实验七　顺序表的排序实验

【实验目的】

掌握以顺序表组织待排序数据元素的排序方法，尤其是希尔排序、快速排序、归并排序。

【实验背景】

排序是将一组具有相同数据类型的数据元素调整为按关键字从小到大(或从大到小)排列的过程。

为方便算法的实现，我们将组织待排元素的顺序表的数据类型定义如下：

```
#define MAXSIZE 20
typedef struct{
    int   r[MAXSIZE +1];        // r[0]为工作单元或闲置
    int   length;               // length 为顺序表的长度
}SqeList;                  //顺序表类型
```

1、直接插入排序

直接插入排序的基本思想是：将整个数据表分成左右两个子表；其中左子表为有序表，右子表为无序表；整个排序的过程就是将右子表中的元素逐个插入到左子表中，直至右子表为空，而左子表成为新的有序表。

算法描述如下：

```
void InsertSort(SqeList *L){              //对顺序表 L 中的元素做直接插入排序
    for ( i=1；i< L->length；i++){
        L->r[0] = L->r[i]；j=i-1；     //将待插入元素存到监视哨 r[0]中
        while(L->r[0]< L->r[j]){       //  寻找插入位置
            L->r[j+1] = L->r[j]；j=j-1；
    }
        L->r[j+1] = L->r[0]；//将待插入元素插入到已排序的序列中
    }
}
```

2、希尔排序

基本思想：先将整个待排序元素序列分割成若干子序列，对每个子序列分别进行直接插入排序，当整个待排序元素序列“基本有序”时，再对全体元素进行一次直接插入排序。

将待排序元素序列调整为基本有序的方法是：选择一个步长值 d ，将元素下标之差为 d 的倍数的元素放在一组(子序列)，在每组内作直接插入排序。

算法描述如下：

```
void ShellInsert(SqeList *L，int di){
// 对顺序表 L 做一趟希尔插入排序，di 为该趟排序的增量
    for(i=1+di；i< L->length；i++)   // 1+di 为第一个子序列的第二个元素的下标
        if(L->r[i] < L->r[i-di]){
            L->r[0]= L->r[i]；//备份 L->r[i] (不做监视哨)
        for( j=i-di；(j>0) && (L->r[0]< L->r[j])；j- =di)
                L->r[j+di]= L->r[j]；
```

```
                L->r[j+di] = L->r[0];
            }
    }
    void ShellSort(SqeList *L，int delta[ ]，int n){
    //对顺序表 L 按增量序列 delta[0]～delta[n-1]做希尔排序
        for(i=0；i<=n-1；i++)
        ShellInsert( L，delta[i])；
    }
```

3、冒泡排序

基本思想是：两两比较待排序元素的关键字，发现它们次序相反时即进行交换，直到没有逆序的元素为止。

冒泡排序算法如下：

```
    void BubbleSort(SqeList *L){                    //对顺序表 L 做冒泡排序
        int i，j，n，change；
        int   x ；
        n= L->length；change=1；                // change 为记录元素交换的标志变量
        for(i=0；i< n-1 && change；++i ){    // 做 n-1 趟排序
            change=0；
            for ( j=0；j<n-i-1；++j)
                if (L->r[j]> L->r[j+1] ){
                    x= L->r[j]；L->r[j]= L->r[j+1]；
                    L->r[j+1]= x；change=1；
}}}
```

4、快速排序

基本思想：在待排序元素中选定一个作为“中间数”，使该数据表中的其他元素的关键字与“中间数”的关键字比较，将整个数据表划分为左右两个子表，其中左边子表任一元素的关键字不大于右边子表中任一元素的关键字；然后再对左右两个子表分别进行快速排序，直至整个数据表有序。

这里，我们选取数据表中的第一个数为“中间数”。

快速排序一次划分的算法描述为：

```
    int Partition( SqeList *H，int left，int right){
    //对顺序表 H 中的 H->r[left]至 H->r[right]部分进行快速排序的一次划分，返回划分后存放中间数的位置(基准位置)
        int   x；
        int   low，high；
        x= H->r[left]；                    // 选择中间数
        low=left；high=right；
        while ( low<high ){
            while (H->r[high]>=x&& low< high ) high--；
                //首先从右向左扫描，查找第一个关键字小于 x.key 的元素
        if ( low <high ){
                H->r[low]= H->r[high]；low++；
            }
```

```
        while (H->r[low]<x &&low<high)   low++;
        //然后从左向右扫描，查找第一个关键字不小于 x.key 的元素
        if ( low<high ){
            H->r[high]= H->r[low]；high--；
        }
    }
    H->r[low]=x；// 将中间数保存到 low=high 的位置
    return low；   // 返回存放中间数的位置
 }
```

