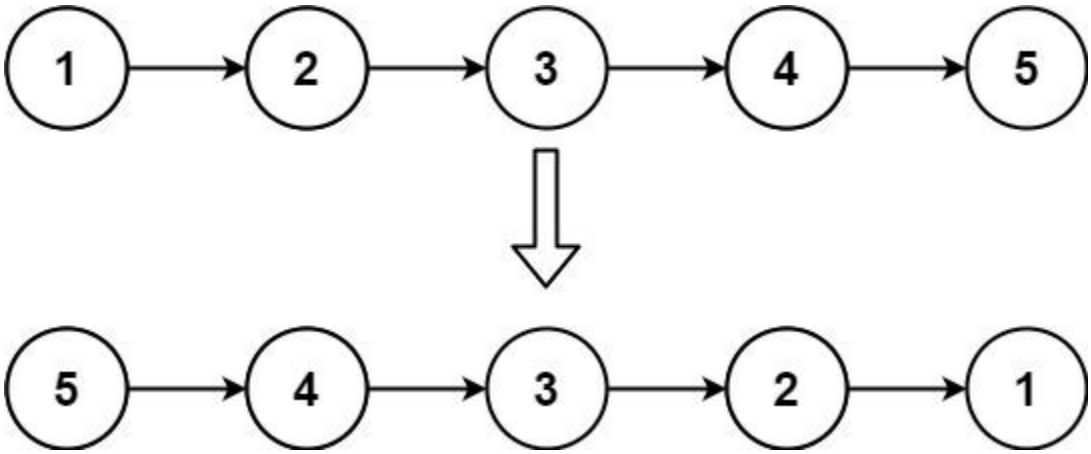
206. Reverse Linked List

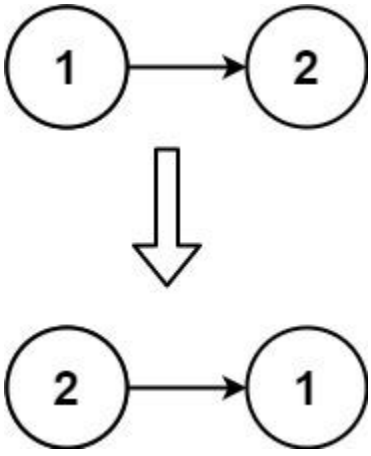
Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:



Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]

Example 2:



Input: head = [1,2]
Output: [2,1]

Example 3:

Input: head = []
Output: []

Constraints:

- The number of nodes in the list is the range `[0, 5000]`.
- `5000 <= Node.val <= 5000`

Follow up: A linked list can be reversed either iteratively or recursively. Could you implement both?

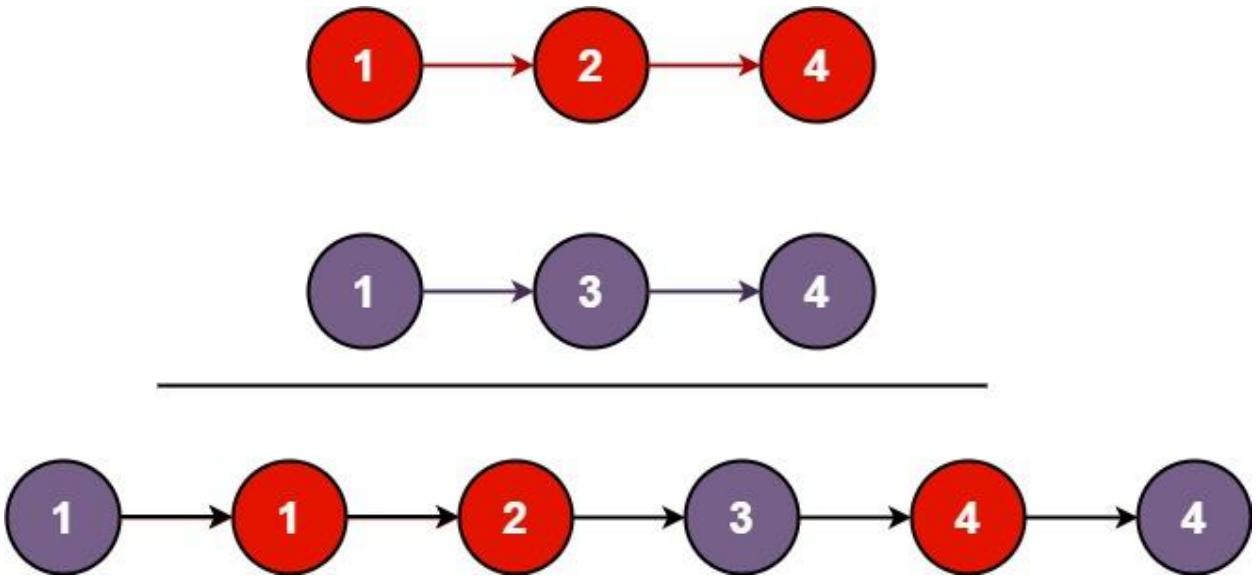
21. Merge Two Sorted Lists

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

Example 1:



Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

Example 2:

Input: list1 = [], list2 = []
Output: []

Example 3:

Input: list1 = [], list2 = [0]
Output: [0]

Constraints:

- The number of nodes in both lists is in the range `[0, 50]`.
- `100 <= Node.val <= 100`
- Both `list1` and `list2` are sorted in **non-decreasing** order.

143. Reorder List

You are given the head of a singly linked-list. The list can be represented as:

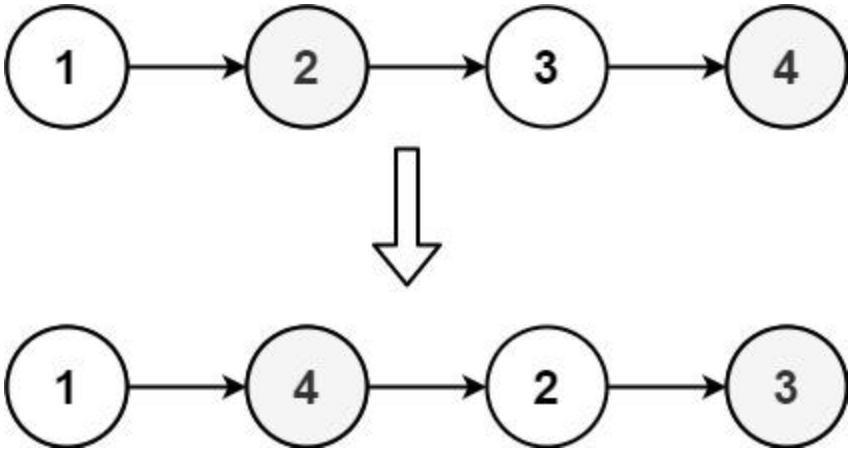
L0 → L1 → … → Ln - 1 → Ln

Reorder the list to be on the following form:

L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …

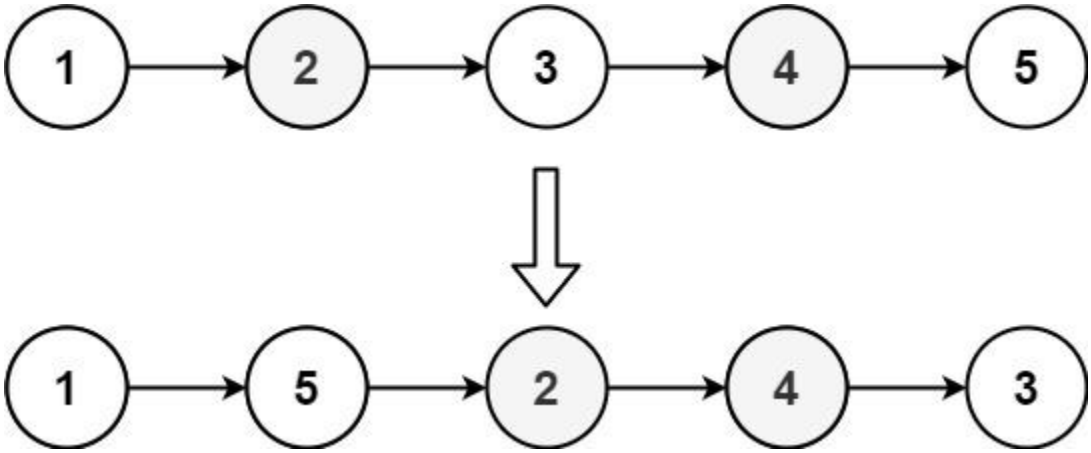
You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:



Input: head = [1,2,3,4]
Output: [1,4,2,3]

Example 2:



Input: head = [1,2,3,4,5]
Output: [1,5,2,4,3]

