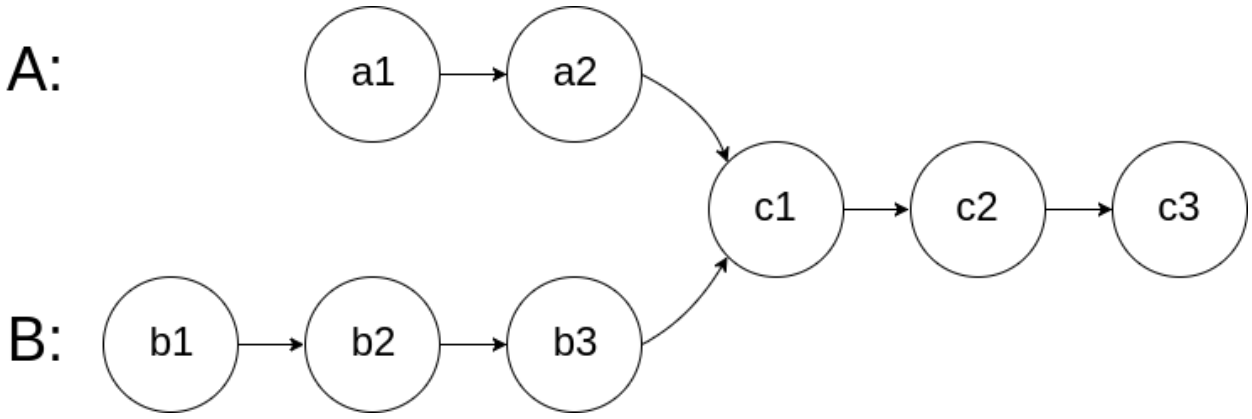
Assignment 3
Linked-List

Total 60 points
Each question 10 marks (8 code , 2 points Time and Space Complexity)

Question 1:

Given the heads of two singly linked-lists `headA` and `headB`, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return `null`.

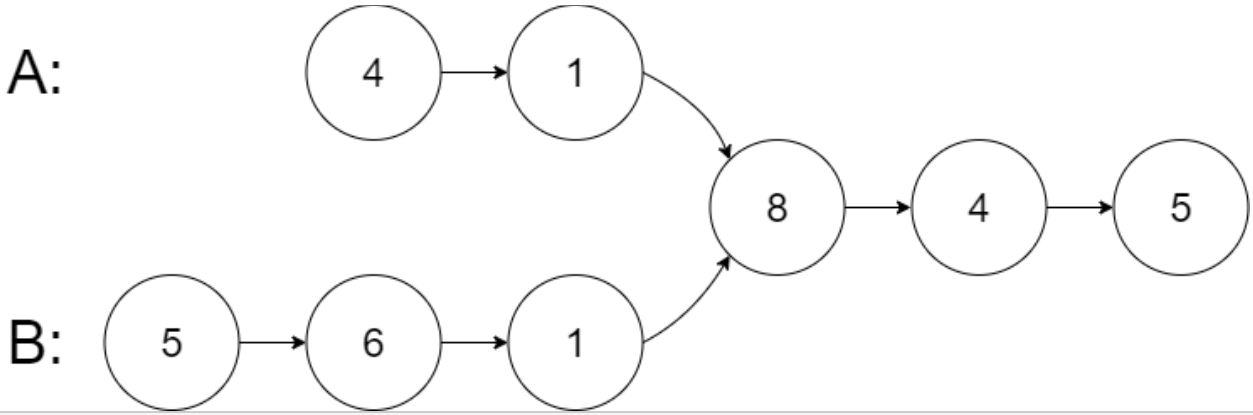
For example, the following two linked lists begin to intersect at node `c1`:



The test cases are generated such that there are no cycles anywhere in the entire linked structure.

Note that the linked lists must **retain their original structure** after the function returns.

