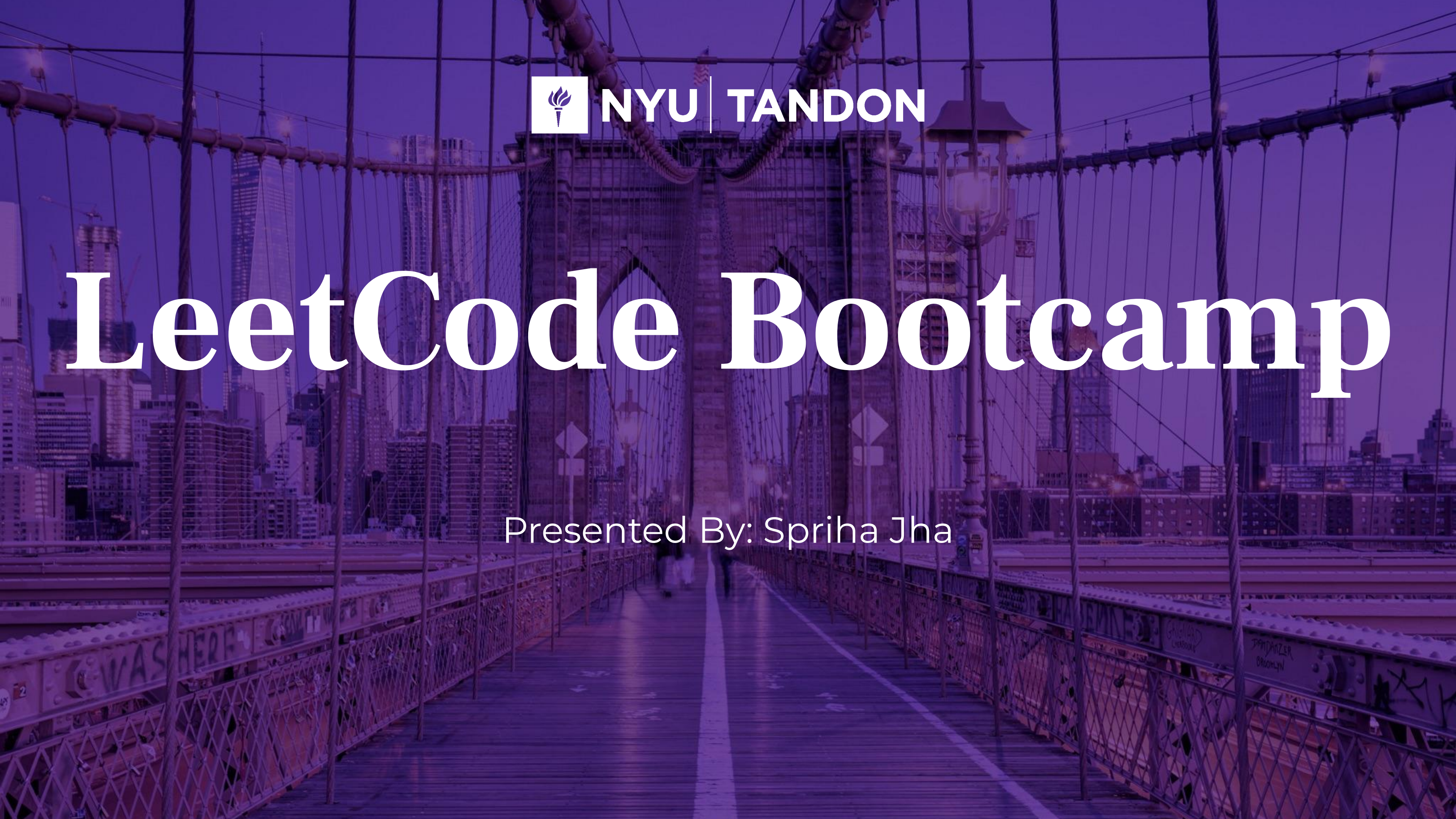


NYU | TANDON

LeetCode Bootcamp

Presented By: Spriha Jha

Session Outline

NYU TANDON

Linked Lists

- Collection of nodes which together represent a sequence. (Dynamic Data Structure)

- Each node contains: **data**, and a **reference** to the next node in the sequence.

