***Linked list***

*'''  
237. Delete Node in a Linked List  
Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.  
Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3,  
the linked list should become 1 -> 2 -> 4 after calling your function.  
'''  
  
# Definition for singly-linked list.***class** ListNode(object):  
 **def** \_\_init\_\_(self, x):  
 self.val = x  
 self.next = None  
  
**class** Solution(object):  
 **def** deleteNode(self, node):  
 *"""* **:type** *node: ListNode* **:rtype***: void Do not return anything, modify node in-place instead.  
 """* node.val = node.next.val  
 node.next = node.next.next

---------------------------------------------------------------------

*'''  
203. Remove Linked List Elements  
Remove all elements from a linked list of integers that have value val.  
  
Example  
Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6  
Return: 1 --> 2 --> 3 --> 4 --> 5  
'''  
# Definition for singly-linked list.***class** ListNode(object):  
 **def** \_\_init\_\_(self, x):  
 self.val = x  
 self.next = None  
  
  
**class** Solution(object):  
 **def** removeElements(self, head, val):  
 *"""* **:type** *head: ListNode* **:type** *val: int* **:rtype***: ListNode  
 """* dummy = ListNode(-1) *# add a dummy node* dummy.next = head *# head might be deleted if head.val == val* cur = dummy *# cur is dynamic, always changing,like a temp pointer* **while** cur != None **and** cur.next != None: *# first time cur.next is head* **if** cur.next.val == val:  
 cur.next = cur.next.next  
 **else**:  
 cur = cur.next  
  
 **return** dummy.next *# neither head nor cur, only dummy.next is right* **'''  
 # wrong  
 cur = head  
 while cur != None:  
 if cur.val == val and cur.next != None:  
 cur = cur.next  
 elif cur.val == val and cur.next == None:  
 cur = None  
 else:  
 cur = cur.next  
 return head  
 '''**

Integer

*'''  
412. Fizz Buzz  
Write a program that outputs the string representation of numbers from 1 to n.  
  
But for multiples of three it should output “Fizz” instead of the number and for the multiples of  
five output “Buzz”. For numbers which are multiples of both three and five output “FizzBuzz”.  
  
Example:  
  
n = 15,  
  
Return:  
[  
 "1",  
 "2",  
 "Fizz",  
 "4",  
 "Buzz",  
 "Fizz",  
 "7",  
 "8",  
 "Fizz",  
 "Buzz",  
 "11",  
 "Fizz",  
 "13",  
 "14",  
 "FizzBuzz"  
]  
'''***def** fizzBuzz(n):  
 a = []  
 **for** x **in** xrange(1,n+1):  
 **if** x%3 == 0 **and** x%5 == 0:  
 a.append(**"FizzBuzz"**)  
 **elif** x%3 == 0:  
 a.append(**"Fizz"**)  
 **elif** x%5 == 0:  
 a.append(**"Buzz"**)  
 **else**:  
 a.append(str(x))  
 **return** a  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 **print** fizzBuzz(1)

--------------------------------------------------------------------------------------------------------------

*'''  
258. Add Digits  
Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.  
For example:  
Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.  
Follow up:  
Could you do it without any loop/recursion in O(1) runtime?  
'''  
  
  
# https://en.wikipedia.org/wiki/Digital\_root***class** Solution(object):  
 **def** addDigits(self, num):  
 *"""* **:type** *num: int* **:rtype***: int  
 """* **if** num == 0:  
 **return** 0  
 **elif** num%9 == 0:  
 **return** 9  
 **else**:  
 **return** num%9

--------------------------------------------------------------------------------------------------------------

String

*'''  
387. First Unique Character in a String  
  
Given a string, find the first non-repeating character in it and return it's index.  
If it doesn't exist, return -1.  
  
