```python
# 80. Remove Duplicates from Sorted Array II
# Input: nums = [1,1,1,2,2,3]
# Output: 5, nums = [1,1,2,2,3,_]
# Explanation: Your function should return k = 5, with the first five
elements of nums being 1, 1, 2, 2 and 3 respectively.
# It does not matter what you leave beyond the returned k (hence they
are underscores).
# Example 2:

# Input: nums = [0,0,1,1,1,1,2,3,3]
# Output: 7, nums = [0,0,1,1,2,3,3,_,_]
# Explanation: Your function should return k = 7, with the first seven
elements of nums being 0, 0, 1, 1, 2, 3 and 3 respectively.
# It does not matter what you leave beyond the returned k (hence they
are underscores).

class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:

        left_pointer = 0
        right_pointer = 0

        while right_pointer < len(nums):

            count = 1
            while right_pointer + 1 < len(nums) and
nums[right_pointer] ==  nums[right_pointer + 1]:

                count += 1
                right_pointer +=1

            for i in range(min(2, count)):

                nums[left_pointer] = nums[right_pointer]
                left_pointer +=1


            right_pointer +=1

        return left_pointer


# 791. Custom Sort String
# Solved
# Medium
# Topics
# Companies
# You are given two strings order and s.
# All the characters of order are unique and were sorted in some
custom order previously.
```

```python
# Permute the characters of s so that they match the order that order
was sorted. More specifically,
# if a character x occurs before a character y in order, then x should
occur before y in the permuted string.

# Return any permutation of s that satisfies this property.



# Example 1:

# Input: order = "cba", s = "abcd"

# Output: "cbad"

# Explanation: "a", "b", "c" appear in order, so the order of "a",
"b", "c" should be "c", "b", and "a".

# Since "d" does not appear in order,
# it can be at any position in the returned string. "dcba", "cdba",
"cbda" are also valid outputs.

# Example 2:

# Input: order = "bcafg", s = "abcd"

# Output: "bcad"

# Explanation: The characters "b", "c", and "a" from order dictate the
order for the characters in s.
# The character "d" in s does not appear in order, so its position is
flexible.

# Following the order of appearance in order, "b", "c",
# and "a" from s should be arranged as "b", "c", "a". "d" can be
placed at any position since
# it's not in order. The output "bcad" correctly follows this rule.
#  Other arrangements like "dbca" or "bcda" would also be valid, as
long as "b", "c", "a" maintain their order.

class Solution:
    def customSortString(self, order: str, s: str) -> str:

        char_cnt = {}
        for i in range(len(s)):
            char_cnt[s[i]] = char_cnt.get(s[i], 0) + 1

        res = ""
        for i in range(len(order)):
            if order[i] in char_cnt and char_cnt[order[i]] > 0:
                res += order[i] * char_cnt[order[i]]
```

```
                char_cnt[order[i]] -= char_cnt[order[i]]
        for val, count in char_cnt.items():
            if count > 0:
                res +=val * count
        return res


# Code
# Testcase
# Testcase
# Test Result
# 735. Asteroid Collision
# Solved
# Medium
# Topics
# Companies
# Hint
# We are given an array asteroids of integers representing asteroids
in a row.

# For each asteroid, the absolute value represents its size, and the
sign represents its direction
#  (positive meaning right, negative meaning left). Each asteroid
moves at the same speed.

# Find out the state of the asteroids after all collisions. If two
asteroids meet,
#  the smaller one will explode. If both are the same size, both will
explode. Two asteroids moving in the same direction will never meet.



# Example 1:

# Input: asteroids = [5,10,-5]
# Output: [5,10]
# Explanation: The 10 and -5 collide resulting in 10. The 5 and 10
never collide.
# Example 2:

# Input: asteroids = [8,-8]
# Output: []
# Explanation: The 8 and -8 collide exploding each other.
# Example 3:

# Input: asteroids = [10,2,-5]
# Output: [10]
# Explanation: The 2 and -5 collide resulting in -5. The 10 and -5
collide resulting in 10.
```

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

        res = []

        for a in asteroids:
            while res and res[-1] > 0 and a < 0:
                positive = res.pop()
                if positive == abs(a):
                    break
                elif positive > abs(a):
                    res.append(positive)
                    break
            else:
                res.append(a)
        return res

"""
Bruteforce - T:O(N) S:O(1)
"""
class SparseVector:
    def __init__(self, nums):
        self.array = nums

    def dotProduct(self, vec):
        result = 0
        for num1, num2 in zip(self.array, vec.array):
            result += num1 * num2
        return result
"""
Hash Set
Time complexity: O(n) for creating the Hash Map; O(L) for calculating
the dot product.
Space complexity: O(L) for creating the Hash Map, as we only store
elements that are non-zero. O(1) for calculating the dot product.
"""
class SparseVector:
    def __init__(self, nums: List[int]):
        self.nonzeros = {}
        for i, n in enumerate(nums):
            if n != 0:
                self.nonzeros[i] = n

    def dotProduct(self, vec: 'SparseVector') -> int:
        result = 0
        # iterate through each non-zero element in this sparse vector
        # update the dot product if the respective index has a non-
zero value in the other vector
        for i, n in self.nonzeros.items():
            if i in vec.nonzeros:
```

```
                result += n * vec.nonzeros[i]
        return result
"""
 Index-value pairing
 T:O(N) S:O(L) for creating <index,value> pairs and O(1) for
calculations
"""
# T:O(N) for creating <index,value> for nonzero values. O(L1+L2) fo
calculatng product
# S: O(L) for <index,value> pairs  O(1) for dot product
class SparseVector:
    def __init__(self, nums: List[int]):
        self.pairs = []

        for i,num in enumerate(nums):
            if num != 0:
                self.pairs.append((i,num))

    # worst case we may be given 2 NON-SPARSE vectors
    # so runtime complexity is O(M+N)   as we may need to go through
both vectors fully
    # as here in problem, constraints is like 2 vectors of SAME size
    # T:O(M+N) S: O(M+N)
    def dotProduct(self, vec: 'SparseVector') -> int:
        result = 0
        # 2 pointers
        p , q = 0, 0

        while p < len(self.pairs) and q < len(vec.pairs):
            if self.pairs[p][0] == vec.pairs[q][0]:
                result += self.pairs[p][1] * vec.pairs[q][1]
                p += 1
                q += 1
            elif self.pairs[p][0] < vec.pairs[q][0]:
                p += 1
            else:
                q += 1

        return result

# 658. Find K Closest Elements
# Solved
# Medium
# Topics
# Companies
# Given a sorted integer array arr, two integers k and x,
# return the k closest integers to x in the array.
# The result should also be sorted in ascending order.

