LeetCode 1

**1.two sum**：找出两数想加等于一个特定的数

方法：哈希，可以认为哈希中查找的时间是O(1)

**5. Longest Palindromic Substring**

方法1：reverse之后转为Longest Common Substring，LCS用动态规划，n\*n的矩阵，最长对角线就是结果，可以只保存上下两行，时间O(n2)，空间O(n)

方法2：LPS从中间展开往两边是一样的，一个字符串可能有2n-1个可能的中心（因为两个字母之间也可能是中心），遍历这些可能的中心。时间O(n2)，空间O(1)

**23. Merge k Sorted Lists**

方法5：用分治，先合并分别合并k/2个list，然后合并这两个list，时间复杂度O(NlogK)，空间复杂度O(1)

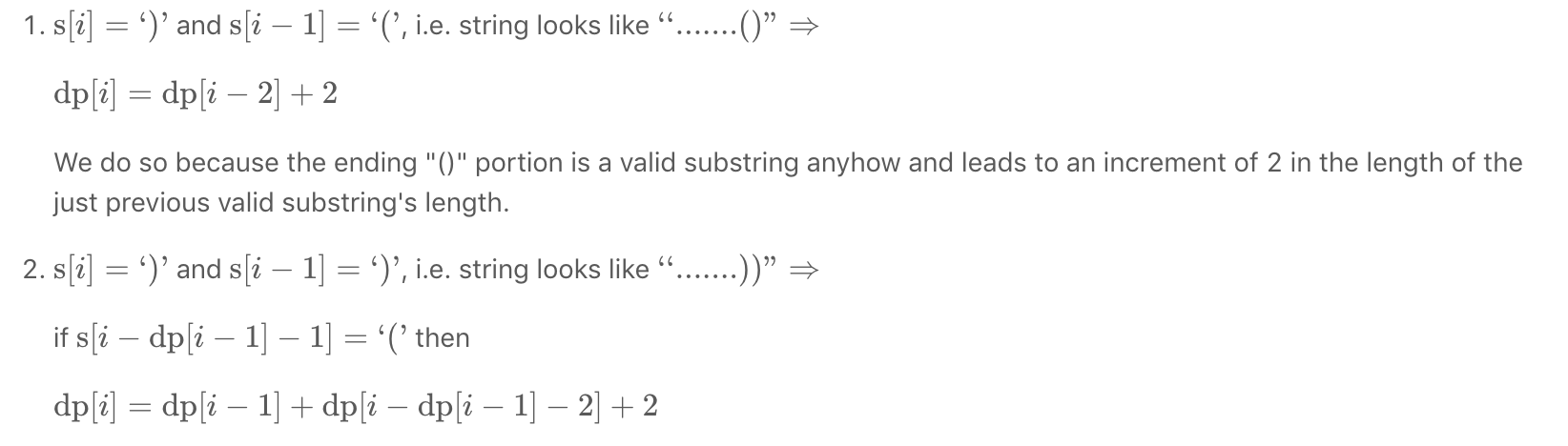
**29. Divide Two Integers**

<https://leetcode.com/problems/divide-two-integers/discuss/427345/Python-24ms-beats-99-with-and-wo-bitwise-operators>

假设结果是1，每次翻一倍，直到再大乘以除数后大于被除数，这个乘数即为结果的第一部分。用被除数减去第一部分乘以除数，将结果当作新除数算出第二部分，如此循环。最后将所有部分相加就是结果

**32. Longest Valid Parentheses**

<https://leetcode.com/problems/longest-valid-parentheses/solution/>



**42. Trapping Rain Water**

用递减栈，当碰到一个数大于栈中所有数时候，所有栈中的点的积水量都是可计算的

**84. Largest Rectangle in Histogram**

用递增栈，当遇到一个数小于刚刚加入的数的时候，栈中所有比它大的数所能形成的面积都是可计算的

**92. Reverse Linked List II反转链表中的一部分**

1.Recursion类似交换数组，将第n个元素和倒数第n个元素互换即可，但是由于链表是单向的，利用递归和回溯的办法使得指针能够向前移

2.Iterative Link Reversal先将反转的这部分数组一个个反转过来（只要将指向后的指针变为指向前即可），然后事先记好这个链表的头和尾，把它和原来的链表连上

**94. Binary Tree Inorder Traversal**

Approach 3: Morris Traversal

Step 1: Initialize current as root

Step 2: While current is not NULL,

If current does not have left child

a. Add current’s value

b. Go to the right, i.e., current = current.right

Else

a. In current's left subtree, make current the right child of the rightmost node

b. Go to this left child, i.e., current = current.left

**95. Unique Binary Search Trees II**

动态规划：假设第i个数为root，那么左子树有i-1个点，右子树有n-i个点

f(0) = 1

f(n) = f(0)\*f(n-1) + f(1)\*f(n-2) + ... + f(n-2)\*f(1) + f(n-1)\*f(0)

