项目说明文档

——8种排序算法的比较

姓名：吴桐欣

学号：1652677

同济大学 软件学院 软件工程专业

目录

项目说明文档——考试报名系统 1

目录 2

1.项目概述 4

1.1项目简介 4

1.2文件目录 4

1.3操作指南 4

1.4注意事项 4

2.思路与设计 5

2.1基本思路 5

2.2设计 5

3.具体实现 6

3.1冒泡排序 6

3.2选择排序 7

3.3直接插入排序 8

3.4希尔排序 9

3.5快速排序 10

3.6堆排序 11

3.7归并排序 14

3.8基数排序 15

4.测试 18

4.1功能测试 18

18

4.2出错测试 19

1.项目概述

1.1项目简介

随机函数产生10000个随机数，用快速排序，直接插入排序，冒泡排序，选择排序的排序方法排序，并统计每种排序所花费的排序时间和交换次数。其中，随机数的个数由用户定义，系统产生随机数。并且显示他们的比较次数。

**1.1.1功能分析**

本项目要求实现功能

1. 产生一定数量的随机数；
2. 进行冒泡排序；
3. 进行选择排序；
4. 进行直接插入排序；
5. 进行希尔排序；
6. 进行快速排序；
7. 进行堆排序；
8. 进行归并排序；
9. 进行基数排序。

1.2文件目录

（1）P10\_1652677\_吴桐欣\_说明文档.docx（本文档）

（2）P10\_1652677\_吴桐欣.exe（可执行文件）

（3）P10\_1652677\_吴桐欣.cpp（源文件）

（4）P10\_1652677\_吴桐欣.h（头文件）

1.3操作指南

（1）运行程序后，将获得程序提示“请输入要产生的随机数的个数：”

用户输入一个数字表示考随机数的个数。

（2）随后程序提示“请选择排序算法：”

用户需要按照所提供的操作数表选择排序算法。

（3）操作数表如下：

1--冒泡排序

2--选择排序

3--直接插入排序

4--希尔排序

5--快速排序

6--堆排序

7--归并排序

8--基数排序

9--退出程序

1.4注意事项

（1）用户不能输入除数字以外的字符

（2）用户输入的操作数不能超出所提供的范围

2.思路与设计

2.1基本思路

每种排序方式用至少一个函数来实现。函数仅在函数内改变数组的值，函数调用结束后，数组顺序保持原来的乱序，以供多次利用、公平比较各个排序的效率。

2.2设计

**2.2.1数据结构**

用vector数组存放产生的随机数。

3.具体实现

3.1冒泡排序

核心代码

void BubbleSort(vector<int> list){

long n=list.size();//元素个数

int count\_comp=0;//记录比较次数

int count\_swap=0;//记录交换次数

clock\_t start=clock();

for (int i=0; i<n-1; i++) {

bool exchange=false;//是否产生交换

for (long j=n-1; j>0; j--) {

count\_comp++;

if (list[j-1]>list[j]) {

//发现逆序，进行交换

SWAP(list[j], list[j-1]);

count\_swap++;

exchange = true;

}

}

if (!exchange) {

//本轮不产生交换，排序完毕

break;

}

}

clock\_t end=clock();

cout<<"冒泡排序所用时间："<<(float)(end-start)/CLOCKS\_PER\_SEC<<" 秒"<<endl;

cout<<"冒泡排序比较次数："<<count\_comp<<endl;

cout<<"冒泡排序交换次数："<<count\_swap<<endl<<endl;

}

说明

使用两个嵌套的for循环，内层表示遍历数组，发现逆序则前后交换数值；外层表示进行至多n-1次的遍历排序。

分别记录数据比较次数和数据交换次数。

排序开始前后分别用clock()函数获得数据，后者减去前者，得到时间差。

3.2选择排序

核心代码

void SelectSort(vector<int> list){

long n=list.size();//元素个数

int count\_comp=0;//记录比较次数

int count\_swap=0;//记录交换次数

clock\_t start=clock();

