一、调试成功程序及说明

1、

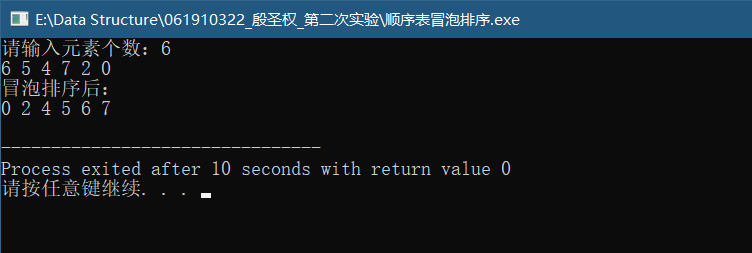
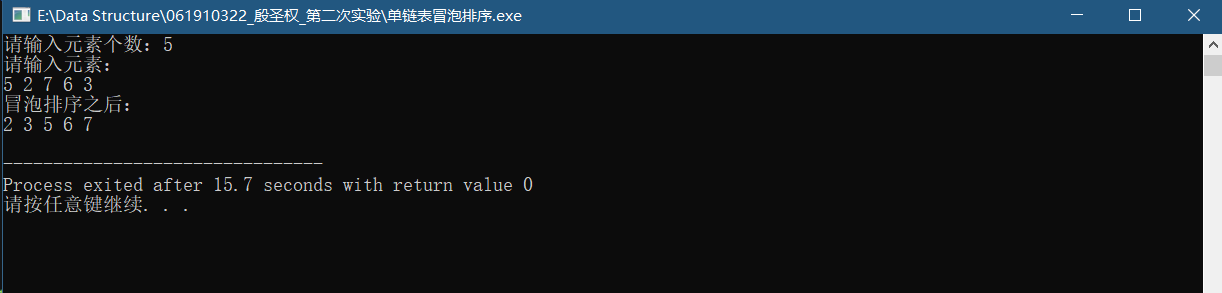
题目：

设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写函数，用选择/冒泡排序算法实现线性表的表排序。

算法思想：

进行n-1次冒泡，相邻元素对比，大的元素向后移动，再继续和相邻元素对比，每次冒泡都把最大的元素放置表尾。

运行结果：



结果分析：时间复杂度：O（n^2）

附源程序。

2、

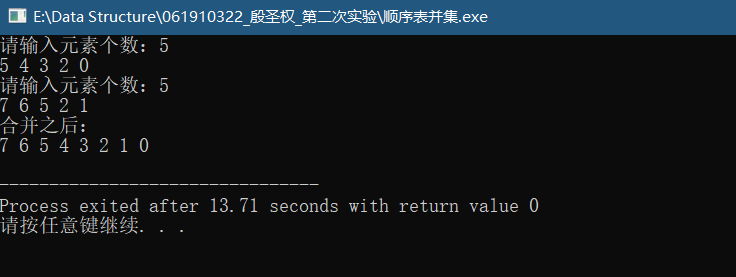
题目：

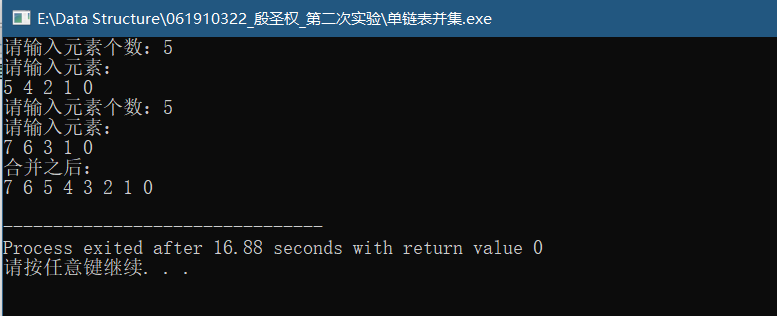
设线性表A、B，元素值为整型，且递减有序，编写函数，实现下列功能：对采用顺序结构和链式结构2种存储结构，要求在A的空间上构成一个新线性表C，其元素为A和B元素的并集，且表C中的元素值递减有序（互不相同）。

