目录

[1 引言 2](#_Toc30327)

[2 数据生成及测试函数 3](#_Toc18728)

[21测试数据生成代码 3](#_Toc10678)

[2.1.1生成测试数据主函数代码 3](#_Toc30999)

[2.1.2生成测试数据实现过程代码 4](#_Toc1798)

[2.2测试参数的获取 5](#_Toc15218)

[3 算法优化 6](#_Toc19504)

[3.1 冒泡排序算法优化 6](#_Toc30181)

[3.2 选择排序算法优化 7](#_Toc29485)

[3.3快速排序算法优化 7](#_Toc8593)

[4 实验分析 8](#_Toc9057)

[4.1算法分析 9](#_Toc14317)

[4.1.1空间复杂度 9](#_Toc10624)

[4.1.2时间复杂度分析 9](#_Toc3986)

[4.1.3稳定性分析： 10](#_Toc5037)

[4.1.4适用类型分析： 13](#_Toc28759)

[4.2对不同数据构型排序时的表现 14](#_Toc15261)

[4.2.1.时间对比 14](#_Toc8558)

[4.2.2赋值，比较次数对比 18](#_Toc5825)

[5 C-字符串数组研究分析 26](#_Toc2408)

[5.1设计BubbleA、B函数前思考题的回答 27](#_Toc9407)

[5.2对于定义函数的分析 27](#_Toc30532)

[5.3对于BubbleA、B子函数即所传参数分析 28](#_Toc22217)

[6 小组分工及组员感想 29](#_Toc9454)

[6.1小组分工 29](#_Toc24208)

[6.2心得体会 29](#_Toc18150)

[6.3对课程的建议 31](#_Toc2531)

[7 结语： 31](#_Toc23007)

[8 致谢： 31](#_Toc14399)

[9 参考文献 31](#_Toc9879)

**排序算法实验研究**

叶思清，徐宏伟，刘远航，潘嘉宝

（计算机工程与科学学院）

摘要：为了更加直观的比较常见排序算法的效率及稳定性，我们针对冒泡、选择、快速三种排序分别进行优化，并且利用不同的数据类型以及数据构型，最终得到了排序算法的运行时间，比较，赋值次数，并且将其可视化处理。通过对图表的分析，最终得到结论。

关键词：排序，优化，算法评析，可视化。

Siqing Ye,Hongwei Xu,Yuanhang Liu,Jiabao Pan

（School of Computer Engineering and Science）

**Abstract:** In order to more intuitively compare the efficiency and stability of common sorting algorithms, we optimize the three kinds of bubbling, selection and fast sorting respectively, and use different data types and data configurations to finally get the running time, comparison and assignment times of the sorting algorithm, and visualize them. Through the analysis of the chart, the final conclusion is reached.

**Key words:** sorting, optimization, algorithm evaluation, visualization.

## 1 引言

排序，是指根据既定的要求，通过元素之间的比较，交换，移动，赋值，使元素重新排列。算法排序是算法基础中非常重要的部分，针对不同的数据构型，不同的算法随着数据量的增大，差异逐渐明显。因此，研究不同算法针对于不同数据构型进行排序时的效率，有助于帮助我们在将来排序时可以选择出最优的排序方法[1]。

我们生活在数据错综复杂的大数据时代，搜集有效数据的能力固然重要，但是对数据的处理能力也不容忽视。对数据处理的第一步应该是整理数据，而整理数据最直观的方法就是按照一定的标准进行数据排序[2]。

在本次实训中，我们对常见排序算法进行学习以及其效率的测定。这样的学习过程，对我们将来进行数据结构以及算法学习打下了坚实的基础，也加深了我们对算法的时间复杂度以及空间复杂度的理解。此外，我们还在理解算法的基础上对算法进行优化处理，并且结合原始算法进行了横向，纵向对比进行优化算法的测评。由于实训课程中涉及到许多陌生的知识点，我们通过查找相关资料以及文献进行学习，这也拓宽了我们的知识面，为接下来的学习奠定了良好的基础[2]。

为了更加直观的展现运行数据，我们小组通过将运行数据导入excel进行数据可视化处理。结合图表信息，对结果进行讨论，总结。

本文将会详细阐述本次实训中本小组的工作内容。

## 2 数据生成及测试函数

我们在理解代码的基础上，了解清楚了每个文件中子函数的具体作用，通过调用子函数完成了不同类型、规模测试数据的生成。然后通过加入记录比较次数，复制次数的语句以及记录排序算法运行时间的时间子函数统计出了赋值次数、比较次数和运行时间三个用来判断时间复杂度以及排序算法效率的参数。

### 2.1测试数据生成代码

生成具有不同规模（由1024倍增至65536），不同数据类型，数据构型（正态分布，均匀分布，完全正序，完全逆序，结构体）的测试数据。

我们通过调用已有的生成函数源代码，生成不同数据类型和构型的测试数据。

|  |
| --- |
| //在头文件部分将引用的子函数的文件引入  **#include "Sorts.h"**  **#include "Score.h"**  **#include "MyRand.h"** |

#### 2.1.1生成测试数据主函数代码

**生成正态分布数据**

|  |
| --- |
| ”MyRand.h”文件  **double** GaussRand(**double** mean, **double** variance);  // 正态分布(高斯分布) mean:均值、数学期望，variance方差 |

**生成均匀分布数据**

|  |
| --- |
| ”MyRand.h”文件  **double** UniformRand(**double** a, **double** b);  // 区间[a, b]上的均匀分布 |

**生成完全正序数据**

|  |
| --- |
| for(i=0; i<n; i++) data[i] = i; |

**生成完全逆序数据**

|  |
| --- |
| for(i=0; i<n; i++) data[i] = n-1-i; |

**生成结构体数据**

|  |
| --- |
| "Score.h"文件  typedef struct Score  {  **char** Id[9];  // 最多存放8个字符  **int** Chinese, Math, English, Physics, Chemistry, Total; // 5门课程成绩及总分  } Score; |

#### 2.1.2生成测试数据实现过程代码

**第一步：申请分配堆内存空间**

|  |
| --- |
| **void** I\_GetMemory(**int** \*\*data, **int** \*\*data0, **int** n)  // 分配堆内存空间，通过参数"返回"首地址  **void** D\_GetMemory(**double** \*\*data, **double** \*\*data0, **int** n)  // 分配堆内存空间，通过参数"返回"首地址  **void** GetMemory(**Score** \*\*data, **Score** \*\*data0, **int** n)  // 分配堆内存空间，通过参数"返回"首地址 |

