各大排序算法时间统计

备注：

对随机数进行排序，随机数的生成范围为（0，100000）

无序：

数据量	算法	时间（s）
100	冒泡排序	0.000
100	选择排序	0.000
100	直接插入排序	0.000
100	归并排序	0.000
100	快速排序	0.000
100	希尔排序	0.000

数据量	算法	时间（s）
1000	冒泡排序	0.002
1000	选择排序	0.002
1000	直接插入排序	0.000
1000	归并排序	0.000
1000	快速排序	0.000
1000	希尔排序	0.000

数据量	算法	时间（s）
10000	冒泡排序	0.231
10000	选择排序	0.247
10000	直接插入排序	0.076
10000	归并排序	0.001
10000	快速排序	0.003
10000	希尔排序	0.002

数据量	算法	时间（s）
100000	冒泡排序	26.307
100000	选择排序	21.816
100000	直接插入排序	5.953
100000	归并排序	0.026
100000	快速排序	0.014
100000	希尔排序	0.040

数据量	算法	时间（s）
1000000	冒泡排序	超过三分钟
1000000	选择排序	超过三分钟
1000000	直接插入排序	超过三分钟
1000000	归并排序	0.164
1000000	快速排序	0.144
1000000	希尔排序	0.295

基本有序：

数据量	算法	时间（s）
100	冒泡排序	0.000
100	选择排序	0.000
100	直接插入排序	0.000
100	归并排序	0.000
100	快速排序	0.000
100	希尔排序	0.000

数据量	算法	时间（s）
1000	冒泡排序	0.001
1000	选择排序	0.001
1000	直接插入排序	0.000
1000	归并排序	0.000
1000	快速排序	0.001
1000	希尔排序	0.000

数据量	算法	时间（s）
10000	冒泡排序	0.112
10000	选择排序	0.117
10000	直接插入排序	0.000
10000	归并排序	0.001
10000	快速排序	0.117
10000	希尔排序	0.001

数据量	算法	时间（s）
100000	冒泡排序	10.968
100000	选择排序	10.609
100000	直接插入排序	0.000
100000	归并排序	0.019
100000	快速排序	栈溢出
100000	希尔排序	0.007

数据量	算法	时间（s）
1000000	冒泡排序	超过三分钟
1000000	选择排序	超过三分钟
1000000	直接插入排序	0.003
1000000	归并排序	0.089
1000000	快速排序	超时栈溢出
1000000	希尔排序	0.078

代码

排序部分

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
int data[1000000]={0};
int fz[1000000]={0};
void BubbleSort(int a[],int n);
void SelectSort(int a[],int n);
void InsertionSort(int a[],int n);

void MergerSort(int a[],int b[],int high,int low);
void Merge(int a[],int low,int m,int high,int b[]);
```

```
12
13  void QuickSort(int a[],int low,int high);
14  int QuickPass(int a[],int low,int high);
15
16  //希尔排序
17  void ShellSort(int *arr, int size);
18  void check();
19
20  int main(){
21       /*int a[5]={23,43,32,12,0};
22       int b[5];
23       ShellSort(a,5);
24       for(int i=0;i<5;i++){
25           printf("%d ",a[i]);
26
27       }*/
28       check();
29       return 0;
30  }
31  void check(){
32       FILE *f;
33       char ch;
34       int count=0;
35       clock_t start, stop; //clock_t为clock()函数返回的变量类型
36       double duration;
37       if((f=fopen("D:\\csjjg\\程序设计综合实践\\排序\\data.txt","r"))==NULL){
38           printf("fail to read!");
39           exit(1);
40       }
41       while(!feof(f)){
42           ch=fgetc(f);
43           while(ch>='0' && ch<='9'){
44               data[count]=(ch-'0')+data[count]*10;
45               ch=fgetc(f);
46
47           }
48           count++;
49       }
50      start=clock();
51      //BubbleSort(data,1000000);
52      //SelectSort(data,1000000);
53      //InsertionSort(data,1000000);
54      //MergerSort(data,fz,999999,0);
55      //QuickSort(data,0,999999);
56      //ShellSort(data,1000000);
57      stop=clock();
58      duration=(double)(stop-start)/CLK_TCK; //CLK_TCK为clock()函数的时间单位，
    即时钟打点
59      printf("%f\n",duration);
60
61  }
62  void BubbleSort(int a[],int n){
63      for(int i=0;i<n-1;i++){
64          for(int j=0;j<n-i-1;j++){
65              int flag;
66             if(a[j]>a[j+1]){
67               flag=a[j];
68               a[j]=a[j+1];
```

