```
##############################  445 Add Two Numbers II ##############################
You are given two non-empty linked lists representing two non-negative integers. The most
significant digit comes first and each of their nodes contain a single digit. Add the two
numbers and return it as a linked list.
You may assume the two numbers do not contain any leading zero, except the number 0 itself.
Follow up:
What if you cannot modify the input lists? In other words, reversing the lists is not allowed.

Example:
Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 8 -> 0 -> 7
## 把两个链表的值依次存入stack, 然后通过add_to_head构建新链表并返回
class Solution(object):
    def addTwoNumbers(self, l1, l2):
        tmp1, tmp2 = l1, l2
        list1, list2 = [], []
        while tmp1:
            list1.append(tmp1.val)
            tmp1 = tmp1.next
        while tmp2:
            list2.append(tmp2.val)
            tmp2 = tmp2.next
        carry = 0
        head = None
        while len(list1) or len(list2) or carry:
            num1 = list1[-1] if len(list1) else 0
            num2 = list2[-1] if len(list2) else 0
            if len(list1):
                list1.pop()
            if len(list2):
                list2.pop()
            newNode = ListNode((num1+num2+carry)%10)
            carry = (num1+num2+carry)/10
            if not head:
                head = newNode
            else:
                newNode.next = head
                head = newNode
        return head

class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        stack<int> sk1, sk2;
        ListNode *tmp(l1);
        while (tmp){
            sk1.push(tmp->val);
            tmp = tmp->next;
        }
        tmp = l2;
        while(tmp){
            sk2.push(tmp->val);
```

```
            tmp = tmp->next;
        }
        int num1, num2, carry(0);
        ListNode *head(NULL);
        while(!sk1.empty() || !sk2.empty() || carry){
            num1 = sk1.empty()?0:sk1.top();
            num2 = sk2.empty()?0:sk2.top();
            ListNode* newNode = new ListNode((num1+num2+carry)%10);
            carry = (num1+num2+carry)/10;
            if(!sk1.empty()) sk1.pop();
            if(!sk2.empty()) sk2.pop();
            if(!head)
                head = newNode;
            else{
                newNode->next = head;
                head = newNode;
            }
        }
        return head;
    }
};
############################### 328 Odd Even Linked List ########################
Given a singly linked list, group all odd nodes together followed by the even nodes. Please
note here we are talking about the node number and not the value in the nodes.
You should try to do it in place. The program should run in O(1) space complexity and O(nodes)
time complexity.

Example:
Given 1->2->3->4->5->NULL,
return 1->3->5->2->4->NULL.
易错：提前把head.next保存起来，不能odd.next = head.next

class Solution(object):
    def oddEvenList(self, head):
        if not head or not head.next:
            return head
        odd, even = head, head.next
        head_next = head.next
        while odd.next and even.next:
            odd.next = even.next
            odd = odd.next
            even.next = odd.next
            even = even.next
        odd.next = head_next
        return head

class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if(!head || !head->next)
            return head;
        ListNode *odd(head), *even(head->next), *even_head(head->next);
        while(odd->next && even->next){
```

```
            odd->next = even->next;
            odd = odd->next;
            even->next = odd->next;
            even = even->next;
        }
        odd->next = even_head;
        return head;
    }
};
############################### 237 Delete Node in a Linked List ##################

Write a function to delete a node (except the tail) in a singly linked list, given only access
to that node.
Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the
linked list should become 1 -> 2 -> 4

class Solution(object):
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-place instead.
        """
        node.val = node.next.val
        node.next = node.next.next

class Solution {
public:
    void deleteNode(ListNode* node) {
        auto next = node->next; //node是一个指针，*node是一个ListNode对象,
        下面这句话等同于node->val = node->next->val; node->next = node->next->next;
        *node = *node->next;
        delete
        next;//删掉指向原来node->next的那个指针，因为这个指针指向的那个node对象已经被复制到node,
        所以现在这个node没有用了,指针也不需要了
    }
};
############################### 234    Palindrome Linked List #################
Given a singly linked list, determine if it is a palindrome.

