数据结构上机实验题实验报告（四）

> 题目: 内部排序算法比较
> 姓名：王磊
> 班级：2020211311
> 学号：2020211538
> 提交日期：2021年12月14日

一 题目描述

1. 输入形式：长度不小于100的伪随机数排序表。
2. 输出形式：每种排序方式的关键字的比较次数和关键字的移动次数。。
3. 程序功能：读入待排序表，分别使用六种排序方法进行排序，给出每种排序方法关键字参加的比较次数和关键字的移动次数并作出分析。

二 程序设计

1. **定义结构变量SqList代表待排序表**

```
typedef struct {
    int l[MAXSIZE + 1];
    int length;
}SqList;
```

2. **概要设计**

主程序中定义了一个长为201（下标为0的位置留作哨兵）的顺序表结构存放需要排序的数据，主程序运行之后，先由随机数生成函数初始化顺序表T，再为T1--T5赋给T完全相同的值，对T和T1--T5分别使用六种排序方法进行排序操作，输出每种排序过程中关键字的比较次数和移动次数。该过程循环5次，即用5个不同的待排序表进行实验。

1. BubbleSort模块调用了swap函数
2. SelectSort模块调用了swap函数
3. QuickSort模块调用了Qsort函数，同时Qsort模块调用了Partition函数，Partition调用了swap函数
4. HeapSort模块调用了HeapAdjust函数，HeapAdjust函数中又调用了swap函数

以下为函数声明

```
void swap(SqList* L, int m, int n);
    //交换L中下标为m和n的元素的位置
void BubbleSort(SqList* L);
    //冒泡排序
void InsertSort(SqList* L);
    //直接插入排序
void SelectSort(SqList* L);
    //简单选择排序
```

```
void QuickSort(SqList* L);
    //快速排序
int Partition(SqList* L, int low, int high, int& CompareCount, int& MoveCount);
    //快速排序中用于确定枢轴位置
void QSort(SqList* L, int low, int high, int& CompareCount, int& MoveCount);
    //对子表进行快速排序的递归程序
void ShellSort(SqList* L);
    //希尔排序
void HeapSort(SqList* L);
    //堆排序
void HeapAdjust(SqList* L, int s, int m, int& CompareCount, int& MoveCount);
    //将表调整为大根堆
```

3. **详细设计**

1. 头文件

```
#include<stdio.h>
#include<stdlib.h>
```

2. 外部变量定义

```
#define MAXSIZE 200
```

3. 结构体定义

```
typedef struct {
    int l[MAXSIZE + 1];
    int length;
}SqList;
```

4. swap交换函数

```
void swap(SqList* L, int m, int n)
{
    int temp = L->l[m];
    L->l[m] = L->l[n];
    L->l[n] = temp;
}
```

5. 冒泡排序

```
void BubbleSort(SqList* L)
{
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 1; i < L->length; i++) {
        for (int k = i + 1; k <= L->length; k++) {
            CompareCount++;
            if (L->l[i] > L->l[k]) {
                swap(L, i, k);
                MoveCount = MoveCount + 3;
            }
        }
    }
    printf("The times of comparision of BubbleSort is %d\n", CompareCount);
    printf("The times of movement of BubbleSort is %d\n", MoveCount);
```

```
}
```

6. 直接插入排序

```
void InsertSort(SqList* L)
{
    int j;
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 2; i < L->length; i++) {
        CompareCount++;
        if (L->l[i] < L->l[i - 1]) {
            L->l[0] = L->l[i];
            MoveCount++;
            for ( j = i - 1; L->l[j] > L->l[0]; j--) {
                L->l[j + 1] = L->l[j];
                CompareCount++;
                MoveCount++;
            }
            CompareCount++;
            L->l[j + 1] = L->l[0];
            MoveCount++;
        }
    }
    printf("The times of comparision of InsertSort is %d\n", CompareCount);
    printf("The times of movement of InsertSort is %d\n", MoveCount);
}
```

7. 简单选择排序

```
void SelectSort(SqList* L)
{
    int min;
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 1; i < L->length; i++) {
        min = i;
        for (int j = i + 1; j <= L->length; j++) {
            CompareCount++;
            if (L->l[min] > L->l[j]) {
                min = j;
            }
        }
        if (i != min) {
            swap(L, i, min);
            MoveCount += 3;
        }
    }
    printf("The times of comparision of SelectSort is %d\n", CompareCount);
    printf("The times of movement of SelectSort is %d\n", MoveCount);
}
```

