# 王争的算法训练营

## 习题课：前缀后缀统计&位运算

# 前缀后缀统计

344. 反转字符串

面试题 16.24. 数对和（例题1）

1. 两数之和

15. 三数之和

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

75. 颜色分类

283. 移动零 已排序未排序指针（例题2）

面试题 16.06. 最小差 类似合并两个有序数组（例题3）

面试题 17.11. 单词距离 类似合并两个有序数组

剑指 Offer 57 - II. 和为s的连续正数序列 （例题1）

剑指 Offer 48. 最长不含重复字符的子字符串 （例题2）

438. 找到字符串中所有字母异位词

76. 最小覆盖子串

121. 买卖股票的最佳时机（例题1）

238. 除自身以外数组的乘积（例题2）

面试题 05.03. 翻转数位

42. 接雨水

53. 最大子序和

191. 位1的个数 （例题1）

461. 汉明距离（例题2）

面试题 05.06. 整数转换

面试题 05.07. 配对交换

面试题 05.01. 插入

面试题 17.04. 消失的数字

剑指 Offer 56 - I. 数组中数字出现的次数

剑指 Offer 56 - II. 数组中数字出现的次数 II

面试题 16.01. 交换数字

231. 2 的幂

# 前缀后缀统计

王争的算法训练营

## 121. 买卖股票的最佳时机（例题1）

给定一个数组 `prices` ，它的第 `i` 个元素 `prices[i]` 表示一支给定股票第 `i` 天的价格。

你只能选择 **某一天** 买入这只股票，并选择在 **未来的某一个不同的日子** 卖出该股票。设计一个算法来计算你所能获取的最大利润。

返回你可以从这笔交易中获取的最大利润。如果你不能获取任何利润，返回 `0` 。

**示例 1：**

```
输入：[7,1,5,3,6,4]
输出：5
解释：在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
     注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格；同时，你不能在买入前卖出股票。
```

**示例 2：**

```
输入：prices = [7,6,4,3,1]
输出：0
解释：在这种情况下, 没有交易完成, 所以最大利润为 0。
```

**提示：**

- $1 <= prices.length <= 10^5$
- $0 <= prices[i] <= 10^4$

# 前缀后缀统计



121. 买卖股票的最佳时机（例题1）

1、暴力
2、前缀后缀统计

```
class Solution {
    public int maxProfit(int[] prices) {
        int n = prices.length;
        int[] max = new int[n];
        int curMax = 0;
        for (int i = n-1; i >= 0; --i) {
            max[i] = curMax;
            if (prices[i] > curMax) curMax = prices[i];
        }
        int result = 0;
        for (int i = 0; i < n; ++i) {
            if (result < max[i]-prices[i]) {
                result = max[i]-prices[i];
            }
        }
        return result;
    }
}
```

## 238. 除自身以外数组的乘积（例题2）

给你一个长度为 *n* 的整数数组 `nums`，其中 *n* > 1，返回输出数组 `output` ，其中 `output[i]` 等于 `nums` 中除 `nums[i]` 之外其余各元素的乘积。

**示例:**

```
输入: [1,2,3,4]
输出: [24,12,8,6]
```

**提示：**题目数据保证数组之中任意元素的全部前缀元素和后缀（甚至是整个数组）的乘积都在 32 位整数范围内。

**说明:** 请**不要使用除法，**且在 O(*n*) 时间复杂度内完成此题。

**进阶：**
你可以在常数空间复杂度内完成这个题目吗？（ 出于对空间复杂度分析的目的，输出数组**不被视为**额外空间。）

```java
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int n = nums.length;
        int[] leftProducts = new int[n];
        int[] rightProducts = new int[n];
        int product = 1;
        for (int i = 0; i < n; ++i) {
            product *= nums[i];
            leftProducts[i] = product;
        }

        product = 1;
        for (int i = n-1; i >= 0; --i) {
            product *= nums[i];
            rightProducts[i] = product;
        }

        int[] result = new int[n];
        for (int i = 0; i < n; ++i) {
            result[i] = 1;
            if (i-1 >= 0) {
                result[i] *= leftProducts[i-1];
            }
            if (i+1 < n) {
                result[i] *= rightProducts[i+1];
            }
        }
        return result;
    }
}
```