# An integer a is closer to x than an integer b if:
```

```python
# |a - x| < |b - x|, or
# |a - x| == |b - x| and a < b


# Example 1:

# Input: arr = [1,2,3,4,5], k = 4, x = 3

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

class Solution:
    def findClosestElements(self, arr: List[int], k: int, x: int) ->
List[int]:

        heap = []

        for i in range(len(arr) ):
            heapq.heappush(heap, ( abs(arr[i] - x), arr[i]) )

        res  = []
        for _ in range(k):
            diff, num = heapq.heappop(heap)
            res.append(num)

        res.sort()
        return res


class Solution:
    def findClosestElements(self, arr: List[int], k: int, x: int) ->
List[int]:

        l,r = 0, len(arr)-1
        while (r-l+1) > k:
            if abs(arr[l]- x) > abs(arr[r]- x):
                l+=1
            else:
                r-=1
        res = []
        for i in range(l,r+1):
            res.append(arr[i])
        return res


# 1004. Max Consecutive Ones III
# Solved
# Medium
# Topics
# Companies
# Hint
```

```python
# Given a binary array nums and an integer k, return the maximum
number of consecutive 1's in the array if you can flip at most k 0's.



# Example 1:

# Input: nums = [1,1,1,0,0,0,1,1,1,1,0], k = 2
# Output: 6
# Explanation: [1,1,1,0,0,1,1,1,1,1,1]
# Bolded numbers were flipped from 0 to 1. The longest subarray is
underlined.
# Example 2:

# Input: nums = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1], k = 3
# Output: 10
# Explanation: [0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]
# Bolded numbers were flipped from 0 to 1. The longest subarray is
underlined.



class Solution:
    def longestOnes(self, nums: List[int], k: int) -> int:

        zeros = 0
        l,r = 0,0
        max_con = 0
        for r in range(len(nums)):
            if nums[r] == 0:
                zeros+=1
            while zeros > k:
                if nums[l] == 0:
                    zeros -=1
                l+=1
            max_con = max(max_con, r-l+1)
        return max_con

class Solution:
    def longestOnes(self, nums: List[int], k: int) -> int:
        max_con = 0
        zeros = {}
        l,r = 0,0

        for r in range(len(nums)):
            if nums[r] == 0:
                zeros[nums[r]] = zeros.get(nums[r], 0) + 1
            while 0 in zeros and zeros[0] > k:
                if nums[l] == 0:
                    zeros[nums[l]] -=1
```

```
                l+=1
            max_con = max(max_con, r-l+1)

        return max_con

# '''
# 767. Reorganize String
# Medium

# Topics
# Companies

# Hint
# Given a string s, rearrange the
# characters of s so that any two
#  adjacent characters are not the same.

# Return any possible rearrangement
#  of s or return "" if not possible.



# Example 1:

# Input: s = "aab"
# Output: "aba"
# Example 2:

# Input: s = "aaab"
# Output: ""
# '''
class Solution:
    def reorganizeString(self, s: str) -> str:
        char_freq = {}
        for c in s:
            char_freq[c] = char_freq.get(c, 0) + 1

        max_heap = [(-freq, char) for char, freq in char_freq.items()]
        heapq.heapify(max_heap)

        res = []
        prev_freq, prev_char = 0, ""

        while max_heap:
            freq, char = heapq.heappop(max_heap)
            res.append(char)

            if prev_freq < 0:
                heapq.heappush(max_heap, (prev_freq, prev_char))

            freq += 1
```

```
            prev_freq, prev_char = freq, char

        if len(res) != len(s):
            return ""

        return "".join(res)

# '''
# 31. Next Permutation
# Medium

# Topics
# Companies
# A permutation of an array of integers is
#  an arrangement of its members into a sequence
#  or linear order.

# For example, for arr = [1,2,3],
# the following are all the permutations of arr:

# [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].
# The next permutation of an array of integers is the
# next lexicographically greater permutation of its integer.
# More formally, if all the permutations of the array
# are sorted in one container according to their lexicographical
order,
# then the next permutation of that
# array is the permutation that follows it in the sorted container.
# If such arrangement is not possible,
# the array must be rearranged as the lowest possible order
#  (i.e., sorted in ascending order).

# For example, the next permutation of arr = [1,2,3] is [1,3,2].
# Similarly, the next permutation of arr = [2,3,1] is [3,1,2].
# While the next permutation of arr = [3,2,1] is [1,2,3]
#  because [3,2,1] does not have a lexicographical larger
rearrangement.
# Given an array of integers nums, find the next permutation of nums.

# The replacement must be in place and use only constant extra memory.



# Example 1:

# Input: nums = [1,2,3]
# Output: [1,3,2]
# Example 2:

# Input: nums = [3,2,1]
# Output: [1,2,3]
```

```python
# Example 3:

# Input: nums = [1,1,5]
# Output: [1,5,1]
# '''




class Solution:
    def swap(self, nums, ind1, ind2):
        temp = nums[ind1]
        nums[ind1] = nums[ind2]
        nums[ind2] = temp

    def reverse(self, nums, beg, end):
        while beg < end:
            self.swap(nums, beg, end)
            beg += 1
            end -= 1

    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        if len(nums) == 1:
            return
        if len(nums) == 2:
            return self.swap(nums, 0, 1)
        dec = len(nums) - 2
        while dec >= 0 and nums[dec] >= nums[dec + 1]:
            dec -= 1
        self.reverse(nums, dec + 1, len(nums) - 1)
        if dec == -1:
            return
        next_num = dec + 1
        while next_num < len(nums) and nums[next_num] <= nums[dec]:
            next_num += 1
        self.swap(nums, next_num, dec)


# Code
# Testcase
# Testcase
# Test Result
# 560. Subarray Sum Equals K
# Solved
# Medium
```

```python
# Topics
# Companies
# Hint
# Given an array of integers nums and an integer k, return the total
number of subarrays whose sum equals to k.

