contents

[Basic Concept 4](#_Toc424253688)

[p1.#206Problem: Reverse Linked List 5](#_Toc424253689)

[Description: 5](#_Toc424253690)

[Test: 5](#_Toc424253691)

[Method1: store every data in the list into program 5](#_Toc424253692)

[Solution1: 6](#_Toc424253693)

[Method2: change each node’s pointer 6](#_Toc424253694)

[Solution2: 7](#_Toc424253695)

[Method3: create a new singly linked list 7](#_Toc424253696)

[Solution3: 7](#_Toc424253697)

[P2.#19 Problem: Remove Nth Node from End of List 8](#_Toc424253698)

[Description: 8](#_Toc424253699)

[Test: 8](#_Toc424253700)

[Method1: use two nodes to go through the linked list twice 8](#_Toc424253701)

[Solution1: 9](#_Toc424253702)

[Discuss: 9](#_Toc424253703)

[P3 .# 203 Problem: Remove Linked List Elements 10](#_Toc424253704)

[Description: 10](#_Toc424253705)

[Test: 10](#_Toc424253706)

[Method 1: 11](#_Toc424253707)

[Solution 1: 11](#_Toc424253708)

[Method 2: 11](#_Toc424253709)

[Solution 2: 11](#_Toc424253710)

[P4 .# 83 Problem: Remove Duplicates from Sorted List 12](#_Toc424253711)

[Description: 12](#_Toc424253712)

[Test: 12](#_Toc424253713)

[Method 1: 12](#_Toc424253714)

[Solution 1: 13](#_Toc424253715)

[Method 2: 13](#_Toc424253716)

[Solution 2: 13](#_Toc424253717)

[P5 .# 21 Problem: Merge Two Sorted Lists 14](#_Toc424253718)

[Description: 14](#_Toc424253719)

[Discuss: 14](#_Toc424253720)

[Test: 14](#_Toc424253721)

[Method 1: 15](#_Toc424253722)

[Solution 1: 15](#_Toc424253723)

[P6 #160 Problem: Intersection of Two Linked Lists 16](#_Toc424253724)

[Description: 16](#_Toc424253725)

[Test: 17](#_Toc424253726)

[Method 1: 17](#_Toc424253727)

[Solution 1: 18](#_Toc424253728)

[Method 2: 18](#_Toc424253729)

[Solution 2: 19](#_Toc424253730)

[P7 .# 2 Problem: Add Two Numbers 20](#_Toc424253731)

[Description: 20](#_Toc424253732)

[Test: 20](#_Toc424253733)

[Method 1: 20](#_Toc424253734)

[Solution 1: 21](#_Toc424253735)

[Discuss 21](#_Toc424253736)

[P8 .# 24 Problem: Swap Nodes in Pairs 23](#_Toc424253737)

[Description: 23](#_Toc424253738)

[Test: 23](#_Toc424253739)

[Method 1: 23](#_Toc424253740)

[Solution 1: 24](#_Toc424253741)

[P9 .# 61 Problem: Rotate List 25](#_Toc424253742)

[Description: 25](#_Toc424253743)

[Given a list, rotate the list to the right by *k* places, where *k* is non-negative. 25](#_Toc424253744)

[Test: 25](#_Toc424253745)

[Method 1: 25](#_Toc424253746)

[Solution 1: 26](#_Toc424253747)

[P10. #148 Problem: Sort List 27](#_Toc424253748)

[Description: 27](#_Toc424253749)

[Test: 27](#_Toc424253750)

[Method 1: 27](#_Toc424253751)

[Solution 1: 28](#_Toc424253752)