算法思想：

先删除a表中重复的元素，然后依次将b表中的元素与a表中的比较，将b元素插入a表中合适的位置

运行结果：





结果分析：时间复杂度：O（n^2）

附源程序。

3、

题目：

输入正整数n、m（m<n），设有n个人坐成一圈，从第1个人开始循环报数，报到m的人出列，然后再从下一个人开始报数，报到m的人又出列，如此重复，直到所有的人都出列为止。要求用链式结构和顺序结构实现，按出列的先后顺序输出每个人的信息。

算法思想：

顺序表：

Status numberOff(SeqList L, int m)

{

int n = 1, i = 0;

while(L.length != 1)//当只剩一个元素时停止循环

{

while(n != m)//从1报数直至报到m值

{

i++;

n++;

if(i >= L.length)//若报数报到最后一个元素则从头开始

{

i -= L.length;

}

}

n = 1;

printf("%d ", L.pData[i]);//输出恰好报到m值的元素值

DeleteElem(L, i+1);//删除恰好报到m值的元素

}

printf("%d \n", L.pData[0]);//输出最后一个元素值

return OK;

}

循环链表：

Status numberOff(ListInfo L, int m)

{

LNode \*pre = L.head;//当前结点前驱

LNode \*p = L.head->next;//当前结点

if(p == L.head) return ERROR;//空表，直接返回

while(L.head->next != L.head)//空表时停止

{

int i = 1;

while(i < m)

{

//向前报数

pre = p;

p = p->next;

i++;

if(p == L.head)//跳过表头

{

pre = p;

p = p->next;

}

}

//输出当前结点并删除

printf("%d ", p->data);

pre->next = p->next;

free(p);

p = pre->next;

if(p == L.head)//跳过表头

{

pre = p;

p = p->next;

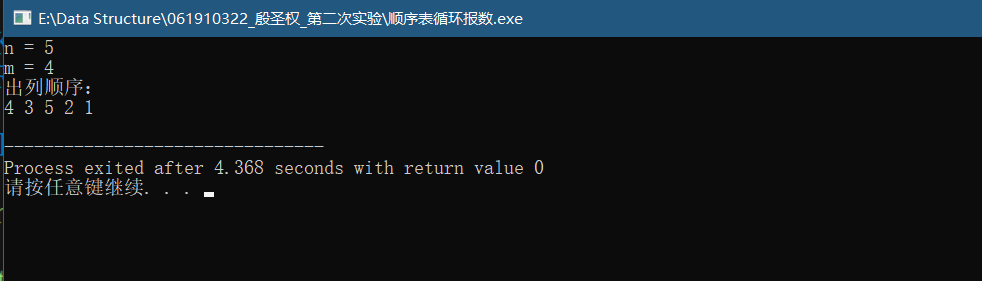
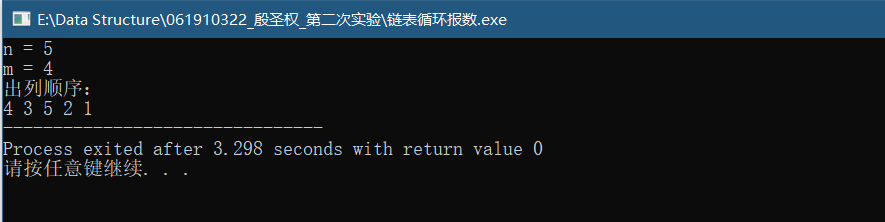
}

}

return OK;

}

运行结果：



结果分析：时间复杂度：O（n\*m）

附源程序。

4、

题目：

CSP题目

题目描述：小明在他的果园里种了一些苹果树，这些苹果树排列成一个圆。为了保证苹果的品质，在种植过程中要进行疏果操作。为了更及时地完成疏果操作，小明会不时地检查每棵树的状态，根据需要进行疏果。检查时，如果发现可能有苹果从树上掉落，小明会重新统计树上的苹果个数（然后根据之前的记录就可以判断是否有苹果掉落了)。在全部操作结束后，请帮助小明统计相关的信息。