# A subarray is a contiguous non-empty sequence of elements within an
array.



# Example 1:

# Input: nums = [1,1,1], k = 2
# Output: 2
# Example 2:

# Input: nums = [1,2,3], k = 3
# Output: 2

class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        prefix_sum_count = {0: 1}
        current_sum = 0
        res = 0
        for num in nums:
            current_sum += num
            if (current_sum - k) in prefix_sum_count:
                res += prefix_sum_count[current_sum - k]

            prefix_sum_count[current_sum] =
prefix_sum_count.get(current_sum, 0) + 1

        return res

# '''
# Data analysts at Amazon are building a utility to identify redundant
words in advertisements.
# They define a string as redundant if the length of the string, /word
= a *V+ b * C, where a and b are given
# integers and Vand Care the numbers of vowels and consonants in the
string word.
# Given a string word, and two integers, a, and b, find the number of
redundant substrings of word
# Note: A substring is a contiguous group of 0 or more characters in a
string. For example- "bcb" is a substring of
# "abba", while "bba" is not.
# Example
# word = "abbacc", a = -1, b = 2.
# The redundant substrings in "abbacc" are shown
```

```
# Substring
# Vowels
# Consonants
# Length
# a *V+b * c
# "abb"
# 1
# 2
# 3
# -1*1 + 2*2 = 3
# "bba"
# 1
# 2
# 3
# -1*1 + 2*2 = 3
# "bac"
# 1
# 2
# 3
# -1*1 + 2*2 = 3
# "acc"
# 1
# 2
# 3
# -1*1 + 2*2 = 3
# "abbacc"
# 2
# 4
# 6
# -1*2 + 2*4 = 6
# There are 5 redundant substrings.
# Function Description
# Complete the function
# getRedundantSubstrings in
# the editor below.
# getRedundantSubstrings has
# the following parameters:
# string word: the string
# int a: the multiplier
# for the number of vowels int b:
#  the m


# '''



def getRedundantSubstrings(word: str, a: int, b: int) -> int:
```

```
    vowels = {'a', 'e', 'i', 'o', 'u'}
    n = len(word)
    ans = 0

    # Hashmap to store frequency of (cost - length) values
    count = {0: 1}

    # Cumulative totals of vowels and consonants
    total_vowels, total_consonants = 0, 0

    for i, char in enumerate(word):
        if char in vowels:
            total_vowels += 1
        else:
            total_consonants += 1

        # Calculate the current cost
        current_cost = a * total_vowels + b * total_consonants
        current_length = i + 1

        # Calculate the difference between cost and length
        diff = current_cost - current_length

        # Add the number of times we've seen this diff before
        ans += count.get(diff, 0)

        # Update the hashmap with the current diff
        count[diff] = count.get(diff, 0) + 1

    return ans

# 189. Rotate Array
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an integer array nums, rotate the array to the right by k
steps, where k is non-negative.



# Example 1:

# Input: nums = [1,2,3,4,5,6,7], k = 3
# Output: [5,6,7,1,2,3,4]
# Explanation:
# rotate 1 steps to the right: [7,1,2,3,4,5,6]
# rotate 2 steps to the right: [6,7,1,2,3,4,5]
# rotate 3 steps to the right: [5,6,7,1,2,3,4]
# Example 2:
```

```python
# Input: nums = [-1,-100,3,99], k = 2
# Output: [3,99,-1,-100]
# Explanation:
# rotate 1 steps to the right: [99,-1,-100,3]
# rotate 2 steps to the right: [3,99,-1,-100]

class Solution:
    def reverseArray(self, nums: List[int], start: int, end : int) ->
None:

        while start < end:

            hold = nums[start]
            nums[start] = nums[end]
            nums[end] = hold

            start +=1
            end -=1



    def rotate(self, nums: List[int], k: int) -> None:
        start_index = 0
        last_index = len(nums) -1

        #incase k is greater than length of array
        k = k % len(nums)

        #reverse array
        self.reverseArray(nums, start_index, last_index)

        #reverse first kth portion of array
        self.reverseArray(nums, start_index, k-1)

        #reverse kth + 1 to end portion of array
        self.reverseArray(nums, k, last_index)


# 55. Jump Game
# Solved
# Medium
# Topics
# Companies
# 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.
```

```python
# Example 1:

# Input: nums = [2,3,1,1,4]
# Output: true
# Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last
index.
# Example 2:

# Input: nums = [3,2,1,0,4]
# Output: false
# Explanation: You will always arrive at index 3 no matter what. Its
maximum jump length is 0, which makes it impossible to reach the

class Solution:

     def canJump(self, nums: List[int]) -> bool:
        goal = len(nums) - 1

        for i in range(goal, -1, -1):

            if i + nums[i] >= goal:
                goal = i

        return goal ==0

    def canJump(self, nums: List[int]) -> bool:

        goal = len(nums) -1

        i = len(nums)
        while i > 0:

            if i-1 + nums[i-1] >= goal:
                goal = i-1

            i-=1


        return goal == 0

# 122. Best Time to Buy and Sell Stock II
# Solved
# Medium
# Topics
# Companies
# You are given an integer array prices where prices[i] is the price
of a given stock on the ith day.

# On each day, you may decide to buy and/or sell the stock. You can
only hold at most one share of the stock at any time. However, you can
```

```
buy it then immediately sell it on the same day.

# Find and return the maximum profit you can achieve.



# Example 1:

# Input: prices = [7,1,5,3,6,4]
# Output: 7
# Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5),
profit = 5-1 = 4.
# Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit
= 6-3 = 3.
# Total profit is 4 + 3 = 7.
# Example 2:

# Input: prices = [1,2,3,4,5]
# Output: 4
# Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5),
profit = 5-1 = 4.
# Total profit is 4.
# Example 3:

# Input: prices = [7,6,4,3,1]
# Output: 0
# Explanation: There is no way to make a positive profit, so we never
buy the stock to achieve the maximum profit of 0.

class Solution:
    def maxProfit(self, prices: List[int]) -> int:

        profit = 0

        for i in range(1, len(prices)):

            if prices[i] > prices[i-1]:
                profit += prices[i] - prices[i-1]

        return profit

# 274. H-Index
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an array of integers citations where citations[i] is the
number of citations a researcher received for their ith paper, return
the researcher's h-index.
```

```python
# According to the definition of h-index on Wikipedia: The h-index is defined as the maximum value
# of h such that the given researcher has published at least h papers that have each been cited at least h times.