对整个数据表进行快速排序，是在一次划分后，对左右子表分别进行快速排序。因此，整个排序过程是一个递归形式的算法。算法描述如下：

```
void QuickSort(SqeList *L，int low，int high){
//对顺序表 L 用快速排序算法进行排序
    int mid；
    if(low<high){
    mid = Partition(L，low，high)；
    QuickSort ( L，low，mid-1)；
    QuickSort ( L，mid+1，high)；
    }
 }
```

5、直接选择排序

直接选择排序的基本思想是：在第 i 趟直接选择排序中，通过 n-i 次关键字的比较，从 n-i+1 个元素中选出关键字最小的元素，与第 i 个元素进行交换。经过 n-1 趟比较，直到数据表有序为止。

直接选择排序的算法可以描述为：

```
void SelectSort(SqeList *L){      // 对顺序表 L 做直接选择排序
    int n，i；
  n=L->length；
    for(i=0；i<n-1；++i){
        k=i；
        for ( j=i+1；j<n；++j)
            if (L->r[j] < L->r[k]) k = j；
        if ( k! =i){
            x= L->r[i]；L->r[i]= L->r[k]；L->r[k]=x；
            }
    }
}
```

6、归并排序

最基本的归并是将两个有序表合并成一个有序表：

若有序表 A，B 同属于一个数据表 R，是 R 的两个有序区，并且 R[low] ~ R[mid] 为有序表 A 的存储区，R[mid＋1] ~ R[high]为有序表 B 的存储区。将其归并为一个有序表 C，存放在数组 R1 中。其算法描述如下：

```
void Merge(RecordType R[]，RecordType R1[]，int low，int mid，int high){
```

```
//数组 R[]中 R[low]～R[mid]和 R[mid+1]～R[high]分别按关键字有序排列，
//将它们合并成一个有序序列，存放在数组 R1 的 R1[low]～R1[high]中
    int i，j，k；
    i=low；j=mid+1；k=low；
    while ( i<=mid && j<=high ){
        if ( R[i]<=R[j]){
            R1[k] = R[i]；++i；
        }
        else{
            R1[k] = R[j]；++j；
        }
        ++k ；
    }
    while(i <= mid) R1[k++] = R[i++]；
    while(j <= mid) R1[k++] = R[j++]；
}
```

归并排序是利用上述归并技术来进行排序的。其基本思想是，将长度为 n 的待排序数据表看成是 n 个长度为 1 的有序表，将这些有序表两两归并，便得到$\lceil n/2 \rceil$个有序表；再将这$\lceil n/2 \rceil$个有序表两两归并，如此反复，直到最后得到长度为 n 的有序表为止。

这样，每次归并操作都是将两个有序表合并成一个有序表，我们称这种方法为 2 路归并排序。

2 路归并的一趟归并算法描述为：

```
void Mergepass(RecordType R[]，RecordType R1[]，int len){
// 对 R 作一趟归并，结果放在 R1 中
    int j，i=0；
    while (i+len*2 <= n){          // i+len*2–1>n-1 时不再排序
        Merge(R，R1，i，i+len-1，i+len*2-1)；
        i = i+len*2 ；
  }
    if (i+len-1 < n-1) Merge(R，R1，i，i+len-1，n-1 )；
    else
        for ( j=i；j<n；j++)
            R1[j] = R[j]；
}
```

对无序表进行 2 路归并排序就是调用上述一趟归并算法，对待排序序列进行若干趟归并。第一趟归并时，每个有序表的长度为 1，即 len＝1。归并后有序表的长度 len 就扩大一倍，即排序中 len 从 1，2，4…到$\lceil n/2 \rceil$，可以用 while 循环实现整个归并过程：

```
MergeSort (SqeList L ){
//对顺序表 L 中的待排序序列进行归并排序结果仍在顺序表 L 中
   int len，n；
   int   *R1；
   n = L.length；len = 1；
   R1= (int *)malloc(sizeof(int)*n)；
  while ( len <= n/2 + 1){
```

```
        Mergepass( L.r，R1，len)；// 一趟归并，结果在 R1 中
        len = 2*len ；
        Mergepass( R1，L.r，len)；// 再次归并，结果在 R 中
        len = 2*len ；
      }
        free(R1)；
  }
```

