# 北京邮电大学软件学院

# 2023-2024学年第1学期实验报告

**课程名称： 数据结构**

**实验名称： 实验七 排序**

**实验完成人：**

**姓名：**\_\_\_\_禄禄鱼\_\_\_\_**学号：**\_\_\_\_\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**（封面 5%）**

**日 期： 2023 年 12 月 25 日**

1. **实验目的**

本实验的主要目的包括：

* + - 通过对常用内部排序算法的比较，深入了解各算法在不同输入数据情况下的关键字比较次数和关键字移动次数。
    - 运用随机数据生成程序，产生多组不同输入数据，以比较不同算法在不同数据集上的表现。
    - 学习并掌握常见的内部排序算法，包括直接插入排序、希尔排序、冒泡排序、快速排序、简单选择排序、堆排序和归并排序。

1. **实验内容**

（说明本实验的内容5%）

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；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。

1. **实验环境**

（说明本实验需要的环境5%）

Clion Windows10

1. **实验过程和实验结果**

1)

实验代码

//  
// Created by 86130 on 2023/12/26.  
//  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
// 交换数组中两个元素的值  
void swap(int \*a**,** int \*b) {  
 int temp = \*a**;** \*a = \*b**;** \*b = temp**;**}  
  
// 打印数组  
void printArray(int arr[]**,** int n) {  
 for (int i = 0**;** i < n**;** i++) {  
 printf("%d "**,** arr[i])**;** }  
 printf("\n")**;**}  
  
// 直接插入排序  
void insertionSort(int arr[]**,** int n**,** int \*compCount**,** int \*moveCount) {  
 for (int i = 1**;** i < n**;** i++) {  
 int key = arr[i]**;** int j = i - 1**;** (\*compCount)++**;** while (j >= 0 && arr[j] > key) {  
 arr[j + 1] = arr[j]**;** (\*moveCount) += 3**;** // 移动次数包括交换3次  
 j--**;** }  
 arr[j + 1] = key**;** }  
}  
  
// 希尔排序  
void shellSort(int arr[]**,** int n**,** int \*compCount**,** int \*moveCount) {  
 for (int gap = n / 2**;** gap > 0**;** gap /= 2) {  
 for (int i = gap**;** i < n**;** i++) {  
 int temp = arr[i]**;** int j = i**;** (\*compCount)++**;** while (j >= gap && arr[j - gap] > temp) {  
 arr[j] = arr[j - gap]**;** (\*moveCount)++**;** j -= gap**;** }  
 arr[j] = temp**;** (\*moveCount)++**;** }  
 }  
}  
  
// 冒泡排序  
void bubbleSort(int arr[]**,** int n**,** int \*compCount**,** int \*moveCount) {  
 for (int i = 0**;** i < n - 1**;** i++) {  
 for (int j = 0**;** j < n - i - 1**;** j++) {  
 (\*compCount)++**;** if (arr[j] > arr[j + 1]) {  
 swap(&arr[j]**,** &arr[j + 1])**;** (\*moveCount) += 3**;** // 移动次数包括交换3次  
 }  
 }  
 }  
}  
  
// 快速排序  
void quickSort(int arr[]**,** int low**,** int high**,** int \*compCount**,** int \*moveCount) {  
 if (low < high) {  
 int pivot = arr[high]**;** int i = low - 1**;** for (int j = low**;** j <= high - 1**;** j++) {  
 (\*compCount)++**;** if (arr[j] < pivot) {  
 i++**;** swap(&arr[i]**,** &arr[j])**;** (\*moveCount) += 3**;** }  
 }  
 swap(&arr[i + 1]**,** &arr[high])**;** (\*moveCount) += 3**;** int partitionIndex = i + 1**;** quickSort(arr**,** low**,** partitionIndex - 1**,** compCount**,** moveCount)**;** quickSort(arr**,** partitionIndex + 1**,** high**,** compCount**,** moveCount)**;** }  
}  
  
// 简单选择排序  
void selectionSort(int arr[]**,** int n**,** int \*compCount**,** int \*moveCount) {  
 for (int i = 0**;** i < n - 1**;** i++) {  
 int minIndex = i**;** for (int j = i + 1**;** j < n**;** j++) {  
 (\*compCount)++**;** if (arr[j] < arr[minIndex]) {  
 minIndex = j**;** }  
 }  
 swap(&arr[minIndex]**,** &arr[i])**;** (\*moveCount) += 3**;** // 移动次数包括交换3次  
 }  
}  
  