**第二步：利用生成测试数据的主函数**

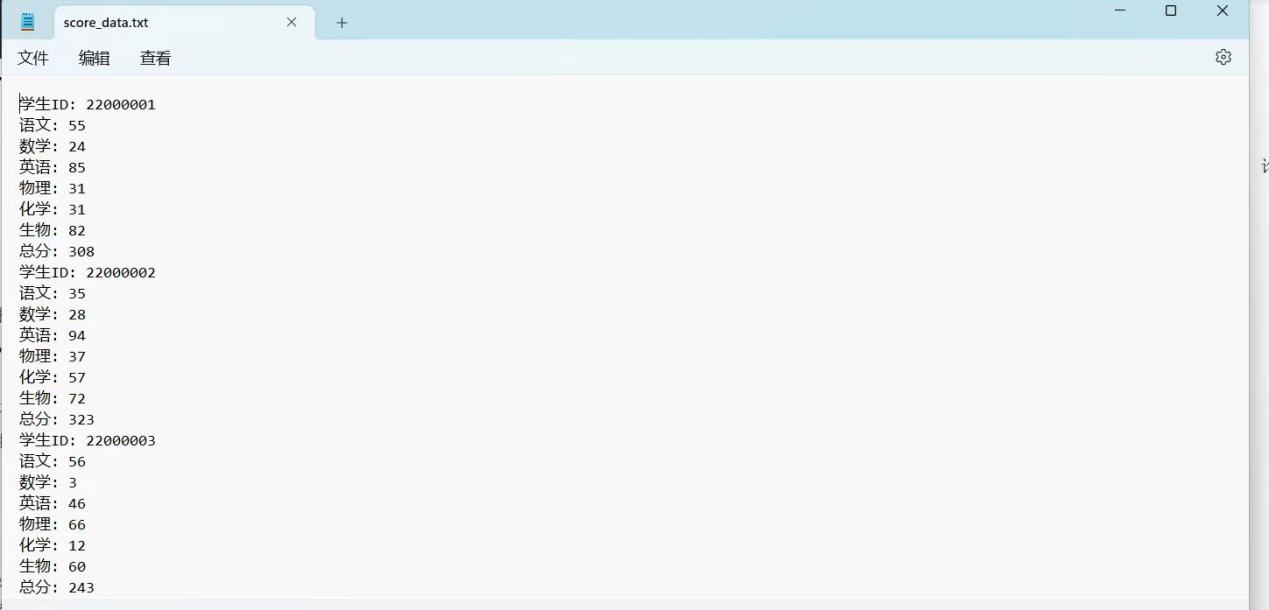
|  |
| --- |
| **void** I\_InitData(**int** \*data, **int** n, **int** flag)  **void** D\_InitData(**double** \*data, **int** n, **int** flag)  **void** InitScore(**struct Score** \*data, **int** n) |

具体测试数据的导出结果见下图1。



**图1.具体数据导出文件的证明**

具体测试数据的生成结果见下图2。



**图2.具体数据生成结果的证明**

### 2.2测试参数的获取

**获取测试参数的主函数**

【注意】int型，double型以及结构体型以int型为例来进行分析。

‘void I\_Test()’该子函数中的包含两个循环。第一重循环实现测试数据量倍增的工作。第二重循环语句对不同类型、规模的数据运用不同排序算法（冒泡，选择，快排）进行参数测试。

在每一次第一重循环中，先利用getMemory()函数来实现堆空间分配的申请，然后利用initData()函数来设置原始数据。

其次，利用第二重循环语句对不同类型、规模的数据运用不同排序算法（冒泡，选择，快排）进行参数测试。要排除数据量在65536，数据构型为完全顺序或者完全逆序时对应快速排序的这一特殊组合，因为递归层次太深，可能导致栈的溢出。

在每一次第二重循环开始时，要用ReSet函数对被上一次循环进行后重新排序的数据进行回复原始数据处理。在循环开始之前，先利用gettime(1)来设置计时起点。然后进入相应的算法排序。排序结束后，利用t[i] = gettime(0)来返回从计时起点到目前所经历的时间。之后再利用Check()函数来检验排序的正确性[1]。

在以上一系列操作结束后，再将我们测试到的时间参数返回，进而帮助我们去分析不同的原始算法，优化算法的时间复杂度。最后一步就是释放堆空间资源，并使指针为空。

该子函数所利用的函数如下：

|  |
| --- |
| I\_GetMemory(&data, &data0, n);// 申请分配堆空间  I\_InitData(data0, n, j);// 设置原始数据  I\_ReSet(data, data0, n);// 恢复原始数据  gettime()// 计时函数  I\_Check(data, n);// 检验排序的正确性  I\_FreeMemory(&data, &data0)；// 释放堆空间资源，并使指针为空 |

## 3 算法优化

在这一部分，我们基于最基础的三种排序算法（冒泡排序，选择排序，快速排序）进行了不同程度的优化，想要达到缩短运行时间的目的。

**3.1 冒泡排序算法优化**

我们组基于基础的冒泡排序进行了三次优化，首先我们设置一个标志位，记录上一次排序是否有交换，没有交换就可以提前结束了。

|  |
| --- |
| for (**int** i = 0; i < a.length - 1; i++) // 外层控制循环次数  {  if (!hasSwap)  break; // 上一轮没有交换，提前结束  } |

其次，我们设置了结束边界记录上一次最后交换的位置，作为下一次循环的结束边界。

最后一次比较说明在那之后的元素都已经排好序，无需再比较，可以避免一些无意义的比较。

|  |
| --- |
| **int** lastSwapIndex = a.length-1;  // 最后一次交换的位置，初始时设为“数组长度 – 1”  **int** end = lastSwapIndex;  // 上一次的最后交换位置作为这次循环的边界 |

最后我们使用双向冒泡排序，它的过程是：先从左往右比较一次，再从右往左比较一次，然后又从左往右比较一次，以此类推。它是为了优化前面的大部分元素都已经排好序的数组的排序，例如对于数组 [2,3,4,5,6,7,8,1]，如果使用普通的冒泡排序，需要比较七次；而换成双向冒泡排序，只需比较三次。

|  |
| --- |
| for (**int** i = left; i < right; i++) { // 保证 a[right] 是最大  for (**int** i = right; i > left; i--) { // 保证 a[left] 是最小 |

**3.2 选择排序算法优化**

基于基本的选择排序，在长度为N的无序数组中，第一次遍历n-1个数，找到最小的数值与第一个元素交换；第二次遍历n-2个数，找到最小的数值与第二个元素交换；直至第n-1次遍历，找到最小的数值与第n-1个元素交换，排序完成。

在优化中，我们使用两个变量，分别指向数组的头尾；每次遍历数组过程中，找到无序表中的最大值与最小值进行交换。主要代码如下：

|  |
| --- |
| //交换两个数据  **void** Swap(**int\*** a, **int\*** b) //选择排序  {  **int** max = begin;  **int** min = begin;  **int** i = 0;  //确定最大值下标以及最小值下标  for (i = begin+1; i <= end; i++)  {  if (arr[i] < arr[min])  {  min = i;  }  if (arr[i] > arr[max])  {  max = i;  }  }  //将最大值插到最后  Swap(&arr[begin], &arr[min]);  if (begin == max) //修正max  {  max = min;  }  Swap(&arr[end], &arr[max]); //将最小值插到最前面  } |