除了输入输出不使用额外的空间

```java
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int n = nums.length;
        int[] result = new int[n];

        int leftProduct = 1;
        for (int i = 0; i < n; ++i) {
            result[i] = leftProduct;
            leftProduct *= nums[i];
        }

        int rightProduct = 1;
        for (int i = n-1; i >= 0; --i) {
            result[i] *= rightProduct;
            rightProduct *= nums[i];
        }
        return result;
    }
}
```

# 前缀后缀统计



面试题 05.03. 翻转数位

给定一个32位整数 `num`，你可以将一个数位从0变为1。请编写一个程序，找出你能够获得的最长的一串1的长度。

**示例 1：**

```
输入：num = 1775(11011101111₂)
输出：8
```

**统计每个0，左边有多少个1，右边有多少个1**

**1、暴力解法**

**2、前缀后缀统计**

**示例 2：**

```
输入：num = 7(0111₂)
输出：4
```

```java
class Solution {
    public int reverseBits(int num) {
        if (num == 0) return 1;
        int[] nums = new int[32];
        for (int i = 0; i < 32; ++i) {
            nums[i] = (num&1);
            num >>= 1;
        }
        int[] leftOneCounts = new int[32];
        int count = 0;
        for (int i = 0; i < 32; ++i) {
            leftOneCounts[i] = count;
            if (nums[i] == 1) {
                count++;
            } else {
                count = 0;
            }
        }

        int[] rightOneCounts = new int[32];
        count = 0;
        for (int i = 31; i >= 0; --i) {
            rightOneCounts[i] = count;
            if (nums[i] == 1) {
                count++;
            } else {
                count = 0;
            }
        }

        int ret = 0;
        for (int i = 0; i < 32; ++i) {
            if (ret < leftOneCounts[i] + rightOneCounts[i] + 1) {
                ret = leftOneCounts[i] + rightOneCounts[i] + 1;
            }
        }
        return ret;
    }
}
```

# 前缀后缀统计



## 42. 接雨水

给定 *n* 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

**示例 1：**

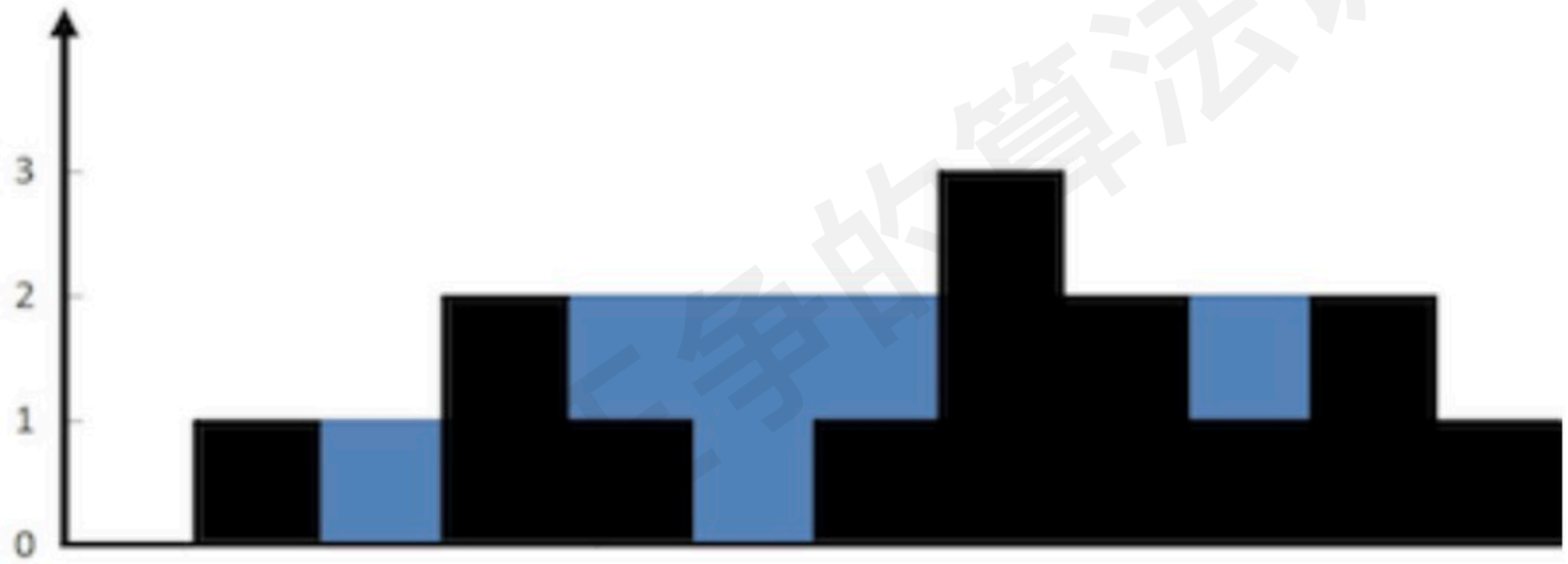

```
输入：height = [0,1,0,2,1,0,1,3,2,1,2,1]
输出：6
解释：上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。
```

