Rearrange a given linked list in-place

Given a **singly linked list** L0 -> L1 -> … -> Ln-1 -> Ln. Rearrange the nodes in the list so that the newly formed list is : L0 -> Ln -> L1 -> Ln-1 -> L2 -> Ln-2 … You are required to do this in place without altering the nodes' values.

Examples:

Input*: 1 -> 2 -> 3 -> 4*
Output*: 1 -> 4 -> 2 -> 3*
Explanation: *Here n = 4, so the correct order is L0->L3->L1->L2*

Input: 1 -> 2 -> 3 -> 4 -> 5
Output: 1 -> 5 -> 2 -> 4 -> 3
Explanation: *Here n = 4, so the correct order is L0->L4->L1->L3->L2*

[Efficient Approach] By Reversing Second Half – O(n) Time and O(1) Space

- *Find the middle of the linked list using the **fast and slow** pointer method. This involves moving **one pointer twice** as fast as the other so that when the faster pointer reaches the end, the **slower pointer** will be at the **middle**.*
- *Reverse the second half of the list starting just **after the middle** node and **split** them in **two parts**.*
- *Merge the two halves together by **alternating nodes** from the first half with nodes from the reversed second half.*

[Naive Approach] Using two nested loops- O(n^2) Time and O(1) Space

*Start with the **head** node as the head of the linked list . For **each node**, perform the following operations:*

- *Traverse the list to find the **last node**.*
- *Disconnect the last node from the list.*
- *Place the removed last node after the current node.*
- *Update **current** to its next node.*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Node {
  public:
    int data;
    Node *next;

    Node(int x) {
        data = x;
        next = NULL;
    }
};

Node *reverselist(Node *head) {
    Node *prev = nullptr, *curr = head, *next;

    while (curr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}

void printList(Node *curr) {

    while (curr != nullptr) {
        cout << curr->data << " ";
        curr = curr->next;
    }
}

// Function to rearrange a linked list
Node *rearrange(Node *head) {

    // Find the middle point using tortoise and hare method
    Node *slow = head, *fast = slow->next;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
    }

    // Split the linked list into two halves
    Node *head1 = head;
    Node *head2 = slow->next;
    slow->next = NULL;
```

```
      // Reverse the second half
    head2 = reverselist(head2);

      // Merge alternate nodes
    head = new Node(0);
    Node *curr = head;
    while (head1 || head2) {

        // First add the element from the first list
        if (head1) {
            curr->next = head1;
            curr = curr->next;
            head1 = head1->next;
        }

        // Then add the element from the second list
        if (head2) {
            curr->next = head2;
            curr = curr->next;
            head2 = head2->next;
        }
    }

    // Return the head of the new list
    return head->next;
}


int main() {

     // singly linked list 1->2->3->4->5
    Node *head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
    head->next->next->next->next = new Node(5);

    head = rearrange(head);
    printList(head);
    return 0;
}
```