**3.3快速排序算法优化**

基于基本的快速排序算法思想，先选取一个基准元素(一般我们将待排序序列中的第一个元素选取为基准元素)，然后将其他元素与基准元素进行比较，比基准元素大的放到基准元素的右边，比基准元素小的放到基准元素的左边。(以基准元素为中心将元素重新分成两个序列，并返回基准元素的下标)，最后将新生成的两个序列继续执行前两步(此处可以用递归实现)。

由于快速排序算法在处理大规模有序数据时，容易由于递归次数过多，导致空间不够自动退出，程序无法继续运行，于是我们用希尔排序替换快速排序，不容易出现空间不够的情况，且速度也较快。代码如下：

|  |
| --- |
| **void** shell\_sort(**int\*** arr,**size\_t** len)  {  for(**int** k=len/2; k>0; k/=2)  {  for(**int** i=k,j; i<len; i++)  {  **int** tmp = arr[i];  for(j=i-k; j>=0 && tmp<arr[j]; j-=k)  {  arr[j+k] = arr[j];  }  arr[j+k] = tmp;  }  }  } |

原理大概是将一个待排序的数组分为若干个以Step为增量的若干个子序列，对这若干个子序列依次进行直接插入排序，在每一次排序完以后的Step呈递减趋势变化，当Step为1时，此时的序列已经基本有序，对整个序列进行直接插入排序。当增量为0时代表数组已经有序。

## 4 实验分析

我们针对不同数据构想和规模的测试数据，分别用三种基础排序算法和优化后的排序算法进行排序，并且测试了排序算法的运行时间以及在排序过程中赋值和比较的次数。然后，我们将运行结果数据导入excel表格进行可视化处理，以测试参数（运行时间，比较赋值次数）和数据类型为分类标准，展现了不同规模的数据在算法优化前后的对比柱状图。通过对运行结果和图表的分析，我们一起总结讨论不同算法的空间复杂度，时间复杂度，稳定性，适用类型等问题。

### 4.1算法分析

#### 4.1.1空间复杂度分析

空间复杂度指的是一个程序在执行时，所占有的临时内存空间大小：S(n)=O(f(n))；n是问题的规模，f(n)为语句关于n所占据的内存的函数。

冒泡排序的辅助变量空间仅仅是一个临时变量，并且不会随着排序规模的扩大而进行改变，所以空间复杂度为O(1)。

选择排序中始终只用到了数组大小的空间，为常数，因此空间复杂度为O(1)。

快速排序主要是由于递归造成的栈空间的使用，最好的情况下其树的深度为：log2(n)

空间复杂度为 O(logn)。而最坏的情况下:需要n-1次调用，每2个数都需要交换，此时退化为冒泡排序空间复杂度为 O(n)。

希尔排序的本质其实就是插入排序。插入排序可以看作是增量d=1的希尔排序，希尔排序可以看作是最后增量d=1的插入排序。二者的空间复杂度都为O（1）[3]。

#### 4.1.2时间复杂度分析

**（1）冒泡排序：**

外循环和内循环以及判断和交换元素的时间开销。

最优的情况也就是开始就已经排序好序了，那么就可以不用交换元素了，由于外层循环为n,内层所需要循环比较的次数为（n-1）（n-2）…1由等差数列求和得时间花销为：[ n(n-1) ] / 2；所以最优的情况时间复杂度为：O( n^2 )。

最差的情况也就是开始的时候元素是逆序的，那么每一次排序都要交换两个元素，则时间花销为：[ 3n(n-1) ] / 2；（其中比上面最优的情况所花的时间就是在于交换元素的三个步骤）；所以最差的情况下时间复杂度为：O( n^2 )；

**（2）选择排序：**

|  |
| --- |
| **void** Efferve()  {  **int** m[5] = { 12, 8, 6, 9, 10 };  **int** max = m[0];  for (**int** i = 0; i < 4; i++)  {  for (**int** j = i; j < 4; j++)  {  if (m[j] < m[j + 1])  {  max = m[j + 1];  m[j + 1] = m[j];  m[j] = max;  }  }  }  } |

当i=0下面循环4次，每次i+1下面循环都执行次4-i次，因此上述循环次数为T=（4-1)）+ （4 -2）+（4 - 3）+ 1；T=[4\*(4-1)]/2次。那当N个数进行排序时，将进行T=[N\*(N-1)]/2次，根据计算方法保留最高次N ^2,因此选择排序的时间复杂度为O(N ^2);

**（3）快速排序：**

由于快速排序用到了递归调用，因此计算其时间复杂度也需要用到递归算法计算

递归算法的时间复杂度公式：T[n] = aT[n/b] + f(n)。

最优情况下时间复杂度：快速排序最优的情况就是每一次取到的元素都刚好平分整个数组，此时的时间复杂度公式则为：T[n] = 2T[n/2] + f(n)；T[n/2]为平分后的子数组的时间复杂度，f[n] 为平分这个数组时所花的时间。

最差情况下时间复杂度：最差的情况就是每一次取到的元素就是数组中最小/最大的，这种情况其实就是冒泡排序了(每一次都排好一个元素的顺序)，此时的时间复杂度为:T[n] = n \* (n-1) = n^2 + n;

综上所述：快速排序最差的情况下时间复杂度为：O( n^2 )

快速排序的平均时间复杂度也是：O(nlogn)

**（4）希尔排序（本质就是多次插入排序）：**

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，每组包含的关键词越来越多，当增量减至1时，整个文件恰被分成一组，算法便终止。希尔排序的[时间复杂度](https://so.csdn.net/so/search?q=%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6&spm=1001.2101.3001.7020)是O(n^(1.3—2))[3]。

#### 4.1.3稳定性分析：

排序算法稳定性指的是:排序算法对于值相等的输入数据，在输出结果中它们的顺序也不变。举具体例子来说，对于一组数据[1,2,2,2],该组数据在进行排序处理之后，去验证三个2的相对位置是否有发生变化。如果没有发生变化，则说明算法稳定性好。反之，不好。

对于排序算法的稳定性分析，我们采用结构体型的测试数据，以总分作为排序依据，以学生的个人ID作为观测指标。具体来说，原本结构体原始测试数据是按照学号的升序进行排列，在使用排序算法后，测试数据将被重新排序。这时可以比较总分相同的学生的ID来进行算法稳定性的测试。如果总分相同的学生的排序依然是按照原始测试数据的排序方式（ID由小到大），那么就说明该排序算法的稳定性较好。

