1. 202-Happy Number

```
class Solution(object):
    def isHappy(self, n):
        """
        :type n: int
        :rtype: bool
        """
        set1=set()
        while n not in set1:
            set1.add(n)
            temp=0
            while n:
                n,val=divmod(n,10)
                temp+=val**2
            n=temp
        return True if n==1 else False
```

2. 204-Count Primes

```
class Solution:
    def countPrimes(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n<3:
            return 0
        primes=[1]*n
        primes[0]=primes[1]=0
        for i in range(2,int(n**0.5)+1):
            if primes[i]:
                primes[i*i:n:i]=[0]*((n-1)//i-i+1)
        return sum(primes)
```

3. 205-Isomorphic Strings

```
class Solution:
    def isIsomorphic(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        if len(s)!=len(t):
            return False
        dic={}
        set1=set()
        for i in range(len(s)):
            if s[i] in dic:
                if t[i]!=dic[s[i]]:
                    return False
            elif t[i] in set1:
                return False
            else:
                dic[s[i]]=t[i]
                set1.add(t[i])
        return True
```

4. 217-Contains Duplicate

```
class Solution:
    def containsDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        return len(nums)!=len(set(nums))
```

5. 219-Contains Duplicate II

```
class Solution:
    def containsNearbyDuplicate(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: bool
        """
        dic={}
        for i,n in enumerate(nums):
            if n in dic and i-dic[n]<=k:
                return True
            dic[n]=i
        return False
```

6. 231-Power of Two

```
class Solution(object):
    def isPowerOfTwo(self, n):
        """
        :type n: int
        :rtype: bool
        """
        if n==0:
            return False
        while n%2==0:
            n/=2
        return n==1
```

7. 242-Valid Anagram

```
class Solution:
    def isAnagram(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        cnt1=collections.Counter(s)
        cnt2=collections.Counter(t)
        return not (cnt1-cnt2) and not (cnt2-cnt1)
```

8. 258-Add Digits

```
class Solution:
    def addDigits(self, num):
        """
        :type num: int
        :rtype: int
        """
        return (num-1)%9+1 if num else 0
```

9. 263-Ugly Number

```
class Solution(object):
    def isUgly(self, num):
        """
        :type num: int
        :rtype: bool
        """
        if num<1:
            return False
        for i in [2,3,5]:
            while num%i==0:
                num/=i
        return num==1
```

10. 268-Missing Number

```
class Solution(object):
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l=len(nums)
        return l*(l+1)/2-sum(nums)
```

11. 283-Move Zeroes

```
class Solution:
    def moveZeroes(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place instead.
        """
        cnt=0
        for i in range(len(nums))[::-1]:
            if nums[i]==0:
                del nums[i]
                cnt+=1
        nums+=[0]*cnt
```

12. 290-Word Pattern

```
class Solution:
    def wordPattern(self, pattern, str):
        """
        :type pattern: str
        :type str: str
        :rtype: bool
        """
        frac=str.split()
        return len(set(zip(pattern,frac)))==len(set(pattern)) and
len(set(zip(pattern,frac)))==len(set(frac)) and len(frac)==len(pattern)
```

13. 292-Nim Game

```
class Solution(object):
    def canWinNim(self, n):
        """
        :type n: int
        :rtype: bool
        """
        return True if n%4 else False
```

14. 326-Power of Three

```
class Solution(object):
    def isPowerOfThree(self, n):
        """
        :type n: int
        :rtype: bool
        """
        return n>0 and 1162261467%n==0
```

15. 342-Power of Four

```
class Solution(object):
    def isPowerOfFour(self, num):
        """
        :type num: int
        :rtype: bool
        """
        tmp=1
        while tmp<num:
            tmp*=4
        return num==tmp
```

16. 344-Reverse String

```
class Solution(object):
    def reverseString(self, s):
        """
        :type s: str
        :rtype: str
        """
        return s[::-1]
```

