

中山大学网络安全学院必修课程

第8节 指针练习

C Programming

岳 晟

网络安全学院 工学园2栋508

yuesh5@mail.sysu.edu.cn

2026年秋季

1. 指针区间原地反转

题目描述

编译器: C (gcc) [编辑](#)

请实现一个函数 `reverse_range`，将整型数组指定下标闭区间内的元素原地逆序存放，区间外的元素保持不变。例如，数组 `1 2 3 4 5 6` 反转区间 `[1, 4]` 后得到 `1 5 4 3 2 6`。

函数接口定义

[c]

[复制内容](#) [格式](#) [全屏](#)

```
1 void reverse_range(int *first, int *last);
```

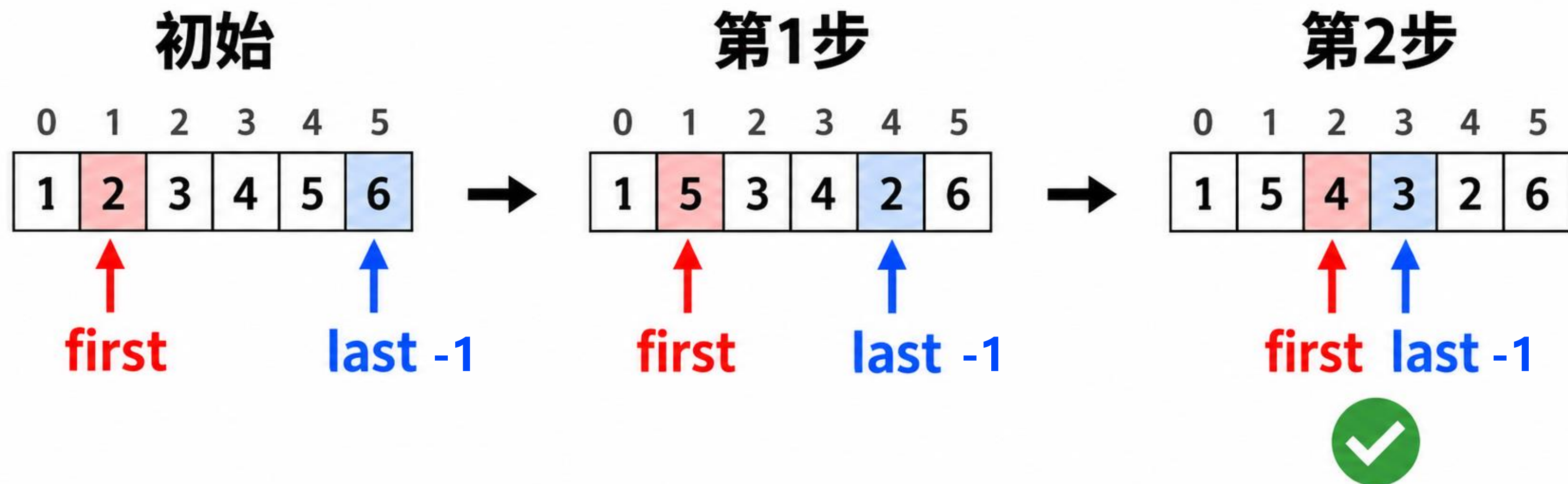
- `first`：指向待反转区间的第一个元素；
- `last`：指向待反转区间最后一个元素之后的位置；
- `[first, last)`：为左闭右开的指针区间，反转不包含 `last` 所指位置。

1. 指针区间原地反转

算法思路：

1. 两个指针从**区间两端向中间**移动
2. 每次**交换**两个指针所指的元素
3. 区间中**不足两个**元素时结束

1. 指针区间原地反转



2. 数组中第k小的元素

题目描述

编译器: C (gcc) [编辑](#)

请实现函数 `kth_smallest`，在一个包含 `n` 个整数的数组中，求出第 `k` 小的元素。第 `k` 小是指将数组中的全部元素按从小到大排序后，位于第 `k` 个位置的数，重复元素也分别计数。例如，数组 `5 2 2 8 3` 从小到大排列为 `2 2 3 5 8`，第 `3` 小的元素为 `3`。

提示：可使用冒泡排序思想。

函数接口定义

[c]

复制内容 格式 全屏

1	<code>int kth_smallest(int *first, int *last, int k);</code>
---	--

- `first`：指向数组第一个有效元素；
- `last`：指向最后一个有效元素之后的位置，`[first, last)` 为左闭右开指针区间；
- `k`：要查找的次序，从 `1` 开始；
- 返回值：区间内第 `k` 小的整数。

排序算法



1 网上购物

在淘宝、京东等购物平台上，可以将商品按照价格从低到高、销量从高到低、评价数量从多到少排列，帮助用户快速筛选商品。



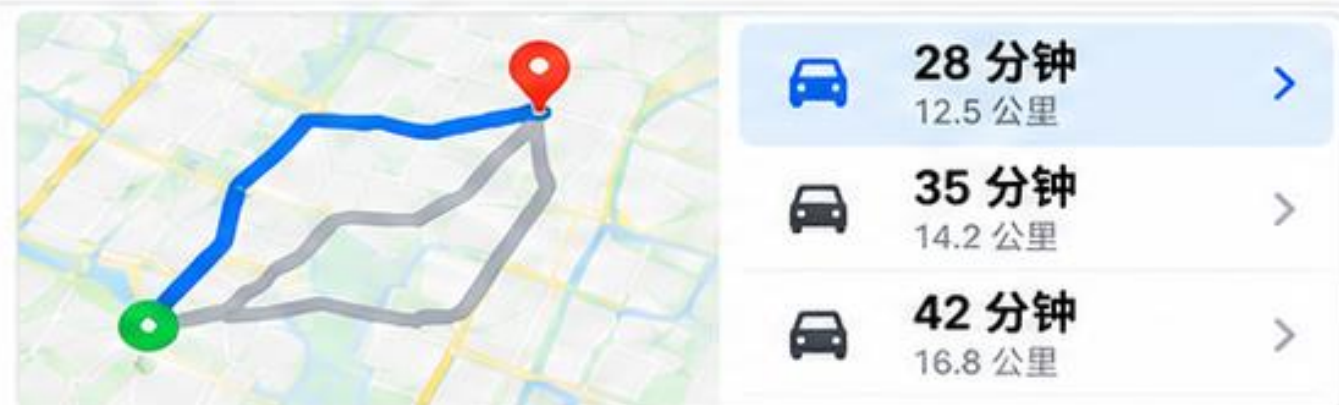
2 文件管理

在电脑的文件夹中，可以按照文件名、修改时间、文件大小排列文件。例如，将最近修改的文件排在最前面。



3 体育比赛排名

马拉松比赛按照完成时间从短到长排名，足球联赛按照积分、净胜球等规则排列球队，确定名次。



4 地图与路线规划

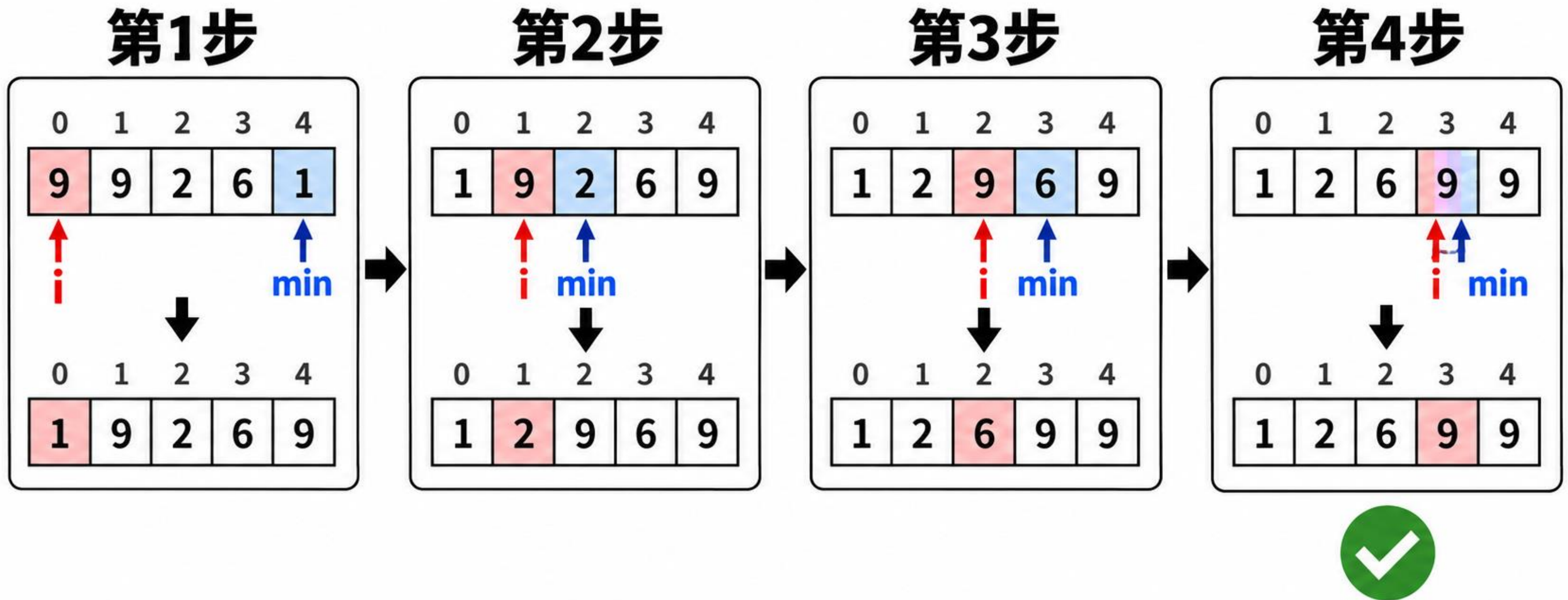
导航软件可以将不同路线按照预计行驶时间、距离或费用排列，让用户选择合适的路线。

排序算法1：选择排序 (Selection Sort)

算法思路：

1. 从未排序区间中找出最小的元素
2. 将最小元素与未排序区间的第一个元素交换
3. 缩小未排序区间，重复上述操作，直到排序完成

排序算法1：选择排序 (Selection Sort)



排序算法1：选择排序 (Selection Sort)

```
#include <stddef.h>

void selection_sort_int(int *nums, size_t n) {
    for (size_t i = 0; i < n - 1; i++) {
        // 找出未排序区间的最小元素
        size_t min_idx = i;
        for (size_t j = i + 1; j < n; j++) {
            if (nums[j] < nums[min_idx]) {
                min_idx = j;
            }
        }
        // 将最小元素交换到未排序区间的开头
        if (min_idx != i) {
            int temp = nums[i];
            nums[i] = nums[min_idx];
            nums[min_idx] = temp;
        }
    }
}
```

