给定一个整数数组（下标从 0 到 n-1， n 表示整个数组的规模），请找出该数组中的最长上升连续子序列。（最长上升连续子序列可以定义为从右到左或从左到右的序列。）

 注意事项

time

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给定 [5, 4, 2, 1, 3], 其最长上升连续子序列（LICS）为 [5, 4, 2, 1], 返回 4.

给定 [5, 1, 2, 3, 4], 其最长上升连续子序列（LICS）为 [1, 2, 3, 4], 返回 4.

思路：比较要是和前一次比较相同就继续加，否则就等于2，在来一个数来记录最大值，这样，只要判断是否和前一次比较相同即可。这样大于并且pre=1，小于pre=0否则就回二。。

class Solution:

# @param {int[]} A an array of Integer

# @return {int} an integer

def longestIncreasingContinuousSubsequence(self, A):

# Write your code here

'''

#耗时1022ms

length=len(A)

if length==0:

return 0

if length==1:

return 1

a=[]

for i in range(length-1):

if A[i]>A[i+1]:

a.append(1)

else:

a.append(0)

b=[]

max1=2

if len(a)==1:

return 2

for i in range(len(a)-1):

if a[i]==a[i+1]:

max1+=1

b.append(max1)

else:

max1=2

return int(max(b))

'''

#第二种方法耗时816ms

length=len(A)

if length==0:

return 0

if length==1:

return 1

a=[]

t=1

if A[0]>=A[1]:

pre=1

else:

pre=0

max1=0

for i in range(length-1):

if A[i]>=A[i+1] and pre==1:

pre=1

t+=1

elif A[i]<A[i+1] and pre==0:

pre=0

t+=1

elif A[i]<A[i+1] and pre==1:

t=2

pre=0

else:

t=2

pre=1

if t>max1:

max1=t

return max1

2.

给出一个包含大小写字母的字符串。求出由这些字母构成的最长的回文串的长度是多少。

数据是大小写敏感的，也就是说，"Aa" 并不会被认为是一个回文串。

 注意事项

假设字符串的长度不会超过 1010。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出 s = "abccccdd" 返回 7

一种可以构建出来的最长回文串方案是 "dccaccd"。

思路：比较简单，只要知道字符串中字符的个数，这样把偶数的留下来，再把大于一的激素拿来，减一个相加最后加一即可。

class Solution:

# @param {string} s a string which consists of lowercase or uppercase letters

# @return {int} the length of the longest palindromes that can be built

def longestPalindrome(self, s):

# Write your code here

length=len(s)

dict={}

nums=0

for i in range(length):

dict[s[i]]=dict.get(s[i],0)+1

if len(dict)==1:

for key in dict:

nums=dict[key]

return nums

flag=True

for i in dict.keys():

if dict[i]%2==0:

nums+=dict[i]

elif dict[i]>=1:

nums+=dict[i]-1

flag=False

if flag==False:

nums+=1

return nums

3. 给一个词典，找出其中所有最长的单词。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

在词典

{

"dog",

"google",

"facebook",

"internationalization",

"blabla"

}

中, 最长的单词集合为 ["internationalization"]

在词典

{

"like",

"love",

"hate",

"yes"

}

中，最长的单词集合为 ["like", "love", "hate"]

思路：这个比较简单也没什么坑，就是统计个数，然后排序，找到最大几个变成list即可，不过能找到最大值就不用排序，应该会快很多。

class Solution:

# @param dictionary: a list of strings

# @return: a list of strings

def longestWords(self, dictionary):

# write your code here

max=-1

t=[]

for i in dictionary :

if len(i)>=max:

max=len(i)

t.append(i)

temp=[]

for j in range(len(t)):

if len(t[j])==max:

temp.append(t[j])

return temp

4. 给定一个整型数组，找出主元素，它在数组中的出现次数严格大于数组元素个数的二分之一。

 注意事项

You may assume that the array is non-empty and the majority number always exist in the array.

