LinkedList
==========

Reverse a Linked List
=====================

```
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        prev = None
        curr = head
        while(curr!=None):
            temp_next = curr.next
            curr.next = prev
            prev = curr
            curr = temp_next

        return prev
```

this 3 pointer solution has a time complexity of O(n)
space complexity is O(1)

Recursion
==========

A function calling itself.

Recursion:
1. There should be a base condition (a condition where we have to stop)
2. We have to break it into smaller problem

Factorial of a number 4

factorial(4) = 4 * factorial(3)

factorial(n) = n * factorial(n-1)

factorial(1) = 1

base condtion - fact(1) = 1
smaller problem - fact(n) = n * fact(n-1)

```
class Solution:
  def factorial(self, num:int) -> int:
    if num == 1:
      return 1
    smaller_prob = self.factorial(num-1)
    return num * smaller_prob

s = Solution()
s.factorial(2)
```

find sum of n natural numbers

input = 10

output = 55

10 numbers

10 + sum of remaining 9

5!

5 * 4!

```
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if (head == None or head.next == None):
            return head
        new_head = self.reverseList(head.next)
        head.next.next = head
        head.next = None
        return new_head
```

time complexity = O(n)

Space complexity = O(n)

==========

Middle of the Linked List
==========

```
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if head == None:
            return head
        temp = head
        count = 0
        while (temp!=None):
            count = count + 1
            temp = temp.next

        middle_ele = count//2 + 1
        temp = head
        i = 1
        while (i < middle_ele):
            temp = temp.next
            i = i+1

        return temp
```

Time Complexity = O(2n) ~ O(n)
space complexity = O(1)

==========

slow pointer and a fast pointer

==========

```
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if (head == None or head.next == None):
            return head
        slow = head
        fast = head

        while (fast!=None and fast.next!=None):
            slow = slow.next
            fast = fast.next.next
```

```
        return slow
```

Merge 2 sorted LinkedLists
==========

```
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) ->
Optional[ListNode]:

        prehead = ListNode(-1)
        curr = prehead

        while list1!=None and list2!=None :
            if list1.val < list2.val:
                curr.next = list1
                list1 = list1.next
            else:
                 curr.next = list2
                 list2 = list2.next
            curr = curr.next
        if list1!=None:
            curr.next = list1
        else:
            curr.next = list2

        return prehead.next
```

Solution using recursion
==========

stopping or base condition for this

```
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) ->
Optional[ListNode]:
        if list1 == None:
            return list2
        elif list2 == None:
            return list1
        elif list1.val < list2.val :
```

```
            list1.next = self.mergeTwoLists(list1.next, list2)
            return list1
        else:
            list2.next  = self.mergeTwoLists(list1, list2.next)
            return list2
```

Remove nth node from the End of LinkedList

==========

```
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNode]:
        temp = head
        count = 0
        while(temp!=None):
            count = count + 1
            temp = temp.next

        i = 1
        temp = head
        if count == n:
            return head.next
        while (i < (count-n)):
            temp = temp.next
            i = i + 1
        temp.next = temp.next.next

        return head
```

==========

```
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNode]:
        dummy = ListNode(-1)
        dummy.next = head
        fast = dummy
        slow = dummy

        i = 1

        while (i <= n):
            fast = fast.next
            i = i + 1

        while (fast.next!=None):
```

```
    fast = fast.next
    slow = slow.next

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