第二章 线性表

回顾

- **研究和解决非数值数据的组织和处理**
  - 描述非数值计算问题的数学模型，不再是数学方程
  - 例如：前述的三个例子：数据的线性结构，树型结构，图
- **算法+数据结构=程序**
  - 算法和数据结构之间的关系
  - 软件系统的框架应当建立在数据之上，而不是建立在操作之上
- **数据结构的作用范畴**
  - 抽象数据对象的数学模型（逻辑结构）例：图状结构
  - 明确操作（运算的定义）例：查找、插入、……
  - 在存储结构上映射数据（存储结构）例：顺序存储
  - 实现操作

回顾

- 算法
  - 五特性：
    - 有穷性、确定性、0至多个输入、1至多个输出、有效性
  - 四要求：
    - 正确性、可读性、健壮性、高效性
- 算法评价
  - 事后统计
  - 事前分析

回顾

- **时间复杂度（大O运算规则）**

  规则1： $kf(n)=O(f(n))$ //大O忽略常数因子

  规则2： if $f(n)=O(g(n))$ and $g(n)=O(h(n))$

  then $f(n)=O(h(n))$ //传递性

  规则3： $f(n)+g(n)=O(max\{f(n),g(n)\})$

  规则4： if $f1(n)=O(g1(n))$ and $f2(n)=O(g2(n))$

  then $f1(n)*f2(n)=O(g1(n)*g2(n))$

第2章 线性表

第2章 线性表

- 2.1 线性表的类型定义
- 2.2 线性表的顺序表示和实现
- 2.3 线性表的链式表示和实现
- 2.4 一元多项式的表示及相加

2.1 线性表的类型定义

例1、26个英文字母组成的字母表

(A，B，C、…、Z)

例2、某实验室从2017年到2023年各种型号的计算机拥有量的变化情况。

(6，17，28，50，92，188)

例3、一副扑克的点数

(2，3，4，…，J，Q，K，A)

2.1 线性表的类型定义

例4、工厂员工健康情况登记表如下：

姓　名	工　号	性　别	年龄	健康情况
王小林	190631	男	18	健康
陈　红	190632	女	20	一般
刘建平	190633	男	21	健康
张立立	190634	男	17	神经衰弱
……	……	……	……	……

2.1 线性表的类型定义

1、定义

若结构是非空有限集，则有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。

→可表示为：$(a_1, a_2, \ldots\ldots, a_n)$

2.1 线性表的类型定义

数据结构：

1）线性表是n(n ≥0)个数据元素的有限序列。

2）含有n个数据元素的线性表是一个数据结构：

List = (D,R)

其中：

$D = \{a_i \mid a_i \in ElemSet, i=1,2,...n, n \geq 0\}$

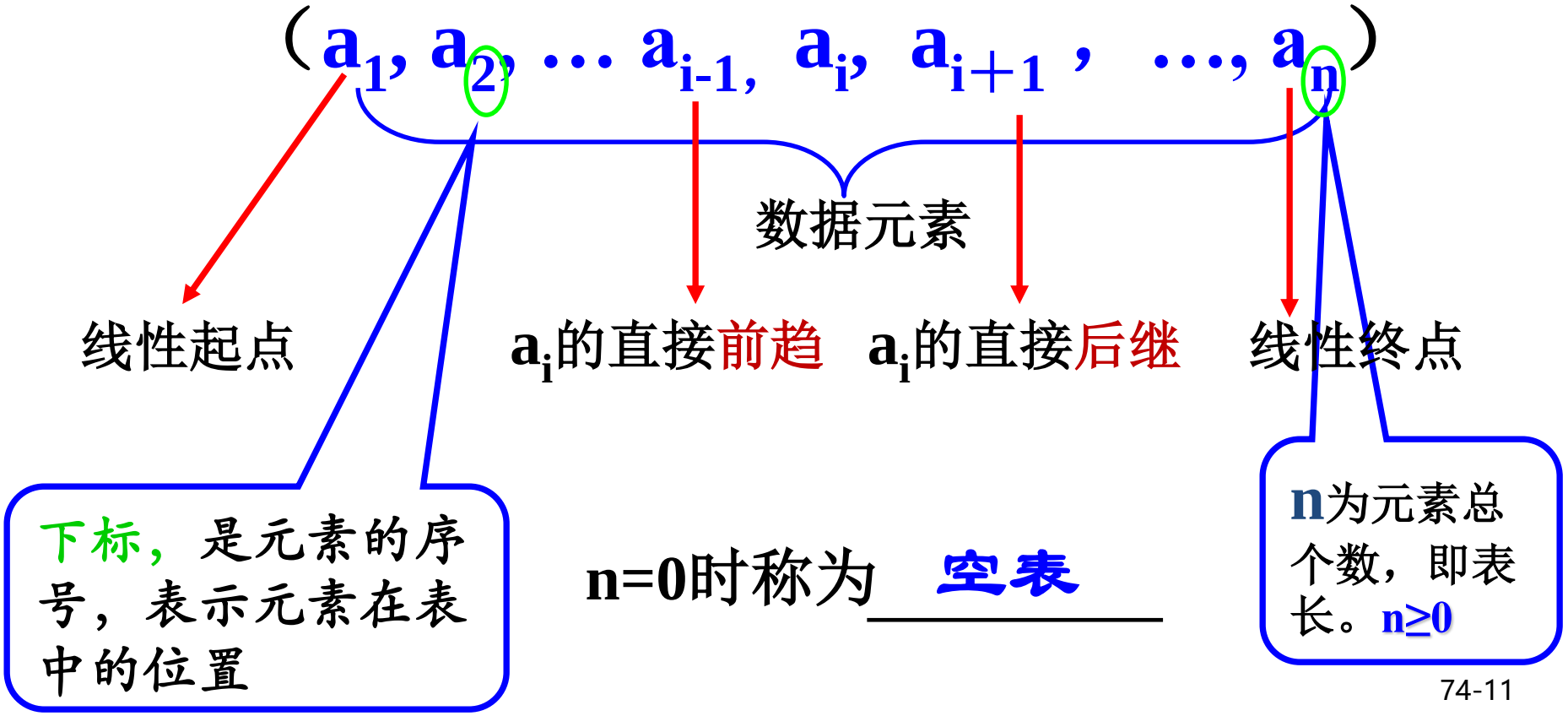
$R = \{< a_{i-1}, a_i > \mid a_{i-1}, a_i \in D, i = 2,3,...n\}$