// 堆排序  
void heapify(int arr[]**,** int n**,** int i**,** int \*compCount**,** int \*moveCount) {  
 int largest = i**;** int left = 2 \* i + 1**;** int right = 2 \* i + 2**;** (\*compCount)++**;** if (left < n && arr[left] > arr[largest]) {  
 largest = left**;** }  
  
 (\*compCount)++**;** if (right < n && arr[right] > arr[largest]) {  
 largest = right**;** }  
  
 if (largest != i) {  
 swap(&arr[i]**,** &arr[largest])**;** (\*moveCount) += 3**;** heapify(arr**,** n**,** largest**,** compCount**,** moveCount)**;** }  
}  
  
void heapSort(int arr[]**,** int n**,** int \*compCount**,** int \*moveCount) {  
 for (int i = n / 2 - 1**;** i >= 0**;** i--) {  
 heapify(arr**,** n**,** i**,** compCount**,** moveCount)**;** }  
  
 for (int i = n - 1**;** i > 0**;** i--) {  
 swap(&arr[0]**,** &arr[i])**;** (\*moveCount) += 3**;** heapify(arr**,** i**,** 0**,** compCount**,** moveCount)**;** }  
}  
  
// 归并排序  
void merge(int arr[]**,** int l**,** int m**,** int r**,** int \*compCount**,** int \*moveCount) {  
 int n1 = m - l + 1**;** int n2 = r - m**;** int L[n1]**,** R[n2]**;** for (int i = 0**;** i < n1**;** i++) {  
 L[i] = arr[l + i]**;** }  
 for (int j = 0**;** j < n2**;** j++) {  
 R[j] = arr[m + 1 + j]**;** }  
  
 int i = 0**,** j = 0**,** k = l**;** while (i < n1 && j < n2) {  
 (\*compCount)++**;** if (L[i] <= R[j]) {  
 arr[k] = L[i]**;** (\*moveCount)++**;** i++**;** } else {  
 arr[k] = R[j]**;** (\*moveCount)++**;** j++**;** }  
 k++**;** }  
  
 while (i < n1) {  
 arr[k] = L[i]**;** (\*moveCount)++**;** i++**;** k++**;** }  
  
 while (j < n2) {  
 arr[k] = R[j]**;** (\*moveCount)++**;** j++**;** k++**;** }  
}  
  
void mergeSort(int arr[]**,** int l**,** int r**,** int \*compCount**,** int \*moveCount) {  
 if (l < r) {  
 int m = l + (r - l) / 2**;** mergeSort(arr**,** l**,** m**,** compCount**,** moveCount)**;** mergeSort(arr**,** m + 1**,** r**,** compCount**,** moveCount)**;** merge(arr**,** l**,** m**,** r**,** compCount**,** moveCount)**;** }  
}  
  
int main() {  
 srand(time(NULL))**;** int n = 100**;** // 表长  
 int dataSets = 5**;** // 数据组数  
  
 for (int i = 0**;** i < dataSets**;** i++) {  
 int arr[n]**;** for (int j = 0**;** j < n**;** j++) {  
 arr[j] = rand() % 1000**;** // 生成0到999的随机数  
 }  
  