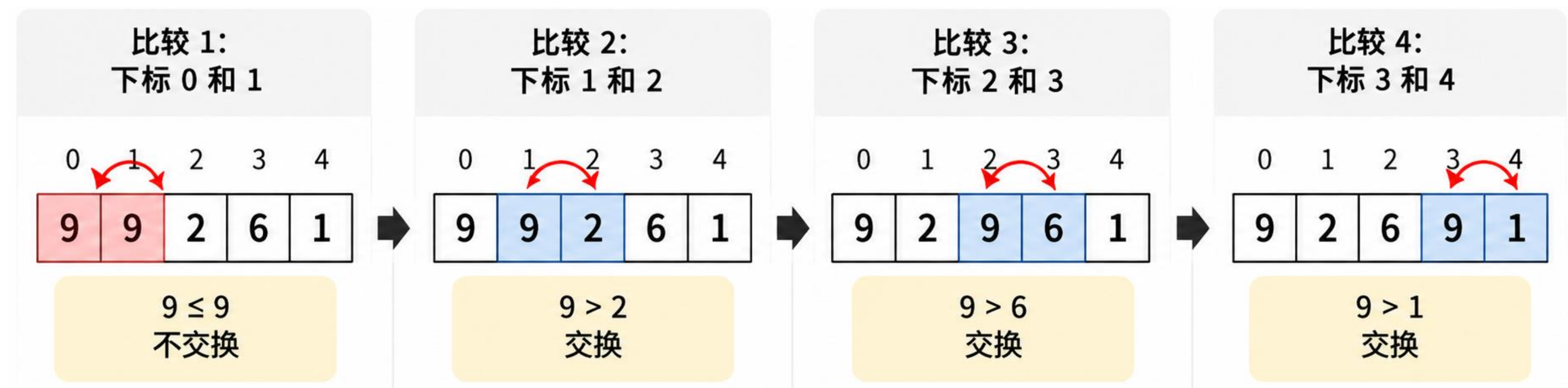
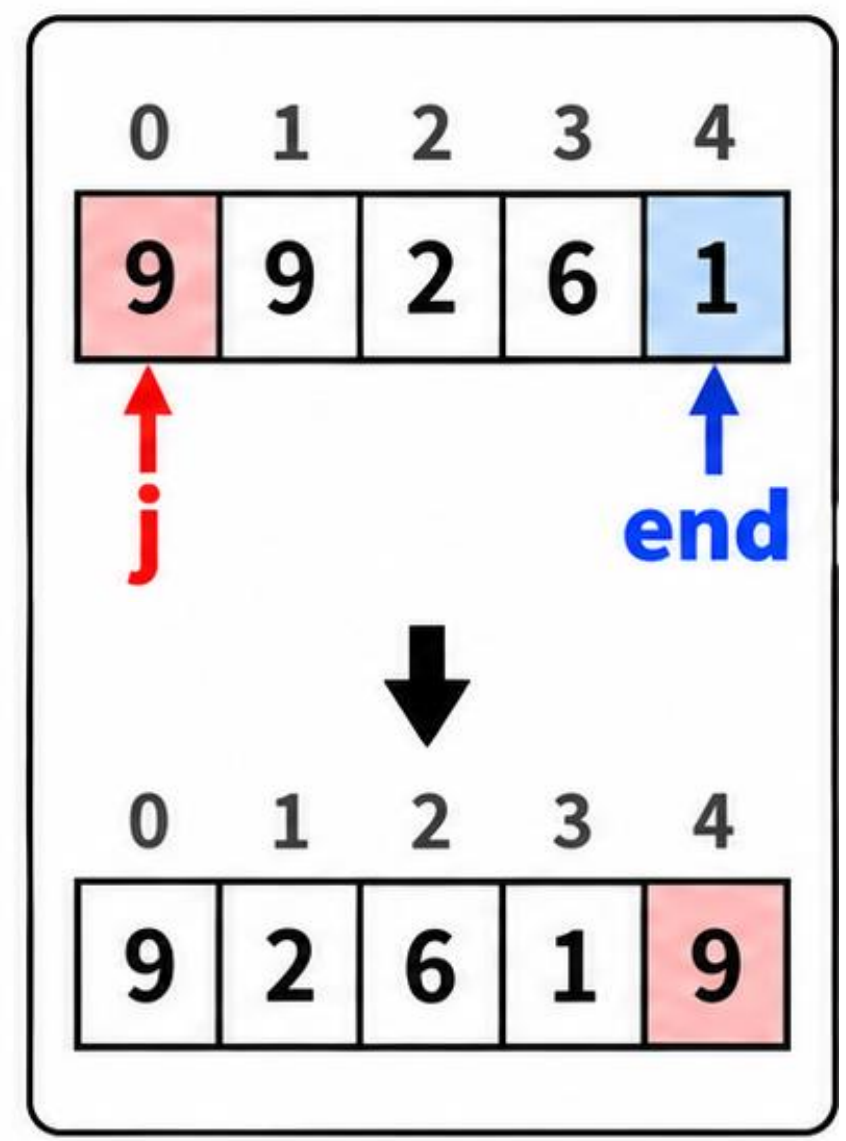
排序算法2：冒泡排序 (Bubble Sort)

算法思路：

1. 从左到右依次比较相邻的两个元素
2. 如果前一个元素大于后一个元素，则交换它们
3. 每轮将未排序区间的最大元素移到末尾，直到排序完成

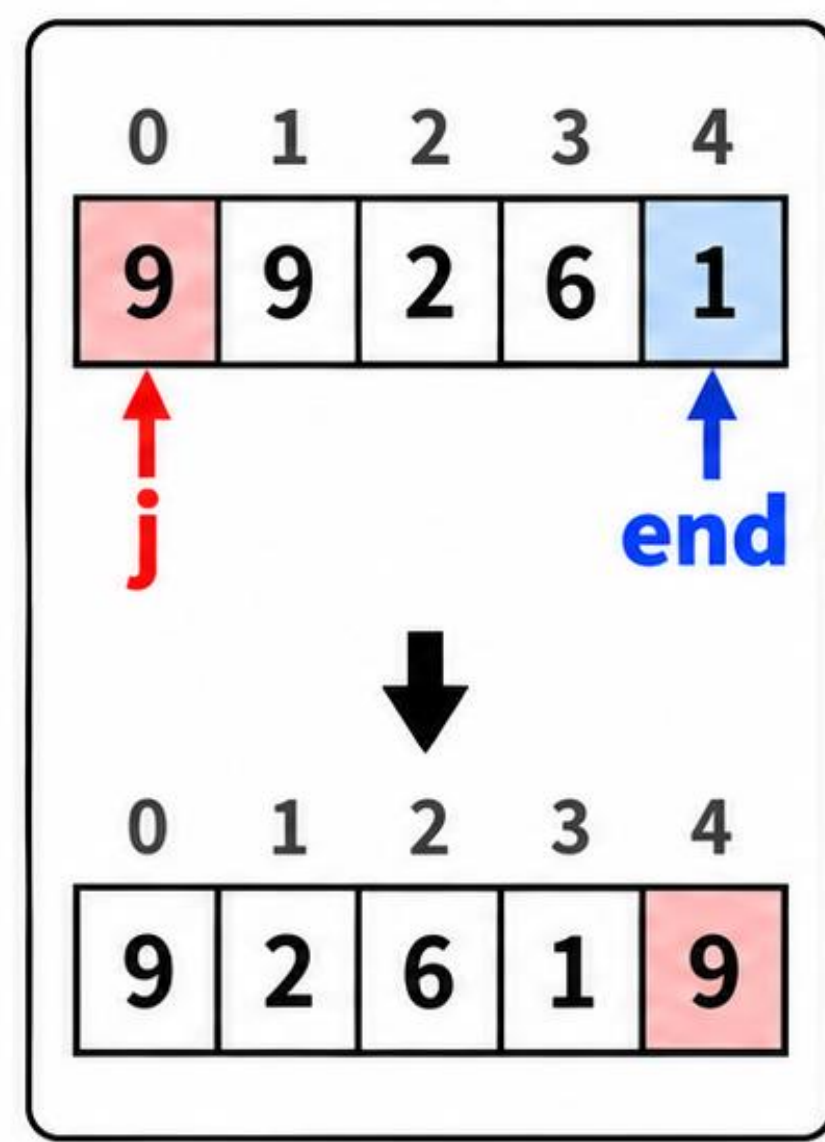
排序算法2：冒泡排序 (Bubble Sort)

第1步

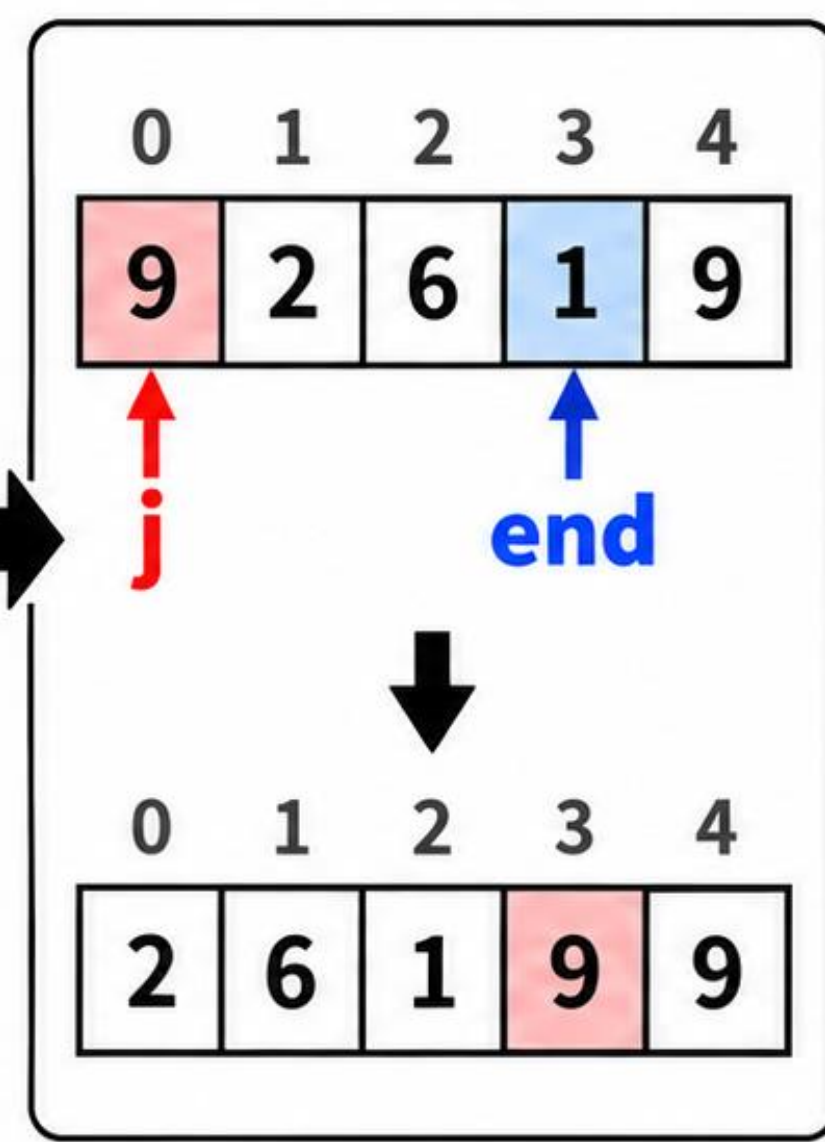


排序算法2：冒泡排序 (Bubble Sort)

