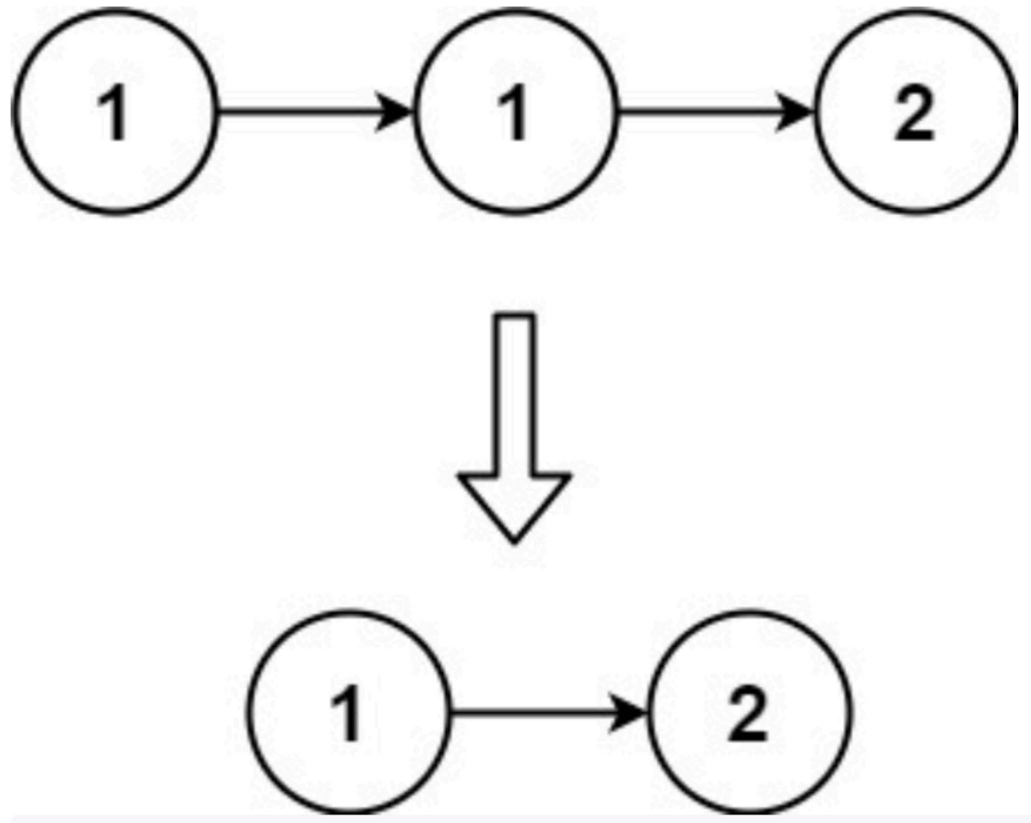
83. Remove Duplicates from Sorted List

Easy 4825 187 Add to List Share

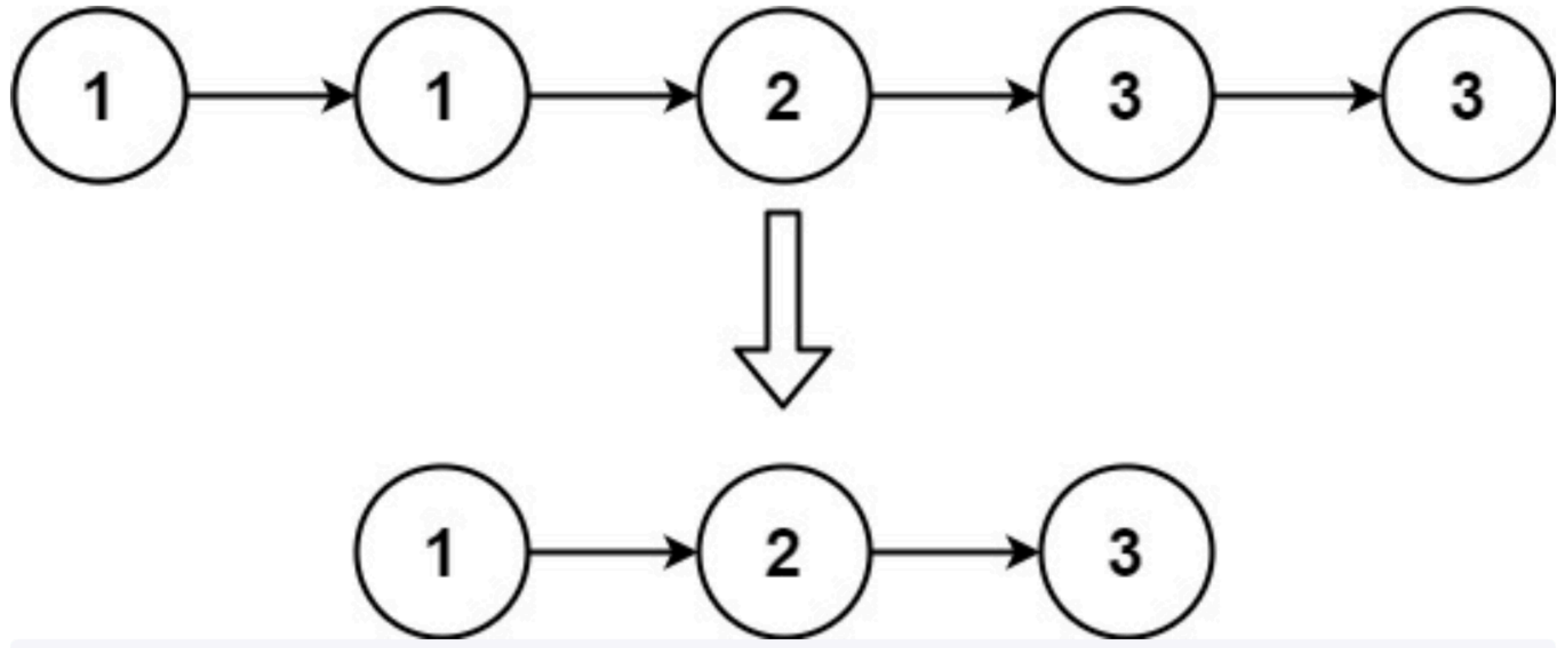
Given the `head` of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list **sorted** as well*.