17. 345-Reverse Vowels of a String

```
class Solution:
    def reverseVowels(self, s):
        """
        :type s: str
        :rtype: str
        """
        s=list(s)
        vowels=set("aeiouAEIOU")
        l=0
        r=len(s)-1
        while l<r:
            while s[l] not in vowels and l<r:
                l+=1
            while s[r] not in vowels and l<r:
                r-=1
            s[l],s[r]=s[r],s[l]
            l+=1
            r-=1
        return ''.join(s)
```

18. 349-Intersection of Two Arrays

```
class Solution:
    def intersection(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        return list(set(nums1)&set(nums2))
```

19. 350-Intersection of Two Arrays II

```
class Solution:
    def intersect(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        dic=collections.Counter(nums1)&collections.Counter(nums2)
        return [x for i in dic for x in [i]*dic[i]]
```

20. 367-Valid Perfect Square

```
class Solution(object):
    def isPerfectSquare(self, num):
        """
        :type num: int
        :rtype: bool
        """
        l,r=1,num
        while l<r:
            m=(l+r)/2
            if m**2<num:
                l=m+1
            else:
                r=m
        return l**2==num
```

21. 383-Ransom Note

```
class Solution:
    def canConstruct(self, ransomNote, magazine):
        """
        :type ransomNote: str
        :type magazine: str
        :rtype: bool
        """
        dic1=collections.Counter(ransomNote)
        dic2=collections.Counter(magazine)
        return not dic1-dic2
```

22. 387-First Unique Character in a String

```
class Solution:
    def firstUniqChar(self, s):
        """
        :type s: str
        :rtype: int
        """
        dic=collections.Counter(s)
        for i,c in enumerate(s):
            if dic[c]==1:
                return i
        return -1
```

23. 389-Find the Difference

```
class Solution(object):
    def findTheDifference(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: str
        """
        return list(collections.Counter(t)-collections.Counter(s))[0]
```

24. 400-Nth Digit

```
class Solution(object):
    def findNthDigit(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n<10:
            return n
        i,p=1,9
        while True:
            n+=p
            p=p*10+9
            i+=1
            if n<p*i:
                return int(str((n+i-1)//i)[(n+i-1)%i])
```

25. 409-Longest Palindrome

```
class Solution:
    def longestPalindrome(self, s):
        """
        :type s: str
        :rtype: int
        """
        dic=collections.Counter(s)
        ans=0
        mark=0
        for c in dic:
            if dic[c]%2==0:
                ans+=dic[c]
            else:
                mark=1
                ans+=(dic[c]-1)
        return ans+mark
```

26. 414-Third Maximum Number

```
class Solution:
    def thirdMax(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        ans=[float('-inf'),float('-inf'),float('-inf')]
        for num in nums:
            if num not in ans:
                if num>ans[0]:
                    ans=[num,ans[0],ans[1]]
                elif num>ans[1]:
                    ans[1:]=[num,ans[1]]
                elif num>ans[2]:
                    ans[2]=num
        return ans[2] if ans[2]!=float('-inf') else ans[0]
```

27. 434-Number of Segments in a String

```
class Solution:
    def countSegments(self, s):
        """
        :type s: str
        :rtype: int
        """
        return len(s.split())
```

28. 438-Find All Anagrams in a String

```
class Solution:
    def findAnagrams(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: List[int]
        """
        cnt1=[0]*26
        cnt2=[0]*26
        l=len(p)
        for c in p:
            cnt1[ord(c)-97]+=1
        for c in s[:l-1]:
            cnt2[ord(c)-97]+=1
        ans=[]
        for i in range(len(s)-l+1):
            cnt2[ord(s[i+l-1])-97]+=1
            if cnt1==cnt2:
                ans.append(i)
            cnt2[ord(s[i])-97]-=1
        return ans
```

29. 441-Arranging Coins

```
class Solution(object):
    def arrangeCoins(self, n):
        """
        :type n: int
        :rtype: int
        """
        return int(math.sqrt(n*2+0.25)-0.5)
```

