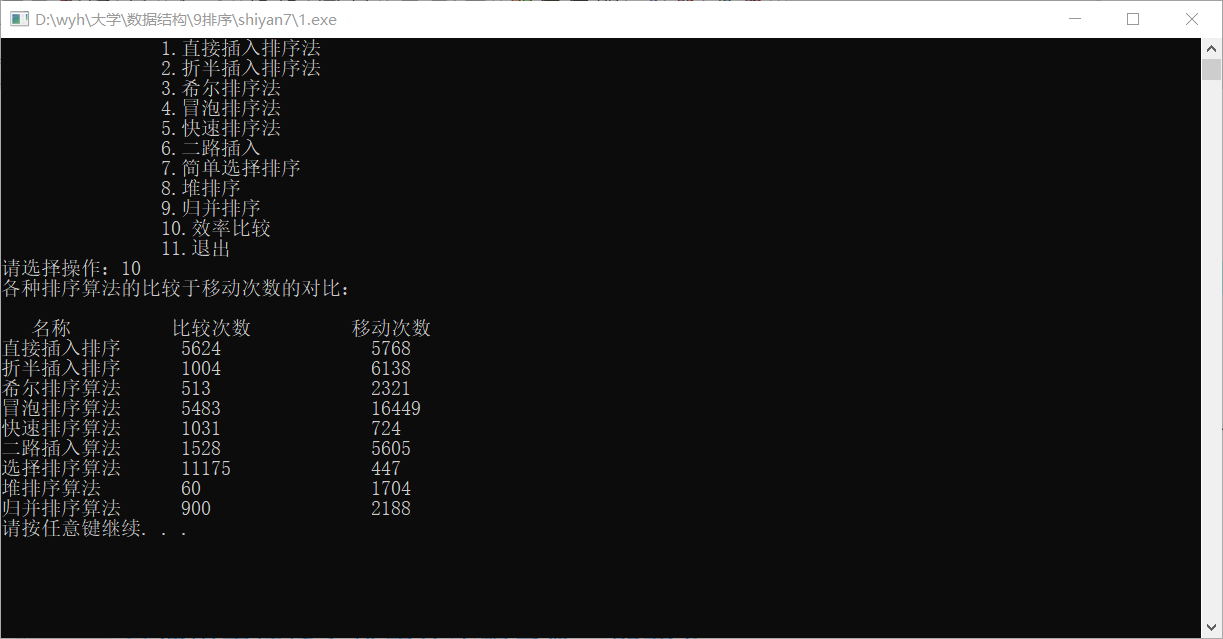
**移动次数有误，调用错函数**

**1.7结果截图**

**9组不同数据得到的排序结果：**

****

**最终结果：（使用9组不同数据）**

**2.统计成绩**

**2.1问题分析**

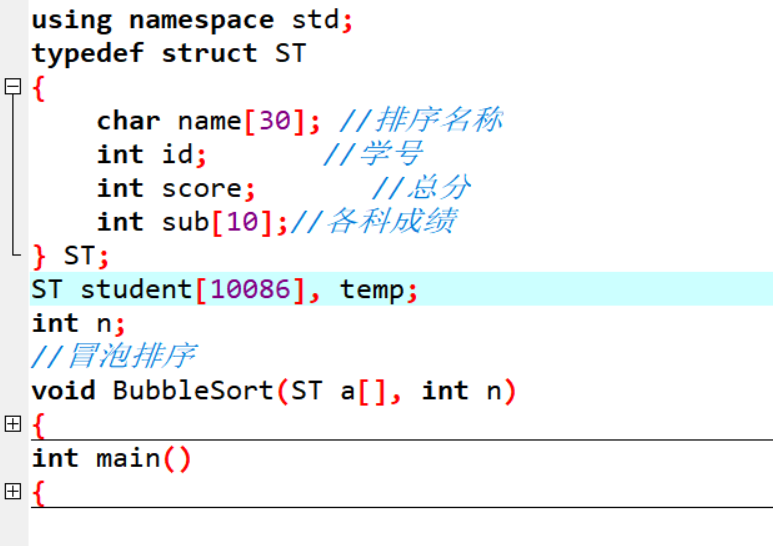
给出n个学生的m门考试的成绩表，每个学生的信息由学号、姓名以及各科成绩组成。对学生的考试成绩进行有关统计，并打印统计表。

首先要按总数高低次序，打印出名次表，分数相同的为同一名次；

其次，要按名次打印出每个学生的学号、姓名、总分以及各科成绩。

**2.2设计方案**

1. **使用struct 结构体记录一个学生的信息**
2. **使用冒泡排序对学生进行排序**
3. **主函数**

****

**2.3算法**

1. **使用struct 结构体记录一个学生的信息**

typedef struct ST

{

char name[30]; //排序名称

int id; //学号

int score; //总分

int sub[10];//各科成绩

} ST;

1. **使用冒泡排序对学生进行排序**

//冒泡排序

void BubbleSort(ST a[], int n)

{

for (int i = 0; i < n - 1; i++)

{

for (int j = n - 1; j > i; j--)

{

if (a[j].score < a[j - 1].score)

{

swap(a[j], a[j - 1]);

}

else if(a[j-1].score == a[j].score && strcmp(a[j-1].name , a[j].name) > 0)//成绩相同时比较姓名

{

swap(a[j], a[j - 1]);

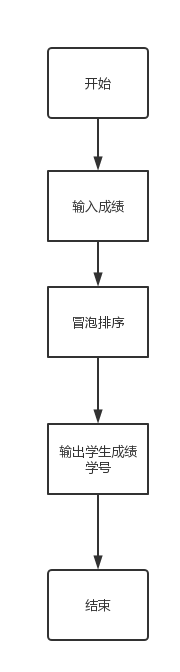
}

}

}

}

**2.4设计图**

****

**2.5程序**

#include <cstdio>

#include <cstdlib>

#include <string.h>

#include <ctime>

#include <iostream>

#define MAX 150

using namespace std;

typedef struct ST

{

char name[30]; //排序名称

int id; //学号

int score; //总分

int sub[10];//各科成绩

} ST;

ST student[10086], temp;

int n;

//冒泡排序

void BubbleSort(ST a[], int n)

{

for (int i = 0; i < n - 1; i++)