特性：均匀性，有序性（线性序列关系）

2.1 线性表的类型定义

2、 线性表逻辑结构

线性表的定义： 用数据元素的有限序列表示



2.1 线性表的类型定义

特点： 在数据元素的非空有限集中，

1）有且仅有一个开始结点；

2）有且仅有一个终端结点；

3）除第一个结点外，集合中的每个数据元素均有且只有一个前驱；

4）除最后一个结点外，集合中的每个数据元素均有且只有一个后继。

简言之，线性结构反映结点间的逻辑关系是

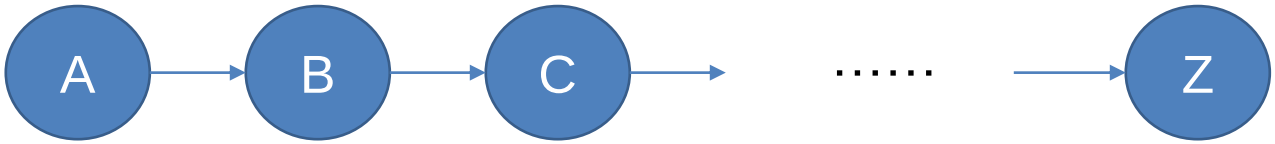
一对一 (1:1)

2.1 线性表的类型定义

例、26个英文字母组成的字母表

(A，B，C、…、Z)

分析：数据元素都是同类型（字母）， 元素间关系是线性的。



2.1 线性表的类型定义

例、工厂员工健康情况登记表如下：

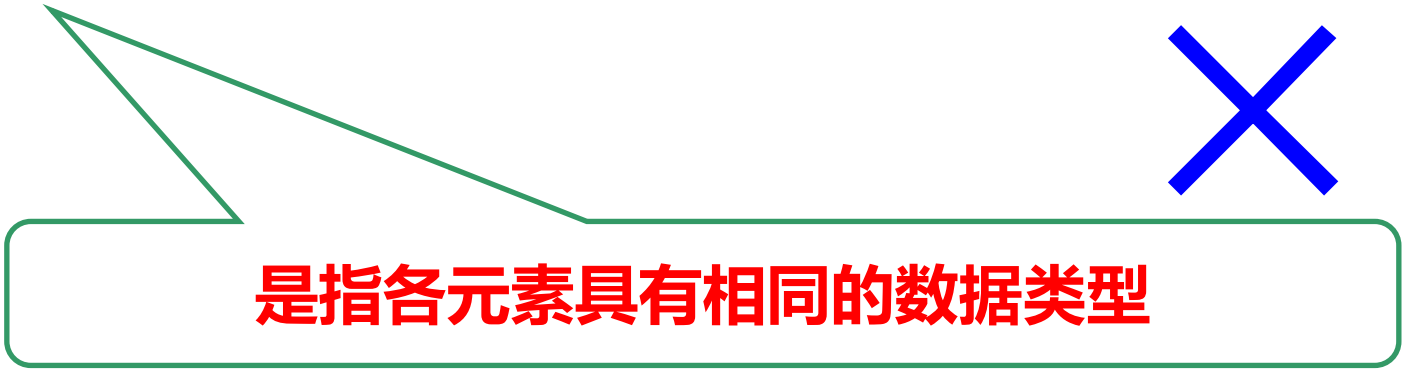
姓　　名	工　　号	性　　别	年龄	健康情况
王小林	190631	男	18	健康
陈　　红	190632	女	20	一般
刘建平	190633	男	21	健康
张立立	190634	男	17	神经衰弱
……	……	…….	…….	…….

注意：同一线性表中的元素必定具有相同特性 （数据类型）！

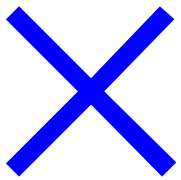
分析：数据元素都是同类型（记录），元素之间关系是线性的。

2.1 线性表的类型定义

①“同一数据逻辑结构中的所有数据元素都具有相同的特性”是指数据元素所包含的数据项的个数都相等。



②线性结构就是线性表



2.1 线性表的类型定义

3、线性表的操作

1）InitList(&L) 初始化，构造一个空的线性表

2）ListLength(L) 求长度,返回线性表中数据元素个数

3）GetElem(L,i,&e) 取表L中第i个数据元素赋值给e

4）LocateElem(L,e) 按值查找，若表中存在一个或多个值为e的结点，返回第一个找到的数据元素的位序，则返回一个特殊值。

5）ListInsert(&L,i,e) 在L中第i个位置前插入新的数据元素e，表长加1。

6）ListDelete(&L,i,e) 删除表中第i个数据元素，e返回其值，表长减1。

2.1 线性表的类型定义

例2-1 假设有两个集合 A 和 B 分别用两个线性表 LA 和 LB 表示，即线性表中的数据元素即为集合中的成员。编写一个算法求一个新的集合C=A∪B，即将两个集合的并集放在线性表LC中。

2.1 线性表的类型定义

例2-1 求两个集合的并,即A=A∪B

分析：设A、B分别由两个线性表LA和LB表示，

要求：将LB中存在而LA中不存在的DE插入到表LA中。

算法思想：① 依次从LB中取出一个数据元素E；
② 判在LA中是否存在；
③ 若不存在，则插入到LA中。

2.1 线性表的类型定义

```
void unionList(List &LA,List LB) {
    int lena,lenb,i;
    ElemType e;
    lena=ListLength(LA);  /*求线性表的长度*/
    lenb=ListLength(LB);
    for (i=1;i<=lenb;i++) {
        GetElem(LB, i, e); /*取LB中第i个数据元素赋给e*/
        if  (!LocateElem(LA, e,  equal))))
            ListInsert(LA,++lena, e); /*LA中不存在和e相同者,则插入到LC中*/
     }
}
```

由于LocateElem(LA,e)运算的时间复杂度为O(ListLength(LA)),
所以本算法的时间复杂度为:
O(ListLength(LA)*ListLength(LB))。

2.1 线性表的类型定义

例2-2 已知线性表LA 和LB 中的数据元素按值非递减有序排列，现要求将LA 和LB 归并为一个新的线性表LC，且LC 中的数据元素仍按值非递减有序排列。

LA = (3, 5, 8, 11)