8. 快速排序

```
void QuickSort(SqList* L)
{
    int MoveCount = 0;
    int CompareCount = 0;
    QSort(L, MoveCount, L->length,CompareCount,MoveCount);
```

```
    printf("The times of comparision of QuickSort is %d\n", CompareCount);
    printf("The times of movement of QuickSort is %d\n", MoveCount);
}
void QSort(SqList* L, int low, int high, int& CompareCount, int& MoveCount)
{
    int pivotloc;
    if (low < high) {
        pivotloc = Partition(L, low, high, CompareCount, MoveCount);
        QSort(L, low, pivotloc - 1, CompareCount, MoveCount);
        QSort(L, pivotloc + 1, high, CompareCount, MoveCount);
    }
}
int Partition(SqList* L, int low, int high, int& CompareCount, int& MoveCount)
{
    int pivotkey;
    L->l[0] = L->l[low];
    pivotkey = L->l[low];
    while (low < high) {
        while (low < high && L->l[high] >= pivotkey) {
            CompareCount++;
            --high;
        }
        CompareCount++;
        swap(L, low, high);
        MoveCount += 3;
        while (low < high && L->l[low] <= pivotkey) {
            CompareCount++;
            low++;
        }
        CompareCount++;
        swap(L, low, high);
        MoveCount += 3;
    }
    return low;
}
```

9. 希尔排序

```
void ShellSort(SqList* L)
{
    int j;
    int MoveCount = 0;
    int CompareCount = 0;
    int increase = L->length;
    do {
        increase = increase / 5 + 1;
        for (int i = increase + 1; i < L->length; i++) {
            CompareCount++;
            if (L->l[i] < L->l[i - increase]) {
                L->l[0] = L->l[i];
                MoveCount++;
                for (j = i - increase; L->l[0] < L->l[j] && j>0; j = j - increase) {
                    CompareCount++;
                    L->l[j + increase] = L->l[j];
                    MoveCount++;
                }
                CompareCount++;
                L->l[j + increase] = L->l[0];
                MoveCount++;
            }
```

```
        }
    } while (increase > 1);
    printf("The times of comparision of ShellSort is %d\n", CompareCount);
    printf("The times of movement of ShellSort is %d\n", MoveCount);
}
```