{

for (int j = n - 1; j > i; j--)

{

if (a[j].score < a[j - 1].score)

{

swap(a[j], a[j - 1]);

}

else if(a[j-1].score == a[j].score && strcmp(a[j-1].name , a[j].name) > 0)//成绩相同时比较姓名

{

swap(a[j], a[j - 1]);

}

}

}

}

int main()

{

cout << "请输入要录入的学生的个数 ： " << endl ;

cin >> n ;

cout << "输入姓名，学号和各科分数 " << endl ;

for( int i = 0 ; i < n ; i ++)

{

cin >> student[i].name >>student[i].id >> student[i].sub[0] >> student[i].sub[1] >> student[i].sub[2] ;

student[i].score = student[i].sub[0] + student[i].sub[1] + student[i].sub[2];

}

BubbleSort(student,n);

int rank=1;//处理有相同分数

cout << "\t\t\t\t" << "按照名次打印学生信息表如下：" << endl ;

//按名次打印学生成绩表及其信息

cout <<"\t"<< "rank"<<"\t" << "name" << "\t" << "id" << "\t" << "score" << "\t" << "english" << "\t" << "chinese" << "\t" << "math" << endl ;

for(int i = 0 ; i < n ; i ++)

{

if((i!=n)&&(student[i].score==student[i+1].score))

{

cout <<"\t"<< rank <<"\t" << student[i].name << " " << "\t"<< student[i].id << " "<< "\t"<< student[i].score <<" " << "\t"<< student[i].sub[0]

<< " "<< "\t"<< student[i].sub[1] << " "<< "\t"<< student[i].sub[2] ;

}

else

{

cout <<"\t"<< rank <<"\t" << student[i].name << " " << "\t"<< student[i].id << " "<< "\t"<< student[i].score <<" " << "\t"<< student[i].sub[0]

<< " "<< "\t"<< student[i].sub[1] << " "<< "\t"<< student[i].sub[2] ;

rank++;

}

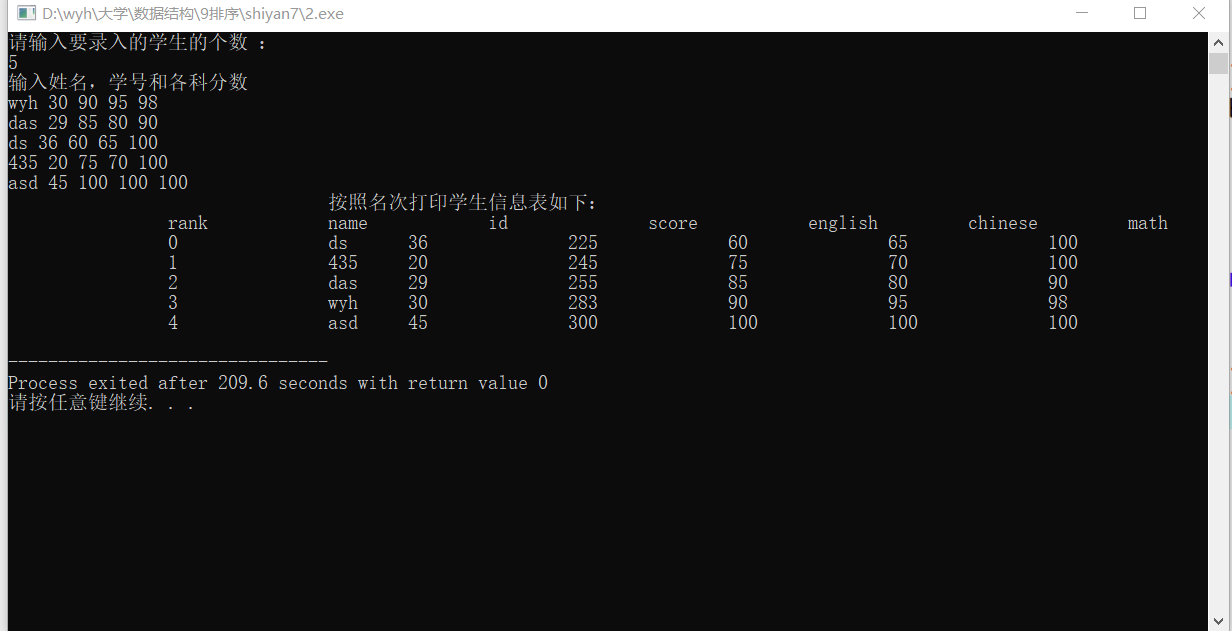
cout << endl ;