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

int min\_i=i;//最小元素的序号

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

count\_comp++;

if (list[min\_i]>list[j]) {min\_i = j;}

}

//交换

SWAP(list[i], list[min\_i]);

count\_swap++;

}

clock\_t end=clock();

cout<<"选择排序所用时间："<<(float)(end-start)/CLOCKS\_PER\_SEC<<" 秒"<<endl;

cout<<"选择排序比较次数："<<count\_comp<<endl;

cout<<"选择排序交换次数："<<count\_swap<<endl<<endl;

}

说明

使用两个嵌套的for循环，内层表示在剩余的数中遍历寻找最小的数，外层表示从小到大依次放置元素。

分别记录数据比较次数和数据交换次数。

排序开始前后分别用clock()函数获得数据，后者减去前者，得到时间差。

3.3直接插入排序

核心代码

void InsertSort(vector<int> list){

long n=list.size();//元素个数

int count\_comp=0;//记录比较次数

int count\_swap=0;//记录交换次数

int temp;//暂存要插入的元素

clock\_t start=clock();

for (int i=1; i<n; i++) {

//获取要插入的元素

temp=list[i];

//从后向前一个个比较

for (int j=i-1; j>=0; j--) {

count\_comp++;

if (list[j]>temp) {

//将前一个元素后移

list[j+1]=list[j];

} else {

//若前面数小于temp，则找到要插入的位置

list[j+1]=temp;

break;

}

}

}

count\_swap=count\_comp;//每次比较都会修改数据，因此交换次数与比较次数相等

clock\_t end=clock();

cout<<"直接插入排序所用时间："<<(float)(end-start)/CLOCKS\_PER\_SEC<<" 秒"<<endl;

cout<<"直接插入排序比较次数："<<count\_comp<<endl;

cout<<"直接插入排序交换次数："<<count\_swap<<endl<<endl;

}

说明

使用两个嵌套的for循环，内层表示在已排好的序列片段中寻找某元素的插入位置，外层表示依次插入数组中的元素。

分别记录数据比较次数和数据交换次数。

排序开始前后分别用clock()函数获得数据，后者减去前者，得到时间差。

3.4希尔排序

核心代码

void ShellSort(vector<int> list){

int count\_comp=0;//记录比较次数

int count\_swap=0;//记录交换次数

long n=list.size();//元素个数

int gap=(int)n;//增量

clock\_t start=clock();

do{

gap = gap/3+1;//计算增量

for (int i=gap; i<n; i+=gap) {

count\_comp++;

//若在子序列中发现逆序

if (list[i-gap]>list[i]) {

int temp=list[i];

//用直接插入法，从后向前比较

for (int j=i-gap; ; j-=gap) {

count\_comp++;

if (list[j]<=temp) {

//小于等于temp，找到temp插入位置

list[j+gap]=temp;

count\_swap++;

break;

} else {

//若比temp大，则后移一位

list[j+gap]=list[j];

count\_swap++;

}

}

}

}

} while (gap>1);

clock\_t end=clock();

cout<<"希尔排序所用时间："<<(float)(end-start)/CLOCKS\_PER\_SEC<<" 秒"<<endl;

cout<<"希尔排序比较次数："<<count\_comp<<endl;

cout<<"希尔排序交换次数："<<count\_swap<<endl<<endl;

}

说明

三层循环嵌套，最外层是每次计算出不同的gap增量数值；中层表示待排序序列被划分成的几个子序列依次分别进行排序；内层表示对一个子序列进行排序（直接插入排序）。

分别记录数据比较次数和数据交换次数。

排序开始前后分别用clock()函数获得数据，后者减去前者，得到时间差。

3.5快速排序

核心代码

//快速排序-三者取中

int& median3(vector<int>& list, const int left, const int right, int& count\_comp, int& count\_swap){

int mid=(left+right)/2;

int k=left;

if (list[mid]<list[k]) {k=mid;}

if (list[right]<list[k]) {k=right;}//left,mid,right三者选最小

if (k!=left) {

SWAP(list[k], list[left]);