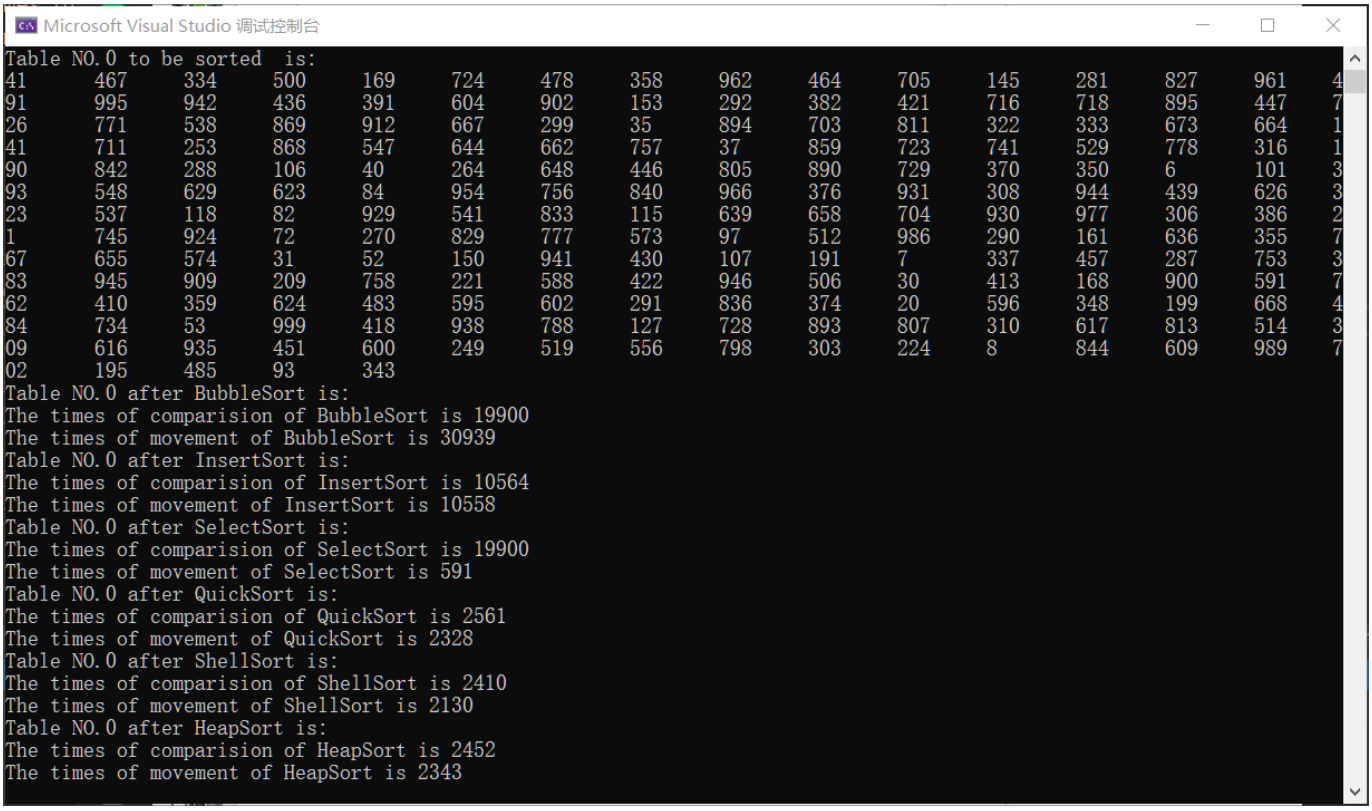
10. 堆排序

```
void HeapSort(SqList* L)
{
    int i;
    int MoveCount = 0;
    int CompareCount = 0;
    for (i = L->length / 2; i > 0; i--) {
        HeapAdjust(L, i, L->length, CompareCount, MoveCount);
    }
    for (i = L->length; i > 1; i--) {
        swap(L, 1, i);
        MoveCount += 3;
        HeapAdjust(L, 1, i - 1, CompareCount, MoveCount);
    }
    printf("The times of comparision of HeapSort is %d\n", CompareCount);
    printf("The times of movement of HeapSort is %d\n", MoveCount);
}
void HeapAdjust(SqList* L, int s, int m, int& CompareCount, int& MoveCount)
{
    int rc = L->l[s];
    MoveCount++;
    for (int i = 2 * s; i <= m; i *= 2) {
        CompareCount++;
        if (i < m && L->l[i] < L->l[i + 1]) {
            i++;
        }
        CompareCount++;
        if (rc >= L->l[i])
            break;
        L->l[s] = L->l[i];
        MoveCount++;
        s = i;
    }
    L->l[s] = rc;
    MoveCount++;
}
```

