实验一　顺序表的基本操作及有序表的合并

2018 级　软件工程 2 班　王泉成　1825122045　2019.09.26

需求分析

问题描述

创建两个有序的顺序表L1和L2，表中元素值由键盘随机输入，再将它们合并为一个新的顺序表L3，合并后L3仍然有序（重复元素只保留一个），最后输出顺序表中的各个元素值

基本要求

包括创建顺序表、插入和删除指定序号的元素、读取表元、获取最大和最小值元素、查找元素、表元素的排序、表元素逆置、顺序表的输入和输出等等；

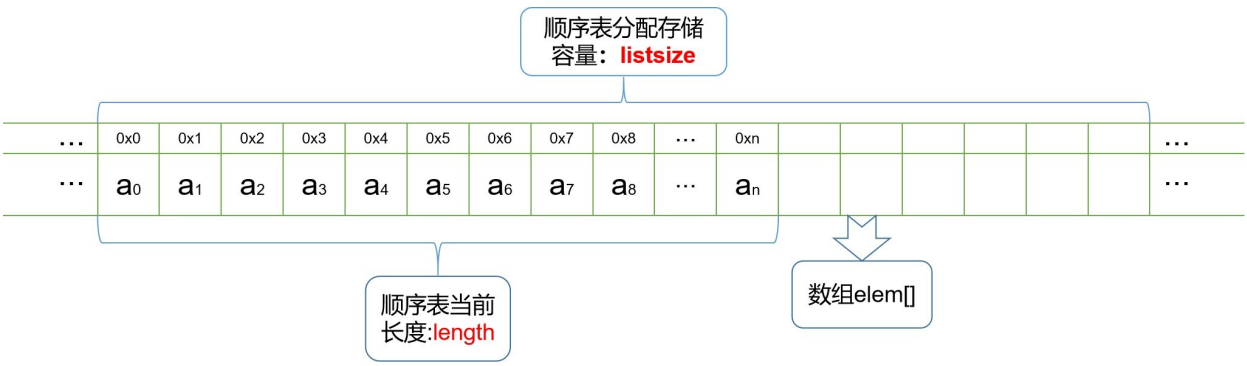
项目设计

数据结构

1.数组
2.顺序表

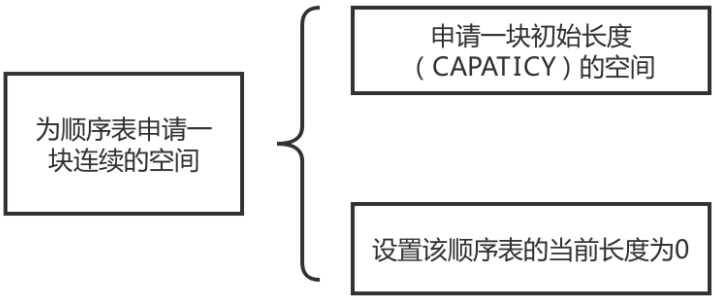
设计思路

1)顺序表的结构体



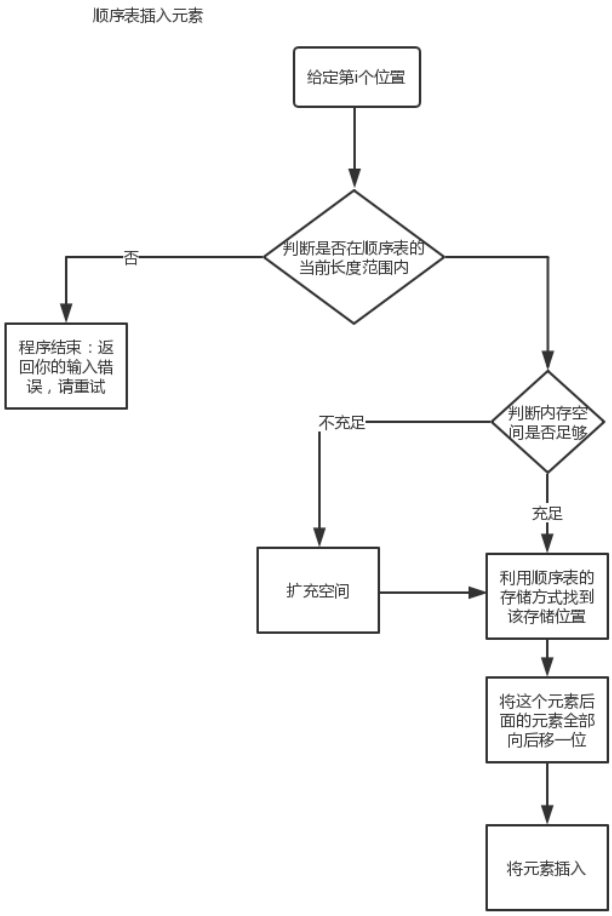

```
typedef struct{
    /*顺序表的数组*/
    ElemType *elem;
    /*当前数组的当前长度*/
    int length;
    /*当前数组的总容量*/
    int capacity;
 }SqList;
```