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出数组**[1,1,1,1,2,2,2]**，返回 1

思路：统计字符出现的个数，排序，找到大于1/2的点然后跳出。当然要是本身有序就不用这么处理。

class Solution:

"""

@param nums: A list of integers

@return: The majority number

"""

def majorityNumber(self, nums):

# write your code here

count=0

for i in range(len(nums)):

if count==0:

a=nums[i]

count=1

else:

if a==nums[i]:

count=count+1

else:

count=count-1

if count>(len(nums)-i)/2:

return a

5. 给你一个包含 *m* x *n* 个元素的矩阵 (*m* 行, *n* 列), 求该矩阵的之字型遍历。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

对于如下矩阵：

[

[1, 2, 3, 4],

[5, 6, 7, 8],

[9,10, 11, 12]

]

返回 [1, 2, 5, 9, 6, 3, 4, 7, 10, 11, 8, 12]

思路：所谓的之字型遍历，就是副对角线遍历。问题就是边界和顺序，激素次遍历是往上偶数次是往下。上下边界是右左右边界是下最后走完n个

class Solution:

# @param: a matrix of integers

# @return: a list of integers

def printZMatrix(self, matrix):

# write your code here

'''

#292ms

row=len(matrix)

colum=len(matrix[0])

a=[matrix[0][0]]

b=[[0 for i in range(colum)] for j in range(row)]

i=0;j=0

t=1

while(1):

while i+1<row and j-1>=0 and b[i+1][j-1]==0:

i+=1

j-=1

a.append(matrix[i][j])

b[i][j]=1

while i-1>=0 and j+1<colum and b[i-1][j+1]==0:

i-=1

j+=1

a.append(matrix[i][j])

b[i][j]=1

if i%2 == 0 :

if j+1<colum:

if b[i][j+1]==0:

j+=1

a.append(matrix[i][j])

b[i][j]=1

elif i+1<row:

if b[i+1][j]==0:

i+=1

a.append(matrix[i][j])

b[i][j]=1

elif i%2 != 0:

if i+1<row:

if b[i+1][j]==0:

i+=1

a.append(matrix[i][j])

b[i][j]=1

elif j+1<colum:

if b[i][j+1]==0:

j+=1

a.append(matrix[i][j])

b[i][j]=1

if i+1>=row and j+1>=colum:

return a

break

'''

#289ms

row=len(matrix)

colum=len(matrix[0])

t=1

temp=[matrix[0][0]]

i=0;j=0

while len(temp)<row\*colum:

if t%2!=0:

while i-1>=0 and j+1<colum:

i-=1

j+=1

temp.append(matrix[i][j])

if i==0:

if j==colum-1:

i+=1

else:

j+=1

elif j==colum-1:

if i==row-1:

temp.append(matrix[i][j])

return temp

else:

i+=1

temp.append(matrix[i][j])

t+=1

else:

while i+1<row and j-1>=0:

i+=1

j-=1

temp.append(matrix[i][j])

if j==0:

if i==row-1:

j+=1

else:

i+=1

elif i==row-1:

if j==colum-1:

temp.append(matrix[i][j])

return temp

else:

j+=1

temp.append(matrix[i][j])

t+=1

return temp

6. 给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的距离。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出一棵如下的二叉树:

1

/ \

2 3

/ \

4 5

这个二叉树的最大深度为3.

思路：递归解，只要求得每个左右节点最大值即可

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left, self.right = None, None

"""

class Solution:

"""

@param root: The root of binary tree.

@return: An integer

"""

def maxDepth(self, root):

# write your code here

if root == None:

return 0;

lDepth = Solution.maxDepth(self , root.left);

rDepth = Solution.maxDepth(self , root.right);

return max(lDepth , rDepth) + 1

7. 给定一个整数数组，找到一个具有最大和的子数组，返回其最大和。

 注意事项

子数组最少包含一个数

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出数组[−2,2,−3,4,−1,2,1,−5,3]，符合要求的子数组为[4,−1,2,1]，其最大和为6

思路：怎么找到最大子数组的和，那么可以把子数组的和表示为二维数组aij，这样ij就可以表示为i到j的数组状态转移方程在哪里，想来就2中一种是头动，一种是尾动。Aij=max（aij-1，aij-1+nums【j】），并标记最大值，这种算法复杂度为o（n2）。自己觉告诉我们要是和小于0则可以舍弃前面的。这样i就自己动了。算法复杂度是o(n)

class Solution:

"""

@param nums: A list of integers

@return: An integer denote the sum of maximum subarray

"""

def maxSubArray(self, nums):

# write your code here

length=len(nums)

max=-8888

sum=0

if length==1:

return nums[0]

for i in range(length):

sum+=nums[i]

if sum>max:

max=sum

if sum<0:

sum=0

return max

8.

