LeetCode做题笔记

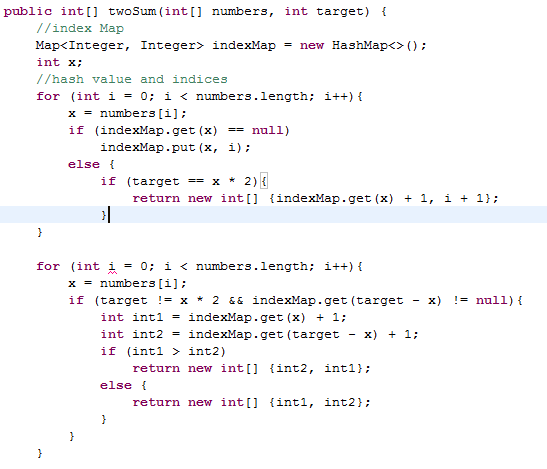
1. Add two numbers：给定一个数集合和一个数，已知集合中有两个数的和是给定数，求这两个加数的index

方法1：暴力，n^2时间复杂度，不推荐

方法2：快速排序nlogn。按集合里数的两倍与target的大小关系对分。对每一个第一部分的数，在另外一个部分二分搜索第二个数：500~ms

方法3：hash，n的时间复杂度，最高纪录420ms

方法3源码：



**经验1：对hashMap的get和put都是非常耗时间的，尽量少做**

**经验2：方法2最好使用随机化快排，效率很高**

1. 把两个链表表示的数加起来：最佳624ms，3%

用长一点的链表做基础，最多只需要new一个新节点

优化建议：进位尽量用数字。如果一个链到头了，另一个没到，应该沿着长链前进。如果进位是0就可以即刻返回，不需要继续前进。

源码：

public class Solution {

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

int sum, progress = 0;

ListNode tmp1 = l1, tmp2 = l2;

for (;;){

sum = l1.val + l2.val + progress;

progress = sum >= 10 ? 1 : 0;

l1.val = l2.val = sum >= 10 ? sum - 10 : sum;

if (l1.next == null){

if (l2.next == null){

if (progress == 1)

l2.next = new ListNode(1);

return tmp2;

}

else {

l2 = l2.next;

}

for (; ;){

if (progress == 0)

return tmp2;

++l2.val;

progress = l2.val >= 10 ? 1 : 0;

l2.val = l2.val >= 10 ? 0 : l2.val;

if (l2.next == null){

if (progress == 1)

l2.next = new ListNode(1);

break;

}

else

l2 = l2.next;

}

return tmp2;

}

if (l2.next == null){

l1 = l1.next;

for (; ;){

if (progress == 0)

return tmp1;

++l1.val;

progress = l1.val >= 10 ? 1 : 0;

l1.val = l1.val >= 10 ? 0 : l1.val;

if (l1.next == null){

if (progress == 1)

l1.next = new ListNode(1);

break;

}

else

l1 = l1.next;

}

return tmp1;

}

l1 = l1.next;

l2 = l2.next;

