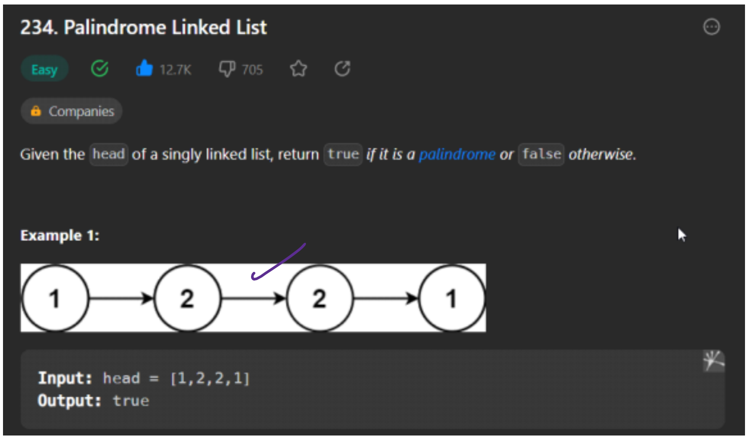
Day 22

19 December 2022 22:02



234. Palindrome Linked List

Easy 12.7K 705

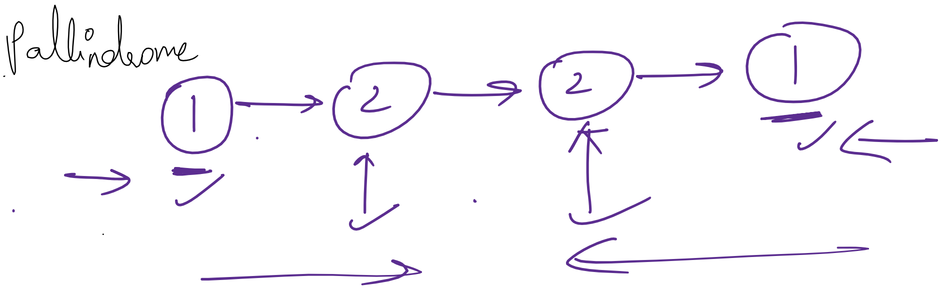
Companies

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

Example 1:

```
Input: head = [1,2,2,1]
Output: true
```

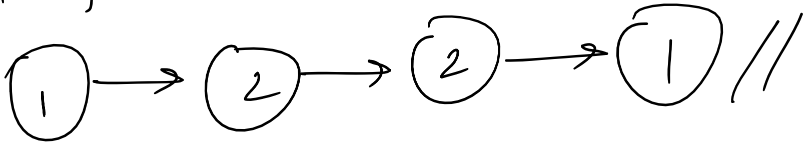
Pallindrome



left	1	2	2	1
right	1	2	2	1

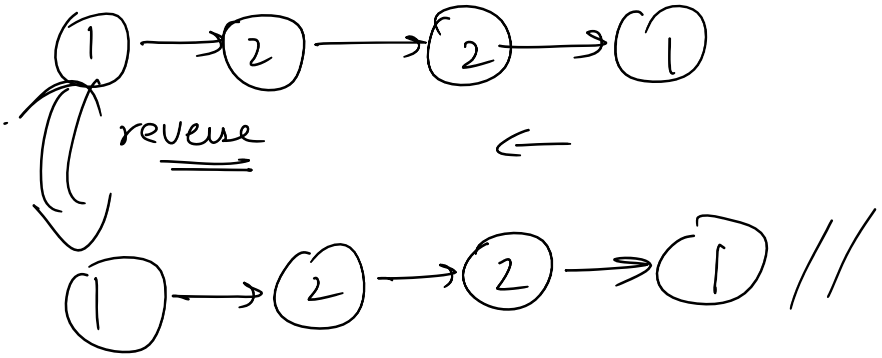
→ Yes

Linked list 1



Linked list 2

Extra



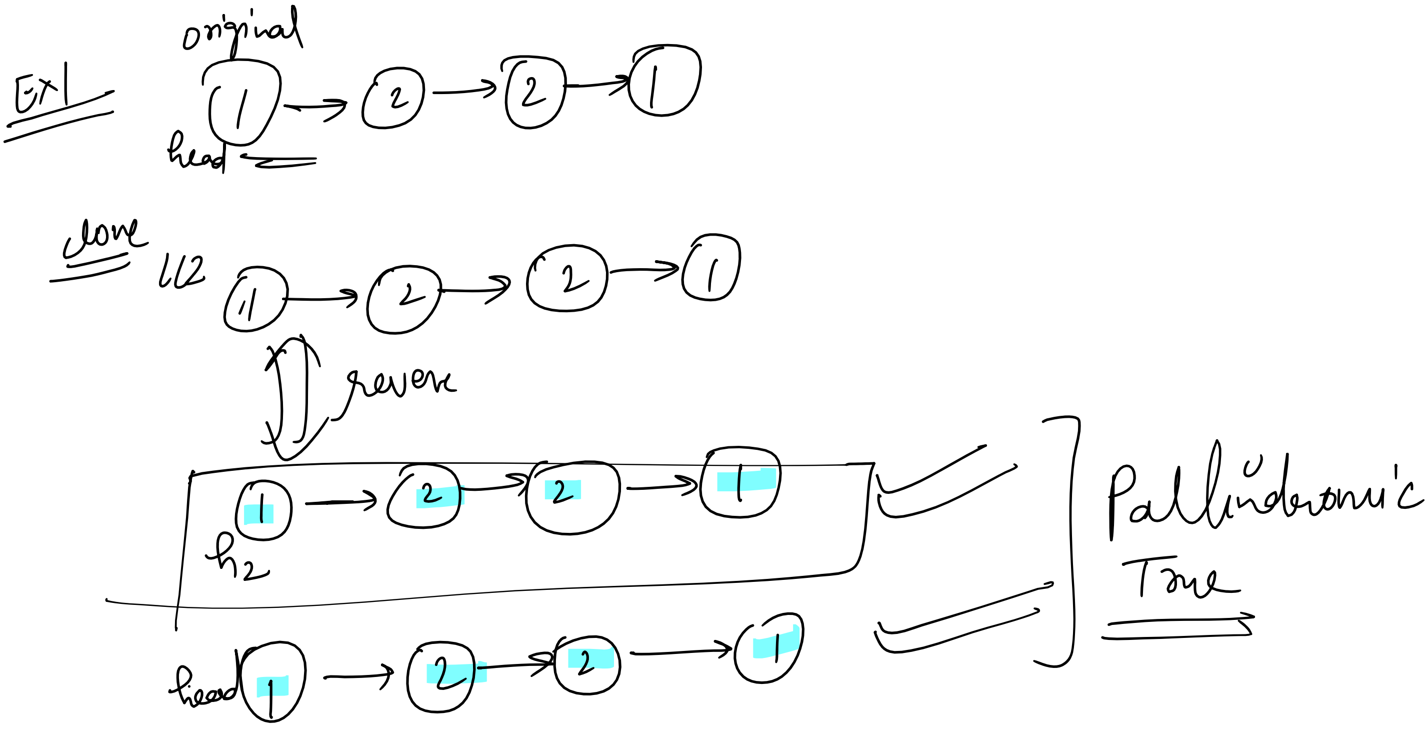
pallindromic ✓

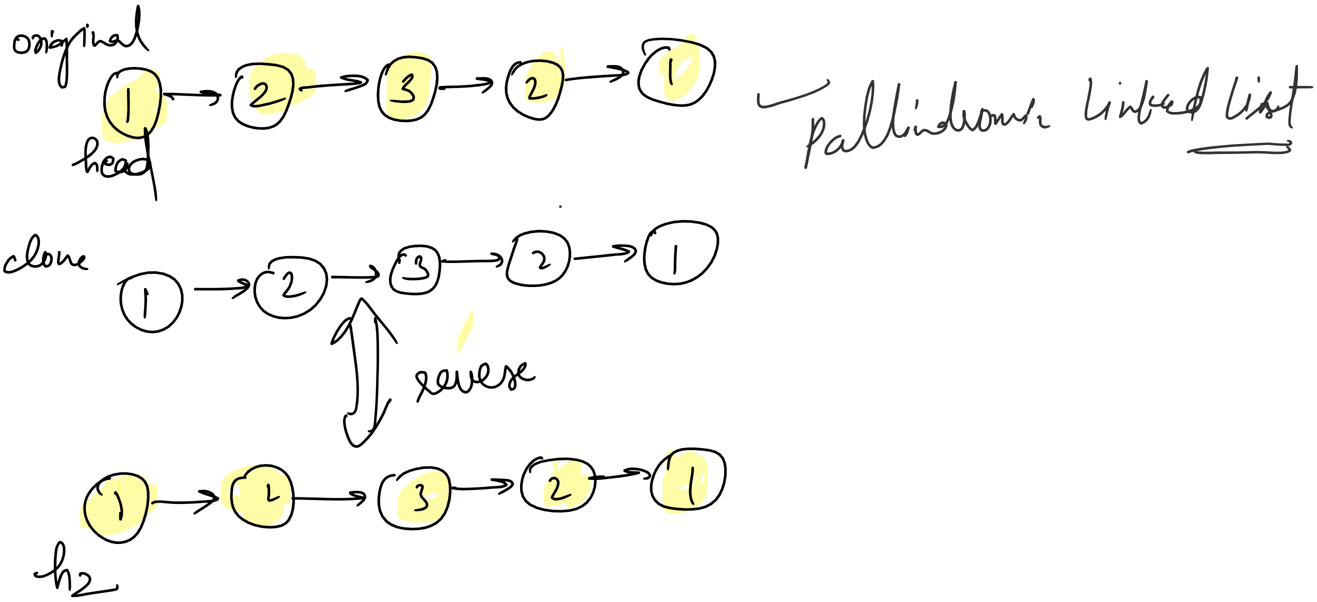
Algorithm

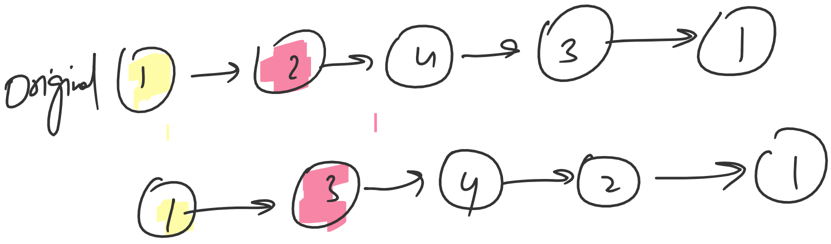
① Copy of Linked List. ✓

② Copy LL Reverse (LL2) ✓

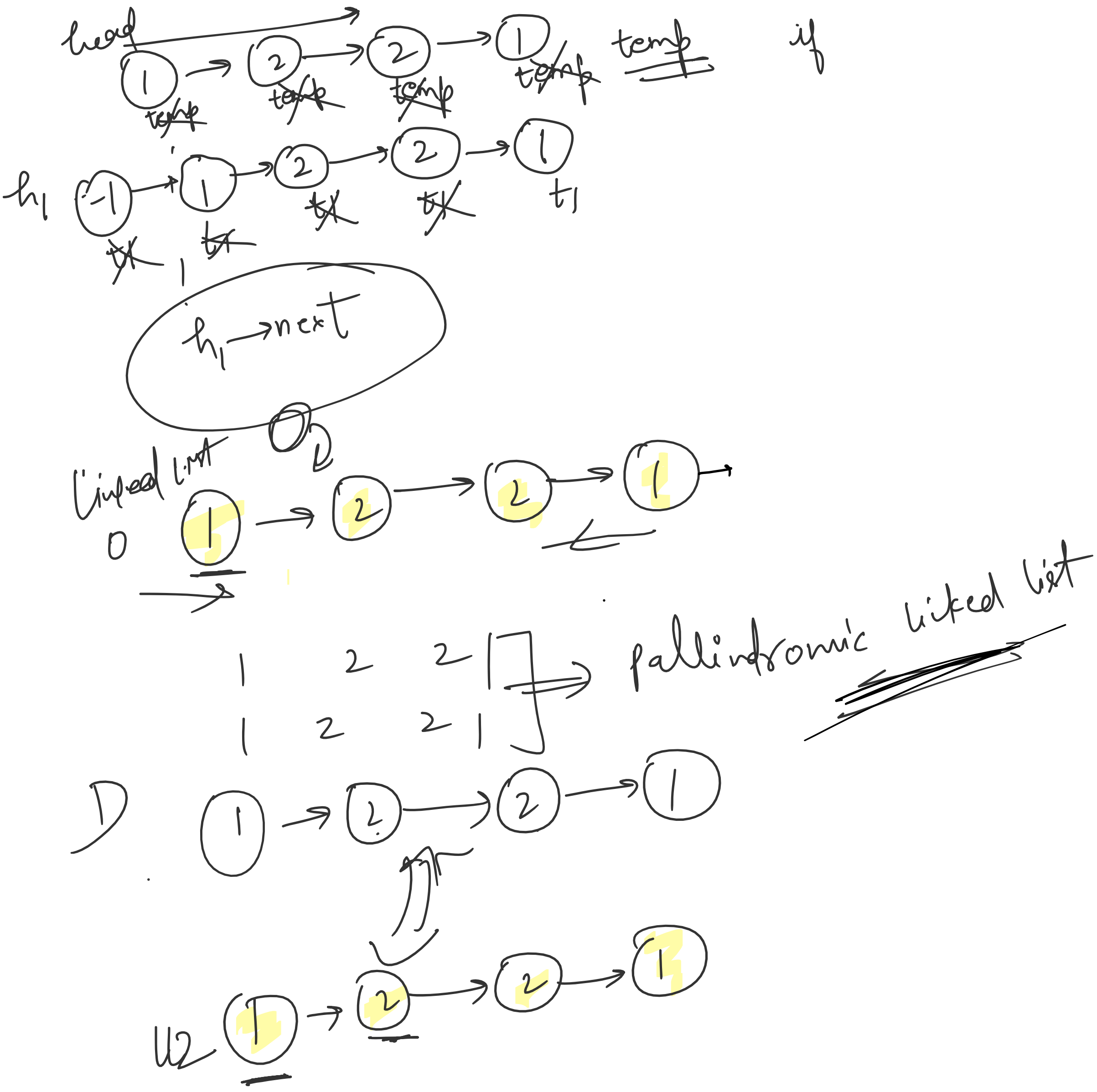
③ Original ⇔ (LL2) Node By Node Compare



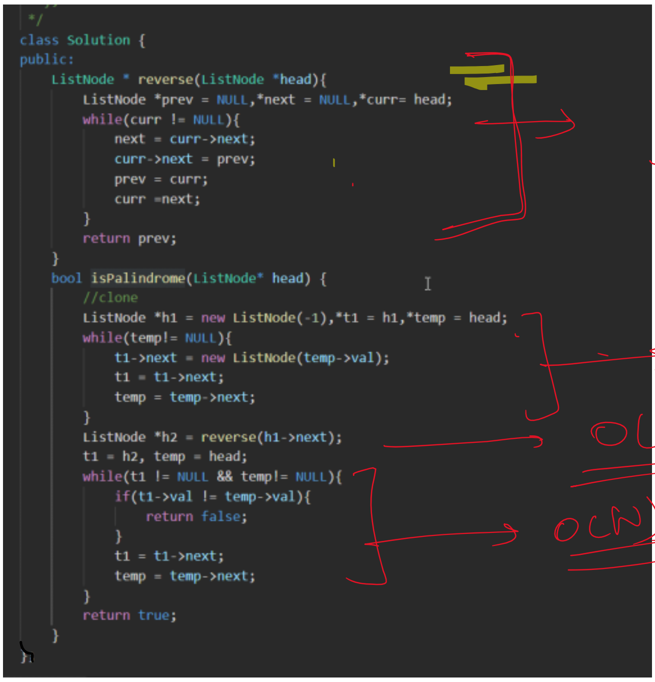




False



pallindronic liked list



```
 */
class Solution {
public:
    ListNode * reverse(ListNode *head){