class Solution {
public:
    bool isPalindrome(ListNode* head) {
        if (!head)
            return true;
        // find the middle node
        ListNode *slow = head;
        ListNode *fast = head;
        while(fast && fast->next)
        {
            fast = fast->next->next;
            slow = slow->next;
        }
        //modify the list:
```

```
        //1->2->3->4 ==> 1->2->3<-4
        //1->2->3->4->5 ==> 1->2->3<-4<-5
        ListNode *pre = slow;
        ListNode *post = slow->next;
        slow->next = nullptr;
        ListNode *temp;
        while(post)
        {
            temp = post->next;
            post->next = pre;
            pre = post;
            post = temp;
        }
        //examine if it satisfy the Palindrome requirements
        ListNode *tail = pre;
        while(tail != head && tail && head)
        {
            if (tail->val != head->val)
                return false;
            tail = tail->next;
            head = head->next;
        }
        return true;
    }
};

################################### 206 Reverse Linked List ###################
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode *pre = nullptr;
        ListNode *post = head;
        ListNode *temp;

        while(post){
            temp = post->next;
            post->next = pre;
            pre = post;
            post = temp;
        }
        return pre;
    }
};
############################### 203 Remove Linked List Elements  ##############
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
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {
        ListNode *dummy = new ListNode(0);
        dummy->next = head;
```

```
        ListNode *tmp = dummy;
        while (tmp && tmp->next){
            if (tmp->next->val == val)
                tmp->next = tmp->next->next;
            else
                tmp = tmp->next;
        }
        return dummy->next;
    }
};
class Solution(object):
    def removeElements(self, head, val):
        if not head:
            return head
        dummy = ListNode(0)
        dummy.next = head
        tmp = dummy
        while tmp and tmp.next:
            if tmp.next.val == val:
                tmp.next = tmp.next.next
            else:
                tmp = tmp.next
        return dummy.next
```

##########################160 Intersection of Two Linked Lists ######################

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

For example, the following two linked lists:

```
A:          a1 → a2
                   ↘
                     c1 → c2 → c3
                   ↗
B:     b1 → b2 → b3
```

begin to intersect at node c1.

```
class Solution(object):
    def getIntersectionNode(self, headA, headB):
        if not headA or not headB:
            return None
        lena, lenb = 0, 0
        tmp = headA
        while tmp:
            tmp = tmp.next
            lena += 1
        tmp = headB
        while tmp:
            tmp = tmp.next
            lenb += 1
        longer = headA if lena >= lenb else headB
        shorter = headA if lena < lenb else headB
        diff = abs(lena - lenb)
        while diff > 0:
            longer = longer.next
            diff -= 1
```

```
        while longer:
            if longer == shorter:
                return longer
            else:
                longer = longer.next
                shorter = shorter.next
        return None

class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        int lena(0), lenb(0);
        ListNode *tmp(headA);
        while (tmp){
            tmp = tmp->next;
            lena++;
        }
        tmp = headB;
        while (tmp){
            tmp = tmp->next;
            lenb++;
        }
        ListNode *longer = lena >= lenb? headA:headB;
        ListNode *shorter = lena < lenb? headA:headB;
        int diff = abs(lena - lenb);
        while(diff){
            longer = longer->next;
            diff--;
        }
        while (longer){
            if (longer == shorter){
                return longer;
            }
            longer = longer->next;
            shorter = shorter->next;
        }
        return NULL;
    }
};
############################## 148   Sort List  ##############################
Sort a linked list in O(n log n) time using constant space complexity.

