**暑期算法训练**

**第一部分：链表：**

1. 翻转链表：

①简要实现思路：

以头节点为标记点，首先将标记点后节点断开，然后将此节点下一个节点连接上首节点，再将标记点与此节点的下一个节点连接，重复操作直到标记点变成末节点即可：如下图：

--->起始链表：

1. ->2-->3-->4-->5

--->第一次交换：（将标记点后节点与首节点交换）

标记：**1**-->**2**-->3-->4-->5

交换：**2-**->**1**-->3-->4-->5

--->第二次交换：（将标记点后节点与下一节点断开并且连接上首节点）

标记：**2-**->**1**-->**3**-->4-->5

交换：**3**-->**2-**->**1**-->4-->5

--->第三次交换：（将标记点后节点与下一节点断开并且连接上首节点）

标记：**3**-->2**-**->**1**-->**4**-->5

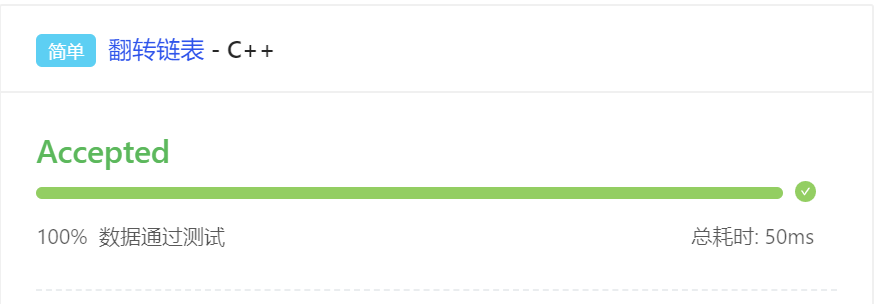
交换：**4**-->**3**-->2-->**1-**->5

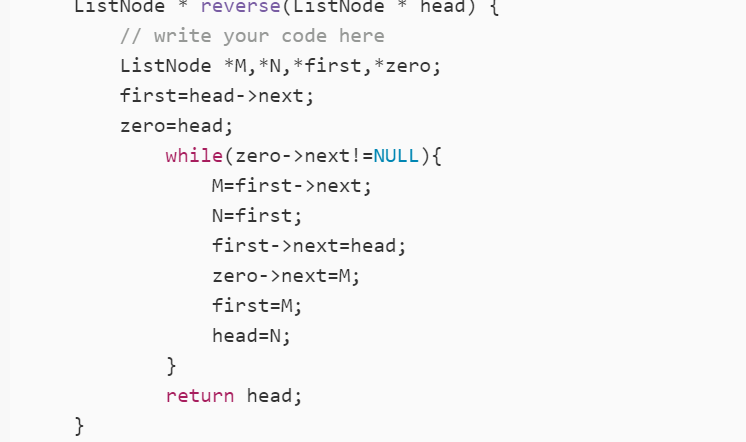
--->第四次交换：（标记点变成末节点，翻转结束）

标记：**4**-->3-->2-->**1**-->**5**

交换：**5**-->**4**-->3-->2-->**1**

②测试及代码截图：





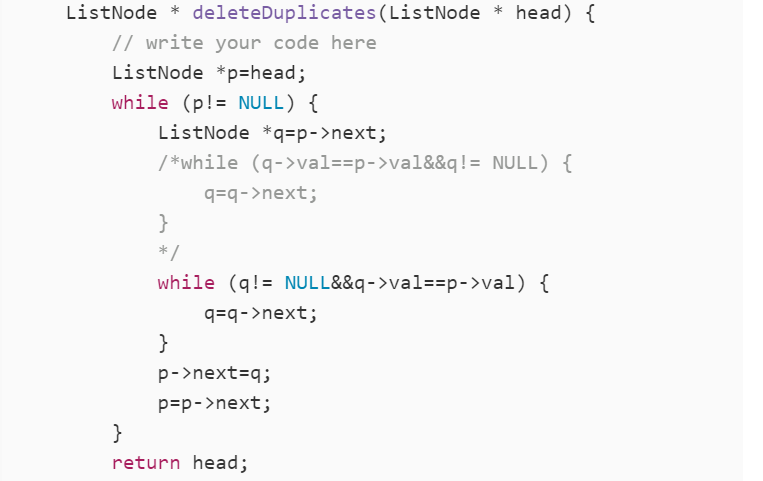
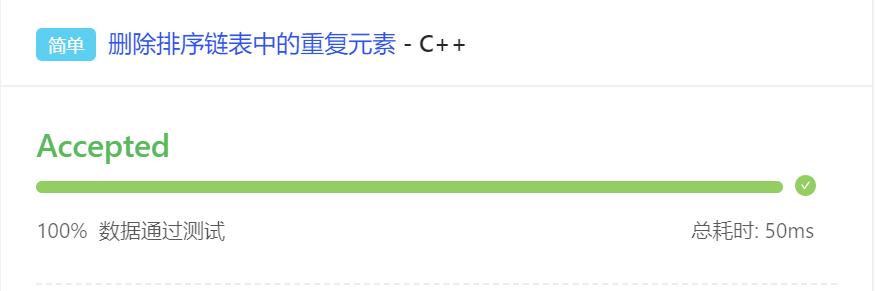
1. 删除排序链表中的排序元素：

（给定一个排序链表，删除所有重复的元素每个元素只留下一个）

①简要实现思路：

遍历全部链表，检测如果当前节点与下一个节点的val相同，则将当前节点指向下下个节点以删除重复节点，当节点为null时退出即可完成。

②测试及代码截图：



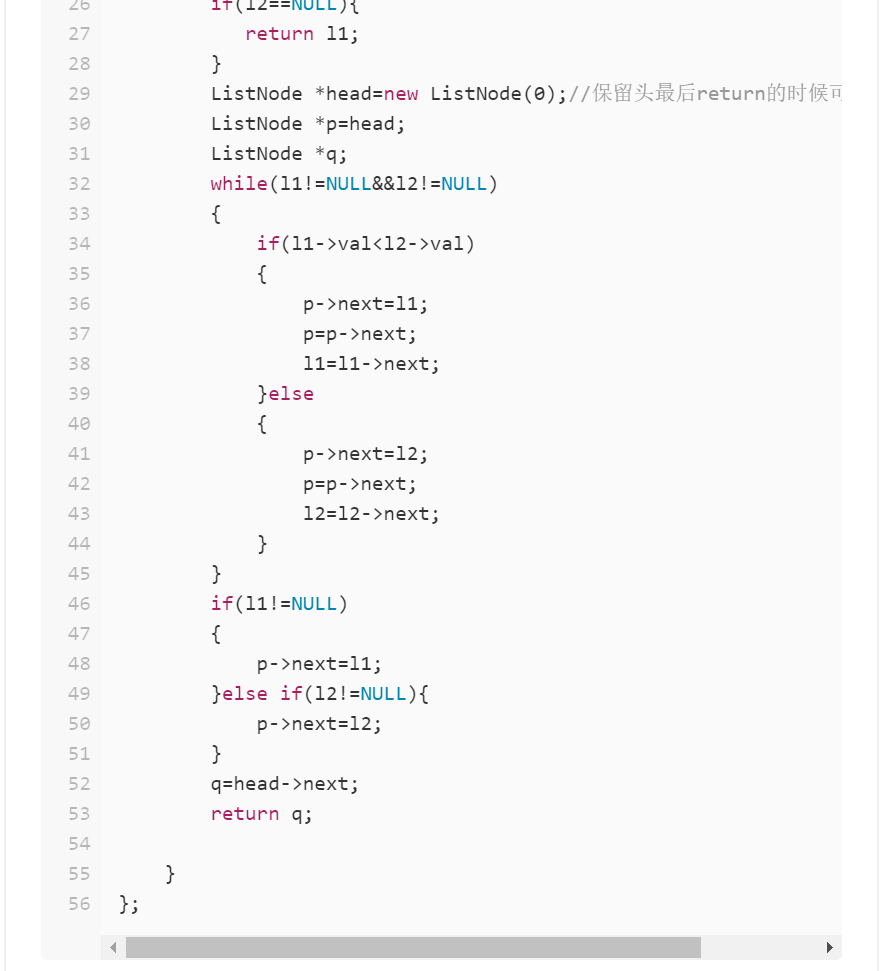
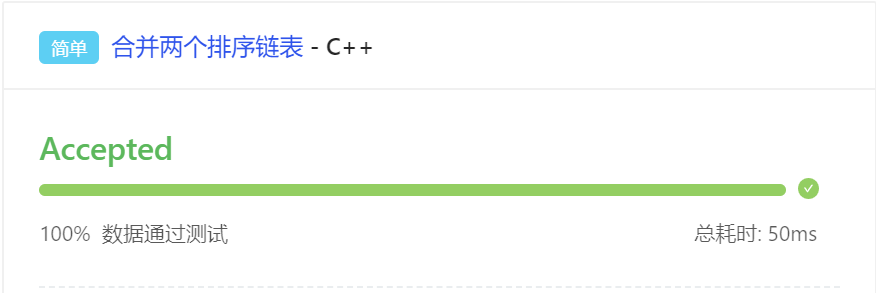
1. 合并两个排序链表：

（将两个排序链表合并为一个新的排序链表）

①简要实现思路：

New一个新的链表，循环遍历两个排序链表，依次比较两个链表节点元素的值，依次排序填入新链表即可。

②测试及代码截图：



1. 链表倒数第n个节点：

（找到单链表倒数第n个节点，保证链表中节点的最少数量为n。）

①简要实现思路：

两次循环，第一次得到链表大小，第二次根据题目要求返回对应节点即可。

②测试及代码截图：



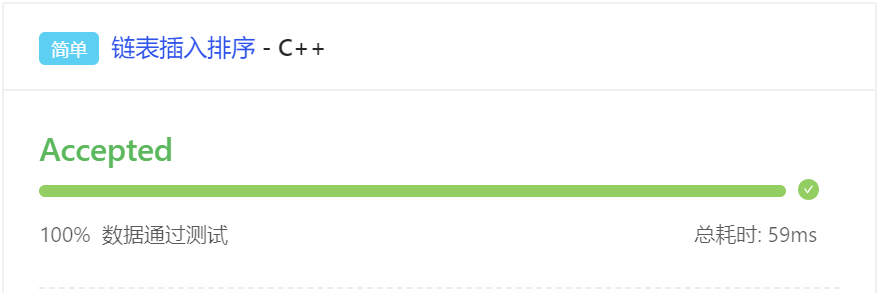
1. 链表插入排序：

（用插入排序对链表排序）

①简要实现思路：

New一个新链表，将原待排序链表中的元素一个一个插入，一开始链表为空，将原链表头直接插入并且直接置空，每次遍历整个新链表寻找插入节点该放入的位置，直到待排序链表节点全部插入结束为止，最后返回新链表即可。