2)创建并初始化顺序表



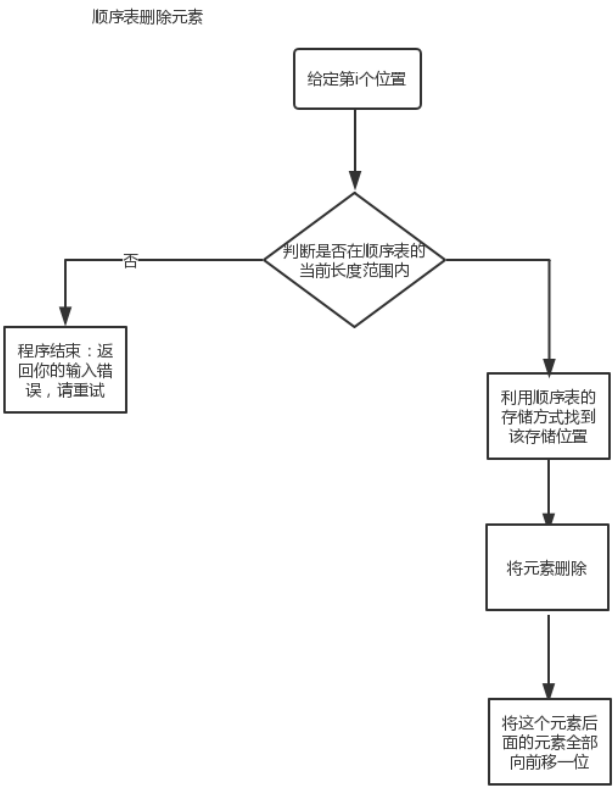

```
Status InitList(SqList &L){
    L.elem=(ElemType*)malloc(BASECAPACITY*sizeof(ElemType));
    L.capacity=BASECAPACITY;
    L.length=0;
    return OK;
   }
```

3)在顺序表插入指定序号i的元素(表示第i+1个元素）



```
Status InsertElem(SqList &L,int i,ElemType e){
  if(i>L.length || i<0){
     return FALSE;
           }
  if(L.length==L.capacity){
    ElemType *newelem;
    L.capacity+=DECAPACITY;
    newelem=(ElemType *)realloc(L.elem,sizeof(ElemType)*L.capacity);
    L.elem=newelem;
        }
    for(int n=0;n<L.length-i;n++){
    *(L.elem+L.length-n)=*(L.elem+L.length-1-n);
    }
    *(L.elem+i)=e;
    L.length+=1;
    return OK;
}
```

