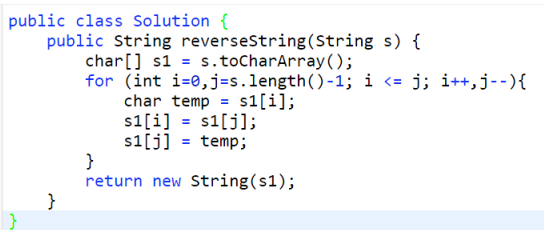
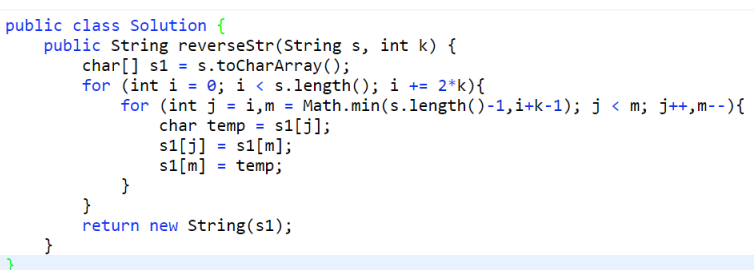
String 题

1. reverse string
2. swap method(two pointer method)； 
3. 用StringBuilder的reverse function；(**return** **new** StringBuilder(s).reverse().toString();)

541. Reverse String II

精髓在于那个Math.min

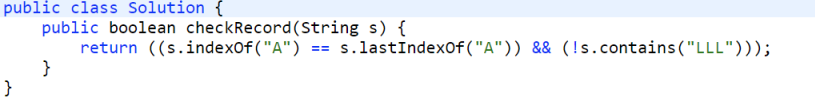
1. Reverse Words in a String[III](https://leetcode.com/problems/reverse-words-in-a-string-iii/)

依旧是swap method，要注意的是java中并没有Python中那样的“”“”.join()函数来合并字符串数组，所以这题不需要用String.split()

1. implement strStr()

Haystack.contains(needle) 相当于Python中的 while needle in haystack

1. Student Attendance Record I



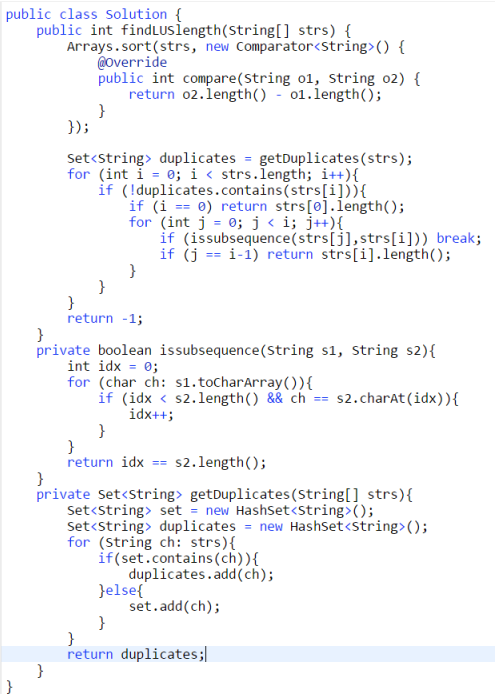
用到了indexOf和lastIndexOf来做一个cheating的计数功能

### Longest Uncommon Subsequence I

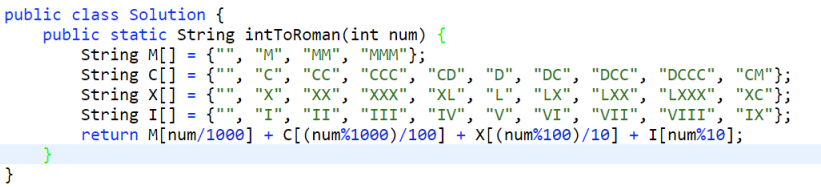
差点就被题目骗了，longest uncommon subsequence其实就是在求两个不同字符串的最大长度，相同则return -1

1. Longest Uncommon Subsequence II
2. 先进行arraysort按长度倒序排序；2、写出判断duplicates和subsequence的子函数；3、11~16行的代码的功能（代码在下一页）
3. Roman to Integer

做的时候查一下罗马数字表，只有前一位对应的值比后一位对应的值小的时候需要减去，其他全为加。



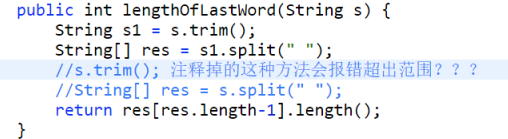
1. Integer to Roman

这种做法说实话没啥意思。。。However，cannot find better method.