# Example 1:

# Input: citations = [3,0,6,1,5]
# Output: 3
# Explanation: [3,0,6,1,5] means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively.
# Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, their h-index is 3.
# Example 2:

# Input: citations = [1,3,1]
# Output: 1
class Solution:

    def hIndex(self, citations: List[int]) -> int:
        h_index = 0

        citations.sort(reverse=True)

        for index, value in enumerate(citations):

            if value >= index + 1:
                h_index +=1

        return h_index

     def hIndex(self, citations: List[int]) -> int:

        n = len(citations)
        citations.sort()

        for i,v in enumerate(citations):
            #translation
            #i have published n - i papers
            #if it has been cited v or more than v times
            #found our answer
            if n - i <= v:
                return n - i

        #otherwise nothing found
        return 0
```

```python
    def hIndex(self, citations: List[int]) -> int:

        #brute fore

        h_index = 0
        for i in range(len(citations)):

            i_th_paper = i +1
            count_ith = 0

            for i in range(len(citations)):

                if citations[i] >= i_th_paper:
                    count_ith +=1

            if count_ith >= i_th_paper:
                h_index = i_th_paper

        return h_index



# 45. Jump Game II
# Solved
# Medium
# Topics
# Companies
# You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

# Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

# 0 <= j <= nums[i] and
# i + j < n
# Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].



# Example 1:

# Input: nums = [2,3,1,1,4]
# Output: 2
# Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.
# Example 2:

# Input: nums = [2,3,0,1,4]
```

```python
# Output: 2

class Solution:
    def jump(self, nums: List[int]) -> int:

        left = 0
        right = 0
        goal = len(nums) -1
        minimum_jump = 0
        while right < goal:

            max_jump = 0
            for i in range(left, right + 1):

                max_jump = max(max_jump, i + nums[i])
            minimum_jump +=1
            left = right + 1
            right = max_jump

        return minimum_jump


# 380. Insert Delete GetRandom O(1)
# Solved
# Medium
# Topics
# Companies
# Implement the RandomizedSet class:

# RandomizedSet() Initializes the RandomizedSet object.
# bool insert(int val) Inserts an item val into the set if not
present. Returns true if the item was not present, false otherwise.
# bool remove(int val) Removes an item val from the set if present.
Returns true if the item was present, false otherwise.
# int getRandom() Returns a random element from the current set of
elements
#  (it's guaranteed that at least one element exists when this method
is called). Each element must have the same probability of being
returned.
# You must implement the functions of the class such that each
function works in average O(1) time complexity.



# Example 1:

# Input
# ["RandomizedSet", "insert", "remove", "insert", "getRandom",
"remove", "insert", "getRandom"]
# [[], [1], [2], [2], [], [1], [2], []]
# Output
```

```
# [null, true, false, true, 2, true, false, 2]

# Explanation
# RandomizedSet randomizedSet = new RandomizedSet();
# randomizedSet.insert(1); // Inserts 1 to the set. Returns true as 1
was inserted successfully.
# randomizedSet.remove(2); // Returns false as 2 does not exist in the
set.
# randomizedSet.insert(2); // Inserts 2 to the set, returns true. Set
now contains [1,2].
# randomizedSet.getRandom(); // getRandom() should return either 1 or
2 randomly.
# randomizedSet.remove(1); // Removes 1 from the set, returns true.
Set now contains [2].
# randomizedSet.insert(2); // 2 was already in the set, so return
false.
# randomizedSet.getRandom(); // Since 2 is the only number in the set,
getRandom() will always return 2.

import random
class RandomizedSet:

    def __init__(self):
        #stores the value
        self.arr = []

        #stores the val and map it to an it index
        self.val_to_index = {}


    def insert(self, val: int) -> bool:
        #all happening in O(1)
        if val in self.val_to_index:
            return False

        #put value and its index
        self.val_to_index[val] = len(self.arr)
        #put value at the end of array
        self.arr.append(val)
        return True



    def remove(self, val: int) -> bool:
        #if item not present, return False
        if val not in self.val_to_index:
            return False

        #if item present
        #get its index
```

```
        index_of_val = self.val_to_index[val]
        #get the index of the last element
        last_val = self.arr[ len(self.arr) - 1]

        #put the last element at the new index
        self.arr[index_of_val] = last_val
        #update last val index to index of deleted
        self.val_to_index[last_val] = index_of_val
        self.arr.pop()
        #delete the entry from the hashtable
        del self.val_to_index[val]

        return True

    def getRandom(self) -> int:

        return random.choice(self.arr)


# Your RandomizedSet object will be instantiated and called as such:
# obj = RandomizedSet()
# param_1 = obj.insert(val)
# param_2 = obj.remove(val)
# param_3 = obj.getRandom()

# 238. Product of Array Except Self
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an integer array nums, return an array answer such that
answer[i] is equal to the product of all the elements of nums except
nums[i].

# The product of any prefix or suffix of nums is guaranteed to fit in
a 32-bit integer.

# You must write an algorithm that runs in O(n) time and without using
the division operation.



# Example 1:

# Input: nums = [1,2,3,4]
# Output: [24,12,8,6]
# Example 2:

# Input: nums = [-1,1,0,-3,3]
# Output: [0,0,9,0,0]
```

```python
# logic is to compute the prefix and posfix then multiple the prefix
and postfix
# nums [1,2,3,4]
# prefix: [1,2,6,24] = prefix[i-1] * nums[i]
# postfix: [24,24,12,4] = postfix[i+1] * nums[i]
# answer = [prefix[i-1]] * postfix[i+1] = [24,12,8,4]

# space :O(N + N) with extra O(N) for answer
# Time: O(N)

# Better solution

#use only one space and manipulate it

class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:

        #calcuate prefix
        prefix = 1
        i = 0
        answer = [0] * len(nums)
        while i < len(nums):

            answer[i] = prefix

            prefix *= nums[i]

            i+=1

        postfix = 1

        #calcuate postfix
        j = len(nums)

        while j > 0:
            answer[j - 1] *= postfix
            postfix = nums[j-1] * postfix
            j -=1

        return answer



# Two Integer Sum II
# Given an array of integers numbers that is sorted in non-decreasing
order.

# Return the indices (1-indexed) of two numbers, [index1, index2],
such that they add up to a given target number target
```

```
#  and index1 < index2. Note that index1 and index2 cannot be equal,
therefore you may not use the same element twice.

# There will always be exactly one valid solution.

# Your solution must use
# O
# (
# 1
# )
# O(1) additional space.

# Example 1:
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        l, r = 0, len(numbers) - 1

        while l < r:
            curSum = numbers[l] + numbers[r]

            if curSum > target:
                r -= 1
            elif curSum < target:
                l += 1
            else:
                return [l + 1, r + 1]

# 134. Gas Station
# Solved
# Medium
# Topics
# Companies
# There are n gas stations along a circular route, where the amount of
gas at the ith station is gas[i].

