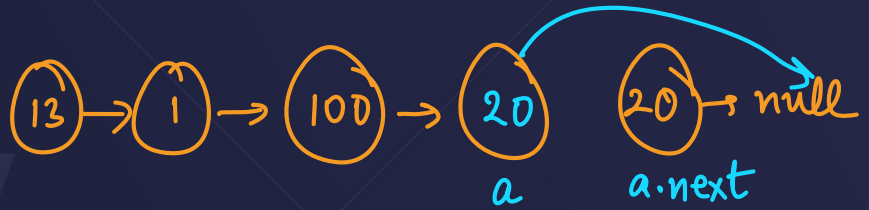
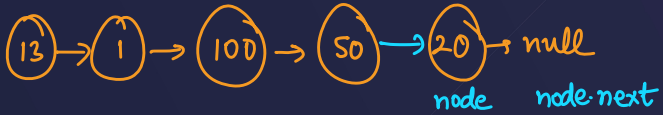
Can we delete a node given the node itself as parameter?

Is there any efficient way, provided that the given node is not the last node?



$a.data = a.next.data$

$a.next = a.next.next$

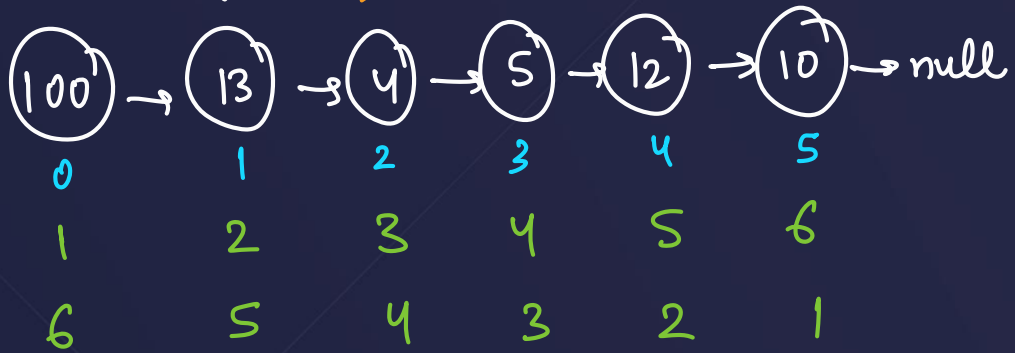


node.val = node.next.val → error

node.next = node.next.next

Finding Nth Node from the end of Linked List

[Only head is given] return node; (LeetCode)

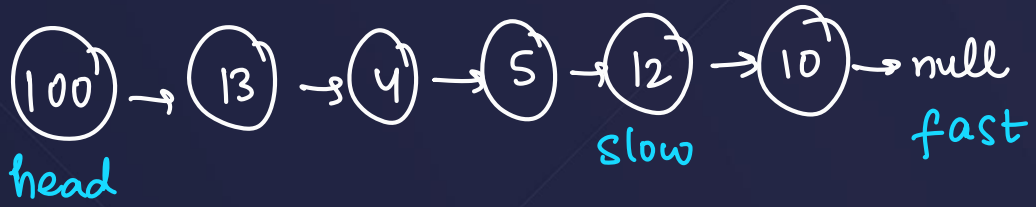


$n = 6$

n^{th} from last = $(m - n + 1)^{th}$ node from start

Finding Nth Node from the end of Linked List (In one traversal)

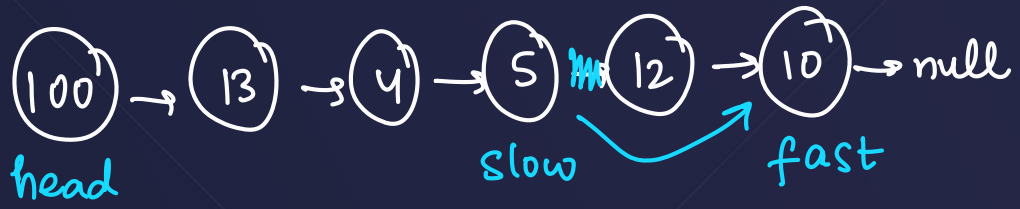
Ex : 2nd node from last




```
Node slow = head;
Node fast = head;
for(int i=1; i<=n; i++){
    fast = fast.next;
}
while(fast!=null){
    slow = slow.next;
    fast = fast.next;
}
return slow;
```

Removing Nth Node from the end of Linked List (in one traversal)

n = 2nd node from last




```
slow.next = slow.next.next;
```

Removing Nth Node from the end of Linked List

n = n^{th} node from end (where m = no. of nodes)

deleteNthFromEnd(a, 6)

Head Node

a b c d e f

100 → 13 → 4 → 5 → 12 → 10 → null

head fast

slow

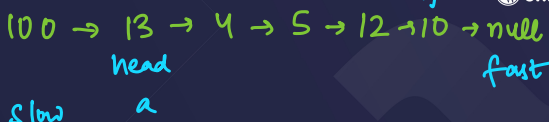
```
public static void deleteNthFromEnd(Node head, int n){
    Node slow = head;
    Node fast = head;
    for(int i=1;i<=n;i++){
        fast = fast.next;
    }
    while(fast.next!=null){
        slow = slow.next;
        fast = fast.next;
    }
    slow.next = slow.next.next;
}
```

100 6

```
if(fast == null){
    head = head.next;
    return;
}
```

```java
public static Node deleteNthFromEnd(Node head, int n){
    Node slow = head;
    Node fast = head;
    for(int i=1;i<=n;i++){
        fast = fast.next;
    }
    if(fast==null){
        head = head.next;
        return head;
    }
    while(fast.next!=null){
        slow = slow.next;
        fast = fast.next;
    }
    slow.next = slow.next.next;
    return head;
}
```