LB = (2, 6, 8, 9, 11, 15, 20)

则

LC = (2, 3, 5, 6, 8, 8, 9, 11, 11, 15, 20)

2.1 线性表的类型定义

```
void MergeList (List LA, List LB, List &LC) {
    int lena, lenb, lenc, i, j, k;
    ElemType ai, bj;
    i = j = l; k = 0;
    InitList(LC);
    lena=ListLength(LA);  lenb=ListLength(LB); /*求线性表的长度*/
    while((i <= lena) && (j <= lenb)) { /*LA 和LB均非空*/
        GetElem(LA, i, ai); GetElem(LB, j, bj);
        if (ai <= bj) {ListInsert(LC, ++k, ai); ++ i;}
        else {Listinsert(Lc, ++k, bj); ++ j; }
    }
    while(i <= lena) {
        GetElem(La, i++, ai); Listlnsert(Lc, ++ k, ai);
    }
    while (j <= lenb) {
        GetElem(Lb, j + + , bj); Listlnsert(Lc, ++ k, bj);
    }
}
```

时间复杂度：
ListInsert()执行次数
O(ListLength(LA)+ListLength(LB))

2.1 线性表的类型定义

4、线性表的抽象数据类型

ADT List{

数据对象：$D = \{a_i \mid a_i \in ElemSet, i=1,2,\cdots n, n \geqslant 0\}$

数据关系：$R1 = \{< a_{i-1}, a_i > \mid a_{i-1}, a_i \in D, i = 2,3,\cdots n\}$

数据操作：

InitList(&L);

ListLength(L);

GetElem(L,i,&e) ;

LocateElem(L,e);

ListInsert(&L,i,e);

ListDelete(&L,i,e);

……

}

5、线性表的物理结构

顺序存储结构与链式存储结构

在确定线性表物理结构的基础上，可以进行相关的操作和编程，用函数实现。

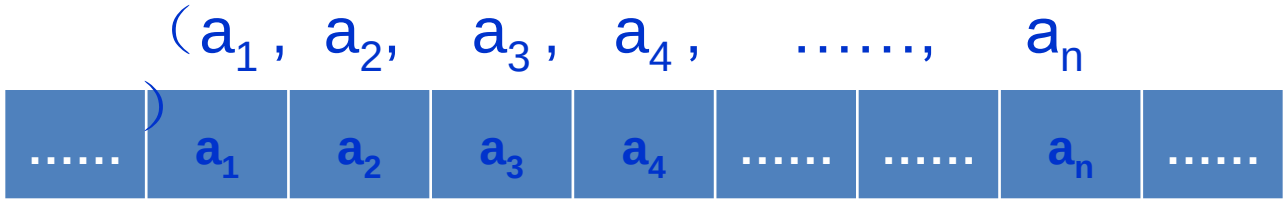
第2章 线性表

- 2.1 线性表的类型定义
- 2.2 线性表的顺序表示和实现
- 2.3 线性表的链式表示和实现
- 2.4 一元多项式的表示及相加

2.2 线性表的顺序表示和实现

线性表的顺序存储结构就是：把线性表中的所有元素按照其逻辑顺序依次存储到从计算机存储器中指定存储位置开始的一块连续的存储空间中。

这样,线性表中第一个元素的存储位置就是指定的存储位置，第i+1个元素(1≤i≤n-1)的存储位置紧接在第i个元素的存储位置的后面。



线性表 ↔ 逻辑结构

顺序表 ↔ 存储结构

特点：逻辑上相邻的元素，物理上也相邻。

2.2 线性表的顺序表示和实现

假定线性表的元素类型为ElemType,则每个元素所占用存储空间大小(即字节数)为

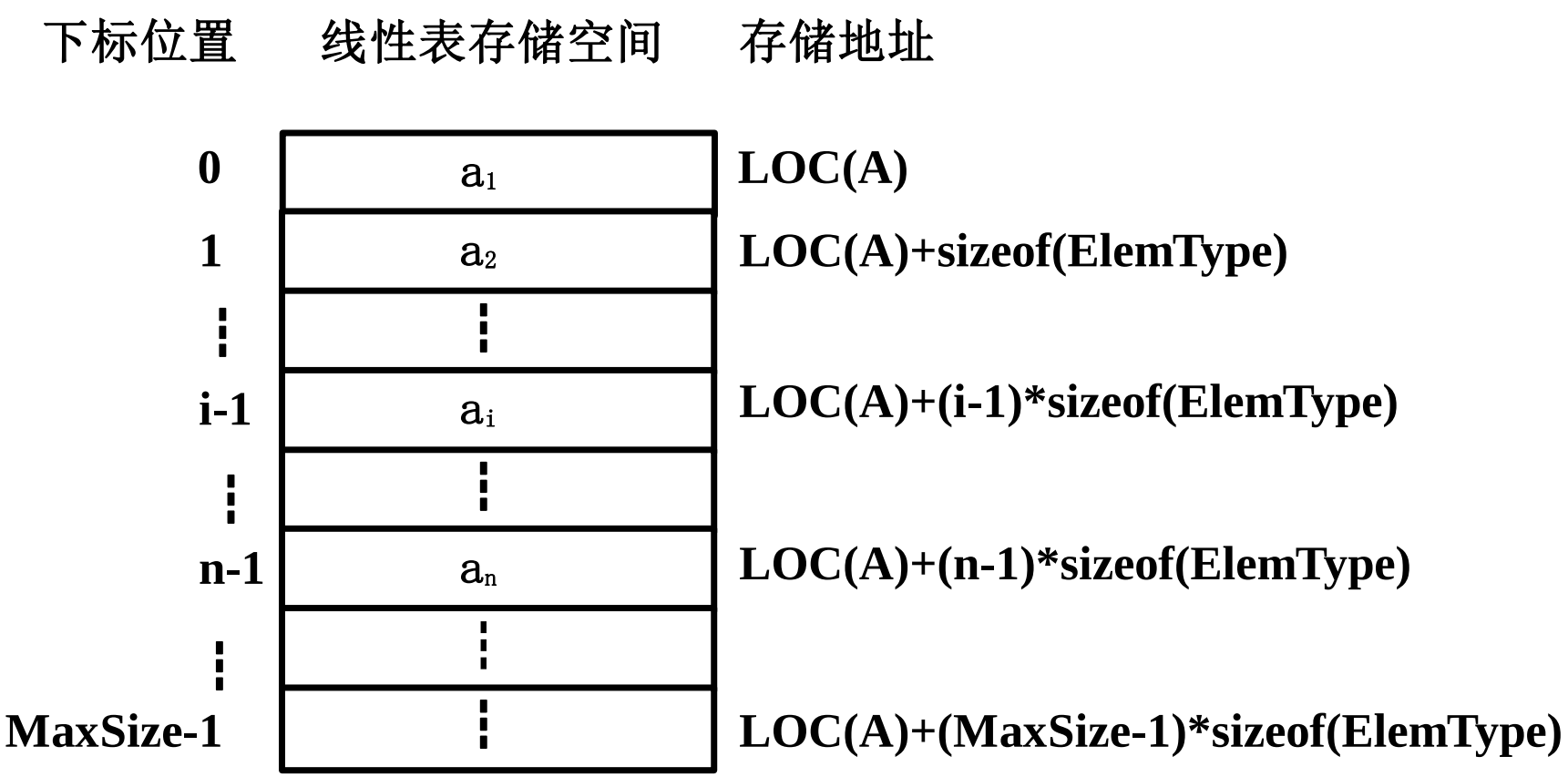
sizeof(ElemType)