1. Reverse Vowels of a String

标准的two pointers method；可以用vowels.indexOf(res[l]) == -1 来代替vowels.contains(res[l]+””), 可以避免进行string comparison.

1. Length of Last Word
2. 先把字符串开头和结尾的空格用s.trim()删除；2、split(“ ”)函数将字符串分开成多个子字符串并读取最后一个子字符串的长度

这里有一个问题待处理：

1. Longest Common Prefix

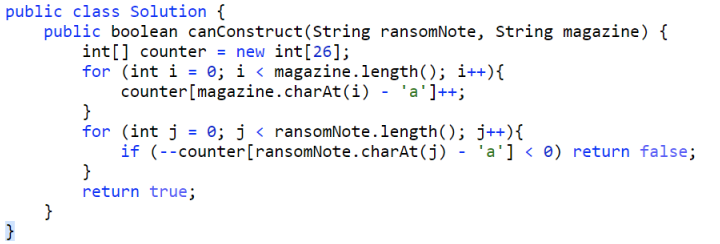
先对字符串数组排序，排序后就只需要比较第一个和最后一个这2个字符串即可！

感觉java中对字符串的处理先用toCharArray转换为字符数组比较不容易出错

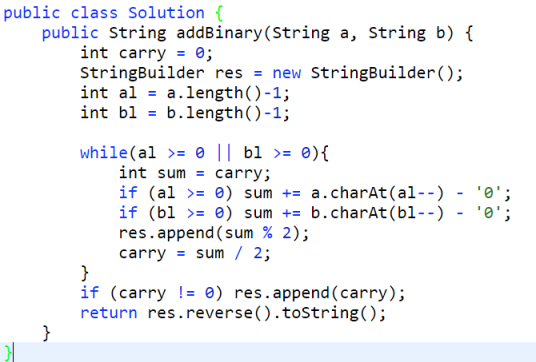
### Detect Capital

正则表达式应该最快，or 基础的判断4种情况

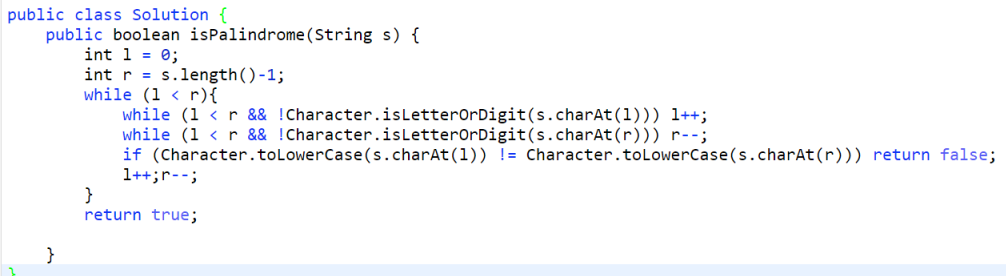
383. Ransom Note

这方法真滴666，好好学习一下，算是一种tricky 的实现map/counter功能的办法，26位代表26个英文单词，ASCII码值代表index

1. Add Binary

位运算；用了Stringbulider便于翻转

125. Valid Palindrome



1. Character.isLetterOrDigit()函数判断其是不是符号；2、将单个字符小写化Character.toLowerCase(char ); 3、two pointer method

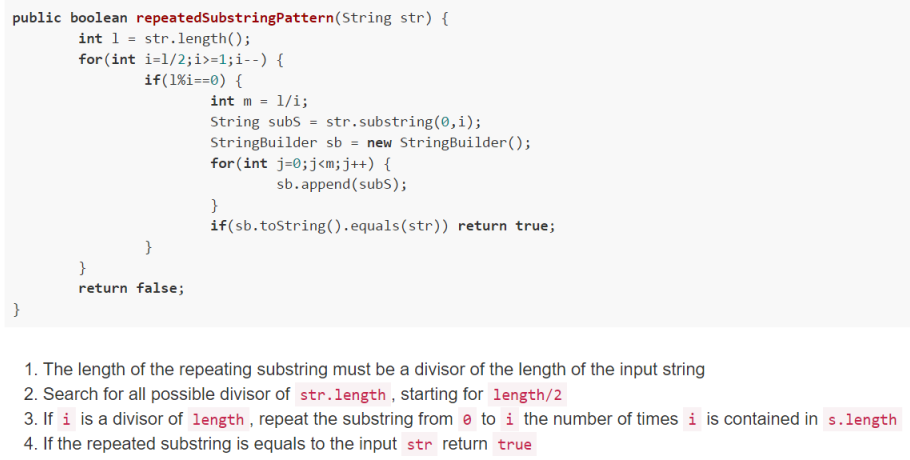