        ListNode *prev = NULL,*next = NULL,*curr= head;
        while(curr != NULL){
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr =next;
        }
        return prev;
    }
    bool isPalindrome(ListNode* head) {
        //clone
        ListNode *h1 = new ListNode(-1),*t1 = h1,*temp = head;
        while(temp!= NULL){
            t1->next = new ListNode(temp->val);
            t1 = t1->next;
            temp = temp->next;
        }
        ListNode *h2 = reverse(h1->next);
        t1 = h2, temp = head;
        while(t1 != NULL && temp!= NULL){
            if(t1->val != temp->val){
                return false;
            }
            t1 = t1->next;
            temp = temp->next;
        }
        return true;
    }
};
```

O(N)

O(N)

O(N)

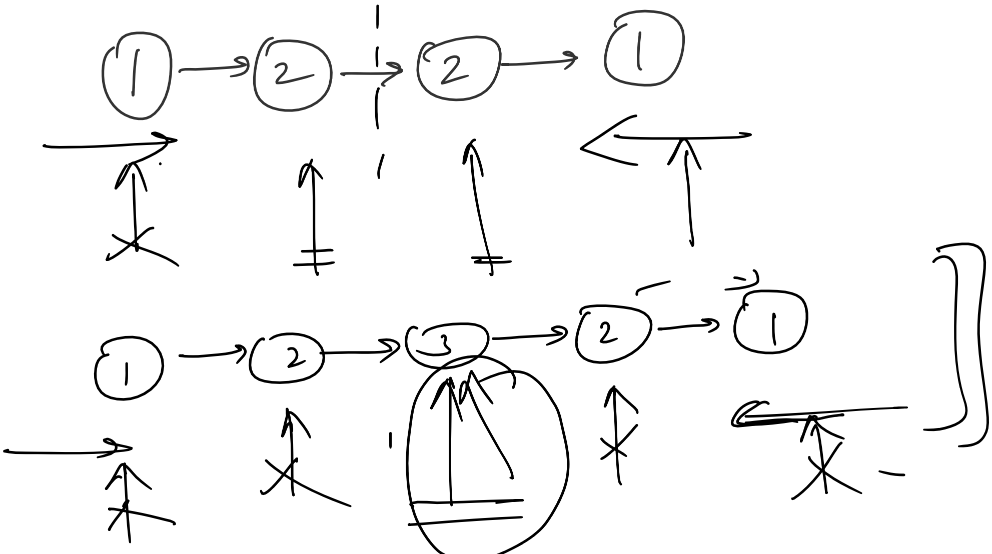
T.C – O(3N) ≈ O(N)

S.C – O(N) → because of clone

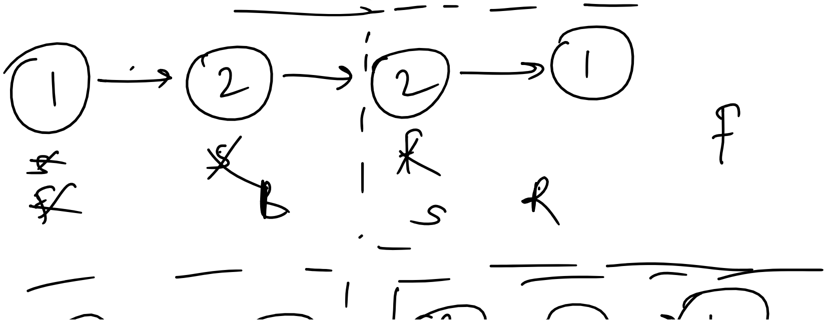
T.C – O(3N)

