Statement

↳ Given the head of a singly linked list and an integer n, remove the n^{th} node from the end of the list and return the head of the modified list.

Naive

↳ Requires two traversals of the linked list.

- First traversal counts the total nodes (N) in the linked list to identify the position of the node to be removed ($N-n+1$).
- Second traversal stop at the $(N-n)^{th}$ node and remove its next node. Then make the next pointer of the $(N-n)^{th}$ node point to the next node of the $(N-n+1)^{th}$ node to skip the target node.

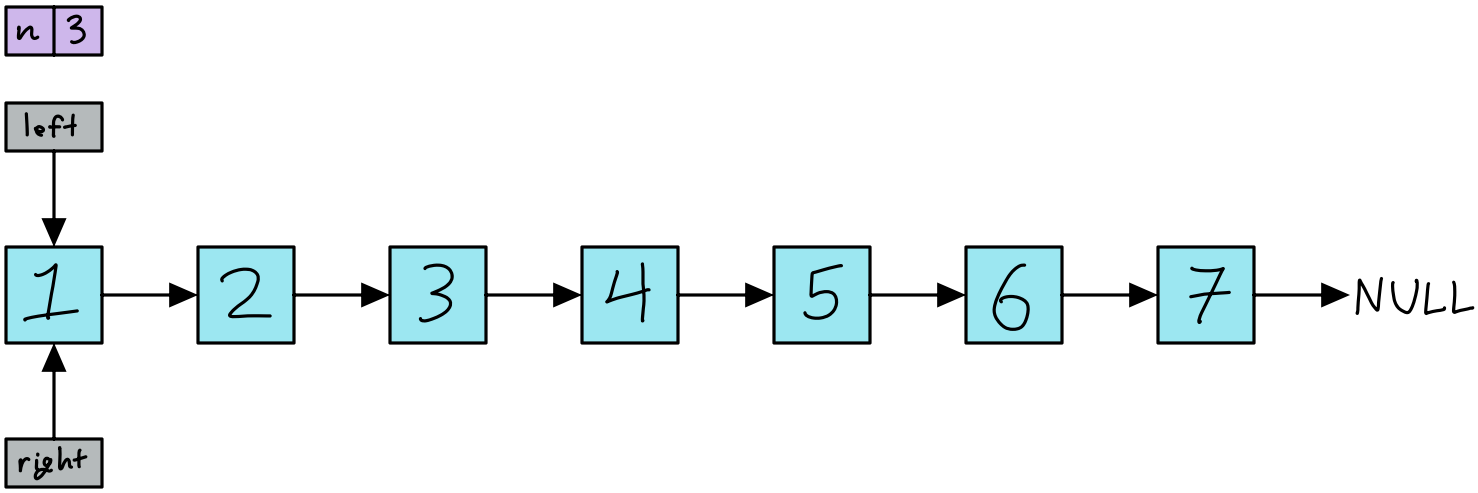
Optimized

↳ Only one traversal of linked list required.

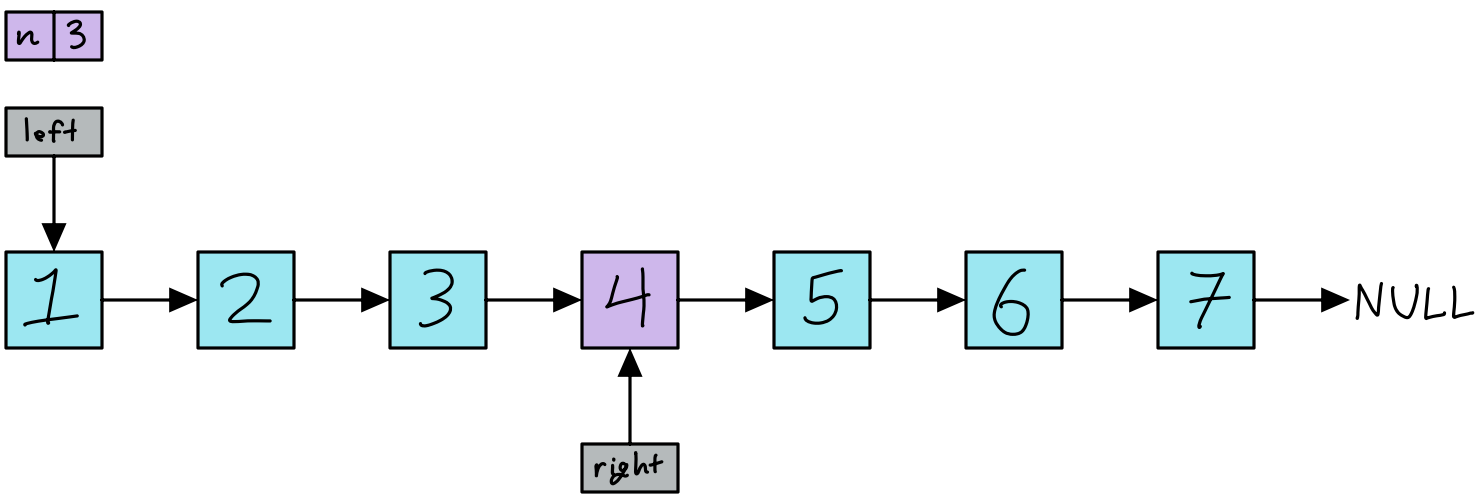
- Initialize left and right pointers
- Move the right pointer n steps forward
- If the right pointer has reached the end of the list, then head is the target node for removal.
  - Return head.next as the new head of the linked list.
- Otherwise, move left and right pointers forward one step at a time.
  - When right pointer reaches the end of the linked list, update left.next to left.next.next.
- Return the head, pointing to the updated linked list with n^{th} node removed.