}//把最小值移到left位置

if (mid!=right && list[mid]<list[right]) {

SWAP(list[right], list[mid]);

}//把中间值，即基准，移到right位置

return list[right];

}

//快速排序-一趟划分

int Partition(vector<int>& list, const int left, const int right, int& count\_comp, int& count\_swap){

int i=left;

int j=right-1;

if (left<right) {

int pivot=median3(list, left, right, count\_comp, count\_swap);

for (; ; ) {

while (i<j && list[i]<pivot) {i++;count\_comp++;}

while (i<j && list[j]>pivot) {j--;count\_comp++;}

if (i<j) {

SWAP(list[i], list[j]);//交换list[i],list[j]的值

count\_swap++;

i++;

j--;

} else {

break;

}

}

if (list[i]>pivot) {

list[right]=list[i];

list[i]=pivot;

count\_swap++;

count\_comp++;

}

}

return i;

}

int count\_comp=0;//记录比较次数

int count\_swap=0;//记录交换次数

clock\_t start=clock();

QuickSort(numbs, 0, (int)(numbs.size()-1), count\_comp, count\_swap);

clock\_t end=clock();

cout<<"快速排序所用时间："<<(float)(end-start)/CLOCKS\_PER\_SEC<<" 秒"<<endl;

cout<<"快速排序比较次数："<<count\_comp<<endl;

cout<<"快速排序交换次数："<<count\_swap<<endl<<endl;

//快速排序

void QuickSort(vector<int> list, const int left, const int right, int& count\_comp, int& count\_swap){

if (left==right) {}

else if (left==right-1) {

count\_comp++;

if (list[left]>list[right]) {

SWAP(list[left], list[right]);

count\_swap++;

}

}

else {

int pivotpos=Partition(list, left, right, count\_comp, count\_swap);//划分

QuickSort(list, left, pivotpos-1, count\_comp, count\_swap);

QuickSort(list, pivotpos+1, right, count\_comp, count\_swap);

}

}

说明

三者取中函数通过比较三个数的大小，获得较为合理的基准，防止划分的子序列长度相差太多。

一趟划分函数根据基准将待排序序列划分为大于基准数、小于基准数两个子序列。

递归调用快速排序函数，终止条件是划分出的子序列只有2个元素。

因涉及递归，计算次数和记录时间不能在函数内进行。

分别记录数据比较次数和数据交换次数。

排序开始前后分别用clock()函数获得数据，后者减去前者，得到时间差。

3.6堆排序

核心代码

//堆排序

void HeapSort(vector<int> list){

int count\_comp=0;//记录比较次数

int count\_swap=0;//记录交换次数

int n=(int)list.size();//元素个数

clock\_t start=clock();

for (int i=(n-2)/2; i>=0; i--) {

siftDown(list, i, n-1, count\_comp, count\_swap);

}

for (int i=n-1; i>=0; i--) {

SWAP(list[0], list[i]);

count\_swap++;

siftDown(list, 0, i-1, count\_comp, count\_swap);

}

clock\_t end=clock();

cout<<"堆排序所用时间："<<(float)(end-start)/CLOCKS\_PER\_SEC<<" 秒"<<endl;

cout<<"堆排序比较次数："<<count\_comp<<endl;

cout<<"堆排序交换次数："<<count\_swap<<endl<<endl;

}

//堆排序-从s到m自顶向下调整

void siftDown(vector<int>& heap, int s, int m, int& count\_comp, int& count\_swap){

int i=s;

int j=2\*i+1;//j是i的左子女

int temp=heap[i];

while (j<=m) {

count\_comp++;

if (j<m && heap[j]<heap[j+1]) {j++;}//j指向两子女中最大者

count\_comp++;

if (temp>=heap[j]){

break;

} else {

count\_swap++;

heap[i]=heap[j];

i=j;

j=2\*i+1;//i降到子女的位置

}

}

heap[i]=temp;

count\_swap++;

