Contents

[3. Longest Substring Without Repeating Characters 1](#_Toc81600915)

[11. Container with most water 1](#_Toc81600916)

[# 15. 3Sum 2](#_Toc81600917)

[# 31. Next Permutation 2](#_Toc81600918)

[# 48. Rotate Image 2](#_Toc81600919)

[# 53. Maximum Subarray 3](#_Toc81600920)

[#55. Jump Game 3](#_Toc81600921)

[# 21. Merge Two Sorted List 5](#_Toc81600922)

[# 268. Missing Number 5](#_Toc81600923)

[# 707.Linked List (singly) 5](#_Toc81600924)

[# 141.Linked List Cycle 7](#_Toc81600925)

[#88. Merge Sorted Array 7](#_Toc81600926)

[# 704. Binary Search 7](#_Toc81600927)

### [3.](https://leetcode.com/problems/longest-substring-without-repeating-characters/) Longest Substring Without Repeating Characters

*# Input: s = "pwwkew" ; Output: 3*def lengthOfLongestSubstring(s: str) -> int:  
 start = maxLength = 0  
 usedChar = {}  
  
 for i in range(len(s)):  
 if s[i] in usedChar and start <= usedChar[s[i]]:  
 start = usedChar[s[i]] + 1  
 else:  
 maxLength = max(maxLength, i - start + 1)  
  
 usedChar[s[i]] = i  
  
 return maxLength  
*#-------------------------------------------------------*

### [11.](https://leetcode.com/problems/container-with-most-water/) Container with most water

from typing import List  
def maxArea(height : List[int]) -> int:  
 result = 0  
 l = 0  
 r = len(height) - 1  
 hmax = max(height)  
  
 while (r-l)\*hmax > result:  
 if(height[l]<height[r]):  
 result = max(result, (r - l) \*height[l])  
 l+=1  
 else:  
 result = max(result, (r - l) \* height[r])  
 r-=1  
  
 return result

*#-------------------------------------------------------*

### **#** [15.](https://leetcode.com/problems/3sum/) **3Sum**

*# finding all three numbers is a list which sum of them is 0*from collections import Counter  
def threeSum(nums):  
 counter = Counter(nums)  
 nums, triplets = list(counter.keys()), set()  
 if counter[0] >= 3:  
 triplets.add((0, 0, 0))  
 positives, negatives = [n for n in nums if n > 0], [n for n in nums if n < 0]  
 for a in negatives:  
 for b in positives:  
 c = -(a + b)  
 if c in counter and ((c != a and c != b) or counter[c] > 1):  
 triplets.add(tuple(sorted([a, b, c])))  
 return triplets  
  
print(threeSum([-1,0,1,2,-1,-4]))  
*#-------------------------------------------------------*

# [31.](https://leetcode.com/problems/next-permutation/) Next Permutation  
def nextPermutation( nums: List[int]) -> None:  
 flag = True  
 i = len(nums) - 1  
 while i > 0:  
 if nums[i] > nums[i - 1]:  
 nums[i:] = sorted(nums[i:])  
 for j in range(i, len(nums)):  
 if nums[i - 1] < nums[j]:  
 temp = nums[j]  
 nums[j] = nums[i - 1]  
 nums[i - 1] = temp  
 return  
 return  
 i -= 1  
 nums.sort()

nums = [1,3,2]  
*#expected [2, 1, 3]*  
nextPermutation(nums)  
print(nums)  
*#-----------------------------------------------------*

### # [48.](https://leetcode.com/problems/rotate-image/) Rotate Image

def rotate(matrix: List[List[int]]) -> None:  
 l = 0  
 r = len(matrix)-1  
  
 while l < r :  
 for i in range(r-l):  
 top , bottom = l, r  
 *# store top left* temp = matrix[top][l+i]  
 *# bottom left to top left* matrix[top][l + i] = matrix[bottom-i][l]  
 matrix[bottom - i][l] = matrix[bottom][r-i]  
 matrix[bottom][r - i] = matrix[top+i][r]  
 matrix[top+i][r] = temp  
  
 l+=1  
 r-=1  
   
matrix = [[1,2,3],[4,5,6],[7,8,9]]  
rotate(matrix)  
*#Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]  
#Output: [[7,4,1],[8,5,2],[9,6,3]]*print(matrix)