class Solution {
public:
    ListNode* sortList(ListNode* head) {
        //recursive end condition
        if (!head || !head->next)
            return head;

        //find the left and right LinkList
        ListNode *fast(head);
        ListNode *slow = new ListNode(0);
        slow->next = head;
```

```
        while(fast && fast->next){
            fast = fast->next->next;
            slow = slow->next;
        }
        ListNode *tmp = slow->next;
        slow->next = NULL;

        //sort left and right recursively
        ListNode *left = sortList(head);
        ListNode *right = sortList(tmp);

        //merge left and right LinkList
        ListNode *dummy_k = new ListNode(0);
        ListNode *i(left), *j(right), *k(dummy_k);
        while (i && j){
            if(i->val < j->val){
                k->next = new ListNode(i->val);
                i = i->next;
            }
            else{
                k->next = new ListNode(j->val);
                j = j->next;
            }
            k = k->next;
        }
        while(i){
            k->next = new ListNode(i->val);
            i = i->next; k = k->next;
        }
        while(j){
            k->next = new ListNode(j->val);
            j = j->next; k = k->next;
        }
        return dummy_k->next;
    }
};
class Solution(object):
    def sortList(self, head):
        if head == None or head.next == None:
            return head
        slow = ListNode(0)
        slow.next = head
        fast = head
        while fast!=None and fast.next != None:
            fast = fast.next.next
            slow = slow.next
        tmp = slow.next
        slow.next = None
        left = self.sortList(head)
        right = self.sortList(tmp)
        i, j = left, right
        dummy_k = ListNode(0)
        k = dummy_k
```

```
        while i and j:
            if i.val < j.val:
                k.next = ListNode(i.val)
                i = i.next
            else:
                k.next = ListNode(j.val)
                j = j.next
            k = k.next
        while i:
            k.next = ListNode(i.val)
            i = i.next; k = k.next
        while j:
            k.next = ListNode(j.val)
            j = j.next; k = k.next
        return dummy_k.next

################################147   Insertion Sort List  ############################
Sort a linked list using insertion sort.
class Solution(object):
    def insertionSortList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        ## -inf -> 0 -> 3 ->4 -> 6 -> 1 -> 2 -> 8 -> 9 -> None
        ##                        prev cur
        ##关键，不仅要记录cur,
        还要记录cur之前的那个节点(也就是已经排好序的节点的尾巴)，如上面的例子，要把1插到0和3之间
        ##要依次更改次序：  6 -> 2, 1-> 3, 0 -> 1, cur = 2
        if not head or not head.next:
            return head
        dummyhead = ListNode(float('-inf'))
        dummyhead.next = head
        cur = head.next
        prev = head
        while cur is not None:
            tmp = dummyhead
            while tmp.next.val < cur.val:
                tmp = tmp.next
            if tmp.next == cur:
                prev = cur
                cur = cur.next
                continue
            prev.next = cur.next
            cur.next = tmp.next
            tmp.next = cur
            cur = prev.next
        return dummyhead.next

class Solution {
public:
    ListNode* insertionSortList(ListNode* head) {
        if (!head || !head->next)
```

```
            return head;
        ListNode *pseudoHead = new ListNode(INT_MIN);
        pseudoHead->next = head;

        ListNode *pre = head;
        ListNode *cur = head->next;
        ListNode *temp;
        while(cur)
        {
            for (temp = pseudoHead; temp->next->val < cur->val; temp =
            temp->next){}//找到tmp->next->val > cur->val的节点
            if (temp->next == cur)
            {
                pre = cur;
                cur = cur->next;
                continue;
            }
            pre->next = cur->next;
            cur->next= temp->next;
            temp->next = cur;
            cur = pre->next;
        }
        return pseudoHead->next;
    }
};
################################143   Reorder List ##############################
Given a singly linked list L: L0→L1→…→Ln-1→Ln,
reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You must do this in-place without altering the nodes' values.
For example,
Given {1,2,3,4}, reorder it to {1,4,2,3}.

