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]:

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

Q4

  class Solution:

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

            if not head: return head

            n1, n2, head2 = head, head.next, head.next

            while n2 and n2.next:

                n1.next, n2.next = n1.next.next, n2.next.next

                n1, n2 = n1.next, n2.next

            n1.next = head2

            return head

Q6

class Solution:

def rotateRight(self, head, k):

if not head:

return head

cur= head

length =1

while cur.next:

cur = cur.next

length+=1

cur.next = head

k= length - (k%length)

while k>0:

cur=cur.next

k-=1

newhead = cur.next

cur.next=None

return newhead

Q7

class Solution:

def nextLargerNodes(self, head: Optional[ListNode]) -> List[int]:

ans = []

stack = []

i = 0

curr = head

while(curr):

ans.append(0)

curr = curr.next

while(head):

while(stack and stack[-1][1] < head.val):

index, \_ = stack.pop()

ans[index] = head.val

stack.append([i, head.val])

i += 1

head = head.next

return ans

Q8

class Solution:

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

dummy = ListNode(0)

dummy.next = head

sum\_ = 0

dict\_ = {sum\_:dummy}

while head:

sum\_ += head.val

dict\_[sum\_] = head

head = head.next

head = dummy

sum\_ = 0

while head:

sum\_ += head.val

head.next = dict\_[sum\_].next

head = head.next

return dummy.next