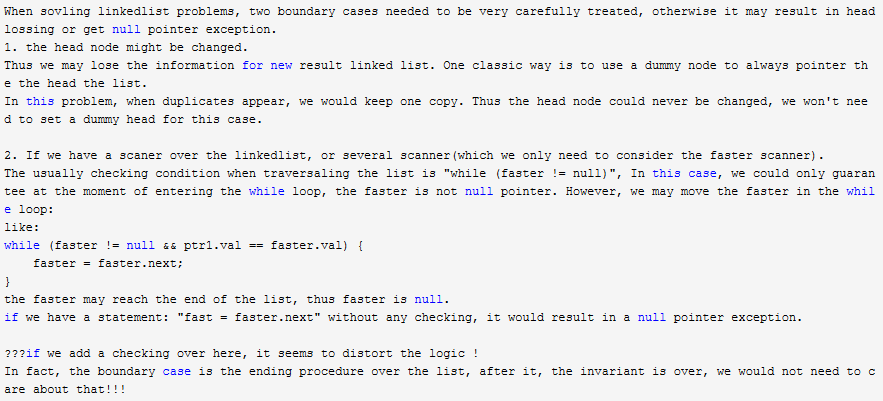
**本周是关于 链表的知识**

# Basic Concept

A linked list is a series of nodes in memory such that:

* There is a starting node.
* Each node contains a pointer that points to the next or child node.
* If a node does not have a child node then its pointer is set to NULL.
* Each node contains data, maybe a lot of it.
* The linked list also has functions that manage the list by performing additions, deletions, changing the data of a node, returning the number of nodes, etc., etc.

Basically, there are two types of linked list, singly-linked list and doubly-linked list. In a singly-linked list every element contains some data and a link to the next element, which allows to keep the structure. On the other hand, every node in a doubly-linked list also contains a link to the previous node. Linked list can be an underlying data structure to implement stack, queue or sorted list.

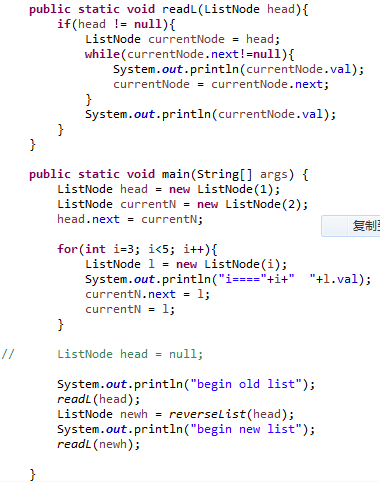


# p1.#206Problem: Reverse Linked List

## Description:

Reverse a singly linked list.

## Test:



## Method1: store every data in the list into program

要返着输出一个单链表。 单链表顾名思义从后一个node 没有办法找到前一个node 最先想到的办法就是遍历两遍此链表。 第一遍将链表中的数据读取出来并存入list，第二遍挨个赋值（从list 的尾向头赋值）。

第一次写完代码后遇到的问题： nonpointerexception 看异常抛出的line 发现是第一次用输入的head 的时候就抛异常了 所以在一进函数的最开头判断 if(head==null) return head;

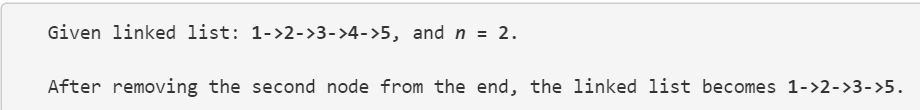
|  |
| --- |
| Solution1: |
|  |
| Method2: change each node’s pointer 进一步，并没有必要在本地把这一堆数据都原样存储下来 太浪费内存了。单链表有方向是没错，我们可以换一个角度利用有方向这一点“改变链表的方向” 即让当前的后一个节点指向前一个节点。 方法一是最直接的把值从新赋予对了就像压入栈中的小球（本来为ABCD），将小球一个一个push出去（成为DCBA）再按这个顺序从前往后放回去成为DCBA。方法二则像装小球的盒子连接方向改变了一下更加快捷高效 |
|  |
| Solution2: |
|  |
| Method3: create a new singly linked list | |
| 此方法为第一，二种方法思路上的合并。 即将所有数据存储了一遍又翻转了链表方向。个人认为是最次的一种方法。 但是涉及到链表节点的添加知识点就权当练习一下了。 | |
| Solution3: | |
|  | |

# P2.#19 Problem: Remove Nth Node from End of List

## Description:

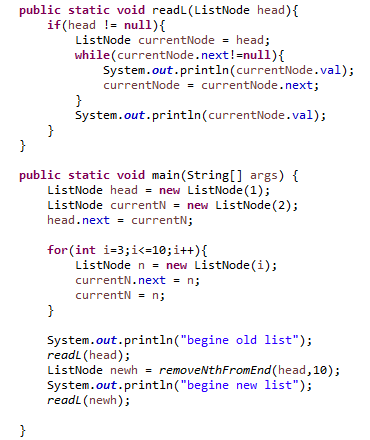
Given a linked list, remove the *n*th node from the end of list and return its head.

