桂林电子科技大学

**数据结构与算法**  实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | **实验六 排序算法的实现与比较** | | | | | | | |  | 辅导员意见：  成绩 辅导员  签 名 |
| 院 系 | 计算机与信息安全学院 | | | 专业 | | 计算机大类 | | |
| 学 号 | 1800300636 | | | 姓名 | | 奚锐 | | |
| 实验日期 | 2019 | 年 | 11 | | 月 | | 2 | 日 |
|  |  | | | | | | | |

**一、实验目的**

（1）掌握几种低效的排序方法；

（2）掌握几种高效的排序方法。

**二、实验具体内容**

**1、实验题目1：**

（1）自行选择至少2种低效的排序算法，比如直接插入排序、二分插入排序、直接选择排序和冒泡排序等；

（2）自行选择至少2种高效的排序算法，比如快速排序、堆排序、基数排序和归并等。

**（2）分析**（必须手写！！！）

**（3）实验代码**（源代码可以打印，算法描述必须手写！！！）

#include <bits/stdc++.h>

using namespace std;

const int NUM =100000;

int a[NUM];

int tempArray[NUM];

clock\_t start, finish;

long long moveNum = 0;

long long compareNum=0;

int sizeodarray;

struct StaticItem{

int index;

string name;

double value;

StaticItem(){

index=0;

name="null";

value=0;

}

StaticItem(string \_name,int \_index){

//单位

string unit;

//统计项目

name=\_name;

//编号

index=\_index;

//统计值

value =0;

}

void upCount(){

value++;

}

void downCount(){

value--;

}

};

class MyStatic{

vector<StaticItem> staticPool;

public :

StaticItem\* add(string name){

staticPool.push\_back(StaticItem(name,rand()));

return &staticPool.back();

}

void toResult(){

string t;

cout<<staticPool.size();

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

cout<<staticPool[i].name<<"的统计结果为"<<staticPool[i].value<<endl;

}

}

} sortStatic;

void newArray(){

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

tempArray[i] = a[i];

}

}

void myswap(int &a, int &b)

{

int t = a;

a = b;

b = t;

}

void print(int \*arr, int n)

{

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

{

cout << arr[i] << " ";

}

cout << endl;

}

void straightInsert(int \*arr, int n,char q)

{

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

{

compareNum++;

int temp = arr[i];

int j;

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

{

if(q=='<'){

if (arr[j] > temp)

{

moveNum++;

arr[j + 1] = arr[j];

}

else

{

break;

}

}

else if(q=='>'){

if (arr[j] < temp)

{

moveNum++;

arr[j + 1] = arr[j];

}

else

{

break;

}

}

}

arr[j + 1] = temp;

//print(arr, n);

}

}

void popSort(int \*arr, int n,char q)

{

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

{

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

{

compareNum++;

if(q=='<'){

if (arr[j + 1] < arr[j])

{

moveNum++;

myswap(arr[j + 1], arr[j]);

}

}

else if(q=='>'){

if (arr[j + 1] > arr[j])

{

moveNum++;

myswap(arr[j + 1], arr[j]);

}

}

}

//print(arr, n);

}

}

int partition(int arr[], int left, int right,char q) //找基准数 划分

{

int i = left + 1;

int j = right;

int temp = arr[left];

while (i <= j)

{

if(q=='<'){

while (arr[i] < temp)

{

compareNum++;

i++;

}

while (arr[j] > temp)

{

compareNum++;

j--;

}

}

else if (q=='>'){

while (arr[i] > temp)

{

compareNum++;

i++;

}

while (arr[j] < temp)

{

compareNum++;

j--;

}

}

if (i < j){

swap(arr[i++], arr[j--]);

moveNum++;

}

else

i++;

}

swap(arr[j], arr[left]);

// print(arr, NUM);

return j;

}

void quick\_sort(int arr[], int left, int right,char q)

{

if (left > right)

return;

int j = partition(arr, left, right,q);

quick\_sort(arr, left, j - 1,q);

quick\_sort(arr, j + 1, right,q);

}

/\*该函数将数组下标范围[l1,r1]和[l2,r2]的有序序列合并成一个有序序列\*/

void merge(int \*arr, int l1, int r1, int l2, int r2,char q)

{

int i = l1; //左半部分起始位置

int j = l2; //右半部分起始位置

int n = (r1 - l1 + 1) + (r2 - l2 + 1); //要合并的元素个数

vector<int> temp(n); //辅助数组

int k = 0; //辅助数组其起始位置

while (i <= r1 && j <= r2)

{ //挑选两部分中最小的元素放入辅助数组中

if(q=='<'){

if (arr[i] < arr[j])

temp[k++] = arr[i++];

else

temp[k++] = arr[j++];

}

else if(q=='>'){

if (arr[i] > arr[j])

temp[k++] = arr[i++];

else

temp[k++] = arr[j++];

}

}

//如果还有剩余，直接放入到辅助数组中

while (i <= r1){

moveNum++;

temp[k++] = arr[i++];

}

while (j <= r2){

moveNum++;

temp[k++] = arr[j++];

}

//更新原始数组元素

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

{

moveNum++;

arr[l1 + i] = temp[i];

}

}

/\*二路归并排序（递归实现）\*/

void MergeSort(int \*arr, int start, int end,char q)

{

if (start < end)

{

int mid = (start + end) >> 1; //分割序列

MergeSort(arr, start, mid,q); //对序列左半部分进行规并排序

MergeSort(arr, mid + 1, end,q); //对序列右半部分进行规并排序

merge(arr, start, mid, mid + 1, end,q); //合并已经有序的两个序列

}

}

int main()

{

sizeodarray=0;

int i=0;

while(cin>>a[i]){

i++;

}

sizeodarray=i;

printf("读入%d个数据\n",i);

newArray();

straightInsert(tempArray, NUM,'<');

printf("straightInsert 对随机数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

straightInsert(tempArray, NUM,'>');

printf("straightInsert 对逆序数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

straightInsert(tempArray, NUM,'>');

printf("straightInsert 对顺序数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

newArray();

popSort(tempArray, NUM,'<');

printf("popSort 对随机数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

popSort(tempArray, NUM,'>');

printf("popSort 对逆序数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

popSort(tempArray, NUM,'>');

printf("popSort 对顺序数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

newArray();

quick\_sort(tempArray, 0, sizeodarray,'<');

printf("quick\_sort 对随机数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

quick\_sort(tempArray, 0, sizeodarray,'>');

printf("quick\_sort 对逆序数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

quick\_sort(tempArray, 0, sizeodarray,'>');

printf("quick\_sort 对顺序数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

newArray();

MergeSort(tempArray, 0, sizeodarray,'<');

printf("MergeSort 对随机数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

MergeSort(tempArray, 0, sizeodarray,'>');

printf("MergeSort 对逆序数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

moveNum = 0,compareNum=0;

MergeSort(tempArray, 0, sizeodarray,'>');

printf("MergeSort 对顺序数排序后 compareNUM:%lld moveNum:%lld\n",compareNum,moveNum);

return 0;

}



**三、实验小结**（必须手写！！！）