Example 1:



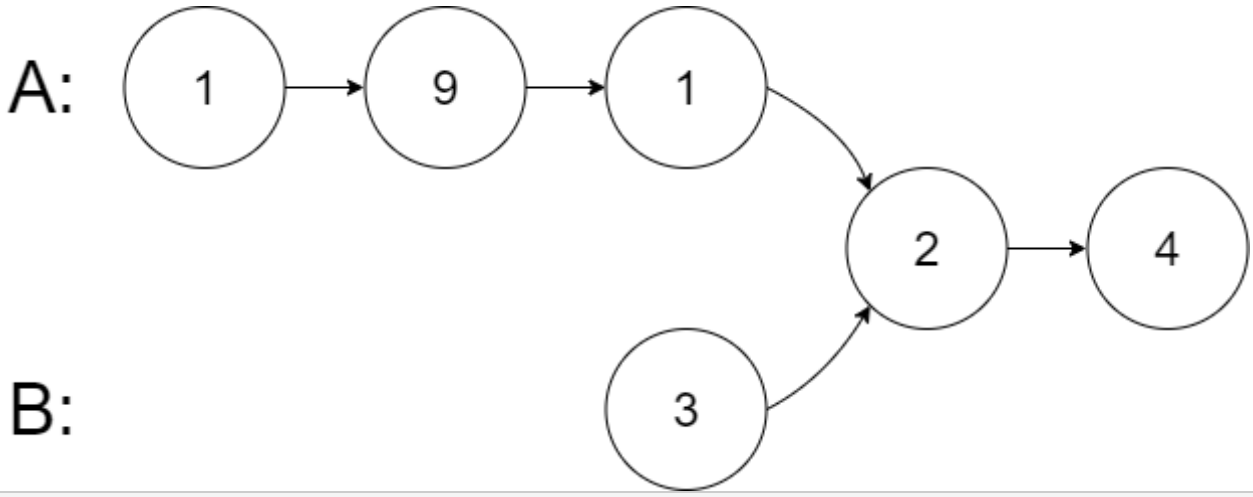
Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3

Output: Intersected at '8'

Explanation: The intersected node's value is 8 (note that this must not be 0 if the two lists intersect).

From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,6,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes befor e the intersected node in B.

Example 2:



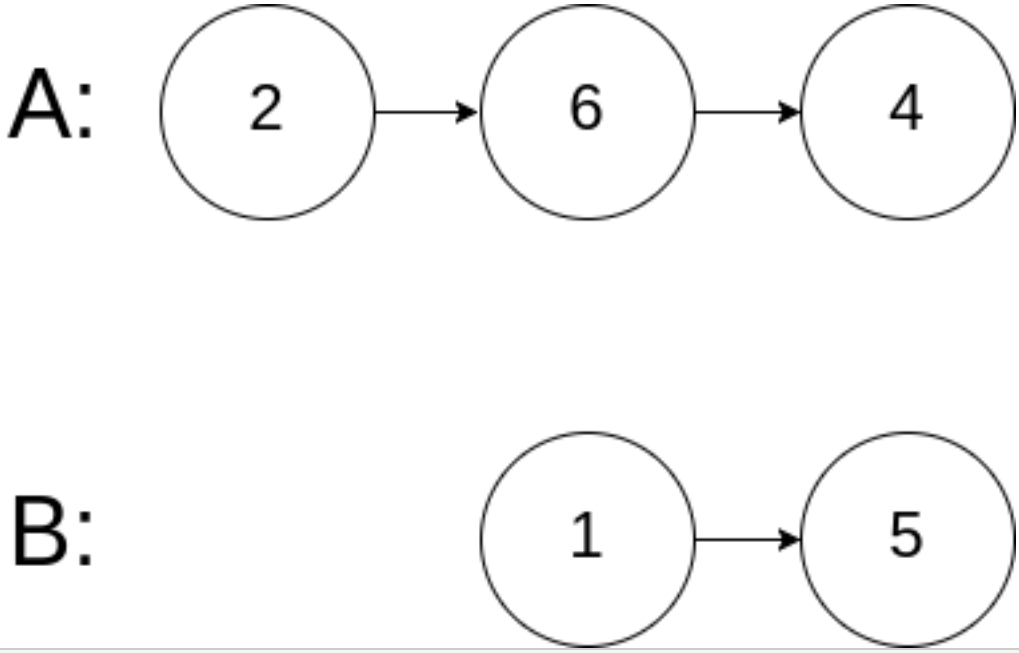
Input: intersectVal = 2, listA = [1,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1

Output: Intersected at '2'

Explanation: The intersected node's value is 2 (note that this must not be 0 if the two lists intersect).

From the head of A, it reads as [1,9,1,2,4]. From the head of B, it reads as [3,2,4]. There are 3 nodes before the intersected node in A; There are 1 node before the i ntersected node in B.

Example 3:



Input: intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2

Output: No intersection

Explanation: From the head of A, it reads as [2,6,4]. From the head of B, it reads a s [1,5]. Since the two lists do not intersect, intersectVal must be 0, while skipA and skipB can be arbitrary values.