**Counter Clockwise**

def rotateMatrixCCW(matrix : List[List[int]])-> List[List[int]]:  
 l , r = 0 , len(matrix)-1  
 temp = 0  
 while l<r:  
 for i in range(r-l):  
 top, bottom = l, r  
 temp = matrix[top+i][l]  
 matrix[top+i][l] = matrix[top][r-i]  
 matrix[top][r-i] = matrix[bottom-i][r]  
 matrix[bottom - i][r] = matrix[bottom][l+i]  
 matrix[bottom][l+i] = temp  
 l+=1  
 r-=1  
  
matrix = [[1, 2, 3, 4],  
 [5, 6, 7, 8],  
 [9, 10,11,12],  
 [13,14,15,16]]  
rotateMatrixCCW(matrix)  
print(matrix)

*#-----------------------------------------------------*

### # [53](https://leetcode.com/problems/maximum-subarray/). Maximum Subarray

Maximum Contiguous Array Sum(subarray): Kadane’s Algorithmdef maxSumSubArray(nums: List[int]) -> int:  
 max\_sum = - pow(10,5)  
 max\_end\_here = 0  
 for i in range(len(nums)):  
 max\_end\_here += nums[i]  
 if max\_end\_here > max\_sum:  
 max\_sum = max\_end\_here  
 if max\_end\_here < 0:  
 max\_end\_here = 0  
 return max\_sum  
l = [-2,1,-3,4,-1,2,1,-5,4]  
*#expected = 6*print(maxSumSubArray(l))

*#-----------------------------------------------------*

### #[55.](https://leetcode.com/problems/jump-game/) Jump Game

You are given an integer array nums. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return true if you can reach the **last index**, or false otherwise.

*#* [*55.*](https://leetcode.com/problems/jump-game/)*Jump Games*def canJump(nums: List[int]) -> bool :  
 max\_jump = 0   
 for i in range(len(nums)):   
 if i >max\_jump:   
 return False   
 max\_jump = max(max\_jump, nums[i]+i)  
 if max\_jump >= len(nums)-1:  
 return True   
nums = [2,3,4,2,1,0,1] *# -> True  
# nums = [2,3,3,2,1,0,1] # -> False*print(canJump(nums))

*#* [*707*](https://leetcode.com/problems/design-linked-list/)*.Linked List (singly)*class Node:  
 def \_\_init\_\_(self, val):  
 self.nextNode = None  
 self.value = val  
  
  
class MyLinkedList:  
  
 def \_\_init\_\_(self):  
 self.head = None  
 self.length = 0  
  
 def get(self, index: int) -> int:  
  
 if index < 0 or index >= self.length:  
 return -1  
  
 if self.head is None:  
 return -1  
  
 curr = self.head  
 for i in range(index):  
 curr = curr.nextNode  
 return curr.value  
  
 def addAtHead(self, val: int) -> None:  
  
 newNode = Node(val)  
 newNode.nextNode = self.head  
 self.head = newNode  
  
 self.length += 1  
  
 def addAtTail(self, val: int) -> None:  
 if self.head == None:  
 self.head = Node(val)  
 else:  
 curr = self.head  
 while curr.nextNode is not None:  
 curr = curr.nextNode  
  
 curr.nextNode = Node(val)  
  
 self.length += 1  
  
 def addAtIndex(self, index: int, val: int) -> None:  
 if index < 0 or index > self.length:  
 return  
  