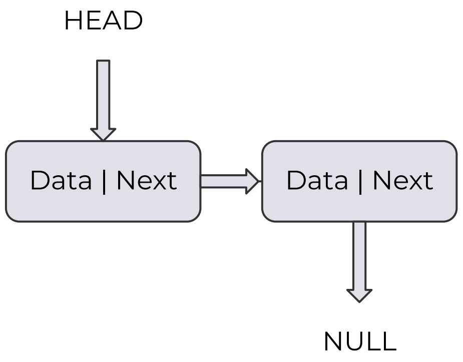
- **Advantage:** Node Insertion/Deletion (given its location) is O(1) whereas in arrays the following elements will have to be shifted.

- **Disadvantage:** Linear access time because elements access by its position is not possible. You have to traverse from the start.

- Differentiator from arrays, memory allocation in our machines. Arrays are allocated memory in one contiguous block. Linked Lists can be scattered throughout.

- It’s usually efficient when it comes to adding and removing most elements, but can be very slow to search and find a single element.

Parts of a Linked List



- A series of **nodes**, which are the elements of the list.

- The starting point of the list is a reference to the first node, which is referred to as the **head.**

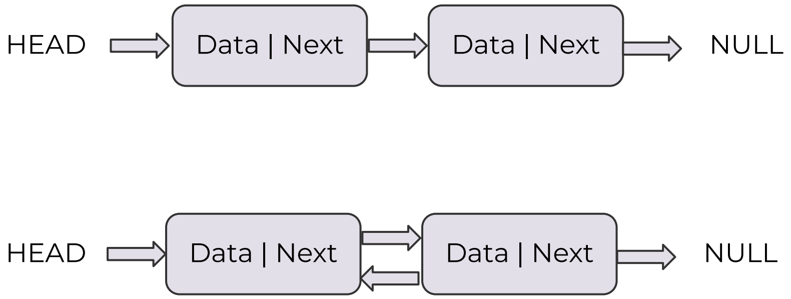
- The end of the list isn’t a node, but rather a node that points to **null**, or an empty value.

- Each node has just two parts: **data**, or the information that the node contains, and a reference to the **next node**.

- A single node has the “address” or a reference to the next node, they don’t need to live right next to one another

NYU | TANDON

Linked Lists



Types:

- Singly Linked List (Uni-direction)
- Doubly Linked List (Bi-direction)
- 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

