Q1

class Node:

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

self.data = data

self.next = None

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

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(" ");

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]:

dummy = ListNode(-1)

dummy.next = head

cur = head

count = 0

while cur:

count += 1

cur = cur.next

pre = dummy

while count >= k:

cur = pre.next

nex = cur.next

for i in range(k-1):

cur.next = nex.next

nex.next = pre.next

pre.next = nex

nex = cur.next

pre = cur

count -= k

return dummy.next

Q4

class Solution:

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

dummy = ListNode(-1)

dummy.next = head

cur = head

count = 0

while cur:

count += 1

cur = cur.next

pre = dummy

while count >= k:

cur = pre.next

nex = cur.next

for i in range(k-1):

cur.next = nex.next

nex.next = pre.next

pre.next = nex

nex = cur.next

pre = cur

count -= k

return dummy.next

Q6

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