Day 4

25 November 2022 21:01

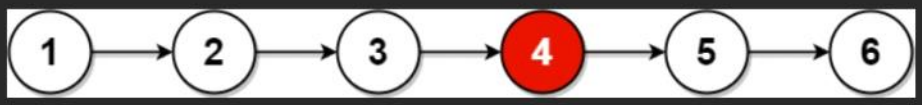
Given the `head` of a singly linked list, return *the middle node of the linked list.*

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

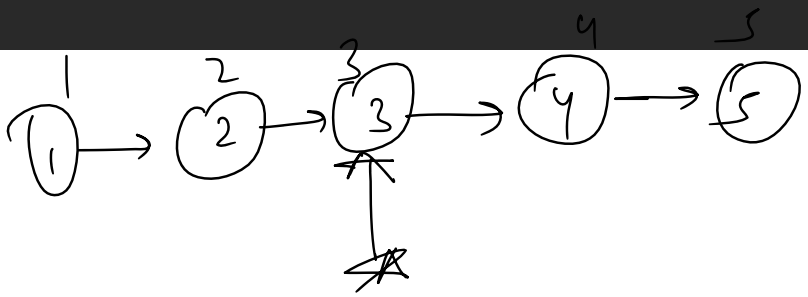
Example 1:

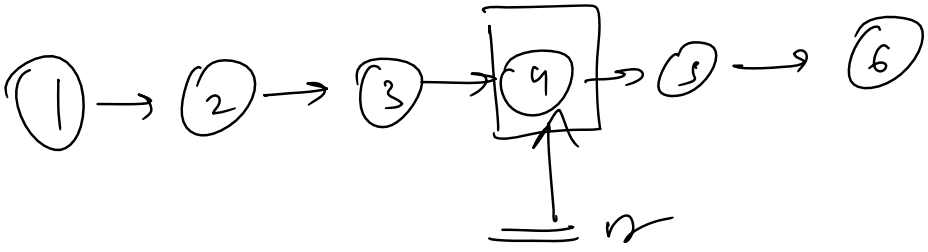

```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

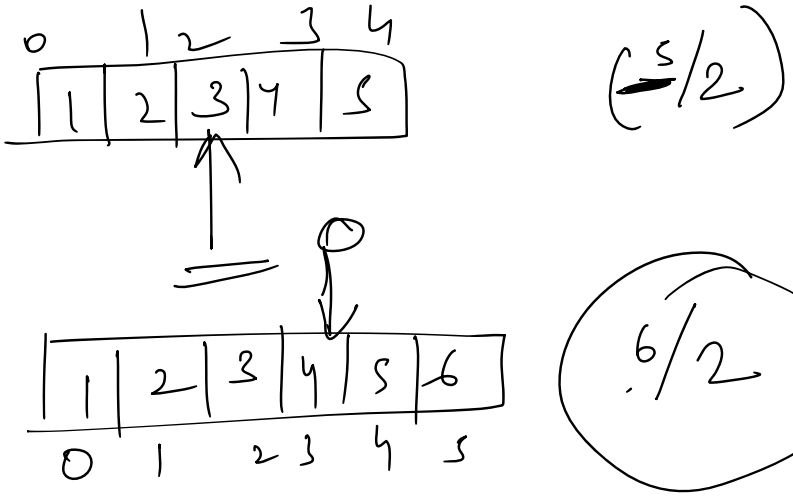
Example 2:




```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.
```

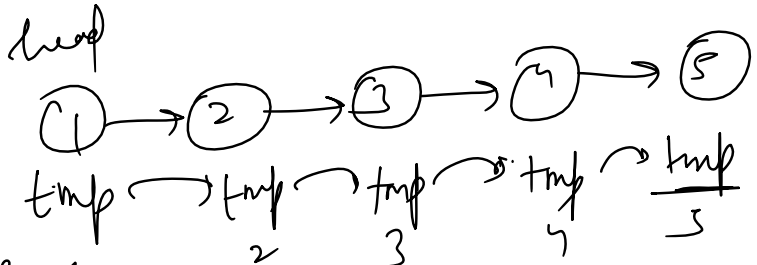






Linked list

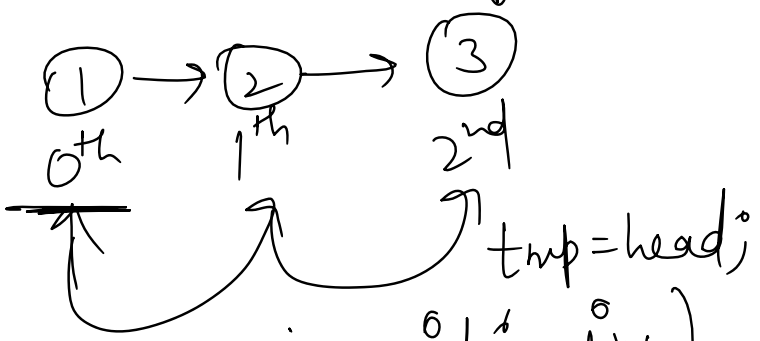
Naive




