

程序设计报告

（2018/19 学年 第二学期）

题 目：**看谁排得快**

|  |  |
| --- | --- |
| **专 业** | **计算机科学与技术** |
| **组长 学号姓名** | **B18030620 张承扬** |
| **组员 学号姓名** | **B18030618 杨楚瀛** |
|  | **B18030621 张琪** |
| **指 导 教 师** | **吴敏** |
| **指 导 单 位** | **计算机学院软件教学中心** |
| **日 期** | **2019.3.29** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **成员分工** | **组长（张承扬）** | **主函数流程、从键盘读入源数据、从文件读入源数据、TimeData基本类、输出报表和图表绘制、文字提示、程序图标** | | | | | |
| **组员（杨楚瀛）** | **冒泡排序、插入排序、二分排序、快速排序、双路快排、SHELL排序、从外部TXT读取报表数据** | | | | | |
| **组员（张琪）** | **随机数生成器、双向冒泡排序、选择排序、快速排序、三路快排、从外部TXT读取报表数据** | | | | | |
| **评分细则** | **评分项** | **优秀** | | **良好** | **中等** | | **差** |
| **遵守机房规章制度** |  | |  |  | |  |
| **上机表现** |  | |  |  | |  |
| **学习态度** |  | |  |  | |  |
| **程序准备情况** |  | |  |  | |  |
| **程序设计能力** |  | |  |  | |  |
| **团队协作** |  | |  |  | |  |
| **课题功能实现** |  | |  |  | |  |
| **算法设计合理性** |  | |  |  | |  |
| **用户界面设计** |  | |  |  | |  |
| **报告内容质量** |  | |  |  | |  |
| **报告格式规范** |  | |  |  | |  |
| **回答问题准确性** |  | |  |  | |  |
| **简短评语** | **教师：**  **年 月 日** | | | | | | |
| **评分等级** | **B18030620 张承扬** | | **B18030618 杨楚瀛** | | | **B18030621 张琪** | |
|  | |  | | |  | |
| **备注** | **评分等级共五种：优秀、良好、中等、及格、不及格** | | | | | | |

**看谁排的快**

**一、课题内容和要求**

看谁排的快是一个与排序相关的课题，无论是考试成绩、工资收入、房价、股票涨幅等各种数据都需要进行排序的工作。对于不同的原始数据集，用不同的方法得到有序序列的速度是不一样的。要求对于用户提供的原始数据，或由系统产生的随机数据集进行排序，用尽可能多的方法实现，分析排序所需要的时间与初始序列之间的关系，从而得出一个结论性的建议，怎样的序列用怎样的方法可以排得最快。

**二、需求分析**

看谁排的快的功能框架图如图1所示。

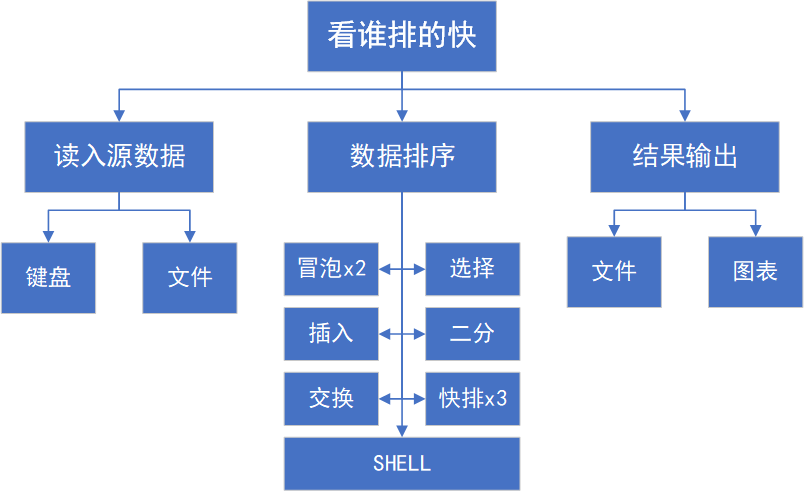


图1 功能框架图

（1）读入源数据：提供直接从键盘获取和从文件读取两种方法，从文件读取分为从程序目录直接读取和指定文件读取。

（2）数据排序：对获取的源数据通过不同排序方式排序，统计实际排序次数与时间通过对象返回；建立临时数组以避免破坏源数据顺序。

（3）结果输出：输出到文件，并绘制图表以便于观察结果。

**三、概要设计**

1 主要存储结构

//TimeData.h 头文件

#ifndef \_TIMEDATA

#define \_TIMEDATA

//TimeData类，用于存储所需统计数据

#include <iostream>

#include <string>

using namespace std;

class TimeData

{

string \_way; //排序方法

double \_time; //排序时间

int \_tangShu; //排序趟数

public:

TimeData(); //无参构造函数

TimeData(string way, double time, int tangShu); //带参构造函数

TimeData(const TimeData &timeData); //拷贝构造函数

string GetWay(); //获取方法名

double GetTime(); //获取运行时间

int GetTangShu(); //获取排序趟数

void SetWay(string way); //设置方法名

void SetTime(double time); //设置运行时间

void SetTangShu(int tangShu); //设置排序趟数

};

#endif // !\_TIMEDATA

//TimeData.cpp 源文件

#include "TimeData.h"

TimeData::TimeData() //无参构造函数

{

\_way = "";

\_time = 0;

\_tangShu = 0;

}

TimeData::TimeData(string way, double time, int tangShu) //带参构造函数

{

\_way=way;

\_time = time;

\_tangShu = tangShu;

}

TimeData::TimeData(const TimeData &timeData) //拷贝构造函数

{

\_way=timeData.\_way;

\_time = timeData.\_time;

\_tangShu = timeData.\_tangShu;

}

string TimeData::GetWay() //获取方法名

{

return \_way;

}

double TimeData::GetTime() //获取运行时间

{

return \_time;

}

int TimeData::GetTangShu() //获取排序趟数

{

return \_tangShu;

}

void TimeData::SetWay(string way) //设置方法名

{

\_way = way;

}

void TimeData::SetTime(double time) //设置运行时间

{

\_time = time;

}

void TimeData::SetTangShu(int tangShu) //设置排序趟数

{

\_tangShu = tangShu;