# You have a car with an unlimited gas tank and it costs cost[i] of
gas to travel from the ith station to its next (i + 1)th station.
# You begin the journey with an empty tank at one of the gas stations.

# Given two integer arrays gas and cost, return the starting gas
station's index if you can travel around the circuit once in the
clockwise direction,
# otherwise return -1. If there exists a solution, it is guaranteed to
be unique



# Example 1:

# Input: gas = [1,2,3,4,5], cost = [3,4,5,1,2]
# Output: 3
```

```
# Explanation:
# Start at station 3 (index 3) and fill up with 4 unit of gas. Your
tank = 0 + 4 = 4
# Travel to station 4. Your tank = 4 - 1 + 5 = 8
# Travel to station 0. Your tank = 8 - 2 + 1 = 7
# Travel to station 1. Your tank = 7 - 3 + 2 = 6
# Travel to station 2. Your tank = 6 - 4 + 3 = 5
# Travel to station 3. The cost is 5. Your gas is just enough to
travel back to station 3.
# Therefore, return 3 as the starting index.
# Example 2:

# Input: gas = [2,3,4], cost = [3,4,3]
# Output: -1
# Explanation:
# You can't start at station 0 or 1, as there is not enough gas to
travel to the next station.
# Let's start at station 2 and fill up with 4 unit of gas. Your tank =
0 + 4 = 4
# Travel to station 0. Your tank = 4 - 3 + 2 = 3
# Travel to station 1. Your tank = 3 - 3 + 3 = 3
# You cannot travel back to station 2, as it requires 4 unit of gas
but you only have 3.
# Therefore, you can't travel around the circuit once no matter where
you start.


class Solution:

    def canCompleteCircuit(self, gas: List[int], cost: List[int]) ->
int:

        #optimal = O(N)
        current_gas = 0
        total_gas = 0
        start_index = 0
        for i in range(len(gas)):
            current_gas += gas[i] - cost[i]
            total_gas += gas[i] - cost[i]

            if current_gas < 0:
                current_gas = 0
                start_index = (i + 1) % len(gas)
                continue

        return start_index if total_gas >= 0 else -1


        #brute force = O(N^2)
    def canCompleteCircuit(self, gas: List[int], cost: List[int]) ->
```

```
int:

        total_gas = 0

        for i in range(len(gas)):
            total_gas = 0

            start_index = i
            for j in range(len(gas)):
                total_gas += gas[start_index  % len(gas)] -
cost[start_index % len(gas)]
                if total_gas < 0:
                    break
                start_index +=1

            if total_gas >= 0:
                return i

        return -1

# 12. Integer to Roman
# Solved
# Medium
# Topics
# Companies
# Seven different symbols represent Roman numerals with the following
values:

# Symbol   Value
# I   1
# V   5
# X   10
# L   50
# C   100
# D   500
# M   1000
# Roman numerals are formed by appending the conversions of decimal
place values from highest to lowest. Converting a decimal place value
into a Roman numeral has the following rules:

# If the value does not start with 4 or 9, select the symbol of the
maximal value that can be subtracted from the input,
# append that symbol to the result, subtract its value, and convert
the remainder to a Roman numeral.
# If the value starts with 4 or 9 use the subtractive form
representing one symbol subtracted from the following symbol,
# for example, 4 is 1 (I) less than 5 (V): IV and 9 is 1 (I) less than
10 (X): IX. Only the following subtractive forms are used: 4 (IV), 9
(IX), 40 (XL), 90 (XC), 400 (CD) and 900 (CM).
# Only powers of 10 (I, X, C, M) can be appended consecutively at most
```

```
3 times to represent multiples of 10. You cannot append 5 (V), 50 (L),
or 500 (D) multiple times. If you need to append a symbol 4 times use
the subtractive form.
# Given an integer, convert it to a Roman numeral.



# Example 1:

# Input: num = 3749

# Output: "MMMDCCXLIX"

# Explanation:

# 3000 = MMM as 1000 (M) + 1000 (M) + 1000 (M)
#  700 = DCC as 500 (D) + 100 (C) + 100 (C)
#   40 = XL as 10 (X) less of 50 (L)
#    9 = IX as 1 (I) less of 10 (X)
# Note: 49 is not 1 (I) less of 50 (L) because the conversion is based
on decimal places
# Example 2:

# Input: num = 58

# Output: "LVIII"

# Explanation:

# 50 = L
#  8 = VIII
# Example 3:

# Input: num = 1994

# Output: "MCMXCIV"

# Explanation:

# 1000 = M
#  900 = CM
#   90 = XC
#    4 = IV


class Solution:
    def intToRoman(self, num: int) -> str:
        ans = ""
        symbol_to_value = {
            "M": 1000,
            "CM": 900,
```

```
            "D": 500,
            "CD": 400,
            "C": 100,
            "XC": 90,
            "L": 50,
            "XL": 40,
            "X": 10,
            "IX": 9,
            "V": 5,
            "IV": 4,
            "I": 1

        }

        for symbol, value in symbol_to_value.items():

            while num >= value:
                ans += symbol
                num -= value


        return ans



    def intToRoman(self, num: int) -> str:
        ans = ""

        symbol = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X",
"IX", "V", "IV", "I"]
        value = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9,  5, 4,
1]
        start_index = 0
        while num > 0:

            if num >= value[start_index]:
                ans += symbol[start_index]
                num -= value[start_index]
            else:
                start_index += 1

        return ans


# 6. Zigzag Conversion
# Solved
# Medium
# Topics
# Companies
# The string "PAYPALISHIRING" is written in a zigzag pattern on a
given number of rows like this: (you may want to display this pattern
```

```
in a fixed font for better legibility)

# P   A   H   N
# A P L S I I G
# Y   I   R
# And then read line by line: "PAHNAPLSIIGYIR"

# Write the code that will take a string and make this conversion
given a number of rows:

# string convert(string s, int numRows);


# Example 1:

# Input: s = "PAYPALISHIRING", numRows = 3
# Output: "PAHNAPLSIIGYIR"
# Example 2:

# Input: s = "PAYPALISHIRING", numRows = 4
# Output: "PINALSIGYAHRPI"
# Explanation:
# P     I    N
# A   L S  I G
# Y A   H R
# P     I
# Example 3:

# Input: s = "A", numRows = 1
# Output: "A"

class Solution:
    def convert(self, s: str, numRows: int) -> str:
        ans = ""
        #if rows is 1, then no zig zag
        if numRows == 1 : return s

        for row in range(numRows):

            jump = (numRows - 1) * 2

            #taking a jump of length jump
            for i in range(row, len(s), jump):

                #add
                ans += s[i]

                #now check if we are at the middle row
                #the number of jumps we take is depedent on
                #()
                if (row > 0 and row < numRows - 1 and i + jump - 2 *
```

```
row < len(s)):
                    ans += s[i + jump - 2 * row]
        return ans


# 151. Reverse Words in a String
# Solved
# Medium
# Topics
# Companies
# Given an input string s, reverse the order of the words.