```
c=1
tmp=head; 1
while(tmp->next!=0)
{   c++;
    tmp=tmp->next;
}
```

c = 5

mid = 5/2 ⇒ 2nd

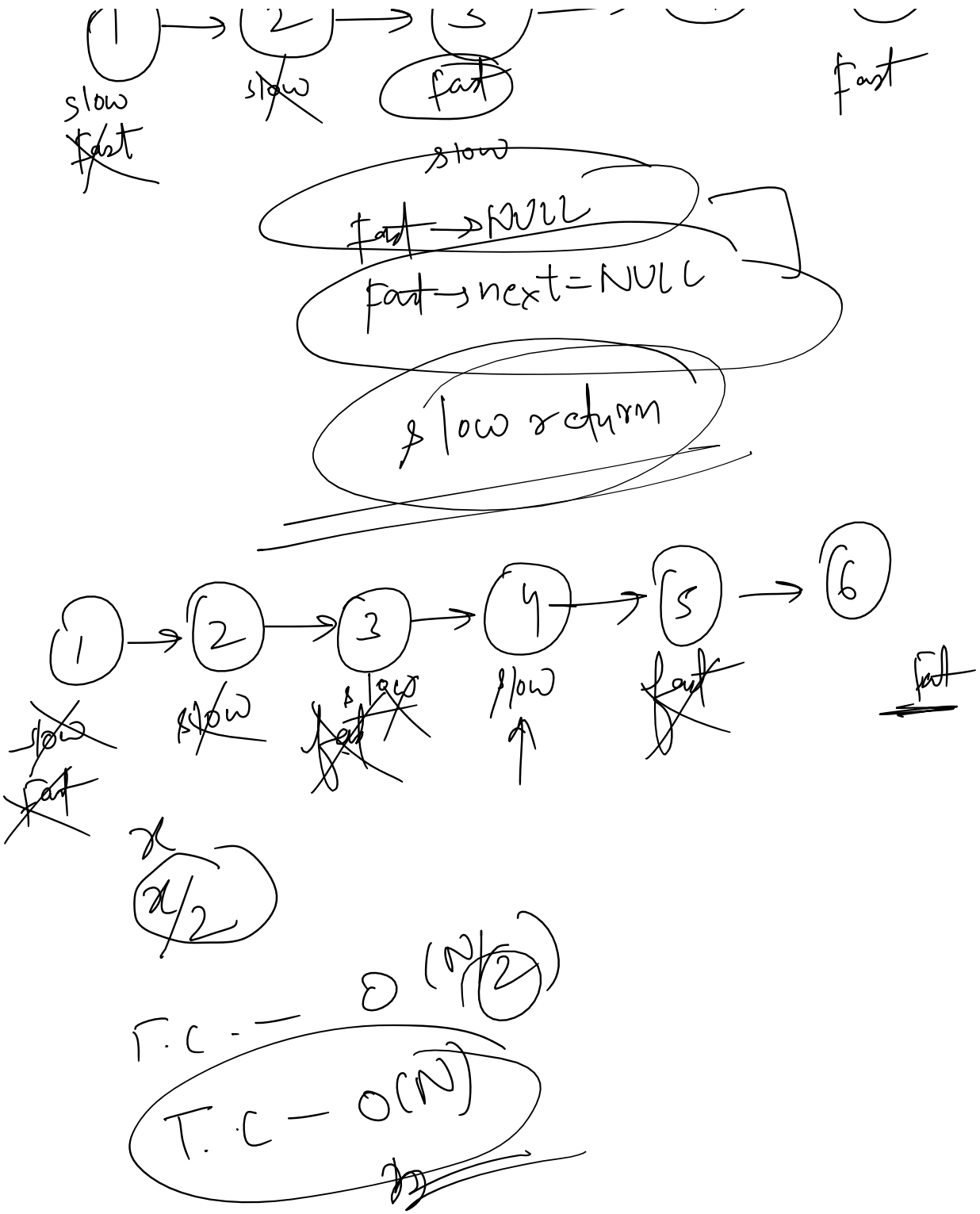



