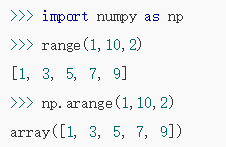
**1. Two sum问题（题号1，easy）**

**（1）range & arange**

range(start, end, step)，返回一个list对象，起始值为start，终止值为end，但不含终止值，步长为step。只能创建int型list。

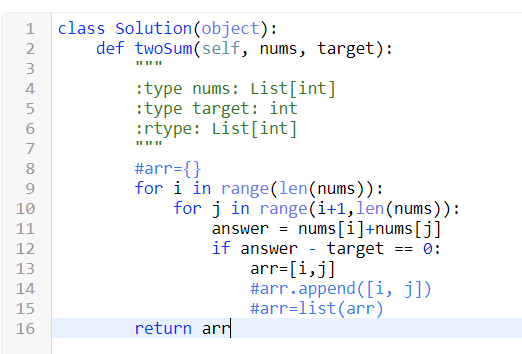
arange(start, end, step)，与range()类似，但是返回一个array对象。需要引入import numpy as np，并且arange可以使用float型数据。



**（2）我的方案**

思路：用每一个数字和其后边所有数字以此相加，找到目标值，返回此时的索引号

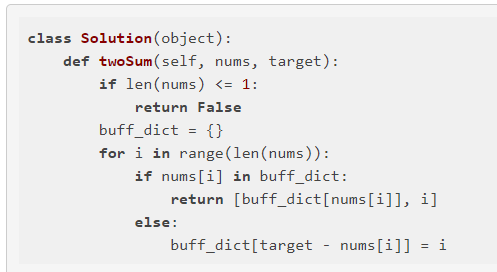
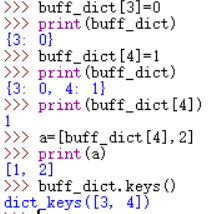
考虑两点：一是不能和自身相加，因此j+1，二是可能有多种结果，所以append;



遗留问题：若用append结果是[[]]形式，怎么把二维数组转一维，但若不考虑重复，可以按上述来写；

运行时间：32ms

**（3）题中方案：**

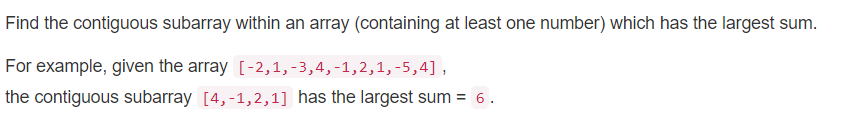
时间：26ms

Nums=[3,2,4], target = 6;

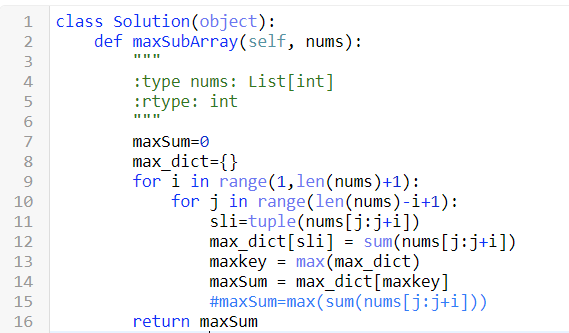
理解：在一开始啥都没有时候创建一个字典，就是说我第一个数和目标差3，那么我就把差的3和我本来3的索引（0）做成一个字典，当后边出现3时，字典就可以查到这个3，然后告诉他前边有个索引是0的3在等它，然后返回这两个3的索引；以此类推，2进来时，告诉他他和目标差4，那么当后边遇到4时，就告诉他前边有个2在等它，把他俩的坐标返回即可，这种方法的好处是不用逐个尝试，前边有记录谁缺什么

**2. Maximum subarray问题（题号53，easy**）

问题：求连续的最大和：

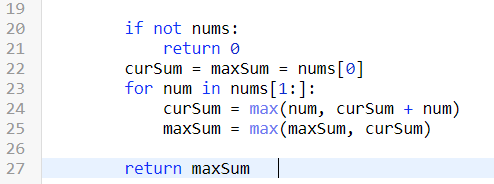


1. 初始思路：创建一个字典，使连续子列表作为键，其和作为值，max这个字典，即可得到其最大值对应的键，但是行不通，因为列表是不可哈希的，因此不可以作为键，大多数的python对象都可以为键，但是列表和字典都是不可哈希的，不能为键，因此可以改为元组



