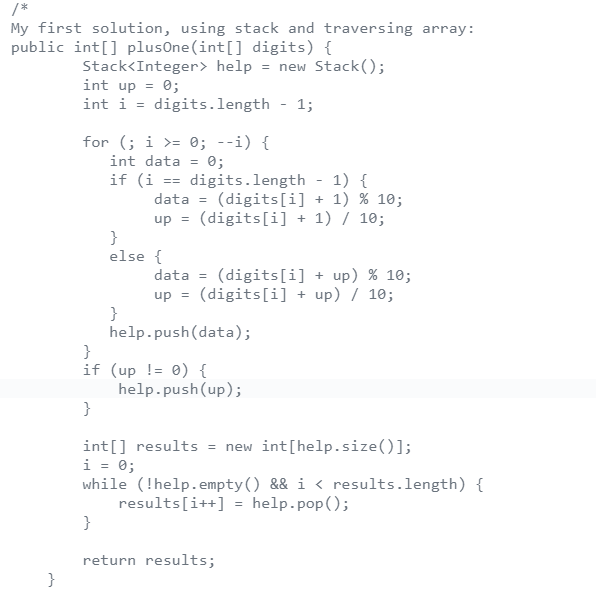
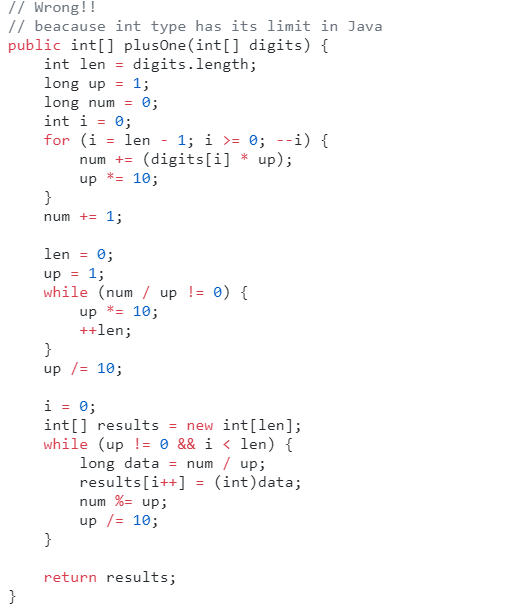
**66. Plus One(+1)**

这题要按位处理，用栈，队列，数组，向量保存每位数字：

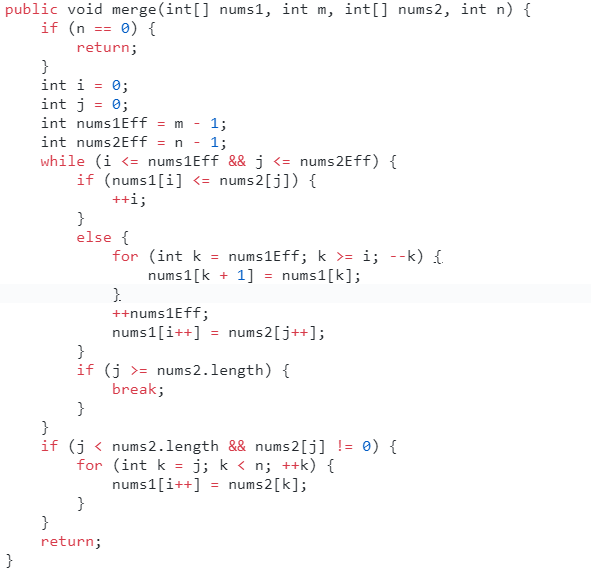


另一个思路，算出总数值后+1再存，这个方法是错误的，因为数值很大的时候有可能超过int类型，导致溢出：



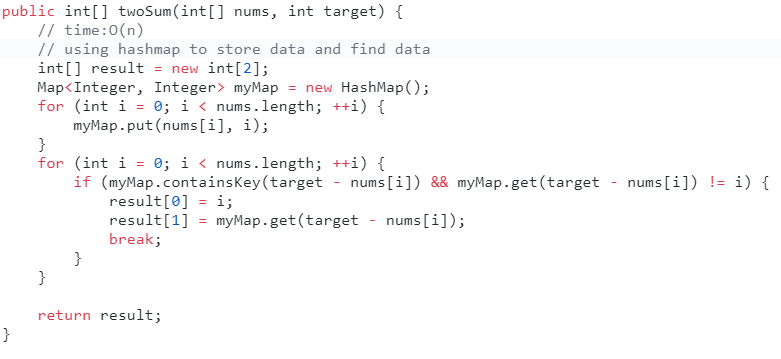
**88. Merge Sorted Array(合并两个有序数组)**

在主数组中找到位置后，将剩余元素后移即可。



1. **Two Sum**

正常思路(小孩子思路)时间复杂度为平方层次，使用Hash Map可以将复杂度降为O(n)，然而空间复杂度为O(n)，因为Hash Map除了存储数据(键值对)外，有containsKey()方法判断是否存在键，不用再内循环一次：