输入格式：

从标准输入读入数据。

第1行包含一个正整数N，表示苹果树的棵数。

第1+i行（1≤i≤N)，每行的格式为mi,ai1,ai2,... ,aimi,。其中，第一个正整数mi表示本行后面的整数个数。后续的mi个整数表示小明对第i棵苹果树的操作记录。若aij (1≤ j ≤mi）为正整数，则表示小明进行了重新统计该棵树上的苹果个数的操作，统计的苹果个数为aij;若为零或负整数，则表示一次疏果操作，去掉的苹果个数是|aij|。

输入保证一定是正确的，满足:

1. ai1>0，即对于每棵树的记录，第一个操作一定是统计苹果个数（初始状态，此时不用判断是否有苹果掉落）；
2. 每次疏果操作保证操作后树上的苹果个数仍为正。

输出格式：

输出到标准输出。

输出只有一行，包含三个整数T、D、E。其中，

1. T为全部疏果操作结束后所有苹果树上剩下的苹果总数（假设每棵苹果树在最后一次统计苹果个数操作后苹果不会因为疏果以外的原因减少);
2. D为发生苹果掉落的苹果树的棵数;
3. E为相邻连续三棵树发生苹果掉落情况的组数。

对于第三个统计量的解释：N棵苹果树A1,A2,...,AN排列成一个圆，那么A1与A2相邻，A2与A3相邻，......，AN-1与AN相邻，AN与A1相邻。如果Ai-1, Ai，Ai+1这三棵树都发生了苹果掉落的情况，则记为一组。形式化的，有

E=|{Ai/Drop(Pred(Ai)）∧Drop(Ai）∧Drop(Succ(Ai)）}|

其中，Drop(Ai）表示苹果树Ai是否发生苹果掉落的情况，Pred(Ai)表示Ai的前一棵树Ai-1（如果i>1）或者AN（如果i=1)，Succ(Ai)表示Ai的后一棵树Ai+1（如果i<N）或者A1（如果i=N）

样例1输入：

4

4 74 -7 -12 -5

5 73 -8 -6 59 -4

5 76 -5 -10 60 -2

5 80 -6 -15 59 0

样例1输出：

222 1 0

样例1解释：

全部操作结束后，第1棵树上剩下的苹果个数为74-7-12-5=50，第2棵为59-4=55，第3棵为60-2=58，第4棵为59-0=59。因此T=50+55+58+59=222。

其中，第3棵树在第2次统计之前剩下的苹果个数为76-5-10=61>60，因此发生了苹果掉落的情况。可以检验其他的树没有这种情况，因此D=1。

没有连续三棵树都发生苹果掉落的情况，因此E=0。

样例2输入：

5

4 10 0 9 0

4 10 -2 7 0

2 10 0

4 10 -3 5 0

4 10 -1 8 0

样例2输出：

39 4 2

样例2解释：

第1、2、4、5棵树发生了苹果掉落的情况，因此D=4。其中，连续三棵树都发生苹果掉落情况的有(5,1,2)和(4,5,1)，因此E=2。

算法思想：

typedef struct ListInfo

{

LNode \*head;

LNode \*tail;

int operateNumber;

int AppleNumber; //苹果数量

int AppleOffTreeNum;//是否掉落苹果

}ListInfo;//一棵苹果树的数据

Status Operation(ListInfo &L)//对每一棵苹果树的操作

{

LNode \*p = L.head->next;

while(p != NULL)//遍历每一个结点

{

if(p->operate > 0)//若大于0

{

if(p == L.head->next)//若是第一个结点

L.AppleNumber += p->operate;//直接赋值给苹果数量

else

{

if(p->operate < L.AppleNumber)// 若不是第一个结点，且比苹果数量少

{

//说明掉落了苹果，掉落棵树赋值为1，并把苹果数量赋值为当前数量

L.AppleOffTreeNum = 1;

L.AppleNumber = p->operate;

}

}

}

else//若小于0，则苹果数量减少

L.AppleNumber += p->operate;

p = p->next;//下一个结点

}

return OK;