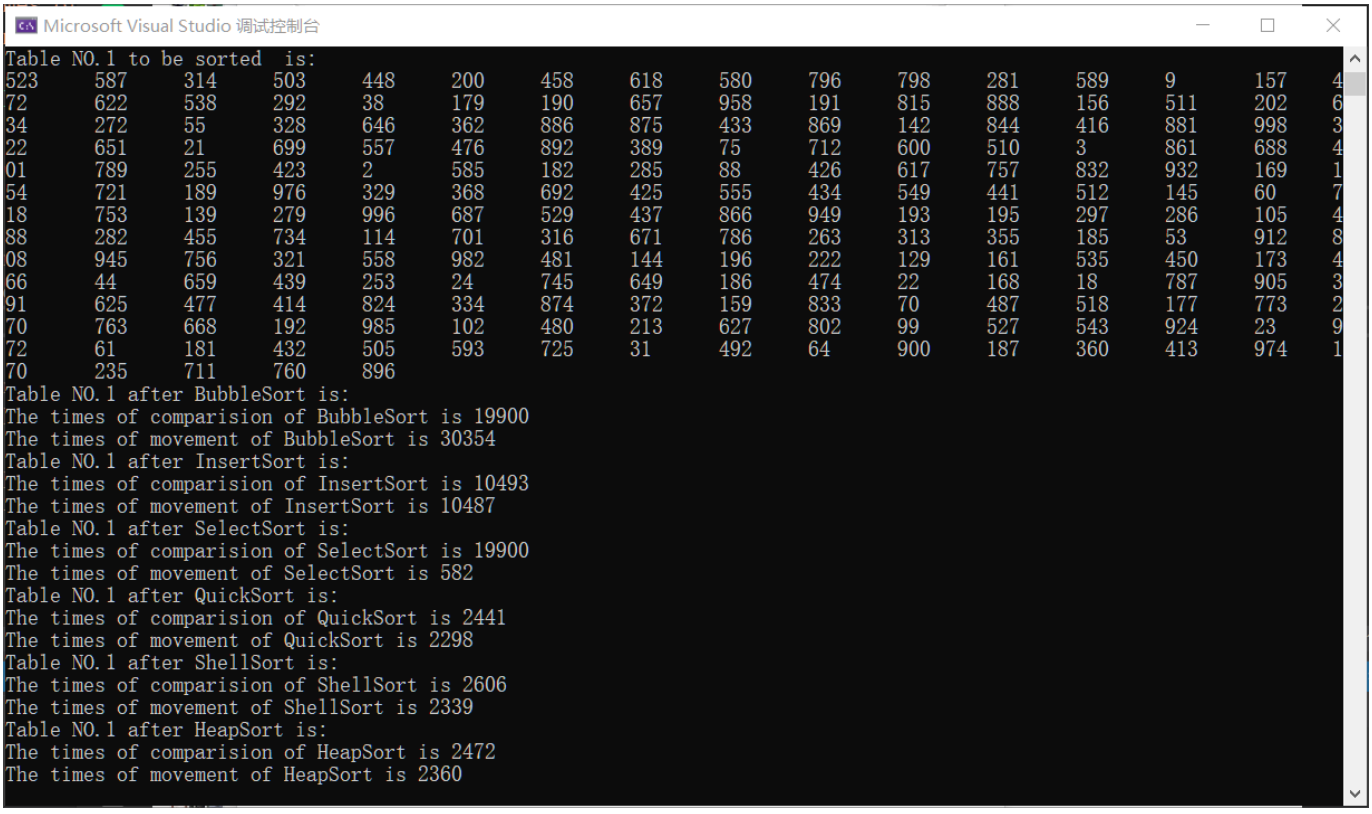
三 调试分析

- 编译环境：Visual Studio2019
- 运行环境：WIN10

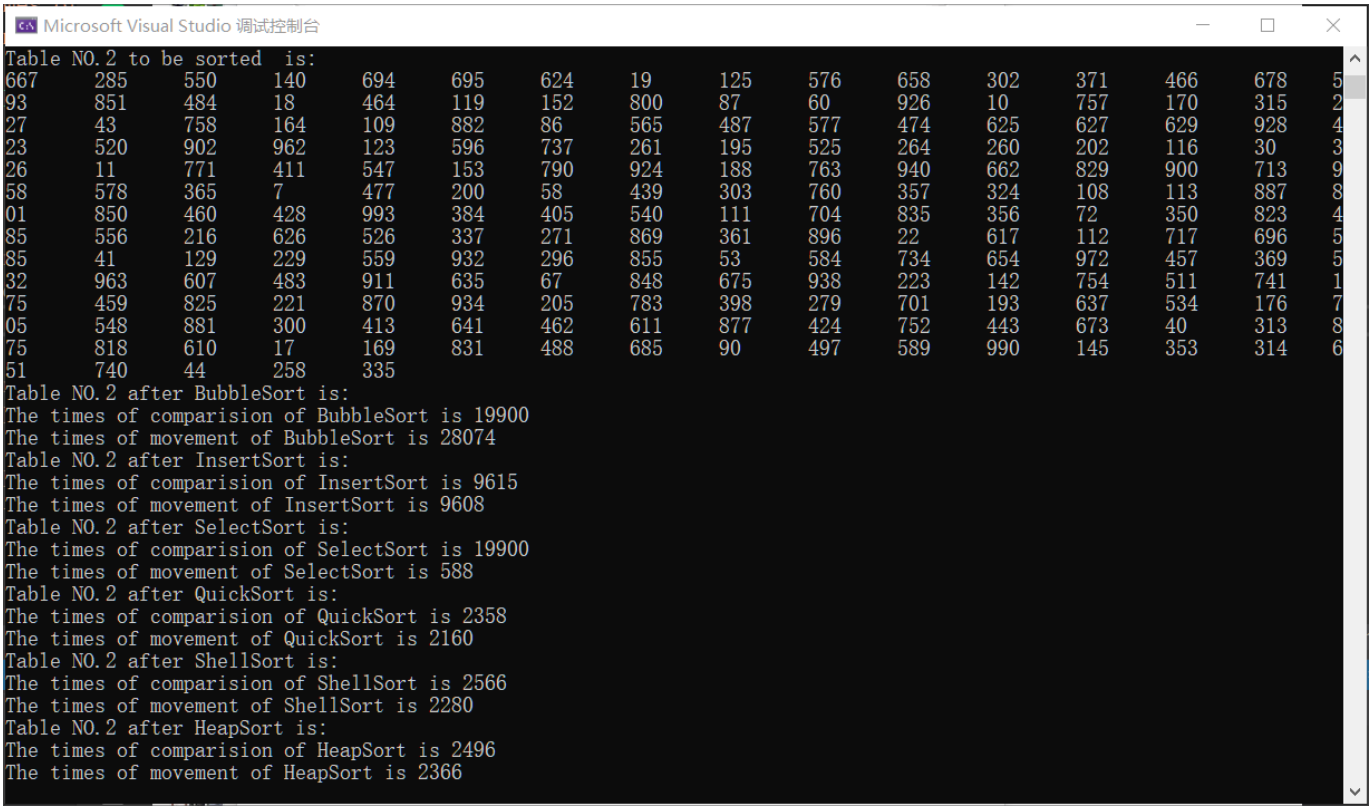
1. 第一组数据运行结果



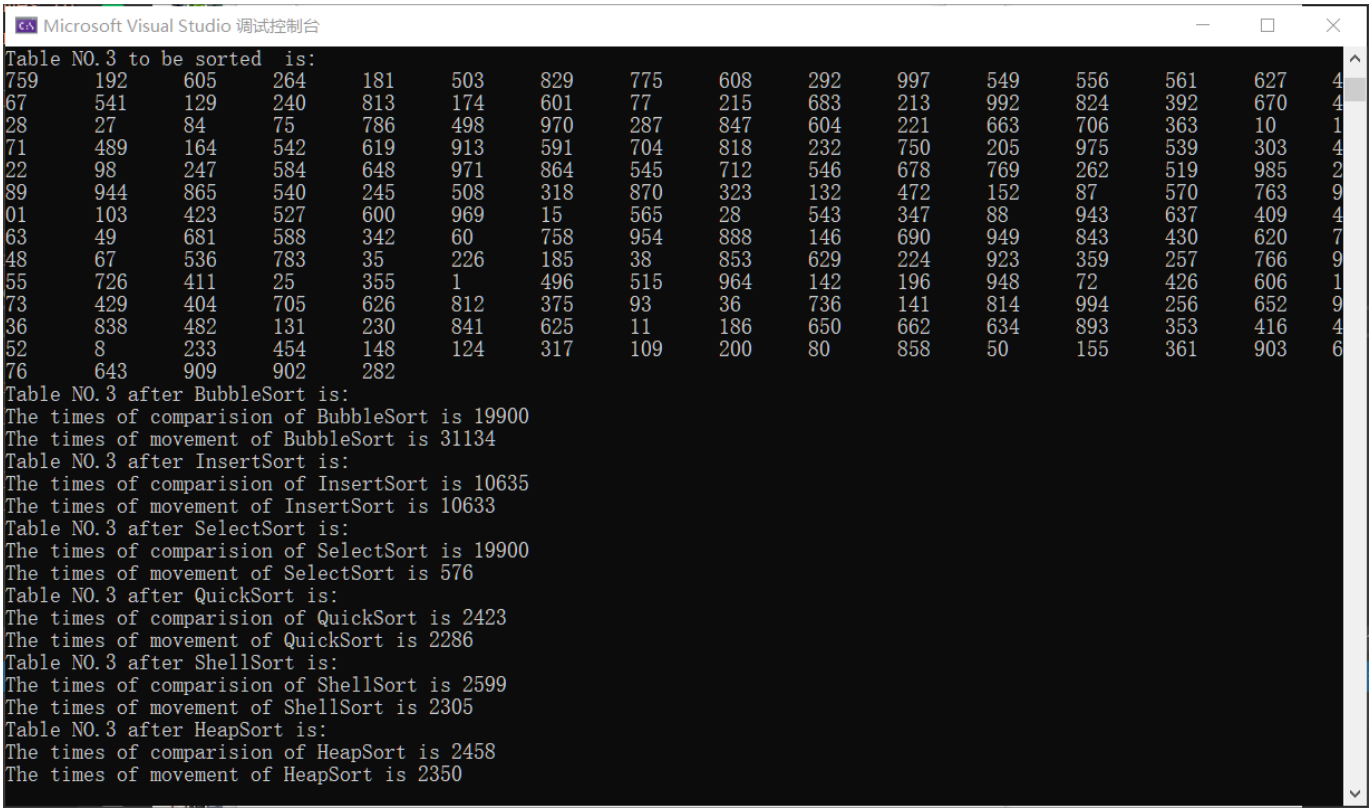
2. 第二组数据运行结果



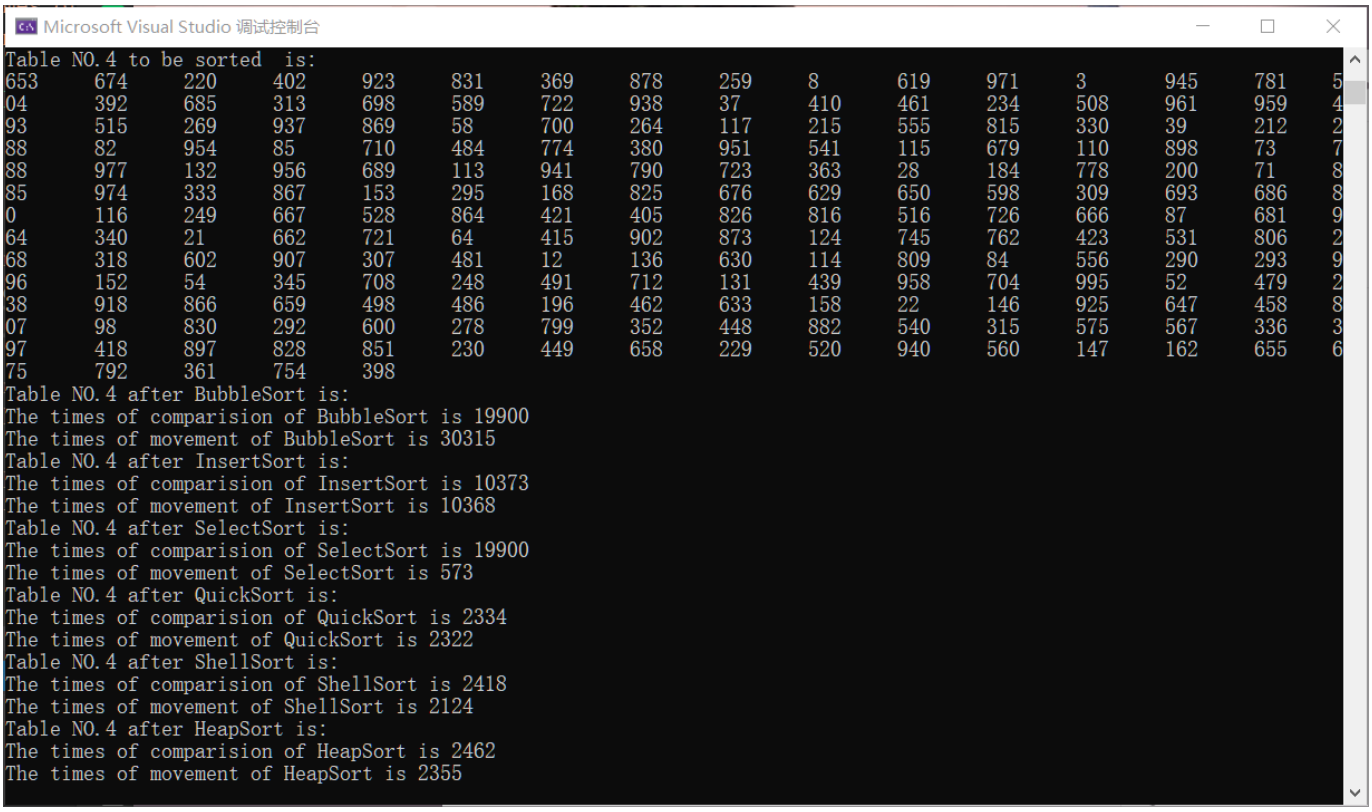
3. 第三组数据运行结果



4. 第四组数据运行结果



5. 第五组数据运行结果



四 工作总结

1. 现有的排序算法的程序实现都比较成熟，该实验的重点和难点在于找到合适的地方插入对关键字移动次数和比较次数的计数变量，大致的规律是在if语句之前加入一次比较，在swap之后加入三次移动。还有一些会出现比较和移动的地方，不同的排序方式都有差别。
2. 又遇到了传参的问题，由于C语言里没有传引用调用，所有只能通过指针的使用来模拟传引用调用，在这个过程中就特别考验对指针概念和符号的理解和掌握，例如在什么地方用*、在什么地方用&，都是需要考虑的问题。
3. 最后的结果基本符合预期，快速排序和堆排序都展现出了在时间复杂度上的优越性。

五 源代码

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 200
typedef struct {
    int l[MAXSIZE + 1];
    int length;
}SqList;