# A word is defined as a sequence of non-space characters. The words
in s will be separated by at least one space.

# Return a string of the words in reverse order concatenated by a
single space.

# Note that s may contain leading or trailing spaces or multiple
spaces between two words. The returned string should only have a
single space separating the words. Do not include any extra spaces.



# Example 1:

# Input: s = "the sky is blue"
# Output: "blue is sky the"
# Example 2:

# Input: s = "  hello world  "
# Output: "world hello"
# Explanation: Your reversed string should not contain leading or
trailing spaces.
# Example 3:

# Input: s = "a good   example"
# Output: "example good a"
# Explanation: You need to reduce multiple spaces between two words to
a single space in the reversed string.


class Solution:
    def reverseWords(self, s: str) -> str:

        s_c = s.split()

        start = 0
        end = len(s_c) - 1

        while start < end:
```

```
            hold = s_c[start]
            s_c[start] = s_c[end]
            s_c[end] = hold

            start +=1
            end -= 1

        ans = ""
        for i in range(len(s_c)):
            if i != len(s_c) -1:
                ans += s_c[i] + " "
            else:
                ans += s_c[i]
        return ans



# 11. Container With Most Water
# Solved
# Medium
# Topics
# Companies
# Hint
# You are given an integer array height of length n. There are n
vertical lines drawn such that the two endpoints of the ith line are
(i, 0) and (i, height[i]).

# Find two lines that together with the x-axis form a container, such
that the container contains the most water.

# Return the maximum amount of water a container can store.

# Notice that you may not slant the container.



# Example 1:


# Input: height = [1,8,6,2,5,4,8,3,7]
# Output: 49
# Explanation: The above vertical lines are represented by array
[1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue
section) the container can contain is 49.
# Example 2:

# Input: height = [1,1]
# Output: 1

class Solution:
```

```python
    def maxArea(self, height: List[int]) -> int:

        start = 0
        end = len(height) - 1
        max_area = 0
        while start < end:

            width = end - start
            min_height = min(height[start], height[end])
            area = width * min_height
            max_area = max(max_area, area)

            if height[start] < height[end]:
                start +=1
            else:
                end -= 1

        return max_area




# Code
# Testcase
# Testcase
# Test Result
# 167. Two Sum II - Input Array Is Sorted
# Solved
# Medium
# Topics
# Companies
# Given a 1-indexed array of integers numbers that is already sorted
in non-decreasing order, find two numbers such that they add up to a
specific target number. Let these two numbers be numbers[index1] and
numbers[index2] where 1 <= index1 < index2 <= numbers.length.

# Return the indices of the two numbers, index1 and index2, added by
one as an integer array [index1, index2] of length 2.

# The tests are generated such that there is exactly one solution. You
may not use the same element twice.

# Your solution must use only constant extra space.



# Example 1:

# Input: numbers = [2,7,11,15], target = 9
# Output: [1,2]
```

```
# Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2
= 2. We return [1, 2].
# Example 2:

# Input: numbers = [2,3,4], target = 6
# Output: [1,3]
# Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 =
3. We return [1, 3].
# Example 3:

# Input: numbers = [-1,0], target = -1
# Output: [1,2]
# Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2
= 2. We return [1, 2]

class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:

        start = 0
        end = len(numbers) - 1

        while start < end:

            if numbers[start] + numbers[end] > target:
                end -=1
            elif numbers[start] + numbers[end] < target:
                start += 1
            else:
                return [start + 1, end + 1]

        return [-1]

# 42. Trapping Rain Water
# Solved
# Hard
# Topics
# Companies
# Given n non-negative integers representing an elevation map where
the width of each bar is 1, compute how much water it can trap after
raining.



# Example 1:


# Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
# Output: 6
# Explanation: The above elevation map (black section) is represented
by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain
water (blue section) are being trapped.
```

```python
# Example 2:

# Input: height = [4,2,0,3,2,5]
# Output: 9
from typing import List

class Solution:
    def trap(self, height: List[int]) -> int:
        # Arrays to store the maximum height to the left and right of
each element
        maxLeft = [0] * len(height)
        maxRight = [0] * len(height)

        # Calculate the max height to the left of each position
        max_left = 0
        for i in range(len(height)):
            maxLeft[i] = max_left
            max_left = max(max_left, height[i])

        # Calculate the max height to the right of each position
        max_right = 0
        for i in range(len(height) - 1, -1, -1):
            maxRight[i] = max_right
            max_right = max(max_right, height[i])

        # Calculate the total trapped water
        totalWater = 0
        for i in range(len(height)):
            # The trapped water at each position is the minimum of the
left and right max heights minus the current height
            waterAtPosition = min(maxLeft[i], maxRight[i]) - height[i]
            if waterAtPosition > 0:
                totalWater += waterAtPosition

        return totalWater

# 424. Longest Repeating Character Replacement
# Solved
# Medium
# Topics
# Companies
# You are given a string s and an integer k. You can choose any
character of the string and change it to any other uppercase English
character. You can perform this operation at most k times.

