**Assignment Questions 13**

**Question 1**

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.

**Examples:**

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

**CODE:**

**class** ListNode:

**def** \_\_init\_\_(self, val**=**0, next**=None**):

self**.**val **=** val

self**.**next **=** next

**def** createNewList(list1, list2):

dummy **=** ListNode() *# Dummy node to simplify the insertion*

curr **=** dummy

**while** list1 **is** **not** **None** **and** list2 **is** **not** **None**:

**if** list1**.**val **>=** list2**.**val:

curr**.**next **=** ListNode(list1**.**val)

list1 **=** list1**.**next

**else**:

curr**.**next **=** ListNode(list2**.**val)

list2 **=** list2**.**next

curr **=** curr**.**next

*# If any of the lists still has remaining nodes, append them to the new list*

**if** list1 **is** **not** **None**:

curr**.**next **=** list1

**if** list2 **is** **not** **None**:

curr**.**next **=** list2

**return** dummy**.**next

**Question 2**

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

**Example 2:**

Input:

LinkedList:

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

Output:

10->12->25->34

**CODE:**

**class** ListNode:

**def** \_\_init\_\_(self, val**=**0, next**=None**):

self**.**val **=** val

self**.**next **=** next

**def** removeDuplicates(head):

**if** **not** head **or** **not** head**.**next:

**return** head

current **=** head

**while** current**.**next:

**if** current**.**val **==** current**.**next**.**val:

current**.**next **=** current**.**next**.**next

**else**:

current **=** current**.**next

**return** head

**Question 3**

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.

**Example 2:**

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.

**CODE:**

**class** ListNode:

**def** \_\_init\_\_(self, val**=**0, next**=None**):

self**.**val **=** val

self**.**next **=** next

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

**if** **not** head **or** k **==** 1:

**return** head

dummy **=** ListNode(0)

dummy**.**next **=** head

prev **=** dummy

count **=** 0

**while** head:

count **+=** 1

**if** count **%** k **==** 0:

prev **=** reverse(prev, head**.**next)

head **=** prev**.**next

**else**:

head **=** head**.**next

**return** dummy**.**next

**def** reverse(prev, next\_node):

last **=** prev**.**next

curr **=** last**.**next

**while** curr **!=** next\_node:

last**.**next **=** curr**.**next

curr**.**next **=** prev**.**next

prev**.**next **=** curr

curr **=** last**.**next

**return** last

**Question 4**

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.

**Example:**

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.

**CODE:**

**class** ListNode:

**def** \_\_init\_\_(self, val**=**0, next**=None**):

self**.**val **=** val

self**.**next **=** next

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

**if** **not** head **or** k **<=** 1:

**return** head

current **=** head

prev **=** **None**

count **=** 0

is\_reverse **=** **True**

**while** current:

**if** is\_reverse **and** count **<** k:

next\_node **=** current**.**next

current**.**next **=** prev

prev **=** current

current **=** next\_node

count **+=** 1

**else**:

prev**.**next **=** current

**for** \_ **in** range(k):

**if** **not** current:

**break**

prev **=** current

current **=** current**.**next

count **=** 0

is\_reverse **=** **not** is\_reverse

return head

**Question 5**

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

**Examples**:

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

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

**CODE:**

**class** ListNode:

**def** \_\_init\_\_(self, val**=**0, next**=None**):

self**.**val **=** val

self**.**next **=** next

**def** deleteLastOccurrence(head, key):

**if** **not** head:

**return** head

dummy **=** ListNode(0)

dummy**.**next **=** head

prev **=** **None**

last\_occurrence **=** **None**

current **=** dummy

*# Traverse the linked list to find the last occurrence of the key*

**while** current**.**next:

**if** current**.**next**.**val **==** key:

last\_occurrence **=** current**.**next

prev **=** current

current **=** current**.**next

*# If the last occurrence is found, delete it by updating the prev.next pointer*

**if** last\_occurrence:

prev**.**next **=** last\_occurrence**.**next

**return** dummy**.**next

**Question 6**

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

**CODE:**

**def** mergeTwoLists(a, b):

dummy **=** ListNode(0)

prev **=** dummy

*# Merge the two lists until one of them reaches its end*

**while** a **and** b:

**if** a**.**val **<=** b**.**val:

prev**.**next **=** a

a **=** a**.**next

**else**:

prev**.**next **=** b

b **=** b**.**next

prev **=** prev**.**next

*# Append the remaining nodes of the non-empty list*

**if** a:

prev**.**next **=** a

**if** b:

prev**.**next **=** b

**return** dummy**.**next

**Question 7**

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

**CODE:**

**class** Node:

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

self**.**data **=** data

self**.**prev **=** **None**

self**.**next **=** **None**

**def** reverseDoublyLinkedList(head):

current **=** head

prev **=** **None**

**while** current:

next\_node **=** current**.**next

current**.**next **=** prev

current**.**prev **=** next\_node

prev **=** current

current **=** next\_node

**return** prev

**Question 8**

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

**Example 1:**

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.

**CODE:**

**class** Node:

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

self**.**data **=** data

self**.**prev **=** **None**

self**.**next **=** **None**

**def** deleteNode(head, position):

*# If position is 1, delete the head node*

**if** position **==** 1:

**if** head **is** **None**:

**return** **None**

new\_head **=** head**.**next

**if** new\_head:

new\_head**.**prev **=** **None**

**return** new\_head

current **=** head

count **=** 1

*# Traverse to the given position*

**while** current **and** count **<** position:

current **=** current**.**next

count **+=** 1

*# If current is None, position is beyond the length of the list*

**if** current **is** **None**:

**return** head

*# Update the pointers of previous and next nodes*

current**.**prev**.**next **=** current**.**next

**if** current**.**next:

current**.**next**.**prev **=** current**.**prev

*# Delete the current node*

current**.**next **=** **None**

current**.**prev **=** **None**

**return** head