以下是针对具体排序算法进行的稳定性分析。

**（1）冒泡排序：**

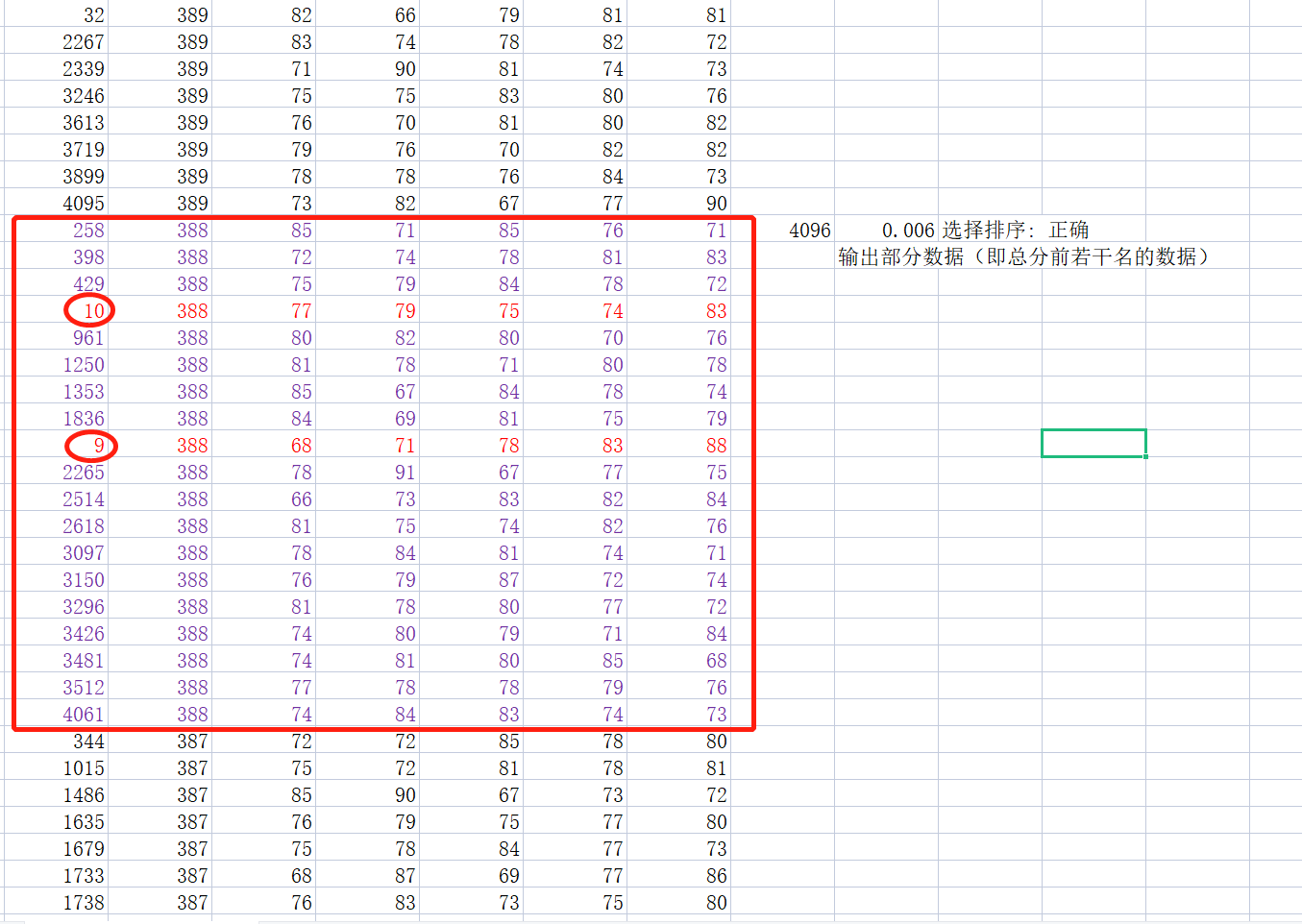
使用冒泡排序的过程中，大的元素会经由交换慢慢地“浮“到数列的顶端””。一次比较两个元素，如果它们的顺序错误就把它们交换过来。那么，即使有两个相同的数据处于相邻的位置，那么冒泡排序是这两个数据也就只是进行比较，但不会进行交换，因此两个相同数据的相对位置不会发生改变。

举个具体的例子来进行说明：假设有一个待排序的序列为2，2，1，4。我们利用冒泡排序将其从小打到进行排序。在第一次遍历过程中，前面两个2会进行比较，但不会交换相对位置，以此类推，在之后的每一次遍历中，如果两个2相邻，那么两个2都只是进行比较，不会交换相对位置，因此冒泡排序是稳定排序。

**（2）选择排序：**

使用选择排序，每次排序均基于前一次排序的结果并在未排序的部分查找最小值，如果发现最小值，则进行互换，否则，进入下一次排序操作。那么在查找到最小值并且进行交换的时候就有可能发生相同值的相对位置发生交换。

举个具体的例子来进行说明：假设有一个待排序的序列为2，3，2，1，4。在进行过第一次遍历之后，第一个2会和1进行位置交换，那么两个2的相对位置就会发生改变，因此选择排序不稳定。具体证明如下图3。