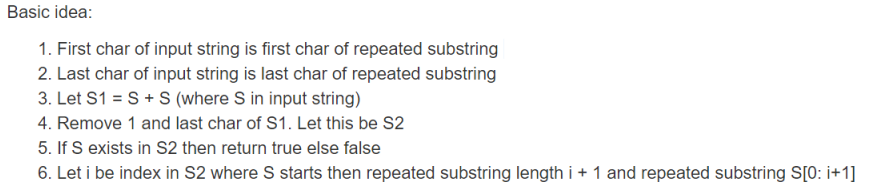
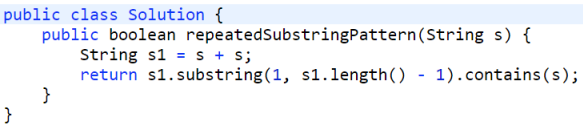
38. Count and Say

用2个StringBuilder来表示temp和res；主要是理解题意，理解了就可以直接straightforward出来程序，注意一下n = 1到n = 2的特殊情况

也可以考虑用hashmap做（字典法，key和value分别对应count and say）

1. Repeated Substring Pattern

3种超屌方法；

1. 
2.  
3. KMP（再议）
4. Number of Segments in a String

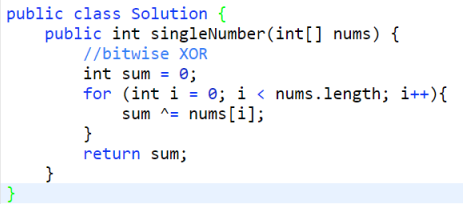
Array类题

1. Missing Number
2. 要求O(n)和constant extra space所以这里不能用sort，用的是数学思想
3. 将所有数字加起来的和与n(n-1)/2的差代表缺少倒数第几位数，相等则输出最后一位+1；
4. Move Zeroes

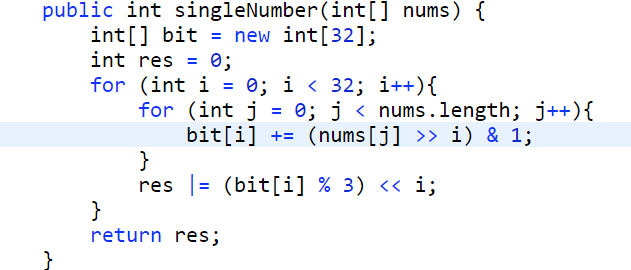
题意：将所有0移到array末尾并不改变其他非0数的顺序；

Easy题，two pointer method，将2个pointer的起始点都设置在idx = 0处，然后向右循环

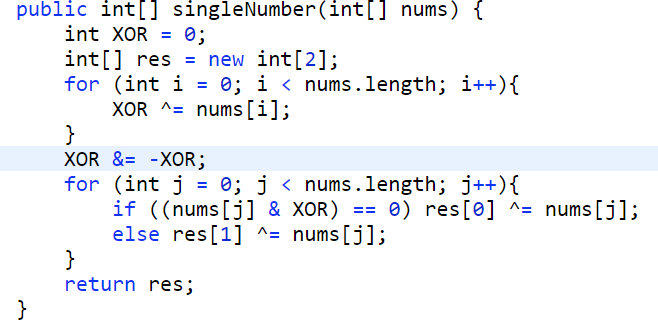
1. Single Number（这之后的4题其实都是位运算的题）

（需知：0^N=N,N^N=0）如果不限制extra space的话还可以用math method（return sum(set(nums))\*2-sum(nums)）/Python版本代码，因为java中HashSet需要初始化，会占用space.