 if index == 0:  
 self.addAtHead(val)  
 else:  
 curr = self.head  
  
 for i in range(index - 1):  
 curr = curr.nextNode  
  
 newNode = Node(val)  
 newNode.nextNode = curr.nextNode  
 curr.nextNode = newNode  
  
 self.length += 1  
  
 def deleteAtIndex(self, index: int) -> None: *# done* if index < 0 or index >= self.length:  
 return  
  
 curr = self.head  
 if index == 0:  
 self.head = curr.nextNode  
 else:  
 for i in range(index - 1):  
 curr = curr.nextNode  
  
 curr.nextNode = curr.nextNode.nextNode  
  
 self.length -= 1  
*# Input: ["MyLinkedList","addAtHead","addAtTail","addAtIndex","get","deleteAtIndex","get"]  
# [[],[1],[3],[1,2],[1],[1],[1]]  
# OutPut: [null,null,null,null,2,null,3]*

*#-----------------------------------------------------*

### **#** [21.](https://leetcode.com/problems/merge-two-sorted-lists/) **Merge Two Sorted List**

**'''Merge two sorted linked lists and return it as a sorted list.  
 The list should be made by splicing together the nodes of the first two lists.'''**def mergeTwoLists(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:  
 pointer = ListNode()  
 head = pointer  
 while l1 and l2:  
 if l1.val < l2.val:  
 pointer.next = l1  
 pointer = pointer.next  
 l1 = l1.next  
 else:  
 pointer.next = l2  
 pointer = pointer.next  
 l2 = l2.next  
 if l1 or l2:  
 pointer.next = l1 if l1 else l2  
 return head.next  
*#input = [1,2,4] and [1,3,4]  
#output = [1,1,2,3,4,4]*

*#-----------------------------------------------------*

### # 268. Missing Number

*#finding the missing number of numbers from 0-n  
#n\*n+1/2 = sum+(missing Number)*def missingNumber(nums: List[int])-> int:  
 return int((len(nums) \* (len(nums) + 1) / 2)-sum(nums))  
  
l= [1,2,4,0]  
print(missingNumber(l))  
*# 3*

*#-----------------------------------------------------*

### # 707.Linked List (singly)

class Node:  
 def \_\_init\_\_(self, val):  
 self.nextNode = None  
 self.value = val  
  
  
class MyLinkedList:  
  
 def \_\_init\_\_(self):  
 self.head = None  
 self.length = 0  
  
 def get(self, index: int) -> int:  
  
 if index < 0 or index >= self.length:  
 return -1  
  
 if self.head is None:  
 return -1  
  
 curr = self.head  
 for i in range(index):  
 curr = curr.nextNode  
 return curr.value  
  
 def addAtHead(self, val: int) -> None:  
  
 newNode = Node(val)  
 newNode.nextNode = self.head  
 self.head = newNode  
  
 self.length += 1  
  
 def addAtTail(self, val: int) -> None:  
 if self.head == None:  
 self.head = Node(val)  
 else:  
 curr = self.head  
 while curr.nextNode is not None:  
 curr = curr.nextNode  
  
 curr.nextNode = Node(val)  
  
 self.length += 1  
  
 def addAtIndex(self, index: int, val: int) -> None:  
 if index < 0 or index > self.length:  
 return  
  
 if index == 0:  
 self.addAtHead(val)  
 else:  
 curr = self.head  
  
 for i in range(index - 1):  
 curr = curr.nextNode  
  
 newNode = Node(val)  
 newNode.nextNode = curr.nextNode  
 curr.nextNode = newNode  
  
 self.length += 1  
  
 def deleteAtIndex(self, index: int) -> None: *# done* if index < 0 or index >= self.length:  
 return  
  
 curr = self.head  
 if index == 0:  
 self.head = curr.nextNode  
 else:  
 for i in range(index - 1):  
 curr = curr.nextNode  
  
 curr.nextNode = curr.nextNode.nextNode  
  
 self.length -= 1  
*# Input: ["MyLinkedList","addAtHead","addAtTail","addAtIndex","get","deleteAtIndex","get"]  
# [[],[1],[3],[1,2],[1],[1],[1]]  
# OutPut: [null,null,null,null,2,null,3]*