**99. Recover Binary Search Tree**

利用Morris traversal

**109. Convert Sorted List to Binary Search Tree**

方法3：模拟中序遍历，因为中序遍历BST就是一个递增序列，所以用递归去做假定的目标树的中序遍历，当遍历到一个点的时候，要么是none，要么就是链表的下一个元素。这样就可以按照中序遍历的顺序构造这颗树

**114. Flatten Binary Tree to Linked List**

前序遍历顺序：对于每个节点，将左子树连接到右节点，将右子树链接到左子树的最右子节点的右边

**116. Populating Next Right Pointers in Each Node**

对于每一层的节点来说，如果它们父节点相同，那么就是左节点的next指向右节点，如果父节点不同，那么就是左边节点的右节点指向右边节点的左节点，由于左右节点通过next相连，所以可以直接获得。故最终的方法是在每一层next计算完成，则在这一层将下一层的next算好再下移到下一层

**121. Best Time to Buy and Sell Stock**

首先记录一个最小值，如果当前值比最小值，更新最小值，否则用当前值减去最小值计算利润更新最大利润（因为最小值是之前存储的，所以一定在前面）

**127. Word Ladder**

快速寻找和一个词只相差一个字母的所有词方法：将这个词中任意一个字母变为星号，用这个模型去匹配，其中星号可以匹配任意字母，只要能匹配上的所有单词互相之间相差一个字母

**128. Longest Consecutive Sequence**

1.用哈希表存数字，这样查找时间是O(1)

2.不遍历重复的序列，如果一个数的前一个数不在数组中，那么这个数才可能是新序列的开始

**134. Gas Station**

定理：如果一个数组的总和非负，那么一定可以找到一个起始位置，从它开始绕数组一圈，累加和一直都是非负的

解：先判断总和，如果为负，则没有解。否则从每一个能出发的位置开始尝试，注意失败之后可以直接从失败的位置开始，因为这条路从原来的非负数开始都失败，从其中任意一个点走也必然失败

**135. Candy**

1.贪心法，维持一个和原数组相同的空数组表示分配的糖果数量，初始值都为1，然后从左扫描，给高rating的加一，再从右扫描，所得数组和就是糖果数量

2.优化空间复杂度，注意到上升阶段，糖果数量是1，2，3……n，下降阶段是m，m-1，……1.而这些数的和可以直接公式计算。对于极大值，应该取两边值中大的那个

**136. Single Number**

利用异或运算的性质，相同的数异或之后为0，0和任何数异或之后还是这个数，所以将所有的数异或，即得解

**137. Single Number II**

通用解法是用数位表示状态转换，的上取整是2，所以需要两个数位表示，转换过程为00→01→10→00，即出现三次之后复位为0，这样在第二位上，只有出现一次的数为1。而各数位之间的关系可以通过真值表写出

**138. Copy List with Random Pointer**

We can avoid using extra space for old node ---> new node mapping, by tweaking the original linked list. Simply interweave the nodes of the old and copied list. For e.g.

Old List: A --> B --> C --> D

InterWeaved List: A --> A' --> B --> B' --> C --> C' --> D --> D'

**141. Linked List Cycle**

**142. Linked List Cycle II**

Floyd's Tortoise and Hare: 快指针每次走两部，慢指针每次一步，如有环，最终在距离环的入口C-H处相遇，其中C为环的长度，H为非环的长度除以C取余

第二阶段可以找到环的入口：通过两个指针，一个指向开始，一个指向刚刚交汇的地方，每次各走一步，可证明相遇的地方就是环的入口

**143. Reorder List**

1.利用快慢指针将链表分为两段，将第二段反转，将第二段交替插入第一段形成最终解

2.利用栈后进先出的特性，第一个出栈的元素就是最后一个元素，插入到第二个节点的位置即可，依次插入一半的结点即可

**146. LRU Cache**

常规LRU利用哈希表和双向链表来实现

**148. Sort List**

自然合并排序法：对于初始给定的数组a，通常存在多个长度大于1的已排好序的子数组段。因此用一次对a的线性扫描就可以找出所有这些排好序的子数组段，然后将相邻的排好序的子数组段两两合并。对于链表来说，只需要常数空间

**149. Max Points on a Line**

用分数表示斜率以处理精度问题，化简最简式用到最大公约数

greatest common divider最大公约数GCD通过欧几里得算法Euclidean algorithm辗转相除法得到：gcd(a,b)=gcd(b, a mod b )，时间复杂度是O(logn)