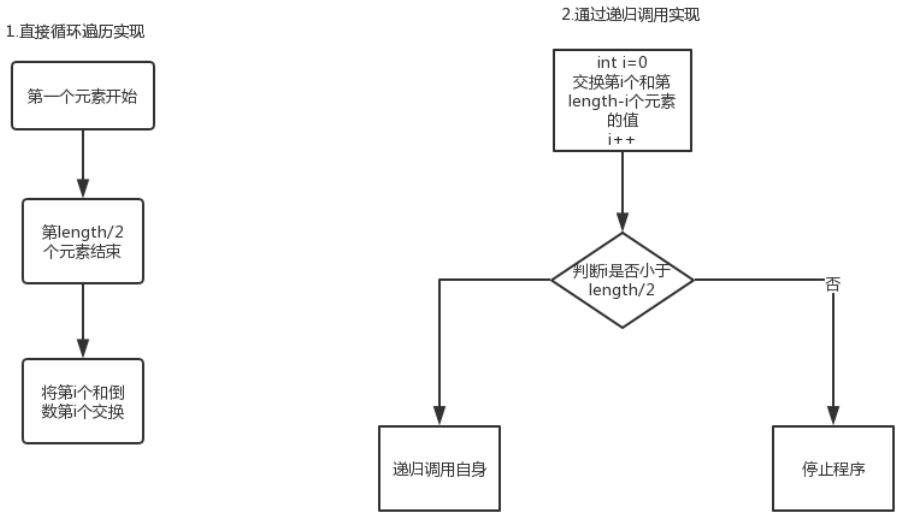
4)顺序表元素的插入




```
    ElemType DeleteElem(SqList &L,int i){
        ElemType e;
        if(i<0 || i>=L.length){
            return ERROR;
```

```
        }
    e=*(L.elem+i-1);
    for(int n=0;n<L.length-i;n++){
        *(L.elem+i+n)=*(L.elem+i+1+n);
    }
    L.length=L.length-1;
    return e;
}
```

5）顺序表的逆置



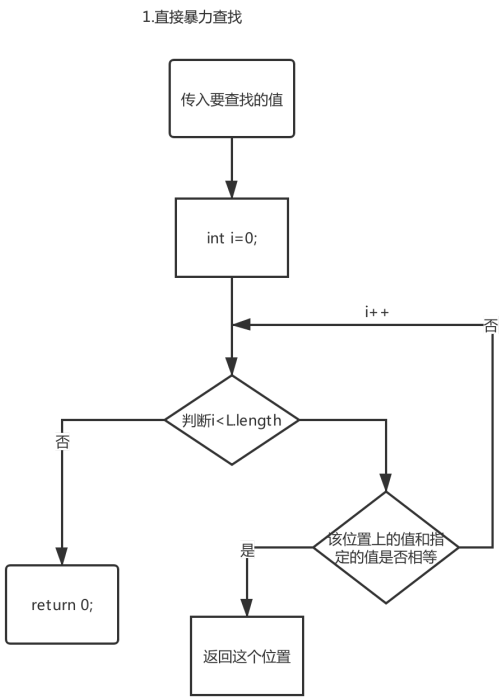

```
/*交换两个元素的方法*/
Status swap(ElemType &a,ElemType &b){
    int t=a;
    a=b;
    b=t;
    return OK;
}
/*将表元素进行逆置*/
void ElemInverse(SqList &L){
    for(int m=0;m<(L.length)/2;m++){
    swap(*(L.elem+m),*(L.length-m-1+L.elem));
    }
}
/*将表元素逆置方法二:递归实现*/
int i=0;
void ElemInverse_2(SqList &L){
    if(i<L.length-i){
        swap(*(L.elem+i),*(L.elem+L.length-i-1));
        i++;
        ElemInverse_2(L);
    }else{
```

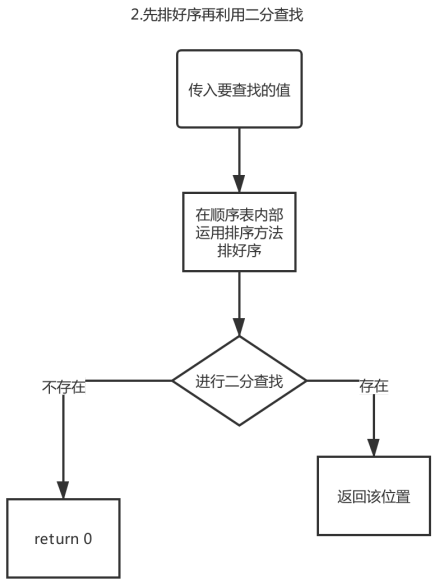
```
            return;
        }
    }
```

