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

作 者 姓 名： 杨煜

学 号： 1850217

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc26282589)

[1.1 项目简介 1](#_Toc26282590)

[1.2 项目要求 1](#_Toc26282591)

[2 实现 1](#_Toc26282592)

[2.1 冒泡排序的实现 1](#_Toc26282593)

[2.1.1 冒泡排序 1](#_Toc26282594)

[2.1.2 冒泡排序核心代码 1](#_Toc26282595)

[2.2 选择排序的实现 2](#_Toc26282596)

[2.2.1 选择排序 2](#_Toc26282597)

[2.2.2 选择排序核心代码 2](#_Toc26282598)

[2.3 直接插入排序的实现 3](#_Toc26282599)

[2.3.1 直接插入排序 3](#_Toc26282600)

[2.3.2 直接插入排序核心代码 3](#_Toc26282601)

[2.4 希尔排序的实现 4](#_Toc26282602)

[2.4.1 希尔排序 4](#_Toc26282603)

[2.4.2 希尔排序核心代码 4](#_Toc26282604)

[2.5 快速排序的实现 5](#_Toc26282605)

[2.5.1 快速排序 5](#_Toc26282606)

[2.5.2 快速排序核心代码 5](#_Toc26282607)

[2.6堆排序的实现 6](#_Toc26282608)

[2.6.1 堆排序 6](#_Toc26282609)

[2.6.2 堆排序核心代码 6](#_Toc26282610)

[2.7 归并排序的实现 9](#_Toc26282611)

[2.7.1 归并排序 9](#_Toc26282612)

[2.7.2 归并排序核心代码 9](#_Toc26282613)

[2.8 基数排序的实现 10](#_Toc26282614)

[2.8.1 基数排序 10](#_Toc26282615)

[2.8.2 基数排序核心代码 10](#_Toc26282616)

[2.9 总体功能的实现 12](#_Toc26282617)

[2.9.1 总体功能 12](#_Toc26282618)

[2.9.2 总体功能核心代码 12](#_Toc26282619)

[3测试 16](#_Toc26282620)

[3.1 100个随机数 16](#_Toc26282621)

[3.2 1000个随机数 17](#_Toc26282622)

[3. 3 10000个随机数 18](#_Toc26282623)

[3. 4 100000个随机数 19](#_Toc26282624)

# 1 分析

## 1.1 项目简介

随机函数产生一百，一千，一万和十万个随机数，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

## 项目要求

请在文档中记录上述数据量下，各种排序的计算时间和存储开销，并且根据实验结果说明这些方法的优缺点。

# 2 实现

## 2.1 冒泡排序的实现

### 2.1.1 冒泡排序

比较相邻的元素。如果第一个比第二个大，就交换他们两个。对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。针对所有的元素重复以上的步骤，除了最后一个。持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

### 2.1.2 冒泡排序核心代码

void BubbleSort(int\* B, int n, long long& compare\_times) {

int temp;

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

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

compare\_times++;

if (B[j] > B[j + 1]) {

temp = B[j];

B[j] = B[j + 1];

B[j + 1] = temp;

}

}

}

}

## 2.2 选择排序的实现

### 2.2.1 选择排序

第一次从R[0]~R[n-1]中选取最小值，与R[0]交换，第二次从R[1]~R[n-1]中选取最小值，与R[1]交换，....，第i次从R[i-1]~R[n-1]中选取最小值，与R[i-1]交换，.....，第n-1次从R[n-2]~R[n-1]中选取最小值，与R[n-2]交换，总共通过n-1次，得到一个按排序码从小到大排列的有序序列。

### 2.2.2 选择排序核心代码

void SelectSort(int\* B, int n, long long& swap\_times)

{

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

{

int min = i;

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

{

if (B[j] < B[min])

{

min = j;

}

}

if (min != i)

{

swap(B[i], B[min]);

swap\_times++;

}

}

}

## 2.3 直接插入排序的实现

### 2.3.1 直接插入排序

当插入第i(i >= 1)时，前面的R[0]，R[1]，……，R[i-1]已经排好序。这时，R[i]的排序码与R[i-1]，r[i-2]，…的排序码顺序进行比较，找到插入位置即将R[i]插入，原来位置上的元素向后顺移。