For example,



**Note:**  
Given *n* will always be valid.  
Try to do this in one pass.

## Test:



## Method1: use two nodes to go through the linked list twice

Although the question said “try to do this in one pass”, the first method in my mind is still using two nodes to go through the linked list separately (twice in total). Node1 and Node2 has n+1 nodes distance and once node1 reached the end, we simply discard the node next to node2 (i.e. node2.next = node2.next.next). Note: be careful with boundary input value

|  |
| --- |
| Solution1: |
|  |
| Discuss: 我想出第一种方法的时候一直以为我的方法属于 “go though the list twice” 的方法。 因为确实是两个节点分别一个一个的往后移动。在我看来这样和 典型的 “go though the list twice” 方法 （先走一遍知道总长度l然后再从头走到l-n的位置删除）没有任何本质区别（毕竟两个指针合起来遍历过的节点数完全一样为 2\*l-n） 唯一的不同是我的方法把两次遍历混在一起了前面的指针走过一个节点后面的指针也往后走一个节点，传统的方法将两次遍历分开了。但是我在网上搜索了一气并没有找到真正意义上 “go through the list once“ 的方法。难道是我理解错题目提示的意思了？。。 |

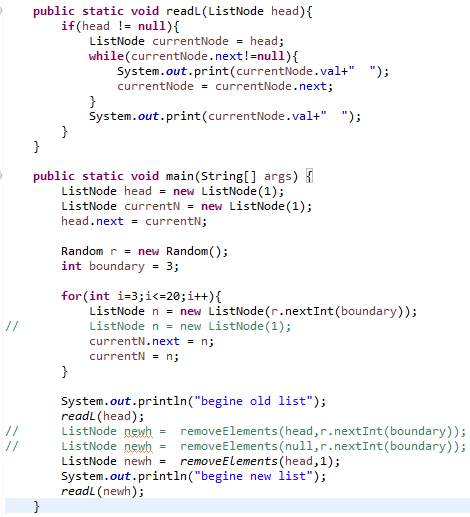
# P3 .# 203 Problem: Remove Linked List Elements

## Description:

Remove all elements from a linked list of integers that have value ***val***.

**Example**  
***Given:*** 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, ***val*** = 6  
***Return:*** 1 --> 2 --> 3 --> 4 --> 5

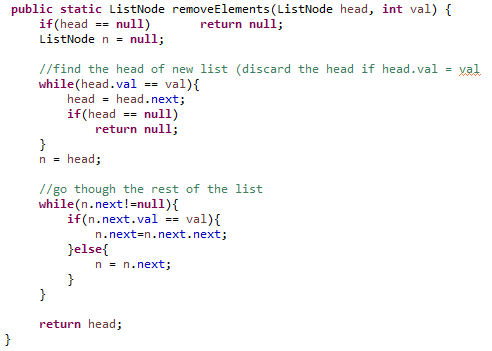
## Test:



## Method 1:

一看完题发现如此直截了当。 还真是送分题呢。。 没啥特别的便利一遍删去val = val的节点。注意特殊情况(1.输入为null 2.开头为需要删除的节点 3.开头连续好几个节点都需要删除 4.结尾的处理)

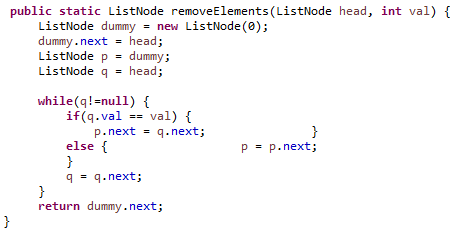
## Solution 1:



## Method 2:

用两个指针记录相邻的两个node 该删删该留留。 个人认为此方法没有第一种方法好 能一个指针过一遍解决的事情为啥要两个指针过两遍解决。。。 不过对头的处理方式值得学习. 用一个傻瓜节点(dummy node)接在头结点的前面可以保证头结点不丢失