**151. Reverse Words in a String**

C语言O(1)空间复杂度做法：先将整个字符串反转，再逐个将单词反转

**152. Maximum Product Subarray**

负数个数为偶数时，分为两种情况，一是舍弃第一个负数及以前，而是舍弃最后一个负数及以后。所以从前向后和从后向前两次遍历即可，注意遇到0要重置

**153. Find Minimum in Rotated Sorted Array**

旋转点之前，数组递增，之后数组也递增，而且之后的所有值小于之前的所有值，二分法找到旋转点

**154. Find Minimum in Rotated Sorted Array II**

如果终点和左边值相等，则左指针右移一位，缩小搜查范围即可，最坏情况，所有数相等，退化为O(n)

**155. Min Stack**

每个值和到它之前的最小值一同入栈

**160. Intersection of Two Linked Lists**

两个指针一起走，短的指针走到底的时候放到长的开头，这样长的走到底的时候短的就指向交汇的位置，数学可证

**162. Find Peak Element**

因为头尾都是负无穷，所以可以使用改版的二分搜索（小心验证

**164. Maximum Gap**

基于桶排序，可以验证的是最大gap的最小值是t=(max−min)/(n−1)，所以当桶的size小于这个值的时候，最大gap就只可能在两个桶之间，而不会在桶内部。两桶之间的比较只需要比较两桶之间的最值即可

**169. Majority Element**

Approach 6: Boyer-Moore Voting Algorithm多数投票算法，设置一个count，每次count大于零的时候选新的candidate，最后使得count大于零的candidate就是要找的值

**172. Factorial Trailing Zeroes**

0只能由2和5相乘得到，判断阶乘质因子中有多少2和5就行了，由于2的数量远多于5的数量，数5即可。由于25=5\*5，所以n数5之后还要对递归对n/5进行数5。所以sol(n)=n/5+sol(n/5)

**187. Repeated DNA Sequences**

基本思路是将每十个字符作为key存入哈希表，如果出现超过两次，就返回

改进方法是注意到四个字符的二进制只有最后3位不同，这样用最后3位表示一个字符，一个int可以表示十个字符的串，这样转化为int的比较和搜索，速度快很多（而且每十个字符不同的只有首尾两个字符，把原先的数字左移三位再加上新的字符即可

**188. Best Time to Buy and Sell Stock IV**

动态规划式local[i][j] = max(global[i - 1][j - 1] + max(diff, 0), local[i - 1][j] + diff)

global[i][j] = max(local[i][j], global[i - 1][j])

其中local[i][j]是第i天j次交易且最后一次交易卖出日在i的最大收益，global[i][j]是i天j次交易的最大利润，diff表示i和i-1天的价格差

**189. Rotate Array**

方法1:从一个位置开始找到它的正确位置，然后把那个位置上的元素再找到正确位置，如此反复

方法2:现将整个数组反序，然后将前k个元素反序，再将后n-k个元素反序

**190. Reverse Bits**

分治法（二分法），每次翻转一半，最后problem size是2的时候将两个位调一下即可

因为长度固定，也可以用迭代实现，先把后16位和前16位互换顺序，在将16位中的每个8位互换顺序，依此类推。

**191.Number of 1 Bits**

1.每次将最后一位1翻转成0，如果这个数变成了0，那么二进制表示里就没有更多的1了2.一个数和前一个数取并（n&n-1）可以将最后一位1翻转成0

**199. Binary Tree Right Side View**

DFS，总是走右子树，当没有子节点的时候回退，注意的是因为可能回退遍历再回退多次，所以遍历的时候需要记录深度，如果当前深度的最右节点已经有了要忽略。BFS同理可以解

**200. Number of Islands**

找到一个1，进行DFS，岛屿数加一，如此反复。需要有个数组标记那些点遍历过，或者允许修改输入的可以直接在原图标零

**201. Bitwise AND of Numbers Range**

找从m到n所有数的共同部分=找m和n左边开始的共同部分

方法1：找一个mask，一开始全1，从右边开始一位一位的设零（右移就行），知道它和m和n与的结果一样

方法2：m和n每次右移一位，直到两个数相等，记录这时的位移长度，再左移回去

方法3：利用n和n-1与每次能将最后一位1置0的性质，将n不断如此操作直到n比m小，此时n就是答案

**202. Happy Number**

非快乐数的循环中必有4

**204. Count Primes**

指数筛选算法埃拉托色尼筛选法（The Sieve of Eratosthenes）：从2开始，每找到一个质数，就标记它的所有倍数，这样依次第一个没有被标记的就是所有的质数