### 2.3.2 直接插入排序核心代码

void InsertSort(int\* B, int n, long long& swap\_times) {

int temp;

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

{

if (B[i] < B[i - 1])

{

temp = B[i];

int j = i - 1;

do

{

B[j + 1] = B[j];

swap\_times++;

j--;

} while (j >= 0 && temp < B[j]);

B[j + 1] = temp;

swap\_times++;

}

}

}

## 2.4 希尔排序的实现

### 2.4.1 希尔排序

先取一个小于n的整数d1作为第一个增量，把文件的全部记录分组。所有距离为d1的倍数的记录放在同一个组中。先在各组内进行直接插入排序；然后，取第二个增量d2<d1重复上述的分组和排序，直至所取的增量 =1( < …<d2<d1)，即所有记录放在同一组中进行直接插入排序为止。

### 2.4.2 希尔排序核心代码

void ShellSort(int\* B, int n, long long& swap\_times)

{

int gap;

for (gap = 3; gap > 0; gap--)

{

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

{

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

{

if (B[j] < B[j - gap])

{

int temp = B[j];

int k = j - gap;

while (k >= 0 && B[k] > temp)

{

B[k + gap] = B[k];

swap\_times++;

k = k - gap;

}

B[k + gap] = temp;

swap\_times++;

}

}

}

}

}

## 2.5 快速排序的实现

### 2.5.1 快速排序

首先设定一个分界值，通过该分界值将数组分成左右两部分。将大于或等于分界值的数据集中到数组右边，小于分界值的数据集中到数组的左边。此时，左边部分中各元素都小于或等于分界值，而右边部分中各元素都大于或等于分界值。 然后，左边和右边的数据可以独立排序。对于左侧的数组数据，又可以取一个分界值，将该部分数据分成左右两部分，同样在左边放置较小值，右边放置较大值。右侧的数组数据也可以做类似处理。重复上述过程，可以看出，这是一个递归定义。通过递归将左侧部分排好序后，再递归排好右侧部分的顺序。当左、右两个部分各数据排序完成后，整个数组的排序也就完成了。

### 2.5.2 快速排序核心代码

void QuickSort(int\* B, const int left, const int right, long long& swap\_times)

{

if (left < right) {

int pivotPos = left;

int pivot = B[left];

for (int i = left + 1; i <= right; i++) {

swap\_times++;

if (B[i] < pivot) {

pivotPos++;

if (pivotPos != i) {

swap(B[pivotPos], B[i]);

}

}

}

B[left] = B[pivotPos];

B[pivotPos] = pivot;

QuickSort(B, left, pivotPos - 1, swap\_times);

QuickSort(B, pivotPos + 1, right, swap\_times);

}

}

## 2.6堆排序的实现

### 2.6.1 堆排序

利用堆的插入，移除，进行排序。

### 2.6.2 堆排序核心代码

class Heap

{

public:

Heap(int size);

Heap(int \*A, int size, long long& swap\_times);

~Heap() { delete[]heap; }

void siftDown(int start, int end, long long& swap\_times);

int length() { return currentSize; }

bool isEmpty() { return currentSize == 0; }

void print();

void swap(int m, int n);

private:

int\* heap;

int currentSize;

};

Heap::Heap(int size) {

heap = new int[size];

if (heap == NULL) {

cerr << "内存分配失败" << endl;

exit(1);

}

currentSize = 0;

}

Heap::Heap(int\* A, int size, long long& swap\_times) {

heap = new int[size];

if (heap == NULL) {

cerr << "内存分配失败" << endl;

exit(1);

}

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

heap[i] = A[i];

}

currentSize = size;

int currentPos = (currentSize - 2) / 2;

while (currentPos >= 0) {

siftDown(currentPos, currentSize - 1, swap\_times);

currentPos--;

}

}

void Heap::siftDown(int start, int end, long long& swap\_times) {

int temp = heap[start];

int i = start;

int j = 2 \* i + 1;

while (j <= end) {

swap\_times++;

if (heap[j] < heap[j + 1] && j < end)

j++;

swap\_times++;

if (temp >= heap[j])

break;

else {

heap[i] = heap[j];

i = j;

j = 2 \* j + 1;