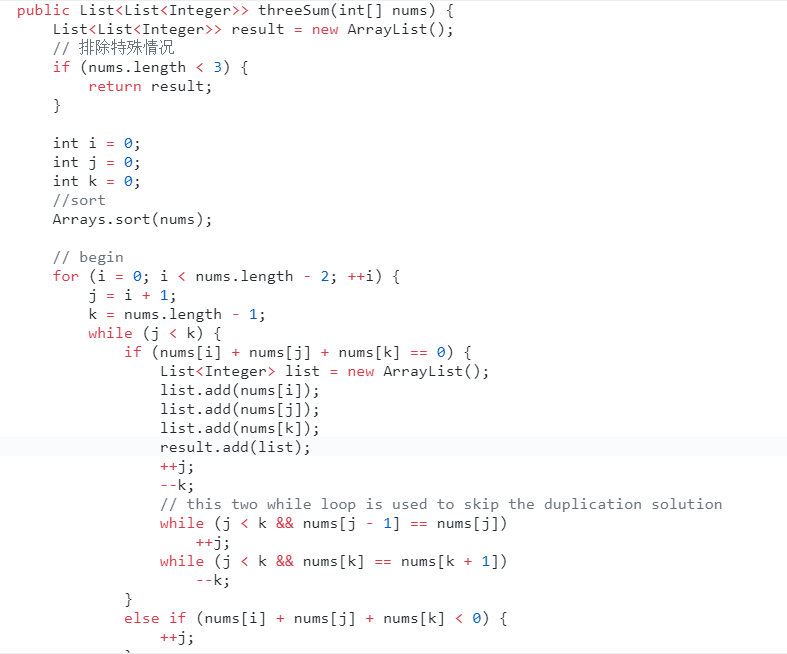


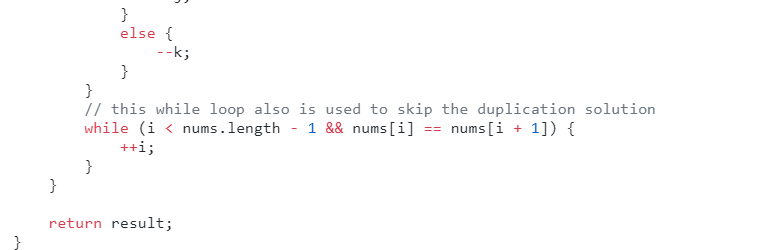
要熟练常用的集合类的方法，比如HashMap中的get(Key)可以获得值。Hash Map中没有下标访问。再比如Queue类型判断空是isEmpty()，不同于Stack的empty()方法。

**15. 3Sum(找到三个下标不同的数a, b, c, 使得a + b + c = 0)**

既然要找到三个满足要求的数，而且这三个数又要按照大小排序，所以需要提前对数组进行排序。因为排序满足O(nlgn)即可，使用Java自带的函数排序Arrays.sort(数组名)，Google此函数的使用。

既然排序好了，可以使用排序的优点，设置两个指标(j, k)，一个从前至后，一个从后到前，将复杂度设为O(n^2)，同时要注意要排除重复的答案，在内循环中使用三层while循环，代码如下：



