

# 1. 位运算

异或  $\wedge$ ，可理解为不进为相加，满足结合律和交换律

$a \wedge a = 0$ ;

$a \wedge 0 = a$ ;

交换a和b

$a = a \wedge b$ ;

$b = a \wedge b$ ;

$a = a \wedge b$ ;

但必须满足是位置不同的（同一内存位置自己异或结果为0）；

```
public static void swap(int[] arr, int a, int b){
    if(a == b){
        return;
    }
    arr[a] = arr[a] ^ arr[b];
    arr[b] = arr[a] ^ arr[b];
    arr[a] = arr[a] ^ arr[b];
}
```

题目：

1. 数组中一种数出现了奇数次，其他的出现了偶数次，找出出现奇数次的这个数（时间复杂度 $O(N)$   
空间复杂度： $O(1)$ ）

思路：出现偶数次的数自己和自己异或后为0，异或满足结合律和交换律，所有元素异或即可得到结果

```
public class main{
    public static int process(int[] arr){
        int res = 0;
        for(int i : arr){
            res ^= i;
        }
        return res;
    }
}
```

2. 数组中两种数出现了奇数次，其他的出现了偶数次，找出出现奇数次的这个数（时间复杂度 $O(N)$   
空间复杂度： $O(1)$ ）

思路：设a和b是出现奇数次的两个数， $a \neq b$ ，即a和b必有一位数不一样，所有元素异或之后的结果就是 $res = a \wedge b \neq 0$ ，当res有一位为1时，则说明这一位a和b不相等，则可将原数组分为两类，该位为1和为0的，将其中一类的全部元素进行异或，可得出其中一个奇数次的数记作res'，再将其与res进行异或，得出另一个数。

补充：提取出一个数最右边的1的位置

```
int rightOne = res & (~res + 1);
```

举例：res: 1 0 1 0 1 1 1 1 0 0

```
~res:  0 1 0 1 0 0 0 0 1 1
~res+1: 0 1 0 1 0 0 0 1 0 0
res & (~res+1) = 0 0 0 0 0 0 1 0 0
```

结果:

```
public class Main{
    public static void process(int[] arr){
        int res = 0;
        for(int curNum : arr){
            res ^= curNum;
        } //得到a^b
        int rightOne = res & (~res + 1); //提取出最右边的1
        int onlyOne = 0; //准备得出res'
        for(int cur : arr){
            if((cur & rightOne) == 1){ //为0为1都可
                onlyOne ^= cur;
            } //得出res'
        }
        System.out.println(onlyOne + " " + (onlyOne ^ res));
    }
}
```