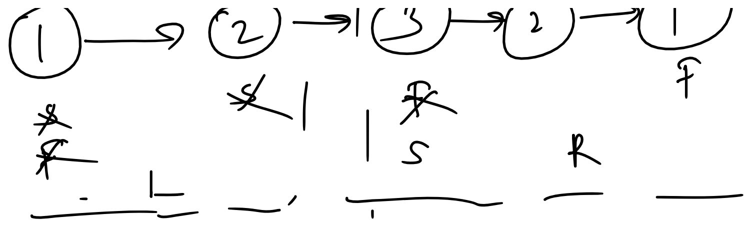
S.C – O(N) ≈

Optimize

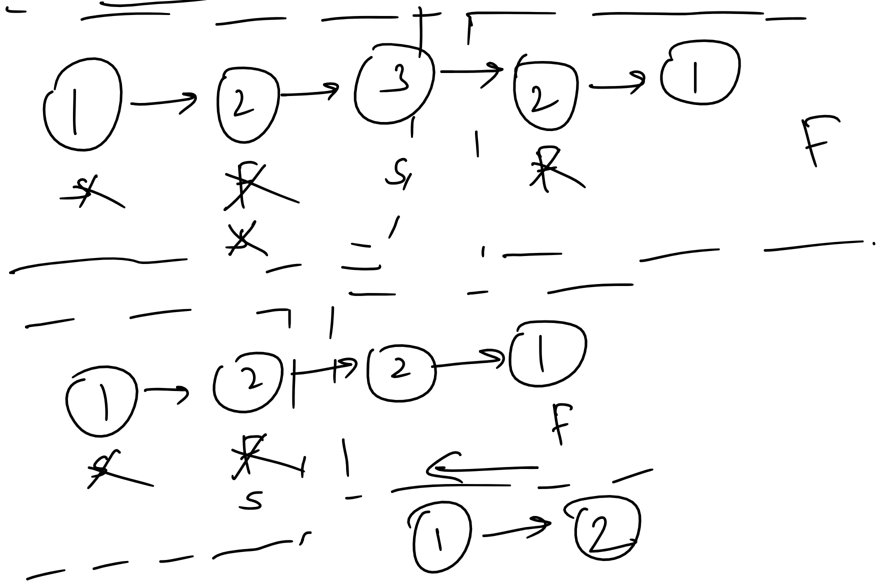


① middle element



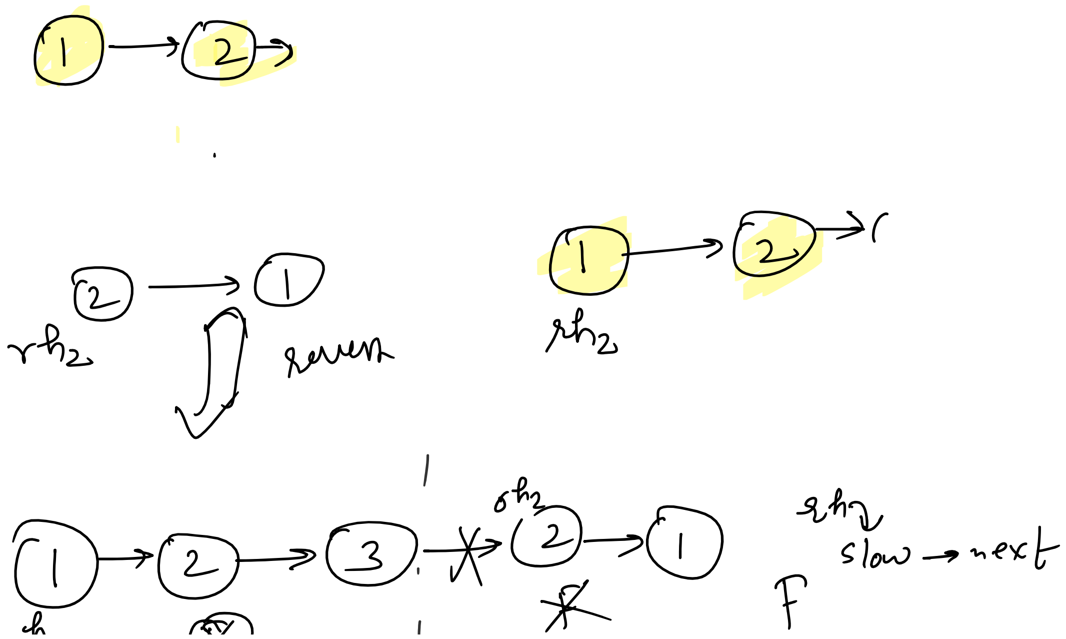


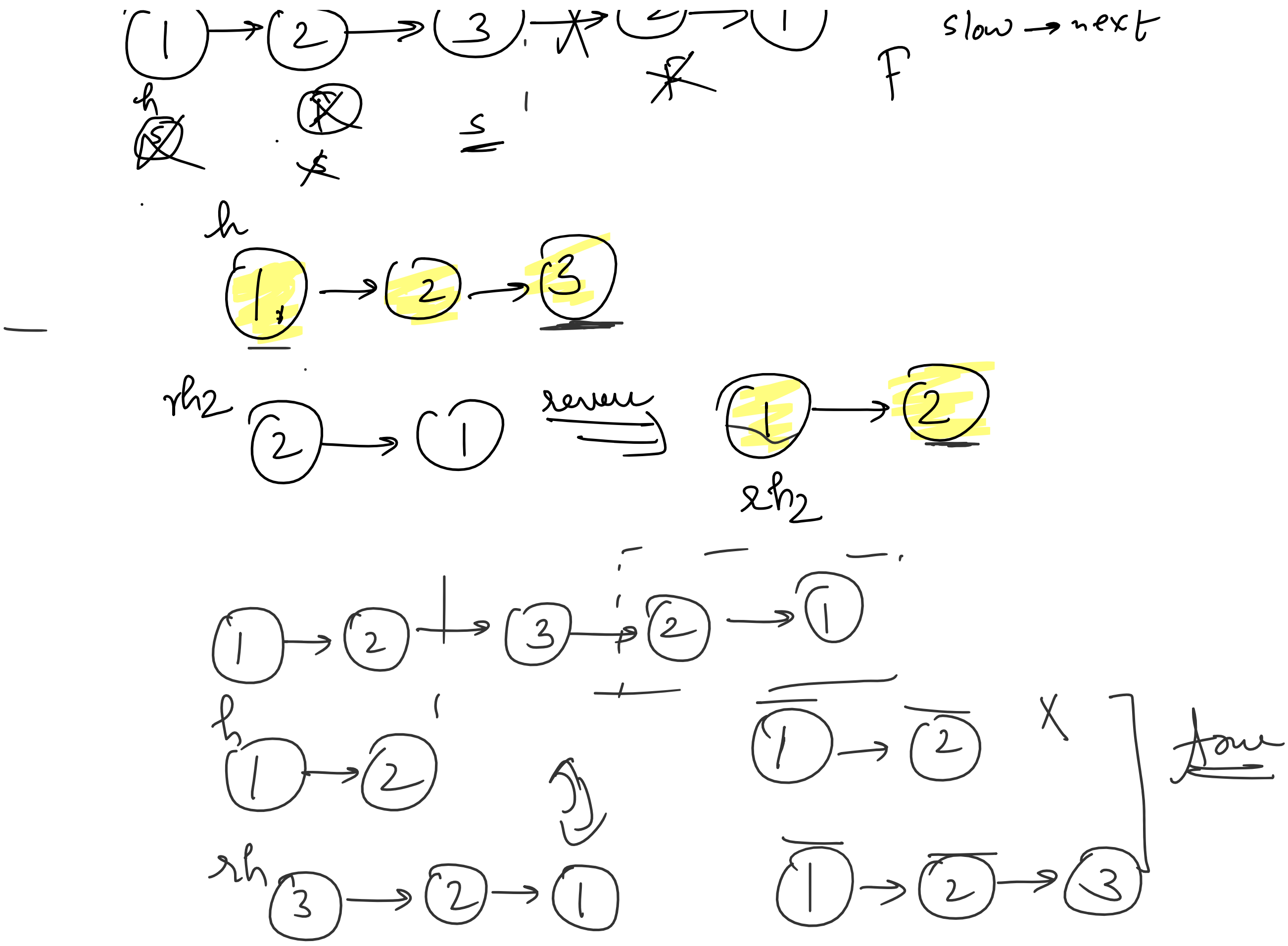
Left Oriented



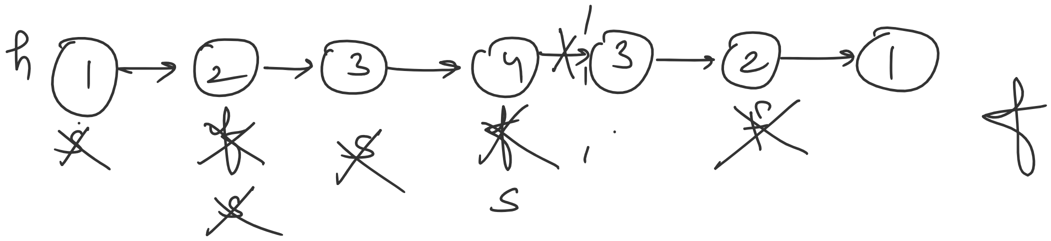
Algorithm

1. middle element find first
2. Right half of LL (Reverse)
3. Node by Node by comparison.