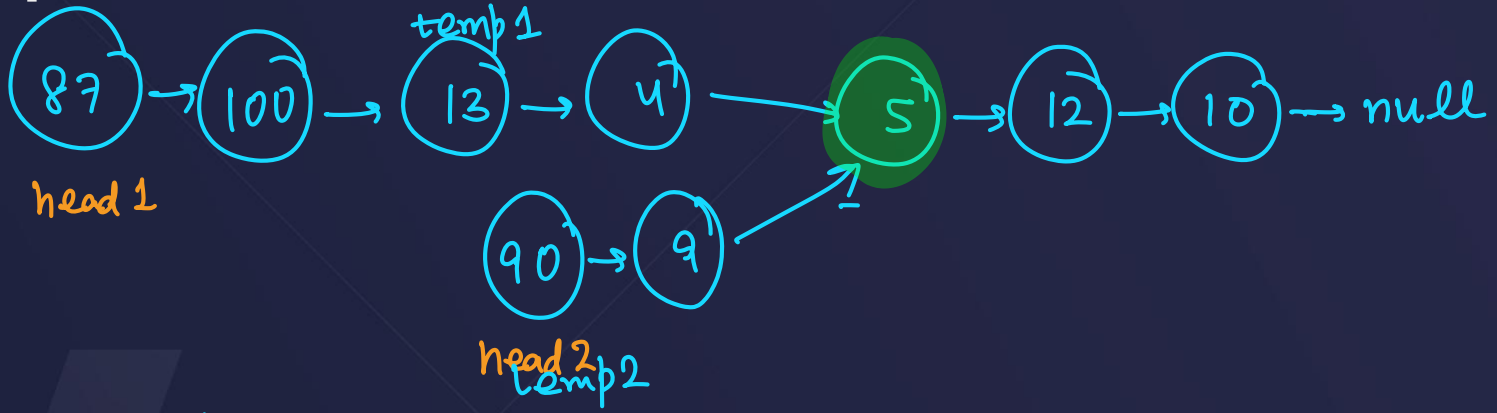
100 6




```java
display(a);
a = deleteNthFromEnd(a,6);
display(a);
```

Finding intersection of two linked lists

Two pointers to the rescue...



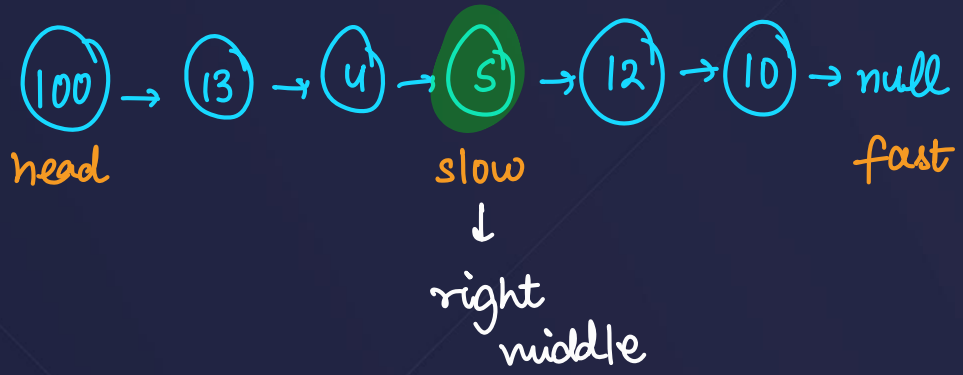
Hint : Find the lengths of both lists

Hint 2 : increment the bigger list by (m - n) steps

m → length of big list

n → length of small list

Finding middle element of a linked list

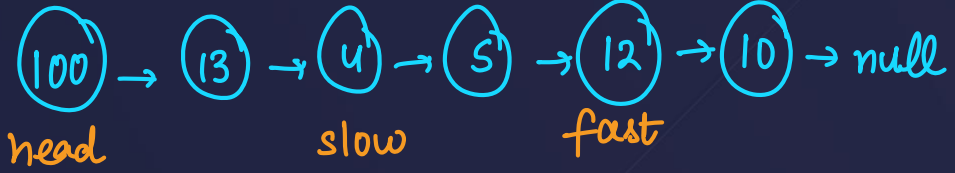


Hint: 1) slow & fast

2) fast → 2x

```
while (fast != null) {
    slow = slow.next;
    fast = fast.next.next;
}
```