**示例 2：**

```
输入：height = [4,2,0,3,2,5]
输出：9
```

## 1）暴力解法

核心思想：
每个柱子之上承载的水量 = min(左侧最高柱子lh，右侧最高柱子rh) - 这个柱子的高度h
总的接水量=每个柱子之上承载水量的总和

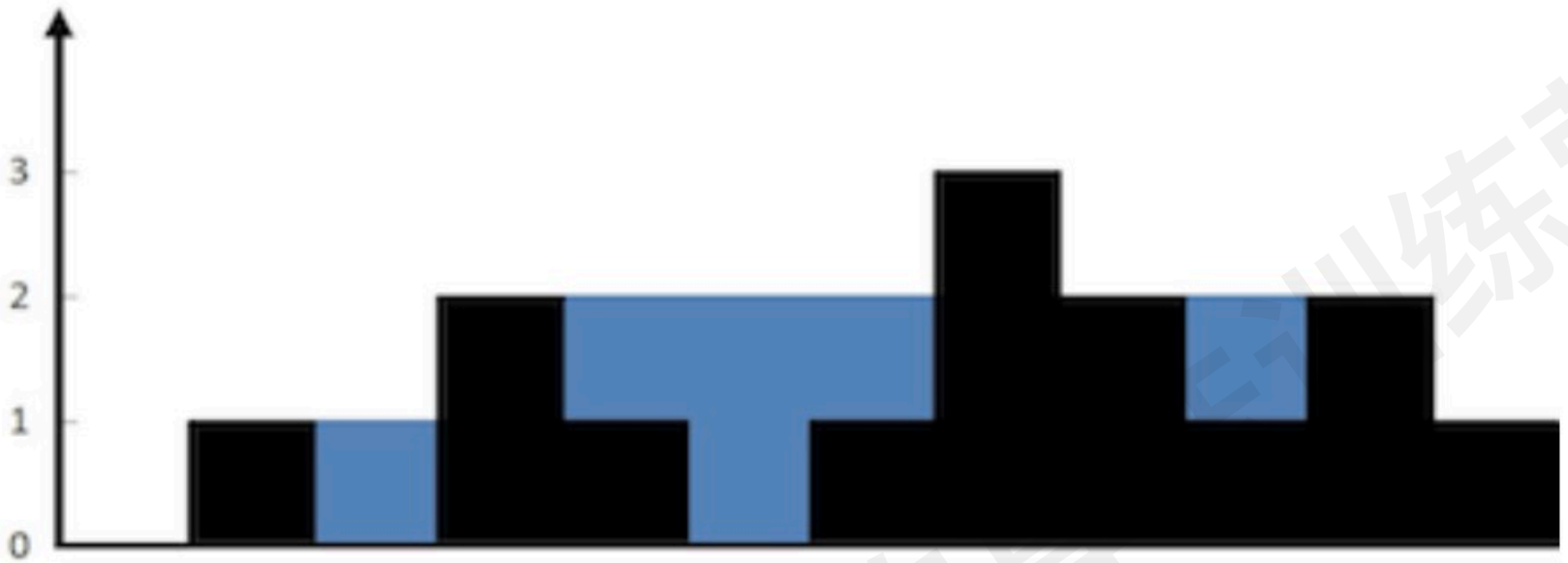


```java
class Solution {
    public int trap(int[] height) {
        int n = height.length;
        int result = 0;
        // 遍历每个柱子h，查找它左边的最高柱子lh，和右边的最高柱子rh
        // 柱子上能承载的雨水=min(lh, rh)-h
        for (int i = 1; i < n-1; ++i) {
            int lh = 0;
            for (int j = 0; j < i; ++j) { // 左侧最高lh
                if (height[j] > lh) lh = height[j];
            }
            int rh = 0;
            for (int j = i+1; j < n; ++j) { // 右侧最高rh
                if (height[j] > rh) rh = height[j];
            }
            int carry = Math.min(lh, rh) - height[i];
            if (carry < 0) carry = 0;
            result += carry;
        }
        return result;
    }
}
```