}

}

heap[i] = temp;

}

void Heap::swap(int m, int n) {

int temp = heap[m];

heap[m] = heap[n];

heap[n] = temp;

}

void Heap::print() {

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

cout << heap[i] << ' ';

}

}

void HeapSort(int\* B, int size, long long& swap\_times) {

Heap maxHeap = Heap(B, size, swap\_times);

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

maxHeap.swap(0, i);

swap\_times++;

maxHeap.siftDown(0, i - 1, swap\_times);

}

}

## 2.7 归并排序的实现

### 2.7.1 归并排序

将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。

### 2.7.2 归并排序核心代码

void Merge(int\* L1, int\* L2, const int left, const int mid, const int right, long long& compare\_times) {

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

L2[i] = L1[i];

}

int s1 = left;

int s2 = mid + 1;

int t = left;

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

compare\_times++;

if (L2[s1] <= L2[s2]) {

L1[t++] = L2[s1++];

}

else {

L1[t++] = L2[s2++];

}

}

while (s1 <= mid) {

compare\_times++;

L1[t++] = L2[s1++];

}

while (s2 <= right) {

compare\_times++;

L1[t++] = L2[s2++];

}

}

void MergeSort(int\* L1, int\* L2, int left, int right, long long& compare\_times) {

if (left < right) {

int mid = (left + right) / 2;

MergeSort(L1, L2, left, mid, compare\_times);

MergeSort(L1, L2, mid + 1, right, compare\_times);

Merge(L1, L2, left, mid, right, compare\_times);

}

}

## 2.8 基数排序的实现

### 2.8.1 基数排序

设置若干个桶，依次扫描待排序的记录R[0]，R[1]，…，R[n-1]，把关键字在某个范围内的记录全都装入到第k个桶里(分配)，然后按序号依次将各非空的桶首尾连接起来(收集)。

### 2.8.2 基数排序核心代码

int MaxBit(int\* B, int n) {

int max = B[0];

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

if (max < B[i]) {

max = B[i];

}

}

int d = 1;

int p = 10;

while (max >= p) {

max = max / 10;

d++;

}

return d;

}

void RadixSort( int\* B, int size, long long& swap\_times) {

int d = MaxBit(B, size);

int\* temp = new int[size];

int count[10];

int radix = 1;

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

for (int j = 0; j < 10; j++) {

count[j] = 0;

}

for (int j = 0; j < size; j++) {

int k = (B[j] / radix) % 10;

count[k]++;

}

for (int j = 1; j < 10; j++) {

count[j] = count[j - 1] + count[j];

}

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

int k = (B[j] / radix) % 10;

temp[count[k] - 1] = B[j];

count[k]--;

}

for (int j = 0; j < size; j++) {

B[j] = temp[j];

}

radix \*= 10;

}

}

## 2.9 总体功能的实现

### 2.9.1 总体功能

首先建立输出的表格，随后依据输入内容，使用switch，分别进行相应的函数调用。

### 2.9.2 总体功能核心代码

int main()

{

cout << "\*\*" << setw(15) << ' ' << "排序算法比较" << setw(15) << ' ' << "\*\*" << endl;

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

cout << "\*\*" << setw(14) << ' ' << "1 --- 冒泡排序" << setw(14) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(14) << ' ' << "2 --- 选择排序" << setw(14) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(14) << ' ' << "3 --- 直接插入排序" << setw(10) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(14) << ' ' << "4 --- 希尔排序" << setw(14) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(14) << ' ' << "5 --- 快速排序" << setw(14) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(14) << ' ' << "6 --- 堆排序" << setw(16) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(14) << ' ' << "7 --- 归并排序" << setw(14) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(14) << ' ' << "8 --- 基数排序" << setw(14) << ' ' << "\*\*" << endl;

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

long num;

cout << "请输入要产生随机数的个数: ";

cin >> num;

srand((unsigned)time(NULL));

int\* A = new int[num];

int\* B = new int[num];

int\* C = new int[num];

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

int item = rand();

A[i]=item;

}

long long swap\_times, compare\_times;

long start, stop;

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

B[i]=A[i];

}

swap\_times = 0;

start = clock();

