

* Reverse the linked list - (Leetcode - 206)

Given the head of singly-linked list, reverse the list and return the reversed list.

I/P - head = [1, 2, 3, 4, 5]

O/P - [5, 4, 3, 2, 1]

- Iterative approach -

Step 1: ListNode* prev = NULL.

Step 2: ListNode* curr = head.

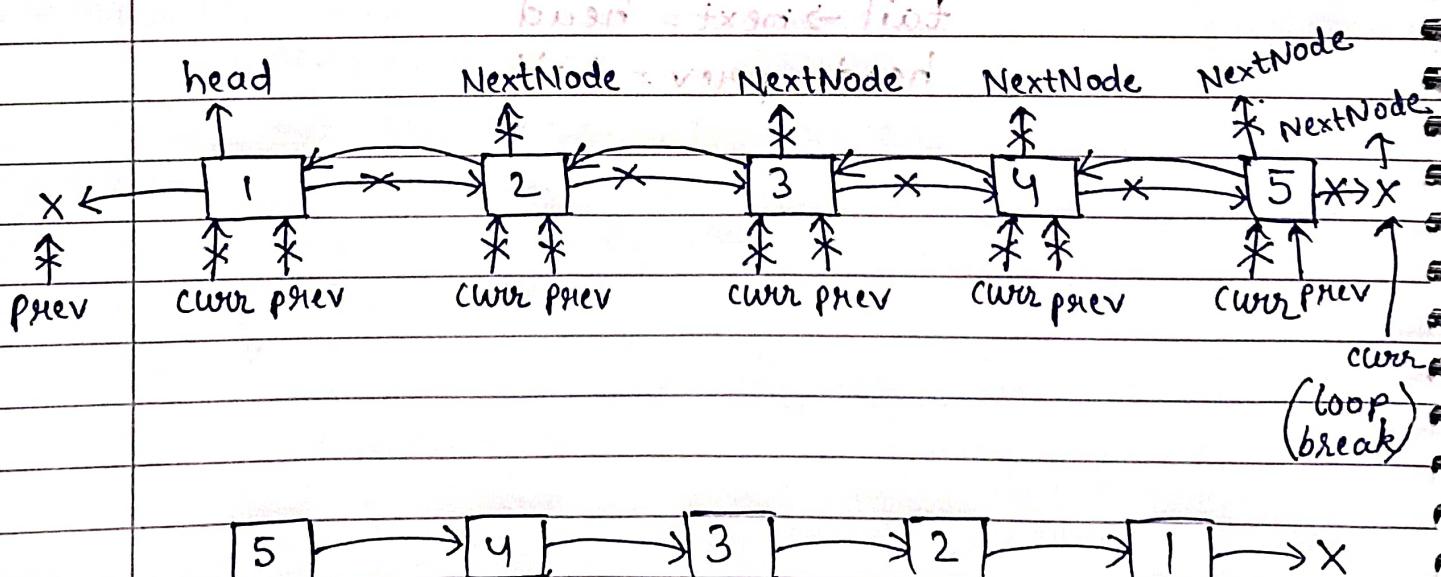
Step 3: (i) ListNode* NextNode = curr -> next.

(ii) curr -> next = prev.

(iii) prev = curr.

(iv) curr = NextNode.

Step 4: Repeat step 3 while (curr != NULL).



Code -

```
class Solution {
public:
    listNode* reverseList (listNode* head) {
        listNode* prev = NULL;
        listNode* curr = head;
        while (curr != NULL) {
            listNode* NextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = NextNode;
        }
        return prev;
    }
};
```

Recursive approach -

Base case - if (`curr == NULL`), return `prev`

Step1: `listNode* NextNode = curr->next.`

Step2: `curr->next = prev.`

Step3: `prev = curr.`

Step4: `curr = NextNode.`

Step5: Make a recursive call.

Code- class solution {

public:

```
listNode* reverseUsingRecursion(listNode*&prev, listNode*&curr)
```

{

if (curr == NULL) return prev;

else {

listNode* NextNode = curr->next;

curr->next = prev;

return reverseUsingRecursion(curr, NextNode);

}

```
listNode* reverseList(listNode* head) {
```

listNode* prev = NULL;

listNode* curr = head;

```
return reverseUsingRecursion(prev, curr);
```

}

LL

* Middle of the Linked List (Leetcode - 876)

Given the head of singly linked list, return the middle node of linked list.