时间复杂度是O(n^2)

空间复杂度是O(1)

## 2）前缀/后缀统计解法



时间复杂度是O(n)
空间复杂度是O(n)

```java
class Solution {
    public int trap(int[] height) {
        int n = height.length;
        // 前缀max
        int[] leftMax = new int[n];
        int max = 0;
        for (int i = 0; i < n; ++i) {
            leftMax[i] = Math.max(max, height[i]);
            max = leftMax[i];
        }
        // 后缀max
        int[] rightMax = new int[n];
        max = 0;
        for (int i = n-1; i >= 0; --i) {
            rightMax[i] = Math.max(max, height[i]);
            max = rightMax[i];
        }
        // 每个柱子之上承载的雨水
        int result = 0;
        for (int i = 1; i < n-1; i++) {
            result += Math.min(leftMax[i], rightMax[i])-height[i];
        }
        return result;
    }
}
```

# 前缀后缀统计

王争的算法训练营

## 53. 最大子序和

1、滑动窗口
2、前缀后缀统计

给定一个整数数组 `nums` ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

**示例 1：**

```
输入：nums = [-2,1,-3,4,-1,2,1,-5,4]
输出：6
解释：连续子数组 [4,-1,2,1] 的和最大，为 6 。
```

**示例 2：**

```
输入：nums = [1]
输出：1
```

**示例 3：**

```
输入：nums = [0]
输出：0
```

**示例 4：**

```
输入：nums = [-1]
输出：-1
```

# 前缀后缀统计

王争的算法训练营

53. 最大子序和　**滑动窗口**

```
class Solution {
    public int maxSubArray(int[] nums) {
        int n = nums.length;
        int maxSum = Integer.MIN_VALUE;
        int sum = 0;
        for (int i = 0; i < n; ++i) {
            if (sum < 0) {
                sum = 0;
            }
            sum += nums[i];
            if (sum > maxSum) {
                maxSum = sum;
            }
        }
        return maxSum;
    }
}
```

# 前缀后缀统计

王争的算法训练营

53. 最大子序和

```
class Solution {
    public int maxSubArray(int[] nums) {
      if (nums.length == 1) return nums[0];
      int sum[] = new int[nums.length];
      int max[] = new int[nums.length];

      int cursum = 0;
      for (int i = 0; i < nums.length; ++i) {
        cursum += nums[i];
        sum[i] = cursum;
      }

      int curmax = Integer.MIN_VALUE;
      for (int i = sum.length-1; i >= 0; --i) {
        if (curmax < sum[i]) curmax = sum[i];
        max[i] = curmax;
      }

      int result = Integer.MIN_VALUE;

      for (int i = 0; i < nums.length; ++i) {
        if (i == 0 && result < max[0]) result = max[0];
        if (i != 0 && result < max[i]-sum[i-1]) result = max[i]-sum[i-1];
      }
      return result;
    }
}
```

