**实 验 报 告**

**实验名： 内排序算法及其性能比较**

**实验课程：** **数据结构**

**学 院： 通信与信息工程学院**

**专 业： 电子信息工程**

**姓 名： 盛立异**

**班 级： B150111**

**学 号： B15011731**

**指导老师： 李华康**

**时 间： 2016.12.29**

**南京邮电大学**

**内排序算法及其性能比较**

1. 问题描述

比较和改进排序算法的性能。

1. 概要设计

程序包括 main.c和ADT.h两个模块，其模块结构图如图1。

main.c

ADT.h

图1 模块结构图

main.c为主程序模块，ADT.h中存放了顺序表结构体和六种算法。主函数通过调用ADT.h中的函数即可实现直接插入排序、冒泡排序、改进的冒泡排序、选择排序、快速排序、改进的快速排序的调用。

1. 详细设计
   1. 数据结构的设计

需要用到一个顺序表，一个边结点，存放的数据类型为一个整形数组用于存放需要排序的数据和两个整形变量用于存放数组的最大值和需要排序的数据个数。

* 1. 模块设计

ADT模块中，InsertSort为直接插入排序、BubbleSort为冒泡排序、BubbleSort1为改进的冒泡排序、SelectSort为选择排序、QuickSort为快速排序、ImprovedSort为改进的快速排序。

* 1. 算法流程图

如图2是InsertSort函数的流程图

开始

int i=1, j;T x;

x = list->Elements[i];

结束

i<list->Size

将x第i-1个元素开始比较，要是x比该元素小则将该元素往后移一位，要是x比该元素大，则x排在该元素后一位

i++

否

是

图2 InsertSort函数

如图3是BubbleSort的流程图

开始

int i = list->Size - 1;

int j;T temp;

BOOL sorted = FALSE;

i>0 && !sorted

如果前面一个元素大于后面一个元素则交换

sorted=FALSE;

sorted = TRUE;

j++

是

否

j<i

否

结束

是

i--

图3 BubbleSort函数

如图4为BubbleSort1函数的流程图

开始

int i = list->Size - 1;

int j,last;T temp;

i>0

如果前面一个元素大于后面一个元素则交换

last=j

last=0

j++

是

否

j<i

否

结束

是

i=last

图4 BubbleSort1函数

如图5为SelectSort函数的流程图

int i=0,j,index;

T temp;

开始

结束

index = i; j=i+1;

i<list->Size-1;

j<list->Size

如果index之后的元素比下标为index的元素大，将其下标赋值给index

交换下标为i和index的两个元素

是

否

是

否

图5 SelectSort函数

如图6为Partition函数的流程图

开始

int i = left, j = right + 1;

T temp;

T pivot = list->Elements[left];

T temp;

T pivot = list->Elements[left];

交换下标为i,j的两个元素

i从左向右找第一个不小于pivot的元素

j从左向右找第一个不大于pivot的元素

是

i<j

否

结束

交换下标left和right的元素

return j

图6 Partition函数

如图7为QSort函数的流程图

开始

int k;

k = Partition(list, left, right);

是

left<right

否

结束

调用自己排序left到k-1的数据和k+1到right的数据

图7 QSort函数

* 1. 算法分析

直接插入排序渐进时间复杂度为O(n^2)，冒泡排序渐进时间复杂度为O(n^2)，选择排序渐进时间复杂度为O(n^2)，快速排序渐进时间复杂度为O(n\*lb(n)).

1. 程序代码

//直接插入排序

void InsertSort(List \*list){

int i, j;

T x;

for (i = 1; i<list->Size; i++){

x = list->Elements[i];

for (j = i - 1; j >= 0 && x<list->Elements[j]; j--){

list->Elements[j + 1] = list->Elements[j];

}

list->Elements[j + 1] = x;

}

}

//冒泡排序

void BubbleSort(List \*list){

int i = list->Size - 1;

int j;

T temp;

BOOL sorted = FALSE;

while (i>0 && !sorted){

sorted = TRUE;

for (j = 0; j<i; j++){

if (list->Elements[j] > list->Elements[j + 1])

Swap(list->Elements[j], list->Elements[j + 1], temp);

sorted = FALSE;

}

i--;

}

}

//改进冒泡排序

void BubbleSort1(List \*list){

int i = list->Size - 1;

int j, last;

T temp;

while (i>0){

last = 0;

for (j = 0; j<i; j++){

if (list->Elements[j] > list->Elements[j + 1]){

Swap(list->Elements[j], list->Elements[j + 1], temp);

last = j;

}

}

i = last;

}

}

//简单选择排序

void SelectSort(List \*list){

int i, j, index;

T temp;

for (i = 0; i<list->Size - 1; i++){

index = i;

for (j = i + 1; j<list->Size; j++){

if (list->Elements[index]>list->Elements[j])

index = j;

}

Swap(list->Elements[i], list->Elements[index], temp);

