****

**Southwest university of science and technology**

《数据结构》大作业

|  |  |
| --- | --- |
| 课题名称 | 题7：内排序算法比较 |
| 专业班级 | 信安1802 |
| 学生姓名 | 侯天铭 |
| 学号+电话 | 5120182377 18990316260 |
| 指导教师 | 曾立胜 |

评分细则

|  |  |  |
| --- | --- | --- |
| 评分项 | 具体评分项（30分） | 得分 |
| 【1】设计过程 | 学习态度（2分） |  |
| 学习进度（3分） |  |
| 【2】设计报告 | 需求分析 （2分） |  |
| 概要设计（5分） |  |
| 详细设计（10分） |  |
| 测试数据及结果（5分） |  |
| 调试分析及总结（3分） |  |
| 总分 |  |  |
| 评阅教师 | 2019 年 月 日 | |

目 录

一、课题描述 ............................................... （4）

二、需求分析 ............................................... （4）

三、概要设计 ............................................... （4）

四、详细设计 ............................................... （9）

五、测试数据及结果 ......................................... （15）

六、调试分析及总结 ......................................... （20）

1. 课题描述

　　在教科书中，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数或大概执行时间。试通过随机数据比较各算法的关键字比较次数和关键字移动次数，以取得直观数据。

1. 需求分析

比较的排序算法包括了：冒泡排序、直接插入排序、直接选择排序、快速排序、希尔排序、堆排序共6种排序方法。在给出相应的排序算法后，选择一种排序，便开始执行相应算法，并得到排序结果以及关键字的比较次数和移动次数。

软件需要实现以下功能：

1. 能对输入数据进行上述6种排序算法并得到结果；
2. 能由用户自己输入需要产生的随机数的个数以及随机数的取值范围；
3. 能通过随机数程序产生待排序的数据，在屏幕上输出产生的随机数，并将这组数据保存在random.txt文本文件中；
4. 能够将随机数程序产生的保存在random.txt中的待排序数据输入到当前程序中；
5. 在每一步操作完成后，都可以通过不同的按键来进入下一步或终止程序；
6. 每一次排序完成后能得到该种排序过程中关键字的比较次数和移动次数，并将其输出到屏幕上；
7. 最后将每一种排序的关键字比较次数、移动次数保存在output.txt文本文件中。
8. 概要设计
9. 数据逻辑结构、存储结构分析：

本程序只包含了一个数据集合，即待排序的数据集合（SqList）：待排序数据data数组、数据长度length。

待排序数据逻辑上是线性结构，存储上可以采用线性表的顺序存储结构，即顺序表存储。产生的随机数文件先保存到txt文件中，然后再读取出来存储在顺序表中进行排序。最后将结果（排序结果，关键字比较次数，关键字移动次数）输出并保存到txt文件中。

顺序表存储结构声明如下：

typedef struct{

int data[MaxSize];

int length;

}SqList;

其中MaxSize为待排序的随机数最多个数，在此设置为了100000。

大致流程图如下：

从当前目录下random.txt中读取数据，放入顺序表

产生随机数，保存到当前目录下random.txt中

将结果输出到当前目录下output.txt中

2）本程序包含11个子程序：

（1）初始化顺序表InitList(SqList &L,int a[],int n)

**参数：**L为结构体变量，即要创建的顺序表；

a数组用以接收产生的随机数；

n为随机数的个数，即顺序表长度。

**功能：**把待排序的随机数存储在顺序表中：将a[k]循环赋值给L.data[i](1<=i<=n)，并将最后的i-1赋给L.length完成顺序表的创建。（考虑到后面堆排序的需求，此时将顺序表从下标1开始建立）。

（2）交换两个变量Swap(int &a,int &b)

**参数：**a为第一个变量，b为第二个变量。

**功能：**交换a,b的值，使用引用符改变实参的值。

（3）冒泡排序BubbleSort(SqList L,double &e,double &ee)