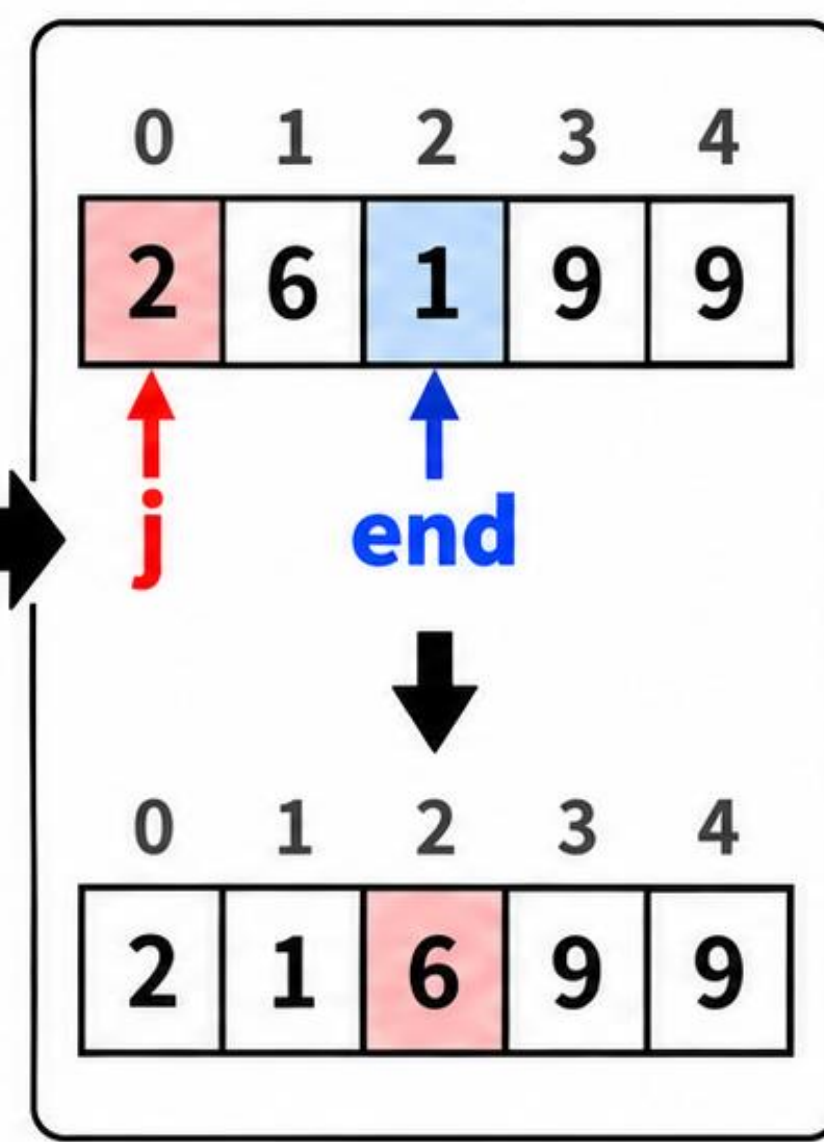
第1步



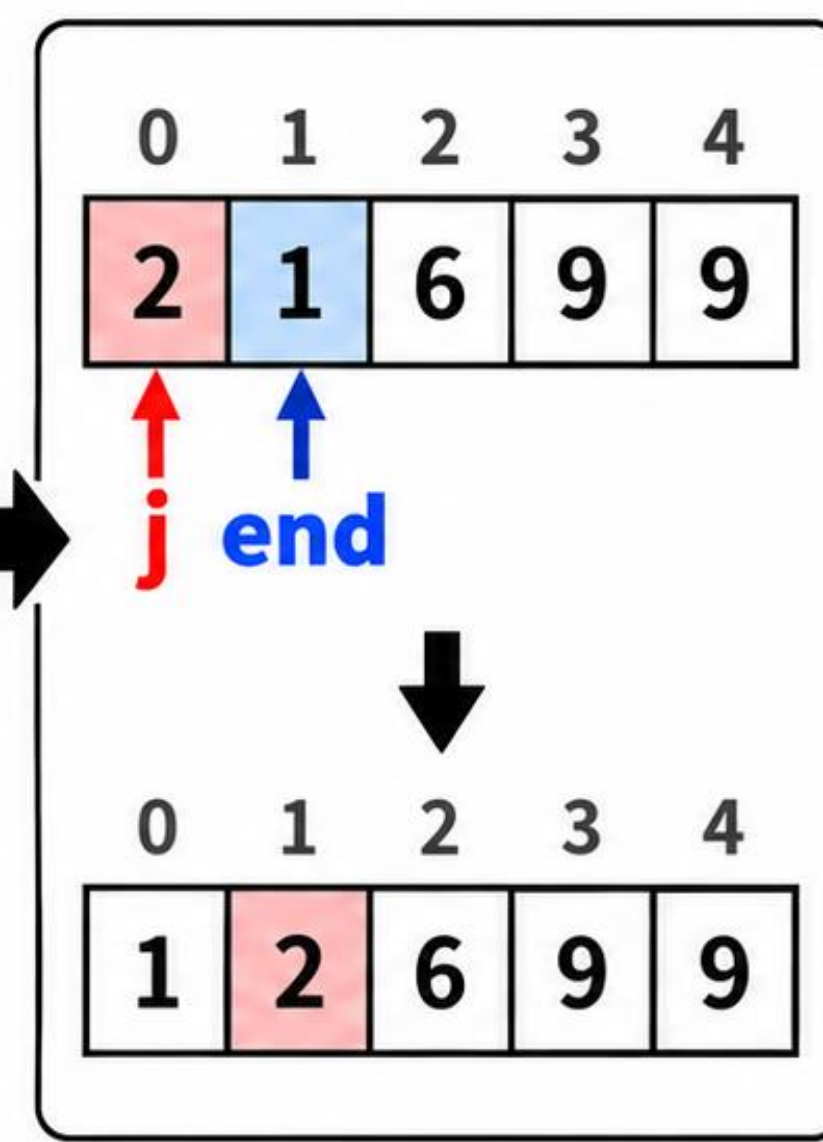
第2步



第3步



第4步



排序算法2：冒泡排序 (Bubble Sort)

```
#include <stddef.h>
void bubble_sort_int(int *nums, size_t n) {
    for (size_t i = 0; i < n - 1; i++) {
        int swapped = 0;
        // 相邻元素比较与交换
        for (size_t j = 0; j < n - i - 1; j++) {
            if (nums[j] > nums[j + 1]) {
                int temp = nums[j];
                nums[j] = nums[j + 1];
                nums[j + 1] = temp;
                swapped = 1;
            }
        }
        // 本轮没有交换，说明数组已经有序
        if (!swapped) {
            break;
        }
    }
}
```

选择排序 vs 冒泡排序

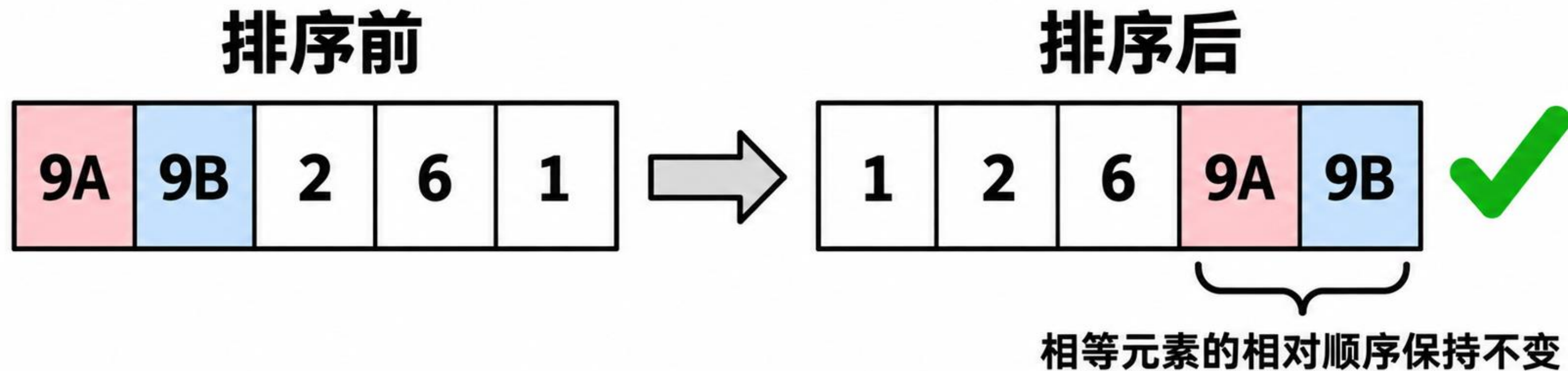
维度	选择排序	冒泡排序
时间复杂度 (最坏)	$O(n^2)$	$O(n^2)$
时间复杂度 (最佳)	$O(n^2)$	$O(n)$
空间复杂度	$O(1)$	$O(1)$
稳定性	不稳定	稳定
交换次数	$O(n)$	$O(n^2)$

时间复杂度是什么意思呢？

- 时间复杂度是衡量一个算法**执行所需时间**随**输入规模增长**而变化的趋势的度量
- 它描述的并不是具体的秒数或毫秒数（因为这会因硬件、编程语言等因素而异），而是描述算法基本操作数**随输入规模增长的数量级**
 - $O(n)$ 意味着当 n 足够大时，存在常数 c 使得计算时间 $T < cn$
 - $O(n^2)$ 意味着当 n 足够大时，存在常数 c 使得计算时间 $T < cn^2$
- 我们关注的是当输入规模 n （例如数组的元素个数）足够大时，运行时间的**主导部分**是什么，因此常数因子和低阶项在 n 很大时会被忽略

排序算法的稳定性是什么意思呢？

- 如果待排序序列中存在**值相等的元素**，在排序完成后，这些相等元素的**相对顺序保持不变**，那么就说这个排序算法是**稳定的**



2. 数组中第k小的元素

题目描述

编译器: C (gcc) [编辑](#)

请实现函数 `kth_smallest`，在一个包含 `n` 个整数的数组中，求出第 `k` 小的元素。第 `k` 小是指将数组中的全部元素按从小到大排序后，位于第 `k` 个位置的数，重复元素也分别计数。例如，数组 `5 2 2 8 3` 从小到大排列为 `2 2 3 5 8`，第 `3` 小的元素为 `3`。

提示：可使用冒泡排序思想。

函数接口定义

[c]