}

}

int Partition(List \*list, int left, int right){

int i = left, j = right + 1;

T temp;

T pivot = list->Elements[left];

do{

do i++; while (list->Elements[i]<pivot);

do j--; while (list->Elements[j]>pivot);

if (i<j) Swap(list->Elements[i], list->Elements[j], temp);

} while (i<j);

Swap(list->Elements[left], list->Elements[j], temp);

return j;

}

void QSort(List \*list, int left, int right){

int k;

if (left<right){

k = Partition(list, left, right);

QSort(list, left, k - 1);

QSort(list, k + 1, right);

}

}

//快速排序

void QuickSort(List \*list){

QSort(list, 0, list->Size - 1);

}

void ISort(List \*list, int left, int right){

int k;

if (right - left + 1 < 10){

int i, j;

T x;

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

x = list->Elements[i];

for (j = i - 1; j >= 0 && x<list->Elements[j]; j--){

list->Elements[j + 1] = list->Elements[j];

}

list->Elements[j + 1] = x;

}

}

else{

k = Partition(list, left, right);

QSort(list, left, k - 1);

QSort(list, k + 1, right);

}

}

//改进快速排序

void ImprovedSort(List \*list){

ISort(list, 0, list->Size - 1);

}

1. 测试和调试

开始界面如图5。

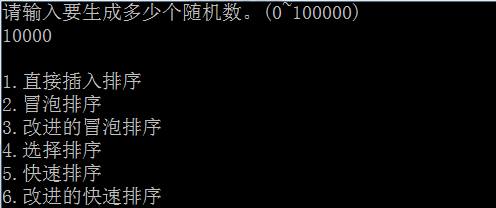


图8 开始界面

测试排序时间界面如图6所示

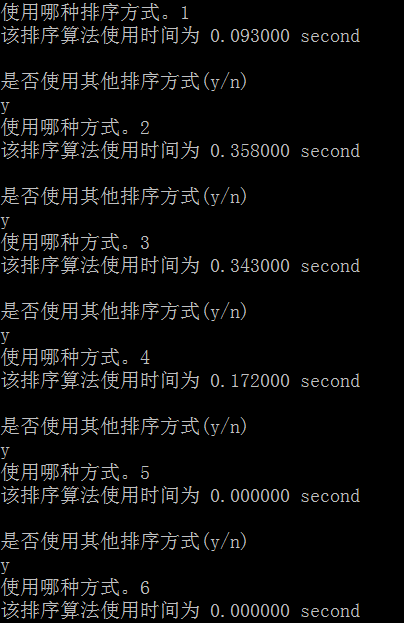


图9 测试排序时间界面

排序算法性能比较如图10。

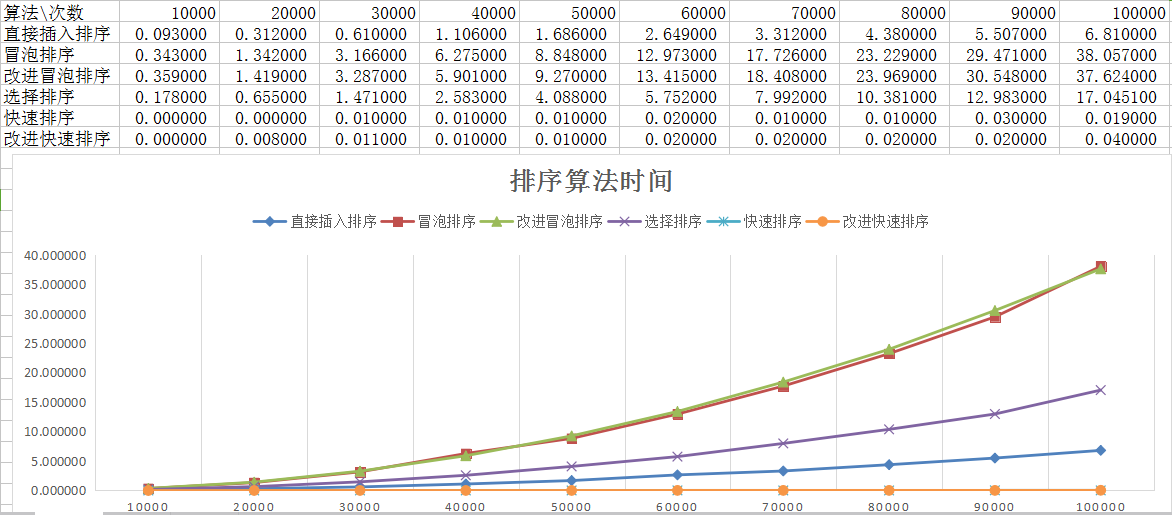


图10 排序算法性能比较

6．实验小结

1）由图10所示快速排序算法和改进快速排序算法的效率最高，冒泡排序和改进冒泡排序的效率最低。

2）由于数据有限的情况下，改进前后的排序算法效率相差不大。