}

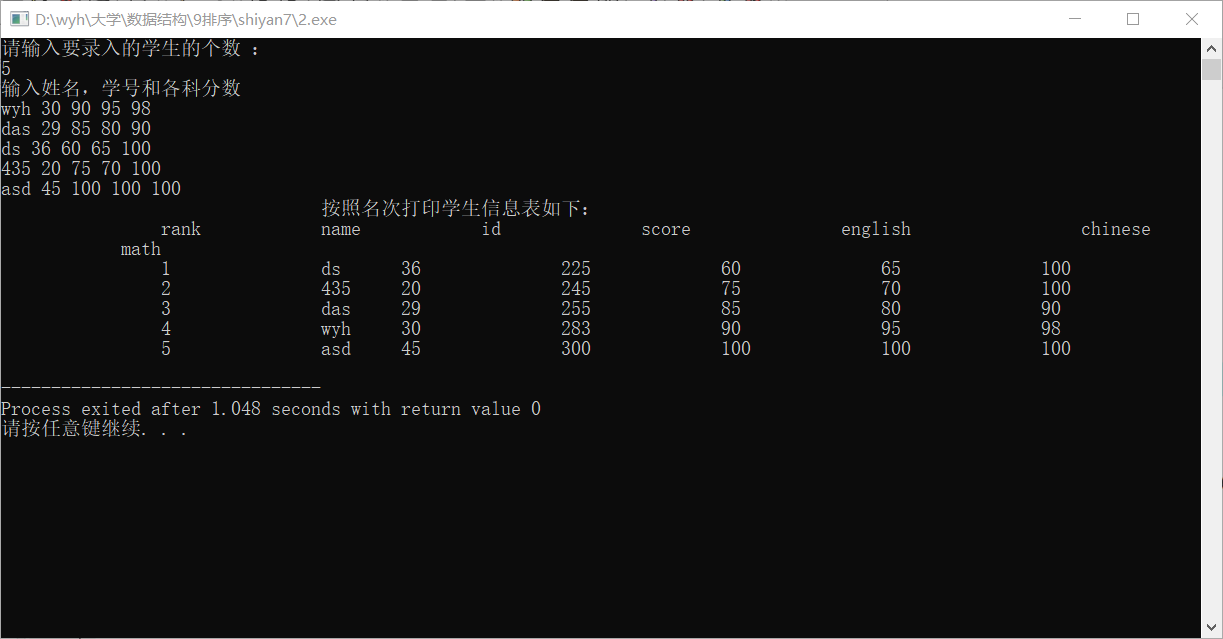
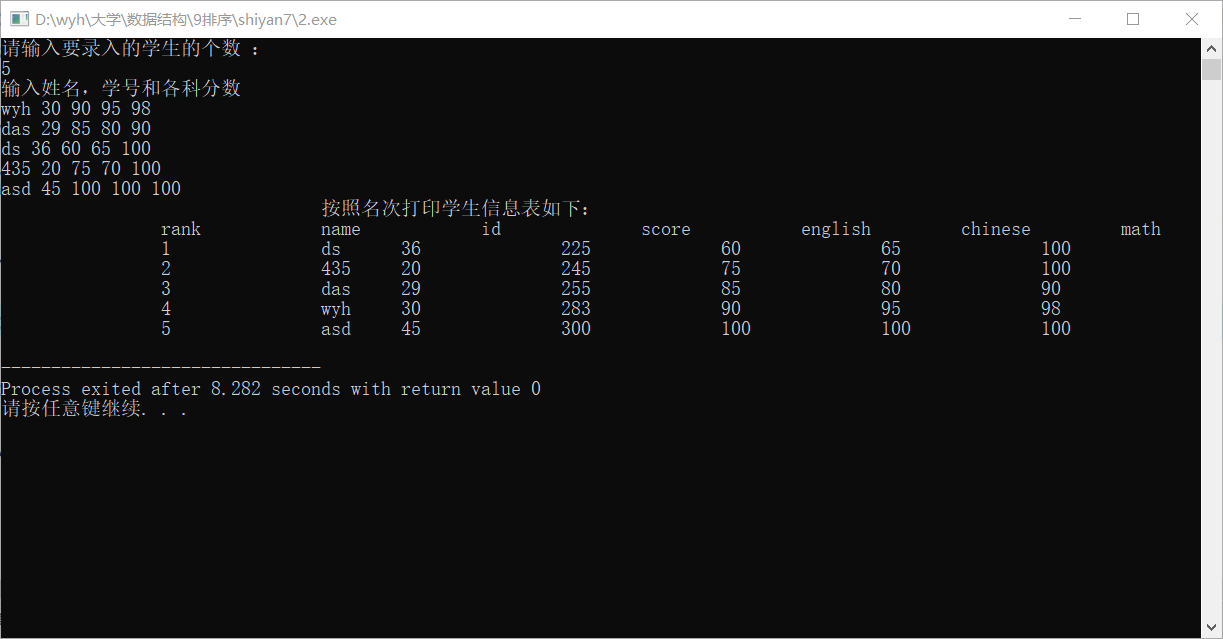
return 0;

}

**2.6调试过程截图**

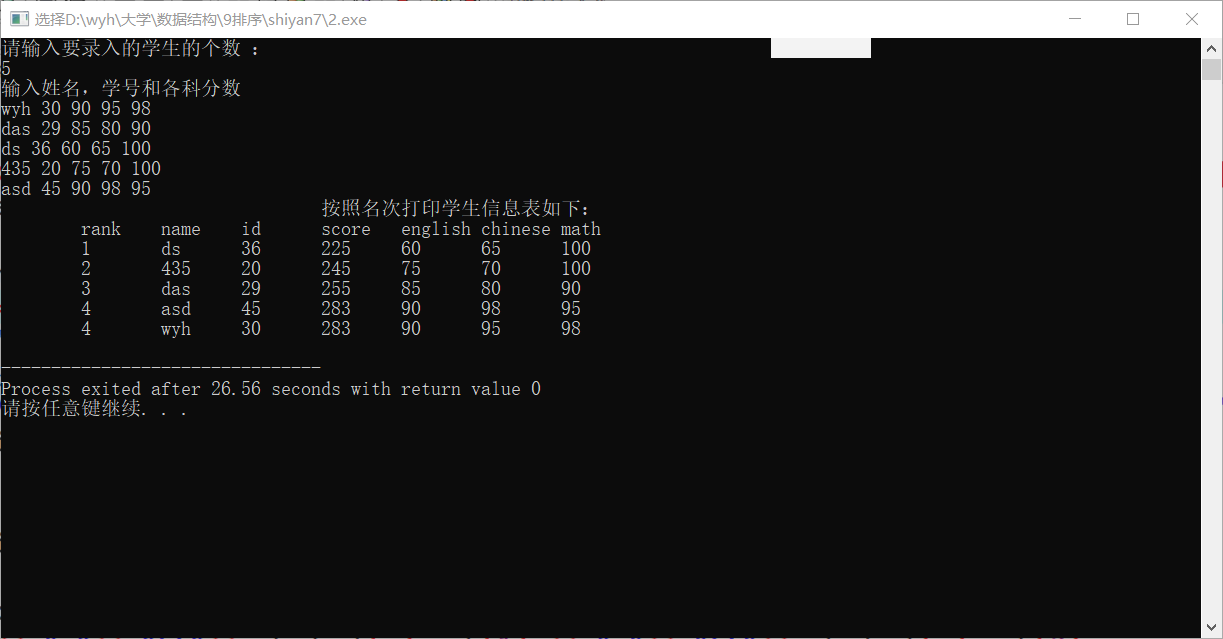
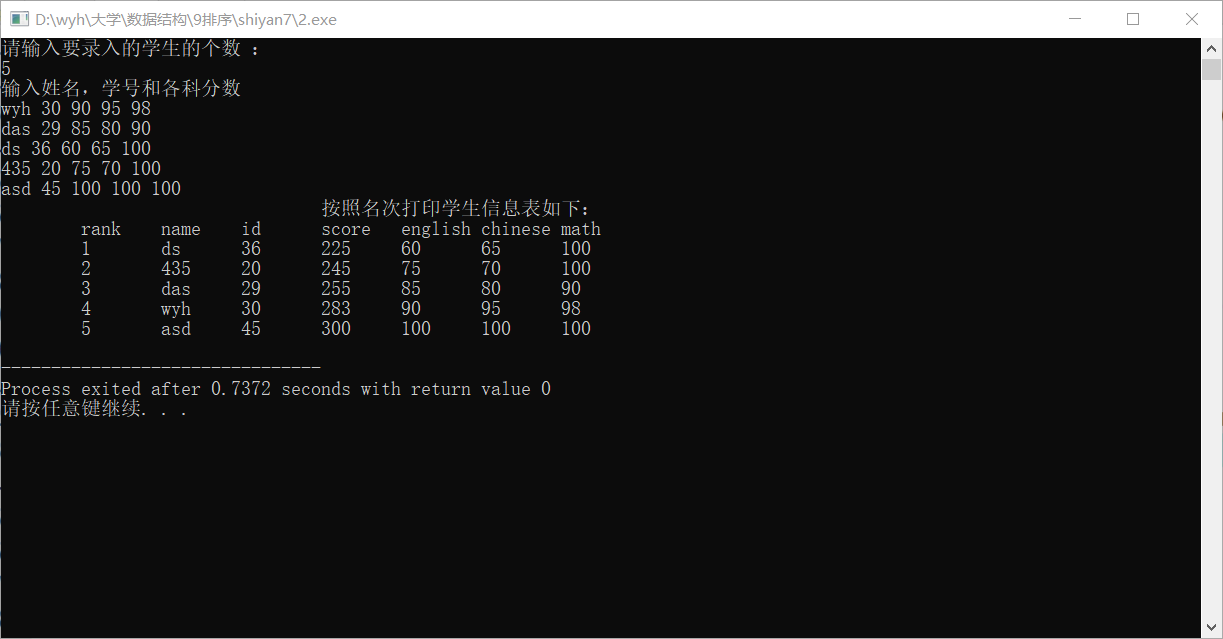
****

