Contents

[Array 3](#_Toc104301463)

[560. Subarray Sum Equals K 3](#_Toc104301464)

[1. Two Sums: 4](#_Toc104301465)

[523. Continuous Subarray Sum 5](#_Toc104301466)

[31. Next Permutation 6](#_Toc104301467)

[670. Maximum Swap 6](#_Toc104301468)

[56. Merge Intervals 7](#_Toc104301469)

[986. Interval List Intersections 8](#_Toc104301470)

[88. Merge Sorted Array 10](#_Toc104301471)

[347. Top K Frequent elements 11](#_Toc104301472)

[215. Kth largest Element in an Array 11](#_Toc104301473)

[713. subarray product less than k 12](#_Toc104301474)

[162. Find Peak Element 13](#_Toc104301475)

[71. Simplify Path 14](#_Toc104301476)

[238.Product of Array Except Self 15](#_Toc104301477)

[973.K Closest Points to Origin 16](#_Toc104301478)

[1570. Dot Product of Two Sparse Vectors 17](#_Toc104301479)

[1762. Buildings with an Ocean view 18](#_Toc104301480)

[1891. Curring Ribbons 19](#_Toc104301481)

[String 20](#_Toc104301482)

[953. Verifying an Alien Dictionary 20](#_Toc104301483)

[249. Group Shifted Strings 20](#_Toc104301484)

[249. Group Shifted String 21](#_Toc104301485)

[1249. Minimum Remove to Make Valid Parentheses 22](#_Toc104301486)

[921. Minimum Add to make Parenthese Valid 23](#_Toc104301487)

[125. Valid Palindrome 24](#_Toc104301488)

[680. Valid Palindrome II 24](#_Toc104301489)

[227. Basic Calculator II 25](#_Toc104301490)

[408. Valid Word Abbreviation 25](#_Toc104301491)

[415. Add Strings 26](#_Toc104301492)

[791. Custom Sort String 26](#_Toc104301493)

[246. Strobogrammatic Number 27](#_Toc104301494)

[Binary Trees 28](#_Toc104301495)

[314. Binary Tree Vertical Order Traversal 28](#_Toc104301496)

[1650. Lowest Common Ancestor of a Binary Tree III 29](#_Toc104301497)

[938. Range Sum of BST 29](#_Toc104301498)

[236. Lowest Common Ancestor of a Binary Tree 30](#_Toc104301499)

[199. Binary Tree Right Side View 31](#_Toc104301500)

[543. Diameter of Binary Tree 32](#_Toc104301501)

[270. Closest Binary Search Tree Value 33](#_Toc104301502)

[Depth First Search and Breath First Search 34](#_Toc104301503)

[721. Accounts Merge 34](#_Toc104301504)

[Structure Design and Linked List 35](#_Toc104301505)

[146. LRU Cache 35](#_Toc104301506)

[234. Palindrome Linked List 37](#_Toc104301507)

[346. Moving Average from Data Stream 38](#_Toc104301508)

[1570. Dot product of two sparse vectors 39](#_Toc104301509)

[Matrix 40](#_Toc104301510)

[766. Teoplitz Matrix 40](#_Toc104301511)

[498. Diagonal Traverse 41](#_Toc104301512)

[1091. Shortest Path in Binary Matrix 41](#_Toc104301513)

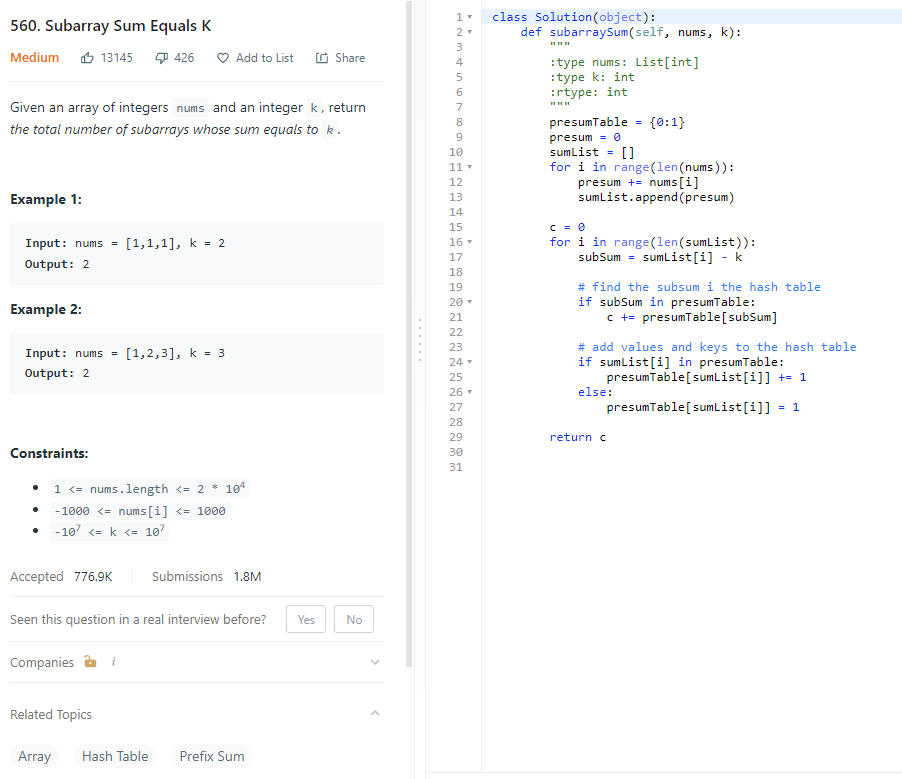
[Recursion: 42](#_Toc104301514)

[339. Nested List Weight Sum 42](#_Toc104301515)

[50. Pow(x,n) 42](#_Toc104301516)

# Array

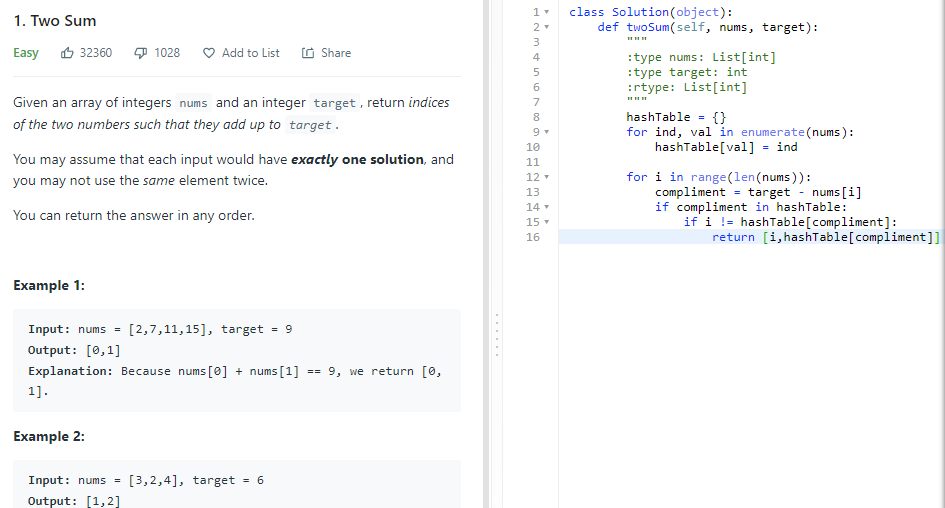
## 560. Subarray Sum Equals K



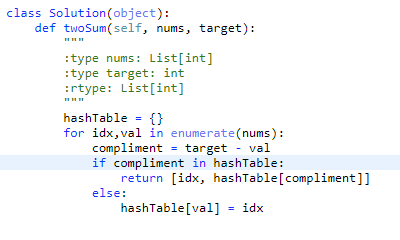
Time: O(n), only traversed the list once

Space: O(n), extra space for the hashtable

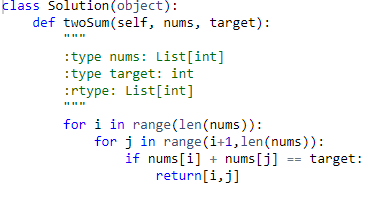
## Two Sums:



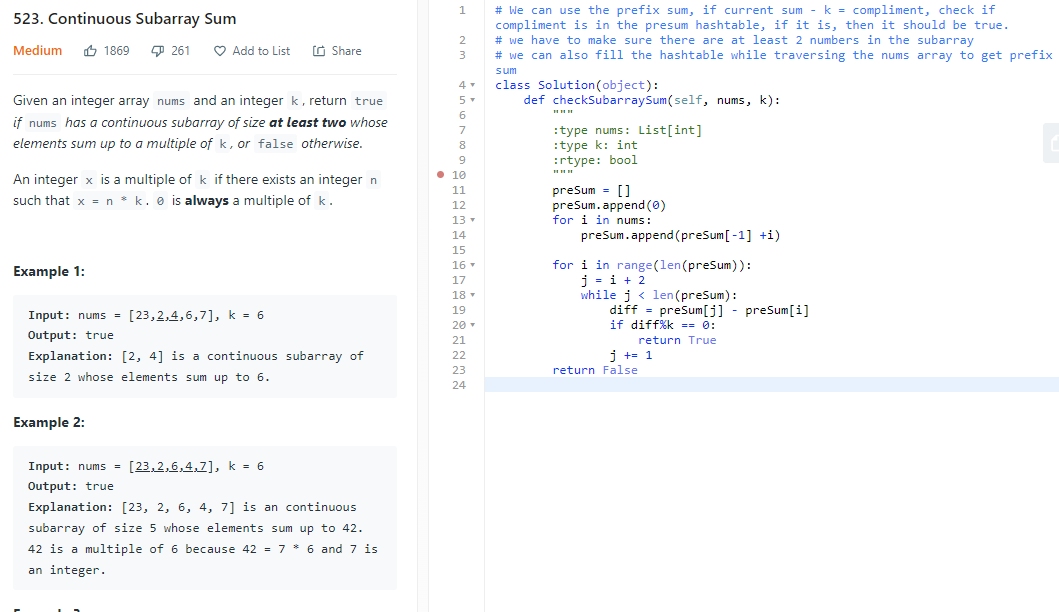
Optimized



Brute Force



## 523. Continuous Subarray Sum

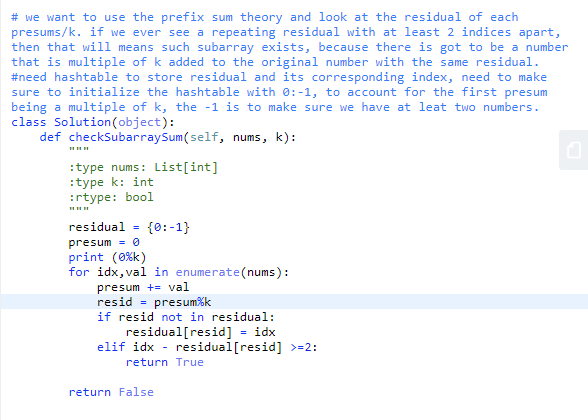


Brute force:



Time: O(n2)

See next page for best solution



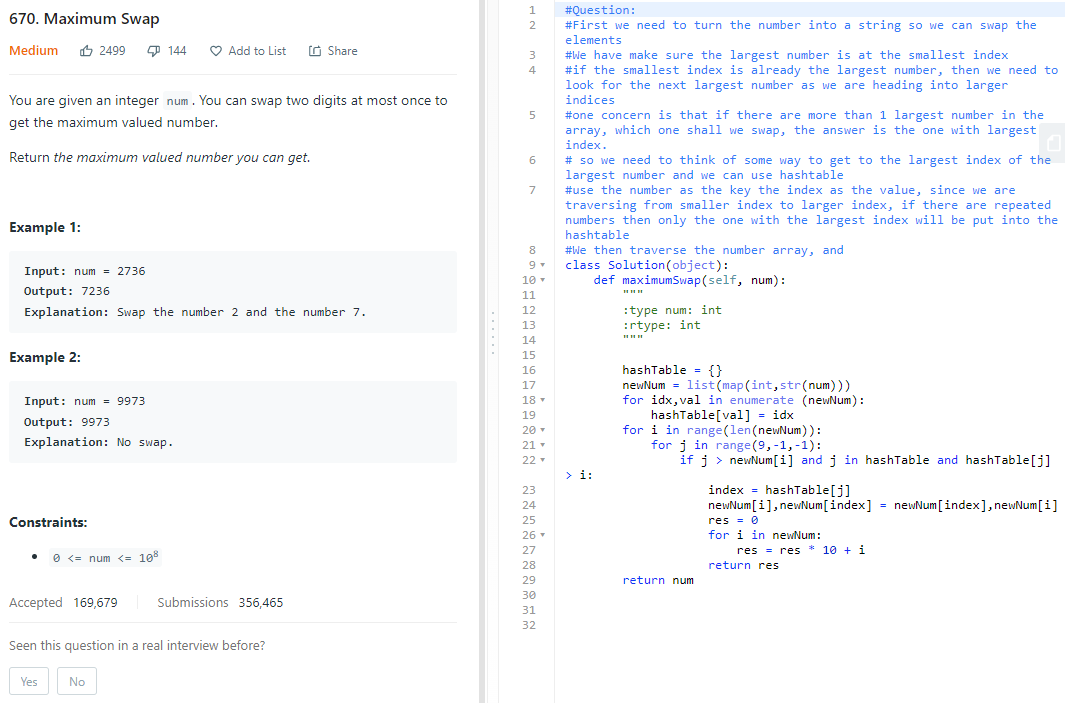
Time:O(n)