②测试及代码截图：





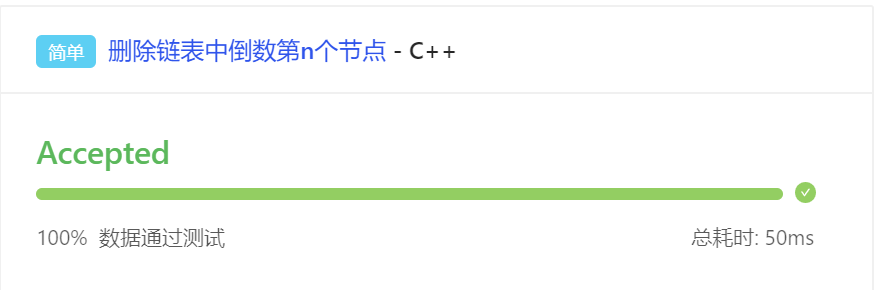
1. 删除链表中倒数第n节点

（给定一个链表，删除链表中倒数第n个节点，返回链表的头节点）

①简要实现思路：

首先遍历整个链表确定链表节点数量，计算出位置后，再次循环到要删除的结点处，将上一节点的next指针指向被删除节点的下一结点即可。

②测试及代码截图：



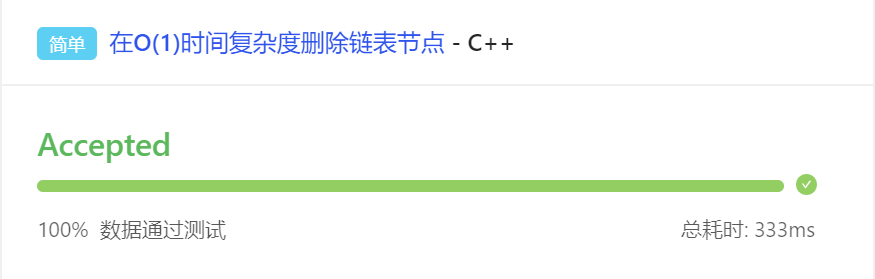
1. 在O（1）时间复杂度内删除链表节点

（给定一个单链表中的一个等待被删除的节点(非表头或表尾)。请在在 O(1) 时间复杂度删除该链表节点。）

①简要实现思路：

由于输入只有一个节点，所以只需要将此节点的下一个结点的val赋值给此节点，再将此节点的next指针指向下下个节点即可（删除这个节点），如果要删除的节点为空，则不进行任何操作。

②测试及代码截图：



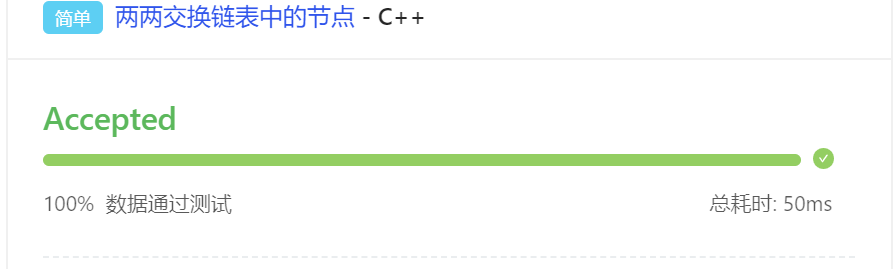
1. 两两交换链表中的节点

（给一个链表，两两交换其中的节点，然后返回交换后的链表）

①简要实现思路：

New一个新链表用于保存返回后的链表，遍历整个原链表，首先保存操作节点两个以后的节点，然后将这两个节点与原链表后续节点断开，再将这两个节点逆序后插入到新链表中，依次操作直到链表遍历结束即可。

②测试及代码截图：



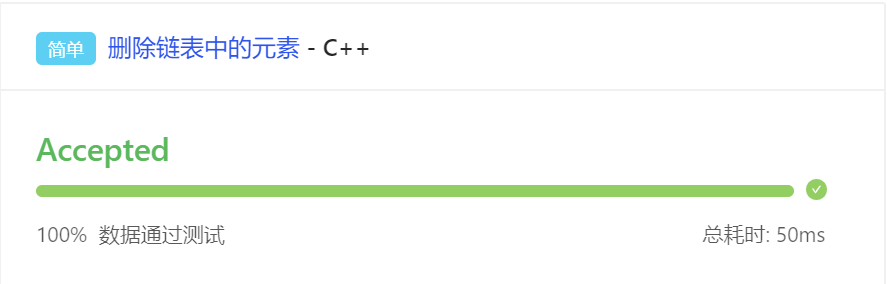
1. 删除链表中的元素

（删除链表中等于给定值 val 的所有节点）

①简要实现思路：

New一个新链表，依旧采用插入法，将原链表每一个节点先断开，再去比较是否值为val，如果不是则插入到新链表，否则直接跳过，遍历结束即可。

②测试及代码截图：



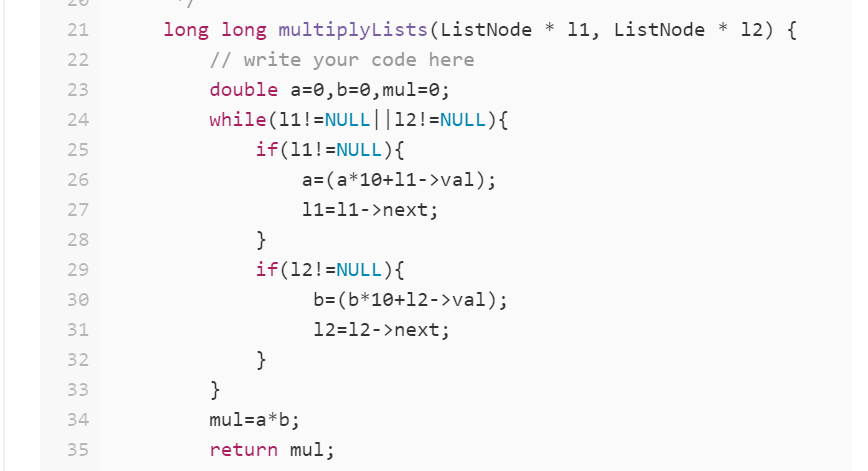
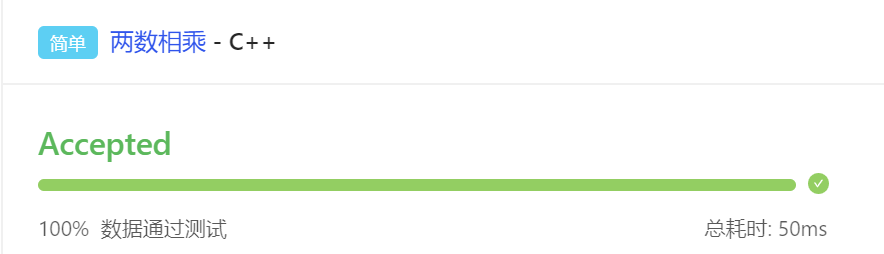
1. 两数相乘

（给出两个链表形式表示的数字,写一个函数得到这两个链表相乘乘积）

①简要实现思路：

依次遍历两个链表，将其值取出组成十进制数字，最后相乘即可。

②测试及代码截图：



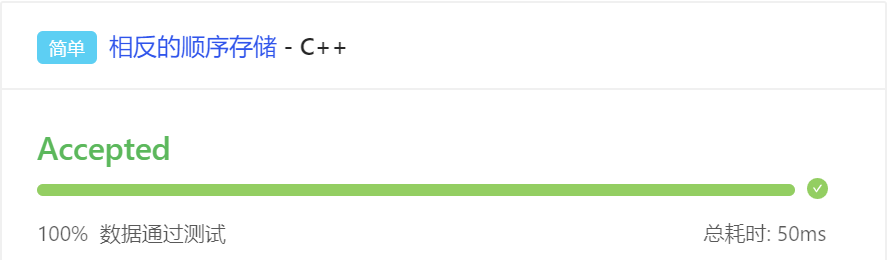
1. 相反的顺序储存

（给出一个链表，并将链表的值以****倒序****存储到数组中）

①简要实现思路：

首先遍历整个链表得到链表大小，然后new一个int类型的数组，再次遍历链表，将链表每一个节点的val逆序储存到数组中，最后将数组中的元素push\_back进入一个vector类型数组中正序输出即可。

②测试及代码截图：



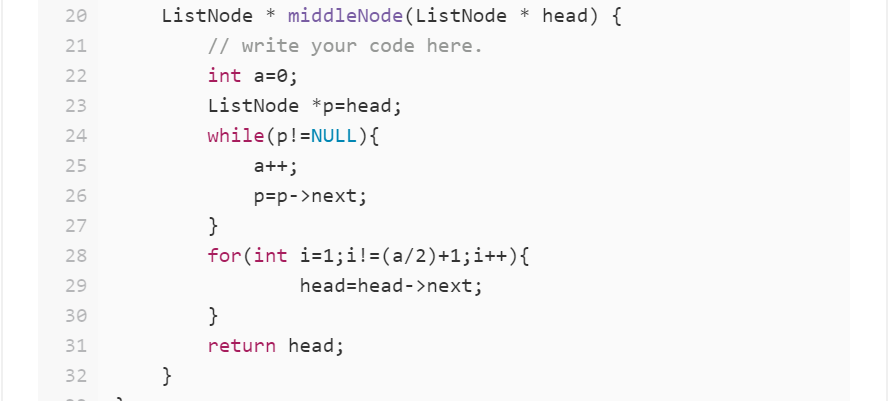
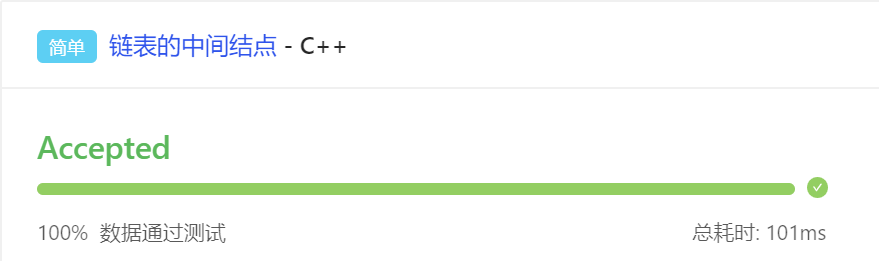
1. 链表中间节点

（给定一个带有头结点 head 的非空单链表，返回链表的中间结点。如果有两个中间结点，则返回第二个中间结点。）

①简要实现思路：

首先遍历整个链表得到链表大小，然后直接计算出来中间节点位置，再一次循环返回此位置节点即可。

②测试及代码截图：



**第二部分：数组：**

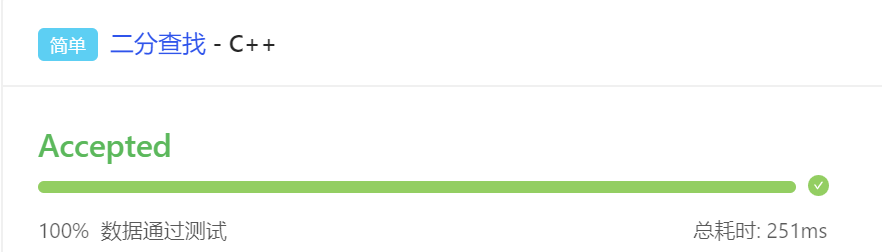
1. 二分查找：

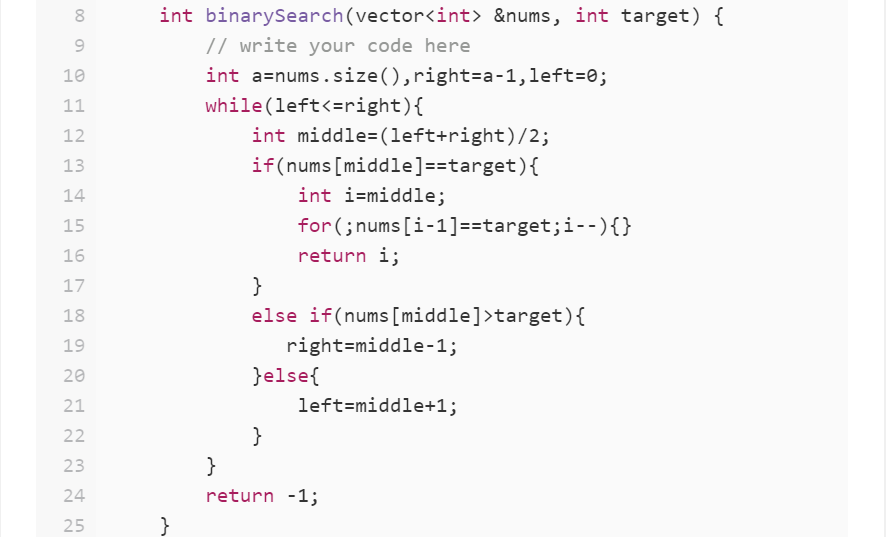
（给定一个排序的整数数组（升序）和一个要查找的整数target，用O(logn)的时间查找到target第一次出现的下标（从0开始），如果target不存在于数组中，返回-1。）