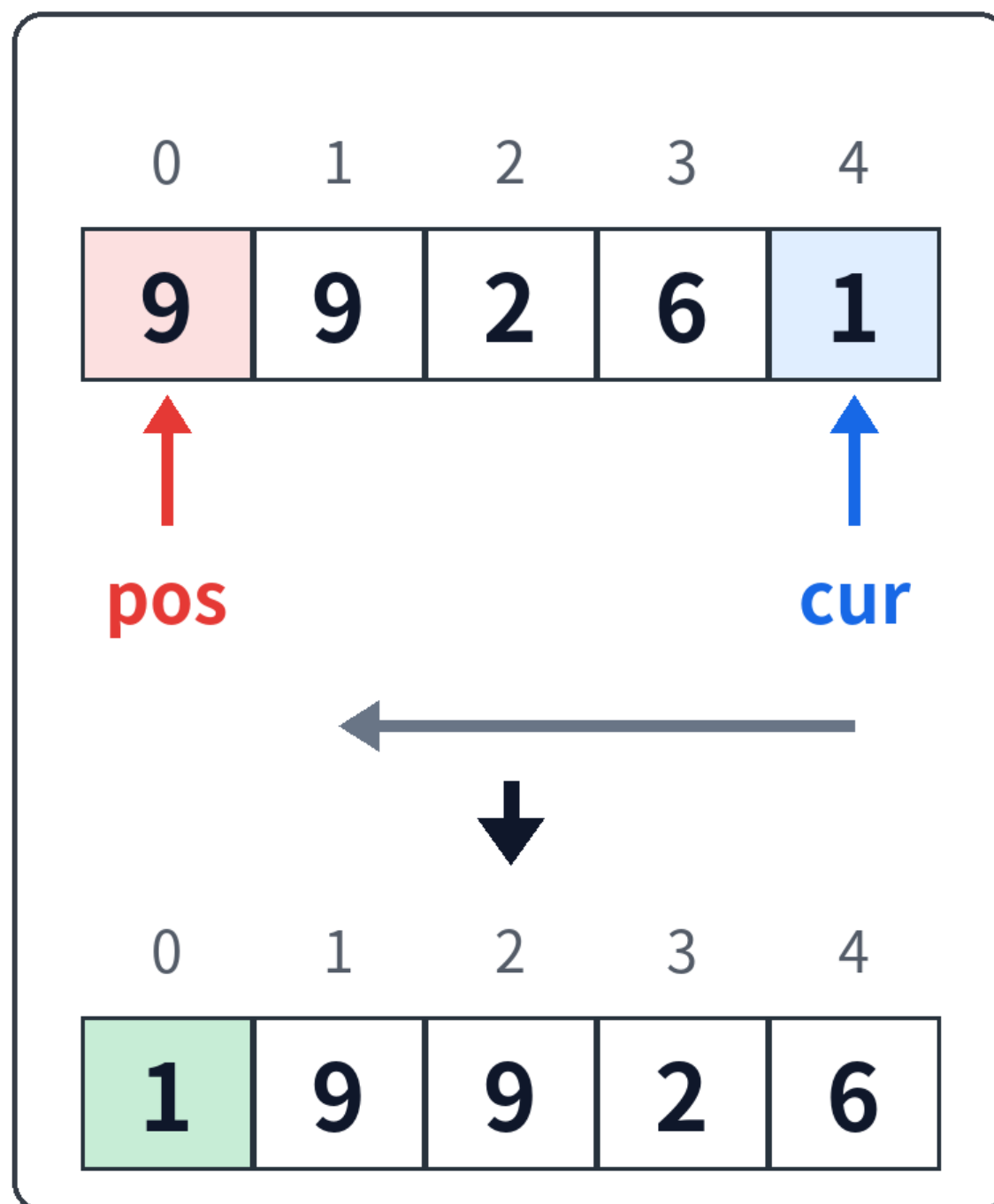
复制内容 格式 全屏

1 int kth_smallest(int *first, int *last, int k);

- `first`：指向数组第一个有效元素；
- `last`：指向最后一个有效元素之后的位置，`[first, last)` 为左闭右开指针区间；
- `k`：要查找的次序，从 `1` 开始；
- 返回值：区间内第 `k` 小的整数。

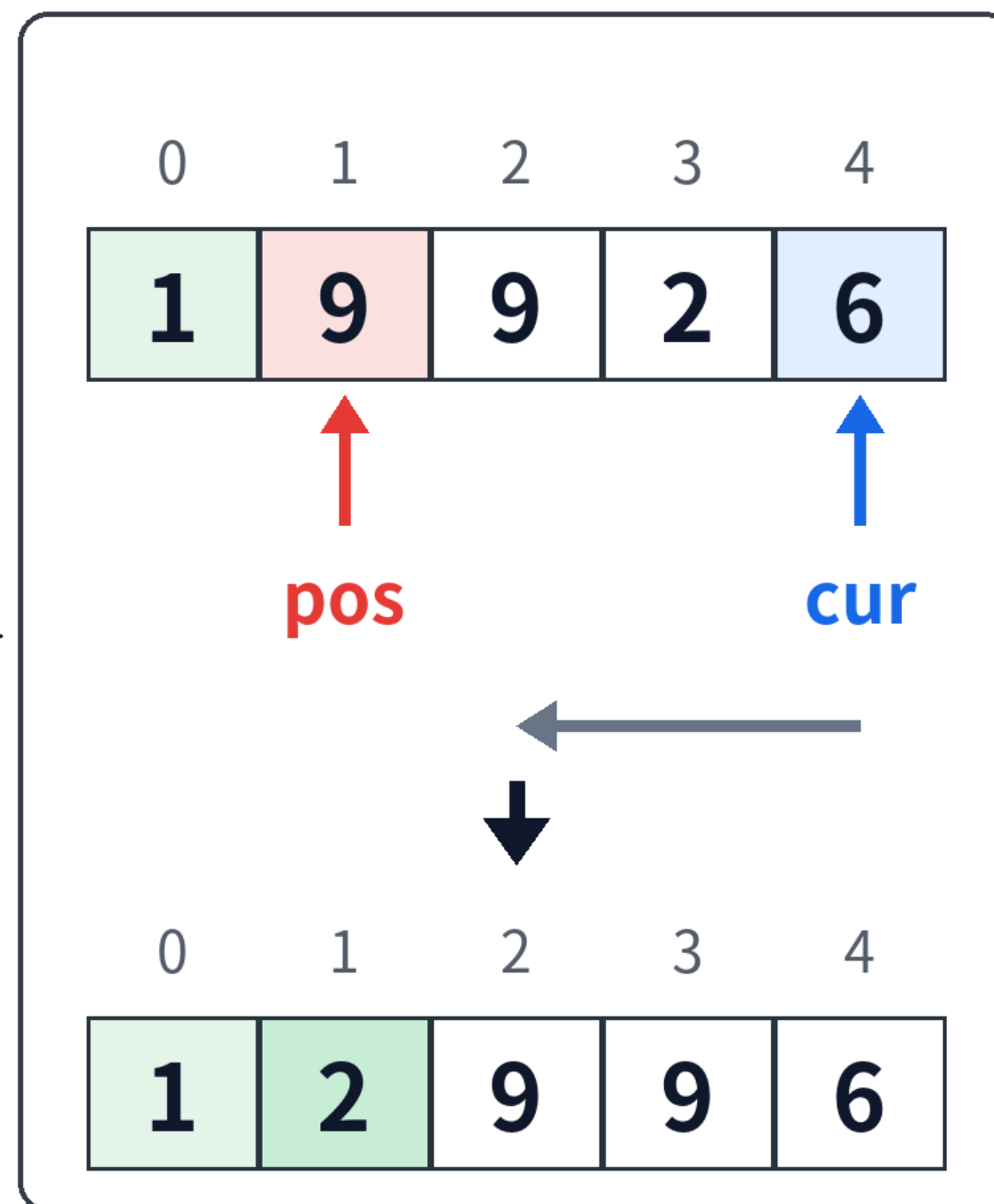
2. 数组中第k小的元素

第1轮

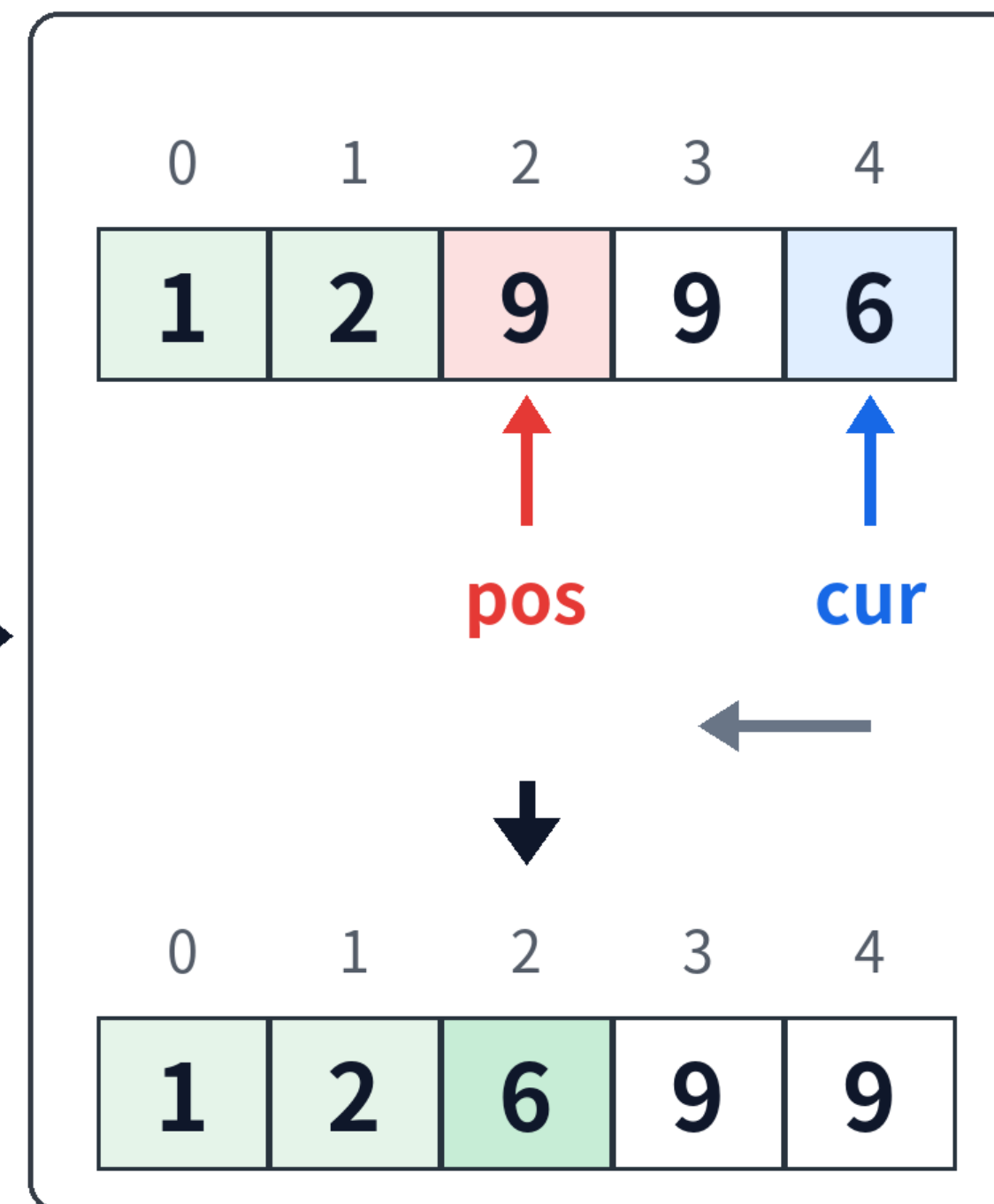


k = 3

第2轮



第3轮



返回 6

3. 移动零

题目描述

编译器: C (gcc) [编辑](#)

给定一个包含整数的数组 `nums`，将所有等于 `0` 的元素移动到数组末尾，其他非零元素必须保持原来的相对顺序。只能修改 `nums` 原有元素，不使用额外数组。

函数接口定义

[c]	
1	<code>void move_zeroes(int nums[], int nums_size);</code>

- `nums`: 待处理的整数数组。
- `nums_size`: 数组的有效元素个数 (`1 <= nums_size <= 100000`)。

输入样例1:

[in]		复制内容	✖
1	7		
2	0 4 0 -2 7 0 5		

输出样例1:

[out]		复制内容	✖
1	4 -2 7 5 0 0 0		

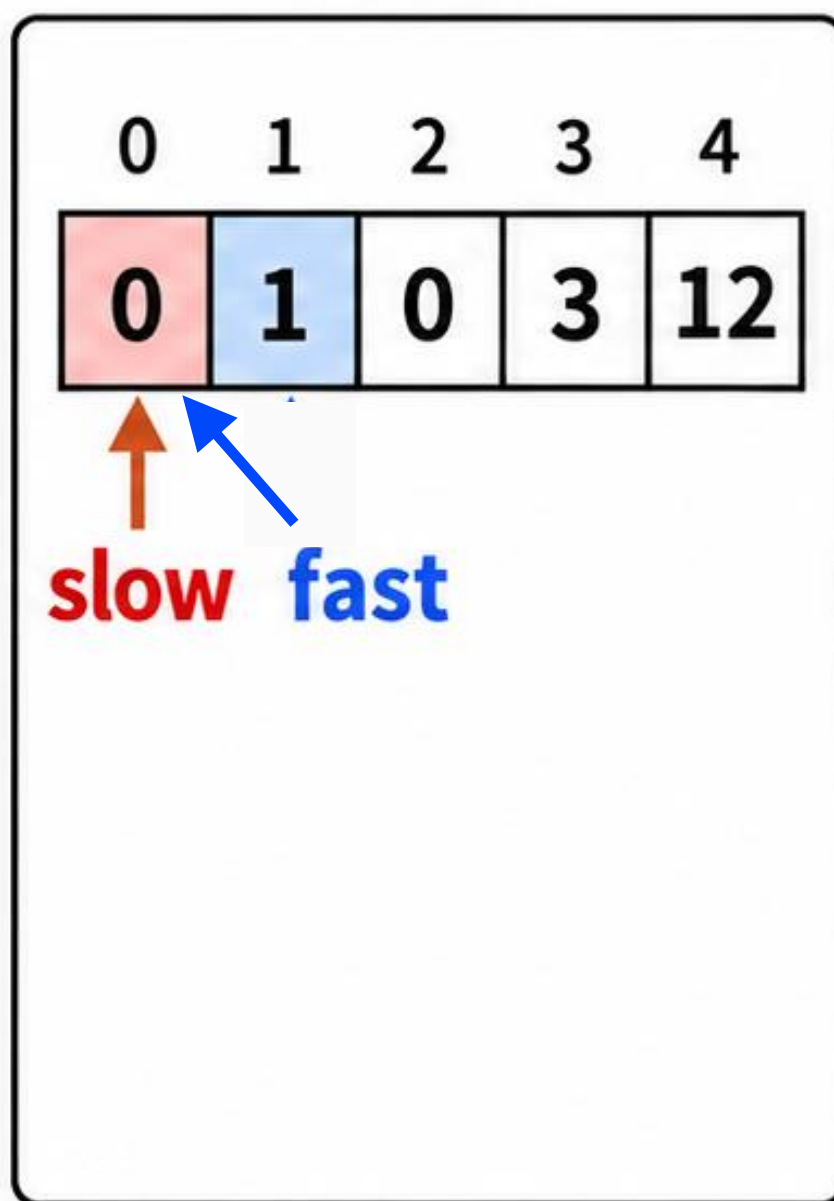
3. 移动零

类似数组去重，算法思路：

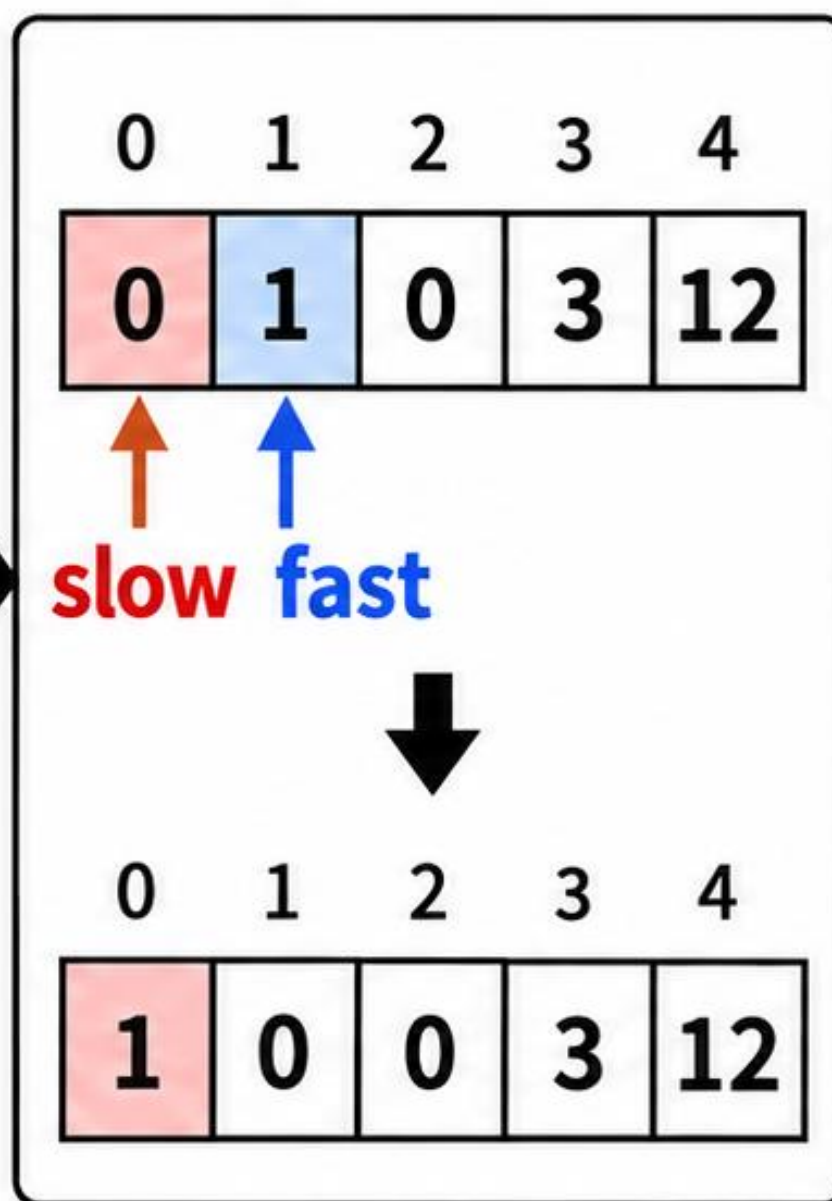
1. 设置**快慢指针**：slow指向下一非零元素应放位置，fast用于遍历数组
2. fast**从左向右**移动，遇非零元素时，将其与**slow所指元素交换**，并将slow向右移动
3. 当fast遍历完整整个数组时结束