【实验任务】

1、程序验证

（1）建立含有若干个整数的顺序表，在此基础上实现顺序表的各种排序算法。

（2）为希尔排序设计建表函数和主函数，要求输出每一趟排序的结果，并通过运行来验证。

2、算法填空 阅读下列快速排序算法，请在一趟排序算法中的适当位置添加代码输出当前一次划分的结果，并通过运行来验证。

```
#include "stdio.h"
#define MAXSIZE 100
typedef struct{
    int key；
    int other；
}RecordType；
typedef struct{
    RecordType r[MAXSIZE+1]；
    int length；
}SqeList；

int Partition(SqeList *H，int left，int right){
    RecordType x；int low，high；x=H->r[left]；low=left；high=right；
    while(low<high){
        while(H->r[high].key>=x.key&& low< high )   high--；
        if(low<high )   {H->r[low]=H->r[high]；low++；}
        while(H->r[low].key<x.key &&low<high)      low++；
        if(low<high)     {H->r[high]= H->r[low]；high--；}
}
H->r[low]=x；return low；
}

void QuickSort(SqeList *L，int low，int high){
    int mid；
    if(low<high)   {
        mid=Partition(L，low，high)；
        QuickSort(L，low，mid-1)；
        QuickSort(L，mid+1，high)；
    }
}
void main(){
```

```
    SqeList L，*p=&L；int i；L.length=8；
    for(i=0；i<L.length；i++)
      scanf("%d"，&L.r[i].key)；
    printf("原数据为：")；
    for(i=0；i<L.length；i++)
      printf("%4d"，L.r[i].key)；
    QuickSort(p，0，7)；
    printf("\n 排序后为：")；
    for(i=0；i<L.length；i++)
      printf("%4d"，L.r[i].key)；
    printf("\n")；
}
```

3、算法设计

（1）改进冒泡排序，试编写一个程序，对一个待排序的数据元素集合进行奇偶转换排序。（奇偶排序是指第一趟对所有奇数的 i，将 a[i]与 a[i+1]进行比较，第二趟是对所有偶数的 i，将 a[i]与 a[i+1]进行比较，每次比较时，若 a[i]>a[i+1]，则将二者交换，重复上述两趟交换进行的过程，直到整个数据表有序。）

（2）在熟悉掌握直接插入排序的基础上，结合下述折半插入排序算法的描述，写出折半插入排序算法的实现过程（函数），并通过运行来验证。

折半插入排序算法构思：

① low=1；high=i-1；//有序表长度为 i-1，第 i 个元素为待插入元素
　　//设置有序表区间，待插入元素送辅助单元

② 若 low>high，得到插入位置，转⑤

③ 若 low≤high，m=(low+high)/2；//取表的中点，并将表一分为二，确定待插入区间

④ 若 r[0]<r[m]，则 high=m-1；//插入位置在左子表中
　否则，low=m+1；　　//插入位置在右子表中
　转②

⑤ high+1 即为待插入位置，从 i-1 到 high+1 的元素逐个后移，然后将 r[0]放在 r[high+1]位置，完成一趟插入排序。

（3）设计算法将一个整型数组调整为这样的数组：所有 3 的倍数在最左边，所有除以 3 余 1 的数在中间，而所有除以 3 余 2 的数在最左边。要求算法的时间尽可能少。

4、实例演练：内部排序算法比较

[问题描述]

各种内部排序算法的时间复杂度分析结果只给出了算法执行时间的阶，或大概执行时间。试通过随机的数据比较各算法的关键字比较次数和关键字移动次数，以取得直观感受。

[基本要求]

① 对以下 10 种常用的内部排序算法进行比较：直接插入排序；折半折入排序；二路插入排序；希尔排序；起泡排序；快速排序；简单选择排序；堆排序；归并排序；基数排序。

② 待排序表的表长不少于 100；其中的数据要用伪随机数产生程序产生；至少要用 5 组不同的输入数据作比较；比较的指标为有关键字参加的比较次数和关键字移动次数（关键字交换计为 3 次移动）。

[测试数据]

由随机产生器决定。

[实现提示]

主要工作是设法在程序中适当的地方插入计数操作。程序还可以包括计算几组数据得出结果波动大小的解释。注意分块调试的方法。