}

2 主要函数流程

（1）输出程序主标题。

（2）程序运行选择：可选择从读入源数据开始从头排序、分析、输出和图表绘制或者利用上一次运行的数据结果直接进行输出和图表绘制，如图2所示。



图2 运行选择图

（3）读入方法选择：不同方法对应不同函数进行读入，如图3所示。



图3 读入方法选择图

（4）进行排序：利用不同排序方法进行排序，并对趟数和时间进行统计，存储于TimeData类的对应成员变量，返回存有数据的对象，如图4所示。



图4 进行排序图

（5）输出文件：输出数据文件，由不同排序方法产生的对象生成对象数组，并以一定格式存储于指定文件，如图5所示。

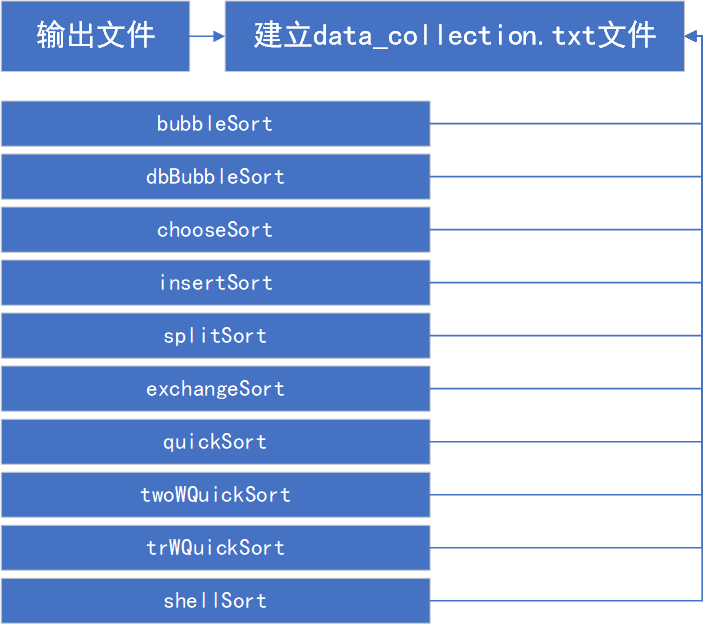


图5 输出文件图

（6）绘制图表：利用特殊符号和已经获取的数据绘制直观的图表，如图6所示。

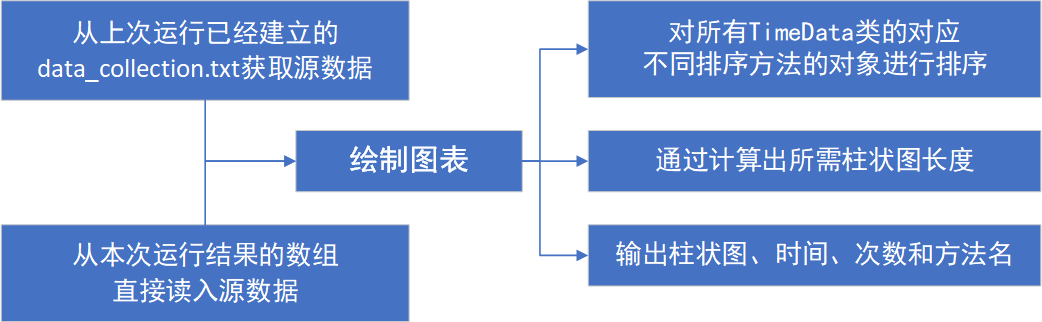


图6 绘制图表图

**四、源程序代码**

//main.cpp 主函数

#include <iostream>

#include <algorithm>

#include <fstream>

#include <ctime>

#include <stdlib.h> //文件处理

#include "ReaderFromKeyboard.h"

#include "ReaderFromFiles.h"

#include "TimeData.h"

#include "DataCollection.h"

#include "GraphDrawer.h"

#include "ShuChu.h" //排序方法

#include "MaoPao.h"

#include "ChooseSort.h"

#include "ChaRu.h"

#include "ErFeng.h"

#include "FastSort.h"

#include "ShuangXiangKuaiPai.h"

#include "Shell.h"

#include "EXChangeSort.h"

#include "TwoWayBubbleSort.h"

#include "ThreeRoadFastSort.h"

using namespace std;

int SIZE\_OF\_ARRAY = 0; //全局数组大小

int main()

{

system("mode con cols=130 lines=40");

system("color 70");

system("title=看谁排得快（香港记者）");

//标题

{

cout << "======================================================" << endl;

cout << "================看谁排得快（香港记者）================" << endl;

cout << "======================================================" << endl;

}

RRRRRESET: //数据读入方法选择

cout << "======================================================" << endl;

cout << "1. 读入源数据进行排序和分析" << endl;

cout << "2. 从data\_collection.txt读入排序结果直接进行图表绘制" << endl;

cout << "======================================================" << endl;

cout << "请选择要执行的程序：";

int chooser = 0;

RESETMARK00: //选择错误重置位置

cin >> chooser;

getchar();

if (chooser != 1 && chooser != 2)

{

cout << "======================================================" << endl;

cout << "选择错误，请重新输入：";

goto RESETMARK00;

}

else if (chooser == 1) //读入源数据进行排序和分析

{

goto PROCESS1;

}

else if (chooser == 2) //从文件读入排序结果

{

FILEERRORA:

if (ShuChu())

{

cout << "======================================================" << endl;

cout << "读取文件失败或文件不存在，回车重试...";

getchar();

goto FILEERRORA;

}

cout << "======================================================" << endl;

cout << "程序已结束，是否退出程序？(y/n)";

char choosere;

RESETMARK00000: //数据错误重置位置

cin >> choosere;

if (choosere != 'y' && choosere != 'n' && choosere != 'Y' && choosere != 'N')

{

cout << "======================================================" << endl;

cout << "选择错误，请重新输入：";

goto RESETMARK00000;

}

else if (choosere == 'y' || choosere == 'Y') //退出程序

{

return 0;

}

else if (choosere == 'n' || choosere == 'N') //RESET

{

goto RRRRRESET;

}

getchar();

return 0;

}

PROCESS1: //数据读入方法选择

cout << "======================================================" << endl;

cout << "1. 从键盘直接输入源数据" << endl;

cout << "2. 从文件读入源数据" << endl;

cout << "3. 随机生成1万个double类型源数据" << endl;

cout << "======================================================" << endl;

cout << "请选择读入源数据的方法：";

chooser = 0;

double \*arr = NULL;

RESETMARK0: //数据错误重置位置

cin >> chooser;

if (chooser != 1 && chooser != 2 && chooser != 3)

{

cout << "======================================================" << endl;

cout << "选择错误，请重新输入：";

goto RESETMARK0;

}

else if (chooser == 1) //从键盘直接输入源数据

{

arr = ReadFromKeyboard();

}

else if (chooser == 2) //从文件读入源数据

{

arr = ReadFromFiles();

}

else if (chooser == 3) //随机生成源数据

{

int i, k = 0;

int arrPre[10000]; //整数数组，保存生成的随机数

arr = new double[10000]; //浮点数数组，保存结果

srand((unsigned)time(NULL));

while (k != 10000) { //生成整数

arrPre[k] = rand(); //注意随机整数的范围

++k;

}

for (i = 0; i != k; ++i) { //输出

arr[i] = arrPre[i] / 100.0;

}

SIZE\_OF\_ARRAY = 10000;

getchar();