错误：理解问题：输出最大值即可，不需要子列表，因此14行又取了一遍字典里的值，但是结果仍有问题，此时结果为后六个数之和5，i,j有问题

1. 题中方案：

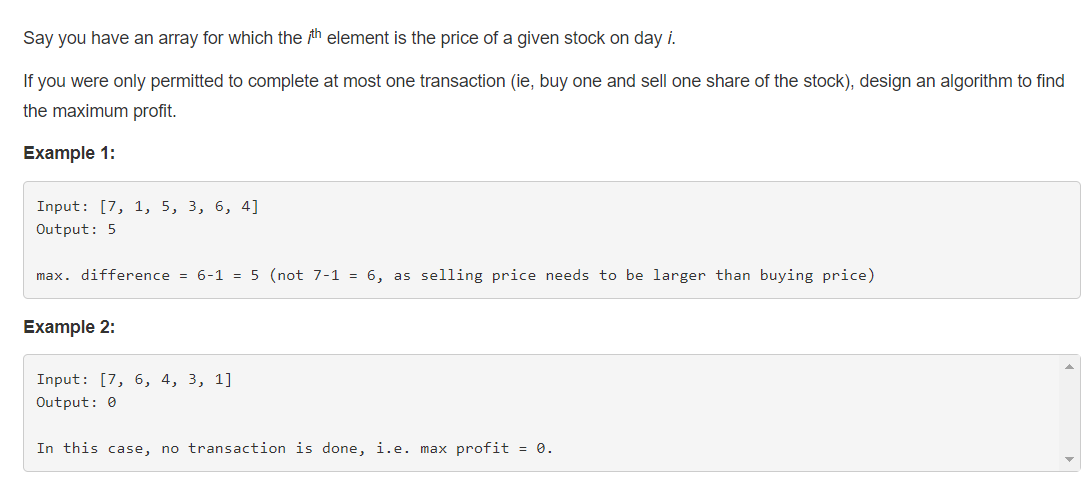


这个思路就是相当于把前两个数先相加，结果与第二个数比，若比它小，则相当于拖后腿，放弃第一个数，从第二个数开始往后加，若比第三个数大，则保留这两个的结果，接着往后加第四个数，结果若比第四个数小则放弃前三个，从第四个数开始加，也就是说从前往后依次相加，每次加两个数，把和与后一个数相比，若大则留作为新的起点，若小则弃，并使后一个数变成新的起点，再将新的数与后一个相加，同理，同时把新的数与和的最大值保存下来，以确定到哪里停止，直至加到最后一个数。

这个方法的优点也在于不用逐个相加，而是先分析数据间的关系，始终排除前边拖后腿的数，并在收集过程中的最值，以便取出，不能不取最值一直加到最后，那样和值有可能变小。

**3. Best Time to Buy and Sell Stock问题（题号121，easy）**

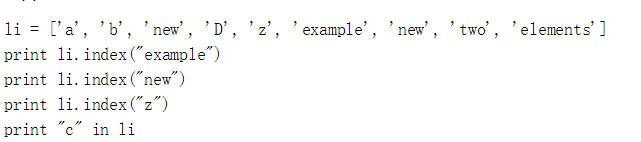
题目：给定一组股票的价格，只有一次交易（一次买入和卖出），设法使获利最大，注意买入一定在卖出之前

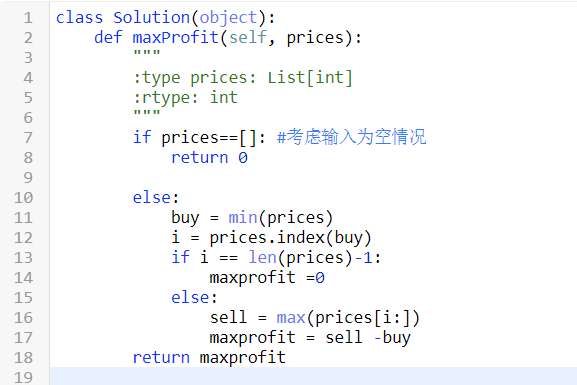


1. 我的方案：

思路：先找最小值，然后返回他的索引，然后找它之后的最大值，二者相减，若之后无最大值（或者说无值），返回为0，

返回数组的索引号用的是index

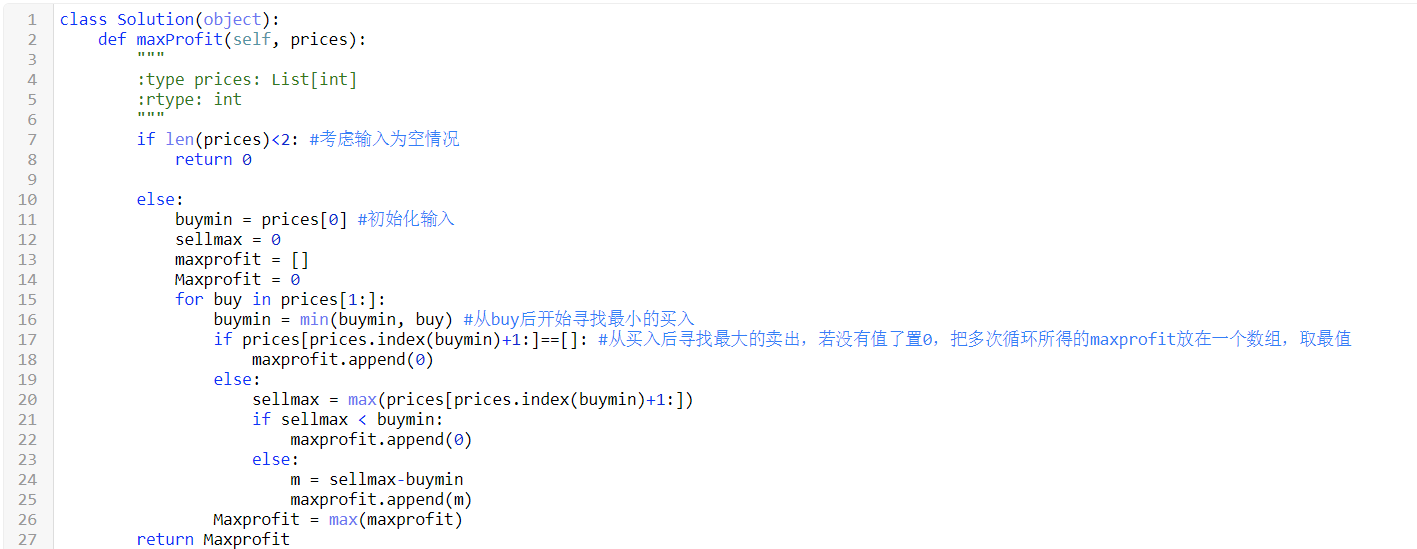




这个思路**有误**，因为不应该找最小值，而是只要买了之后有更高的价格，均可以卖出。如输入[2,4,1]，最小值1后边没有值了，但是可以2买入，4卖出

**因此需要改**：

思路：在循环内，先确定一个买入的最小值，然后从其后找最大值，若没有值了返回为0，若有最大值则计算最大利润 ；在循环外，往后继续找最小值，若找到新的最小值，再从其后找最大值，将循环中所得的最大值不断补充到利润数组里，最后取出数组中的最大值即可