NYU | TANDON

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

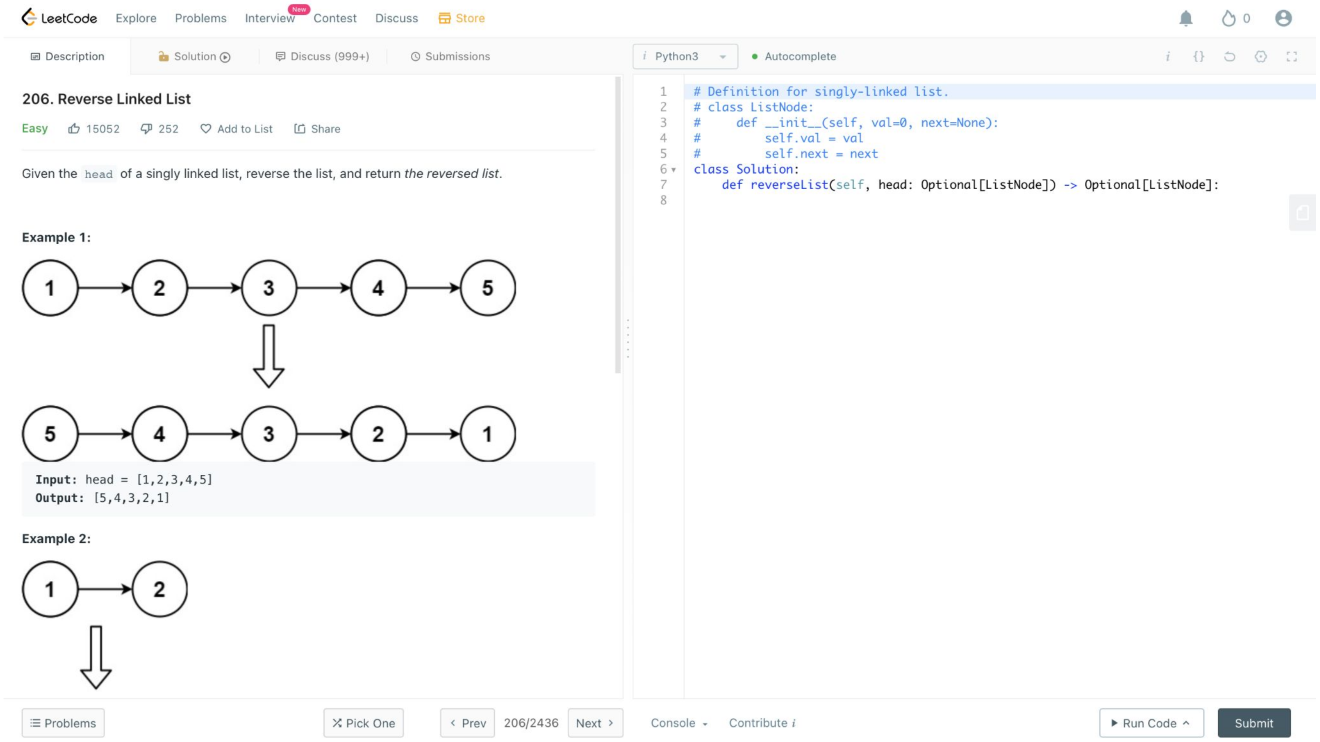
Map out your solution before you write any code. Avoid too much time trying to find the perfect solution. Validate your solution early and often.

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



206. Reverse Linked List

Easy 15052 252 Add to List Share

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]
```

Example 2:

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

```

NYU | TANDON

PROBLEM 1

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
```

- While traversing the list, we can change the current node's next pointer to point to its previous element.

- Since a node does not have reference to its previous node, we must store its previous element beforehand.

- We also need another pointer to store the next node before changing the reference.

- Do not forget to return the new head reference at the end!

Complexity Analysis

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

Space complexity : O(1)

PROBLEM 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
```