①简要实现思路：

由于数组已经排序，每次将数组中间位置元素与target比较，若是小于target则将左边界重置为中间位置的右一个数字，若是大于target则将右边界重置为中间位置的左一个数字，依次继续二分下去，如果target等于中间值，由于要求要为第一次出现的target，所以从中间值向左递减，直到找到第一个序号即可。

②测试及代码截图：





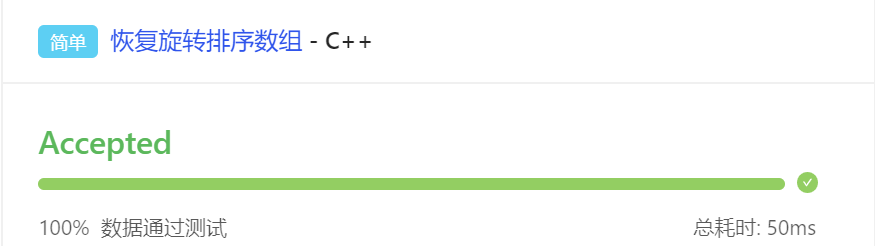
1. 恢复旋转排序数组：

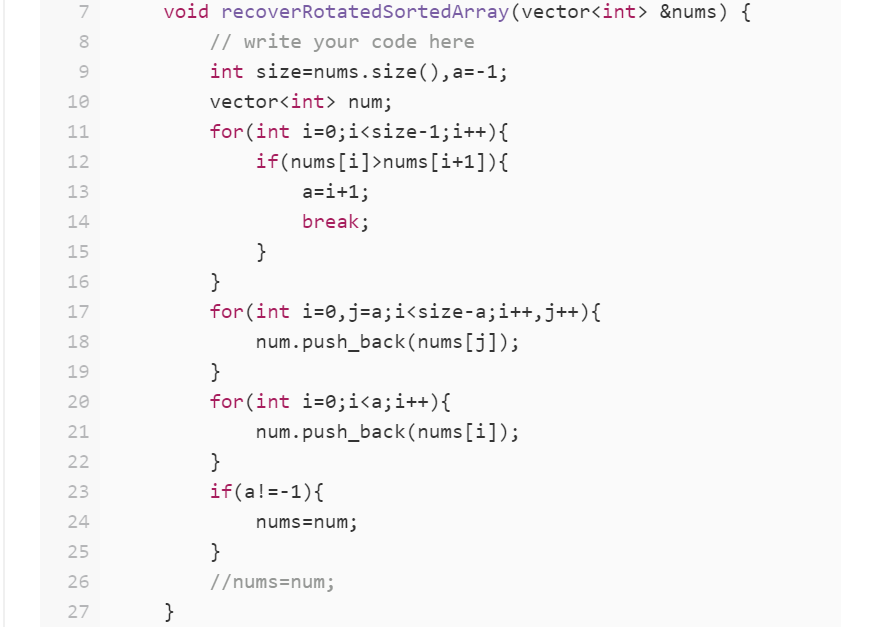
（给定一个****旋转****排序数组，在原地恢复其排序（升序））

①简要实现思路：

由于是要求原地升序，所以先定义一个vector数组，然后循环原数组找到旋转排序的断点，再按照从大到小的顺序将后部分先push进入新数组，再将数组前部分push进去，最后由于要求“原地”，所以将新数组赋值给原数组。

②测试及代码截图：





1. 两数之和：

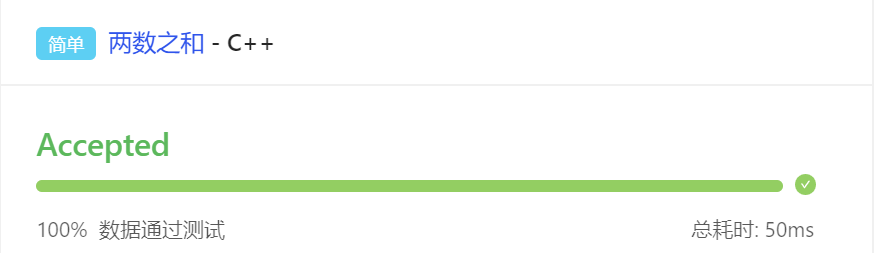
（给一个整数数组，找到两个数使得他们的和等于一个给定的数 target

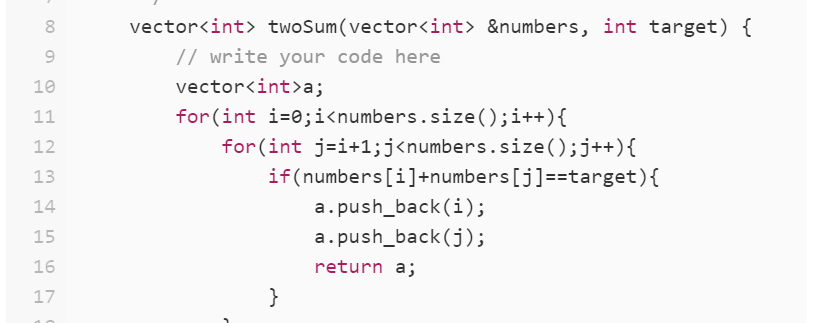
你需要实现的函数twoSum需要返回这两个数的下标, 并且第一个下标小于第二个下标。注意这里下标的范围是 0 到 n-1）

①简要实现思路：

分别对两个数组进行遍历即可，由于要求第一个下表小于第二个下标，所以第二个for循环从i+1开始，一旦找到满足target的序号直接返回即可。

②测试及代码截图：





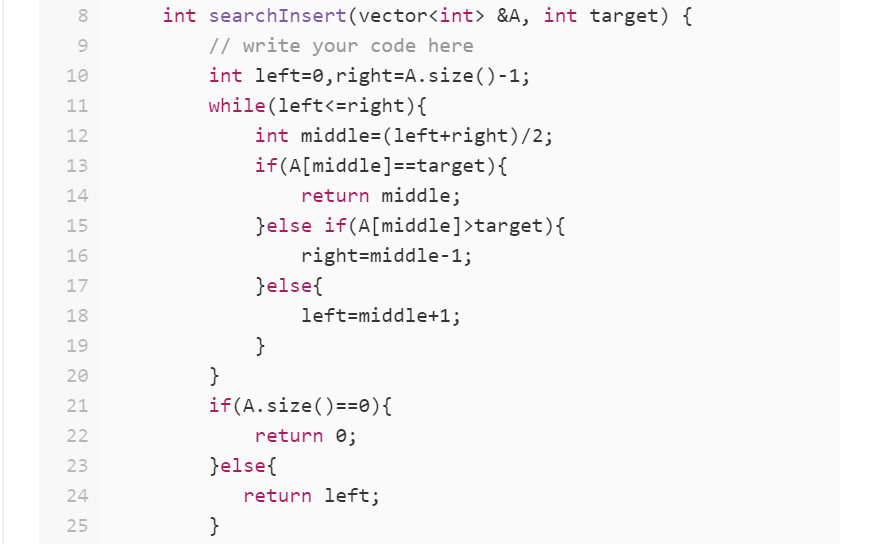
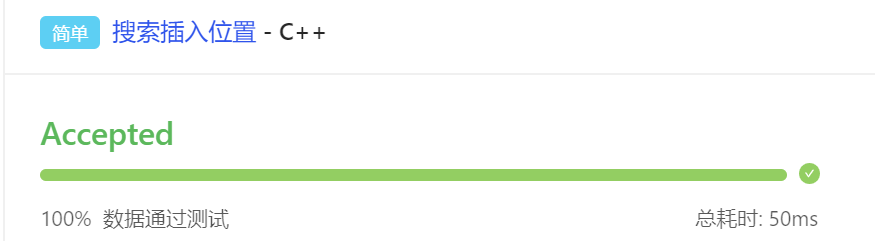
1. 搜索插入位置：

（给定一个排序数组和一个目标值，如果在数组中找到目标值则返回索引。如果没有，返回到它将会被按顺序插入的位置）

①简要实现思路：

使用二分法查找即可，如果无目标值，则需要将目标值插入二分法结束时退出的位置。

②测试及代码截图：



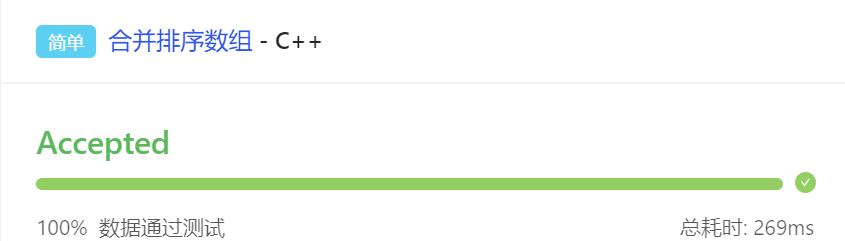
1. 合并排序数组：

（合并两个排序的整数数组A和B变成一个新的数组）

①简要实现思路：

由于A数组最后是要被返回的数组，所以就先把A数组的值放入一个新的a数组，然后对a数组与B数组进行遍历，一个一个比较其大小，然后依次插入A数组中，最后返回A数组即可。

②测试及代码截图：





1. 中位数：

（给定一个未排序的整数数组，找到其中位数）

①简要实现思路：

此处中位数很简单找，重点是排序算法，我百度查到了一种快速排序算法，如果用已经学的冒泡等排序算法速度会很慢，当然这里也可以直接调用lintcode自带的sort函数进行排序，不过还是侧重解释一下网上查到的快速排序算法（分治法）：**https://blog.csdn.net/hurmishine/article/details/53769531**

该方法基本思像就是：

1．先从数列中取出一个数作为基准数。

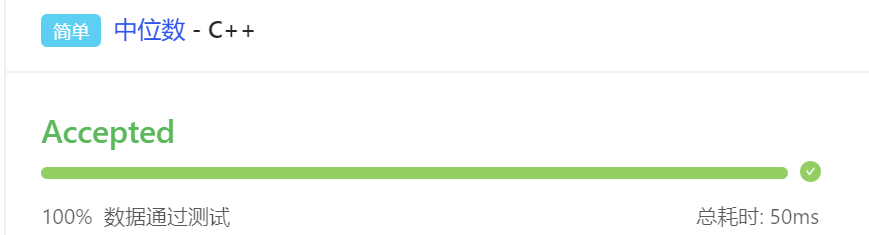
2．分区过程，将比这个数大的数全放到它的右边，小于或等于它的数全放到它的左边。

3．再对左右区间重复第二步，直到各区间只有一个数。

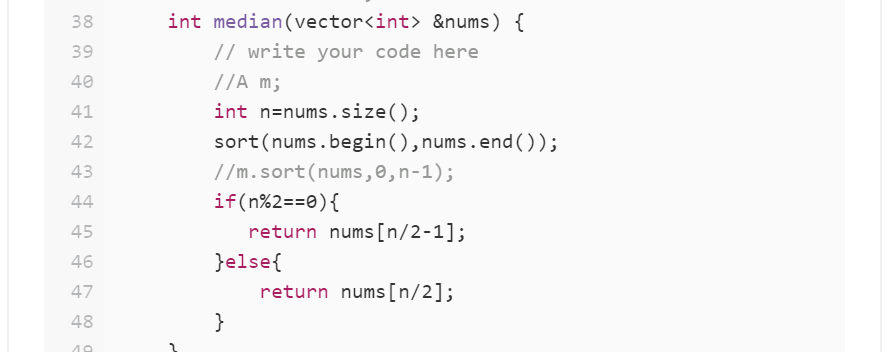
具体实现就是通过**挖坑法**，首先从左边随便确定一个基准数，此时在左侧就形成了一个坑，再从数组右侧向左找一个比基准数小的数字，找到以后填入之前基准数的位置去填充那个坑，此时在右侧又形成了一个坑，再从左往右找一个比基准数大于等于的数字，找到以后填到右侧形成的那个坑，直到遍历到左右运动指针指向同一个位置，再将基准数放置于这个位置即可完成一次分区。