3. 移动零

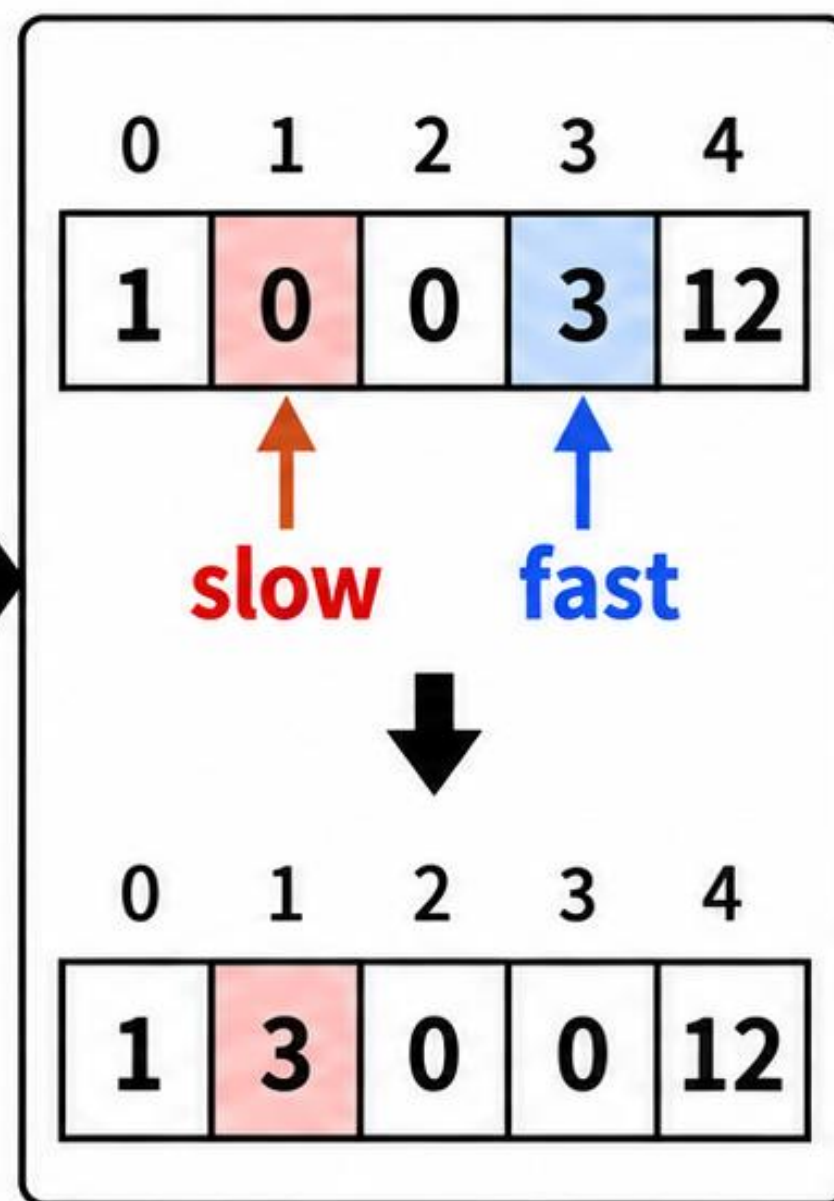
第0步



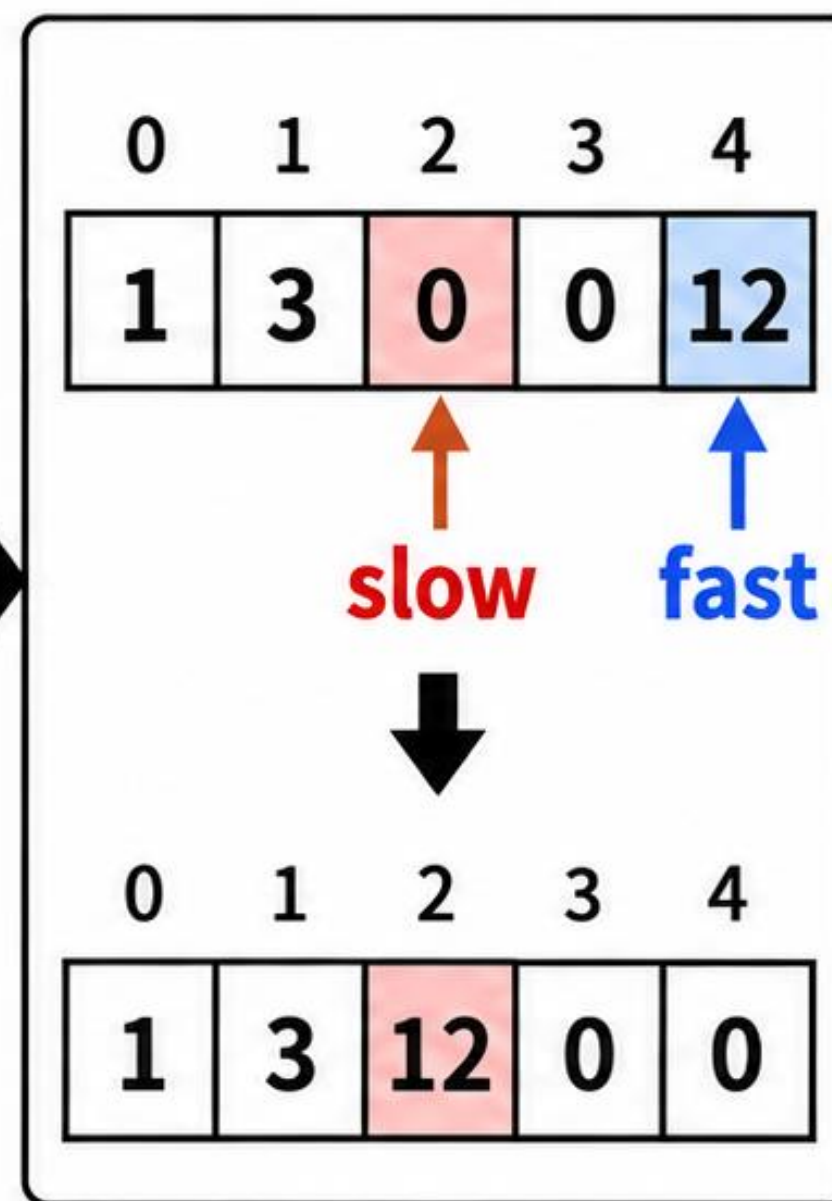
第1步



第2步



第3步



4. 合并两个有序数组

给定两个按非递减顺序排列的整数序列，长度分别为 `m` 和 `n`。数组 `nums_1` 已预留 `m+n` 个元素的存储空间，其中前 `m` 个是有效数据、后 `n` 个是占位空间；数组 `nums_2` 有 `n` 个有效数据。请将两个序列合并为一个非递减序列，并把全部结果直接存入 `nums_1`。要求在 `nums_1` 内完成合并，不使用额外数组。

函数接口定义

[c]

复制内容 格式 全屏

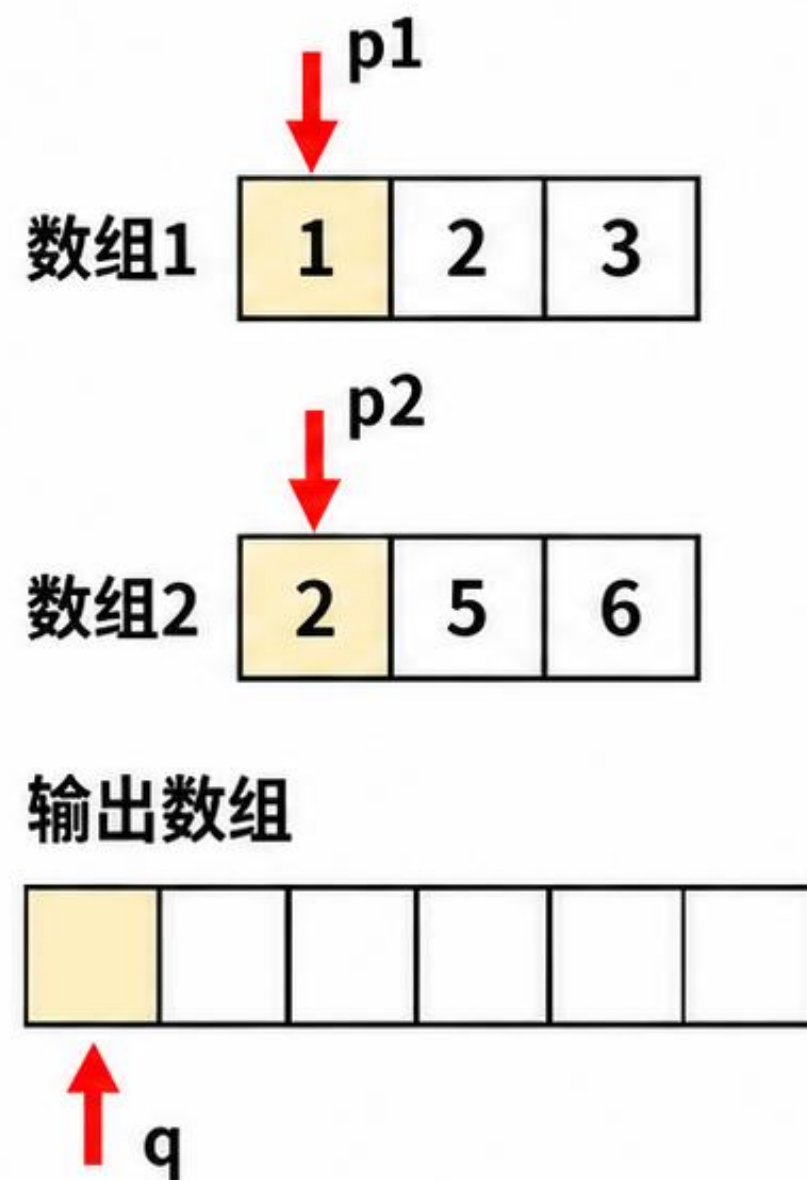
```
1 void merge(int nums_1[], int nums_1_size, int m, int nums_2[], int nums_2_size, int n);
```

- `nums_1`: 数组 1。
- `nums_1_size`: 数组 1 的容量，等于 `m+n`。
- `m`: 数组 1 中原有的有效元素个数。
- `nums_2`: 数组 2。
- `nums_2_size`: 数组 2 的容量。
- `n`: 数组 2 中原有的有效元素个数（`0 <= m,n`，`1 <= m+n <= 100000`）。

