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： 祝新元

学 号： 1751629

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](#_Toc533792363)

[1.1 项目简介 1](#_Toc533792364)

[2 设计 2](#_Toc533792365)

[2.1 主程序设计 2](#_Toc533792366)

[3 八种排序算法 2](#_Toc533792367)

[3.1 冒泡排序 2](#_Toc533792368)

[3.1.1 基本思想 2](#_Toc533792369)

[3.1.2 核心代码 2](#_Toc533792370)

[3.1.3 冒泡排序的数据记录 3](#_Toc533792371)

[3.1.4分析 3](#_Toc533792372)

[3.2 选择排序 4](#_Toc533792373)

[3.2.1 基本思想 4](#_Toc533792374)

[3.2.2 核心代码 4](#_Toc533792375)

[3.2.3选择排序的数据记录 6](#_Toc533792376)

[3.2.4分析 6](#_Toc533792377)

[3.3 直接插入排序 7](#_Toc533792378)

[3.3.1 基本思想 7](#_Toc533792379)

[3.3.2 核心代码 7](#_Toc533792380)

[3.3.3 直接插入排序的数据记录 9](#_Toc533792381)

[3.3.4分析 9](#_Toc533792382)

[3.4 希尔排序 10](#_Toc533792383)

[3.4.1 基本功能 10](#_Toc533792384)

[3.4.2 核心代码 10](#_Toc533792385)

[3.4.3 希尔排序的数据记录 12](#_Toc533792386)

[3.4.4分析 12](#_Toc533792387)

[3.5 快速排序 13](#_Toc533792388)

[3.5.1 基本功能 13](#_Toc533792389)

[3.5.2核心代码 13](#_Toc533792390)

[3.5.3 快速排序的数据记录 15](#_Toc533792391)

[3.5.4分析 15](#_Toc533792392)

[3.6 堆排序 16](#_Toc533792393)

[3.6.1 基本功能 16](#_Toc533792394)

[3.6.2核心代码 16](#_Toc533792395)

[3.6.3 堆排序的数据记录 18](#_Toc533792396)

[3.6.4分析 18](#_Toc533792397)

[3.7 归并排序 19](#_Toc533792398)

[3.7.1 基本功能 19](#_Toc533792399)

[3.7.2核心代码 19](#_Toc533792400)

[3.7.3 归并排序的数据记录 21](#_Toc533792401)

[3.7.4分析 21](#_Toc533792402)

[3.8 基数排序 22](#_Toc533792403)

[3.8.1 基本功能 22](#_Toc533792404)

[3.8.2核心代码 22](#_Toc533792405)

[3.8.3 基数排序的数据记录 24](#_Toc533792406)

[3.8.4分析 24](#_Toc533792407)

[3.9 总体系统的实现 25](#_Toc533792408)

[3.9.1 总体系统核心代码 25](#_Toc533792409)

[3.9.2 总体系统截屏示例 26](#_Toc533792410)

# 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。

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 设计

## 2.1 主程序设计

系统首先实现对屏幕的初始化，然后根据用户所输入的操作码（order）执行对应的排序算法。在每种排序算法前后分别得到时间来计算算法需要时间。

# 3 八种排序算法

## 3.1 冒泡排序

### 3.1.1 基本思想

在要排序的一组数中，对当前还未排好序的范围内的全部数，自上而下对相邻的两个数依次进行比较和调整，让较大的数往下沉，较小的往上冒。即：每当两相邻的数比较后发现它们的排序与排序要求相反时，就将它们互换。

### 3.1.2 核心代码

void BubbleSort(int a[],int n)

{

double start,finish;

start=(double)clock(); //计算开始时间

bool exchange=false; //判断有没有交换

int i,j;

long long int sortSum=0; //交换次数

long long int compareSum=0; //比较次数

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

{

exchange=false;

for(j=n-1;j>i;j--)

{

//如果前一项大,则交换,交换次数+1

if(a[j-1]>a[j])

{

int temp=a[j];

a[j]=a[j-1];

a[j-1]=temp;

sortSum++;

exchange=true;

}

compareSum++;

}

if(exchange==false) break;

}

finish=(double)clock();