再递归调用，分别对上一次分区结束的左边进行挖坑填入，然后对右边进行挖坑填入，直到左右两个区都只剩下一个值则结束排序。

②测试及代码截图：



**直接sort代码：**



**自行排序代码：（最终二者耗时一致）**



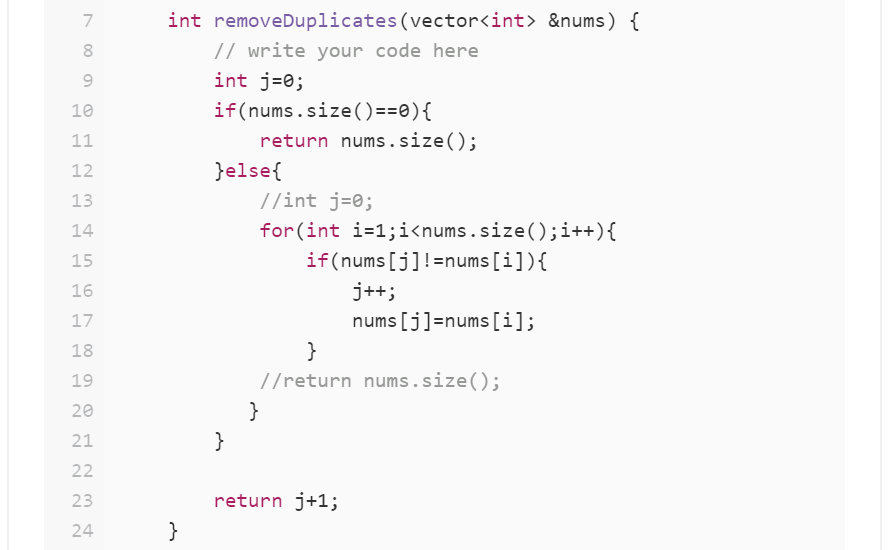
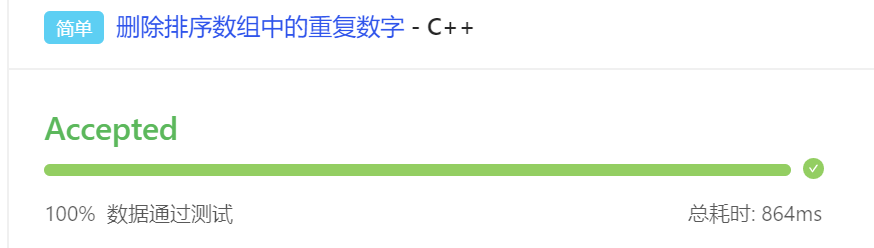
1. 删除排序数组中的重复数字：

（给定一个排序数组，在原数组中“删除”重复出现的数字，使得每个元素只出现一次，并且返回“新”数组的长度，不要使用额外的数组空间，必须在不使用额外空间的条件下原地完成）

①简要实现思路：

此处借鉴了笔记区同学们的“双指针法”。使用一个慢指针和一个快指针同时进行遍历，如果快指针所在的值与慢指针相同则不移动慢指针，否则将慢指针加一同时赋值为快指针的值，以此达到消除在一起的重复数字的目的。

②测试及代码截图：



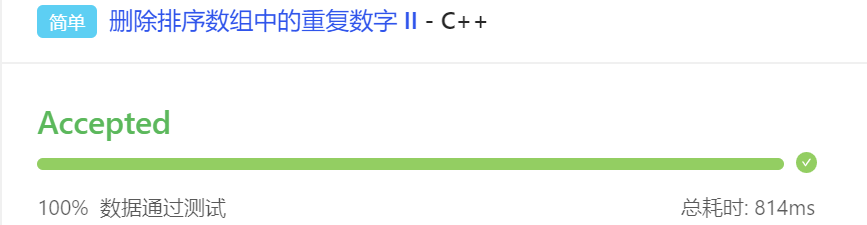
1. 删除排序数组中的重复数字（2）

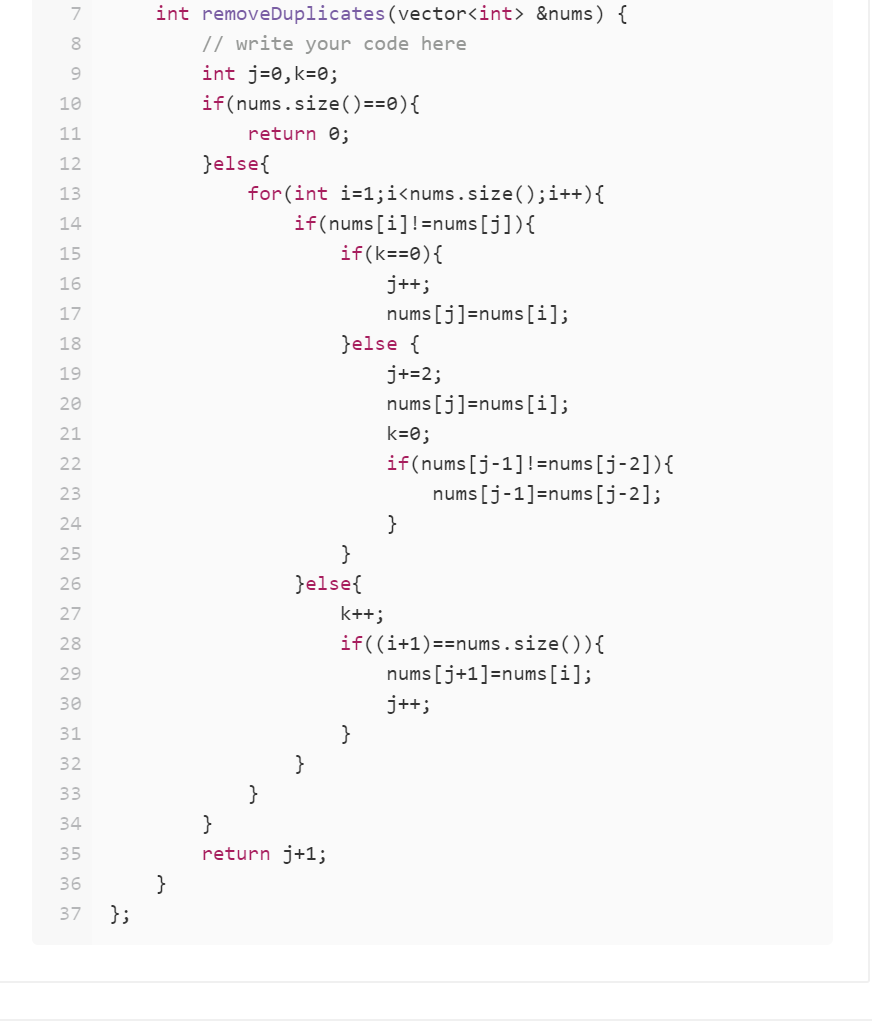
（给你一个排序数组，删除其中的重复元素，使得每个数字最多出现两次，返回新的数组的长度，如果一个数字出现超过2次，则这个数字最后保留两个）

①简要实现思路：

根据上一题的分析，此处依旧使用双指针法，只不过需要一个k来计数，因为最多不超过2个重复数字，因此有略微变化，若是k为0，则j++，否则j+=2。

②测试及代码截图：





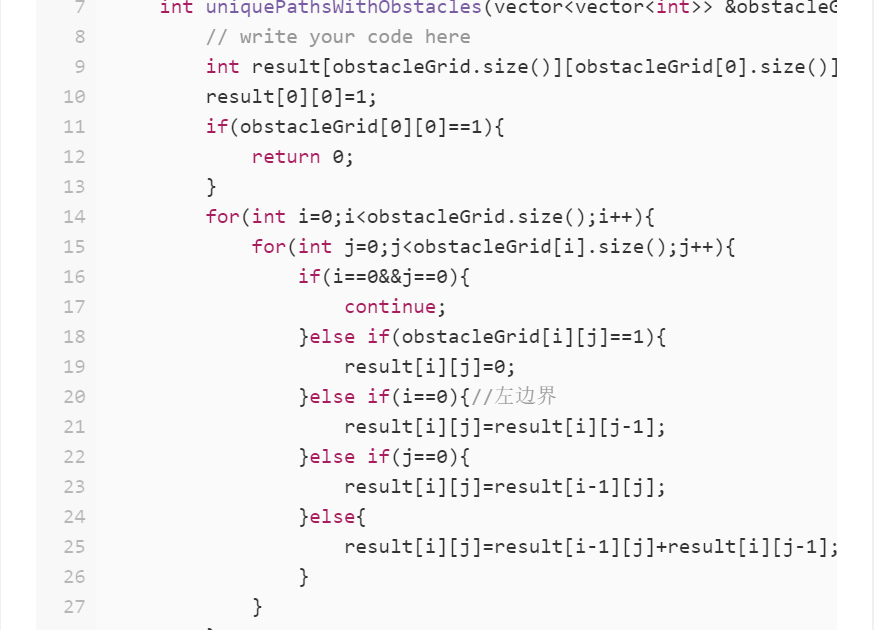
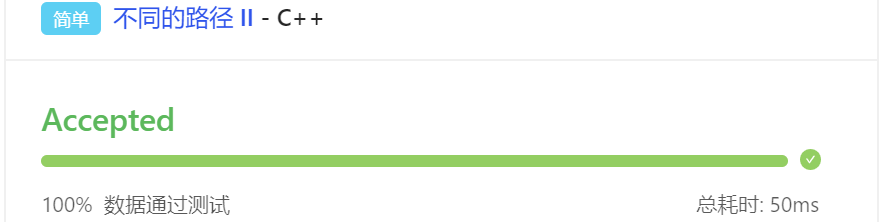
（9）不同的路径||

（"[不同的路径](http://www.lintcode.com/problem/unique-paths/" \o "不同的路径" \t "https://www.lintcode.com/problem/unique-paths-ii/_blank)" 的跟进问题：现在考虑网格中有障碍物，那样将会有多少条不同的路径？网格中的障碍和空位置分别用 1 和 0 来表示）

①简要实现思路：

此处参考了笔记区的同学的解答，由题可知，网格中每一个位置均可从左侧或者上侧达到，依次类推，也就是说只要得到左侧位置和上侧位置可以有多少路径得到，再相加即可得到当前位置有多少路径。由此拓展到全网格，全局遍历一次即可。

②测试及代码截图：



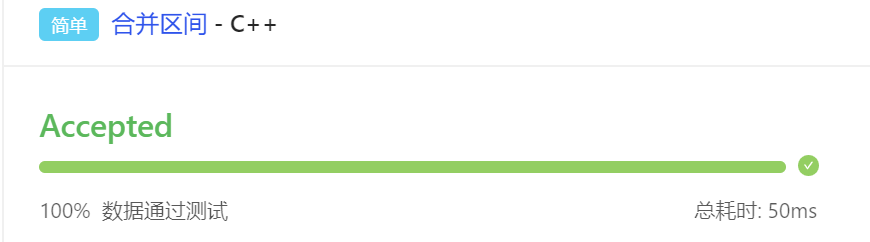
（10）合并区间：

（给出若干闭合区间，合并所有重叠的部分）

①简要实现思路：

首先将所有区间按照左端点大小排序（用冒泡算法会大大耽误时间，所以采用自带的sort算法，但是sort函数的判断大小功能以及不能使用了，所以要编写一个判断函数，只根据括号左边大小去进行判断），然后定义一个interval类型的变量用于储存每一个区间，首先直接把左端点赋值进去，然后再去遍历接下来的区间，然后可以合并就进行合并，不能就直接确定右端点，再push进result数组，遍历一遍即可。