}

说明

siftDown函数用while循环调整成为最大堆。

堆排序函数先for循环多次调用siftDown函数，使序列调整为堆。

再用for循环令每次循环都将得到的最大堆的堆顶放置在序列后面。

分别记录数据比较次数和数据交换次数。

排序开始前后分别用clock()函数获得数据，后者减去前者，得到时间差。

3.7归并排序

核心代码

//归并排序函数

void MergeSort(vector<int>& list, int left, int right, int& count\_comp, int& count\_swap){

if (left>=right) {

return;

}

int mid=(left+right)/2;

MergeSort(list, left, mid, count\_comp, count\_swap);//左子序列排序

MergeSort(list, mid+1, right, count\_comp, count\_swap);//右子序列排序

Merge(list, left, mid, right, count\_comp, count\_swap);//合并左右子序列

}

//归并排序-合并

void Merge(vector<int>& list, const int left, const int mid, const int right, int& count\_comp, int& count\_swap){

vector<int> tempList(list);

int s1=left,s2=mid+1,t=left;//s1，s2是检测指针，t是存放指针

while(s1<=mid && s2<=right){

count\_comp++;

count\_swap++;

if (tempList[s1]<=tempList[s2]) {

list[t++]=tempList[s1++];

} else {

list[t++]=tempList[s2++];

}

}

while (s1<=mid) {

list[t++]=tempList[s1++];

count\_swap++;

}

while (s2<=right) {

list[t++]=tempList[s2++];

count\_swap++;

}

}

vector<int> copyList(numbs);

int count\_comp=0;//记录比较次数

int count\_swap=0;//记录交换次数

clock\_t start=clock();

MergeSort(copyList, 0, (int)(numbs.size()-1), count\_comp, count\_swap);

clock\_t end=clock();

cout<<"归并排序所用时间："<<(float)(end-start)/CLOCKS\_PER\_SEC<<" 秒"<<endl;

cout<<"归并排序比较次数："<<count\_comp<<endl;

cout<<"归并排序交换次数："<<count\_swap<<endl<<endl;

说明

合并函数将两个分别有序的子序列合并为一个有序序列。

递归调用归并排序函数，每次先划分出子序列，分别用归并函数处理两个子序列，再用merge函数将两个子序列合并。

因涉及函数递归调用，所以计数和记录时间差不能在函数内进行。

分别记录数据比较次数和数据交换次数。

排序开始前后分别用clock()函数获得数据，后者减去前者，得到时间差。

3.8基数排序

核心代码

void RadixSort(vector<int> list){

int n=(int)list.size();//元素个数

int count\_comp=0;//记录比较次数

int count\_swap=0;//记录交换次数

vector<int> tempList(list);

clock\_t start=clock();

for (long place=1; ; place\*=10) {

int k=0;//存放指针

bool change=false;

for (int radix=0;; radix++) {

for (int i=0; i<tempList.size();i++) {

count\_comp++;

if ((tempList[i]%(10\*place))/place == radix) {

if (list[k]!=tempList[i]) {

list[k]=tempList[i];//放入结果列中

count\_swap++;

change=true;

}

k++;//存放指针移至下一个位置

if (i!=tempList.size()-1) {

//将此数移到列尾

int temp\_i=i;

while(temp\_i!=tempList.size()-1){

tempList[temp\_i]=tempList[temp\_i+1];

temp\_i++;

}

i--;//要使下一步从i位置开始

}

tempList.pop\_back();

}

}

if (radix>9 || k>=n) {

break;

}

}

if (change==false) {

break;

}

tempList=list;

}

clock\_t end=clock();

cout<<"基数排序所用时间："<<(float)(end-start)/CLOCKS\_PER\_SEC<<" 秒"<<endl;

cout<<"基数排序比较次数："<<count\_comp<<endl;

cout<<"基数排序交换次数："<<count\_swap<<endl<<endl;

