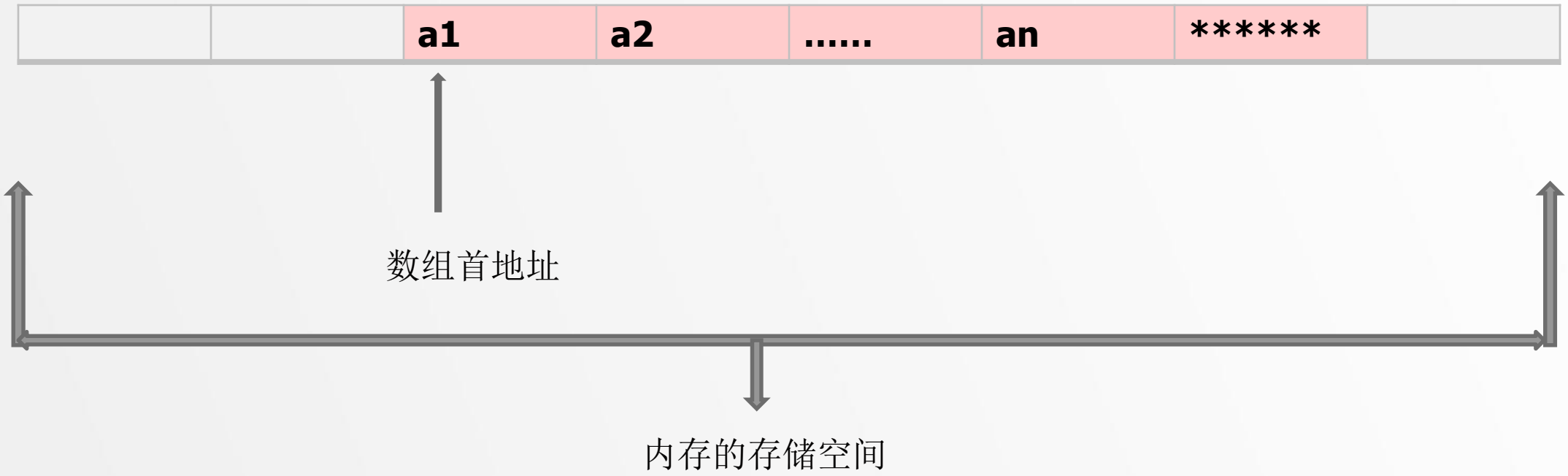
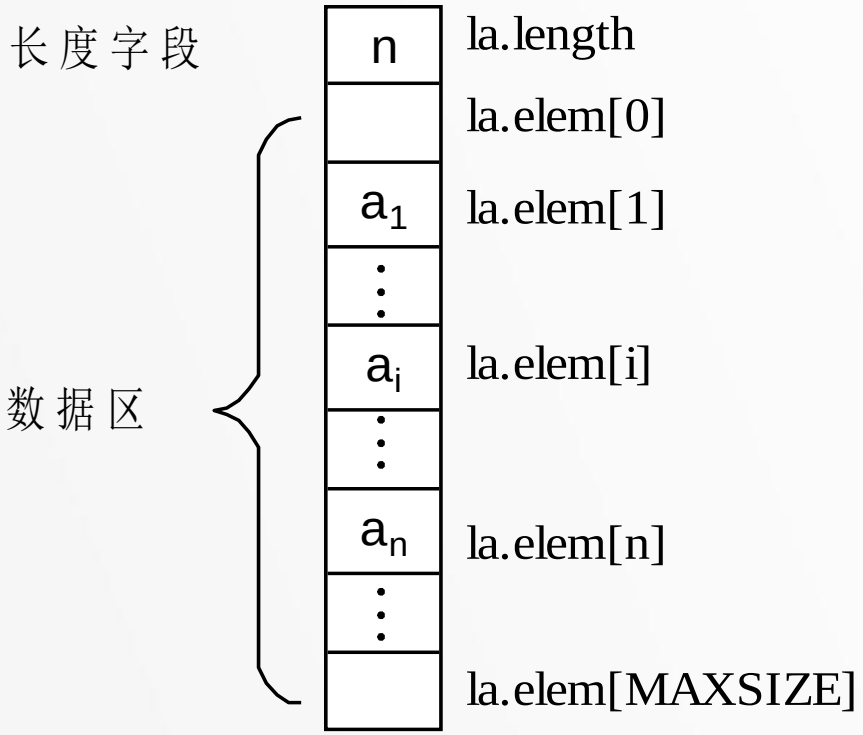
顺序存储结构的存储