class Solution {
public:
    void reorderList(ListNode* head) {
        if (!head || !head->next || !head->next->next)
            return;
        ListNode *slow(head), *fast(head);
        while (fast && fast->next){
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode *tmp = slow->next;
        slow->next = NULL;
        //1->2->3->4 变为:  1->2->3<-4; 3->NULL;
        ListNode *prev(slow), *post(tmp);
        while(post){
            tmp = post->next;
            post->next = prev;
            prev = post;
            post = tmp;
        }
```

```
        ListNode *tail(prev), *tmp1, *tmp2;
        cout << tail->val << endl;
        while(1){
            tmp1 = head->next;
            tmp2 = tail->next;
            if(!tmp1 || !tmp2)
                break;
            head->next = tail;
            tail->next = tmp1;
            head = tmp1;
            tail = tmp2;
        }
    }
};
class Solution(object):
    def reorderList(self, head):
        if (not head) or (not head.next) or (not head.next.next):
            return
        slow, fast = head, head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
        tmp = slow.next
        slow.next = None
        post = tmp
        prev = slow
        while post.next:
            tmp = post.next
            post.next = prev
            prev = post
            post = tmp
        post.next = prev

        list1 = head
        list2 = post
        while True:
            tmp1 = list1.next
            tmp2 = list2.next
            if not tmp1 or not tmp2:
                break
            list1.next = list2
            list2.next = tmp1
            list1 = tmp1
            list2 = tmp2
############################## 142  Linked List Cycle II  ###################
Given a linked list, return the node where the cycle begins. If there is no cycle, return null.
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        unordered_multiset<ListNode*> st;
        ListNode *tmp = head;
        while(tmp){
            st.insert(tmp);
```

```
            tmp = tmp->next;
            if (st.count(tmp) == 2)
                return tmp;
        }
        return nullptr;
    }
};
class Solution(object):
    def detectCycle(self, head):
        if not head:
            return None
        nodes = set()
        tmp = head
        while tmp:
            if tmp in nodes:
                return tmp
            nodes.add(tmp)
            tmp = tmp.next
        return None
############################## 141  Linked List Cycle ####################
Given a linked list, determine if it has a cycle in it.
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *fast(head), *slow(head);
        while (fast && fast->next){
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast)
                return 1;
        }
        return 0;
    }
};
class Solution(object):
    def hasCycle(self, head):
        if not head or not head.next:
            return False
        slow, fast = head, head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
            if slow == fast:
                return True
        return False
########################138. Copy List with Random Pointer ####################
A linked list is given such that each node contains an additional random pointer which could
point to any node in the list or null.
# Definition for singly-linked list with a random pointer.
# class RandomListNode(object):
#     def __init__(self, x):
#         self.label = x
#         self.next = None
```

```
#         self.random = None
class Solution(object):
    def copyRandomList(self, head):
        """
        :type head: RandomListNode
        :rtype: RandomListNode
        """
        #原list:  1-> 2-> 3; 1.random = x (x belong to (1,2,3,None))
        #新list:  1'->2'->3'; 1'.random = x'(x' belong to (1',2',3',None))
        #建立字典node_dic = {1:1', 2:2', 3:3'}
        #这样透过node_dic[1.random]就能得到x'
        #即:  1'.random = node_dic.get(1.random, None)
        node_dic = {}
        dummy = RandomListNode(0)
        node = dummy
        tmp = head
        while tmp:
            nextNode = RandomListNode(tmp.label)
            node_dic[tmp] = nextNode
            node.next = nextNode
            node = node.next
            tmp = tmp.next
        tmp = head
        node = dummy.next
        while tmp:
            node.random = node_dic.get(tmp.random, None)
            tmp = tmp.next
            node = node.next
        return dummy.next
class Solution {
public:
    RandomListNode *copyRandomList(RandomListNode *head) {
        RandomListNode *tmp1 = head;
        RandomListNode *dummy = new RandomListNode(0);
        RandomListNode *tmp2 = dummy;
        unordered_map <RandomListNode*, RandomListNode*> mp;
        while(tmp1){
            RandomListNode *newNode =  new RandomListNode(tmp1->label);
            mp.insert(make_pair(tmp1, newNode));
            tmp1 = tmp1->next;
            tmp2->next = newNode;
            tmp2 = tmp2->next;
        }
        tmp2 = dummy->next;
        tmp1 = head;
        while(tmp1){
            tmp2->random = mp.find(tmp1->random) == mp.end()? NULL:mp[tmp1->random];
            tmp1 = tmp1->next;
            tmp2 = tmp2->next;
        }
        return dummy->next;
    }
};
```

```
##########################92    Reverse Linked List II#################
Reverse a linked list from position m to n. Do it in-place and in one-pass.

