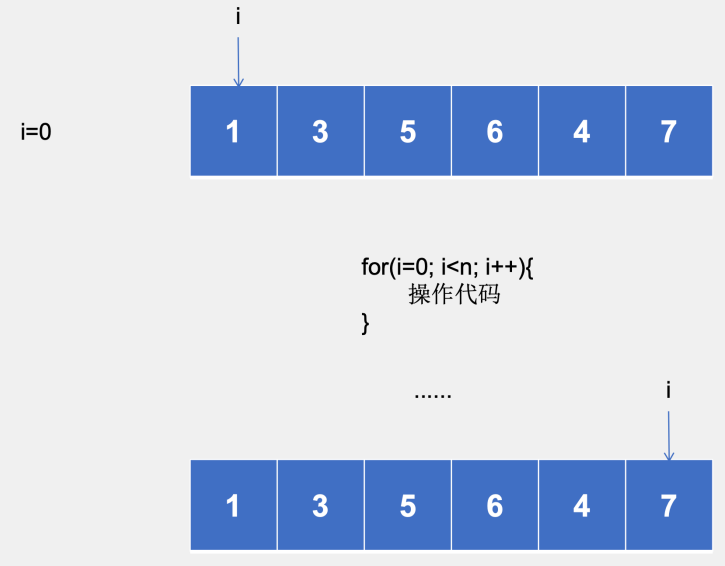
强化_算法基础

算法题1

现有一个整数数组nums，其中有n个均不相等的整数，要求设计一个算法可以判断该数组是否为单调数组。

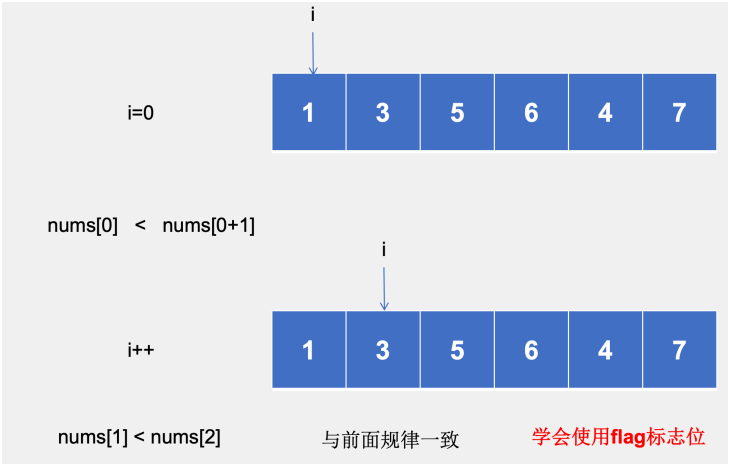
```
// 例:
nums=[1,2,4,9] n=4
// 输出:
true

nums=[1,3,5,4,9,10] n=6
// 输出:
false
```

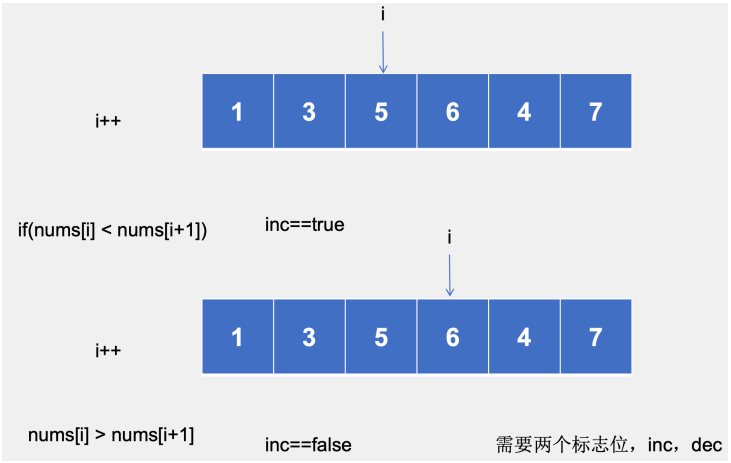
基本算法思想：遍历数组，判断数组中任意元素与其后面元素的大小关系。
完整代码=基本算法思想扩写+逐字翻译(画图帮助理清细节)

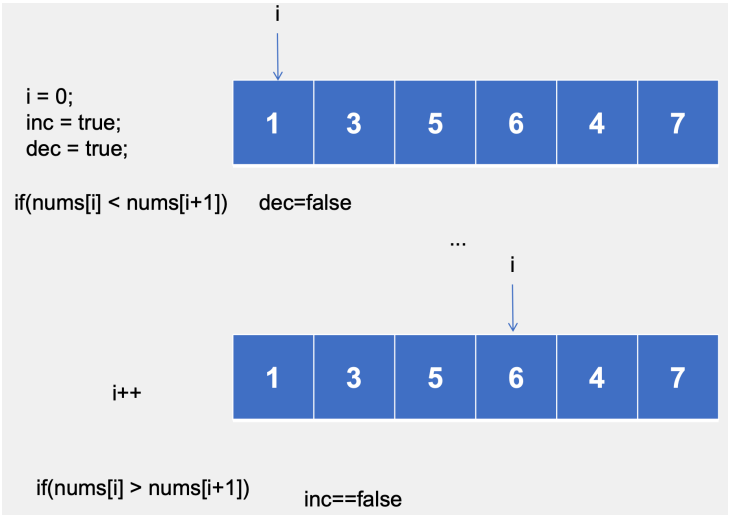



```
for(i=0; i<n; i++){ // 判断是否为单调递减数组
    if(nums[i] < nums[i+1]){
        // 不是一个单调递减数组
    }
}
```

```
for(i=0; i<n; i++){ // 判断是否为单调递增数组
    if(nums[i] > nums[i+1]){
        // 不是一个单调递增数组
    }
}
```