Explanation: The two lists do not intersect, so return null.

Constraints:

- The number of nodes of listA is in the m.
- The number of nodes of listB is in the n.
- $0 <= m, n <= 3 * 10^4$
- $1 <= Node.val <= 10^5$

- 0 <= skipA <= m
- 0 <= skipB <= n
- intersectVal is 0 if listA and listB do not intersect.
- intersectVal == listA[skipA] == listB[skipB] if listA and listB intersect.
-

Question2 :

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.

Input: head = [1,2,6,3,4,5,6], val = 6

Output: [1,2,3,4,5]

Example 2:

Input: head = [], val = 1

Output: []

Example 3:

Input: head = [7,7,7,7], val = 7

Output: []

Question 3:

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.

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]

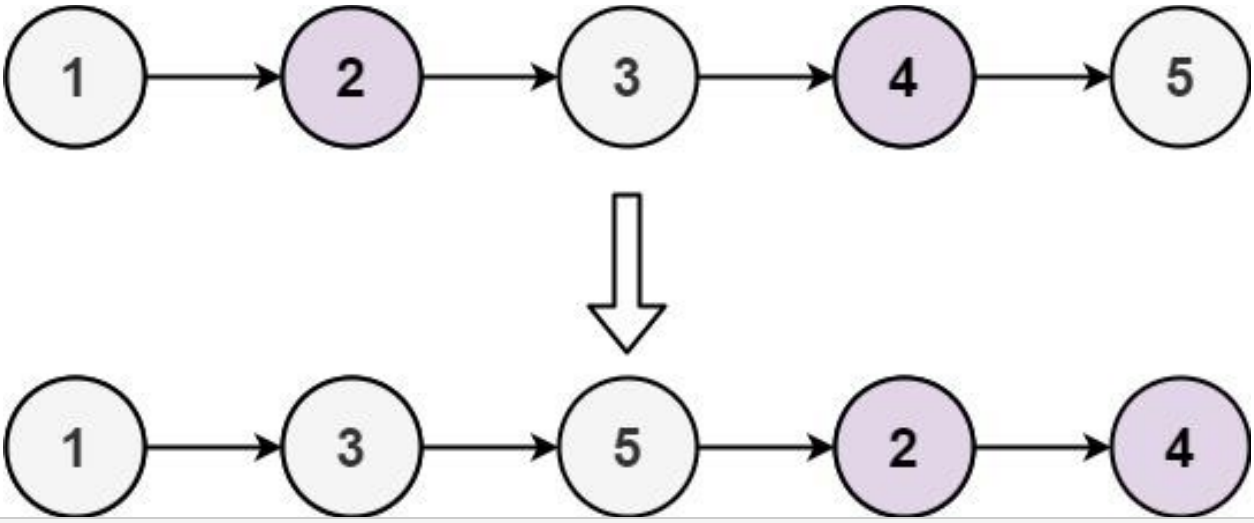
Question 4:

Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

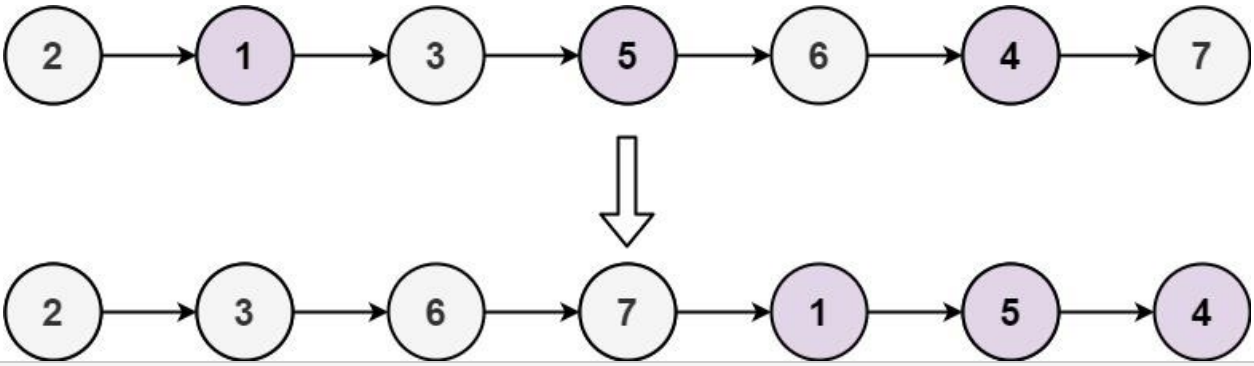
Example 1:



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

Output: [1,3,5,2,4]

Example 2:



Input: head = [2,1,3,5,6,4,7]

Output: [2,3,6,7,1,5,4]

Constraints:

- n == number of nodes in the linked list
- $0 <= n <= 10^4$
- $-10^6 <= Node.val <= 10^6$

Question 5: Given the head of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle** node.