Space: O(n)

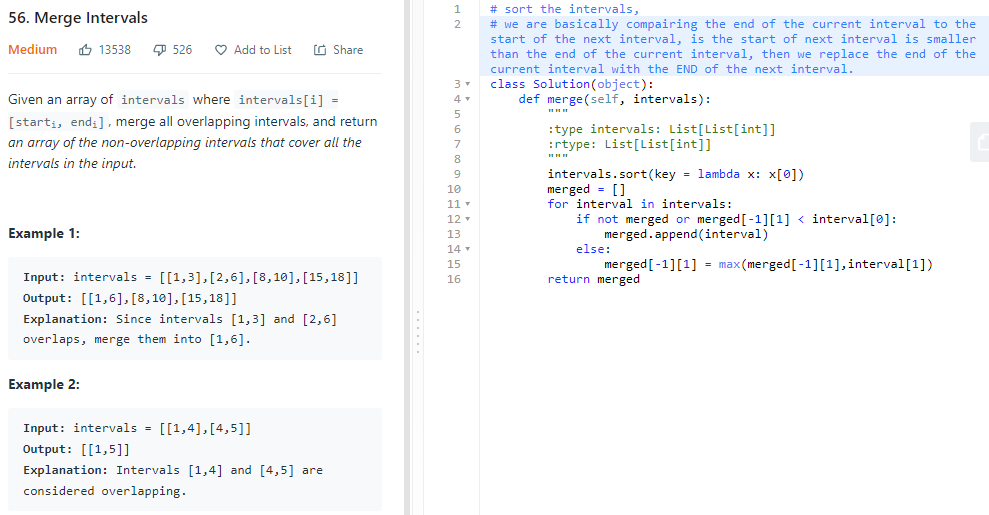
## 31. Next Permutation



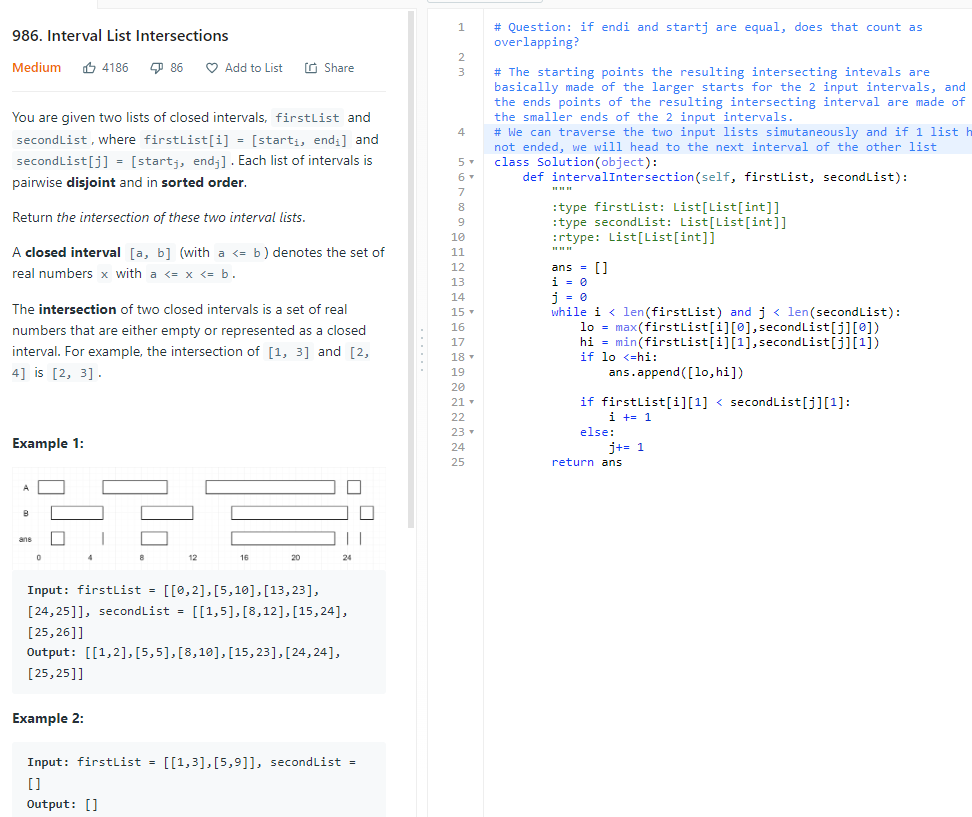
## 670. Maximum Swap



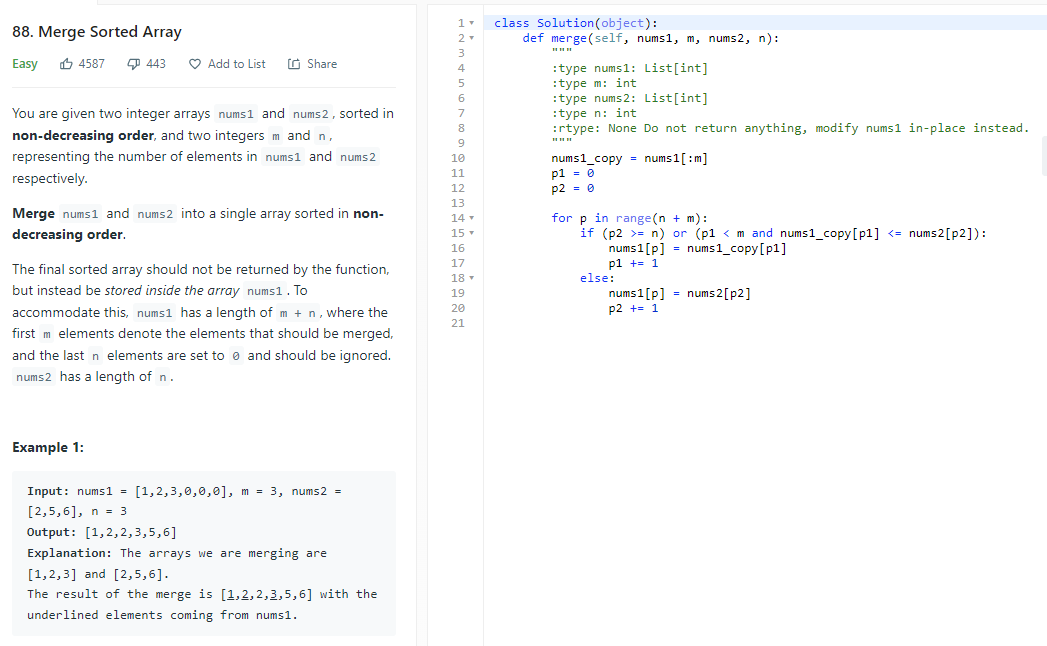
## 56. Merge Intervals



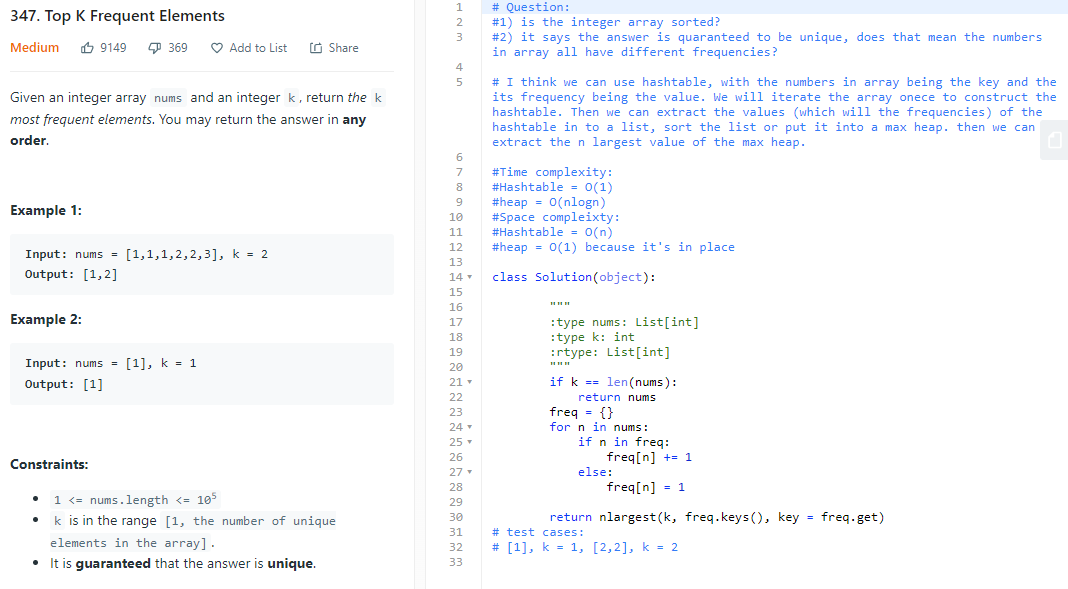
## 986. Interval List Intersections

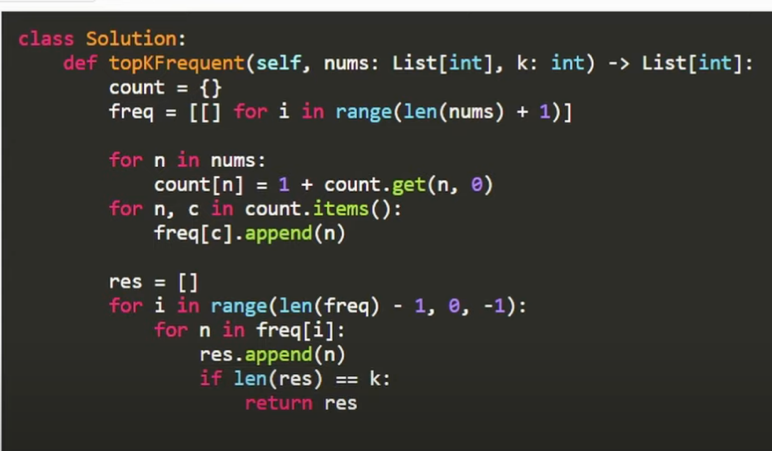


## 88. Merge Sorted Array



## 347. Top K Frequent elements





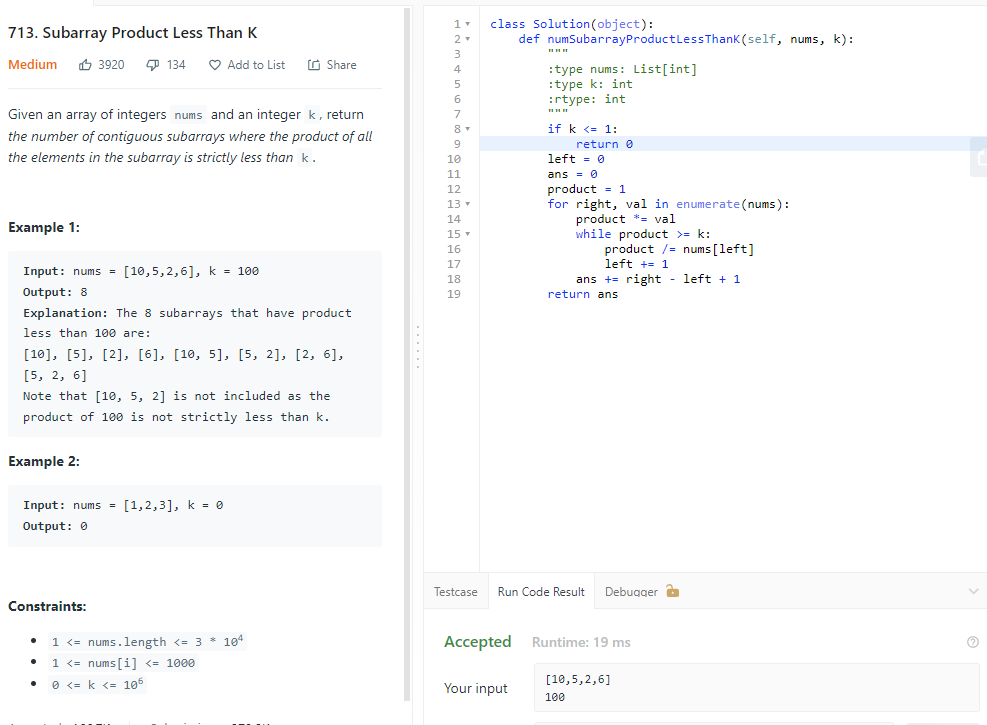
## 215. Kth largest Element in an Array



Time: Average O(n), worst O(n2)