空间开销大，可以通过位操作节省空间

**206. Reverse Linked List**

利用三个指针，每次可以将前两个元素的顺序翻转，即又指向后变为指向前，依次进行到最后一位即可

**212. Word Search II**

用单词树存放词典可以大大增加查找效率（也便于利用回溯）。然后对板中没个字母进行DFS看是否能找到匹配的单词，回溯法

**213. House Robber II**

分别算出抢1到n-1家和2到n家的最大值，两个最大值中取大的即为所求

**214. Shortest Palindrome**

本质是找到从零开始最长的回文子串，然后将后面的内容逆序放到前面即可。

如果将字符串复制一份并首尾相连，那么问题的前半部分就变为找字符串前面和后面相同的部分，即是KMP问题的一部分，用KMP算法可以在O(n)时间求解。注意相连时中间需要加一个特殊字符以区分两段

**215. Kth Largest Element in an Array**

利用最小堆。如果数组中的元素都是不同的，可以利用快排partition，将partition的位置和k比较可以得到下一次partition的部分

**216. Combination Sum III**

同理运用递归，个人觉得可以限制每次结束的数字加速

**218. The Skyline Problem**

把输入转换成左端点+高度和右端点+高度两个变量，按照所有的端点序排序，维持一个最大堆，然后从左往右扫描，如果遇到左端点，就将高度加入堆，然后记录当前的最大值，如果是右端点，则不断删除最大值知道最大值对应的右端点超出当前右端点的值。这样中途堆中存在的某些长方形可能已经遍历结束了，但是不影响结果，因为要么它们的高度比最大值小，要么它们在到达堆顶的时候会被删除

**220. Contains Duplicate III**

保持k大小的窗口，用一种数据结构（集合）存储其中的数，对于每个新加数，看看窗口中有没有跟这个数构成符合要求的数的组合

**221. Maximal Square**

动态规划：dp(i,j)=min(dp(i−1,j),dp(i−1,j−1),dp(i,j−1))+1.

**222. Count Complete Tree Nodes**

分别获得左右子树的最大高度，如果相等，说明左子树满，右子树完全，递归求右子树即可，如果不等，说明右子树满，左子树完全，递归求左子树即可

**224. Basic Calculator**

只有加减和括号。方法1：用栈，但是因为栈反着算结果不对，所以先把表达式反向，但是注意多位数的数字不能反过来。方法2：因为加法具有结合律和交换律，所以把减法变成加相反数，则不用反向

**227. Basic Calculator II**

有加减乘除，没有括号。用栈，减法变加法，每次先算乘除，再算加减

**229. Majority Element II**

找出所有出现次数大于n/3的数，易知这种数最多两个，意味着这两个数出现的频率还是比其他任意一个数要高。所以可用摩尔投票法，即出现一个新的数，这两个数的票数都可以减一。注意题目没有假设这样的数一定存在，所以需要二次检验

**231. Power of Two**

利用二的指数幂中二进制表示只有一个1这一属性

**233. Number of Digit One**

数学问题，需要找规律

**236. Lowest Common Ancestor of a Binary Tree**

DFS找这两个点，找到第一个点时，记录点，每次回退时候，如果当前点就是记录点，记录点上移，当找到第二个点的时候，记录点在的位置就是最小公共祖先

**239. Sliding Window Maximum**

方法1：用双向队列记录数的下标（主要是解决数据重复的问题，否则也可以记录数），每当窗口增加或者移动时候，以此删除比增加的数还要小的数，即在当前数据前面又比当前数字小的数对所有含有当前数的窗口结果不会产生影响，视为无效数据，可以在队列中删除。由于没个数都是进队列一次，出队列一次，所以复杂度是O(n)

方法2：把数组按照个数为k分组，最后一组可能不是全部数据。以组为单位，从左到右方向，逐个计算组内遇到的最大数，记录为新的数组leftMaxArray；再从右到左方向，逐个计算组内遇到的最大数，记录为新的数组rightMaxArray.。因为从左到右方向–>是最大数，从右到左方向<–也是最大数，所以在某个位置index, k个数里面最多只能跨两个域，所以Max(leftMaxArray[index], rightMaxArray[index - k + 1])就是最大数。实际时间复杂度可以优化为O(2n)

**240. Search a 2D Matrix II**

注意两次二分搜索没有办法解这道题。从右上角开始搜索，如果小于当前数，则可以删除这一行，反之可以删除这一列，这样最多删除n次就可以找到。实际实现时从右上角往左或者网下便利即可

**241. Different Ways to Add Parentheses**

方法1：分治或者递归或者DFS

方法2：动态规划。dp[i][j] 表示在第i个数字到第j个数字之间范围内的子串添加不同括号所能得到的不同值的整型数组

**247. Shortest Word Distance II**

用哈希表记录每个单词出现的位置，然后比较两个单词的数组即可