 int compCount**,** moveCount**;** // 测试并比较每个排序算法  
 printf("数据集 %d:\n"**,** i + 1)**;** // 插入排序  
 compCount = 0**;** moveCount = 0**;** insertionSort(arr**,** n**,** &compCount**,** &moveCount)**;** printf("插入排序: 比较次数：%d，移动次数：%d\n"**,** compCount**,** moveCount)**;** // 希尔排序  
 compCount = 0**;** moveCount = 0**;** shellSort(arr**,** n**,** &compCount**,** &moveCount)**;** printf("希尔排序: 比较次数：%d，移动次数：%d\n"**,** compCount**,** moveCount)**;** // 冒泡排序  
 compCount = 0**;** moveCount = 0**;** bubbleSort(arr**,** n**,** &compCount**,** &moveCount)**;** printf("冒泡排序: 比较次数：%d，移动次数：%d\n"**,** compCount**,** moveCount)**;** // 快速排序  
 compCount = 0**;** moveCount = 0**;** quickSort(arr**,** 0**,** n - 1**,** &compCount**,** &moveCount)**;** printf("快速排序: 比较次数：%d，移动次数：%d\n"**,** compCount**,** moveCount)**;** // 选择排序  
 compCount = 0**;** moveCount = 0**;** selectionSort(arr**,** n**,** &compCount**,** &moveCount)**;** printf("选择排序: 比较次数：%d，移动次数：%d\n"**,** compCount**,** moveCount)**;** // 堆排序  
 compCount = 0**;** moveCount = 0**;** heapSort(arr**,** n**,** &compCount**,** &moveCount)**;** printf("堆排序: 比较次数：%d，移动次数：%d\n"**,** compCount**,** moveCount)**;** // 归并排序  
 compCount = 0**;** moveCount = 0**;** mergeSort(arr**,** 0**,** n - 1**,** &compCount**,** &moveCount)**;** printf("归并排序: 比较次数：%d，移动次数：%d\n"**,** compCount**,** moveCount)**;** printf("\n")**;** }  
  
 return 0**;**}

实验结果

数据集 1:

插入排序: 比较次数：99，移动次数：7374

希尔排序: 比较次数：503，移动次数：862

冒泡排序: 比较次数：4950，移动次数：7374

快速排序: 比较次数：568，移动次数：1128

选择排序: 比较次数：4950，移动次数：297

堆排序: 比较次数：1232，移动次数：1698

归并排序: 比较次数：533，移动次数：672

数据集 2:

插入排序: 比较次数：99，移动次数：7464

希尔排序: 比较次数：503，移动次数：908

冒泡排序: 比较次数：4950，移动次数：7464

快速排序: 比较次数：589，移动次数：1014

选择排序: 比较次数：4950，移动次数：297

堆排序: 比较次数：1260，移动次数：1740

归并排序: 比较次数：549，移动次数：672

数据集 3:

插入排序: 比较次数：99，移动次数：7590

希尔排序: 比较次数：503，移动次数：922

冒泡排序: 比较次数：4950，移动次数：7590

快速排序: 比较次数：650，移动次数：1158

选择排序: 比较次数：4950，移动次数：297

堆排序: 比较次数：1262，移动次数：1743

归并排序: 比较次数：551，移动次数：672

数据集 4:

插入排序: 比较次数：99，移动次数：6897

希尔排序: 比较次数：503，移动次数：990

冒泡排序: 比较次数：4950，移动次数：6897

快速排序: 比较次数：643，移动次数：1062

选择排序: 比较次数：4950，移动次数：297

堆排序: 比较次数：1282，移动次数：1773

归并排序: 比较次数：540，移动次数：672

数据集 5:

插入排序: 比较次数：99，移动次数：6708

希尔排序: 比较次数：503，移动次数：865

冒泡排序: 比较次数：4950，移动次数：6708

快速排序: 比较次数：597，移动次数：1143

选择排序: 比较次数：4950，移动次数：297

堆排序: 比较次数：1280，移动次数：1770

归并排序: 比较次数：551，移动次数：672

2)

实验代码

1. #include <stdio.h>  
   #include <stdlib.h>  
     
   // 学生结构体  
   struct Student {  
    int id**;** char name[50]**;** int scores[5]**;** // 假设有5门课程  
    int totalScore**;** int rank**;**}**;**// 快速排序实现  
   void quickSort(struct Student \*student**,** int low**,** int high) {  
    if (low < high) {  
    int pivot = student[high].totalScore**;** int i = low - 1**;** for (int j = low**;** j <= high - 1**;** j++) {  
    if (student[j].totalScore > pivot) {  
    i++**;** struct Student temp = student[i]**;** student[i] = student[j]**;** student[j] = temp**;** }  
    }  
     
    struct Student temp = student[i + 1]**;** student[i + 1] = student[high]**;** student[high] = temp**;** int partitionIndex = i + 1**;** quickSort(student**,** low**,** partitionIndex - 1)**;** quickSort(student**,** partitionIndex + 1**,** high)**;** }  
   }  
     
   // 计算总分  
   void calculateTotalScore(struct Student \*student**,** int numStudents**,** int numCourses) {  
    for (int i = 0**;** i < numStudents**;** i++) {  
    student[i].totalScore = 0**;** for (int j = 0**;** j < numCourses**;** j++) {  
    student[i].totalScore += student[i].scores[j]**;** }  
    }  
   }  
     