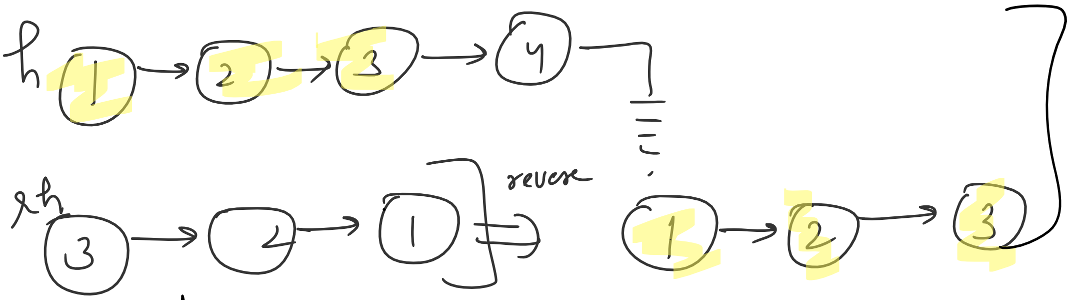




Algo
① middlee element (left)
② Right वाले Part को reverse करना
③ Node by Node

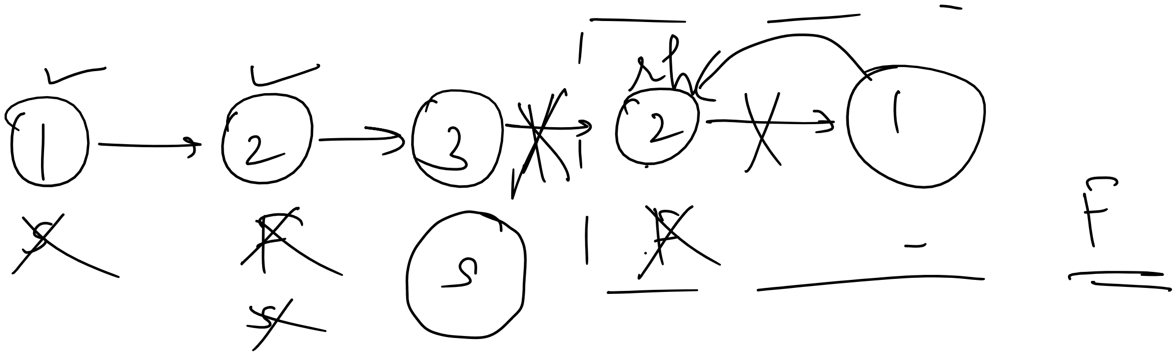


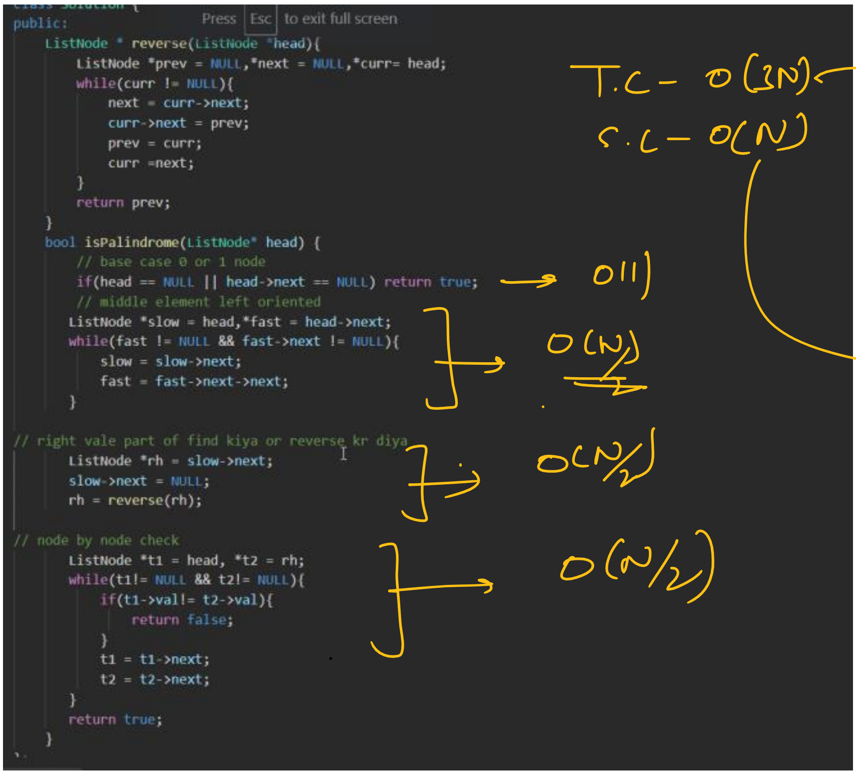
rh = s → next



return true;

return true;





Press Esc to exit full screen

```
class Solution {
public:
    ListNode * reverse(ListNode *head){
        ListNode *prev = NULL,*next = NULL,*curr= head;
        while(curr != NULL){
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr =next;
        }
        return prev;
    }
    bool isPalindrome(ListNode* head) {
        // base case 0 or 1 node
        if(head == NULL || head->next == NULL) return true;
        // middle element left oriented
       ListNode *slow = head,*fast = head->next;
       while(fast != NULL && fast->next != NULL){
           slow = slow->next;
           fast = fast->next->next;
       }

// right vale part of find kiya or reverse kr diya
       ListNode *rh = slow->next;
       slow->next = NULL;
       rh = reverse(rh);

// node by node check
       ListNode *t1 = head, *t2 = rh;
       while(t1!= NULL && t2!= NULL){
           if(t1->val!= t2->val){
               return false;
           }
           t1 = t1->next;
           t2 = t2->next;
       }
       return true;
    }
};
```

T.C – O(3N)

S.C – O(N)

→ O(1)

→ O(N)

→ O(N/2)

→ O(N/2)

Optimization

T.C – O(3N/2) ~ O(N)

S.C – O(1)

T.C – O(3N/2)

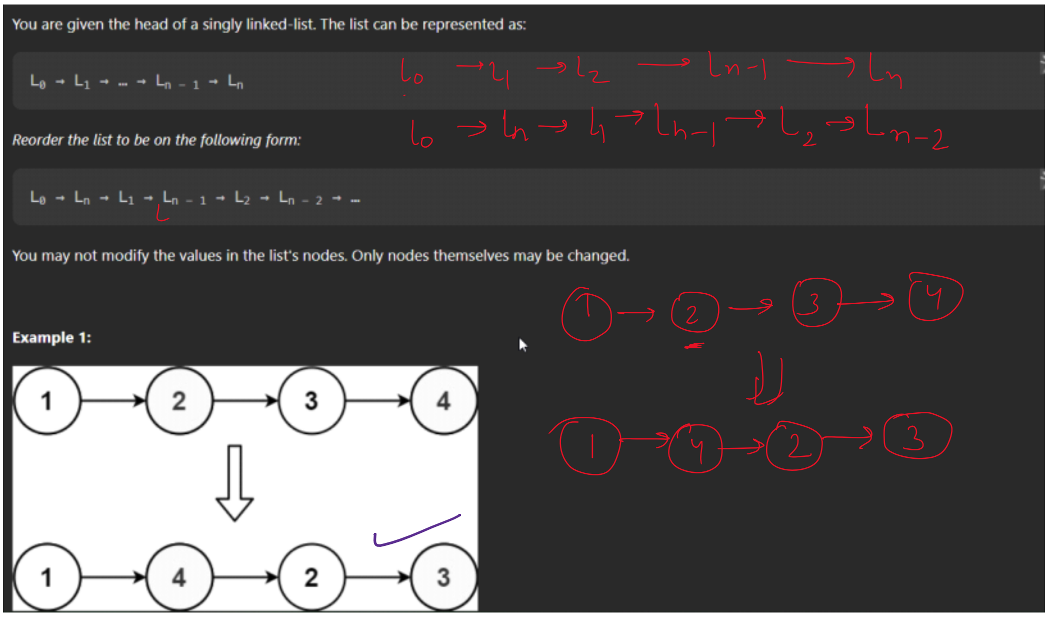
S.C – O(1)

Algorithm

① middle element ①

Right side of linked list] reverse

Node by Node



You are given the head of a singly linked-list. The list can be represented as:

```
L0 → L1 → … → Ln - 1 → Ln
```

$l_0 \to l_1 \to l_2 \to l_{n-1} \to l_n$

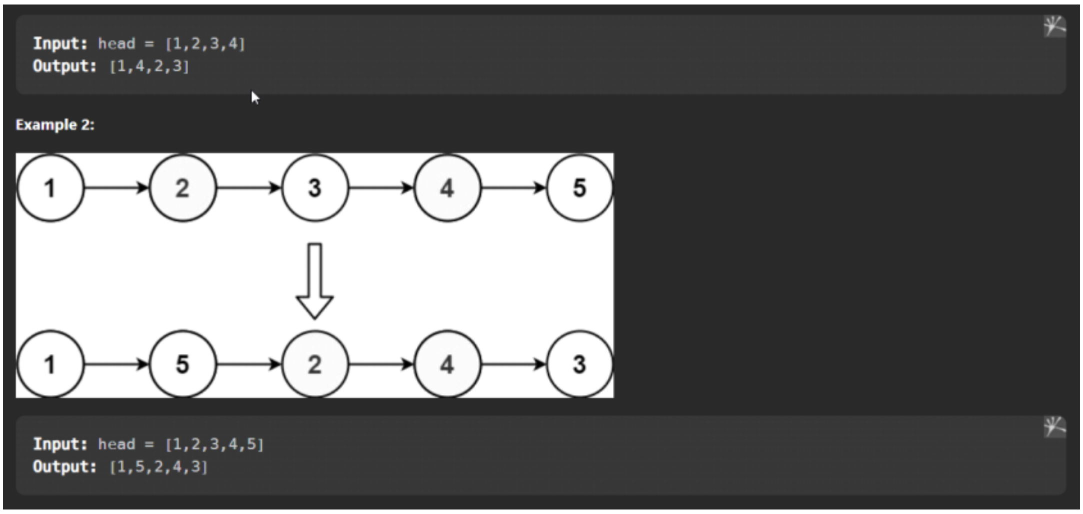
$l_0 \to l_n \to l_1 \to l_{n-1} \to l_2 \to L_{n-2}$

