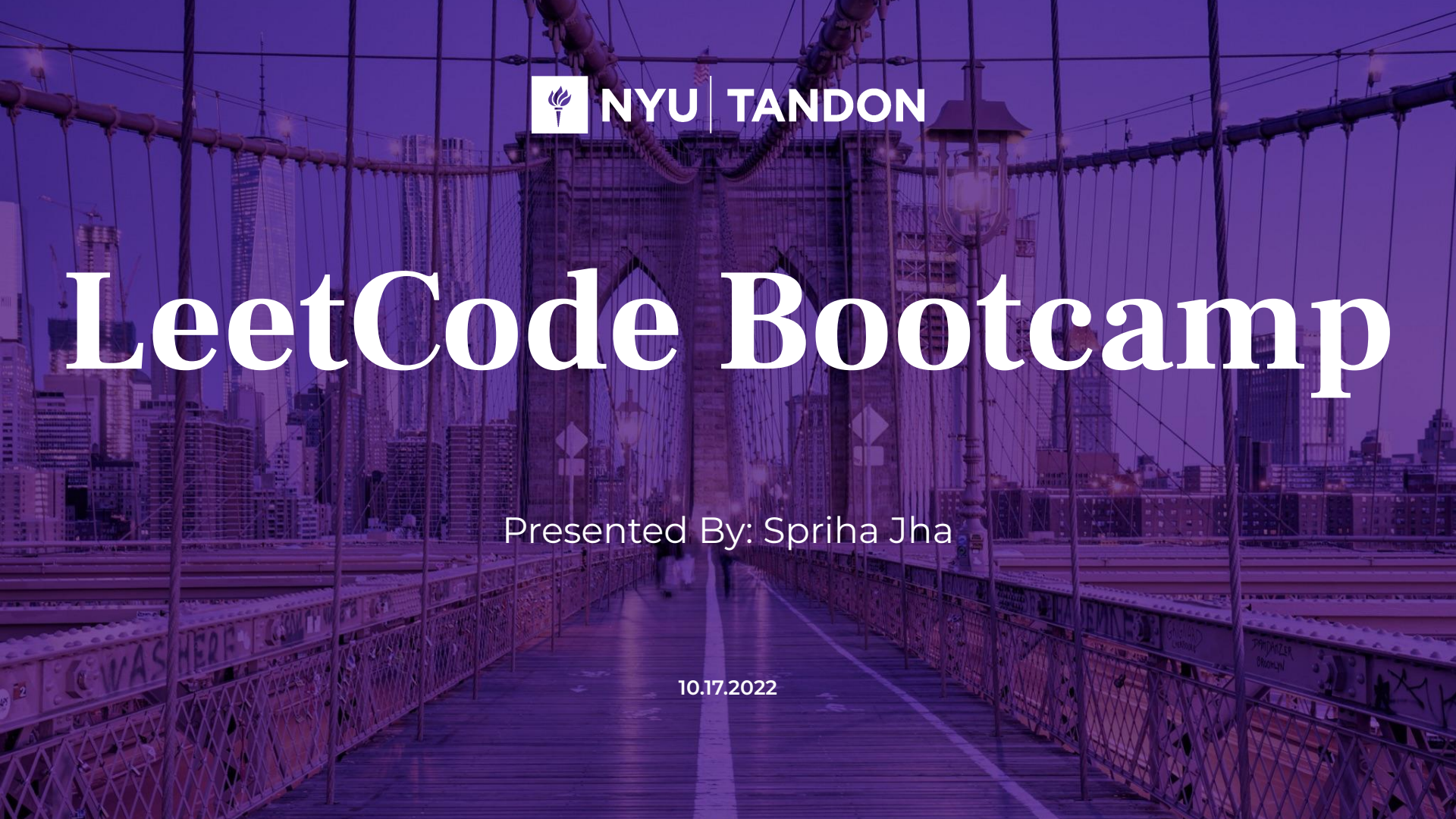


LeetCode Bootcamp

Presented By: Spriha Jha

10.17.2022

Session Outline

01. Introduction to Linked Lists (Guide)

02. Problem Sets

03. Debrief & Q/A

Linked Lists

Types:

- Singly Linked List
- Doubly Linked List
- Circular Linked List

Corner cases:

- Empty linked list (head is null)
- Single node
- Two nodes
- Linked list has cycles.

