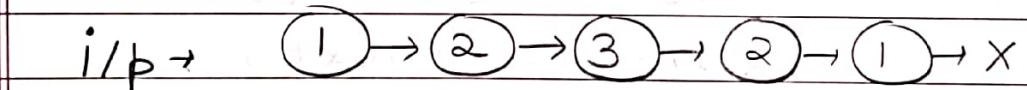
22/04/2023

Q1 Check your Linked List is palindrome or not.



o/p → True

Palindrome → Same when read either from left and right, then it is a palindrome.

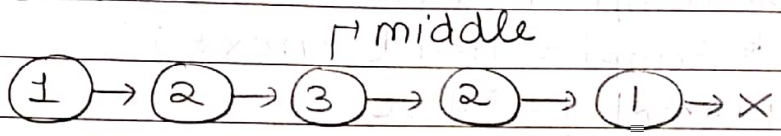
Approach-1
Create a new linked list and that linked list will be the reverse of the input linked list and then compare both the linked list.

Time complexity = O(n)
Space complexity = O(n)

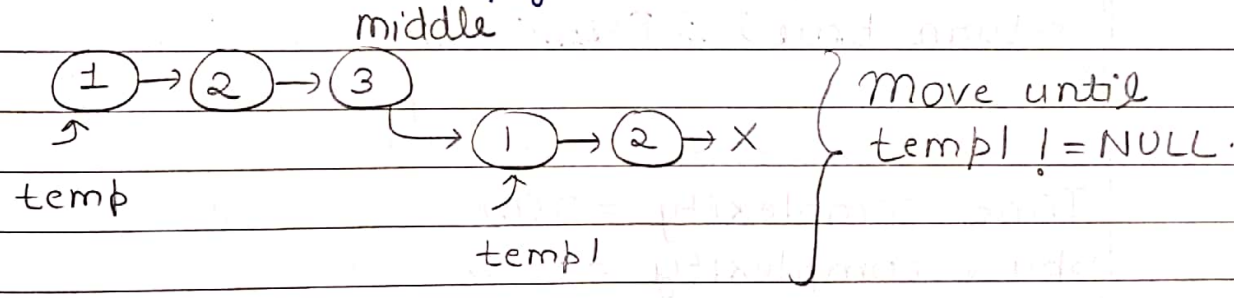
Approach-2
Copy the data of linked list to the array and then via 2 pointer approach, check for palindrome. However this again has

space complexity of O(n). We need to do in O(1) space.

Approach - 3



1) Reverse the linked list from middle and for that we need to find the middle of the linked list.
2) Now start comparing both the half and if we find any value not equal, simply return false. Do this until we reach NULL. If false was not returned then simply return true.



Code

```
bool checkPalindrome (Node * & head) {
    //Empty linked list case
    if (head == NULL) {
        return true;
    }
    //Single node is a palindrome
    if (head -> next == NULL) {
        return true;
    }
    //Step-1 ⇒ Find middle
    Node * middle = getMiddle (head);
    //Step-2 ⇒ Reverse the linked list after middle
```

To get $\frac{n}{2}$ as middle
fast = head->next;
Done in prev class

Done in LL class-2

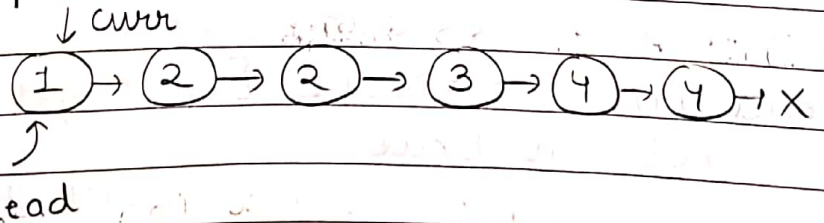
```
Node * reverseLLHead = reverse(middle->next);
middle->next = reverseLLHead; // Optional
//Step-3 Compare both linked lists.
Node * temp = head;
Node * temp1 = head->next;
while (temp1 != NULL) {
        //Not a palindrome
        if (temp->data != temp1->data) {
                return false;
        }
        temp = temp->next;   } Move forward
        temp1 = temp1->next; } in both linked
}                              list
return true; // Palindrome
}
```

Time complexity = O(n)
Space complexity = O(1)

Q2 Remove duplicates from sorted linked list.