Examples:  
s = "leetcode"  
return 0.  
  
s = "loveleetcode",  
return 2.  
'''***import** string  
  
**class** Solution(object):  
 **def** firstUniqChar(self, s):  
 *"""* **:type** *s: str* **:rtype***: int  
 """  
  
  
 # 75 ms* L = []  
 **for** c **in** string.ascii\_lowercase:  
 **if** s.count(c) == 1: *# s.count costs O(n)* L.append(s.find(c)) *# s.find costs O(n)* **if** len(L) >= 1:  
 **return** min(L)  
 **else**:  
 **return** -1  
  
  
 **'''  
 # 99ms  
 return min([s.find(c) for c in string.ascii\_lowercase if s.count(c)==1] or [-1])  
 '''  
  
  
 '''  
 # time exceed: go through each element until one meet requirement  
 for i in xrange(len(s)):  
 if s.count(s[i]) == 1:  
 return i  
 return -1  
 '''  
  
 '''  
 # wrong: set change the original string order  
 S = set(s)  
 for x in S:  
 if s.count(x) == 1:  
 return s.index(x)  
 return -1  
 '''  
  
 '''  
 Difference between find and index?  
 str.find returns -1 when it does not find the substring.  
  
 >>> line = 'hi, this is ABC oh my god!!'  
 >>> line.find('?')  
 -1  
  
 While str.index raises ValueError:  
 >>> line.index('?')  
 Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
 ValueError: substring not found  
 '''**

**---------------------------------------------------------------------------***'''  
344. Reverse String  
Write a function that takes a string as input and returns the string reversed.  
  
Example:  
Given s = "hello", return "olleh".  
  
'''***def** reverseString0(s):  
 L = len(s)  
 a = **""  
 for** i **in** xrange(1,L+1):  
 a += s[-i]  
 **return** a  
  
**from** collections **import** deque  
**def** reverseString(s):  
 L = len(s)  
 a = deque()  
 **for** i **in** xrange(L):  
 a.appendleft(s[i])  
 **return ''**.join(a)  
  
  
**print** reverseString(**"abc defg"**)

---------------------------------------------------------------------------

*'''  
383. Ransom Note  
Given an arbitrary ransom note string and another string containing letters from all the magazines,  
write a function that will return true if the ransom note can be constructed from the magazines ;  
otherwise, it will return false.  
  
Each letter in the magazine string can only be used once in your ransom note.  
  
Note:  
You may assume that both strings contain only lowercase letters.  
  
canConstruct("a", "b") -> false  
canConstruct("aa", "ab") -> false  
canConstruct("aa", "aab") -> true  
'''***class** Solution(object):  
 **def** canConstruct(self, ransomNote, magazine):  
 *"""* **:type** *ransomNote: str* **:type** *magazine: str* **:rtype***: bool  
 """* **"""  
 str.count(sub, start= 0,end=len(string))  
 list.count(obj)  
  
 """** *# 69ms, beat > 90%* s = set(ransomNote)  
 **for** x **in** s:  
 **if** ransomNote.count(x) > magazine.count(x):  
 **return** False  
 **return** True  
  
  
 **'''  
 # 182ms  
 return not collections.Counter(ransomNote) - collections.Counter(magazine)  
 '''  
  
 '''  
 # time limit exceeds  
 L = list(magazine)  
 for x in ransomNote:  
 if L.count(x):  
 L.remove(x)  
 else:  
 return False  
 return True  
 '''** *# could not pass  
 # Input: "aa", "ab"  
 # Output: ture  
 # expected: false* **'''  
 if (set(ransomNote) - set(magazine)) == set([]):  
 return True  
 else:  
 return False  
 '''** *# could not pass  
 # Input: "fffbfg", "effjfggbffjdgbjjhhdegh"  
 # Output: false  
 # Expected: true* **'''  
 if magazine.count(ransomNote)==0:  
 return False  
 else:  
 return True  
 '''**