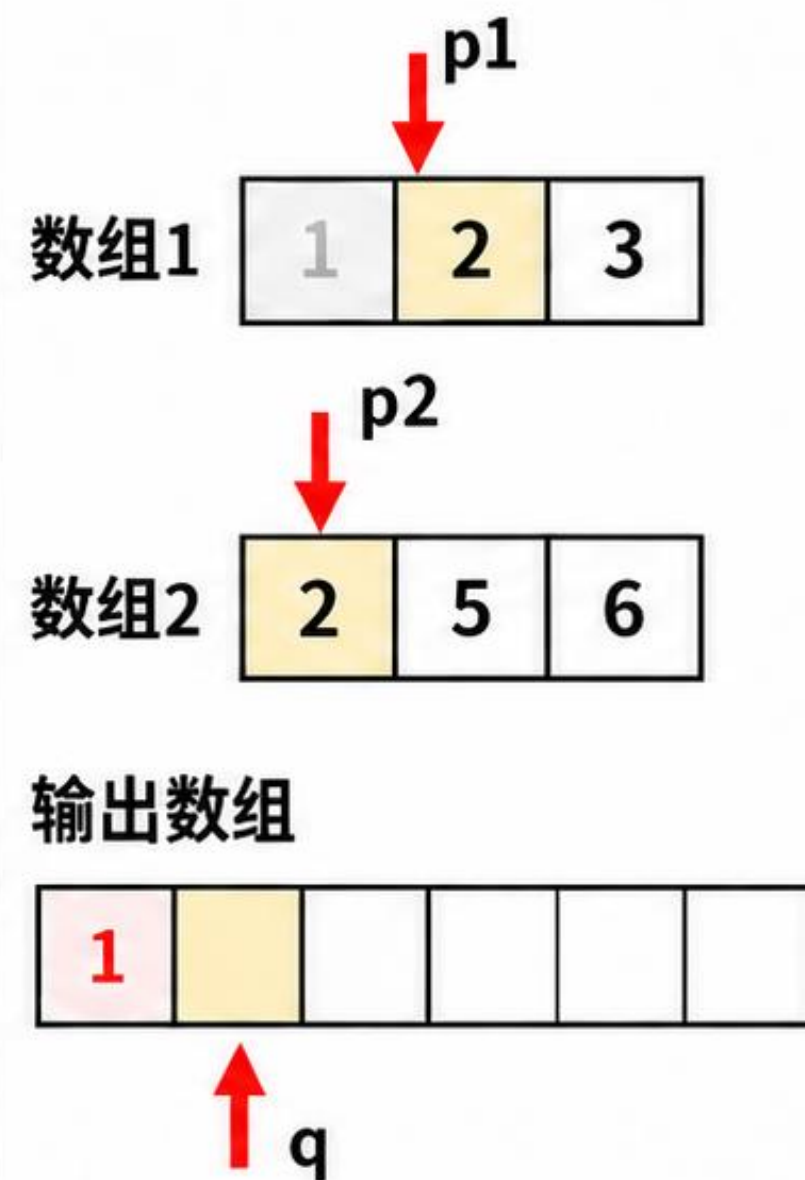
4. 合并两个有序数组

方法一：用一个 $m+n$ 的临时数组，总将最小的数写入新数组，直到写完

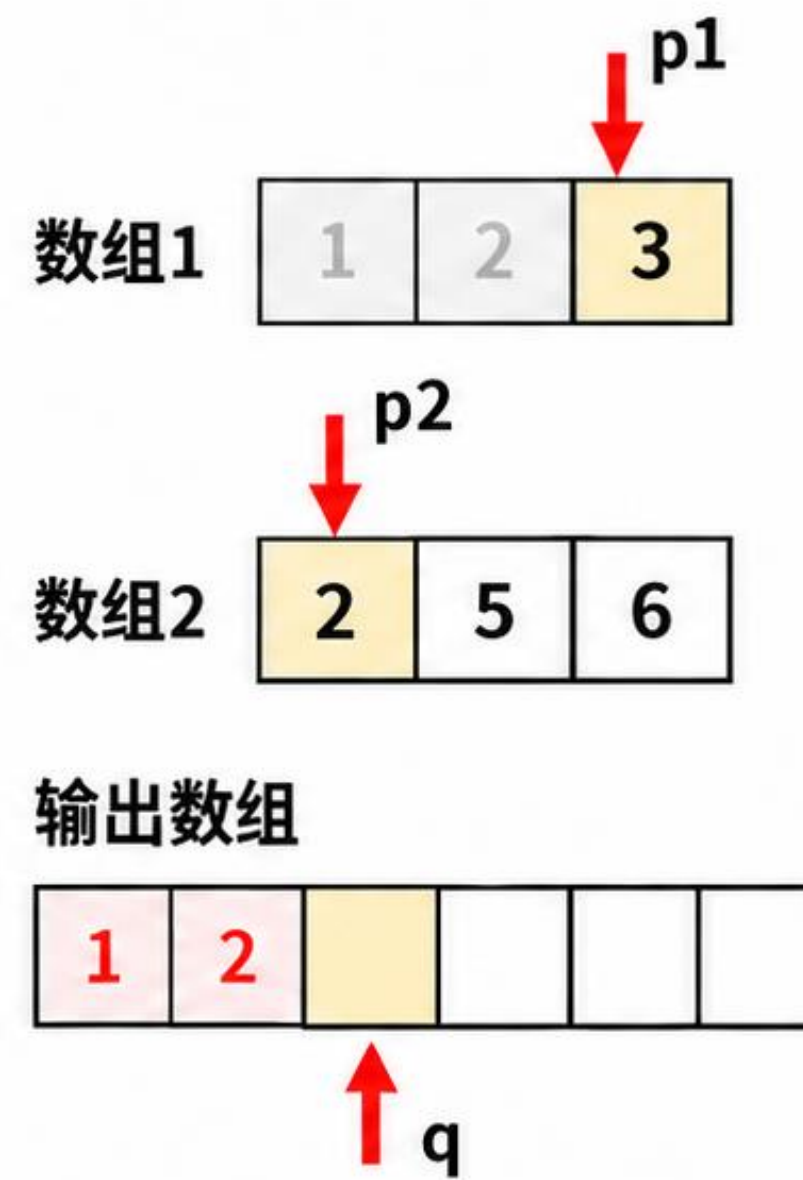
第0步：初始化



第1步：写入 1



第2步：写入 2

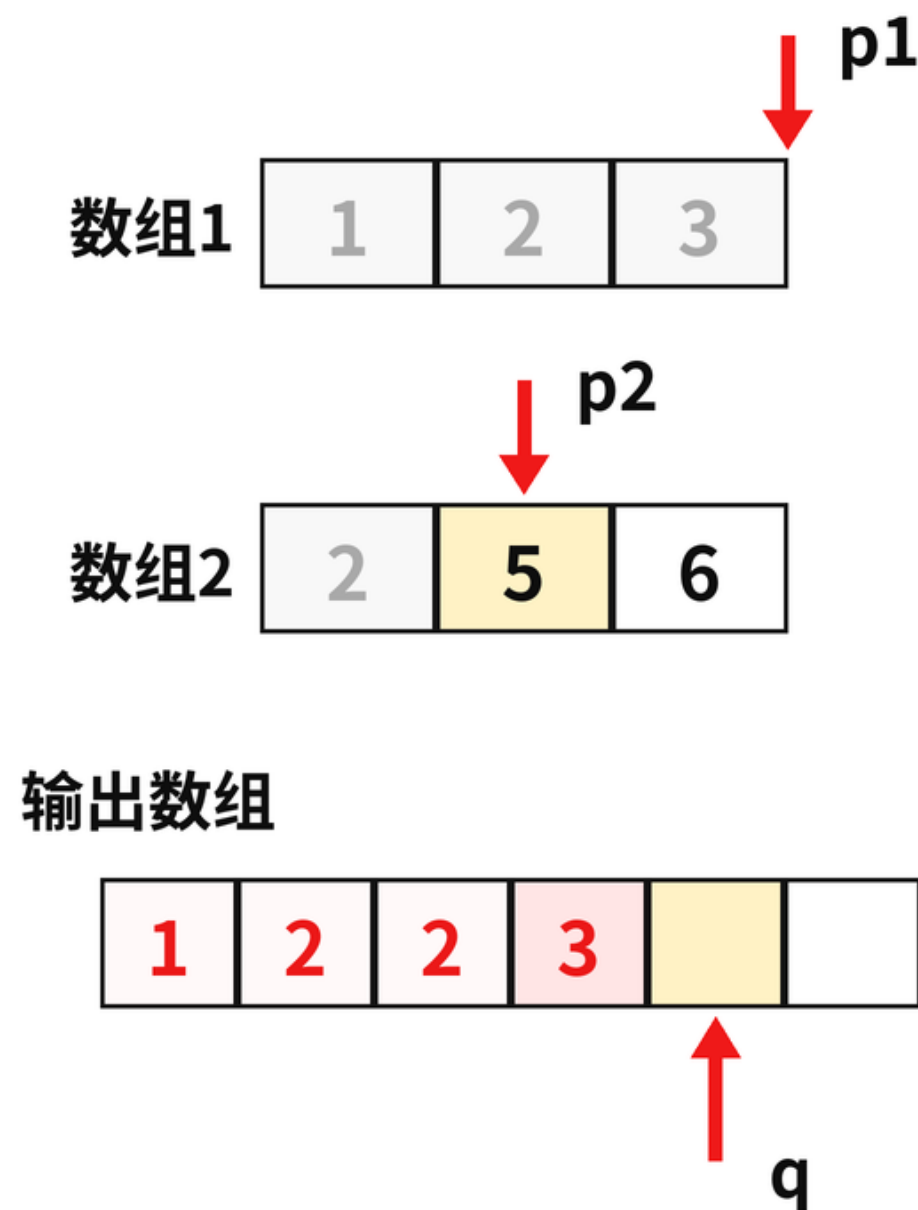


第3步：写入 2

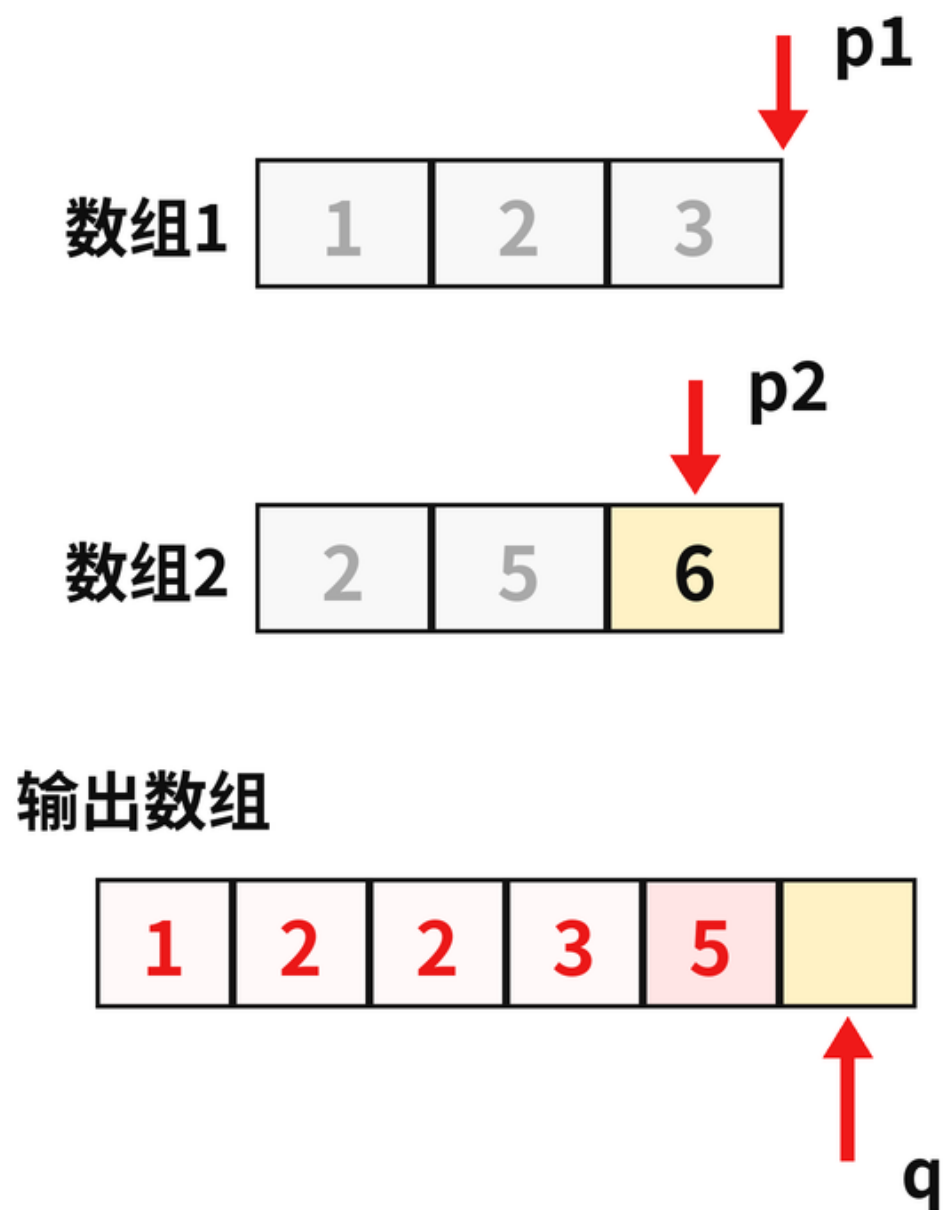


4. 合并两个有序数组

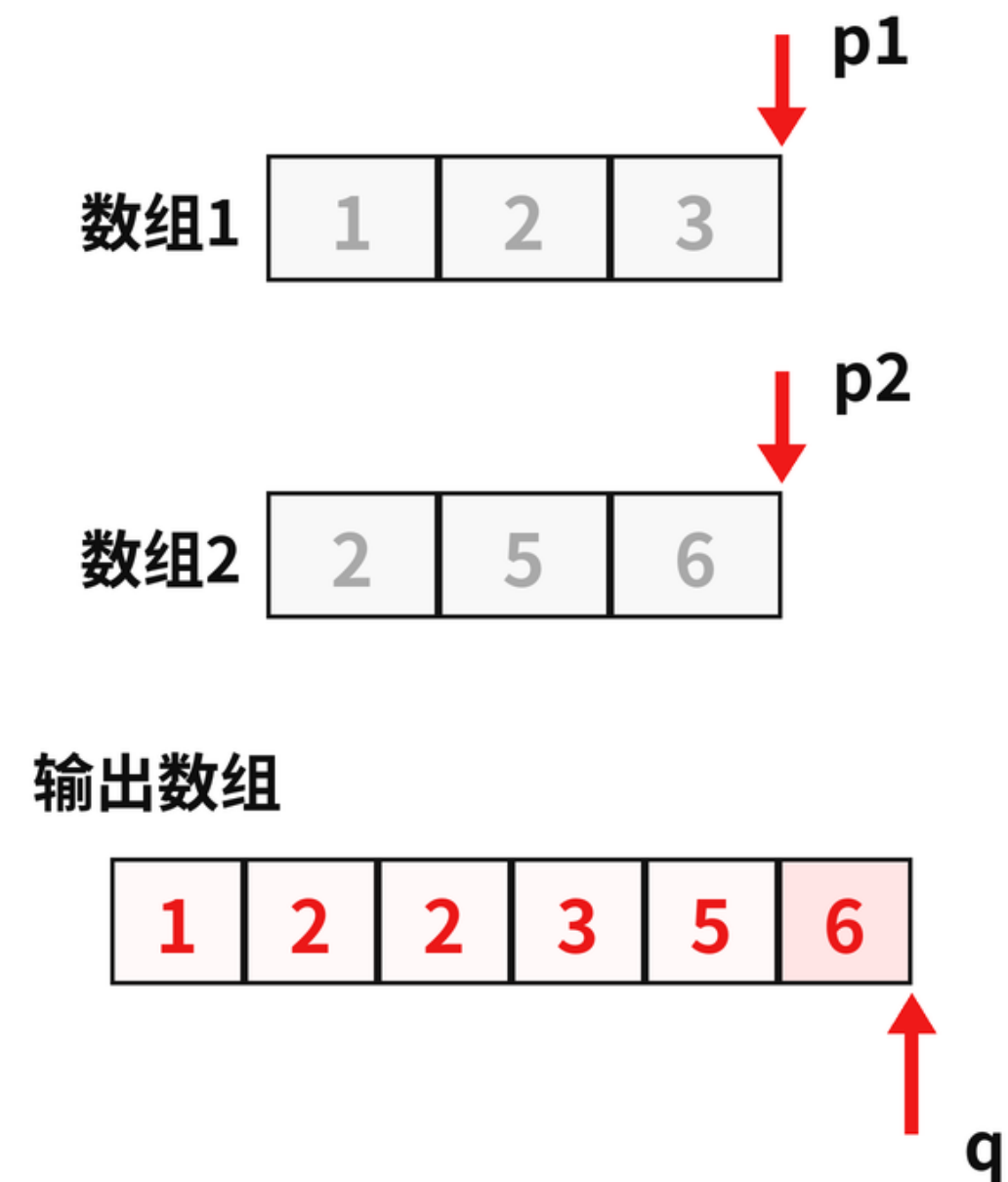
第4步：写入 3



第5步：写入 5



第6步：写入 6



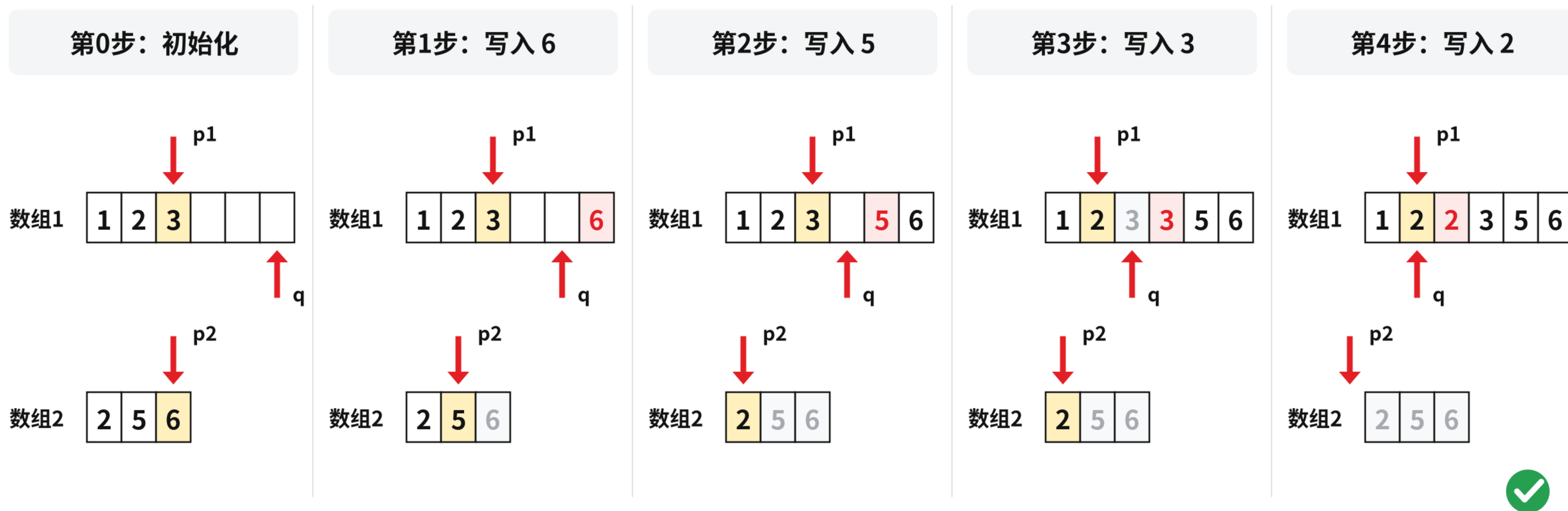
4. 合并两个有序数组

```
void merge1(int *nums1, int m, int *nums2, int n) {
    int temp[m + n];
    int *p1 = nums1, *p1e = nums1 + m;
    int *p2 = nums2, *p2e = nums2 + n;
    int *q = temp;
    // 1. 比较两个数组的当前元素，写入较小值
    while (p1 < p1e && p2 < p2e) {
        if (*p1 <= *p2) *q++ = *p1++;
        else *q++ = *p2++;
    }
    // 2. 将剩余元素依次写入临时数组
    while (p1 < p1e) *q++ = *p1++;
    while (p2 < p2e) *q++ = *p2++;

    // 3. 将临时数组复制回 nums1
    for (int *p = temp; p < q; p++) *nums1++ = *p;
}
```

4. 合并两个有序数组

方法二：利用nums1的尾部空间，总将最大的数从后往前写入nums1，直到数组2全部写入



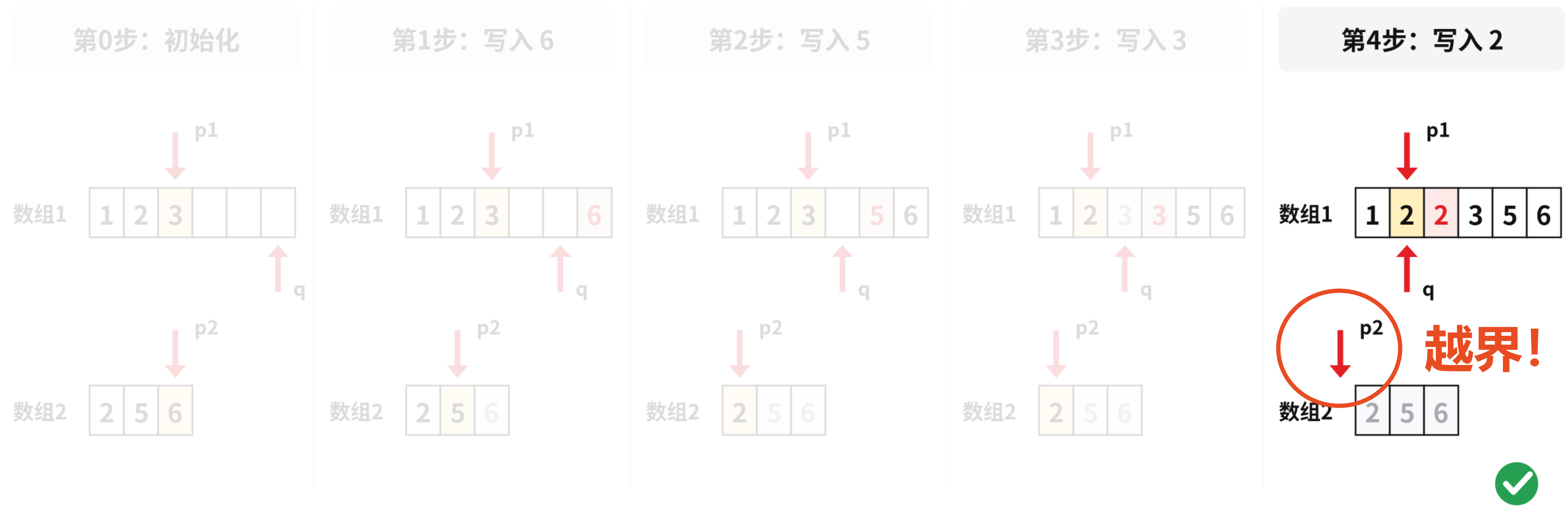
4. 合并两个有序数组

```
void merge2(int *a1, int m, int *a2, int n) {  
    int *p1 = a1 + m - 1;  
    int *p2 = a2 + n - 1;  
    int *q = a1 + m + n - 1;  
  
    while (p2 >= a2) {  
        if (*p1 >= *p2) {  
            *q = *p1;  
            q--;  
            p1--;  
        }  
        else {  
            *q = *p2;  
            q--;  
            p2--;  
        }  
    }  
}
```