基本算法思想：遍历数组，判断数组中任意元素与其后面元素的大小关系。

算法思想：遍历数组，判断任意当前元素与其后紧邻元素的大小关系是否与其他元素之间一致。设

置两个标志位来标记递增或递减，当前元素小于后一个元素时就认为是递增，反之认为是递减，遍历一遍数组后根据标志位即可得出结果。

```
bool isMonotonic(int* nums, int numsSize) {
    bool inc = true, dec = true;
    for(int i=0;i<numsSize-1;i++) {
        if(nums[i]<nums[i+1]){// 递增
            dec = false;
        }else if(nums[i]>nums[i+1]){// 递减
            inc = false;
        }
    }
    return dec || inc;
}
```

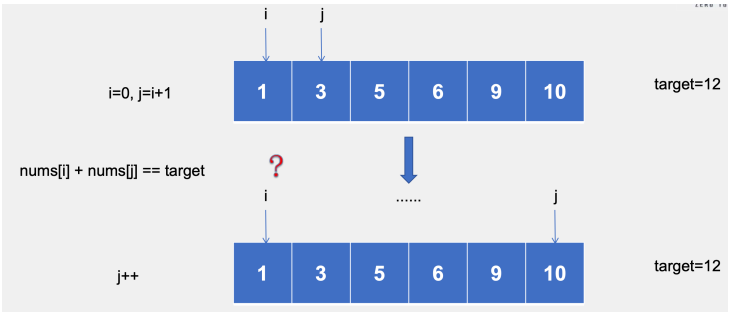
算法题2

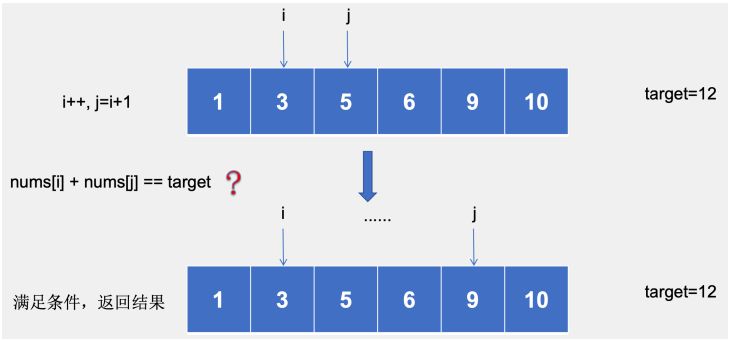
现有一个整数数组nums和一个目标值target，请你在该数组中找出两个数字，其和正好等于目标值，最后返回这两个数字的数组下标。假设一定存在满足该条件的两个数字 且答案是唯一的。

```
// 例:
nums=[3,2,6,9], target=11
// 输出:
[1,3]

nums=[1,3,5,7,9,10], target=11
// 输出:
[0, 5]
```

基本算法思想：遍历数组，判断数组中任意两个元素之和是否为target。

```
// 双指针法
for(i=0; i<边界; i++){
    for(j=初始值; j<边界; j++){
        // 核心代码
    }
}
```





基本算法思想：遍历数组，判断数组中任意两个元素之和是否为target.

算法思想：暴力法即可解出此题，遍历数组，判断数组中任意两个元素之和是否为target。使用两个工作指针i和j，当固定住nums[i]时，j从i+1位置出发去遍历剩余的表 寻找值为target-nums[i]的元素，找到了符合条件的元素之后将其下标返回。

```
int* twoSum(int* nums, int numsSize, int target, int* returnSize) {
    int i, j;
    int *result=(int*)malloc(sizeof(int)*2);    // 返回结果
    for(i=0; i<numsSize; i++) {
        for(j=i+1; j<numsSize; j++) {
            if(nums[i]+nums[j]==target){    // 满足条件
                result[0]=i;
                result[1]=j;
                *returnSize=2;
                return result;
            }
        }
    }
    return result;
}
```

暴力解法模板（链表同理）

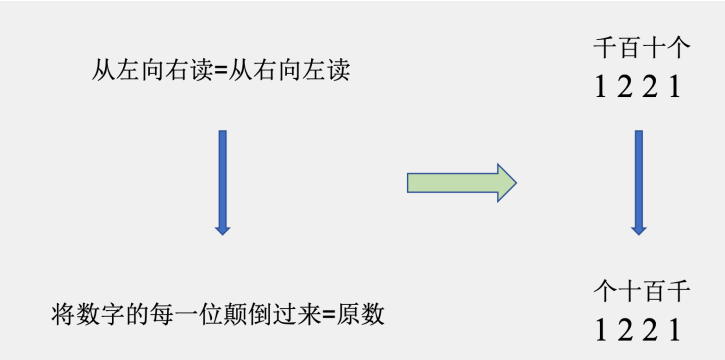
```
for(i=0; i<边界; i++){
    for(j=初始值; j<边界; j++){
        // 核心代码
    }
}
```