**参数：**L为结构体变量，即存储了待排序随机数的顺序表；

e为冒泡排序过程中关键字的比较次数；

ee为冒泡排序过程中关键字的移动次数。

**功能：**使用冒泡排序对随机数序列进行排序，并将排序结果输出，同时计算出排序过程中关键字的比较次数和移动次数。

（4）插入排序InsertSort(SqList L,double &a1,double &aa)

**参数：**L为结构体变量，即存储了待排序随机数的顺序表；

a1为插入排序过程中关键字的比较次数；

aa为插入排序过程中关键字的移动次数。

**功能：**使用插入排序对随机数序列进行排序，并将排序结果输出，同时计算出排序过程中关键字的比较次数和移动次数。

（5）选择排序SelectSort(SqList L,double &f,double &ff)

**参数：**L为结构体变量，即存储了待排序随机数的顺序表；

f为选择排序过程中关键字的比较次数；

ff为选择排序过程中关键字的移动次数。

**功能：**使用选择排序对随机数序列进行排序，并将排序结果输出，同时计算出排序过程中关键字的比较次数和移动次数。

（6）快速排序区间划分partition（SqList &L,int s,int t,double &c,double &cc）

**参数：**L为结构体变量，即存储了待排序随机数的顺序表；

s为当前划分区间的起始位置（数组下标）；

t为当前划分区间的终点位置（下标）；

c为划分过程中关键字的比较次数；

cc为划分过程中关键字的移动次数。

**功能：**实现快速排序。快速排序的实质就是不断地调用partition算法，第一次划分以序列首元素为标准，小的放到其左边，大的放到其右边，从而将首元素放到了一个合适的位置（归位）。然后再对第一次划分后的左区间、右区间分别进行划分，以此类推，直到某区间中只剩1个或没有元素，停止递归返回上一级，最终实现快速排序。

（7）快速排序QuickSort(SqList &L,int s,int t,double &c,double &cc)

**参数：**K为结构体变量，即存储了待排序随机数的顺序表；（这里是因为快速排序是递归算法，为了能在输出排序结果的同时不干预其余排序而重新创建了另一个顺序表并使用引用符实现实参的改变）；

s为划分算法的起始位置；

t为划分算法的终点位置；

c为快速排序（划分）过程中关键字的比较次数；

cc为快速排序（划分）过程中关键字的移动次数。

**功能：**使用快速排序（递归，不断调用partition算法进行划分）对随机数序列进行排序，同时计算出排序过程中关键字的比较次数和移动次数。

（8）希尔排序ShellSort(SqList L,double &b,double &bb)

**参数：**L为结构体变量，即存储了待排序随机数的顺序表；

b为希尔排序过程中关键字的比较次数；

bb为希尔排序过程中关键字的移动次数。

**功能：**使用希尔排序对随机数序列进行排序，并将排序结果输出，同时计算出排序过程中关键字的比较次数和移动次数。

（9）筛选构建大根堆sift(SqList &D,int low,int high,double &d,double &dd)

**参数：**D为结构体变量，即存储了待排序随机数的顺序表（看成是一棵二叉树的顺序存储结构）；

low指向构建堆的过程中树的根节点；

high为构建出的堆中节点的最大编号；

d为构建堆的过程中关键字的比较次数；

dd为构建堆的过程中关键字的移动次数。

**功能：**第一次，使用所有数据构建大根堆，往后每经过一次排序即挑出最大值（取出根节点）后，利用剩余的节点继续构建大根堆。

（10）堆排序HeapSort(SqList D,double &d,double &dd)

**参数：**D为结构体变量，即存储了待排序随机数的顺序表；

d为堆排序过程中关键字的比较次数；

dd为堆排序过程中关键字的移动次数。

**功能：**使用堆排序（循环调用sift筛选构建大根堆，并交换首尾元素）对随机数序列进行排序，并将排序结果输出，同时计算出排序过程中关键字的比较次数和移动次数。

（11）产生随机数RanDom(int n,int N,int b[])