## Solution 2:



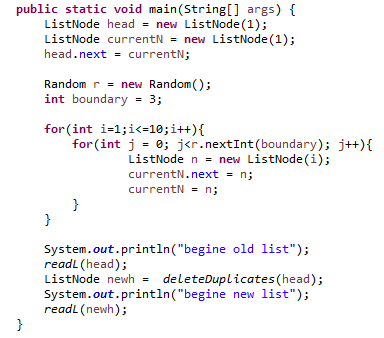
# P4 .# 83 Problem: Remove Duplicates from Sorted List

## Description:

Given a sorted linked list, delete all duplicates such that each element appear only *once*.

For example,  
Given 1->1->2, return 1->2.  
Given 1->1->2->3->3, return 1->2->3.

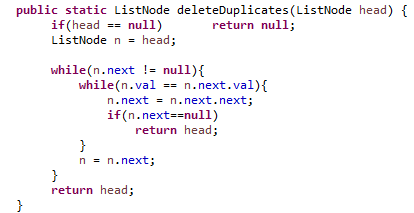
## Test:



## Method 1:

此题跟上一题如出一辙反正是已经排序了的list相当于val会随着node的变化而变化。改编自p3 method1.

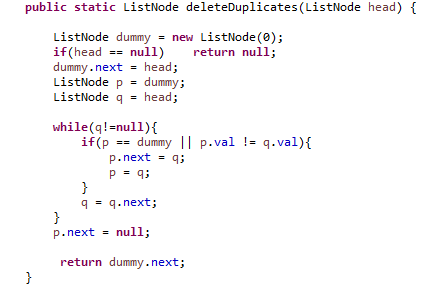
## Solution 1:



## Method 2:

此题跟上一题如出一辙反正是已经排序了的list相当于val会随着node的变化而变化。改编自p3 method 2. 用两个指针 + 笨蛋节点

## Solution 2:



# P5 .# 21 Problem: Merge Two Sorted Lists

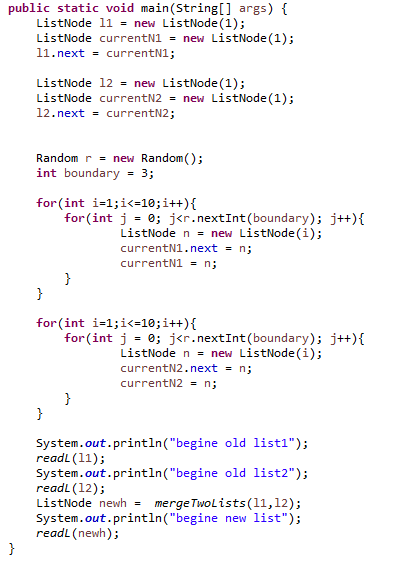
## Description:

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

## Discuss:

题目说 two sorted lists 并没有说 two sorted lists in the same order 初步设想有三种情况 A.都为从大到小 B.都为从小到大 C.一个从大到小 一个从小到大 . 我一开始误以为得分ABC三个case 分别计算。但是我只写了一种从小打到排序的情况就过了。。。。 。 是我理解错问题了 还是题目简化成了从小到大的情况

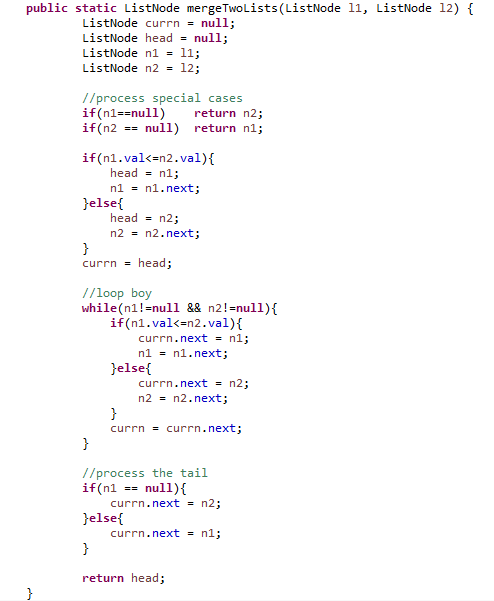
## Test:



## Method 1:

题目将要求说的很明确了返回的新链表需为给出的两个List的拼接 只能两个指针n1,n2分别遍历l1,l2. 一个currentNode 指针存合并链表的节点 一个head 存头。 每次n1.val 与n2.val 比大小 小的那个赋值给 currentNode后往前挪一位。 注意处理好开始的特殊情况与结尾部分。

## Solution 1:

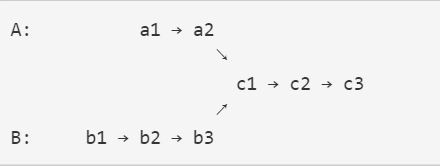


# P6 #160 Problem: Intersection of Two Linked Lists

## Description:

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:

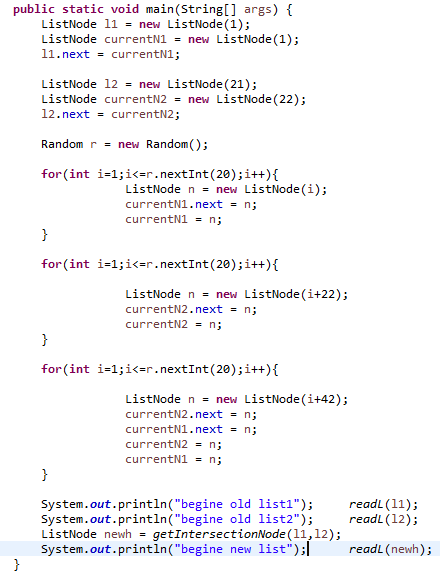


Begin to intersect at node c1.

**Notes:**

* If the two linked lists have no intersection at all, return null.
* The linked lists must retain their original structure after the function returns.
* You may assume there are no cycles anywhere in the entire linked structure.
* Your code should preferably run in O(n) time and use only O(1) memory.

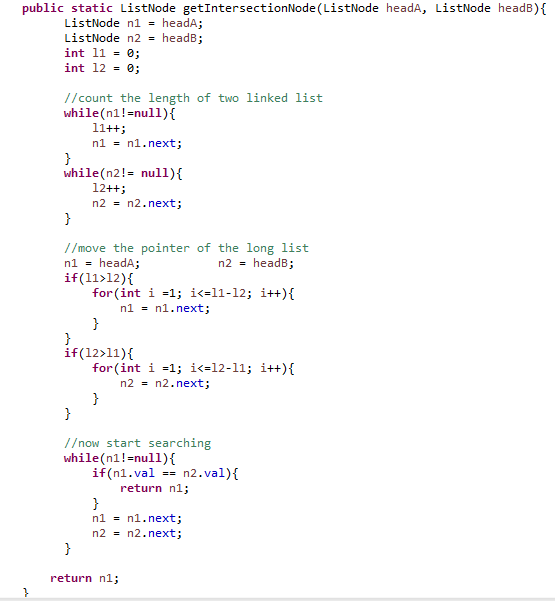
## Test:



## Method 1:

介于题目给的存储空间的条件，将所有值都存下来排序数重复并不是一个很好的选择，弃之。 时间复杂度为O(n) ，并没有严格像空间复杂度一样限制为O(1) 说明要遍历链表有限的次数。 设公共长度为a, A链表长度为 a+b B链表长度为a+c 设c>b 则如若两个链表有合并，合并位置最前为从A链表的头，B链表的c-b位置开始。 所以解法一为先分别遍历两个量表得长度l1,l2 将长的链表指针n1从头开始向后移动abs(l1-l2)个node后开始与短链表的指针（此时在头部）n2 一一对比值并记录移动的个数m 当n1与n2所指内容一样时终止循环

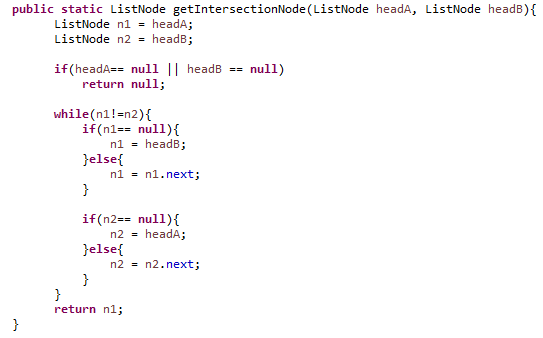
## Solution 1:



## Method 2:

神算法。。。 将从A的头开始的指针遍历完 list A 以后遍历 list B 将从 B 开始的指针便利完list B 以后遍历 list A。 这样弥补了A与B不一样长的问题 只要他们有 intersection 两个指针总会指向相同的地方

## Solution 2:



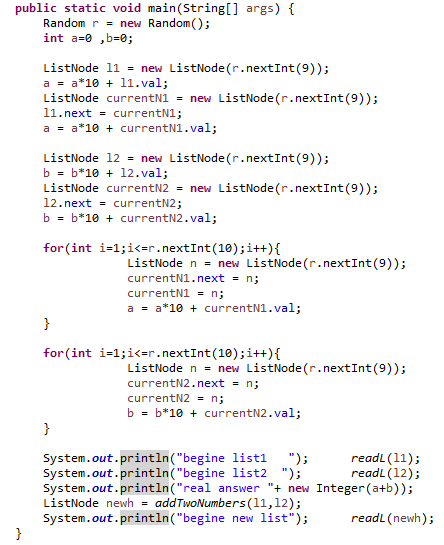
# P7 .# 2 Problem: Add Two Numbers

## Description:

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

**Input:** (2 -> 4 -> 3) + (5 -> 6 -> 4)  
**Output:** 7 -> 0 -> 8

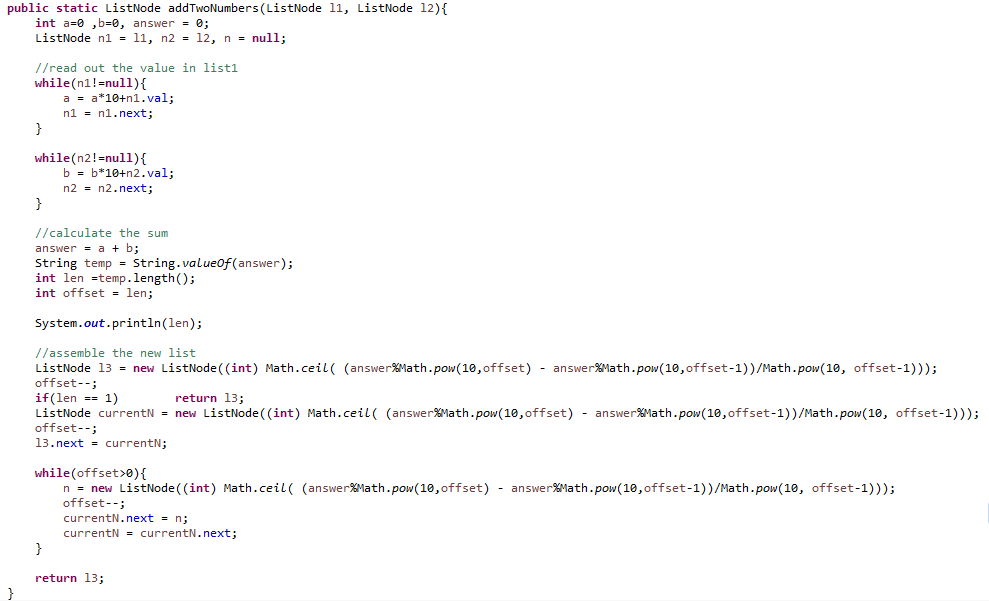
## Test:



## Method 1:

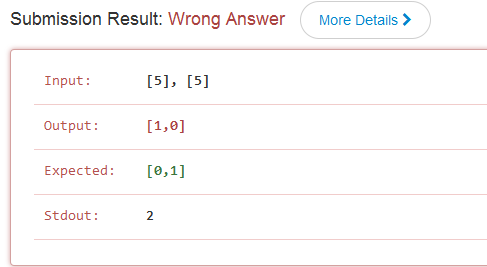
数的加法处理的难点在进位 （正如数的减法处理的难点在退位）. 但链表从前往后为位数由高至低所以增加了难度。方法一为投机取巧的方法。。。 先把数取出来，让java 算加法再把算好的答案放入一个新的链表返回去。 即将求和与链表赋值分开进行 规避了问题的难点