If there are two middle nodes, return the second middle node.

Example 1 - odd no. of nodes

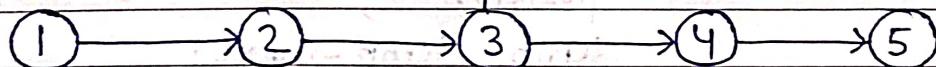
I/P - head = [1, 2, 3, 4, 5]

O/P - [3, 4, 5]

Example 2 - even no. of nodes

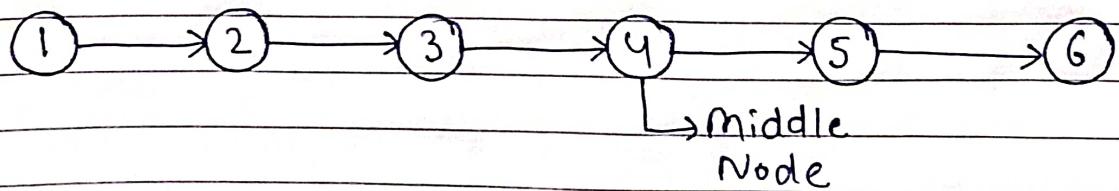
I/P - head = [1, 2, 3, 4, 5, 6]

O/P - [4, 5, 6]



$$n = 5$$

$$\text{Middle node} = \frac{n+1}{2} = \frac{5+1}{2} = 3$$



$$n = 6$$

$$\text{Middle node} = \frac{n+1}{2} = \frac{6+1}{2} = 4$$

Code -

class Solution {

public:

```
int findLength(ListNode*& head) {
    ListNode* temp = head;
    int len = 0;
    while (temp != NULL) {
        len++;
        temp = temp->next;
    }
    return len;
}
```

ListNode* MiddleNode(ListNode* head) {

int n = findLength(head);

int position = $\frac{n+1}{2}$; // head - size

ListNode* temp = head;

while (position != 1) {

position--;

} // temp = temp->next;

return temp;

}; // if size is odd then return middle node

}; // if size is even then return next to middle node

else {

return;

}; // if size is even then return next to middle node

— 11 —

* Tortoise algorithm - Middle of linked list

Slow and fast pointers →

Slow - increment by 1 step.

Fast - increment by 2 steps.

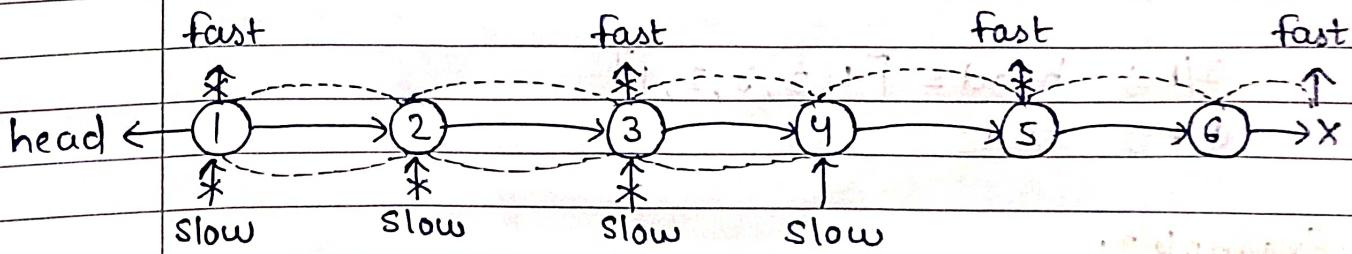
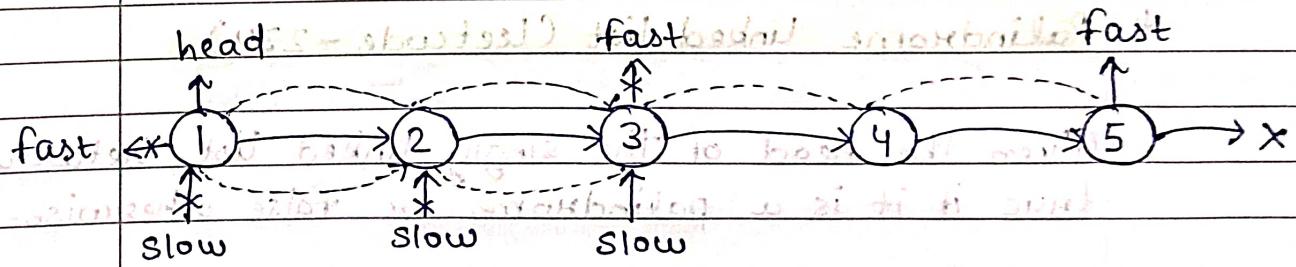
If fast pointer 2 step increment kar paya,
only then slow pointer 1 step se increment
krega.