void swap(SqList* L, int m, int n);
void BubbleSort(SqList* L);
void InsertSort(SqList* L);
void SelectSort(SqList* L);
void QuickSort(SqList* L);
int Partition(SqList* L, int low, int high, int& CompareCount, int& MoveCount);
void QSort(SqList* L, int low, int high, int& CompareCount, int& MoveCount);
void ShellSort(SqList* L);
void HeapAdjust(SqList* L, int s, int m, int& CompareCount, int& MoveCount);
void HeapSort(SqList* L);
```

```
int main()
{
    int s;
    SqList T, T1, T2, T3, T4, T5;
    for (int i = 0; i < 5; i++) {
        bool check[1000] = { 0 };
        printf("Table NO.%d to be sorted  is:\n", i);
        for (int j = 1; j < (MAXSIZE + 1); j++) {
            s = rand() % 1000;
            while (check[s] == 1) {
                s = rand() % 1000;
            }
            check[s] = 1;
            T.l[j] = s;
            printf("%d\t", s);
        }
        T.length = MAXSIZE;
        printf("\n");
        T1 = T;
        T2 = T;
        T3 = T;
        T4 = T;
        T5 = T;
        printf("Table NO.%d after BubbleSort is:\n", i);
        BubbleSort(&T);
        printf("Table NO.%d after InsertSort is:\n", i);
        InsertSort(&T1);
        printf("Table NO.%d after SelectSort is:\n", i);
        SelectSort(&T2);
        printf("Table NO.%d after QuickSort is:\n", i);
        QuickSort(&T3);
        printf("Table NO.%d after ShellSort is:\n", i);
        ShellSort(&T4);
        printf("Table NO.%d after HeapSort is:\n", i);
        HeapSort(&T5);
    }
    return 0;
}
void swap(SqList* L, int m, int n)
{
    int temp = L->l[m];
    L->l[m] = L->l[n];
    L->l[n] = temp;
}
void BubbleSort(SqList* L)
{
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 1; i < L->length; i++) {
        for (int k = i + 1; k <= L->length; k++) {
            CompareCount++;
            if (L->l[i] > L->l[k]) {
                swap(L, i, k);
                MoveCount = MoveCount + 3;
            }
        }
    }
    printf("The times of comparision of BubbleSort is %d\n", CompareCount);
    printf("The times of movement of BubbleSort is %d\n", MoveCount);
}
void InsertSort(SqList* L)
```