Visualization

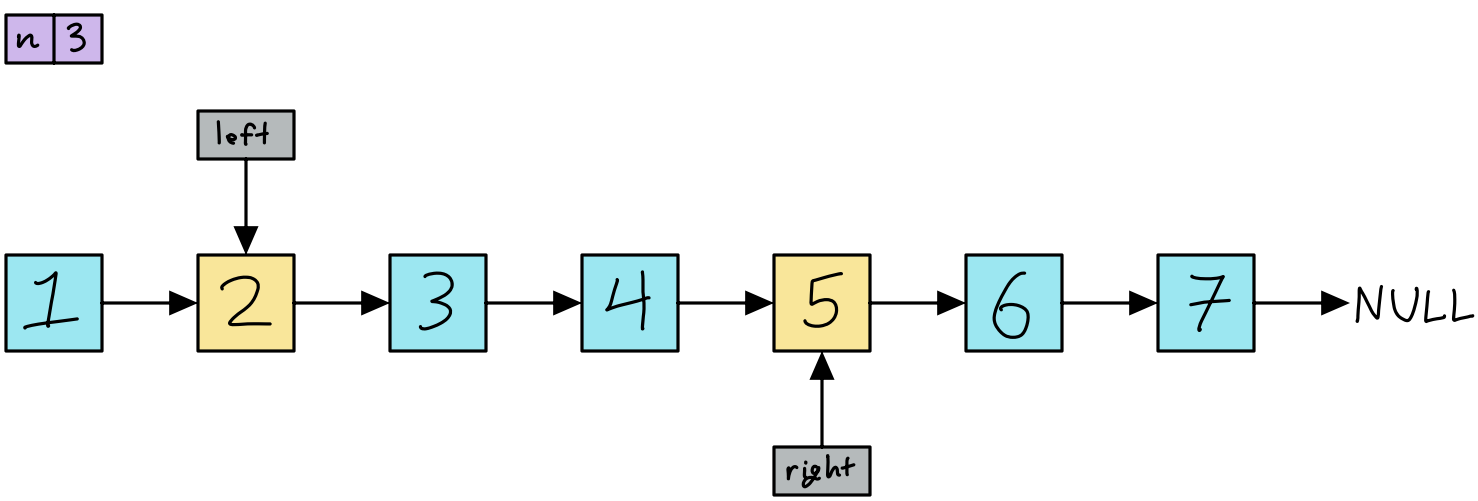
i) Initially, both right and left pointers point to the head node of the linked list.