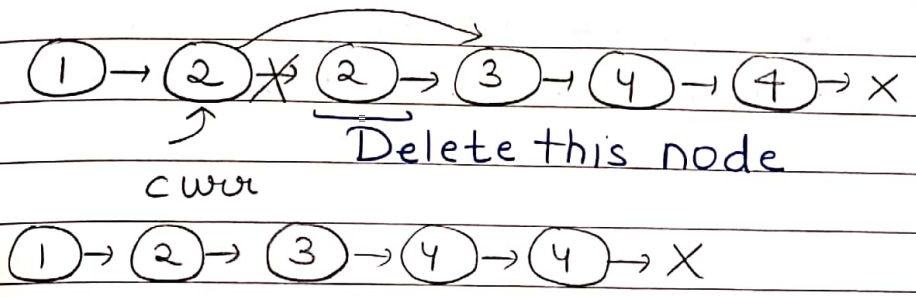
i/p → (1) → (2) → (2) → (3) → (4) → (4) → X
o/p → (1) → (2) → (3) → (4) → X

Approach-1



compare curr->data and curr->next->data. If not equal, then simply move curr forward. If found equal, then move

curr → next to curr → next → next.



Similarly we will be able to delete the node with value/data → 4.

Code

```
void removeDuplicates (Node * & head) {
    //Empty linked list
    if (head == NULL) {
        return;
    } //Single node can't be duplicate
    if (head → next == NULL) {
        return;
    }
    //Follow steps of approach
    Node * curr = head;
    while (curr != NULL) {

        if ((curr → next != NULL) && (curr → data
            == curr → next → data)) {
            //Change the pointers
            Node * temp = curr → next;
            curr → next = curr → next → next;
            //Delete node for not wasting memory
            temp → next = NULL;
            delete temp;
        }
```

```
        else { // Simply move curr forward
            curr = curr -> next;
        }
    }
}
```

Time complexity = O(n)
Space complexity = O(1)

Q3 Sort 0s, 1s and 2s in the linked list.

i/p → (2) → (1) → (2) → (0) → (1) → X
o/p → (0) → (1) → (1) → (2) → (2) → X

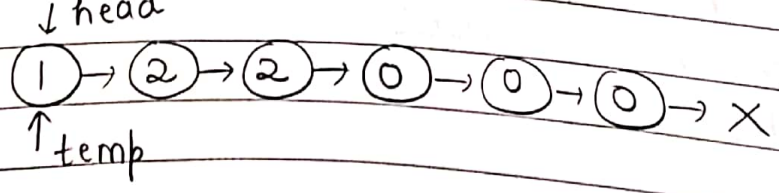
Approach - 1

1) Simply store number of 0s, 1s and 2s in the linked list
2) Now first add 0s, 1s and 2s with the help of count stored.

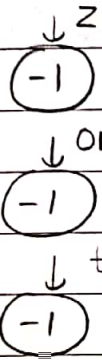
Time complexity = O(n)
Space complexity = O(1)

This approach has one issue that here data replacement is done but in interview, the person might say that data replacement is not allowed.

Approach - 2



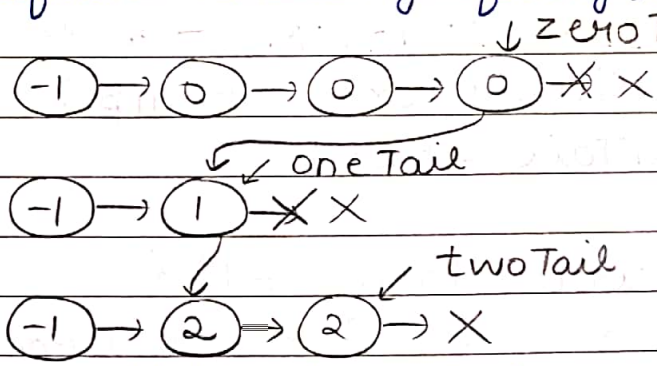
1) Create 3 dummy nodes




```
temp = head;
head = head->next;
temp->next = NULL;
```

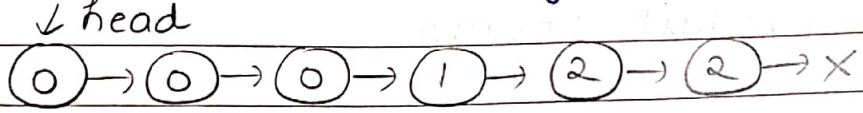
So after traversing fully, we get

2)



Join all the linked list

3) Remove all the dummy nodes



return the head.

