**Linked List**

**Delete Node in Linked List:**

There is a singly-linked list head and we want to delete a node node in it.

You are given the node to be deleted node. You will **not be given access** to the first node of head.

All the values of the linked list are **unique**, and it is guaranteed that the given node node is not the last node in the linked list.

Delete the given node. Note that by deleting the node, we do not mean removing it from memory. We mean:

* The value of the given node should not exist in the linked list.
* The number of nodes in the linked list should decrease by one.
* All the values before node should be in the same order.
* All the values after node should be in the same order.

**Custom testing:**

* For the input, you should provide the entire linked list head and the node to be given node. node should not be the last node of the list and should be an actual node in the list.
* We will build the linked list and pass the node to your function.
* The output will be the entire list after calling your function.

**Example 1:**

A diagram of a given node

Description automatically generated with medium confidence

**Input:** head = [4,5,1,9], node = 5

**Output:** [4,1,9]

**Explanation:** You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.

Trick: Convert node to be deleted to node next to it

Time and space complexity O(1)

A screenshot of a computer program

Description automatically generated with medium confidence

**Remove Nth Node from End of List:**

Given the head of a linked list, remove the nth node from the end of the list and return its head.

**Example 1:**

A picture containing circle, diagram, screenshot, line

Description automatically generated

**Input:** head = [1,2,3,4,5], n = 2

**Output:** [1,2,3,5]

Trick: Use 2 pointers and a dummy node. Time complexity O(n). <https://www.youtube.com/watch?v=XVuQxVej6y8&t=301s>

N = 2

1 -> 2 -> 3 -> 4 -> 5



1 -> 2 -> 3 -> 4 -> 5 Shift to right by 1 until right point to at the end of the list (points at “null”)



Now left pointer is at node we want to delete and right pointer is pointing to “null”

But how do we delete node? Use dummy node. Initialize left pointer at dummy node.

Dummy -> 1 -> 2 -> 3 -> 4 -> 5







Then update left to remove node at N by updating pointer

Dummy -> 1 -> 2 -> 3 -> 4 -> 5



Return new link list with dummy.next

A screenshot of a computer code

Description automatically generated with medium confidence

**Reverse Linked List:**

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

**Example 1:**

A picture containing circle, diagram, line

Description automatically generated

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

**Output:** [5,4,3,2,1]

Iterative solution: Use 2 pointers, previous and current. Swap pointers until current pointer reaches end of linked list

Time complexity O(n); space O(1)

Null -> 1 -> 2 -> 3 -> Null



Null <- 1 -> 2 -> 3 -> Null



Null <- 1 <- 2 -> 3 -> Null



Null <- 1 <- 2 <- 3 -> Null



Return previous pointer since equal to new head

A screenshot of a computer code

Description automatically generated with medium confidence

Recursive solution: Split into subproblems. Maintain last node as new head.

Time and space complexity O(n)

Null -> 1 -> 2 -> 3 -> Null



Null -> 1 -> 2 -> 3 -> Null



Null -> 1 -> 2 <- 3 -> Null



Null <- 1 -> 2 <- 3 -> Null

A screenshot of a computer program

Description automatically generated with medium confidence

<https://www.youtube.com/watch?v=G0_I-ZF0S38>

**Merge Two Sorted Lists:**

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*

**Example 1:**

A picture containing circle, weightlifting

Description automatically generated

**Input:** list1 = [1,2,4], list2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

Trick: Create dummy node to avoid edge case of empty list <https://www.youtube.com/watch?v=XIdigk956u0>

If one list is longer than the other, just take the remaining nodes in the longer list and insert into output since already in order

Complexity: time - O(n + m); space – O(1) since we are shifting pointers

A screenshot of a computer program

Description automatically generated with medium confidence

**Palindrome Linked List:**

Given the head of a singly linked list, return trueif it is apalindromeorfalseotherwise.

**Example 1:**

A picture containing circle, line

Description automatically generated

**Input:** head = [1,2,2,1]

**Output:** true

Trick 1: Put node values in an array and check indices. Time and space complexity O(n)

A screenshot of a computer code

Description automatically generated with medium confidence

Solution in O(1) memory:

Trick 2: Use a fast and slow pointer. Both pointers start at the head, but one moves faster than the other. The idea is that when the fast pointer reaches the end of the list, the slow pointer will be somewhere in the middle. We can check the values in the beginning and the end of the list just like we did with array algorithm, we just need to reverse the pointers.

A screenshot of a computer program

Description automatically generated with medium confidence

<https://www.youtube.com/watch?v=yOzXms1J6Nk>

**Linked List Cycle:**

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 trueif there is a cycle in the linked list. Otherwise, return false.



**Example 1:**



A picture containing sketch, drawing, diagram, circle

Description automatically generated



**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).

Trick: Tortoise and hare algorithm (fast and slow pointers)

Time complexity O(n); space O(1)

If we reach null, then there’s no cycle. Fast pointer would get there first.

Slow and fast pointer will meet at same node if there is a cycle (see notes above).

Slow pointer move 1 position and fast moves 2 positions.

Why is time complexity always n?

How big can closing gap (10 in example above) possibly be?



It can only be the entire length of the list (n-1). So how many iterations to get n - 1 = 0? Roughly n so complexity O(n) where n is the length of the cycle which could be entire linked list.

<https://www.youtube.com/watch?v=gBTe7lFR3vc>

A screen shot of a computer code

Description automatically generated with low confidence