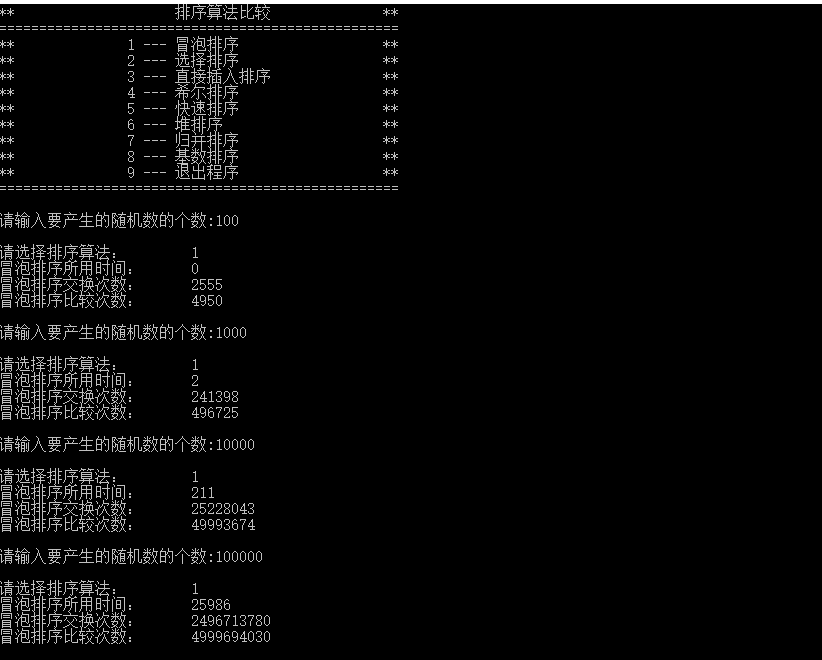
cout<< left <<setw(24)<<"冒泡排序所用时间："<<finish-start<<endl;

cout<< left <<setw(24)<<"冒泡排序交换次数："<<sortSum<<endl;

cout<< left <<setw(24)<<"冒泡排序比较次数："<<compareSum<<endl;

}

### 3.1.3 冒泡排序的数据记录



### 3.1.4分析

100

存储开销：1

1000

存储开销：1

10000

存储开销：1

100000

存储开销：1

平均时间复杂度：O（n^2）

优点：稳定

缺点：排序速度低下

## 3.2 选择排序

### 3.2.1 基本思想

在要排序的一组数中，选出最小（或者最大）的一个数与第1个位置的数交换；然后在剩下的数当中再找最小（或者最大）的与第2个位置的数交换，依次类推，直到第n-1个元素（倒数第二个数）和第n个元素（最后一个数）比较为止。

### 3.2.2 核心代码

//选择排序

void SelectSort(int a[],int n)

{

double start,finish;

start=(double)clock(); //计算开始时间

long long int sortSum=0; //交换次数

long long int compareSum=0; //比较次数

int k=0;

//寻找最大的

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

{

k=i; //用k记录最大值的位置

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

{

//如果有更加大的就记录位置

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

{

k=j;

}

compareSum++;

}

if(k!=i)//如果位置不一样就交换

{

int temp=a[i];

a[i]=a[k];

a[k]=temp;

sortSum++;

}

compareSum++;

}

finish=(double)clock();

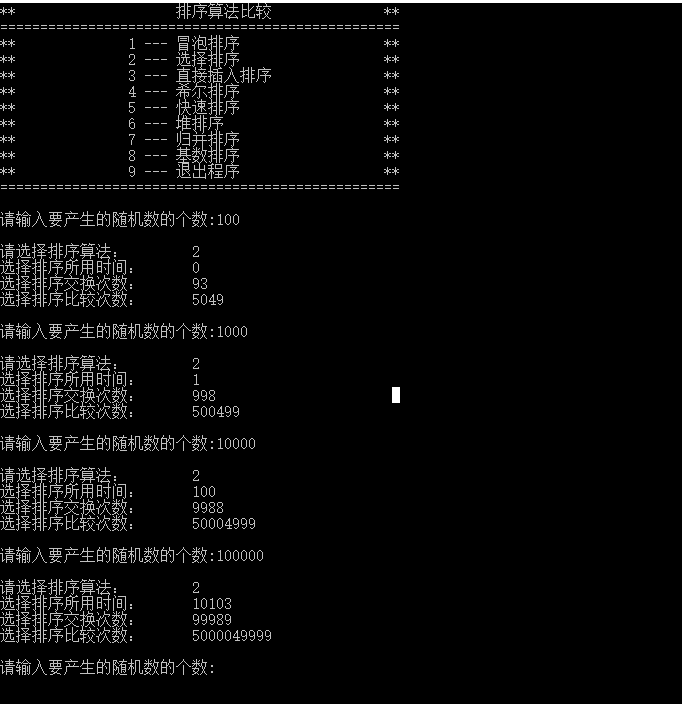
cout<< left <<setw(24)<<"选择排序所用时间："<<finish-start<<endl;

cout<< left <<setw(24)<<"选择排序交换次数："<<sortSum<<endl;

cout<< left <<setw(24)<<"选择排序比较次数："<<compareSum<<endl;