For example:
Given 1->2->3->4->5->NULL, m = 2 and n = 4,
return 1->4->3->2->5->NULL.
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int m, int n) {
        ListNode *dummy_head = new ListNode(0);
        dummy_head->next = head;
        ListNode* temp = dummy_head;
        int cnt = 1;
        while (cnt < m){
            temp = temp->next;
            cnt++;
        }
        ListNode *start = temp;
        ListNode *end = temp->next;
        ListNode *prev(NULL), *post(end);
        cnt = m;
        while(cnt <= n){
            ListNode *tmp = post->next;
            post->next = prev;
            prev = post;
            post = tmp;
            cnt++;
        }
        start->next = prev;
        end->next = post;
        return dummy_head->next;
    }
};
class Solution(object):
    def reverseBetween(self, head, m, n):
        if not head or not head.next:
            return head
        dummy = ListNode(0)
        dummy.next = head
        tmp = dummy
        cnt = 1
        while cnt < m:
            tmp = tmp.next
            cnt += 1
        end = tmp
        prev = prev_copy = tmp.next
        post = prev.next
        prev.next = None
        while cnt < n:
            tmp = post.next
            post.next = prev
            prev = post
            post = tmp
```

```
            cnt += 1
        end.next = prev
        prev_copy.next = post
        return dummy.next
##############################86    Partition List##########################
Given a linked list and a value x, partition it such that all nodes less than x come before
nodes greater than or equal to x.
You should preserve the original relative order of the nodes in each of the two partitions.

For example,
Given 1->4->3->2->5->2 and x = 3,
return 1->2->2->4->3->5.
class Solution(object):
    def partition(self, head, x):
        """
        :type head: ListNode
        :type x: int
        :rtype: ListNode
        """
        if not head or not head.next:
            return head
        dummy = ListNode(0)
        dummy.next = head
        prev = dummy
        curr = head
        while curr and curr.next:
            if curr.val >= x:
                if curr.next.val < x:
                    tmp = curr.next.next
                    curr.next.next = prev.next
                    prev.next = curr.next
                    curr.next = tmp
                    prev = prev.next
                else:
                    curr = curr.next
            else:
                curr = curr.next
                prev = prev.next
        return dummy.next
########################83   Remove Duplicates from Sorted List##################
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.
class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head or not head.next:
            return head
        tmp = head
```

```
        while tmp.next:
            if tmp.val == tmp.next.val:
                tmp.next = tmp.next.next
                continue
            tmp = tmp.next
        return head
#########################82  Remove Duplicates from Sorted List II###################
Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct
numbers from the original list.
For example,
Given 1->2->3->3->4->4->5, return 1->2->5.
Given 1->1->1->2->3, return 2->3.
class Solution(object):
    def deleteDuplicates(self, head):
        if not head or not head.next:
            return head
        dummy = ListNode(0)
        dummy.next = head
        last = dummy
        curr = head.next
        while curr:
            if last.next.val == curr.val:
                while curr and curr.val == last.next.val:
                    curr = curr.next
                last.next = curr
                if curr:
                    curr = curr.next
            else:
                last = last.next
                curr = curr.next
        return dummy.next
#########################61    Rotate List#################################
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.