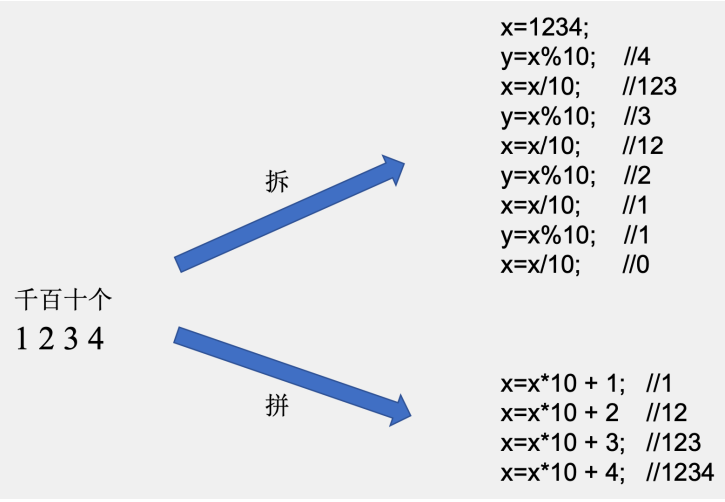
算法题3

现有一个整数x，如果从左向右看和从右向左看是一模一样的整数，我们就称该数字为回文数。

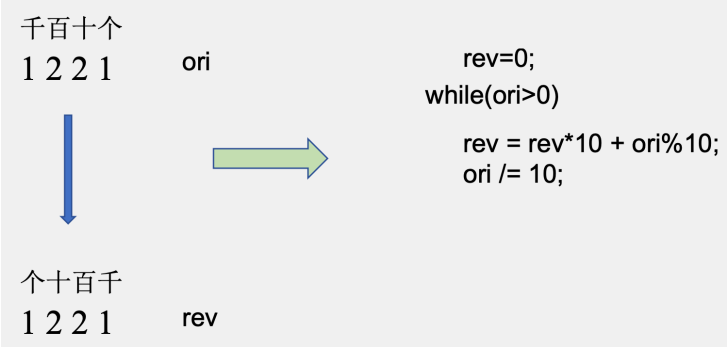
```
// 例:
x = 616
// 输出:
true

x = -11
// 输出:
false
```

基本算法思想：逆置数字后与原数相比是否一样。





如何考虑边界条件?



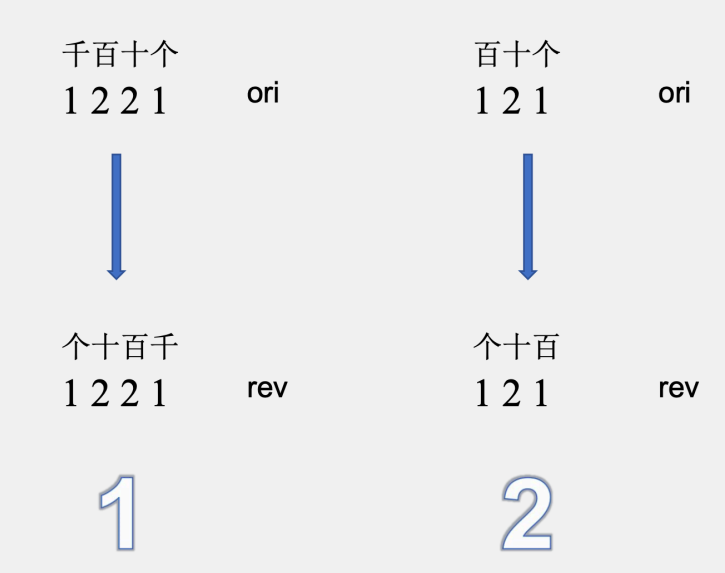
基本算法思想：逆置数字后与原数相比是否一样。

算法思想：题干里面已经说了“回文数字就是从左向右看和从右向左看一样的数字”，含义其实就是把它的个十百千位颠倒过来和以前一模一样。所以只需要使用一个变量来存放这个颠倒后的数字，然后再与原数字比大小即可，这里需要注意的是，如果一个很大的非回文数颠倒过来可能会导致int类型溢出，所以要用long来存储颠倒后的数字。例子已经暗示输入有负数了，记得先进行边界条件的判断。

```
bool isPalindrome(int x) {
    if(x<0) {   // 判断边界条件
        return false;
    }
    long elem=0;
    int ori=x;
    while(x>0) {
        elem=elem*10+x%10;
        x=x/10;
    }
    return ori==(int)elem;
}
```

优化算法

1. 偶数位的回文数字颠倒过来一半的数字即和剩余的数字一样了；
2. 奇数位的回文数字正中间那一位数字无所谓是几都不会影响其回文的性质；



优化算法思想：对于偶数位数字只颠倒一半的数字即可与原数字剩余部分进行比较；对于奇数位数字只需要颠倒一半的数字后去掉最后一位再与原数字剩余部分比较即可。例子已经暗示输入有负数了，记得先进行边界条件的判断。

```
bool isPalindrome(int x) {
    // 判断边界条件：除奇偶数外，还需要考虑什么特殊情况？如1010。
    if(x<0||x%10==0&&x!=0) {
        return false;
    }
    int num=0;
    while(x>num) {
        num=num*10+x%10;
        x=x/10;
    }
    return x==num||x==num/10;
}
```