} //排序

cout << "======================================================" << endl;

cout << "源数据导入完成，回车进行排序...";

getchar();

double \*arrTemp = new double[SIZE\_OF\_ARRAY];

cout << "======================================================" << endl;

cout << "1. 冒泡排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData bubbleSort = MaoPao(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << bubbleSort.GetTime() << "μs " << "排序趟数" << bubbleSort.GetTangShu() << "次" << endl;

cout << "2. 双向冒泡排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData dbBubbleSort = TwoWayBubbleSort(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << dbBubbleSort.GetTime() << "μs " << "排序趟数" << dbBubbleSort.GetTangShu() << "次" << endl;

cout << "3. 选择排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData chooseSort = ChooseSort(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << chooseSort.GetTime() << "μs " << "排序趟数" << chooseSort.GetTangShu() << "次" << endl;

cout << "4. 插入排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData insertSort = ChaRu(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << insertSort.GetTime() << "μs " << "排序趟数" << insertSort.GetTangShu() << "次" << endl;

cout << "5. 二分排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData splitSort = ErFeng(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << splitSort.GetTime() << "μs " << "排序趟数" << splitSort.GetTangShu() << "次" << endl;

cout << "6. 交换排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData exchangeSort = ChangeSort(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << exchangeSort.GetTime() << "μs " << "排序趟数" << exchangeSort.GetTangShu() << "次" << endl;

cout << "7. 快速排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData quickSort = FastSort(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << quickSort.GetTime() << "μs " << "排序趟数" << quickSort.GetTangShu() << "次" << endl;

cout << "8. 双路快速排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData twoWQuickSort = quicksortT(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << twoWQuickSort.GetTime() << "μs " << "排序趟数" << twoWQuickSort.GetTangShu() << "次" << endl;

cout << "9. 三路快速排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData trWQuickSort = ThreeRoadFastSort(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << trWQuickSort.GetTime() << "μs " << "排序趟数" << trWQuickSort.GetTangShu() << "次" << endl;

cout << "10. SHELL排序中... ";

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

arrTemp[i] = arr[i];

}

TimeData shellSort = Shell(arrTemp, SIZE\_OF\_ARRAY);

cout << "用时" << shellSort.GetTime() << "μs " << "排序趟数" << shellSort.GetTangShu() << "次" << endl;

FILEERROR: //文件处理

cout << "======================================================" << endl;

cout << "排序完成，回车输出至文件...";

getchar();

TimeData \*dataArray = new TimeData[10]; //建立结果数组

dataArray[0] = bubbleSort;

dataArray[1] = dbBubbleSort;

dataArray[2] = chooseSort;

dataArray[3] = insertSort;

dataArray[4] = splitSort;

dataArray[5] = exchangeSort;

dataArray[6] = quickSort;

dataArray[7] = twoWQuickSort;

dataArray[8] = trWQuickSort;

dataArray[9] = shellSort;

if (DataCollection(dataArray)) //如果输出失败

{

cout << "======================================================" << endl;

cout << "文件生成失败，回车尝试重新生成...";

goto FILEERROR;

}; //对对象数组进行排序分析

int i, j;

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

{

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

{

if (dataArray[j].GetTime() > dataArray[j + 1].GetTime())

{

swap(dataArray[j], dataArray[j + 1]);

}

}

}

DRAWERROR: //图表绘制重置点

cout << "======================================================" << endl;

cout << "文件成功生成于程序目录，回车输出处理后数据并查看图表...";

getchar();

if (GraphDrawer(dataArray)) //如果绘图失败

{

cout << "======================================================" << endl;

cout << "图表绘制失败，回车尝试重新生成...";

goto DRAWERROR;

}; //以追加方式打开文件输出

fstream opener;

opener.open("data\_collection.txt", ios::app);

opener << "=====以上为未排序前源数据=====" << endl;

opener << "=====以下为排序分析完成后数据=====" << endl << endl;

for (int i = 0; i < 10; i++)

{

opener << dataArray[i].GetWay() << " " << dataArray[i].GetTime() << " " << dataArray[i].GetTangShu() << endl;

}

cout << "======================================================" << endl;

cout << "程序已结束，是否退出程序？(y/n)";

char choosere;

RESETMARK000: //数据错误重置位置

cin >> choosere;

if (choosere != 'y' && choosere != 'n' && choosere != 'Y' && choosere != 'N')

{

cout << "======================================================" << endl;

cout << "选择错误，请重新输入：";

goto RESETMARK000;

}

else if (choosere == 'y' || choosere == 'Y') //退出程序

{

return 0;

}

else if (choosere == 'n' || choosere == 'N') //RESET

{

goto RRRRRESET;

}

getchar();

return 0;

}

//ReaderFromFiles.h 从文件读取源数据

double\* ReadFromFiles();

//ReaderFromFiles.cpp 从文件读取源数据

#include <iostream>

#include <vector>

using namespace std;

extern int SIZE\_OF\_ARRAY; //全局数组大小

double\* ReadFromFiles()

{

cout << "======================================================" << endl;

cout << "======================从文件读入======================" << endl;

cout << "======================================================" << endl;

FILE \*fp = NULL;

RESETMARK2: //文件错误重置位置

cout << "1. 从程序同目录读入\"array.txt\"源数据" << endl;

cout << "2. 读入指定文件" << endl;

cout << "======================================================" << endl;

cout << "请选择读入文件的方法：";

int chooserTMP = 0;

RESETMARKFILE: //选择错误重置位置

cin >> chooserTMP;

if (chooserTMP != 1 && chooserTMP != 2)

{

getchar();

cout << "======================================================" << endl;

cout << "选择错误，请重新输入：";

goto RESETMARKFILE;

}

else if (chooserTMP == 1) //从程序同目录赌读入"array.txt"

{

cout << "======================================================" << endl;

cout << "即将从程序同目录下读入\"array.txt\"，按回车确认...";

getchar();

getchar();

fp = fopen("array.txt", "r");

}

else if (chooserTMP == 2) //读入指定文件

{

cout << "======================================================" << endl;

cout << "请输入所需读取文件的完整目录：" << endl;

getchar();

char arraTemp[100];

gets\_s(arraTemp, 100);

fp = fopen(arraTemp, "r");

}

if (!fp)

{

cout << "======================================================" << endl;

cout << "打开文件失败，请重新选择" << endl;

cout << "======================================================" << endl;

goto RESETMARK2;

}

std::vector<double> \*arrayTemp = new vector<double>; //使用动态大小的vector容器实现从文件中读取不定数量的数据

while (!feof(fp)) {

double tmp = 0;

fscanf(fp, "%lf", &tmp);

arrayTemp->push\_back(tmp);