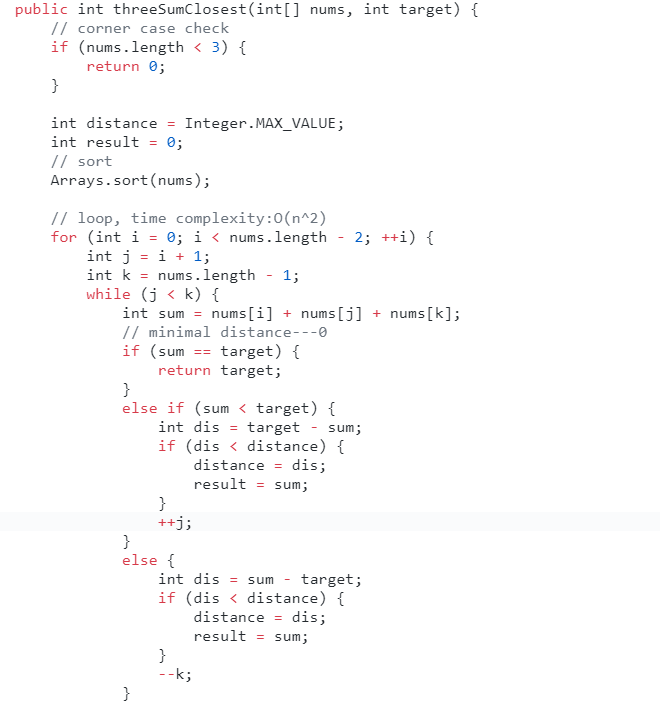


最后不要忘记corner case check(边缘条件)，当数组元素小于3时不需要检查。

这题即使复杂度为平方级别，也会出现exceed time limit的情况，所以这三个while循环跳过重复元素的条件缺一不可。

**16. 3Sum Closest**

3Sum的延申，思路和3Sum几乎一样，可以巩固思路。



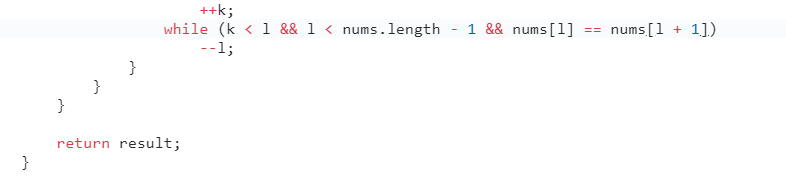


注意Java中最大/小整数表示是Integer.MAX\_VALUE(Integer.MIN\_VALUE)，C++中是INT\_MAX(INT\_MIN)。

**18. 4Sum**

思路同3Sum差不多。





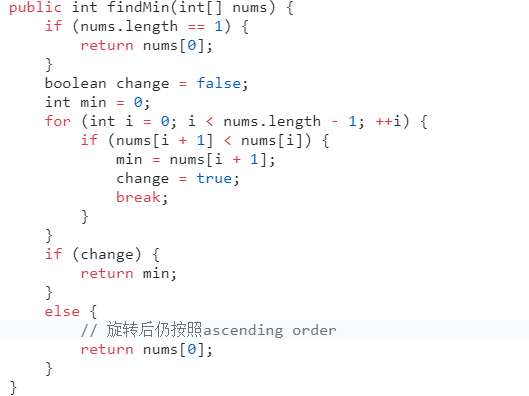
注意两个if语句和两个while循环用来跳过重复值，其中两个if语句不能换成while循环，因为这样会造成Time Limit Exceeded。For循环还是用continue好。