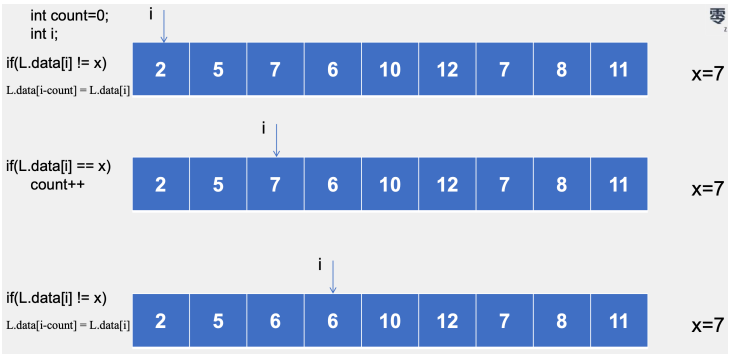
算法题4

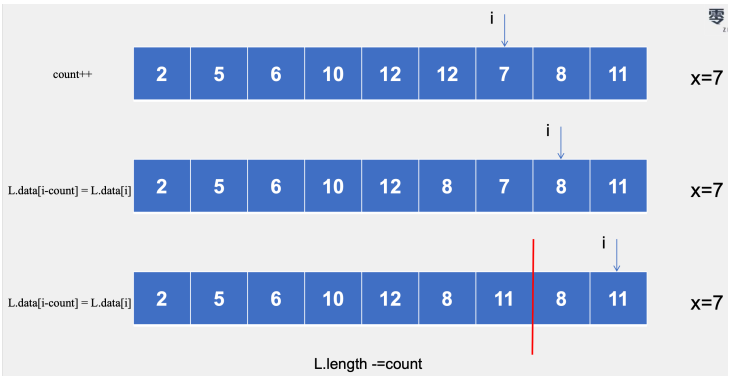
现有一个长度为n(n<100)的顺序表L，要求删除表中全部值为x的元素，返回删除后 的顺序表。顺序表的MaxSize为100。

1. 写出顺序表的数据结构定义。
2. 用c或c++实现代码，关键语句给出注释。
3. 写出算法的时间复杂度和空间复杂度。

```
// 例:
L=[3,5,6,9,2,3,10,3,11], x=3
// 输出:
[5,6,9,2,10,11]

L=[2,5,6,9,11,5,30,45,66], x=5
// 输出:
[2,6,9,11,30,45,66]
```

基本算法思想：遍历数组，将所有值为x的元素删除。





基本算法思想：遍历数组，将所有值为x的元素删除。

算法思想：经过观察可以发现，最终序列与原始序列相比，每一个值不等于x的元素前移的位置都等于它前面值为x的元素个数，所以只需要一个计数变量count来记录当前遍历到的x的个数，每一个不

等于x的元素直接前移count个位置即最终位置。这是本题的最佳做法，时间复杂度和空间复杂度都是最优。

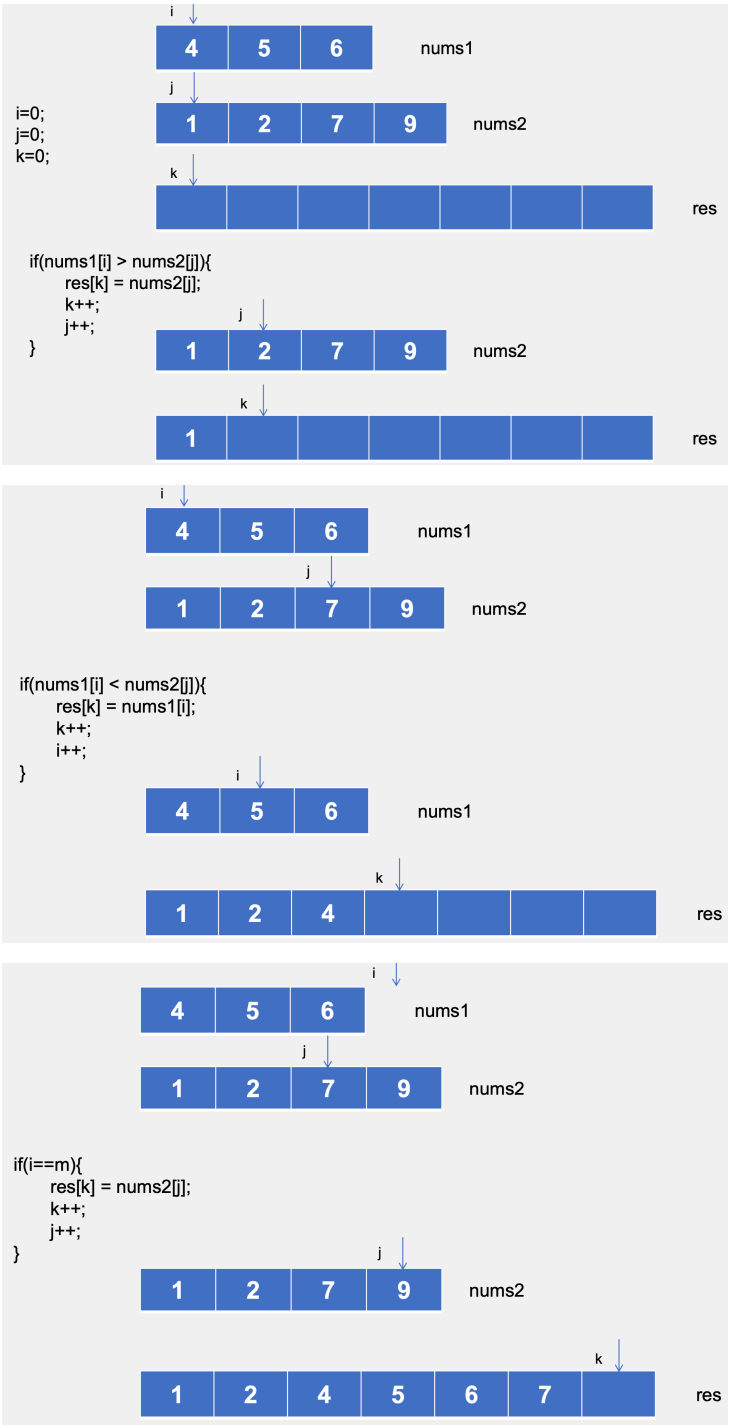
```
#define MaxSize 100

typedef struct {
    int data[MaxSize];
    int length;
}SqList;

SqList removeElement(SqList& L, int n, int x){
    int count=0;
    int i;
    for(i=0; i<n; i++) {
        if(L.data[i]==x) {
            count++;
        } else {
            L.data[i-count]=L.data[i];  // 直接放到最终位置
        }
    }
    L.length-=count;
    return L;
}
```