}

### 3.2.3选择排序的数据记录



### 3.2.4分析

100

存储开销：1

1000

存储开销：1

10000

存储开销：1

100000

存储开销：1

平均时间复杂度：O（n^2）

优点：交换次数少

缺点：不稳定

## 3.3 直接插入排序

### 3.3.1 基本思想

整个序列分为有序区和无序区，取第一个元素作为初始有序区，然后第二个开始，依次插入到有序区的合适位置，直到排好序

### 3.3.2 核心代码

//直接插入排序

void InsertSort(int a[],int n)

{

double start,finish;

start=(double)clock(); //计算开始时间

long long int sortSum=0; //交换次数

long long int compareSum=0; //比较次数

int temp,j;

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

{

//发现未排序好区域第一个元素比已排序好的最后一个元素小，说明要加入到前面

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

{

//记录这个值

temp=a[i];

j=i-1;

//不停和前一个值比较,如果前一个值更加大就让那个值向后移动

while(j>=0 && temp<a[j])

{

a[j+1]=a[j];

sortSum++;

compareSum++;

if(j!=0) j--;

else break;

}

a[j]=temp;

}

compareSum++;

}

finish=(double)clock();

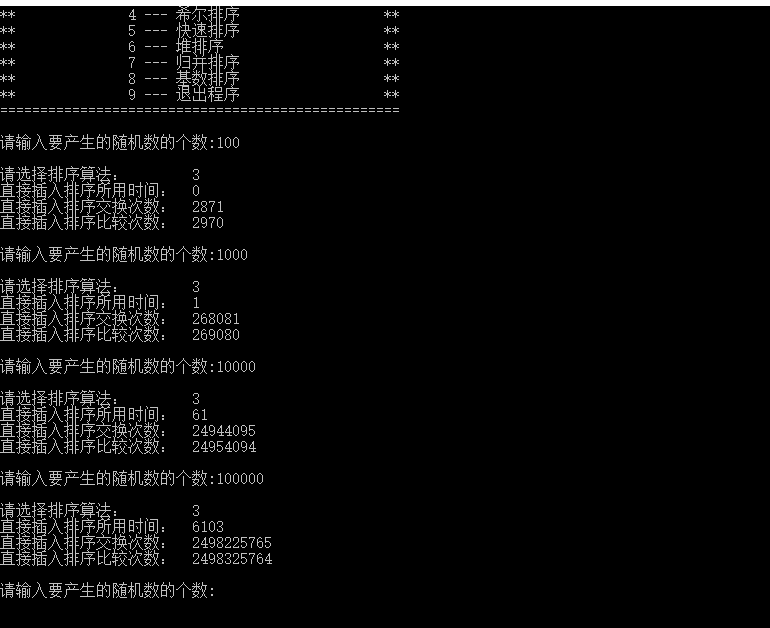
cout<< left <<setw(24)<<"直接插入排序所用时间："<<finish-start<<endl;

cout<< left <<setw(24)<<"直接插入排序交换次数："<<sortSum<<endl;

cout<< left <<setw(24)<<"直接插入排序比较次数："<<compareSum<<endl;

}

### 3.3.3 直接插入排序的数据记录



### 3.3.4分析

100

存储开销：1

1000

存储开销：1

10000

存储开销：1

100000

存储开销：1

平均时间复杂度：O（n^2）

优点：稳定，快速

缺点：比较次数不一定，插入后移动数据多

## 3.4 希尔排序

### 3.4.1 基本功能

先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行依次直接插入排序。

### 3.4.2 核心代码

//希尔排序

void ShellSort(int a[],int n)

{

double start,finish;

start=(double)clock(); //计算开始时间

long long int sortSum=0; //交换次数

long long int compareSum=0; //比较次数

int j;

int gap=n;

int temp;

do{

gap=gap/3+1; //求下一个gap

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

{

if(a[i]<a[i-gap]) //如果前gap的值更加大，说明要排序

{

temp=a[i];

j=i-gap;