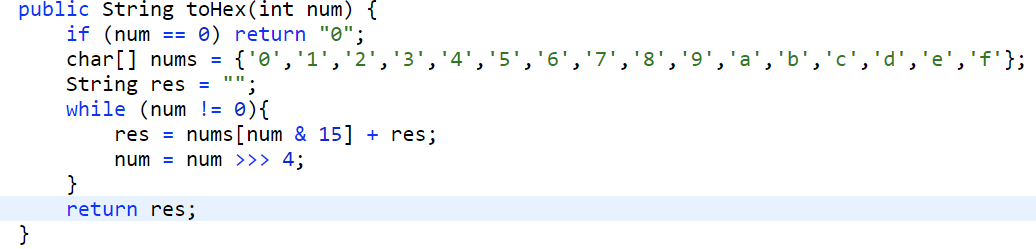
1. Single Number II

int数据类型有32位，可以用32变量存储这N个元素中各个二进制位上1出现的次数，最后再进行mod 3的操作，若结果为1则为所求的元素中的一位；O(32\*N)，这是个通用解法，适用于出现k次。

1. Single Number III

这道题其实是很巧妙的利用了[Single Number 单独的数字](http://www.cnblogs.com/Given an array of integers, every element appears three times except for one. Find that single one.  Note: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory" \t "http://www.cnblogs.com/grandyang/p/_blank)的解法，因为那道解法是可以准确的找出只出现了一次的数字，但前提是其他数字必须出现两次才行。而这题有两个数字都只出现了一次，那么我们如果能想办法把原数组分为两个小数组，不相同的两个数字分别在两个小数组中，这样分别调用[Single Number 单独的数字](http://www.cnblogs.com/Given an array of integers, every element appears three times except for one. Find that single one.  Note: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory" \t "http://www.cnblogs.com/grandyang/p/_blank)的解法就可以得到答案。那么如何实现呢，首先我们先把原数组全部异或起来，那么我们会得到一个数字，这个数字是两个不相同的数字异或的结果，我们取出其中任意一位为‘1’的位，为了方便起见，我们用 a &= -a 来取出最右端为‘1’的位，然后和原数组中的数字挨个相与，那么我们要求的两个不同的数字就被分到了两个小组中，分别将两个小组中的数字都异或起来，就可以得到最终结果了

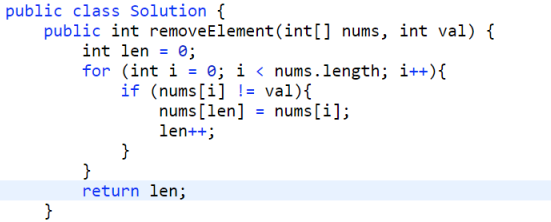
1. Convert a Number to Hexadecimal

直接使用位运算的方法是最为简便的.

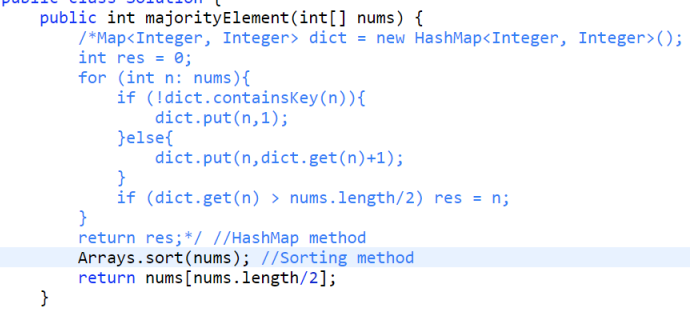
1. Remove Duplicates from Sorted Array

Two pointer method,要注意的地方就是return的值是原来数组的前xx位（idx从1开始算！！！mdzz），表示的是新数组length，不用新建新的array节省space.

27. Remove Element

由于只要输出array的前几位所以可以采用覆盖的方法从idx = 0位开始直接将不等于val的项覆盖过去.

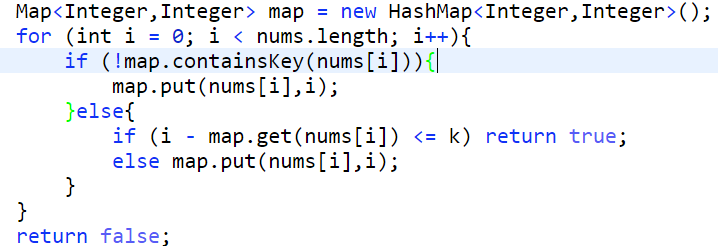
1. Majority Element

这边主要是记一下java里面HashMap的用法，类似于python中的dictionary，要特别注意的是<>中的是Integer不能是int；BTW，这题用sort法超快

1. Contains Duplicate

Easy题; 2 methods: 1、用HashSet，若元素已经存在于set中则return true 2、sort法，sort之后依次比较前后2元素是否相等