}

SIZE\_OF\_ARRAY = arrayTemp->size();

double \*backArray = new double[SIZE\_OF\_ARRAY];

for (int i = 0; i < SIZE\_OF\_ARRAY; i++)

{

backArray[i] = arrayTemp->at(i);

}

fclose(fp); delete arrayTemp;

return backArray;

}

//ReaderFromKeyboard.h 从键盘读入源数据

double\* ReadFromKeyboard();

//ReaderFromKeyboard.cpp 从键盘读入源数据

#include <iostream>

using namespace std;

extern int SIZE\_OF\_ARRAY; //全局数组大小

double\* ReadFromKeyboard()

{

cout << "======================================================" << endl;

cout << "======================从键盘读入======================" << endl;

cout << "======================================================" << endl;

cout << "请输入需要排序的数据个数，回车确认：";

RESETMARK1: //数据错误重置位置

int arrayNum = 0; //数据个数

cin >> arrayNum;

if (arrayNum == 0)

{

cout << "======================================================" << endl;

cout << "数据个数为0，不需要排序，请重新输入需要排序的数据个数：";

goto RESETMARK1;

}

else if (arrayNum == 1)

{

cout << "======================================================" << endl;

cout << "数据个数为1，不需要排序，请重新输入需要排序的数据个数：";

goto RESETMARK1;

}

else if (arrayNum < 0)

{

cout << "======================================================" << endl;

cout << "数据个数小于0，无法排序，请重新输入需要排序的数据个数：";

goto RESETMARK1;

}

else if (arrayNum > 100)

{

cout << "======================================================" << endl;

cout << "数据个数大于100，请通过文件输入需要排序的数据个数：";

goto RESETMARK1;

}

SIZE\_OF\_ARRAY = arrayNum; //设置数组大小

cout << "请输入需要排序的数据，以半角空格分隔，回车确认：" << endl;

double \*origArray = new double[arrayNum]; //传递用数据数组

for (int i = 0; i < arrayNum; i++) {

scanf("%lf", &origArray[i]);

}

getchar();

return origArray;

}

//TimeData.h 基本类，见上基本储存结构

//TimeData.cpp 基本类，见上基本储存结构

//resource.h 图标

#define IDI\_ICON1 102

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NEXT\_RESOURCE\_VALUE 103

#define \_APS\_NEXT\_COMMAND\_VALUE 40001

#define \_APS\_NEXT\_CONTROL\_VALUE 1001

#define \_APS\_NEXT\_SYMED\_VALUE 101

#endif

#endif

//resource.cpp 图标

#include "resource.h"

#define APSTUDIO\_READONLY\_SYMBOLS

#include "winres.h"

#undef APSTUDIO\_READONLY\_SYMBOLS

#if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_CHS)

LANGUAGE LANG\_CHINESE, SUBLANG\_CHINESE\_SIMPLIFIED

#ifdef APSTUDIO\_INVOKED

1 TEXTINCLUDE

BEGIN

"resource.h\0"

END

2 TEXTINCLUDE

BEGIN

"#include ""winres.h""\r\n"

"\0"

END

3 TEXTINCLUDE

BEGIN

"\r\n"

"\0"

END

#endif

IDI\_ICON1 ICON "C:\\Users\\Kafuziroh\\Documents\\GitHub\\HongkongJournalist\\DSR CA LOGO.ico"

#endif

#ifndef APSTUDIO\_INVOKED

#endif

//DataCollection.h 数据输出至文本

#ifndef \_DATACOLLECTION

#define \_DATACOLLECTION

#include "TimeData.h"

int DataCollection(TimeData dataArray[]);

#endif // !\_DATACOLLECTION

//DataCollection.cpp 数据输出至文本

#include <iostream>

#include <fstream>

#include "TimeData.h"

using namespace std;

int DataCollection(TimeData dataArray[]) {

ofstream ofile("data\_collection.txt");

if (!ofile)

{

cout << "建立文件失败，请检查用户是否有程序所在文件夹的写入权限" << endl;

return 1; //失败为1

}

/\*ofile << "=====原始排序数据输出=====" << endl;\*/

for (int i = 0; i < 10 ; i++)

{

ofile << dataArray[i].GetWay() << " " << dataArray[i].GetTime() << " " << dataArray[i].GetTangShu() << endl;

}

ofile << endl;

return 0; //成功为0

}

// GraphDrawer.h 图表绘制

#include "TimeData.h"

int GraphDrawer(TimeData arr[]);

// GraphDrawer.cpp 图表绘制

#include <iostream>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

int GraphDrawer(TimeData arr[])

{

cout << "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■" << endl;

cout << "■ 数据图表 ■" << endl;

cout << "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■" << endl << endl;

for (int k = 0; k < 10; k++)

{

for (int i = 0; i <= (int((arr[k].GetTime()) / double(arr[9].GetTime() / 20))) \* 2; i++)

//以数组总大小决定长度范围避免图表过长

{

cout << "■";

}

cout << " " << arr[k].GetTime() << "μs" << " " << arr[k].GetTangShu() << "次" << arr[k].GetWay() << endl << endl;

}

cout << "■ 最快方法 ■ " << arr[0].GetWay() << " " << arr[0].GetTime() << "μs" << " " << arr[0].GetTangShu() << "次排序" << endl << "■ 最慢方法 ■ " << arr[9].GetWay() << " " << arr[9].GetTime() << "μs" << " " << arr[9].GetTangShu() << "次排序" << endl;

return 0;

}

//ShuChu.h 读取TXT

int ShuChu();

//ShuChu.cpp 读取TXT

#include <iostream>

#include <fstream>

#include <sstream>

#include "TimeData.h"

#include "GraphDrawer.h"

using namespace std;

int ShuChu()

{

ifstream ifile("data\_collection.txt");

if (!ifile)

{

return 1;

}

int i = 0, j;

double k;

TimeData str[10];

char ste[20];

string line;

while (getline(ifile, line))

{

if (i > 9)

{

break;

}

stringstream ss;

ss << line;

ss >> ste >> k >> j;

str[i].SetWay(ste);

str[i].SetTime(k);

str[i].SetTangShu(j);

i++;

}

int gap;

TimeData temp;

for (gap = 5; gap > 0; gap /= 2) //分组进行排序

{

for (i = gap; i < 10; i++)

{

for (j = i - gap; j >= 0 && str[j].GetTime() > str[j + gap].GetTime(); j -= gap)

{

temp = str[j];

str[j] = str[j + gap];

str[j + gap] = temp;

}

}

}

GraphDrawer(str);

ifile.close();

return 0;

}

//ChaRu.h 插入排序

#include "TimeData.h"

TimeData ChaRu(double str[], int len);

//ChaRu.cpp 插入排序

#include <iostream>

#include <windows.h>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

TimeData ChaRu(double str[], int len)

