Three pointers

DUMMY NODES QUESTIONS

(Q) Sort a linked list of 0's, 1's, and 2's

(m1) Changing data in the nodes - O(n) time complexity, O(1) space complexity

- Traverse the list and count the number of 0s, 1s and 2s. Let it be n1, n2, n3 respectively
- Traverse the list again and fill first n_1 nodes with 0, next n_2 nodes with 1 and remaining with 2

```
void sort_0s_1s_2s (node* &head)
{
    int no_zeroes = 0;
    int no_ones = 0;
    int no_twos = 0;

    // traversing the LL to count 0s, 1s and 2s
    node* ptr = head;   // creating a pointer at start of LL
    while (ptr != NULL)
    {
        if (ptr->data == 0)
        {
            no_zeroes++;
        }
        else if (ptr->data == 1)
        {
            no_ones++;
        }
        else if (ptr->data == 2)
        {
            no_twos++;
        }
        ptr = ptr->next;
    }

    // traversing the LL again to change data
    ptr = head;  // creating a pointer at start of LL
    for (int i = 1; i<= no_zeroes ; i++) //filling first no_zeroes nodes with 0s
    {
        ptr -> data = 0;
        ptr = ptr -> next;
    }
    for (int i = 1; i<= no_ones ; i++)  //filling next no_ones nodes with 1s
    {
        ptr -> data = 1;
        ptr = ptr -> next;
    }
    for (int i = 1; i<= no_twos ; i++)  //filling next no_twos nodes with 2s
    {
        ptr -> data = 2;
        ptr = ptr -> next;
    }
}
```

```
int main()
{
    // beacause initially since the LL is empty, head will be pointing to nothing
    node* head = NULL;

    // creating the LL
    insert_at_end(head,0);
    insert_at_end(head,1);
    insert_at_end(head,0);
    insert_at_end(head,0);
    insert_at_end(head,1);
    insert_at_end(head,2);
    insert_at_end(head,1);
    insert_at_end(head,2);
    insert_at_end(head,1);
    display(head);
    cout << endl;

    sort_0s_1s_2s(head);
    display(head);
}
```

(m2) Changing links - three pointers - O(n) time complexity, O(1) space complexity

```
node* sort_0s_1s_2s (node* &head)
{
    // edge/corner cases
    if (head == NULL || head->next == NULL)
    {
        return head;
    }

    // creating dummmy nodes for each of the three LL, this will prevent many NULL checks
    node* dummy_node_zero = new node(-1);
    node* dummy_node_one = new node(-1);
    node* dummy_node_two = new node(-1);

    // adding heads to dummy nodes
    node* head_0_LL = dummy_node_zero;
    node* head_1_LL = dummy_node_one;
    node* head_2_LL = dummy_node_two;

    // creating pointer to traverse through the three LL
    node* ptr_0 = head_0_LL;
    node* ptr_1 = head_1_LL;
    node* ptr_2 = head_2_LL;

    // traversing the original LL and while traversing if we encounter 0 then putting it in zeroes_LL and similarly for 1s and 2s
    node* ptr_org = head; // creating a pointer to original given LL
    while (ptr_org != NULL)
    {
        if (ptr_org->data == 0)
        {
            ptr_0->next = ptr_org;
            ptr_0 = ptr_0->next;
            ptr_org = ptr_org->next;
        }
        else if (ptr_org->data == 1)
        {
            ptr_1->next = ptr_org;
            ptr_1 = ptr_1->next;
            ptr_org = ptr_org->next;
        }
        else
        {
            ptr_2->next = ptr_org;
            ptr_2 = ptr_2->next;
            ptr_org = ptr_org->next;
        }
    }

    //combining the 3 LL as 1 LL with zeroes_LL at start, followed by ones_LL and then twoes_LL
    ptr_0->next = (head_1_LL->next);
    ptr_1->next = head_2_LL->next;
    ptr_2->next = NULL;

    // returning the head of the combined LL
    return head_0_LL->next;
}
```