1. Contains Duplicate II（存在相同元素且间距小于等于k）

用hashmap来处理；

1. Contains Duplicate III

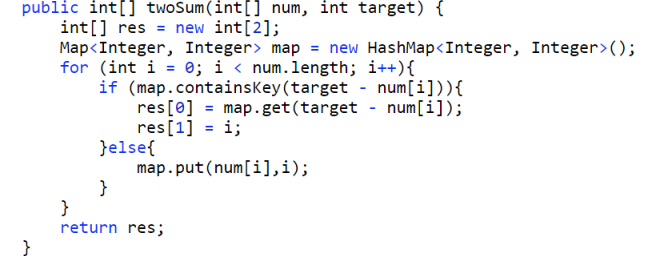
上题的follow up，有点难，涉及到BST和bucket解法和treeset解法

1. Max Consecutive Ones

求连续的1的最大连续长度，int[] nums中只有0和1；

Sum法：将连续的1的和加起来，return最大的sum值

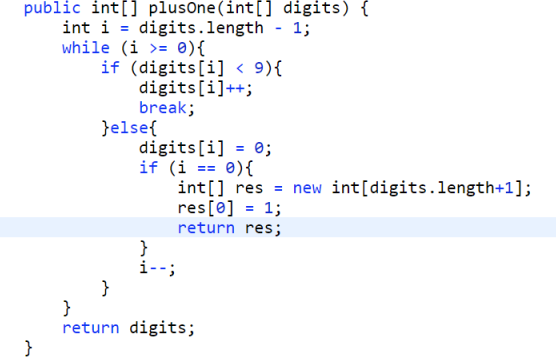
1. Best Time to Buy and Sell Stock
2. Best Time to Buy and Sell Stock II
3. Two Sum

HashMap解法，很机智的方法

1. Two Sum II - Input array is sorted

因为相比上一题array（数组）已经是sorted的了，所以可以继续用hashmap也可以用two pointer method，同时用Binary Search可以提高运行速度

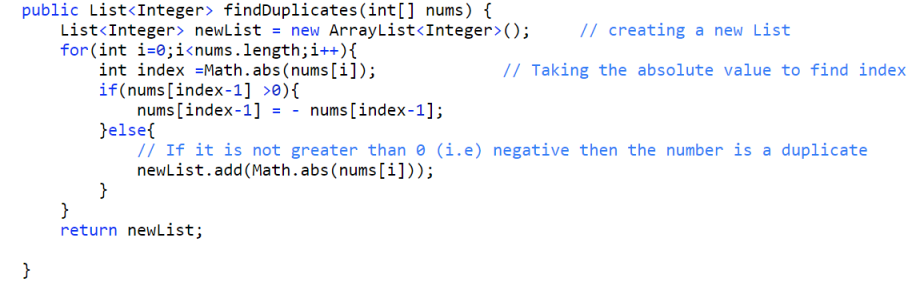
1. Plus One

这道题不难但是暴露了很多细节问题！一开始想通过add(0, 1)的方式直接对array进行修改，但之后发现error，因为add函数是针对arraylist的；之后我想着用array.aslist()转为list再add再转回来，又报error，查资料发现array.aslist()转出来的是array这个数组的静态内部类，并非真正的arraylist，它有set, get, contains的功能但是不能添加删除元素； 最后这题干脆还是新建个数组比较方便

1. Can Place Flowers

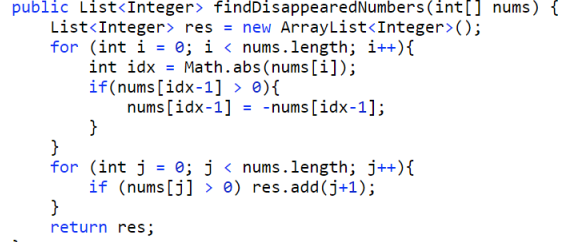
这题就用straightforward的方法，别想什么数学骚操作，下一个

1. Find All Duplicates in an Array

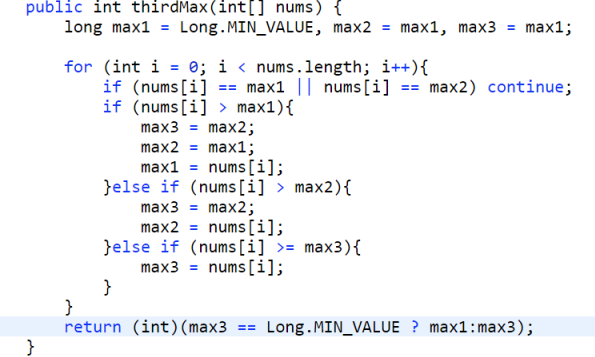


这段程序之后复习时也要好好看看，真的没想到这种做法！！

1. Find All Numbers Disappeared in an Array

整体思路和上一题差的不多，主要还是mark 为负数的方法。

1. Third Maximum Number

这题真的是。。。一言难尽啊；一开始declare 3个max值都是Integer.MIN\_VALUE但是当给的数组里有这个数的时候就很特么蛋疼，用count计算max赋值次数的话需要sort，这样复杂度就高了，后来发现这里可以声明为Long type的MIN\_VALUE，哇，感觉发现新大陆！；除了这种方法之外，还可以选择使用Treeset这种数据结构