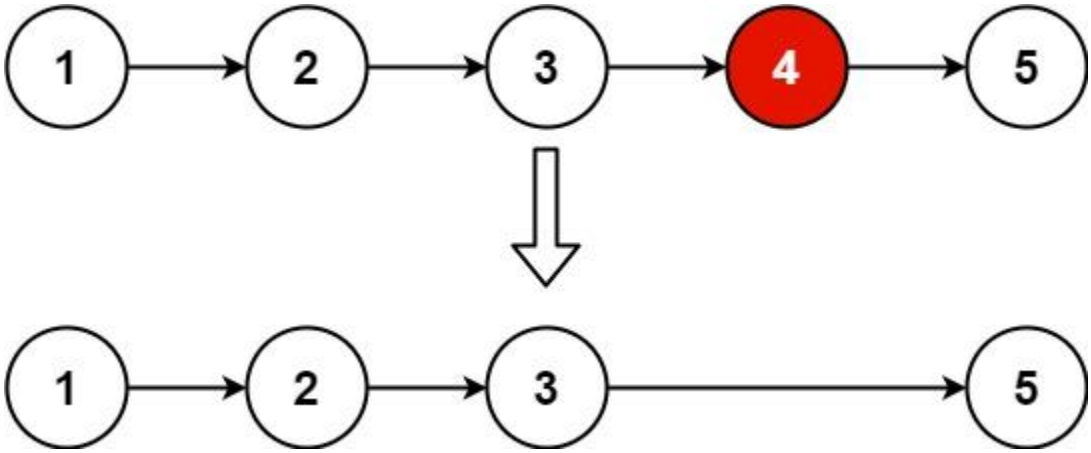
Constraints:

- The number of nodes in the list is in the range `[1, 5 * 104]`.
- `1 <= Node.val <= 1000`

[19. Remove Nth Node From End of List]

Given the `head` of a linked list, remove the `nth` node from the end of the list and return its head.

Example 1:



Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]

Example 2:

Input: head = [1], n = 1
Output: []

Example 3:

Input: head = [1,2], n = 1
Output: [1]

Constraints:

- The number of nodes in the list is `sz`.
- `1 <= sz <= 30`
- `0 <= Node.val <= 100`
- `1 <= n <= sz`

Follow up: Could you do this in one pass?

138. Copy List with Random Pointer

A linked list of length `n` is given such that each node contains an additional random pointer, which could point to any node in the list, or `null`.

Construct a **deep copy** of the list. The deep copy should consist of exactly `n` **brand new** nodes, where each new node has its value set to the value of its corresponding original node. Both the `next` and `random` pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. **None of the pointers in the new list should point to nodes in the original list**.

For example, if there are two nodes `X` and `Y` in the original list, where `X.random --> Y`, then for the corresponding two nodes `x` and `y` in the copied list, `x.random --> y`.

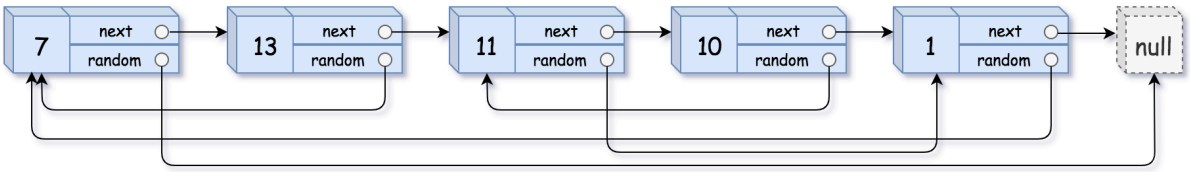
Return *the head of the copied linked list*.

The linked list is represented in the input/output as a list of `n` nodes. Each node is represented as a pair of `[val, random_index]` where:

- `val`: an integer representing `Node.val`
- `random_index`: the index of the node (range from `0` to `n-1`) that the `random` pointer points to, or `null` if it does not point to any node.

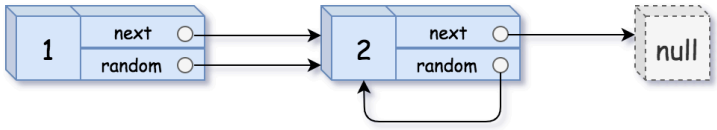
Your code will **only** be given the `head` of the original linked list.