算法题5

现有两个非递减数组nums1和nums2，其中nums1有m个元素，nums2有n个元 素，要求在不改变大小关系的情况下将两个数组合并为一个数组。

```
// 例:
nums1=[3,5,6,9], m=4, nums2=[1,3,5,7], n=4
// 输出:
[1,3,3,5,5,6,7,9]

nums1=[11,30,45], m=3, nums2=[5, 10] n=2
// 输出:
[5,10,11,30,45]
```

基本算法思想：分别遍历两个数组，将较小的元素依次写入结果数组。



基本算法思想：分别遍历两个数组，将较小的元素依次写入结果数组。

算法思想：申请一个大小为m+n的结果数组，从前向后遍历nums1和nums2进行比较，将其中较小的元素写入结果数组，当一个数组比完后将另一个数组的值直接写入结果数组即可。

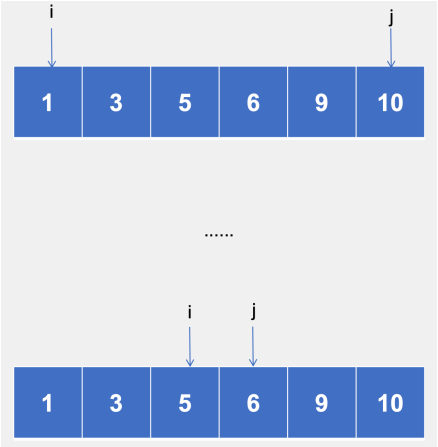
```
int* merge(int* nums1, int m, int* nums2, int n) {
    int *res=malloc(sizeof(int*)*(m+n));
    int i=0, j=0, k=0;
    int cur;
    while(i<m || j<n) {
        if(i==m) {  // nums1遍历完成
            cur=nums2[j++];
        }else if(j==n) {    // nums2遍历完成
        cur=nums1[i++];
        }else if(nums1[i]<nums2[j]) {   // nums1的元素更小
            cur=nums1[i++];
        }else { // nums2的元素更小
            cur=nums2[j++];
        }
        res[k++]=cur;
    }
    return res;
}
```

算法题6

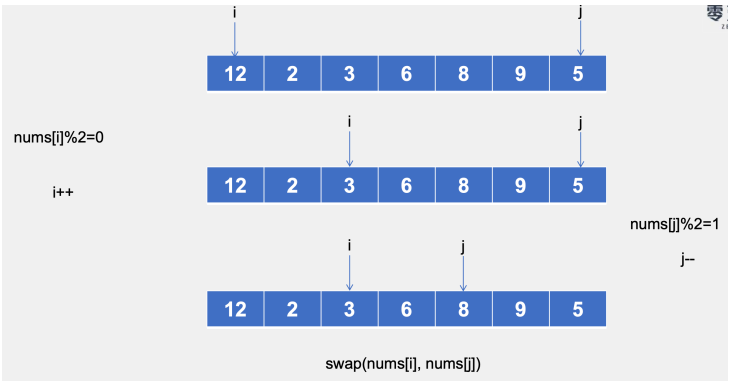
现有数组nums，一共有numSize个元素，要求你设计一个算法使其中的偶数元 素都在数组的前半部分，奇数都在数组的后半部分，彼此之间的顺序没有要求。

```
// 例:
nums=[5,2,3,6,8,9,4]
// 输出:
[4,2,8,6,3,9,5]

nums1=[0]
// 输出:
[0]
```

基本算法思想：遍历数组，将数组后半部分的偶数放到前面，将数组前半部分的奇数放到后面。




```
i=0; j=n-1;
while(i<j){
    操作代码;
    看情况i++;
    看情况j--;
}
```



基本算法思想：遍历数组，将数组后半部分的偶数放到前面，将数组前半部分的奇数放到后面。

算法思想：利用**双指针的思想**，偶数指针从前向后遍历数组，奇数指针从后向前遍历，当偶数指针遍历到奇数，奇数指针遍历到偶数时，两者交换值即可，当两个指针相遇的时候遍历完成。