--------------------------------------------------------------------------

*'''  
389. Find the Difference  
Given two strings s and t which consist of only lowercase letters.  
String t is generated by random shuffling string s and then add one more letter at a random position.  
Find the letter that was added in t.  
Example:  
Input:  
s = "abcd"  
t = "abcde"  
Output:  
e  
Explanation:  
'e' is the letter that was added.  
'''***class** Solution(object):  
 **def** findTheDifference(self, s, t):  
 *"""* **:type** *s: str* **:type** *t: str* **:rtype***: str  
 """  
  
 # 42ms: ord and chr* diff = 0  
 **for** i **in** range(len(s)): *# first -,then +* diff -= ord(s[i]) *# ord(x): Converts a single character to its integer value.* diff += ord(t[i])  
 diff += ord(t[-1]) *# after add the final, diff is positive* **return** chr(diff) *# chr(x): Converts an integer to a character.  
  
  
 # 52ms ^XOR,ord and chr* code = 0  
 **for** ch **in** s + t: *# ord(x): Converts a single character to its integer value.* code ^= ord(ch) *# 0^ch=ch, ch^ch=0, the rest is the final result* **return** chr(code) *# chr(x): Converts an integer to a character.  
  
  
 # 66ms: dictionary  
 # dict.get(key, default=None), default is the Value to be returned in case key does not exist.* dic = {}  
 **for** ch **in** s:  
 dic[ch] = dic.get(ch, 0) + 1 *# record all the character in dic* **for** ch **in** t:  
 **if** dic.get(ch, 0) == 0: *# if this char did not exist or no more exists* **return** ch  
 **else**:  
 dic[ch] -= 1  
  
  
 *# 82ms: collections.Counter* **from** collections **import** Counter  
 **return** (Counter(t)-Counter(s)).popitem()[0]  
 **'''  
 what is Counter?  
 https://docs.python.org/2/library/collections.html#collections.Counter  
 >>> from collections import Counter  
 >>> x = "abcde"  
 >>> y = "abc"  
 >>> Counter(x)-Counter(y)  
 Counter({'e': 1, 'd': 1})  
 >>> (Counter(x)-Counter(y))  
 Counter({'e': 1, 'd': 1})  
 >>> (Counter(x)-Counter(y)).popitem()  
 ('e', 1)  
 >>> (Counter(x)-Counter(y)).popitem()[0]  
 'e'  
 '''** *# 92ms: list remove* li\_t = list(t)  
 **for** x **in** s:  
 li\_t.remove(x)  
 **return ''**.join(li\_t) *# convert list to string* **'''  
 wrong solution: filter or replace, if duplicates, could not pass s="a",t="aa"  
  
 1.  
 return filter(lambda x: x not in s,t)  
  
 2.  
 for x in s:  
 t = t.replace(x,"")  
 return t  
  
 '''**

--------------------------------------------------------------------------

Tree

*'''  
100. Same Tree  
Given two binary trees, write a function to check if they are equal or not.  
Two binary trees are considered equal if they are structurally identical and the nodes have the same value.  
'''  
  
# Definition for a binary tree node.  
# class TreeNode(object):  
# def \_\_init\_\_(self, x):  
# self.val = x  
# self.left = None  
# self.right = None***class** Solution(object):  
 **def** isSameTree(self, p, q):  
 *"""* **:type** *p: TreeNode* **:type** *q: TreeNode* **:rtype***: bool  
 """* **'''  
 # 48 ms  
 if p == None and q == None:  
 return True  
 elif (p != None and q != None) and (p.val == q.val) and self.isSameTree(p.left,q.left) and self.isSameTree(p.right,q.right):  
 return True  
 else:  
 return False  
 '''** *# 35 ms* **if** p **and** q:  
 **return** p.val == q.val **and** self.isSameTree(p.left, q.left) **and** self.isSameTree(p.right, q.right)  
 **return** p **is** q *# when p == None and q == None: (p is q) = True; otherwise (p is q) = False  
  
 # 45 ms  
 #return p and q and p.val == q.val and self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right) or p is q  
  
 # 39 ms  
 #return p and q and p.val == q.val and all(map(self.isSameTree, (p.left, p.right), (q.left, q.right))) or p is q*

