

Homework 9.21 9.23

1.14 判断下列函数 $f(n)$ 和 $g(n)$,当 $n \rightarrow \infty$ 时, 哪个函数增长最快?

(1) $f(n)=10^2+\ln(n!+10^{n^3})$ $g(n)=2n^4+n+7$

(2) $f(n)=(\ln(n!)+5)^2$ $g(n)=13n^{2.5}$

(3) $f(n)=n^{2.1} + \sqrt{n^4 + 1}$ $g(n)=(\ln(n!))^2+n$

(4) $f(n)=2^{(n^3)} + (2^n)^2$ $g(n)=2^{(n^2)} + n^5$

答:

(1) $\lim_{n \rightarrow \infty} \frac{10^2 + \ln(n! + 10^{n^3})}{2n^4 + n + 7} = 0$ $g(n)$ 增长快

```
1 In[1]:= Limit[(10^2 + Log[n! + 10^(n^3)])/(2 n^4 + n + 7), {n -> Infinity}]
2 Out[1]= 0
```

(2) $\lim_{n \rightarrow \infty} \frac{(\ln(n!)+5)^2}{13n^{2.5}} = 0$ $g(n)$ 增长快

```
1 In[2]:= Limit[(Log[n!] + 5)^2/(13 n^(2.5)), {n -> Infinity}]
2 Out[2]= 0
```

(3) $\lim_{n \rightarrow \infty} \frac{n^{2.1} + \sqrt{n^4 + 1}}{(\ln(n!))^2 + n} = \infty$ $f(n)$ 增长快

```
1 In[3]:= Limit[(n^2.1 + Sqrt[n^4 + 1])/((Log[n!])^2 + n), {n -> Infinity}]
2 Out[3]= \[Infinity]
```

(4) $\lim_{n \rightarrow \infty} \frac{2^{(n^3)} + (2^n)^2}{2^{(n^2)} + n^5} = \infty$ $f(n)$ 增长快

```
1 In[4]:= Limit[(2^(n^3) + (2^n)^2)/(2^(n^2) + n^5), {n -> Infinity}]
2 Out[4]= \[Infinity]
```

2.11 设顺序表 va 中的数据元素递增有序。试写一算法, 将 x 插入到顺序表的适当位置上, 以保持该表的有序性。

类C描述:

```
1 Status ListInsert(SqList &va, ElemType x){
2     //顺序表va非递减有序,插入x并保序, 0<=va.length<va.listsize
3     if(va.length >= va.listsize){           //存储空间已满
4         newbase = (ElemType*)realloc(va.elem,
5         (va.listsize + LISTINCREMENT)*sizeof(ElemType));
```

```

6         if(!newbase)    exit(OVERFLOW);    //空间分配失败
7         va.elem = newbase;
8         va.listsize += LISTINCREMENT;
9     }
10    i = va.length + 1;    //从后往前
11    while(i > 1 && x < va.elem[i-1]){
12        va.elem[i] = va.elem[i-1];    //边比较边移动
13        i--;
14    }
15    va.elem[i] = x;
16    ++va.length;
17    return OK;
18 }//ListInsert

```

完整C实现：（假设所有元素为整型）

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #define OK          1
4  #define ERROR       0
5  #define OVERFLOW    -2
6  #define LIST_INIT_SIZE 100    //初始分配容量
7  #define LISTINCREMENT 10    //分配增量
8  typedef int Status;
9
10 typedef struct {
11     int *elem;    //存储空间基址
12     int length;    //当前长度
13     int listsize;    //当前分配容量
14 }SqlList;
15
16 Status InitList(SqlList &va){//建表
17     va.elem = (int*)malloc(LIST_INIT_SIZE*sizeof(int));
18     if(!va.elem){
19         printf("建表出错\n");
20         exit(OVERFLOW);
21     }
22     va.length = 0;
23     va.listsize = LIST_INIT_SIZE;
24     return OK;
25 }
26
27 Status InputList(SqlList &va,int n){//输入
28     int i;
29     if(n < 1 || n > va.listsize){
30         printf("输入出错\n");
31         return ERROR;
32     }
33     printf("请按照非递减原则输入元素,以回车隔开: \n");
34     for(i = 1;i <= n;i++){
35         scanf("%d",&va.elem[i]);
36     }
37     va.length = n;
38     return OK;
39 }
40 Status OutputList(SqlList va,int n,int flag){//输出
41     int i;

```