Example 1:



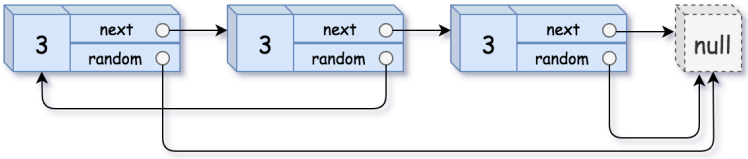
Input: head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
Output: [[7,null],[13,0],[11,4],[10,2],[1,0]]

Example 2:



Input: head = [[1,1],[2,1]]
Output: [[1,1],[2,1]]

Example 3:



Input: head = [[3,null],[3,0],[3,null]]
Output: [[3,null],[3,0],[3,null]]

Constraints:

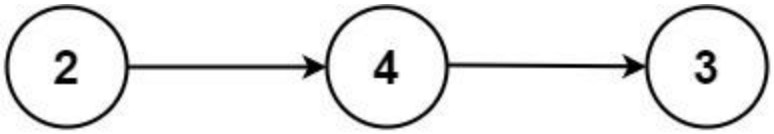
- `0 <= n <= 1000`
- `104 <= Node.val <= 104`
- `Node.random` is `null` or is pointing to some node in the linked list.

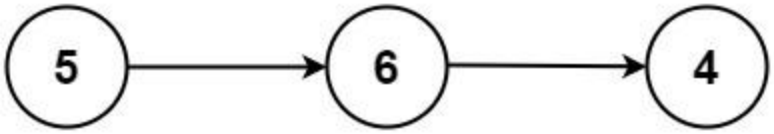
2. Add Two Numbers

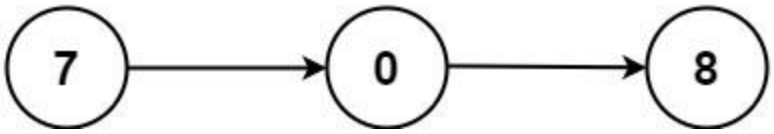
You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:







Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.

Example 2:

Input: l1 = [0], l2 = [0]
Output: [0]

Example 3:

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: [8,9,9,9,0,0,0,1]

Constraints:

- The number of nodes in each linked list is in the range `[1, 100]`.
- `0 <= Node.val <= 9`
- It is guaranteed that the list represents a number that does not have leading zeros.

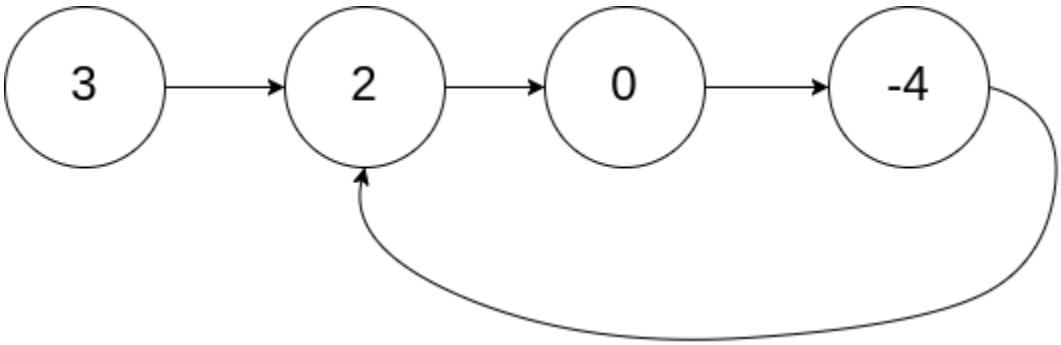
141. Linked List Cycle

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

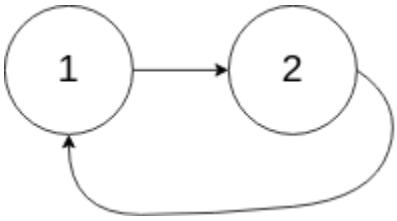
Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:



Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:



Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Example 3:

Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.

Constraints:

- The number of the nodes in the list is in the range `[0, 104]`.
- `105 <= Node.val <= 105`
- `pos` is `1` or a **valid index** in the linked-list.

Follow up: Can you solve it using `O(1)` (i.e. constant) memory?

287. Find the Duplicate Number

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and using only constant extra space.

Example 1:

Input: nums = [1,3,4,2,2]
Output: 2

Example 2:

Input: nums = [3,1,3,4,2]
Output: 3

Example 3:

Input: nums = [3,3,3,3,3]
Output: 3

Constraints:

- `1 <= n <= 105`
- `nums.length == n + 1`
- `1 <= nums[i] <= n`
- All the integers in `nums` appear only **once** except for **precisely one integer** which appears **two or more** times.