Reorder the list to be on the following form:

```
L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …
```

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:

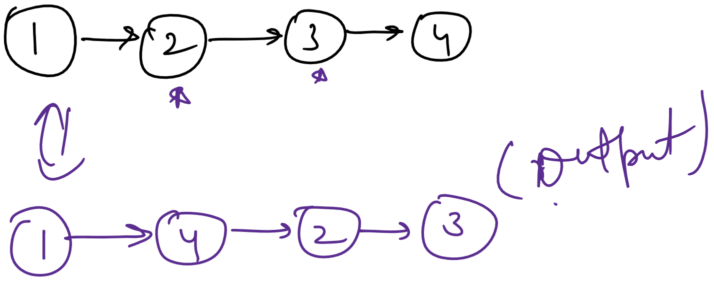


```
Input: head = [1,2,3,4]
Output: [1,4,2,3]
```

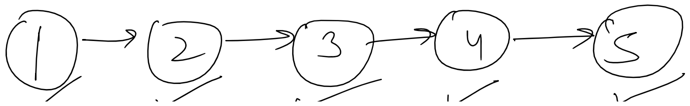
Example 2:

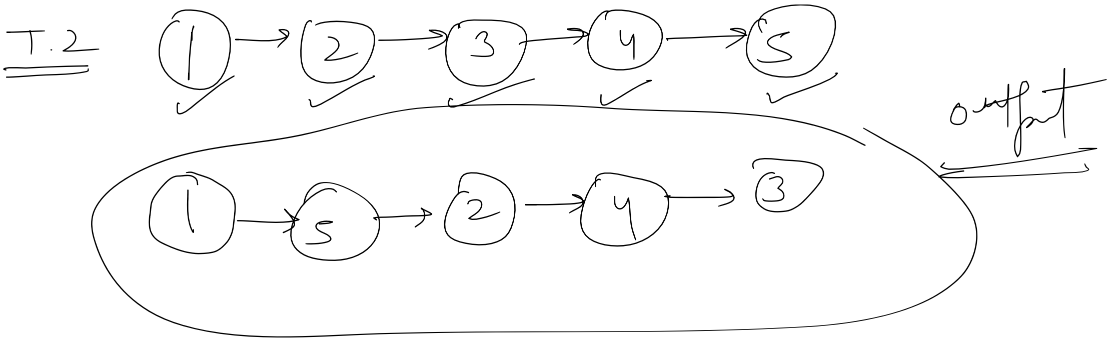
```
Input: head = [1,2,3,4,5]
Output: [1,5,2,4,3]
```

