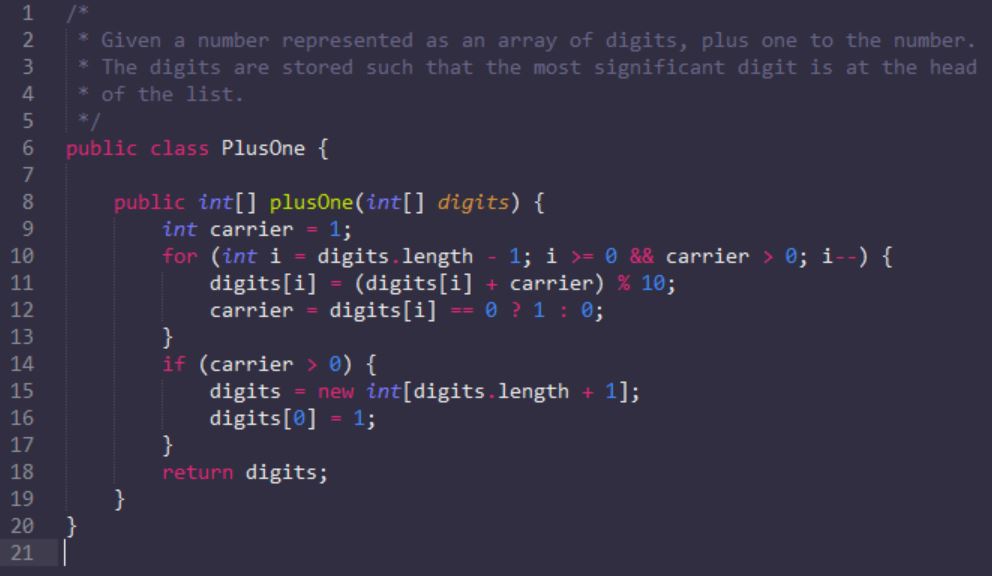
**1** #66

### Problem: Plus One

**Description:**

Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.



这道题的关键在于处理进位。因为每次只是+1，所以情况分别有1：+1之后不进位，只是在当前数值上+1；2：+1之后和等于10，当前数位成为0，然后进位1（特殊情况有进位之后一位数变成两位数，两位数成为三位数等等）

所以我们用循环来处理这个问题，我们从Array的最低位开始循环，我们对于现有位（最低位）加一，如果和不等于10，那么继续循环，如果等于10，则将该数位设为0，再将最低位的前一位加一，如果最高位（most significant）等于10，则需要重新建立一个新数组，最高位1，其余位都是0（还没实现）

**2** #119

**Problem:** Pascal's Triangle II

**Description:**

Given an index *k*, return the *k*th row of the Pascal's triangle.

For example, given *k* = 3,  
Return [1,3,3,1].

**Note:**  
Could you optimize your algorithm to use only *O*(*k*) extra space?



Pascal's Triangle I和II这两道题主要思路都是用二维数组储存整个三角形

不同于Pascal’s Triangl 前一题，这里我们仅仅需要得到的第k层的集合，但只能使用O(k)的空间。所以不能用前面二维数组的方式，只能使用一位数组滚动计算。

在前一题，我们知道，帕斯卡三角的计算公式是这样的，A[k][n] = A[k-1][n-1] + A[k-1][n]。

假设现在数组存放的第3层的数据，[1, 3, 3, 1]，如果我们需要计算第4层的数据，如果我们从前往后计算，譬如A[4][2]= A[3][1] + A[3][2]，也就是4，但是因为只有一个数组，所以需要将4这个值覆盖到2这个位置，那么我们计算A[4][3]的时候就会出现问题了，因为这时候A[3][2]不是3，而是4了。

为了解决这个问题，我们只能从后往前计算，仍然是上面那个例子，我们实现计算A[4][3] = A[3][2] + A[3][3]，也就是6，我们将6直接覆盖到3这个位置，但不会影响我们计算A[4][2]，因为A[4][2] = A[3][1] + A[3][2]，已经不会涉及到3这个位置了。