# 位运算

344. 反转字符串

面试题 16.24. 数对和（例题1）

1. 两数之和

15. 三数之和

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

75. 颜色分类

283. 移动零 已排序未排序指针（例题2）

面试题 16.06. 最小差 类似合并两个有序数组（例题3）

面试题 17.11. 单词距离 类似合并两个有序数组

剑指 Offer 57 - II. 和为s的连续正数序列 （例题1）

剑指 Offer 48. 最长不含重复字符的子字符串 （例题2）

438. 找到字符串中所有字母异位词

76. 最小覆盖子串

53. 最大子序和（例题1）

121. 买卖股票的最佳时机（例题2）

238. 除自身以外数组的乘积（例题3）

面试题 05.03. 翻转数位

42. 接雨水

191. 位1的个数 （例题1）

461. 汉明距离（例题2）

面试题 05.06. 整数转换

面试题 05.07. 配对交换

面试题 05.01. 插入

面试题 17.04. 消失的数字

剑指 Offer 56 - I. 数组中数字出现的次数

剑指 Offer 56 - II. 数组中数字出现的次数 II

面试题 16.01. 交换数字

231. 2 的幂

# 位运算

王争的算法训练营

## 191. 位1的个数 （例题1）

编写一个函数，输入是一个无符号整数（以二进制串的形式），返回其二进制表达式中数字位数为 '1' 的个数（也被称为汉明重量）。

**示例 1：**

```
输入：00000000000000000000000000001011
输出：3
解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。
```

**示例 2：**

```
输入：00000000000000000000000010000000
输出：1
解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。
```

**示例 3：**

```
输入：11111111111111111111111111111101
输出：31
解释：输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。
```

# 位运算

## Java特殊情况：逻辑位移

王争的算法训练营

```java
public class Solution {
    // you need to treat n as an unsigned value
    public int hammingWeight(int n) {
        int oneCount = 0;
        int mask = 1;
        for (int i = 0; i < 32; i++) {
            if ((n & mask) != 0) {
                oneCount++;
            }
            mask <<= 1;
        }
        return oneCount;
    }
}
```

```java
public class Solution {
    // you need to treat n as an unsigned value
    public int hammingWeight(int n) {
        int oneCount = 0;
        int i = 1;
        while (n != 0) {
            if ((n&1)==1) oneCount++;
            n >>= 1;
        }
        return oneCount;
    }
}
```

无法处理负数

**示例 1：**

```
输入：00000000000000000000000000001011
输出：3
解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。
```

**示例 2：**

```
输入：00000000000000000000000010000000
输出：1
解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。
```

**示例 3：**

```
输入：11111111111111111111111111111101
输出：31
解释：输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。
```

# 位运算



## 461. 汉明距离（例题2）

两个整数之间的 汉明距离 指的是这两个数字对应二进制位不同的位置的数目。

给你两个整数 `x` 和 `y`，计算并返回它们之间的汉明距离。

**示例 1：**

```
输入：x = 1, y = 4
输出：2
解释：
1   (0 0 0 1)
4   (0 1 0 0)
      ↑   ↑
上面的箭头指出了对应二进制位不同的位置。
```

**示例 2：**

```
输入：x = 3, y = 1
输出：1
```

**提示：**

- $0 <= x, y <= 2^{31} - 1$

461. 汉明距离（例题2）

```
class Solution {
    public int hammingDistance(int x, int y) {
        int r = x ^ y;
        int mask = 1;
        int count = 0;
        for (int i = 0; i < 31; i++) {
            if ((r & mask) != 0) count++;
            mask *= 2;
        }
        return count;
    }
}
```

## 面试题 05.06. 整数转换

整数转换。编写一个函数，确定需要改变几个位才能将整数A转成整数B。

**示例1:**

```
输入：A = 29 （或者0b11101），B = 15（或者0b01111）
输出：2
```

**示例2:**

```
输入：A = 1，B = 2
输出：2
```

**提示:**

1. A，B范围在[-2147483648, 2147483647]之间

面试题 05.06. 整数转换

```
class Solution {
    public int convertInteger(int A, int B) {
        int C = A ^ B;
        int count = 0;
        for (int i = 0; i < 32; ++i) {
            if ((C&1) == 1) {
                count++;
            }
            C >>= 1;
        }
        return count;
    }
}
```

