项目说明文档

数据结构课程设计

——8种排序算法的比较案例

同济大学

Tongji University

目 录

[1 分析 1](#_Toc9178)

[1.1 背景分析 1](#_Toc4852)

[1.2 功能分析 1](#_Toc6491)

[2 设计 1](#_Toc31349)

[2.1 数据结构设计 1](#_Toc27622)

[3 实现 2](#_Toc17481)

[3.1 冒泡排序算法的实现 2](#_Toc13356)

[3.1.1冒泡排序算法流程图 2](#_Toc15307)

[3.1.2冒泡排序算法核心代码 2](#_Toc6090)

[3.2 直接插入排序的实现 3](#_Toc295)

[3.2.1 直接插入排序流程图 3](#_Toc2868)

[3.2.2 直接插入排序核心代码 3](#_Toc30728)

[3.3 希尔排序算法的实现 4](#_Toc23896)

[3.3.1 希尔排序算法流程图 4](#_Toc4892)

[3.3.2 希尔排序算法核心代码 4](#_Toc31014)

[3.4 快速排序算法的实现 5](#_Toc18359)

[3.4.1 快速排序算法流程图 5](#_Toc21991)

[3.4.2 快速排序算法核心代码 5](#_Toc2519)

[3.5 堆排序算法的实现 7](#_Toc11767)

[3.5.1 堆排序算法的流程图 7](#_Toc12153)

[3.5.2 堆排序算法的核心代码 7](#_Toc6208)

[3.6 归并排序算法的实现 8](#_Toc23279)

[3.6.1 归并排序算法流程图 8](#_Toc29869)

[3.6.2 归并排序算法核心代码 8](#_Toc6063)

[3.7基数排序算法实现 10](#_Toc10470)

[3.7.1 基数排序算法流程图 10](#_Toc25485)

[3.7.2基数排序算法核心代码 10](#_Toc21591)

[4 测试 12](#_Toc28059)

[4.1 功能测试（time函数放大1000） 12](#_Toc26330)

[4.1.1 排序数为10的测试 12](#_Toc21544)

[4.1.2 排序数为100的测试 14](#_Toc18574)

[4.1.3 排序数为1000的测试 16](#_Toc8224)

[4.1.4 排序数为10000的测试 18](#_Toc32270)

[4.2 算法性能分析 20](#_Toc7617)

# 1 分析

## 1.1 背景分析

算法是数据结构中不可缺少的部分，将一组数据进行排序是算法中的重要部分，排序有多种算法。按目前看，有冒泡排序，交换排序，直接插入排序，希尔排序，快速排序，堆排序，归并排序， 8种排序方法。我们先输入我们要排序的随机数的个数，再建立一个动态的一位数组进行随机数的储存，通过选择界面来计算不同算法的比较时间和比较次数

## 1.2 功能分析

作为一个简易的算法分析程序，我们通过time（）函数来计算程序运行所需要的时间，用change记录比较交换的次数，选择界面来调用8种不同的算法

# 2 设计

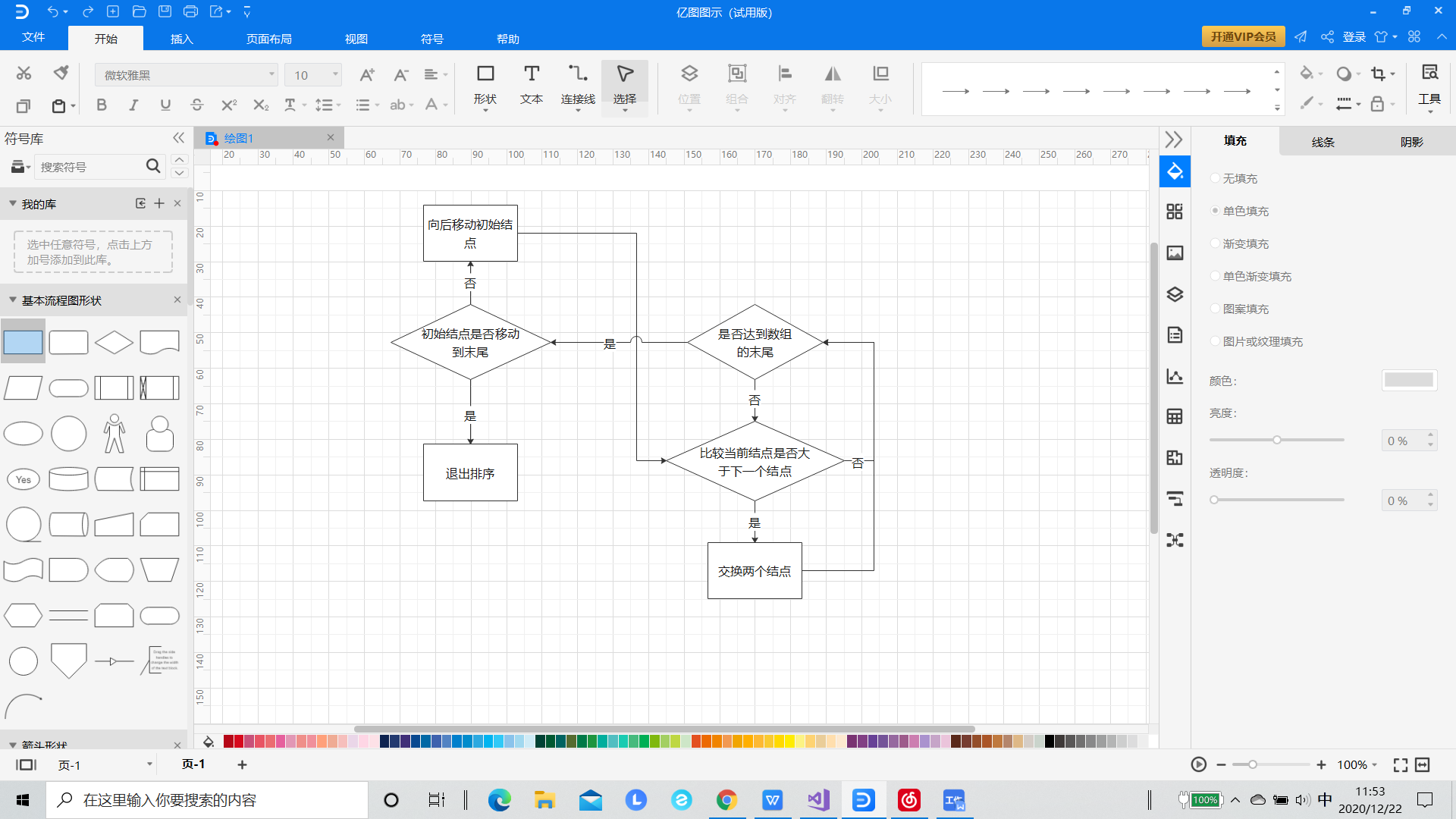
## 2.1 数据结构设计

如上功能分析所述，该系统可以开辟一个动态的一维数组进行数据的储存，在主函数种声明好算法的函数名到时调用。

# 3 实现

## 3.1 冒泡排序算法的实现

### 3.1.1冒泡排序算法流程图



### 3.1.2冒泡排序算法核心代码

void bubblesort(int \*a, int &change, int &time, int number)