## 2. 排序

### 选择排序 (时间复杂度 $O(N^2)$ 空间复杂度 $O(1)$ )

每次只交换一次

- 遍历数组
- 每次遍历都要遍历数组没有 sorted 部分找到当前最大 / 最小元素放到正确的位置上

```
public static void selectionSort(int[] arr){
    if (arr == null || arr.length < 2){
        return;
    } //排除异常情况
    for (int i = 0; i < arr.length-1; i++){ //i ~ N-1
        int minIndex = i;
        for(int j = i + 1; j < arr.length; j++){ //i ~ N-1 上找最小值的下标
            minIndex = arr[j] < arr[minIndex] ? j : minIndex;
        }
        swap(arr, i, minIndex);
    }
}

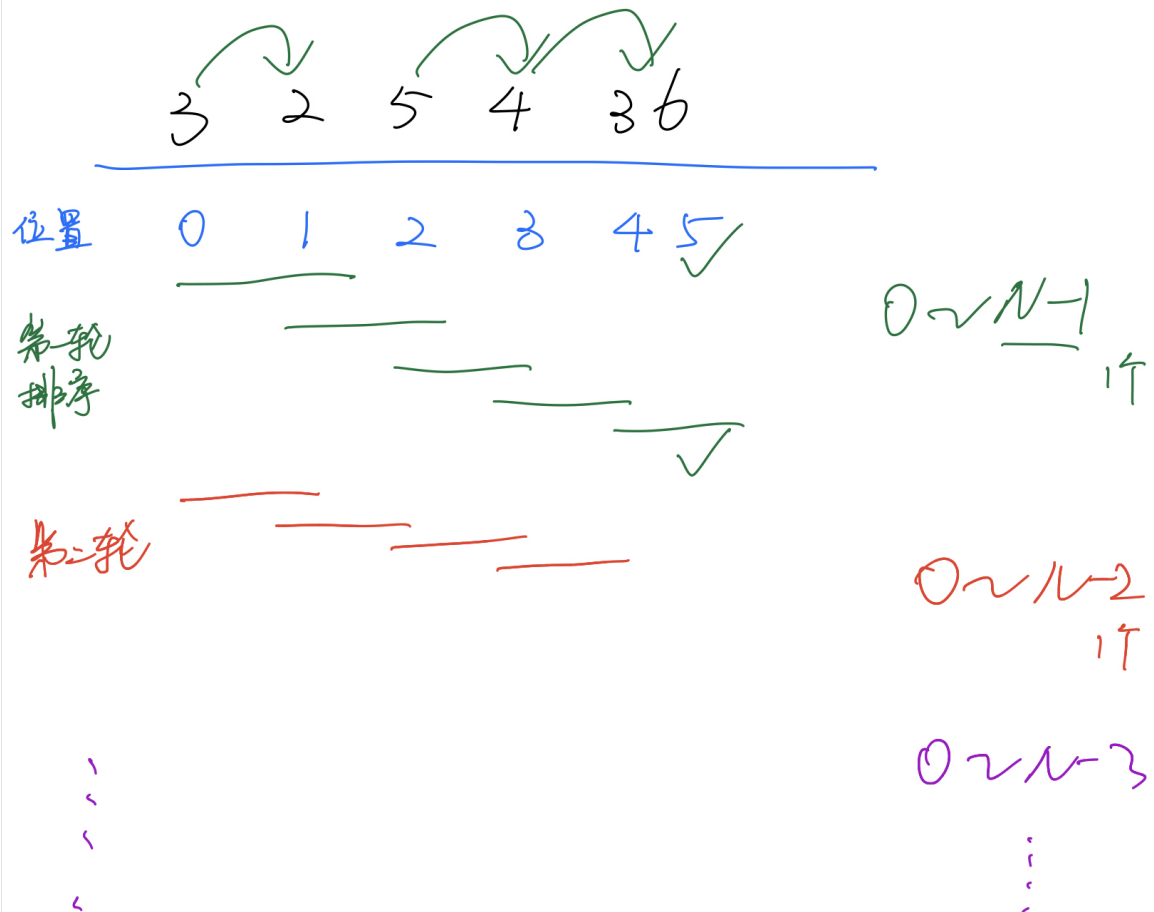
//交换操作
public static void swap(int[] arr, int i, int j){
    int tmp = arr[i];
    arr[i] = arr[j];
    arr[j] = tmp;
}
```

## 冒泡排序 (时间复杂度 $O(N^2)$ 空间复杂度 $O(1)$ )

相邻两个数比较, 交换, 最大的往右排, 相对小的往左排

- 遍历数组
- 每次遍历让所有的元素 (除了最后) 跟下一个元素比, 如果出现 inversion 就换过来
- 每次 `outerloop` 遍历就会让当前没有 sorted 部分找出一个最大元素放到正确的位置上 (被换到那)

举例子:



CSDN @weixin\_45377141

```
public static void bubbleSort(int[] arr){
    if(arr == null || arr.length < 2){
        return;
    }
    for (int e = arr.length - 1; e > 0 ; e--){ //0~e
        for(int i = 0; i < e; i++){
            if(arr[i] > arr[i + 1]){
                swap(arr, i , i+1);
            }
        }
    }
}
```