30. 443-String Compression

```
class Solution:
    def compress(self, chars):
        """
        :type chars: List[str]
        :rtype: int
        """
        cur=chars[0]
        cnt=0
        ans=[]
        for c in chars+[' ']:
            if c==cur:
                cnt+=1
            else:
                ans+=[cur]
                if cnt!=1:
                    ans+=list(str(cnt))
                cnt=1
                cur=c
        chars[:]=ans
        return len(chars)
```

31. 448-Find All Numbers Disappeared in an Array

```
class Solution:
    def findDisappearedNumbers(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        return list(set(range(1,len(nums)+1))-set(nums))
```

32. 453-Minimum Moves to Equal Array Elements

```
class Solution(object):
    def minMoves(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        return sum(nums)-min(nums)*len(nums)
```

33. 455-Assign Cookies

```
class Solution:
    def findContentChildren(self, g, s):
        """
        :type g: List[int]
        :type s: List[int]
        :rtype: int
        """
        g.sort()
        s.sort()
        l=len(g)
        ans=0
        for x in s:
            if ans<l and x>=g[ans]:
                ans+=1
        return ans
```

34. 459-Repeated Substring Pattern

```
class Solution:
    def repeatedSubstringPattern(self, s):
        """
        :type s: str
        :rtype: bool
        """
        return s in (s+s)[1:-1]
```

35. 461-Hamming Distance

```
class Solution:
    def hammingDistance(self, x, y):
        """
        :type x: int
        :type y: int
        :rtype: int
        """
        return bin(x^y).count('1')
```

36. 475-Heaters

```
class Solution:
    def findRadius(self, houses, heaters):
        """
        :type houses: List[int]
        :type heaters: List[int]
        :rtype: int
        """
        ans=0
        heaters=[-math.inf]+sorted(heaters)+[math.inf]
        houses.sort()
        i=0
        for house in houses:
            while house>heaters[i]:
                i+=1
            ans=max(ans,min(heaters[i]-house,house-heaters[i-1]))
        return ans
```

37. 476-Number Complement

```
class Solution:
    def findComplement(self, num):
        """
        :type num: int
        :rtype: int
        """
        return pow(2,num.bit_length())-num-1
```

38. 479-Largest Palindrome Product

```
class Solution:
    def largestPalindrome(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n==1:
            return 9
        for a in range(2,10**n):
            left=10**n-a
            right=int(str(left)[::-1])
            if a**2-4*right>=0:
                x=a-(a**2-4*right)**0.5
                if x//2==x/2:
                    return (10**n*left+right)%1337
```

39. 482-License Key Formatting

```
class Solution:
    def licenseKeyFormatting(self, S, K):
        """
        :type S: str
        :type K: int
        :rtype: str
        """
        S=S.replace('-','').upper()[::-1]
        ans=[]
        for i in range(0,len(S),K):
            ans.append(S[i:i+K])
        return '-'.join(ans)[::-1]
```

40. 485-Max Consecutive Ones

```
class Solution:
    def findMaxConsecutiveOnes(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        ans=0
        tmp=0
        nums.append(0)
        for num in nums:
            if num:
                tmp+=1
            else:
                ans=max(ans,tmp)
                tmp=0
        return ans
```

41. 496-Next Greater Element I

```
class Solution:
    def nextGreaterElement(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        dic={}
        stack=[]
        for i in nums2:
            while stack and stack[-1] < i:
                dic[stack.pop()] = i
            stack.append(i)
        return [dic.get(i, -1) for i in nums1]
```

42. 507-Perfect Number

```
class Solution(object):
    def checkPerfectNumber(self, num):
        """
        :type num: int
        :rtype: bool
        """
        return num in [6,28,496,8128,33550336]
```

43. 628-Maximum Product of Three Numbers

```
class Solution(object):
    def maximumProduct(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l,s=heapq.nlargest(3,nums),heapq.nsmallest(2,nums)
        x,y=l[1]*l[2],s[0]*s[1]
        ans=max(x,y)*l[0]
        return ans
```