顺序存储结构的实现

- 用结构来整合
- 空间如何分配?
  - –静态： 数组
  - –动态： 指针



实现实例（C 语言描述）

```
#define LIST_INIT_SIZE 100
#define LIST_INCREAMENT 10

typedef int ElemType;

typedef struct SqList
{
ElemType *elem;
int length;
int list_size;
}SqList, *Ptr;
typedef Ptr SqlListPtr;
```

线性表顺序存储结构上基本操作的实现

① 初始化--创建线性表

```
Status List_Init(SqListPtr L)
{
    Status s = success;
    L->list_size = LIST_INIT_SIZE;
    L->length = 0;
    L->elem = (ElemType *)malloc(sizeof(ElemType)*L->list_size);
    if (L->elem == NULL)
        s=fatal;
    return s;
}
```

算法时间复杂度：**O(1)**

线性表顺序存储结构上**基本操作的实现**

② 查找

- 按位置查找
  - 检查位置是否合法?
  - 返回相应信息
  - 快
- 按值查找
  - 逐个比较
  - 复杂度分析
    - 最好
    - 最差
    - 平均

线性表顺序存储结构上基本操作的实现

② 查找----按位置

```
Status List_Retrival(SqListPtr L, int pos, ElemType *elem)
{
    Status s = range_error;
    if (L){
        if ((pos-1) >= 0 && (pos-1) < L->length){
            *elem = L->elem[pos-1];
            s = success;
        }
    }
    else
    s = fatal;
    return s;
}
```

算法时间复杂度：O(1)

线性表顺序存储结构上基本操作的实现

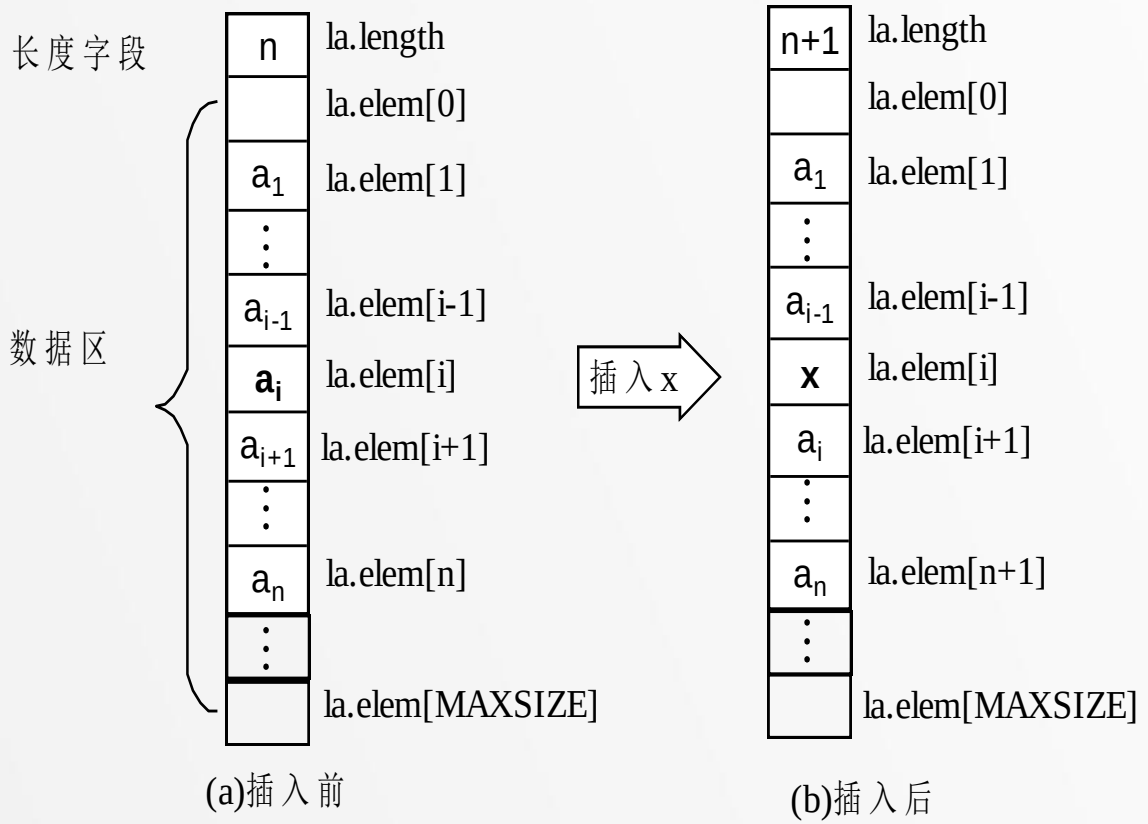
② 查找----按值查找位置

```
Status List_Locate(SqListPtr L, ElemType elem, int *pos)
{
    Status s = range_error;
    if (L){
        for (int i = 0; i < L->length; ++i){
            if (L->elem[i] == elem){
                *pos = i+1;
                s = success;
                break;
            }
        }
    }
    else s = fatal;
        return s;
}
```

