Linked List 2 ¶

Question
https://leetcode.com/problems/remove-nth-node-from-end-of-list/ (https://leetcode.com/problems/remove-nth-node-from-end-of-list/)

Solution-1: Brute Force

1. Find length of linked list
2. Reach the len-nth node
3. Delete it

Solution-2: Two Pointer

1. Create a pointer p1=head
2. Move pointer p1 n steps
3. Create another pointer p2=head
4. Move both p1 and p2, 1 step ahead till p1 reaches end
5. Delete p2

C#

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     public int val;
 *     public ListNode next;
 *     public ListNode(int val=0, ListNode next=null) {
 *         this.val = val;
 *         this.next = next;
 *     }
 * }
 */
public class Solution {
    public ListNode RemoveNthFromEnd(ListNode head, int n) {
        var iterator = head;
        var counter = 1;

        // here we count the nodes in the list
        while (iterator.next != null)
        {
            iterator = iterator.next;
            counter++;
        }

        iterator = head;
        int total = counter;

        // if n equals node count then we return the list except firs
t node
        if (total == n)
        {
            return head.next;
        }
```

```
        slow = head
        fast = head
        # n times move fast pointer 1 step
        for i in range(n):
            fast = fast.next

        # if n == len(list)
        if not fast: # fast == NULL
            return head.next

        while fast.next:
            slow = slow.next
            fast = fast.next
        slow.next = slow.next.next
        return head
# [1,2,3,4,5] n =2
#           F
#       S

# [1, 2, 3] n=3
#           F=NULL
```

Java

```
    public ListNode removeNthFromEnd(ListNode head, int n) {
            int length = 0;
            ListNode temp = head;
```

C++

```
    ListNode* removeNthFromEnd(ListNode* head, int n) {

            ListNode* slow=head;
            ListNode* fast=head;
            for(int i=0;i<n;i++){
                fast =fast->next;
            }

            if(fast==nullptr) return head->next;

            while( fast->next!=nullptr){
                slow=slow->next;
                fast=fast->next;
            }

            ListNode* temp=slow->next;
            slow->next= slow->next->next;
            delete temp;

            return head;
        }
```

Question
https://leetcode.com/problems/middle-of-the-linked-list/ (https://leetcode.com/problems/middle-of-the-linked-list/)

```
  1 2 3 4 5 6
  s
    F

  1 2 3 4 5
  s
    F
TC: O(N)
SC: O(1)
```

JAVA

```
public ListNode middleNode(ListNode head) {
    if(head==null || head.next==null) return head;

        ListNode slow = head;
        ListNode fast = head; // NOTICE THIS
        while(fast!=null && fast.next!=null){
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

public ListNode middleNode(ListNode head) {
    if(head==null || head.next==null) return head;

        ListNode slow = head;
        ListNode fast = head.next; // NOTICE THIS
        while(fast!=null){
            slow = slow.next;

            fast = fast.next;
            if (fast!=null) fast = fast.next;
        }
        return slow;
    }
```

Python

```
        if head == None:
            return
        slow = head
        fast = head
        # check if fast and fast next is not equals to None run the l
oop
        while fast != None and fast.next != None:
            slow = slow.next
            fast = fast.next.next # advance the fast pointer two step
s in every iteration when fast reaches to the end of the linkedlist s
low pointer stops at middile of the node.
        return slow
```

```
public ListNode middleNode(ListNode head) {

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

        ListNode slow = head;
        ListNode fast = head;

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

In [ ]:
```
1
```

Question
https://leetcode.com/problems/reverse-linked-list/ (https://leetcode.com/problems/reverse-linked-list/)

In [ ]:
```
1  ```Python
2
3     1->2->3->4->5->X
4   p c   n
5
6  p = x
7  c = head
8  n = c->next
9
10 ```
```

TC: O(N)
SC: O(1)
C++-1

```
        ListNode* reverseList(ListNode* head) {
            if (head == NULL) return head;

            ListNode *prev=NULL;
            ListNode *curr=head;
            ListNode *next=head->next;

            while( curr != NULL ) { // TODO: verify
                curr->next = prev;
```

C++-2

```
    class Solution {
    public:
        ListNode* reverseList(ListNode* head) {

            ListNode *prev=NULL;
            ListNode *curr=head;
            ListNode *next=NULL;

            while( curr != NULL ) {
                next = curr->next;
                curr->next = prev;
                prev = curr;
                curr = next;
            }

            return prev;

        // x<-1<-2<-3  x
        //          p  c,n
        }
    };
```

TC: O(2N) = O(N)
SC: O(N)

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        std::vector<int> data;
        ListNode *temp = head;
        while(temp != NULL) {
            data.push_back(temp->val);
            temp=temp->next;
        }

        // start from beginning again
```

In [ ]:
```
1
```

Question
https://leetcode.com/problems/merge-two-sorted-lists/ (https://leetcode.com/problems/merge-two-sorted-lists/)

C++
TC: O((m+n) log (m+n))
SC: O(m+n)

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

        vector<int> data;
```

Reccursive

TC: O(m+n)

SC: O(m+n) including stack for recursion

```
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        if (list1 == null)
                return list2;
        if (list2 == null)
```

Iterative

TC: O(m+n)

SC: O(1)

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

       if(list1 == NULL) {
```

```
**DIY**
Write recursive solution for https://leetcode.com/problems/reverse-linked-list/
https://leetcode.com/problems/reverse-nodes-in-k-group/
```