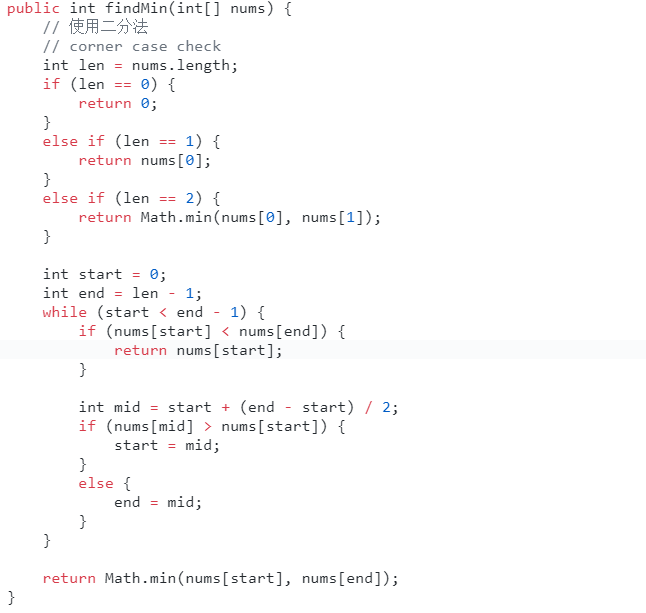
**扩展：kSum**

见PDF文件。

**153. Find Minimum in Rotated Sorted Array**

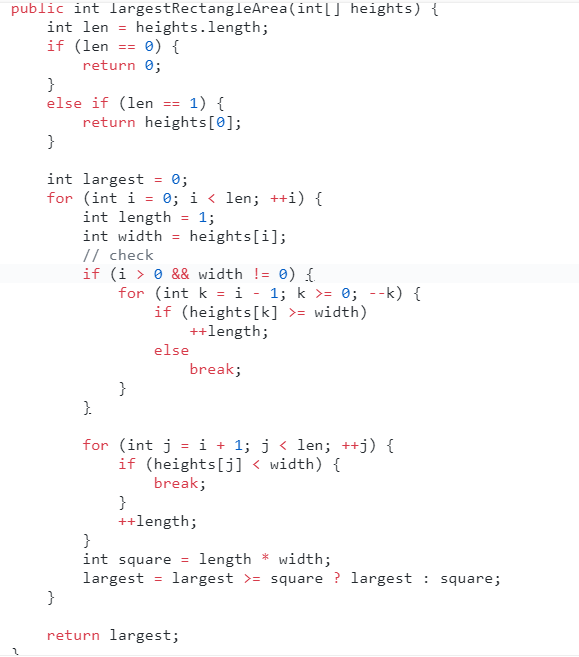
自法如下：



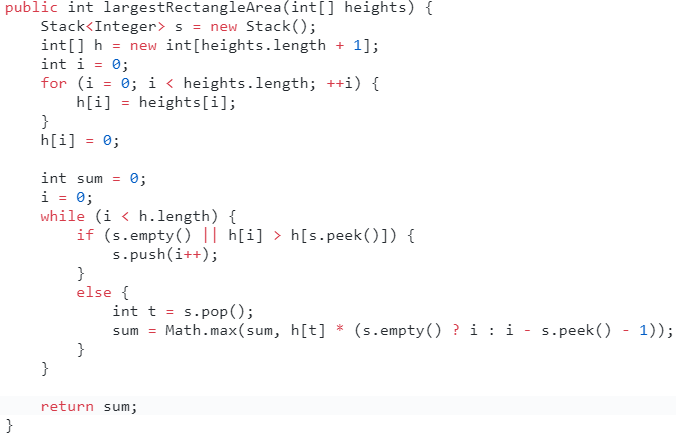
按照PDF所说，可以采用二分法：  


While循环注意要是start < end – 1，而不能start < end。

**84. Largest Rectangle in Histogram**

****

自法，遍历每一个bar，向左向右遍历找到高度小于bar的临界点，再计算面积，时间复杂度是O(n^2)，当然这种方法没有技术含量而且过不了大数据测试。

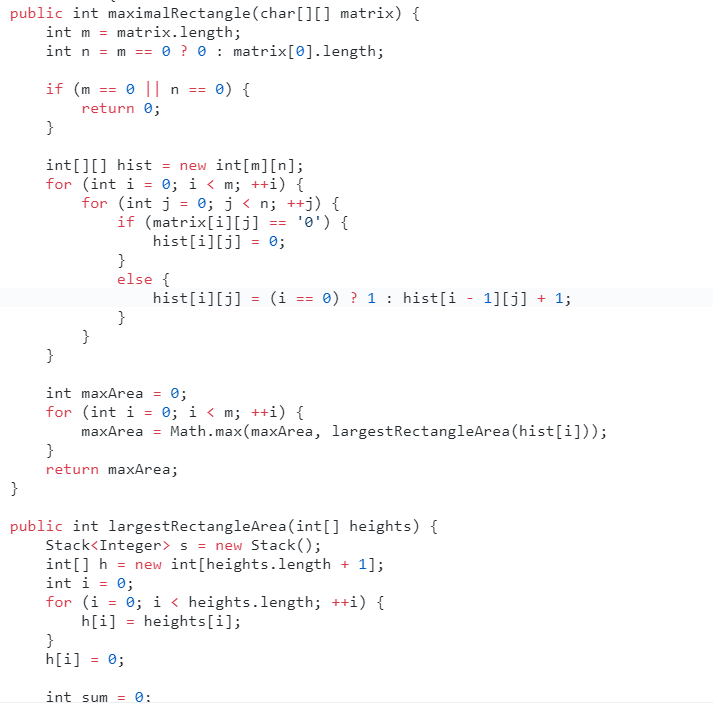


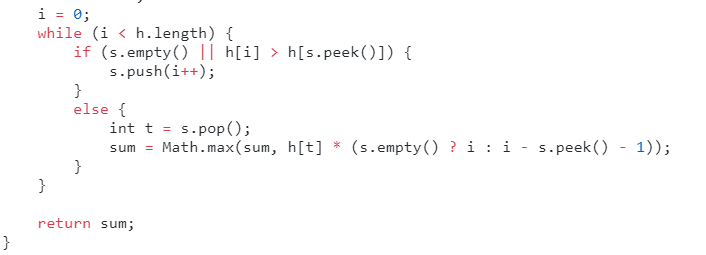
牺牲空间复杂度换取时间复杂度的例子…但感觉很难想得出来。

分析以下，看PDF。

最后末尾加个0处理最后一个bar高度。

**85. Maximal Rectangle**

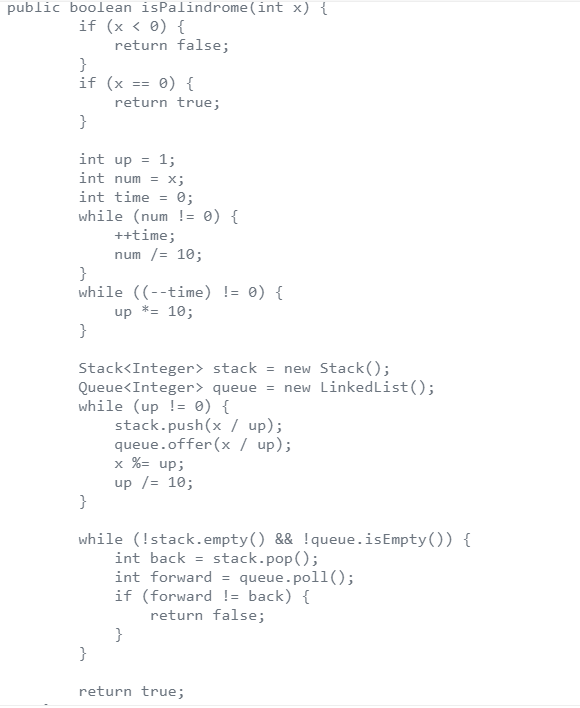