6）读取顺序表中的元素:直接遍历取值

```
/*读取表中元素的方法*/
 void visit(ElemType e){
      printf("%d  ",e);
    }
 /*读取顺序表中的元素*/
 Status ListTraverse(SqList L){
      for(int i=0;i<=L.length-1;i++){
         visit(L.elem[i]);
         }
         return OK;
    }
```

7)顺序表中元素的查找

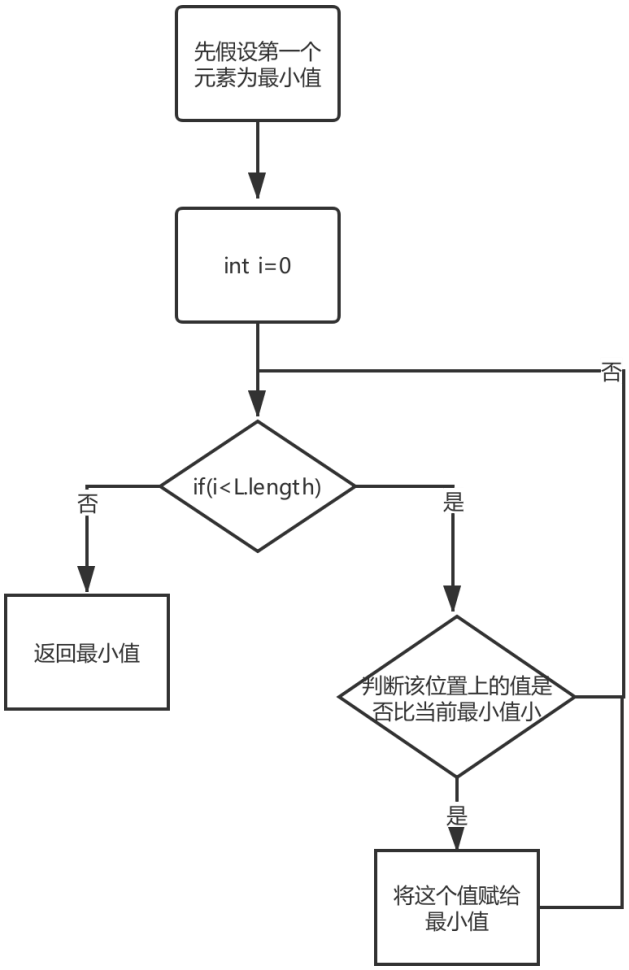





```
 void LocateElem(SqList L,ElemType e){
    bool exist=false;
    for(int i=1;i<L.length;i++){
        if(*(L.elem+i-1)==e){
            exist=true;
          }
    }
    if(exist==true){
        printf("该顺序表中存在%d这个元素\n",e);
    }else{
```

```
            printf("该顺序表中不存在%d这个元素\n",e);
        }
    }
```