{

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

double dqFreq; //计时器频率

LARGE\_INTEGER f; //计时器频率

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

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

{

int it = str[i];

int iu = i - 1;

while ((iu >= 0) && (it < str[iu])) //将新插入的数与之前的一个数相比较，若大则放置在最后一位，若小则放在比它大和比它小的数中间

{

str[iu + 1] = str[iu];

iu--;

}

str[iu + 1] = it;

}

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

//乘以1000000把单位由秒化为微秒，精度为1000 000/（cpu主频）微秒

TimeData pp("插入排序", run\_time, len - 1);

return pp;

}

//ChooseSort.h 选择排序

#include "TimeData.h"

TimeData ChooseSort(double a[], int n);

//ChooseSort.cpp 选择排序

#include <iostream>

#include <windows.h>

#include <algorithm>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

TimeData ChooseSort(double a[], int n) //选择法排序

{

int i, j, k;

double tmp;

double dqFreq; //计时器频率

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

LARGE\_INTEGER f;

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

for (i = 0; i < n - 1; i++) //控制循环次数，一共n个数需要n-1次循环

{

k = i; //用k标记循环次数i

for (j = i + 1; j < n; j++) //如果i不等于k说明a[i]不是最小

{

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

k = j;

}

if (i != k) //如果i不等于k说明a[i]不是最小

{ //若i和k相等就说明a[i]是本次循环最小的值

tmp = a[i];

a[i] = a[k];

a[k] = tmp;

} //数组替换

}

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

// 乘以1000000把单位由秒化为微秒，精度为1000 000 / （cpu主频）微秒

TimeData aaa("选择排序", run\_time, n - 1); //方法名称，所用时间，排序趟数

return aaa;

}

//ErFeng.h 二分排序

#include "TimeData.h"

TimeData ErFeng(double str[], int len);

//ErFeng.cpp 二分排序

#include <iostream>

#include <windows.h>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

int sun = 0;

TimeData ErFeng(double str[], int len)

{

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

double dqFreq; //计时器频率

LARGE\_INTEGER f; //计时器频率

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

double temp;

int low, high, mid, i, j;

for (i = 1; i < len; i++)

{

temp = str[i]; //保存当前元素

low = 0;

high = i - 1;

while (low <= high) //在a[low...high]中折半查找

{

mid = (low + high) / 2; //找到中间元素

if (str[mid] > temp) //如果当前元素比中间元素小，当前元素要插入到中间元素的左侧

{

high = mid - 1;

}

else //如果当前元素比中间元素大，当前元素要插入到中间元素的右侧

{

low = mid + 1;

}

}

for (j = i - 1; j > high; --j) //元素后移

{

str[j + 1] = str[j];

}

str[high + 1] = temp; //插入

sun++;

}

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

//乘以1000000把单位由秒化为微秒，精度为1000 000/（cpu主频）微秒

TimeData pp("二分排序", run\_time, sun);

return pp;

}

//EXChangeSort.h 交换排序

#include "TimeData.h"

TimeData ChangeSort(double a[], int n);

//EXChangeSort.cpp 交换排序

#include <iostream>

#include <windows.h>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

TimeData ChangeSort(double a[], int n) //交换法排序

{

int i = 0, j;

double tmp;

double dqFreq; //计时器频率

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

LARGE\_INTEGER f;

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

for (i = 0; i < n - 1; i++) //将每一个循环的第一个数与后面的数逐一比较

{

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

{

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

{

tmp = a[j];

a[j] = a[i];

a[i] = tmp;

} //每一个循环第一个数字与数组中找到的更小数字交换

}

}

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

// 乘以1000000把单位由秒化为微秒，精度为1000 000 / （cpu主频）微秒

TimeData aaa("交换排序", run\_time, n - 1); //方法名称，所用时间，排序趟数

return aaa;

}

//FastSort.h 快速排序

#include "TimeData.h"

int partition(double arr[], int l, int r);

void \_\_quicksort(double arr[], int l, int r);

TimeData FastSort(double arr[], int n);

//FastSort.cpp 快速排序

#include <iostream>

#include <algorithm>

#include <windows.h>

#include "TimeData.h"

using namespace std;

int TS = 0; //排序趟数统计器

int partition(double arr[], int l, int r)

{

double v = arr[l];

int j;

j = l;

for (int i = l + 1; i <= r; i++)

{

if (arr[i] < v)

{

swap(arr[j + 1], arr[i]);

j++;

}

}

swap(arr[l], arr[j]);

return j;

}

void \_\_quicksort(double arr[], int l, int r)

{

TS++; //排序趟数++

if (l >= r)

return;

int p = partition(arr, l, r);

\_\_quicksort(arr, l, p - 1);

\_\_quicksort(arr, p + 1, r);

}

TimeData FastSort(double arr[], int n)

{

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

double dqFreq; //计时器频率

LARGE\_INTEGER f; //计时器频率

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

\_\_quicksort(arr, 0, n - 1);

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

//乘以1000000把单位由秒化为微秒，精度为1000000/（CPU主频）微秒

TimeData a("快速排序", run\_time, TS); //方法名称，所用时间，排序趟数

return a;

}

//MaoPao.h 冒泡排序

#include "TimeData.h"

TimeData MaoPao(double str[], int len);

//MaoPao.cpp 冒泡排序

#include <iostream>

#include <Windows.h>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

TimeData MaoPao(double str[], int len)

{

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

double dqFreq; //计时器频率

LARGE\_INTEGER f; //计时器频率

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

for (int i = len; i > 0; i--) //总共需要排序

{

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

{

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

{

double temp = str[j]; //数组前后替换

str[j] = str[j + 1];

str[j + 1] = temp;

}

}

}

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

//乘以1000000把单位由秒化为微秒，精度为1000 000/（cpu主频）微秒

TimeData pp("冒泡排序", run\_time, len - 1);

return pp;

}

//Shell.h SHELL排序

#include "TimeData.h"

TimeData Shell(double str[], int len);

//Shell.cpp SHELL排序

#include <iostream>

#include <windows.h>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

TimeData Shell(double str[], int len)

{

int sumBBB = 0;

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

double dqFreq; //计时器频率

LARGE\_INTEGER f; //计时器频率

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

int i, j, gap;

double temp;

for (gap = len / 2; gap > 0; gap /= 2) //分组进行排序

{

for (i = gap; i < len; i++)

{

for (j = i - gap; j >= 0 && str[j] > str[j + gap]; j -= gap)

{

temp = str[j];

str[j] = str[j + gap]; //将相差gap位置的两个数

str[j + gap] = temp;

}

}

sumBBB++;

}

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

//乘以1000000把单位由秒化为微秒，精度为1000 000/（cpu主频）微秒

TimeData pp("SHELL排序", run\_time, sumBBB);

return pp;

}

//ShuangXiangKuaiPai.h 双路快排

#include "TimeData.h"

int partitionT(double arr[], int l, int r);

void \_\_quicksortT(double arr[], int l, int r);