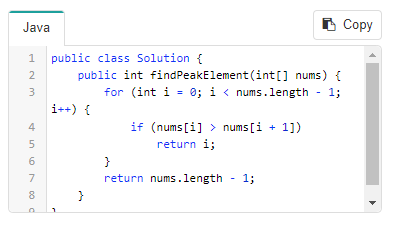
Space: O(n), recursive

## 713. subarray product less than k



## 162. Find Peak Element





## 71. Simplify Path



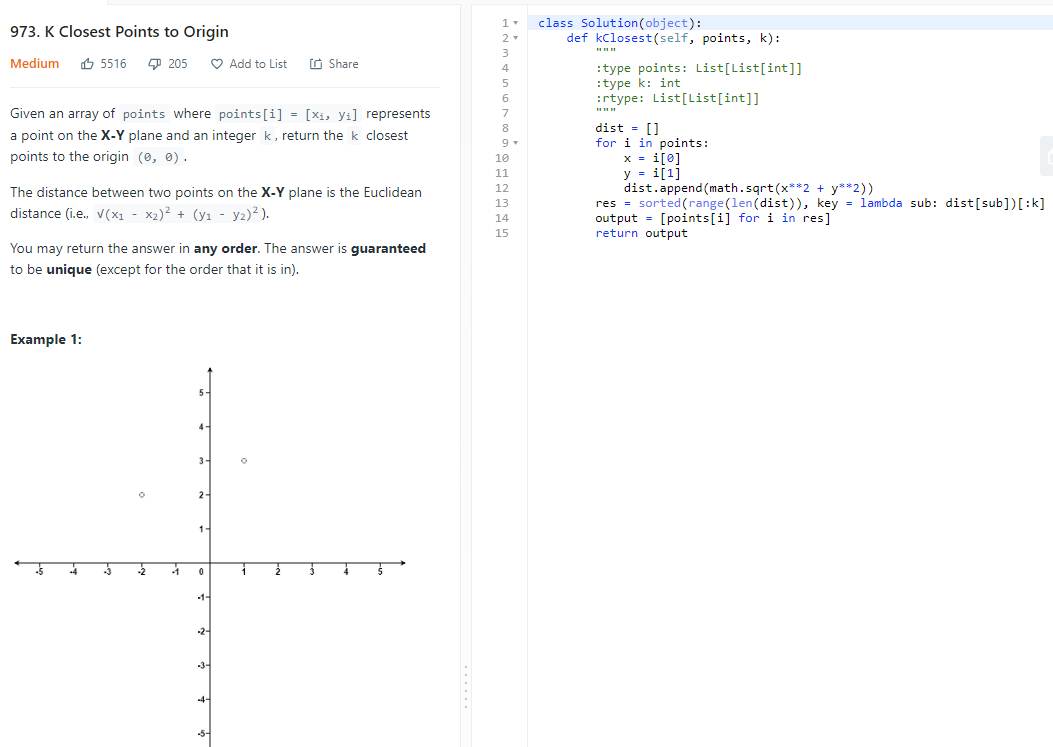
Time: O(n)

Space: O(n)

## 238.Product of Array Except Self

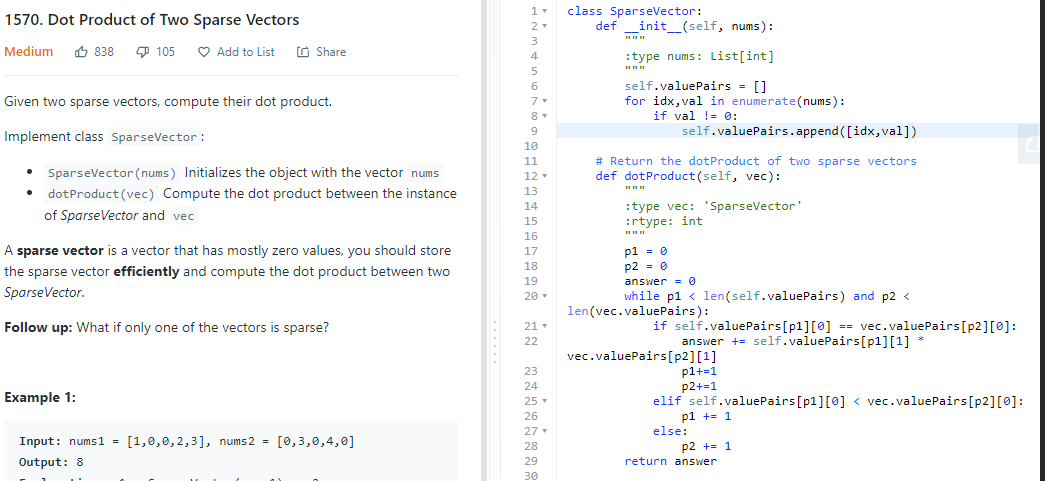


## 973.K Closest Points to Origin



## 1570. Dot Product of Two Sparse Vectors

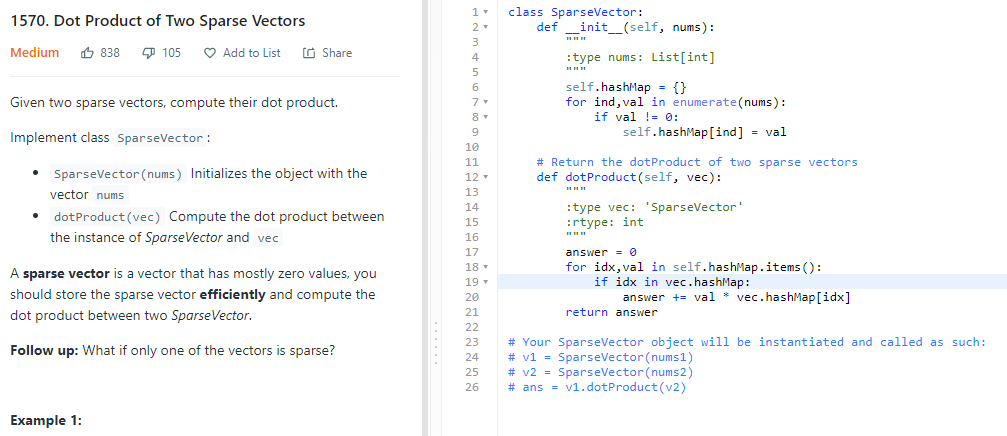
2 pointers



Time: max(O(n),O(v)), v is the number of non-zero items, O(v) is for calculation of dot products

Space: O(L), L is the number of nonzero items

HashMap:



Time: max(O(n),O(v)), v is the number of non-zero items, O(v) is for calculation of dot products

Space: O(L), L is the number of nonzero items

## 1762. Buildings with an Ocean view



Time: O(n)

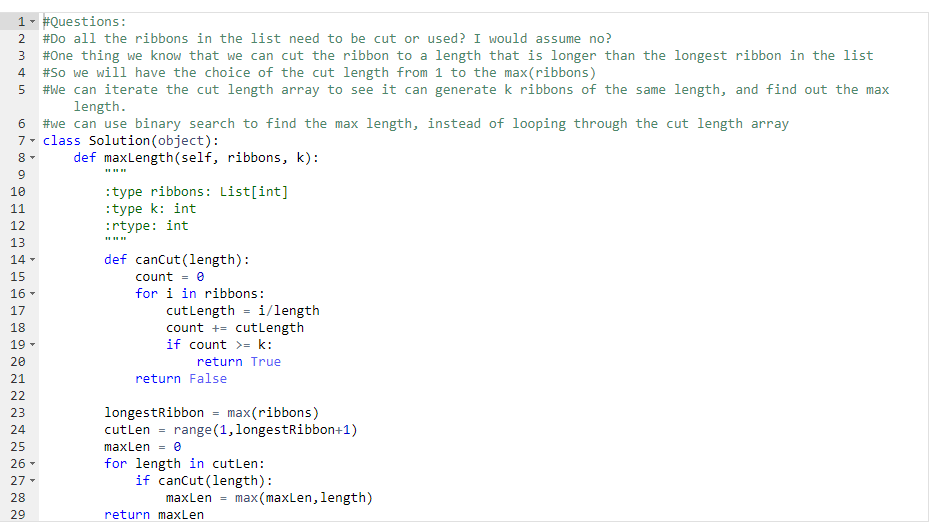
Space:O(n)

## 1891. Curring Ribbons



Time:O(logn), space:O(1)

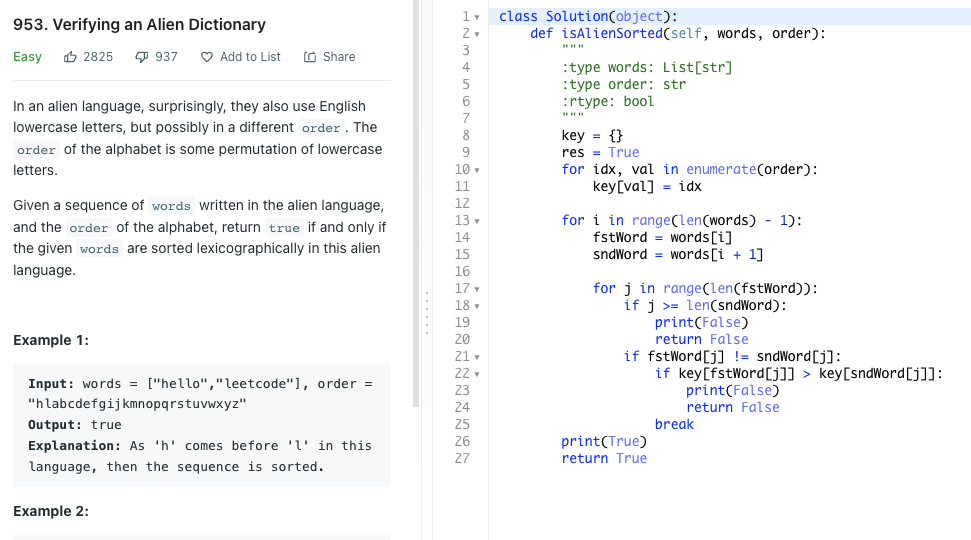
Brute force:



Time O(n2), spaceO(1)

# String

## 953. Verifying an Alien Dictionary



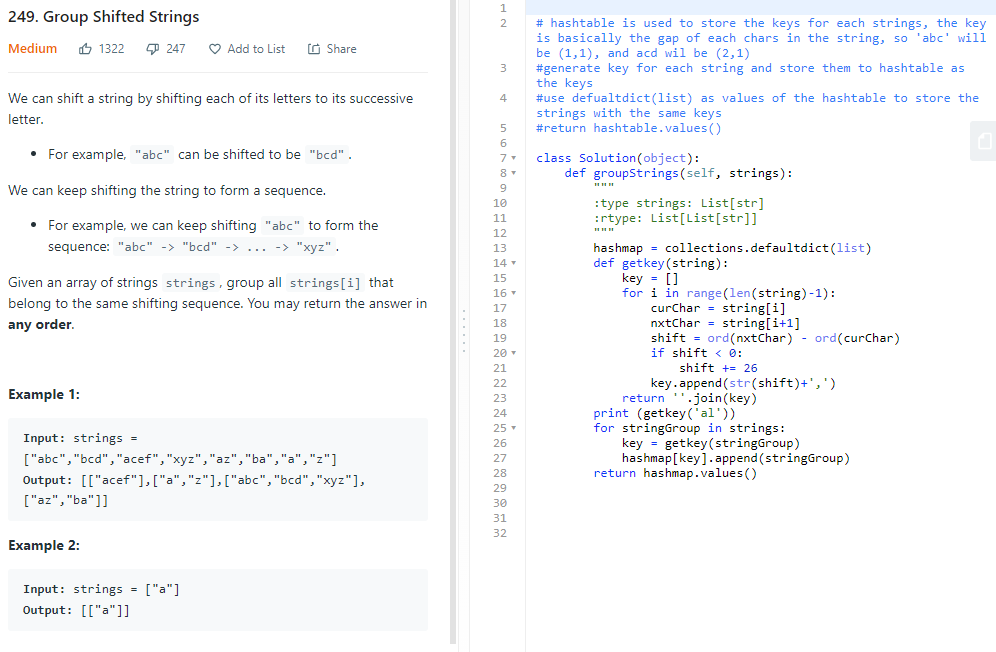
## 249. Group Shifted Strings



Time:O(n\*m), n is the number so strings, m is the number of char in the string

Space: O(n\*m), n is the number of strings

## 249. Group Shifted String



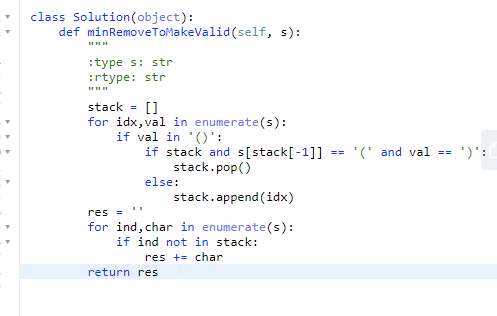
Time:O(n\*m), n is the number of strings, m is the max length of a string

Space: O(nm), same idea as above

## 1249. Minimum Remove to Make Valid Parentheses



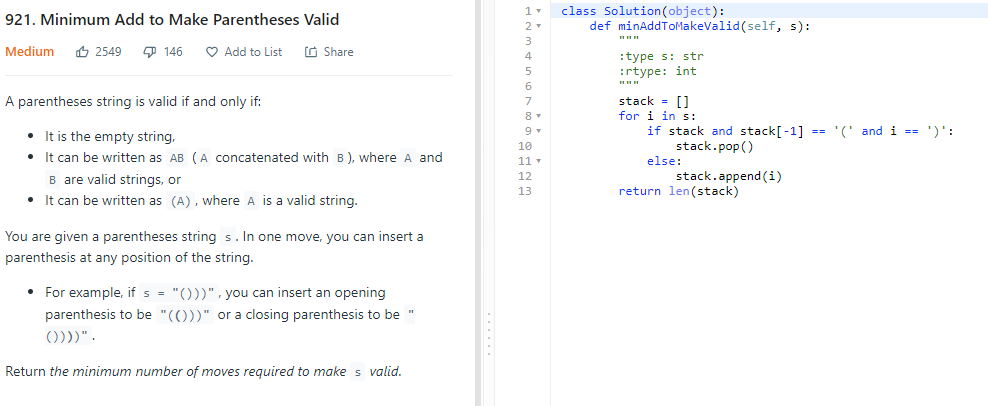
Best solution!!