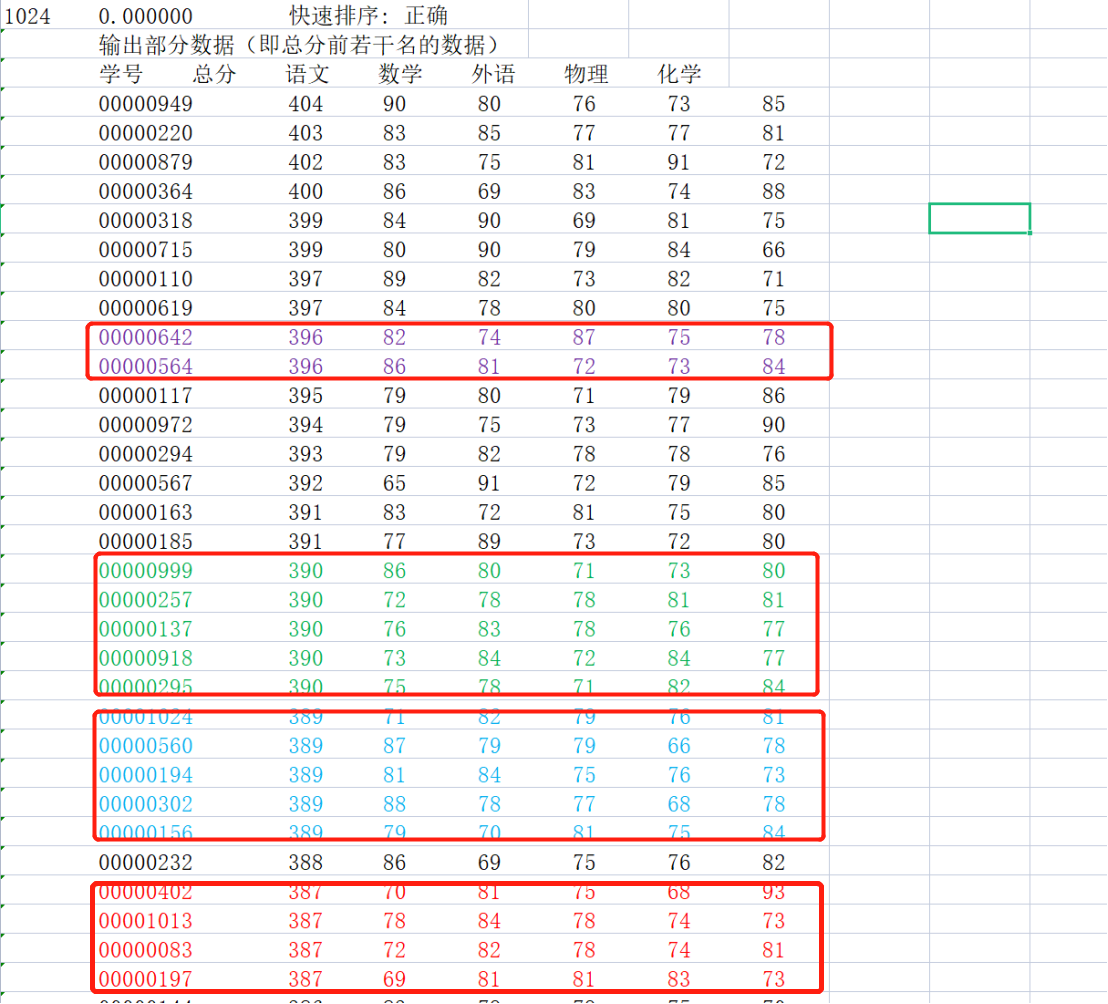
**图3.选择排序不稳定的证明**

**（3）快速排序**：

快速排序的过程中，会有两个地方导致算法排序不稳定，具体分析如下。

快速排序的步骤是选取基准数字，将基准数字放在列表头部（尾部）。然后开始遍历，如果数字比基准大，则位置不变，如果比基准小，则将该值与前面第一个比基准大的值交换位置（此处就会引起相同值的相对位置的变换，即排序不稳定），如果前面没有比基准大的数字就不用替换了。

完成一遍历后，交换头部的基准数字和最后一个比基准数字小的数字的位置，可以保证基准数字前面的值都小，后面的都大。在此交换过程中也会出现排序的不稳定。因此快速排序不稳定。具体证明见下图4。



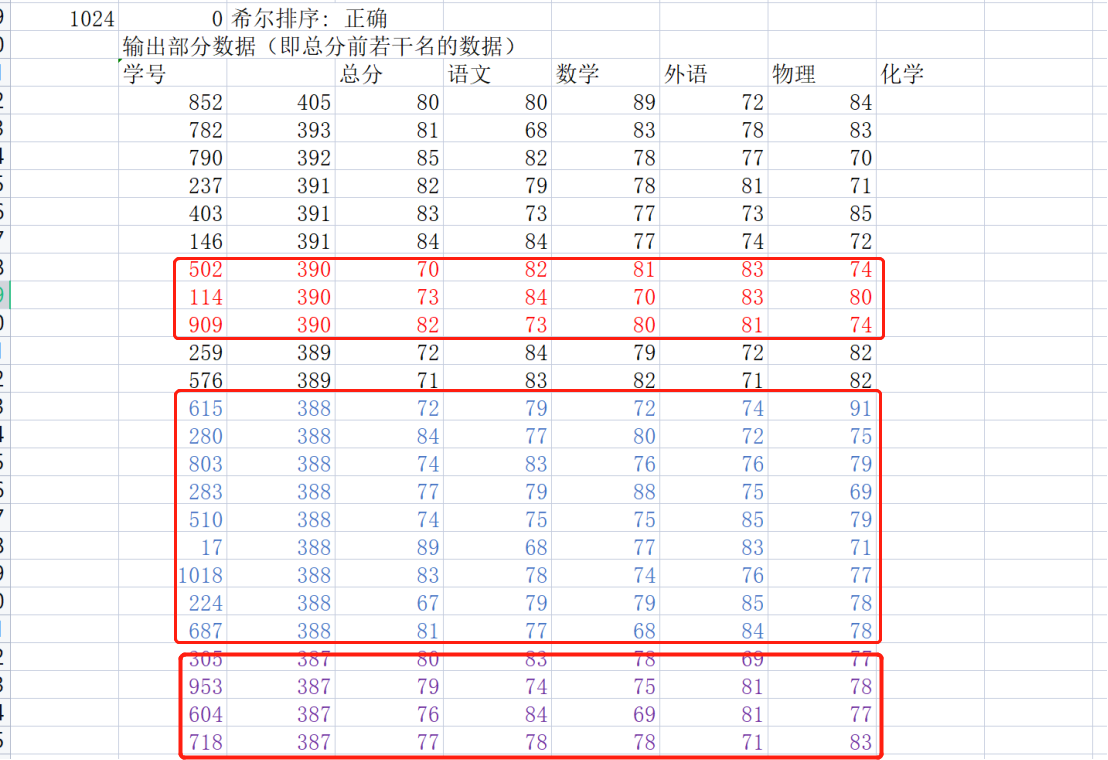
**图4.快速排序不稳定的证明**

在进行排序算法优化之前，各个学生按照学号升序排列。进行排序之后，我们观察总分相同的学生的学号排列来进行对比。如果总分相同，学号仍然是升序排列，那么就说明该排序算法稳定，反之不稳定。

根据图3，我们可以发现在排序之后，总分相同的学生不再以学号升序排列，因此我们可以判断快速排序是不稳定的。

**（4）希尔排序：**

希尔排序的本质是多次插入排序。我们已知一次[插入排序](https://www.baidu.com/s?wd=%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)是稳定的，不会改变相同元素的相对顺序，但在不同的[插入排序](https://www.baidu.com/s?wd=%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)过程中，相同的元素可能在各自的[插入排序](https://so.csdn.net/so/search?q=%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F&spm=1001.2101.3001.7020)[中移动](https://www.baidu.com/s?wd=%E4%B8%AD%E7%A7%BB%E5%8A%A8&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，最后其稳定性就会被打乱，因此[希尔排序](https://www.baidu.com/s?wd=shell%E6%8E%92%E5%BA%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)是不稳定的。具体证明见下图5.