```

42     if(n < 1 || n > va.listsize) return ERROR;
43     if(va.length == 0) return ERROR;
44     printf((flag++) == 1 ? "\n原表为: \n" : "\n插入后: \n");
45     for(i = 1; i <= n; i++)
46         printf("%d ", va.elem[i]);
47     return OK;
48 }
49 Status ListInsert(SqList &va, int x){//插入
50     int *newbase, i;
51     if(va.length >= va.listsize){                //存储空间已满
52         newbase = (int*)realloc(va.elem,
53             (va.listsize + LISTINCREMENT)*sizeof(int));
54         if(!newbase) exit(OVERFLOW);            //空间分配失败
55         va.elem = newbase;
56         va.listsize += LISTINCREMENT;
57     }
58     i = va.length + 1;
59     while(i > 1 && x < va.elem[i-1]){
60         va.elem[i] = va.elem[i-1];                //边比较边移动
61         i--;
62     }
63     va.elem[i] = x;
64     ++va.length;
65     return OK;
66 }
67
68 int main(){
69     SqList va;
70     int n, x;
71     InitList(va);
72     printf("请输入元素个数: ");
73     scanf("%d", &n);
74     InputList(va, n);
75     OutputList(va, n, 1);
76     printf("\n请输入带插入元素: ");
77     scanf("%d", &x);
78     ListInsert(va, x);
79     OutputList(va, n+1, 0);
80     return 0;
81 }

```

结果测试:

1、输入5个数据: 2 4 6 8 10, 插入7

```

请输入元素个数: 5
请按照非递减原则输入元素, 以回车隔开:
2
4
6
8
10
原表为:
2      4      6      8      10
请输入带插入元素: 7
插入后:
2      4      6      7      8      10

```

2、输入6个数据: 2 5 8 9 9 10, 插入9

```

请输入元素个数: 6
请按照非递减原则输入元素, 以回车隔开:
2
5
8
9
9
10

原表为:
2      5      8      9      9      10
请输入带插入元素: 9

插入后:
2      5      8      9      9      9      10

```

3、输入错误n=1000

```

请输入元素个数: 1000
输入出错

```

时空分析: **InitList**、**InputList**、**OutputList**的时间复杂度为 $O(n)$ ，**ListInsert**的最坏情况下时间复杂度为 $O(n)$ 。空间复杂度均为 $O(n)$ 。其中 n 为表长 $L.length$ 。

2.19 已知线性表中的元素以值递增有序排列，并以单链表作存储结构。试写一高效的算法，删除表中所有值大于 $mink$ 且小于 $maxk$ 的元素(若表中存在这样的元素)，同时释放被删结点空间，并分析你的算法的时间复杂度(注意: $mink$ 和 $maxk$ 是给定的两个参变量，它们的值可以和表中的元素相同，也可以不同)。

类C描述:

```

1  Status ListDelete(LinkList &L, int mink, int maxk){
2  //带头结点L, 非递减有序, 删除满足mink<x<maxk的元素并释放空间
3      q = L; p = L->next;          //辅助指针
4      if(!L || !p) return ERROR;
5      if(mink >= maxk) return ERROR;
6      if(p->data >= maxk) return OK; //链表最小值大于maxk
7      while(p){
8          if(p->data > mink && p->data < maxk){
9              t = p;
10             q->next = p->next;
11             p = p->next;
12             free(t);
13         }
14         else{
15             q = p;
16             p = p->next;
17         }
18     }
19     return OK;
20 }//ListDelete

```

完整C实现: (假设所有元素为整型)

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #define OK      1
4  #define ERROR   0
5  typedef int Status;
6
7  typedef struct LNode {
8      int      data;