Example 1:



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

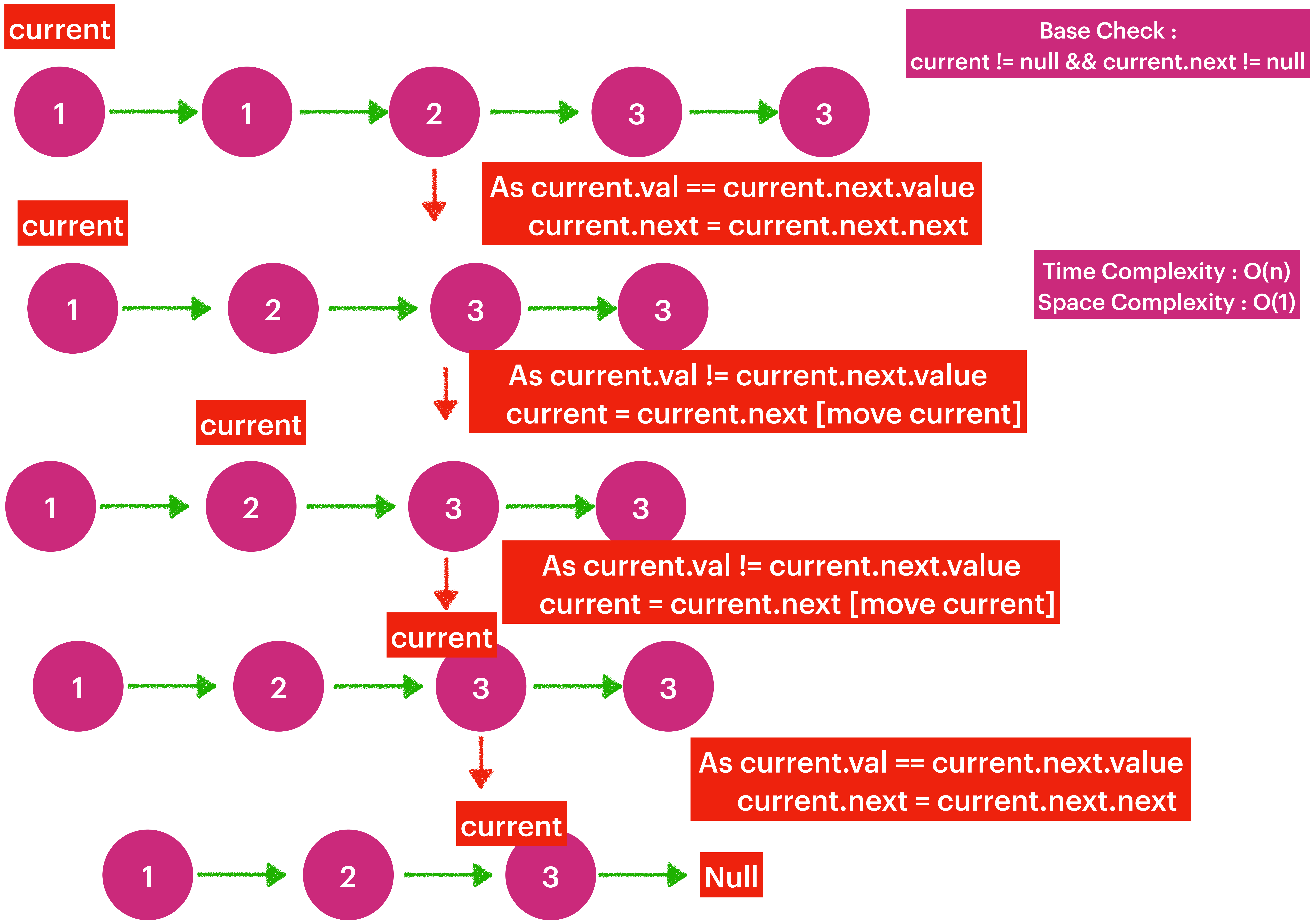
Example 2:



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

Constraints:

- The number of nodes in the list is in the range `[0, 300]`.
- `-100 <= Node.val <= 100`
- The list is guaranteed to be **sorted** in ascending order.