整个线性表所占用存储空间的大小为:

n*sizeof(ElemType)

其中,n表示线性表的长度。

2.2 线性表的顺序表示和实现



2.2 线性表的顺序表示和实现

L = sizeof(ElemType)

$Loc(a_{i+1}) = Loc(a_i) + L$

一般来说，线性表的第i个元素a_i的存储位置为：

$Loc(a_i) = Loc(a_1) + (i-1)*L$

其中$Loc(a_1)$是线性表的第一个数据元素a_1的存储位置，通常称作线性表的起始位置或基地址。

2.2 线性表的顺序表示和实现

- 用“物理位置”相邻来表示线性表中数据元素之间的逻辑关系。

- 根据线性表的顺序存储结构的特点，只要确定了存储线性表的**起始位置**，线性表中任一数据元素都可随机存取，所以，线性表的顺序存储结构是一种**随机存取**的存储结构。

2.2 线性表的顺序表示和实现

例：设有一维数组M，下标的范围是0到9，每个数组元素用相邻的4个字节存储。存储器按字节编址，设存储数组元素M[0]的第一个字节的地址是98，则M[3]的第一个字节的地址是多少?

110

解：已知地址计算通式为：

$$LOC(a_i) = LOC(a_1) + L * (i-1)$$

但此题要注意下标起点略有不同：

$$LOC(M_{[3]}) = 98 + 4 \times (4-1) = 110$$

2.2 线性表的顺序表示和实现

顺序存储方法:用一组地址连续的存储单元依次存储线性表的元素。

可以利用数组V[n]来实现，
也可以用指针变量表示

数组→顺序存储结构。

2.2 线性表的顺序表示和实现

```
#define MaxSize 100   //或者N,或者是一个常数
typedef struct
{     ElemType data[MaxSize];
      int length;
} SqList;   /*顺序表类型*/
```

其中，data成员存放元素，length成员存放线性表的实际长度。

注意：由于C/C++中数组的下标从0开始，线性表的第i个元素a_i存放顺序表的第i-1位置上。为了清楚，将a_i在逻辑序列中的位置称为逻辑位序，在顺序表中的位置称为物理位序。

2.2 线性表的顺序表示和实现

1. 建立顺序表

其方法是将给定的含有n个元素的数组的每个元素依次放入到顺序表中，并将n赋给顺序表的长度成员。算法如下：

```
void CreateList(SqList *&L,ElemType a[],int n)
/*建立顺序表*/
{
    int i;
    L=(SqList *)malloc(sizeof(SqList));
    for (i=0;i<n;i++)
        L->data[i]=a[i];
    L->length=n;
}
```

void *malloc(unsigned int size)

2.2 线性表的顺序表示和实现

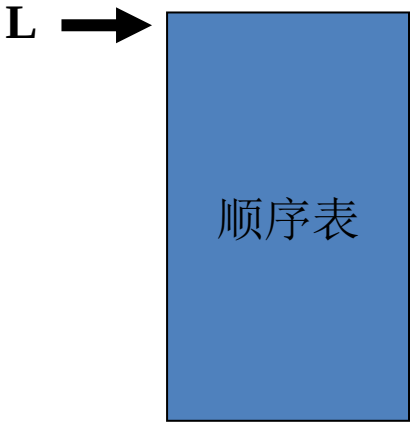
2. 顺序表基本运算算法

(1) 初始化线性表InitList(L)

该运算的结果是构造一个空的线性表L。实际上只需将length成员设置为0即可。

```
void InitList(SqList *&L)   //引用型指针{
    L=(SqList *)malloc(sizeof(SqList));
     /*分配存放线性表的空间*/
    if (!L) exit(OVERFLOW);
    L->length=0;
}
```

本算法的时间复杂度为O(1)。



2.2 线性表的顺序表示和实现

2. 顺序表基本运算算法

(2) 销毁线性表DestroyList(L)

该运算的结果是释放线性表L占用的内存空间。

```
void DestroyList(SqList *&L)
{
    free(L);
}
```

本算法的时间复杂度为O(1)。

2.2 线性表的顺序表示和实现

2. 顺序表基本运算算法

(3) 判定是否为空表ListEmpty(L)

该运算返回一个值表示L是否为空表。若L为空表，则返回1，否则返回0。

```
int ListEmpty(SqList *L)
{
        return(L->length==0);
}
```

本算法的时间复杂度为O(1)。

2.2 线性表的顺序表示和实现

2. 顺序表基本运算算法

(4) 求线性表的长度ListLength(L)

该运算返回顺序表L的长度。实际上只需返回length成员的值即可。

```
int ListLength(SqList *L)
{
        return(L->length);
}
```

本算法的时间复杂度为O(1)。

2.2 线性表的顺序表示和实现

2. 顺序表基本运算算法

(5) 输出线性表DispList(L)

