1.Pow(x, n)
2.寻找两个正序数组的中位数
3.数组中的逆序对
4.翻转对
5.计算右侧小于当前元素的个数
6..分割数组的最大值
7.x的平方根
8.旋转数组的最小数字
9.最长递增子序列
10.俄罗斯套娃问题

50. Pow(x, n)

难度 中等 707 收藏 分享 切换为英文 接收动态 反馈

实现 pow(*x*, *n*) ，即计算 x 的 n 次幂函数（即，x^n）。

示例 1：

```
输入：x = 2.00000, n = 10
输出：1024.00000
```

示例 2：

```
输入：x = 2.10000, n = 3
输出：9.26100
```

示例 3：

```
输入：x = 2.00000, n = -2
输出：0.25000
解释：2^-2 = 1/2^2 = 1/4 = 0.25
```



```python
class Solution:
    def myPow(self, x: float, n: int) -> float:
        def quick(x, n):
            ans = 1.0
            x_con = x
            while n:
                if n & 1 == 1:
                    ans *= x_con
                x_con *= x_con
                n >>= 1
            return ans
        return quick(x, n) if n >= 0 else 1.0 / quick(x, -n)
```

4. 寻找两个正序数组的中位数

难度 困难 4359 收藏 分享 切换为英文 接收动态 反

给定两个大小分别为 `m` 和 `n` 的正序（从小到大）数组 `nums1` 和 `nums2` 。请你找出并返回这两个正序数组的 **中位数** 。

示例 1：

```
输入：nums1 = [1,3], nums2 = [2]
输出：2.00000
解释：合并数组 = [1,2,3] ，中位数 2
```

示例 2：

```
输入：nums1 = [1,2], nums2 = [3,4]
输出：2.50000
解释：合并数组 = [1,2,3,4] ，中位数 (2 + 3) / 2 = 2.5
```

示例 3：

```
输入：nums1 = [0,0], nums2 = [0,0]
输出：0.00000
```

示例 4：

```
输入：nums1 = [], nums2 = [1]
输出：1.00000
```

```python
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2:
        def getkelement(k):
            """
            - 主要思路：要找到第 k (k>1) 小的元素，那么就取 pivot1
            - 这里的 "/" 表示整除
            - nums1 中小于等于 pivot1 的元素有 nums1[0 .. k/2-2]
            - nums2 中小于等于 pivot2 的元素有 nums2[0 .. k/2-2]
            - 取 pivot = min(pivot1, pivot2)，两个数组中小于等于
            - 这样 pivot 本身最大也只能是第 k-1 小的元素
            - 如果 pivot = pivot1，那么 nums1[0 .. k/2-1] 都不可
            - 如果 pivot = pivot2，那么 nums2[0 .. k/2-1] 都不可
            - 由于我们 "删除" 了一些元素（这些元素都比第 k 小的元素要
            """
            idx1, idx2 = 0, 0
            while True:
                # 特殊情况
                if idx1 == m:
                    return nums2[idx2 + k - 1]
                if idx2 == n:
                    return nums1[idx1 + k - 1]
                if k == 1:
                    return min(nums1[idx1], nums2[idx2])
                # 正常情况
                newidx1 = min(idx1 + k // 2 - 1, m - 1)
                newidx2 = min(idx2 + k // 2 - 1, n - 1)
                pivot1, pivot2 = nums1[newidx1], nums2[newidx
                if pivot1 < pivot2:
                    k -= newidx1 - idx1 + 1
                    idx1 = newidx1 + 1
                else:
                    k -= newidx2 - idx2 + 1
                    idx2 = newidx2 + 1

        m, n = len(nums1), len(nums2)
        total = m + n
        if total % 2 == 1:
            return getkelement((total + 1) // 2)
        else:
            return (getkelement(total // 2) + getkelement(tot
```

剑指 Offer 51. 数组中的逆序对

难度 困难 501 收藏 分享 切换为英文 接收动态 反

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输
组，求出这个数组中的逆序对的总数。

示例 1:

```
输入: [7,5,6,4]
输出: 5
```

```python
class Solution:
    def reversePairs(self, nums: List[int]) -> int:
        n = len(nums)
        tmp = [0] * n
        return self.merge_sort(nums, tmp, 0, n - 1)

    def merge_sort(self, nums, tmp, left, right):
        if left >= right:
            return 0
        mid = (left + right) >> 1
        res = self.merge_sort(nums, tmp, left, mid) + \
              self.merge_sort(nums, tmp, mid + 1, right) + \
              self.find_reverse(nums, left, right)
        i, j, k = left, mid + 1, left
        while i <= mid and j <= right:
            if nums[i] <= nums[j]:
                tmp[k] = nums[i]
                i += 1
            else:
                tmp[k] = nums[j]
                j += 1
            k += 1
        while i <= mid:
            tmp[k] = nums[i]
```



```python
            k += 1
            i += 1
        while j <= right:
            tmp[k] = nums[j]
            k += 1
            j += 1
        nums[left:right + 1] = tmp[left:right + 1]
        return res

    def find_reverse(self, nums, left, right):
        res, mid = 0, (left + right) >> 1
        j = mid + 1
        for i in range(left, mid + 1):
            while j <= right and nums[i] > nums[j]:
                res += mid + 1 - i
                j += 1
        return res
```