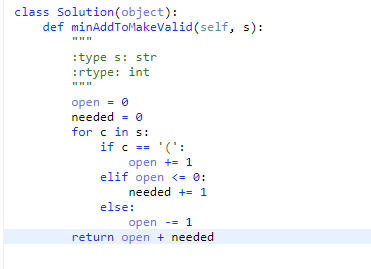
Time: O(n)

Space: O(n)

## 921. Minimum Add to make Parenthese Valid

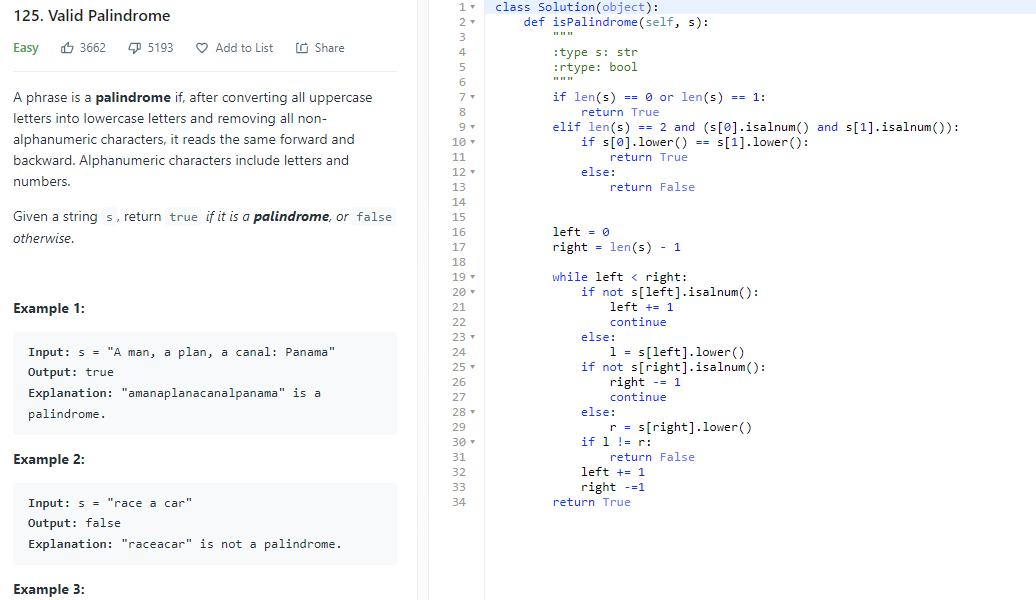


Time: O(n), space O(n)

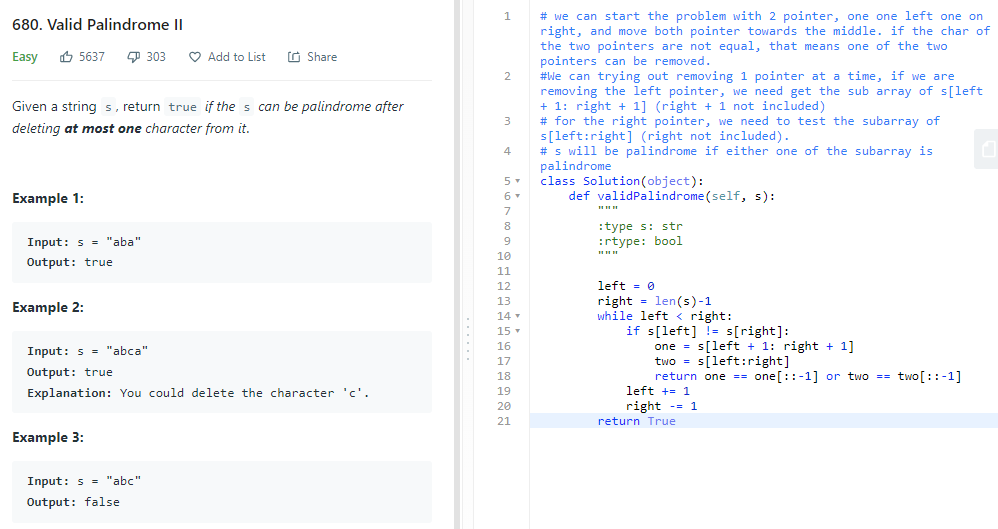


Time:O(n), space:O(1)

## 125. Valid Palindrome



## 680. Valid Palindrome II



Time: O(n)

Space: O(n) because I’m make copies of the array

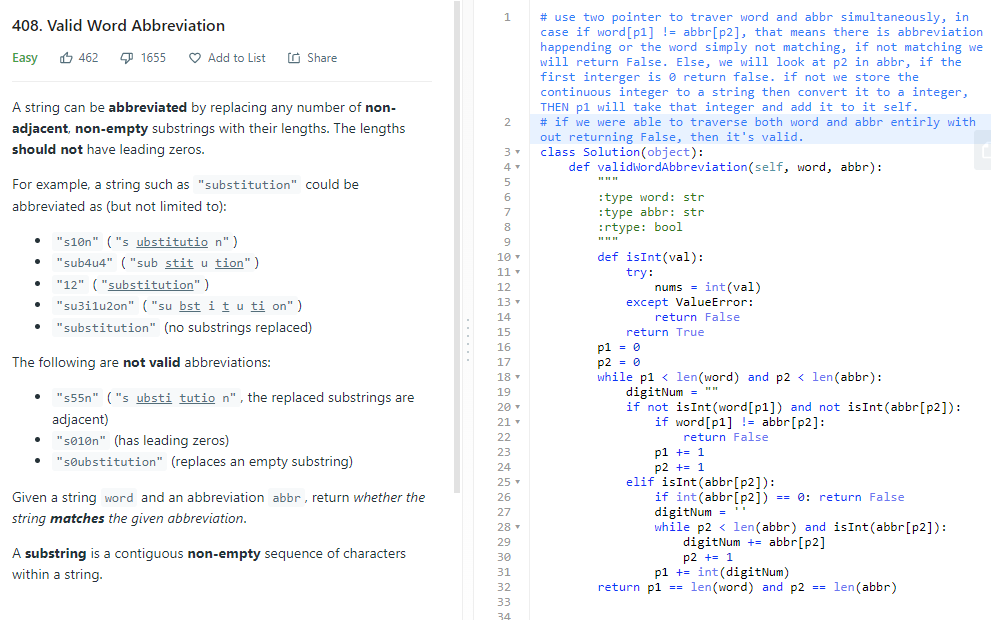
## 227. Basic Calculator II



Time: O(n)

Space: O(n), using the stack

## 408. Valid Word Abbreviation



Time: O(N+M), word string and abbr string, Space:O(1)

## 415. Add Strings



Time: O(n), n is max(len(num1,num2).

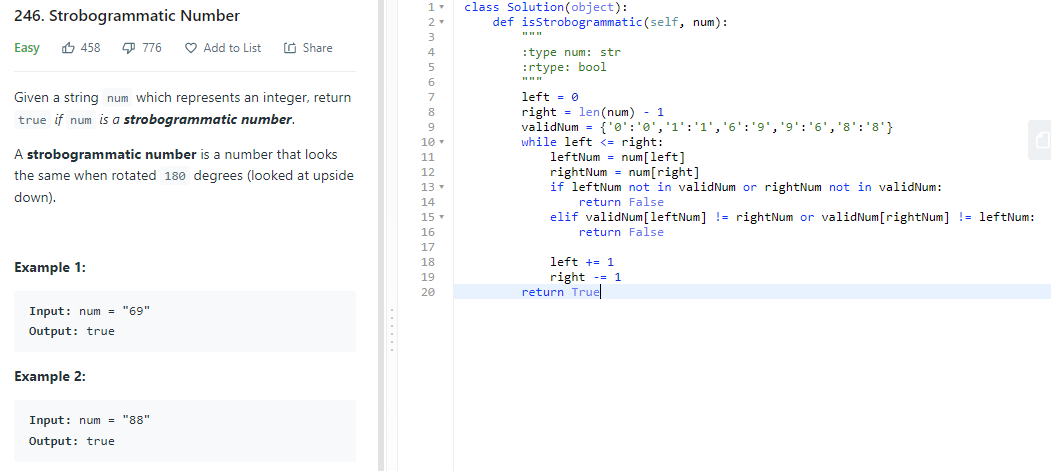
Sapce: O(n), n is max(len(num1,num2). Because the returned resulting array

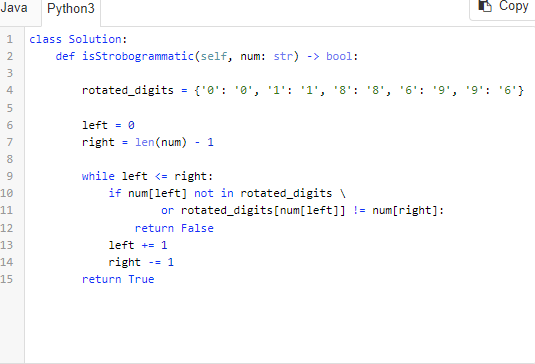
## 791. Custom Sort String



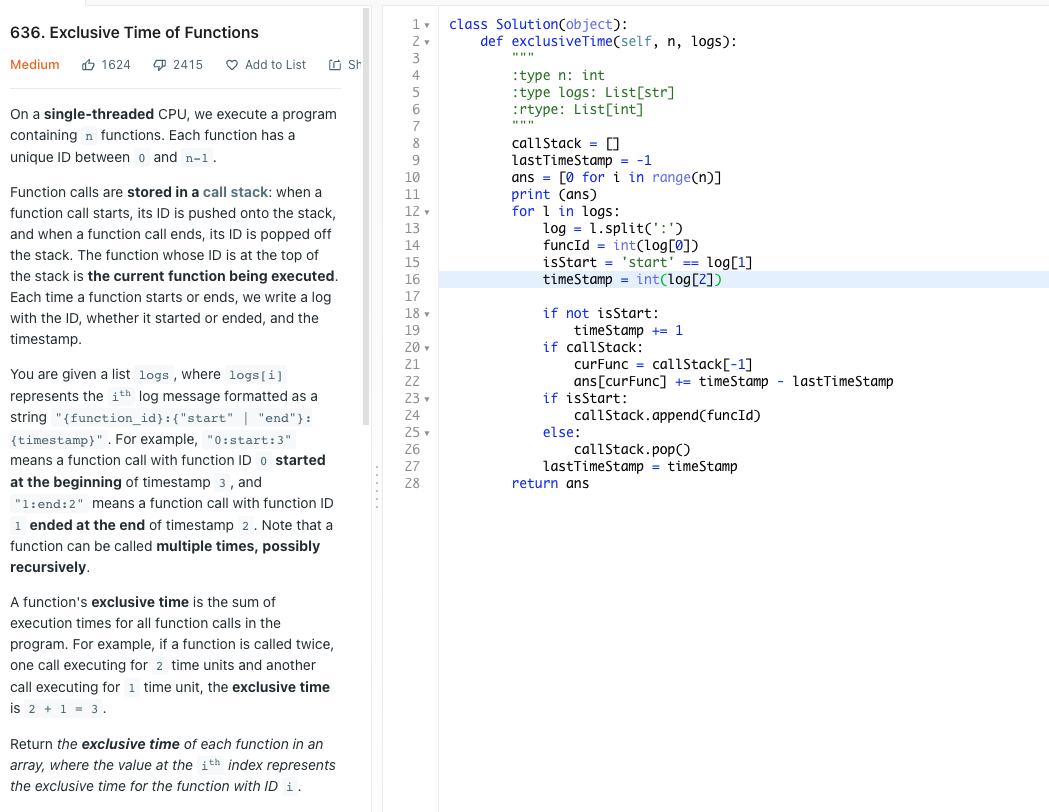
Time: O(len(s)+len(order)), space: O(n), n being the len(s)