}

}

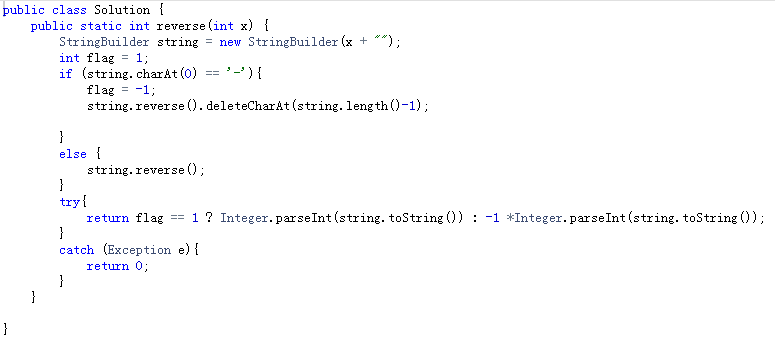
}

**经验2：leetcode不靠谱啊，速度不一定对**

1. 把一个整数数位逆转，注意符号，不能溢出

最好记录404ms，前5%

**经验3：类库比较快。而且好像早上比较快？**



官方提示：

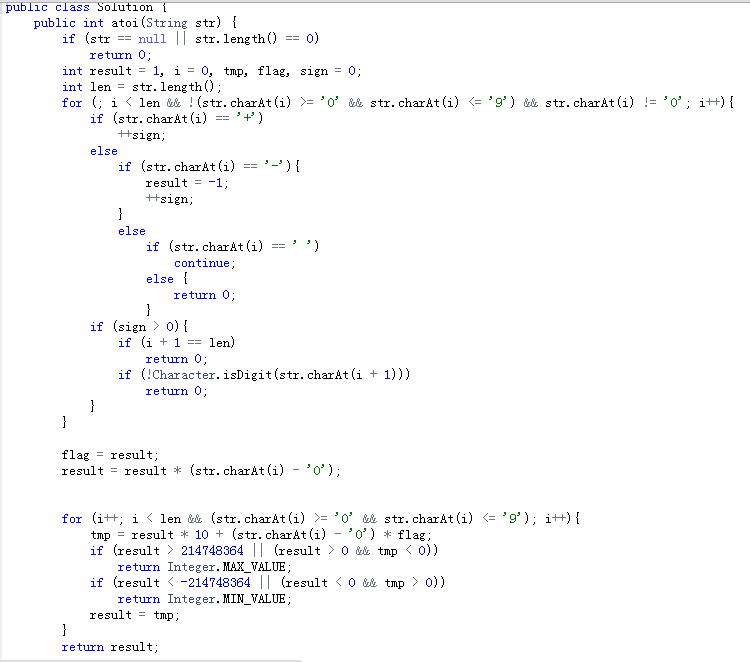
To check for overflow/underflow, we could check if ret > 214748364 or ret < –214748364 before multiplying by 10. On the other hand, we do not need to check if ret == 214748364, why? 溢出是2147483647， 214748364乘10不溢出

1. 手写atoi，一大堆判断。。。最好记录408ms，约为5%。

经验4：**如何经济的判断溢出**

**经验5：让不合法的输入第一时间return 0;**

**经验6：对于javascript里那一堆isxxx的函数，对应在Java的Character类下**



1. 求两个升序数组的中位数(8, hard)

网上绝大部分解法都是错误的，和我犯了一样的错误。三个小时啊啊啊啊啊白费了

经验7：求平均数应该除以2.0

错误算法：

**分别求出序列A 和B 的中位数，设为a 和b，求序列A 和B 的中位数过程**

**1）若a=b，则a 或b 即为所求中位数，算法结束。**

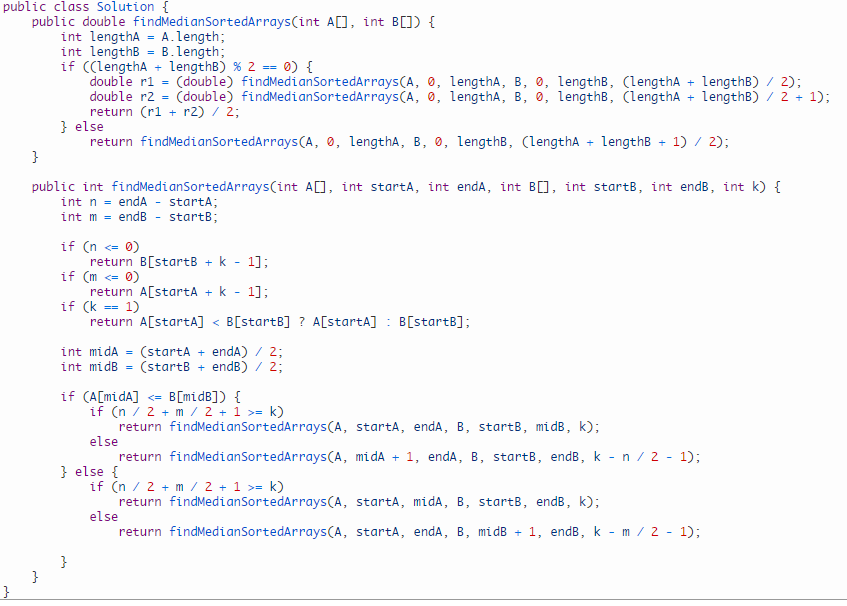
**2）若a<b，则舍弃序列A 中较小的一半，同时舍弃序列B 中较大的一半,要求舍弃的长度相等；**