该运算当线性表L不为空时,顺序显示L中各元素的值。

```
void DispList(SqList *L)
{
        int i;
        if (ListEmpty(L)) return;
        for (i=0;i<L->length;i++)
                printf("%c",L->data[i]);
        printf("\n");
}
```

本算法中基本运算为for循环中的printf语句，故时间复杂度为:

O(L->length)或O(n)

2.2 线性表的顺序表示和实现

2. 顺序表基本运算算法

(6) 求某个数据元素值GetElem(L,i,e)

该运算返回L中第i(1≤i≤ListLength(L))个元素的值,存放在e中。

```
int GetElem(SqList *L,int i,ElemType &e)
{
        if (i<1 || i>L->length)  return 0;
        e=L->data[i-1];
        return 1;
}
```

本算法的时间复杂度为O(1)。

2.2 线性表的顺序表示和实现

2. 顺序表基本运算算法

(7) 按元素值查找LocateElem(L,e)

p25～p26算法2.6

该运算顺序查找第1个值域与e相等的元素的位序。若这样的元素不存在,则返回值为0。

```
int LocateElem(SqList *L, ElemType e)
{
        int i=0;
        while (i<L->length && L->data[i]!=e) i++;
        if (i>=L->length)return 0;
        else  return i+1;
}
```

本算法中基本运算为while循环中的i++语句,故时间复杂度为:

O(L->length)或O(n)

2.2 线性表的顺序表示和实现

2. 顺序表基本运算算法

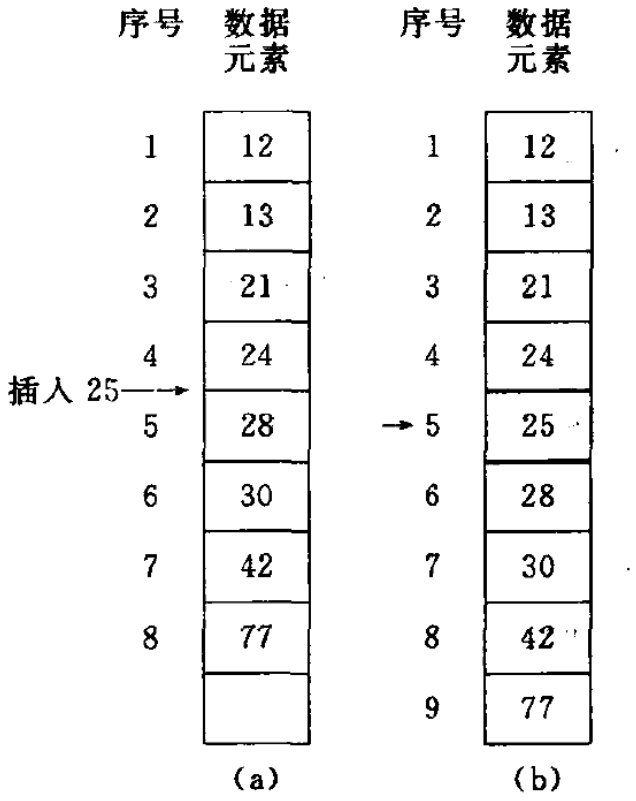
(8) 插入数据元素ListInsert(L,i,e)

该运算在顺序表L的第i个位置(1≤i≤ListLength(L)+1)上插入新的元素e。

插入前：L=（a1, ... , ai-1, ai, ... ,an）
插入后：L=（a1, ... , ai-1, b, ai, ... ,an ）

数据元素ai-1 和a; 之间的逻辑关系发生了变化

2.2 线性表的顺序表示和实现



为了在线性表的第4 和第5 个元素之间插入一个值为25 的数据元素，则需将第5 个至第8 个数据元素依次往后移动一个位置。

2.2 线性表的顺序表示和实现

算法思想：

① 进行合法性检查，1<=i<=n+1；

② 判断线性表是否已满；

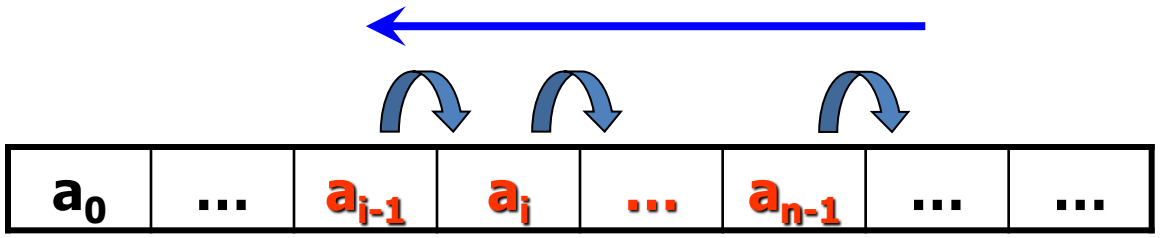
③ 将第n个至第i个元素逐一后移一个单元；

④ 在第i个位置处插入新元素；

⑤ 将表的长度加1。

2.2 线性表的顺序表示和实现

```
int ListInsert(SqList *&L,int i,ElemType e) {
    int j;
    if (i<1 || i>L->length+1)        return 0;
    i--;  /*将顺序表逻辑位序转化为elem下标即物理位序*/
    for (j=L->length;j>i;j--) L->data[j]=L->data[j-1];
        /*将data[i]及后面元素后移一个位置*/
    L->data[i]=e;
    L->length++;     /*顺序表长度增1*/
    return 1;
}
```



逻辑位序　1　　i　　i+1　　n　　MaxSize

2.2 线性表的顺序表示和实现

顺序表动态存储结构

```
#define LIST_INIT_LENTH 100 //或者N,或者是一个常数
#define LISTINCREMENT 10 //线性表存储空间的分配增量
typedef  struct {
      ElemType   *elem;
      int  length; //当前长度
      int  listsize; // 当前分配的存储容量
} SqList;
```

初始化算法发生了变化：p23算法2.3

2.2 线性表的顺序表示和实现

```
Status ListInsert_sq(SqList &L,int i,ElemType e){
  if (i<1 || i>L.length+1)  return ERROR;
  if (L.length >= L.listsize) {
     newbase=(ElemType*)realloc(L.elem,
              (L.listsize+LISTINCREMENT)*sizeof(ElemType));
     if (!newbase) exit(OVERFLOW);
     L.elem = newbase;
     L.listsize+=LISTINCREMENT;
   }
   q=&(L.elem[i-1]);
   for (p=&(L.elem[L.length-1]);p>=q;--p)
     *(p+1) = *p;
   *q=e;
   ++L.length;
   return OK;
}
```

```
void *realloc(void *p,unsigned int size)
```