Base Check :
current != null && current.next != null

As current.val == current.next.value
current.next = current.next.next

Time Complexity : O(n)
Space Complexity : O(1)

As current.val != current.next.value
current = current.next [move current]

As current.val != current.next.value
current = current.next [move current]

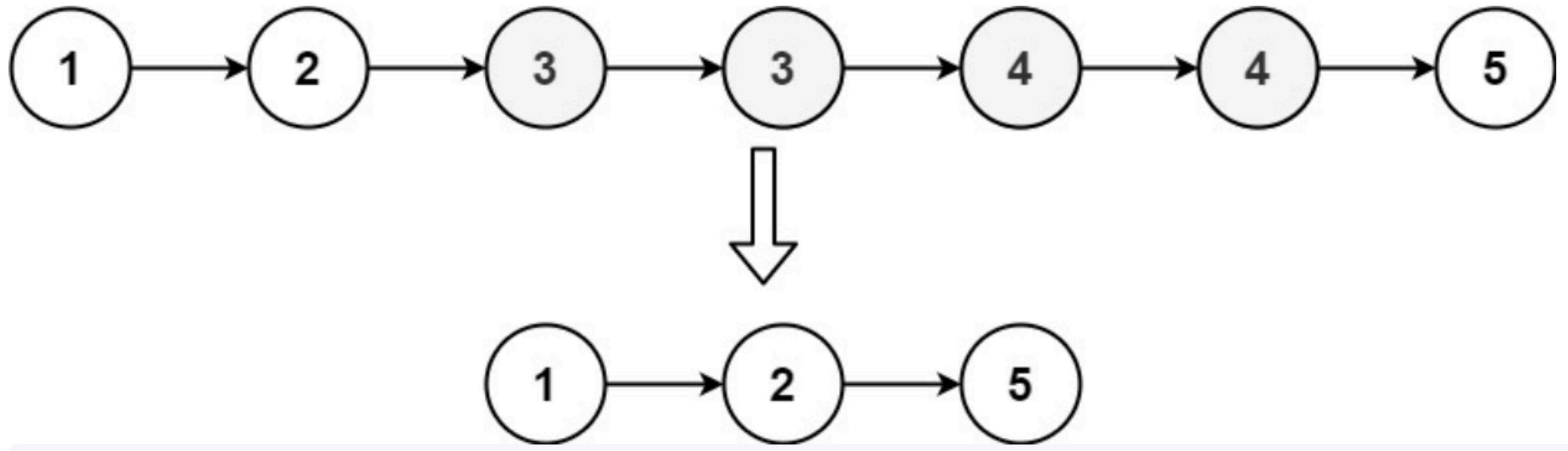
As current.val == current.next.value
current.next = current.next.next