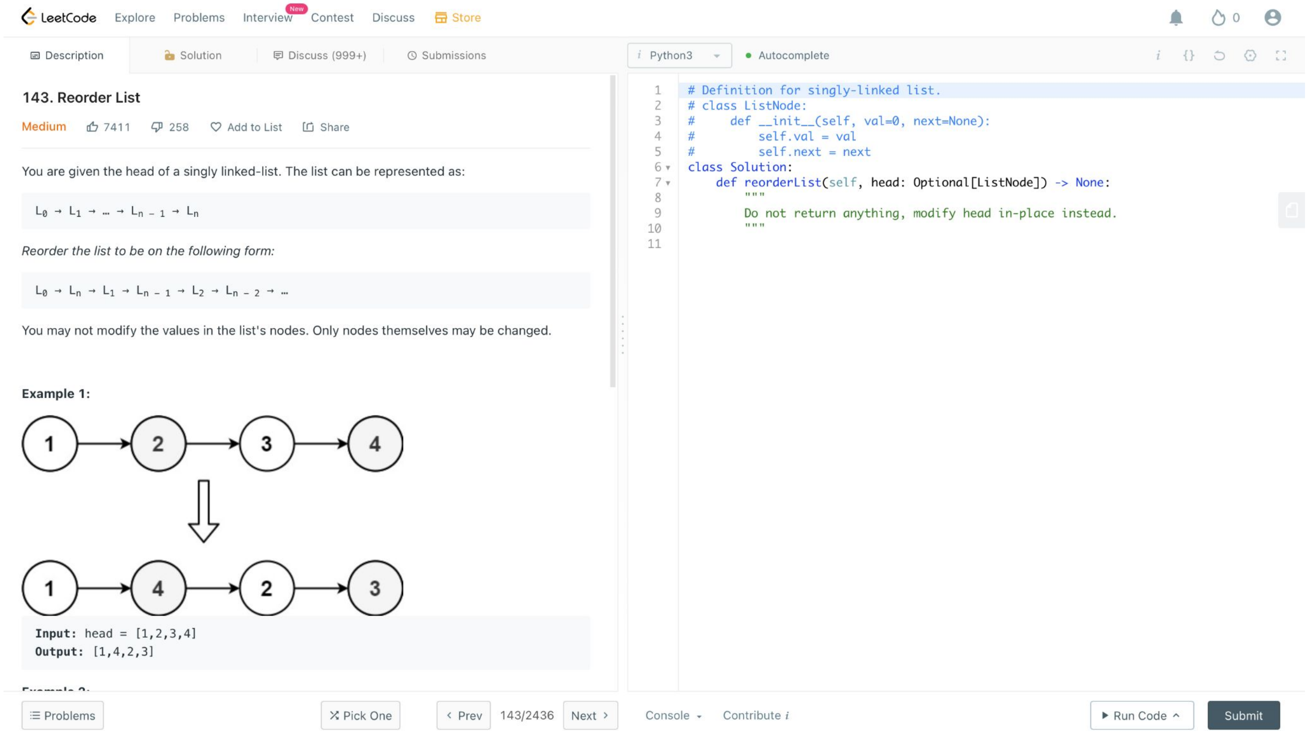
Complexity Analysis

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.

NYU | TANDON

Problem 2: Reorder List



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:

$$\begin{tikzcd} 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \end{tikzcd}$$

$$\Downarrow$$

$$\begin{tikzcd} 1 \arrow[r] & 4 \arrow[r] & 2 \arrow[r] & 3 \end{tikzcd}$$

```
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.
        """
```

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
```

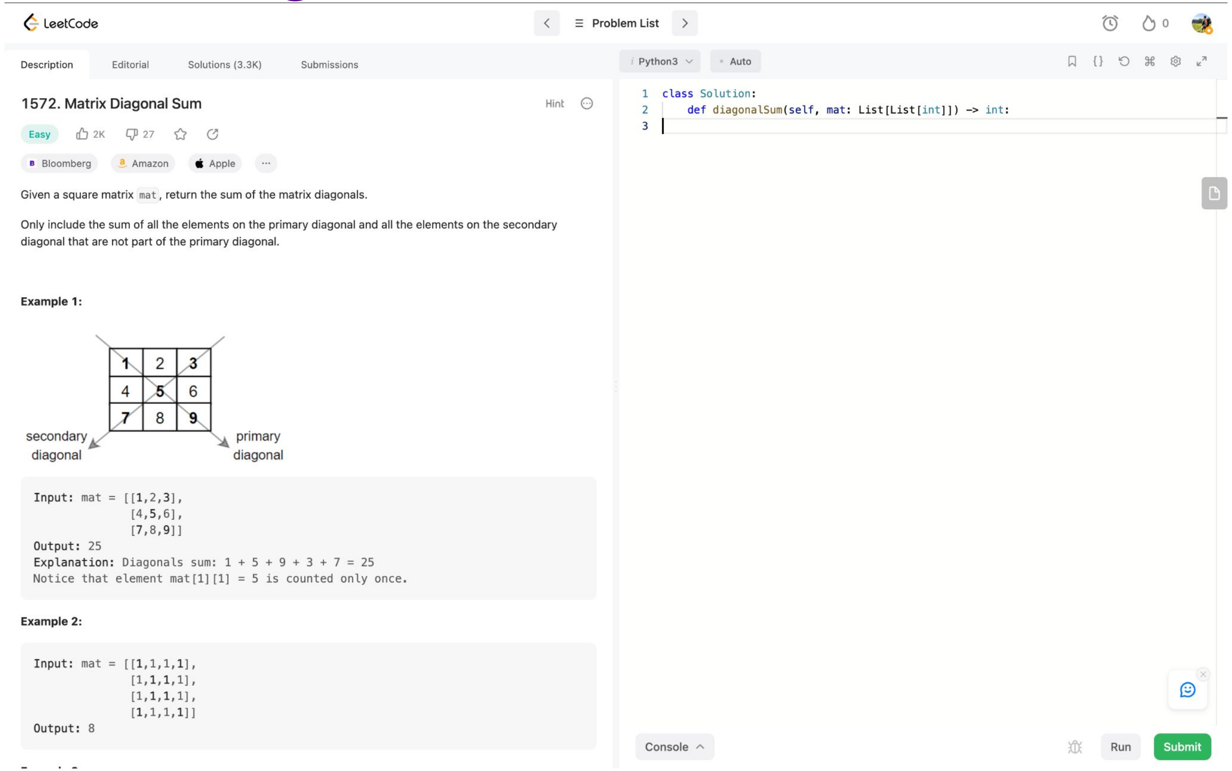
Complexity Analysis

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 02

Problem 3: Matrix diagonal sum

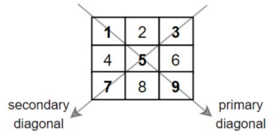


1572. Matrix Diagonal Sum

Given a square matrix `mat`, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:



1	2	3
4	5	6
7	8	9