## Solution 1:

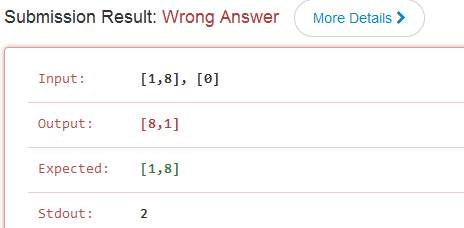


## Discuss

我写完这个答案以后上去测就撞见鬼了。。。 5+5 难道不是应该结果为10吗。。。为啥expected 结果为 01 我读书少你不要骗我。。。。



然后我再想了一下觉得难道是输入为从高位至地位但是输出要求从低位至高位？ 好吧 怪我喽。。。 于是我开心地改了我的代码可以把答案从低位到高位输出了又提交了一次结果：



你是在逗我吗！！！ 为啥这次18+0 = 18 我给你反着从个位输出 81了你说我应该输出 18。 我真的无语了 这题是几个意思！！！

我别的能操作进位的方法也不想了。。。 待我把题是啥意思搞明白再说吧。。。 （未完待续）

# P8 .# 24 Problem: Swap Nodes in Pairs

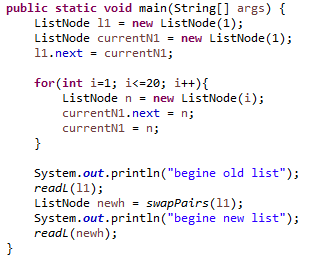
## Description:

Given a linked list, swap every two adjacent nodes and return its head.

For example,  
Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may **not** modify the values in the list, only nodes itself can be changed.

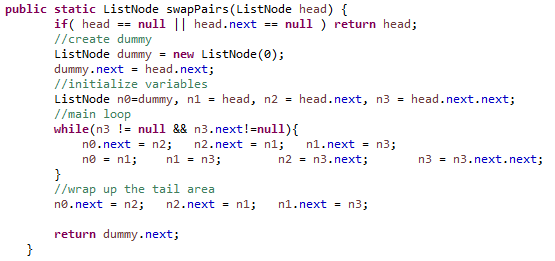
## Test:



## Method 1:

先排除两个特殊情况：1.传入的值为null 2.整个list 只有一个节点 也无所谓交换 剩下情况可以保证有两个节点。 吸取之前的经验， 头会变用笨蛋（dummy）。 因为需要两两交换所以本地至少得有四个指针。 两个指向要交换的节点 一个指向被交换的两个节点的前一个节点，一个指向链表中的后一个节点，这样就可以保证在交换的过程中不会造链表的丢失。 此题不难，就是有点绕 搞清楚关系了就可以很容易的做出来了。

## Solution 1:



# P9 .# 61 Problem: Rotate List

## Description:

### Given a list, rotate the list to the right by *k* places, where *k* is non-negative.

For example:  
Given 1->2->3->4->5->NULL and *k* = 2,  
Return 4->5->1->2->3->NULL.