   // 打印名次表和学生详细信息  
   void printRankAndDetails(struct Student \*student**,** int numStudents**,** int numCourses) {  
    printf("名次表:\n")**;** printf("%-5s%-10s%-10s%-10s%-10s%-10s%-10s%-10s%-10s\n"**,** "名次"**,** "学号"**,** "姓名"**,** "总分"**,** "高数"**,** "JavaSE"**,** "计组"**,** "形式语言"**,** "数据结构")**;** for (int i = 0**;** i < numStudents**;** i++) {  
    printf("%-5d%-10d%-10s%-10d"**,** student[i].rank**,** student[i].id**,** student[i].name**,** student[i].totalScore)**;** for (int j = 0**;** j < numCourses**;** j++) {  
    printf("%-10d"**,** student[i].scores[j])**;** }  
    printf("\n")**;** }  
   }  
     
   int main() {  
    int numStudents = 100**;** // 学生人数  
    int numCourses = 5**;** // 课程数量  
     
    struct Student students[numStudents]**;** // 模拟生成学生数据  
    for (int i = 0**;** i < numStudents**;** i++) {  
    students[i].id = 1001 + i**;** sprintf(students[i].name**,** "Student%d"**,** i + 1)**;** for (int j = 0**;** j < numCourses**;** j++) {  
    students[i].scores[j] = rand() % 51 + 50**;** // 假设成绩在50~100之间  
    }  
    }  
     
    // 计算总分  
    calculateTotalScore(students**,** numStudents**,** numCourses)**;** // 快速排序  
    quickSort(students**,** 0**,** numStudents - 1)**;** // 设置名次  
    for (int i = 0**;** i < numStudents**;** i++) {  
    if (i == 0 || students[i].totalScore < students[i - 1].totalScore) {  
    students[i].rank = i + 1**;** } else {  
    students[i].rank = students[i - 1].rank**;** }  
    }  
     
    // 打印结果  
    printRankAndDetails(students**,** numStudents**,** numCourses)**;** return 0**;**}

