**Algorithm to Concatenate Two Singly Linked Lists L and M:**

The goal is to concatenate two singly linked lists L and M into a new list L' such that L' contains all the nodes of L followed by all the nodes of M.

**Steps:**

1. **Check if either list is empty**:
   * If L is empty (i.e., L.head == null), then L' should be a copy of M (i.e., L' = M).
   * If M is empty (i.e., M.head == null), then L' should be a copy of L (i.e., L' = L).
2. **Traverse the list L to find its last node**:
   * Start from L.head and traverse the list until you reach the last node (i.e., the node where next == null).
3. **Link the last node of L to the first node of M**:
   * Once the last node of L is found, update its next pointer to point to the first node of M.
4. **Update the head of the concatenated list**:
   * The head of the concatenated list L' should still be the head of L, as L comes first in L'.

**Pseudocode:**

Concatenate(L, M):

if L is empty:

L' = M

return L'

if M is empty:

L' = L

return L'

current = L.head

while current.next is not null:

current = current.next

current.next = M.head

L' = L # The head of L is the head of the concatenated list

return L'

**Explanation:**

1. **Handling Empty Lists**:
   * If either L or M is empty, the concatenated list is simply the non-empty list, as there's nothing to concatenate.
2. **Traversing L**:
   * We start from the head of L and use a current pointer to traverse through each node until we reach the last node (i.e., current.next == null).
3. **Connecting L and M**:
   * Once the last node of L is found, we link its next pointer to the head of list M (current.next = M.head), effectively appending M to the end of L.
4. **Returning the Concatenated List**:
   * Since L already represents the concatenated list starting from its head, we simply return L.

**Time Complexity:**

* The algorithm requires traversing the entire list L to find its last node, which takes O(n)O(n)O(n), where nnn is the number of nodes in L.
* Therefore, the overall time complexity of this algorithm is O(n)O(n)O(n), where nnn is the number of nodes in L.

**Example:**

Let’s assume we have the following two lists:

* L: 1 -> 2 -> 3
* M: 4 -> 5 -> 6

After concatenating them, the new list L' will be:

* L': 1 -> 2 -> 3 -> 4 -> 5 -> 6