```
for(int i=1; i<=mid; i++)
{ tmp= tmp->next;
}
```

tmp

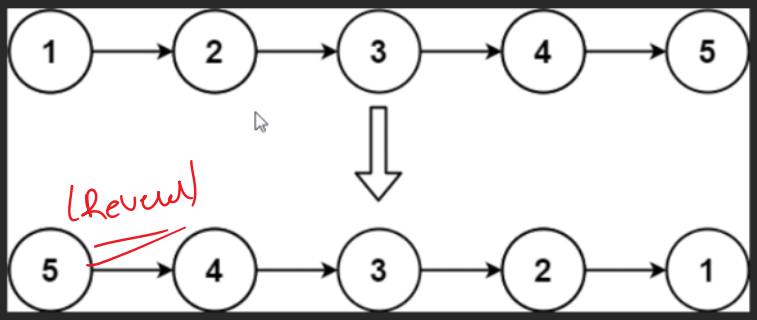
T.C - O(N)

S.C - O(1)

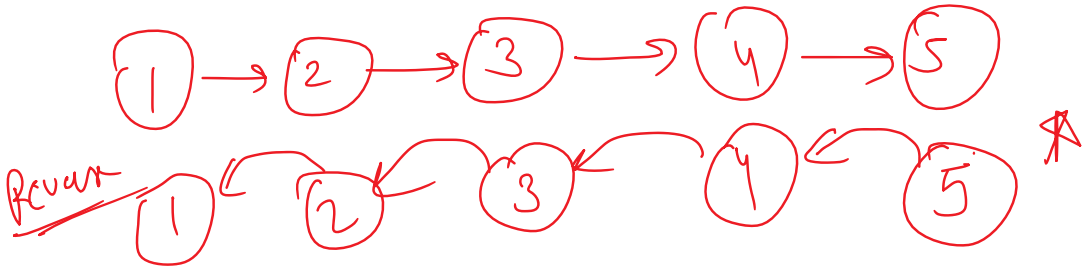

1
2
3
4
5
6
slow
fast
slow
fast
fast
slow
fast->NULL
fast->next=NULL
slow return
n
n/2
T.C - O(N/2)
T.C - O(N)

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:




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

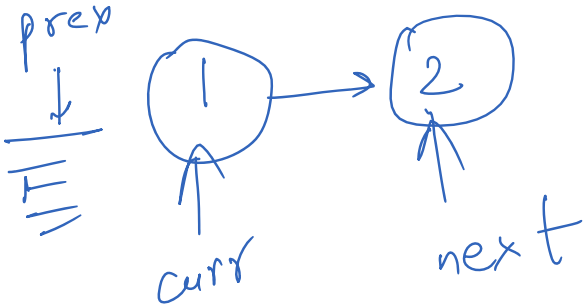


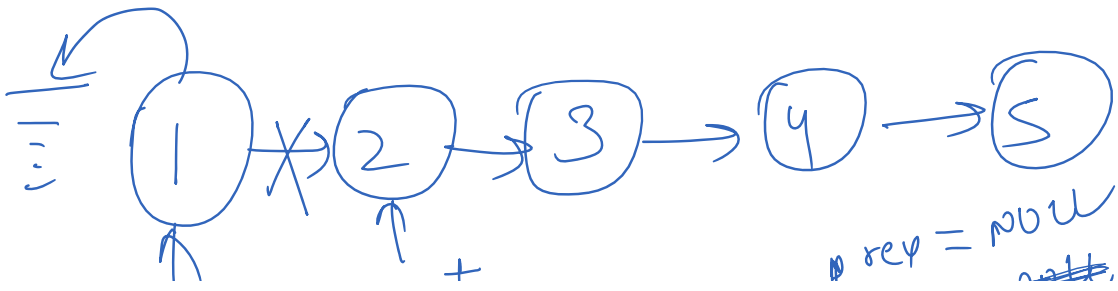