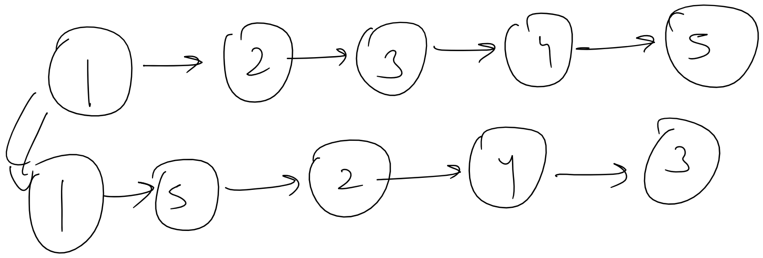
T1

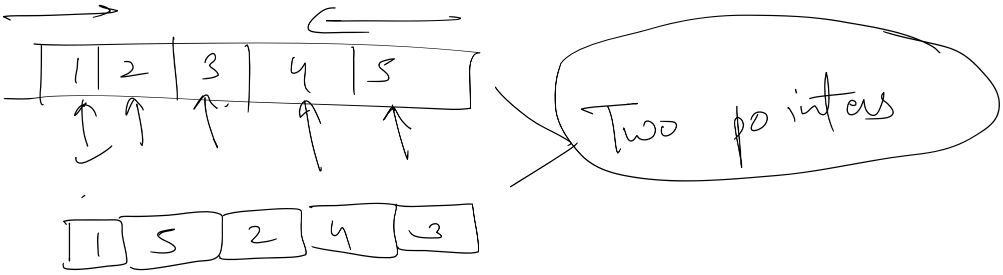


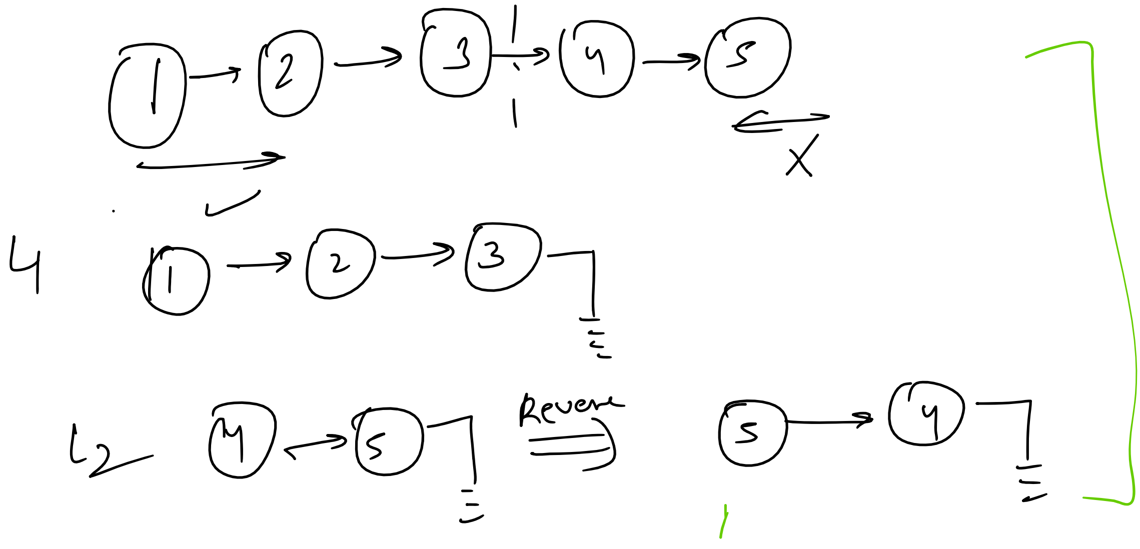
T.2

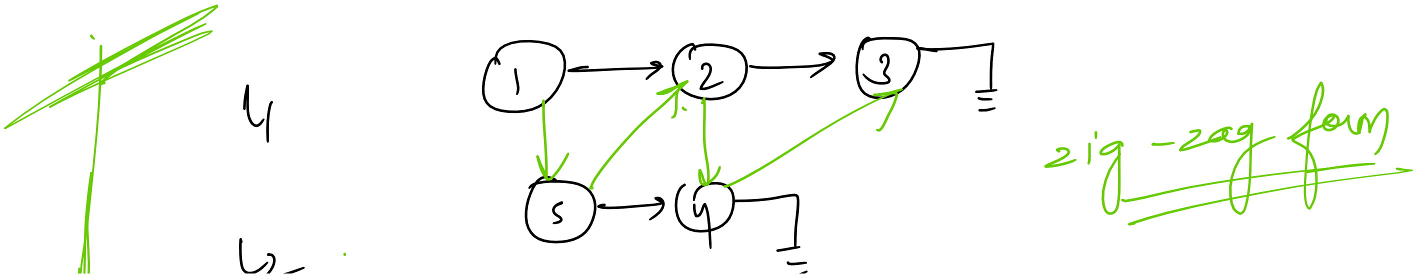


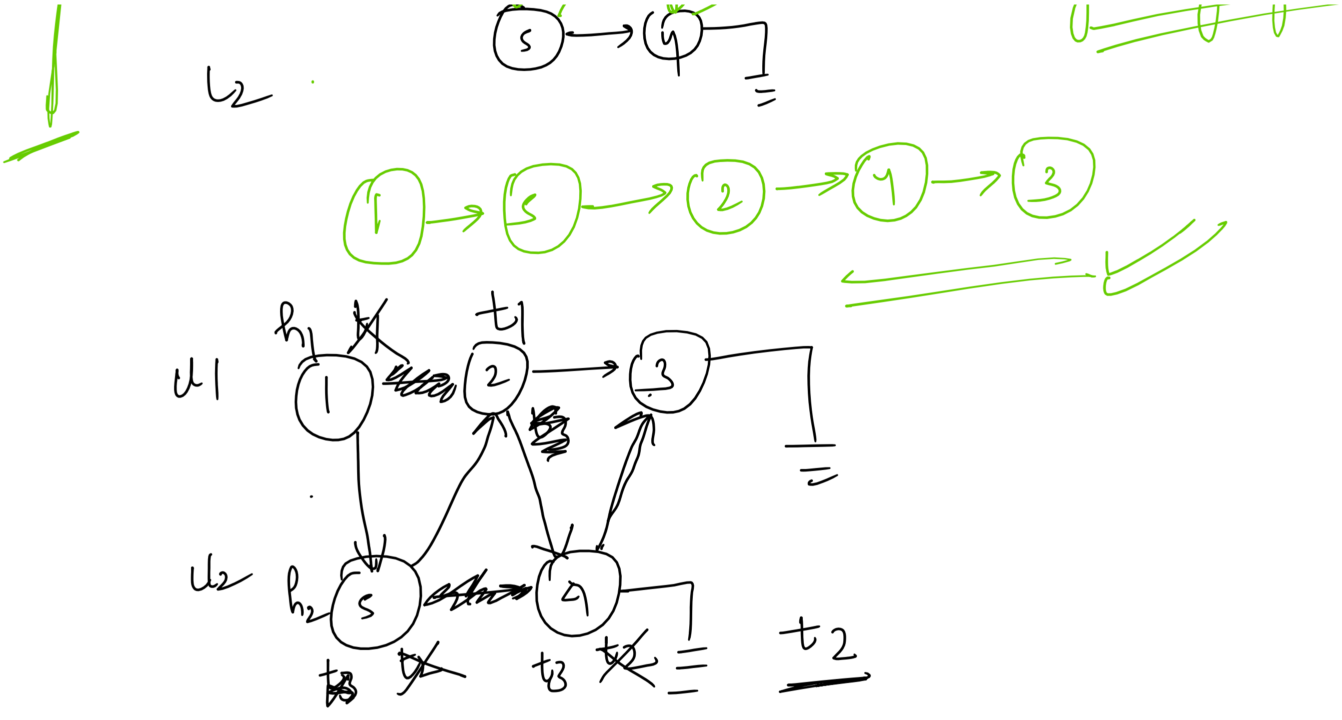

I.2
1 → 2 → 3 → 4 → 5
1 → 5 → 2 → 4 → 3
output

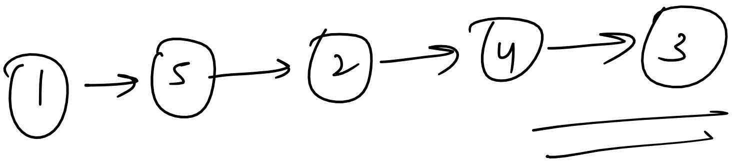

1 → 2 → 3 → 4 → 5
1 → 5 → 2 → 4 → 3

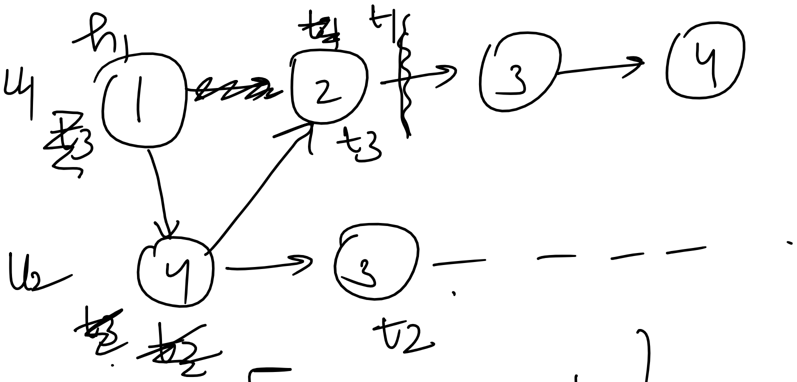

1 2 3 4 5
1 5 2 4 3
Two pointers


1 → 2 → 3 | → 4 → 5
X
L1 1 → 2 → 3
L2 4 → 5
Reverse
5 → 4


L1
L2
1 → 2 → 3
5 → 4
zig-zag form







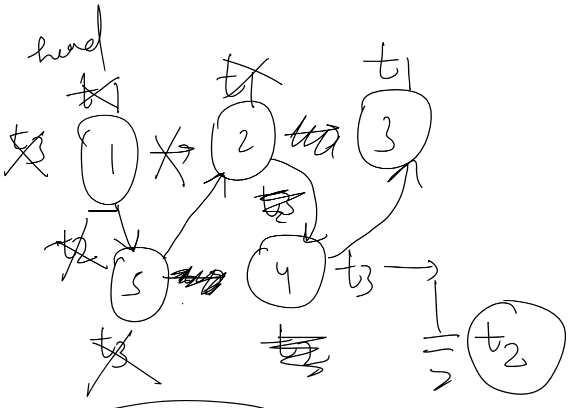
$[t_3 \rightarrow next = t_2]$

Algorithm

1. Pallindromic Linked list
   1. Middle Element
   2. Reverse Right vala part
   ~~3~~ 2i

~~③ zi~~

② zig-zag pattern

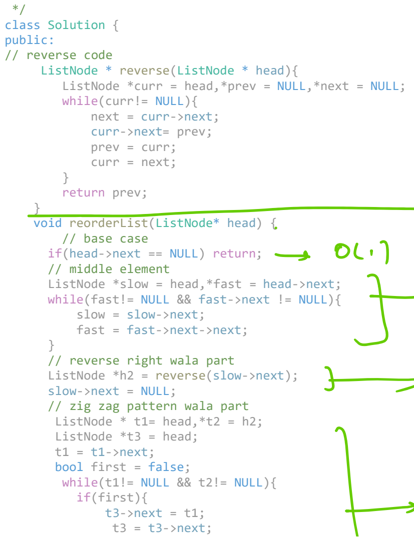


$t_2 == NULL$

$t_3 \rightarrow next = t1$

$t_1 == NULL$

$t_3 \rightarrow next = t_2$



```
 */
class Solution {
public:
// reverse code
     ListNode * reverse(ListNode * head){
        ListNode *curr = head,*prev = NULL,*next = NULL;
        while(curr!= NULL){
            next = curr->next;
            curr->next= prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
    void reorderList(ListNode* head) {
        // base case
      if(head->next == NULL) return;
      // middle element
      ListNode *slow = head,*fast = head->next;
      while(fast!= NULL && fast->next != NULL){
          slow = slow->next;
          fast = fast->next->next;
      }
      // reverse right wala part
      ListNode *h2 = reverse(slow->next);
      slow->next = NULL;
      // zig zag pattern wala part
       ListNode * t1= head,*t2 = h2;
       ListNode *t3 = head;
       t1 = t1->next;
       bool first = false;
        while(t1!= NULL && t2!= NULL){
          if(first){
              t3->next = t1;
               t3 = t3->next;
```

→ O(1)

→ O(N/2)

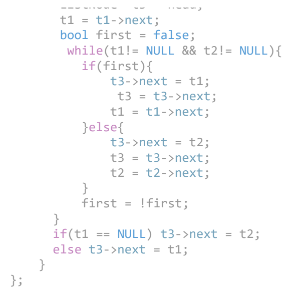
→ O(N)2)

→ O(N/2)

T.C – O(N)

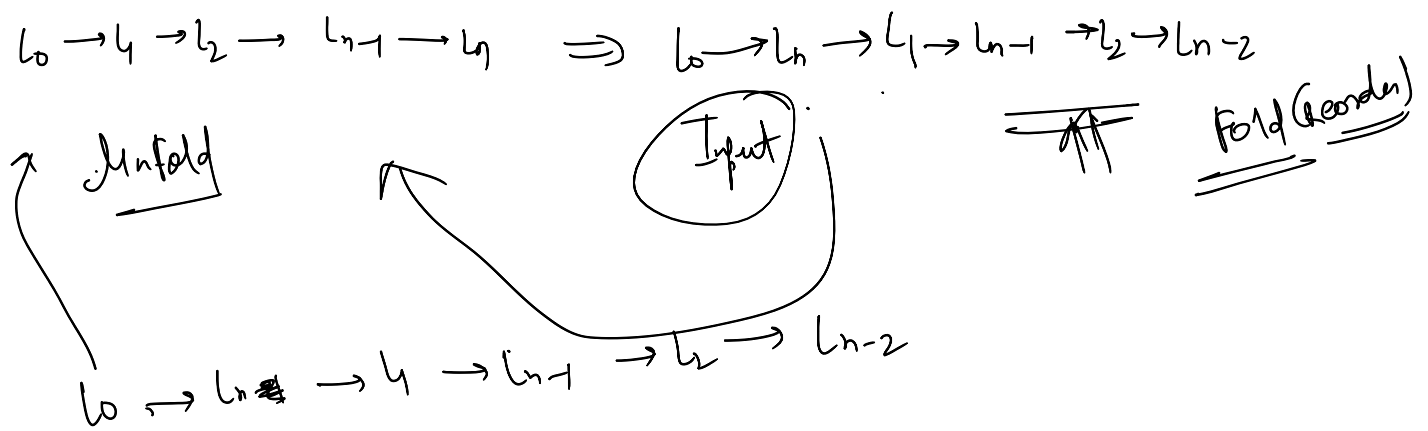
T.C – O(3N/2)

S.C – O(1)

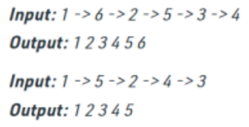


```
        t1 = t1->next;
        bool first = false;
         while(t1!= NULL && t2!= NULL){
            if(first){
                t3->next = t1;
                 t3 = t3->next;
                t1 = t1->next;
            }else{
                t3->next = t2;
                t3 = t3->next;
                t2 = t2->next;
            }
            first = !first;
        }
        if(t1 == NULL) t3->next = t2;
        else t3->next = t1;
    }
};
```

t → O(N/2)



$L_0 \rightarrow L_1 \rightarrow L_2 \rightarrow L_{n-1} \rightarrow L_n \Rightarrow L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2}$
Unfold
Input
Fold (Reorder)
$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2}$