2.2 线性表的顺序表示和实现

算法的复杂度。

- **所需移动结点的次数不仅依赖于表的长度，而且还与插入位置有关。**
- **当i=n+1，最好情况，其时间复杂度O（1）；**
- **当i=1时，结点后移语句将循环执行n次，需移动表中所有结点，这是最坏情况.**

2.2 线性表的顺序表示和实现

- **时间复杂度为O（n）。**

  由于插入可能在表中任何位置上进行，因此需分析算法的平均复杂度

  在长度为n的线性表中第i个位置上插入一个结点，令$E_{in}(n)$表示移动结点的期望值（即移动的平均次数），则在第i个位置上插入一个结点的移动次数为n-i+1。故

$$E_{in} = \sum_{i=1}^{n+1} p_i (n - i + 1)$$

2.2 线性表的顺序表示和实现

假设在表中任何位置($1 \leqq i \leqq n+1$)上插入结点的机会均等，则

$$p_1=p_2=p_3=\ldots=p_{n+1}=1/(n+1)$$

因此，在等概率插入的情况下，

$$E_{in}=\sum_{i=1}^{n+1} p_i(n-i+1)=\frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1)=\frac{n}{2}$$

虽然$E_{is}(n)$中n的的系数较小，但就数量级而言，它仍然是线性阶的。因此算法的平均时间复杂度为O(n)。

$$T(n)=O(n)$$

2.2 线性表的顺序表示和实现

本算法中注意以下问题：

(1) 顺序表中数据区域有MAXSIZE个存储单元，所以在向顺序表中做插入时先检查表空间是否满了，在表满的情况下不能再做插入，否则产生溢出错误。

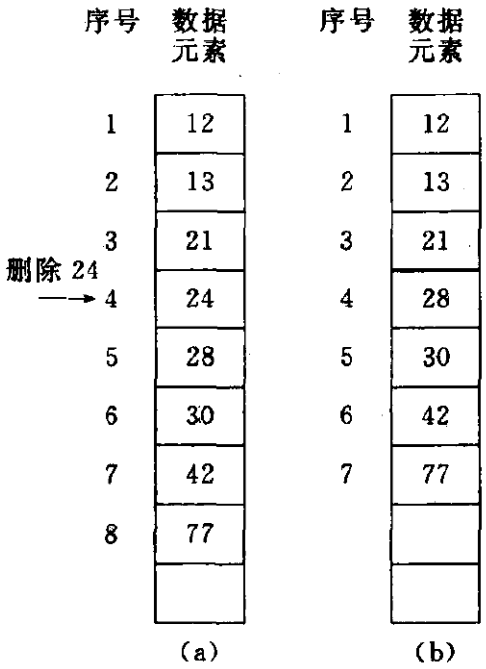
(2) 要检验插入位置的有效性，这里 i 的有效范围是：1<=i<=n+1，其中 n 为原表长。

(3) 注意数据的移动方向。

2.2 线性表的顺序表示和实现

(9) 删除数据元素ListDelete(L,i,e)

删除顺序表L中的第i(1≤i≤ListLength(L))个元素。
删除前：L=（a1,...,ai-1,ai,ai+1,...,an）
删除后：L=（a1,...,ai-1,ai+1,...,an）



为了删除第4 个数据元素，必须将从第5 个至第8 个元素都依次往前移动一个位置。

2.2 线性表的顺序表示和实现

算法思想：

① 进行合法性检查，i是否满足1<=i<=n；

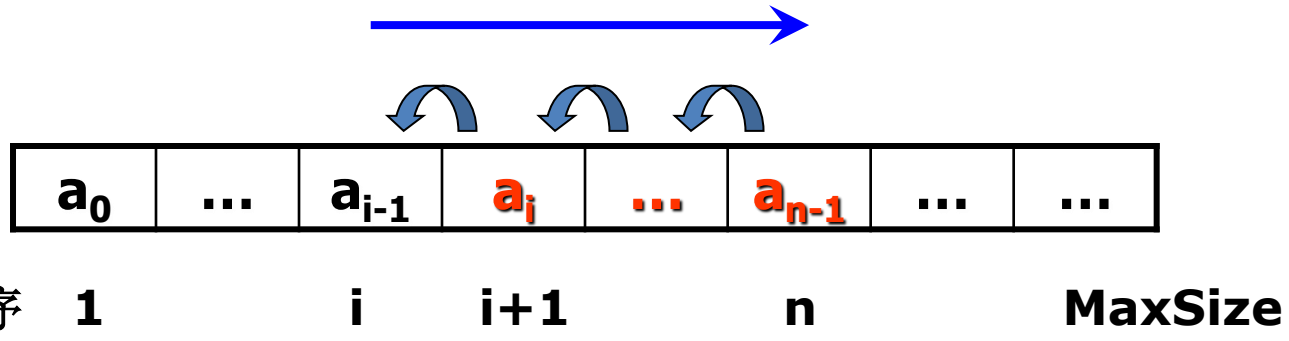
② 判线性表是否已空，v.last=0；

③ 将第i+1至第n个元素逐一向前移一个位置；

④ 将表的长度减1。

2.2 线性表的顺序表示和实现

```
int ListDelete(SqList *&L,int i,ElemType &e) {
    int j;
    if (i<1 || i>L->length)   return 0;
    i--;       /*将顺序表逻辑位序转化为elem下标即物理位序*/
    e=L->data[i];
    for (j=i;j<L->length-1;j++) L->data[j]=L->data[j+1];
        /*将data[i]之后的元素后前移一个位置*/
    L->length--;   /*顺序表长度减1*/
    return 1;
}
```



2.2 线性表的顺序表示和实现

```
Status ListDelete_sq(SqList &L,int i,ElemType &e) {
  if (i<1||i>L.length) return ERROR;
  p=&(L.elem[i-1]);
  e=*p;
  q=L.elem+L.length-1; //表尾元素结点
  for (++p;p<=q;++p) *(p-1)=*p;
  --L.length;
  return OK
}
```

