Q1. Given two linked list of the same size, the task is to create a new linked list using those linked lists. The condition is that the greater node among both linked list will be added to the new linked list.

**def** newList(root1, root2):

    ptr1 **=** root1

    ptr2 **=** root2

    root **=** None

**while** (ptr1 !**=** None) :

        temp **=** Node(0)

        temp.next **=** None

        # Compare for greater node

**if** (ptr1.data < ptr2.data):

            temp.data **=** ptr2.data

**else**:

            temp.data **=** ptr1.data

**if** (root **==** None):

            root **=** temp

**else** :

            ptr **=** root

**while** (ptr.next !**=** None):

                ptr **=** ptr.next

            ptr.next **=** temp

        ptr1 **=** ptr1.next

        ptr2 **=** ptr2.next

**return** root

Q2. Write a function that takes a list sorted in non-decreasing order and deletes any duplicate nodes from the list. The list should only be traversed once.

class Solution:

    def deleteDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:

        curr=head

        while (curr):

            while curr.next and curr.next.val==curr.val:

                curr.next=curr.next.next

            curr=curr.next

        return head

Q3. Reverse k node

class Solution:

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

curr = head

for \_ in range(k):

if not curr: return head

curr = curr.next

prev = None

curr = head

for \_ in range(k):

nxt = curr.next

curr.next = prev

prev = curr

curr = nxt

head.next = self.reverseKGroup(curr, k)

return prev

Q4. Given a linked list, write a function to reverse every alternate k nodes (where k is an input to the function) in an efficient way. Give the complexity of your algorithm.

**def** kAltReverse(head, k):

    prev **=** None

    curr **=** head

    temp **=** None

    tail **=** None

    newHead **=** None

    join **=** None

    t **=** 0

**while** (curr !**=** None) :

        t **=** k

        join **=** curr

        prev **=** None

**while** (curr !**=** None **and** t > 0):

            t **=** t **-** 1

            temp **=** curr.next

            curr.next **=** prev

            prev **=** curr

            curr **=** temp

**if** (newHead **==** None):

            newHead **=** prev

**if** (tail !**=** None):

            tail.next **=** prev

        tail **=** join

        tail.next **=** curr

        t **=** k

**while** (curr !**=** None **and** t > 0):

            t **=** t **-** 1

            prev **=** curr

            curr **=** curr.next

        tail **=** prev

**return** newHead

Q5. Given a linked list and a key to be deleted. Delete last occurrence of key from linked. The list may have duplicates.

**def** deleteLast(head, x):

    temp **=** head

    ptr **=** None

**while** (temp !**=** None):

        # If found key, update

**if** (temp.data **==** x):

            ptr **=** temp

        temp **=** temp.next

    # If the last occurrence is the last node

**if** (ptr !**=** None **and** ptr.next **==** None):

        temp **=** head

**while** (temp.next !**=** ptr):

            temp **=** temp.next

        temp.next **=** None

    # If it is not the last node

**if** (ptr !**=** None **and** ptr.next !**=** None):

        ptr.data **=** ptr.next.data

        temp **=** ptr.next

        ptr.next **=** ptr.next.next

**return** head

Q6. Given two sorted linked lists consisting of **N** and **M** nodes respectively. The task is to merge both of the lists (in place) and return the head of the merged list.

class Solution:

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

        cur = dummy = ListNode()

        while list1 and list2:

            if list1.val < list2.val:

                cur.next = list1

                list1, cur = list1.next, list1

            else:

                cur.next = list2

                list2, cur = list2.next, list2

        if list1 or list2:

            cur.next = list1 if list1 else list2

        return dummy.next

Q7. Given a **Doubly Linked List**, the task is to reverse the given Doubly Linked List.

**def** reverse(self):

        temp **=** None

        current **=** self.head

**while** current **is** **not** None:

            temp **=** current.prev

            current.prev **=** current.next

            current.next **=** temp

            current **=** current.prev

**if** temp **is** **not** None:

            self.head **=** temp.prev

Q8. Given a doubly linked list and a position. The task is to delete a node from given position in a doubly linked list.

**def** deleteNode(head\_ref, del\_):

**if** (head\_ref **==** None **or** del\_ **==** None):

**return**

**if** (head\_ref **==** del\_):

        head\_ref **=** del\_.next

**if** (del\_.next !**=** None):

        del\_.next.prev **=** del\_.prev

**if** (del\_.prev !**=** None):

        del\_.prev.next **=** del\_.next

**return** head\_ref

**def** deleteNodeAtGivenPos(head\_ref,n):

**if** (head\_ref **==** None **or** n <**=** 0):

**return**

    current **=** head\_ref

    i **=** 1

**while** ( current !**=** None **and** i < n ):

        current **=** current.next

        i **=** i **+** 1

**if** (current **==** None):

**return**

    deleteNode(head\_ref, current)

**return** head\_ref