## Test:

测试方法跟上一题的并无大差 就不贴了

## Method 1:

虽然名字叫的这么高大上 什么旋转跳跃我闭着眼， 但其实借鉴那个p6#160题的神算法只需如下几步：

1. 遍历链表一遍，记录链表总长度n

2. 将此时的tail.next = head 即将链表头尾相连形成环

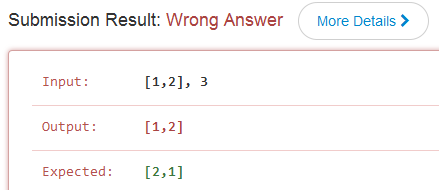
3. 继续向后走n-k个node

4. 要返回的头尾此节点的next， 用小笨蛋 dummy 记录

5. 将此节点的next = null 打破环

6. 返回 dummy

于是我天真地实现了以下去尝试发现还有这个鬼： k 是可以大于n的！



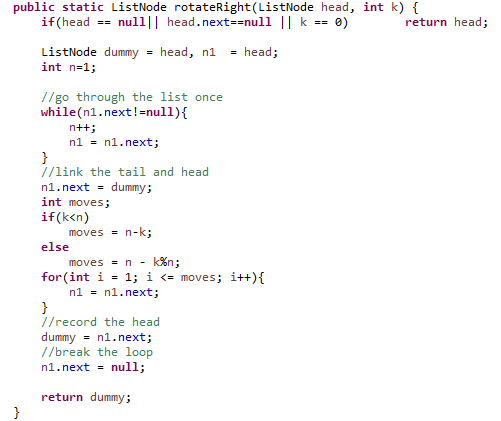
好吧。。。 怪我喽

第三步应该改成

If(k<n) 继续向后走n-k个node

Else继续向后走n - k%n个node

## Solution 1:

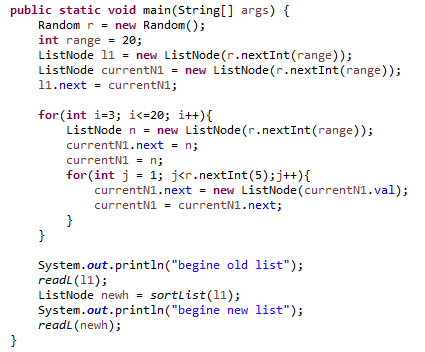


# P10. #148 Problem: Sort List

## Description:

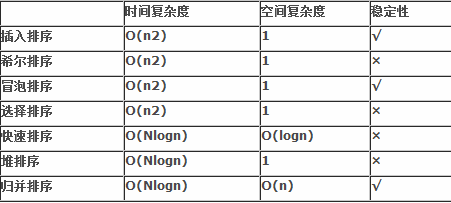
Sort a linked list in *O*(*n* log *n*) time using constant space complexity.

## Test:

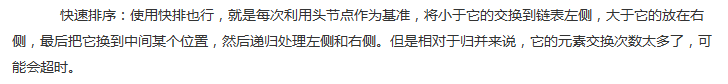
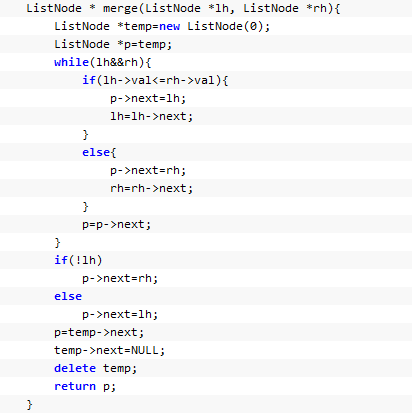
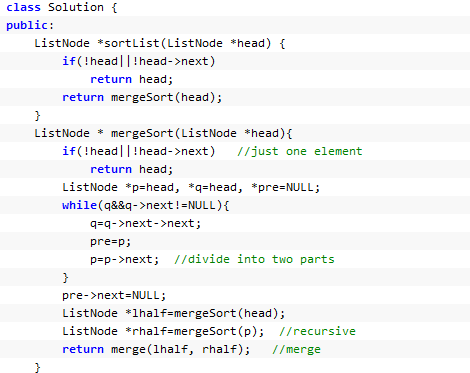
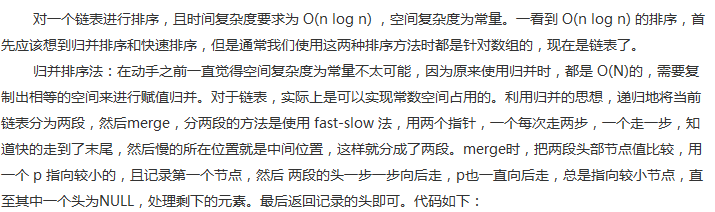


## Method 1:

这就是一条披着链表皮的排序题 只是因为单链表取值的限制多了好多限制。。。根据题目的要求 时间复杂度为O(nlog(n)) 空间复杂度为1。 上网百度了一下各种排序算法的复杂度表： 这么一看还真是没活头。。 那好吧。。。 我基础薄弱不会用链表实现后三种排序啊。。。 于是又百度出了下面的solution1.。。



## Solution 1:



原文链接：

<http://blog.csdn.net/jiadebin890724/article/details/21334059>

请叫我ddl oriented 。。。 待我这个周末把排序方法系统地学习总结一下。。。。