Finding middle element of a linked list

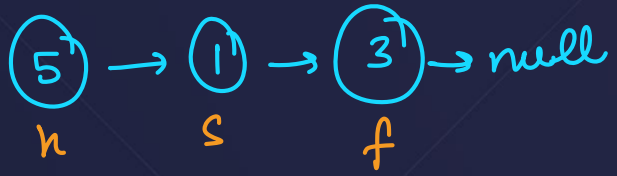


For Even no. of nodes → fast.next.next != null

↓

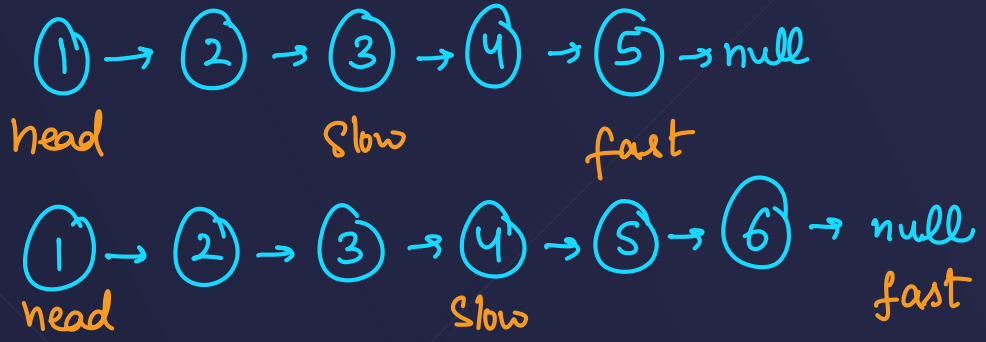
△ left middle

Finding middle element of a linked list



For odd number of nodes → fast.next != null

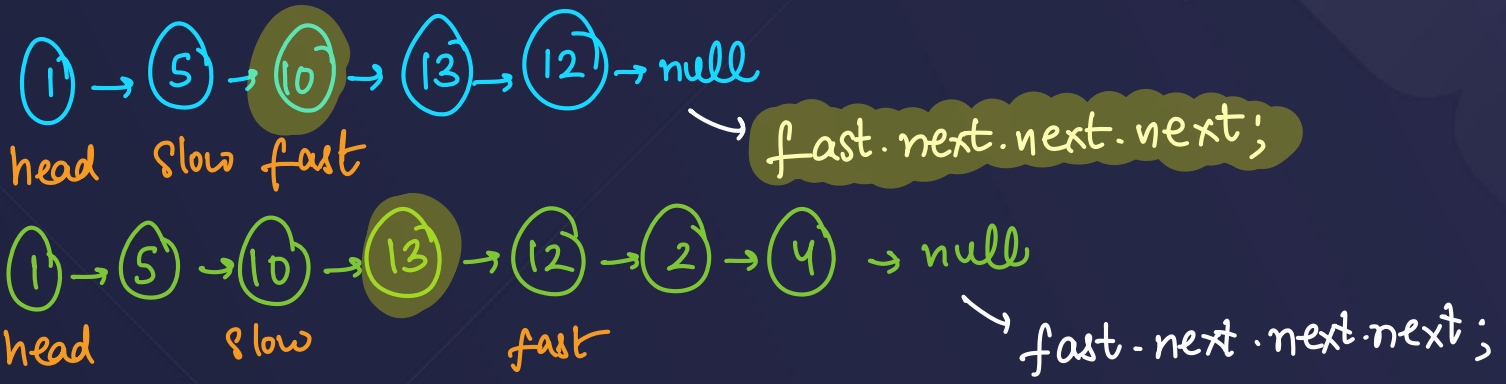
Finding middle element of a linked list




```
while (fast.next!=null && fast!=null){
    slow = slow.next;
    fast = fast.next.next;
}
```

Deleting the middle element of the linked list

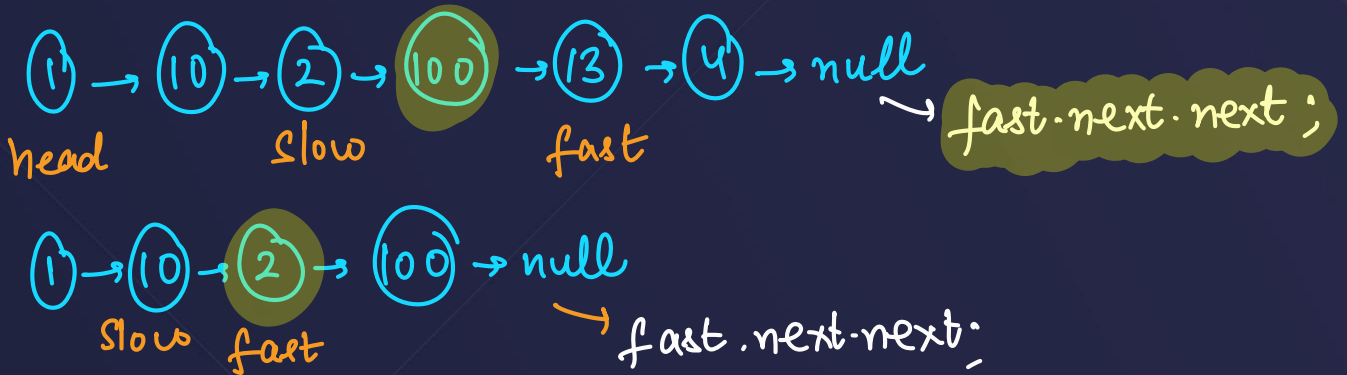
Odd :




```
while (                    ){
    slow = slow.next;
    fast = fast.next.next;
}
```

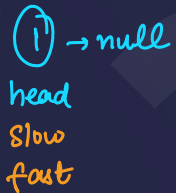
Deleting the middle element of the linked list

Even → Right middle has to be deleted

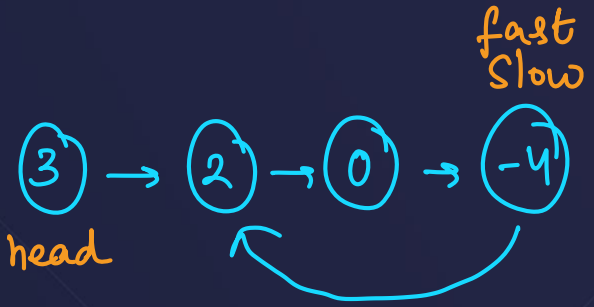



```
while (          ) {
|     slow = slow.next;
|     fast = fast.next.next;
}
```

```java
public ListNode deleteMiddle(ListNode head) {
    ListNode slow = head;
    ListNode fast = head;
    while(fast.next.next!=null && fast.next.next.next!=null){
        slow = slow.next;
        fast = fast.next.next;
    }
    // 1 2 3 4 5
    // slow -> 2, 3-> slow.next, 4 -> slow.next.next
    slow.next = slow.next.next;
    return head;
}
```

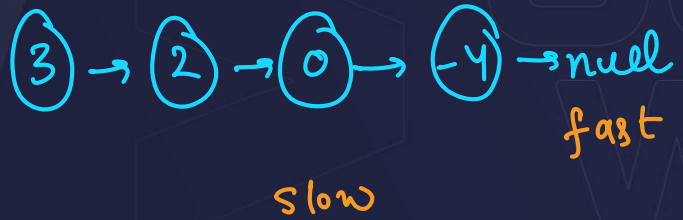


Cycle in a linked list

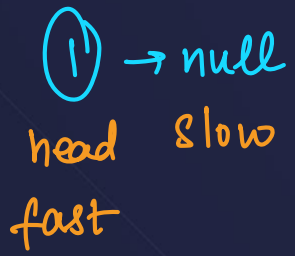



```
while (fast != null) {
    slow = slow.next;
    fast = fast.next.next;
    if (fast == slow) return true;
}
return false;
```