1. Recursion
2. Iterative

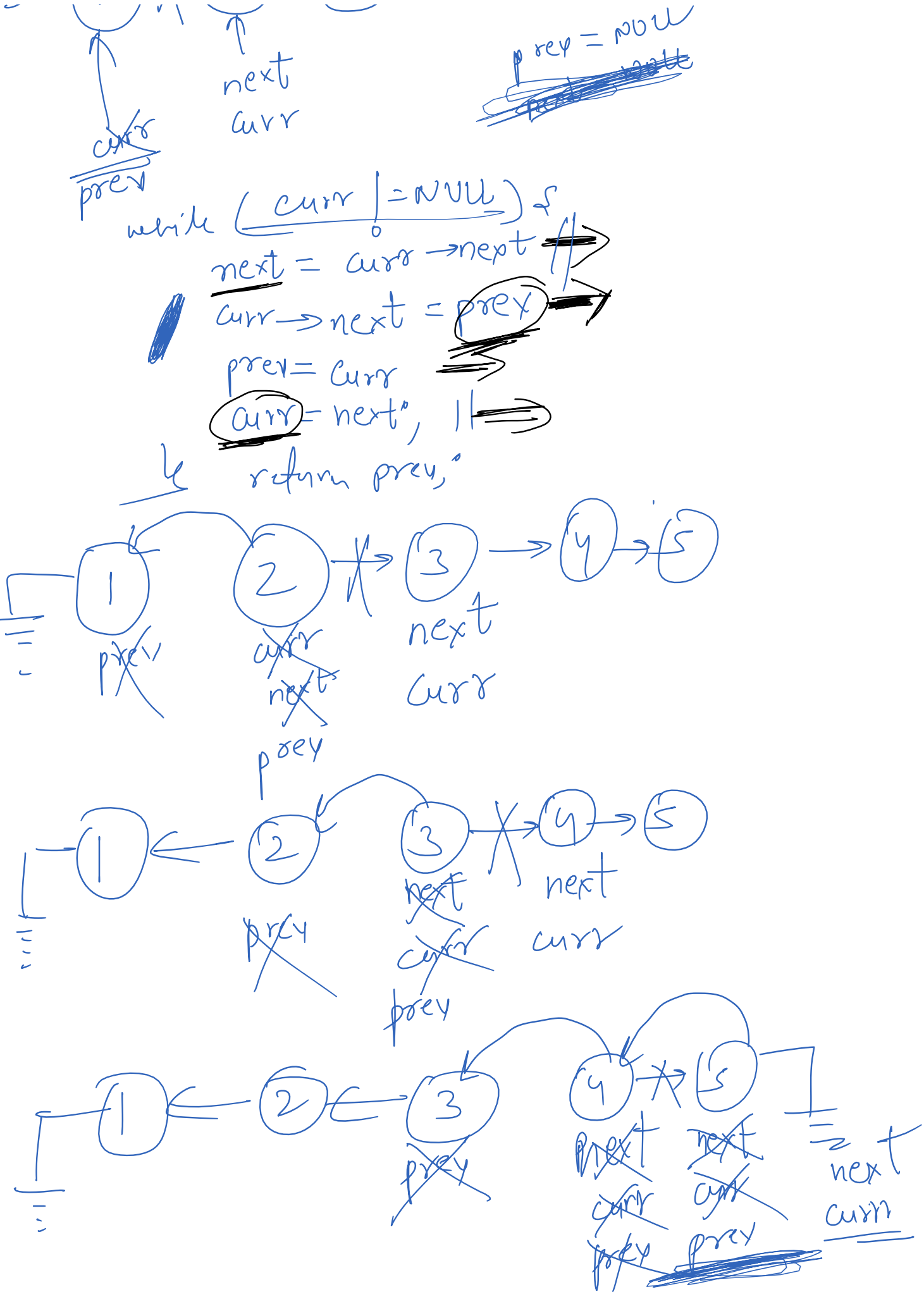
prev → previous Node

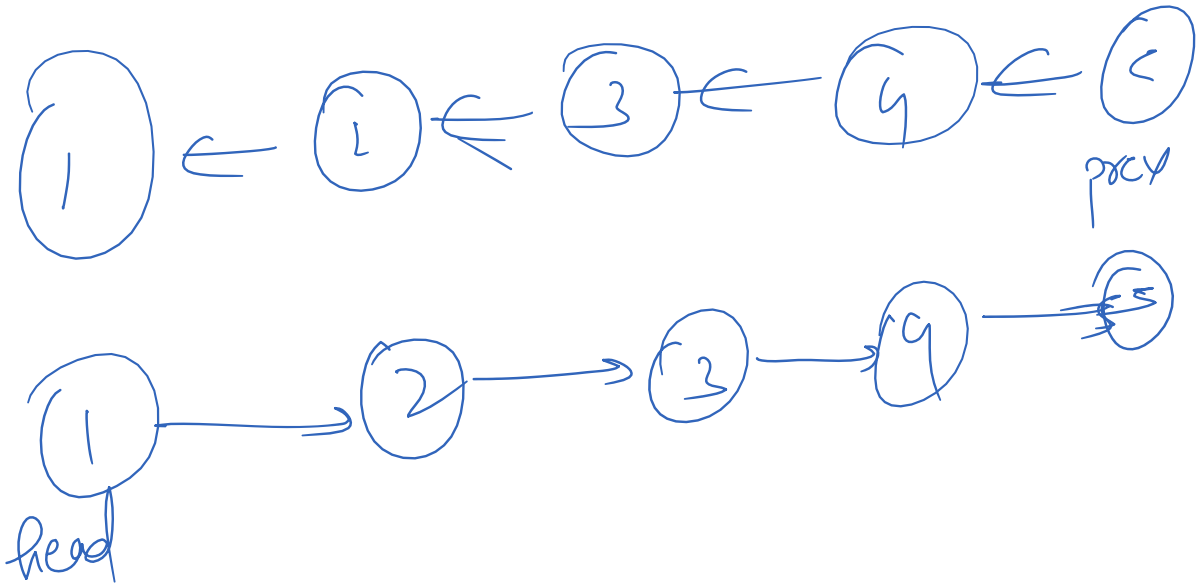
curr → current Node

next → current ~~to~~ next node






curr
prev
next
curr
prev = null
while (curr != NULL) {
next = curr->next
curr->next = prev
prev = curr
curr = next;
}
return prev;
1
2
3
4
5
prev
curr
next
curr
next
curr
prev
1
2
3
4
5
prev
next
curr
next
curr
prev
1
2
3
4
5
prev
prev
next
curr
next
curr
prev
prev
next
curr




