姓名： 毛翊尧 班级： A1611 学号： 31号

**实验六 排序**

一、实验目的

1、掌握内部排序的基本算法；

2、分析比较内部排序算法的效率。

二、实验预习

说明以下概念

1、简单排序：将一组记录按某关键字递增或递减的顺序排序。

2、希尔排序：又称“缩小增量排序”属于插入排序类的方法。

3、快速排序：通过一趟排序将待排记录分割成独立的两部分，其中一部分记录的关键字均

比另一部分记录的关键字小，则可分别对这两部分记录继续进行排序，以达到整个序列有序。

4、堆排序：只需要一个记录大小的辅助空间，每个待排序的记录仅占有一个存储空间。

三、实验内容和要求

1. 运行下面程序：

#include <stdlib.h>

#include <stdio.h>

#define MAX 50

int slist[MAX]; /\*待排序序列\*/

void insertSort(int list[], int n);

void createList(int list[], int \*n);

void printList(int list[], int n);

void heapAdjust(int list[], int u, int v);

void heapSort(int list[], int n);

/\*直接插入排序\*/

void insertSort(int list[], int n)

{

int i = 1, j = 0, node = 0, count = 1;

printf("对序列进行直接插入排序：\n");

printf("初始序列为：");

printList(list, n);

for(i = 2; i <= n; i++)

{

node = list[i];

j = i - 1;

while(j >= 0 && node < list[j])

{

list[j+1] = list[j];

--j;

}

list[j+1] = node;

printf("第%d次排序结果：", count++);

printList(list, n);

}

}

/\*堆排序\*/

void heapAdjust(int list[], int u, int v)

{

int i = u, j , temp = list[i];

j = 2 \* i;

while (j <= v)

{

if(j < v && list[j] < list[j+1])

j++; /\*若右孩子较大，则把j修改为右孩子的下标\*/

if(temp < list[j])

{

list[i] = list[j]; /\*将list[j]调到父亲的位置上\*/

i = j;

j = 2 \* i; /\*修改i和j的值，以便继续向下筛选\*/

}

else

break; /\*筛选完成，终止循环\*/

}

list[i] = temp; /\*被筛结点的值放入最终位置\*/

}

void heapSort(int list[], int n)

{

int i = 0, temp = 0, count = 1;

printf("对序列进行堆排序：\n");

printf("初始序列为：");

printList(list, n);

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

heapAdjust(list, i, n); /\*建立初始堆\*/

printf("建立的初始堆为：");

printList(list, n);

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

{/\*循环，完成堆排序\*/

temp = list[1];

list[1] = list[i];

list[i] = temp; /\*将第一个元素同当前区间内最后一个元素对换\*/

heapAdjust(list, 1 , i-1); /\*筛选出list[1]结点\*/

printf("第%d次排序结果：", count++);

printList(list, n);

}

}

/\*生成待排序序列\*/

void createList(int list[], int \*n)

{

int i = 1,a;

printf("请输入待排序序列(长度小于50,以输入一个字符结束):\n");

while(scanf("%d",&a)==1)

{

list[i] = a;

i++;

}

\*n=i-1;

getchar();

}

/\*输出排序结果\*/

void printList(int list[], int n)

{

int i = 1;

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

{

printf(" %d ", list[i]);

if(i % 10 ==0 && i != 0)

printf("\n");

}

printf("\n");

}

int main()

{

int choice=1,length;

while(choice!=0)

{

printf("\n");

printf("\*\*\*\*\* 内部排序算法演示程序 \*\*\*\*\*\n");

printf("\n1. 直接插入排序 \n");

printf("\n2. 堆排序 \n");

printf("\n0. 退出\n");

printf("\n请选择：");

scanf("%d", &choice);

getchar();

switch(choice)

{

case 1:

{

createList(slist, &length);

insertSort(slist, length);

break;

}

case 2:

{

createList(slist, &length);

heapSort(slist, length);

break;

}

default:choice=0;

}

printf("\n");

}

return 0;

}

输入待排序序列：49 38 65 97 13 27 49（以输入一个字符作为结束）

1. 直接插入排序运行结果（写出每一趟的状态）：

第1次排序结果： 38 49 65 97 13 27 49

第2次排序结果： 38 49 65 97 13 27 49

第3次排序结果： 38 49 65 97 13 27 49

第4次排序结果： 13 38 49 65 97 27 49

第5次排序结果： 13 27 38 49 65 97 49

第6次排序结果： 13 27 38 49 49 65 97

2）堆排序运行结果（写出每一趟的状态）：

第1次排序结果： 65 49 49 38 13 27 97

第2次排序结果： 49 38 49 27 13 65 97

第3次排序结果： 49 38 13 27 49 65 97

第4次排序结果： 38 27 13 49 49 65 97

第5次排序结果： 27 13 38 49 49 65 97

第6次排序结果： 13 27 38 49 49 65 97

2、在1题中补充希尔排序算法。

算法代码：

void ShellSort(SqList &L)   {

  printf("希尔排序\n");

int i, j;

int dk = 1;//增量

while(dk <=L.length/3)

dk = 3\*dk+1;//增大增量

while(dk>0)  {

  dk = 3;//减小增量

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

   L.r[0].key = L.r[i].key;

j = i;

   while ((j >= dk) && (L.r[j-dk].key > L.r[0].key))   {

L.r[j].key = L.r[j-dk].key;

j -= dk; }

   L.r[j].key = L.r[0].key;

}

}

}

输入待排序序列：49 38 65 97 13 27 49（以输入一个字符作为结束）

运行结果（写出每一趟的状态）：

38 49 65 97 13 27 49

13 38 49 65 97 27 49

13 27 38 49 65 97 49

13 27 38 49 49 65 97

3、在1题中补充快速排序算法。

算法代码：int Partition(SqList &L,int low,int high) {

 //分割区域函数  L.r[0] = L.r[low];

 int pivotkey = L.r[low].key;//一般将顺序表第一个元素作为支点  while(low < high)

 {

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

  L.r[low] = L.r[high];

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

  L.r[high] = L.r[low];  }

 L.r[low] = L.r[0];//返回枢轴位置

return low;

输入待排序序列：49 38 65 97 13 27 49（以输入一个字符作为结束）

运行结果（写出每一趟的状态）：

初始关键字 ： 49 38 65 97 13 27 49

完成一趟排序： {27 38 13} 49 {97 65 49}

分别进行快速排序： {13} 27 {38} {49 65} 97 49 65

有序序列：13 27 38 49 49 65 97

四、实验小结

请比较各个排序算法的性能。

时间性能上， 快速排序 > 堆排序 > 合并排序 > 插入排序 > 冒泡排序

而在交换次数上，合并排序 > 快速排序 > 堆排序 > 冒泡排序 > 插入排序