Hint-1 : Use slow & fast



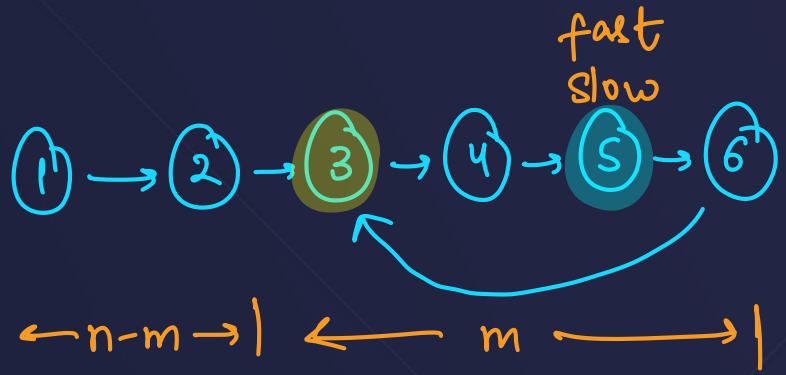
Cycle in a linked list




```
ListNode slow = head;
ListNode fast = head;
while(fast!=null){
    slow = slow.next;
    fast = fast.next.next;
    if(fast==slow) return true;
}
return false;
```

Cycle in a linked list

Proof



Nodes = n

Cycle length = m

$m \leq n$

Slow → 1

fast → 2

s: $n - m + x$

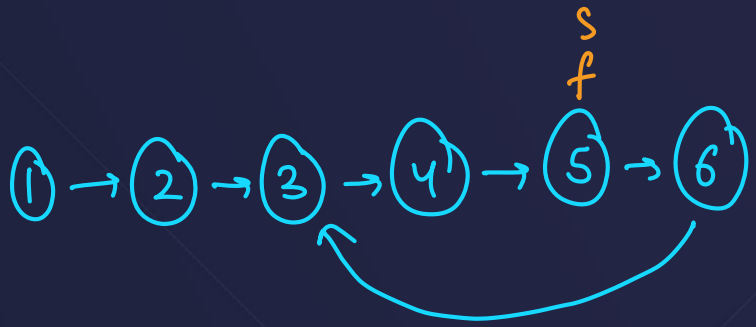
f: $n + x$

→ $n + x = 2[n - m + x]$

→ $n + x = 2n - 2m + 2x$

⇒ $2m - n = x$

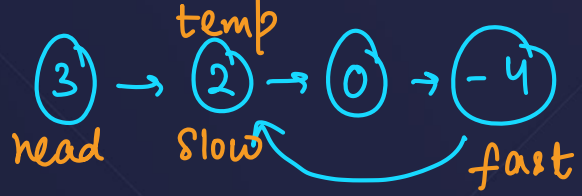
Cycle in a linked list



s → 1
f → 3

Practice → Interview

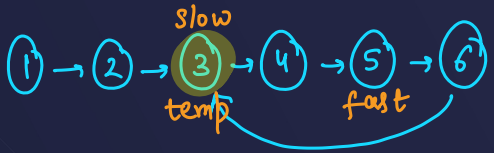
Find out the node where the cycle begins, i.e. the node at which the tail node points.



Cycle Detection + Number of nodes in cycle

Cycle Detection + Interesting Observation

H.W. leetcode → LL Cycle 2 ✓




```
1  while(fast!=null){
2  |   slow = slow.next;
3  |   fast = fast.next.next;
4  |   if(fast==slow) break;
5  }
6  Node temp = head;
```

```
7  while(temp!=slow){
8  |       temp = temp.next;
9  |       slow = slow.next;
10 }
11 return slow;
```

What do you think?

Which of the following algorithms is not feasible to implement in a linked list?

(a) Linear Search
(b) Merge Sort
(c) Insertion Sort
(d) Binary Search

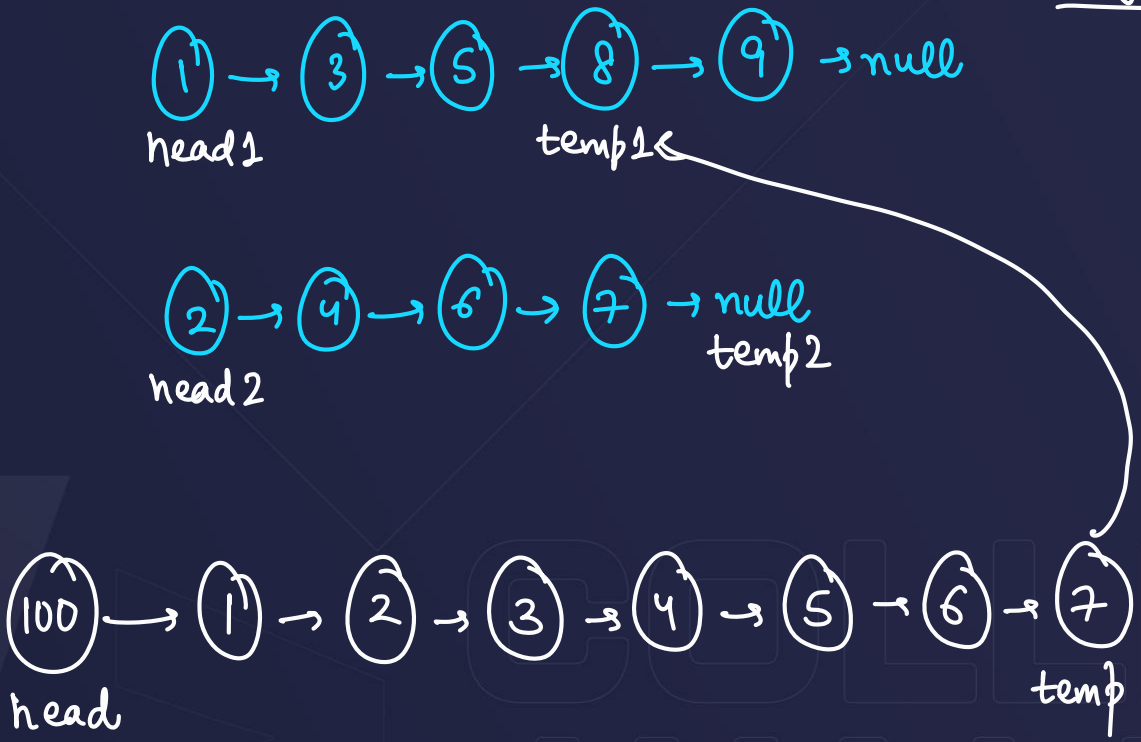
What do you think?