```
            a[j+1]=flag;
          }
        }
    }
}

void SelectSort(int a[],int n){
    for(int i=0;i<n-1;i++){
        int min=i;
        for(int j=i+1;j<n;j++){
            if(a[j]<a[i]){
                min=j;
            }
        }
        int flag;
        flag=a[i];
        a[i]=a[min];
        a[min]=flag;

    }
}

void InsertionSort(int a[],int n){
    for(int i=1;i<n;i++){
       int x=a[i];
       int j=i-1;
       while(j>=0 && a[j]>x){
           a[j+1]=a[j];
           j--;
       }
       a[j+1]=x;
    }
}

//归并
void MergerSort(int a[],int b[],int high,int low){
    if(low>=high){
        return ;//规模不超过1，不需要排序
    }
    int m= (low + high)/2;
    MergerSort(a,b,m,low);//前一半子序列排序
    MergerSort(a,b,high,m+1);//后一半子序列排序
    Merge(a,low,m,high,b);//归并两段有序子序列
    for(int i=low;i<=high;i++){//移动回原数组
        a[i]=b[i];
    }

}
void Merge(int a[],int low,int m,int high,int b[]){
    int i=low;//前段有序子序列起点
    int j=m+1;//后段有序子序列起点
    int k=i;//归并结果起始下标
    while(i<=m && j<=high){
        if(a[i]<a[j]){
            b[k++]=a[i++];
        }else{
            b[k++]=a[j++];
        }
```

```
    }
    while(i<=m){
        b[k++]=a[i++];
    }
    while(j<=high){
        b[k++]=a[j++];
    }
}
//快速排序
void QuickSort(int a[],int low,int high){
    if (low>=high)
    {
        return;
    }
    int pivot=QuickPass(a,low,high);
    QuickSort(a,low,pivot-1);
    QuickSort(a,pivot+1,high);

    }
int QuickPass(int a[],int low,int high){
    int x=a[low];
    while(low<high){
        while(low<high && x<=a[high])
        high--;
        if(low==high){
            break;
        }
        a[low++]=a[high];
        while(low<high && x>=a[low])
        low++;
        if(low==high){
            break;
        }
        a[high--]=a[low];

    }
    a[low]=x;
    return low;
}

//希尔排序

void ShellSort(int *a, int len)
{
    int i, j, k, tmp, gap;  // gap 为步长
    for (gap = len / 2; gap > 0; gap /= 2) {  // 步长初始化为数组长度的一半，每次遍历后步长减半，
        for (i = 0; i < gap; ++i) { // 变量 i 为每次分组的第一个元素下标
            for (j = i + gap; j < len; j += gap) { //对步长为gap的元素进行直插排序，当gap为1时，就是直插排序
                tmp = a[j];  // 备份a[j]的值
                k = j - gap;  // j初始化为i的前一个元素（与i相差gap长度）
                while (k >= 0 && a[k] > tmp) {
                    a[k + gap] = a[k]; // 将在a[i]前且比tmp的值大的元素向后移动一位
                    k -= gap;
                }
                a[k + gap] = tmp;
```

```
            }
        }
    }
}
```

生成随机数部分

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include<math.h>

int main()
{
    srand(time(NULL));//先种种子
    int i,j,s=0;
    FILE *fp = NULL;

    fp = fopen("D:\\csjjg\\程序设计综合实践\\排序\\data.txt","a");//在指定目录下创建.txt文件
    for(i = 0 ; i < 1000000 ; i ++) //产生随机数
    {
        j = rand()%100000 ;
        fprintf(fp,"%d\n",j); //把随机数写进文件
    }
    fclose(fp); //关闭文件
    return 0;
    }
```

总结

对于无序的数据平均性能最好的为快速排序算法，有序数据对快速排序的效率不利

无序数据数据量很大时冒泡、直接插入、选择排序法消耗时间过于长