82. Remove Duplicates from Sorted List II

Medium 5743 167 Add to List Share

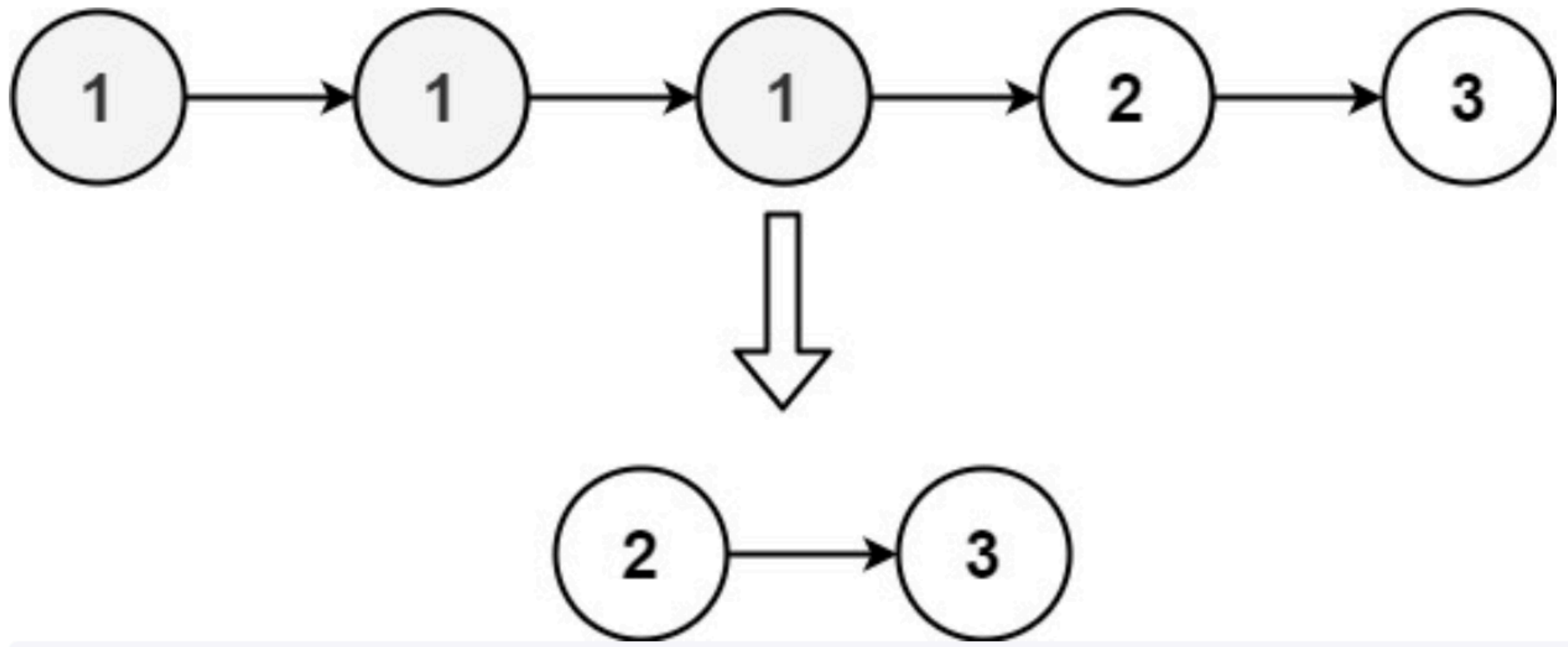
Given the `head` of a sorted linked list, *delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list*. Return *the linked list* ***sorted*** *as well*.

Example 1:



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

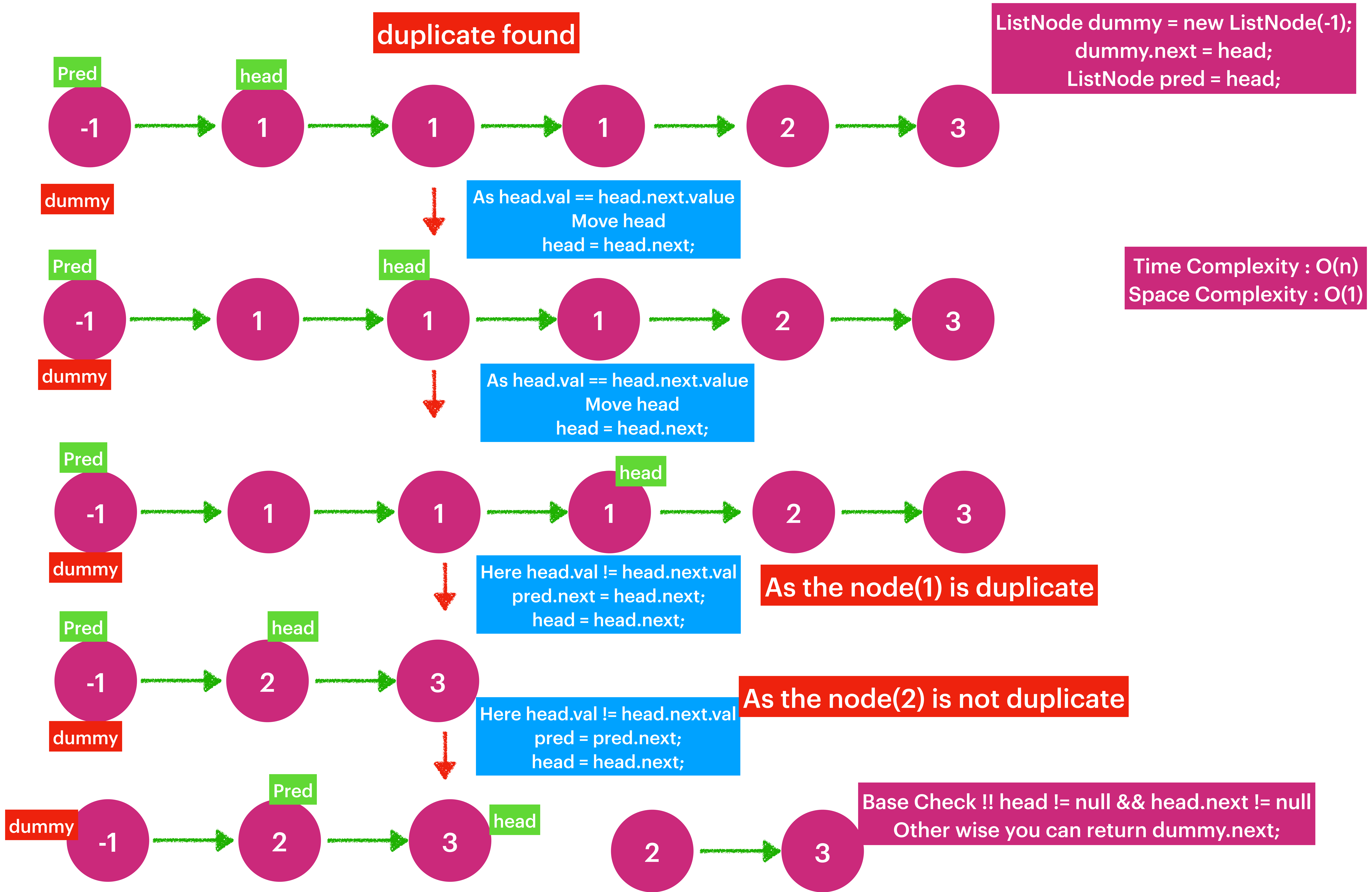
Example 2:



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

Constraints:

- The number of nodes in the list is in the range `[0, 300]`.
- `-100 <= Node.val <= 100`
- The list is guaranteed to be **sorted** in ascending order.



duplicate found

ListNode dummy = new ListNode(-1);
dummy.next = head;
ListNode pred = head;

As head.val == head.next.value
Move head
head = head.next;

Time Complexity : O(n)
Space Complexity : O(1)

As head.val == head.next.value
Move head
head = head.next;

Here head.val != head.next.val
pred.next = head.next;
head = head.next;

As the node(1) is duplicate

Here head.val != head.next.val
pred = pred.next;
head = head.next;

As the node(2) is not duplicate

Base Check !! head != null && head.next != null
Other wise you can return dummy.next;

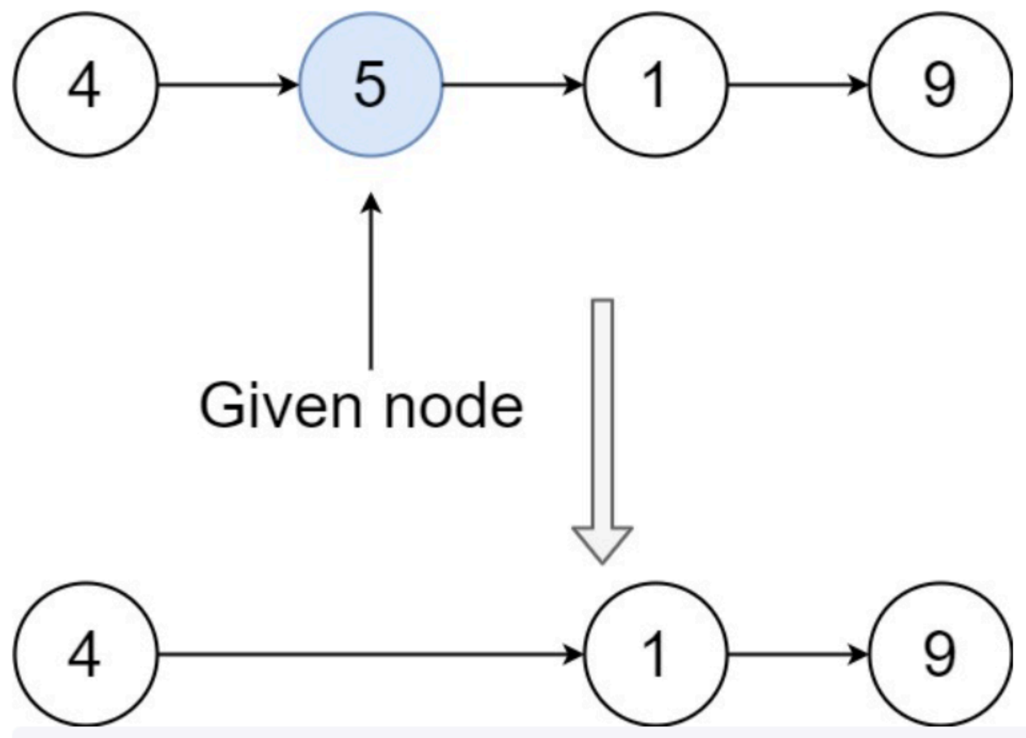
237. Delete Node in a Linked List

Easy 4478 11724 Add to List Share

Write a function to **delete a node** in a singly-linked list. You will **not** be given access to the `head` of the list, instead you will be given access to **the node to be deleted** directly.