**参数：**n为随机数的个数；

N为随机数的取值范围（【0-N】）；

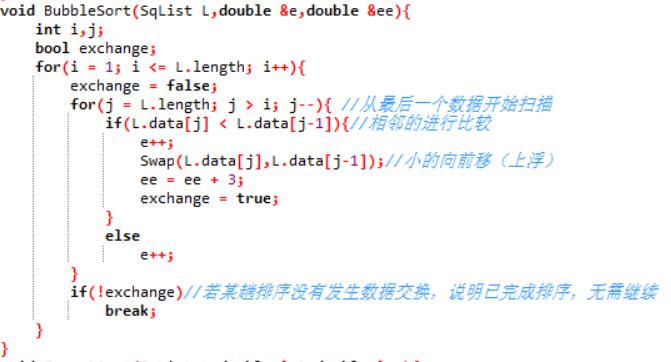
b数组用以存储产生的随机数。

**功能：**连续生成一系列随机数存储在数组b中，并依次输出产生的随机数。

1. 详细设计
2. 冒泡排序：

冒泡排序的思路是：每一轮排序时，从最后一个数据开始，相邻两个数据进行比较，将较小的那个往序列前方移动，以此到达一种将最小的元素“上浮”到序列首元素的效果。每轮排序后确定一个最小值，对剩余的序列继续排序。由于每一轮排序都能将当前序列中的最小值放到序列的开头，从而在多轮循环后达到排序效果。假如共有n个数据需要进行排序，最好情况（正序）下，算法的时间复杂度为O(n)；最坏情况（反序）下，时间复杂度为O(n²)。算法的平均时间复杂度为O(n²)。无额外空间，因此空间复杂度为O(1)。

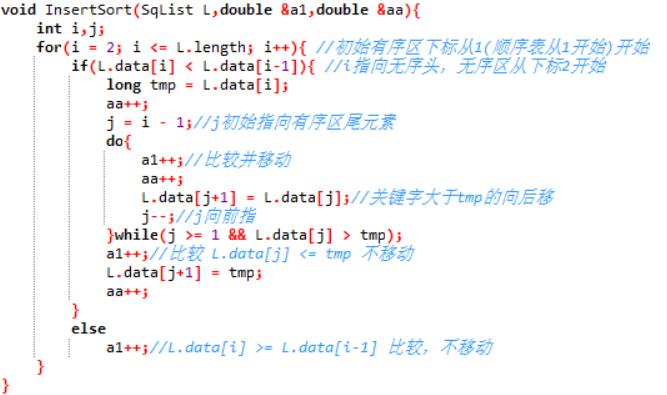
具体代码如下：



1. 直接插入排序：

直接插入排序的思路是：将整个序列分为有序区与无序区两个部分，第一轮循环即初始情况下有序区只含下标为0元素，用i指向无序区首元素，j指向有序区尾元素，若反序（R[i]<R[j]）则先将R[i]保存，再将j指向元素后移，然后j继续往前扫描，直至到头或不存在反序，将原R[i]放到合适位置。每轮排序完成后，有序区数据增加1，无序区数据减少1。假如共有n个数据需要进行排序，最好情况（正序）下，算法的时间复杂度为O(n)；最坏情况（反序）下，时间复杂度为O(n²)。算法的平均时间复杂度为O(n²)。无额外空间，空间复杂度为O(1)。