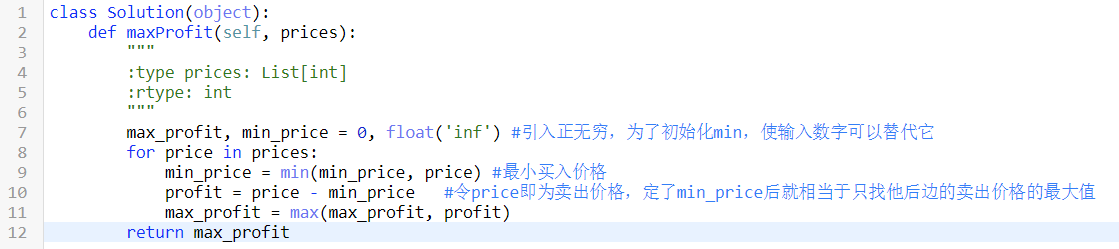


问题：超时26ms

1. 更改版：

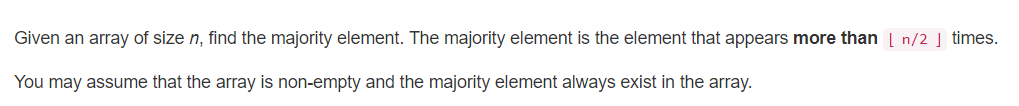
思路：前一部分与我思路一样，先给定一个最小值，但是巧妙之处在于他用了一个正无穷也就是inf来初始化最小值，这样第一个输入很容易就把它代替了，然后用prices数组的每一个数以此作为卖出价格，这样的好处在于不用考虑初始数据是不是空，或者是不是只有一个，买入和卖出的初值都是有的，并且如这种情况相减能保证为0；之后在循环中用price来做卖出值，然后及时取最大更新利润，就不用利润数组的出现了

与我的主要区别在于第10行的加入，用price当卖出，当最小买入定了的时候相当于用price卖出来以此与之作差，因此相当于自动把握了时间顺序去其后找卖出，同时还会更新新的最小值，使步骤简化了许多，值得学习。

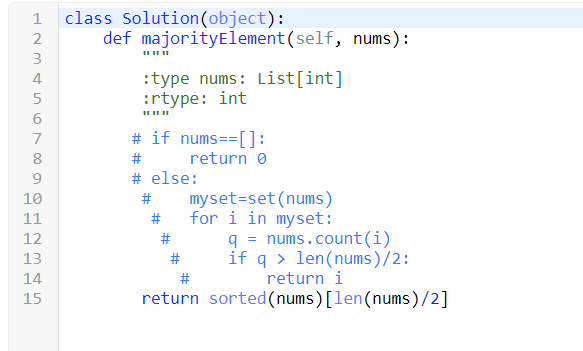


**4. Majority Element（题号169，easy）**

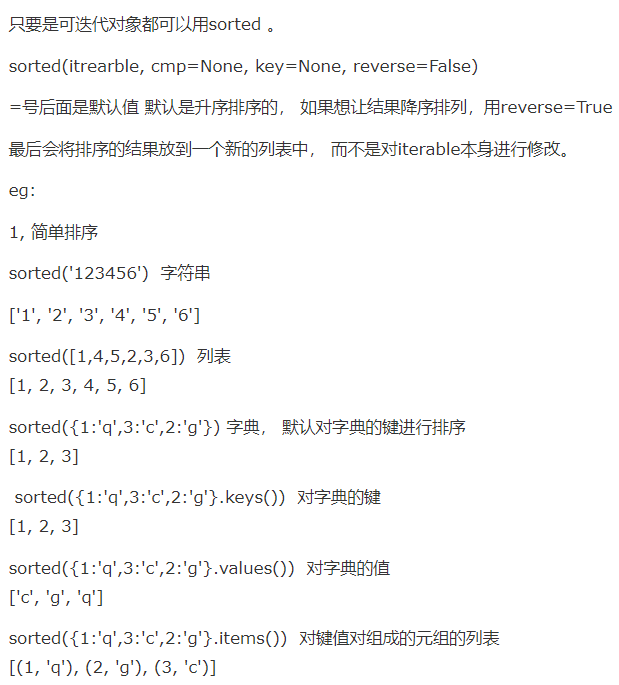
找出数组中出现数目大于一半的元素

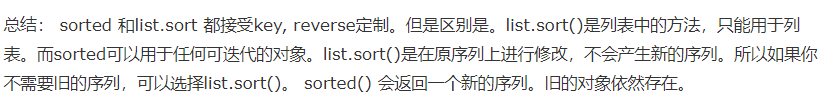


1. 我的思路：用set把非重复元素找出来，再用count逐个计数，选出大于半数的元素
2. 别人思路：用sorted把数组排序，输出其中位数（排序后的中位数一定过半了）