**3**

#189

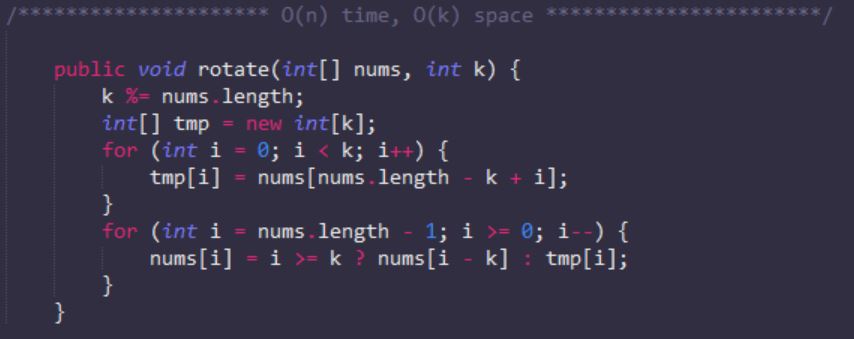
### Problem: Rotate Array

### Description:

Rotate an array of *n* elements to the right by *k* steps.

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].

**Note:**  
Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.



## **以n-k为界，分别对数组的两边进行逆置，然后再对整个数组进行一次逆置**

这个算法的实现原理我能理解，但是为什么要这样做呢？推导过程是从以结果为导向来推导。

以数组arr[5] = {1, 2, 3, 4, 5};为例。

如果n=5，k=2，则旋转结果为{4, 5, 1, 2, 3}

看上去很像整个数组的逆置{5, 4, 3, 2, 1}

但是还需要变换一下才能得到想要的结果：

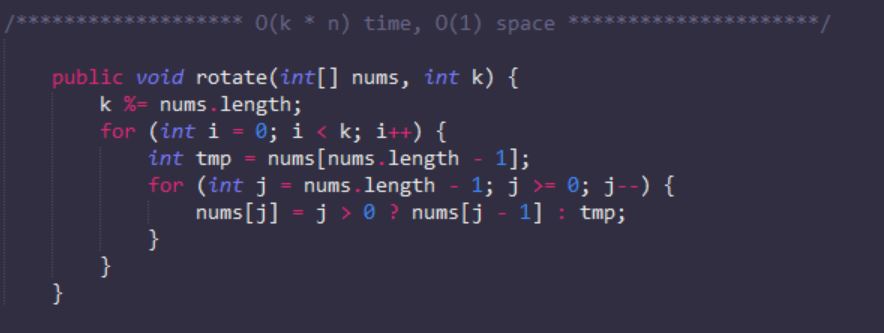
发现如果将前面(0,k)和后面(k,n)的两部分分别逆置之后就会得到最终的结果。

即

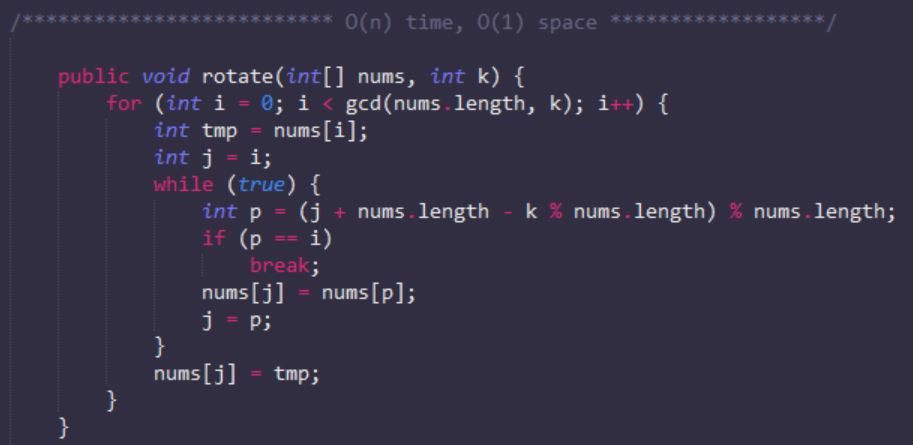
{5, 4} => {4, 5}

{1, 2, 3} => {3, 2, 1}

于是就有了先将数组前后两部分逆置，然后再将整个数组逆置的解法。



这个比较容易想到，第一个想法就是写个循环一个个移动，时间复杂度很高



这个gcd的解法很难想到，是从网上看的。这个算法是比较难想到的，这个算法优点是在执行了GCD(n, k)次后即可停止，GCD(Greatest Common Divisor)–最大公约数

如果n与k互质，下面的赋值过程就能完成所有值的赋值(设数组为X[0…n-1],长度为n)： t = X[0]

X[0] = X[i mod n]