## 插入排序 ( $O(N^2)$ )

给定一个数组,做到0~0, 0~1, 0~2, ..., 0~N-1位置有序, 实现整体排序

- 遍历数组
- 首先做到 0-0 有序, 这个很明显本身就做到了
- 接着我们要做到 0-1 有序, 所以先看在 1 号元素
  - 如果比前一个小就交换, 交换完了再看前面有没有要交换的了, (前面没数了, 也挺了), 我们此时做到了 0-1 有序
  - 如果比前一个大, 不需要交换
- 接着我们要做到 0-2 有序, 所以先看在 2 号元素
  - 如果比前一个小就交换, 交换完了再看前面有没有要交换的了, 一直换到对的地方
  - 如果比前一个大, 不需要交换
- 接着我们要做到 0-3 有序, 所以先看在 3 号元素
  - 如果比前一个小就交换, 交换完了再看前面有没有要交换的了, 一直换到对的地方
  - 如果比前一个大, 不需要交换

... 最终我们做到了 `0-nums.length-1` 有序了

代码:

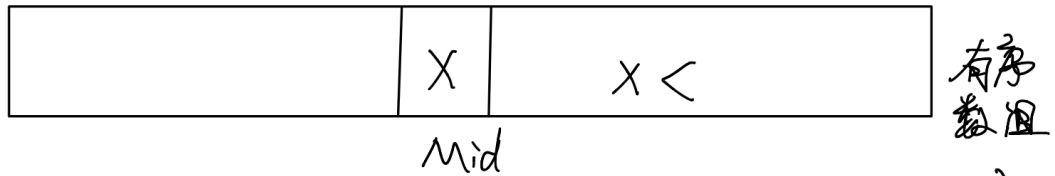
```
public static void insertionSort(int[] arr){
    if (arr == null || arr.length < 2){
        return;
    }
    //0~0有序
    //0~i想有序
    for (int i = 1; i < arr.length; i++){
        for (int j = i - 1; j >= 0 && arr[j] > arr[j + 1]; j--){
            swap ( arr, j, j+1);
        } //从后往前看, 当前数往左换到不能换, 停止
    }
}
```

## 3. 二分法

问题一: 如何在一个有序数组中找某个是否存在?

思路: 遍历, 时间复杂度 $O(N)$ , 二分法

经典二分:



num?

二分法: (1) 首先找出数组中间的数 X  
比较 X 与 num 大小

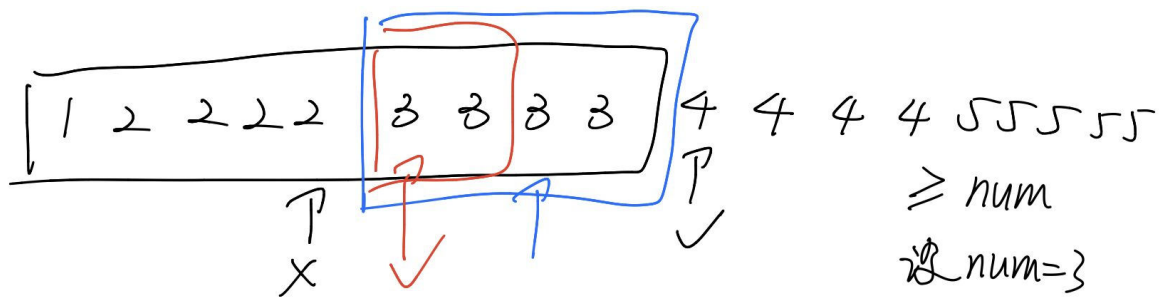
若  $X > num$ , 则 X 右边无 num, 往 X 左边找  
 $X < num$ , 则往 X 右边找  
 $X = num$ , 返回

$O(\log N)$

CSDN @weixin\_45377141

```
function BSearch(arr, num) {
    let L = 0;
    let R = arr.length - 1;
    let mid = 0;
    while(L <= R) {
        mid = L + (L - R) / 2;
        if (arr[mid] == num) {
            return true;
        } else if (arr[mid] < num) {
            L = mid + 1;
        } else {
            R = mid - 1;
        }
    }
    return arr[L] == num;
}
```