**3）若a>b，则舍弃序列A 中较大的一半，同时舍弃序列B 中较小的一半，要求舍弃的长度相等；**

**在保留的两个升序序列中，重复过程1）、2）、3），直到两个序列中只含一个元素时为止，较小者即为所求的中位数。**

原因：应该在任何一个数列剩两个元素的时候就停止程序，因为有可能中位数恰好就是这两个元素的平均数，不能舍去。但这时要分析的情况太多，不优。

正确算法

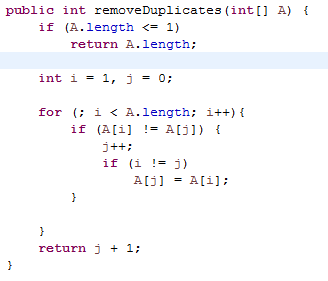


这个算法的主要思想就在于找到一个数，这个数前面有(n+m)/2个数

1. 给定一个有序序列，求不同的元素个数并且返回不同序列，要求原地返回，O(1)空间(26, easy)

15分钟，第一次就AC了略开心，最好记录406ms貌似是前1%！虽然这个时间不靠谱

没啥可优化的了，感觉几乎没有废代码

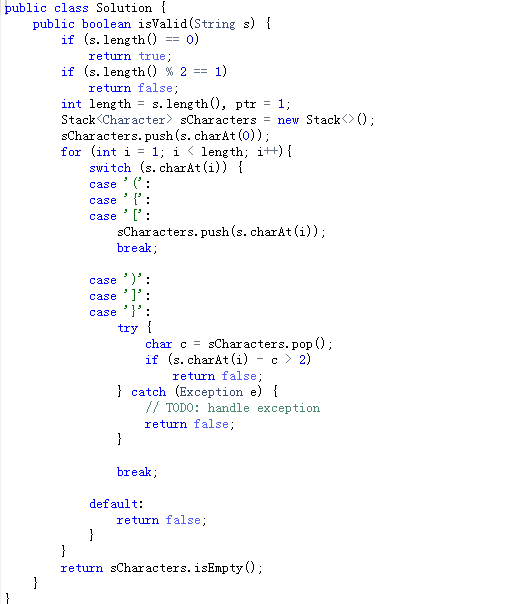


经验？：真的会有公司考这么简单？

1. 括号匹配。(20, easy)