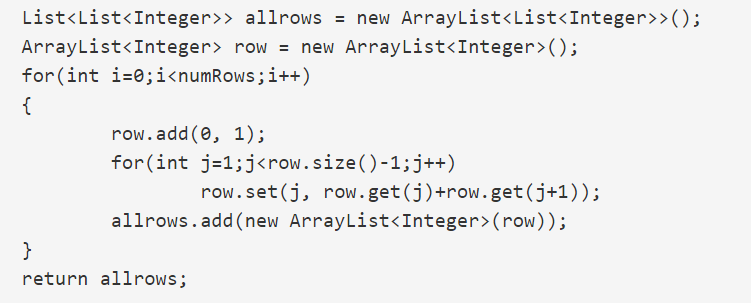
http://www.cnblogs.com/skywang12345/p/3311268.html#a1 附上一篇介绍Treeset的博文（也可以直接去API文档查）

1. Maximum Product of Three Numbers

3个数的最大乘积只有可能是最小的两个数和最大的数的乘积（2个或3个为负数） or 最大的3个数的乘积（2个或3个为正数）；eg：可以取一个长度为5的例子试试看，结果显然是吻合的；这题我自己用了Arrays.sort()，但若是追求最低的时间复杂度，应该不用sort而用上面那题的排序的方法，最后 **return** Math.max(max1\*max2\*max3, max1\*min1\*min2);

1. Pascal's Triangle

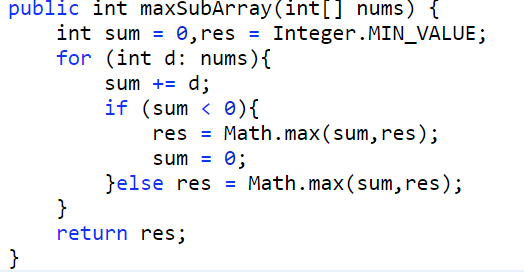
2种办法，1、引入一个temp list，头尾都加0再前后2项相加；2、如下图code

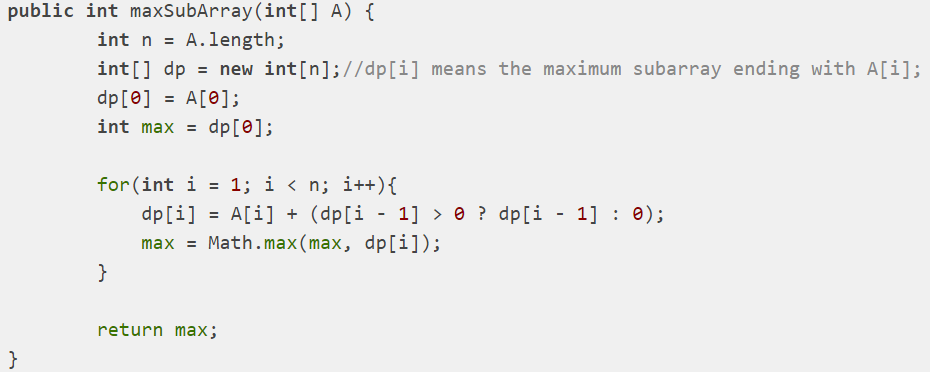
这种方法从第二行开始就每次在首位加一个1，然后第2~length-1位就依次前后2项相加即可

1. Pascal's Triangle II

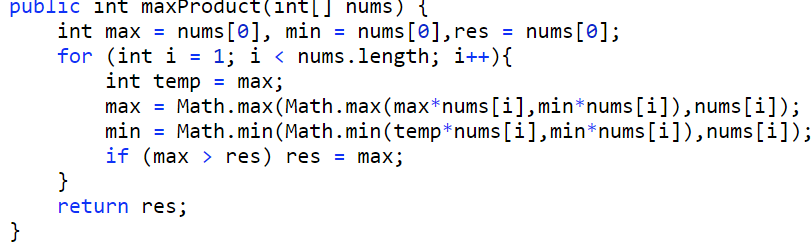
这题用了上题中的第二种方法，相当于118题中的for loop删除掉最后一句（当然第一种也是一样就是）