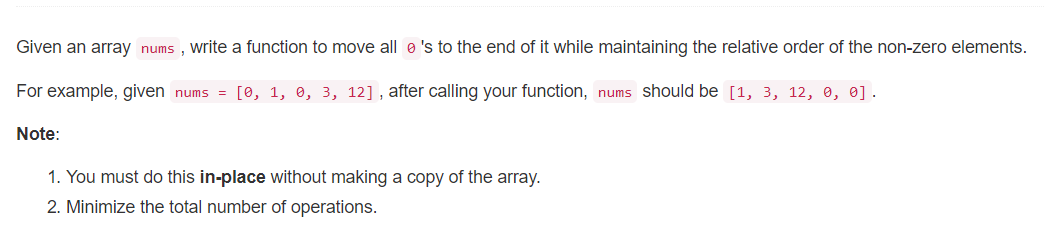


**Sorted()与list.sort()的用法**



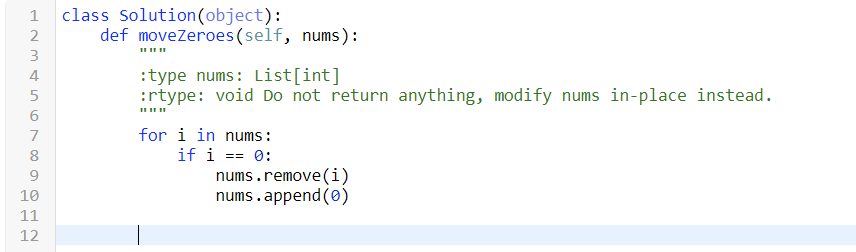


**5. Move Zeros（题号283，easy）**



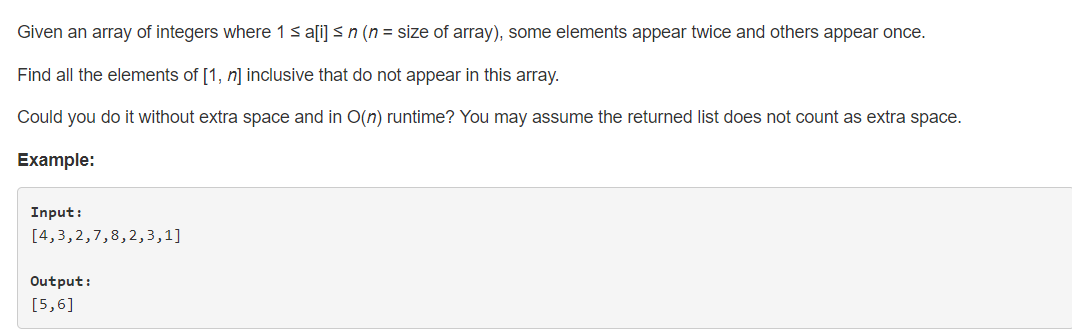
题目：把数组中的0移到后边，非零元素顺序保持不变

思路：逐个检测元素，是0删掉，后边增加0，非零不管

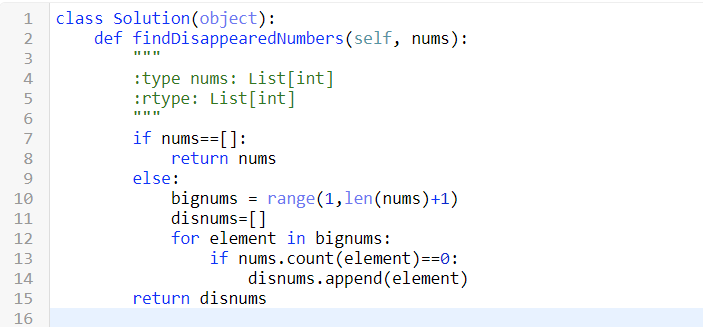


**6. Find All Numbers Disappeared in an Array（题号448，easy）**

题目：找到数组中消失的元素，注意n是数组长度



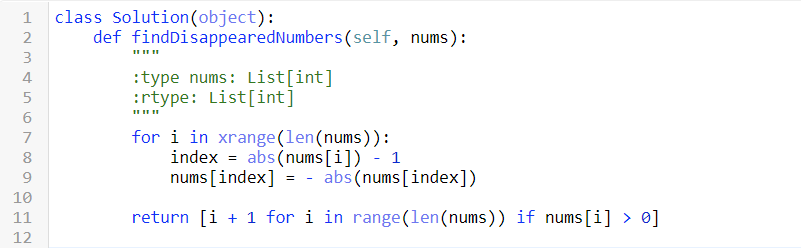
思路：构建一个大的数组，即1到n的，把原数组当做小的数组，然后在小的数组中计数大数组的每一个元素，要是计数为0，则说明确实，把这个放在缺失元素的数组里



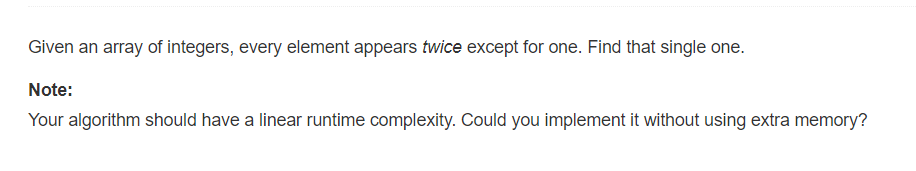
问题：超时

解决的方案：

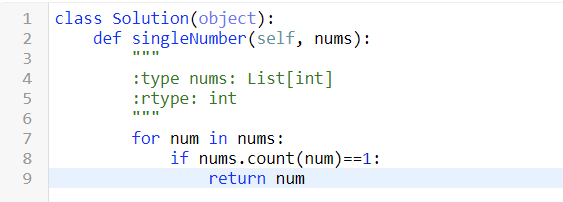
第一次遍历：用相反数的形式标记出现过的数字（下标）；第二次遍历：找出没有被标记的数字（下标），并返回



**7. 找出单个数字（题号136，easy）**



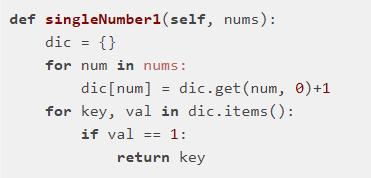
我的思路：遍历数组，计数每个元素的个数，若为1则输出该元素



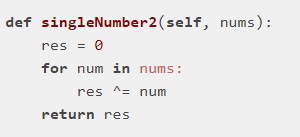
问题：超时

解决方案：

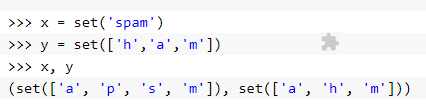
法一：建立字典，把个数作为值，用get函数，get函数返回字典中要查找的值，若不存在时返回默认值