## 面试题 05.07. 配对交换

配对交换。编写程序，交换某个整数的奇数位和偶数位，尽量使用较少的指令（也就是说，位0与位1交换，位2与位3交换，以此类推）。

**示例1:**

```
输入: num = 2（或者0b10）
输出 1 （或者 0b01）
```

**示例2:**

```
输入: num = 3
输出: 3
```

**提示:**

1. `num` 的范围在[0, 2^30 - 1]之间，不会发生整数溢出。

面试题 05.07. 配对交换

```
class Solution {
    public int exchangeBits(int num) {
        int ret = 0;
        //num - [0, 2^30 - 1]
        //所以位数是30个位
        for(int i = 0; i <= 30; i += 2){
            //奇位与偶位
            int a1 = (num & (1 << i));
            int b1 = (num & (1 << (i+1)));
            //如果是1,则加上交换位;0则加上也没用
            if(a1 != 0) ret |= (1 << (i+1));
            if(b1 != 0) ret |= (1 << i);
        }
        return ret;
    }
}
```

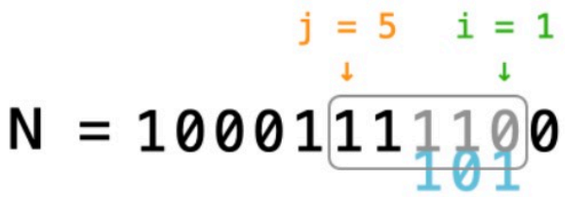
# 位运算



## 面试题 05.01. 插入

给定两个整型数字 `N` 与 `M`，以及表示比特位置的 `i` 与 `j`（`i <= j`，且从 0 位开始计算）。

编写一种方法，使 `M` 对应的二进制数字插入 `N` 对应的二进制数字的第 `i ~ j` 位区域，不足之处用 `0` 补齐。具体插入过程如图所示。




题目保证从 `i` 位到 `j` 位足以容纳 `M`， 例如： `M = 10011`，则 `i～j` 区域至少可容纳 5 位。

**示例1:**

```
输入：N = 1024(10000000000), M = 19(10011), i = 2, j = 6
输出：N = 1100(10001001100)
```

**示例2:**

```
输入：  N = 0, M = 31(11111), i = 0, j = 4
输出：N = 31(11111)
```

```java
class Solution {
    public int insertBits(int N, int M, int i, int j) {
        int[] nbits = new int[32];
        int[] mbits = new int[32];
        int nbitsNum = 0;
        int mbitsNum = 0;
        int mask = 1;
        for (int k = 0; k < 32; ++k) {
            if ((N&mask)!=0) {
                nbits[k] = 1;
            }
            mask <<= 1;
        }
        mask = 1;
        for (int k = 0; k < 32; ++k) {
            if ((M&mask)!=0) {
                mbits[k] = 1;
            }
            mask <<= 1;
        }
        for (int k = i; k <= j; ++k) {
            nbits[k] = mbits[k-i];
        }
        mask = 1;
        int ret = 0;
        for (int k = 0; k < 32; ++k) {
            ret += (nbits[k] * mask);
            mask <<= 1;
        }
        return ret;
    }
}
```

# 位运算



## 面试题 17.04. 消失的数字

数组 `nums` 包含从 `0` 到 `n` 的所有整数，但其中缺了一个。请编写代码找出那个缺失的整数。你有办法在O(n)时间内完成吗?