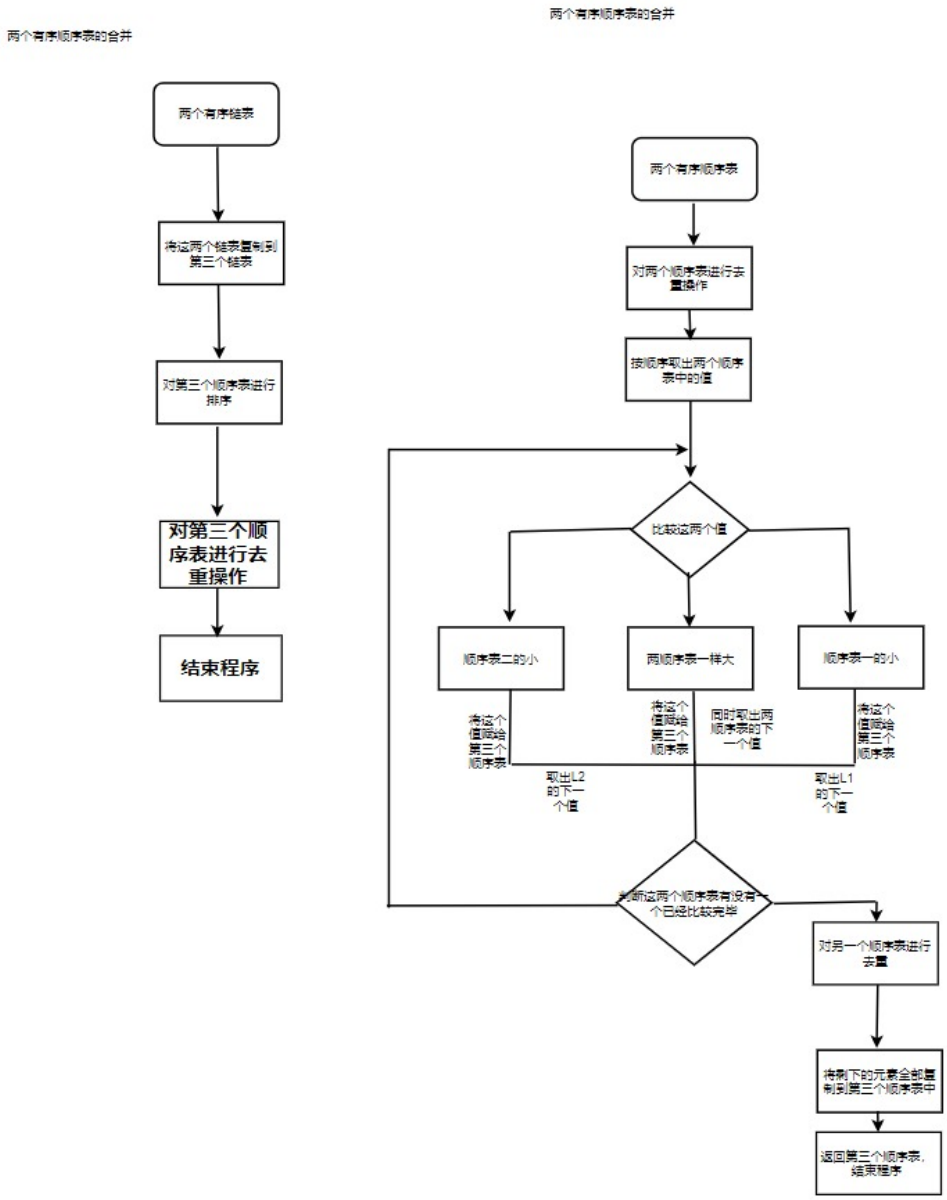
8)求顺序表的最大值和最小值




```
/*获取顺序表中元素最大值*/
ElemType MaxElem(SqList L){
    ElemType max;
    max=*(L.elem);
    for(int i=1;i<L.length;i++){
        if(*(L.elem+i)>max){
            max=*(L.elem+i);
           }
       }
       return max;
   }
/*获取顺序表中的最小值*/
ElemType MinElem(SqList L){
    ElemType min;
    min=*(L.elem);
    for(int i=1;i<L.length;i++){
        if(*(L.elem+i)<min){
            min=*(L.elem+i);
           }
```

```
        }
        return min;
    }
```