法二：按位异或



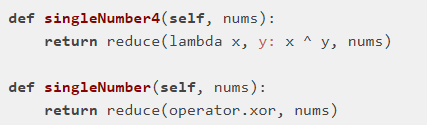
法三：用set删除数组中的重复元素



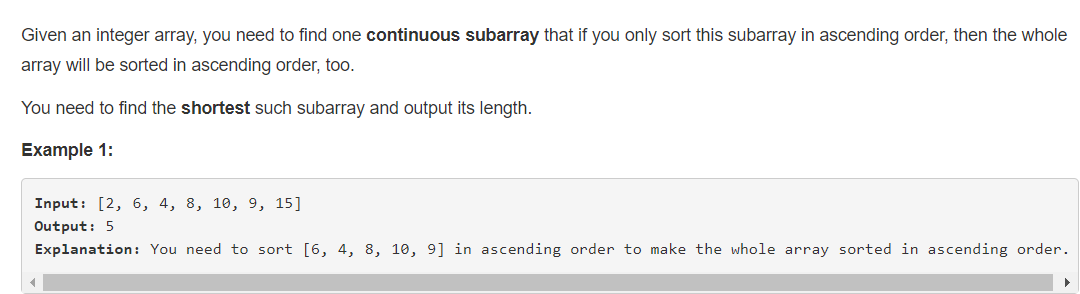


法四：reduce函数：





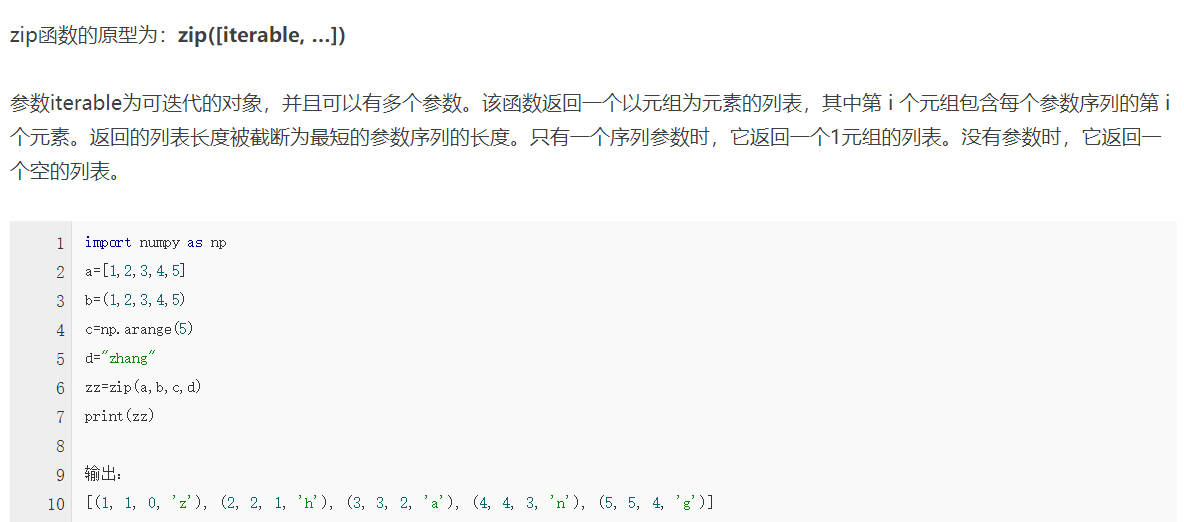
**8. 最短乱序子集（题号581，easy）**

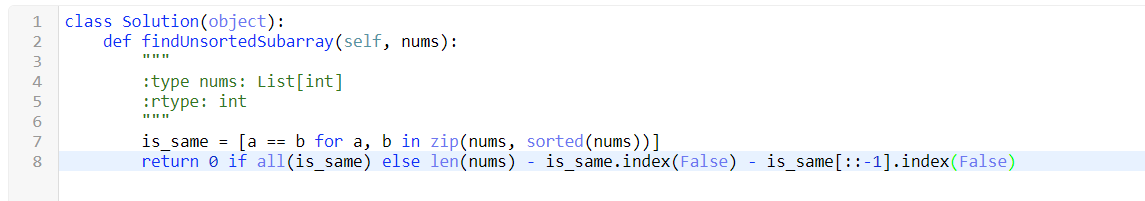


题目：找出数组中乱序的最短子集

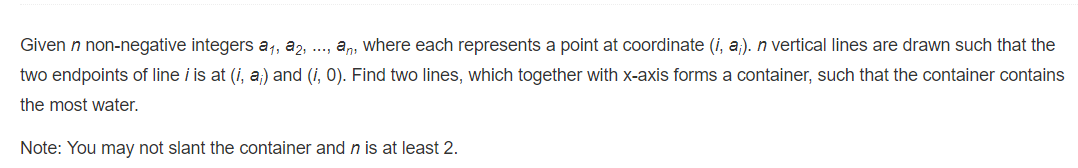
参考方案：

把乱序和正序的每个元素组成一个元素对（zip），判断它俩是否相等（a==b），相等返回True，不等返回False，最短乱序的子集就是在第一个和倒数第一个（[::-1]）之间的元素，因此用总长度减去前后的长度，即可得中间乱序的长度



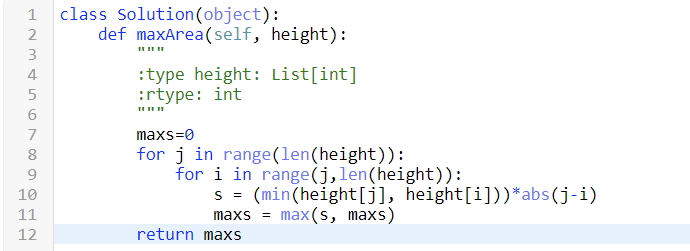


**9. 容器含水最多（题号11，medium）**



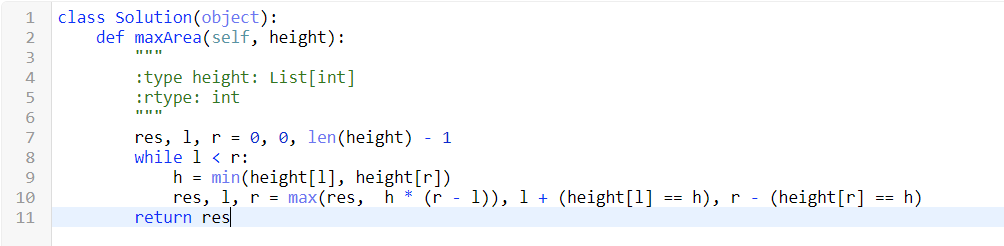
数组元素与标号组成对画在坐标系里，任取两根垂直与x轴的线，求与x轴围成的最大面积