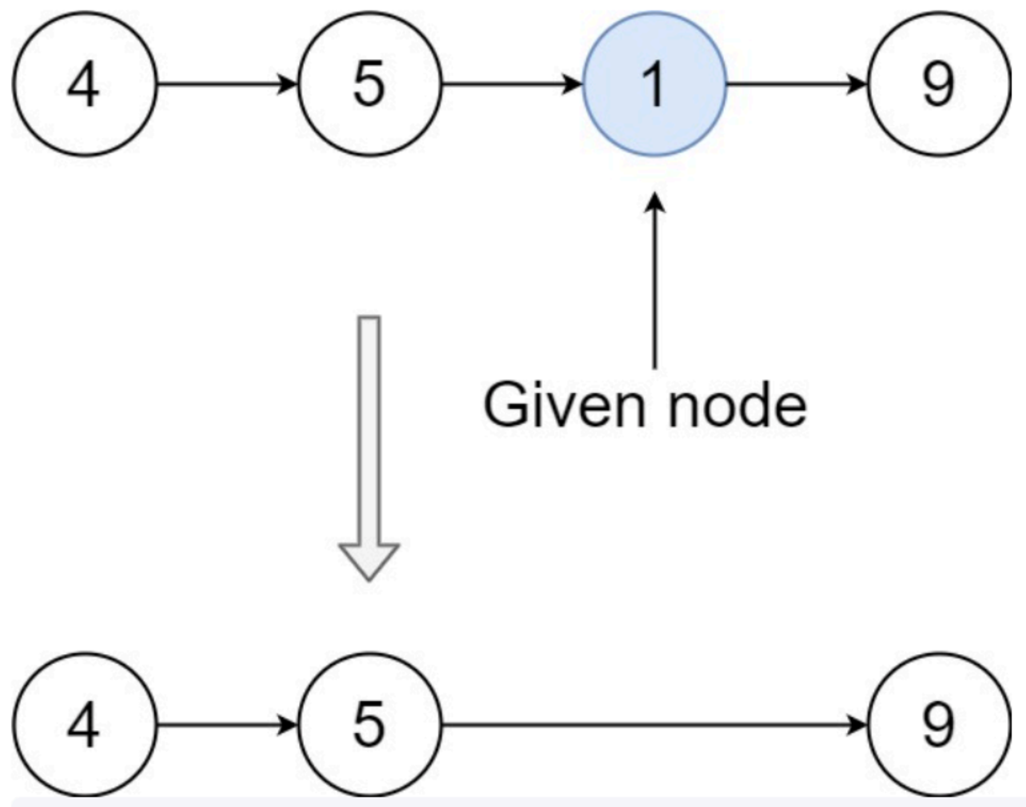
It is **guaranteed** that the node to be deleted is **not a tail node** in the list.

Example 1:



```
Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the
linked list should become 4 -> 1 -> 9 after calling your function.
```

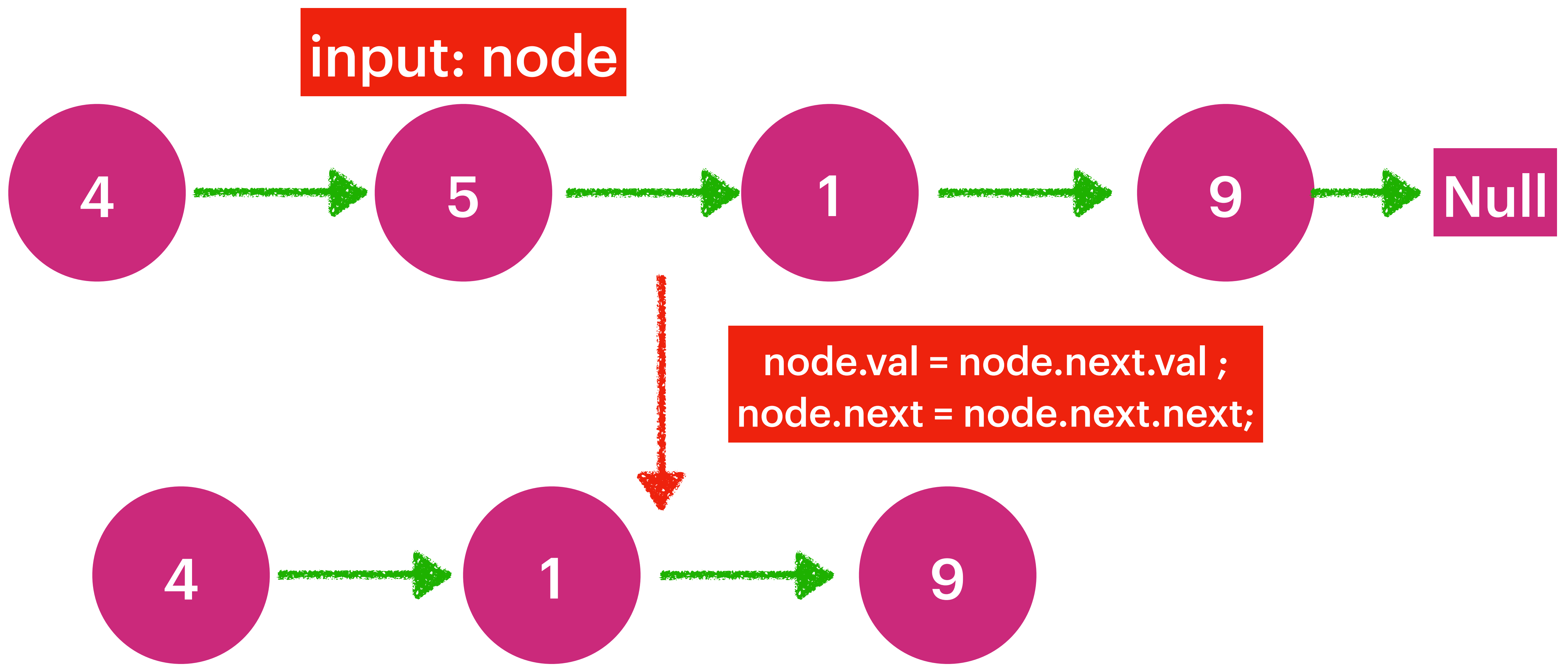
Example 2:



```
Input: head = [4,5,1,9], node = 1
Output: [4,5,9]
Explanation: You are given the third node with value 1, the linked
list should become 4 -> 5 -> 9 after calling your function.
```