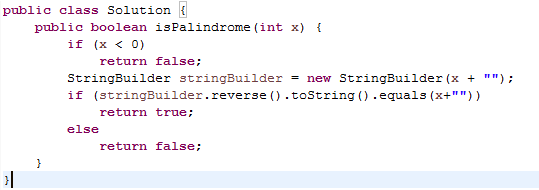
最好记录430ms，前10%。稍微用了点小聪明，不过不好（使用异常做判断）

**经验8：使用Stack比使用数组效率高很多，对这个题而言**

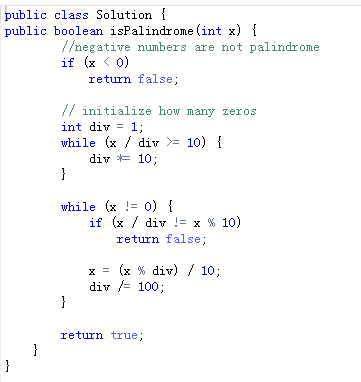


经验？：真的会有公司考这么简单？

1. 判断一个数是不是回文数，不能用额外空间（这点好奇怪，不用额外空间连循环都没法跑了）（9, easy）（投机取巧法）



正规做法：少少优化。大概能跑到前50%，最好记录713ms



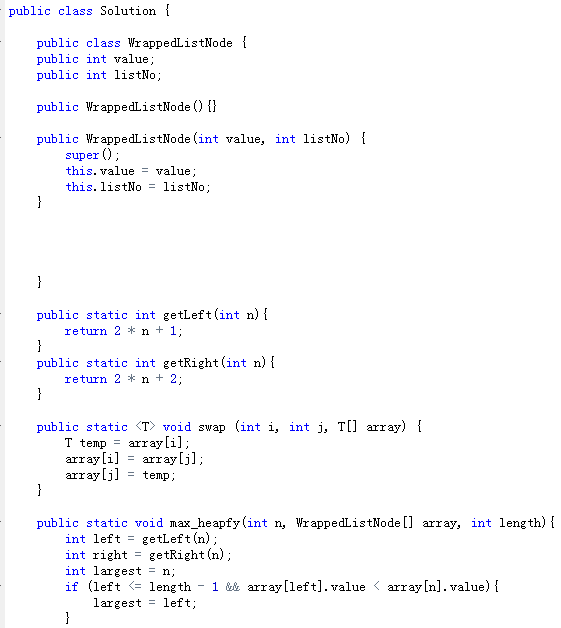
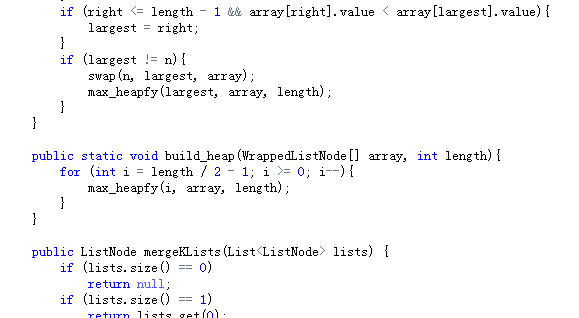
经验？：真的会有公司考这么简单？

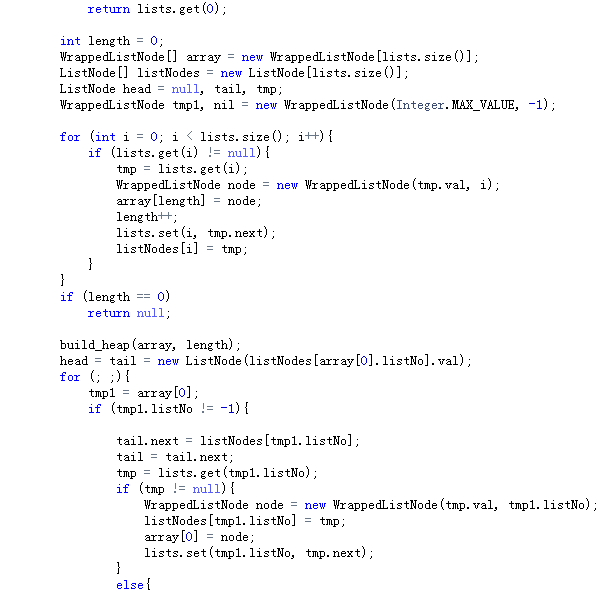
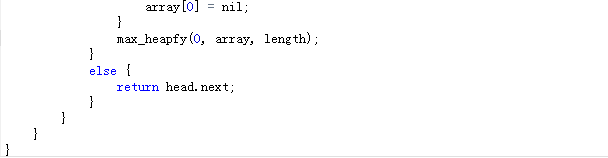
1. 归并K个有序链表。使用堆来做，一开始把K个链表的第一个元素放进数组，然后建堆。之后取出第一个元素（如果这个元素是nil元素，直接退出）放进归并后的链表，再从这个元素所在的链表取第一个元素放到原来元素的位子上，然后重新维护堆性质。如果那个链表已经没有元素，那就插入一个nil元素。