## 246. Strobogrammatic Number



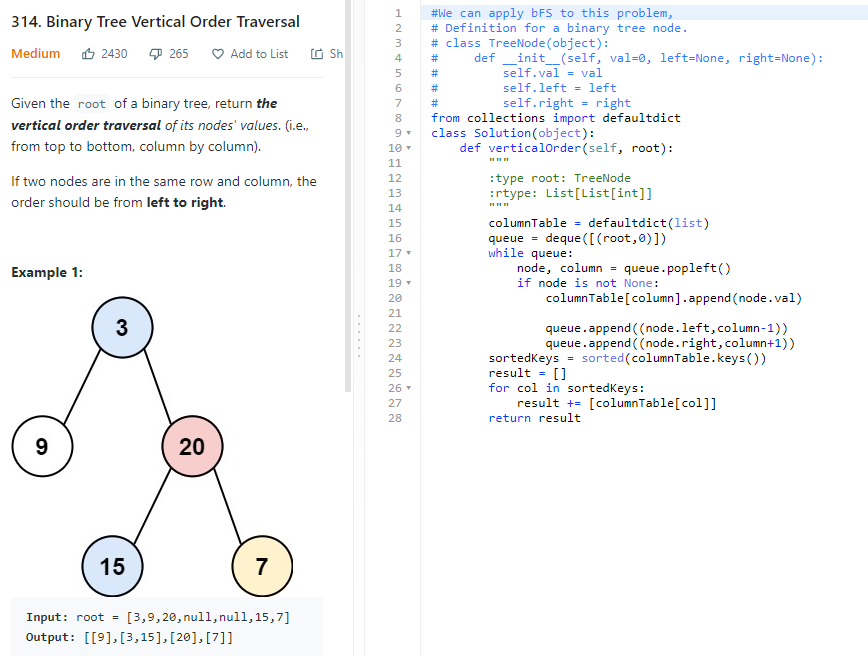


## 636. Exclusive Time of Functions



# Binary Trees

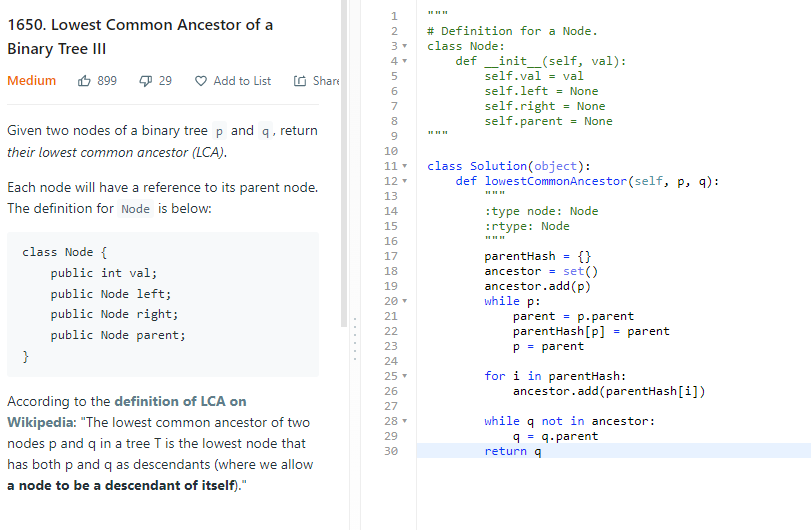
## 314. Binary Tree Vertical Order Traversal



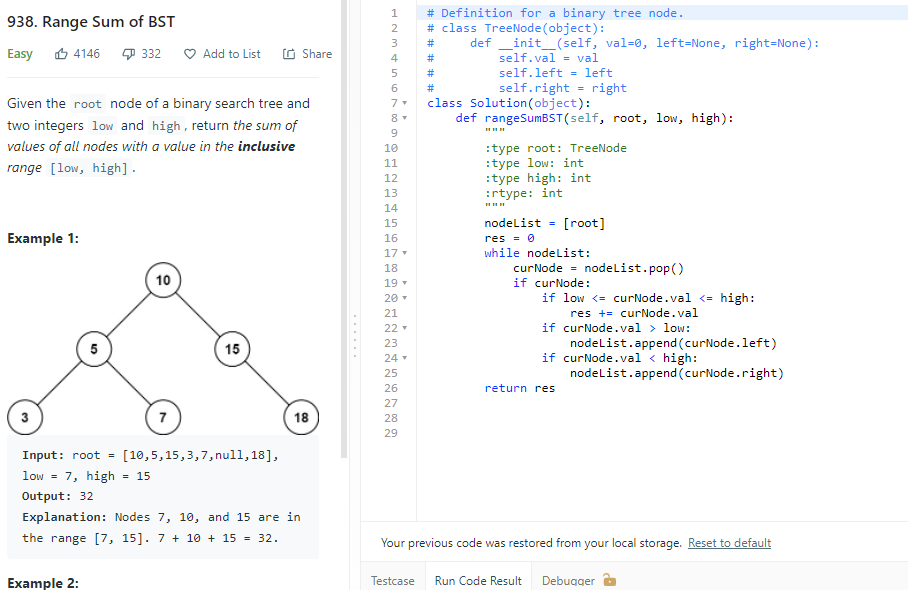
Time: O(n) since we will be visiting all notes, O(nlogn) because of sorting

Space: O(n), think about the hash keys values, the que and the result array

## 1650. Lowest Common Ancestor of a Binary Tree III

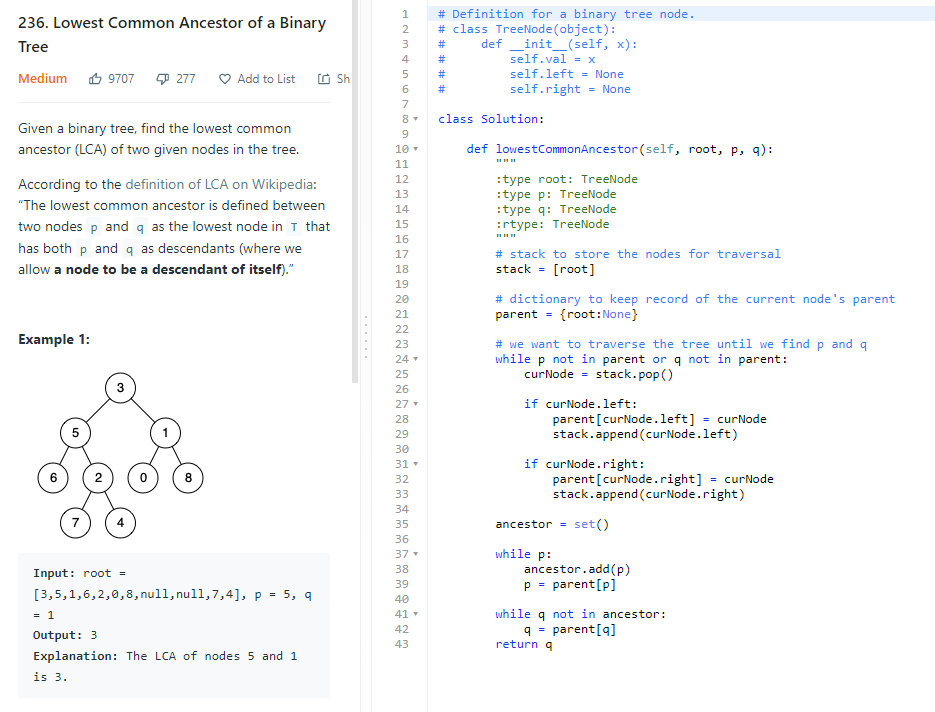


## 938. Range Sum of BST

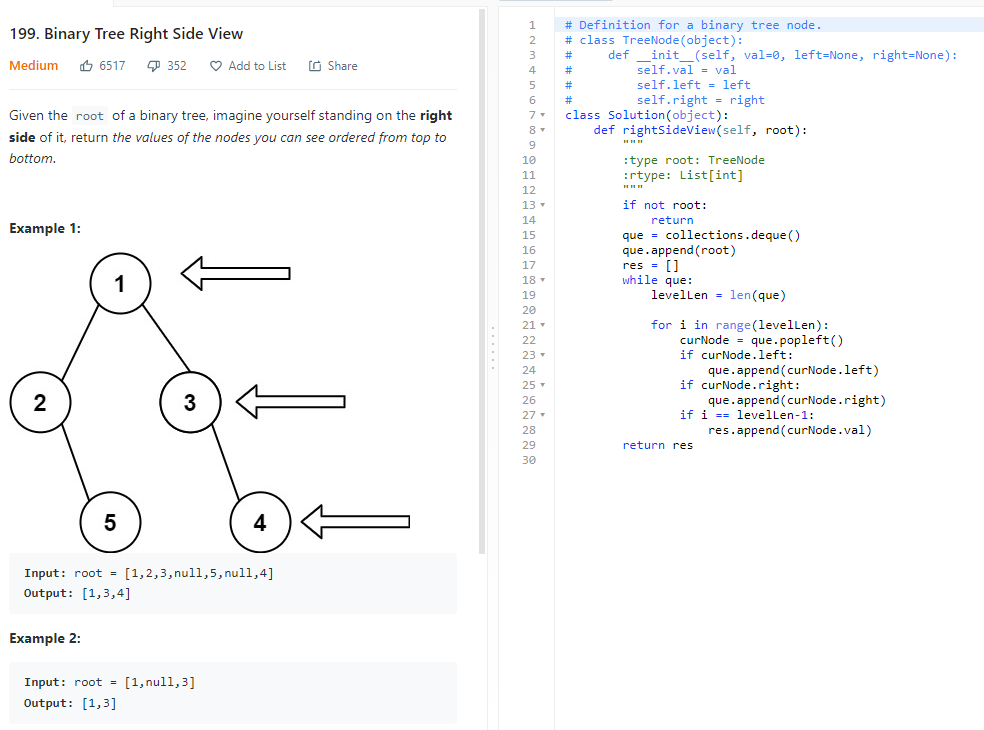


## 236. Lowest Common Ancestor of a Binary Tree

To know the LCA of the two notes, we first have to find out what ancestor do p and q have individually. We can traverse the tree from root node until we find both p and q. and as we were traversing, we will get data on which node is parent or child of which node. We can use a hashTable to do this, the key will be the nodes and values be their parent pointer. We first initialize the hash table with {root: None}, then as we visits the children, their direct parent become the value they correspond to. Once p and q are both found, we can list out all the parents p have in a set, let’s call it parentset, and then we look at q, if q is in the parentset that means q is the LCA. We can keep finding the parents of q using the original hashtable, is p is found to be the parent of q, then p is the LCA, otherwise the LCA is the node at the end of the parentSet.



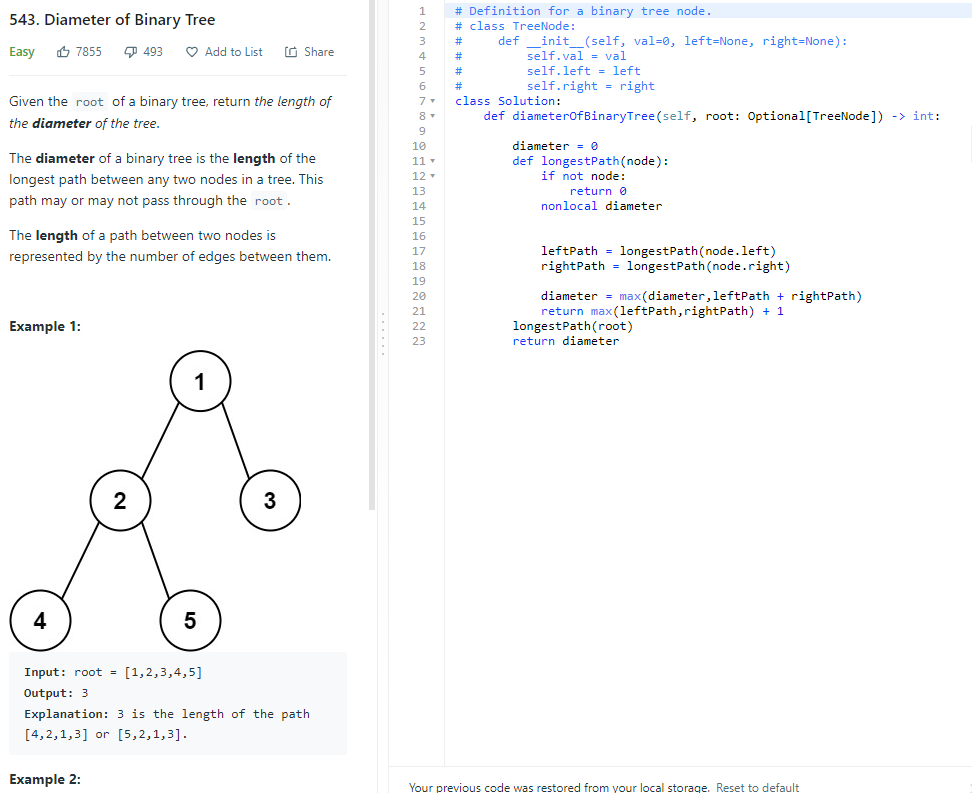
## 199. Binary Tree Right Side View



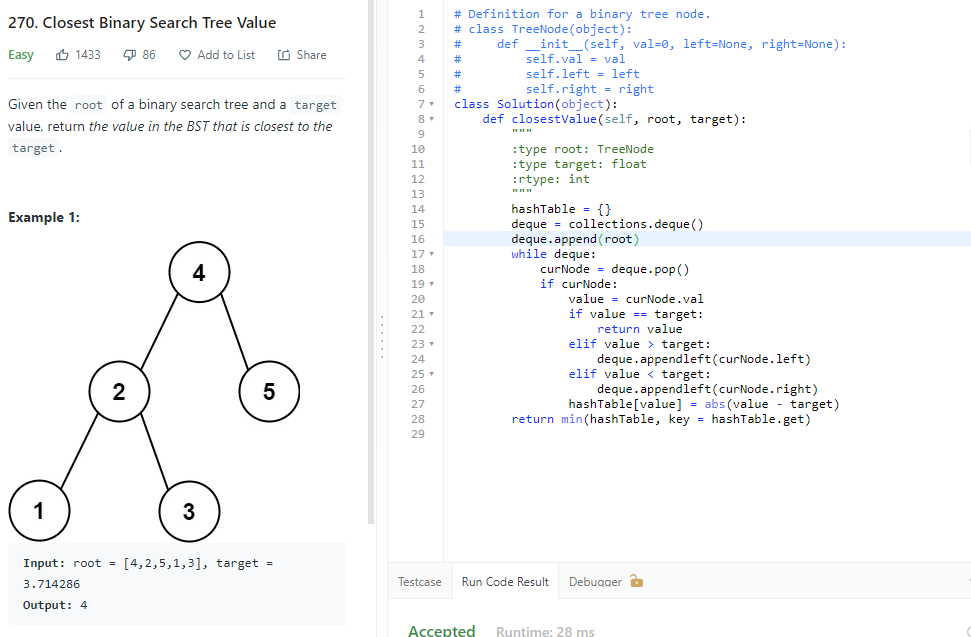
Time: O(n), we may have to traverse all notes