Constraints:

- The number of the nodes in the given list is in the range `[2, 1000]`.
- `-1000 <= Node.val <= 1000`
- The value of each node in the list is **unique**.
- The `node` to be deleted is **in the list** and is **not a tail** node



node.val = node.next.val ;
node.next = node.next.next;

Delete node(5)

Input guarantees the gives node(5)

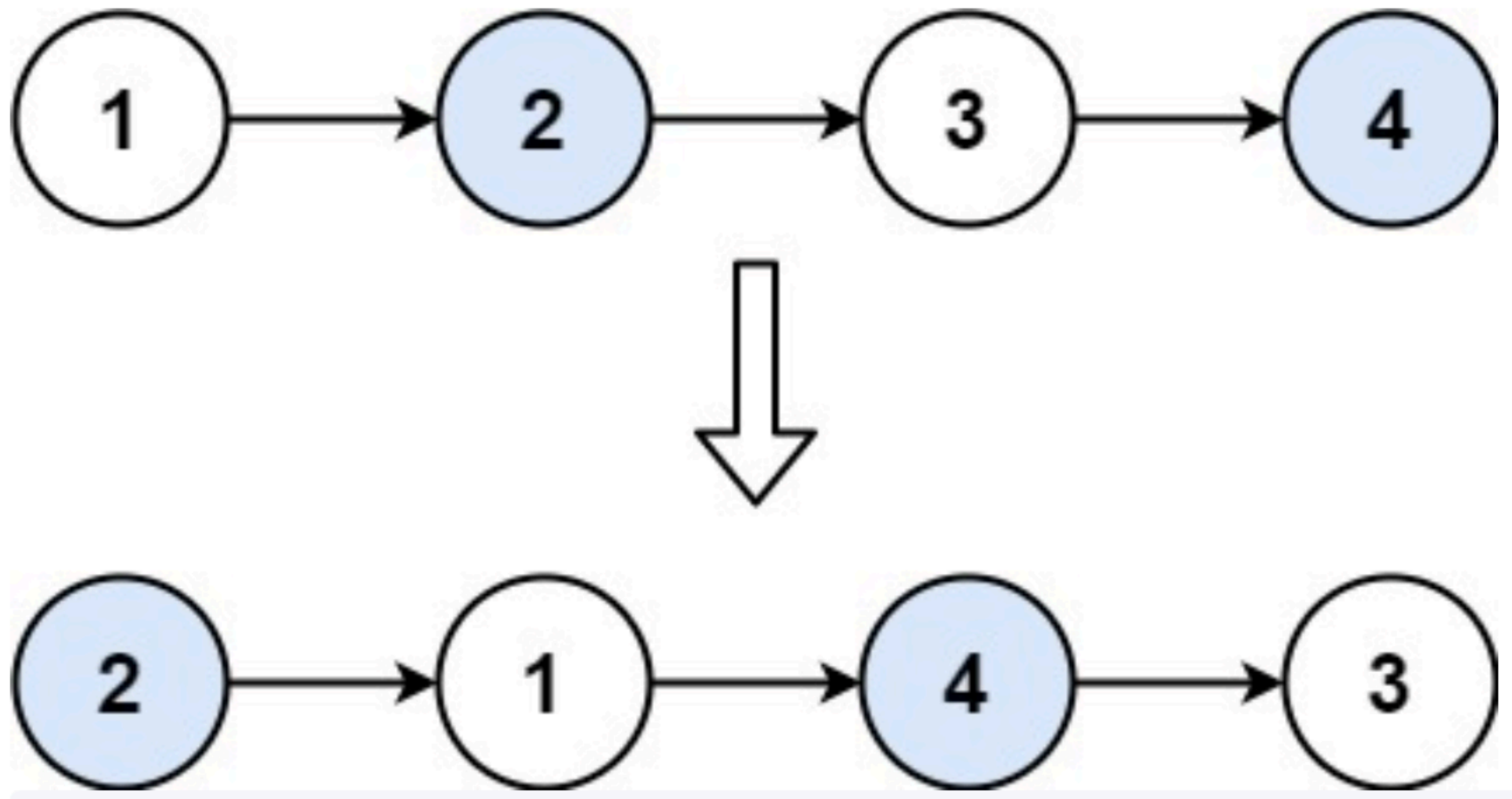
Time Complexity : O(1)
Space Complexity : O(1)