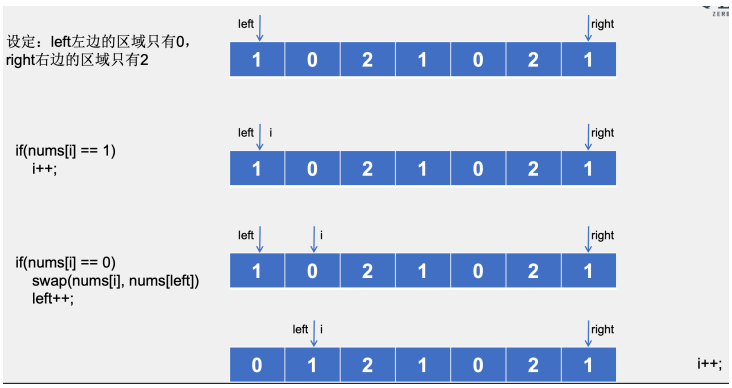
```
int* sortArrayByParity(int* nums, int numsSize) {
    if(numsSize==1) return nums;    // 边界条件
    int even=0, odd=numsSize-1;
    while(even<odd) {
        while(even<odd && nums[even]%2==0) even++;  // 偶数
        while(even<odd && nums[odd]%2==0) odd--;    // 奇数
        if(even<odd) {  // 互换
            int temp=nums[even];
            nums[even++]=nums[odd];
            nums[odd--]=temp;
        }
    }
    return nums;
}
```

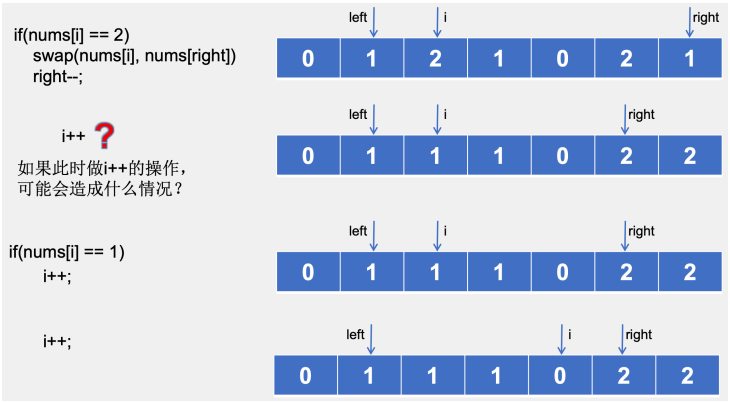
算法题7

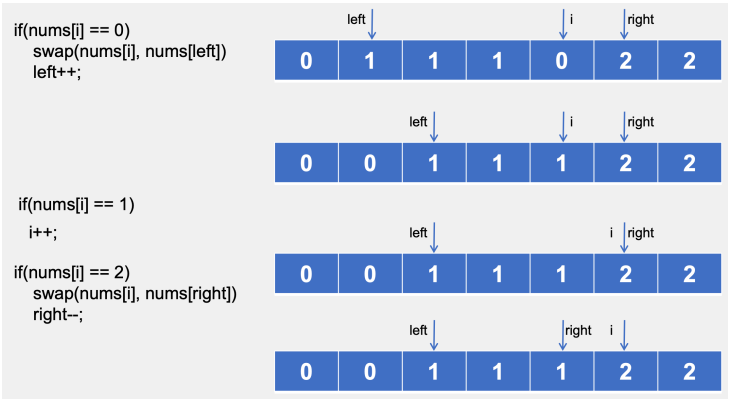
现有一个数组nums，一共有n个元素，数组内只包含0，1，2三种元素，现在要求设计一个算法将这三种元素进行排序，最终使得0都位于数组的前部，2都位于数组的后部，1都位于数组中间位置。

```
// 例:
nums=[0,1,2,2,1,0], n=6
// 输出:
[0,0,1,1,2,2]

nums=[2,1,0], n=3
// 输出:
[0,1,2]
```

算法思想：遍历数组，将位于数组后半部分的0交换到数组前面，将位于数 组前半部分的2交换到数组后面。







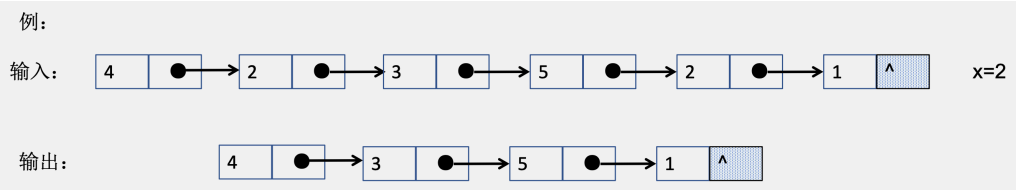
算法思想：遍历数组，将位于数组后半部分的0交换到数组前面，将位于数组前半部分的2交换到数组后面。

算法思想：本题使用双指针法，一个工作指针left放在表首，作为数字0的边界；一个工作指针right放在表尾，作为数字2的边界。在从前向后遍历的过程中，只要遇到数字0就将其与边界left交换，只要遇到数字2就将其与边界right进行交换，遍历结束即为规定顺序。

