北京邮电大学软件学院

2016-2017学年第1学期实验报告

课程名称：\_算法与数据结构\_

实验名称：\_排序\_

实验完成人： \_刘浩博\_

姓名：\_刘浩博\_ 学号：\_2015212086\_ 成绩：\_\_\_\_\_\_\_\_\_\_\_

指导老师： \_贾红娓\_

日期： 2017 年 1 月 17 日

⼀.**实验⽬的**

本次实验旨在集中对⼏个专门的问题作较为深⼊的探讨和理解，不强调对某些特定的

编程技术的训练。

**⼆. 实验内容**

**内部排序算法⽐较**

**[问题描述]**

　　各种内部排序算法的时间复杂度分析结果只给出了算法执⾏时间的阶，或⼤概执⾏时

间。试通过随机的数据⽐较各算法的关键字⽐较次数和关键字移动次数，以取得直观感受。

**[基本要求]**

　　（1） 对以下9种常⽤的内部排序算法进⾏⽐较：直接插⼊排序；折半折⼊排序；⼆路

插⼊排序；希尔排序；起泡排序；快速排序；简单选择排序；堆排序；归并排序。

　　（2） 待排序表的表长不少于100；其中的数据要⽤伪随机数产⽣程序产⽣；⾄少要⽤

5组不同的输⼊数据作⽐较；⽐较的指标为有关键字参加的⽐较次数和关键字移动次数（关

键字交换计为3次移动）。

**[测试数据]**

　　由随机产⽣器决定。

**[实现提⽰]**

　　主要⼯作是设法在程序中适当的地⽅插⼊计数操作。程序还可以包括计算⼏组数据得

出结果波动⼤⼩的解释。注意分块调试的⽅法。

**三. 实验环境**

Dev-C++ 5.11

**四. 实验结果**

达到实验要求，程序可完成实验目的，且无错误出现。

**五. 附录**

**Sort.cpp:**

#include<stdio.h>

#include<stdlib.h>

#define MAXSIZE 100

typedef int KeyType;

typedef struct{

KeyType key; //关键字项

}RedType; //记录类型

typedef struct{

RedType r[MAXSIZE+1];

int comp=0;//关键字比较次数

int move=0;//关键字移动次数

int length;

}SqList;

//随机产生数字

void random(SqList &L){

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

L.r[i].key=rand()%9999+1;

L.comp=L.move=0;

}

//输出比较指标

void PrintInfo(SqList &L){

printf("Compare times:%d\nMove times:%d\n",L.comp,L.move);

}

//直接插入排序

void InsertSort(SqList &L){

int i,j;

for(i=2;i<=L.length;++i){

L.comp++;

if(L.r[i].key<L.r[i-1].key){

L.r[0]=L.r[i]; L.move+=3;

L.r[i]=L.r[i-1]; L.move+=3;

L.comp++;

for(j=i-2;L.r[0].key<L.r[j].key;--j,L.comp++){

L.r[j+1]=L.r[j]; L.move+=3;

}

L.r[j+1]=L.r[0]; L.move+=3;

}

}

PrintInfo(L);

}

//希尔排序

void ShellInsert(SqList &L,int dk){

int i,j;

for(i=dk+1;i<=L.length;i++,L.comp++){

if(L.r[i].key<L.r[i-dk].key){

L.r[0]=L.r[i]; L.move+=3;

}

for(j=i-dk;j>0&&L.r[0].key<L.r[j].key;j-=dk,L.comp++){

L.r[j+dk]=L.r[j]; L.move+=3;

}

L.r[j+dk]=L.r[0]; L.move+=3;

}

}

void ShellSort(SqList &L,int dlta[],int t){

for(int k=0;k<t;++k)

ShellInsert(L,dlta[k]);

}

//起泡排序

void BubbleSort(SqList &L){

int i,times=1;//times为排序趟数

bool chg=false;//chg为交换标志

while(!chg){

chg=true;

for(i=1;i<=L.length-times;i++,L.comp++){

if(L.r[i].key>L.r[i+1].key){

L.r[0]=L.r[i];

L.r[i]=L.r[i+1];