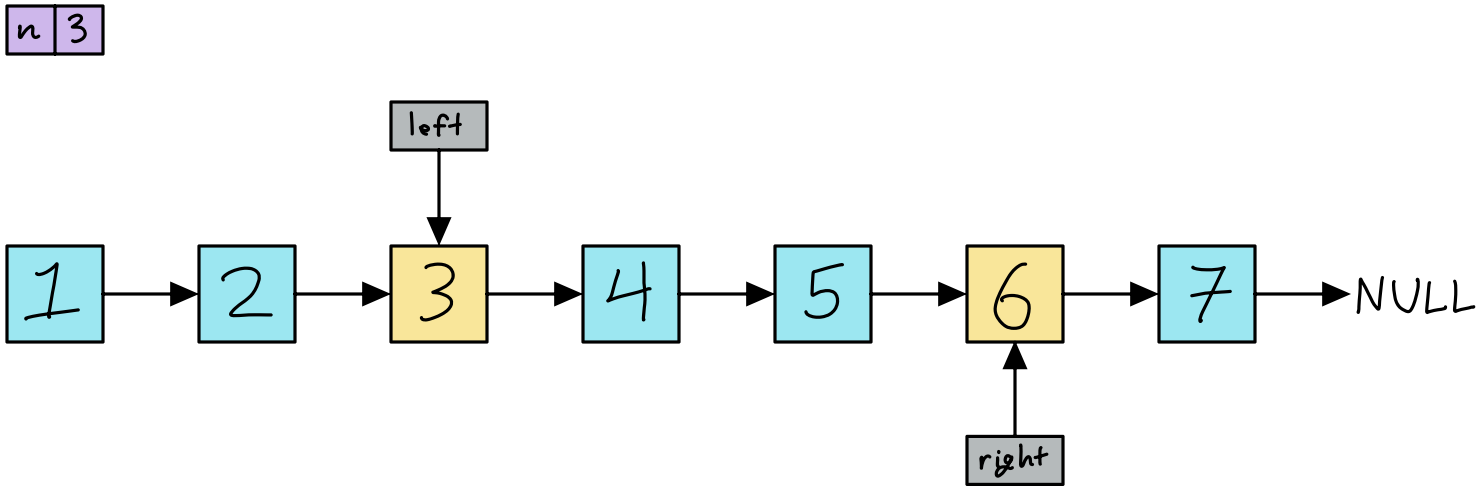
ii) Move the pointer n steps forward from the beginning



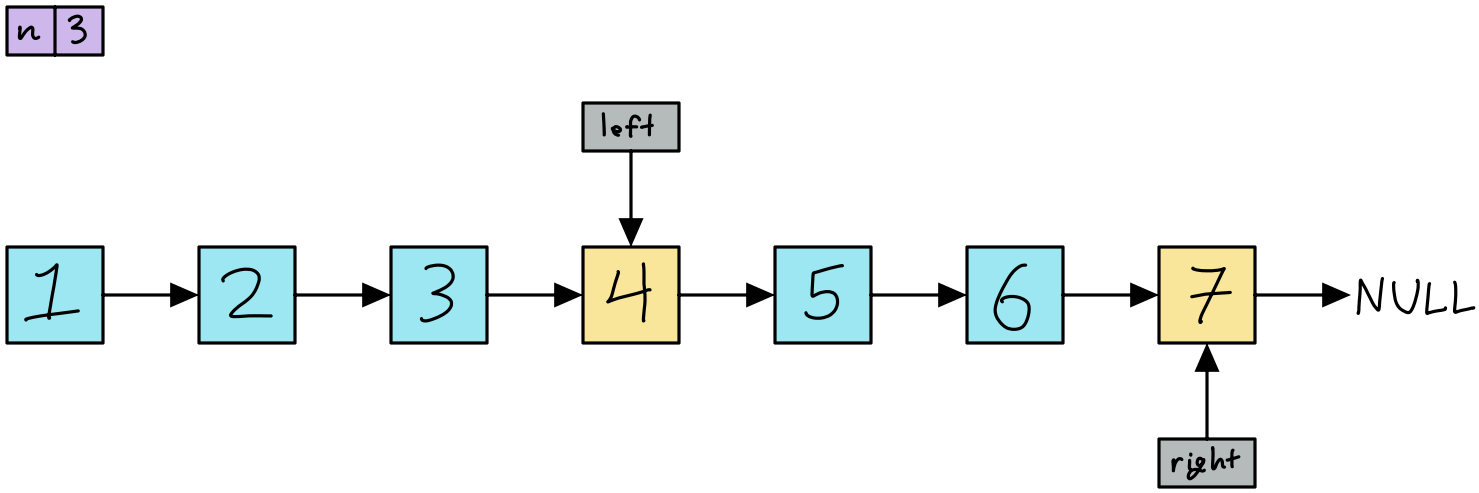
iii) Move the right and left pointers one step forward.