```cpp
ListNode* reverseList(ListNode* head) {
    ListNode * prev= nullptr,*next = nullptr,*curr = head;
    while(curr != nullptr){
        next = curr->next;
        curr->next = prev;
        prev= curr;
        curr = next;
    }
    return prev;
}
```

T.C - O(N)
S.C - O(1)

Recursive

```cpp
ListNode * helper(ListNode * prev,ListNode *curr, ListNode *next){
    if(curr == nullptr) return prev;
    next = curr->next;
    curr->next = prev;
    prev = curr;
    curr = next;
    return helper(prev,curr,next);
}
ListNode* reverseList(ListNode* head) {
    if(head == nullptr || head->next == nullptr) return head;
    return helper(nullptr,head,nullptr);
}
```

T.C - O(N)

T.C – O(N)

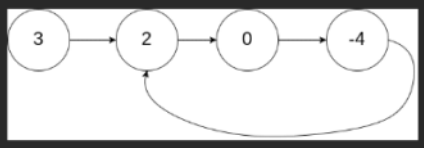
S.C – O(N)

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

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

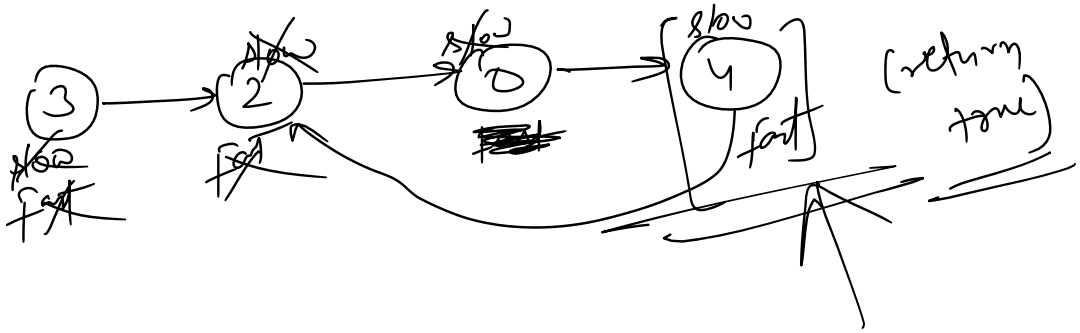
Return `true` *if there is a cycle in the linked list.* Otherwise, return `false`.

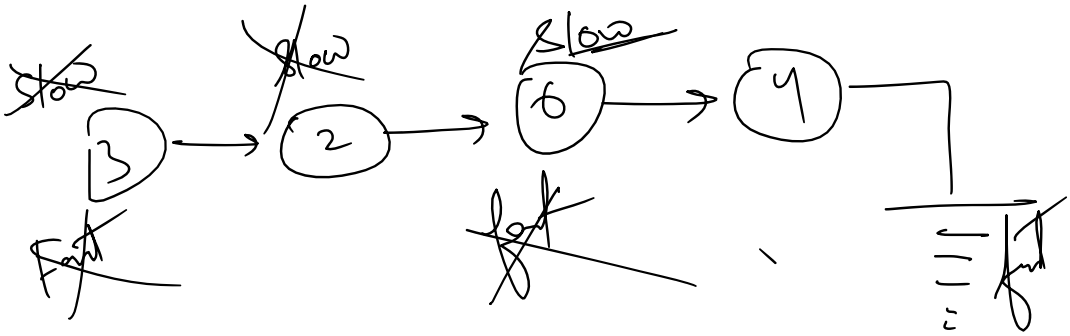
Example 1:

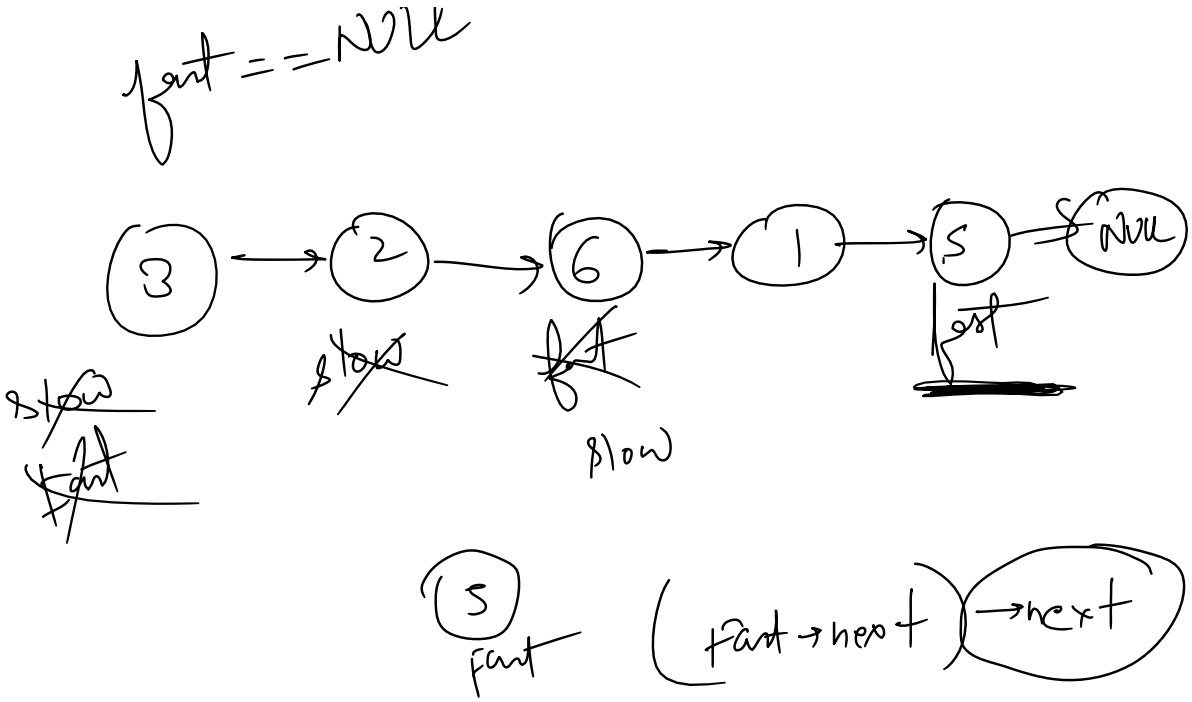