- 需将第i+1至第L.length个元素向前移动一个位置

2.2 线性表的顺序表示和实现

时间复杂度分析:最坏情况是删除第1个元素，此时要前移n-1个元素，因此:T（n）=O（n）

本算法注意以下问题：

（1）删除第i个元素，i的取值为 1<=i<=n ,否则第i个元素不存在，因此，要检查删除位置的有效性。

（2）当表空时不能做删除，因表空时 L->last的值为-1，条件（i<1 || i>L->length+1）也包括了对表空的检查。

（3）删除 a_i 之后，该数据已不存在，如果需要，先取出 a_i ，再做删除。

2.2 线性表的顺序表示和实现

- 结点的移动次数也是由表长n和位置i决定。
- 若i=n，最好情况，时间复杂度O(1)，
- 若i=1，则前移语句将循环执行n-1次，时间复杂度O(n)。
- 平均复杂度。在长度为n的线性表中删除一个结点，令$E_{de}(n)$表示所需移动结点的平均次数，删除表中第i个结点的移动次数为n-i，故

$$E_{de} = \sum_{i=1}^{n} p_i(n-1)$$

式中，pi表示删除表中第i个结点的概率.

2.2 线性表的顺序表示和实现

在等概率条件下

$$p_1=p_2=p_3=\ldots=p_n=1/n$$

由此可得：

$$\mathrm{E}_{de} == \sum_{i=1}^{n} p_i(n-i) = \frac{1}{n}\sum_{i=1}^{n+1}(n-i) = \frac{n-1}{2}$$

即在顺序表上做删除运算，平均要移动表中约一半的结点，平均时间复杂度也是O(n)。

2.2 线性表的顺序表示和实现

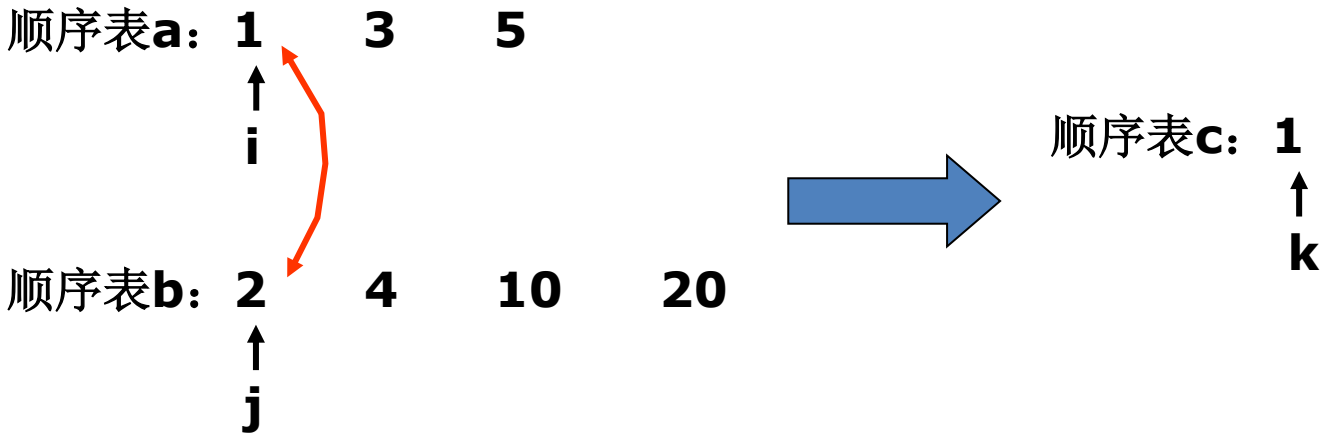
(10)线性表合并 (La、Lb为按顺序排列线性表)

将两个元素有序(从小到大)的顺序表合并成一个有序顺序表。

思路:

将两个顺序表进行二路归并。依次扫描通过A和B的元素，比较当前的元素的值，将较小值的元素赋给C，如此直到一个线性表扫描完毕，然后将未完的那个顺序表中余下部分赋给C即可。C的容量要能够容纳A、B两个线性表相加的长度。

2.2 线性表的顺序表示和实现



归并到顺序表c中 ← k记录c中元素个数

1(i=0) ⇔ 2(j=0) ⇒ 将1(i=1)插入r(k=1)

3(i=1) ⇔ 2(j=0) ⇒将2(j=1)插入r(k=2)

3(i=1) ⇔ 4(j=1) ⇒将3(i=2)插入r(k=3)

5(i=2) ⇔ 4(j=1) ⇒将4(j=2)插入r(k=4)

5(i=2) ⇔ 10(j=2) ⇒将5(j=3)插入r(k=5)

将b中余下元素插入c中。

2.2 线性表的顺序表示和实现

(10)线性表合并 (La、Lb为按顺序排列线性表)

```
void MergeList(SqList La,SqList Lb,SqList &Lc){
  pa=La.elem; pb=Lb.elem;
  Lc.listsize = Lc.length= La.length+Lb.length;
  pc=Lc.elem=(ElemType*)malloc(Lc.listsize*sizeof(ElemType));
  if (!Lc.elem) exit(OVERFLOW);
  pa_last=La.elem+La.length-1;
  pb_last=Lb.elem+Lb.length-1;
  while (pa<=pa_last&&pb<=pb_last) {
     if(*pa<=*pb) *pc++=*pa++;
      else *pc++=*pb++; }
  while (pa<=pa_last) *pc++=*pa++;
  while (pb<=pb_last) *pc++=*pb++;
}
```

时间复杂度
O(La. length+ Lb. length)

2.2 线性表的顺序表示和实现

```
#define MaxSize  100
typedef struct {
  ElemType data[MaxSize];
  int length;
} SqList;
void MergeList_Sq(SqList La,SqList Lb,SqList &Lc) {
  int i=0, j=0,k=0;
  InitList (Lc);
  while (i<=La.length-1 && j<=Lb.length-1) {
    if (La.data[i]<=Lb.data[j]) Lc.data[k++]=La.data[i++];
    else Lc.data[k++]=Lb.data[j++]; }
  while (i<=La.length-1) Lc.data[k++]=La.data[i++];
  while (j<=Lb.length-1) Lc.data[k++]=Lb.data[j++];
  Lc.length = k;
}
```