# Return the length of the longest substring containing the same
letter you can get after performing the above operations.
```

```python
# Example 1:

# Input: s = "ABAB", k = 2
# Output: 4
# Explanation: Replace the two 'A's with two 'B's or vice versa.
# Example 2:

# Input: s = "AABABBA", k = 1
# Output: 4
# Explanation: Replace the one 'A' in the middle with 'B' and form
"AABBBBA".
# The substring "BBBB" has the longest repeating letters, which is 4.
# There may exists other ways to achieve this answer too.

class Solution:
    def characterReplacement(self, s: str, k: int) -> int:
        # Initialize the left pointer of the sliding window
        l = 0
        # Dictionary to count the frequency of each character in the
current window
        char_map = {}
        # Variable to store the length of the longest valid substring
        res = 0

        # Iterate with the right pointer over each character in the
string
        for r in range(len(s)):
            # Increment the count of the current character in the
character map
            char_map[s[r]] = char_map.get(s[r], 0) + 1

            # Check if the current window is valid:
            # (window length) - (most frequent character count) should
be <= k
            # This condition checks if the number of characters that
need replacement
            # to make all characters in the window the same is within
the allowed limit 'k'
            while (r - l + 1) - max(char_map.values()) > k:
                # If not valid, shrink the window from the left by
moving the left pointer 'l' to the right
                # and decrement the count of the character that is
being removed from the window
                char_map[s[l]] -= 1
                l += 1

            # Update the result with the maximum length of a valid
window found so far
            res = max(res, r - l + 1)
```

```
        # Return the length of the longest valid substring
        return res


# 567. Permutation in String
# Attempted
# Medium
# Topics
# Companies
# Hint
# Given two strings s1 and s2, return true if s2 contains a
permutation of s1, or false otherwise.

# In other words, return true if one of s1's permutations is the
substring of s2.



# Example 1:

# Input: s1 = "ab", s2 = "eidbaooo"
# Output: true
# Explanation: s2 contains one permutation of s1 ("ba").
# Example 2:

# Input: s1 = "ab", s2 = "eidboaoo"
# Output: false

class Solution:
    #Time limit exceeded
    def checkInclusion(self, s1: str, s2: str) -> bool:

        def backtrack(start, word):

            if len(start) == len(s1) and start in s2:
                return True

            for i in range(len(word)):
                excluded = word[:i] + word[i+1:]
                if backtrack(start+word[i], excluded):
                    return True

            return False

        return backtrack("", s1)


class Solution:
    # Time limit exceeded in the original implementation
    def checkInclusion(self, s1: str, s2: str) -> bool:
        # Initialize character frequency counts for s1 and a sliding
```

```
window in s2
        count_s1 = [0] * 26  # Frequency array for characters in s1
        count_s2 = [0] * 26  # Frequency array for the current window
in s2

        # Populate the frequency array for s1
        for i in range(len(s1)):
            # Map the character to its index by calculating its offset
from 'z'
            count_s1[ord('z') - ord(s1[i])] += 1

        # Initialize the left pointer of the sliding window
        l = 0

        # Iterate with the right pointer over each character in s2
        for r in range(len(s2)):
            # Add the current character to the count of the current
window in s2
            count_s2[ord('z') - ord(s2[r])] += 1

            # Shrink the window if its size exceeds the length of s1
            while (r - l + 1) > len(s1):
                # Check if the frequency array of the current window
matches s1's array
                if count_s1 == count_s2:
                    return True
                else:
                    # If not, remove the leftmost character from the
window and increment the left pointer
                    count_s2[ord('z') - ord(s2[l])] -= 1
                    l += 1

        # Final check to see if the last window matches after the loop
completes
        return count_s1 == count_s2


class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
        # Initialize frequency counts for s1 and the current window in
s2
        count_s1 = [0] * 26  # Frequency array for characters in s1
        count_s2 = [0] * 26  # Frequency array for the current window
in s2

        # If s1 is longer than s2, s2 can't contain a permutation of
s1
        if len(s1) > len(s2):
            return False
```

```
        # Populate the frequency array for s1
        for char in s1:
            count_s1[ord(char) - ord('a')] += 1

        # Initial window setup: populate the frequency array for the
first window of s2
        for i in range(len(s1)):
            count_s2[ord(s2[i]) - ord('a')] += 1

        # Initialize the number of matches between count_s1 and
count_s2
        matches = 0
        for i in range(26):
            if count_s1[i] == count_s2[i]:
                matches += 1

        # Sliding window over the rest of s2
        l = 0
        for r in range(len(s1), len(s2)):
            if matches == 26:
                return True

            # Add the new character to the window
            index = ord(s2[r]) - ord('a')
            count_s2[index] += 1
            # Check if the new character added makes a match
            if count_s1[index] == count_s2[index]:
                matches += 1
            elif count_s1[index] + 1 == count_s2[index]:
                matches -= 1

            # Remove the character that's no longer in the window
            index = ord(s2[l]) - ord('a')
            count_s2[index] -= 1
            # Check if the character removed maintains the match
            if count_s1[index] == count_s2[index]:
                matches += 1
            elif count_s1[index] - 1 == count_s2[index]:
                matches -= 1

            # Move the left pointer
            l += 1

        # Final check after the loop completes
        return matches == 26

# 76. Minimum Window Substring
# Attempted
# Hard
# Topics
```

```python
# Companies
# Hint
# Given two strings s and t of lengths m and n respectively, return
the minimum window
# substring
#  of s such that every character in t (including duplicates) is
included in the window. If there is no such substring, return the
empty string "".

# The testcases will be generated such that the answer is unique.



# Example 1:

# Input: s = "ADOBECODEBANC", t = "ABC"
# Output: "BANC"
# Explanation: The minimum window substring "BANC" includes 'A', 'B',
and 'C' from string t.
# Example 2:

# Input: s = "a", t = "a"
# Output: "a"
# Explanation: The entire string s is the minimum window.
# Example 3:

# Input: s = "a", t = "aa"
# Output: ""
# Explanation: Both 'a's from t must be included in the window.
# Since the largest window of s only has one 'a', return empty string.

class Solution:
  #TLE
    def minWindow(self, s: str, t: str) -> str:

        def getCount(sub_s):
            count_s = [0] * 128
            for i in range(len(sub_s)):
                count_s[ord(sub_s[i])] += 1
            return count_s
        def compare(count_t, count_s):
            for i in range(len(count_t)):
                if count_s[i] - count_t[i] < 0:
                    return False
            return True

        count_t = [0] * 128
        for i in range(len(t)):
            count_t[ord(t[i])] += 1  # Fixed here to use t instead of
s
```

```
        resLen = float('inf')
        res = [-1, -1]

        for r in range(len(s)):
            for l in range(r + 1):  # Corrected range to include `r` properly
                count_sub_s = getCount(s[l:r + 1])
                print(s[l:r + 1])
                # Check if current substring matches and is smaller
                if compare(count_t, count_sub_s)  and (r - l + 1) < resLen:
                    print("yes")
                    resLen = r - l + 1
                    res = [l, r]

        return s[res[0]:res[1] + 1] if resLen != float('inf') else ""



class Solution:
    def minWindow(self, s: str, t: str) -> str:
        if not s or not t:
            return ""

        count_t = {}
        for char in t:
            count_t[char] = 1 + count_t.get(char, 0)

        count_s = {}
        have, need = 0, len(count_t)  # 'need' is the number of distinct characters
        res, resLen = [-1, -1], float('inf')
        l = 0

        for r in range(len(s)):
            count_s[s[r]] = 1 + count_s.get(s[r], 0)