*'''  
104. Maximum Depth of Binary Tree  
Given a binary tree, find its maximum depth.  
The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.  
'''***class** Solution(object):  
 **def** maxDepth1(self, root): *# slower, Runtime: 102 ms  
 """* **:type** *root: TreeNode* **:rtype***: int  
 """* **if not** root:  
 **return** 0  
 **if not** root.left **and not** root.right:  
 **return** 1  
 **elif** root.left **and not** root.right:  
 **return** self.maxDepth(root.left) + 1 *# when do recursive, use self.maxDepth, rather than maxDepth* **elif** root.right **and not** root.left:  
 **return** self.maxDepth(root.right) + 1  
 **else**:  
 **return** max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1  
  
 **def** maxDepth(self, root): *# faster, Runtime: 65 ms  
 """* **:type** *root: TreeNode* **:rtype***: int  
 """* **if** root == None:  
 **return** 0  
 **if** root.left == None **and** root.right == None:  
 **return** 1  
 **elif** root.left **and** root.right == None:  
 **return** self.maxDepth(root.left) + 1 *# when do recursive, use self.maxDepth, rather than maxDepth* **elif** root.right **and** root.left == None:  
 **return** self.maxDepth(root.right) + 1  
 **else**:  
 **return** max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1  
  
 **def** maxDepth(self, root): *# Medium, Runtime: 72 ms  
 """* **:type** *root: TreeNode* **:rtype***: int  
 """* **if not** root:  
 **return** 0  
 **else**:  
 **return** max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1  
  
**'''  
Test case explanation  
[]  
Empty tree.  
The root is a reference to NULL (C/C++), null (Java/C#/Javascript), None (Python), or nil (Ruby).  
  
[1,2,3]  
 1  
 / \  
 2 3  
  
[1,null,2,3]  
 1  
 \  
 2  
 /  
 3  
  
[5,4,7,3,null,2,null,-1,null,9]  
 5  
 / \  
 4 7  
 / /  
 3 2  
 / /  
 -1 9  
'''  
  
'''  
what's the difference between not x and x == None?  
not x will also return True for everything that evaluates to False in a boolean context.  
Some examples:  
>>> x = ()  
>>> not x  
True  
  
>>> x = []  
>>> not x  
True  
  
>>> x = ''  
>>> not x  
True  
  
>>> x = 0  
>>> not x  
True  
  
>>> x is None  
False  
  
So if your code should act differently when x is None as opposed to x being an empty list,  
tuple, string, the number zero, ...  
then use x == None or x is None instead of not x  
'''**

--------------------------------------------------------------------------------------------------------------------------------------

*'''  
226. Invert Binary Tree  
GInvert a binary tree.  
  
 4  
 / \  
 2 7  
 / \ / \  
1 3 6 9  
to  
 4  
 / \  
 7 2  
 / \ / \  
9 6 3 1  
'''  
  
# Definition for a binary tree node.  
# class TreeNode(object):  
# def \_\_init\_\_(self, x):  
# self.val = x  
# self.left = None  
# self.right = None***class** Solution(object):  
 **def** invertTree(self, root):  
 *"""* **:type** *root: TreeNode* **:rtype***: TreeNode  
 """* **'''  
 if root == None:  
 return None  
 else:  
 root.left, root.right = self.invertTree(root.right), self.invertTree(root.left)  
 return root  
 '''  
  
 if** root:  
 root.left, root.right = self.invertTree(root.right), self.invertTree(root.left)  
 **return** root

-----------------------------------------------------------------------------------------------------------------------------------------

Array

*'''  
27. Remove Element  
Given an array and a value, remove all instances of that value in place and return the new length.  
Do not allocate extra space for another array, you must do this in place with constant memory.  
The order of elements can be changed. It doesn't matter what you leave beyond the new length.  
  