问题二: 在有序数组中找大于等于某个数最左侧的位置



① 先找出中点位置，看是否满足条件，满足，则记住该位置， $t$

②  $t$  左侧的中间位置，不满足，则往右侧二分

③ 中间位置满足，记住位置，往左侧继续二分

④ 看该位置数是否满足，继续看 没有数了，停 则找到

CSDN @weixin\_45377141

```
function BSfindLeftMax(arr, num) {
    let L = 0;
    let R = arr.length - 1;
    let mid = 0;
    let leftMaxIndex = -1;
    while (L < R) {
        mid = L + (L - R) / 2;
        if (arr[mid] >= num) {
            leftMaxIndex = mid;
            R = mid - 1;
        } else {
            L = mid + 1;
        }
    }
    return leftMaxIndex;
}
```

问题三：局部最小值，在一个无序数组中，任意两个相邻的元素不相等，定义一个局部最小

局部最小:

$$\begin{array}{cc} a & b \\ \text{位置: } 0 & 1 \end{array}$$

若  $a < b$ , 则 0 位置局部最小

$$\begin{array}{cc} b < a \\ N-2 & N-1 \end{array}$$

则  $N-1$  位置局部最小

$$a > b < c$$

$$i-1 \quad i \quad i+1 \quad \text{则 } i \text{ 位置局部最小}$$

① 先看 0~1 位置是否满足

② 看  $N-2 \sim N-1$  位置是否满足

③ 若不满足即

$$\begin{array}{ccc} \swarrow & \nwarrow & \nwarrow \\ 0 & 1 & N-2 \quad N-1 \end{array}$$

则 0~ $N-1$  位置必存在局部最小

取  $m$  位置, 判断  $m-1 \quad m \quad m+1$

$m-1 < m$ , 则 递归二分

4  
5

CSDN @weixin\_45374111

```
function BSFindMin(arr) {  
    // 无序数组中寻找到任意一个局部最小值  
    if (arr.length < 1 || arr === null) return;  
    let L = 0;  
    let R = arr.length - 1;  
    let mid = 0;  
    if (arr[L] < arr[L + 1]) {  
        return arr[L];  
    } else if (arr[R] < arr[R - 1]) {  
        return arr[R];  
    }  
    while (L <= R) {  
        mid = L + ((R - L) >> 1);  
    }  
}
```

```

        if (arr[mid - 1] > arr[mid] && arr[mid + 1] > arr[mid]) {
            return mid;
        } else if (arr[mid - 1] < arr[mid]) {
            R = mid - 1;
        } else if (arr[mid + 1] < arr[mid]) {
            L = mid + 1;
        }
    }
    return arr[L];
}

```

总结：优化流程的标准1.数据状况 2.问题标准

## 对数器

同一个问题可能有多个方法，例如方法a，是我们想测试的方法，方法b是好实现，但时间复杂度不好的方法。所以采用对数器的方法，使用随机样本产生器，用方法a和b产生 `res1` 和 `res2`，比较结果进行分析调整。

比如：

```

public static void main(String[] args){
    int testTime = 500000; //测试次数
    int maxSize = 100; //最大长度
    int maxValue = 100; //最大值
    boolean succeed = true;
    for (int i = 0; i < testTime; i++){
        int[] arr1 = generateRandomArray(maxSize, maxValue); //产生一个随机数组
        int[] arr2 = copyArray(arr1);
        insertionSort(arr1); //插入排序
        comparator(arr2); //对数器方法排序
        if (!isEqual(arr1, arr2)) { //每个位置的值是否一样
            //打印arr1
            //打印arr2
            succeed = false;
            break;
        }
    }
    System.out.println(succeed? "Nice!" : "No!");
}

```

版权声明：本文为CSDN博主「andy.wang0502」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：[https://blog.csdn.net/weixin\\_45377141/article/details/124713657](https://blog.csdn.net/weixin_45377141/article/details/124713657)