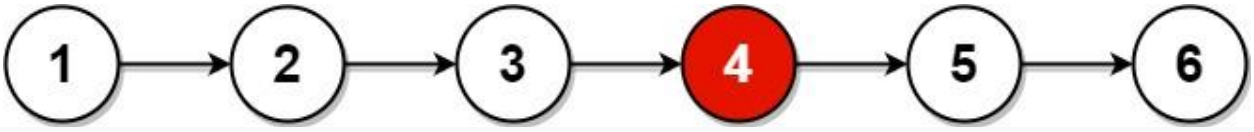
Example 1:

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

Output: [3,4,5]

Explanation: The middle node of the list is node 3.

Example 2:



Input: head = [1,2,3,4,5,6]

Output: [4,5,6]

Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

Constraints:

- The number of nodes in the list is in the range `[1, 100]`.
- `1 <= Node.val <= 100`

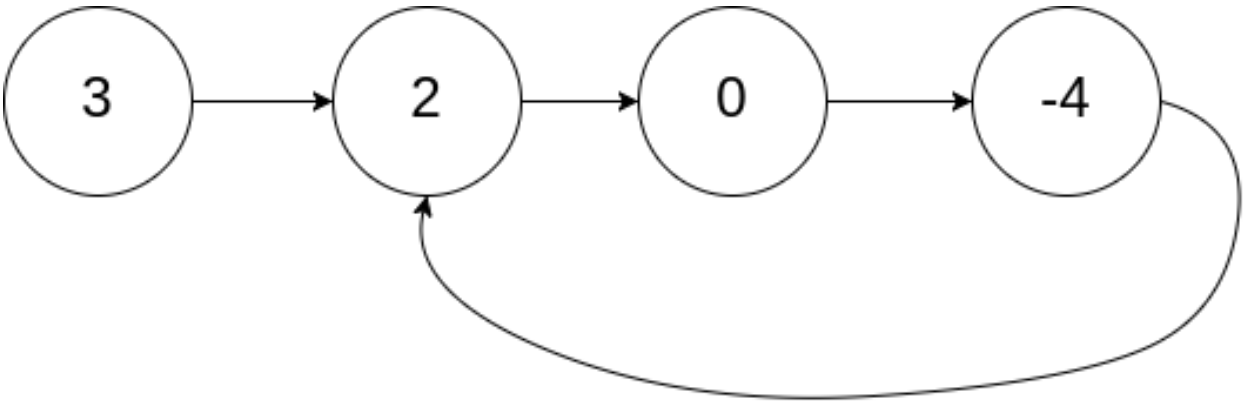
Question 6:

Given the `head` of a linked list, return *the node where the cycle begins. If there is no cycle, return* `null`.

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 (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter**.

Do not modify the linked list.

Example 1:

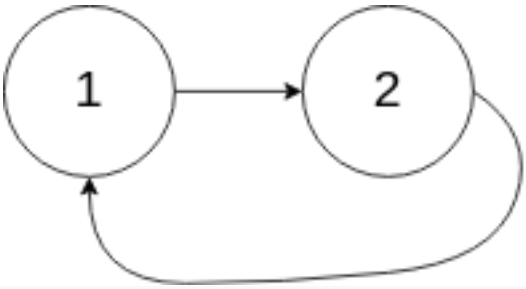


Input: head = [3,2,0,-4], pos = 1

Output: tail connects to node index 1

Explanation: There is a cycle in the linked list, where tail connects to the second node.

Example 2:



Input: head = [1,2], pos = 0

Output: tail connects to node index 0

Explanation: There is a cycle in the linked list, where tail connects to the first node.

Example 3:

Input: head = [1], pos = -1

Output: no cycle

Explanation: There is no cycle in the linked list.

Constraints:

- The number of the nodes in the list is in the range $[0, 10^4]$.
- $-10^5 <= Node.val <= 10^5$
- pos is -1 or a **valid index** in the linked-list.