②测试及代码截图：



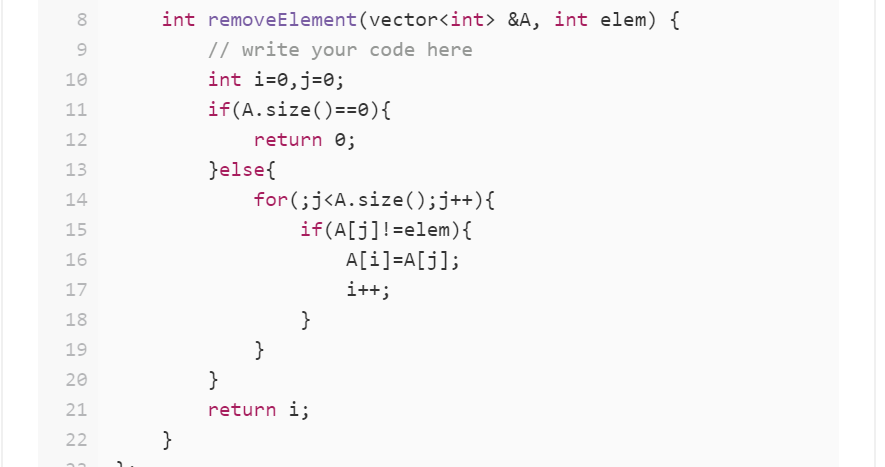
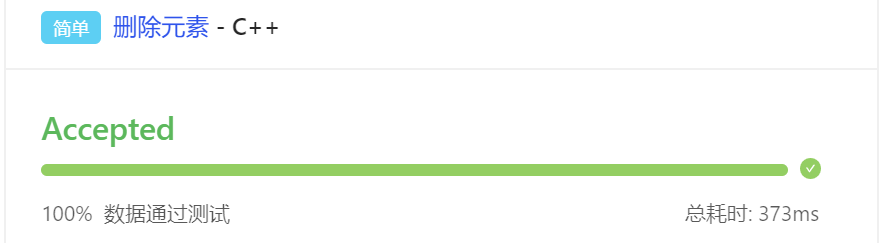
1. 删除元素

（给定一个数组和一个值，在原地删除与值相同的数字，返回新数组的长度，元素的顺序可以改变，并且对新的数组不会有影响）

①简要实现思路：

依旧用双指针法，i指针保留当前位置，j指针进行遍历，如果j遇到的数字与elem不相同，则i位置的值不变，且i自加，否则i不变进而保证最后得到的数组数目与目标相同。

②测试及代码截图：



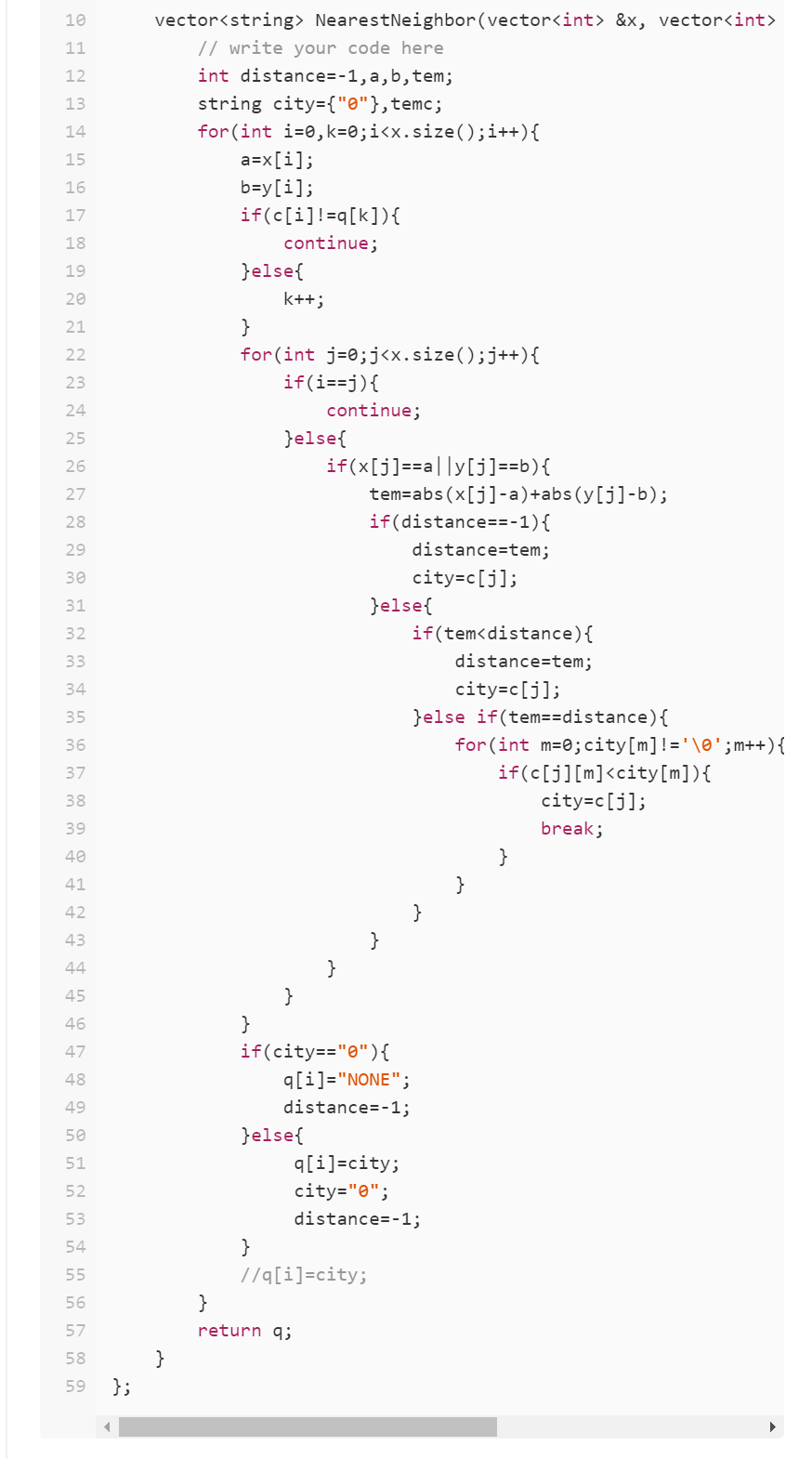
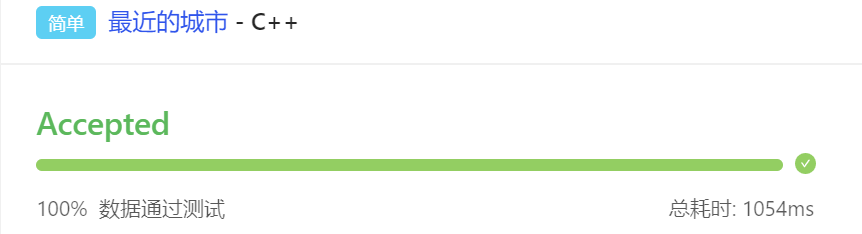
1. 最近的城市

（在一个二维平面上有许多城市，所有的城市都有着自己的名字c[i]*c*[*i*]，以及位置坐标(x[i],y[i])(*x*[*i*],*y*[*i*])（都为整数）。有q*q*组询问，每组询问给出一个城市名字，你需要回答离该城市距离最近的，x*x*相同或者y*y*相同的城市名称）

①简要实现思路：

首先要将城市和被询问的位置对应起来，也就是说q[k]==c[i]，因为距离计算的for循环是以c[i]的i来定位的，要避免自己城市纳入计算范畴。然后就定义一个string类型的city和一个distance，对城市坐标进行遍历（同时用上述方法取除自己城市），在满足x或者y相同的情况下，若是distance较小则直接替换并且重新标记city，如果distance相同则对这两个城市名称进行判断取字典序最小的那个，遍历结束后将city直接push进入结果数组，如果找不到城市与他x或者y相同，则push“NONE”，最后输出即可。

②测试及代码截图：



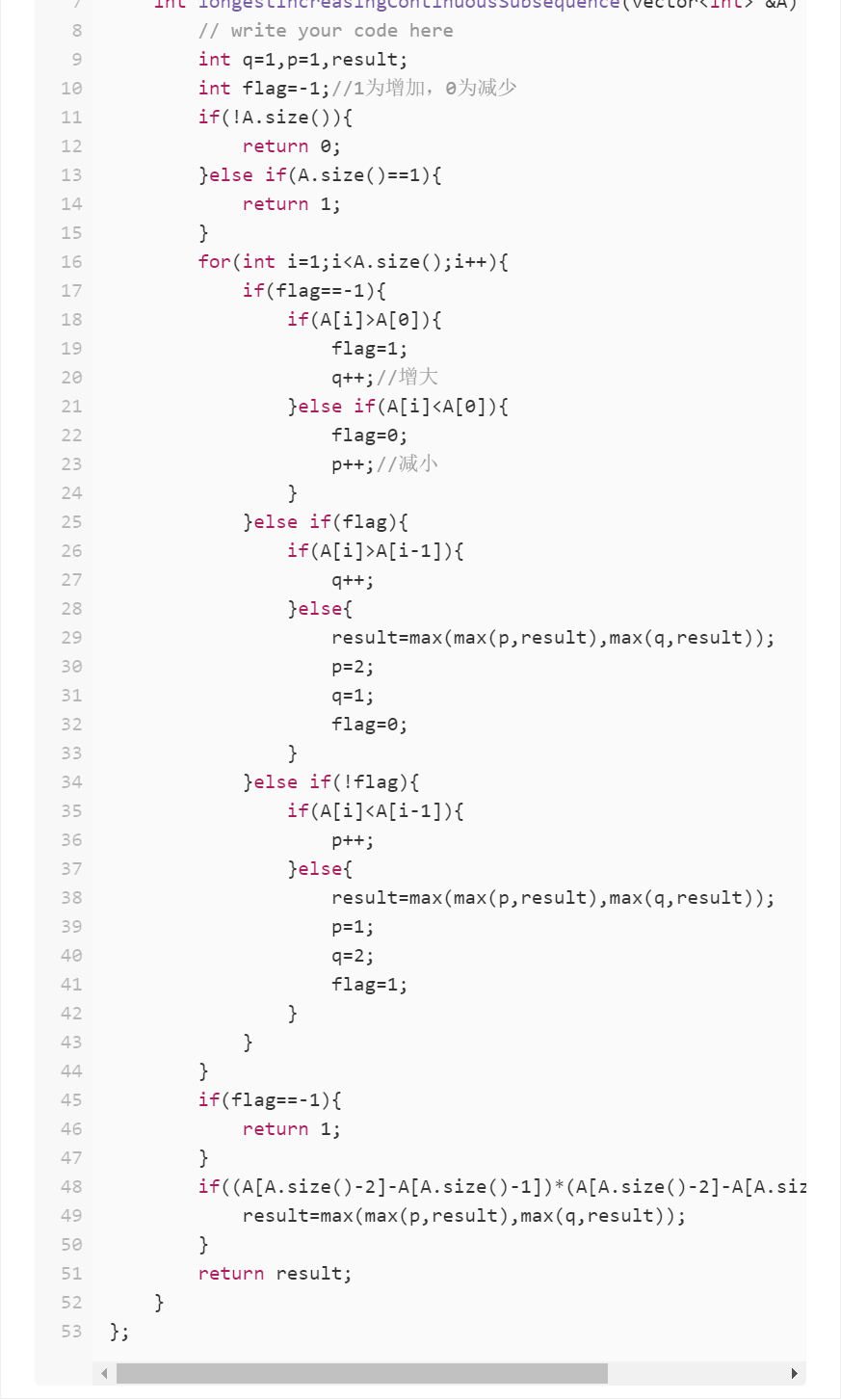
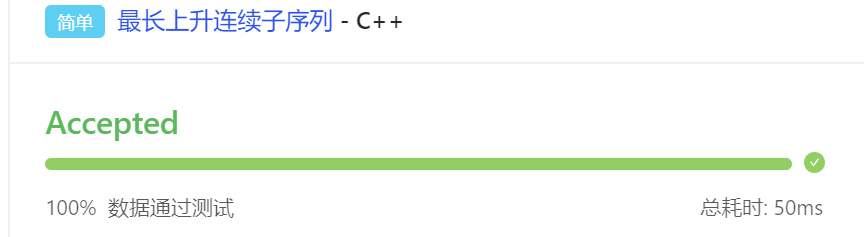
1. 最长上升连续子序列