Example:  
Given input array nums = [3,2,2,3], val = 3  
Your function should return length = 2, with the first two elements of nums being 2.  
'''***class** Solution(object):  
 **def** removeElement(self, nums, val):  
 *"""* **:type** *nums: List[int]* **:type** *val: int* **:rtype***: int  
 """* **while**(nums.count(val)!= 0):  
 nums.remove(val)  
 **return** len(nums)

---------------------------------------------------------------------------------------------------------------------------------------------------

*'''  
283. Move Zeroes  
Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.  
For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].  
  
Note:  
You must do this in-place without making a copy of the array.  
Minimize the total number of operations.  
'''***class** Solution(object):  
 **def** moveZeroes(self, nums):  
 *"""* **:type** *nums: List[int]* **:rtype***: void Do not return anything, modify nums in-place instead.  
 """  
  
 # 59ms* L = len(nums)  
 p = 0  
 **for** i **in** xrange(L):  
 **if** nums[i] != 0:  
 nums[i],nums[p] = nums[p],nums[i]  
 p += 1  
  
 *# 282ms* **for** x **in** nums:  
 **if** x == 0:  
 nums.remove(x)  
 nums.append(0)

-------------------------------------------------------------------------------------------------------------------------------------------------

*'''  
Given two arrays, write a function to compute their intersection.  
  
Example:  
Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2].  
  
Note:  
Each element in the result must be unique.  
The result can be in any order.  
'''***class** Solution(object):  
 **def** intersection(self, nums1, nums2):  
 *"""* **:type** *nums1: List[int]* **:type** *nums2: List[int]* **:rtype***: List[int]  
 """  
 # 45 ms* **return** list(set(nums1)&set(nums2))  
  
 **'''  
 # 52 ms  
 return list(set(nums1)-(set(nums1)-set(nums2)))  
 '''**

--------------------------------------------------------------------------------------------------------------------------------------------

*'''  
350. Intersection of Two Arrays II  
Given two arrays, write a function to compute their intersection.  
  
Example:  
Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2, 2].  
  
Note:  
Each element in the result should appear as many times as it shows in both arrays.  
The result can be in any order.  
  
Follow up:  
What if the given array is already sorted? How would you optimize your algorithm?  
What if nums1's size is small compared to nums2's size? Which algorithm is better?  
What if elements of nums2 are stored on disk, and the memory is limited such that you cannot  
load all elements into the memory at once?  
'''***from** collections **import** Counter  
  
**class** Solution(object):  
 **def** intersect(self, nums1, nums2):  
 *"""* **:type** *nums1: List[int]* **:type** *nums2: List[int]* **:rtype***: List[int]  
 """  
  
 # 52 ms* c1, c2 = Counter(nums1), Counter(nums2)  
 **return** sum([[num] \* min(c1[num], c2[num]) **for** num **in** c1 & c2], [])  
  
  
 **'''  
 # 82 ms  
 S = set(nums1)&set(nums2)  
 L= []  
 for x in S:  
 L += [x] \* min(nums1.count(x),nums2.count(x))  
 return L  
 '''**

---------------------------------------------------------------------------------------------------------------------------------------------------

Integer

*'''  
371. Sum of Two Integers  
Calculate the sum of two integers a and b, but you are not allowed to use the operator + and -.  
Example:  
Given a = 1 and b = 2, return 3.  
'''***def** getSum(self, a, b):  
 MAX\_INT = 0x7FFFFFFF  
 MIN\_INT = 0x80000000  
 MASK = 0x100000000  
 **while** b:  
 a, b = (a ^ b) % MASK, ((a & b) << 1) % MASK  
 **return** a **if** a <= MAX\_INT **else** ~((a % MIN\_INT) ^ MAX\_INT)  
  
**'''  
What's hexadecimal ?  
As each hexadecimal digit represents four binary digits (bits),  
it allows a human-friendly representation of binary-coded values.  
For example, a single byte can have values ranging from 00000000 to 11111111,  
in binary, but this is more conveniently represented as 00 to FF in hexadecimal.  
  