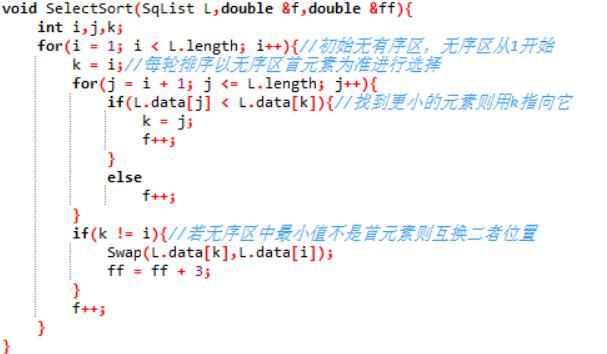
具体代码如下：



1. 选择排序：

选择排序的思路是：每次将无序区中的最小值选出，与序列第一个元素交换（放到序列的首位置），然后在剩下的数据中再次进行选择，交换。每轮排序完成后，有序区数据加一，无序区数据减一。在每次进行选择的时候，以当前序列首元素为准（用k指向），遇到更小的则移动k的位置指向更小值。多次循环后达到排序目的。假如共有n个数据需要进行排序，由于选择排序每一轮都需要从无序区中选出最小值，因此无论初始序列如何排列，所需进行比较的次数相同，平均时间复杂度为O(n²)。空间复杂度为O(1)。

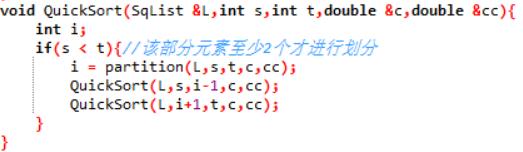
具体代码如下：



1. 快速排序：

快速排序的思路是：反复调用划分算法进行区域的划分，所以快速排序实际上就是不断划分的过程。具体的划分算法下面会提。

快速排序代码如下：



看着是个递归算法，不断在执行QuickSort，但实际上就是对划分算法的不断调用。因此快速的关键在于划分算法。

划分算法：在待排序的n个元素中选取一个元素（通常取第一个元素）作为基准，将其放在适当位置（比它小的放在前一部分，比它大的放在后一部分），于是序列被划分成了两部分。随后对划分出来的两部分分别重复以上步骤（递归），直到每部分只有一个元素或为空为止。

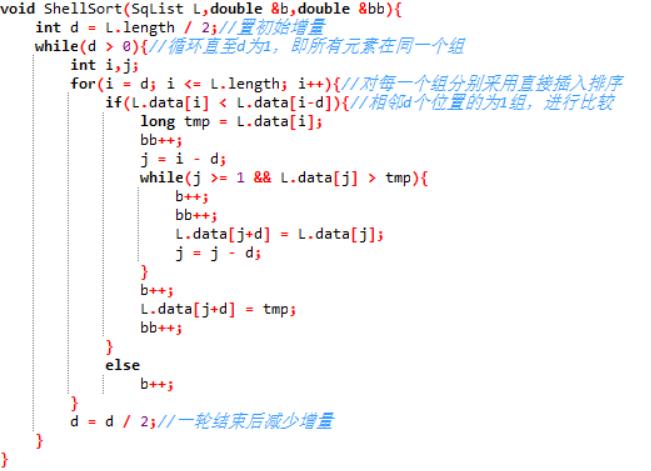


快速排序最好情况下，即每一次划分出的左右两区间长度大致相等，时间复杂度为O(nlog2n)，空间复杂度为O(log2n)；最坏情况，即每次划分选取的元素都是最小（或最大）值，导致左区间或右区间为空，时间复杂度为O(n^2)，空间复杂度为O(n)。平均时间复杂度为O(nlog2n)，空间复杂度为O(log2n)。

1. 希尔排序：

希尔排序是直接插入排序的改进，又称为减少增量的排序，基本思路如下：假如待排序的数据有n个，则初始增量d置为n/2，将整个序列划分成d个组，相邻d的元素放到同一个组中，对每一个组分别进行直接插入排序，然后减少增量（除以2），再次重复分组与排序。希尔排序的效率取决于所取增量d，一般认为其平均时间复杂度为O(n^1.3)。空间复杂度为O(1)。