我的思路：定一根线，变另一根线，求面积，找最值



问题：超时；

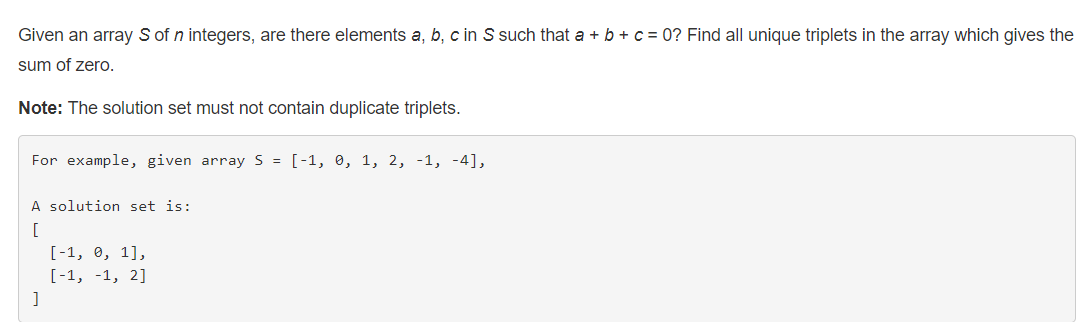
解决方案：



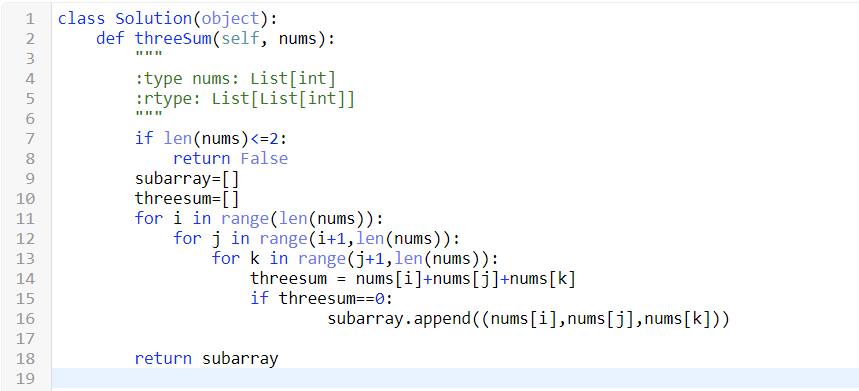
思路与我一致，但是亮点在于，既然是面积那么线的前后顺序无关，也就是说i和j与j和i表示了同一块面积，因此可以把l 和r一个从前往后，一个从后往前更新，只要两个没走岔了，就都有面积，然后找出最大即可，时间省了一半

**10. 3Sum（题号15，medium）**

题目：找出数组中求和为0的三个元素，并且要求加和情况唯一，即每个子集合元素不重复

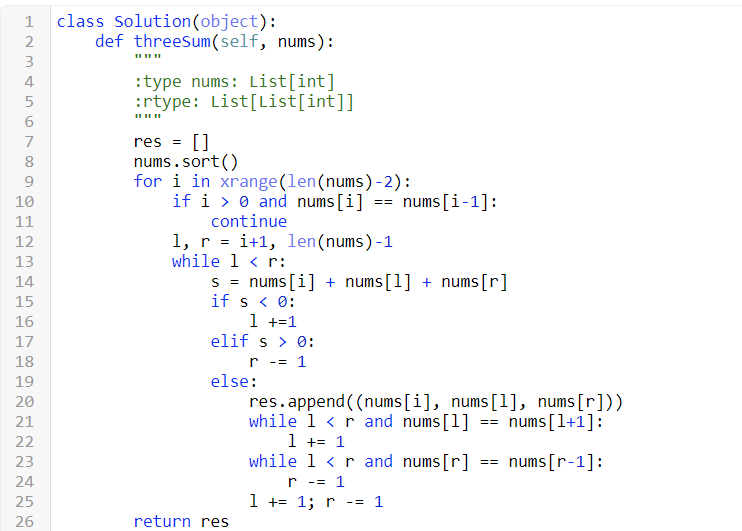


我的思路：所有三个元素加和，找出为0的情况，但是并没有解决元素唯一问题



解决方案：

为了简化运算，先给数组排序，这样前边都是小于0的，后边是大于0的，前两个数从前边开始，最后一个数从后往前，直到第二个数与第三个数相遇为止。巧妙之处有二，一是排序后若加和大于零，则往小调节第三个数，若小于0则往大调节第二个数，不用盲目求和；二是解决重复问题，因为排序，若有相同的就在一起了，因此第二个数与第三个数移动的过程中，遇到重复的跳过即可，就能够避免相同的情况