1. Maximum Subarray

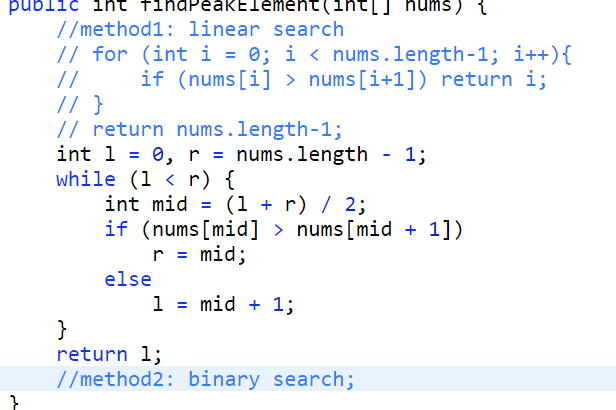
method 1、由于只需要返回最大值并不需要返回最大区间所以相对比较简单，只要从左往右加，sum<0时重置下sum=0就可以了

Method 2、DP；

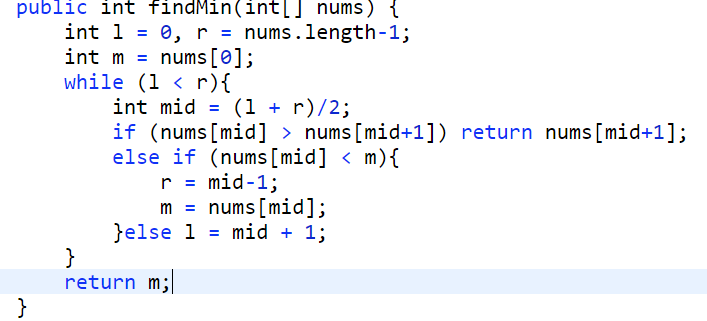
1. Maximum Product Subarray

Loop through the array, each time remember the max and min value for the previous product, the most important thing is to update the max and min value: we have to compare among max \* A[i], min \* A[i] as well as A[i], since this is product, a negative \* negative could be positive.

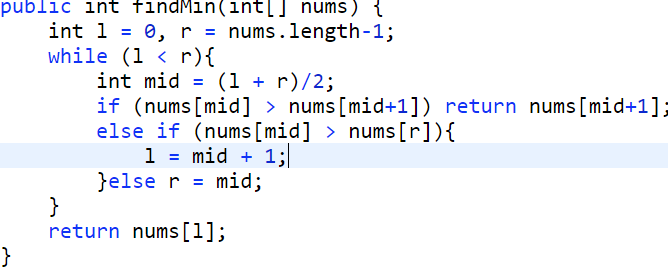
1. Find Peak Element



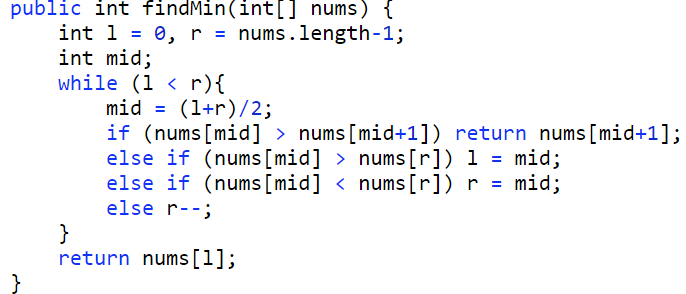
1. Find Minimum in Rotated Sorted Array



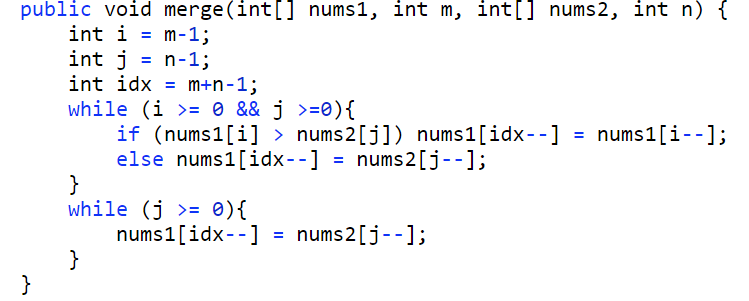
--------------------------------------------------------------------------------------