44. 633-Sum of Square Numbers

```
class Solution(object):
    def judgeSquareSum(self, c):
        """
        :type c: int
        :rtype: bool
        """
        l=0
        r=int(c**0.5)
        while l<=r:
            t=l*l+r*r
            if t==c:
                return True
            elif t<c:
                l+=1
            else:
                r-=1
        return False
```

45. 672-Bulb Switcher II

```
class Solution:
    def flipLights(self, n, m):
        """
        :type n: int
        :type m: int
        :rtype: int
        """
        if m==0 or n==0:
            return 1
        if n==1:
            return 2
        if n==2:
            if m==1:
                return 3
            return 4
        if n>=3:
            if m==1:
                return 4
            if m==2:
                return 7
        return 8
```

46. 728-Self Dividing Numbers

```
class Solution(object):
    def selfDividingNumbers(self, left, right):
        """
        :type left: int
        :type right: int
        :rtype: List[int]
        """
        ans=[]
        for x in range(left,right+1):
            y=x
            while y:
                val=y%10
                if val==0 or x%(val):
                    break
                y//=10
            if not y:
                ans.append(x)
        return ans
```

47. 754-Reach a Number

```
class Solution:
    def reachNumber(self, target):
        """
        :type target: int
        :rtype: int
        """
        target=abs(target)
        x=math.ceil((2*target+0.25)**0.5-0.5)
        return x if (x*(x+1)/2-target)%2==0 else x+1+x%2
```

48. 877-Stone Game

```
class Solution:
    def stoneGame(self, piles):
        """
        :type piles: List[int]
        :rtype: bool
        """
        return True
```

49. 914-X of a Kind in a Deck of Cards

```
class Solution:
    def hasGroupsSizeX(self, deck):
        """
        :type deck: List[int]
        :rtype: bool
        """
        dic=collections.defaultdict(int)
        for i in deck:
            dic[i]+=1
        x=dic[deck[0]]
        for i in dic.values():
            while i%x!=0:
                i,x=x,i%x
        return x!=1
```

50. 942-DI String Match

```
class Solution:
    def diStringMatch(self, S):
        """
        :type S: str
        :rtype: List[int]
        """
        l=0
        r=len(S)
        ans=[]
        for c in S:
            if c=='I':
                ans.append(l)
                l+=1
            else:
                ans.append(r)
                r-=1
        ans.append(l)
        return ans
```

51. 7- Reverse Integer

```
    class Solution(object):
        def reverse(self, x):
            """
            :type x: int
            :rtype: int
            """
            if x < 0:
                    x = str(x)[1:]
                    y = -int(str(x)[::-1])
            else:
                    y = int(str(x)[::-1])
            if y < -2 ** 31 or y > 2 ** 31 - 1:
                    return 0
            else:
            return y
```

52. 9-Palindrome Number

```
class Solution:
        def isPalindrome(self, x):
                if str(x) == str(x)[::-1]:
                        return True
                else:
                return False
```

53. 11-Container With Most Water

```
class Solution:
        def maxArea(self, height):
                begin = 0
                end = len(height)-1
                ans = 0
                while begin < end:
                        ans = max(ans,(end-begin) * min(height[begin],height[end]))
                        if height[begin] > height[end]:
                                end -= 1
                        else:
                                begin += 1
        return ans
```

54. 12-Integer to Roman

```
class Solution:
        def intToRoman(self, num):
                unit = ['','I','II','III','IV','V','VI','VII','VIII','IX']
                ten = ['','X','XX','XXX','XL','L','LX','LXX','LXXX','XC']
                hundred = ['','C','CC','CCC','CD','D','DC','DCC','DCCC','CM']
                thousand = ['','M','MM','MMM']


                d = [unit,ten,hundred,thousand]
                num = str(num)[::-1]
                ans = []
                for i in range(len(num)):
                        ans.append(d[i][int(num[i])])
        return "".join(ans[::-1])
```