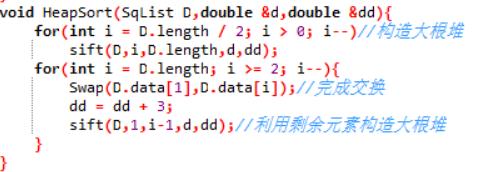
具体代码如下：



1. 堆排序：

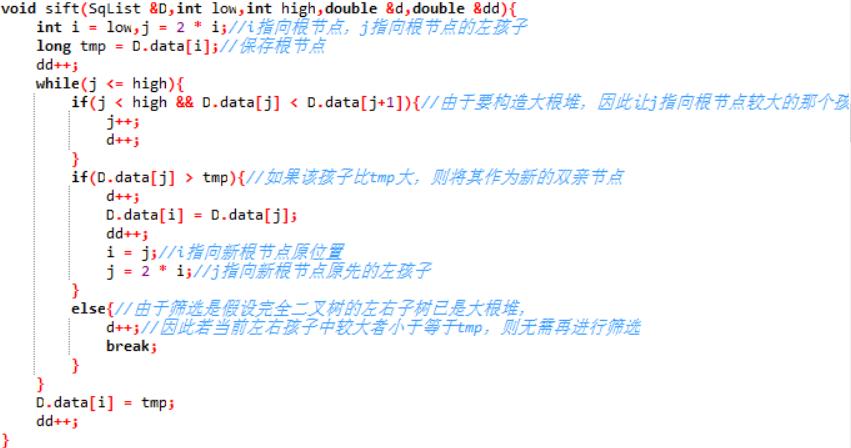
堆排序的排序过程与简单选择排序类似，只是挑选最大值、最小值的方法不同。简单选择排序每轮都需要进行两两比较来选出最大或最小值，而堆排序是利用二叉树的顺序存储结构采取构建大根堆或小根堆的形式，将最大值或最小值放在根节点即序列首位（下标为1）。因此进行选择时可直接得到最大值或最小值。

堆排序时，首先构造大根堆，选中首元素完成一轮交换，由于打乱了原有顺序，因此再将剩下的元素又一次构建大根堆，选中首元素完成交换……具体代码如下:



就像快速排序的关键是划分partition算法一样，堆排序的关键在于构建大根堆的筛选sift算法。筛选的主要流程为：将无序数组看成是二叉树的顺序存储结构，用i指向树的根节点，保存在tmp中，j指向根节点中较大的那个孩子，若该孩子比tmp大，则将其调整到双亲节点的位置，同时i,j指向分别指向该孩子节点的原位置和原左孩子，继续往下筛选。根据算法前提：进行筛选的时候假设完全二叉树的左右子树已是大根堆，因此若某轮循环中某节点的左右孩子中较大者小于等于tmp，则将tmp放到该根节点的位置，完成筛选。

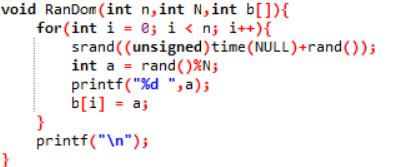
具体代码如下：



堆排序的时间主要由建立初始堆和反复重建堆两部分的时间构成。另外，堆排序和简单选择排序一样，无论初始序列如何，最好、最坏、平均时间复杂度都是O(nlog2n)。空间复杂度为O(1)。

1. 生成随机数：

代码如下:



在这里，使用srand()函数来播种，我们常会以time时间来作为参数，但由于for循环执行太快，不到1s就运行完成了，而time得到的时间只能精确到秒，因此1s内time根本没有改变。所以若只使用time作为参数，那么实际上并没有重新播种，产生也不是随机数，而是相同数字。因此我在参数内又加入了一个rand()。初始rand()会得到一个随机数，但此时是作为了srand的参数，就意味着进行了重新播种，然后在下面再使用rand()得到一个随机数作为待排序数据。循环，在srand()参数里又使用time(null)+rand()作为参数，虽然time(null)没变，但此时的rand()与第一次rand()产生的是不同的随机数，所以是完成了重新播种的，然后再使用rand()获取随机数作为待排序数据，再次循环，参数中rand()又与上次不同……