}

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

{

CreateList(L[i]);

Operation(L[i]);

T += L[i].AppleNumber; //每一棵苹果树的苹果数量的累加

D += L[i].AppleOffTreeNum;//掉落苹果的苹果树的棵数累加

}

//计算连续三棵树掉落的组数

//1、只包含有头或尾的连续的三棵树

for(i=2; i<=N-1; i++)

if(L[i-1].AppleOffTreeNum + L[i].AppleOffTreeNum + L[i+1].AppleOffTreeNum == 3)

E += 1;

//2、包含头和尾连续的三棵树

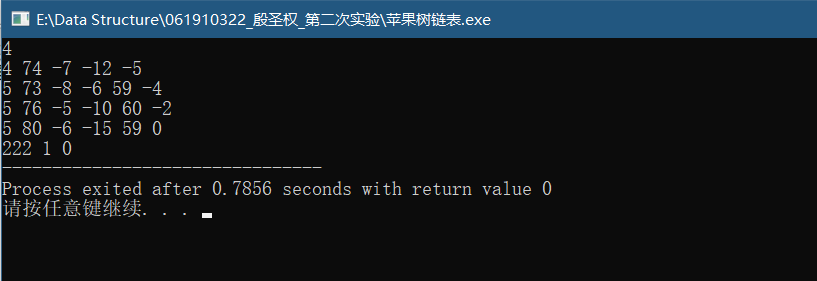
if(L[N].AppleOffTreeNum + L[1].AppleOffTreeNum + L[2].AppleOffTreeNum == 3)

E += 1;

if(L[N-1].AppleOffTreeNum + L[N].AppleOffTreeNum + L[1].AppleOffTreeNum == 3)

E += 1;

运行结果：



结果分析：时间复杂度：O（n）

附源程序。

5、

题目：

CSP题目

问题描述：小H和小W来到了一条街上，两人分开买菜，他们买菜的过程可以描述为，去店里买一些菜然后去旁边的一个广场把菜装上车，两人都要买n种菜，所以也都要装n次车。具体的，对于小H来说有n个不相交的时间段[a1,b1],[a2,b2]…[an,bn]在装车，对于小W来说有n个不相交的时间段[c1,d1],[c2,d2]…[cn,dn]在装车。其中，一个时间段[s, t]表示的是从时刻s到时刻t这段时间，时长为t-s。  
　　由于他们是好朋友，他们都在广场上装车的时候会聊天，他们想知道他们可以聊多长时间。  
输入格式  
　　输入的第一行包含一个正整数n，表示时间段的数量。  
　　接下来n行每行两个数ai，bi，描述小H的各个装车的时间段。  
　　接下来n行每行两个数ci，di，描述小W的各个装车的时间段。  
输出格式  
　　输出一行，一个正整数，表示两人可以聊多长时间。  
样例输入  
4  
1 3  
5 6  
9 13  
14 15  
2 4  
5 7  
10 11  
13 14  
样例输出  
3  
数据规模和约定  
　　对于所有的评测用例，1 ≤ n ≤ 2000, ai < bi < ai+1，ci < di < ci+1,对于所有的i(1 ≤ i ≤ n)有，1 ≤ ai, bi, ci, di ≤ 1000000。  
　　给两个人设定两个数组t[i]，t1[i] 当装车时置1，当t[i]=t1[i]时 总数sum++。

算法思想：

typedef struct LNode//储存每个时间段的结点

{

struct LNode \*next;

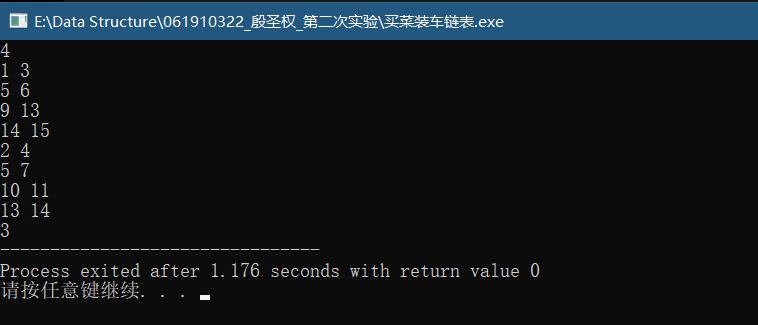
int startTime;//开始时间

int endTime;//结束时间

}LNode;