//相隔gap来不断后移元素

do{

a[j+gap]=a[j];

j=j-gap;

sortSum++;

compareSum++;

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

a[j+gap]=temp;

}

compareSum++;

}

}while(gap>1);

finish=(double)clock();

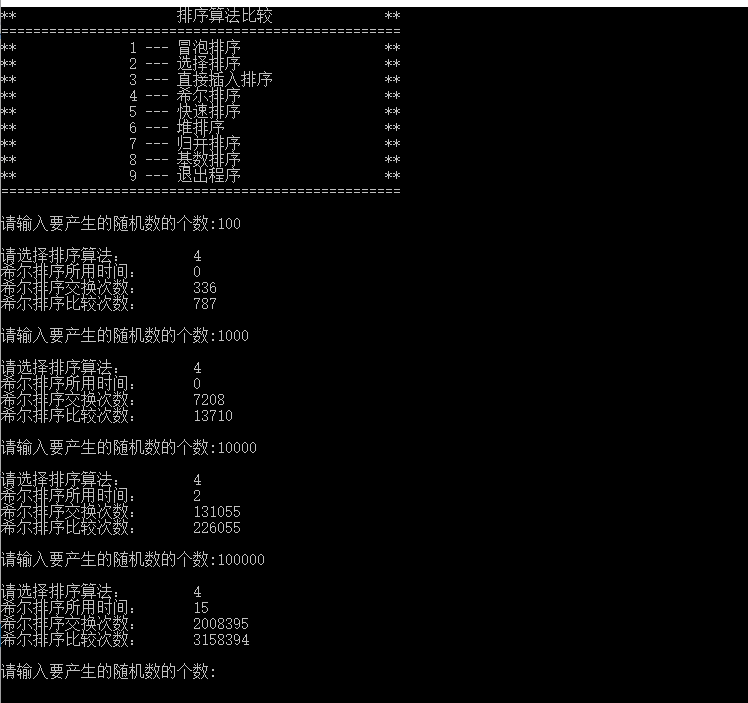
cout<< left <<setw(24)<<"希尔排序所用时间："<<finish-start<<endl;

cout<< left <<setw(24)<<"希尔排序交换次数："<<sortSum<<endl;

cout<< left <<setw(24)<<"希尔排序比较次数："<<compareSum<<endl;

}

### 3.4.3 希尔排序的数据记录



### 3.4.4分析

100

存储开销：2

1000

存储开销：2

10000

存储开销：2

100000

存储开销：2

平均时间复杂度：O（n^1.3）

优点： 容易执行，效率高

缺点： 不稳定

## 3.5 快速排序

### 3.5.1 基本功能

1）选择一个基准元素,通常选择第一个元素或者最后一个元素,

2）通过一趟快速排序的记录分割成独立的两部分，其中一部分记录的元素值均比基准元素值小。另一部分记录的 元素值比基准值大。

3）此时基准元素在其排好序后的正确位置

4）然后分别对这两部分记录用同样的方法继续进行排序，直到整个序列有序。

### 3.5.2核心代码

//快速排序

int PartSort(int a[],int l,int r,long long int& sortSum,long long int &compareSum)

{

//基准元素

int pivotpos=l;

int pivot=a[l];

//检查整个序列，进行划分

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

{

//小于基准的划到左边

if(a[i]<pivot)

{

pivotpos++;

if(pivotpos!=i)

{

int temp=a[pivotpos];

a[pivotpos]=a[i];

a[i]=temp;

sortSum++;

}

}

compareSum++;

}

//将基准元素归位

a[l]=a[pivotpos];

a[pivotpos]=pivot;

sortSum++;

//返回基准元素位置

return pivotpos;

}

void myQuickSort(int a[],int left,int right,long long int& sortSum,long long int& compareSum)

{

if(left<right)

{

int pivotpos=PartSort(a,left,right,sortSum,compareSum);

myQuickSort(a,left,pivotpos-1,sortSum,compareSum);