实验结果

| 名次 | 学号 | 姓名 | 总分 | 高数 | JavaSE | 计组 | 形式语言 | 数据结构 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1023 | Student23 | 449 | 73 | 82 | 97 | 99 | 98 |
| 2 | 1003 | Student3 | 427 | 94 | 94 | 75 | 98 | 66 |
| 3 | 1070 | Student70 | 420 | 97 | 97 | 85 | 79 | 62 |
| 4 | 1094 | Student94 | 418 | 99 | 96 | 83 | 51 | 89 |
| 5 | 1088 | Student88 | 417 | 67 | 91 | 99 | 81 | 79 |
| 5 | 1015 | Student15 | 417 | 58 | 89 | 79 | 98 | 93 |
| 7 | 1065 | Student65 | 416 | 57 | 96 | 77 | 96 | 90 |
| 8 | 1062 | Student62 | 413 | 99 | 67 | 75 | 96 | 76 |
| 9 | 1010 | Student10 | 412 | 73 | 95 | 60 | 100 | 84 |
| 9 | 1066 | Student66 | 412 | 91 | 83 | 66 | 82 | 90 |
| 11 | 1009 | Student9 | 410 | 91 | 95 | 58 | 89 | 77 |
| 12 | 1051 | Student51 | 407 | 85 | 83 | 97 | 62 | 80 |
| 12 | 1006 | Student6 | 407 | 90 | 80 | 100 | 82 | 55 |
| 14 | 1050 | Student50 | 404 | 70 | 77 | 90 | 81 | 86 |
| 15 | 1030 | Student30 | 401 | 85 | 78 | 67 | 97 | 74 |
| 16 | 1032 | Student32 | 400 | 97 | 83 | 52 | 95 | 73 |
| 16 | 1086 | Student86 | 400 | 76 | 76 | 90 | 85 | 73 |
| 16 | 1053 | Student53 | 400 | 84 | 98 | 77 | 89 | 52 |
| 19 | 1075 | Student75 | 399 | 92 | 82 | 52 | 95 | 78 |
| 20 | 1033 | Student33 | 398 | 69 | 85 | 83 | 85 | 76 |
| 20 | 1056 | Student56 | 398 | 52 | 100 | 61 | 92 | 93 |
| 20 | 1016 | Student16 | 398 | 100 | 62 | 92 | 65 | 79 |
| 23 | 1095 | Student95 | 394 | 84 | 64 | 90 | 67 | 89 |
| 24 | 1027 | Student27 | 393 | 64 | 78 | 59 | 99 | 93 |
| 25 | 1080 | Student80 | 392 | 70 | 91 | 93 | 71 | 67 |
| 26 | 1042 | Student42 | 391 | 97 | 96 | 88 | 51 | 59 |
| 26 | 1043 | Student43 | 391 | 52 | 93 | 80 | 70 | 96 |
| 28 | 1054 | Student54 | 390 | 71 | 70 | 86 | 75 | 88 |
| 28 | 1072 | Student72 | 390 | 63 | 92 | 76 | 90 | 69 |
| 28 | 1004 | Student4 | 390 | 82 | 87 | 58 | 83 | 80 |
| 31 | 1055 | Student55 | 389 | 88 | 68 | 96 | 61 | 76 |
| 32 | 1031 | Student31 | 388 | 97 | 57 | 67 | 75 | 92 |
| 33 | 1040 | Student40 | 386 | 74 | 80 | 88 | 66 | 78 |
| 33 | 1082 | Student82 | 386 | 61 | 92 | 64 | 91 | 78 |
| 33 | 1087 | Student87 | 386 | 58 | 96 | 61 | 97 | 74 |
| 33 | 1069 | Student69 | 386 | 79 | 78 | 74 | 60 | 95 |
| 37 | 1085 | Student85 | 385 | 96 | 95 | 62 | 52 | 80 |
| 37 | 1017 | Student17 | 385 | 95 | 76 | 62 | 63 | 89 |
| 39 | 1097 | Student97 | 382 | 78 | 51 | 86 | 69 | 98 |
| 40 | 1092 | Student92 | 381 | 87 | 81 | 76 | 87 | 50 |
| 40 | 1011 | Student11 | 381 | 97 | 52 | 72 | 65 | 95 |
| 40 | 1001 | Student1 | 381 | 91 | 55 | 60 | 81 | 94 |
| 43 | 1071 | Student71 | 380 | 66 | 75 | 83 | 84 | 72 |
| 43 | 1076 | Student76 | 380 | 50 | 87 | 73 | 76 | 94 |
| 43 | 1074 | Student74 | 380 | 85 | 77 | 71 | 65 | 82 |
| 43 | 1028 | Student28 | 380 | 63 | 98 | 100 | 54 | 65 |
| 47 | 1093 | Student93 | 379 | 84 | 85 | 99 | 56 | 55 |
| 47 | 1021 | Student21 | 379 | 89 | 55 | 99 | 77 | 59 |
| 49 | 1046 | Student46 | 375 | 56 | 95 | 89 | 77 | 58 |
| 49 | 1089 | Student89 | 375 | 75 | 55 | 76 | 76 | 93 |
| 51 | 1044 | Student44 | 373 | 61 | 83 | 84 | 83 | 62 |