```
Input: mat = [[1,2,3],
              [4,5,6],
              [7,8,9]]
Output: 25
Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25
Notice that element mat[1][1] = 5 is counted only once.
```

Example 2:

```
Input: mat = [[1,1,1,1],
              [1,1,1,1],
              [1,1,1,1],
              [1,1,1,1]]
Output: 8
```

```python
class Solution:
    def diagonalSum(self, mat: List[List[int]]) -> int:

```

PROBLEM 3

Approach:

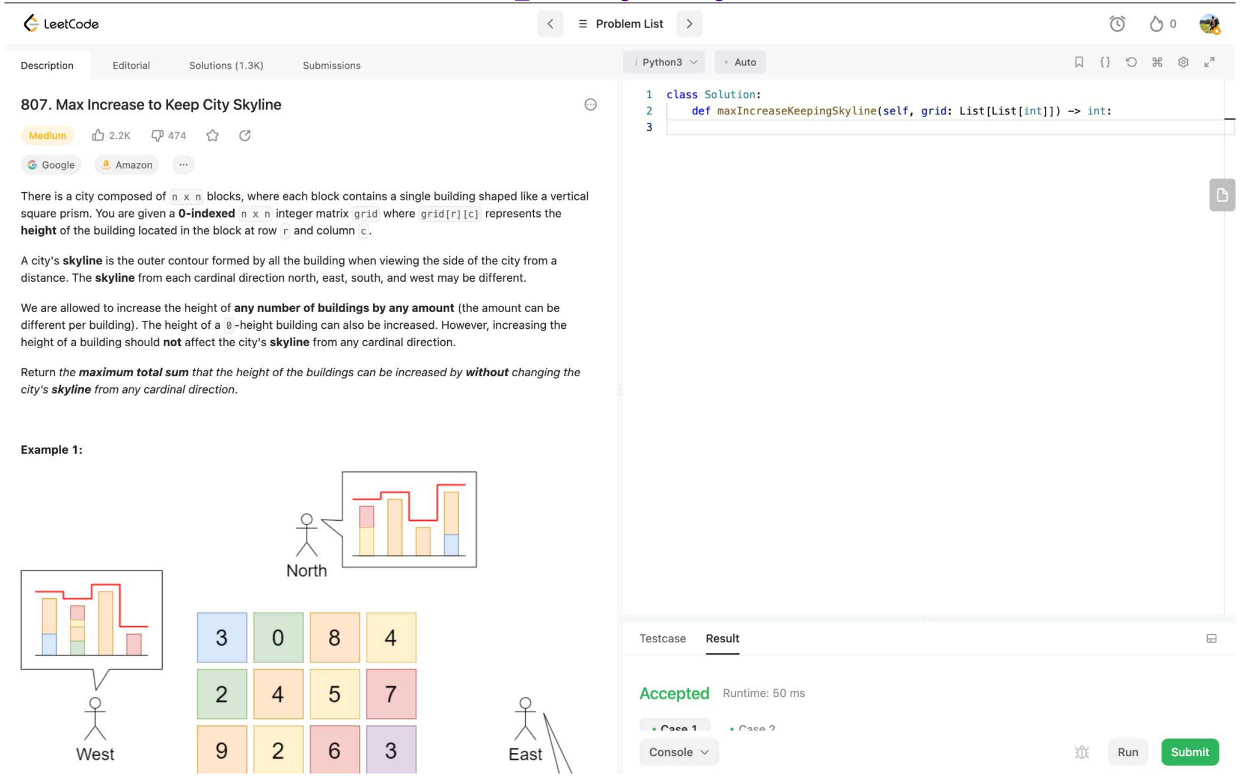
```
def diagonalSum(self, mat: List[List[int]]) -> int:
    n = len(mat)
    ans = 0