class Solution {
public:
    //in this function: k < sizeof the linklist
    ListNode *rotateRightHelper(ListNode *head, int k)
    {
        ListNode *fast = head;
        ListNode *slow = head;
        //让fast和slow拉开k的距离
        while(k > 0)
        {
            fast = fast->next;
            k--;
        }
        //fast和slow同时前进，直到fast到最后一个节点
        while(fast->next)
        {
```

```
            fast = fast->next;
            slow = slow->next;
        }
        //fast指向原来的head
        fast->next = head;
        head = slow->next;//slow->next是新的head
        slow->next = nullptr;//slow->next指向null
        return head;

    }
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next)
            return head;
        ListNode *tmp = head;
        // find out the size of the linklist
        int size = 1;
        while(tmp->next)
        {
            tmp = tmp->next;
            size++;
        }
        // if size <= k, k = k%size;
        if (size <= k)
            rotateRightHelper(head, k%size);
        else
            rotateRightHelper(head, k);
    }
};

############################25   Reverse Nodes in k-Group#########################
Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.
k is a positive integer and is less than or equal to the length of the linked list. If the
number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.
You may not alter the values in the nodes, only nodes itself may be changed.
Only constant memory is allowed.
For example,
Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5
 /* 假设k = 3; head是个虚节点。假设start = head;
     * null --> 1 --> 2 --> 3 --> 4 --> 5--> 6 --> 7 --> 8 -->....
     *   |      |     |
     * start   pre   cur
     *      (new_start)
     * -------------------经过一次reverse内for循环以后：------------------------
     * null --> 1 <-- 2 <-- 3     4 --> 5 --> 6 --> 7 --> 8 -->....
     *   |                 |     |
     * start               pre  cur
     * -------------------for循环后面两句以后：------------------------
     * null --> 3 --> 2 --> 1 --> 4 --> 5 --> 6 --> 7 --> 8 -->....
     *   |      |           |     |
```

```
     * start   pre        (new_start)  cur
     * ----------------return new_start,然后new_start又作为参数start开始下一轮--------------
     * null --> 3 --> 2 --> 1 --> 4 --> 5 --> 6 --> 7 --> 8 -->....
     *                      |     |     |
     *                    start  pre   cur
     *                         (new_start)
     * -------------------经过一次reverse内for循环以后：------------------------
     * null --> 3 --> 2 --> 1 --> 4 <-- 5 <-- 6     7 --> 8 -->....
     *                      |                |     |
     *                    start              pre   cur
     * -------------------for循环后面两句以后：------------------------
     * null --> 3 --> 2 --> 1 --> 6 --> 5 --> 4 --> 7 --> 8 -->....
     *                      |     |           |     |
     *                    start  pre     (new_start)  cur
     * ----------------return new_start,然后new_start又作为参数start开始下一轮--------------
     * null --> 3 --> 2 --> 1 --> 6 --> 5 --> 4 --> 7 --> 8 --> 9 -->...
     *                                        |     |     |
     *                                      start  pre   cur
     * ..............
     *
     * */

class Solution {
public:
    //find the length of the linklist
    int findLength(ListNode* head)
    {
        int size = 0;
        ListNode *cur = head;
        while(cur){
            size++;
            cur = cur->next;
        }
        return size;
    }
    // return the newStart node
    ListNode *reverseOnce(ListNode* start, int k)
    {
        ListNode *pre = start->next;
        ListNode *post = pre->next;
        ListNode *newStart = start->next;
        ListNode *tmp;
        for (int i = 1; i < k; i++)
        {
            tmp = post->next;
            post->next = pre;
            pre = post;
            post = tmp;
        }
        start->next->next = post;
        start->next = pre;
        return newStart;
    }
```

```
    ListNode* reverseKGroup(ListNode* head, int k) {
        int times = findLength(head) / k;
        ListNode *pseudoHead = new ListNode(0);
        pseudoHead->next = head;
        ListNode *temp = pseudoHead;
        for (int i = 0; i < times; i++)
        {
            temp = reverseOnce(temp, k);
        }
        return pseudoHead->next;
    }
};
########################24  Swap Nodes in Pairs##################
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.