Follow up:

- How can we prove that at least one duplicate number must exist in `nums`?
- Can you solve the problem in linear runtime complexit

146. LRU Cache

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Example 1:

Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
LRUCache lRUCache = new LRUCache(2);

lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);   // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);   // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
lRUCache.get(1);   // return -1 (not found)
lRUCache.get(3);   // return 3
lRUCache.get(4);   // return 4

Constraints:

- `1 <= capacity <= 3000`
- `0 <= key <= 104`
- `0 <= value <= 105`
- At most `2 * 105` calls will be made to `get` and `put`

23. Merge k Sorted Lists

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6

Example 2:

Input: lists = []

Output: []

Example 3:

Input: lists = [[]]
Output: []

Constraints:

- `k == lists.length`
- `0 <= k <= 104`
- `0 <= lists[i].length <= 500`
- `104 <= lists[i][j] <= 104`
- `lists[i]` is sorted in **ascending order**.
- The sum of `lists[i].length` will not exceed `104`.

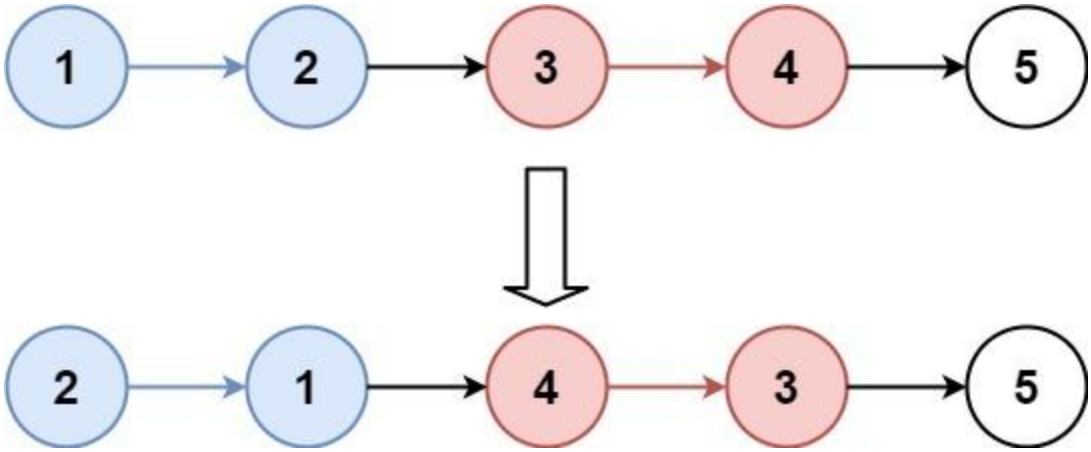
25. Reverse Nodes in k-Group

Given the `head` of a linked list, reverse the nodes of the list `k` at a time, and return *the modified list*.

`k` is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of `k` then left-out nodes, in the end, should remain as it is.

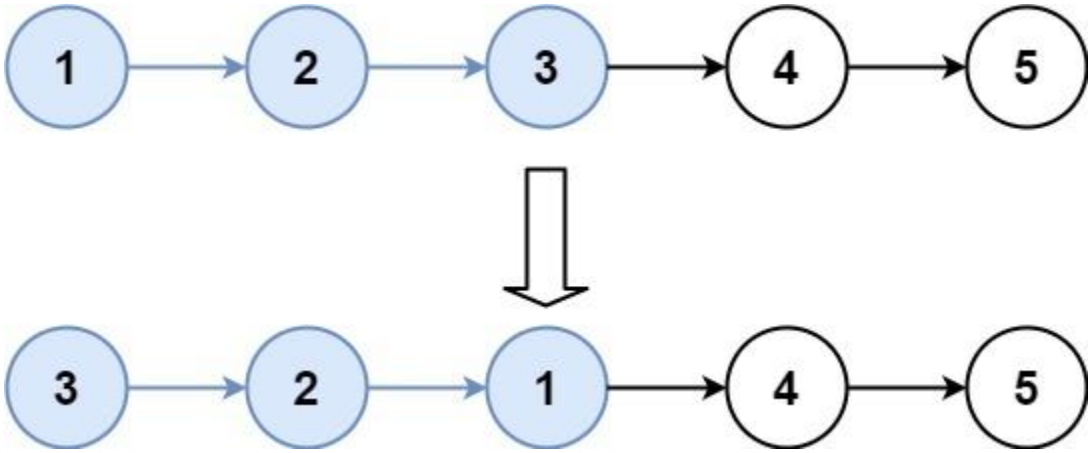
You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:



Input: head = [1,2,3,4,5], k = 2
Output: [2,1,4,3,5]

Example 2:



Input: head = [1,2,3,4,5], k = 3
Output: [3,2,1,4,5]

Constraints:

- The number of nodes in the list is `n`.
- `1 <= k <= n <= 5000`
- `0 <= Node.val <= 1000`

Follow-up: Can you solve the problem in `O(1)` extra memory space?