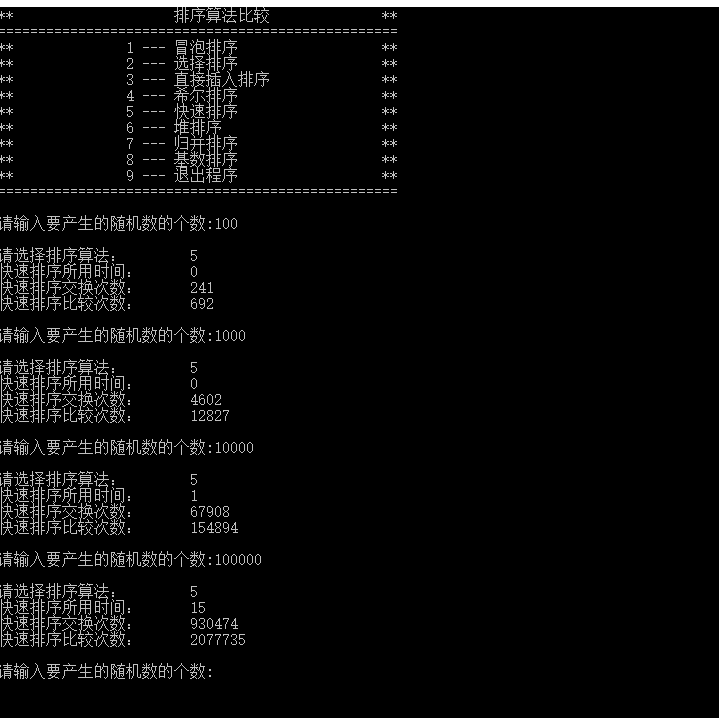
myQuickSort(a,pivotpos+1,right,sortSum,compareSum);

}

compareSum++;

}

### 3.5.3 快速排序的数据记录



### 3.5.4分析

100

存储开销：2

1000

存储开销：2

10000

存储开销：2

100000

存储开销：2

平均时间复杂度：O（nlogn）

优点： 效率高，移动数据少

缺点： 不稳定

## 3.6 堆排序

### 3.6.1 基本功能

初始时把要排序的n个数的序列看作是一棵顺序存储的二叉树（一维数组存储二叉树），调整它们的存储序，使之成为一个堆，将堆顶元素输出，得到n 个元素中最小(或最大)的元素，这时堆的根节点的数最小（或者最大）。然后对前面(n-1)个元素重新调整使之成为堆，输出堆顶元素，得到n 个元素中次小(或次大)的元素。依此类推，直到只有两个节点的堆，并对它们作交换，最后得到有n个节点的有序序列。

### 3.6.2核心代码

//堆排序

void Swap(int& i,int& j)

{

int temp=i;i=j;j=temp;

}

void siftDown(int a[],int start,int m,long long int& sortSum,long long int& compareSum)//从节点start到m从上到下比较，如果子女的值小于

{ //父节点的值则调整

int i=start;

int j=2\*i+1;

int temp=a[i];

while(j<=m)

{

if(j<m && a[j]<a[j+1])j++; //j->min(heap[j],heap[j+1])

else if(temp>=a[j])break;

//the max of child up

else

{

a[i]=a[j];

i=j;

j=2\*j+1;

sortSum++;

}

compareSum+=3;

}

a[i]=temp;

}

void HeapSort(int a[],int n)

{

double start,finish;

start=(double)clock(); //计算开始时间

long long int sortSum=0; //交换次数

long long int compareSum=0; //比较次数

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

{

siftDown(a,i,n-1,sortSum,compareSum);

}

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

{

Swap(a[0],a[i]);

sortSum++;

siftDown(a,0,i-1,sortSum,compareSum);

}

finish=(double)clock();

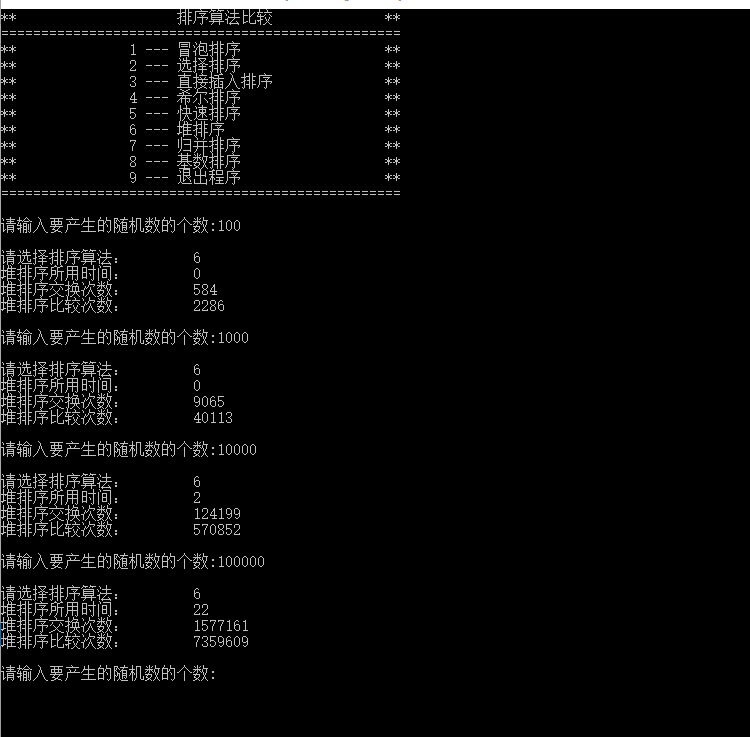
cout<< left <<setw(24)<<"堆排序所用时间："<<finish-start<<endl;