9)两个有序链表的合并



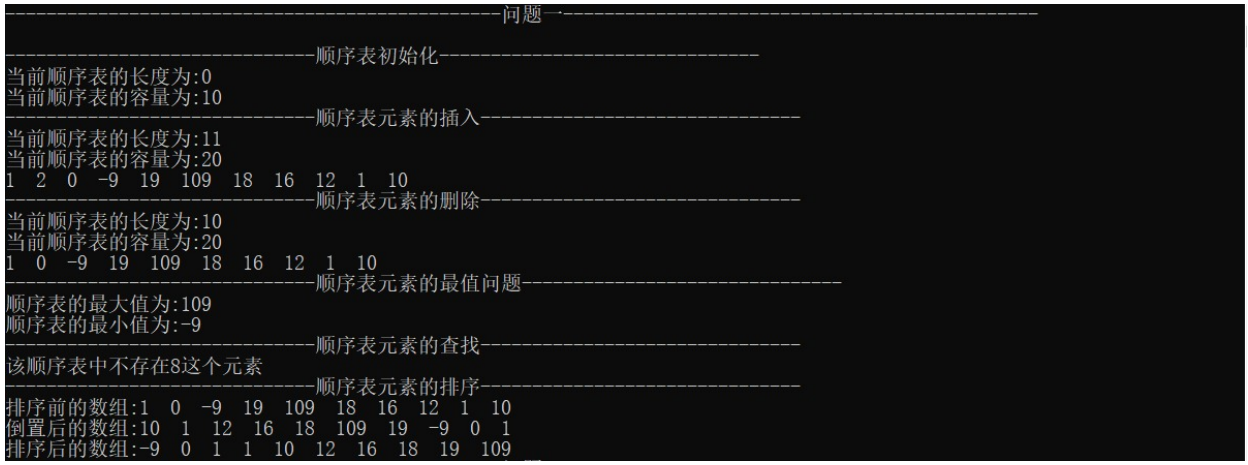

```
/*两个有序顺序表的简单合并(直接将两个数组合并就可以)*/
SqList UnionList(SqList L,SqList Q){
    SqList U;
    /*空间分配的大小为前两个顺序表的长度相加*/
    U.elem=(ElemType *)malloc((L.length+Q.length)*sizeof(ElemType));
    U.length=0;
    for(int i=0;i<L.length;i++){
        /*将第一个顺序表复制到第三个顺序表中*/
        *(U.elem+i)=*(L.elem+i);
        U.length=U.length+1;
    }
    for(int j=0;j<Q.length;j++){
        /*将第二个顺序表复制到第三个顺序表中*/
        *(U.elem+L.length+j)=*(Q.elem+j);
```

```
            U.length=U.length+1;
        }
        return U;
    }
    /*对于排好序的数组去除大小相同的元素*/
    Status NoSame(SqList &L){
        for(int i=1;i<L.length-1;i++){
            if(*(L.elem+i)==*(L.elem+i-1)){
                DeleteElem(L,i);
                /*每删除一个元素(第i个元素)后，还要将原顺序表的第(i)个元素和第(i+2)个元素进行对比，所以要进行减减操作*/
                i--;
            }
        }
        return OK;
    }
    /*边比较边排序*/
    SqList UnionList_2(SqList L,SqList Q){
        SqList U;
        /*空间分配的大小为前两个顺序表的长度相加*/
        U.elem=(ElemType *)malloc((L.length+Q.length)*sizeof(ElemType));
        U.capacity=L.length+Q.length;
        U.length=0;
        int i=0,j=0,k=0;
        for(;(*(L.elem+i)<*(L.elem+L.length))&&(*(j+Q.elem)<*(Q.elem+Q.length-1))&&(k<U.capacity);k++){

            if( (*(L.elem+i)==*(L.elem+i+1)) &&i<L.length-1){
            i=i+1;
            }

            if(( *(Q.elem+j)==*(Q.elem+j+1) )&&j<Q.length-1){
            j=j+1;
            }
            if(*(L.elem+i)<*(Q.elem+j)){
                *(U.elem+k)=*(L.elem+i);
                i++;
                U.length++;
            }else if(*(L.elem+i)>*(Q.elem+j)){
                *(U.elem+k)=*(Q.elem+j);
                j++;
                U.length++;
            }else{
                *(U.elem+k)=*(L.elem+i);
                U.length++;
                i++;
                j++;
            }
        }
        if(*(L.elem+i)==*(L.elem+L.length)){
            for(;j<Q.length;j++){
            if(*(Q.elem+j)==*(Q.elem+j+1))
```