时间复杂度为O(n)，空间复杂度为O(1)。

1. 测试数据及结果

（1）测试数据：

有效数据：1000，10000，100000

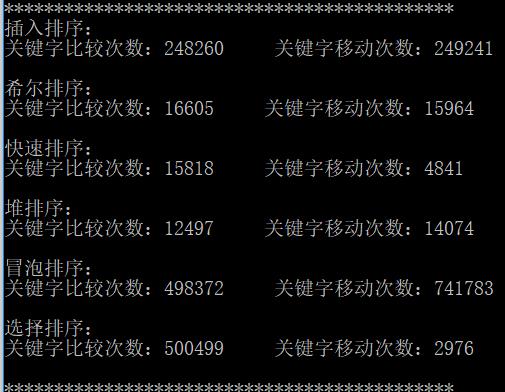
无效数据：1000000，0，-1，asdas

在此，通过随机数个数n分别为1000,10000,100000三种情况来分别对比6种排序过程中关键字的比较次数和移动次数取得直观数据，进而直接比较各种内部排序算法的时间复杂度。另外，该程序过滤了一些恶意的输入信息，只能输入限制范围内的数字。

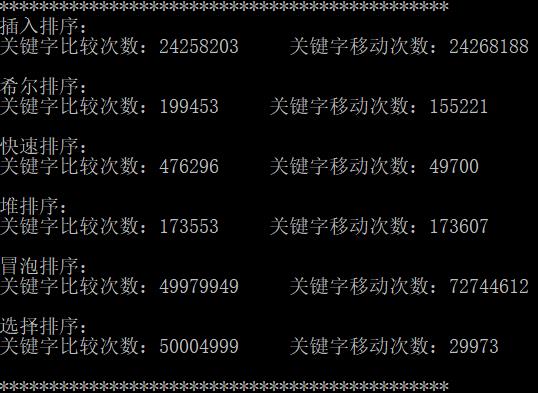
|  |  |  |
| --- | --- | --- |
| 输入 | 期望输出 | 测试结果 |
| 1000 | 6种排序过程中关键字的比较次数、移动次数 | 符合期望输出 |
| 10000 | 6种排序过程中关键字的比较次数、移动次数 | 符合期望输出 |
| 100000 | 6种排序过程中关键字的比较次数、移动次数 | 符合期望输出 |
| 1000000 | 提示错误，重新输入 | 符合期望输出 |
| 0 | 提示错误，重新输入 | 符合期望输出 |
| -1 | 提示错误，重新输入 | 符合期望输出 |
| asdas | 提示错误，重新输入 | 符合期望输出 |

具体情况见以下截图：

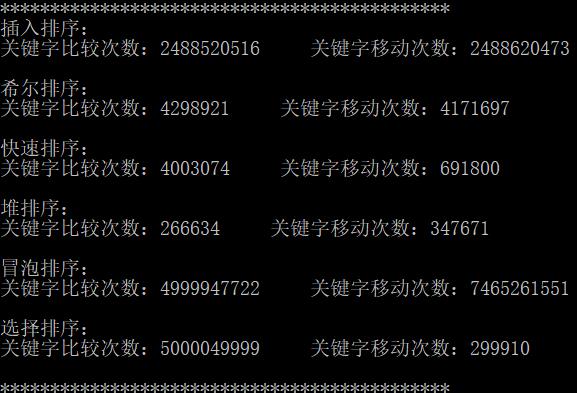
n为1000：



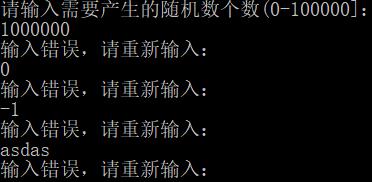
n为10000：



n为100000：



外加报错提示：



（2）测试结果：

根据程序运行结果可得：当待排序数据量不断增加时，插入排序和冒泡排序的关键字比较次数、移动次数，以及选择排序的关键字比较次数明显上升，大大超过了其余排序。

结合各排序算法的理论时间复杂度，冒泡排序为O(n^2)，直接插入排序为O(n^2)，选择排序为O(n^2)，快速排序为O(nlog2n)，希尔排序为O(n^1.3)，堆排序为O(nlog2n)。对照程序运行结果：冒泡排序、插入排序、选择排序关键字比较次数、移动次数相近且最高；希尔排序、快速排序、堆排序三者较低且不相上下。因此总体来说，程序运行得到的各内排序算法的关键字比较次数、移动次数所反映出来的排序速度与理论时间复杂度基本一致。

