Q1

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

# Function to insert node in a linked list

**def** insert(root, item):

    temp **=** Node(0)

    temp.data **=** item

    temp.next **=** None

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

        root **=** temp

**else** :

        ptr **=** root

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

            ptr **=** ptr.next

        ptr.next **=** temp

**return** root

# Function which returns 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

**def** display(root):

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

**print**(root.data, "->", end **=** " ")

        root **=** root.next

    print(" ");

# Driver Code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    root1 **=** None

    root2 **=** None

    root **=** None

    # First linked list

    root1 **=** insert(root1, 5)

    root1 **=** insert(root1, 2)

    root1 **=** insert(root1, 3)

    root1 **=** insert(root1, 8)

**print**("First List: ", end **=** " ")

    display(root1)

    # Second linked list

    root2 **=** insert(root2, 1)

    root2 **=** insert(root2, 7)

    root2 **=** insert(root2, 4)

    root2 **=** insert(root2, 5)

**print**("Second List: ", end **=** " ")

    display(root2)

    root **=** newList(root1, root2)

**print**("New List: ", end **=** " ")

    display(root)

Q2

class Solution:

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

if not head:

return None

curr = head

while curr.next:

if curr.val == curr.next.val:

curr.next = curr.next.next

else:

curr = curr.next

return head

Q3

class Solution:

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

nodes = []

cnt = 0

cpy = head

while head:

nodes.append(head)

head = head.next

cnt += 1

if cnt == k:

cnt = 0

m,n = 0,k-1

while m<n:

nodes[m].val,nodes[n].val = nodes[n].val,nodes[m].val

m,n = m+1,n-1

nodes = []

return cpy