Space: O(width) because of the que, the last level could have N/2 tree nodes

## 543. Diameter of Binary Tree



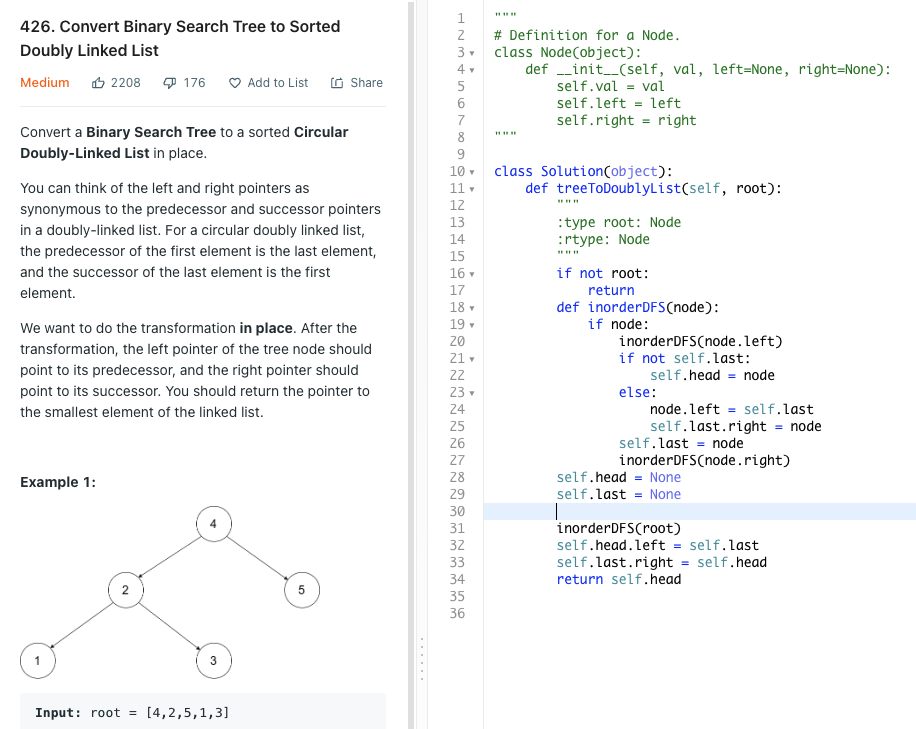
## 270. Closest Binary Search Tree Value



Time: O(n)

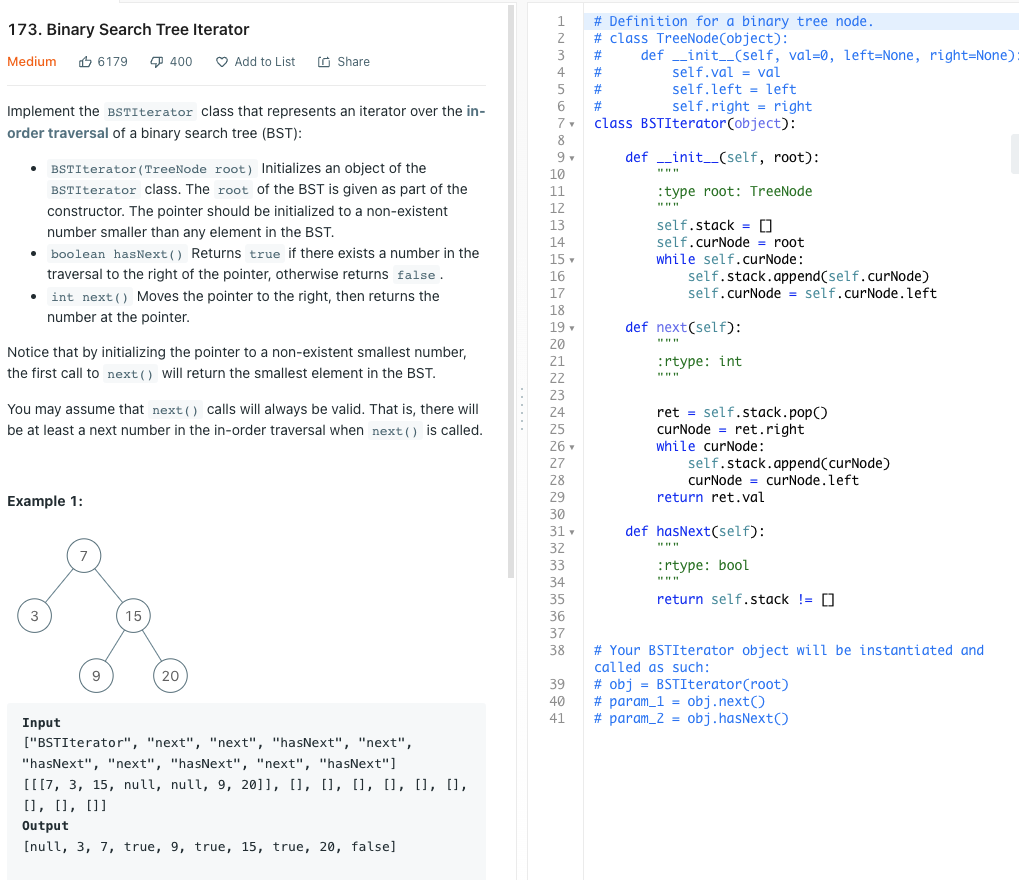
Space:O(n) for the hashtable

## 426. Convert Binary Search Tree to Sorted Doubly Linked List



Time: O(n), space: O(n)

## 173. Binary Search Tree Iterator

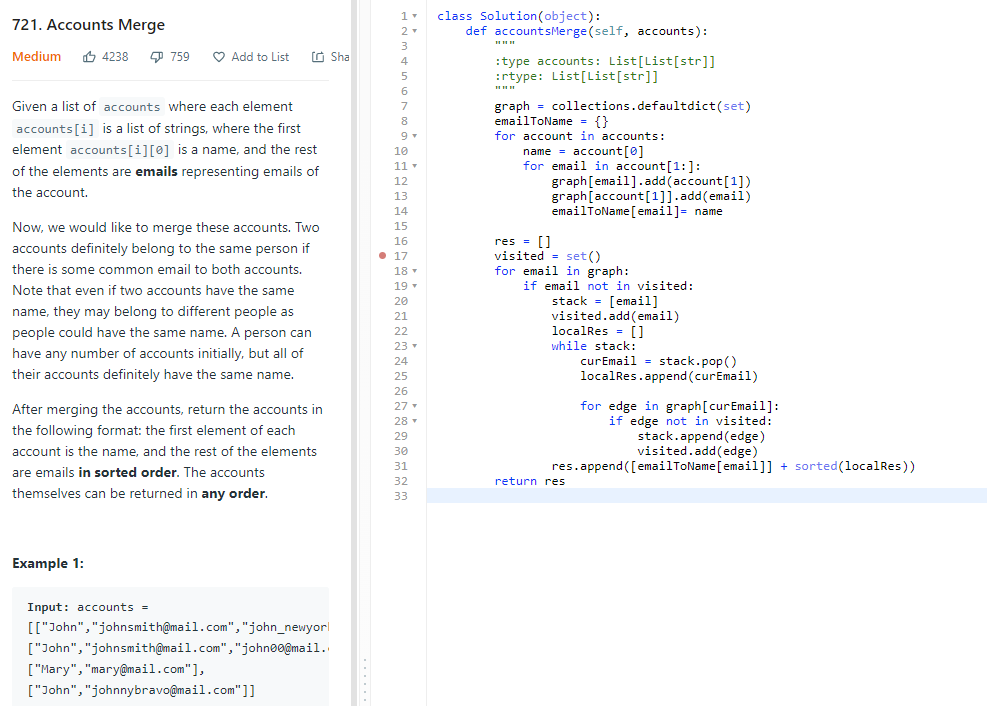


Time: O(1) and O(n) in worse case due to we have to find the left most node

Space:O(n), n is the # of nodes I the tree

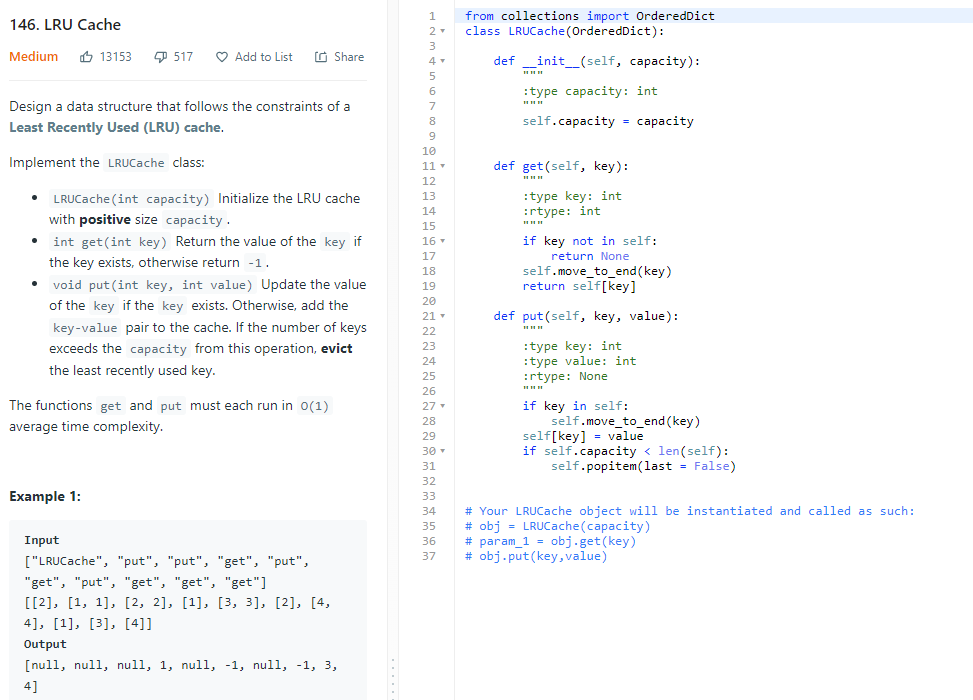
# Depth First Search and Breath First Search