算法时间复杂度：**O(N)**

③ 插入元素操作

- 第i个数据元素之前插入新数据元素x
  - $(a_1,a_2,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n) \rightarrow (a_1,a_2,\ldots,a_{i-1},x,a_i,a_{i+1},\ldots,a_n)$



③ 插入元素操作

- 步骤：

  (1) 检查插入位置是否合法，如果合法则继续，否则退出；

  (2) 判表是否已占满；因为是事先分配空间，可能存在所分配存储空间全部被占用的情况，此时也不能实现插入。

  (3) 若前面检查通过则数据元素依次向后移动一个位置；为避免覆盖原数据，应从最后一个向前依次移动。

  (4) 新数据元素放到恰当位置；

  (5) 表长加1。

③ 插入元素操作

```
Status List_Insert(SqListPtr L, int pos, ElemType elem)
{
    Status s = range_error;
    if ((pos-1) >= 0 && (pos-1) <= L->length){
        if (L && L->length <L-> list_size ){
            for (int i = L->length-1; i >= pos-1; --i){
                L->elem[i+1] = L->elem[i];
            }
            L->elem[pos-1 ] = elem;
            L->length++;
            s = success;
        }
    }
    else s = fail;
    return s;
}
```

插入算法的时间性能分析

移动次数：

在第i（1≤ i≤ n+1）个位置上插入 e，移动 n - i + 1个元素。

设在第i个位置上作插入的概率为P_i，则平均移动数据元素的次数：

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

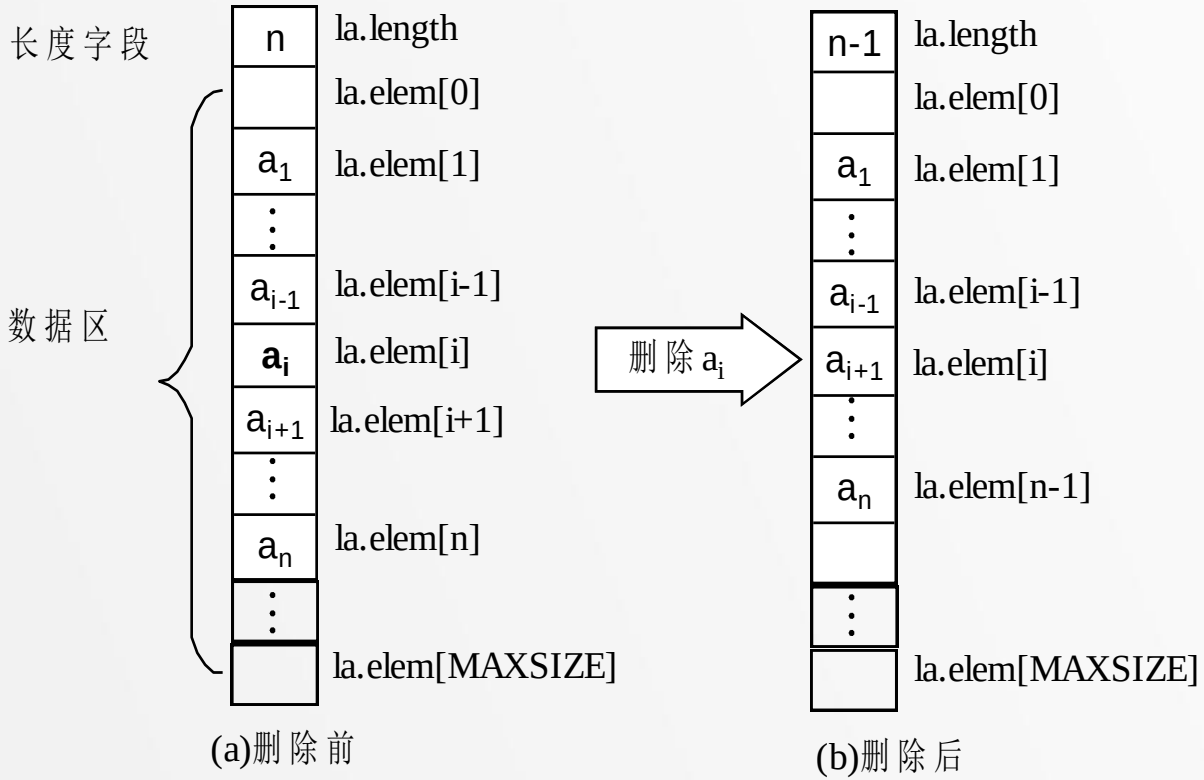
设：