Input: *1 -> 6 -> 2 -> 5 -> 3 -> 4*

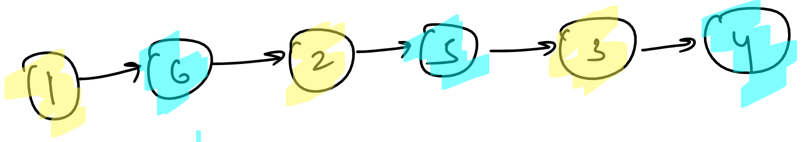
Output: *1 2 3 4 5 6*

Input: *1 -> 5 -> 2 -> 4 -> 3*

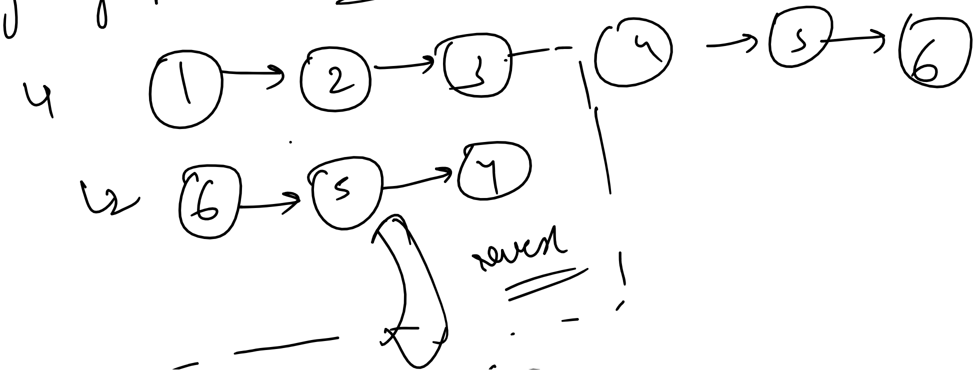
Output: *1 2 3 4 5*

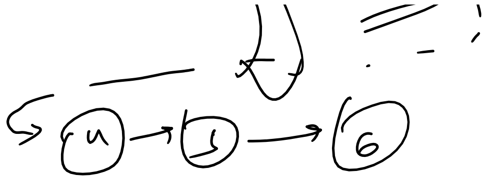
$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \Rightarrow L_0 \rightarrow L_1 \rightarrow L_2 \rightarrow L_3 \rightarrow L_n$

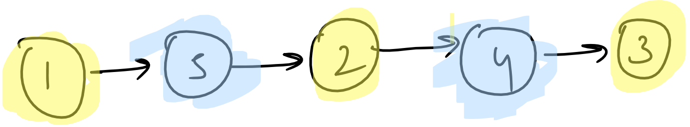
(folded)



zig-zag pattern recurr

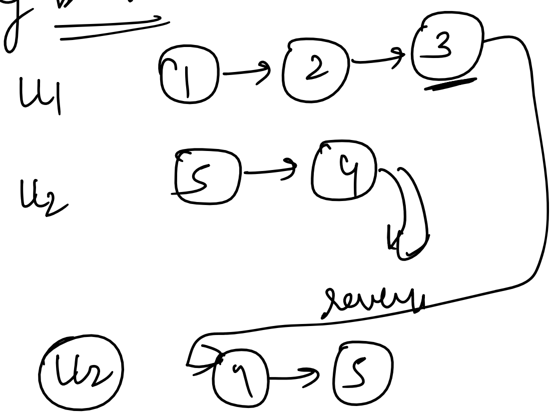




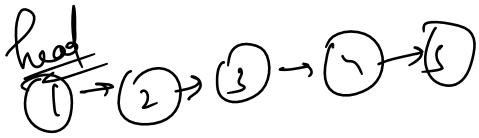


Algorithm

zig-zag break



① reverse

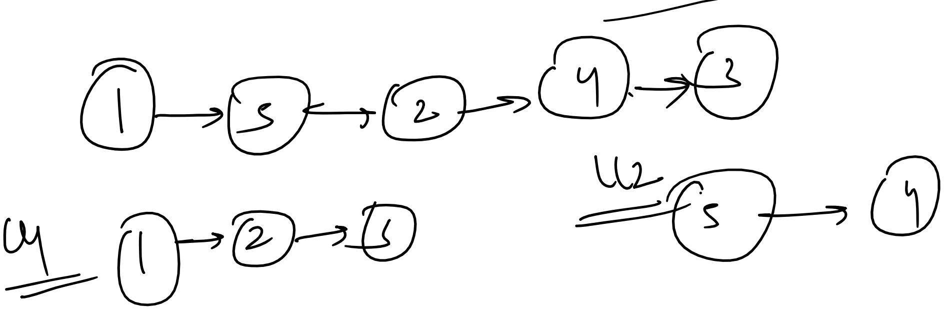


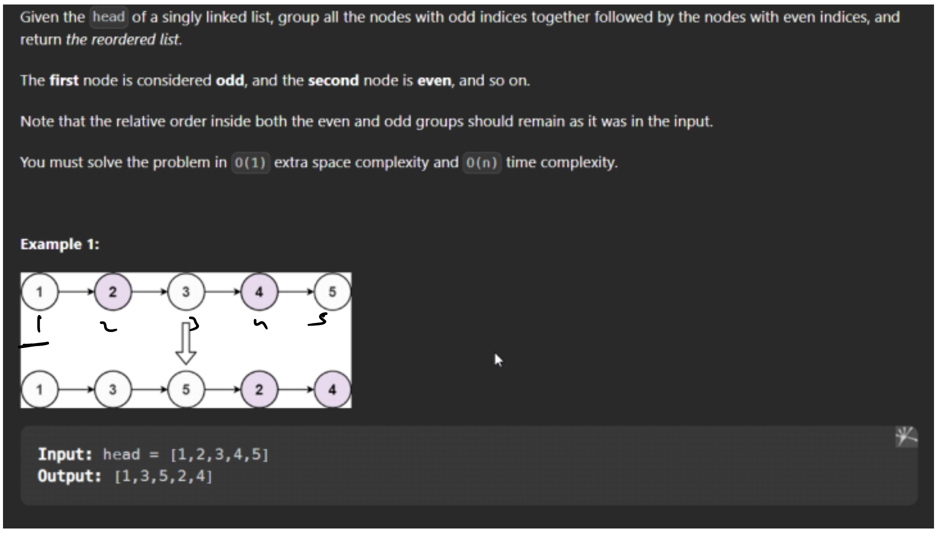
Algorithm

① Unwind zig-zag pattern ✓

② reverse l2

③ Attach l1 tail to l2 head.





Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

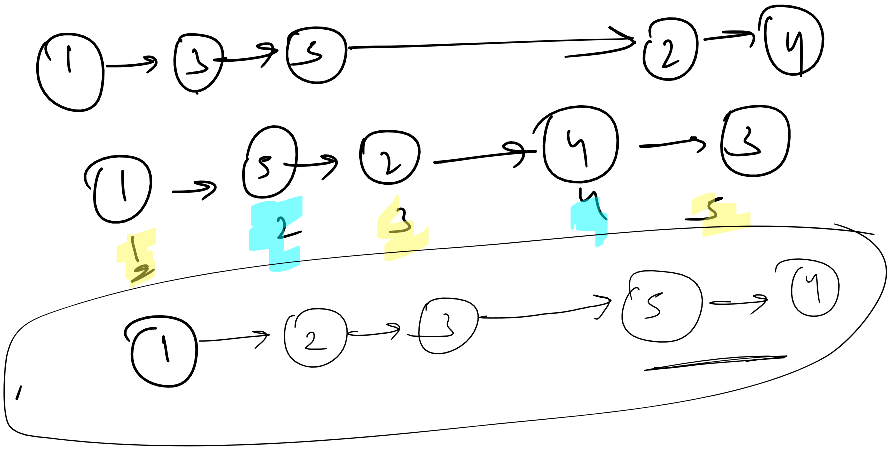
The **first** node is considered **odd**, and the **second** node is **even**, and so on.

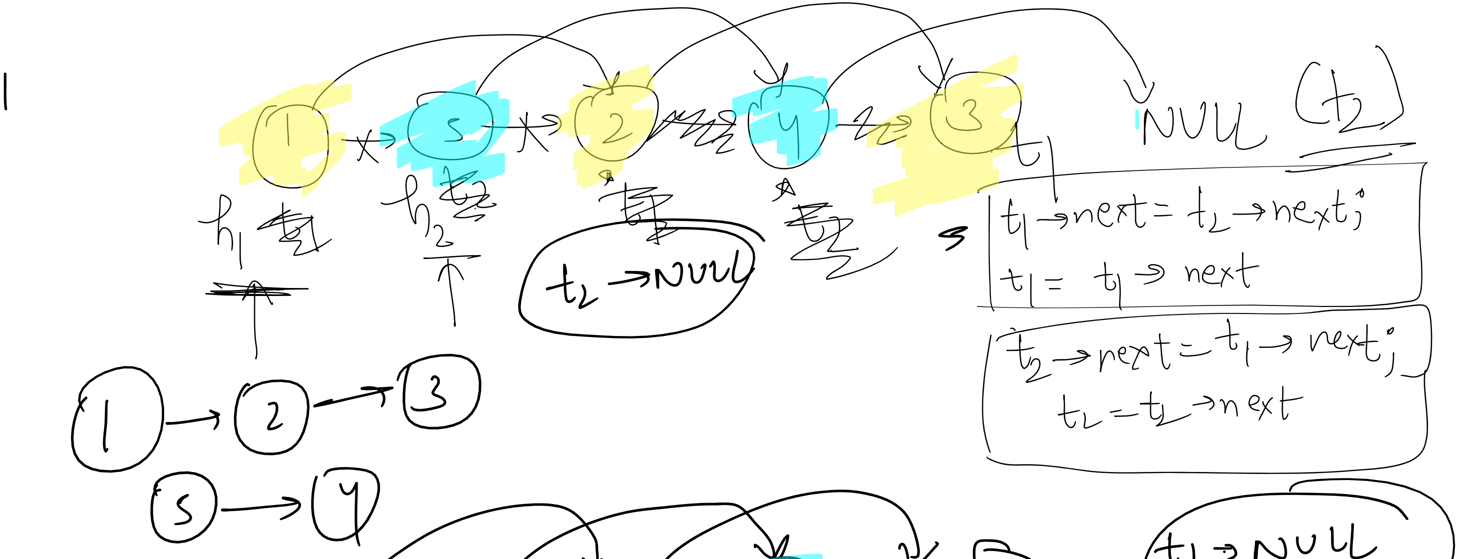
Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