Code

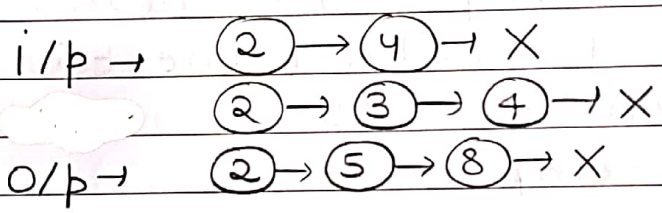
```
Node * sortLinkedList (Node * & head) {
    //Creation of dummy nodes
    Node * zeroHead = new Node (-1);
    Node * zeroTail = zeroHead;
    Node * oneHead = new Node (-1);
```

```
Node *oneTail = oneHead;
Node *twoHead = new Node(-1);
Node * twoTail = twoHead;
// Traverse original linked list
Node * curr = head;
while (curr != NULL) {
    // Handling node with 0 value
    if (curr->data == 0) {
        // Isolate node
        Node * temp = curr;
        curr = curr->next;
        temp->next = NULL;
        // Append 0 in the zeroHead LL
        zeroTail->next = temp;
        zeroTail = temp;
    }
    else if (curr->data == 1) {
        // Same steps for isolation of
        0 node
        // Append 1 in the oneHead LL
        oneTail->next = temp;
        oneTail = temp;
    }
    else {
        // Same steps for isolation of
        0 node
        // Append 2 in the twoHead LL
        twoTail->next = temp;
        twoTail = temp
    }
}
```

```
// Join them and delete dummy nodes
// Modify oneHead LL
Node * temp = oneHead;
oneHead = oneHead -> next;
temp -> next = NULL;
delete temp;
//Modify twoHead LL
temp = twoHead;
twoHead = twoHead -> next;
temp -> next = NULL;
delete temp;
//Join List
if (oneHead != NULL) {
    //oneHead LL is not empty
    zeroTail -> next = oneHead; //Attach
    if (twoHead != NULL) { //twoHead LL not empty
        oneTail -> next = twoHead; <- Connect
    }
}
else { //oneHead LL is empty.
    // Attach 0 LL and 2 LL
    if (twoHead != NULL) {
        zeroTail -> next = twoHead;
    }
}
//Delete dummy node of 0 LL
temp = zeroHead;
zeroHead = zeroHead -> next;
temp -> next = NULL;
delete temp;
//Return new head of LL
return zeroHead;
}
```

TC = linear
SC = O(1)

Q4 Add 2 numbers represented by linked list.



1) Reverse both the linked list
2) Add both the linked list
3) Reverse the answer obtained & that is the answer.

Code

```
Node * add (Node * &h1, Node * &h2){
    //Step 1 Reverse both LL
    h1 = reverse (h1);
    h2 = reverse (h2);
    //Step 2 Add the LL
    Node * ansHead = NULL;
    Node * ansTail = NULL;
    int carry = 0;
    while (head1 != NULL && head2 != NULL){
    //Find sum
    int sum = carry + h1→data + h2→data;
    //Find digit
    int digit = sum % 10;
    //Find carry
    carry = sum/10;
    Node * newNode = new Node (digit);
    if (ansHead == NULL){
        //First node to insert
```

```
        ansHead = newNode;
        ansTail = newNode;
    }
    else { // Not 1st node
        ansTail -> next = newNode;
        ansTail = newNode;
    }
h1 = h1 -> next;
h2 = h2 -> next;
}
// 1st LL is bigger
while (h1 != NULL) {
    int sum = carry + h1 -> data;
    int digit = sum % 10;
    carry = sum / 10;
    Node * newNode = new Node (digit);
    ansTail -> next = newNode;
    ansTail = newNode;
    h1 = h1 -> next;
}
// IInd LL is bigger
while (h2 != NULL) {
    // Same steps followed above
}
// Handle the carry
while (carry != 0) {
    int sum = carry;
    int digit = sum % 10;
    carry = sum / 10;
    Node * newNode = new Node (digit);
    ansTail -> next = newNode;
    ansTail = newNode;
}
```

```
//Reverse the ans LL
ansHead = reverse(ansHead);
return ansHead;
}
```

Note → reverse function has been discussed in class 2 of linked list.

```
Handle extra cases
if (head1 == NULL)
        return head2;
if (head2 == NULL)
        return head1;
```

Time complexity = O(max(m,n)) → linear
Space complexity = O(1)