# 数组

与list集合的互换

## 对于已经排好序的：

1可以使用两个指针一前一后进行遍历

2 二分查找

## 对于没有排序的：

1 可以先进行排序，再进行操作

2 如果不排序

一般设置变量，用于存储每个阶段的值

使用map或者set集合（对于map，键值分别对应元素值和下标或者出现次数）

## 递推的思想：

前面的操作会影响后面的操作，这时要找规律或者一个递推公式。

比如爬楼梯，dp[n]=dp[n-1]+dp[n-2]；格雷码，n位格雷码可以由n-1位格雷码转换而成。

## 左右两个因素

如果结果是由左右两个因素进行决定，不是单方向的决定，可以设两个数组，一个从前向后遍历时存储值，一个从后向前遍历时存储值。

## 异或

使用异或

交换律：A XOR B = B XOR A

结合律: A XOR B XOR C = A XOR (B XOR C) = (A XOR B) XOR C

自反性： A XOR B XOR B = A XOR 0 = A

只能处理偶数次出现的情况，可以用来判断哪些数只出现奇数次。

# 单链表

单链表只能从前往后遍历，如果不符合，可以先反转或者使用栈。

反转：

ListNode Reverse(ListNode head){

ListNode pre = null;

While(head!=null) {

ListNode tmp = head.next;

Head.next = pre;

Pre = head;

Head = tmp;

}

Return pre;

}

使用俩指针fast和slow或者使用hashmap计数

# 字符串

技巧1 前部分翻转 后部分翻转 整体翻转

技巧2 kmp算法 contains方法

# 链表

链表的题目一般都是节省空间，加强代码实现

涉及到逆序操作，可以使用栈实现（比如判断是否为回文）

判断两个链表是否相交 是否有环 以及找到第一次相交点 环的入口点

/\*

\* 判断链表是否有环

\* 1 如果一个有环一个无环，则肯定不相较

\* 2 如果都无环，则遍历到尾部，看看最后一个节点是否相等即可

\* 如果要找到第一个相交的点，则让长的链表先走len1-len2步，再同时走，找到相等的节点即可

\* 3 如果都有环，则先找到各自的环入口点(快慢指针，第一次相遇后，快指针从头节点开始，每次走一步，慢指针继续向前走一步，直至再次相遇）

\* 如果两个入口点相等，则肯定相交

\* 如果要找到第一个相交的点，则需要让长的链表（从头节点到相交的节点）先走len1-len2步，然后同时走，找到相等的点即可

\* 如果两个入口点不相等，则选其中一个相交点，继续向后走，看看 是否会经历另外一个相交点，如果经历了，则相交，否则不相交

\*/

这是一道比较综合的题目。将链表中关于环和相交的问题聚集在一起了，关键是要思路清晰。