**图5.希尔排序不稳定的证明**

在进行排序算法优化之前，各个学生按照学号升序排列。进行排序之后，我们观察总分相同的学生的学号排列来进行对比。如果总分相同，学号仍然是升序排列，那么就说明该排序算法稳定，反之不稳定。

根据图5，我们可以发现在排序之后，总分相同的学生不再以学号升序排列，因此我们可以判断快速排序是不稳定的。

#### 4.1.4适用类型分析：

选择排序方法时，主要需考虑以下因素：数据表的大小，即待排序数据元素的个数；数据构型；对排序的稳定性要求。

我们通过分析不同类型和规模的数据运行后花费的时间的结果数据进行具体分析，得出了在不同类型和规模下，应该使用哪一种排序算法。具体分析如下：

**（1）正态分布：**

当数据成正态分布时，我们发现：当数据量为1024时，六种算法运行时间均为0；当数据量为32768时，快速排序用时0.023秒，希尔排序用时0.015秒，而优化后的冒泡排序和选择排序分别用时2.079秒和0.535秒；当数据量为66536时，快速排序用时0.079秒，希尔排序用时0.016秒，而优化后的冒泡排序和选择排序分别用时8.555秒和2.21秒

因此，我们得出结论如下：如果数据量居于4096至65536区间，快速排序，希尔排序的时间明显短于冒泡排序和选择排序。如果数据量居于0至4096区间，由于排序时间太短，几种不同的排序算法之间运行时间的差异无法体现，因此几种排序算法均可以采用。

**（2）均匀分布：**

当数据成均匀分布时，我们发现：当数据量为1024时，六种算法运行时间均为0；当数据量为32768时，快速排序用时0.015秒，希尔排序用时0.015秒，而优化后的冒泡排序和选择排序分别用时2.043秒和0.519秒；当数据量为66536时，快速排序用时0.094秒，希尔排序用时0.007秒，而优化后的冒泡排序和选择排序分别用时7.875秒和2.061秒。

因此，我们得出结论如下：如果数据量居于2048至65536区间，快速排序，希尔排序的时间明显短于冒泡排序和选择排序。如果数据量居于0至1024区间，由于排序时间太短，几种不同的排序算法之间运行时间的差异无法体现，因此几种排序算法均可以采用。

针对以上四种排序算法，我们举个具体列子进行具体分析。当数据量为50时，适合使用冒泡和选择排序，但是当数据量为30000时，适合使用快速排序。当重复数据较多时，适合使用选择排序。如果数据本身已经部分有序或者完全有序，则适合插入排序。

### 4.2对不同数据构型排序时的表现

#### 4.2.1时间对比

**（1）整型数据**

**图6.整型数据（正态分布）时间比较**

分析：

1. 对于1024组数据量：因为数据量太小，所以时间太短，无法比较。
2. 对于32768组数据量：冒泡排序优化后，时间缩短了接近27%，选择排序优化后，时间缩短了接近24.5%，快速排序用希尔排序进行替换后，时间缩短了35%。
3. 对于65536组数据量：冒泡排序优化后，时间缩短了接近27%，选择排序优化后，时间缩短了接近23%，快速排序用希尔排序进行替换后，时间缩短了80%。

**图7.整型数据（均匀分布）时间比较**

分析：

1. 对于1024组数据量：因为数据量太小，所以时间太短，无法比较。
2. 对于32768组数据量：冒泡排序优化后，时间缩短了接近25%，选择排序优化后，时间缩短了接近17%，快速排序用希尔排序进行替换后，时间并没有减少。
3. 对于65536组数据量：冒泡排序优化后，时间缩短了接近61%，选择排序优化后，时间缩短了接近61%，快速排序用希尔排序进行替换后，时间缩短了93%。

**图8.整型数据（完全顺序）时间比较**

分析：

1. 对于1024组数据量：因为数据量太小，所以时间太短，无法比较。
2. 对于32768组数据量：冒泡排序优化后，时间缩短至0，选择排序优化后，时间缩短了接近57%，快速排序用希尔排序进行替换后，时间缩短至0。
3. 对于65536组数据量：冒泡排序优化后，时间缩短至0，选择排序优化后，时间缩短了接近57%，快速排序用希尔排序进行替换后，由于递归次数过多而终止。

**图9.整型数据（完全逆序）时间比较**

分析：

1. 对于1024组数据量：因为数据量太小，所以时间太短，无法比较。
2. 对于32768组数据量：冒泡排序优化后，时间增加了3%，选择排序优化后，时间缩短了接近13%，快速排序用希尔排序进行替换后，时间缩短趋近于0。
3. 对于65536组数据量：冒泡排序优化后，时间增加了1%，选择排序优化后，时间缩短了接近14%，

**(2)双精度浮点型数据**

双精度浮点型数据的分析结果同整型数据，以下只展现几张图例，结论不再赘述。

**图10.双精度浮点型数据（正态分布）时间比较**

**图11.双精度浮点型数据（均匀分布）时间比较**

**图12.双精度浮点型数据（完全顺序）时间比较**

**图13.双精度浮点型数据（完全逆序）时间比较**

**（3）结构体型**

**图14.结构体型数据时间比较**

分析：

1. 对于1024组数据量：因为数据量太小，所以时间太短，无法比较。
2. 对于32768组数据量：冒泡排序优化后，时间缩短了接近77%，选择排序优化后，时间缩短了接近99.5%，快速排序，时间缩短了99.2%。
3. 对于65536组数据量：冒泡排序优化后，时间缩短了接近80%，选择排序优化后，时间缩短了接近99.6%，快速排序用希尔排序进行替换后，时间缩短了99.6%。

#### 4.2.2赋值，比较次数对比

**（1）整型数据**

**图15.整型数据（正态分布）比较次数**

分析：

1. 对于1024组数据量：冒泡排序优化后，次数减少了接近34%，选择排序优化后，次数减少了接近50%，快速排序用希尔排序进行替换后，次数减少了99.7%。
2. 对于32768组数据量：冒泡排序优化后，次数减少了接近34.5%，选择排序优化后，次数减少了接近50%，快速排序用希尔排序进行替换后，次数增加了80.3%。
3. 对于65536组数据量：冒泡排序优化后，次数减少了接近35%，选择排序优化后，次数减少了接近50%，快速排序用希尔排序进行替换后，次数增加了107.4%。

**图16.整型数据（正态分布）赋值次数**

分析：

1. 对于1024组数据量：冒泡排序优化后，次数增加了3.4%，选择排序优化后，次数增加了90%，快速排序用希尔排序进行替换后，次数增加了537%。
2. 对于32768组数据量：冒泡排序优化后，次数增加了0.15%，选择排序优化后，次数增加了82%，快速排序用希尔排序进行替换后，次数增加了975%。
3. 对于65536组数据量：冒泡排序优化后，次数增加了0.14%，选择排序优化后，次数增加了82%，快速排序用希尔排序进行替换后，次数增加了1079%。

**图17.整型数据（均匀分布）比较次数**

分析：

1. 对于1024组数据量：冒泡排序优化后，次数减少了接近32.5%，选择排序，次数减少了接近50%，快速排序用希尔排序进行替换后，次数增加了27%。
2. 对于32768组数据量：冒泡排序优化后，次数减少了接近34%，选择排序，次数减少了接近50%，快速排序用希尔排序进行替换后，次数增加了178%。
3. 对于65536组数据量：冒泡排序优化后，次数减少了接近34%，选择排序，次数减少了接近50%，快速排序用希尔排序进行替换后，次数增加了212%。