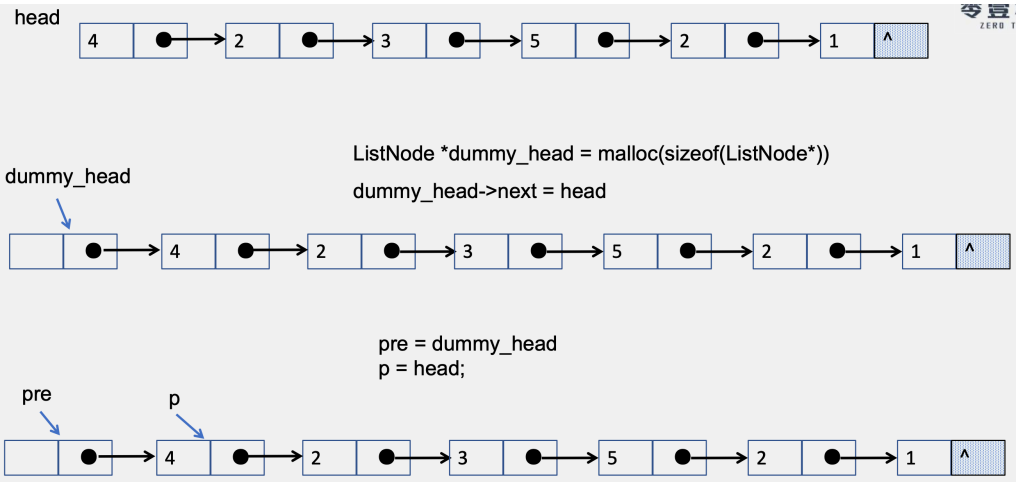
```
void sortColors(int* nums, int numsSize) {
    int left=0, right=numsSize-1;   // left代表红色，right代表蓝色
    for(int i=0; i<=right; i++) {
        if(nums[i]==0) {
            swap(nums, left, i);
            left++;
        }
        if(nums[i]==2) {
            swap(nums, i, right);
            right--;
            i--;    // 防止漏算
        }
    }
}
void swap(int* nums, int i, int j) {
    int temp=nums[i];
    nums[i]=nums[j];
    nums[j]=temp;
}
```