24. Swap Nodes in Pairs

Medium 6968 294 Add to List Share

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

Example 1:



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

Example 2:

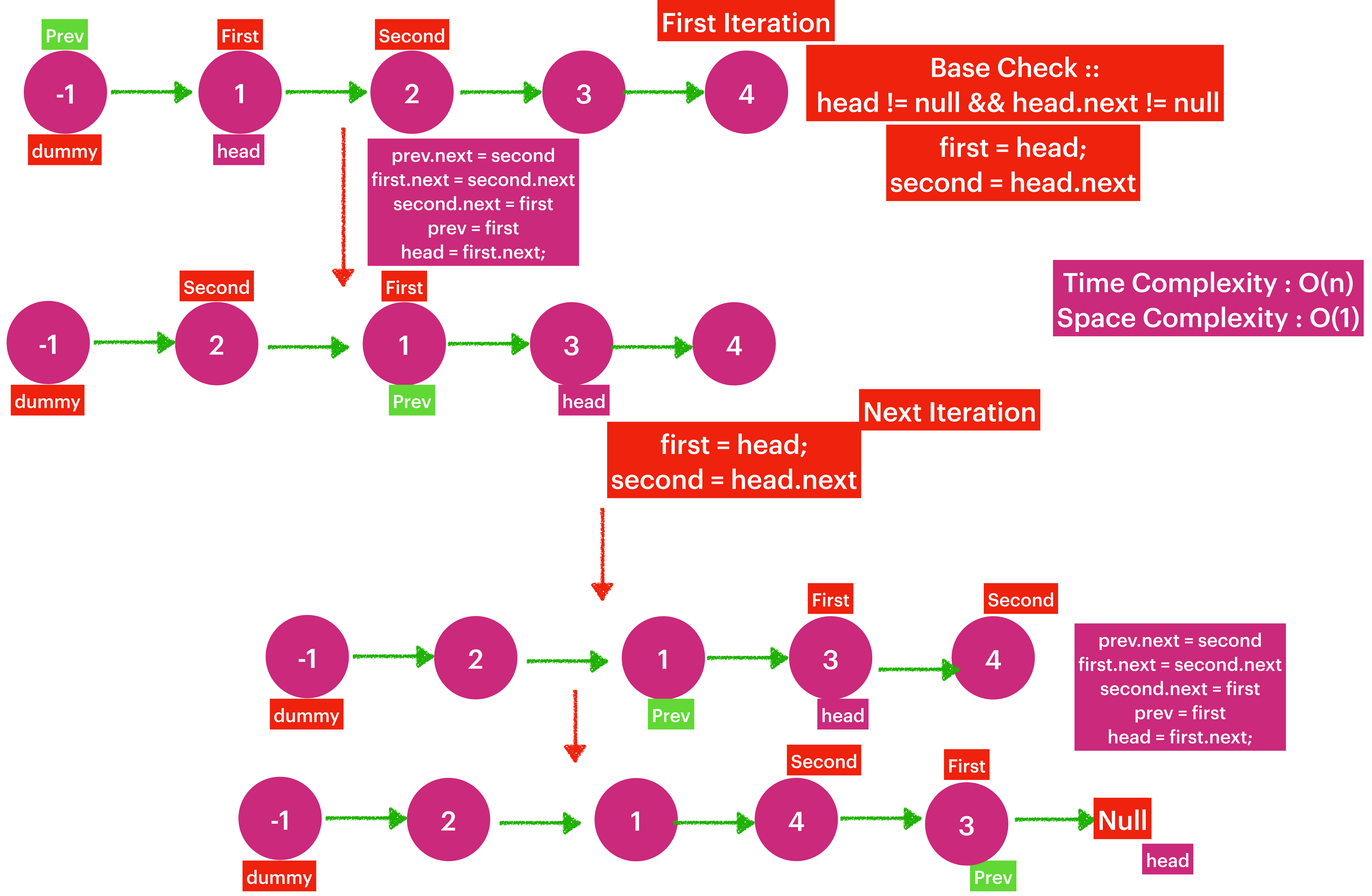
```
Input: head = []
Output: []
```

Example 3:

```
Input: head = [1]
Output: [1]
```

Constraints:

- The number of nodes in the list is in the range `[0, 100]`.
- `0 <= Node.val <= 100`



First Iteration

Base Check ::
head != null && head.next != null

```
first = head;
second = head.next
```

```
prev.next = second
first.next = second.next
second.next = first
prev = first
head = first.next;
```

Time Complexity : O(n)
Space Complexity : O(1)

Next Iteration

```
first = head;
second = head.next
```

```
prev.next = second
first.next = second.next
second.next = first
prev = first
head = first.next;
```