{

int timestart = clock();

for (int n=0; n < number; n++)

{

for (int m=0; m < number-1-n; m++)

{

int temp;

if (a[m] > a[m+1])

{

temp = a[m]; a[m] = a[m + 1]; a[m + 1] = temp;

change++;

}

}

}

int timeend = clock();

time = timeend - timestart;

## 3.2 直接插入排序的实现

### 3.2.1 直接插入排序流程图

### 

### 3.2.2 直接插入排序核心代码

void insertsort(int \*a, int &change, int &time, int number)

{

int timestart = clock();

for (int n = 1; n < number; n++)

{

for (int m = 0; m < n; m++)

{

int temp;

if (a[m] > a[n])

{

temp = a[n];

for (int y = n - 1; y >= m; y--)

{

a[y + 1] = a[y];

change++;

}

a[m] = temp;

}

}

}

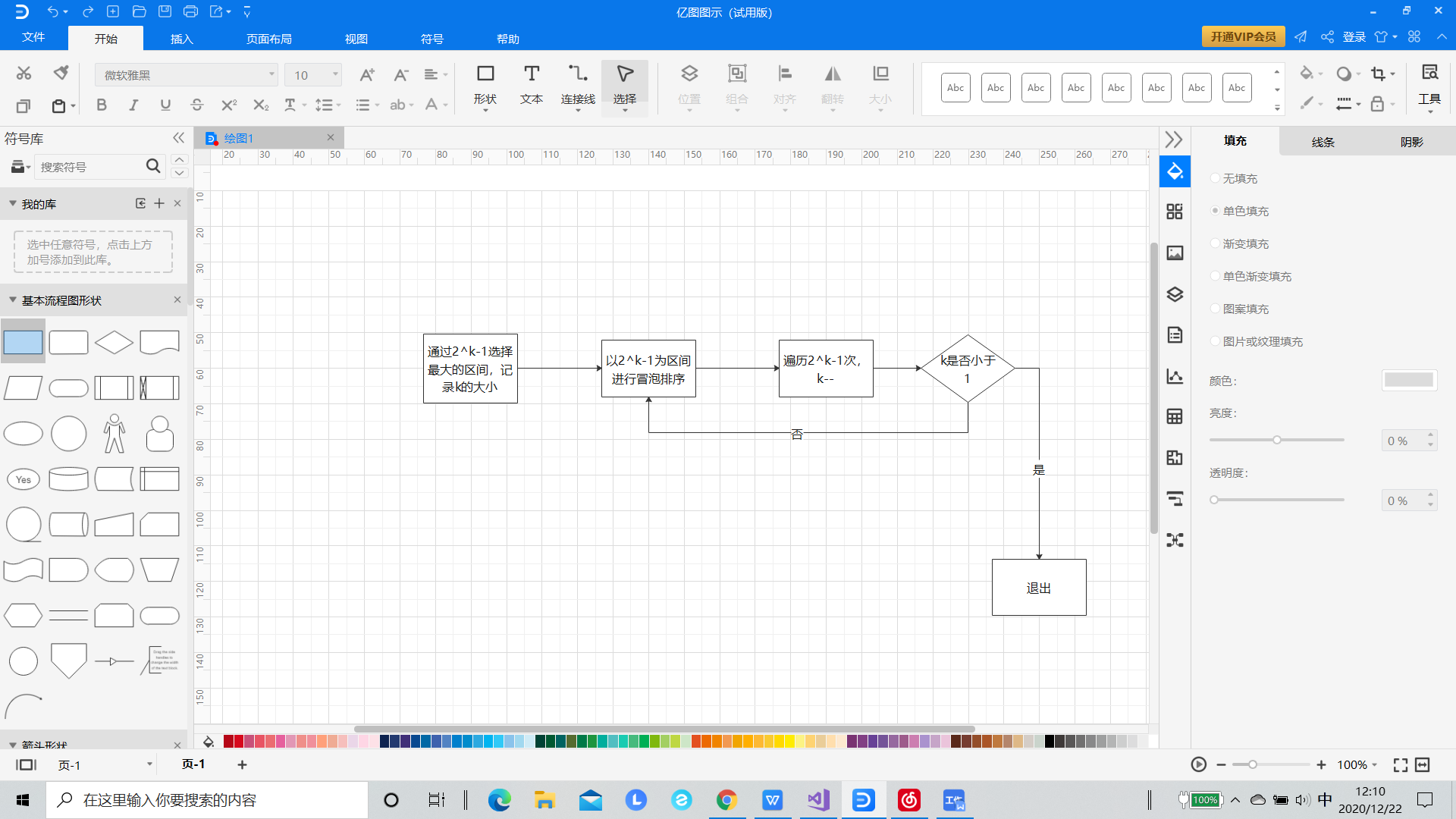
int timeend = clock();

time = timeend - timestart;

}

## 3.3 希尔排序算法的实现

### 3.3.1 希尔排序算法流程图



### 3.3.2 希尔排序算法核心代码

void shellsort(int \*a, int &change, int &time, int number)

{

int k = 1; int delta;

for (;; k++)

{

if (number <= pow(2, k + 1) - 1 && number >= pow(2, k)-1) { break; }

}

int timestart = clock();

for (; k >= 1; k--)

{

delta = pow(2, k) - 1;

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

{

for (int n = 0 + i; n < number; n = n + delta)

{

for (int m = 0 + i; m < number - delta - n; m = m + delta)

{

int temp;

if (a[m] > a[m + delta])

{

temp = a[m]; a[m] = a[m + delta]; a[m + delta] = temp;

change++;