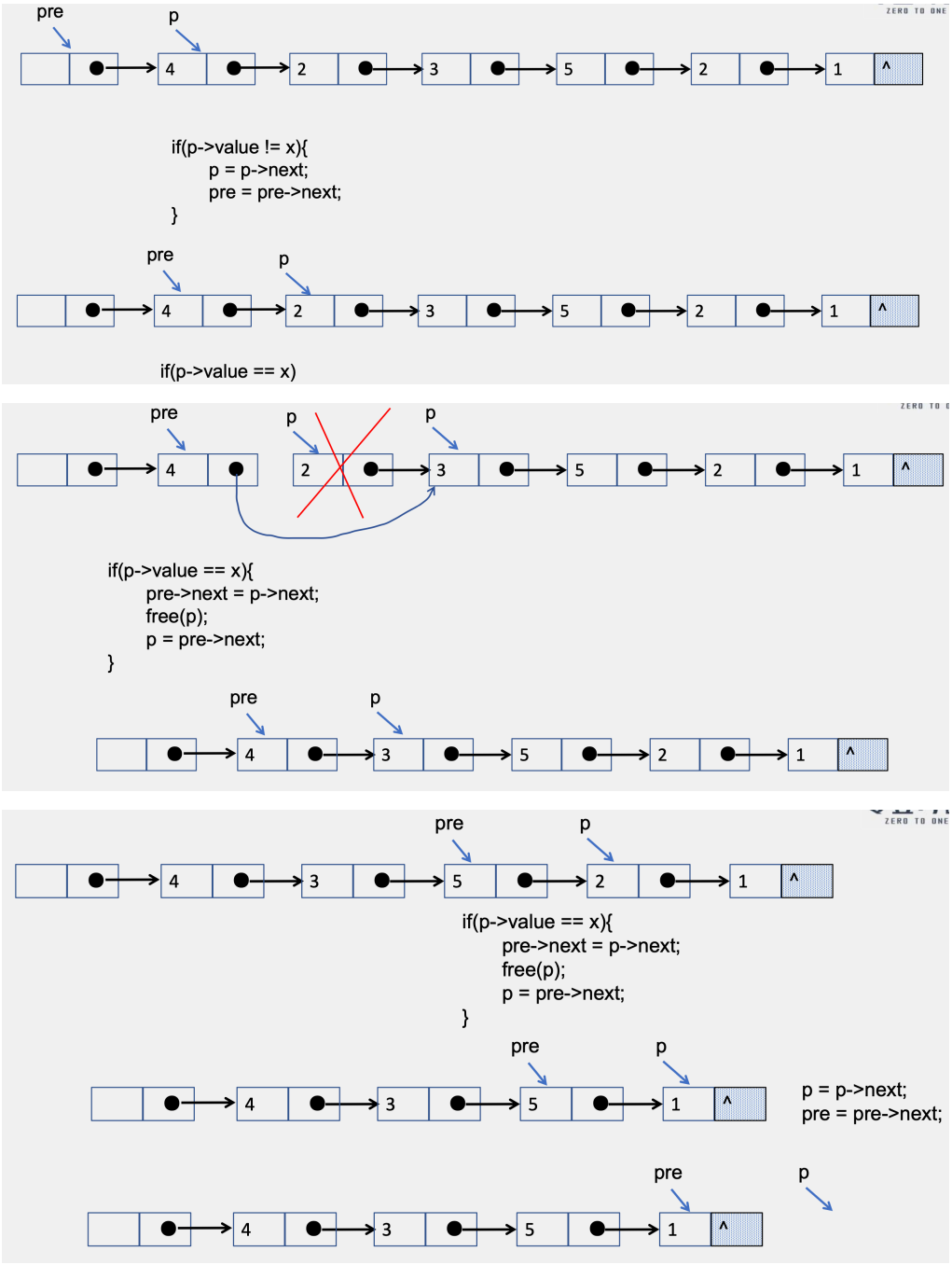
算法题8

现有一个不带头结点的单链表head，要求设计一个算法删除其中值为x的结点。



基本算法思想：遍历链表，删除其中值为x的结点。





基本算法思想：遍历链表，删除其中值为x的结点。

算法思想：顺序遍历链表，删除其中值为x的结点。为了防止删除时断链，额外声明一个pre指针记录待删除结点的前驱结点。

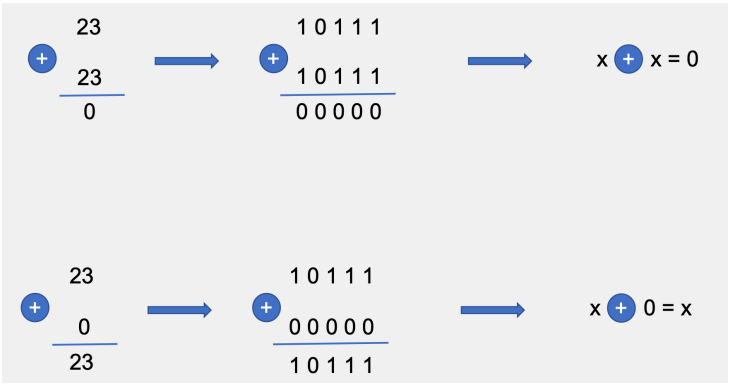
```
// Definition for singly-linked list.
typedef struct {
    int val;
    struct ListNode *next;
}ListNode;
ListNode* delete(ListNode* head, int x) {
    ListNode *dummy_head=malloc(sizeof(ListNode*)); // 虚头节点
    ListNode *p=head;
    ListNode *pre=dummy_head;
    while(p) {
        if(p->value==x) {   // 删除x
            pre->next=p->next;
            free(p);
            p=pre->next;
        }else { // 向后遍历
            p=p->next;
            pre=pre->next;
        }
    }
    return dummy_head->next;
}
```

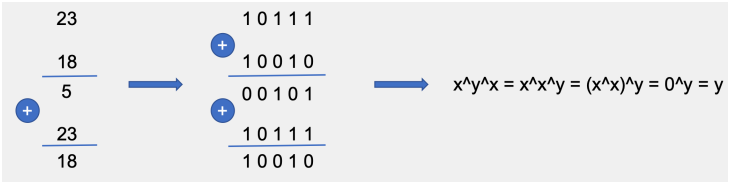
算法题9

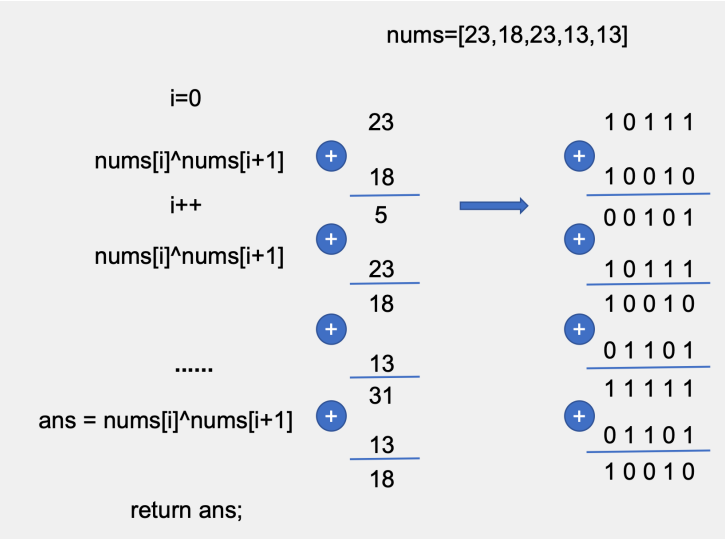
现有一个整数数组nums，如果只有一个数字出现一次，其余的数字都出现两次，返回那个只出现了一次的元素。一共有多少种解法?

```
// 例:
nums=[6,5,5,6,3]
// 输出:
3

nums=[11,11,45]
// 输出:
45
```

```c
int singleNumber(int* nums, int numsSize) {
    int i, j;
    for(i=0; i<numsSize-1; i++) {   // 冒泡排序
        bool flag=false;
        for(j=numsSize-1; j>i; j--) {
            if(nums[j-1]>nums[j]) {
                swap(nums, j-1, j);
                flag=true;
            }
        }
        if(flag==false) break;
    }
    for(i=0; i<numsSize-1; i+=2) {  // 查找
        if(nums[i]!=nums[i+1]) {
            return nums[i];
        }
    }
    return nums[numsSize-1];
}
void swap(int* nums, int i, int j) {    // 对换函数
    int temp=nums[i];
    nums[i]=nums[j];
    nums[j]=temp;
}
```








```
int singleNumber(int* nums, int numsSize) {
    int ans=nums[0];
    for(int i=1; i<numsSize; i++) {
        ans=ans^nums[i];
    }
    return ans;
}
```

这里要注意两个异或小知识：

1. 相同的数异或得0；
2. 0与任何数异或得任何数。