```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).
```

Example 2:



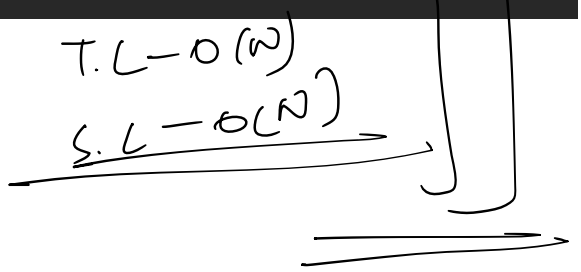





```c
bool hasCycle(ListNode *head) {
    if(head == NULL || head->next == NULL) return false;
    ListNode * slow  = head;
    ListNode * fast = head;
    do{
        slow = slow->next;
        fast = fast->next->next;
        if(slow == fast){
            return true;
        }
    }while(fast!= NULL && fast->next!= NULL);
    return false;
}
```

T.C - O(N)

```cpp
bool helper(ListNode * slow,ListNode * fast){
    if(fast == NULL || fast->next == NULL) return false;
    slow = slow->next;
    fast = fast->next->next;
    if(slow == fast) return true;
    return helper(slow,fast);
}
bool hasCycle(ListNode *head) {
   if(head == NULL || head->next == NULL) return false;
   return helper(head,head);
}
```

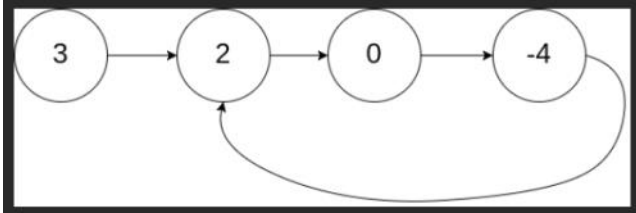


Given the `head` of a linked list, return *the node where the cycle begins. If there is no cycle, return* `null`.

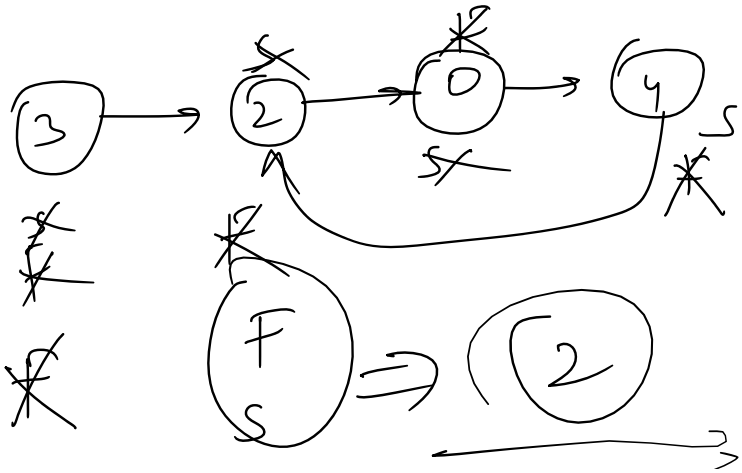
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter**.

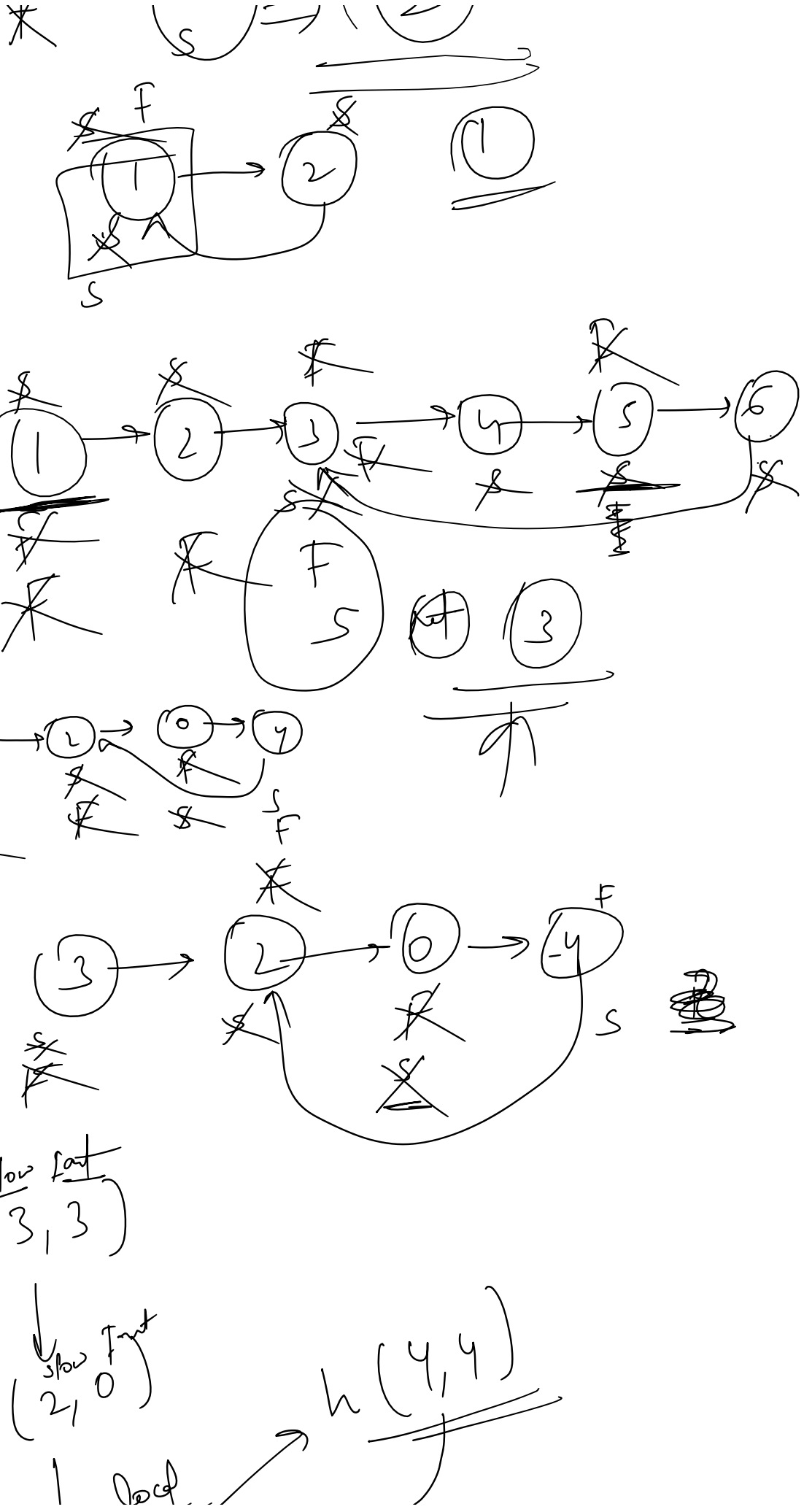
Do not modify the linked list.

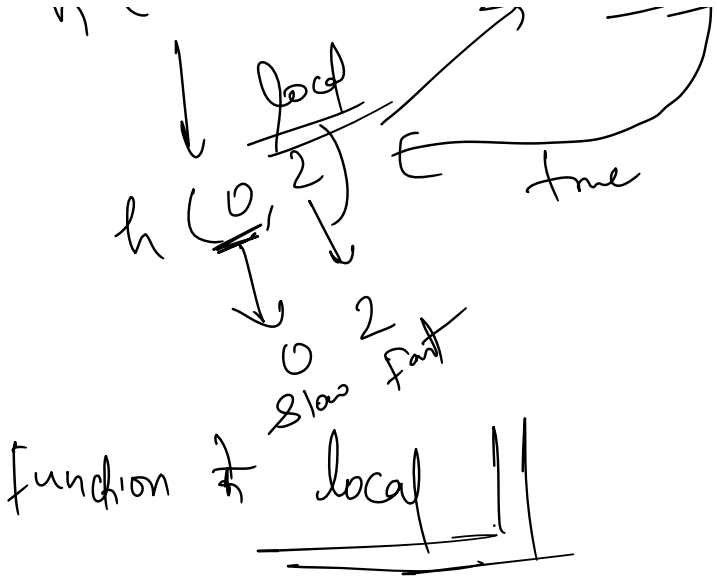
Example 1:




```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the second node.
```




slow fast
3,3
slow fast
2,0
4,4




```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
//  if(head == NULL || head->next == NULL) return NULL;
//         ListNode * slow  = head;
//         ListNode * fast = head;
//         bool flag = false;
//         do{
//             slow = slow->next;
//             fast = fast->next->next;
//             if(slow == fast){
//                 flag = true;
//                 break;
//             }
//         }while(fast!= NULL && fast->next!= NULL);
//         if(!flag) return NULL;
//         fast = head;
//         while(fast != slow){
//             slow = slow->next;
//             fast = fast->next;
//         }
//         return slow;
ListNode * position  = NULL;
bool helper(ListNode *slow,ListNode *fast){
    if(fast == NULL || fast->next == NULL) return false;
    slow = slow->next;
    fast = fast->next->next;
    if(slow == fast){
      position = slow;
      return true;
    }
    return helper(slow,fast);
}
ListNode *helper2(ListNode * slow, ListNode * fast){
    if(slow == fast) return slow;
     slow = slow->next;
     fast = fast->next;
     return helper2(slow,fast);
}
ListNode *detectCycle(ListNode *head) {
        if(head == NULL || head->next == NULL) return NULL;
        ListNode *slow = head;
        ListNode *fast = head;
        if(!helper(slow,fast))return NULL;
        // cout << slow->val << "  " << fast->val << endl;
        return helper2(position,head);
    }
};
```

loop condition ⟺ Base Case