```
void merge2(int *a1, int m, int *a2, int n) {  
    int *p1 = a1 + m - 1;  
    int *p2 = a2 + n - 1;  
    int *q = a1 + m + n - 1;  
  
    while (p2 >= a2) {  
        if (p1 >= a1 && *p1 >= *p2) {  
            *q = *p1;  
            q--;  
            p1--;  
        }  
        else {  
            *q = *p2;  
            q--;  
            p2--;  
        }  
    }  
}
```

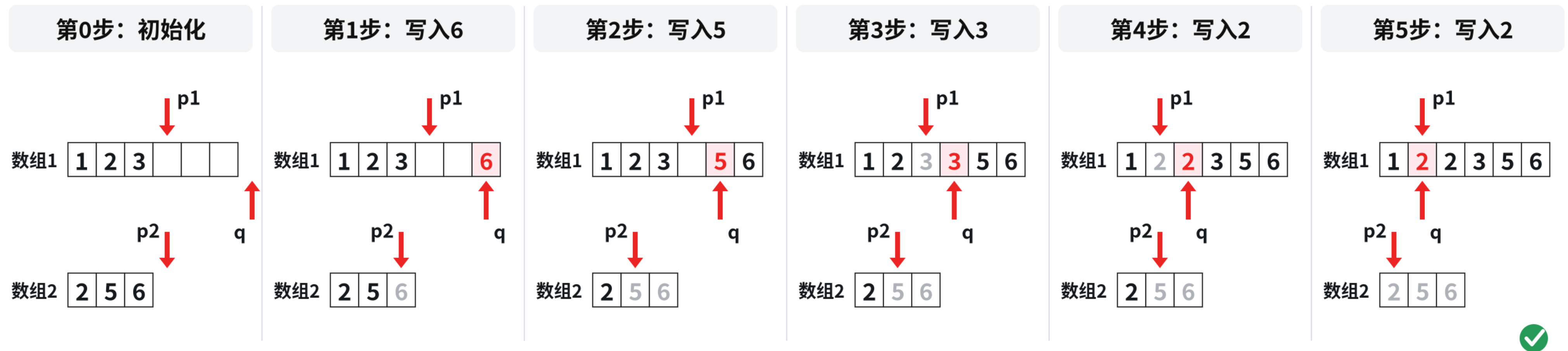
代码还有什么隐患？

4. 合并两个有序数组



4. 合并两个有序数组

方法二：利用nums1的尾部空间，总将最大的数从后往前写入nums1，直到数组2全部写入（指针指向要处理位置的**后一个位置**）



6. 矩阵置零

题目描述

编译器: C (gcc) [编辑](#)

给定一个 `rows` 行、`cols` 列的整数矩阵。如果矩阵中某个元素原本为 0，就将该元素所在的整行和整列全部置为 0。所有置零操作均应根据修改前的矩阵确定，修改过程中产生的新 0 不会触发新的置零操作。

函数接口定义

[c]		复制内容 格式 全屏	
1	void set_zeroes(int rows, int cols, int matrix[rows][cols]);		

- `rows`: 矩阵行数;
- `cols`: 矩阵列数;
- `matrix`: 待处理的二维整型数组，须在原数组中修改其元素。

两个有序序列的多重交集

题目描述

[编辑](#)

给定两个按非递减顺序排列的整数序列 **A** 和 **B**，求它们的多重交集，即交集中保留重复元素。对于任意整数 **x**，它在交集中出现的次数，等于它在 **A** 和 **B** 中出现次数的较小值。交集仍按非递减顺序输出。

例如 **A = [1,1,2,3,5,5]**，**B = [1,2,2,5,5,5]**，结果是 **[1,2,5,5]**。

输入格式

- 第一行输入两个整数 **n m** ($0 \leq n, m \leq 3000$)，分别表示序列 **A** 和 **B** 的长度；
- 第二行输入 **n** 个整数，表示序列 **A**；当 **n = 0** 时，此行为空；
- 第三行输入 **m** 个整数，表示序列 **B**；当 **m = 0** 时，此行为空；
- 所有元素均在 $[-10^6, 10^6]$ 范围内，两个输入序列均已按非递减顺序排列。

输出格式

第一行输出交集的元素个数 **t**。若 **t > 0**，第二行输出这 **t** 个整数，整数间用一个空格分隔，行尾无空格；若 **t = 0**，则仅输出一行 **0**。

纸上得来终觉浅