}

说明

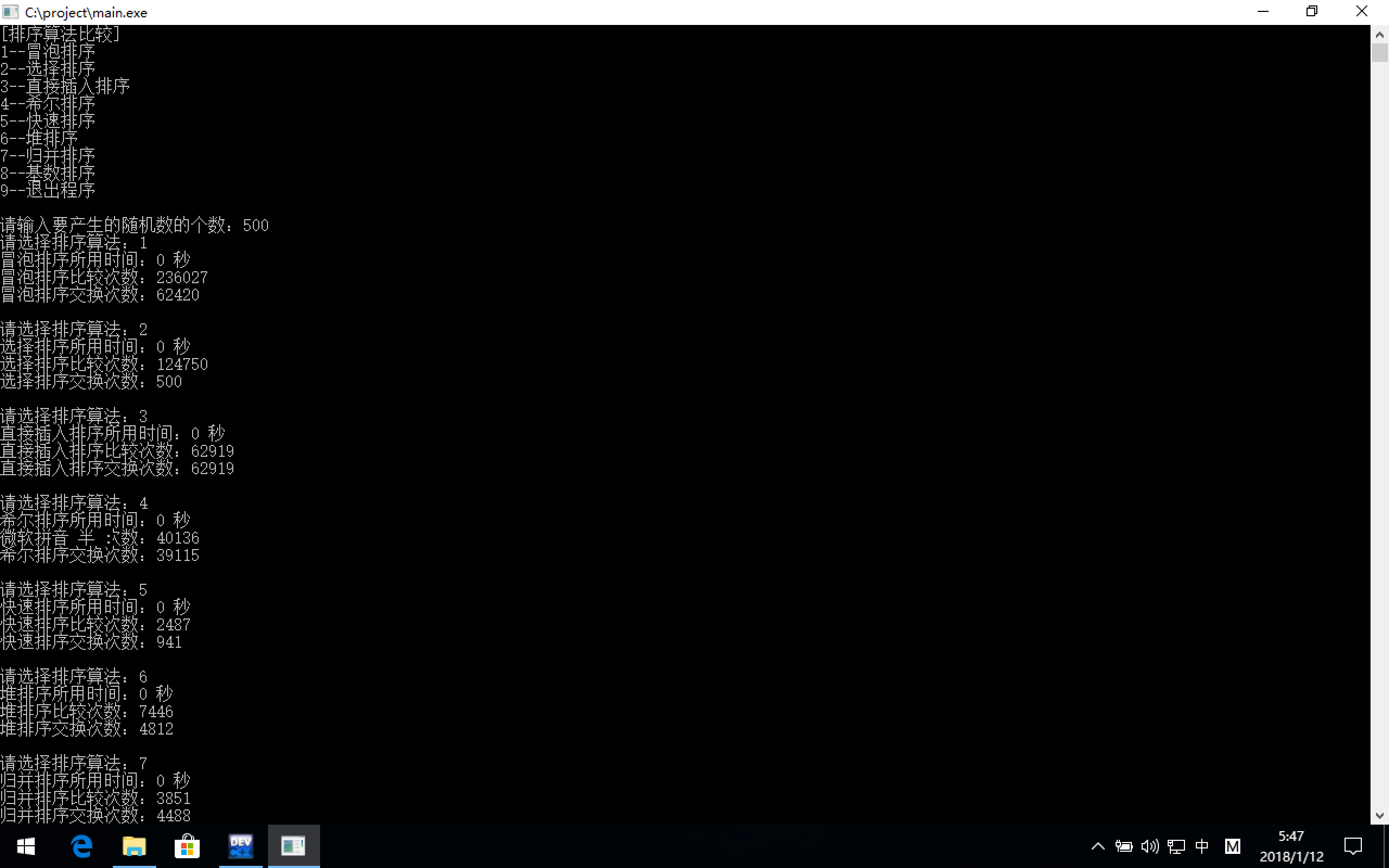
首先选出每次比较的数值位，每个位有0～9共10个基数，根据基数对序列进行排序。