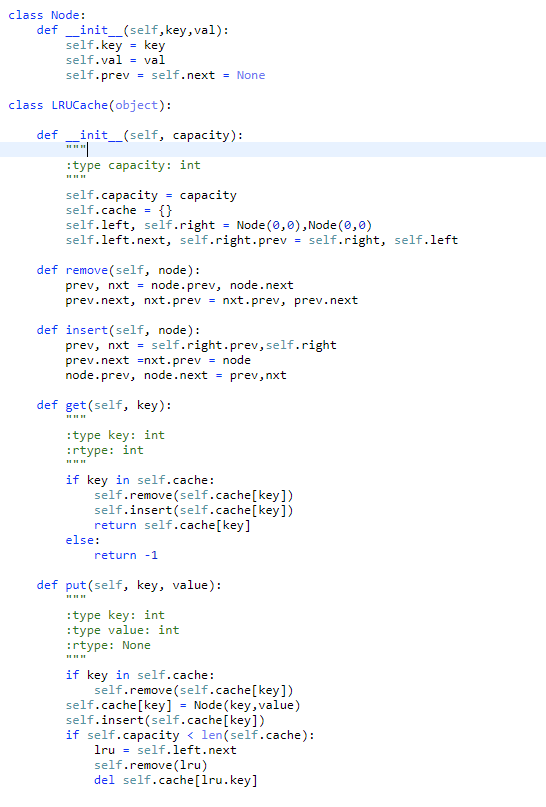
## 721. Accounts Merge



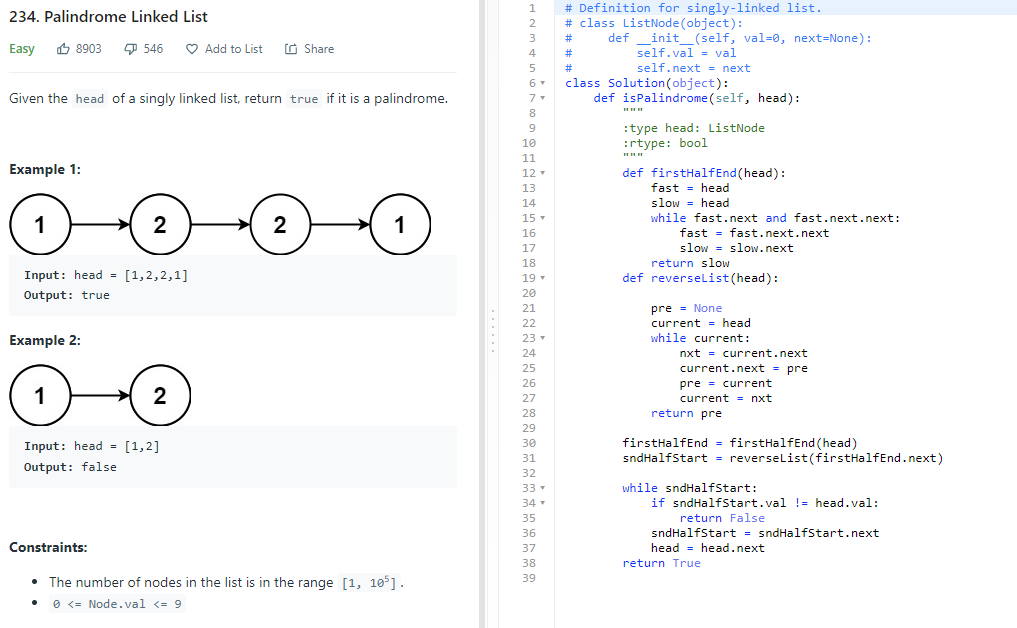
# Structure Design and Linked List

## 146. LRU Cache



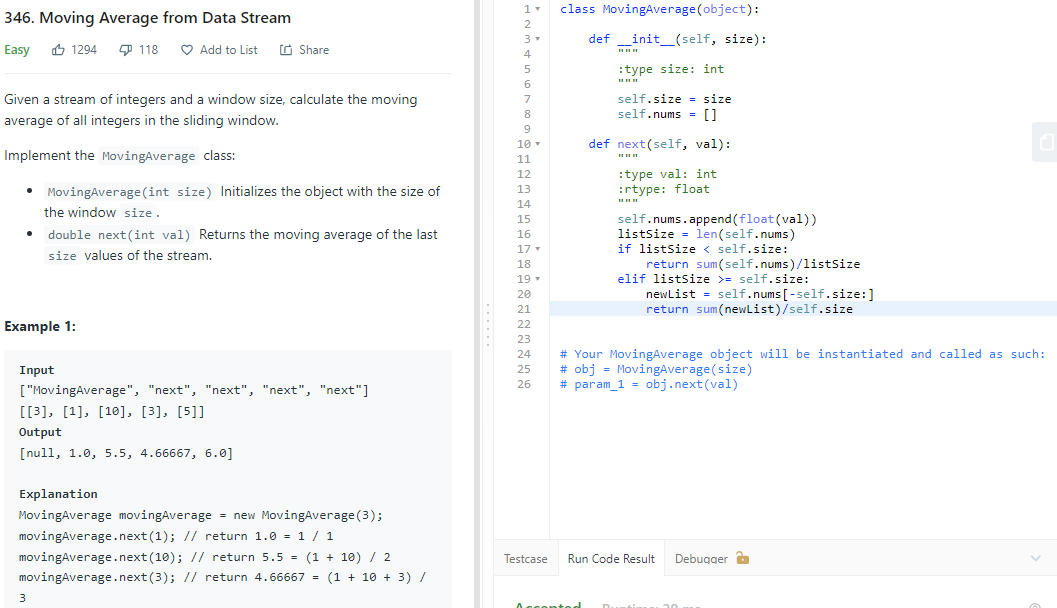


## 234. Palindrome Linked List



## 346. Moving Average from Data Stream

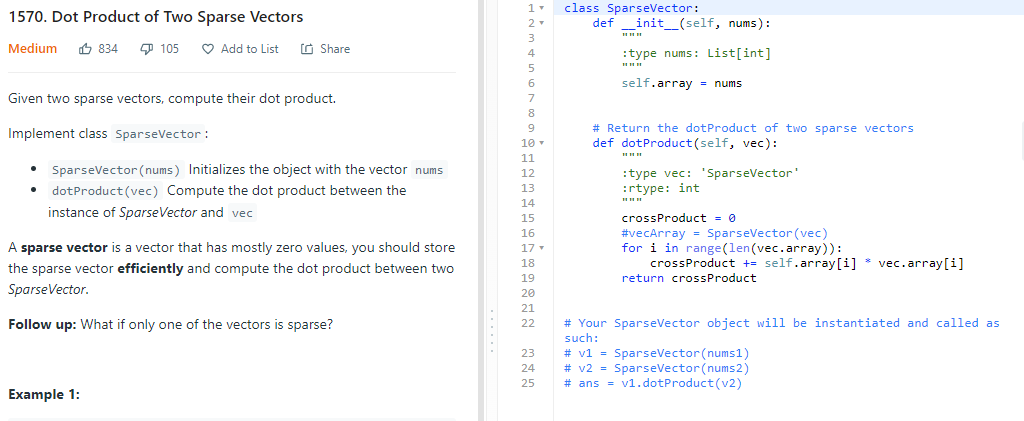


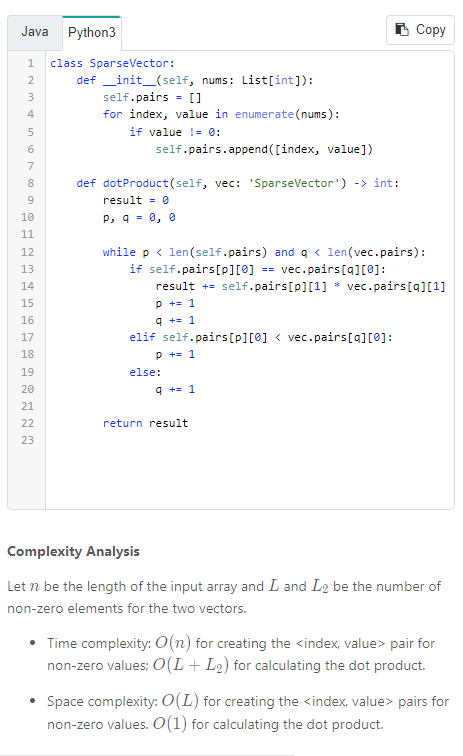
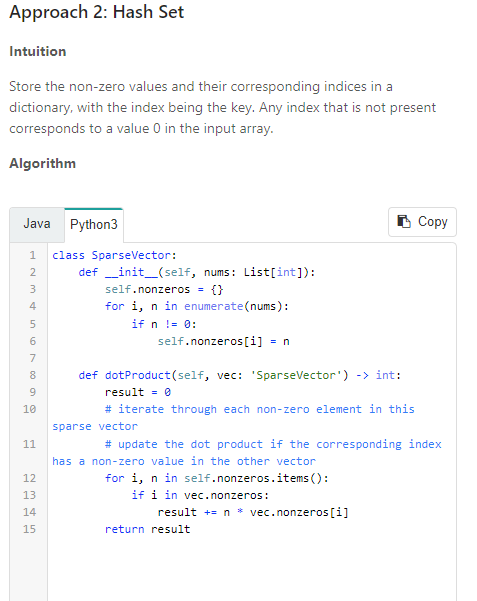


Time O(n)

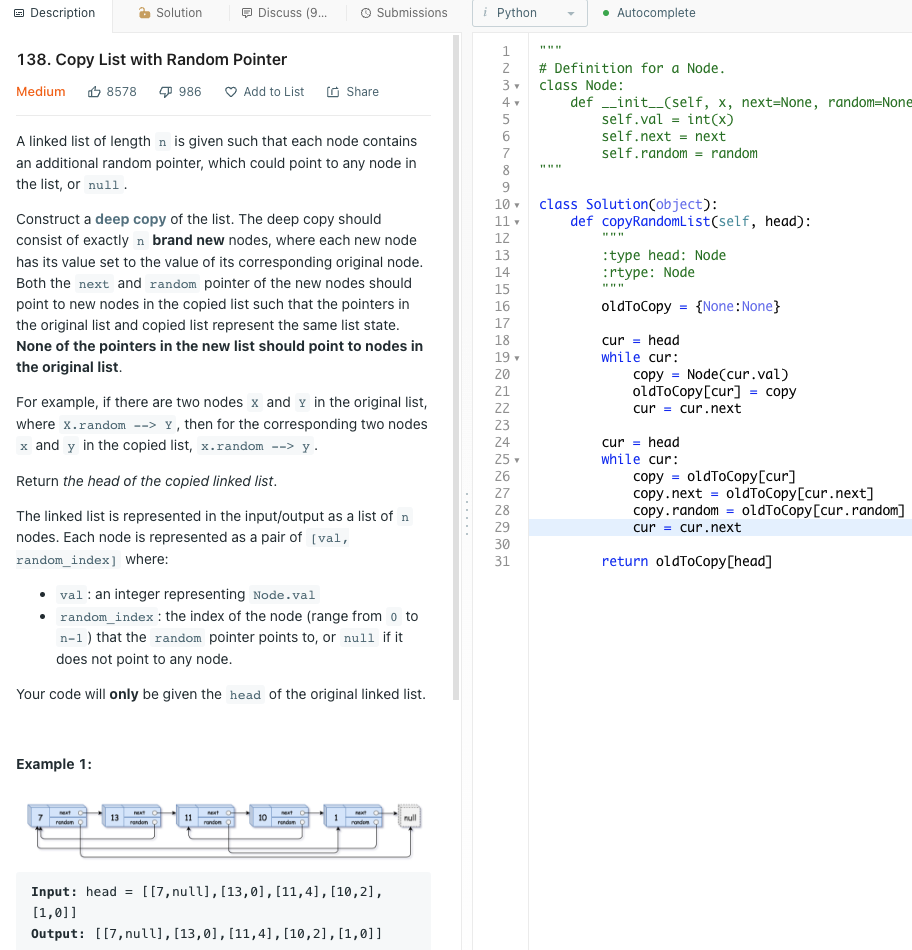
Space: O(n)