BubbleSort(B, num, swap\_times);

stop = clock();

cout << "冒泡排序所用时间：" << (stop - start) << "ms" << endl;

cout << "冒泡排序交换次数：" << swap\_times << endl;

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

B[i] = A[i];

}

swap\_times = 0;

start = clock();

SelectSort(B, num, swap\_times);

stop = clock();

cout << "选择排序所用时间：" << (stop - start) << "ms" << endl;

cout << "选择排序交换次数：" << swap\_times << endl;

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

B[i] = A[i];

}

swap\_times = 0;

start = clock();

InsertSort(B, num, swap\_times);

stop = clock();

cout << "直接插入排序所用时间：" << (stop - start) << "ms" << endl;

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

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

B[i] = A[i];

}

swap\_times = 0;

start = clock();

ShellSort(B, num, swap\_times);

stop = clock();

cout << "希尔排序所用时间：" << (stop - start) << "ms" << endl;

cout << "希尔排序交换次数：" << swap\_times << endl;

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

B[i] = A[i];

}

swap\_times = 0;

start = clock();

QuickSort(B, 0, num - 1, swap\_times);

stop = clock();

cout << "快速排序所用时间：" << (stop - start) << "ms" << endl;

cout << "快速排序交换次数：" << swap\_times << endl;

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

B[i] = A[i];

}

swap\_times = 0;

start = clock();

HeapSort(B, num, swap\_times);

stop = clock();

cout << "堆排序所用时间：" << (stop - start) << "ms" << endl;

cout << "堆排序交换次数：" << swap\_times << endl;

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

B[i] = A[i];

}

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

C[i] = A[i];

}

compare\_times = 0;

start = clock();

MergeSort(B, C, 0, num - 1, compare\_times);

stop = clock();

cout << "归并排序所用时间：" << (stop - start) << "ms" << endl;

cout << "归并排序比较次数：" << compare\_times << endl;

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

B[i] = A[i];

}

swap\_times = 0;

start = clock();

RadixSort(B, num, compare\_times);

stop = clock();

cout << "基数排序所用时间：" << (stop - start) << "ms" << endl;

cout << "基数排序交换次数：" << swap\_times << endl;

delete []A;

delete []B;

delete []C;

return 0;