}

}

}

}

}

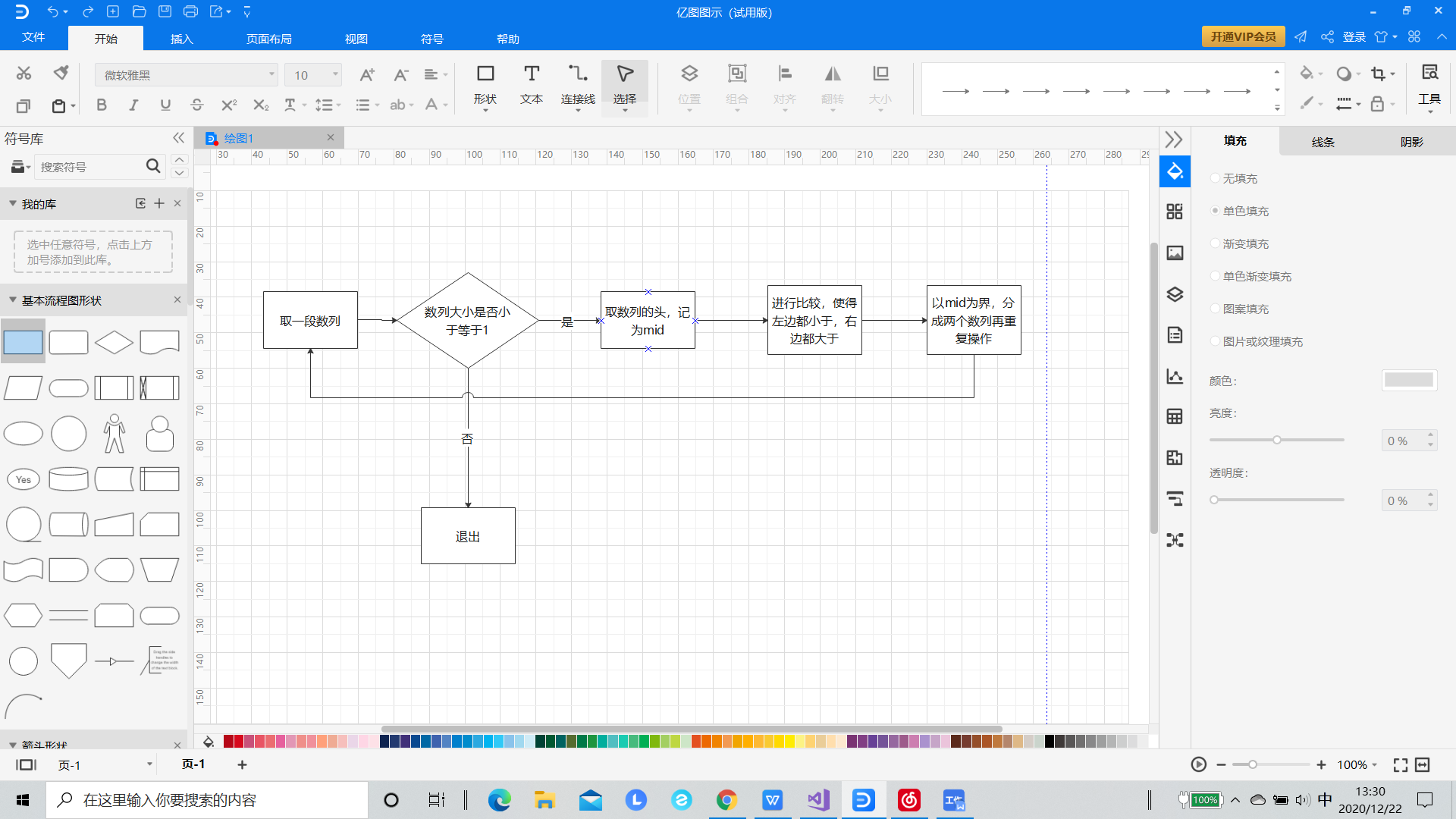
int timeend = clock();

time = timeend - timestart;

}

## 3.4 快速排序算法的实现

### 3.4.1 快速排序算法流程图



### 3.4.2 快速排序算法核心代码

int quickswap(int \*a, int head, int end, int &change) //将一个无序数列的头数据放到中间，左边都小于，右边都大于

{

int x = head; int y = end;

int k = a[head];

while (x < y)

{

for (;x<y; y--)

{

if (k >= a[y]) { a[x] = a[y]; x++; change++; break; }

}

for (; x<y; x++)

{

if (a[x] > k) { a[y] = a[x]; y--; change++; break; }

}

}

a[x] = k;

return x;

}

int quickswap2(int \*a, int head, int end, int &change)

{

if (head<end)

{

int temp = quickswap(a, head, end,change); //递归调用，直到一个数列中只有一个数为止

quickswap2(a, head, temp - 1,change);

quickswap2(a, temp + 1, end,change);

}

else

{

return 0;

}

}

void quicksort(int \*a, int &change, int &time, int number)

{

int head = 0; int end = number-1;

int timestart = clock();

quickswap2(a, head, end,change);

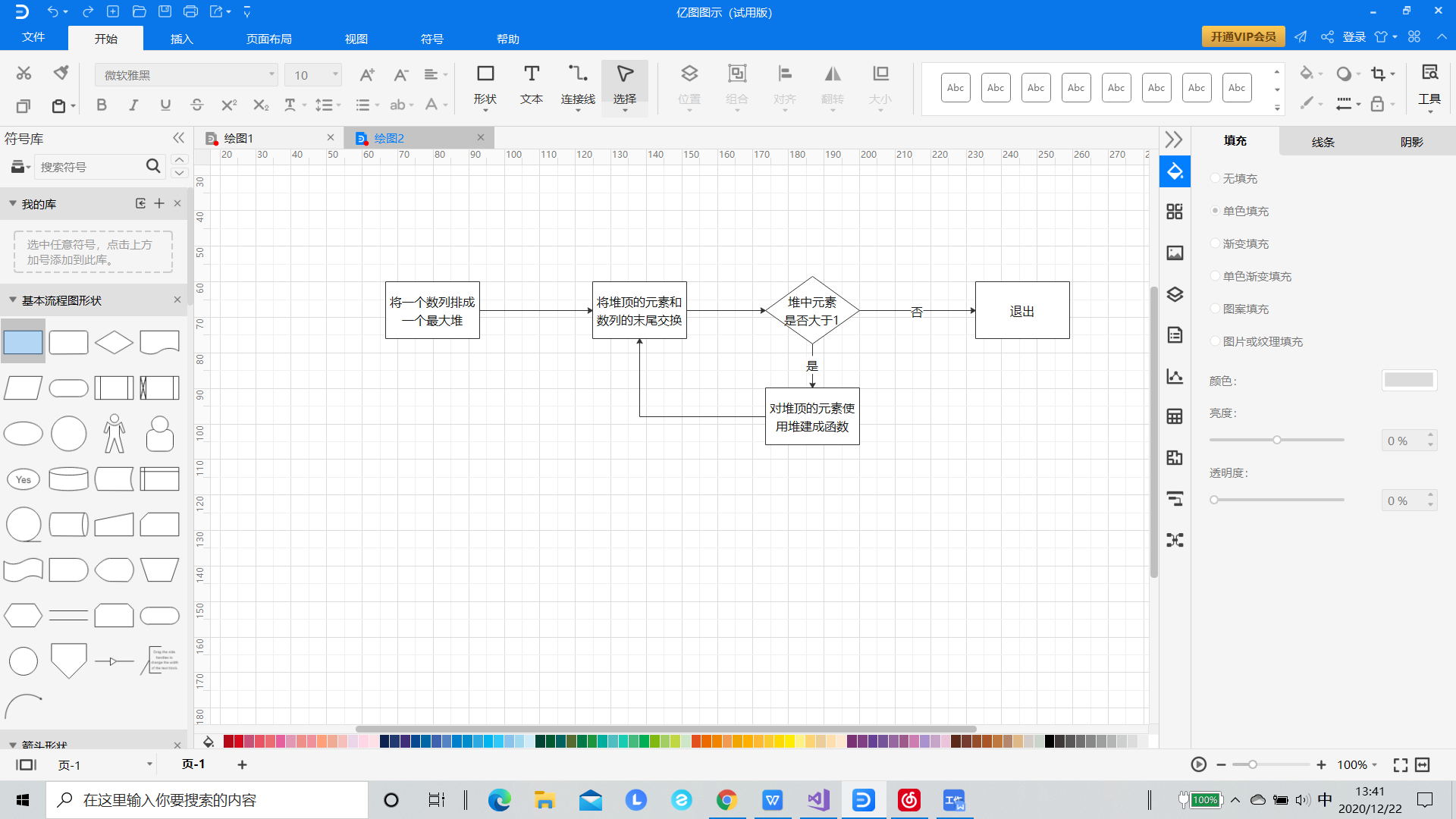
int timeend = clock();

time = timeend - timestart;