```

```

9      struct LNode      *next;
10 }LNode,*LinkList;
11
12 Status CreateList(LinkList &L){//建表，带头结点，尾插法
13     LinkList p,rear;
14     int i,n;
15     L = (LinkList)malloc(sizeof(LNode));    //头结点
16     if(!L) return ERROR;
17     L->data = NULL; L->next = NULL;
18     rear = L;
19     printf("请输入元素的个数: ");
20     scanf("%d",&n);
21     printf("请按照非递减原则，输入元素: \n");
22     for(i = 0;i < n;i++){
23         p = (LinkList)malloc(sizeof(LNode));
24         if(!p) return ERROR;
25         p->data = NULL; p->next = NULL;
26         scanf("%d",&p->data);
27         rear->next = p;
28         rear = p;
29     }
30     rear->next = NULL;
31     return OK;
32 }
33
34 Status PrintList(LinkList L,int flag){//打印
35     LinkList p;
36     if(!L) return ERROR;
37     p = L->next;
38     printf((flag++)==1?"\n原表为: \n":"\n处理后: \n");
39     if(!p) printf("\n表已空\n");    //表已经删空
40     while(p){
41         printf("%d  ",p->data);
42         p = p->next;
43     }
44     printf("\n");
45     return OK;
46 }
47
48 Status ListDelete(LinkList &L,int mink,int maxk){//删除
49     LinkList p,q,t;
50     q = L;  p = L->next;    //辅助指针
51     if(!L || !p) return ERROR;
52     if(mink >= maxk){
53         printf("阈值输错\n");
54         return ERROR;
55     }
56     if(p->data >= maxk) return OK;    //链表最小值大于maxk
57     while(p){
58         if(p->data > mink && p->data < maxk){
59             t = p;
60             q->next = p->next;
61             p = p->next;
62             free(t);
63         }
64         else{
65             q = p;
66             p = p->next;

```

```

67     }
68 }
69 return OK;
70 }
71
72 int main(){
73     LinkList L;
74     int n,mink,maxk;
75     CreateList(L);
76     PrintList(L,1);
77     printf("请输入mink和maxk: \n");
78     scanf("%d%d",&mink,&maxk);
79     ListDelete(L,mink,maxk);
80     PrintList(L,0);
81     return 0;
82 }

```

结果测试:

- 1、输入6个数据: 1 5 7 8 10 15, 阈值为3 9
- 2、输入5个数据: 22 44 56 78 91, 阈值为1 100
- 3、输入3个数据: 5 6 7, 阈值为1 3
- 4、输入错误阈值100 10

<pre> 请输入元素的个数: 6 请按照非递减原则, 输入元素: 1 5 7 8 10 15 原表为: 1 5 7 8 10 15 请输入mink和maxk: 3 9 处理后: 1 10 15 </pre>	<pre> 请输入元素的个数: 5 请按照非递减原则, 输入元素: 23 44 56 78 91 原表为: 23 44 56 78 91 请输入mink和maxk: 1 100 处理后: 表已空 </pre>	<pre> 请输入元素的个数: 3 请按照非递减原则, 输入元素: 5 6 7 原表为: 5 6 7 请输入mink和maxk: 1 3 处理后: 5 6 7 </pre>	<pre> 请输入元素的个数: 1 请按照非递减原则, 输入元素: 1 原表为: 1 请输入mink和maxk: 100 10 阈值输错 处理后: 1 </pre>
---	--	--	--

时空分析: CreateList、PrintList的时间复杂度为 $O(n)$, ListDelete的最坏情况下时间复杂度为 $O(n)$ 。空间复杂度均为 $O(n)$ 。其中 n 为表长。

