分治法

分治法的本意是把一个程度复杂或者计算规模大的问题转化为多个规模小的问题，然后再把小规模问题的答案结合起来得到问题的解，从而解决该问题的一种方法。

Leetcode 第 50 题：实现 pow(*x*, *n*)，即计算 x 的 n 次幂函数。

示例 1:
输入: 2.00000, 10
输出: 1024.00000

示例 2:
输入: 2.10000, 3
输出: 9.26100

示例 3:
输入: 2.00000, -2
输出: 0.25000
解释: 2-2 = 1/22 = 1/4 = 0.25

说明:
-100.0 < x < 100.0

n 是 32 位有符号整数，其数值范围是 [−231, 231 − 1] 。

解题思路：
这是求解幂级函数函数的问题，可考虑使用 x*Pow(x,n-1)方式简化问题，不过这样虽然能解决问题，但是时间复杂度为 O（n）太高，不能通过。为了把时间复杂度降下来，我们可以考虑使用求解 Pow(x,n//2),这样一来只需要 log(n)的时间复杂度就可以了，然后我们在考虑 n 为奇数偶数的问题，若 n 为偶数，则输出为 Pow(x,n//2)**2,否则输出为 x*[Pow(x,n//2)**2],done。

代码如下（递归实现）：

```
class Solution:
    def myPow(self, x: float, n: int) -> float:
        def pos(x,n):
            if n==0:
                return 1.0
            y=pos(x,n//2)
```

```
            return y*y if n % 2 == 0 else x*y*y

        return pos(x,n) if n>=0 else 1.0/pos(x,-1*n)
```

Leetcode 第 53 题最大子序列和：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:

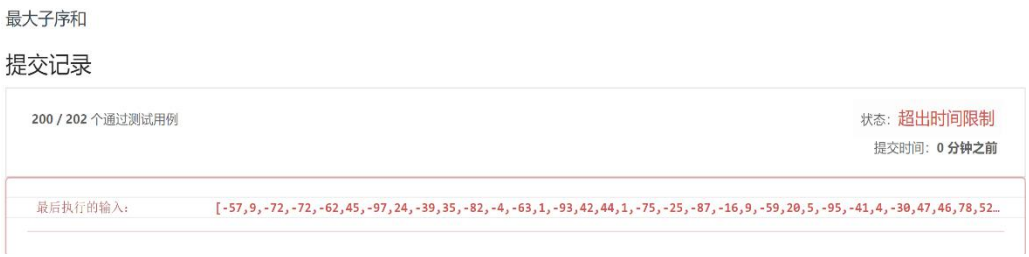
输入: [-2,1,-3,4,-1,2,1,-5,4]

输出: 6

解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。

解法一：暴力法
解题思路：遍历所有的可能，找出最大的子序列和值，时间复杂度 O(n^2)
代码如下：

```
n=len(nums)
        max_v=-float("INF")
        if n==1:
            return nums[0]
        else:
            for i in range(n):
                for j in range(i+1,n+1):
                    max_v=max(max_v,sum(nums[i:j]))
            return max_v
```



解法二：分治法（时间复杂度：O（NlogN））
解题思路：把该问题可以划分为规模更小(N/2)的问题，把数组从中划分为两部分。如此一来就有三种可能：1）最大连续子序列在前半 2）最大连续子序列在后半段 3）最大连续子序列连跨两段。把三种可能都求出来，最后求个最大值即为结果
代码如下：

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

```
        if n==0:return 0
        elif n==1:return nums[0]
        else:
            mid=(n-1)//2
            max_l=self.maxSubArray(nums[:mid+1])
            max_r=self.maxSubArray(nums[mid+1:])
            max1=nums[mid]
            max2=nums[mid+1]
            for i in range(mid+1):
                max1=max(max1,sum(nums[i:mid+1]))
            for j in range(mid+1,n+1):
                max2=max(max2,sum(nums[mid+1:j]))
            return max(max_l,max_r,max1+max2)
```

改善版：

```
n=len(nums)
        if n==0:return 0
        elif n==1:return nums[0]
        else:
            mid=(n-1)//2
            max_l=self.maxSubArray(nums[:mid+1])
            max_r=self.maxSubArray(nums[mid+1:])
            max1,sum_l=nums[mid],0
            max2,sum_r=nums[mid+1],0
            i,j=mid,mid+1
            while i>=0:
                sum_l+=nums[i]
                max1=max(max1,sum_l)
                i-=1
            while j<n:
                sum_r+=nums[j]
                max2=max(max2,sum_r)
```

```
            j+=1
        return max(max_l,max_r,max1+max2)
```

解法三：动态规划(时间复杂度：O(N))
解题思路：构造动态规划数组 dp，以某个点为结尾求其连续子序列的最大值，如果 dp[i-1]>0,则对 dp[i]=dp[i-1]+nums[i],否则，dp[i]=nums[i]
代码如下：

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        #方法一：dp
        n=len(nums)
        if n==0:
            return 0
        elif n==1:
            return nums[0]
        else:
            max_v=nums[0]
            dp=[0]*n
            dp[0]=nums[0]
            for i in range(1,n):
                if dp[i-1]>0:
                    dp[i]=dp[i-1]+nums[i]
                    max_v=max(dp[i],max_v)
                else:
                    dp[i]=nums[i]
                    max_v=max(dp[i],max_v)
            return max_v
```

Leetcode 第 169 题-多数元素：给定一个大小为 n 的数组，找到其中的多数元素。多数元素是指在数组中出现次数大于 n/2 的元素。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

方法一：字典法（时间复杂度：O(N)）。

解题思路：统计所有元素的次数，然后求出次数大于 n/2 的那个。

代码如下：

```
from collections import Counter
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        dic=Counter(nums)
        n=len(nums)
        thre=n//2
        for num in nums:
            if dic[num]>thre:
                return num
```

解法二：投票法（时间复杂度：O（N）空间复杂度：O（1））

解题思路：两两 pk，如果相同，就给计数器加一，如果不同，就相互抵消，从头开始。

代码如下：

```
n=len(nums)
        if n==1:return nums[0]
        i,count,result=1,1,nums[0]
        while i<n:
            if nums[i]!=result:
                count-=1
                if count==0 and i+1<n:
                    result=nums[i+1]
                    i+=2
                    count=1
                else:
                    i+=1
            else:
                count+=1
                i+=1
        return result
```

解法三：排序法（时间复杂度：O（NlgN））

解题思路：由于有一个元素的次数大于 N//2，则排序后，中间元素则为该元素。

代码如下：

```
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        n=len(nums)
        nums=sorted(nums)
        return nums[n//2]
```

方法四：分治法（时间复杂度：**O（NlgN）**）

解题思路：把数组拆分为两半，求出每一半出现次数最多的值，则会得出两个竞争对手，二者在原数组 PK，则可得出现次数超过 n//2 的数字。
代码如下：

```
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        n=len(nums)
        if n==0:
            return None
        if n==1:
            return nums[0]
        mid=(N-1)//2
        left=self.majorityElements(nums[:mid+1])
        right=self.majorityElements(nums[mid+1:])

        If left==right:
            return left
        elif nums.count(left)>nums.count(right):
            return left
        else:
            return right
```