55. 14-Longest Common Prefix

```
class Solution:
    def longestCommonPrefix(self, strs):
        if len(strs) == 0:
            return ''
        for i in range(len(strs[0])):
            letter = strs[0][i]
            for string in strs:
                if len(string) <= i or string[i] != letter:
                    return strs[0][:i]

    return strs[0]
```

56. 26-Remove Duplicates from Sorted Array

```
class Solution:
    def removeDuplicates(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums) == 0:
            return 0
        cnt = 1
        mark = nums[0]
        for i in range(1,len(nums)):
            if nums[i] != mark:
                nums[cnt] = nums[i]
                mark = nums[i]
                cnt += 1
    return cnt
```

57. 27-Remove Element

```
class Solution:
    def removeElement(self, nums, val):
        """
        :type nums: List[int]
        :type val: int
        :rtype: int
        """
        cnt = 0
        for i in range(len(nums)):
            if nums[i] != val:
                    nums[cnt] = nums[i]
                    cnt += 1
        return cnt
```

58. 28-Implement strStr()

```
class Solution:
    def strStr(self, haystack, needle):
        if needle:
            return haystack.find(needle)
        else:
        return 0
```

59. 34-Find First and Last Position of Element in Sorted Array

```
class Solution:
    # returns leftmost (or rightmost) index at which `target` should be inserted in sorted
    # array `nums` via binary search.
    def extreme_insertion_index(self, nums, target, left):
        lo = 0
        hi = len(nums)
        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] > target or (left and target == nums[mid]):
                hi = mid
            else:
                lo = mid+1
        return lo
    def searchRange(self, nums, target):
        left_idx = self.extreme_insertion_index(nums, target, True)
        # assert that `left_idx` is within the array bounds and that `target`
        # is actually in `nums`.
        if left_idx == len(nums) or nums[left_idx] != target:
            return [-1, -1]
    return [left_idx, self.extreme_insertion_index(nums, target, False)-1]
```

60. 35-Search Insert Position

```
class Solution:
    def searchInsert(self, nums, target):
        if nums[-1] < target:
            return len(nums)
        for i in range(len(nums)):
            if nums[i] == target or nums[i] > target:
            return i
```

61. 50-Pow(x, n)

```
class Solution:
    def myPow(self, x, n):
        """
        :type x: float
        :type n: int
        :rtype: float
        """
        if n < 0:
            x = 1./x
            n = -n
        power = 1
        while n:
            if n & 1:
                power *= x
            x *= x
            n >>= 1
    return power
```

62. 58-Length of Last Word

```
class Solution:
    def lengthOfLastWord(self, s):
        s = s.rstrip()
        if not s:
            return 0
        index = len(s) - 1
        cnt = 0
        while index >= 0 and s[index] != ' ':
            cnt += 1
            index -= 1
    return cnt
```

63. 63-Unique Paths II

```
class Solution:
    def uniquePathsWithObstacles(self, obstacleGrid):
        """
        :type obstacleGrid: List[List[int]]
        :rtype: int
        """
        if obstacleGrid[0][0] == 1:
            return 0
        width,length = len(obstacleGrid), len(obstacleGrid[0])
        dp = [0 for j in range(length+1)]
        dp[1] = 1
        for i in range(1,width+1):
```

```
                    for j in range(1,length+1):
                        if i==1 and j==1:
                            continue
                        if obstacleGrid[i-1][j-1] == 0:
                            dp[j] = dp[j] + dp[j-1]
                        else:
                            dp[j] = 0
                return dp[length]
```

64. 64-Minimum Path Sum

```
class Solution:
    def minPathSum(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        # get the rows number
        m = len(grid)

        # get the columns number
        n = len(grid[0])

        # calculate the 1st column
        for i in range(1,m):
            grid[i][0] = grid[i][0] + grid[i-1][0]

        # calculate the 1st row
        for i in range(1,n):
            grid[0][i] = grid[0][i] + grid[0][i-1]

        for i in range(1, m):
            for j in range(1, n):
                grid[i][j] += grid[i-1][j] if grid[i-1][j]<grid[i][j-1] else grid[i][j-1]
        return grid[m-1][n-1]
```