Step 1: Set slow and fast pointer at head.

Step 2: Increment fast by 2 steps. } until fast approaches

Step 3: Increment slow by 1 step. } NULL.

Step 4: return slow.



Code -

```

class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        // Getting fast from while
        ListNode* slow = head;
        ListNode* fast = head;
        // fast is mid element + 1st
        while (fast != NULL) {
            // fast from if fast->next;
            if (fast != NULL) {
                fast = fast->next;
                slow = slow->next;
            }
        }
        // getting spot from while
        return slow; // left boundary of 2nd half
    }
};

```

* Palindrome linked list (Leetcode - 234)

Given the head of the singly linked list, return true if it is a palindrome or false otherwise.

Example 1:

I/P \Rightarrow head = [1, 2, 3, 2, 1]O/P \Rightarrow true

Example 2:

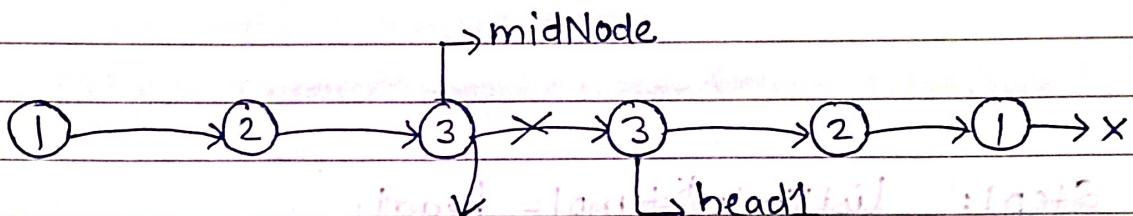
I/P \Rightarrow head = [1, 2]O/P \Rightarrow false

Step 1: Break into two halves.

Step 2: Reverse the second half.

Step 3: Compare the two linked lists.

Break into two halves -

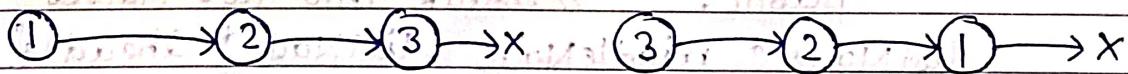


Step 1: find *midNode*.