Techniques:

- Sentinel/dummy nodes
- Two pointers
- Modification operations

PART 02

Problem Sets

Steps to approach the question:

Understand the problem

Take time to carefully read through the problem from start to finish is critical in finding the correct and complete solution to the problem in hand.

Code your solution

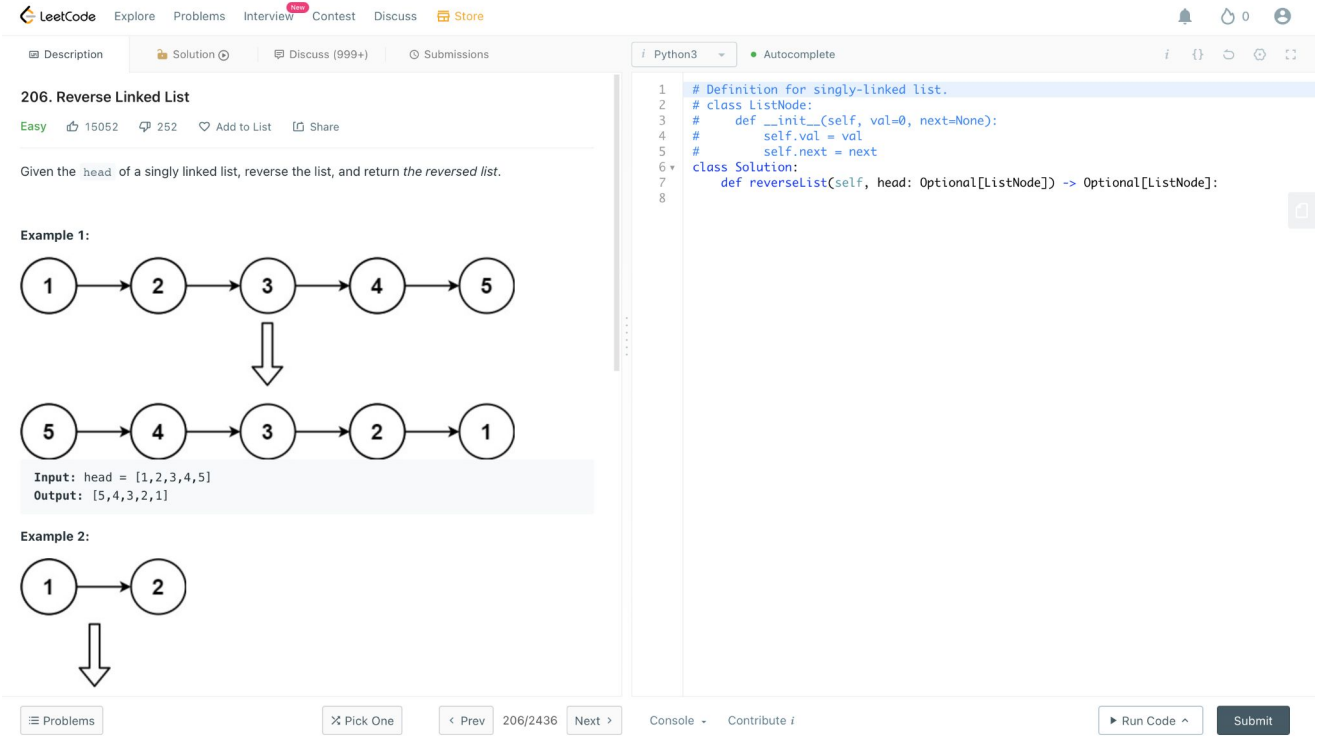
Map out your solution before you write any code. Avoid too much time trying to find the perfect solution. Validate your solution early and often.

Manage your time

Don’t forget, you have multiple questions to complete within a said time. Make sure you allocate enough time to carefully consider all problems.