优化版本

### Find Minimum in Rotated Sorted Array II

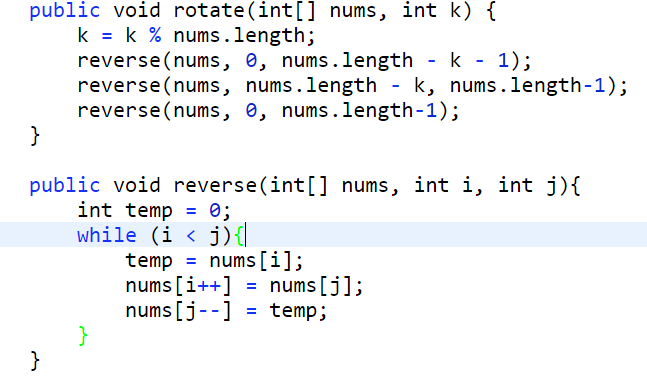
这两题让我发现binary search中要特别注意mid+1和mid-1的赋值（给low和high），容易产生一些边界范围的问题，稳妥起见可以选择和这题一样只用low=mid和high = mid，开始做的时候就一直是错在用了high = mid-1。

1. Merge Sorted Array（test case给我恶心到了。。。）

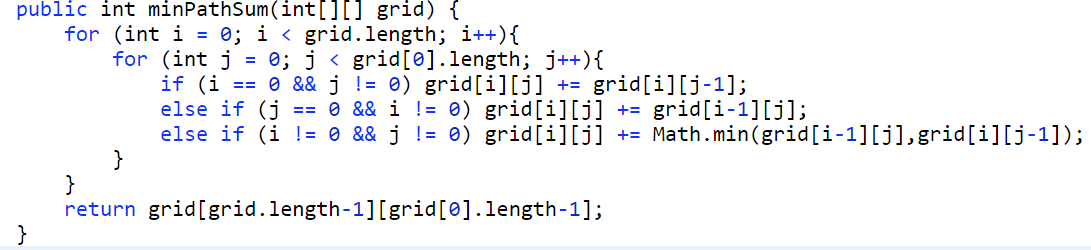
merge sort其实只要将2个array从最后一位向前开始比较即可，将较大的那个赋值给nums1的最后一位（因为nums1的size是>= m+n的，所以不会超出范围）

1. Rotate Array //For example, with *n* = 7 and *k* = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

The basic idea is that, for example, nums = [1,2,3,4,5,6,7] and k = 3, first we reverse [1,2,3,4], it becomes[4,3,2,1]; then we reverse[5,6,7], it becomes[7,6,5], finally we reverse the array as a whole, it becomes[4,3,2,1,7,6,5] ---> [5,6,7,1,2,3,4].



1. Minimum Path Sum

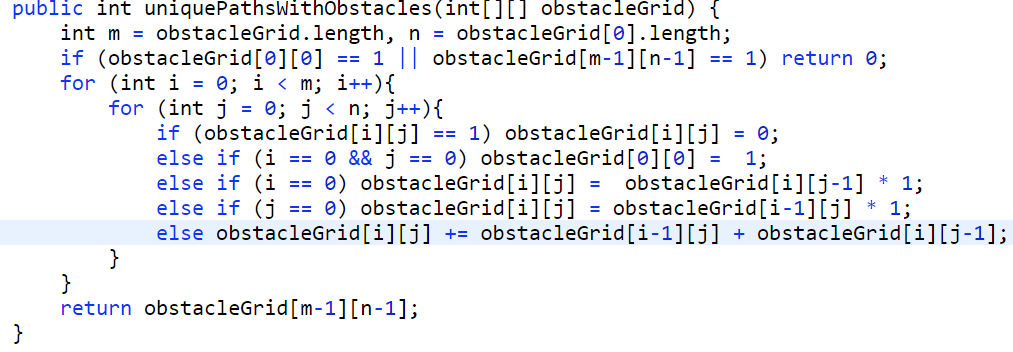


DP solution, 思路就是用每一个grid[i][j]来存储到（i，j）这个点的时候的最短距离

1. Unique Paths

还是DP solution，每一格存储的是到这个点的所有可能性

1. Unique Paths II

上一题的follow up；核心思想是2点：1、遇到1就赋值为0；2、第一行和第一列的值赋值规则是上一项乘以1(若有障碍物则此障碍物之后的都会变成0);

1. Minimum Moves to Equal Array Elements

Emmmmmm，注意审题，一次move指的是n-1个元素同时+1,；所以这道题就逆向思维，转换为只有一个元素需要减1.