2. Add Two Numbers

class Solution: def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:

```
    h1 = l1
    h2 = l2
    carry = 0
    prev = None
    while h1 and h2:
        num =  h1.val+h2.val +carry
        if num < 10:
            h1.val = num
            carry = 0
        else:
            num = num%10
            h1.val = num
            carry = 1
        prev = h1
        h1 = h1.next
        h2 = h2.next
    if h2:
        prev.next = h2
        h1 = h2
    while h1:
        num =  h1.val+carry
        if num < 10:
            h1.val = num
            carry = 0
        else:
            num = num%10
            h1.val = num
            carry = 1
        prev = h1
        h1 = h1.next
    if carry:
        prev.next = Node(1)
    return l1
```

19. Remove Nth Node From End of List

```
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNo
de]:
        dummy = ListNode(0, head)
        fast = slow = dummy

        for _ in range(n):
            fast = fast.next

        while fast.next:
            fast = fast.next
            slow = slow.next

        slow.next = slow.next.next
        return dummy.next
```

---

61. Rotate List

```
class Solution:
    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        if not head or not head.next or k == 0:
            return head

        #Compute The length
        curr = head
        length = 1
        while curr.next:
            curr = curr.next
            length+=1

        curr.next = head #Make the list Circular

        #Find the Tail
        k = length - (k%length)
        while k:
            curr = curr.next
            k-=1

        #Break the circle and return the New Head
        newHead = curr.next
        curr.next = None
        return newHead
```

138. Copy List with Random Pointer

```
class Solution:
    def copyRandomList(self, head: 'Optional[Node]') -> 'Optional[Node]':
        cur=head
        i=0
        dummy=Node(100)
        head2=dummy
        mapp=[]
        original=[]
        while cur:
            original.append(cur.val)
            cur.val=i
            node=Node(i)
            mapp.append(node)
            dummy.next=node
            dummy=dummy.next
            cur=cur.next
            i+=1

        cur=head2.next
        i=0
        while head:
            if head.random is not None:
                cur.random=mapp[head.random.val]
            else:
                cur.random=None
            head=head.next
            cur.val=original[i]
            cur=cur.next
            i+=1

        return head2.next
```

141. Linked List Cycle

```
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        slow = head
        fast = head
        while fast != None and fast.next!= None:
            fast = fast.next.next
            slow = slow.next
            if fast == slow:
                return True

        return False
```

142. Linked List Cycle II

```
class Solution:
    def detectCycle(self, head: Optional[ListNode]) -> Optional[ListNode]:
        slow = head
        fast = head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next

            if slow == fast:
                slow = head
                while fast != slow:
                    fast = fast.next
                    slow = slow.next
                return slow
        return None
```

148. Sort List

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def divide_LL(self,head):
        slow = fast = head
        prev = None
        while fast and fast.next:
            prev = slow
            slow = slow.next
            fast = fast.next.next
        if prev:
            prev.next = None
        return slow

    def merge(self,l1,l2):
        dummy = ListNode(-1)
        cur = dummy
        while l1 and l2:
            if l1.val < l2.val:
                cur.next = l1
                l1= l1.next
            else:
                cur.next = l2
                l2 = l2.next
            cur = cur.next

        cur.next = l1 if l1 else l2
        return dummy.next

    def merge_sort(self,node):
        if not node or not node.next:
            return node

        mid_head = self.divide_LL(node)
        left = self.merge_sort(node)
        right = self.merge_sort(mid_head)
        return self.merge(left,right)
    def sortList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        return self.merge_sort(head)
```

160. Intersection of Two Linked Lists

```
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> Optional[Lis
tNode]:
        l1 = headA
        l2 = headB
        while l1 != l2:
            l1 = l1.next if l1 else headB
            l2 = l2.next if l2 else headA
        return l1
```

206. Reverse Linked List

```
class Solution:
    def reverseList(self, root: Optional[ListNode]) -> Optional[ListNode]:

        def reverse(head):
            if not head or not head.next:
                return head

            new_head = reverse(head.next)
            front = head.next
            front.next = head
            head.next = None
            return new_head

        return reverse(root)
```

234. Palindrome Linked List

```
class Solution:
    def isPalindrome(self, head: Optional[ListNode]) -> bool:
        cur = head
        prev = None
        data = ""
        while cur:
            data += str(cur.val)
            temp = cur
            cur = cur.next
            temp.next = prev
            prev = temp
        return data[::-1] == data
```

237. Delete Node in a Linked List

```
class Solution:
    def deleteNode(self, node):
        node.val = node.next.val
        if node.next.next != None:
            node.next = node.next.next
        else:
            node.next = None
```

328. Odd Even Linked List

```
class Solution:
    def oddEvenList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head or not head.next:
            return head

        odd = head
        even = head.next
        even_head = even

        while even and even.next:
            odd.next = even.next
            odd = odd.next
            even.next= odd.next
            even = even.next
        odd.next = even_head
        return head
```

876. Middle of the Linked List

```
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head:return head

        slow = head
        fast = head
        while fast != None and fast.next != None:
            fast = fast.next.next
            slow = slow.next
        return slow
```

2095. Delete the Middle Node of a Linked List

```
class Solution:
    def deleteMiddle(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head or head.next == None:
            return None
        dummy = ListNode(-1)
        dummy.next = head
        slow = fast =head
        prev  = None
        while fast and fast.next:
            prev = slow
            slow = slow.next
            fast = fast.next.next

        prev.next = slow.next
        return dummy.next
```