2.21 试写一算法, 实现顺序表的就地逆置,即利用原表的存储空间将线性表 (a_1, a_2, \dots, a_n) 逆置为 $(a_n, a_{n-1}, \dots, a_1)$ 。

类C描述:

```

1 Status ListInverse(SqList &L){
2     //顺序表就地逆置, 不另开空间
3     if(L.length == 0) return ERROR;
4     j = L.length;
5     for(i = 1; i < j; i++)
6         L.elem[i] <-> L.elem[j--]; //两端向中间进行
7     return OK;
8 } //ListInverse

```

完整C实现: (假设所有元素为整型)

```

1 #include<stdio.h>

```

```

2  #include<stdlib.h>
3  #define OK          1
4  #define ERROR       0
5  #define OVERFLOW    -2
6  #define LIST_INIT_SIZE 100    //初始分配容量
7  #define LISTINCREMENT 10     //分配增量
8  typedef int Status;
9
10 typedef struct {
11     int *elem;    //存储空间基址
12     int length;   //当前长度
13     int listsize; //当前分配容量
14 }SqList;
15
16 Status InitList(SqList &L){//建表
17     L.elem = (int*)malloc(LIST_INIT_SIZE*sizeof(int));
18     if(!L.elem){
19         printf("建表出错\n");
20         exit(OVERFLOW);
21     }
22     L.length = 0;
23     L.listsize = LIST_INIT_SIZE;
24     return OK;
25 }
26
27 Status InputList(SqList &L,int n){//输入
28     int i;
29     if(n < 1 || n > L.listsize){
30         printf("输入出错\n");
31         return ERROR;
32     }
33     printf("请输入元素,以回车隔开: \n");
34     for(i = 1;i <= n;i++){
35         scanf("%d",&L.elem[i]);
36     }
37     L.length = n;
38     return OK;
39 }
40
41 Status OutputList(SqList L,int n,int flag){//输出
42     int i;
43     if(n < 1 || n > L.listsize) return ERROR;
44     if(L.length == 0) return ERROR;
45     printf((flag++) == 1 ? "\n原表为: \n" : "\n逆置后: \n");
46     for(i = 1;i <= n;i++){
47         printf("%d ",L.elem[i]);
48     }
49     return OK;
50 }
51
52 Status ListInverse(SqList &L){//就地逆置
53     int temp,i,j;
54     if(L.length == 0) return ERROR;
55     j = L.length;
56     for(i = 1;i < j;i++){
57         temp = L.elem[i];
58         L.elem[i] = L.elem[j];
59         L.elem[j--] = temp;
60     }
61     return OK;

```

```

60 }
61
62 int main(){
63     SqList L;
64     int n;
65     InitList(L);
66     printf("请输入元素个数: ");
67     scanf("%d",&n);
68     InputList(L,n);
69     OutputList(L,n,1);
70     ListInverse(L);
71     OutputList(L,n,0);
72     return 0;
73 }

```

结果测试:

1、输入5个数据: 2 6 17 10 8

```

请输入元素个数: 5
请输入元素,以回车隔开:
2
6
17
10
8

原表为:
2      6      17      10      8
逆置后:
8      10      17      6      2

```

2、输入6个数据: 6 9 10 21 55 4

```

请输入元素个数: 6
请输入元素,以回车隔开:
6
9
10
21
55
4

原表为:
6      9      10      21      55      4
逆置后:
4      55      21      10      9      6

```

3、输入错误 n=1000

```

请输入元素个数: 1000
输入出错

```

时空分析: InitList、InputList、OutputList的时间复杂度为 $O(n)$, ListInverse的时间复杂度为 $O(\frac{n}{2})$ 。空间复杂度均为 $O(n)$ 。其中n为表长L.length。

