**实验报告（6.2）**

**姓 名： 鲁日中天 学 号： 2016211339**

**班 级： 2016211308 日 期：2018/1/2**

**分工情况：个人完成**

上机作业

# 实验6.2

目标：

按下述原则编写快排的非递归算法：  
(1)一趟排序之后，若子序列已有序(无交换)，则不参加排序，否则先对长度较短的子序列进行排序，且将另一子序列的上、下界入栈保存；  
(2)若待排记录数£3，则不再进行分割，而是直接进行比较排序之。

实验原理:

用非递归的快速排序，一趟排序之后，若子序列已有序(无交换)，则不参加排序，否则先对长度较短的子序列进行排序，且将另一子序列的上或下界入栈保存；若待排记录数<3，则不再进行分割，而是直接进行比较排序。

**操作步骤：**

1. **按照要求输入你需要输入的数字个数**
2. **然后开始输入数字，数字之间用空格隔开，输入完毕后回车**
3. **程序会按照排序序列输出**

**算法思想：**

1. **先定义一个类，里面有数组，数组的大小n , 然后所有的操作均为该类中的函数方法。**
2. **主函数里创建一个Array对象，然后调用其中的方法排序，输出**

**核心算法为：**

1. void sort(int head , int last) 当检测到序列中不足三个以上元素时，调用该函数，直接交换排序
2. void Qsort() 与 int GetPartion(int head , int last) 为快排中用到的函数，

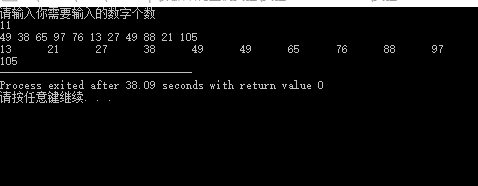
在Qsort()中，一趟排序之后，若子序列已有序(无交换)，则不参加排序，否则先对长度较短的子序列进行排序，且将另一子序列的上或下界入栈保存；若待排记录数<3，则不再进行分割，而是直接调用sort(head , last)进行比较排序。

而int GetPartion(int head , int last)的算法和之前的的并无大的区别

1. 另外，push(SS , head , last) , pop(SS , \*head , \*last) 用来出栈，入栈，

其中，push的head , last 为int 型 ， pop的为int\* 。

**测试截图：**

****

**问题体会：**

1. **一开始没太注意指针传值问题，在pop()函数中，把head,last直接传入而没有取地址，导致出错。**

## 源代码：

#include<stdio.h>

#include<iostream>

#include<stdlib.h>

#include<string.h>

using namespace std ;

class Array{

public:

int array[10000] ;

int n ; //存储数字的个数

struct stack{

int S[1000] ;

int top; //用来表示栈顶指针

};

struct stack SS; //定义一个栈

void push(struct stack \*SS , int head , int last){

SS->S[SS->top] = head ;

SS->top++ ;

SS->S[SS->top] = last ;

SS->top++ ;

}

void pop(struct stack \*SS , int \*head , int \*last){

SS->top-- ;

\*last = SS->S[SS->top] ;

SS->top-- ;

\*head = SS->S[SS->top] ;

}

Array(){ //构造函数

SS.top = 0 ;

cout<<"请输入你需要输入的数字个数"<<endl ;

cin>>n ;

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

cin>>array[i] ;

}

}

void swap(int head ,int last){ //交换函数

int temp = array[head] ; //设置中间暂存变量

array[head] = array[last] ;

array[last] = temp ;

}

void sort(int head , int last){

if(last - head == 1){ //子序列只含有两个元素

if(array[head] > array[last])

swap(head , last) ;

}

else{

if(array[head] > array[head+1])

swap(head , head+1) ;

if(array[head+1] > array[last])

swap(head+1 , last) ;

if(array[head] > array[head+1])

swap(head , head+1) ;

}

}

int GetPartion(int head , int last){ //快排时交换表中记录，使枢轴记录到位

int key = array[head] ;

while(head < last){

while(head < last && array[last] >= key){

last-- ;

}

swap(head , last) ;

while(head < last && array[head] <= key){

head++ ;

}

swap(head , last) ;

}

return head ;

}

void Qsort(){ //快排函数

int head = 0 , last = n-1 ;

while(head<last){

if(last - head > 2){ //当前子序列长度大于3且未排好序

int partion = GetPartion(head , last) ;

if(last-partion > partion-head){

push(&SS , partion+1 , last) ; //长的序列入栈

last = partion - 1 ; //短的留下

}

else{

push(&SS , head , partion-1) ;

head = partion + 1 ;

}

}

else if(head < last && last - head < 3){ //当前子序列长度小于3且未排序

sort(head , last) ; //进行比较排序

head = last ; //表示序列已经排好序

}

else{ //将剩下没排好序的子列出栈

pop(&SS , &head , &last) ; //将head 以及last从栈中取出

}

}

Qsort2(0 , n-1) ;

}

void Qsort2(int head , int last){

if(head < last)

{

int partion = GetPartion(head , last);

Qsort2(head , partion-1);

Qsort2(partion+1 , last);

}

}

void output(){

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

cout<<array[i]<<"\t" ;

}

}

};

main(){

Array test ;

test.Qsort() ;

test.output() ;

}