PART 02

Problem 1: Reverse Linked List



NYU TANDON

Approach: Iterative

```
def reverseList(self, head: ListNode) -> ListNode:

    prev = None
    curr = head

    while curr:
        next_temp = curr.next
        curr.next = prev
        prev = curr
        curr = next_temp

    return prev
```

Complexity Analysis

Time complexity : Assume that n is the list's length, the time complexity is O(n).

Space complexity : O(1)

Approach: Recursion

```
def reverseList(self, head: ListNode) -> ListNode:

    if (not head) or (not head.next):
        return head

    p = self.reverseList(head.next)
    head.next.next = head
    head.next = None

    return p
```

Complexity Analysis

Time complexity : Assume that n is the list's length, the time complexity is O(n).

Space complexity : O(n) The extra space comes from implicit stack space due to recursion. The recursion could go up to n levels deep.

Problem 2: Middle of the Linked List

LeetCode Explore Problems Interview Contest Discuss Store

Description | Solution | Discuss (999+) | Submissions

876. Middle of the Linked List

Easy 7287 194 Add to List Share

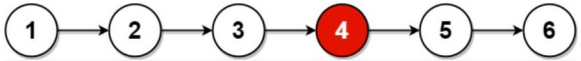
Given the `head` of a singly linked list, return *the middle node of the linked list.*

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.
```

Constraints:

Problems | Pick One | Prev 47/69 Next

Python3 | Autocomplete

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:

```

Console | Contribute | Run Code | Submit

NYU TANDON

Approach:

```
def middleNode(self, head: ListNode) -> ListNode:
    arr = [head]

    while arr[-1].next:
        arr.append(arr[-1].next)

    return arr[len(arr) // 2]
```

Complexity Analysis

Time complexity: $O(N)$, where N is the number of nodes in the given list.
Space complexity: O(N), the space used by array.

PART 02

Problem 3: Reorder List

LeetCode Explore Problems Interview Contest Discuss Store

Description Solution Discuss (999+) Submissions

Python3 Autocomplete

143. Reorder List

Medium 7411 258 Add to List Share

You are given the head of a singly linked-list. The list can be represented as:

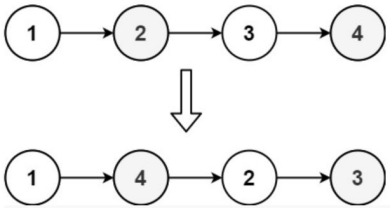
$L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$

Reorder the list to be on the following form:

$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:



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

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reorderList(self, head: Optional[ListNode]) -> None:
        """
        Do not return anything, modify head in-place instead.
        """
```

Problems Pick One Prev 143/2436 Next Console Contribute i Run Code Submit

NYU TANDON

Approach

```
def reorderList(self, head: ListNode) -> None:
    if not head:
        return

    # find the middle of linked list [Problem 876] in 1->2->3->4->5->6 find 4
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    # reverse the second part of the list [Problem 206] convert 1->2->3->4->5->6 into 1->2->3->4 and 6->5->4 reverse the second half in-place
    prev, curr = None, slow
    while curr:
        curr.next, prev, curr = prev, curr, curr.next

    # merge two sorted linked lists [Problem 21] merge 1->2->3->4 and 6->5->4 into 1->6->2->5->3->4
    first, second = head, prev
    while second.next:
        first.next, first = second, first.next
        second.next, second = first, second.next
```

Complexity Analysis

Time complexity : O(N) There are three steps here. To identify the middle node takes O(N) time. To reverse the second part of the list, one needs N/2 operations. The final step, to merge two lists, requires N/2 operations as well. In total, that results in O(N) time complexity.

Space complexity : O(1), since we do not allocate any additional data structures.

PART 06

Q/A

Slack Hours

Join Slack Workspace!

Office Hours: Tuesday (10AM - 1PM)

Problem Assignments

01. Linked List Cycle (Easy)

02. Merge Two Sorted Lists (Easy)

03. Delete N Nodes After M Nodes of a Linked List (Easy)

04. Linked List Cycle II (Medium)

05. Remove Nth Node From End of List (Medium)

06. Swapping Nodes in a Linked List (Medium)

07. Merge k Sorted Lists (Hard)


NYU TANDON
Thank you!
Upcoming: Binary Trees (10/24)