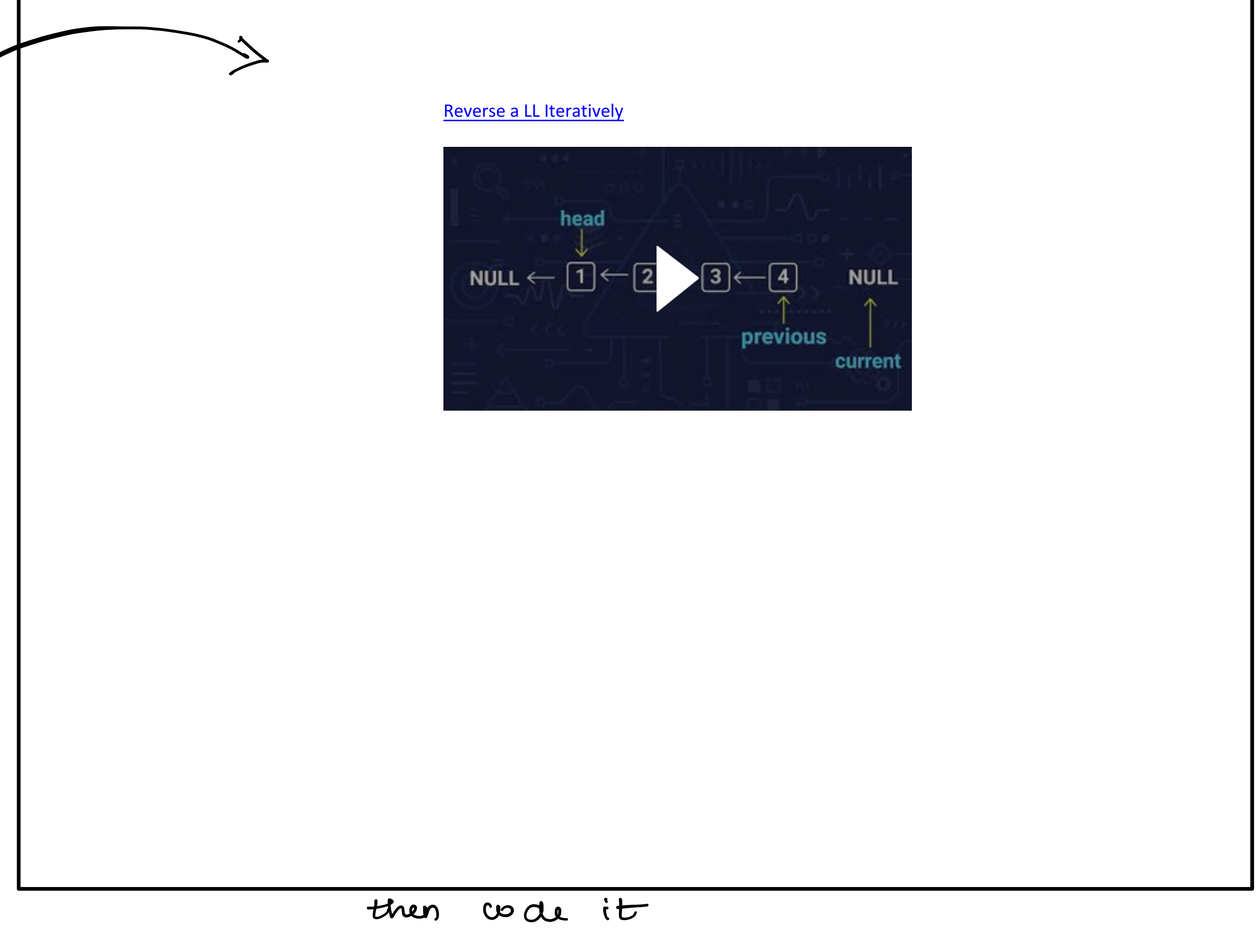
SLIDING POINTERS QUESTION

V.V.Imp

(Q) Reverse the given linked list.

→ O(n) time comp, O(1) space comp

think math first



then code it

(Q) Check if given linked list is palindrome or not

reverse the linked list using above method and then compare the two linked lists