| 51 | 1035 | Student35 | 373 | 90 | 82 | 91 | 54 | 56 |
| 53 | 1024 | Student24 | 372 | 65 | 52 | 68 | 98 | 89 |
| 54 | 1064 | Student64 | 371 | 80 | 60 | 87 | 60 | 84 |
| 54 | 1002 | Student2 | 371 | 66 | 53 | 83 | 84 | 85 |
| 56 | 1052 | Student52 | 370 | 52 | 96 | 91 | 55 | 76 |
| 57 | 1007 | Student7 | 368 | 91 | 50 | 82 | 62 | 83 |
| 58 | 1041 | Student41 | 366 | 79 | 67 | 64 | 59 | 97 |
| 58 | 1025 | Student25 | 366 | 66 | 80 | 82 | 71 | 67 |
| 60 | 1100 | Student100 | 363 | 64 | 64 | 51 | 96 | 88 |
| 60 | 1081 | Student81 | 363 | 75 | 88 | 79 | 66 | 55 |
| 62 | 1098 | Student98 | 361 | 91 | 74 | 75 | 69 | 52 |
| 62 | 1090 | Student90 | 361 | 54 | 88 | 76 | 93 | 50 |
| 62 | 1026 | Student26 | 361 | 82 | 92 | 69 | 52 | 66 |
| 65 | 1018 | Student18 | 360 | 58 | 69 | 82 | 58 | 93 |
| 66 | 1061 | Student61 | 357 | 85 | 65 | 72 | 80 | 55 |
| 66 | 1020 | Student20 | 357 | 66 | 52 | 92 | 80 | 67 |
| 68 | 1063 | Student63 | 356 | 54 | 56 | 93 | 80 | 73 |
| 69 | 1059 | Student59 | 355 | 57 | 60 | 56 | 92 | 90 |
| 70 | 1057 | Student57 | 354 | 61 | 50 | 89 | 98 | 56 |
| 71 | 1038 | Student38 | 350 | 91 | 65 | 64 | 72 | 58 |
| 71 | 1022 | Student22 | 350 | 86 | 53 | 61 | 52 | 98 |
| 71 | 1073 | Student73 | 350 | 77 | 100 | 54 | 61 | 58 |
| 74 | 1047 | Student47 | 346 | 58 | 82 | 55 | 91 | 60 |
| 75 | 1060 | Student60 | 345 | 76 | 88 | 62 | 67 | 52 |
| 75 | 1013 | Student13 | 345 | 75 | 76 | 55 | 56 | 83 |
| 77 | 1036 | Student36 | 344 | 50 | 62 | 95 | 83 | 54 |
| 78 | 1099 | Student99 | 343 | 75 | 61 | 67 | 69 | 71 |
| 78 | 1083 | Student83 | 343 | 58 | 69 | 79 | 74 | 63 |
| 80 | 1029 | Student29 | 342 | 54 | 97 | 59 | 80 | 52 |
| 80 | 1091 | Student91 | 342 | 70 | 84 | 55 | 69 | 64 |
| 82 | 1045 | Student45 | 340 | 75 | 56 | 90 | 67 | 52 |
| 82 | 1078 | Student78 | 340 | 54 | 58 | 78 | 82 | 68 |
| 84 | 1019 | Student19 | 339 | 62 | 53 | 94 | 51 | 79 |
| 85 | 1084 | Student84 | 337 | 87 | 57 | 85 | 55 | 53 |
| 85 | 1049 | Student49 | 337 | 70 | 55 | 84 | 76 | 52 |
| 85 | 1014 | Student14 | 337 | 66 | 63 | 67 | 87 | 54 |
| 85 | 1005 | Student5 | 337 | 56 | 68 | 76 | 50 | 87 |
| 85 | 1096 | Student96 | 337 | 85 | 57 | 62 | 74 | 59 |
| 90 | 1079 | Student79 | 336 | 59 | 67 | 95 | 64 | 51 |
| 90 | 1068 | Student68 | 336 | 72 | 52 | 66 | 73 | 73 |
| 92 | 1077 | Student77 | 335 | 69 | 61 | 56 | 97 | 52 |
| 93 | 1048 | Student48 | 331 | 75 | 71 | 59 | 75 | 51 |
| 94 | 1034 | Student34 | 327 | 56 | 54 | 62 | 82 | 73 |
| 95 | 1058 | Student58 | 325 | 51 | 54 | 78 | 58 | 84 |
| 96 | 1037 | Student37 | 323 | 69 | 92 | 52 | 60 | 50 |
| 97 | 1039 | Student39 | 322 | 55 | 51 | 65 | 93 | 58 |
| 98 | 1008 | Student8 | 321 | 72 | 64 | 84 | 51 | 50 |
| 98 | 1067 | Student67 | 321 | 58 | 63 | 61 | 80 | 59 |
| 100 | 1012 | Student12 | 312 | 57 | 52 | 50 | 90 | 63 |

（问题分析，设计方案、算法、设计图、程序、调试过程截图、运行结果截图等70%）

1. **实验心得**

（10%）

通过本次实验，我深入了解了各种内部排序算法的实现原理和性能特点。通过比较关键字比较次数和关键字移动次数，我对算法的效率有了更直观的认识。同时，通过统计学生成绩的实验，我掌握了对数据进行排序和统计的方法，这对今后在实际工作中处理大量数据将会有很大帮助。整体而言，本次实验拓展了我的算法和数据处理能力。

窗体顶端