Python: long and int ?  
in my machine cywin:  
>>> print type(0x7FFFFFFFFFFFFFFF)  
<type 'int'>  
  
>>> print type(0x7FFFFFFFFFFFFFFF + 1)  
<type 'long'>  
  
  
python return if else?  
These three versions are the same.  
1)  
if(A > B):  
 return A+1  
return A-1  
  
2)  
if(A > B):  
 return A+1  
else:  
 return A-1  
  
3)return A+1 if A > B else A-1  
  
'''  
  
'''  
Python整数不是固定的32位，所以需要做一些特殊的处理，  
代码里的将一个数对0x100000000取模（注意：Python的取模运算结果恒为非负数），  
是希望该数的二进制表示从第32位开始到更高的位都同是0（最低位是第0位），以在0-31位上模拟一个32位的int。  
  
python因为自动整数越界为long。用python要用0x10000000 (33位）做mask取模保持int。  
'''  
  
'''  
32位的int，正数的范围是(0,0x7FFFFFFF),负数(0x80000000,0xFFFFFFFF)  
'''  
  
  
'''  
# First iteration (a is 20, b is 20)  
10100 ^ 10100 == 00000 # makes a 0  
(10100 & 10100) << 1 == 101000 # makes b 40  
  
# Second iteration:  
000000 ^ 101000 == 101000 # Makes a 40  
(000000 & 101000) << 1 == 0000000 # Makes b 0  
  
  
What do the masks do?  
All the masks ensures that the value is an integer.  
Since the maximum possible int (32 bits) is 2147483647, if you add 2 to this value,  
like you did in your example,the int overflows and you get a negative value.  
You have to force this in Python, because it doesn't respect this int boundary as strongly  
typed languages like Java and C++ have defined.  
  
Consider the following:  
def get\_sum(a, b):  
 while b:  
 a, b = (a ^ b), (a & b) << 1 # without the masks  
 return a  
  
print get\_sum(2147483647, 2)  
outputs: 2147483649  
  
while  
print Solution().getSum(2147483647, 2)  
outputs: -2147483647  
  
due to the overflow.  
'''  
  
'''  
Python has more than 32 bits for integers. You can try to run "print 2 \*\* 31"  
Python would shows the exact number correctly,  
while other languages like Java would not. Java only recognizes -2 \*\* 31 to 2 \*\* 31 - 1.  
  
How does integers presented in Python differ from integers in 32-bit e.g. Java?  
From what I heard, Python has 64 bits. (Please let me know if I am wrong. )  
So 1 in Python would look like 0x0000000000000001, but it looks like 0x00000001 in 32-bit format.  
-1 in Python would look like 0xFFFFFFFFFFFFFFFF, but it looks like 0xFFFFFFFF in 32-bit format.  
  
It seems that the input given by LC is in 32-bit format.  
Since Python would treat it as positive with 1 on the 32 position,  
we have to use mask to treat it as negative.  
'''**

**-----------------------------------------------------------**

*'''  
442. Find All Duplicates in an Array  
Given an array of integers, 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once.  
Find all the elements that appear twice in this array.  
Could you do it without extra space and in O(n) runtime?  
  
Example:  
Input:  
[4,3,2,7,8,2,3,1]  
  
Output:  
[2,3]  
  
similar: 448  
'''***class** Solution(object):  
 **def** findDuplicates(self, nums):  
 *"""* **:type** *nums: List[int]* **:rtype***: List[int]  
 """  
  
 # 305ms* dic = {}  
 li\_dup = []  
  
 **for** x **in** nums:  
 dic[x] = dic.get(x,0) + 1  
  
 **for** x **in** nums:  
 **if** dic[x] == 2:  
 li\_dup.append(x)  
 dic[x] = -1  
  
 **return** li\_dup  
  
  
 **'''  
 # 315ms  
 li\_dup = []  
 for x in nums:  
 if nums[abs(x)-1] < 0:  
 li\_dup.append(abs(x))  
 else:  
 nums[abs(x)-1] \*= -1  
 return li\_dup  
 '''**

**-------------------------------------------**

*'''  
448. Find All Numbers Disappeared in an Array  
Given an array of integers where 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once.  
Find all the elements of [1, n] inclusive that do not appear in this array.  
Could you do it without extra space and in O(n) runtime? You may assume the returned list does not count as extra space.  
  