（给定一个整数数组（下标从 0 到 n-1， n 表示整个数组的规模），请找出该数组中的最长上升连续子序列。（最长上升连续子序列可以定义为从右到左或从左到右的序列））

①简要实现思路：

首先使用flag来检测前面序列是递增还是递减，若是递增则将q自增，若是递减则p自增，这两个用于检测前面的子序列递增或者递减的数量。若下一个数字不满足前面序列的增减变化，则此时将结果与result对比，将较大者存入result，然后重置q，p继续遍历直至结束。由于存在一直递增或者递减的情况，这种情况下result还未被赋值，所以设置最后一个if，如果数列单增或者单减，则return一个p，q，result的最大值。

②测试及代码截图：



1. 加一

（给定一个非负数，表示一个数字数组，在该数的基础上+1，返回一个新的数组。该数字按照数位高低进行排列，最高位的数在列表的最前面）

①简要实现思路：

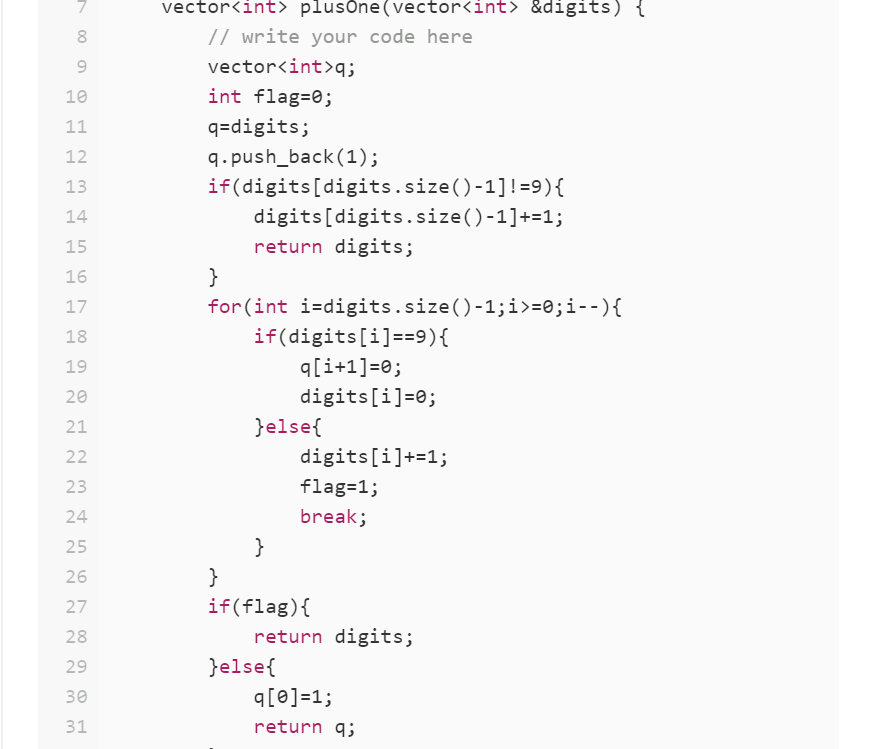
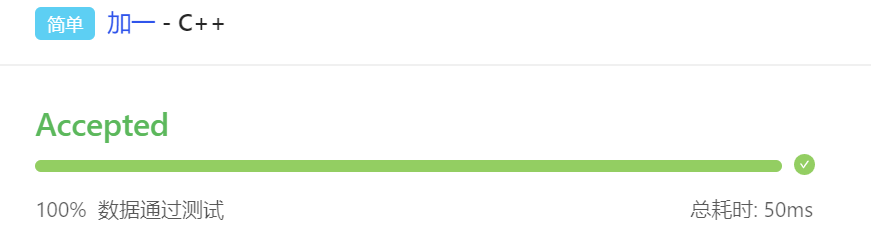
分三种情况讨论，

第一种情况：如果数字数组的最低位不为9，则直接将最低位加一返回即可。

第二种情况：如果数字数组全为9，则建立一个新数组，并且将最高位赋值为1，其他位全部为0（个数为原数组的大小）。

第三种情况：某一个数字为9，且不是最低位，则找到这个数字并且将上一位加一，自身变成0即可。

②测试及代码截图：



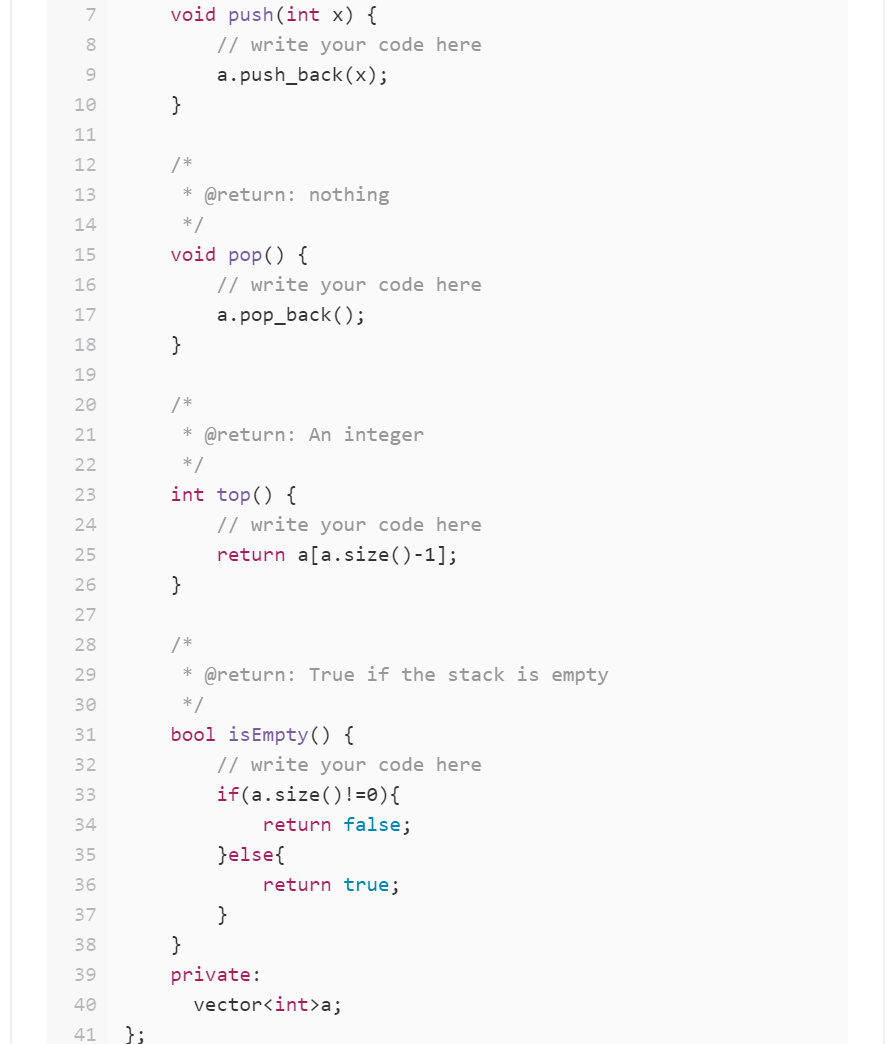
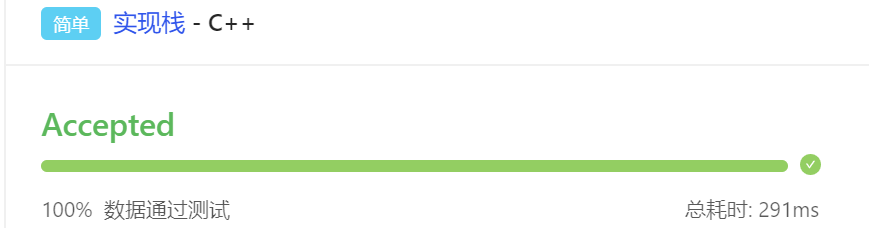
1. 实现栈

（实现一个栈，可以使用除了栈之外的数据结构）

①简要实现思路：

直接用数列的对应函数代替即可。

②测试及代码截图：



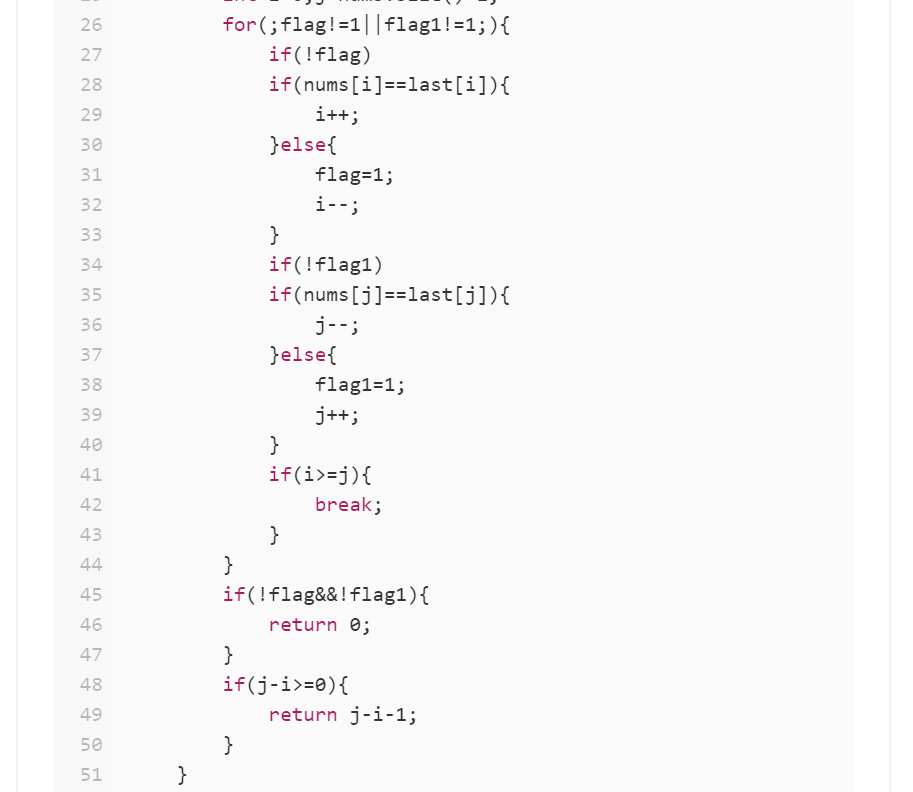
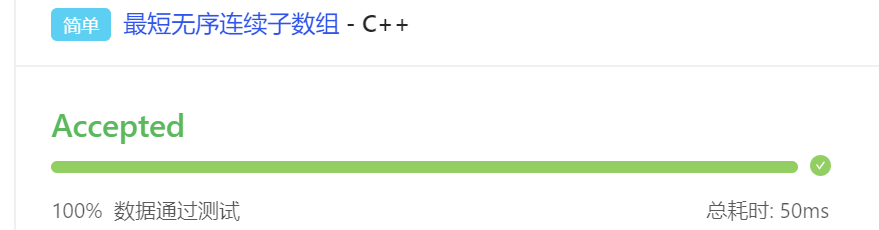
1. 最短无序连续子数组