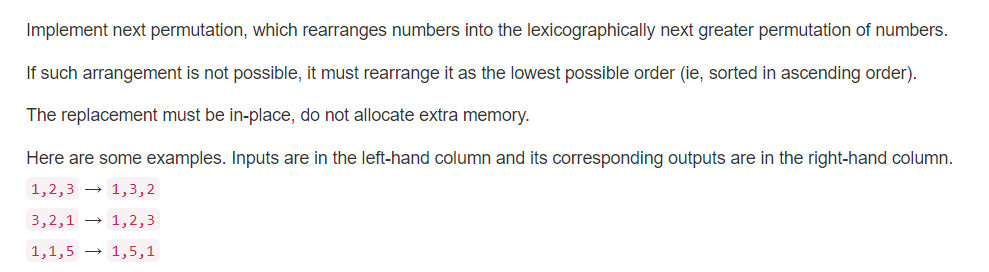


里边xrange需要说明：



**11. Next Permutation（题号31，medium）**

题目：求下一个排列比当前更大的排列，若没有这样的排列，则给它升序排列



题意理解：eg:123-132-213-231-312-321(将123进行排列组合，保证下一个比前一个大，“下一个函数”需要输入排列组合，输出下一个)，我理解的是只排列排序时的第一步

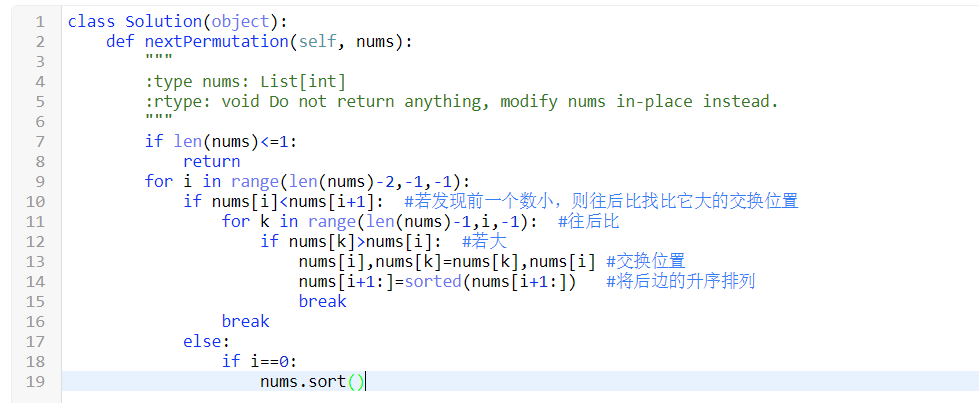
思路：

（1）由于下一个数比上一个数大，因此需要从后往前扫描，找到递增的位置设为元素i,j，(i小于j)

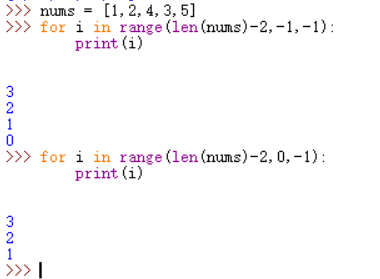
（2）由于下一个数虽然大于上一个数，且最接近上一个数，因此找到元素i，在i元素后面找到最接近i且大于i的元素k。由于i后面的元素都是降序排列的，只需要从后往前扫描找到第一个比i大的元素即可

（3）找到将i和k交换位置，然后将k后面的元素递增排序

（4）找不到，则将全部元素递增排序

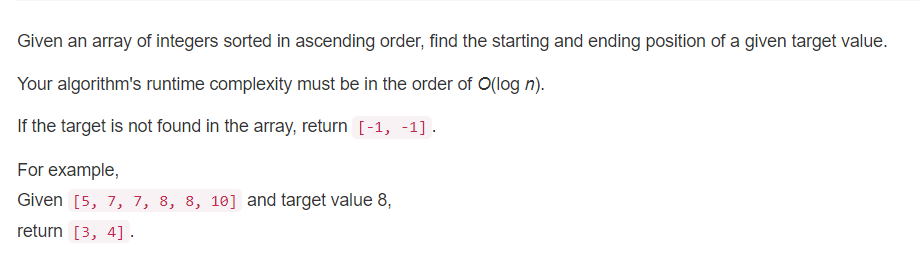


再补充一点range的用法，它有三个参数，第一个是起点，第二个是终点，第三个是步长，相当于起点一步一步加步长，最后到终点为止得到这么一列数，如果倒着取，那个起点就是原终点，终点是原起点（但这时原起点不会被包含，所以取-1，相当于原起点的左边），步长是-1

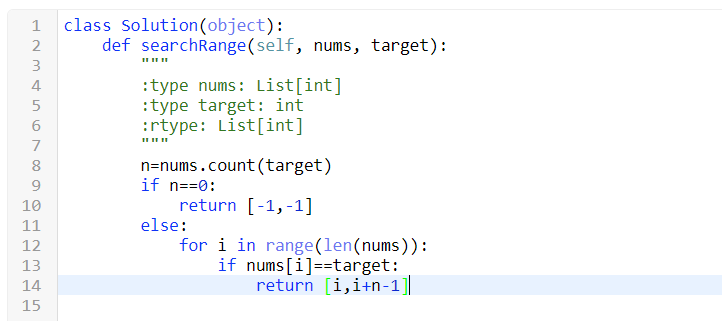


**12. Search for a Range（题号34，medium）**

题目：在一个升序排列的数组中，寻找目标元素的起点和终点索引，若没有这个元素，返回[-1,-1]

