**Linked List**

1. Add two numbers

class Solution:

    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:

        current = ListNode(0)

        temp = current

        carry = 0

        while l1 or l2 or carry:

            num1 = l1.val if l1 else 0

            num2 = l2.val if l2 else 0

            value = num1 + num2 + carry

            # when value > 9, only ten's digit is stored otherwise entire num

            current.next = ListNode(value%10)

            # Store the carry

            carry = value // 10

            # increment the linked lists

            l1 = l1.next if l1 else None

            l2 = l2.next if l2 else None

        return temp.next

2. Add two numbers 2

3. Remove Nth node from the end

class Solution:

    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:

        '''

        Approach 1: Move first till nth node and

        then move both first and second till first reaches end

        TC: O(n)

        SC: O(1)

        '''

        # base case

        if head.next is None:

            return

        temp = ListNode(0)

        temp.next = head

        i = 0

        first = temp

        second = temp

        while first.next is not None:

            if i < n:

                first = first.next

                i += 1

            else:

                second = second.next

                first = first.next

        # second will move till a node lesser than the node to be removed.

        # update the second's next with address of the next next node from second

        second.next = second.next.next

        return temp.next

3 Delete Node in a Linked List

4 Odd Even Linked List

class Solution:

    def oddEvenList(self, head: ListNode) -> ListNode:

        '''

        Approach 1:

        TC: O(n)

        SC: O(1)

        '''

        # base case

        if head is None:

            return

        odd\_head = head     # Initialize odd head

        even\_head = head.next # initialize even head

        temp = even\_head

        while  odd\_head.next is not None and temp.next is not None:

            odd\_head.next = temp.next

            odd\_head = odd\_head.next

            temp.next = odd\_head.next

            temp = temp.next

        # after odd nodes are linked and even nodes are linked. link the odd\_head to even\_head

        odd\_head.next = even\_head

        return head

5. Merge two sorted lists

6. Merge k sorted lists

7. Partition list

class Solution(object):

    def partition(self, head, x):

        '''

        Approach: Create a two lists, one list with nodes less than x and other greater than or equal to x

        Link last node of smaller list with head of the larger list.

        TC: O(n)

        SC: O(1)

        '''

        # before and after are the two pointers used to create two list

        # before\_head and after\_head are used to save the heads of the two lists.

        # All of these are initialized with the dummy nodes created.

        before = before\_head = ListNode(0)

        after = after\_head = ListNode(0)

        while head:

            # If the original list node is lesser than the given x,

            # assign it to the before list.

            if head.val < x:

                before.next = head

                before = before.next

            else:

                # If the original list node is greater or equal to the given x,

                # assign it to the after list.

                after.next = head

                after = after.next

            # move ahead in the original list

            head = head.next

        # Last node of "after" list would also be ending node of the reformed list

        after.next = None

        # Once all the nodes are correctly assigned to the two lists,

        # combine them to form a single list which would be returned.

        before.next = after\_head.next

        return before\_head.next

8. Linked List cycle

class Solution:

    def hasCycle(self, head: ListNode) -> bool:

        # fast and slow pointer

        fast, slow = head, head

        # loop until the fast pointer reaches the last node

        while fast is not None and fast.next is not None:

            slow = slow.next

            fast = fast.next.next

            if slow == fast:

                return True

        return False

9. Linked List cycle 2

class Solution(object):

    def getIntersect(self, head):

        tortoise = head

        hare = head

        # A fast pointer will either loop around a cycle and meet the slow

        # pointer or reach the `null` at the end of a non-cyclic list.

        while hare is not None and hare.next is not None:

            tortoise = tortoise.next

            hare = hare.next.next

            if tortoise == hare:

                return tortoise

        return None

    def detectCycle(self, head):

        if head is None:

            return None

        # If there is a cycle, the fast/slow pointers will intersect at some

        # node. Otherwise, there is no cycle, so we cannot find an entrance to

        # a cycle.

        intersect = self.getIntersect(head)

        if intersect is None:

            return None

        # To find the entrance to the cycle, we have two pointers traverse at

        # the same speed -- one from the front of the list, and the other from

        # the point of intersection.

        ptr1 = head

        ptr2 = intersect

        while ptr1 != ptr2:

            ptr1 = ptr1.next

            ptr2 = ptr2.next

        return ptr1

10. Reverse Linked List

class Solution:

    def reverse(self, head: ListNode) -> ListNode:

        prev = None

        current = head

        while current is not None:

            next\_node = current.next

            current.next = prev

            prev = current

            current = next\_node

        return prev

11. Reverse LL between

'''