}

## 3.5 堆排序算法的实现

### 3.5.1 堆排序算法的流程图



### 3.5.2 堆排序算法的核心代码

先建立一个最大堆函数adjustheap

void adjustheap(int \*&a, int &change, int number, int current)//该点的顶大堆

{

int temp = a[current];

for (int n = 2 \* current + 1; n < number; n = n \* 2 + 1)

{

if (n + 1 < number&&a[n + 1] > a[n]) { n = n+ 1; }

if (a[n] > temp)

{

a[current] = a[n];

current = n; change++;

}

else { break; }

}

a[current] = temp;

}

再进行我们的堆排序算法的编写

void heapsort(int \*a, int &change, int &time, int number)

{

int max = number;

//建立一个顶大堆

int timestart = clock();

for (int current = number / 2 - 1; current >= 0; current--)

{

adjustheap(a, change, number, current);

}

//从堆顶将最大的元素放到最后

for (int n=0;n<number;)

{

int temp = a[number-1];

a[number - 1] = a[0]; change++;

a[0] = temp;

number--;

adjustheap(a, change, number, 0);

}

int timeend = clock();

time = timeend - timestart;

## }

## 3.6 归并排序算法的实现

### 3.6.1 归并排序算法流程图

### 

### 3.6.2 归并排序算法核心代码

void merge(int \*a, int \*b, int &change, int head, int end, int middle)//将一个数列分成两个数列进行比较，链接成一个数列

{

int x = head; int y = middle + 1; int k = head;

//比较排序

while (x<middle+1&&y<end+1)

{

if (a[x] < a[y])

{

b[k] = a[x]; change++;

k++; x++;

}

else

{

b[k] = a[y]; change++;

k++; y++;

}

}

//剩余链接

while (x != middle + 1)

{

b[k] = a[x]; change++;

k++; x++;

}

while (y != end + 1)

{

b[k] = a[y]; change++;

k++; y++;

}

//将整个链赋值回去

for (int n = head; n <= end; n++)

{

a[n] = b[n];

}

}

int mergesort(int \*a, int \*b, int &change, int head,int end)//递归调用

{

int middle;

if (head < end)

{

middle = head + (end-head) / 2;

mergesort(a, b, change, head, middle);

mergesort(a, b, change, middle + 1, end);

merge(a, b, change, head, end, middle);//链接两个有序的数列

return 0;

}

}

void mergesort2(int \*a, int \*b, int &change, int &time, int head, int end)

{

int timestart = clock();

mergesort(a, b, change, head, end);

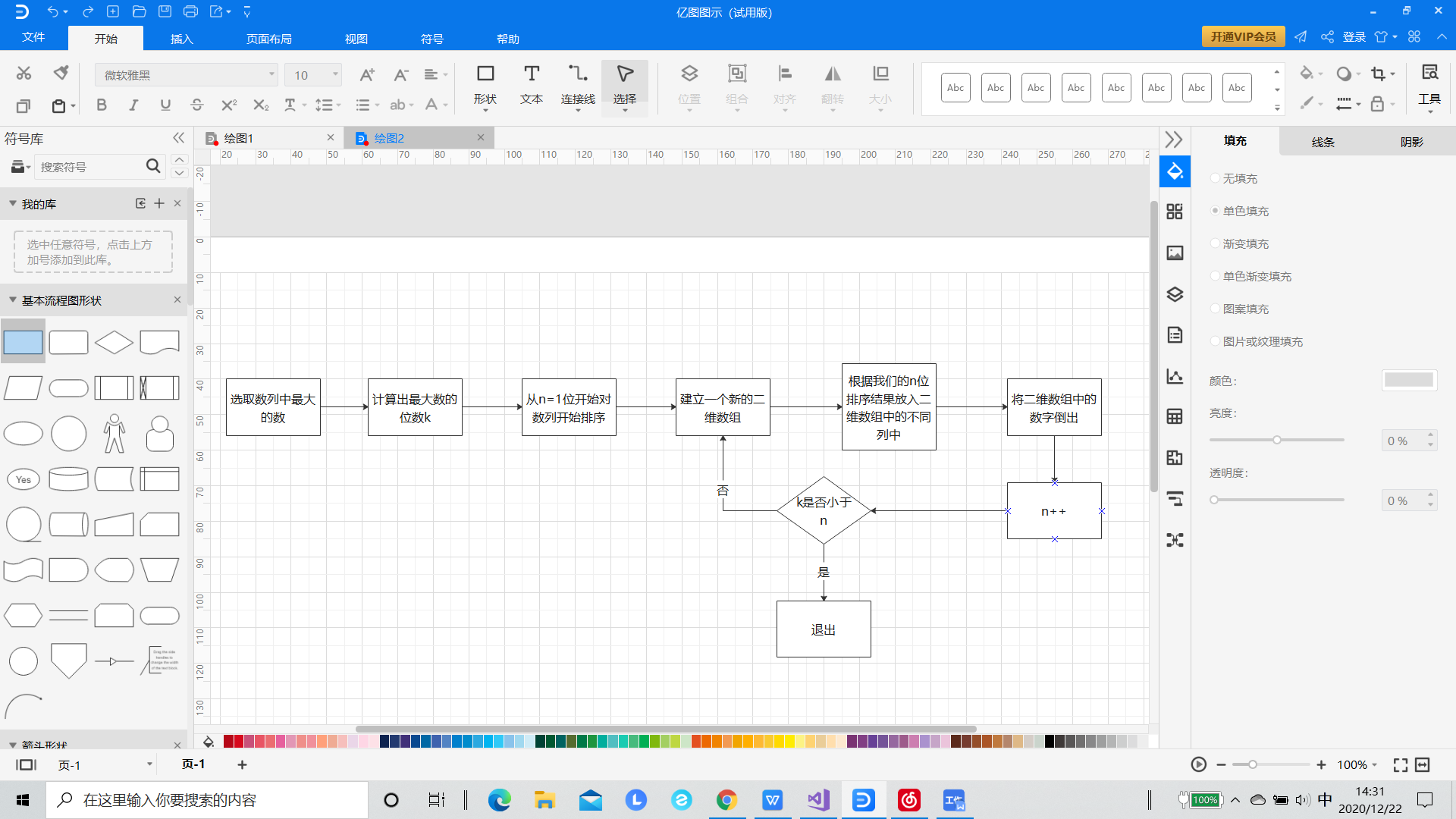
int timeend = clock();

time = timeend - timestart;

# }

# 3.7基数排序算法实现

### 3.7.1 基数排序算法流程图



### 3.7.2基数排序算法核心代码

void bucket(int \*&a, int&change, int loop,int number)

{

int putin[10][10000] = {};

int k = pow(10, loop - 1);

int n, m;

for (n=0; n < number; n++)

{

int row = (a[n] / k) % 10;

for (m=0; m < 10000; m++)

{

if (putin[row][m] == NULL)

{

change++;

putin[row][m] = a[n]; break;

}

}

}

//放回去

int temp = 0;

for (n = 0; n < 10; n++)

{

for (m = 0; m < 10000; m++)

{

if (putin[n][m] == NULL)