2.2 线性表的顺序表示和实现

顺序表应用举例

例　设计一个算法,将x插入到一个有序(从小到大排序)的线性表(顺序存储结构即顺序表)的适当位置上,并保持线性表的有序性。

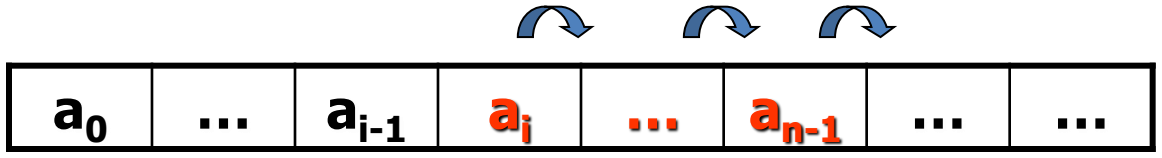
解:先通过比较在顺序表L中找到存放x的位置i,然后将x插入到L.elem[i]中,最后将顺序表的长度增1。

2.2 线性表的顺序表示和实现

```
void Insert(SqList &L,ElemType x) {
    int i=0,j;
    if (L.length >= L.listsize) return OVERFLOW;
    while (i<L.length && L.elem[i]<x)
        i++;
    for (j=L.length-1;j>=i;j--)
        L.elem[j+1]=L.elem[j];
    L.elem[i]=x;
    L.length++;
}
```

查找插入位置

元素后移一个位置



逻辑位序 1 i i+1 n listsize

2.2 线性表的顺序表示和实现

例 **已知长度为n的线性表A采用顺序存储结构,编写一个时间复杂度为O(n)、空间复杂度为O(1)的算法,该算法删除线性表中所有值为item的数据元素。**

解:用k记录顺序表A中等于item的元素个数,边扫描A边统计k,并将不为item的元素前移k个位置,最后修改A的长度。对应的算法如下:

2.2 线性表的顺序表示和实现

```
void delnode1(SqList &A,ElemType item)
{     int k=0,i;  /*k记录值不等于item的元素个数*/
      for  (i=0;i<A.length;i++)
          if (A.elem[i]!=item)
          {      A.elem[k]=A.data[i];
                 k++;    /*不等于item的元素增1*/
           }
      A.length=k;  /*顺序表A的长度等于k*/
}
```

算法1：类似于建顺序表

2.2 线性表的顺序表示和实现

```
void delnode2(SqList &A,ElemType item)
{     int k=0,i=0; /*k记录值等于item的元素个数*/
      while (i<A.length)
      {    if (A.elem[i]==item)   k++;
           else A.elem[i-k]=A.elem[i]; /*当前元素前移k个位置*/
           i++;
      }
     A.length=A.length-k;  /*顺序表A的长度递减*/
}
```

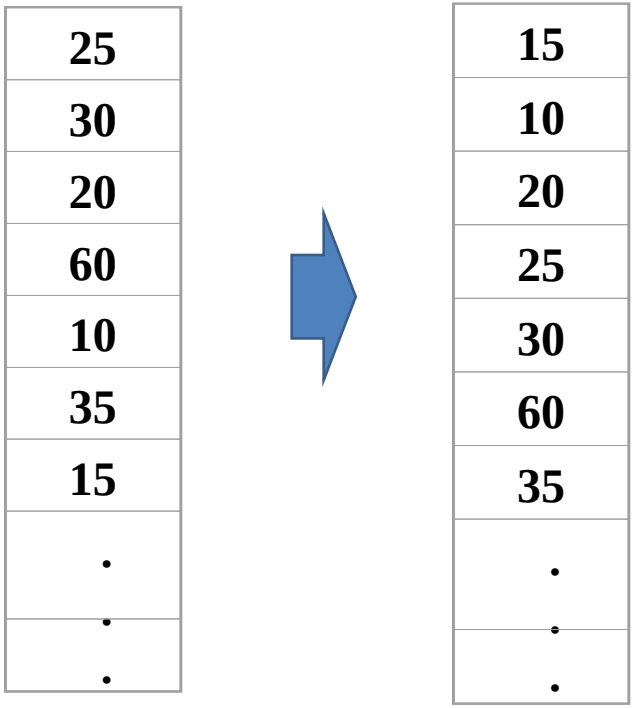
算法2

2.2 线性表的顺序表示和实现

上述算法中只有一个while循环,时间复杂度为O(n)。算法中只用了i,k两个临时变量,空间复杂度为O(1)。

2.2 线性表的顺序表示和实现

例 将顺序表 (a_1，a_2，…，a_n) 重新排列为以 a_1 为界的两部分：a_1 前面的值均比 a_1 小，a_1 后面的值都比 a_1 大(这里假设数据元素的类型具有可比性，不妨设为整型)，操作前后如图所示。这一操作称为划分。a_1 也称为基准。



2.2 线性表的顺序表示和实现

基本思路：

从第二个元素开始到最后一个元素，逐一向后扫描：（1）当前数据元素 aI 比 a1 大时，表明它已经在 a1 的后面，不必改变它与 a1 之间的位置，继续比较下一个。

（2）当前结点若比 a1 小，说明它应该在 a1 的前面，此时将它上面的元素都依次向下移动一个位置，然后将它置入最上方。

2.2 线性表的顺序表示和实现

```
void part(SeqList  *L) {
   int i,j;
   datatype x,y;
   x=L->data[0];    /* 将基准置入 x 中*/
   for(i=1;i<=L->last;i++)
   if(L->data[i]<x)  {
      y = L->data[i];
      for(j=i-1;j>=0;j--)  /*移动*/
         L->data[j+1]=L->data[j];
      L->data[0]=y;
   }
}
```

2.2 线性表的顺序表示和实现

本算法中，有两重循环，外循环执行n - 1次，内循环中移动元素的次数与当前数据的大小有关，当第 i 个元素小于 a_1 时，要移动它上面的 i-1个元素，再加上当前结点的保存及置入，所以移动 i-1+2次，在最坏情况下，a_1 后面的结点都小于 a_1，故总的移动次数为：

$$\sum_{i=2}^{n}(i-1+2)=\sum_{i=2}^{n}(i+1)=\frac{n*(n+3)}{2}$$

即最坏情况下移动数据时间性能为O(n ^{2})。

这个算法简单但效率低。

正在答疑