*#--------------------------------------------------------------*

# [141.](https://leetcode.com/problems/linked-list-cycle/)Linked List Cycle *#-105 <= Node.val <= 105  
# input = [3,2,0,-4]  
# pos= 2 True , pos = -1 False*def linkedListCycle(head: Optional[ListNode]):  
 while head != None:  
 if head.val == 10e6 :  
 return True  
 else :  
 head.val = 10e6 *# mark the visited node* head = head.next  
 return False  
  
*#--------------------------------------------------------------*

### #[88.](https://leetcode.com/problems/merge-sorted-array/) Merge Sorted Array

**'''  
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3  
Output: [1,2,2,3,5,6]  
'''***# 88. Merge Sorted Array*def merge(nums1 : List[int], n: int ,nums2: List[int], m )->None :  
 if n == 0:  
 nums1[:] = nums2[:]  
 return  
 if m == 0:  
 return  
 if nums1[n-1]<= nums2[0]:  
 nums1[n:] = nums2[:]  
 return  
 else:  
 pointer = m+n-1  
 i , j = n-1 , m-1  
 while i >=0 and j >=0 :  
 if nums1[i] > nums2[j]:  
 nums1[pointer] = nums1[i]  
 i-=1  
 else:  
 nums1[pointer] = nums2[j]  
 j-=1  
  
 pointer-=1  
 if j >= 0:  
 nums1[0:pointer+1] = nums2[0:j+1]  
  
nums1: List[int] = [2,0]  
merge(nums1,1,[1],1)  
print(nums1)  
*#--------------------------------------------------------*

# [704.](https://leetcode.com/problems/binary-search/) Binary Searchdef search(nums: List[int], target: int) -> int:  
 pointerL = 0  
 pointerR = len(nums)-1  
 while pointerR - pointerL > 1:  
 n = int((pointerR - pointerL)/2)+pointerL  
  
 if nums[n] == target:  
 return n  
  
 if nums[n] > target:  
 pointerR = n-1  
  
 if nums[n] < target:  
 pointerL = n+1  
  
 if nums[pointerL] == target:  
 return pointerL  
 if nums[pointerR] == target:  
 return pointerR  
 return -1  
  
nums = [-1,0,3,5,9,12]  
target = 9  
print(search(nums, target))  
*#------------------------------------------------------*

# 104.Maximum depth of Binary Tree([Link](https://leetcode.com/problems/maximum-depth-of-binary-tree/))class TreeNode:  
 def \_\_init\_\_(self, val=0, left=None, right=None):  
 self.val = val  
 self.left = left  
 self.right = right  
  
class SolutionMaxDepth:  
 def maxDepth(self, root: Optional[TreeNode]) -> int:  
 if root in None:  
 return 0  
  
 max\_depth = 1  
 def travers(root : TreeNode, depth ):  
 if not root.left and not root.right:  
 nonlocal max\_depth  
 max\_depth = max(max\_depth, depth)  
 return max\_depth  
 if root.left:  
 travers(root.left , depth+1 )  
 if root.right:  
 travers(root.right, depth+1 )  
 travers(root, max\_depth)  
 return max\_depth  
  
newBT1 = TreeNode(3)  
newBT2 = TreeNode(9)  
newBT3 = TreeNode(20) *# 3 -> 1*newBT1.left = newBT2 *# / \*newBT1.right = newBT3 *# 9 20 -> 2*newBT4 = TreeNode(15) *# / \*newBT5 = TreeNode(7) *# 15 7 -> 3*newBT3.left = newBT4 *# depth = 3*newBT3.right = newBT5  
depth = SolutionMaxDepth()  
print(depth.maxDepth(newBT1))

*#------------------------------------------------------*