TC: O(n)

SC: O(1)

'''

class Solution:

    def reverseBetween(self, head: ListNode, m: int, n: int) -> ListNode:

        current, previous = head, None

        count = 0

        # base case

        if m==n:

            return head

        # iterate till the start of the sublist

        while current is not None and count < m-1:

            previous = current

            current = current.next

            count += 1

        # store the node before the sublist

        node\_before\_reverse = previous

        # store the first node of the sublist

        node\_sublist = current

        count = 0

        # Reverse the sublist

        while current is not None and count < n-m + 1:

            next\_node = current.next

            current.next = previous

            previous = current

            current = next\_node

            count += 1

        # link the last node of the list before reversing the next nodes to the last node of the list to be reversed

        if node\_before\_reverse is not None:

            node\_before\_reverse.next = previous

        else:

            head = previous

        node\_sublist.next = current

        return head

12. Reverse LL k groups alternatively

'''

TC: O(n)

SC: O(1)

'''

class Solution:

    def reverseKGroup(self, head: ListNode, k: int) -> ListNode:

        # base case

        if k<= 1 or head is None:

            return head

        current, previous = head, None

        # Similar to reverse a sublist but have to do it for multiple sublists. Infinite loop and break it

        # when it reaches end.

        while True:

            node\_before = previous

            lastnode\_sublist = current

            i = 0

            # reverse

            while current is not None and i<k:

                next\_node = current.next

                current.next = previous

                previous = current

                current = next\_node

                i += 1

            # link the node before reverse to the last node of the reversed sublist.

            if node\_before is not None:

                node\_before.next = previous

            else:

                head = previous

            # node before reversal should point to the start of the next sublist.

            lastnode\_sublist.next = current

            # breaking the loop condiition

            if current is None:

                break

            # update the previous as the last reversed node.

            previous = lastnode\_sublist

            # Continue the same for next sublist.

        return head

13. Palindrome Linked List

'''

Approach 1: Find the middle and reverse the 2nd half of the LL, compare both

TC: O(n)

SC: O(1)

'''

class Solution:

    def isPalindrome(self, head: ListNode) -> bool:

        if head is None:

            return True

        # Find the end of first half and reverse second half.

        first\_half\_end = self.end\_of\_first\_half(head)

        second\_half\_start = self.reverse\_list(first\_half\_end.next)

        # Check whether or not there's a palindrome.

        result = True

        first\_position = head # first node of the 1st list

        second\_position = second\_half\_start # first node of the 2nd list

        while result and second\_position is not None:

            if first\_position.val != second\_position.val:

                result = False

            first\_position = first\_position.next

            second\_position = second\_position.next

        # Restore the list and return the result.

        first\_half\_end.next = self.reverse\_list(second\_half\_start)

        return result

    def end\_of\_first\_half(self, head):

        fast = head

        slow = head

        while fast.next is not None and fast.next.next is not None:

            fast = fast.next.next

            slow = slow.next

        return slow

    def reverse\_list(self, head):

        previous = None

        current = head

        while current is not None:

            next\_node = current.next

            current.next = previous

            previous = current

            current = next\_node

        return previous

14. Remove Duplicates from Sorted List

15. Remove Duplicates from Sorted List II

16. Sort list

17. Intersection of Two Linked Lists

18. Plus One list

'''

TC: O(n)

SC: O(1)

'''

class Solution:

    def reverseKGroup(self, head: ListNode, k: int) -> ListNode:

        # base case

        if k<= 1 or head is None:

            return head

        current, previous = head, None

        # Similar to reverse a sublist but have to do it for multiple sublists. Infinite loop and break it

        # when it reaches end.

        while True:

            node\_before = previous

            lastnode\_sublist = current

            i = 0

            # reverse

            while current is not None and i<k:

                next\_node = current.next

                current.next = previous

                previous = current

                current = next\_node

                i += 1

            # link the node before reverse to the last node of the reversed sublist.

            if node\_before is not None:

                node\_before.next = previous

            else:

                head = previous

            # node before reversal should point to the start of the next sublist.

            lastnode\_sublist.next = current

            # breaking the loop condiition

            if current is None:

                break

            # update the previous as the last reversed node.

            previous = lastnode\_sublist

            # Continue the same for next sublist.

        return head

19. Copy list with random pointer

20. Flatten doubly LL

21. Swap nodes in pairs

22. Convert BST to sorted doubly LL

23. Insert to sorted circular LL

24. Remove LL elements

25. Convert sorted list to BST

26. Reorder list