X[i mod n] = X[2i mod n]

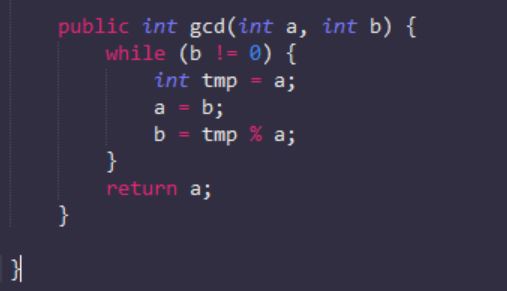
……

X[(n-2)*i mod n] = X[(n-1)*i mod n]

X[(n-1)\*i mod n] = t

以上操作已经把包括{0,1,…,n-1}所有元素放到了最终位置上，每次完成一个元素的放置。 根据以上我们得到了一个中间结论，如果n和k互质，我们就可以一次完成所有元素的纺织。那么如果n和k不是互质的呢？

那就是让n和k互质。即让k’=k/gcd(k, n),n’=n/gcd(k,n)。这样就构成了一对互质的数。这就意味着需要把整个数组的每g(g=gcd(k,n))个元素组成块，每个块的k’和n’就互质了。每次相当于把g中的一个元素移到最终的位置，由于g是块元素，每个块含有g个元素，所以总共需要g次移动。



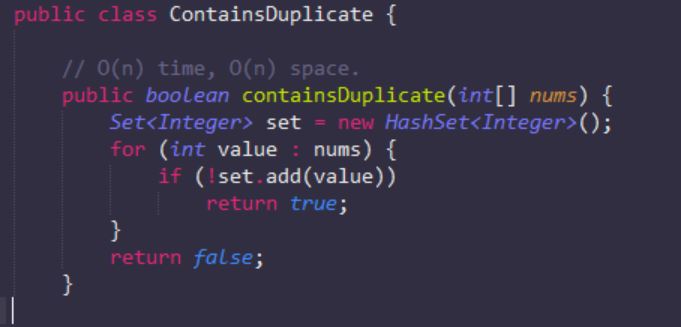
**4**

#217

### Problem: Contains Duplicates

**Description:**

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.



思路是借用Hashtable，遍历数组，对于数组中每一个元素，检查它是否在hashtable中，如果在，说明该元素已经出现过，是重复的；如果不在，说明是新元素，加入hashtable中就可以

**5**

#55

**Problem:** Jump Game

**Description:**

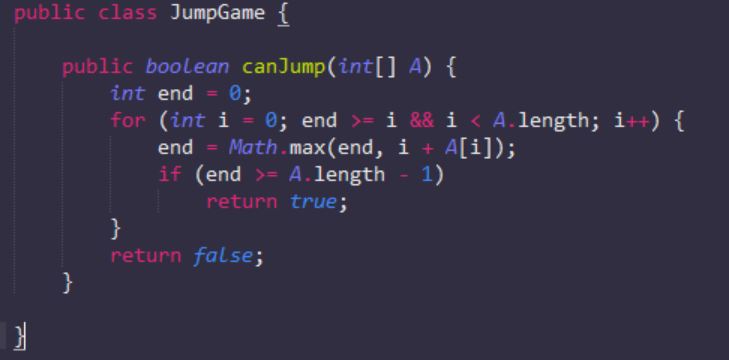
Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:  
A = [2,3,1,1,4], return true.

A = [3,2,1,0,4], return false.



