#include "stdio.h"

#include "stdlib.h"

#define MAX 10000

#define EQ(a,b) ((a)==(b))

#define LT(a,b) ((a)<(b))

#define LQ(a,b) ((a)<=(b))

//排序表的类型定义

typedef int KeyType;

typedef struct

{ KeyType key;

} RecType;

typedef struct

{ RecType r[100];

int length;

} SqList;

//交换数据

void Swap(int &a, int &b)

{

int temp = a;

a = b;

b = temp;

}

// 排序表的建立

void Create(SqList &L)

{ int i,n;

printf("\n请输入表长:"); scanf("%d",&n);

printf("请输入%d元素:",n);

L.r[0].key=-100;

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

scanf("%d",&(L.r[i].key));

L.length=n; }

// 遍历排序表(输出哨兵)

void Traverse(SqList &L)

{ int i;

printf("\n(%d)\t",L.r[0].key);

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

printf("%5d",L.r[i].key); }

//冒泡排序

void BubbleSort(SqList &L, int size)

{

int i, j;

for(i=1;i<=size-1;i++)

{

for(j=size;j>i;j--)

if(LT(L.r[j].key ,L.r[j-1].key))

Swap(L.r[j].key,L.r[j-1].key);

Traverse(L);}

}

//冒泡排序的改进算法

void BubbleSort2(SqList &L){

int start=1,end=L.length;

int i;

KeyType source[MAX];

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

source[i]=L.r[i].key;

while(start<=end) //如果还有元素没有确定其位置

{

for(i=start;i<end;i++) //寻找剩余元素的最大数

if(source[i]>source[i+1])

{

Swap(source[i],source[i+1]);

}

end--; //找到最大数

for(i=end;i>start;i--) //寻找剩余元素的最小元素

if(source[i]<source[i-1])

{

Swap(source[i],source[i-1]);

}

start++; //找到一个最小数

}

printf("%4d",source[i]);

}

// 快速排序

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

{ KeyType pivotkey;

L.r[0]=L.r[low]; //用子表的第一个记录做枢轴

pivotkey=L.r[low].key; //枢轴记录关键字

while(low<high) //从表的两端交替向中间扫描

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

L.r[low]=L.r[high]; //将比枢轴小的记录移动到低端

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

L.r[high]=L.r[low];} //将比枢轴大的记录移动到高端

L.r[low]=L.r[0]; //枢轴记录到位

Traverse(L);

return low; } //返回枢轴位置

//递归形式的快速排序

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

{ int pivotloc;

if (low<high) //长度大于1

{ pivotloc=Partition(L,low,high); //将记录一分为二

QSort(L,low,pivotloc-1); // 对低子表递归排序，pivotloc是枢轴的位置

QSort(L,pivotloc+1,high); } // 对高子表递归排序

}

//快速排序的改进算法

void Merge(SqList &L, int start, int mid, int end)

{

RecType temp1[MAX], temp2[MAX];

int n1, n2;

int i, j, k;

n1 = mid - start + 1; //将原记录分成两部分的第一段长度

n2 = end - mid; //将原记录分成两部分的第二段长度

for(i=0;i<n1;i++)

temp1[i] =L.r[start + i]; //将第一段里的元素存储到temp1[i]

for(i=0;i<n2;i++)

temp2[i] =L.r[mid + i + 1]; //将第二段里的元素存储到temp2[i]

temp1[n1].key = temp2[n2].key =MAX;//每段元素为赋最大值

i = j = 0;

for(k=start;k<=end;k++)

{

if(LT(temp1[i].key,temp2[j].key))

{

L.r[k] = temp1[i]; //比较两段里面的元素使基本有序

i++;

}

else

{

L.r[k] = temp2[j];

j++;

}

}

}

//递归形式进行排序

void RecMerge(SqList &L, int start, int end)

{

int i;

if(start < end)

{

i = (start + end) / 2; //取mid值

RecMerge(L, start, i); //对前一段排序

RecMerge(L, i+1, end); //对后一段排序

Merge(L, start, i, end); //合并排序

Traverse(L);

}

}

//合并排序

void MergeSort(SqList &L, int size)

{

RecMerge(L, 0, size);

Traverse(L);

}

// 简单选择排序

void SelectSort1(SqList &L)

{ int i,j,min;

for(i=0; i<=L.length; i++) //选择第i小的记录

{

min=i;

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

if(LT(L.r[j].key,L.r[min].key))

min = j; //在L.[i...L.length]中选择最小的记录

Swap(L.r[i].key, L.r[min].key);//与第i个交换

Traverse(L);

}

}

//简单选择排序的改进:二元选择排序

void SelectSort2(SqList &L)

{

int i ,j , min ,max,flag;

flag=0;

int n=L.length;

flag=L.length;

for (i=1;i<=int(n/2);i++)

{

min = i; max = i ;

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

{

if (LT(L.r[max].key,L.r[j].key))

{ max = j ; continue ;} //确认最大值位置

if (LT(L.r[j].key,L.r[min].key))

{ min = j; continue;} //确认最小值位置

}

if(max!=i||min!=flag)

{

if(min>=max&&max!=i)

{Swap(L.r[min].key,L.r[i].key);

Swap(L.r[max].key,L.r[flag].key);}

else

{Swap(L.r[max].key,L.r[flag].key);

Swap(L.r[min].key,L.r[i].key);}

}

else

Swap(L.r[max].key,L.r[min].key);

flag--; //将最大值最小值放入指定位置

Traverse(L);

}

}

//菜单

int MainMenue()

{

// fflush( stdin );

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menue \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\* \*\*\n");

printf("\*\* 1. BubbleSort \*\*\n");

printf("\*\* 2. BubbleSort2 \*\*\n");

printf("\*\* 3. QSort \*\*\n");

printf("\*\* 4. MergeSort \*\*\n");

printf("\*\* 5. SelectSort1 \*\*\n");

printf("\*\* 6. SelectSort2 \*\*\n");

printf("\*\* 7. Exit. \*\*\n");

printf("\*\* \*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//主函数

int main()

{

int flag = 1;

char ch[10];

SqList L;

Create(L);

while ( flag )

{

MainMenue();

printf("Please input your choice(1~7): ");

gets( ch );

switch ( ch[0] )

{

case '1':

BubbleSort(L,L.length); exit(0);

case '2':

BubbleSort2(L); exit(0);

case '3':

QSort(L,1,L.length); exit(0);

case '4':

MergeSort(L,L.length); exit(0);

case '5':

SelectSort1(L); exit(0);

case '6':

SelectSort2(L); exit(0);

case '7': exit(0);

default:

printf("Input error!\n"); break;

}

}

}