    for i in range(n):
        # Add elements from primary diagonal.
        ans += mat[i][i]
        # Add elements from secondary diagonal.
        ans += mat[n - 1 - i][i]
    # If n is odd, subtract the middle element as its added twice.
    if n % 2 != 0:
         ans -= mat[n // 2][n // 2]

    return ans
```

Complexity Analysis

Time complexity: $O(N)$, iterating over primary and secondary diagonals.
Space complexity: O(1), the space used by array.

NYU | TANDON

Problem 4: Max increase to keep city skyline



LeetCode

Description Editorial Solutions (1.3K) Submissions

807. Max Increase to Keep City Skyline

Medium 2.2K 474

Google Amazon

There is a city composed of `n x n` blocks, where each block contains a single building shaped like a vertical square prism. You are given a **0-indexed** `n x n` integer matrix `grid` where `grid[r][c]` represents the **height** of the building located in the block at row `r` and column `c`.

A city's **skyline** is the outer contour formed by all the building when viewing the side of the city from a distance. The **skyline** from each cardinal direction north, east, south, and west may be different.

We are allowed to increase the height of **any number of buildings by any amount** (the amount can be different per building). The height of a `0`-height building can also be increased. However, increasing the height of a building should **not** affect the city's **skyline** from any cardinal direction.

Return *the **maximum total sum** that the height of the buildings can be increased by **without** changing the city's **skyline** from any cardinal direction.*

Example 1:

Python3 Auto

```python
class Solution:
    def maxIncreaseKeepingSkyline(self, grid: List[List[int]]) -> int:

```

Testcase Result

Accepted Runtime: 50 ms

Case 1 Case 2

Console Run Submit

PROBLEM 4

Approach:

```
def maxIncreaseKeepingSkyline(self, grid: List[List[int]]) -> int:
    row_maxes = [max(row) for row in grid]
    col_maxes = [max(col) for col in zip(*grid)]

    return sum(min(row_maxes[r], col_maxes[c]) - val
            for r, row in enumerate(grid)
            for c, val in enumerate(row))
```

Complexity Analysis

Time complexity: *O*(*N*^2), iterating through every cell of the grid.
Space complexity: O(N), the space used by row_maxes and col_maxes.

NYU | TANDON

PART 06

Q/A

Problem Assignments

01. Linked List Cycle (Easy)

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

03. Rotate Image (Medium)

04. Set Matrix Zeroes (Medium)

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

06. Minimum Path Sum (Medium)

07. Merge k Sorted Lists (Hard)

NYU | TANDON

Thank you!

Upcoming: Graph, Stack, Queue