**排名应该从1开始**

****

**输出格式不规范，没有对齐**

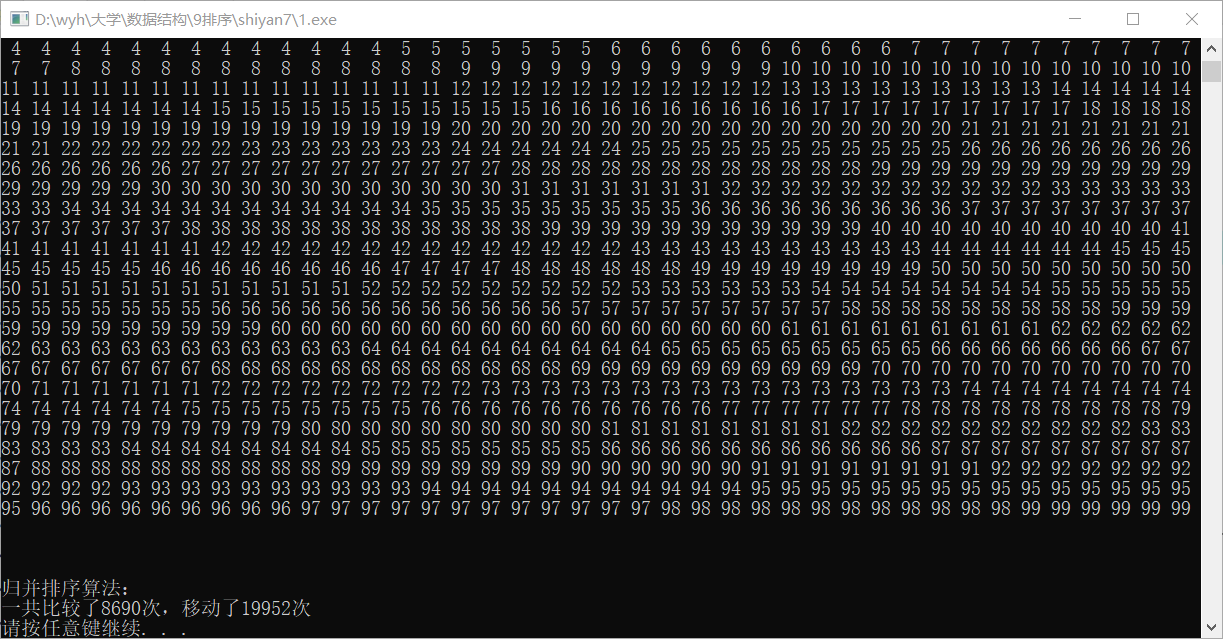
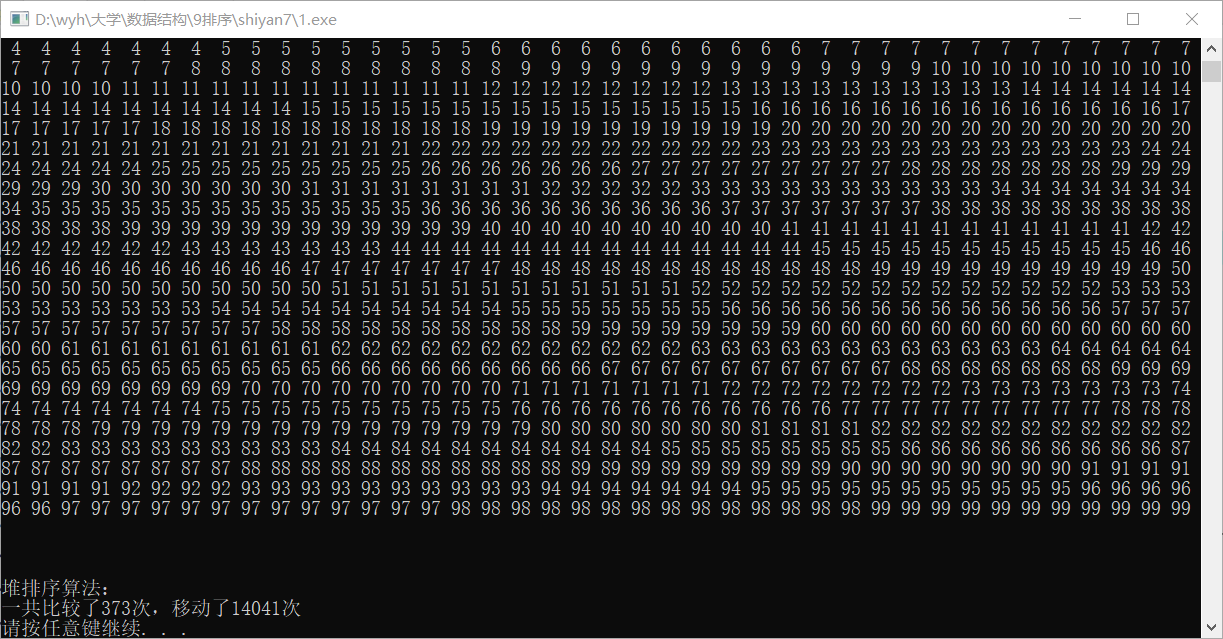
