Q.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.

Input: list1 = 5->2->3->8

list2 = 1->7->4->5

Output: New list = 5->7->4->8

Input:list1 = 2->8->9->3

list2 = 5->3->6->4

Output: New list = 5->8->9->4

Q. 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.

For example if the linked list is 11->11->11->21->43->43->60 then removeDuplicates() should convert the list to 11->21->43->60.

**Example 1:**

Input:

LinkedList:

11->11->11->21->43->43->60

Output:

11->21->43->60

Input:

LinkedList:

10->12->12->25->25->25->34

Output:

10->12->25->34

Q. Given a linked list of size **N**. The task is to reverse every **k** nodes (where k is an input to the function) in the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should be considered as a group and must be reversed (See Example 2 for clarification).

**Example 1:**

Input:

LinkedList: 1->2->2->4->5->6->7->8

K = 4

Output:4 2 2 1 8 7 6 5

Explanation:

The first 4 elements 1,2,2,4 are reversed first

and then the next 4 elements 5,6,7,8. Hence, the

resultant linked list is 4->2->2->1->8->7->6->5.

Input:

LinkedList: 1->2->3->4->5

K = 3

Output:3 2 1 5 4

Explanation:

The first 3 elements are 1,2,3 are reversed

first and then elements 4,5 are reversed.Hence,

the resultant linked list is 3->2->1->5->4.

Q. 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.

Inputs: 1->2->3->4->5->6->7->8->9->NULL and k = 3

Output: 3->2->1->4->5->6->9->8->7->NULL.

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

Input: 1->2->3->5->2->10, key = 2

Output: 1->2->3->5->10

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

Input: 1->2->3->5->2->10, key = 2

Output: 1->2->3->5->10

Q. 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.

**Examples:**

Input: a: 5->10->15, b: 2->3->20

Output: 2->3->5->10->15->20

Input: a: 1->1, b: 2->4

Output: 1->1->2->4

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

**Example:**

Original Linked list 10 8 4 2

Reversed Linked list 2 4 8 10

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

Input:

LinkedList = 1 <--> 3 <--> 4

x = 3

Output:1 3

Explanation:After deleting the node at

position 3 (position starts from 1),

the linked list will be now as 1->3.

Input:

LinkedList = 1 <--> 5 <--> 2 <--> 9

x = 1

Output:5 2 9

Ans: Sure! I can help you with those tasks. Here are the solutions to the given problems:

1. Creating a new linked list using two linked lists:

```python

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

def create\_new\_list(list1, list2):

if not list1 or not list2:

return None

new\_list = None

tail = None

node1 = list1

node2 = list2

while node1 and node2:

if node1.data >= node2.data:

new\_node = Node(node1.data)

node1 = node1.next

else:

new\_node = Node(node2.data)

node2 = node2.next

if not new\_list:

new\_list = new\_node

tail = new\_node

else:

tail.next = new\_node

tail = new\_node

while node1:

new\_node = Node(node1.data)

tail.next = new\_node

tail = new\_node

node1 = node1.next

while node2:

new\_node = Node(node2.data)

tail.next = new\_node

tail = new\_node

node2 = node2.next

return new\_list

```

2. Deleting duplicate nodes from a sorted linked list:

```python

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

def remove\_duplicates(head):

if not head:

return None

current = head

while current and current.next:

if current.data == current.next.data:

current.next = current.next.next

else:

current = current.next

return head

```

3. Reversing every k nodes in a linked list:

```python

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

def reverse\_k\_nodes(head, k):

if not head or not head.next or k < 2:

return head

def reverse\_sublist(start, end):

prev = None

current = start

next\_node = None

while current != end:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

start.next = end

if start == head:

head = prev

else:

prev\_start.next = prev

return head

count = 1

prev\_start = None

current = head

while current:

if count % k == 0:

head = reverse\_sublist(prev\_start.next, current.next)

prev\_start = current

current = current.next

else:

current = current.next

count += 1

return head

```

4. Reversing every alternate k nodes in a linked list:

```python

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

def reverse\_alternate\_k\_nodes(head, k):

if not head or not head.next or k < 2:

return head

def reverse\_sublist(start, end):

prev = None

current = start

next\_node = None

while current != end:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

start.next = end

return prev

count = 1

prev\_start = None

current = head

while current:

if count % (2 \* k) == 1:

prev\_start = current

current = current.next

elif count % (2 \* k) == 0:

prev\_start.next = reverse\_sublist(prev\_start.next, current.next)

prev\_start = current

current = current.next

else:

current = current.next

count += 1

return head

```

5. Deleting the last occurrence of a key in a linked list:

```python

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

def delete\_last\_occurrence(head, key):

if not head:

return None

last\_occurrence = None

current = head

while current:

if current.data == key:

last\_occurrence = current

current = current.next

if not last\_occurrence:

return head

if last\_occurrence == head:

head = head.next

return head

current = head

while current.next != last\_occurrence:

current = current.next

current.next = last\_occurrence.next

return head

```

6. Merging two sorted linked lists:

```python

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

def merge\_sorted\_lists(head1, head2):

if not head1:

return head2

if not head2:

return head1

if head1.data <= head2.data:

merged\_head = head1

head1 = head1.next

else:

merged\_head = head2

head2 = head2.next

current = merged\_head

while head1 and head2:

if head1.data <= head2.data:

current.next = head1

head1 = head1.next

else:

current.next = head2

head2 = head2.next

current = current.next

if head1:

current.next = head1

else:

current.next = head2

return merged\_head

```

7. Reversing a doubly linked list:

```python

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

self.prev = None

def reverse\_doubly\_linked\_list(head):

if not head or not head.next:

return head

current = head

prev\_node = None

while current:

next\_node = current.next

current.next = prev\_node

current.prev = next\_node

prev\_node = current

current = next\_node

head = prev\_node

return head

```

8. Deleting a node from a given position in a doubly linked list:

```python

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

self.prev = None

def delete\_node\_from\_position(head, position):

if not head:

return None

current = head

if position == 1:

head = current.next

if head:

head.prev = None

return head

count = 1

while current and count != position:

current = current.next

count += 1

if not current:

return head

current.prev.next = current.next

if current.next:

current.next.prev = current.prev

return head

```

The complexity of each algorithm is as follows:

1. Creating a new linked list: O(N), where N is the size of the larger linked list.

2. Deleting duplicate nodes: O(N), where N

is the number of nodes in the linked list.

3. Reversing every k nodes: O(N), where N is the number of nodes in the linked list.

4. Reversing every alternate k nodes: O(N), where N is the number of nodes in the linked list.

5. Deleting the last occurrence of a key: O(N), where N is the number of nodes in the linked list.

6. Merging two sorted linked lists: O(N+M), where N and M are the sizes of the two linked lists.

7. Reversing a doubly linked list: O(N), where N is the number of nodes in the linked list.

8. Deleting a node from a given position: O(N), where N is the number of nodes in the linked list.