Which of the following algorithms is not feasible to implement in a linked list?

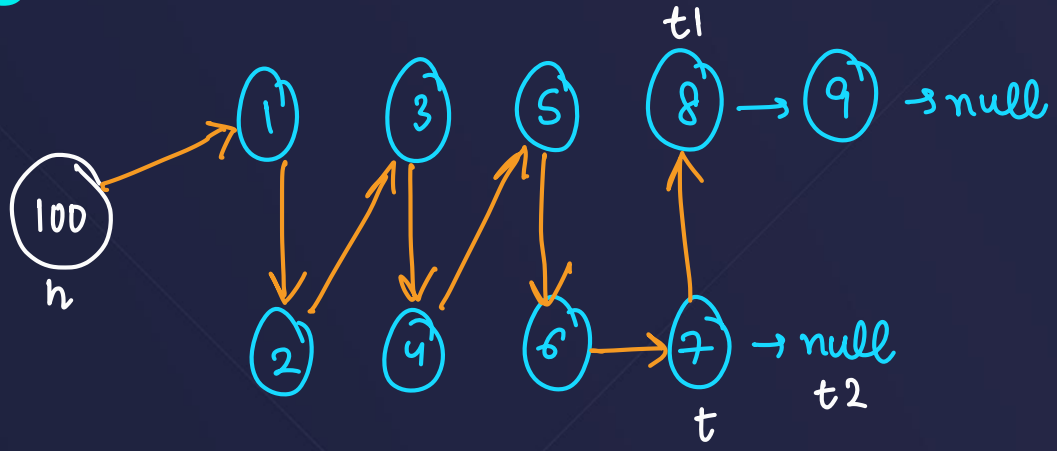
(a) Linear Search
(b) Merge Sort
(c) Insertion Sort
(d) **Binary Search**

Merge two sorted linked lists

Using Extra Space



Merge two sorted linked lists (Without Extra Space)




```
t.next = t1
t = t1
t1 = t1.next
```

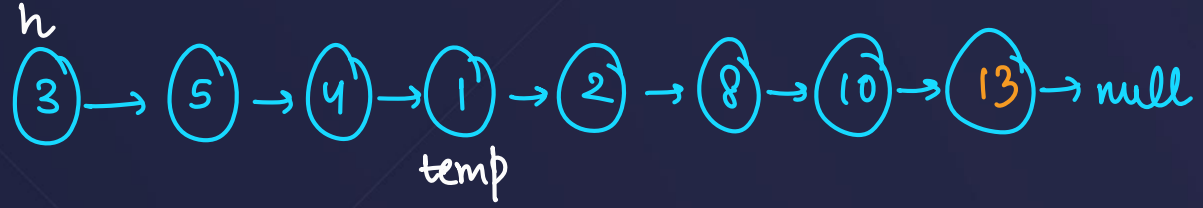
```
Node h = new Node(100);
Node t = h;
```

```
while (t1 != null && t2 != null){
}
if (t2 == null){
    t.next = t1;
```

h2 h1

Practice (H.W. Consider)

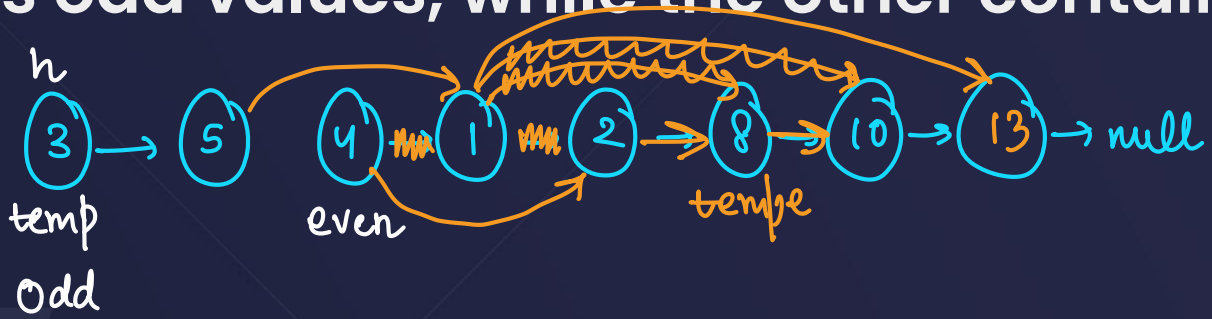
Given a linked list, split it into two lists such that one contains odd values, while the other contains even values.