****



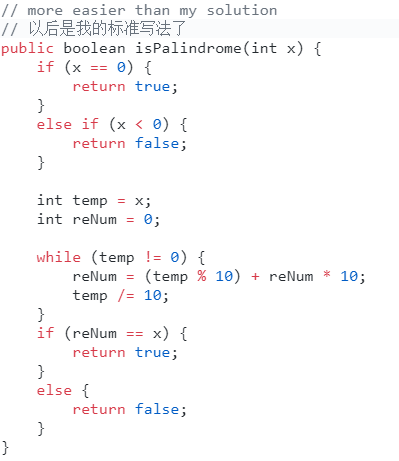
承接上题柱状图求最大矩形面积，这个矩形可以看成有m个柱状图(m层)，依次求面积即可。

注意矩形的初始化。

**9. Palindrome Number(确定是不是回文数字)**

自己的方法空间复杂度为O(n)，而且代码冗长：  


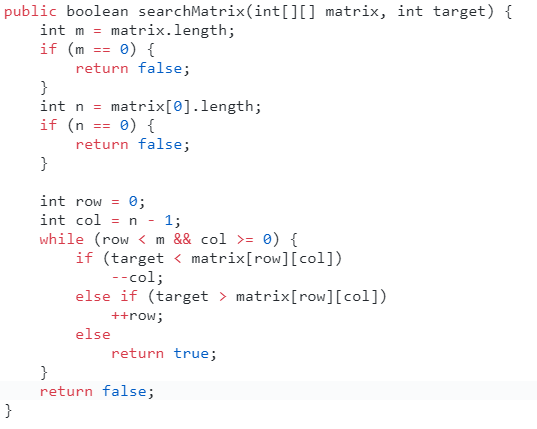
看看别人的：



分析：对原数字的处理一定要是while循环内的判断语句temp != 0，而不是其他的比如x % 10 != 0之类的，因为要确认10001的特殊情况。此外，reNum的计算也是有技巧的。

**74. Search a 2D Matrix**

如果没有题目的两个条件，正常情况下利用用brute force(暴力破解)方法，查询复杂度为O(n^2)，但正因为有这两个条件，可以采用类似二分法的方法：

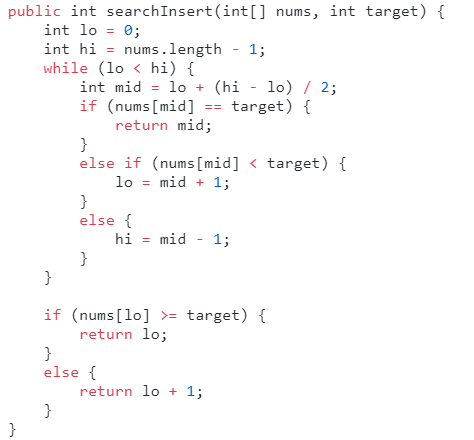


注意两个参数的选择，一个是行，一个是列，从右上角的元素开始搜索，而不是两个参数都是行的情况。

这样的时间复杂度为O(n)。

**35. Search Insert Position**

最简单的方法是线性复杂度，但显然可以用二分法，自己写的如下：



**162. Find Peak Element**

可用二分法。一定要想清楚为什么能用二分法。

这里略。