Example:  
  
Input:  
[4,3,2,7,8,2,3,1]  
  
Output:  
[5,6]  
  
similar to 442  
'''***class** Solution(object):  
 **def** findDisappearedNumbers(self, nums):  
 *"""* **:type** *nums: List[int]* **:rtype***: List[int]  
 """  
  
   
 # without extra space, O(n)335ms* **for** i **in** xrange(len(nums)):  
 index = abs(nums[i]) - 1  
 nums[index] = - abs(nums[index])  
  
 **return** [i + 1 **for** i **in** xrange(len(nums)) **if** nums[i] > 0]  
   
  
 **'''  
 # use set 242ms  
 return set(range(1,len(nums)+1)) - set(nums)  
 '''  
   
 '''  
 # use filter: run out of time  
 li\_num = []  
 for i in xrange(1,len(nums)+1):  
 li\_num.append(i)  
 return filter(lambda x: x not in nums,li\_num)  
 '''  
  
 '''  
 # use dictionary 254ms  
 dic = {}  
 li\_disappear = []  
 for x in nums:  
 dic[x] = dic.get(x,0)+1  
  
 for i in xrange(1,len(nums)+1):  
 if dic.get(i,0) == 0:  
 li\_disappear.append(i)  
 return li\_disappear  
 '''  
  
 '''  
 difference between range and xrange  
 range(): range(1, 10) returns a list from 1 to 10 numbers & hold whole list in memory.  
 xrange(): Like range(), but instead of returning a list, returns an object that generates the numbers in the range on demand. For looping, this is lightly faster than range() and more memory efficient. xrange() object like an iterator and generates the numbers on demand.(Lazy Evaluation)  
 In [1]: range(1,10)  
 Out[1]: [1, 2, 3, 4, 5, 6, 7, 8, 9]  
 In [2]: xrange(10)  
 Out[2]: xrange(10)  
 In [3]: print xrange.\_\_doc\_\_  
 xrange([start,] stop[, step]) -> xrange object  
 '''**

**---------------------------------------------**

*'''  
453. Minimum Moves to Equal Array Elements  
Given a non-empty integer array of size n, find the minimum number of moves required to make all array elements equal,   
where a move is incrementing n - 1 elements by 1.  
  
Example:  
  
Input:  
[1,2,3]  
  
Output:  
3  
  
Explanation:  
Only three moves are needed (remember each move increments two elements):  
  
[1,2,3] => [2,3,3] => [3,4,3] => [4,4,4]  
'''***class** Solution(object):  
 **def** minMoves(self, nums):  
 *"""* **:type** *nums: List[int]* **:rtype***: int  
 """  
 # think in oppisite way, times of incrementing n - 1 elements by 1 is equal to times of decrementing 1 elements by 1.  
 # consider the original state and final state, the difference of sum divide by step is the frequenct times.* step = 1  
 **return** (sum(nums) - len(nums)\* min(nums)) / step

**---------------------------------------------**

Bit operation

*----------------------------------------------------------------------*

*'''  
The Hamming distance between two integers is the number of positions at which the corresponding bits are different.  
Given two integers x and y, calculate the Hamming distance.  
  
Note:  
0 ≤ x, y < 2^31.  
  