Practice

Homework

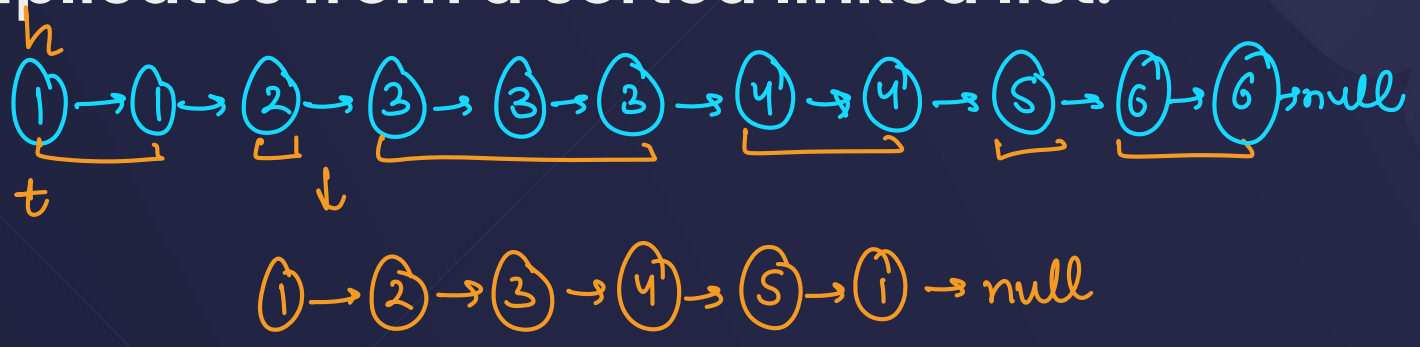
Given a linked list, split it into two lists such that one contains odd values, while the other contains even values.



Practice

Homework

Remove duplicates from a sorted linked list.



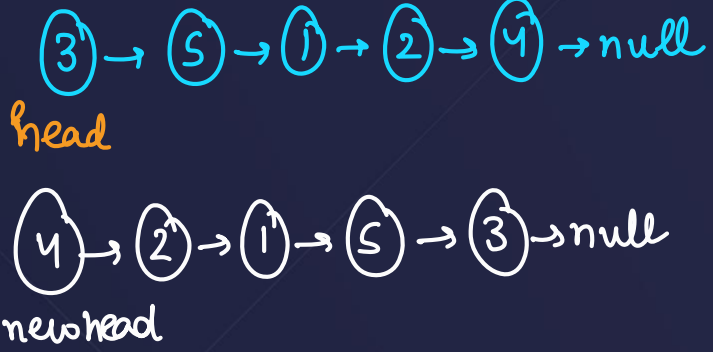

```
if (temp.next.val == temp.val)
```

Recursion for Reversal

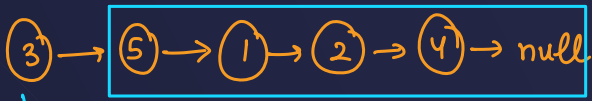
O(n) → TC

S.C. → O(n)

Reverse a linked list and return its new head.



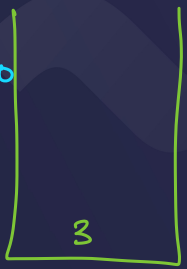
display( ) , revdisplay(Node head);



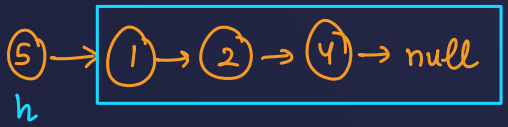
1) head.next
isko rev karlo

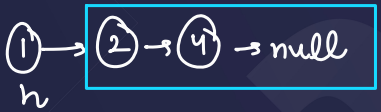
2) head, head.next
interchange,
2.5) null

3) return the
new head



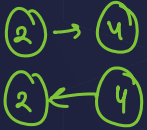
2) h.next.next = h





Base case

if (h.next == null) return h;



a = reverse(a);

```java
public static Node reverse(Node head){
    if(head.next==null) return head;
    Node newHead = reverse(head.next);
    head.next.next = head; // interchan
    head.next = null;
    return newHead;
}
```


```java
public static Node reverse(Node head){
    if(head.next==null) return head;
    Node newHead = reverse(head.next);
    head.next.next = head; // interchan
    head.next = null;
    return newHead;
}
```

```java
public static Node reverse(Node head){
    if(head.next==null) return head;
    Node newHead = reverse(head.next);
    head.next.next = head; // interchan
    head.next = null;
    return newHead;
}
```

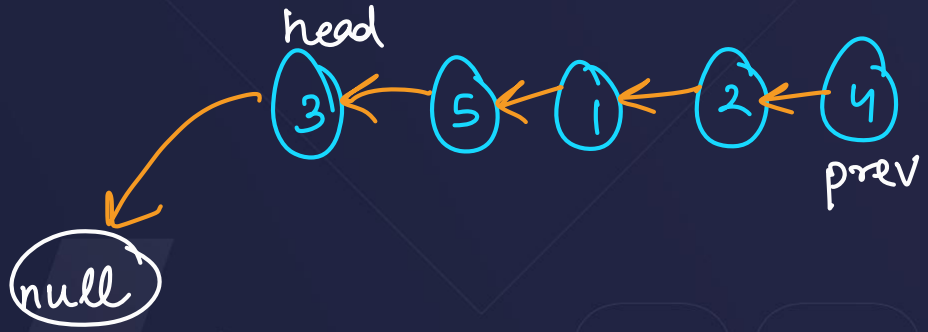
```java
public static Node reverse(Node head){
    if(head.next==null) return head;
    Node newHead = reverse(head.next);
    head.next.next = head; // interchan
    head.next = null;
    return newHead;
}
```

```java
public static Node reverse(Node head){
    if(head.next==null) return head;
    Node newHead = reverse(head.next);
    head.next.next = head; // interchan
    head.next = null;
    return newHead;
}
```

Can we reverse a linked list without recursion?

prev, curr, agla

Are two pointers enough?...



