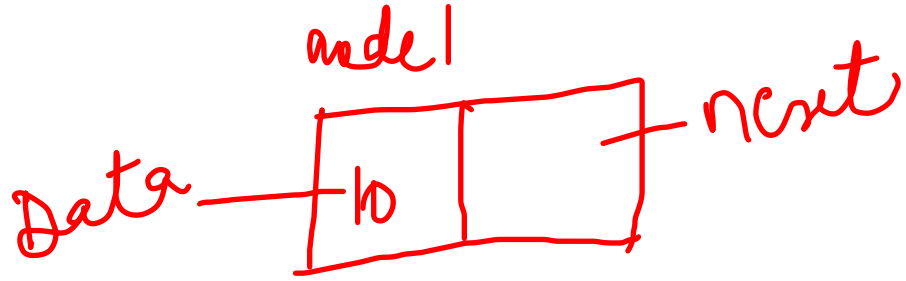
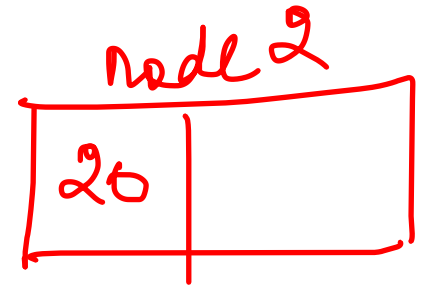
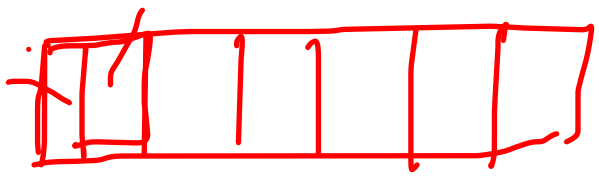
Linked List







generic type

```
public class node<T>{

    int data;

    Node<T> next;
```

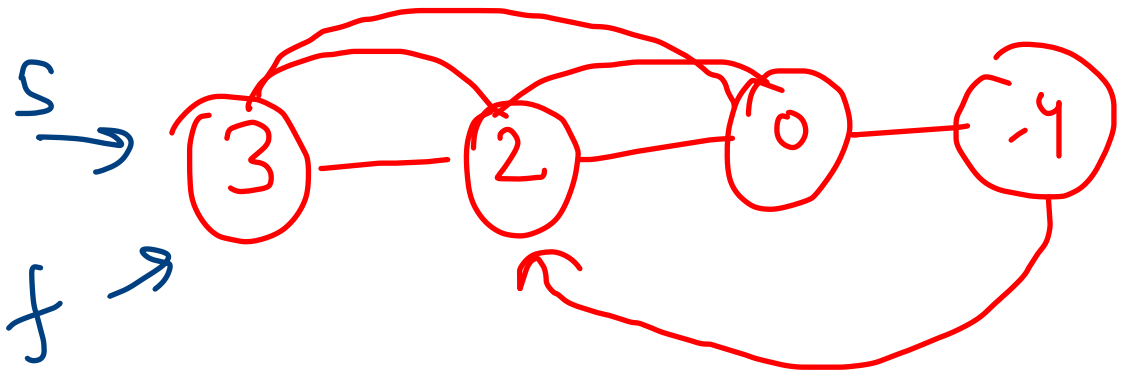
```
node(T data){
    this.data = data;
    next = null;
}
```

```java
import java.util.LinkedList;
 class node<T>{
    T data;
    node<T> next;

    node(T data){
        this.data = data;
        next = null;
    }
}

class Main {
    public static void main(String[] args) {
       node<Integer> node1 = new node<Integer>(10);
       System.out.println(node1.data);
       System.out.println(node1.next);

       node<Integer> node2 = new node<Integer>(20);
       node1.next = node2;
       System.out.println(node2);
       System.out.println(node1.next);
       System.out.println(node2.data);
       System.out.println(node2.next);
    }
}
```