}

# 3测试

## 3.1 100个随机数

**实验结果：**

**冒泡排序所用时间：0ms**

**冒泡排序交换次数：4950**

**选择排序所用时间：0ms**

**选择排序交换次数：93**

**直接插入排序所用时间：0ms**

**直接插入排序交换次数：2601**

**希尔排序所用时间：0ms**

**希尔排序交换次数：1134**

**快速排序所用时间：0ms**

**快速排序交换次数：619**

**堆排序所用时间：0ms**

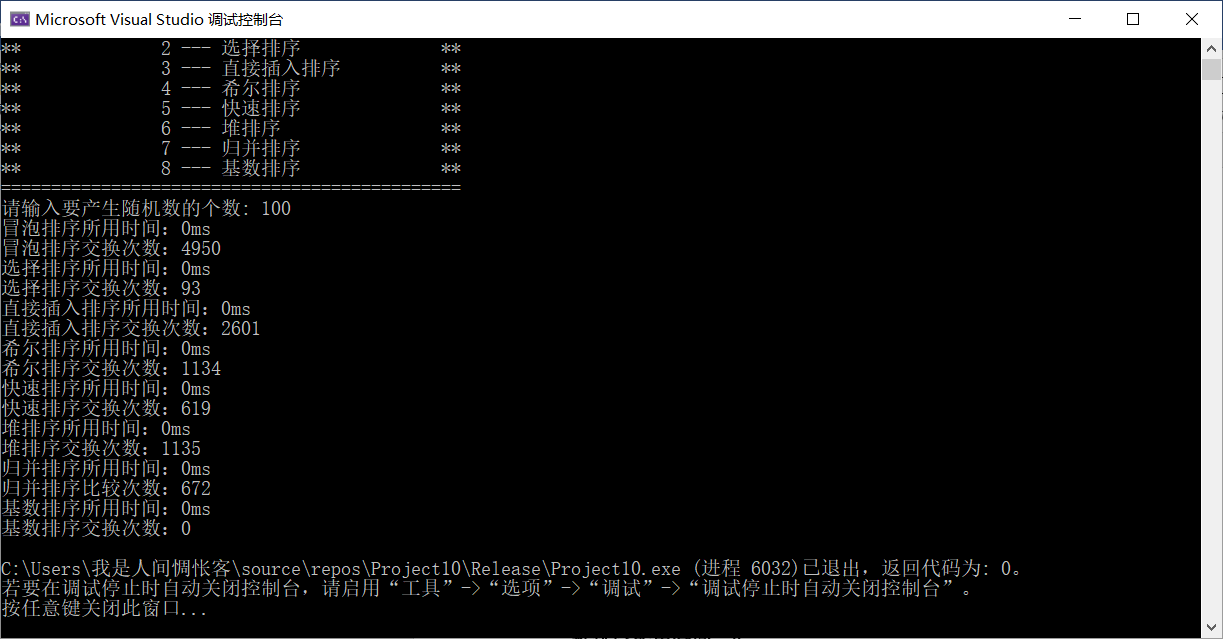
**堆排序交换次数：1135**

**归并排序所用时间：0ms**

**归并排序比较次数：672**

**基数排序所用时间：0ms**

**基数排序交换次数：0**



## 3.2 1000个随机数

**实验结果：**

**冒泡排序所用时间：2ms**

**冒泡排序交换次数：499500**

**选择排序所用时间：1ms**

**选择排序交换次数：992**

**直接插入排序所用时间：1ms**

**直接插入排序交换次数：247855**

**希尔排序所用时间：1ms**

**希尔排序交换次数：88012**

**快速排序所用时间：0ms**

**快速排序交换次数：10942**

**堆排序所用时间：0ms**

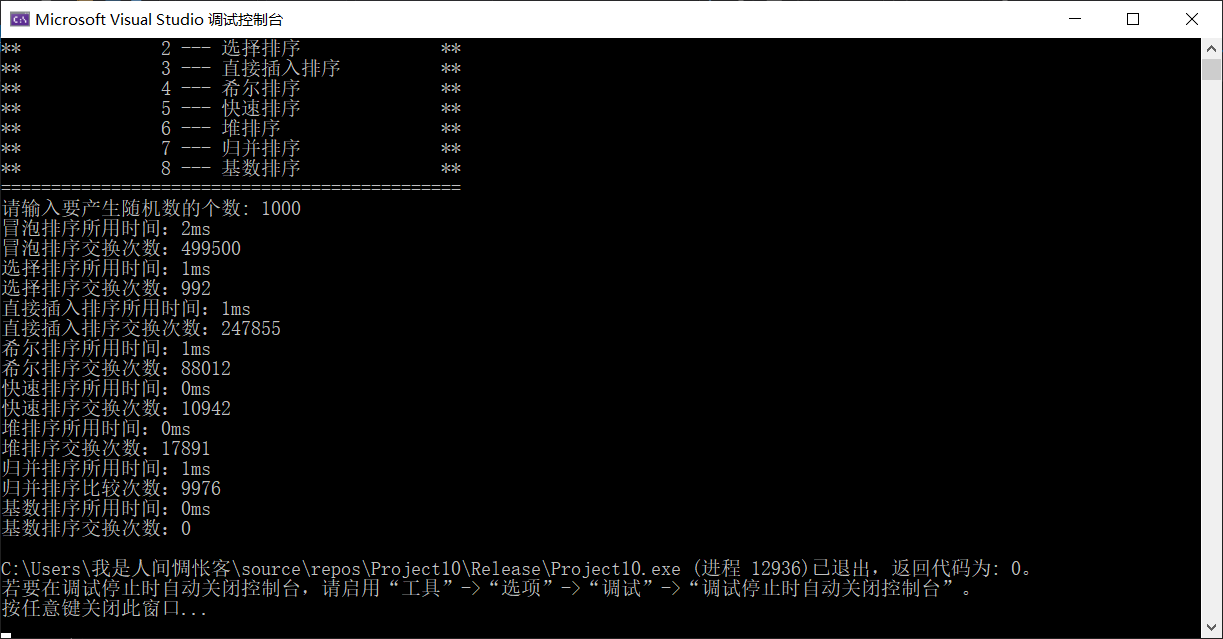