65. 66-Plus One

```
class Solution(object):
    def plusOne(self, digits):
        """
        :type digits: List[int]
        :rtype: List[int]
        """
        pos = len(digits) - 1
```

```
                while True:
                    temp = digits[pos] + 1
                    if temp > 9:
                        digits[pos] = temp % 10
                        pos -= 1
                        if pos == -1:
                            digits.insert(0,1)
                            return digits
                    else:
                        digits[pos] = temp
                    return digits
```

66. 67-Add Binary

```
class Solution(object):
    def addBinary(self, a, b):
        """
        :type a: str
        :type b: str
        :rtype: str
        """
        return bin(int(a,2) + int(b,2))[2:]
```

67. 69-Sqrt(x)

```
class Solution(object):
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        return int(math.sqrt(x))
```

68. 70-Climbing Stairs

```
    class Solution:
        def climbStairs(self, n):
            """
            :type n: int
            :rtype: int
            """
            res = [i for i in range(n+1)]
            for i in range(3, n+1):
                res[i] = res[i-1] + res[i-2]
        return res[n]
```

69. 88-Merge Sorted Array

```
class Solution:
    def merge(self, nums1, m, nums2, n):
        end_pos = m + n - 1
        while m > 0 and n > 0:
            if nums1[m-1] > nums2[n-1]:
                nums1[m+n-1] = nums1[m-1]
                m -= 1
            else:
                nums1[m+n-1] = nums2[n-1]
                n -= 1
        if n > 0:
            nums1[:n] = nums2[:n]
```

70. 121-Best Time to Buy and Sell Stock

```
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if prices == []:
            return 0
        ans = 0
        min_num = prices[0]
        for i in range(1,len(prices)):
            min_num = min(prices[i],min_num)
            ans = max(ans,prices[i]-min_num)
        return ans
```

71. 125-Valid Palindrome

```
    class Solution(object):
        def isPalindrome(self, s):
            """
            :type s: str
            :rtype: bool
            """
            temp = []
            for i in s:
                if i.isalnum():
                    temp.append(i.lower())
        return temp == temp[::-1]
```

72. 136-Single Number

```
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        res = 0
        for i in nums:
            res ^= i
        return res

class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        return reduce(lambda x,y:x^y , nums)
```

73. 137-Single Number II

```
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        d = dict()
        for i in nums:
            if i not in d:
                d[i] = 1
            else:
                d[i] += 1
        for item in d:
            if d[item] == 1:
                return item

class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        return (sum(set(nums)) *    3 - sum(nums)) // 2
```

74. 167-Two Sum II - Input array is sorted

```
class Solution(object):
    def twoSum(self, numbers, target):
        """
        :type numbers: List[int]
        :type target: int
        :rtype: List[int]
        """
        d = dict(zip(numbers,range(1,len(numbers)+1)))
        for i in numbers:
            if target-i in d:
                if target-i == i:
                    return [d[i]-1,d[i]]
                return [d[i],d[target-i]]
```

75. 169-Majority Element

```
class Solution(object):
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        maj = nums[0]
        cnt = 1
        for i in nums[1:]:
            if i == maj:
                cnt += 1
            else:
                if cnt == 0:
                    maj = i
                    cnt = 1
                else:
                    cnt -= 1
        return maj
```

76. 172-Factorial Trailing Zeroes

```
class Solution(object):
    def trailingZeroes(self, n):
        """
        :type n: int
        :rtype: int
        """
        cnt = 0
        while n >= 5:
            n -= (n % 5)
            n /= 5
            cnt += n
        return cnt
```

77. 189-Rotate Array

```
class Solution(object):
    def reverseOrder(self,nums,begin,end):
        while begin < end:
            nums[begin],nums[end] = nums[end],nums[begin]
            begin += 1
            end -=1
    def rotate(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: void Do not return anything, modify nums in-place instead.
        """
        k %= len(nums)
        self.reverseOrder(nums,0,len(nums)-k-1)
        self.reverseOrder(nums,len(nums)-k,len(nums)-1)
        self.reverseOrder(nums,0,len(nums)-1)
```