最好貌似50%

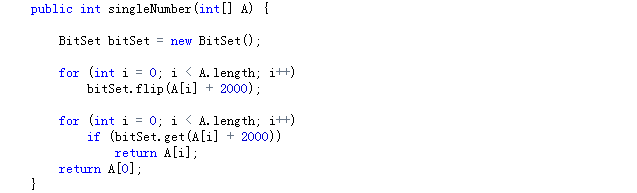
截图累死了。趁机用了用泛型方法，真好使！哨兵nil也不错，感谢算法导论！

经验10：维护堆性质后不能减少length，因为不能保证length-1位的元素一定是nil

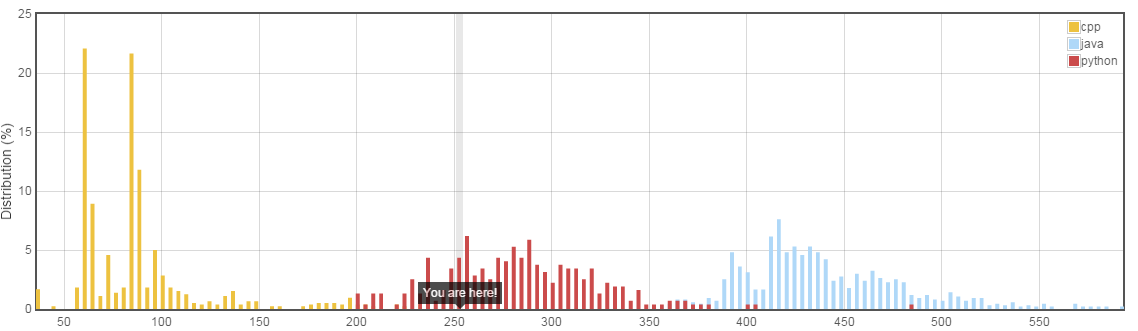
 

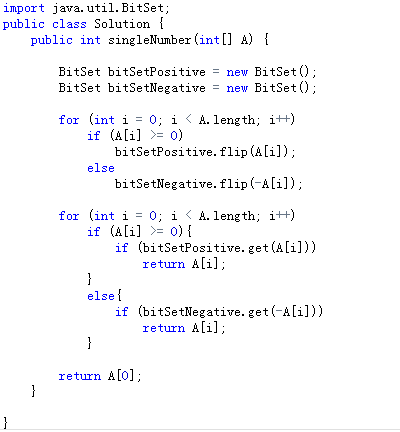
1. 找出一组数里那个单蹦的（其他都是一对）先放一个投机取巧法，十八岁以下严禁模仿！