cout<< left <<setw(24)<<"堆排序交换次数："<<sortSum<<endl;

cout<< left <<setw(24)<<"堆排序比较次数："<<compareSum<<endl;

}

### 3.6.3 堆排序的数据记录



### 3.6.4分析

100

存储开销：2

1000

存储开销：2

10000

存储开销：2

100000

存储开销：2

平均时间复杂度：O（nlogn）

优点： 效率高，移动数据少

缺点： 不稳定

## 3.7 归并排序

### 3.7.1 基本功能

归并（Merge）排序法是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列。

### 3.7.2核心代码

//归并排序

void merge(int L1[],int L2[],int left,int mid,int right,long long int& sortSum,long long int& compareSum)

{

//把待归并元素序列L1复制到辅助数组L2中，再从L2归并到L1中

for(int k=left;k<=right;k++)

{

L2[k]=L1[k];

}

int s1=left;

int s2=mid+1; //s1,s2是检测指针

int t=left; //t是存放指针

//两个表都没有检测完

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

{

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

{

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

sortSum++;

}

else

{

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

sortSum++;

}

compareSum+=2;

}

//第一个表没有检测完

while(s1<=mid)

{

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

sortSum++;

}

//第二个表没有检测完

while(s2<=right)

{

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

sortSum++;

}

}

void mergeSort(int a[],int b[],int left,int right,long long int& sortSum,long long int& compareSum)

{

if(left>=right){return;}

int mid=(left+right)/2; //花费为两个子序列

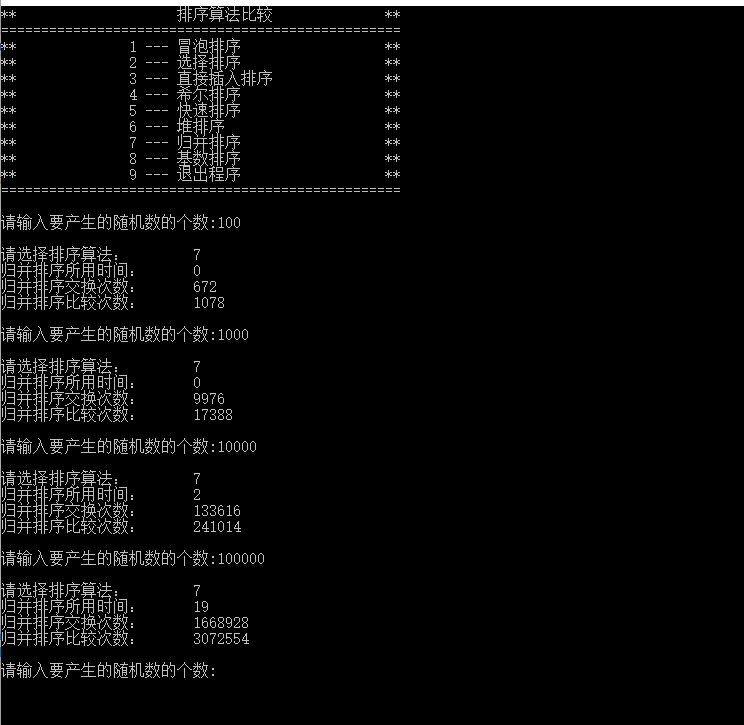
mergeSort(a,b,left,mid,sortSum,compareSum); //对左侧子序列进行排序

mergeSort(a,b,mid+1,right,sortSum,compareSum); //对右侧子序列进行排序

merge(a,b,left,mid,right,sortSum,compareSum); //合并

}

### 3.7.3 归并排序的数据记录



### 3.7.4分析

100

存储开销：100

1000

存储开销：1000

10000

存储开销：10000

100000

存储开销：100000

平均时间复杂度：O（nlogn）

优点： 概念简单，稳定

缺点： 空间复杂度高

## 3.8 基数排序

### 3.8.1 基本功能

先按照低位排序，然后收集；再按照高位排序，然后再收集；依次类推，直到最高位。有时候有些属性是有优先级顺序的，先按低优先级排序，再按高优先级排序。最后的次序就是高优先级高的在前，高优先级相同的低优先级高的在前。