493. 翻转对

难度 困难 294 收藏 分享 切换为英文 接收动态 反馈

给定一个数组 `nums` ，如果 `i < j` 且 `nums[i] > 2*nums[j]` 我们就将 `(i, j)` 称作
翻转对。

你需要返回给定数组中的重要翻转对的数量。

示例 1:

```
输入: [1,3,2,3,1]
输出: 2
```

示例 2:

```
输入: [2,4,3,5,1]
输出: 3
```

```python
class Solution:
    def reversePairs(self, nums: List[int]) -> int:
        n = len(nums)
```

```
        tmp = [0] * n
        return self.merge_sort(nums, tmp, 0, n - 1)

    def merge_sort(self, nums, tmp, left, right):
        if left >= right:
            return 0
        mid = (left + right) >> 1
        res = self.merge_sort(nums, tmp, left, mid) + \
              self.merge_sort(nums, tmp, mid + 1, right) + \
              self.find_reverse(nums, left, right)
        i, j, k = left, mid + 1, left
        while i <= mid and j <= right:
            if nums[i] <= nums[j]:
                tmp[k] = nums[i]
                i += 1
            else:
                tmp[k] = nums[j]
                j += 1
            k += 1
        while i <= mid:
            tmp[k] = nums[i]
            i += 1
            k += 1
        while j <= right:
            tmp[k] = nums[j]
            j += 1
            k += 1
        nums[left:right + 1] = tmp[left:right + 1]
        return res

    def find_reverse(self, nums, left, right):
        res, mid = 0, (left + right) >> 1
        j = mid + 1
        for i in range(left, mid + 1):
            while j <= right and nums[i] > 2 * nums[j]:
                res += mid + 1 - i
```



```
                j += 1
        return res
```

315. 计算右侧小于当前元素的个数

难度 困难 632 收藏 分享 切换为英文 接收动态 反馈

给定一个整数数组 *nums*，按要求返回一个新数组 *counts*。数组 *counts* 有该性质： `counts` 是 `nums[i]` 右侧小于 `nums[i]` 的元素的数量。

示例：

```
输入：nums = [5,2,6,1]
输出：[2,1,1,0]
解释：
5 的右侧有 2 个更小的元素 (2 和 1)
2 的右侧仅有 1 个更小的元素 (1)
6 的右侧有 1 个更小的元素 (1)
1 的右侧有 0 个更小的元素
```

```python
class Solution:
    def countSmaller(self, nums: List[int]) -> List[int]:
        self.ans = [0] * len(nums)
        tmp = []
        for idx, val in enumerate(nums):
            tmp.append((val, idx))
        self.merge_sort(tmp)
        return self.ans

    def merge_sort(self, nums):
        if len(nums) < 2:
            return nums
        mid = len(nums) >> 1
        left = self.merge_sort(nums[:mid])
        right = self.merge_sort(nums[mid:])
        return self.merge(left, right)

    def merge(self, left, right):
        res = []
        i, j = 0, 0
        while i < len(left) and j < len(right):
            if left[i][0] > right[j][0]:
                res.append(right[j])
                j += 1
            else:
                res.append(left[i])
                self.ans[left[i][1]] += j
                i += 1
        if i == len(left):
            res += right[j:]
        else:
            for k in range(i, len(left)):
                self.ans[left[k][1]] += j
            res += left[i:]
        return res
```


410. 分割数组的最大值

难度 困难 533 收藏 分享 切换为英文 接收动态 反

给定一个非负整数数组 `nums` 和一个整数 `m` ，你需要将这个数组分成 `m` 个非空的连续子

设计一个算法使得这 `m` 个子数组各自和的最大值最小。

示例 1：

```
输入：nums = [7,2,5,10,8], m = 2
输出：18
解释：
一共有四种方法将 nums 分割为 2 个子数组。 其中最好的方式是将其分为 [7,2,5] 
[10,8] 。
因为此时这两个子数组各自的和的最大值为18，在所有情况中最小。
```

示例 2：

```
输入：nums = [1,2,3,4,5], m = 2
输出：9
```

示例 3：

```
输入：nums = [1,4,4], m = 3
输出：4
```

```python
class Solution:
    def splitArray(self, nums: List[int], m: int) -> int:
        def check(x):
            total, cnt = 0, 1
            for num in nums:
                if total + num > x:
                    total = num
                    cnt += 1
                else:
                    total += num
            return cnt <= m

        left, right = max(nums), sum(nums)
        while left < right:
            mid = (left + right) >> 1
            if check(mid):
                right = mid
            else:
                left = mid + 1
        return left
```


69. x 的平方根

难度 简单 752 收藏 分享 切换为英文 接收动态 反

实现 `int sqrt(int x)` 函数。

计算并返回 *x* 的平方根，其中 *x* 是非负整数。

由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。

示例 1:

> **输入:** 4
> **输出:** 2

示例 2:

> **输入:** 8
> **输出:** 2
> **说明:** 8 的平方根是 2.82842...,
> 由于返回类型是整数，小数部分将被舍去。

```python
class Solution:
    def mySqrt(self, x: int) -> int:
        if x == 0:
            return 0

        left = 1
        right = x // 2

        while left < right:
            mid = (left + right + 1) >> 1
            square = mid * mid

            if square > x:
                right = mid - 1
            else:
                left = mid
        return left
```