**图18.整型数据（均匀分布）赋值次数**

分析：

1. 对于1024组数据量：冒泡排序优化后，次数增加了0.96%，选择排序，次数增加了103%，快速排序用希尔排序进行替换后，次数增加了580%。
2. 对于32768组数据量：冒泡排序优化后，次数减少了0.25%，选择排序，次数增加了104%，快速排序用希尔排序进行替换后，次数增加了1142%。
3. 对于65536组数据量：冒泡排序优化后，次数减少了0.11%，选择排序，次数增加了105%，快速排序用希尔排序进行替换后，次数增加了1259%。

**图19.整型数据（完全顺序）比较次数**

**图20.整型数据（完全顺序）赋值次数**

分析：

1. 针对于冒泡排序算法：优化前后比较次数明显减少，但是赋值次数都为零。
2. 针对于选择排序：优化前后比较次数明显减少，但是优化前赋值次数为零，优化后赋值次数明显增加。因此，可以得出选择排序在优化之后不再稳定。
3. 针对于快速排序：当数据量较小时，优化之后比较次数降为零，但是赋值次数明显增加。

**图21.整型数据（完全逆序）比较次数**

**图22.整型数据（完全逆序）比较次数**

分析：

1. 对于1024组数据量：冒泡排序优化后，次数没有发生变化，选择排序，次数增加了17100%，快速排序用希尔排序替代后，次数增加了1051%。
2. 对于32768组数据量：冒泡排序优化后，次数没有发生变化，选择排序，次数增加了546166%，快速排序用希尔排序替代后，次数增加了1675%。
3. 对于65536组数据：冒泡排序优化后，次数增加了93%，选择排序，次数增加了1092300%，

快速排序和希尔排序在本组中出现异常，故不再分析。

**（2）双精度浮点型数据**

双精度浮点型数据的分析结果同整型数据，以下只展现几张图例，结论不再赘述。

**图23.双精度浮点型数据（正态分布）比较次数**

**图24.双精度浮点型数据（正态分布）赋值次数**

**图25.双精度浮点型（均匀分布）比较次数**

**图26.双精度浮点型数据（均匀分布）赋值次数**

**图27.双精度浮点型数据（完全顺序**

**图28.双精度浮点型数据（完全顺序）赋值次数**

**图29.双精度浮点型数据（完全逆序）比较次数**

**图30.双精度浮点型数据（完全逆序）赋值次数**

## 5 C-字符串数组研究分析

在对C-字符串数组展开排序研究前，我们小组认为首先要对各种区域、数组、指针等概念有深入地了解，于是我们小组查找了各种区域、数组、指针等概念的定义与用途，作为基础知识储备，以便于理解sort.c文件中的代码，然后进行研究分析，最后补全BubbleA、B代码。

### 5.1设计BubbleA、B函数前思考题的回答

**（1）如何比较两个字符串的内容？**

答：两个字符串的内容可以通过使用 ‘strcmp’ 函数进行比较。 ‘strcmp’ 函数会返回一个整数值，用于表示两个字符串的比较结果。如果返回值为负数，则表示第一个字符串小于第二个字符串；如果返回值为正数，则表示第一个字符串大于第二个字符串；如果返回值为零，则表示两个字符串相等。

**（2）存储在什么区域的字符串能交换其内容？**

答：存储在可写的内存区域（如栈区、堆区）的字符串可以交换其内容。在这种情况下，可以使用字符串操作函数（如 ‘strcpy’ 或 ‘memcpy’）来交换字符串的内容。

**（3）若不能交换字符串的内容，排序操作中交换什么？**

答：如果存储的字符串内容不可交换（例如存储在只读的代码区），则在排序操作中可以交换指向字符串的指针。通过交换指针，可以改变字符串在数组中的位置，从而实现排序。

### 5.2对于定义函数的分析

|  |
| --- |
| **char** strA[][NUM] = { "enter", "number", "C/C++程序设计", "size", "指针", "begin", "of", "函数", "cat", "case", "program", "certain", "a", "cake", "side" };  **char**\* strB[] = { "enter", "number", "C/C++程序设计", "size", "指针", "begin", "of", "函数", "cat", "case", "program", "certain", "an", "cake", "side" }; |

上述定义 ‘strA’ 是一个二维字符数组，存储了多个C-字符串，它是在栈区分配的，每个C-字符串都是以null字符结尾，存储在全局数据区。‘strB’是一个指针数组，每个指针指向一个C-字符串常量。它也是在栈区分配的，指向的C字符串常量存储在代码区，是只读的。

|  |
| --- |
| **int** n1 = sizeof(strA) / sizeof(\*strA), n2 = sizeof(strB) / sizeof(\*strB); |

这边的n1表示为二维数组‘strA’里面含有多少串字符串；n2表示指针数组‘strB’中共有几个指针分别指向不同的字符串。

### 5.3对于BubbleA、B子函数即所传参数分析

对于BubbleA(strA, n1)所传参数strA即二维数组的首地址则由下方代码的str数组指针所接收，通过指针去访问和获取strA数组的内容，从而进行排序，但是不改变strA数组存储的数据。故接下来只需利用冒泡排序以及strcmp函数进行字符串排序，所补充BubbleA 完整代码如下：

|  |
| --- |
| **void** BubbleA(**char**(\*str)[NUM], **int** size)  {  for (**int** i = 0; i < size - 1; i++)  {  for (**int** j = 0; j < size - i - 1; j++)  {  if (**strcmp**(str[j], str[j + 1]) > 0)  {  **char** temp[NUM];  **strcpy**(temp, str[j]);  **strcpy**(str[j], str[j + 1]);  **strcpy**(str[j + 1], temp);  }  }  }  } |

对于BubbleB(strB, n2)所传参数strB即指针数组的首地址由下方代码同类型的指针数组str所接受，也是通过指针去访问和获取strB数组的内容，从而进行排序，但是不改变strB数组存储的数据。接下来的排序过程同BubbleA，在此不过多赘述，所补充BubbleB完整代码如下：

|  |
| --- |
| **void** BubbleB(**char\*** str[], **int** size)  {  for (**int** i = 0; i < size - 1; i++)  {  for (**int** j = 0; j < size - i - 1; j++)  {  if (**strcmp**(str[j], str[j + 1]) > 0)  {  **char\*** temp = str[j];  str[j] = str[j + 1];  str[j + 1] = temp;  }  }  }  } |