TimeData quicksortT(double arr[],int n);

//ShuangXiangKuaiPai.cpp 双路快排

#include <iostream>

#include <algorithm>

#include "TimeData.h"

#include <Windows.h>

using namespace std;

int sumAAA = 0; //计算趟数所用

int partitionT(double arr[], int l, int r)

{

double v = arr[l];

int j;

j = l;

for (int i = l + 1; i <= r; i++)

{

if (arr[i] < v)

{

swap(arr[j + 1], arr[i]);

j++;

}

}

swap(arr[l], arr[j]);

return j;

}

void \_\_quicksortT(double arr[], int l, int r)

{

sumAAA++;

if (l >= r)

return;

int p = partitionT(arr, l, r);

\_\_quicksortT(arr, l, p - 1);

\_\_quicksortT(arr, p + 1, r);

}

TimeData quicksortT(double arr[], int n)

{

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

double dqFreq; //计时器频率

LARGE\_INTEGER f; //计时器频率

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

\_\_quicksortT(arr, 0, n - 1);

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

//乘以1000000把单位由秒化为微秒，精度为1000 000/（cpu主频）微秒

TimeData pp("双路快速排序", run\_time, sumAAA);

return pp;

}

//ThreeRoadFastSort.h 三路快排

#include "TimeData.h"

template <typename T>

void \_\_quicksort3(T arr[], int l, int r);

TimeData ThreeRoadFastSort(double arr[], int n);

//ThreeRoadFastSort.cpp 三路快排

#include <iostream>

#include <Windows.h>

#include <algorithm>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

int sumbbb = 0; //计算趟数所用

template <typename T>

void \_\_quicksort3(T arr[], int l, int r) //三路排序

{

sumbbb++;

if (l >= r)

return;

T v = arr[l];

int lt = l;

int gt = r + 1;

int i = l + 1;

while (i < gt)

{

if (arr[i] < v)

{

swap(arr[i], arr[lt + 1]);

lt++;

i++;

}

else if (arr[i] > v)

{

swap(arr[i], arr[gt - 1]);

gt--;

}

else

{

i++;

}

}

swap(arr[l], arr[lt]);

\_\_quicksort3(arr, l, lt - 1);

\_\_quicksort3(arr, gt, r);

}

TimeData ThreeRoadFastSort(double arr[], int n)

{

int i, j, k;

double tmp;

double dqFreq; //计时器频率

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

LARGE\_INTEGER f;

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

\_\_quicksort3(arr, 0, n - 1);

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

// 乘以1000000把单位由秒化为微秒，精度为1000 000 / （cpu主频）微秒

TimeData pp("三路快速排序", run\_time, sumbbb); //方法名称，所用时间，排序趟数

return pp;

}

//TwoWayBubbleSort.h 双向冒泡排序

#include "TimeData.h"

TimeData TwoWayBubbleSort(double a[], int n);

//TwoWayBubbleSort.cpp 双向冒泡排序

#include <iostream>

#include <windows.h>

#include "TimeData.h"

using namespace std;

extern int SIZE\_OF\_ARRAY;

TimeData TwoWayBubbleSort(double number[], int n)

{

double dqFreq; //计时器频率

double run\_time;

\_LARGE\_INTEGER time\_start; //开始时间

\_LARGE\_INTEGER time\_over; //结束时间

LARGE\_INTEGER f;

QueryPerformanceFrequency(&f);

dqFreq = (double)f.QuadPart;

QueryPerformanceCounter(&time\_start); //计时开始

int left = 0, right = SIZE\_OF\_ARRAY - 1, shift = 1;

int i;

while (left < right)

{

for (i = left; i < right; i++)

{

if (number[i] > number[i + 1])

{

swap(number[i], number[i + 1]);

shift = i;

}

}

right = shift;

for (i = right - 1; i >= left; i--)

{

if (number[i] > number[i + 1])

{

swap(number[i], number[i + 1]);

shift = i + 1;

}

}

left = shift;

}

QueryPerformanceCounter(&time\_over); //计时结束

run\_time = 1000000 \* (time\_over.QuadPart - time\_start.QuadPart) / dqFreq;

// 乘以1000000把单位由秒化为微秒，精度为1000 000 / （cpu主频）微秒

TimeData aaa("双向冒泡排序", run\_time, 2 \* (n - 1)); //方法名称，所用时间，排序趟数

return aaa;

}

**五、测试数据及其结果分析**

运行后，程序的主菜单界面如图7。

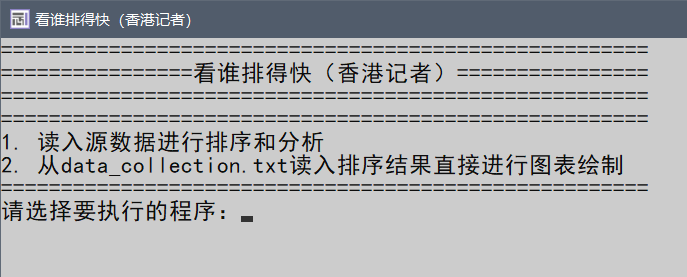


图7 主菜单图

（1）读入源数据进行排序和分析，见图8.