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

表明：顺序表上插入操作需移动表中一半的数据元素。
时间复杂度为O(n)。
最坏情况是在第1个元素前插入(i=1)，需要后移n个元素。

④ 删除元素操作

- 删除操作
  - $(a_1,a_2,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n)$ ➔ $(a_1,a_2,\ldots,a_{i-1},a_{i+1},\ldots,a_n)$



(a)删除前　　(b)删除后

④ 删除元素操作

- 步骤
  - (1) 检查删除位置是否合法；
  - (2) 若检查通过，数据元素依次向前移动一个位置；
  - (3) 表长减1。

④ 删除元素操作

```
Status List_delete(SqListPtr L, int pos)
{
    Status s = range_error;
    if ((pos-1) >= 0 && (pos-1) < L->length){
        If(L && L->length > 0){
            for (int i = pos ; i < L->length; ++i){
                L->elem[i-1] = L->elem[i ];
            L->length--;
            s=success;
            }
        }
    }
    else s = fail;
    return s;
}
```

时间复杂性分析

基本操作：移动元素
最好：删最后一个，移动0
最差：删第一个，移动n-1
平均：删除第i个元素，移动n-i 个数据元素，概率pi=1/n

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

思考讨论题

- 已知线性表A={1,3,5},B={2,4,5,6}
- 请采用线性表的基本操作实现C=A∩B

答案：
(1) 初始化空线性表C
(2) 依次访问线性表B的元素List_Retrival，元素存放在elem参数中
(3) 查询elem是否在线性表A当中List_Locate
(4) 如果返回查询状态是失败，则将该元素插入C线性表当中List_Insert