78. 190-Reverse Bits

```
class Solution:
    # @param n, an integer
    # @return an integer
    def reverseBits(self, n):
        temp = ("0" * 32 + bin(n)[2:])[-32:][::-1]
        return int(temp,2)

class Solution:
    # @param n, an integer
    # @return an integer
    def reverseBits(self, n):
        temp = bin(n)[2:].zfill(32)[::-1]
        return int(temp,2)
```

79. 191-Number of 1 Bits

```
class Solution(object):
    def hammingWeight(self, n):
        """
        :type n: int
        :rtype: int
        """
        return bin(n).count('1')
```

80. 746-Min Cost Climbing Stairs

```
class Solution(object):
    def minCostClimbingStairs(self, cost):
        """
        :type cost: List[int]
        :rtype: int
        """
        if len(cost) == 0 or len(cost) == 1:
            return 1
        a, b = cost[0], cost[1]
        for index in range(2, len(cost)):
            a, b = b, cost[index] + min(a, b)
        return min(a,b)
```

81. 867-Transpose Matrix

```
class Solution(object):
    def transpose(self, A):
        R, C = len(A), len(A[0])
        ans = [[None] * R for _ in xrange(C)]
        for r, row in enumerate(A):
            for c, val in enumerate(row):
                ans[c][r] = val
        return ans
```

82. 868-Binary Gap

```
class Solution(object):
    def binaryGap(self, N):
        A = [i for i in xrange(32) if (N >> i) & 1]
        if len(A) < 2: return 0
        return max(A[i+1] - A[i] for i in xrange(len(A) - 1))
```

83. 884-Uncommon Words from Two Sentences

```
class Solution(object):
    def uncommonFromSentences(self, A, B):
        count = {}
        for word in A.split():
            count[word] = count.get(word, 0) + 1
        for word in B.split():
            count[word] = count.get(word, 0) + 1

        #Alternatively:
        #count = collections.Counter(A.split())
        #count += collections.Counter(B.split())

        return [word for word in count if count[word] == 1]
```

84. 888-Fair Candy Swap

```
class Solution(object):
    def fairCandySwap(self, A, B):
        Sa, Sb = sum(A), sum(B)
        setB = set(B)
        for x in A:
            if x + (Sb - Sa) / 2 in setB:
                return [x, x + (Sb - Sa) / 2]
```

85. 893-Groups of Special-Equivalent Strings

```
class Solution(object):
    def numSpecialEquivGroups(self, A):
        def count(A):
            ans = [0] * 52
            for i, letter in enumerate(A):
                ans[ord(letter) - ord('a') + 26 * (i%2)] += 1
            return tuple(ans)
        return len({count(word) for word in A})
```

86. 896-Monotonic Array

```
class Solution(object):
    def isMonotonic(self, A):
        return (all(A[i] <= A[i+1] for i in xrange(len(A) - 1)) or
                all(A[i] >= A[i+1] for i in xrange(len(A) - 1)))
```

87. 905-Sort Array By Parity

```
class Solution(object):
    def sortArrayByParity(self, A):
        A.sort(key = lambda x: x % 2)
        return A
```

88. 908-Smallest Range I

```
class Solution(object):
    def smallestRangeI(self, A, K):
        return max(0, max(A) - min(A) - 2*K)
```

89. 914-X of a Kind in a Deck of Cards

```
class Solution(object):
    def hasGroupsSizeX(self, deck):
        from fractions import gcd
        vals = collections.Counter(deck).values()
        return reduce(gcd, vals) >= 2
```

90. 917-Reverse Only Letters

```
class Solution(object):
    def reverseOnlyLetters(self, S):
        letters = [c for c in S if c.isalpha()]
        ans = []
        for c in S:
            if c.isalpha():
                ans.append(letters.pop())
            else:
                ans.append(c)
        return "".join(ans)
```