{

break;

}

a[temp] = putin[n][m]; temp++; change++;

}

}

}

void bucketsort(int \*a, int &change, int &time, int number)

{

int max; int loop;

max = 0; loop = 1;

int timestart = clock();

for (int n = 0; n < number; n++)

{

if (a[n] > max) { max = a[n]; }

}

for (;;)

{

max = max / 10;

if (max > 0)

{

loop++;

}

else { break; }

}

for (int n=1;n<=loop;n++)

{

bucket(a, change,n,number);

}

int timeend = clock();

time = timeend - timestart;

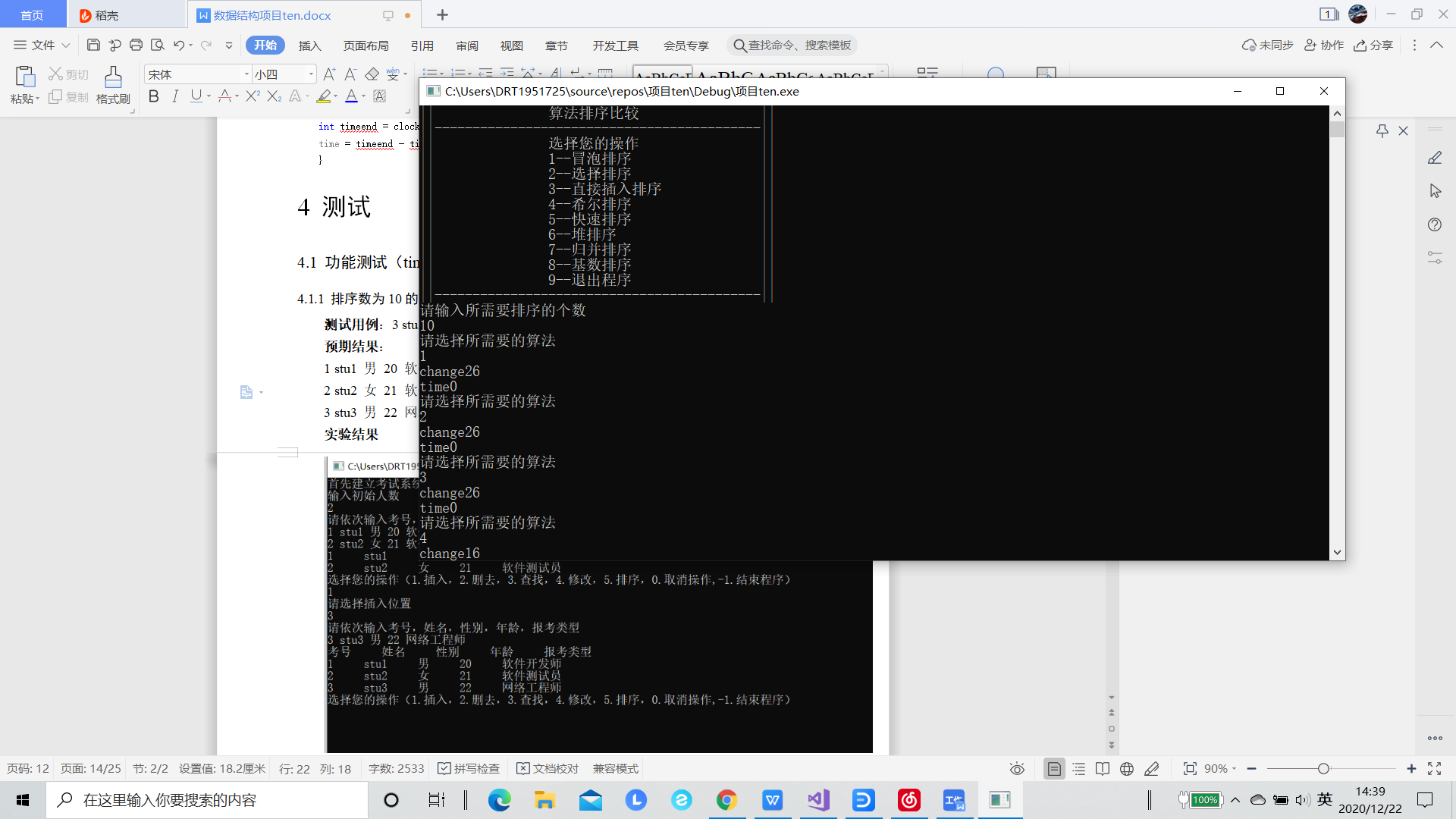
}

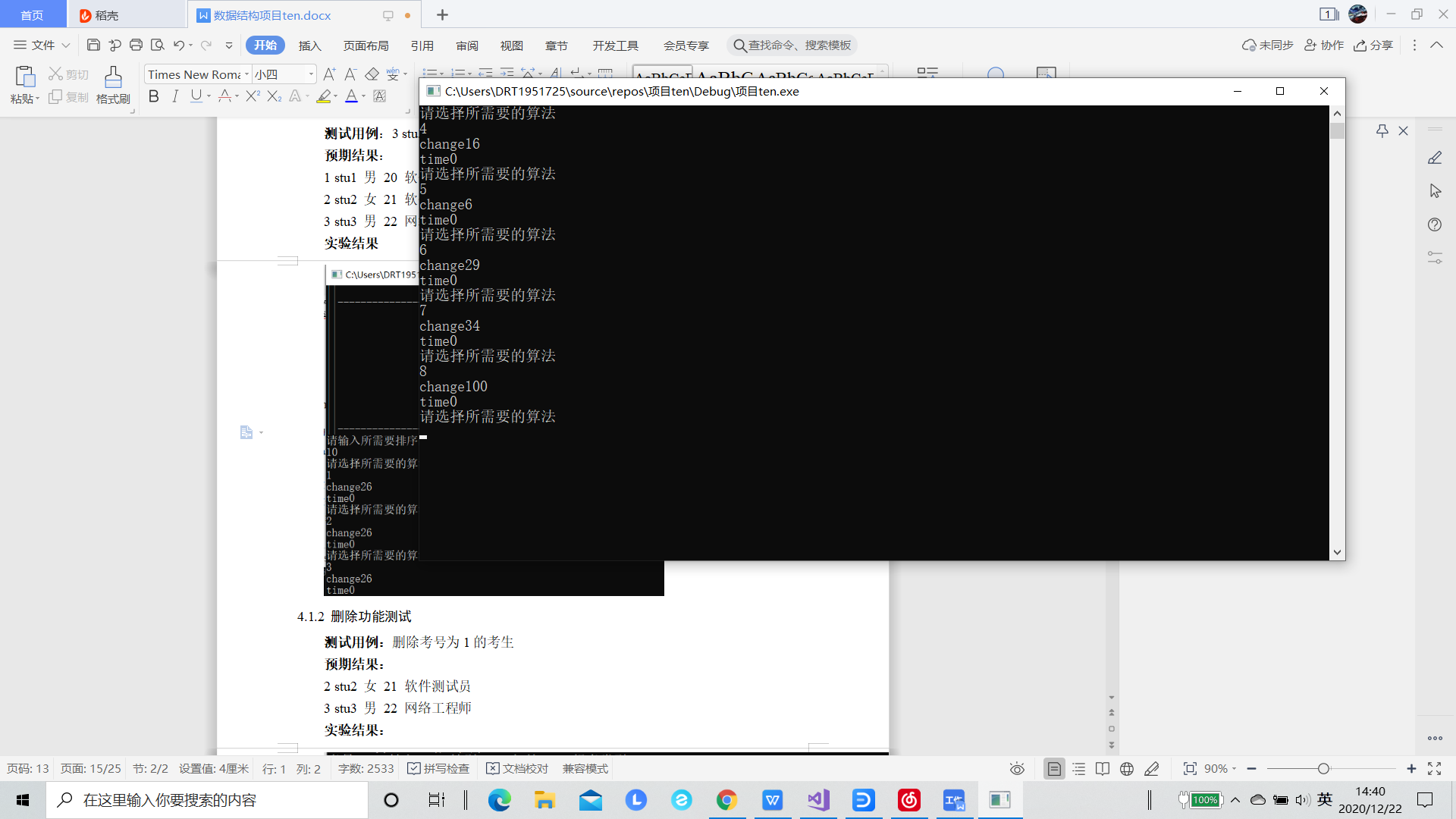
# 4 测试

## 4.1 功能测试（time函数放大1000）

### 4.1.1 排序数为10的测试

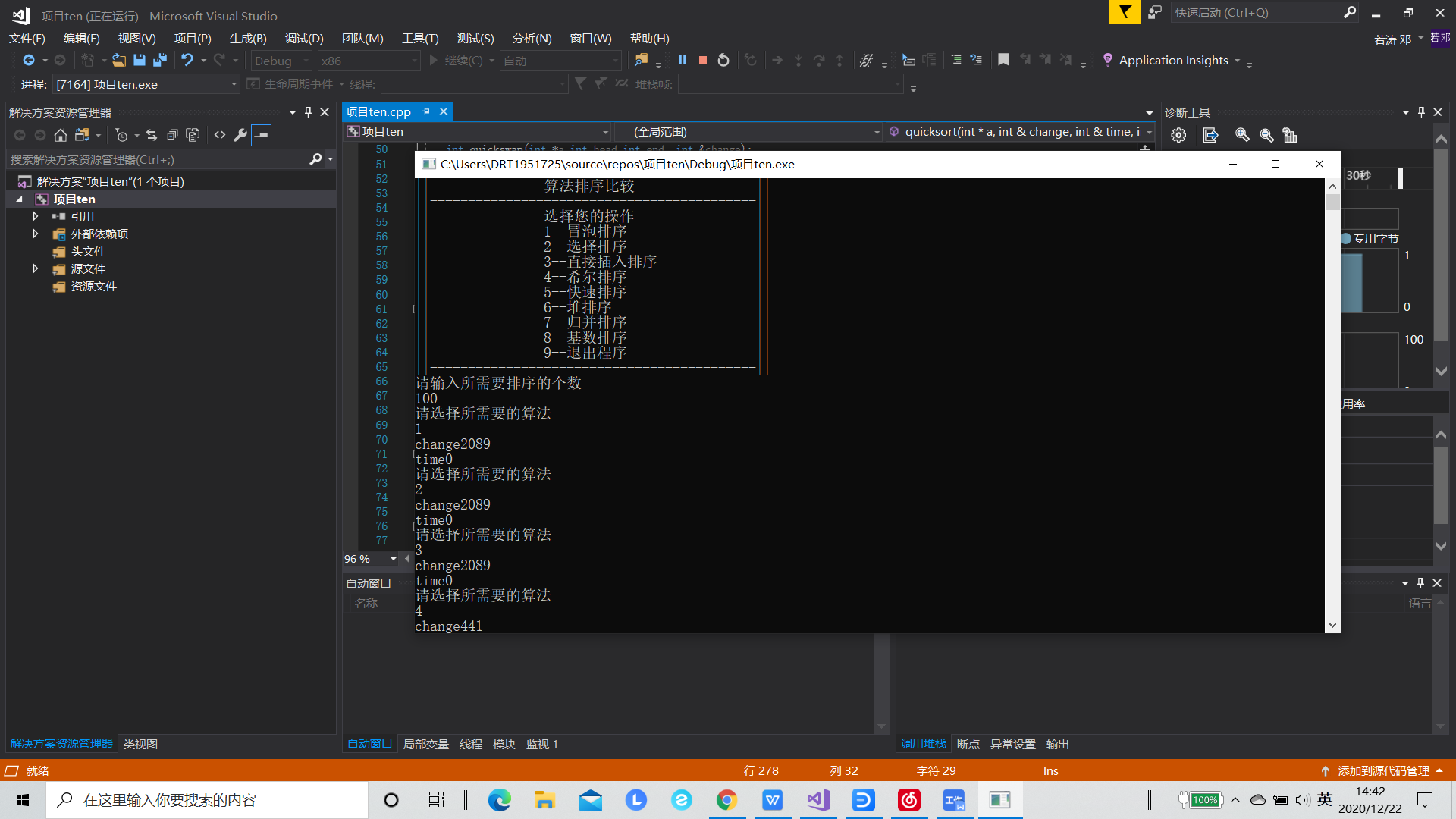