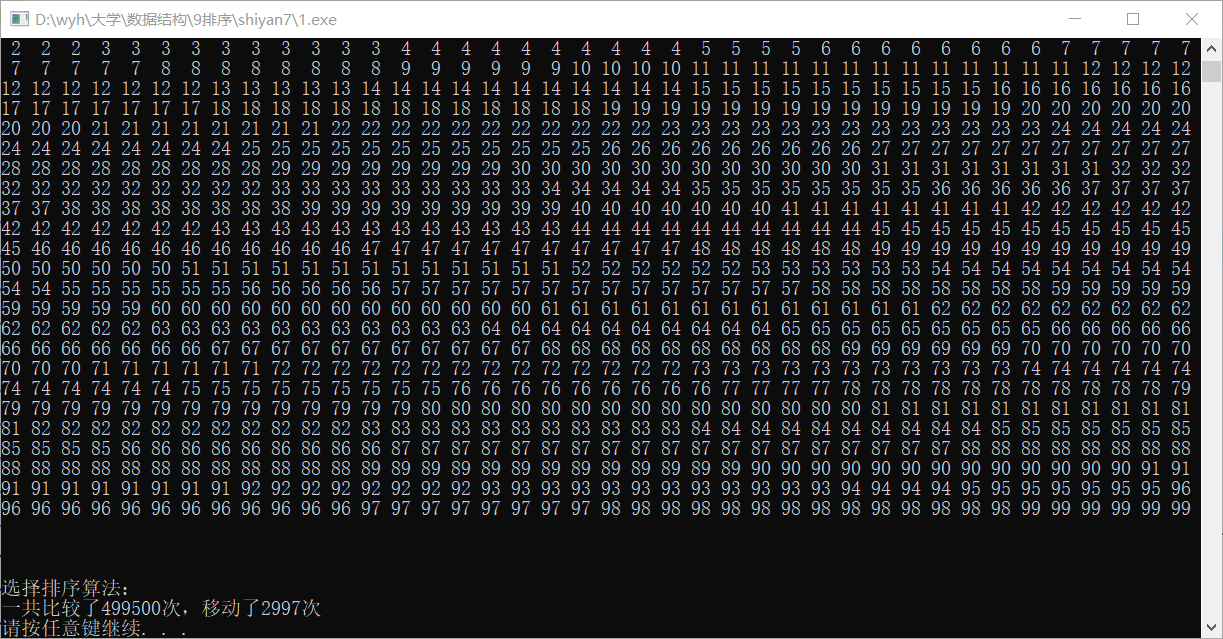
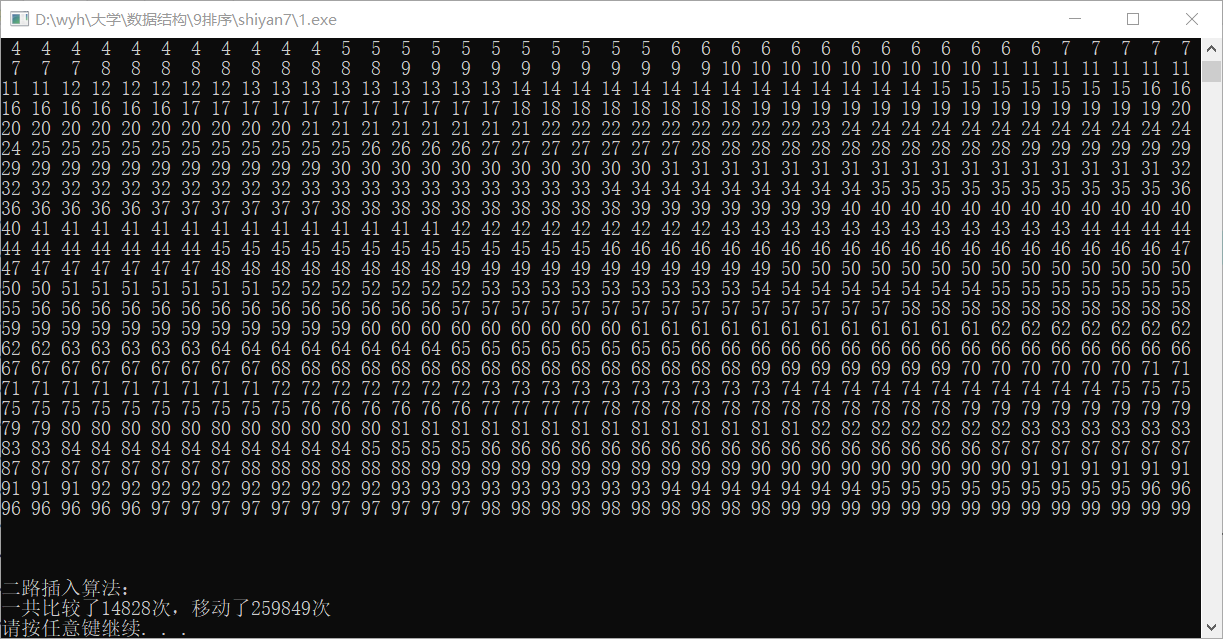