**注意：** 本题相对书上原题稍作改动

**示例 1：**

```
输入：[3,0,1]
输出：2
```

**示例 2：**

```
输入：[9,6,4,2,3,5,7,0,1]
输出：8
```

1、排序
2、位图
3、求和
4、位运算

# 位运算



面试题 17.04. 消失的数字

```
class Solution {
    public int missingNumber(int[] nums) {
        int n = nums.length;
        int ret = 0;
        for (int i = 0; i <= n; ++i) {
            ret ^= i;
        }
        for (int i = 0; i < n; ++i) {
            ret ^= nums[i];
        }
        return ret;
    }
}
```

# 位运算



## 剑指 Offer 56 - I. 数组中数字出现的次数

一个整型数组 `nums` 里除两个数字之外，其他数字都出现了两次。请写程序找出这两个只出现一次的数字。要求时间复杂度是O(n)，空间复杂度是O(1)。

**示例 1：**

```
输入：nums = [4,1,4,6]
输出：[1,6] 或 [6,1]
```

**示例 2：**

```
输入：nums = [1,2,10,4,1,4,3,3]
输出：[2,10] 或 [10,2]
```

**限制：**

- `2 <= nums.length <= 10000`

# 位运算



## 剑指 Offer 56 - I. 数组中数字出现的次数

```
class Solution {
    public int[] singleNumbers(int[] nums) {
        int xorResult = 0;
        int n = nums.length;
        for (int i = 0; i < n; ++i) {
            xorResult ^= nums[i];
        }
        int tag = 1;
        while ((xorResult & tag) == 0) {
            tag = tag << 1;
        }
        int a = 0;
        int b = 0;
        for (int i = 0; i < n; ++i) {
            if ((nums[i] & tag) == 0) {
                a ^= nums[i];
            } else {
                b ^= nums[i];
            }
        }
        return new int[] {a, b};
    }
}
```

# 位运算



## 剑指 Offer 56 - II. 数组中数字出现的次数 II

在一个数组 `nums` 中除一个数字只出现一次之外，其他数字都出现了三次。请找出那个只出现一次的数字。

**示例 1：**

```
输入：nums = [3,4,3,3]
输出：4
```

**示例 2：**

```
输入：nums = [9,1,7,9,7,9,7]
输出：1
```

**限制：**

- `1 <= nums.length <= 10000`
- `1 <= nums[i] < 2^31`

# 位运算



## 剑指 Offer 56 - II. 数组中数字出现的次数 II

```
class Solution {
    public int singleNumber(int[] nums) {
        int n = nums.length;
        int[] bits = new int[32];
        int mask = 1;
        for (int i = 0; i < 32; ++i) {
            for (int j = 0; j < n; ++j) {
                if ((nums[j]&mask) != 0) {
                    bits[i] = (bits[i] + 1) % 3;
                }
            }
            mask <<= 1;
        }
        int result = 0;
        mask = 1;
        for (int i = 0; i < 32; ++i) {
            if (bits[i] == 1) {
                result += mask;
            }
            mask <<= 1;
        }
        return result;
    }
}
```

面试题 16.01. 交换数字

编写一个函数，不用临时变量，直接交换 `numbers = [a, b]` 中 `a` 与 `b` 的值。

**示例：**

```
输入: numbers = [1,2]
输出: [2,1]
```

**提示：**

- `numbers.length == 2`
- `-2147483647 <= numbers[i] <= 2147483647`

面试题 16.01. 交换数字

```
class Solution {
    public int[] swapNumbers(int[] numbers) {
        if(numbers[0]==numbers[1]) return numbers;
        numbers[0]^=numbers[1];
        numbers[1]^=numbers[0];
        numbers[0]^=numbers[1];
        return numbers;
    }
}
```

# 位运算



## 231. 2 的幂

给你一个整数 `n`，请你判断该整数是否是 2 的幂次方。如果是，返回 `true` ；否则，返回 `false` 。

如果存在一个整数 `x` 使得 `n == 2^x` ，则认为 `n` 是 2 的幂次方。

**示例 1：**

```
输入：n = 1
输出：true
解释：2^0 = 1
```

**示例 2：**

```
输入：n = 16
输出：true
解释：2^4 = 16
```

**示例 3：**

```
输入：n = 3
输出：false
```

231. 2 的幂

```
class Solution {
    public boolean isPowerOfTwo(int n) {
        while (n != 0) {
            if ((n & 1) == 1) {
                if ((n >> 1) == 0) return true;
                else return false;
            }
            n >>= 1;
        }
        return false;
    }
}
```

# 提问环节

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料