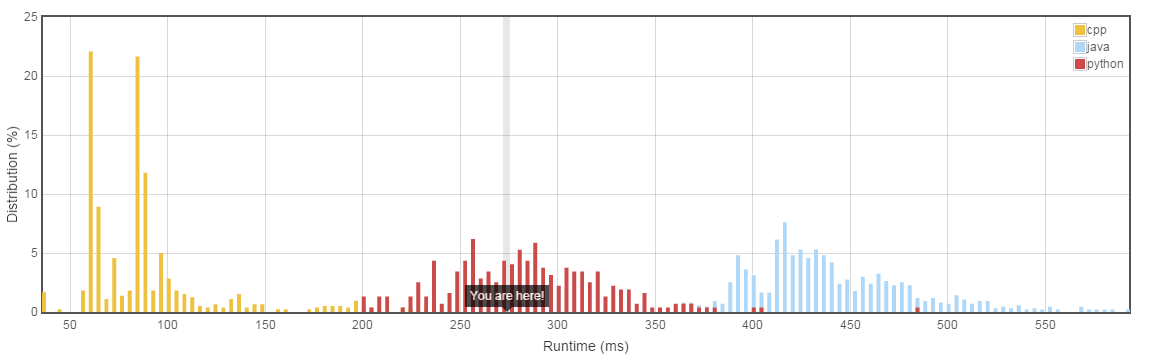
我的运行时间是这样的(Java)，亮爆了有木有！有木有啊！！！



正确做法：使用BitSet，正负分开统计



总觉得时间复杂度没有变高啊？



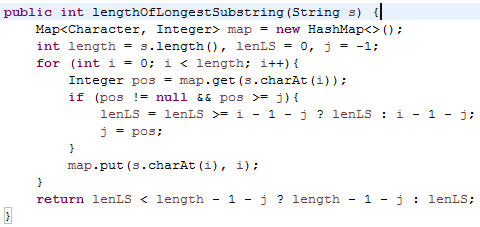
果然，还是算法好，要么就是今天机子跑得格外快！又不是啥特殊算法。。。

**经验11：BitSet好东西，真心好！谁用谁知道！**

1. 找到一个字符串里最长的无重复字符的子串（leetcode第3题，medium）

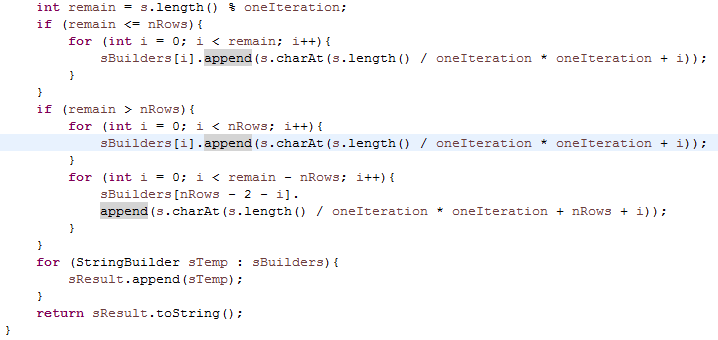
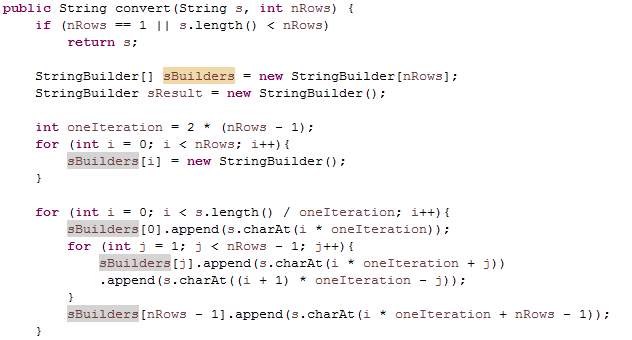
O(N)的算法，时间最好280ms，大概前2%左右，这个算法已经不能更优化了。

大概的思路是：找到一对相同的之后，计算这一对的后面那个与j（上一对的前面那个+1）之间的距离。为了不单独处理所以给了j -1的初始值。（其实这个算法一开始有两个指针，后来我发现好像一个指针也能跑，但是两个到一个指针稍稍有点难说这个区别）



1. 把输入的字符串排成Z字形输出（leetcode第6题，easy），没啥可优化的。

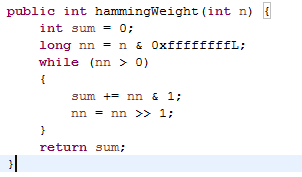
O(N)的算法，时间最好429ms，大概前5%左右。



这就是输入3的时候

1. 求一个无符号数的二进制表示有多少个1.（leetcode第191题，easy）。最好时间

主要难点是位运算吧。而且Java直接给负数略坑啊

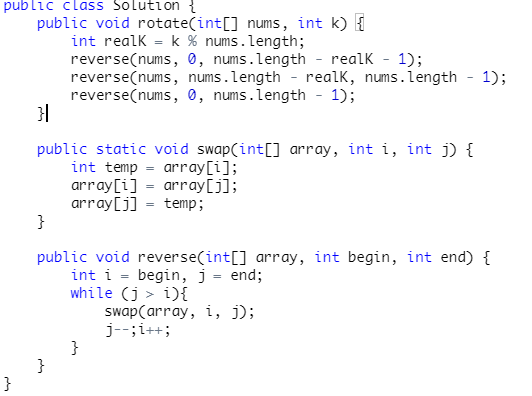


**经验12：取余n等于& (n-1)，当n是2的幂的时候。**

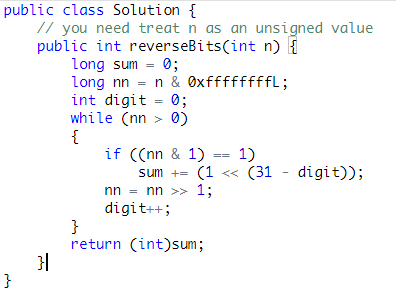
**经验13：long做直接数运算时，直接数后面要加L表示这个数用LONG。**