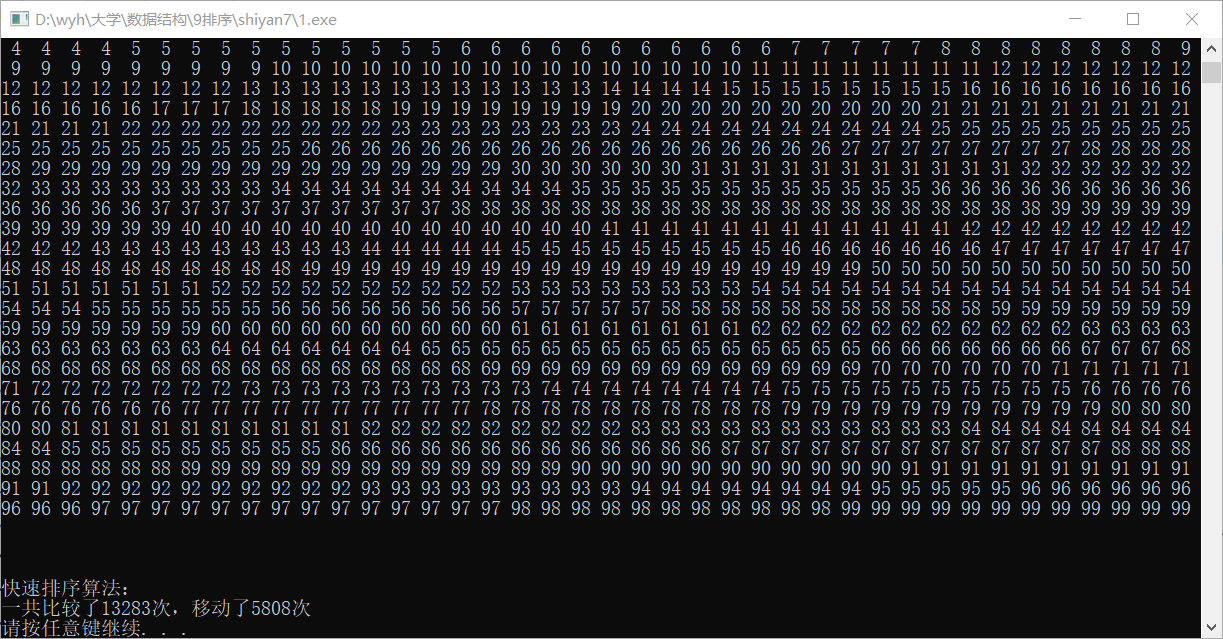
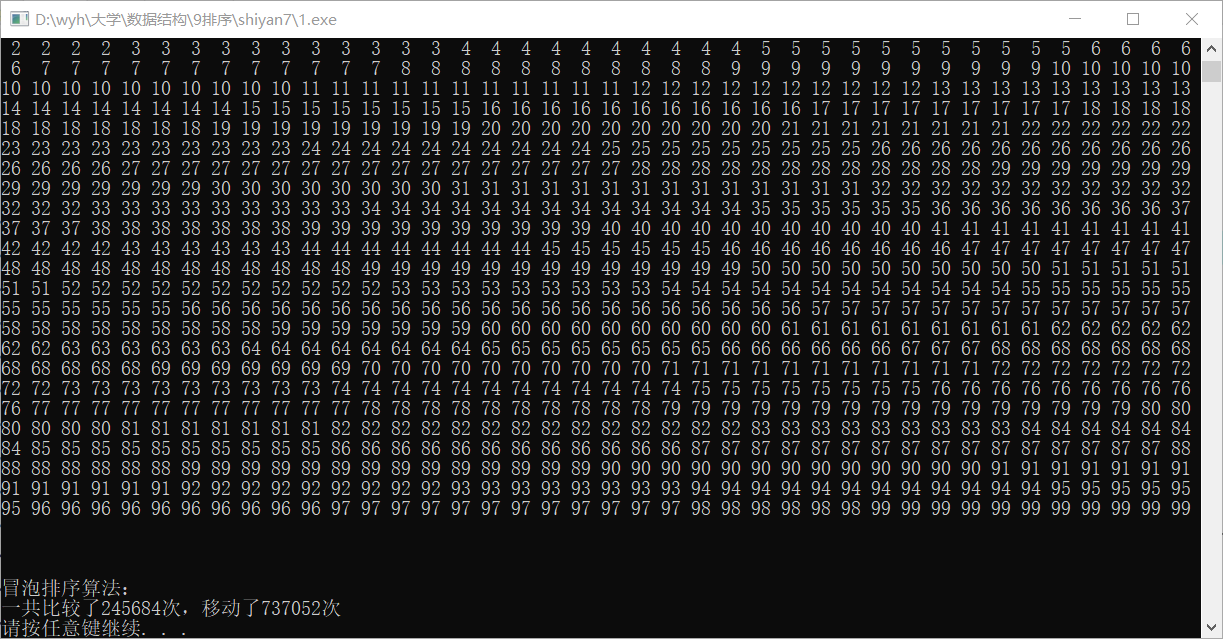
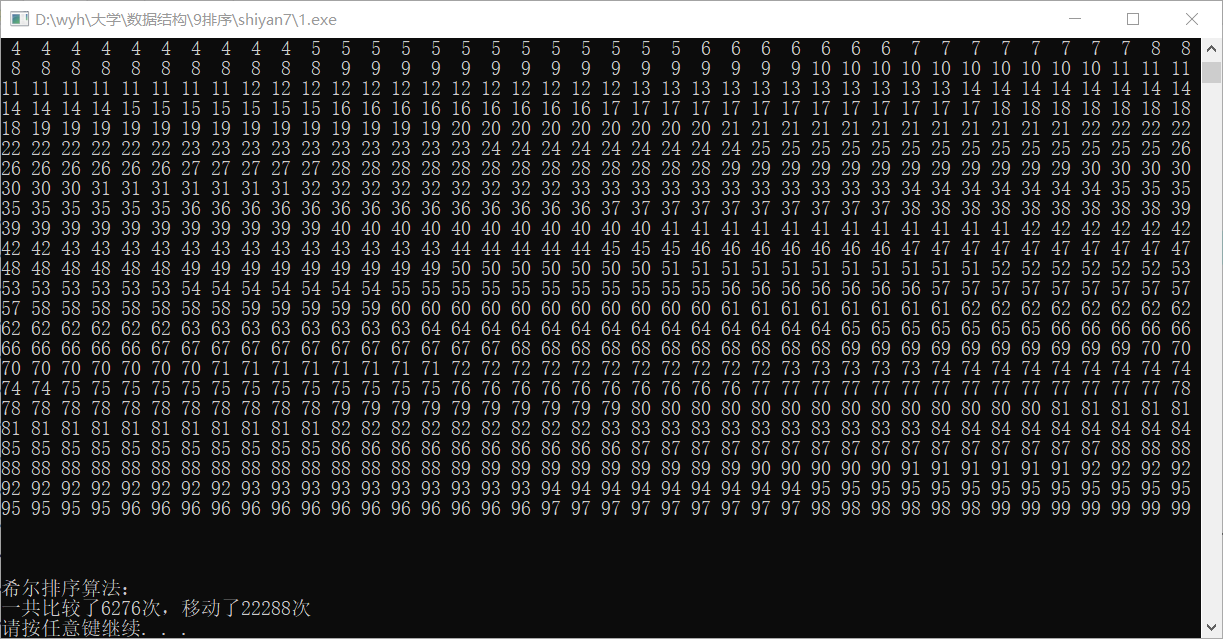