### 3.8.2核心代码

//基数排序

int getMaxDigit(int a[],int n)

{

int digit=1;

int base=10;

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

{

while(a[i]>=base)

{

base\*=10;

digit++;

}

}

return digit;

}

void LSDSort(int a[],int n,long long int& sortSum,long long int& compareSum)

{

int maxDigit=getMaxDigit(a,n);

int count[10];

int \*tmp=new int[n];

int radix = 1;

int i, j, k;

for (i = 1; i <= maxDigit; i++)//进行maxdiDigit次排序

{

//每次分配前清空计数器

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

{

count[j] = 0;

}

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

{

k = (a[j] / radix) % 10;//统计每个桶中的记录数

count[k]++;

}

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

{

count[j] = count[j - 1] + count[j]; //将tmp中的位置依次分配给每个桶

}

//将所有桶中记录依次收集到tmp中

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

{

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

tmp[count[k] - 1] = a[j];

count[k]--;

}

//将临时数组的内容复制到data中

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

{

a[j] = tmp[j];

}

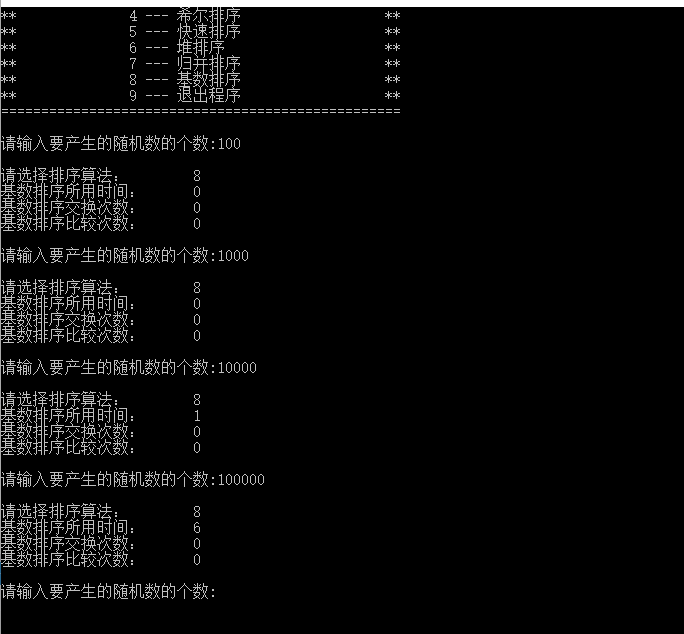
radix = radix \* 10;

}

delete[] tmp;

}

### 3.8.3 基数排序的数据记录



### 3.8.4分析

100

存储开销：112

1000

存储开销：1012

10000

存储开销：10012

100000

存储开销：100012

平均时间复杂度：O（d（n+radix））

优点：稳定，控制机构简单，快速

缺点：空间复杂度高

## 3.9 总体系统的实现

### 3.9.1 总体系统核心代码

while(1)

{

cout << endl;

srand(int(time(0)));

int n;

cout << "请输入要产生的随机数的个数:";

cin >> n;

int \*a = new int[n];

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

{

a[i] = rand();

}

cout<<endl;

for(int i=0;i<n;i++) {a[i]=rand();}

int order;

cout<< left <<setw(24)<<"请选择排序算法：";

cin>>order;

switch(order)

{

case 1:{BubbleSort(a,n); break;}

case 2:{SelectSort(a,n); break;}

case 3:{InsertSort(a,n); break;}

case 4:{ShellSort(a,n); break;}

case 5:{QuickSort(a,n); break;}

case 6:{HeapSort(a,n); break;}

case 7:{MergeSort(a,n); break;}

case 8:{RadixSort(a,n); break;}

case 9: {return 0; }

}

}

### 3.9.2 总体系统截屏示例