（给定一个整数数组，你需要找到一个****连续子数组****，如果你只按升序对这个子数组进行排序，那么整个数组也将按升序排序，你需要找到****最短的****这样的子数组并输出它的长度）

①简要实现思路：

首先需要对数组进行排序，排序结束以后，将排序后的数组与排序前的数组从两边分别开始向中间一一对应判断，当排序后数组与排序前数组出现不一样的地方时，就说明中间包含的就是最短无序连续子数组。

②测试及代码截图：



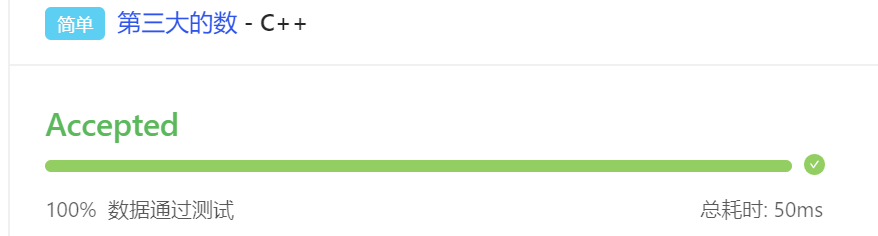
1. 第三大的数

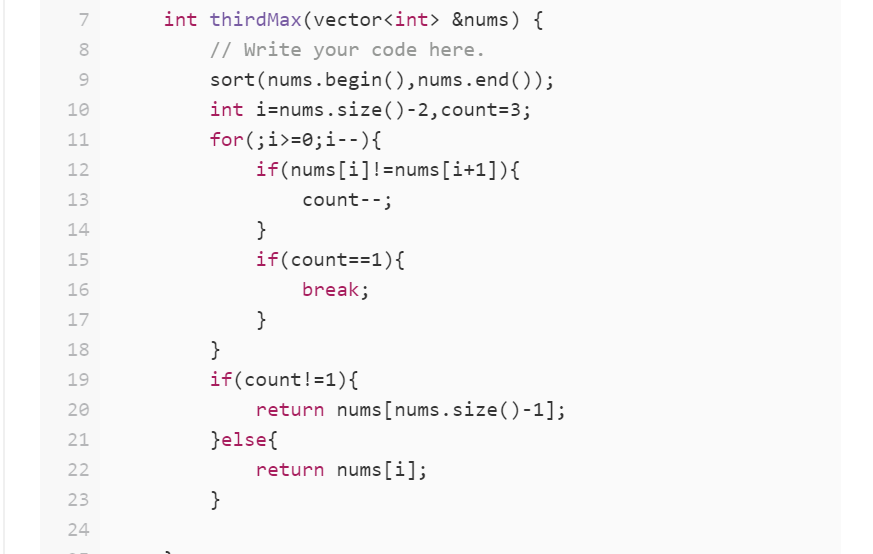
（给定一个非空数组，返回此数组中第三大的数。如果不存在，则返回数组中最大的数。要求算法时间复杂度必须是O(n)）

①简要实现思路：

首先对数组进行排序，此时采用自带的sort函数进行排序加快运行速度。然后对数组进行反向遍历，如果数字相同则跳过，找到第三大的数字输出即可。

②测试及代码截图：





1. 重复计数

（给定一个整数数组，您的任务是计算重复数组元素的数量。 重复是被定义为两个或更多相同的元素。 例如，在数组[1, 2, 2, 3, 3, 3]中，两个2是一个重复，三个3是一个重复。你需要返回一个数组，包含所有非唯一（重复）值的数字，且顺序与给定的数组相同。）

①简要实现思路：

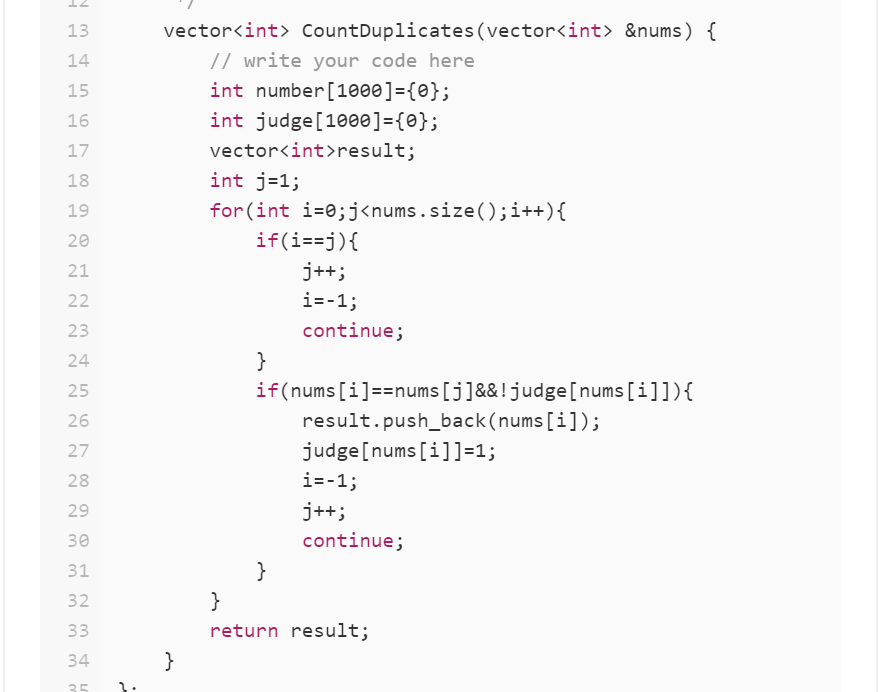
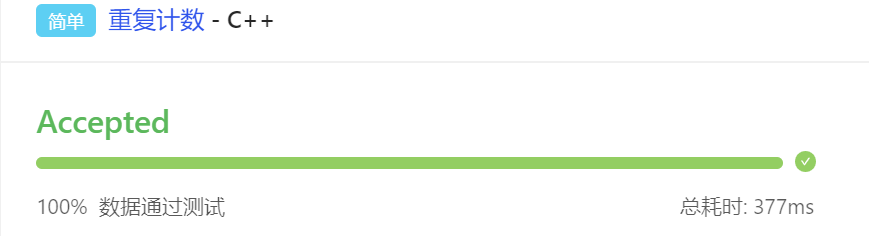
依旧采取双指针法。首先确定此题目的要求是要，返回重复数字并且返回顺序是“先重复者先返回”而不是“先出现者先返回”。

按照双指针的思想，慢指针从0开始，快指针从1开始。若是满足快指针数字等于慢指针数字（肯定是第一次出现的数字），且判断数字显示此数字并未在结果数组中出现过，则将此数字放入结果数组，并且修改判断数组的信息。

然后将快指针自加，慢指针置为0。

若是直到快慢指针重合仍未找到重复的数字，则说明这一区间没有重复数字，所以将快指针自加，慢指针置为0重新开始，当快指针到最后一个时结束。

②测试及代码截图：



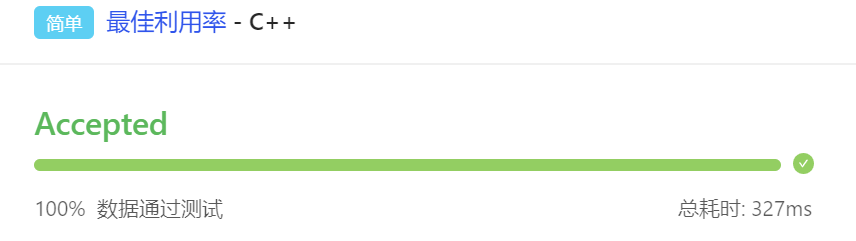
1. 最佳利用率

（给两个排序的数组。 从两个数组中各取取一个数，这两个数之和需要小于或等于k， 需要找到两数之和最大的索引组合。返回一对包含两个列表的索引。如果有多个两数之和相等的索引答案，你应该选择第一个数组索引最小的索引对。  
1）两数的总和<= k  
2）总和是最大的  
3）两个数组索引都尽量最小）

①简要实现思路：

定义一个index序列数组，两个数组依次从0开始遍历，如果满足两数之和小于等于k且大于之前的sum，则直接替换sum和index，由于都是从0开始向后遍历的，所以第一次的索引已经是最小的了。

②测试及代码截图：



**第三部分：栈：**

1. 最长有效括号

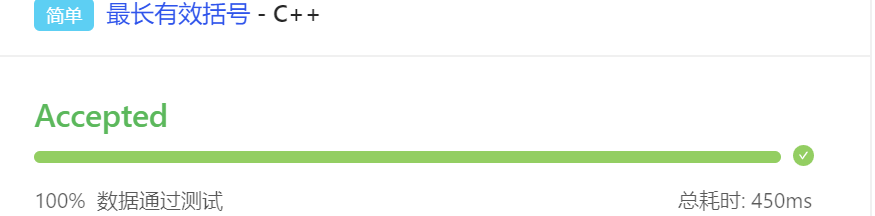
（给出一个只包含'('和')'的字符串，找出其中最长的左右括号正确匹配的合法子串）

①简要实现思路：

运用栈的方法，首先对原数组遍历一遍，依次将括号放入堆栈，如果正好放入的括号构成（）则可以消去，将元素出栈，否则无法消去，将现在遍历的括号序号放入栈。在遍历结束以后，栈中元素的可能情况有”((((...”“)))..”

“)(((..”“..)))(”，所以将栈中元素每两个之间的间隙大小就是其包含的有效括号大小，故依次计算距离保存最大值，然后将元素出栈，最后和栈底最后一个元素序号（其前面所有的均为有效括号）对比，取最大值即可。

②测试及代码截图：



1. 用栈模拟汉诺塔问题

（在经典的汉诺塔问题中，有 3 个塔和 N 个可用来堆砌成塔的不同大小的盘子。要求盘子必须按照从小到大的顺序从上往下堆 （如，任意一个盘子，其必须堆在比它大的盘子上面）。同时，你必须满足以下限制条件：

(1) 每次只能移动一个盘子。  
(2) 每个盘子从堆的顶部被移动后，只能置放于下一个堆中。  
(3) 每个盘子只能放在比它大的盘子上面。

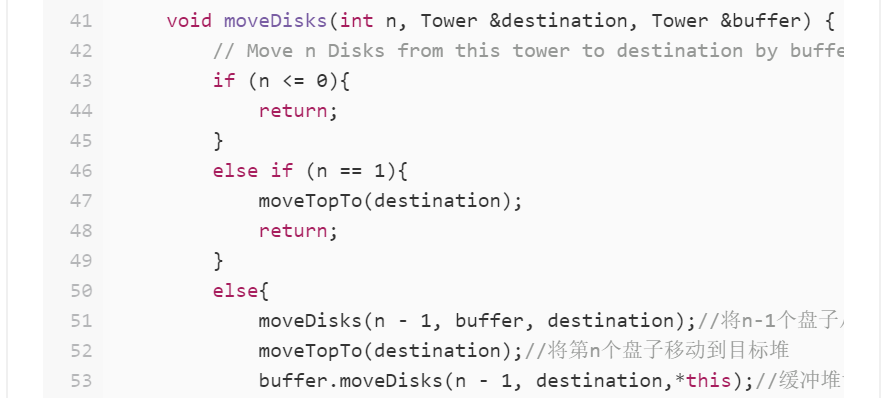
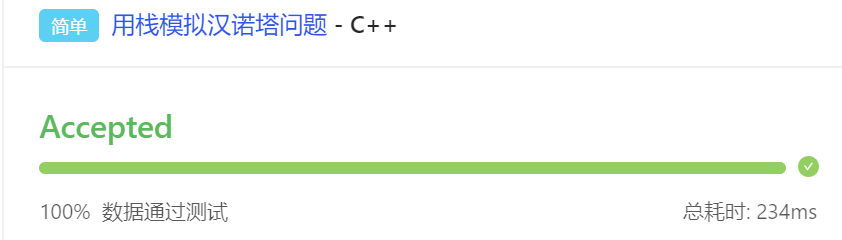
请写一段程序，实现将第一个堆的盘子移动到最后一个堆中