**6**

#1

### Problem: Two Sum

**Description:**

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

**Input:** numbers={2, 7, 11, 15}, target=9  
**Output:** index1=1, index2=2



**7**

#35

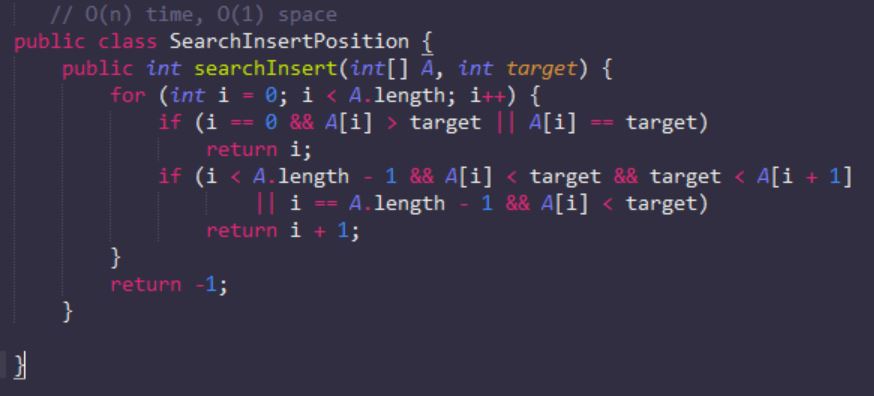
### Problem: Search Insert Position

**Description:**

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.  
[1,3,5,6], 5 → 2  
[1,3,5,6], 2 → 1  
[1,3,5,6], 7 → 4  
[1,3,5,6], 0 → 0



### 8

#62

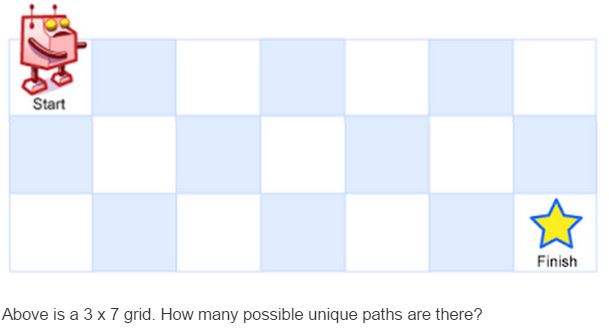
### Problem: Unique Paths

**Description:**

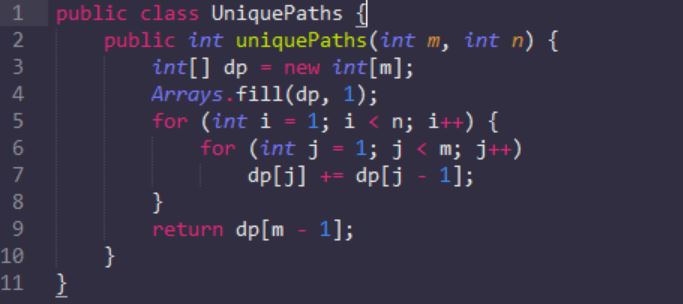
A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?



**Note:** *m* and *n* will be at most 100.



**9**

#81

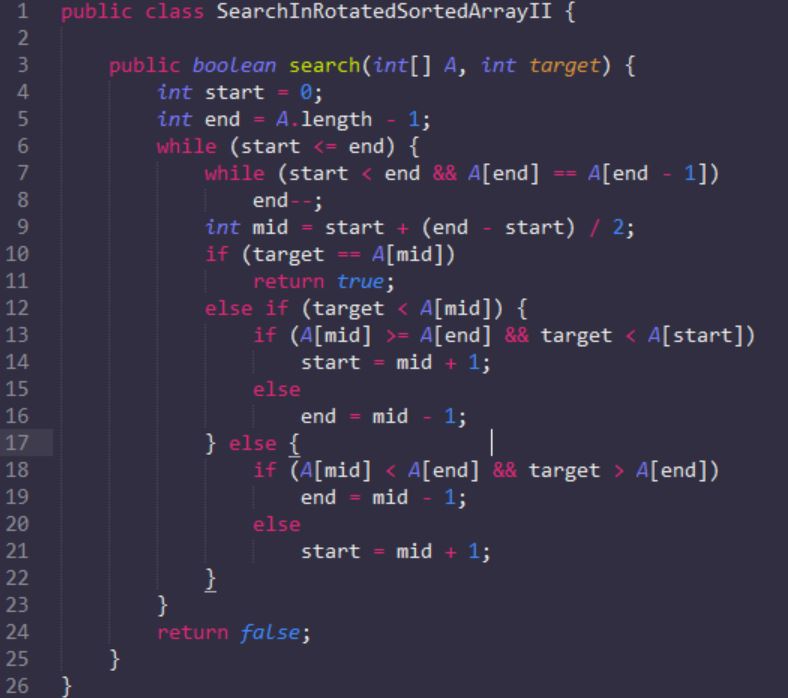
### Problem: Search in Rotated Sorted Array II

### Description:

Follow up for "Search in Rotated Sorted Array":  
What if *duplicates* are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array.



**10**

#31

### Problem: Next Permutation

**Description:**

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.  
1,2,3 → 1,3,2  
3,2,1 → 1,2,3  
1,1,5 → 1,5,1