1) agla = curr.next

2) curr.next = prev

3) prev = curr

4) curr = agla

prev curr agla

Palindrome Linked List

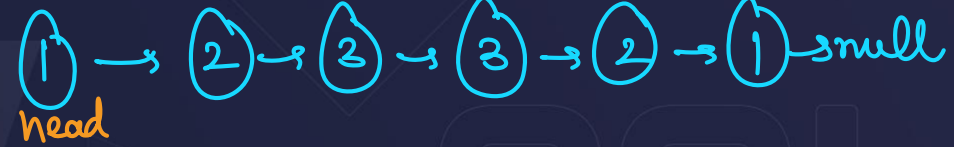
Check whether a linked list is palindrome or not?

public static boolean isPalindrome(Node head){

M-I

}

List



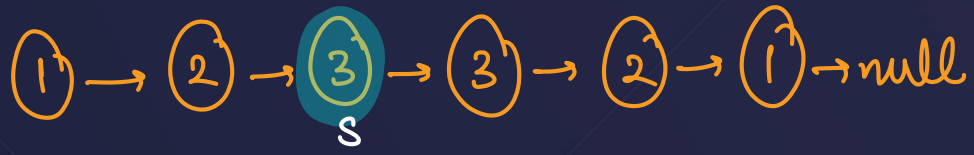
Duplicate

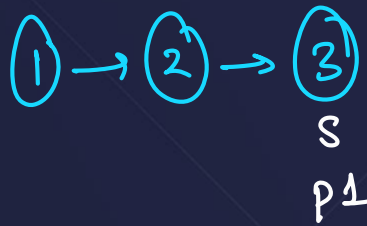
Copy (Deep)

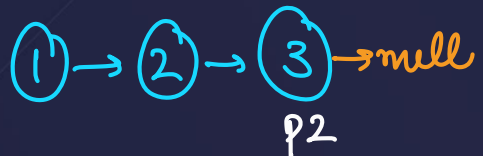
Palindrome Linked List

Check whether a linked list is palindrome or not?

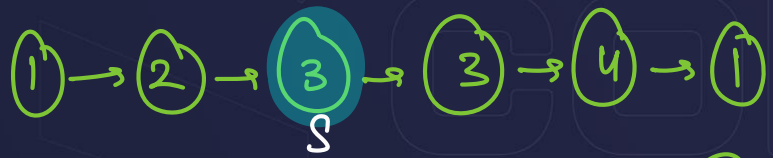
Step 1 :

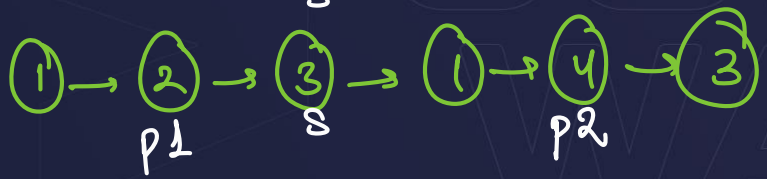






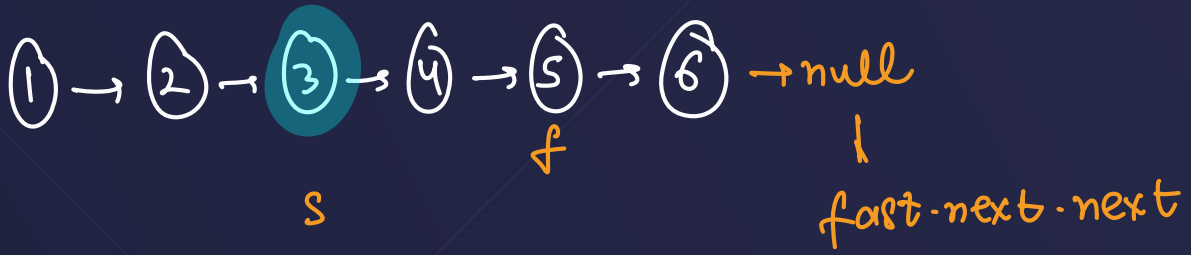
1) Left Middle
2) Right half ko reverse
3) p1, p2

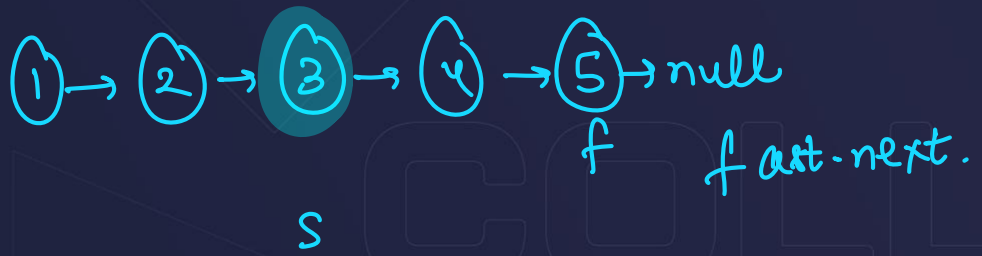




Palindrome Linked List

Check whether a linked list is palindrome or not?

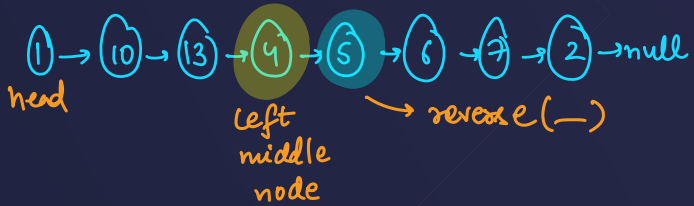


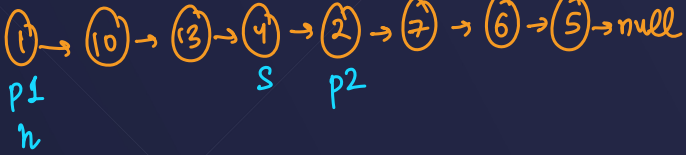


Practice [Leetcode → 2130]

Find the maximum twin sum of a linked list of even length.

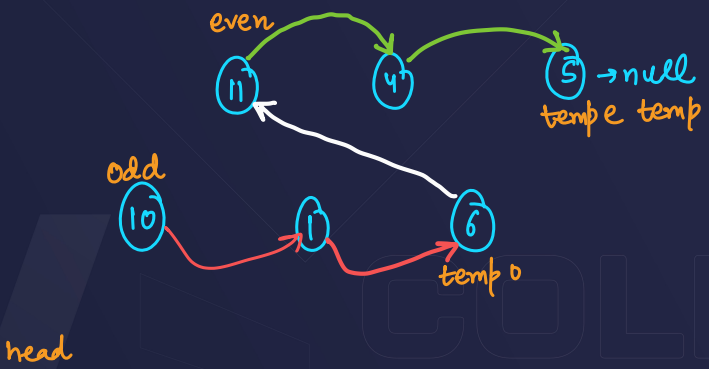
Twin of any node at $(i)^{th}$ index is the node at $(n-i-1)^{th}$ index. Twin sum is the sum of values of a node and its twin.