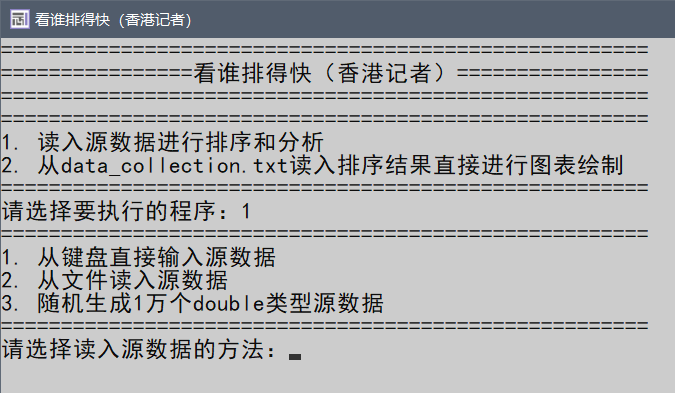


图8 读入方法选择图

（2）选择读入源数据方法，三种方法对应见图9、图10、图11。

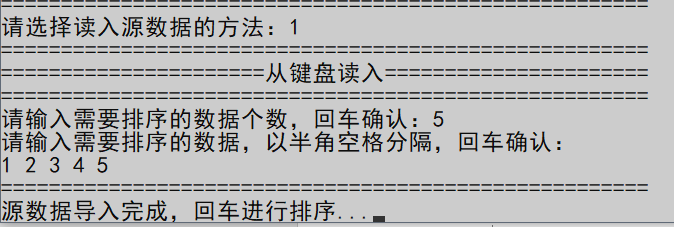


图9 键盘读入图

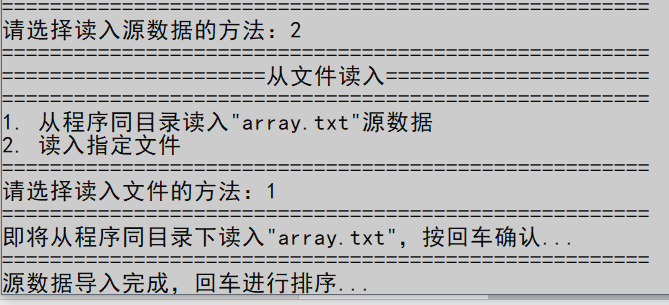


图10 文件读入图

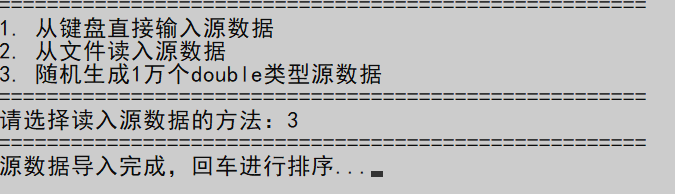


图11 随机数读入图

（3）排序测试，见图12。

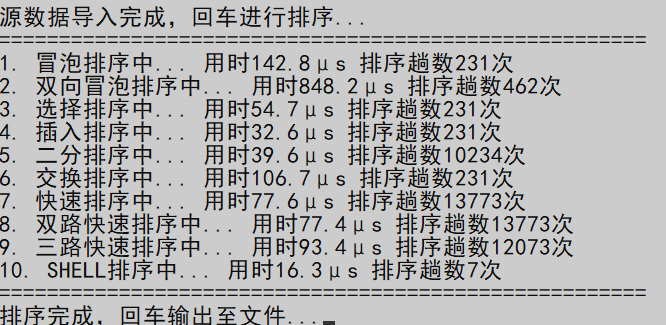


图12 排序测试图

（4）输出测试，见图13、图14。

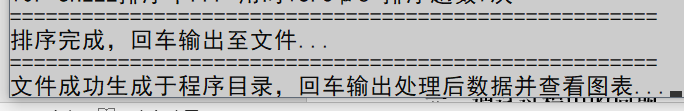


图13 输出图



图14 输出文件图

（5）图表绘制测试，见图15、图16。



图15 图表

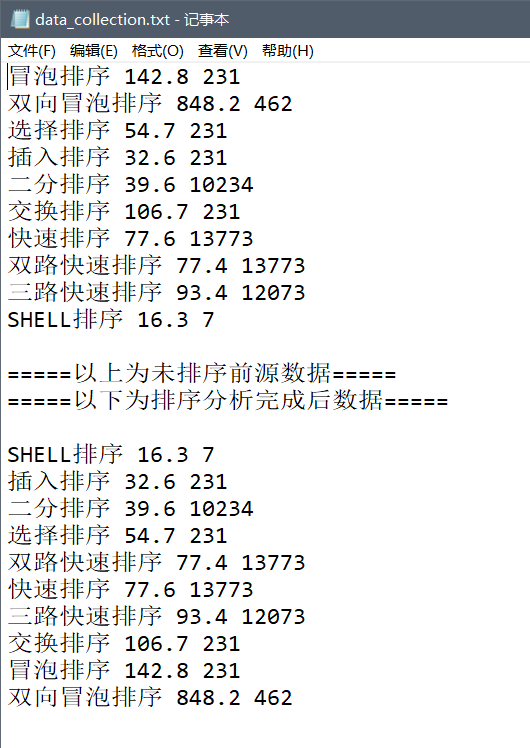


图16 最终文件图

（6）直读测试，见图17。

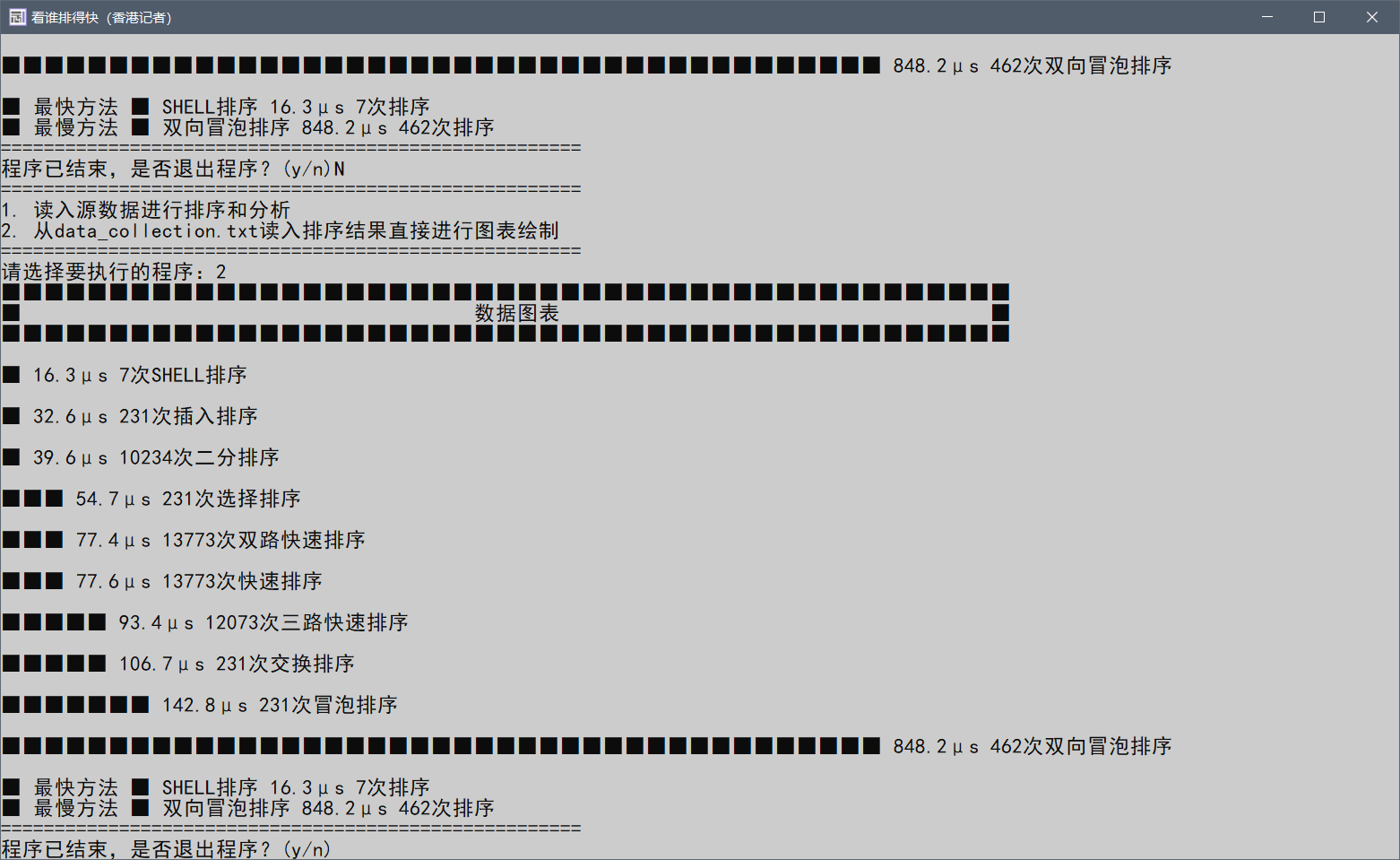


图17 直读测试图

结果分析如下。

（1）在输入菜单选项时如果输入的内容不是选项有的的数字，而是其他数字，系统将继续执行等待下次正确的输入。

（2）在需要输入所须查找的名称处如输入名称不在系统中，系统提示错误并要求重新输入。

（3）在读取文件或者生成文件时产生问题，系统会申请重试。

（4）程序结束后可以决定退出或从头再次运行。

**六、调试过程中的问题**

（1）TimeData类函数问题：在从文件读取数据并建立TimeData类的对象时发现无法输入数据，后发现是由于TimeData的成员变量为私有成员变量，由于只设置了Get函数而未设置Set函数而导致无法输入数据，增加Set函数解决问题。（张承扬）

（2）自动下一步问题：在选择文件输入方法步骤输入数据选择文件输入方法后，下一步正式开始读取文件自动进行而未成功通过回车指令确认；后发现原因是后文确认所用getchar()函数吸收了前面步骤的回车所导致，多增加一个getchar()函数解决。（张承扬）

（3）在编写系统随机生成的随机数据集时，有两种函数可供使用，C++ 函数库中的随机数生成器中的rand()不需要参数，它会返回一个从0到最大随机数的任意整数，最大随机数的大小通常是固定的一个大整数，rand()产生的随机数在每次运行的时候都与上一次相同，所以某种意义上来说并不是绝对的随机数，会存在一定的偶然性；而另一函数srand（），参数seed是rand()的种子，用来初始化rand()的起始值，返回一个[seed, RAND\_MAX（0x7fff）)间的随机整数。但是程序要求生成或给定的数不一定都是int型的数，还有小数的存在，所以要避免这类问题的出现，需要让系统生成的任意整数除以10、100、1000之类的数，以形成浮点数，再放在程序里运行排序并输出。（张琪）

（4）读取D盘根目录下的目标文件“data\_collection.txt”时，在TimeData类里有几种不同排序方法的字符串对象，无法用C语言的结构体放在数组中以便读取并且输出，于是我们尝试了书上ifstream读取TimeData类里面的数组，想用用for循环读取，编译器报错无法读取。所以我们选择用C++中的getline链接数组与后面需要读取的排序时间和排序趟数，先使用ss串读取一行的内容再读取之后的每一行TimeData类对象的内容，就做到了读取数组的功能。（张琪）