Example:  
Input: x = 1, y = 4  
Output: 2  
  
Explanation:  
1 (0 0 0 1)  
4 (0 1 0 0)  
 ↑ ↑  
  
The above arrows point to positions where the corresponding bits are different.  
'''***class** Solution(object):  
 **def** hammingDistance(self, x, y):  
 *"""* **:type** *x: int* **:type** *y: int* **:rtype***: int  
 """* **return** list(bin(x^y)).count(**"1"**) *# bin: Convert an integer number to a binary string.*

*---------------------------------------------------------------------------*

*'''  
371. Sum of Two Integers  
Calculate the sum of two integers a and b, but you are not allowed to use the operator + and -.  
Example:  
Given a = 1 and b = 2, return 3.  
'''***def** getSum(self, a, b):  
 MAX\_INT = 0x7FFFFFFF  
 MIN\_INT = 0x80000000  
 MASK = 0x100000000  
 **while** b:  
 a, b = (a ^ b) % MASK, ((a & b) << 1) % MASK  
 **return** a **if** a <= MAX\_INT **else** ~((a % MIN\_INT) ^ MAX\_INT)  
  
**'''  
What's hexadecimal ?  
As each hexadecimal digit represents four binary digits (bits),  
it allows a human-friendly representation of binary-coded values.  
For example, a single byte can have values ranging from 00000000 to 11111111,  
in binary, but this is more conveniently represented as 00 to FF in hexadecimal.  
  
Python: long and int ?  
in my machine cywin:  
>>> print type(0x7FFFFFFFFFFFFFFF)  
<type 'int'>  
  
>>> print type(0x7FFFFFFFFFFFFFFF + 1)  
<type 'long'>  
  
  
python return if else?  
These three versions are the same.  
1)  
if(A > B):  
 return A+1  
return A-1  
  
2)  
if(A > B):  
 return A+1  
else:  
 return A-1  
  
3)return A+1 if A > B else A-1  
  
'''  
  
'''  
Python整数不是固定的32位，所以需要做一些特殊的处理，  
代码里的将一个数对0x100000000取模（注意：Python的取模运算结果恒为非负数），  
是希望该数的二进制表示从第32位开始到更高的位都同是0（最低位是第0位），以在0-31位上模拟一个32位的int。  
  
python因为自动整数越界为long。用python要用0x10000000 (33位）做mask取模保持int。  
'''  
  
'''  
32位的int，正数的范围是(0,0x7FFFFFFF),负数(0x80000000,0xFFFFFFFF)  
'''  
  
  
'''  
# First iteration (a is 20, b is 20)  
10100 ^ 10100 == 00000 # makes a 0  
(10100 & 10100) << 1 == 101000 # makes b 40  
  
# Second iteration:  
000000 ^ 101000 == 101000 # Makes a 40  
(000000 & 101000) << 1 == 0000000 # Makes b 0  
  
  
What do the masks do?  
All the masks ensures that the value is an integer.  
Since the maximum possible int (32 bits) is 2147483647, if you add 2 to this value,  
like you did in your example,the int overflows and you get a negative value.  
You have to force this in Python, because it doesn't respect this int boundary as strongly  
typed languages like Java and C++ have defined.  
  
Consider the following:  
def get\_sum(a, b):  
 while b:  
 a, b = (a ^ b), (a & b) << 1 # without the masks  
 return a  
  
print get\_sum(2147483647, 2)  
outputs: 2147483649  
  
while  
print Solution().getSum(2147483647, 2)  
outputs: -2147483647  
  
due to the overflow.  
'''  
  
'''  
Python has more than 32 bits for integers. You can try to run "print 2 \*\* 31"  
Python would shows the exact number correctly,  
while other languages like Java would not. Java only recognizes -2 \*\* 31 to 2 \*\* 31 - 1.  
  
How does integers presented in Python differ from integers in 32-bit e.g. Java?  
From what I heard, Python has 64 bits. (Please let me know if I am wrong. )  
So 1 in Python would look like 0x0000000000000001, but it looks like 0x00000001 in 32-bit format.  
-1 in Python would look like 0xFFFFFFFFFFFFFFFF, but it looks like 0xFFFFFFFF in 32-bit format.  
  
It seems that the input given by LC is in 32-bit format.  
Since Python would treat it as positive with 1 on the 32 position,  
we have to use mask to treat it as negative.  
'''**

---------------------------------------------------------------------