给定一个未排序的整数数组，找到其中位数。

中位数是排序后数组的中间值，如果数组的个数是偶数个，则返回排序后数组的第N/2个数。

**样例**

给出数组**[4, 5, 1, 2, 3]**， 返回 3

给出数组**[7, 9, 4, 5]**，返回 5

最简单的是直接排序，然后找，还有就是二分法查找中位数，复杂度要低些。最好的是线性时间查找。

def Partition(a, low,high, x):

i=low

high-=1

while (a[i]!= x ) :

i+=1

a[low],a[i]=a[i],a[low]

while (low < high):

while low < high and a[high] >= x:

high-=1

a[low] = a[high]

while low < high and a[low] <x:

low+=1

a[high] = a[low]

a[low] = x

return low

def S\_sort(a,low,high):

n = high - low

if n==1:

return a[low]

remind=n%5

for i in range(0,n//5):

quick\_sort(a, i\*5+low,low+i \* 5 + 4)

a[low+i],a[low+i \* 5 + 2]=a[low+i \* 5 + 2], a[low+i]

h=n//5

if remind !=0:

quick\_sort(a, high-remind,high-1)

a[low+h],a[high-remind+(remind+1)//2-1]=a[high-remind+(remind+1)//2-1], a[low+h]

x = S\_sort(a, low,low+h+1)

else:

x = S\_sort(a, low,low+h)

return x

def Select(a, low,high, k):

x=S\_sort(a, low, high)

j= Partition(a, low, high, x)

q = j - low + 1

if (q == k) :

return x

elif (q>k):

return Select(a, low, j , k)

else :

return Select(a, j+1, high, k - q)

def quick\_sort(lists, left, right):

# 快速排序

if left >= right:

return lists

key = lists[left]

least = left

max1 = right

while left < right:

while left < right and lists[right] >= key:

right -= 1

lists[left] = lists[right]

while left < right and lists[left] <= key:

left += 1

lists[right] = lists[left]

lists[right] = key

quick\_sort(lists,least, left - 1)

quick\_sort(lists, left + 1, max1)

return lists

a = [ 8, 4,0, -89, -12, 1, 36, 789, 21, 54,2,6,5,9,39,24,0]

t= Select(a, 0,len(a),10)

print(t)

9.

给出若干闭合区间，合并所有重叠的部分。

**样例**

给出的区间列表 => 合并后的区间列表：

[ [

[1, 3], [1, 6],

[2, 6], => [8, 10],

[8, 10], [15, 18]

[15, 18] ]

]

思路应该比较简单，就是比较两区间，分为有重叠和和没重叠，没重叠的部分直接入栈，有重叠的部分比较如同插入样，比较断点值选择小小大大的区间合并策略。当然也可以选择不增加额外空间。代码不写了

10. 合并两个排序的整数数组A和B变成一个新的数组。

 注意事项

你可以假设A具有足够的空间（A数组的大小大于或等于m+n）去添加B中的元素。

**样例**

给出 A = [1, 2, 3, empty, empty], B = [4, 5]

合并之后 A 将变成 [1,2,3,4,5]

思路：就像两幅扑克牌一样大小，分别拿出来比较。复杂度是o（m+n）别的也没啥好办法。必须在A中插入还是比较烦的，那么插入一个就要后移一个，这样算法复杂度很高。如果直接利用原来的空间即按照大的排列这样就避免的移位操作。

1. **class** Solution {
2. **public**:
3. /\*\*
4. \* @param A: sorted integer array A which has m elements,
5. \*           but size of A is m+n
6. \* @param B: sorted integer array B which has n elements
7. \* @return: void
8. \*/
9. **void** mergeSortedArray(**int** A[], **int** m, **int** B[], **int** n) {
10. **int** k=m+n-1;
11. **int** i=m-1;
12. **int** j=n-1;
13. **while**(i>=0 && j>=0)
14. {
15. **if**(A[i]>B[j])
16. A[k--]=A[i--];
17. **else**
18. A[k--]=B[j--];
19. }
20. **while**(j>=0)
21. {
22. A[k--]=B[j--];
23. }
24. }
25. };