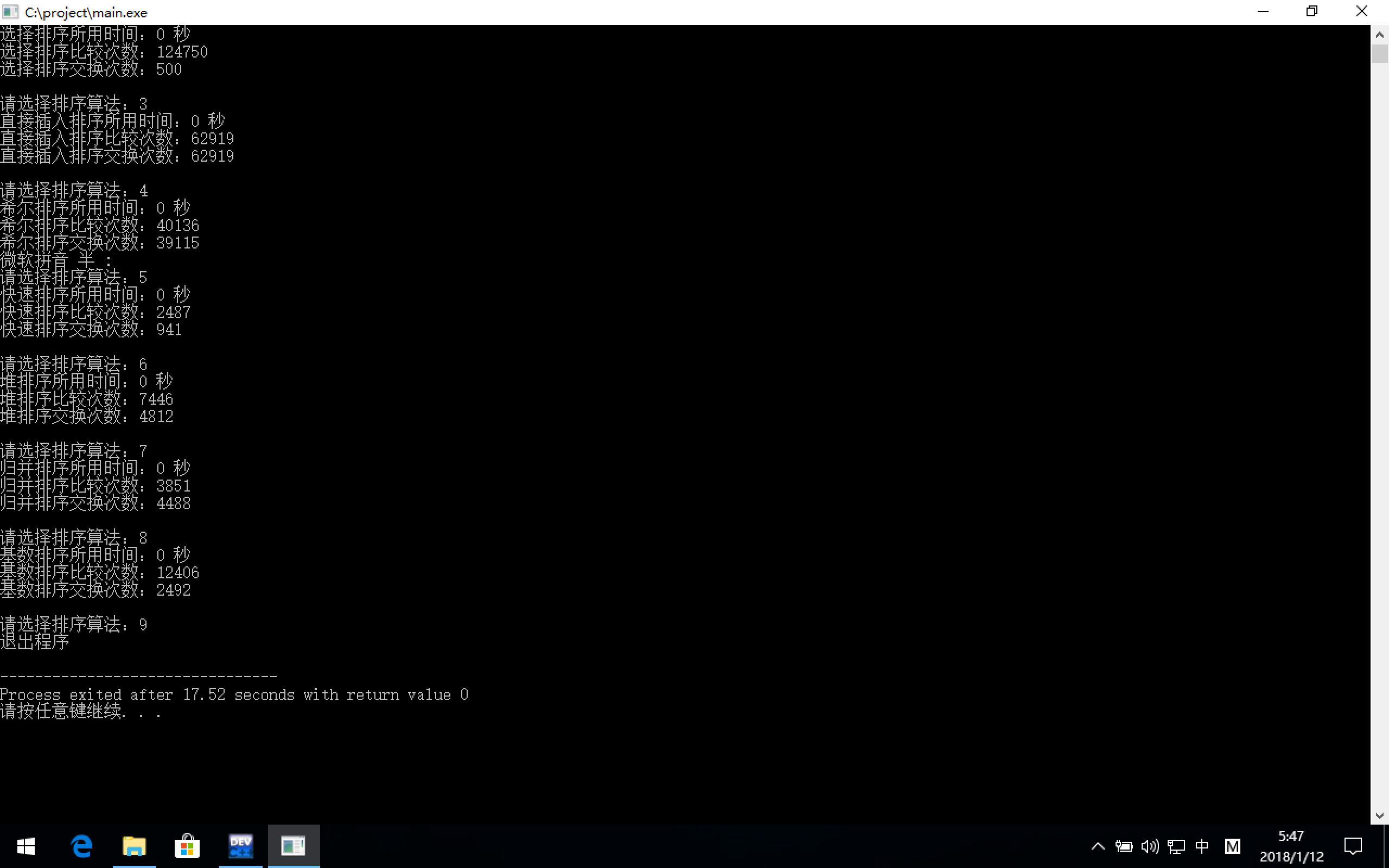
分别记录数据比较次数和数据交换次数。

排序开始前后分别用clock()函数获得数据，后者减去前者，得到时间差。

4.测试