91. 921-Minimum Add to Make Parentheses Valid

```
class Solution(object):
    def minAddToMakeValid(self, S):
        ans = bal = 0
        for symbol in S:
            bal += 1 if symbol == '(' else -1
            # It is guaranteed bal >= -1
            if bal == -1:
                ans += 1
                bal += 1
        return ans + bal
```

92. 922-Sort Array By Parity II

```
class Solution(object):
    def sortArrayByParityII(self, A):
        N = len(A)
        ans = [None] * N
        t = 0
        for i, x in enumerate(A):
            if x % 2 == 0:
                ans[t] = x
                t += 2
        t = 1
        for i, x in enumerate(A):
            if x % 2 == 1:
                ans[t] = x
                t += 2
        # We could have also used slice assignment:
        # ans[::2] = (x for x in A if x % 2 == 0)
        # ans[1::2] = (x for x in A if x % 2 == 1)
        return ans
```

93. 925-Long Pressed Name

```
class Solution(object):
    def isLongPressedName(self, name, typed):
        g1 = [(k, len(list(grp))) for k, grp in itertools.groupby(name)]
        g2 = [(k, len(list(grp))) for k, grp in itertools.groupby(typed)]
        if len(g1) != len(g2):
            return False
        return all(k1 == k2 and v1 <= v2
                   for (k1,v1), (k2,v2) in zip(g1, g2))
```

94. 932-Beautiful Array

```
class Solution:
    def beautifulArray(self, N):
        memo = {1: [1]}
        def f(N):
            if N not in memo:
                odds = f((N+1)/2)
                evens = f(N/2)
                memo[N] = [2*x-1 for x in odds] + [2*x for x in evens]
            return memo[N]
        return f(N)
```

95. 933-Number of Recent Calls

```
class RecentCounter(object):
    def __init__(self):
        self.q = collections.deque()

    def ping(self, t):
        self.q.append(t)
        while self.q[0] < t-3000:
            self.q.popleft()
        return len(self.q)
```

96. 937-Reorder Log Files

```
class Solution(object):
    def reorderLogFiles(self, logs):
        def f(log):
            id_, rest = log.split(" ", 1)
            return (0, rest, id_) if rest[0].isalpha() else (1,)
        return sorted(logs, key = f)
```

97. 939-Minimum Area Rectangle

```
class Solution(object):
    def minAreaRect(self, points):
        columns = collections.defaultdict(list)
        for x, y in points:
            columns[x].append(y)
        lastx = {}
        ans = float('inf')
        for x in sorted(columns):
            column = columns[x]
            column.sort()
            for j, y2 in enumerate(column):
                for i in xrange(j):
                    y1 = column[i]
                    if (y1, y2) in lastx:
                        ans = min(ans, (x - lastx[y1,y2]) * (y2 - y1))
                    lastx[y1, y2] = x
        return ans if ans < float('inf') else 0
```

98. 941-Valid Mountain Array

```
class Solution(object):
    def validMountainArray(self, A):
        N = len(A)
        i = 0
        # walk up
        while i+1 < N and A[i] < A[i+1]:
            i += 1
        # peak can't be first or last
        if i == 0 or i == N-1:
            return False
        # walk down
        while i+1 < N and A[i] > A[i+1]:
            i += 1
        return i == N-1
```

99. 942-DI String Match

```
class Solution(object):
    def diStringMatch(self, S):
        lo, hi = 0, len(S)
        ans = []
        for x in S:
            if x == 'I':
                ans.append(lo)
                lo += 1
            else:
                ans.append(hi)
                hi -= 1
        return ans + [lo]
```

100. 944-Delete Columns to Make Sorted

```
class Solution(object):
    def minDeletionSize(self, A):
        ans = 0
        for col in zip(*A):
            if any(col[i] > col[i+1] for i in xrange(len(col) - 1)):
                ans += 1
        return ans
```