class Solution(object):
    def swapPairs(self, head):
        if not head or not head.next:
            return head
        prev, post, newHead = head, head.next, head.next
        while True:
            tmp = prev
            prev.next = post.next
            post.next = prev
            if prev.next:
                prev = prev.next
            else:
                break
            if prev.next:
                post = prev.next
            else:
                break
            tmp.next = post
        return newHead

class Solution(object):
    def swapPairs(self, head):
        if not head or not head.next:
            return head
        tmp = head.next
        head.next = self.swapPairs(tmp.next)
        tmp.next = head
        return tmp
############################23  Merge k Sorted Lists########################
Merge k sorted linked lists and return it as one sorted list. Analyze and describe its
complexity.
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        //multimap 是按照key的升序排列的
```

```
        //把各个节点的head的val：各个节点的head指针，依次插入multimap

        //这样multimap里就存放了所有的链表的首节点的值（升序）以及其指针，*multimap.begin()是当前的最小的值

        //接下来，依次把multimap的首元素的值，插入新链表，擦去multimap的最小键值对，并把对应的指针向后移动（这就是为什么要记录节点的指针）
        //同时，把后面结点的值及其指针插入multimap
        //重复以上操作，直到multimap为空
        multimap<int, ListNode*> mp;
        for(ListNode* node : lists){
            if(node)
                mp.insert(make_pair((*node).val, node));
        }
        ListNode* ans = new ListNode(0);
        ListNode* tmp = ans;

        while(!mp.empty()){
            ListNode *minNode = mp.begin()->second;
            tmp->next = new ListNode(minNode->val);
            mp.erase(mp.begin());
            if( minNode->next ){
                minNode = minNode->next;
                mp.insert(make_pair( minNode->val, minNode ));
            }
            tmp = tmp->next;
        }
        return ans->next;
    }
};
##########################21    Merge Two Sorted Lists#########################
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(!l1)
            return l2;
        if(!l2)
            return l1;
        ListNode *temp1 = l1;
        ListNode *temp2 = l2;
        ListNode *ans = new ListNode(0);
        ListNode *temp = new ListNode(0);
        ans = temp;
        while(temp1 && temp2){
            if(temp1->val < temp2->val){
                temp->next = new ListNode(temp1->val);
                temp1 = temp1->next;
            }
            else{
                temp->next = new ListNode(temp2->val);
                temp2 = temp2->next;
            }
            temp = temp->next;
```

```
        }
        while(temp1){
            temp->next = new ListNode(temp1->val);
            temp1 = temp1->next;
            temp = temp->next;
        }
        while(temp2){
            temp->next = new ListNode(temp2->val);
            temp2 = temp2->next;
            temp = temp->next;
        }
        return ans->next;

    }
};
########################19   Remove Nth Node From End of List####################
Given a linked list, remove the nth node from the end of list and return its head.
For example,
   Given linked list: 1->2->3->4->5, and n = 2.
   After removing the second node from the end, the linked list becomes 1->2->3->5.
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        if(!head->next)
            return NULL;
        ListNode* prev = head;
        ListNode* post = head;
        int cnt = 0;
        while (cnt < n){
            if(!post)
                break;
            post = post->next;
            cnt++;
        }
        if(!post)
           return head->next;
        while (post->next){
            post = post->next;
            prev = prev->next;
        }
        prev->next = prev->next->next;
        return head;
    }
};
############################2   Add Two Numbers####################
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode *head = new ListNode(0);
        ListNode *tmp = new ListNode(0);
        head = tmp;
```

```
        int carry = 0;
        int num1, num2;
        while(l1 || l2 || carry){
            num1 = (l1 == NULL)? 0:l1->val;
            num2 = (l2 == NULL)? 0:l2->val;
            tmp->next = new ListNode((num1+num2+carry)%10);
            carry = (num1+num2+carry)/10;
            l1 = (l1 != NULL)? l1->next:NULL;
            l2 = (l2 != NULL)? l2->next:NULL;
            tmp = tmp->next;
        }
        return head->next;
    }
};
```

##