2.24 假设有两个按元素值递增有序排列的线性表A和B,均以单链表作存储结构,请编写算法将A表和B表归并成一个按元素值递减有序(即非递增有序,允许表中含有值相同的元素)排列的线性表C,并要求利用原表(即A表和B表)的结点空间构造C。

类C描述:

```

1 Status MergeList(LinkList &A,LinkList &B,LinkList &C){
2     //将两带头结点的非递减单链表AB,合并为带头结点的非递增单链表C,使用原表结点空间。
3     if(!A || !B || (!A->next && !B->next))
4         return ERROR;

```



```

5     pa = qa = A;    pb = qb = B;
6     pa = pa->next;  pb = pb->next;
7     A->next = NULL; C = A; //用A的头结点充当C的头结点
8     while(pa && pb){
9         if(pa->data < pb->data){ //将最小值入A
10            qa = pa;
11            pa = pa->next;
12            qa->next = A->next;
13            A->next = qa;
14        }
15        else{
16            qb = pb;
17            pb = pb->next;
18            qb->next = A->next;
19            A->next = pb;
20        }
21    } //while
22    while(pa){ //如果pa剩余
23        qa = pa;
24        pa = pa->next;
25        qa->next = A->next;
26        A->next = qa;
27    }
28    while(pb){ //如果pb剩余
29        qb = pb;
30        pb = pb->next;
31        qb->next = A->next;
32        A->next = pb;
33    }
34    pb = B; free(pb);
35    return OK;
36 } //MergeList

```

完整C实现:

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #define OK          1
4  #define ERROR       0
5  typedef int Status;
6
7  typedef struct LNode {
8      int data;
9      struct LNode *next;
10 }LNode,*LinkList;
11
12 Status CreateList(LinkList &L){ //建表, 带头结点
13     LinkList p,rear;
14     int i,n;
15     L = (LinkList)malloc(sizeof(LNode));
16     if(!L) return ERROR;
17     L->data = NULL; L->next = NULL;
18     rear = L;
19     printf("请输入元素的个数: ");
20     scanf("%d",&n);
21     printf("请按照非递减原则, 输入元素: \n");
22     for(i = 0; i < n; i++){

```

```

23     p = (LinkedList)malloc(sizeof(LNode));
24     if(!p) return ERROR;
25     p->data = NULL; p->next = NULL;
26     scanf("%d",&p->data);
27     rear->next = p;
28     rear = p;
29 }
30 rear->next = NULL;
31 return OK;
32 }
33
34 Status PrintList(LinkedList L,int flag){
35     LinkedList p;
36     if(!L) return ERROR;
37     p = L->next;
38     switch(flag){
39         case 1: printf("表A为: \n");break;
40         case 2: printf("表B为: \n");break;
41         case 3: printf("表C为: \n");break;
42         default:break;
43     }
44     while(p){
45         printf("%d ",p->data);
46         p = p->next;
47     }
48     printf("\n");
49     return OK;
50 }
51
52 Status MergeList(LinkedList &A,LinkedList &B,LinkedList &C){
53     LinkedList pa,qa,pb,qb;
54     if(!A || !B || (!A->next && !B->next))
55         return ERROR;
56     pa = A; qa = pa;
57     pb = B; qb = pb;
58     pa = pa->next;
59     pb = pb->next;
60     A->next = NULL;
61     C = A;//用A的头结点充当C的头结点
62     while(pa && pb){
63         if(pa->data < pb->data){//将最小值入A
64             qa = pa;
65             pa = pa->next;
66             qa->next = A->next;
67             A->next = qa;
68         }
69         else{
70             qb = pb;
71             pb = pb->next;
72             qb->next = A->next;
73             A->next = qb;
74         }
75     }//while
76     while(pa){//如果pa剩余
77         qa = pa;
78         pa = pa->next;
79         qa->next = A->next;
80         A->next = qa;

```