）

①简要实现思路：

此题需要我们编写两个函数“移动顶部盘子”“移动n个盘子”，移动顶部盘子好办，直接使用已有的add函数即可，然后将顶部元素出栈。移动n个盘子的话，如果n小于等于0则退出，如果n等于1则调用移动顶部盘子即可，如果n大于1，则先将n-1个盘子移动到缓冲柱子**（调用自己，递归）**，然后将底部盘子移动到目标柱子，最后将缓冲柱子上的盘子移动过来即可（需要对缓冲柱子对象调用moveDisks函数，同时由于缓冲堆的盘子数应该为0，所以此时的缓冲堆是上一次执行时的那个空的堆栈，所以用this）。

②测试及代码截图：



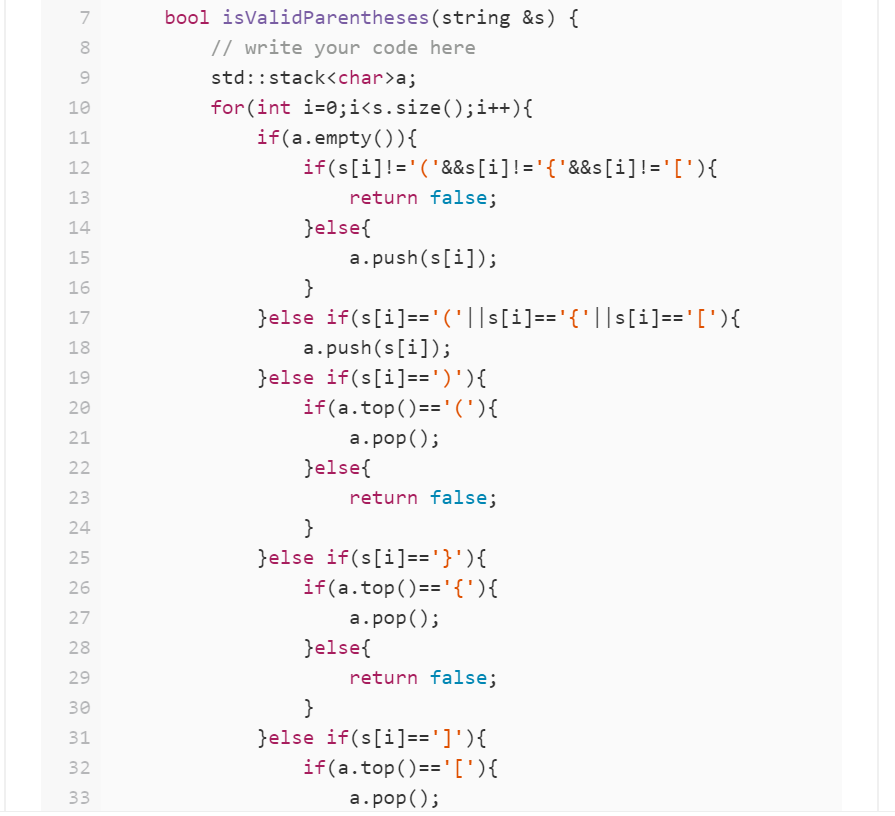
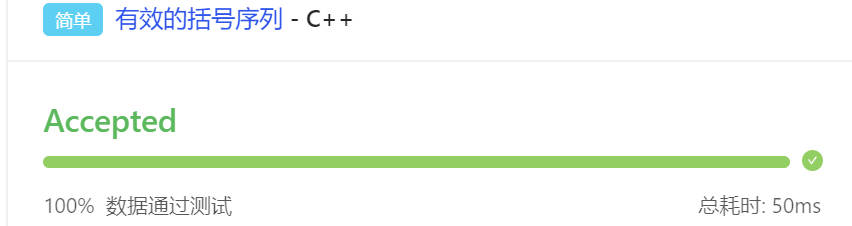
1. 有效的括号序列

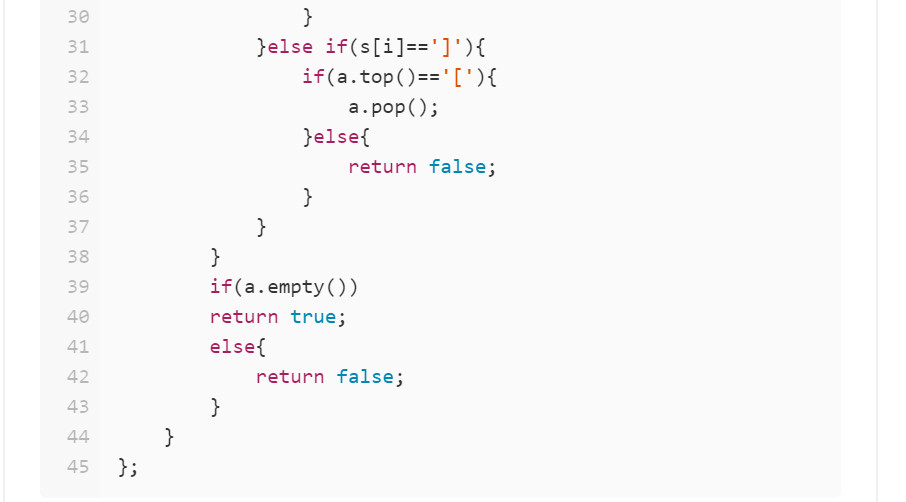
（给定一个字符串所表示的括号序列，包含以下字符： '(', ')', '{', '}', '[' and ']'， 判定是否是有效的括号序列，括号必须依照 "()" 顺序表示， "()[]{}" 是有效的括号，但 "([)]" 则是无效的括号）

①简要实现思路：

由于题目要求必须要'(', ')', '{', '}', '[' and ']'在一块，所以直接遍历一遍即可，每一次将元素与栈顶元素进行比较，如果满足"()[]{}"的配对情况之一则出栈，否则入栈，如果在遍历结束以后，栈中没有元素且中间不被打断，则返回true，否则返回false。

②测试及代码截图：





1. 实现栈

（此题在数组中出现过，此处不再赘述）

1. 移掉k位数字

（给定一个以字符串表示的非负整数 num，移除这个数中的 k 位数字，使得剩下的数字最小）

①简要实现思路：

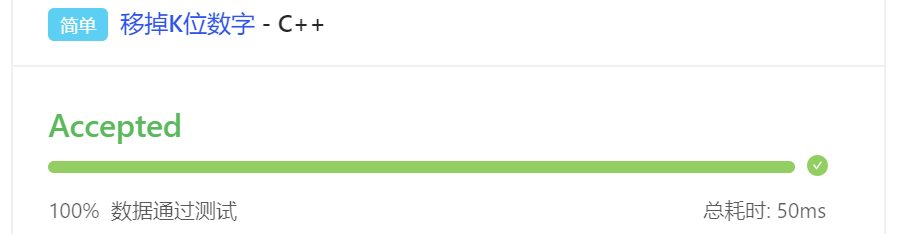
此题参考了网站：（单调栈+贪心算法）

<https://blog.csdn.net/weixin_42956047/article/details/106364279>

使用贪心算法构建一个从栈底到栈顶的递增栈，也就是要遍历全部数组，并且尽量保证栈顶为最大数字，从而使得数字最后最小，当栈顶元素大于当前元素时，若是k不为0则出栈，直到不满足为止。如果恰好一直递增那也需要去除k个元素，则直接从栈顶开始去除k个元素。

完成上述步骤后还有一个非常关键的步骤：**将数字转换成字符并且从栈底到栈顶逆序输出字符串**，我百度得知”to\_string（）函数”可以将将数字变成字符串，但是网上的逆序函数都用不了，后来我发现可以**利用string类中的“strcat函数”或者“+”运算符，将栈顶元素放在前面直接用+即可实现逆序输出**。

②测试及代码截图：





1. 棒球游戏

（现在你是棒球比赛分记录员。

给定一个字符串数组，每一个字符串可以是以下4种中的其中一个：

（1）整数 (一个回合的分数): 直接表示这回合你得到的分数。

（2）"+" (一个回合的分数): 表示这回合你获得的分数为前两个 有效 分数之和。

（3）"D" (一个回合的分数): 表示这回合你得到的分数为你上一次获得的有效分数的两倍。

（4）"C" (一种操作，而非一个回合的分数): 表示你上回合的有效分数是无效的，需要移除。  
每一轮的操作都是永久性的，可能会影响之前和之后的一轮。

你需要返回在所有回合中获得总分数。

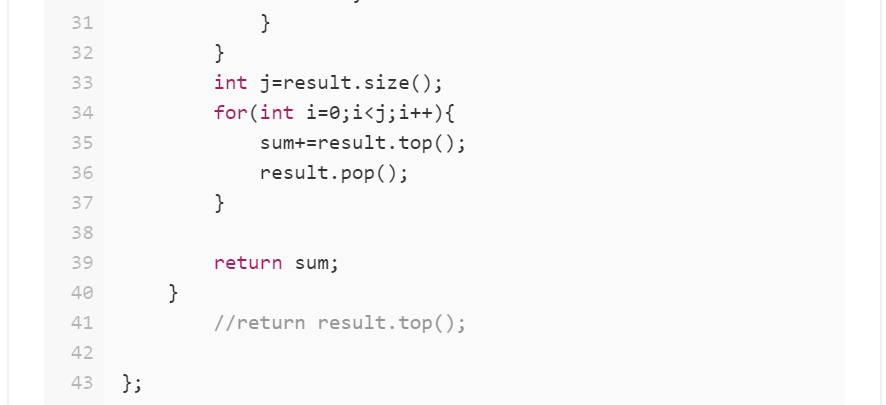
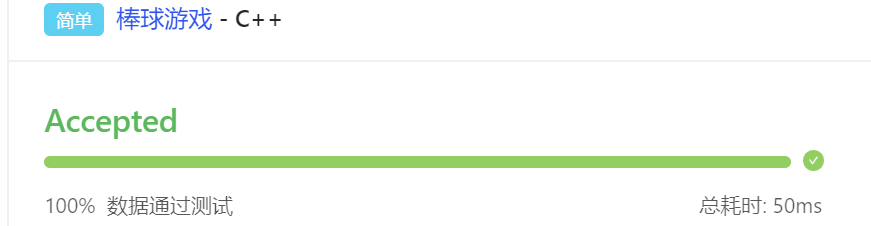
）

①简要实现思路：

首先这里有一个非常重要的函数**“std::stoi（）”**可以将**字符串转换为数字**，网上提供的其他的函数都不行。

此题首先用栈来记录每一轮的分数，再根据对应的字符串对栈中元素进行计算操作即可，最后直接遍历整个栈求和得到总分。

②测试及代码截图：



1. 下一个更大的数

（你有两个数组 nums1和nums2****（互不重复）****，其中nums1是nums2的子集。 在nums2的相应位置找到nums1所有元素的下一个更大数字，nums1中的数字x的下一个更大数字是nums2中x右边第一个更大的数字。 如果它不存在，则为此数字输出-1）

①简要实现思路：

此题首先建立一个栈来维护答案，通过两次遍历，在nums2中找到数字x后向后寻找下一个更大的数，找到则直接push进栈，否则将-1 push进栈，每次循环后都将栈顶元素赋值给答案数组，在遍历结束以后即可。（其实完全不用栈也可以很简单解决）。

②测试及代码截图：