**堆排序交换次数：17891**

**归并排序所用时间：1ms**

**归并排序比较次数：9976**

**基数排序所用时间：0ms**

**基数排序交换次数：0**



## 3. 3 10000个随机数

**实验结果：**

**冒泡排序所用时间：146ms**

**冒泡排序交换次数：49995000**

**选择排序所用时间：58ms**

**选择排序交换次数：9988**

**直接插入排序所用时间：19ms**

**直接插入排序交换次数：24964093**

**希尔排序所用时间：15ms**

**希尔排序交换次数：8436005**

**快速排序所用时间：2ms**

**快速排序交换次数：149288**

**堆排序所用时间：4ms**

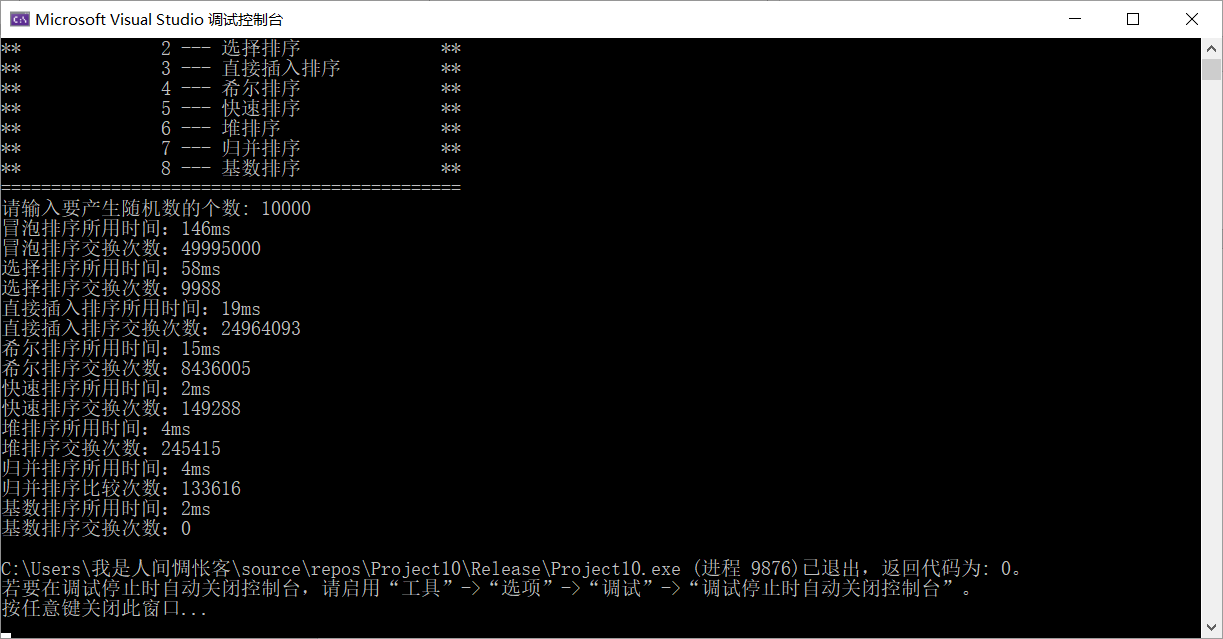
**堆排序交换次数：245415**

**归并排序所用时间：4ms**

**归并排序比较次数：133616**

**基数排序所用时间：2ms**

**基数排序交换次数：0**



## 3. 4 100000个随机数

**实验结果：**

**冒泡排序所用时间：14921ms**

**冒泡排序交换次数：4999950000**

**选择排序所用时间：5392ms**

**选择排序交换次数：99995**

**直接插入排序所用时间：1555ms**

**直接插入排序交换次数：2503382173**

**希尔排序所用时间：610ms**

**希尔排序交换次数：837195145**

**快速排序所用时间：9ms**

**快速排序交换次数：2029361**

**堆排序所用时间：14ms**

**堆排序交换次数：3119037**

**归并排序所用时间：11ms**

**归并排序比较次数：1668928**

**基数排序所用时间：5ms**

**基数排序交换次数：0**