L.r[i+1]=L.r[0];

L.move+=3;

chg=false;

}

}

times++;

}

PrintInfo(L);

}

//快速排序

int Partition(SqList &L,int low,int high){

int pivotkey;

L.r[0]=L.r[low];

pivotkey=L.r[low].key;

while(low<high){

while(low<high&&L.r[high].key>=pivotkey){

--high;

L.comp++;

}

L.comp++;

L.r[low]=L.r[high]; L.move+=3;

while(low<high&&L.r[low].key<=pivotkey){

++low;

L.comp++;

}

L.comp++;

L.r[high]=L.r[low]; L.move+=3;

}

L.r[low]=L.r[0]; L.move+=3;

return low;

}

void QSort(SqList &L,int low,int high){

int pivotloc;

if(low<high){

pivotloc=Partition(L,low,high);

QSort(L,low,pivotloc-1);

QSort(L,pivotloc+1,high);

}

}

//简单选择排序

int SelectMinKey(SqList &L,int i){

int j=i;

for(i;i<=L.length;i++){

L.comp++;

if(L.r[i].key<L.r[j].key)

j=i;

}

return j;

}

void SelectSort(SqList &L){

int i,j;

for(i=1;i<L.length;++i){

j=SelectMinKey(L,i);

if(i!=j){

L.r[0]=L.r[i];

L.r[i]=L.r[j];

L.r[j]=L.r[0];

L.move+=3;

}

}

}

//堆排序

void HeapAdjust(SqList &H,int s,int m){

RedType rc;

int j;

rc=H.r[s];

for(j=2\*s;j<=m;j\*=2){

H.comp++;

if(j<m&&H.r[j].key<H.r[j+1].key)

++j;

H.comp++;

if(rc.key>=H.r[j].key)

break;

H.r[s]=H.r[j]; s=j; H.move+=3;

}

H.r[s]=rc;

}

void HeapSort(SqList &H){

int i;

for(i=H.length/2;i>0;--i)

HeapAdjust(H,i,H.length);

for(i=H.length;i>1;--i){

H.r[0]=H.r[1];

H.r[1]=H.r[i];

H.r[i]=H.r[0];

H.move+=9;

HeapAdjust(H,1,i-1);

}

}

//归并排序

void Merge(SqList &L,RedType SR[],RedType TR[],int i,int m,int n){

int j,k;

for(j=m+1,k=i;i<=m&&j<=n;++k,L.comp++){

if(SR[i].key<SR[j].key){

TR[k]=SR[i++];

L.move+=3;

}

else{

TR[k]=SR[j++];

L.move+=3;

}

if(i<=m)

for(int s=k;s<=n;s++){

TR[s]=SR[s-k+i];

L.move+=3;

}

if(j<=n)

for(int s=k;s<=n;s++){

TR[s]=SR[s-k+j];

L.move+=3;

}

}

}

void MSort(SqList &L,RedType SR[],RedType TR1[],int s,int t){

int m;

RedType TR2[MAXSIZE];

if(s==t){

TR1[s]=SR[s];

L.move+=3;

}

else{

m=(s+t)/2;

MSort(L,SR,TR2,s,m);

MSort(L,SR,TR2,m+1,t);

Merge(L,TR2,TR1,s,m,t);

}

}

void MergeSort(SqList &L){

MSort(L,L.r,L.r,1,L.length);

}

int main(){

int i=0;

int dlta[5]={1,3,5,7,9};

SqList L;

L.length=MAXSIZE;

random(L);printf("--直接插入排序--\n");InsertSort(L);

random(L);printf("--希尔排序--\n");ShellSort(L,dlta,5);PrintInfo(L);

random(L);printf("--起泡排序--\n");BubbleSort(L);

random(L);printf("--快速排序--\n");QSort(L,1,L.length);PrintInfo(L);

random(L);printf("--简单选择排序--\n");SelectSort(L);PrintInfo(L);

random(L);printf("--堆排序--\n");HeapSort(L);PrintInfo(L);

random(L);printf("--归并排序--\n");MergeSort(L);PrintInfo(L);

return 0;

}

**结果截图：**