## 1570. Dot product of two sparse vectors

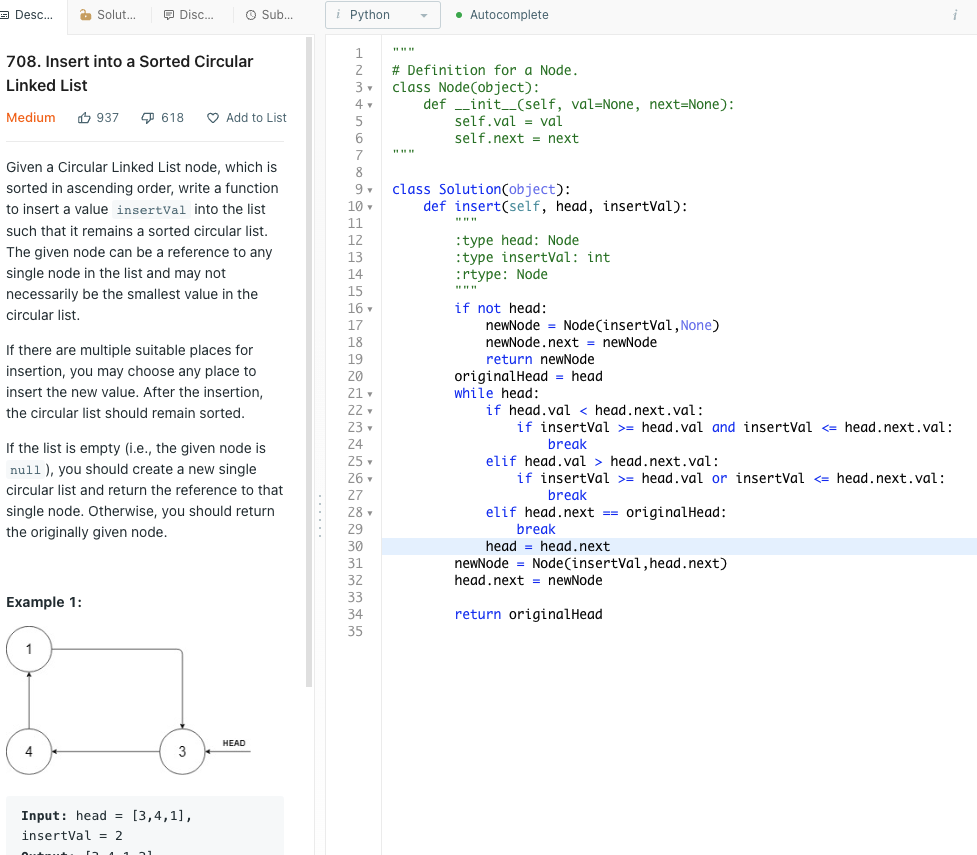




## 138. Copy List with Random Pointer



## 708. Insert into a Sorted Circular Linked List



## 528. Random Pick with Weight

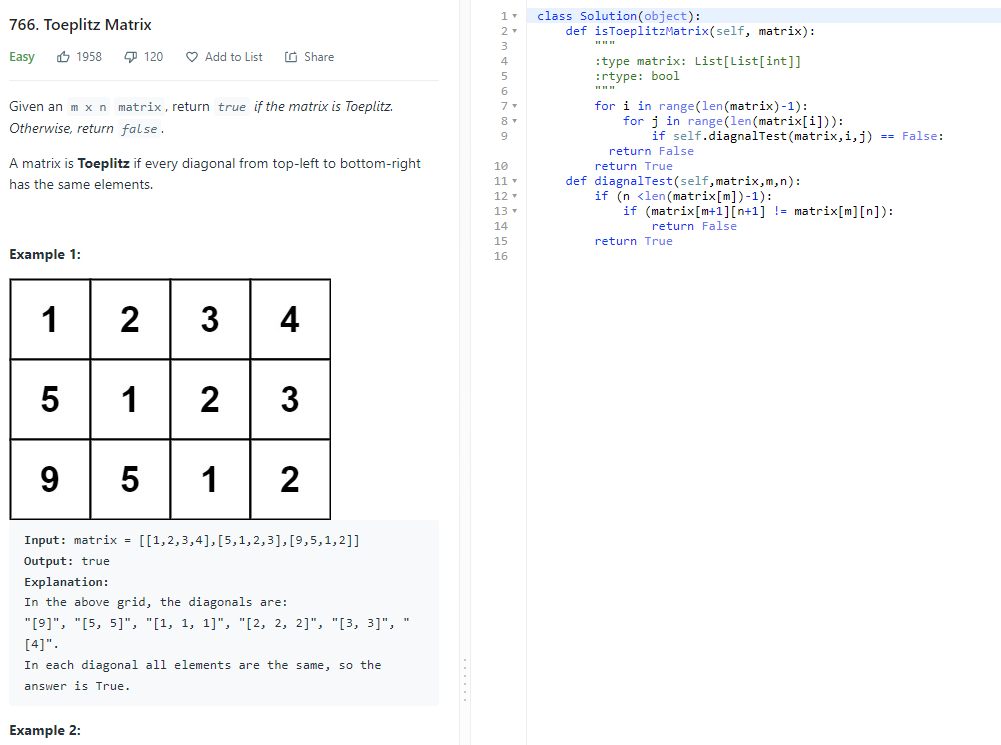


Time: O(n) for constructor O(logn) for binary search

Space: O(n) for the constructor

# Matrix

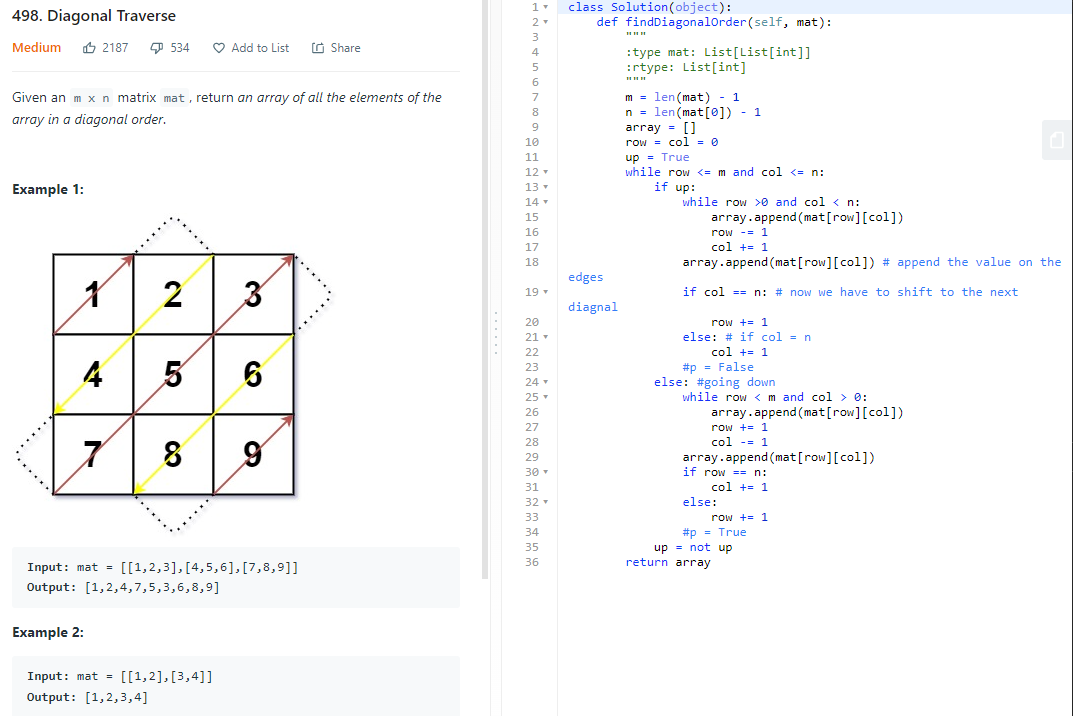
## 766. Teoplitz Matrix



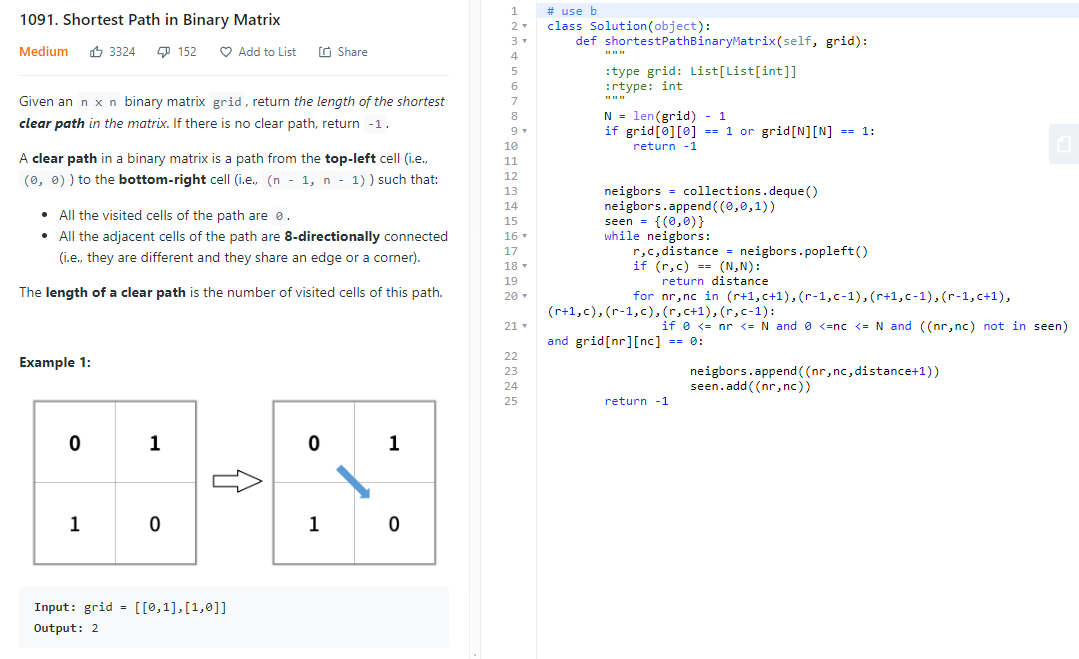
Time: O(mn), m and n is the number or rows and columns

Space, O(m+n)

## 498. Diagonal Traverse



## 1091. Shortest Path in Binary Matrix

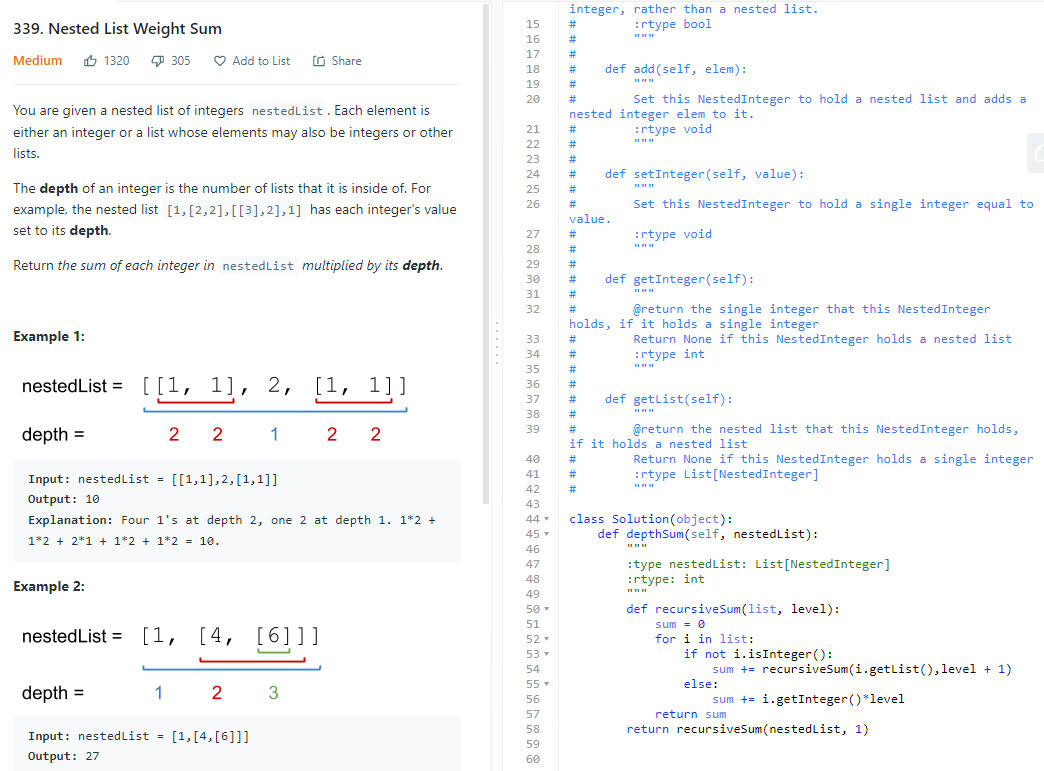


Time:O(n)

Space:O(1)

# Recursion:

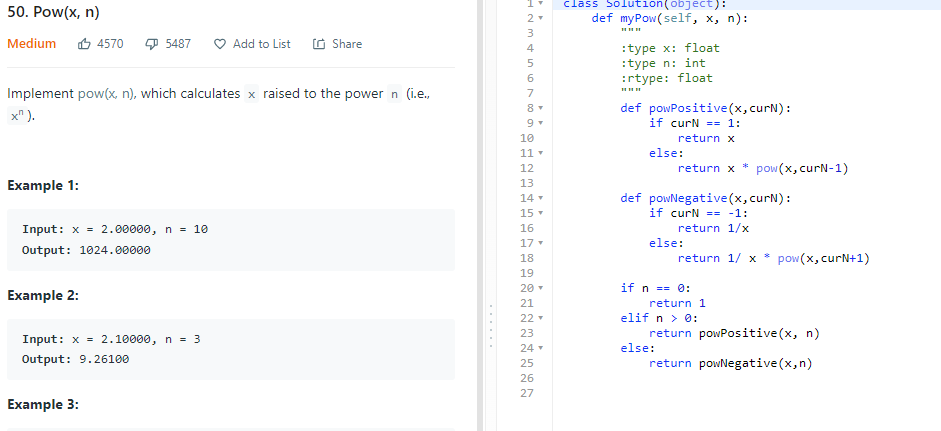
## 339. Nested List Weight Sum



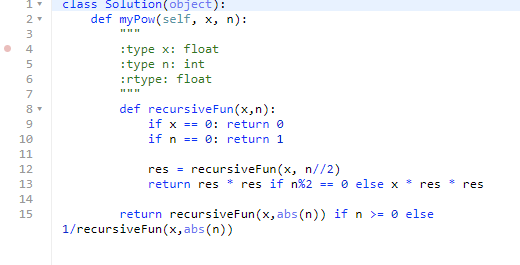
Time: O(2N), 1 N is the recursive call, the other N is the iteration inside the recursive call

Space: O(N), N being the number of recursive calls

## 50. Pow(x,n)



Time: O(n), space:O(1)



Time: O(logn), space: O(logn)