Output

```
10
null
node@2a139a55
node@2a139a55
20
null
```

141. Linked List Cycle



s	f
3	3
2	0
0	2
-4	-4

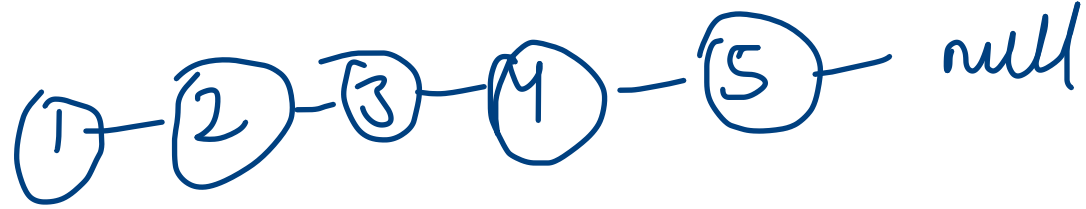
true

slow = slow.next

fast = fast.next.next

slow == fast

-4 == -4



S	f
1	1
2	3
3	5
4	null

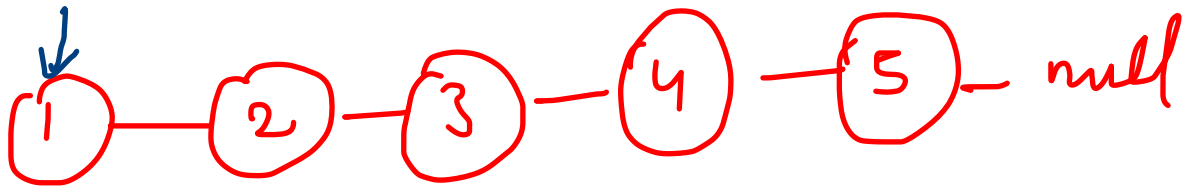
false

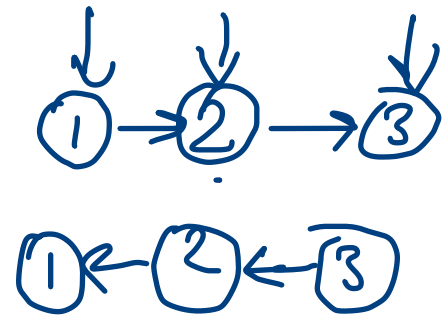
```java
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;

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

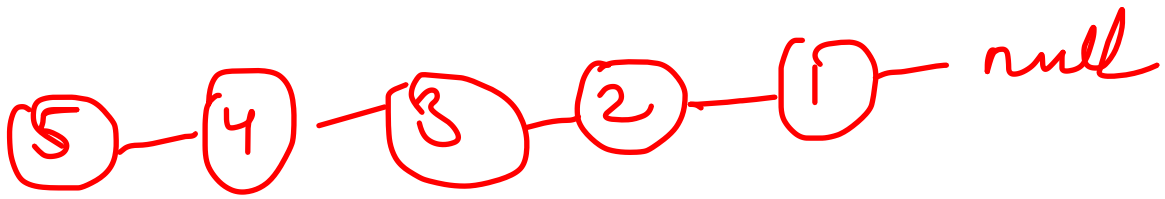
TC → O(n)

SC → O(1)

Reverse LL







Iterative Approach

3 pointers

1. prev - represent the previous node (initially NULL)
2. curr - current node (starts from head)
3. next - temporary stores the value of next (curr.next)

prev – null

curr – 1 (head)

1-2-3-4→5-null

1-2

Iteration 1

1.1 next = curr.next ⇒ 2

1.2 Reverse the link ⇒ curr.next = prev ⇒ 1 → null

1.3 move prev = 1

1.4 Move curr = 2

prev = 1 → null

curr = 2

I2

next = 3

curr.next = prev = 2 → 1 → null

prev = 2

curr = 3

2 → 1 → null

prev = null

curr = 1

next = 2 (store curr.next)

next = curr.next // next = 2

curr.next = prev ; // curr.next = null

prev = null , curr.next = null

1 → null (1 originally pointing to 2, we changed to null)

next = curr.next → 2

curr.next = prev → null

prev = curr → 1

curr = next → 2

2 - 1 - null

(2) 1-2-3-4-(5)

prev = 2→1→null

curr = 3

I3 next = curr.next = 4

curr.next = prev; // 3-2-1-null

prev = 3

curr = 4

I4) next = 5

curr.next = prev. // 4-3-2-1-null

prev = 4