1. 将数组向右旋转。（LeetCode第189题）

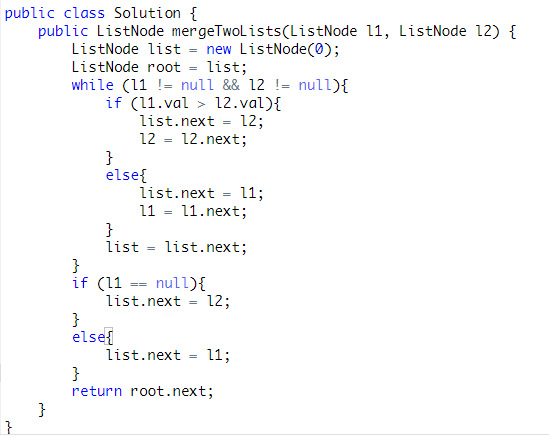
这个算法只使用O(1)的空间，原载于编程珠玑。我只能说，精彩而美妙！



1. 把一个整数的二进制表示倒过来并输出这个新二进制对应的整数。（LeetCode190， easy）。最好成绩300ms，前30%。还是2的幂使用位操作可以大大简化



1. 归并两个链表，leetcode 23，easy。太简单了没啥可说。。。。



1. 在一个数组里寻找所有满足a+b+c=0的a,b,c。LeetCode 15，Medium
   1. 方法一：先用0做pivot，对所有的数进行一次partition，区分开正负数。然后对于每一个数，在符号相反的区域内调用two sum。这个算法可行，时间复杂度理论上是O(N^2)，但是用Set之类的数据结构比较多，拖慢了整体的速度。最后无论怎么优化都没成功。
   2. 方法二：先对数组做快排，然后从最小的数开始用I，j做循环，在J后面用二分查找找第三个数
   3. 方法三：对方法二的优化，二分查找应该返回比要查找的数小的最大的数，这样之后的二分查找能够更快完成，我以后抽空优化~
2. 创建一个Stack，要求push pop top和 getMin都要在常数时间内完成。（LeetCode155， easy）

主要想法：两个栈一个存最小值一个是普通栈，最小栈要求只压入不大于栈顶元素的元素。出栈时若两站栈顶元素相同则同时pop，否则只pop普通栈的元素

1. 在一列数里寻找三个数a, b和c，要求a+b+c最接近target（Leetcode16, Medium）

把3sum的binarySearch改成二分查找最接近的元素就好了，整体算法不变。不跳过重复元素。

1. 删去一个list里倒数第n个，one-pass (LeetCode19, Easy)

三个指针一前一后一头，搞定

1. 一个排好序的数组被右移了，搜索一个数字
   1. 线性搜索，不说了，投机取巧. O(N)
   2. 线性搜索到分界线，在分界线后面二分搜索。也是O(N)
   3. 改进后的二分搜索：分类讨论思想，有七种情况
      1. 如果mid和target一样，那不用找了，return
      2. 如果mid比target小，分三种情况讨论
         1. Mid比right小，target也比right小，说明target在以mid+1为起始，right为终止的已排序序列里，直接调用标准二分查找即可。
         2. Mid比right小，target比right大，说明mid右侧的上升序列里没有target，应该在左侧继续查找：right=mid-1;
         3. Mid比right大，说明mid左边的上升序列里没有target，应该在右侧继续查找：left=mid+1；

2和3都不能使用标准二分查找，因为序列不是排好序的.

* + 1. 如果mid比target大，分三种情况讨论，和上面三种情况差不多

这个方法所有的情况都能保证O(LgN)的时间复杂度，应该是最优了。