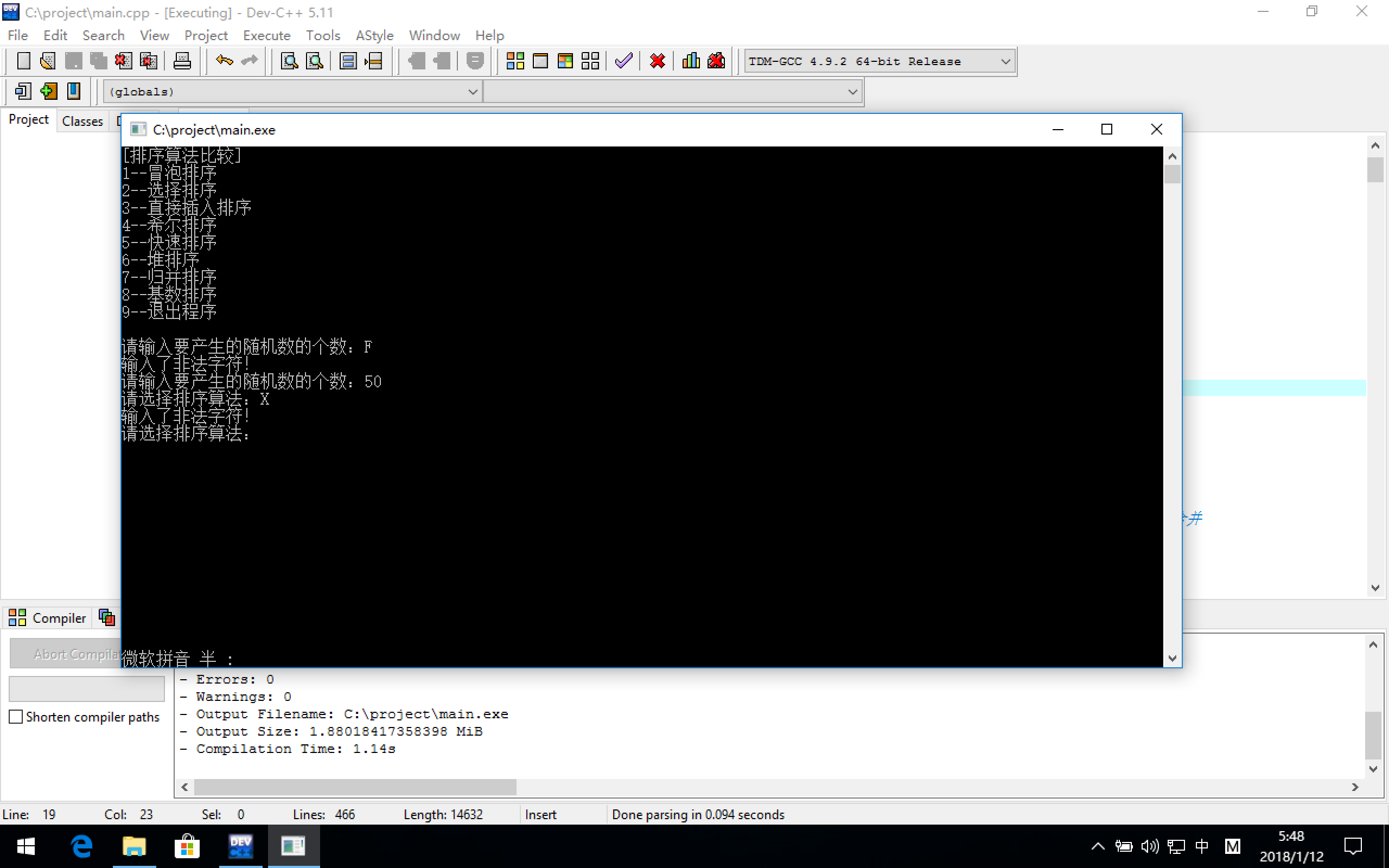
4.1功能测试





4.2出错测试

4.2.1输入非法字符



4.2.2输入数字不符合要求