Example 1:

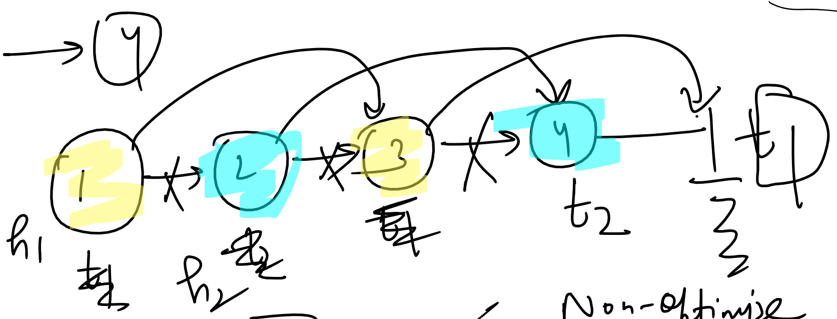
```
Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]
```





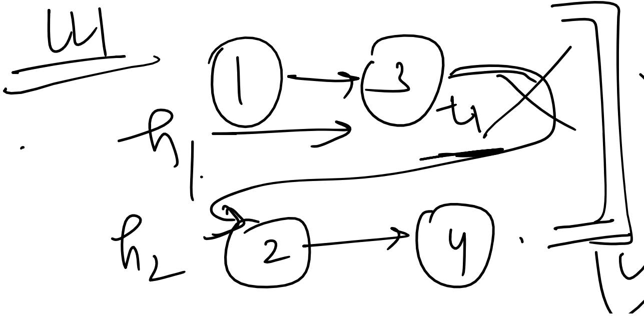
t1→next = t2→next;
t1 = t1→next

t2→next = t1→next;
t2 = t2→next

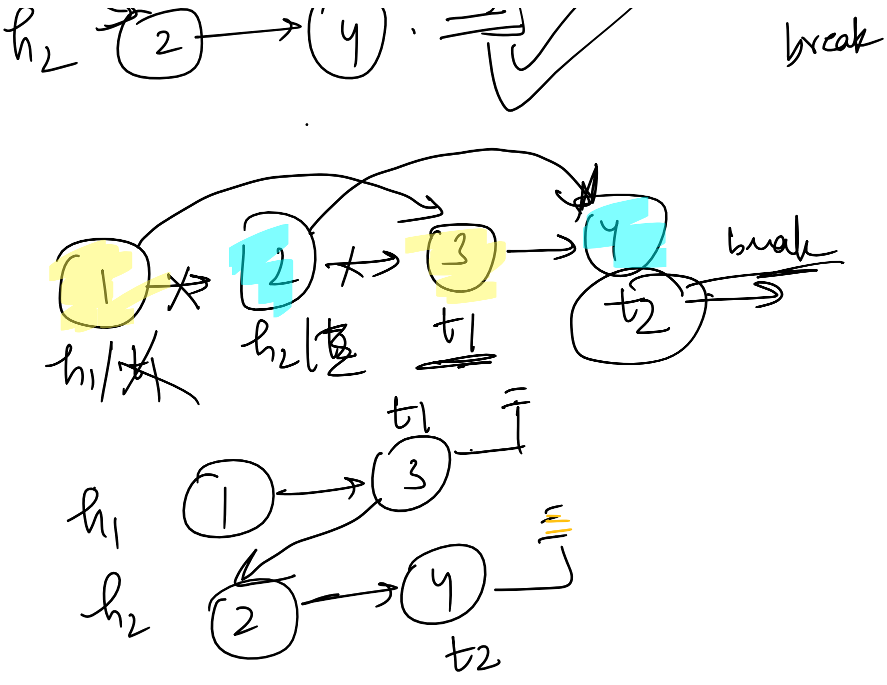


t1→NULL

Non-optimize



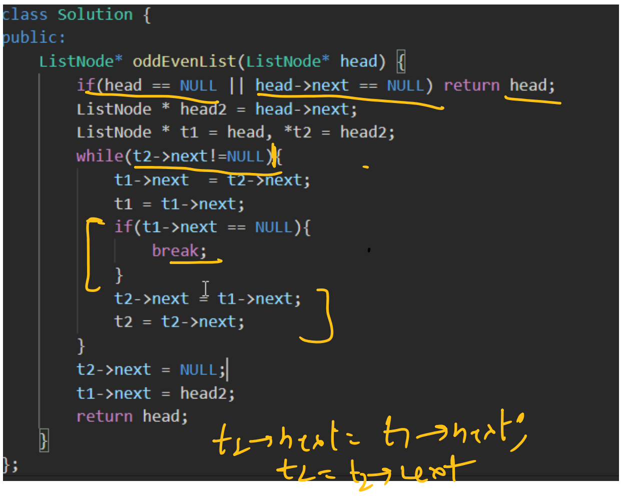
break [t1→next==NULL
t1→next==NULL



break | t1
t2->next == NULL

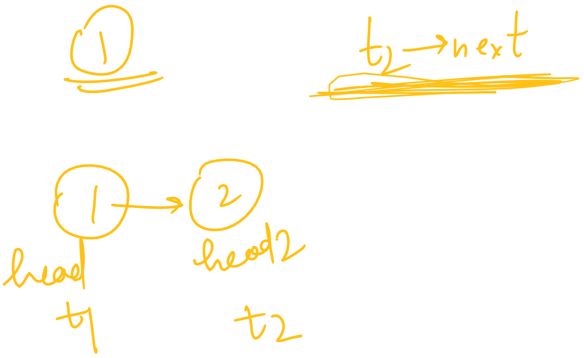
t1->next = t2->next;
t1 = t1->next

t2->next = t1->next;
t2 = t2->next



```
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if(head == NULL || head->next == NULL) return head;
        ListNode * head2 = head->next;
        ListNode * t1 = head, *t2 = head2;
        while(t2->next!=NULL){
            t1->next  = t2->next;
            t1 = t1->next;
            if(t1->next == NULL){
                break;
            }
            t2->next = t1->next;
            t2 = t2->next;
        }
        t2->next = NULL;
        t1->next = head2;
        return head;
    }
};
```

t2->next = t1->next;
t2 = t2->next



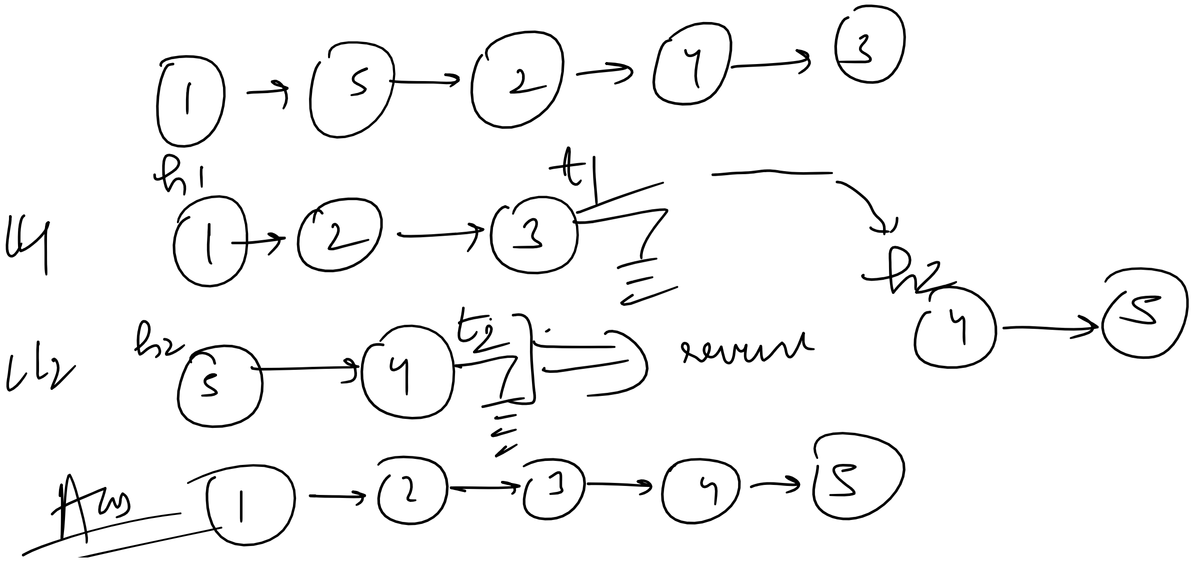
t1->next = t2->next
t1 = t1->next

~~t2->next~~

T.C - O(N)

S.C - O(1)

Unfold



Unfold

zig-zag — $O(N/2)$

rem — $O\left(\frac{N}{2}\right)$

T.C → $O(N)$

S.C — $O(1)$

1. Pallindromic LL
2. Fold (Reordn) ''
3. Unfold ( ) ''
4. odd-even pattern ''