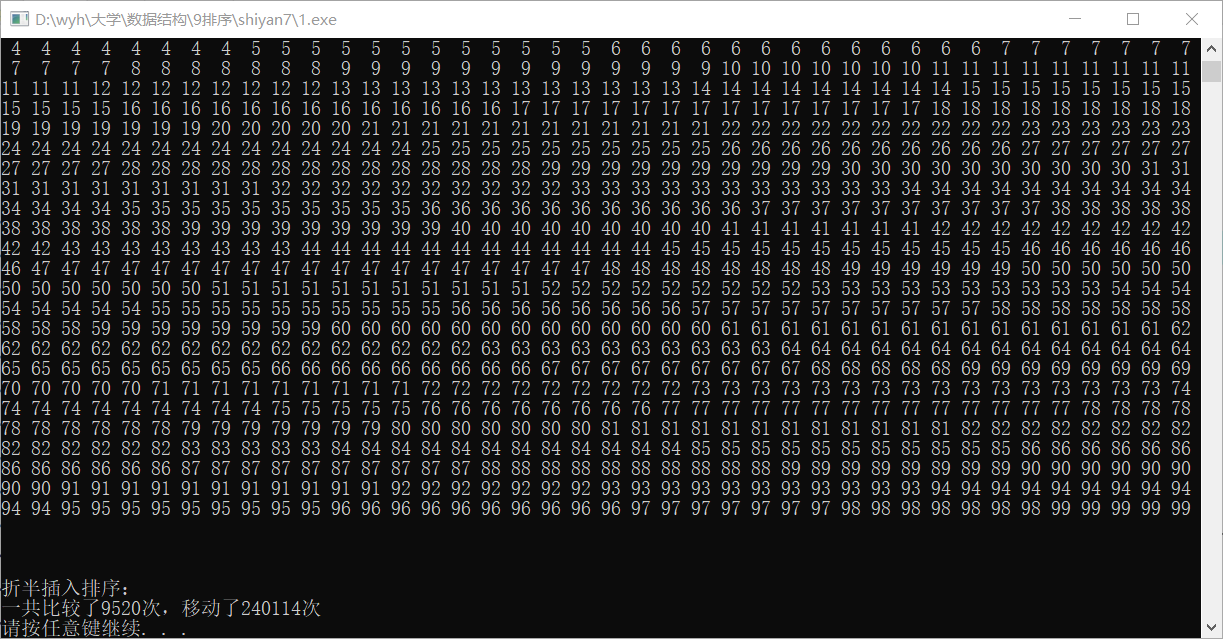
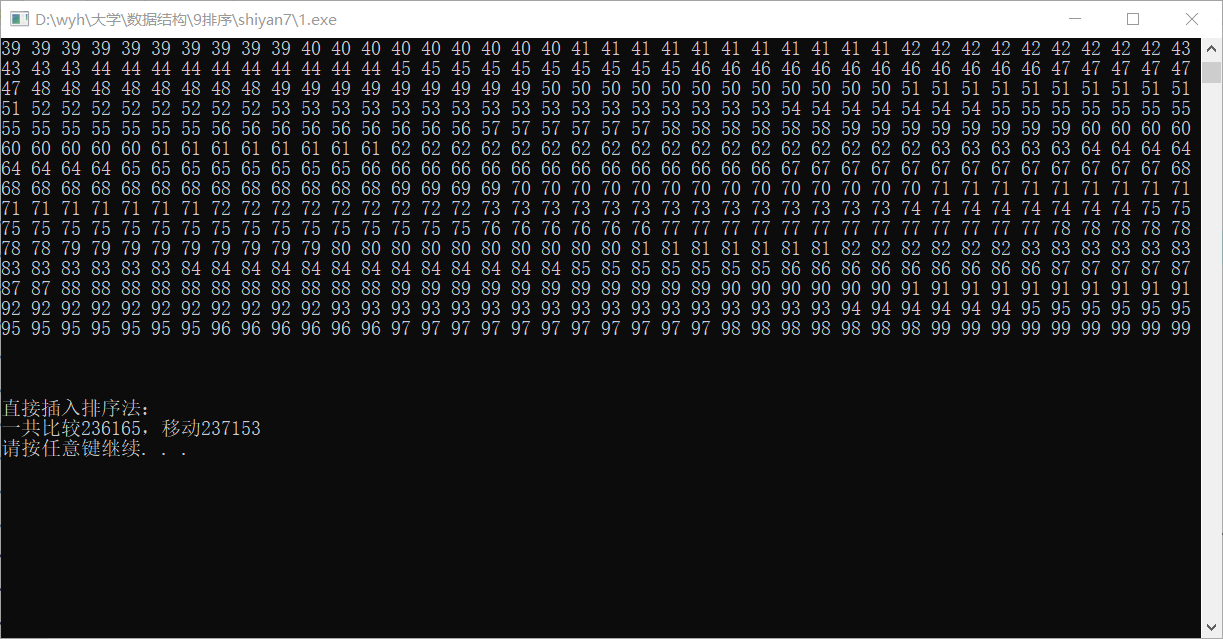
**2.7运行结果截图**