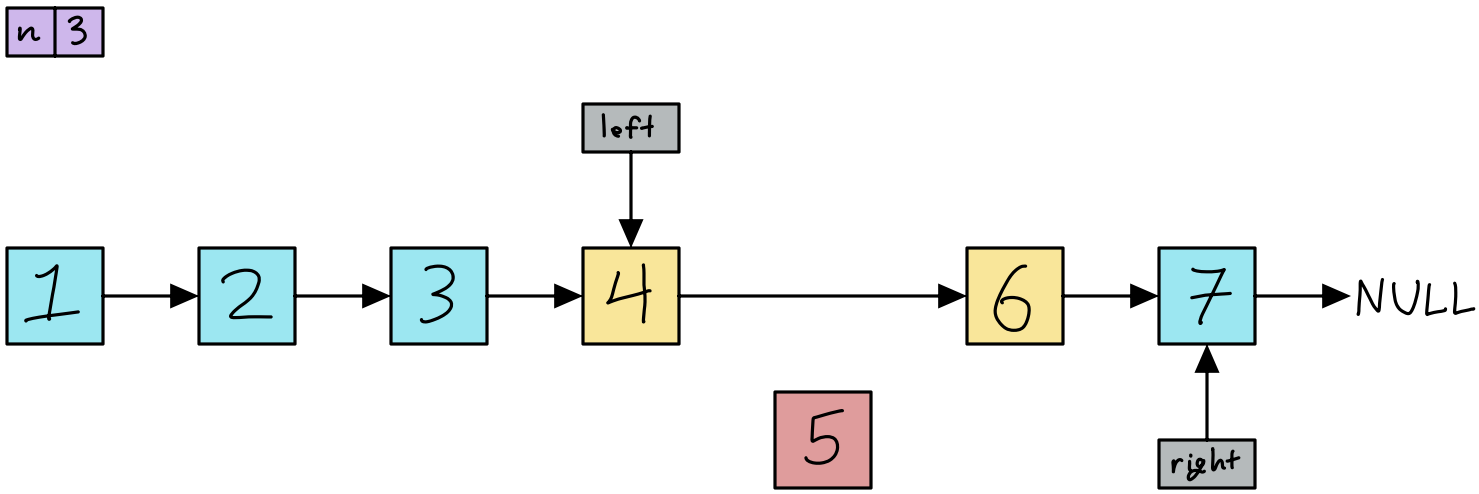
iv) Move the right and left pointers one step forward.



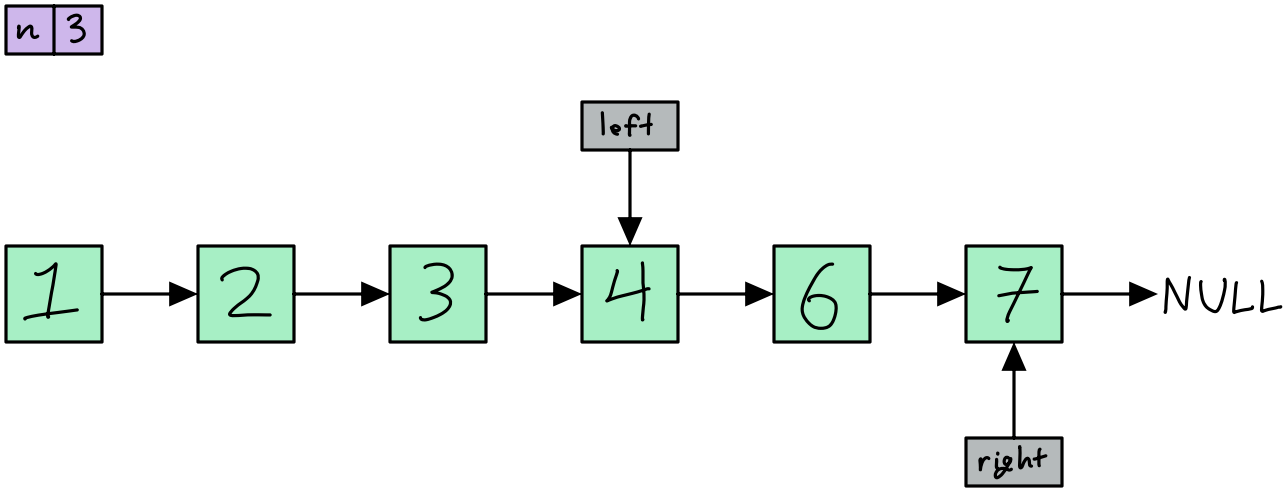
v) Move the right and left pointers one step forward. The left pointer has reached the node. The right pointer has reached one node before the n^{th} last node.



vi) Relink the left node to the node next to left's next node.



vii) The 3rd last node (5) has been removed from the linked list.



Code

```
ListNode* RemoveNthLastNode(ListNode *head, int n) {
    ListNode* right = head;
    ListNode* left = head;

    for (int i = 0; i < n; i++) {
        right = right->next;
    }

    if (!right) {
        return head->next;
    }

    while (right->next) {
        right = right->next;
        left = left->next;
    }

    left = left->next->next;

    return head;
}
```

Time Complexity

↳ The time complexity is $O(N)$, where N is the number of nodes in the linked list.

Space Complexity

↳ The space complexity is $O(1)$ because we use constant space to store two pointers.