```

81     }
82     while(pb){//如果pb剩余
83         qb = pb;
84         pb = pb->next;
85         qb->next = A->next;
86         A->next = qb;
87     }
88     pb = B; free(pb);
89     return OK;
90 }//MergeList
91
92 int main(){
93     LinkList A,B,C;
94     CreateList(A);
95     CreateList(B);
96     PrintList(A,1);
97     PrintList(B,2);
98     MergeList(A,B,C);
99     PrintList(C,3);
100    return 0;
101 }

```

结果测试:

- 1、输入A为3 5 7 9 11, B为2 4 6:
- 2、输入A为3 6, B空:

```

请输入元素的个数: 5
请按照非递减原则, 输入元素:
3
5
7
9
11
请输入元素的个数: 3
请按照非递减原则, 输入元素:
2
4
6
表A为:
3      5      7      9      11
表B为:
2      4      6
表C为:
11     9      7      6      5      4      3      2

请输入元素的个数: 2
请按照非递减原则, 输入元素:
3
6
请输入元素的个数: 0
请按照非递减原则, 输入元素:
表A为:
3      6
表B为:
表C为:
6      3

```

时空分析: CreateList、PrintList的时间复杂度为 $O(n)$, 其中 n 为表长。MergeList的最坏情况下时间复杂度为 $O(n)$, 其中 n 为A, B表长之和。

2.29 已知A,B和C为三个递增有序的线性表, 现要求对A表作如下操作: 删去那些既在B表中出现又在C表中出现的元素。试对顺序表编写实现上述操作的算法, 并分析你的算法的时间复杂度(注意:题中没有特别指明同一表中的元素值各不相同)。

类C描述:

```

1  Status DeleteList(Sqlist &A,Sqlist B,Sqlist C){
2      if(!A.length) return ERROR;
3      for(i = 0;i < A.length;i++)
4          for(j = 0;j < B.length;j++)
5              if(A.elem[i] == B.elem[j])
6                  for(k = 0;k < C.length;k++)
7                      if(A.elem[i] == C.elem[k]){

```

```

8         for(t = i;t < A.length;t++)
9             A.elem[t] = A.elem[t + 1];
10        A.elem[t + 1] = NULL;
11        --A.length;
12        --i;
13    }
14 }

```

完整C实现:

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #define OK          1
4  #define ERROR       0
5  #define OVERFLOW    -2
6  #define LIST_INIT_SIZE 100    //初始分配容量
7  #define LISTINCREMENT 10     //分配增量
8  typedef int Status;
9
10 typedef struct {
11     int *elem;    //存储空间基址
12     int length;   //当前长度
13     int listsize; //当前分配容量
14 }SqList;
15
16 Status InitList(SqList &L){//建表
17     L.elem = (int*)malloc(LIST_INIT_SIZE*sizeof(int));
18     if(!L.elem){
19         printf("建表出错\n");
20         exit(OVERFLOW);
21     }
22     L.length = 0;
23     L.listsize = LIST_INIT_SIZE;
24     return OK;
25 }
26
27 Status InputList(SqList &L){//输入
28     int i,n;
29     printf("请输入元素个数: ");
30     scanf("%d",&n);
31     if(n < 1 || n > L.listsize){
32         printf("输入出错\n");
33         return ERROR;
34     }
35     printf("请按照非递减原则输入元素,以回车隔开: \n");
36     for(i = 0;i < n;i++)
37         scanf("%d",&L.elem[i]);
38     L.length = n;
39     return OK;
40 }
41
42 Status OutputList(SqList L,int flag){//输出
43     int i,n;
44     n = L.length;
45     if(n < 1 || n > L.listsize) return ERROR;
46     if(n == 0) return ERROR;
47     switch(flag){