curr = 5

I5) next = null

curr.next = prev;

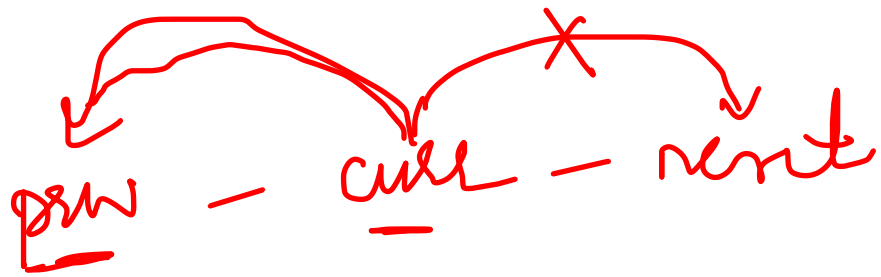
// 5-4-3-2-1-null

prev = 5

curr = null

(loop exit)

return prev;

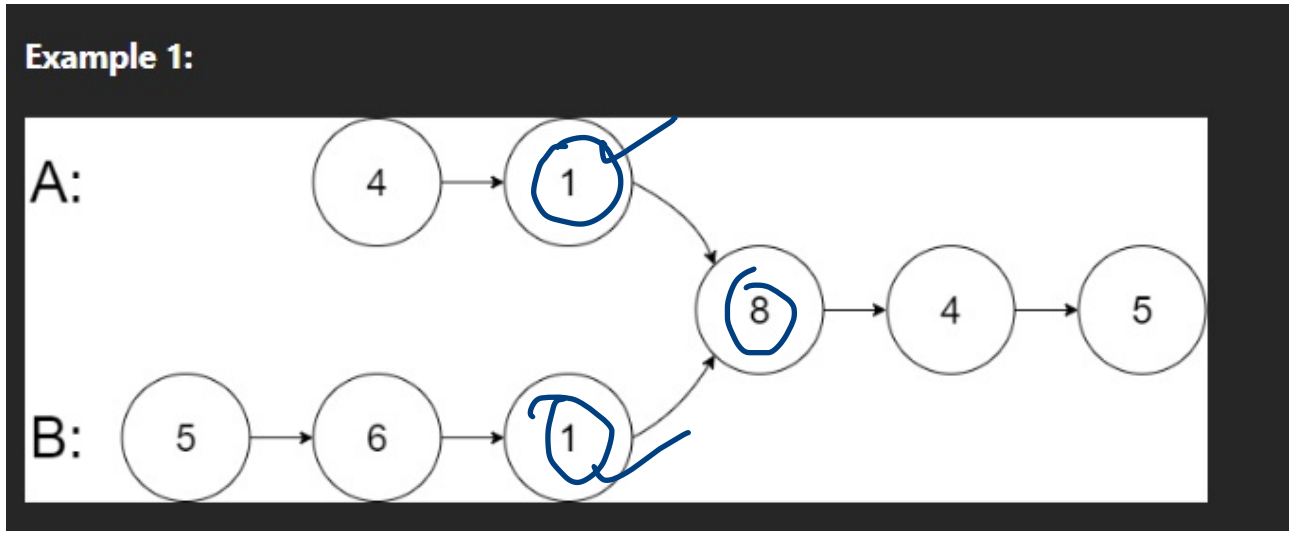


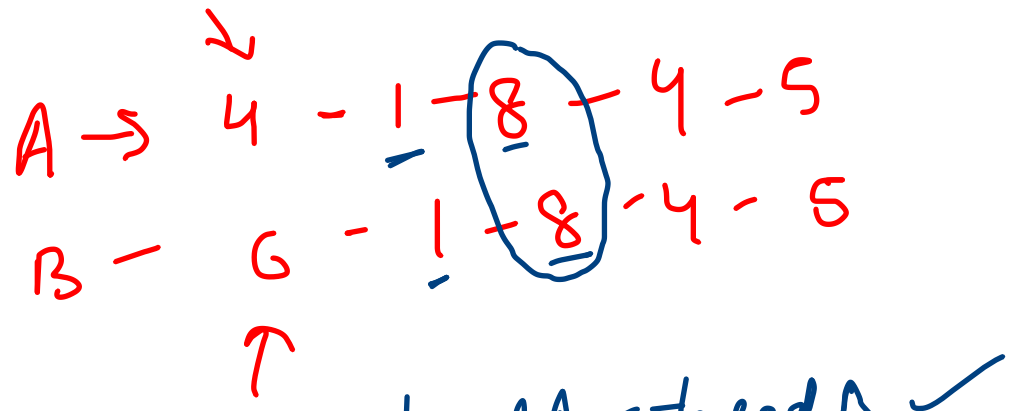

```java
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;

        while(curr != null){
            ListNode next = curr.next; // store the next node
            curr.next = prev; // reverse the link
            prev = curr; // move prev forward
            curr = next; // move next forward
        }
        return prev;
    }
}
```

TC → O(n)

SC → O(1)

Intersect of two LL





A → 4 → 1 → 8 → 4 → 5 (5) count 1 = 5

B → ~~5~~ → 6 → 1 → 8 → 4 → 5 (6) count 2 = 6

count 1 > count 2

count 2 > count 1

6 > 5 T

```java
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

        ListNode q1=headA,q2=headB;
        int count1=0,count2=0;
        while(q1!=null){
            count1++;
            q1=q1.next;
        }
        while(q2!=null){
            count2++;
            q2=q2.next;
        }
        while(count1>count2){
            headA=headA.next;
            count1--;
        }
        while(count1<count2){
            headB=headB.next;
            count2--;
        }
        while(headA!=headB){
                headA=headA.next;
                headB=headB.next;
            }
            return headA;
    }
}
```

O(n) — length of L1

n — length of L2

n — skip nodes of L1

n — skip nodes of L2

n

TC → O(n)

SC → O(1)