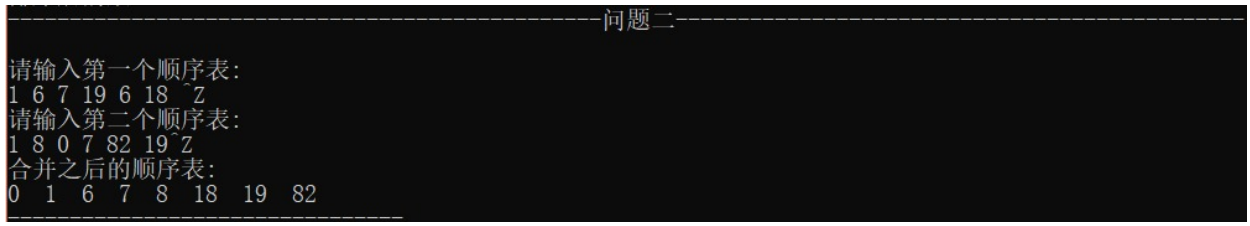
```
            j=j+1;
        *(U.elem+k)=*(Q.elem+j);
        k++;
        U.length++;
        }
    }else if(*(j+Q.elem)==*(Q.elem+Q.length)){
        for(;i<L.length;i++){
        if(*(L.elem+i)==*(L.elem+i+1))
            i=i+1;
        *(U.elem+k)=*(L.elem+i);
        k++;
        U.length++;
        }
    }
    return U;
}
/*实现两个有序顺序表的合并，合并后也有序，并且只保留顺序表中重复元素的一个*/
SqList Synthesis(){
    SqList L;
    SqList Q;
    InitList(Q);
    InitList(L);
    printf("请输入第一个顺序表:\n");
    InputElem(L);
    ElemBubbling(L);
    /*连续输入操作要相清空缓存区，才能进行在此输入*/
    fflush(stdin);
    printf("请输入第二个顺序表:\n");
    InputElem(Q);
    ElemBubbling(Q);
    SqList U=UnionList(L,Q);
    ElemBubbling(U);
    NoSame(U);
    return U;
}
 /*实现两个有序顺序表的合并，合并后也有序，并且只保留顺序表中重复元素的一个*/
SqList Synthesis_2(){
    SqList L;
    SqList Q;
    InitList(Q);
    InitList(L);
    printf("请输入第一个顺序表:\n");
    InputElem(L);
    ElemBubbling(L);
    /*连续输入操作要相清空缓存区，才能进行在此输入*/
    fflush(stdin);
    printf("请输入第二个顺序表:\n");
    InputElem(Q);
    ElemBubbling(Q);
    SqList U=UnionList_2(L,Q);
    return U;
}
```

测试结果

- 测试一



- 测试二



总结分析

回顾

在顺序表的插入和删除中，要让数据插入:总最后一个元素开始往后移，而删除时需要从下一个元素开始往前移，两个顺序表的合并时，利用第二种做法时，每插入一个元素第三个顺序表的当前长度都要加一，在判断自身链表前后两个元素时，应该用if不能用while

时间复杂度

排序算法:冒泡排序，时间复杂度为$O(n^2)$
查找:暴力查找，时间复杂度$O(n)$
最值:时间复杂度$O(n)$
逆置:交换实现，时间复杂度$O(n)$
合并顺序表：时间复杂度$O(n^2)$

改进设想

排序:归并排序
查找:先归并排序+二分查找
逆置:数组的复制
变量命名方面要多注意一些实际的含义

空间复杂度

空间复杂度为$O(n)$

经验和体会

1.当需要多次输入时，每一次输入完毕后必须先清空缓存区，才能接收下一次输入的数据
2.每次申请新节点时必须同时为其申请存储空间，否则数据无法存储
3.申请指针时不用申请节点空间，指针用完要释放对应的内存
4.每插入一个元素后必须要自身长度自加，删除元素必须要自身长度自减
5.申请指针还是申请节点还是申请顺序表要分清楚
6.多注意变量的命名规则
7.&字符的作用:①取别名②取地址
8.在需要在函数内改变链表L的时候，需要加&,不需要改变顺序表的时候就不要加