```

```

48         case 1: printf("\n表A为: \n");break;
49         case 2: printf("\n表B为: \n");break;
50         case 3: printf("\n表C为: \n");break;
51         case 4: printf("\n处理后: \n");break;
52         default:break;
53     }
54     if(!L.elem[0]) printf("\n表已空\n");
55     for(i = 0;i < n;i++)
56         printf("%d ",L.elem[i]);
57     return OK;
58 }
59
60 Status DeleteList(Sqlist &A,Sqlist B,Sqlist C){
61     int i,j,k,t;
62     if(!A.length) return ERROR;
63     for(i = 0;i < A.length;i++)
64         for(j = 0;j < B.length;j++)
65             if(A.elem[i] == B.elem[j])
66                 for(k = 0;k < C.length;k++)
67                     if(A.elem[i] == C.elem[k]){
68                         for(t = i;t < A.length - 1;t++)
69                             A.elem[t] = A.elem[t + 1];
70                         A.elem[t + 1] = NULL;
71                         --A.length;
72                         --i;
73                     }
74     return OK;
75 }
76
77 int main(){
78     Sqlist A,B,C;
79     int n,x;
80     InitList(A);
81     InitList(B);
82     InitList(C);
83     InputList(A);
84     InputList(B);
85     InputList(C);
86     OutputList(A,1);
87     OutputList(B,2);
88     OutputList(C,3);
89     DeleteList(A,B,C);
90     OutputList(A,4);
91     return 0;
92 }

```

结果测试:

- 1、输入A为3 4 5, B为4 6, C为4 8 9
- 2、输入A为3 4 5 5 6, B为5 7 8 9 11, C为5
- 3、输入A为6 9, B为7 7, C为5 8

请输入元素个数: 3 请按照非递减原则输入元素, 以回车隔开: 3 4 5 请输入元素个数: 2 请按照非递减原则输入元素, 以回车隔开: 4 6 请输入元素个数: 3 请按照非递减原则输入元素, 以回车隔开: 4 8 9 表A为: 3 4 5 表B为: 4 6 表C为: 4 8 9 处理后: 3 5	请输入元素个数: 5 请按照非递减原则输入元素, 以回车隔开: 3 4 5 5 6 请输入元素个数: 5 请按照非递减原则输入元素, 以回车隔开: 5 7 8 9 11 请输入元素个数: 1 请按照非递减原则输入元素, 以回车隔开: 5 表A为: 3 4 5 5 6 表B为: 5 7 8 9 11 表C为: 5 处理后: 3 4 6	请输入元素个数: 2 请按照非递减原则输入元素, 以回车隔开: 6 9 请输入元素个数: 2 请按照非递减原则输入元素, 以回车隔开: 7 7 请输入元素个数: 2 请按照非递减原则输入元素, 以回车隔开: 5 8 表A为: 6 9 表B为: 7 7 表C为: 5 8 处理后: 6 9
--	---	--

时空分析: InitList、InputList、OutputList的时间复杂度为 $O(n)$, DeleteList的时间复杂度为 $O(n^4)$ 。空间复杂度均为 $O(n)$ 。

- 回过头来发现, 本算法没有利用到A,B,C的非递减性质, 且时间复杂度过高, 现进行优化:

```

1  Status DeleteList_New(Sqlist &A, Sqlist B, Sqlist C){
2      while(i < A.length && j < B.length && k < C.length){
3          if(B.elem[j] < C.elem[k])    j++; //推进
4          else if(B.elem[j] > C.elem[k]) k++;
5          else{
6              x = B.elem[j]; //找到B, C相同值
7              while(B.elem[j] == x) j++;
8              while(C.elem[k] == x) k++;
9              while(i < A.length && A.elem[i] < x)
10                 A.elem[t++] = A.elem[i++]; //移动到待删值
11                 while(i < A.length && A.elem[i] == x) i++; //删除
12             }
13         }
14         while(i < A.length) A.elem[t++] = A.elem[i++]; //处理剩余元素
15         A.length = t;
16         while(t < i)    A.elem[t++] = NULL;
17         return OK;
18     }

```

时间复杂度为 $O(n)$, $n = \min\{A.length, B.length, C.length\}$ 。