**实验结果**

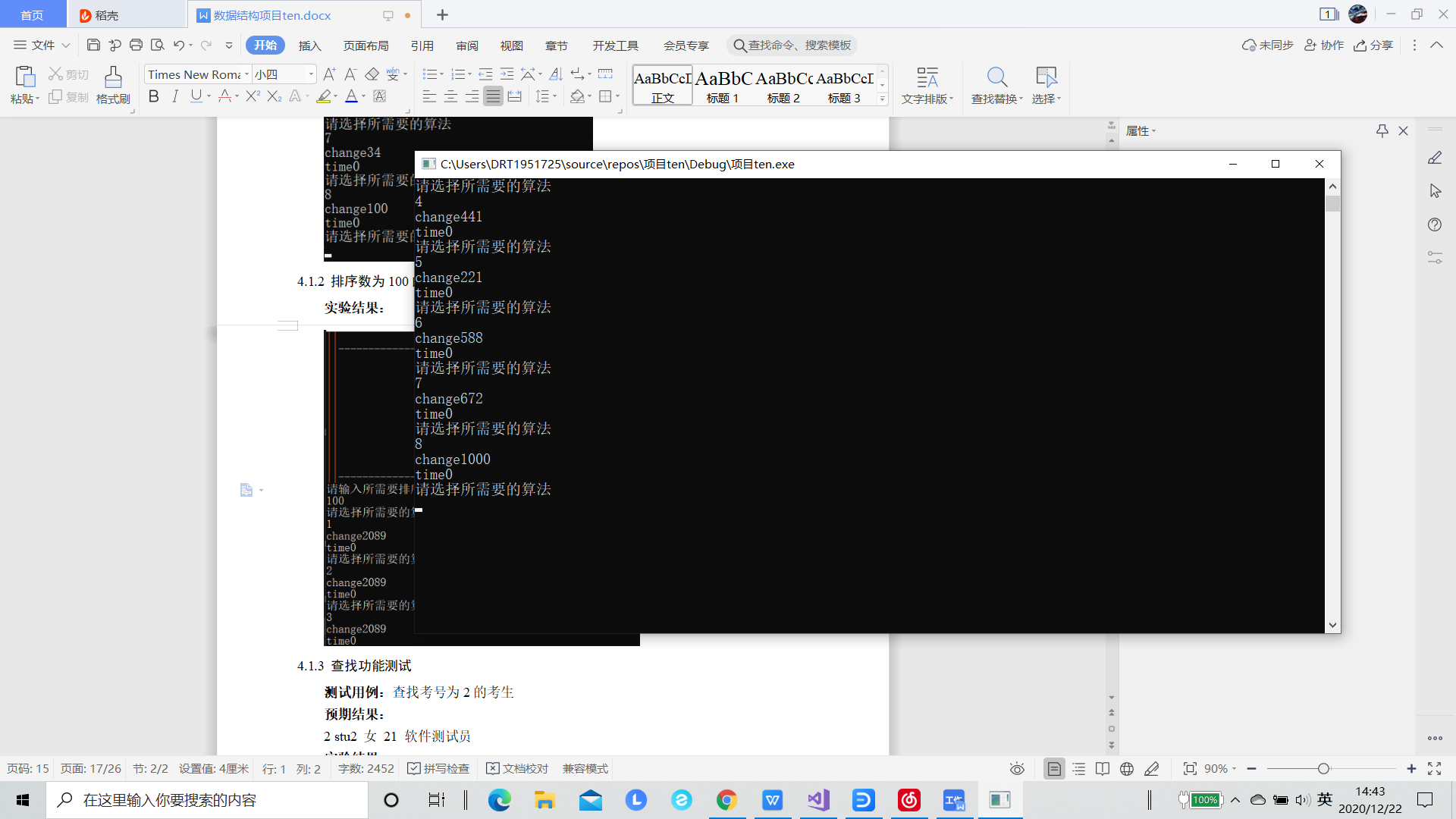




### 4.1.2 排序数为100的测试

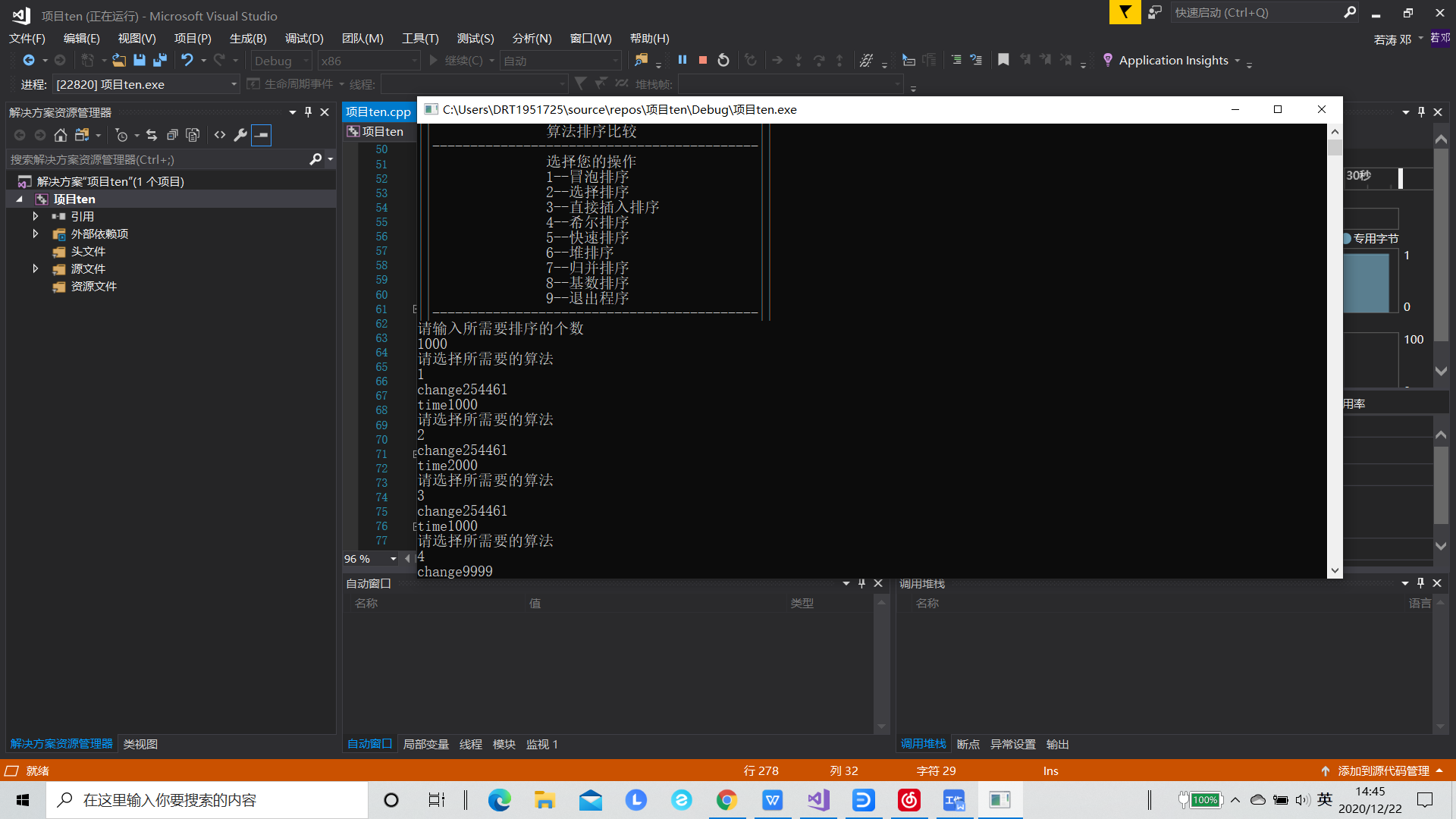
**实验结果：**

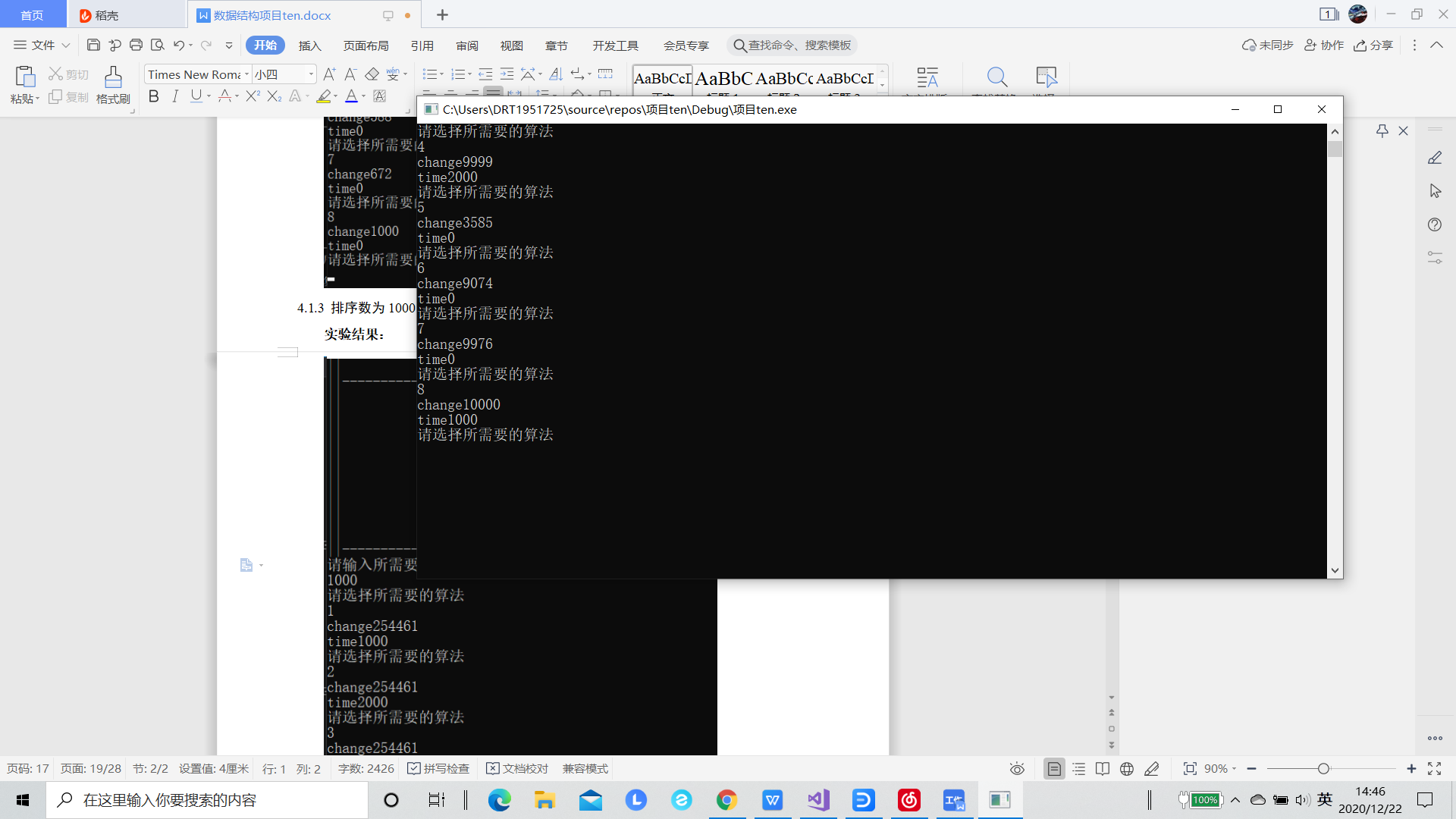




### 4.1.3 排序数为1000的测试

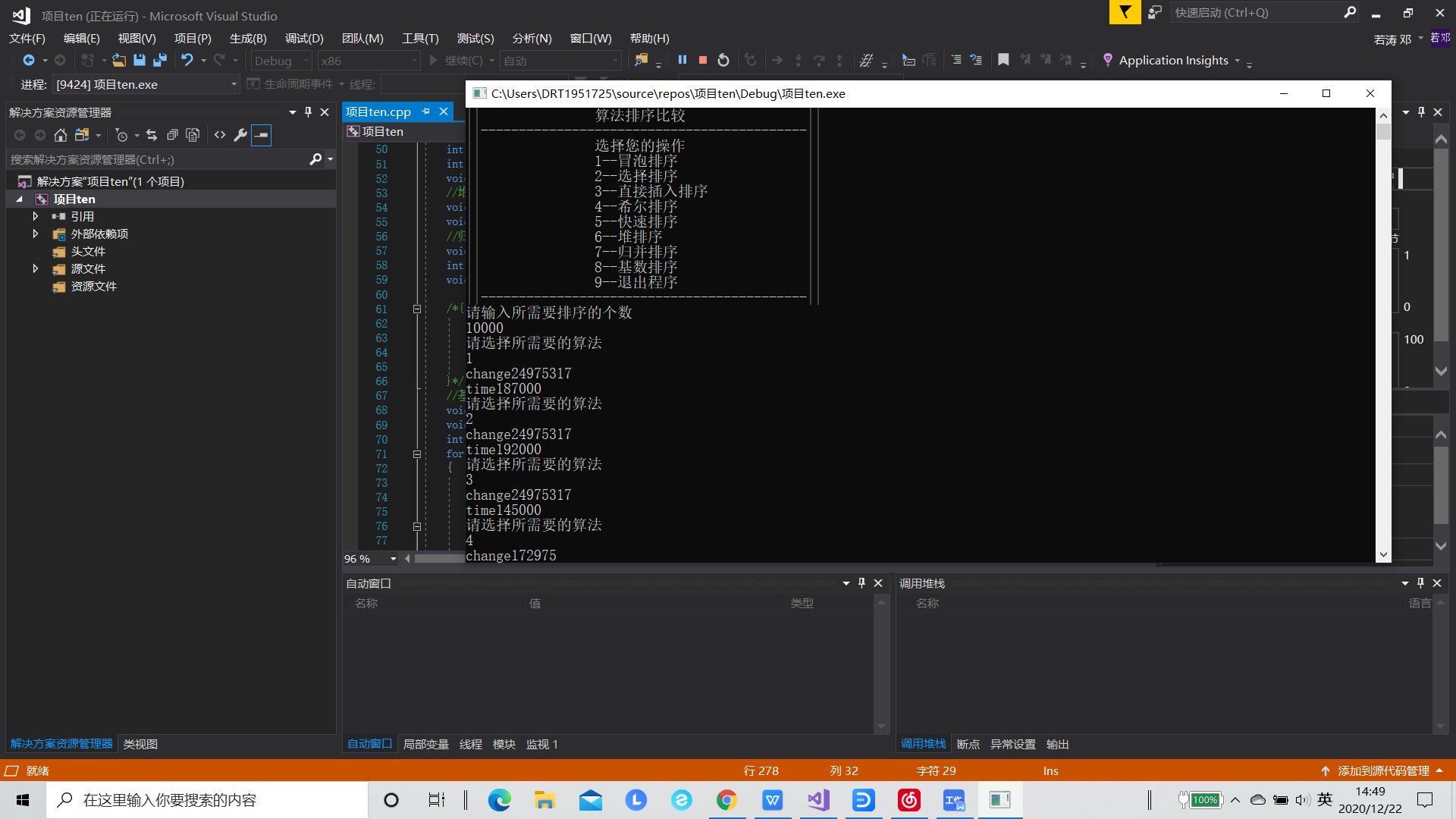
**实验结果：**





### 4.1.4 排序数为10000的测试

**实验结果：**



### 

## 4.2 算法性能分析

从上述的实验结果不难看出，冒泡排序，选择排序，直接插入排序，三种排序方法代码简单方便，但遇到较大的数时比较次数极多，时间使用长，希尔排序比较上述三种算法数据的交换次数要少点，但对于用时来说并没有加快。快速排序是遇到一个大数的最优选择，它的比较次数少，而且用时最短。如果担心快速排序总是变成交换排序，可以使用堆排序和归并排序，两者虽然比起快速排序的比较次数要更多，但是时间的复杂度较为稳定。基数排序比较特殊，由于要不断地输入数据到桶中，并且再次倒出来，它的交换次数是固定的，其用时间也介于几种简单排序和复杂排序当中。