以W末尾时刻为主，分为三种情况，W的末尾时刻在H开始时刻之前，在H开始和末尾时刻之间，在H末尾时刻之后，分情况讨论

运行结果：



结果分析：时间复杂度：O（n）

附源程序。

6、

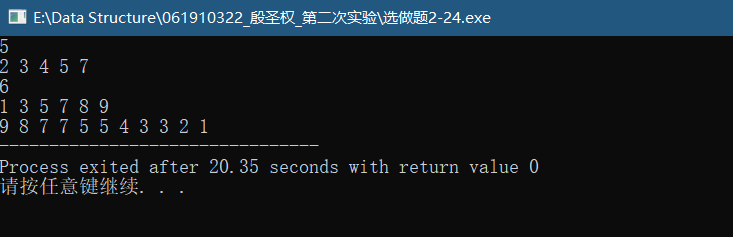
题目：

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表。

算法思想：

每次将b表第一个元素依次与a表元素比较，将b表结点摘除并插入到a表中适当位置，最后逆置a表。

运行结果：



结果分析：时间复杂度：O（n^2）

附源程序。

7、

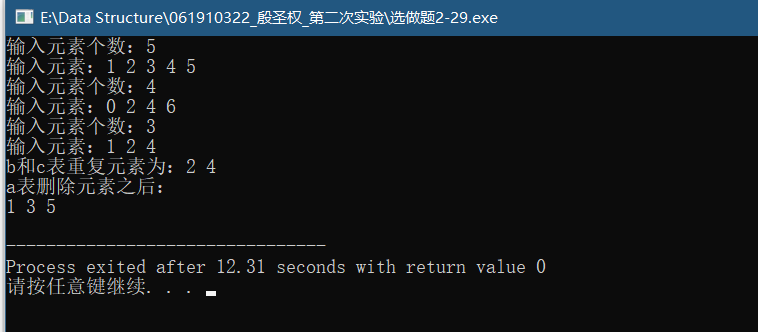
题目：

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）。

算法思想：

先找出既在b表又在c表的元素并存在d数组，在a表中删除和c表相同的元素。

运行结果：



结果分析：时间复杂度：O（n^2）

附源程序。

8、

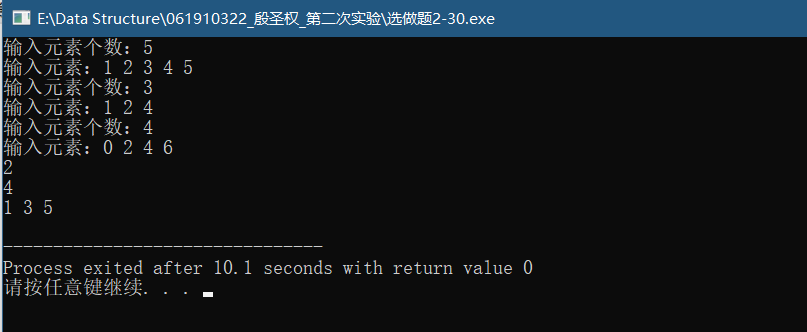
题目：

2.30 要求同2.29题。试对单链表编写算法，请释放A表中的无用结点空间。

算法思想：

找到一个b表和c表共有元素之后立即判断a表是否有相同元素，若有则删除。

运行结果：



结果分析：时间复杂度：O（n^2）

附源程序。

二、代码行数及小结

代码行数：

第一题：顺序表：77；链表：103

第二题：顺序表：121；链表：146

第三题：顺序表：84；链表：197

第四题：102

第五题：160

第六题：119

第七题：116

第八题：128

小结：

第二题没仔细看题目，一开始没有在A表原有空间构建新表，而是另外开辟新空间，我吸取了本题的教训，下次会注意审题。