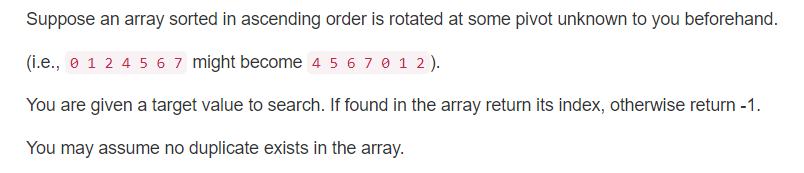


思路：先计数目标元素个数，若为0直接返回[-1,-1]，若不为0，则寻找其起点，然后加上n-1则为终点



**13. Search in Rotated Sorted Array（题号33，medium）**

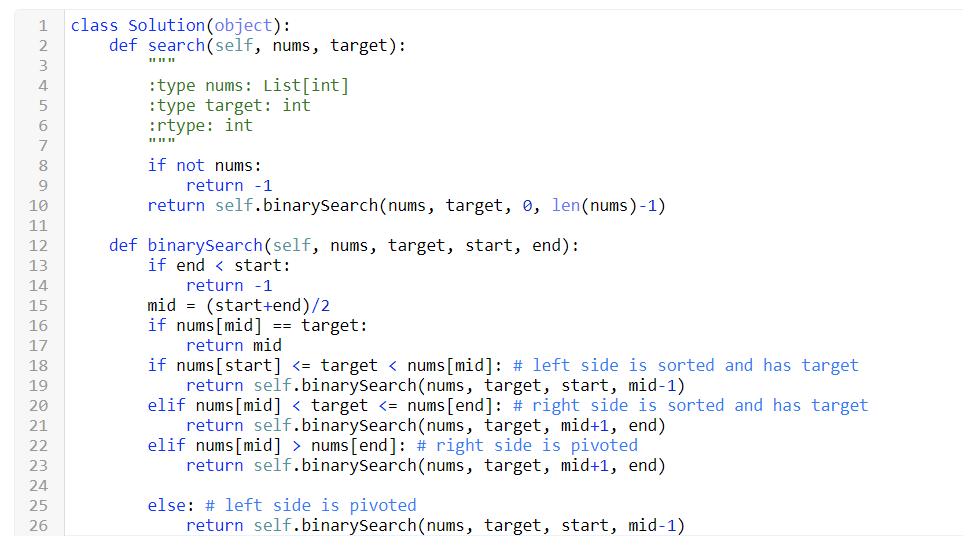
题目：在一个部分旋转的数组中找到目标元素



理解意图：由于数组部分旋转，因此不能用一般的二分法查找，但是每一部分是升序排列的，因此小部分内可以用二分法，这个题的亮点在于二分法的迭代以及分情况讨论：

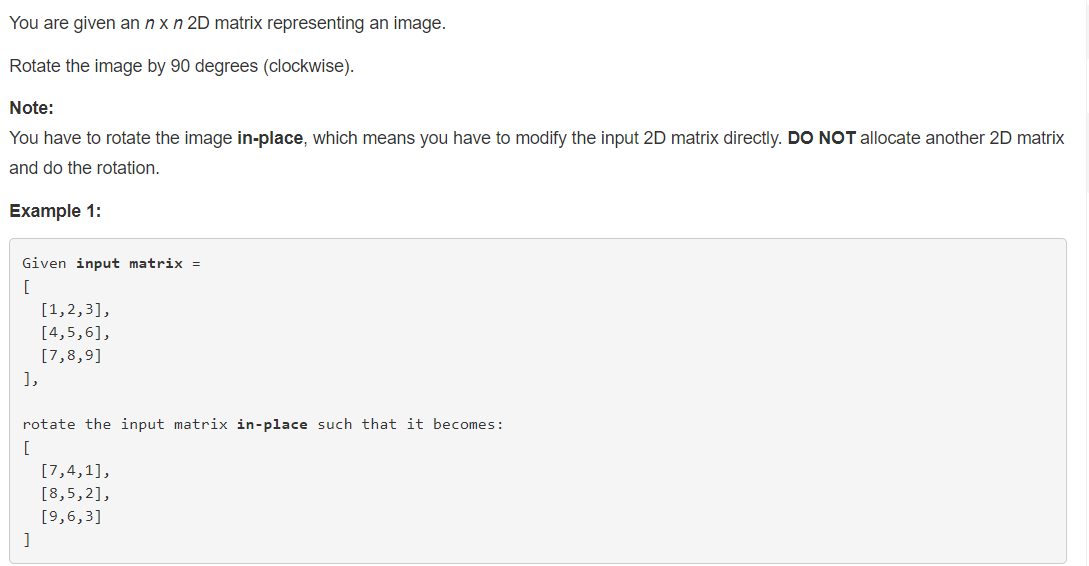
输入三个元素：起点，终点以及目标，然后求终点，如果：

1. 目标在起点与中点之间，则在左半部分继续找；
2. 目标在中点与终点之间，则在右半部分继续找；
3. 若都不符合两者情况，那么当中点大于终点时，右侧旋转了，并且由于不符合情况（1），因此往右接着找；
4. 若不符合上述三种，则左侧旋转了，往左接着找



**14. Rotate Image（题号48，medium）**

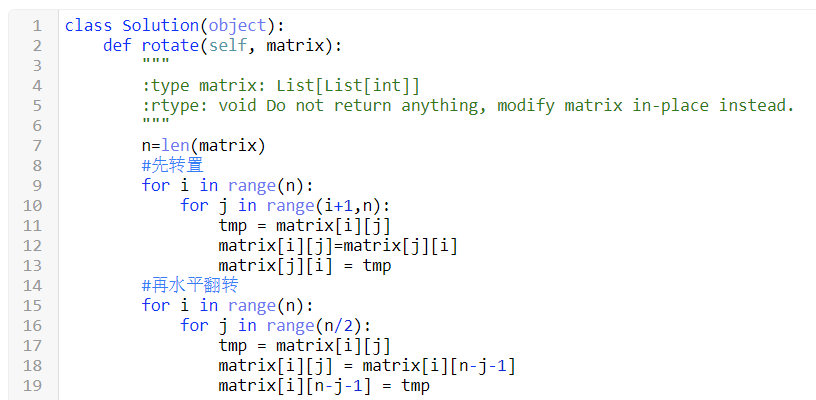
题目：将矩阵顺时针旋转90度



思路：先转置，在水平翻转

1. 转置就是两个维度索引对调
2. 水平翻转就是第一个维度（行，i）不变，第二个维度两个对称位置的元素对调

求对称的位置时，j最大到n/2，因为相当于只有一半的索引在变换，总长减去一半的即可，注意n-j-1的使用



15.