****

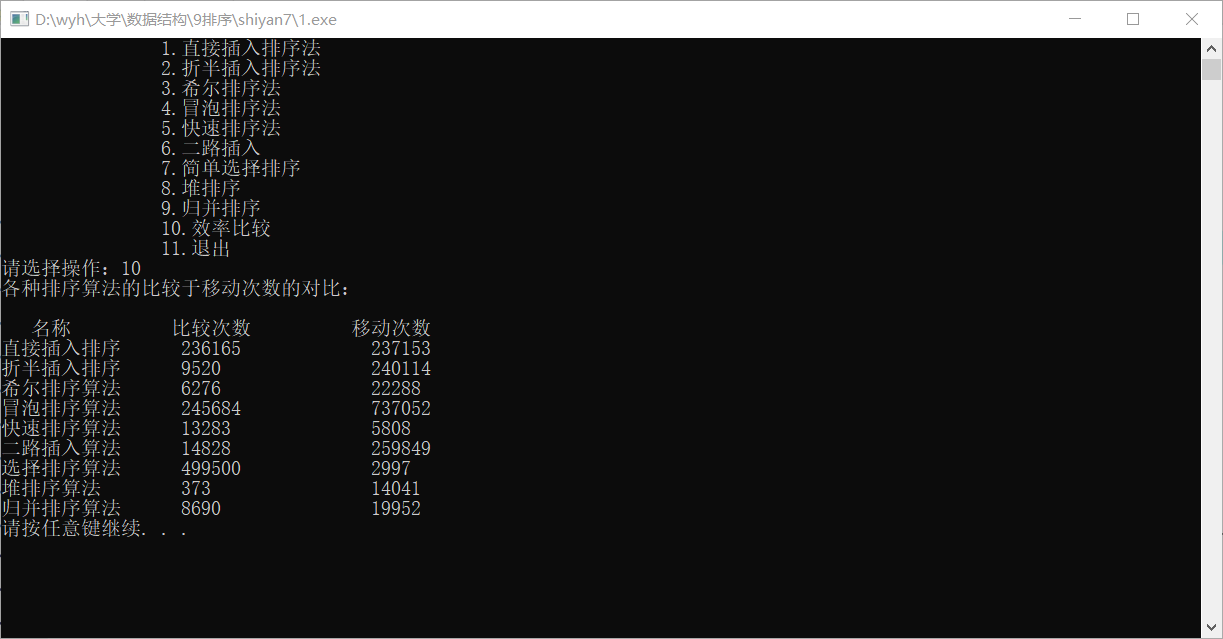
**特殊情况：**成绩相同的情况

**3选做**

将排序数组大小改为1000

****

**结果：**

****

1. **实验心得**

本次实验，我了解了常用的9种排序：直接插入排序；折半折入排序；二路插入排序；希尔排序；起泡排序；快速排序；简单选择排序；堆排序；归并排序。排序是计算机内经常进行的一种操作，其目的是将一组“无序”的记录序列调整为“有序”的记录序列。分[内部排序](https://baike.baidu.com/item/%E5%86%85%E9%83%A8%E6%8E%92%E5%BA%8F" \t "https://baike.baidu.com/item/%E6%8E%92%E5%BA%8F/_blank)和[外部排序](https://baike.baidu.com/item/%E5%A4%96%E9%83%A8%E6%8E%92%E5%BA%8F" \t "https://baike.baidu.com/item/%E6%8E%92%E5%BA%8F/_blank)，若整个排序过程不需要访问外存便能完成，则称此类排序问题为内部排序。反之，若参加排序的记录数量很大，整个序列的排序过程不可能在[内存](https://baike.baidu.com/item/%E5%86%85%E5%AD%98/103614" \t "https://baike.baidu.com/item/%E6%8E%92%E5%BA%8F/_blank)中完成，则称此类排序问题为外部排序。内部排序的过程是一个逐步扩大记录的有序序列长度的过程。

以下是几种主要的排序方法：

冒泡排序：每一轮排序后最大（或最小）的数将移动到数据序列的最后，理论上总共要进行n(n-1）/2次交换。首先比较a[1]与a[2]的值，若a[1]大于a[2]则交换两者的值，否则不变。再比较a[2]与a[3]的值，若a[2]大于a[3]则交换两者的值，否则不变。

插入排序：所有排序方法中最简单的一种算法，其主要的实现思想是将数据按照一定的顺序一个一个的插入到有序的表中，最终得到的序列就是已经排序好的数据。

选择排序：对于具有 n 个记录的无序表遍历 n-1 次，第 i 次从无序表中第 i 个记录开始，找出后序关键字中最小的记录，然后放置在第 i 的位置上。

希尔排序：已知一组无序数据a[1]、a[2]、……a[n]，需将其按升序排列。发现当n不大时，[插入排序](https://baike.baidu.com/item/%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F" \t "https://baike.baidu.com/item/%E6%8E%92%E5%BA%8F/_blank)的效果很好。首先取一增量d(d<n)，将a[1]、a[1+d]、a[1+2d]……列为第一组，a[2]、a[2+d]、a[2+2d]……列为第二组……，a[d]、a[2d]、a[3d]……列为最后一组以次类推，在各组内用插入排序，然后取d'<d，重复上述操作，直到d=1。

快速排序：已知一组无序数据a[1]、a[2]、……a[n]，需将其按升序排列。首先任取数据a[x]作为基准。比较a[x]与其它数据并排序，使a[x]排在数据的第k位，并且使a[1]~a[k-1]中的每一个数据<a[x]，a[k+1]~a[n]中的每一个数据>a[x]，然后采用分治的策略分别对a[1]~a[k-1]和a[k+1]~a[n]两组数据进行[快速排序](https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F" \t "https://baike.baidu.com/item/%E6%8E%92%E5%BA%8F/_blank)。

在第二个实验中，运用一个结构体和数组对学生的成绩排序，且运用了冒泡排序。

这两个实验让我明白了只有多实践，才能更好的理解掌握数据结构的精髓。从而避免基础语法错误，让代码变得更简洁高效。如此才能准确高效的解决问题。在今后的编程过程中要更注重代码整体设计的提纲记录，用更多的注释，让自己的思路更清晰。