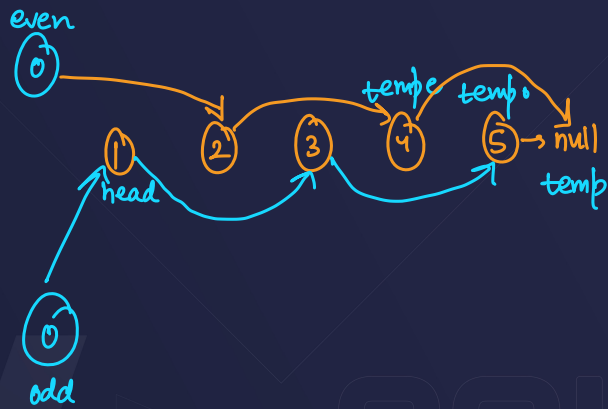





```
int max = Integer.MIN_VALUE;
```

LeetCode 328 : Odd Even Linked List






```
while(temp!=null){
    tempo.next = temp;
    temp = temp.next;
    tempo = tempo.next;

    tempe.next = temp;
    temp = temp.next;
    tempe = tempe.next;
}
```

Practice

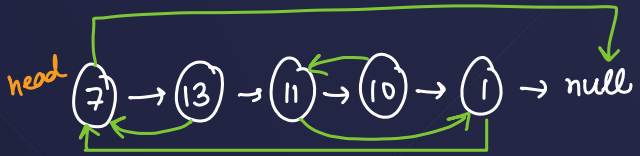
Copy List with Random Pointer: Construct a deep copy of a linked list where each node contains an additional random pointer, which could point to any node in the list or null.

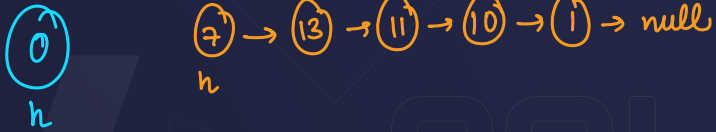
An interesting question!

Step - 1 : Deep Copy

Step 2 : Connect the lists

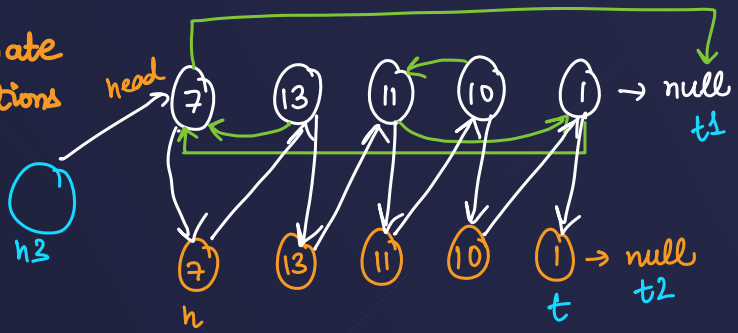
Step-1 Deep Copy → next





Step-2

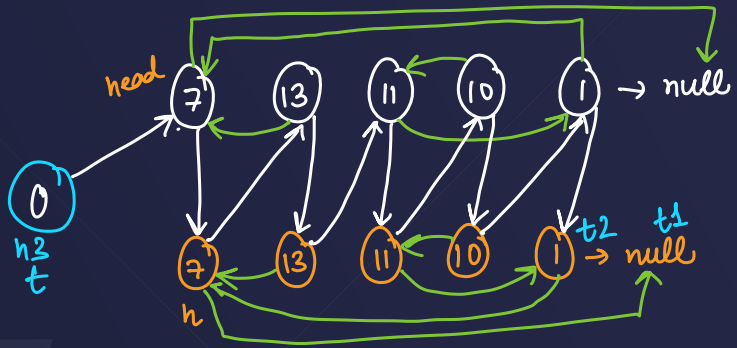
Alternate Connections




```
t.next = t1
t1 = t1.next
t = t.next
```

```
t.next = t2
t2 = t2.next
t = t.next
```

Step-3 Assigning Random Pointers



t = h3

if (t1.random = null) t2.random = null

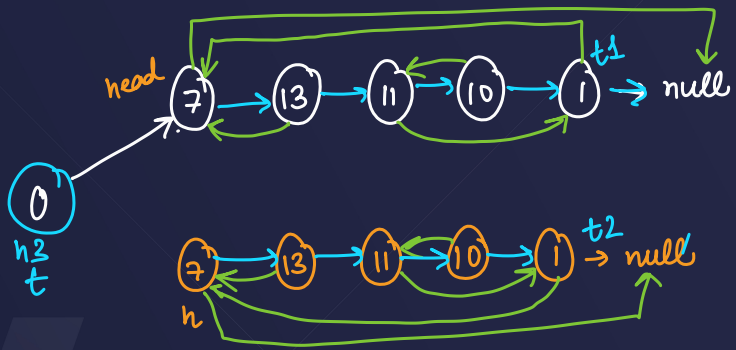
t2.random = t1.random.next.

while

→ t1 = t1.next.next;

→ t2 = t2.next.next;

Step-4 Seperating the lists




```
t1.next = t2.next;
t1 = t1.next
t2.next = t1.next
t2 = t2.next;
```

```
Node head2 = new Node(0);
Node temp2 = head2;
Node temp1 = head;
// creating deep copy -  7 13 11 10 1
while(temp1!=null){
    Node t = new Node(temp1.val);
    temp2.next = t;
    temp2 = t;
    temp1 = temp1.next;
}
```