1. **调试分析及总结**
2. 调试过程中遇到的问题以及解决的方法描述。

主要问题有三：

一：初次完成程序的时候，我点击运行，跟着程序自带的提示，输入xxx怎么怎么样，输入xxx怎么怎么样，就这样走完了整个步骤。随机数的个数以及取值范围能够正确输入，同时每一种排序的排序结果，排序过程中关键字的比较次数、移动次数均能正确得到且输出，自认为已经不存在大问题了。但是随后我意识到，之所以我能得到正确的结果，是因为我是严格按照程序的提示或说是要求一步步往下走的，当然能得到正确结果，但如果我反其道而行之呢？当我在输入点不按照提示进行输入，而是胡乱输入一些东西，程序自然也就死了。于是我意识到该程序容错率太低。

解决：于是我开始对每一个输入点均进行输入内容的判断，只有当输入内容符合要求时才会进入下一步操作，由此解决了该问题。

二：无法进行100000个数据的排序。其实最初完成程序的时候是能进行100000个数据的排序的，但由于十万个数据还算挺多，当初没运行完成我就退出了。到程序最终完成开始写报告时，却怎么也不能进行100000个数据排序了，现在也没想明白为什么。

解决：未解决。

三：由于我设置了每次生成随机数后会输出随机数，然后在每一次排序结束后均会先输出排序结果（开始是为了检查排序的正确性，后来就不想删了），再输出关键字的比较次数和移动次数，由此虽然更直观，但也带来了一些问题。如果随机数较多，那么输出所有的随机数都需要一部分时间，后面进行排序，每次排序又要再输出一遍排序结果，显得界面很乱，内容很多。除此之外，为了将快速排序的结果输出，只能使用引用符&，但是又不能因为该算法影响其他的排序算法，因此我是直接新增了一个顺序表，空间开销增大。但是这个情况严格来说倒也不算是个大问题，所以也没有修改。

四：堆排序。由于堆排序是利用了二叉树的顺序存储结构特征，因此数据是从下标1开始存储，但是最开始建立的顺序表是从0开始存储，在所有算法均完善后我才意识到这个问题，为了简便，也是重新创建了一个顺序表从1开始存储，用于堆排序。虽然解决了堆排序，但空间开销仍增大了。

实际上问题三，如果不输出排序结果，就不存在这个问题，所以暂时不打算修改；对问题四，如果把原顺序表改为从1开始存储就不需要再新建表，但是随之需要更改其他很多内容，较为麻烦，因此暂也未作修改。

1. 收获体会。

体会一：

根据实际问题考虑数据的存储结构。以本题为例（相对简单）：要对一串数据进行排序，最简单最直接也是最先想到的就是应用顺序表，概念清楚，操作简单，并且能完成功能要求。

体会二：

也是主要体会。在进行程序设计的时候，不能仅从自己（设计者）的角度出发。就拿之前那个例子，我自己是设计者，我设计为输入‘1’进入下一步，输入‘0’结束程序，当我在使用这个程序的时候肯定会严格按照提示进行操作，因为我知道，按照提示做，就能得到正确的结果。但实际情况是，不是说只能输入‘1’或‘0’，而是只有输入‘1’或‘0’才能让程序继续或退出。但从用户的角度出发，往往是提示输入‘1’或‘0’，我就不输入‘1’或‘0’，我偏要输入别的，那怎么办？这也是在程序设计过程中需要考虑到的问题。总结起来：从用户角度出发，提升程序健壮性。