```
{
    int j;
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 2; i < L->length; i++) {
        CompareCount++;
        if (L->l[i] < L->l[i - 1]) {
            L->l[0] = L->l[i];
            MoveCount++;
            for ( j = i - 1; L->l[j] > L->l[0]; j--) {
                L->l[j + 1] = L->l[j];
                CompareCount++;
                MoveCount++;
            }
            CompareCount++;
            L->l[j + 1] = L->l[0];
            MoveCount++;
        }
    }
    printf("The times of comparision of InsertSort is %d\n", CompareCount);
    printf("The times of movement of InsertSort is %d\n", MoveCount);
}
void SelectSort(SqList* L)
{
    int min;
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 1; i < L->length; i++) {
        min = i;
        for (int j = i + 1; j <= L->length; j++) {
            CompareCount++;
            if (L->l[min] > L->l[j]) {
                min = j;
            }
        }
        if (i != min) {
            swap(L, i, min);
            MoveCount += 3;
        }
    }
    printf("The times of comparision of SelectSort is %d\n", CompareCount);
    printf("The times of movement of SelectSort is %d\n", MoveCount);
}
int Partition(SqList* L, int low, int high, int& CompareCount, int& MoveCount)
{
    int pivotkey;
    L->l[0] = L->l[low];
    pivotkey = L->l[low];
    while (low < high) {
        while (low < high && L->l[high] >= pivotkey) {
            CompareCount++;
            --high;
        }
        CompareCount++;
        swap(L, low, high);
        MoveCount += 3;
        while (low < high && L->l[low] <= pivotkey) {
            CompareCount++;
            low++;
        }
        CompareCount++;
```

```
        swap(L, low, high);
        MoveCount += 3;
    }
    return low;
}
void QSort(SqList* L, int low, int high, int& CompareCount, int& MoveCount)
{
    int pivotloc;
    if (low < high) {
        pivotloc = Partition(L, low, high, CompareCount, MoveCount);
        QSort(L, low, pivotloc - 1, CompareCount, MoveCount);
        QSort(L, pivotloc + 1, high, CompareCount, MoveCount);
    }
}
void QuickSort(SqList* L)
{
    int MoveCount = 0;
    int CompareCount = 0;
    QSort(L, MoveCount, L->length,CompareCount,MoveCount);
    printf("The times of comparision of QuickSort is %d\n", CompareCount);
    printf("The times of movement of QuickSort is %d\n", MoveCount);
}



void ShellSort(SqList* L)
{
    int j;
    int MoveCount = 0;
    int CompareCount = 0;
    int increase = L->length;
    do {
        increase = increase / 5 + 1;
        for (int i = increase + 1; i < L->length; i++) {
            CompareCount++;
            if (L->l[i] < L->l[i - increase]) {
                L->l[0] = L->l[i];
                MoveCount++;
                for (j = i - increase; L->l[0] < L->l[j] && j>0; j = j - increase) {
                    CompareCount++;
                    L->l[j + increase] = L->l[j];
                    MoveCount++;
                }
                CompareCount++;
                L->l[j + increase] = L->l[0];
                MoveCount++;
            }
        }
    } while (increase > 1);
    printf("The times of comparision of ShellSort is %d\n", CompareCount);
    printf("The times of movement of ShellSort is %d\n", MoveCount);
}
void HeapAdjust(SqList* L, int s, int m, int& CompareCount, int& MoveCount)
{
    int rc = L->l[s];
    MoveCount++;
    for (int i = 2 * s; i <= m; i *= 2) {
        CompareCount++;
        if (i < m && L->l[i] < L->l[i + 1]) {
            i++;
        }
```

```
        CompareCount++;
        if (rc >= L->l[i])
            break;
        L->l[s] = L->l[i];
        MoveCount++;
        s = i;
    }
    L->l[s] = rc;
    MoveCount++;
}
void HeapSort(SqList* L)
{
    int i;
    int MoveCount = 0;
    int CompareCount = 0;
    for (i = L->length / 2; i > 0; i--) {
        HeapAdjust(L, i, L->length, CompareCount, MoveCount);
    }
    for (i = L->length; i > 1; i--) {
        swap(L, 1, i);
        MoveCount += 3;
        HeapAdjust(L, 1, i - 1, CompareCount, MoveCount);
    }
    printf("The times of comparision of HeapSort is %d\n", CompareCount);
    printf("The times of movement of HeapSort is %d\n", MoveCount);
}
```