1. **试分别用顺序表和单链表作为存储结构，实现将线性表(a0,a1,...an-1)就地逆置的操作，所谓"就地"指辅助空间应为O(1)。**

解答：

1. 顺序表：

要将该表逆置，可以将表中的开始结点与终端结点互换，第二个结点与倒数第二个结点互换，如此反复，就可将整个表逆置了。

算法如下：

#define List\_Size 100  // 线性表存储空间的初始分配量为100

typedef struct{

ElemType  \*elem; // 向量data用于存放表结点

int  length;  // 当前的表长度

} Sqlist;

  //以上为定义表结构

void Invert\_Sq( Sqlist &L){

//逆置顺序表L

n=L.length; m=n/2;

for ( i=0 ; i < m; i++){

temp = L.elem[ i ];

L .elem[ i ] = L.elem[ n - 1 - i ] ;

L.elem[ n - 1 - i ] = temp ; //交换数据逆置

}

}

//Invert\_Sq

2. 单链表：

可以利用指针的指向转换来达到表逆置的目的。

算法如下：

typedef struct LNode{  //结点类型定义

ElemType  data;  //结点的数据域

Struct LNode  \*next; //结点的指针域

}LNode;

typedef  LNode  \*LinkList;

//以上为定义表结构

LinkList Invert\_L( LinkList L ){

//逆置单链表L

LNode \*p ,\*q ;//设置两个临时指针变量

if( L&&L->next){

//当链表不是空表或单结点时，将链表中的除头结点外的所有结点顺着链表指针，由前往后将每个结点依次从链表中摘下，作为第一个结点插入到带头结点链表中。这样就可以得到逆置的链表。

L = L->next;

p = L;

q = p->next;

p -> next = NULL; //将开始结点变成带头结点链表的终端结点，原来的第二个结点成为新的无头链表的开始结点

while (q){

//每次循环将无头结点链的后一个结点变成带头结点链的开始结点

p = q;

q = q-> next ;

p->next = L;

L = p;

}

return L;

}

return L; //如是空表或单结点表，直接返回头指针。

//当链表为空表或只有一个结点时，该链表的逆置链表与原表相同。

}

//Invert\_L

1. **设顺序表L是一个递减有序表，试设计一个算法：将e插入到L中，并使L仍为一个有序表。**

解答：

因已知顺序表L是递减有序表，所以只要从头找起找到第一个比它小(或相等)的结点数据，把e插入到这个数所在的位置就是了。算法如下：

void Insert\_DecreaseList( Seqlist \*L , ElemType e){

if( L.length >= List\_Size)

Error(“Overflow!”); //L已满，给出相应信息

for ( i=L.length ; (i > 0) && (L.elem[ i-1] < e) ; i--)

L.elem[ i ]=L.elem[ i - 1 ];   // 比较并移动元素

L.elem[ i ] =e; //插入e到L中

L.length++;  //修正L的长度，令其增1

}

//Insert\_DecreaseList

1. **设La和Lb是两个单链表，表中元素递增有序，试设计一个算法：将La和Lb就地归并成一个按元素值递减有序的单链表Lc，请分析算法的时间复杂度。**

解答：

根据已知条件，La和Lb是两个递增有序表，所以可以先取La表的表头建立空的Lc表。然后同时扫描La表和Lb表，将两表中最大的结点从对应表中摘下，并作为开始结点插入Lc表中。如此反复，直到La表或Lb表为空。最后将不为空的La表或Lb表中的结点依次摘下并作为开始结点插入Lc表中。这时，得到的Lc表就是由La表和Lb表归并成的一个按元素值递减有序的单链表Lc。并且辅助空间为O(1)。

算法如下：

LinkList MergeSort\_L ( LinkList La , LinkList Lb LinkList &Lc) {

// 归并两个带头结点的递增有序表为一个带头结点递减有序表

pa=La->next; //pa指向La表开始结点

pb=Lb->next; //pb指向Lb表开始结点

Lc=pc=La;

Lc->next=NULL; //取La表的表头建立空的Lc表

while ( pa && pb) {

if ( pb->data <= pa->data ) {

// 当Lb中的元素小于等于La中当前元素时，将pa表的开始结点摘下

q=pa;pa=pa->next;

}

else{

// 当Lb中的元素大于La中当前元素时，将pb表的开始结点摘下

q=pb;pb=pb->next;

}

       q->next=pc->next;

pc->next=q; //将摘下的结点q作为开始结点插入Lc表

}

//若pa表非空，则处理pa表

while(pa)

{

       q=pa;pa=pa->next;

       q->next=pc->next;

pc->next=q;

}

//若pb表非空，则处理pb表

while(pb)

{

       q=pb;pb=pb->next;

       q->next=pc->next;

pc->next=q;

}

delete Lb;//回收Lb表的头结点空间

return Lc;

}

该算法的时间复杂度分析如下：

算法中有三个while 循环，其中第二个和第三个循环只执行一个。每个循环做的工作都是对链表中结点扫描处理。整个算法完成后，A表和B表中的每个结点都被处理了一遍。所以若A表和B表的表长分别是m和n，则该算法的时间复杂度O(m+n)