## 6 小组分工及组员感想

### 6.1小组分工

本小组全体成员每个阶段，如代码阅读理解，代码整合应用，排序算法优化，运行数据处理，论文构思和写作均有参与。但是各有侧重负责的板块。具体负责内容如下。

刘远航：生成数据优化（选择排序，快速排序），排序算法优化，计次计数代码整合，论文写作（算法优化部分）。

叶思清：生成测试数据，计次计数代码整合，记录时间代码，论文写作（摘要，引言，数据生成及测试函数部分，实验分析部分）。

徐宏伟：排序算法优化（冒泡排序），运行数据可视化处理，论文写作，论文格式调整。

潘嘉宝：字符串排序，论文写作（C-字符串数组研究分析部分）、格式调整。

### 6.2心得体会

刘远航：对于我这种只有大一秋季学期学了一点C语言的人来说，这次计算机暑期实训是非常有难度和挑战性的。艰难方显勇毅，磨砺始得玉成，这也是我选计算机得一部分原因吧。在项目进行的过程中，我们遇到了很多困难，想了很多办法，有过调试半天也运行不出来的抓耳挠腮，也有过看出问题关键后的茅塞顿开，走过一些弯路，出过许多错误，所幸我们并没有放弃，虽然我们四个计算机基础都很薄弱，但我们也尽了很大的努力，好几天都是从早上忙到晚上，谁说站在光里的才算英雄。在实训中，我也提高了各方面的能力，团队合作能力，搜索资源能力等等，也学到了一些知识，对三种排序方法有了更深的理解，也学到的是如何去分析一个算法并去优化它以及一些优化算法的基本思路，相当于对所学C语言的一种复习吧。路漫漫其修远兮，计科之路，从此始矣。

叶思清：这次夏季实训对于我来说是一次较大的挑战。课程开始之初，老师只给出了我们需要完成的任务，至于过程中那些全新的知识点需要我们自己去搜索学习。我认为这也是本次课程最具有挑战性的地方。在本次课程开始时，虽然老师已经提供了许多的文件资料以及源代码，但是当时我们并不知道每个文件以及源代码的具体作用以及应用方法，所以我们就尝试自己编写程序实现测试数据的生成。但是在之后的课程中，当老师讲述了工程的概念之后，我们才意识到可以将子函数的源代码文件全部放在同一个文件夹，然后直接调用源代码文件实现测试数据的生成。于是我们重新去阅读理解代码的含义，在搞清楚每个源代码的作用后进行直接调用，最终也生成了测试数据。虽然一开始浪费了两天的时间，但是这两天也让我们对生成不同构型测试数据的代码有了更为深刻的理解。同时，这次经验教训也让我们在之后的学习中，知道该如何高效应用已有的资料。我们应该做的第一步是通过读代码弄清楚代码的功能，然后直接进行调用。

潘嘉宝：回想起这一周多的经历，心中感慨万分。还记得刚看完作业要求和老师所给代码时的不知所措与茫然；还记得前两天在我们组组员的理解下一点一点取得进展的希望；也还记得第三天突然明白要把老师给的代码整合到一起运行的恍然大悟；更忘不了第四天在同一个地方，运用五台机器都跑不出想要的结果，一整天毫无进展的崩溃；然后接下来第五第六天，虽然程序可以跑通了，但是加入比较次数和赋值次数之后也还有大大小小的bug，一直不断地在调试，也是经常因为没有注意到一点点小细节而调试了好久好久，最后猛然发现的追悔莫及……万幸，最后代码都如愿能够跑下来。看着最后跑完结果的两个代码，一个732行，另一个943行，我感觉内心那块一直悬着的石头终于落下来了，也终于得以不再纠结于代码。但是发现大作业也还远远没有结束，后续的报告也还是需要我们投入很多时间。但经历了跑代码多次绝望之后，报告遇到的问题也都不算什么，小组几个成员一起合作了一两天就完成得差不多了。

大概完成任务后，真的想再次感慨这几天的辛苦不易，连续好几天从早上8点奋斗到晚上10点，除了中午短短休息半个小时，还有吃饭，其余的时间都投入到这次的大作业中。也特别庆幸有三个十分给力的队友，遇到困难一起解决，相互鼓励，我们本次的小组作业才得以完成，为大家的这几天的辛苦都点个赞。

最后，我想真诚地向这一周多给予我或者我们小组帮助过的人表示感谢，他/她们分别有：老师、撑起全组的大组长、莽村哥、还有一位想着深藏功与名的队友、还有帮助我突破栈上限问题的panda哥哥和panther哥哥，也还有我的小学同学fjp，黛玉学姐，还有与我们共同奋战的战友们——zyw,wy等等，以上感谢不分先后顺序，都是我真心想要感谢的人，因为有他们，我在迷茫与彷徨中逐渐找到方向，本次小组作业才得以顺利完成。再次对所有帮助过我的人给予我内心最诚挚的感谢！

徐宏伟：大抵是因为编程能力不足，开始没有懂老师给的资料，忙活了两天才发现可以做成工程文件，但也不算全无收获，至少复习了一下C语言知识。感谢队友的帮助，我可以一点点理解代码，领悟排序精髓。迄今为止，这是我遇见最给力的队友，我感觉我就是最摸鱼的那个。每天虽然要忙很久，虽然空闲时间很少，但是每天真的很充实，大概这就是劳动的快乐，不同于在虚拟空间中荒度时间，切实的劳动才能让我们有所收获。

### 6.3对课程的建议

刘远航：对很多没基础的同学来说太难了，会打击到一些心理素质不够强的同学的自信心，磨灭其对计算机的热情。老师可以适当讲解，缓解同学们压力，我觉得可以设计一些有趣的程序，增加课堂的有趣性。

叶思清：由于本门课程对于计算机基础知识的要求较强，我认为可以将本门课程安排在大二的夏季实训课程，这样在有一定知识储备的基础下，可以更加好的完成本门课程的相关任务。

潘嘉宝：建议老师在一开始帮助大家很好地解读好文件的要求，告诉大家大致要做的方向，不至于说很多同学在一开始都不知道要做什么，要怎么做。

徐宏伟：希望实训难度稍微降低一点，编程能力弱的同学才能够有更多的参与感。

## 7 结语：

通过本门课程的学习，加深了我们对三种基础排序算法的理解，对我们的能力有了极大的提升，也为之后的学习打下了坚实的基础。本次课程的学习强度大，时间紧凑，这锻炼了我们的团队合作能力，沟通协商能力，自主学习能力，短时间快速获取信息的能力，而这些能力是在将来的学习中不可或缺的重要能力。

## 8 致谢：

感谢任课老师的谆谆教诲，同时也感谢小组成员的辛勤付出。感谢老师给我们锻炼自身能力的机会，希望在将来的学习生活中能够迎难而上，自强不息。

## 9 参考文献

1. 杨绣丞,李彤,赵娜等.计算排序算法设计与分析[J].计算机应用研究,2014,31(03):658-662+695.
2. 刘珏玉.计算排序法及其应用[J].物探化探计算技术,1989(04):351-356.
3. 崔宁城,黄光南,李红星等.插入排序快速推进旅行时计算方法[J].石油物探,2020,59(06):851-862.
4. 高园. 新型排序问题的计算复杂性研究[D].郑州大学,2018.
5. 徐辉荣.排序计算方法初析[J].水利科技与经济,2008,No.83(05):359-360.