Step 2: *listNode* head1 = midNode->next;*

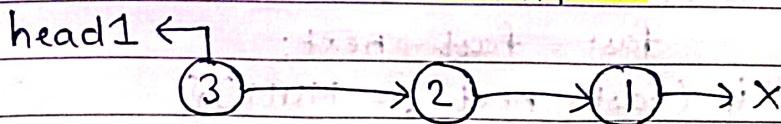
Step 3: *midNode->next = NULL.*

Now, the two halves are →

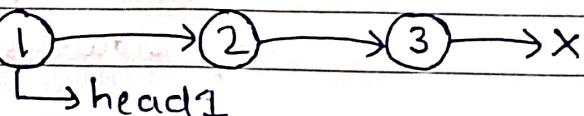


First half Second half

Reverse the second half -



After reversing -

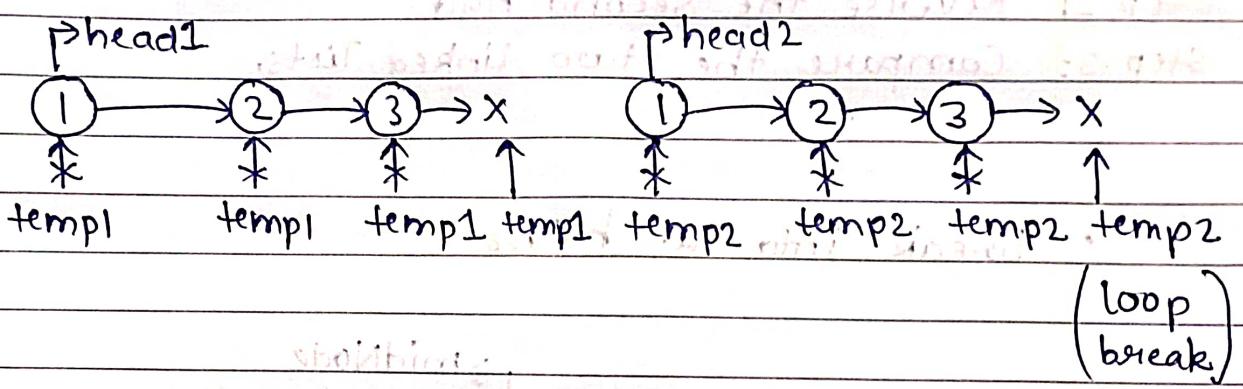


Step 1: *listNode* prev = NULL,*

Step 2: *listNode* curr = head1.*

Step 3: *head1 is updated now,*

- Compare the two linked lists -



- Step 1: `listNode *temp1 = head1;`
- Step 2: `listNode *temp2 = head2;`
- Step 3: (i) `if (temp1->val != temp2->val)` return false
(ii) else, `temp1 = temp1->next` & `temp2 = temp2->next`
- Step 4: Repeat the Step 3 until (`temp2 != NULL`).
- Step 5: Otherwise return true.

Code - class Solution {

```

public: // break into two halves
listNode* middleNode (listNode*&head) {
    listNode* slow = head;
    listNode* fast = head;

    while (fast->next != NULL) {
        fast = fast->next;
        if (fast->next != NULL) {
            fast = fast->next;
            slow = slow->next;
        }
    }
    return slow;
}

```

LL

// reverse the second half

```
listNode * reverseUsingRecursion(listNode * &prev,  
                                listNode * &curre)  
{  
    if (curre == NULL) {  
        return prev;  
    }  
    listNode * NextNode = curre->next;  
    curre->next = prev;  
    return reverseUsingRecursion(curre, NextNode);  
}
```

// compare the two linked lists

```
bool compareList(listNode * &head1, listNode * &head2)  
{  
    listNode * temp1 = head1;  
    listNode * temp2 = head2;  
  
    while (temp2 != NULL) {  
        if (temp1->val != temp2->val) {  
            return false;  
        }  
        temp1 = temp1->next;  
        temp2 = temp2->next;  
    }  
    return true;  
}
```

```

bool isPalindrome(listNode* head) {
    listNode* midNode = middleNode(head);
    listNode* head1 = midNode->next;
    midNode->next = NULL;

    listNode* prev = NULL;
    listNode* curr = head1;
    head1 = reverseUsingRecursion(prev, curr);
}

```

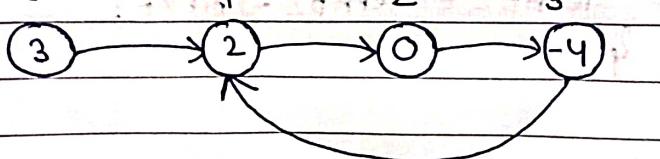
Chap 2 shall return compareList(head, head1);

* Linked list cycle (Leetcode - 141)

Given head, the head of the linked list, determine if linked list has cycle in it.

Return true if there is a cycle otherwise return false.

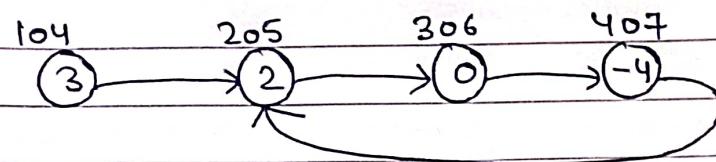
Example 1 -



I/P - head = [3, 2, 0, -4], pos = 1

O/P - true

Logic - Agr linked list traverse karte time koi address repeat kar gaye, it means loop is present.



By default, in map, all addresses are marked false. Mark visited addresses true and if any address is re-visited, it means cycle is present. So, in that case, return true.

Using map

	Address	Status	
	104	T	
	205	T	
	306	T	
	407	T	loop present

Code - class solution

public:

```
bool hasCycle (listNode* head){
```

```
map<listNode*, bool> table;
```

```
listNode* temp = head;
```

```
while (temp != NULL) {
```

```
if (table[temp] == false) {
```

```
table[temp] = true;
```

```
}
```

```
else {
```

```
return true; //cycle is present
```

```
}
```

```
temp = temp->next;
```

```
}
```

```
return false; //Agr loop se bahr aa gaye, it
```

```
{ } ; // means cycle is absent.
```