（5）读入数组的时候，cin后面显示没有与这些操作数匹配的 ">>" 运算符，之后就一直无法理解其中的原理，于是就去尝试了定义inf的文件流类，机遇TimeData类，重载输入流运算符">>" ，在TimeData.h头文件里的public中用友元函数重载，之后便能正确使用">>"运算符，并且在main函数文件中缺少了#include <string> 的库头文件。（张琪）

（6）在统计排序的子函数时，我都用了一个全局变量sum，来统计所经历的趟数，但运行之后报错了，因为sum是需要作为一个返回值的，所以做全局变量如果有多个文件同时使用的话，会报错，所以，改变了每个函数中全局变量的变量名。（杨楚瀛）

（7）在编写从文件读入数据时，我们要读取的数据有三个类型，因此c的读文件方式不适用与我们的情况，而在我们使用c++读文件方法时，一开始我们使用ifstream读取数据并用for循环将其存入我们自己写的类TimeData数组里，但是因为我们读出的数据中第一个是string类型的，用inf.getline（str，50），将其存入一个字符数组里会导致会将后面同一行的数据也读到这个数组里，而且用for循环读也无法自己换行，于是我们采用了定义一个没有初始值的string类型的变量line，这样getline（str，line），，应为不设置默认值所以默认为读到回车停止操作，我们先用一个串流ss读取一行的所有内容，然后再在这个串流里分别读取我们需要的内容；而且一开始用于读取string类型数的那个数组我只定义了10的长度，导致了爆栈，之后运行后发现错误将其改正了过来。（杨楚瀛）

**七、课程设计总结**

张承扬：通过这次设计，我充分体会到了需求分析、整体规划和基础代码的重要性。在正式开始设计之前，一定要正确进行需求分析，不然就会导致后续的工作目标混乱与时间的浪费。整体规划非常重要，在开始写多文件工程是一定要事先讨论好对接整合的需要，否则整合入主程序进行调用时会产生各种各样的问题。基础代码之所以是基础，因为其支撑着所有的分代码，TimeData类出现的问题会影响到所有用到它的函数，会导致进度大幅拖慢，所以检查仔细必不可少。

杨楚瀛：这次我们的程序设计的程序是“看谁排的快“，主要功能是需要实现用不同方法输入后，调用不同的排序函数来对其进行排序，并返回排序类型、排序时间和排序趟数三个参数，将其汇入文件，然后再输入到内存进行排序来看那个排序函数的速度最快，再把输到屏幕上进行展示。在写一个个子函数时我们用多文件工程的方法，使我们的程序调理更加清晰，并在此过程中对各自写的排序方法有了更深的了解。因为需要返回string类型、double类型、int类型的三个值，我们写了一个类用于储存和读入这些数值，更加方便了主函数子函数之间值的传递；在读入文件和读出文件方面，c的读写文件方法并不十分适用于我们的情况，我们就通过讨论查资料，自学课本的方法自学了读写文件的方法，并运用了最简便的一种来读写文件 ，大大的加强了我们的自学能力。

张琪：本周的程序设计课让我感触很多，我们根据班级人员的熟悉程度组成小组，学习的同时也让同学关系更进一步，并且小组成员根据个人程序设计知识掌握情况进行合理分工，动手实践，将整个程序设计工作化整为零，为今后进入工作岗位提供了宝贵的经验方法。程序设计将我们所学的知识活学活用，各尽自己所能，发挥自身优势，系统地构建了整个程序，在进行程序设计的同时也互相交流各自的想法，扩大了自身的知识面，找到不一样的思路和方法来解决问题，既做成了一套程序，也让自己的知识层面更上一层楼，找到解决问题的同时也能够及时发现自身存在的编码问题，以纠正自身的误解，也能够在以后少走弯路，为之后的人生打下了坚实的基础。