            # Check if current character's count in window matches the required count in t
            if s[r] in count_t and count_s[s[r]] == count_t[s[r]]:
                have += 1

            # Shrink the window until it no longer satisfies the condition
            while have == need:
                # Update result if the current window is smaller
                if (r - l + 1) < resLen:
                    resLen = r - l + 1
                    res = [l, r]
```

```
                # Shrink the window from the left
                count_s[s[l]] -= 1
                if s[l] in count_t and count_s[s[l]] < count_t[s[l]]:
                    have -= 1
                l += 1

        # Return the minimum window substring
        return s[res[0]:res[1] + 1] if resLen != float('inf') else ""



# 239. Sliding Window Maximum
# Attempted
# Hard
# Topics
# Companies
# Hint
# You are given an array of integers nums, there is a sliding window
of size k which is moving from the very left of the array to the very
right.
# You can only see the k numbers in the window. Each time the sliding
window moves right by one position.

# Return the max sliding window.



# Example 1:

# Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
# Output: [3,3,5,5,6,7]
# Explanation:
# Window position                Max
# ---------------               -----
# [1  3  -1] -3  5  3  6  7       3
#  1 [3  -1  -3] 5  3  6  7       3
#  1  3 [-1  -3  5] 3  6  7       5
#  1  3  -1 [-3  5  3] 6  7       5
#  1  3  -1  -3 [5  3  6] 7       6
#  1  3  -1  -3  5 [3  6  7]      7
# Example 2:

# Input: nums = [1], k = 1
# Output: [1]

 class Solution:
    #TLE
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
```

```
        res = []
        l = 0
        for r in range(len(nums)):

            while (r-l+1) == k:
                sub_max = max( nums[l:r+1] )
                res.append(sub_max)
                l+=1

        return res



class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        output = []
        q = deque()  # index
        l = r = 0

        while r < len(nums):
            while q and nums[q[-1]] < nums[r]:
                q.pop()
            q.append(r)

            if l > q[0]:
                q.popleft()

            if (r -l+ 1) == k:
                output.append(nums[q[0]])
                l += 1
            r += 1

        return output

# 15. 3Sum
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an integer array nums, return all the triplets [nums[i],
nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] +
nums[j] + nums[k] == 0.

# Notice that the solution set must not contain duplicate triplets.



# Example 1:
```

```
# Input: nums = [-1,0,1,2,-1,-4]
# Output: [[-1,-1,2],[-1,0,1]]
# Explanation:
# nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
# nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
# nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
# The distinct triplets are [-1,0,1] and [-1,-1,2].
# Notice that the order of the output and the order of the triplets
does not matter.
# Example 2:

# Input: nums = [0,1,1]
# Output: []
# Explanation: The only possible triplet does not sum up to 0.
# Example 3:

# Input: nums = [0,0,0]
# Output: [[0,0,0]]
# Explanation: The only possible triplet sums up to 0.

class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:

        nums.sort()
        ans = []
        for i in range(len(nums)):

            #make sure we are not reusing the same nums[i]
            #if current nums[i] is same as previous
            #we have found it possible 3sum or it doesn't exist
            if i > 0 and nums[i] == nums[i - 1]:
                continue

            left = i + 1
            right = len(nums) - 1

            while left < right:

                if nums[i] + nums[left] + nums[right] > 0:
                    right -= 1
                elif nums[i] + nums[left] + nums[right] < 0:
                    left += 1
                else:
                    ans.append([nums[i], nums[left], nums[right]])
                    left +=1

                    #now make sure we are not reusing left again

                    while nums[left] == nums[left - 1] and left <
right:
```

```
                        left += 1
        return ans


# Code


# Testcase
# Testcase
# Test Result
# 209. Minimum Size Subarray Sum
# Solved
# Medium
# Topics
# Companies
# Given an array of positive integers nums and a positive integer
target, return the minimal length of a
# subarray
#  whose sum is greater than or equal to target. If there is no such
subarray, return 0 instead.



# Example 1:

# Input: target = 7, nums = [2,3,1,2,4,3]
# Output: 2
# Explanation: The subarray [4,3] has the minimal length under the
problem constraint.
# Example 2:

# Input: target = 4, nums = [1,4,4]
# Output: 1
# Example 3:

# Input: target = 11, nums = [1,1,1,1,1,1,1,1]
# Output: 0

class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:

        min_size = float('inf')
        sum = 0
        left_pointer = 0

        for i in range(len(nums)):

            sum += nums[i]

            while sum >= target:
```

```
                window = i - left_pointer + 1
                min_size = min(min_size, window)
                sum -= nums[left_pointer]
                left_pointer += 1

        return min_size if min_size != float('inf') else 0


# 3. Longest Substring Without Repeating Characters
# Solved
# Medium
# Topics
# Companies
# Hint
# Given a string s, find the length of the longest
# substring
#  without repeating characters.



# Example 1:

# Input: s = "abcabcbb"
# Output: 3
# Explanation: The answer is "abc", with the length of 3.
# Example 2:

# Input: s = "bbbbb"
# Output: 1
# Explanation: The answer is "b", with the length of 1.
# Example 3:

# Input: s = "pwwkew"
# Output: 3
# Explanation: The answer is "wke", with the length of 3.
# Notice that the answer must be a substring, "pwke" is a subsequence
and not a substring.

class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:

        if len(s) == 1:
            return 1

        lon_sub = 0
        char_set = set()
        left_pointer = 0
        for i in range(len(s)):

            while s[i] in char_set:
```

```
                char_set.remove(s[left_pointer])
                left_pointer +=1
            window = i - left_pointer + 1
            lon_sub = max(lon_sub, window)
            char_set.add(s[i])

        return lon_sub

# 36. Valid Sudoku
# Solved
# Medium
# Topics
# Companies
# Determine if a 9 x 9 Sudoku board is valid. Only the filled cells
need to be validated according to the following rules:

# Each row must contain the digits 1-9 without repetition.
# Each column must contain the digits 1-9 without repetition.
# Each of the nine 3 x 3 sub-boxes of the grid must contain the digits
1-9 without repetition.
# Note:

# A Sudoku board (partially filled) could be valid but is not
necessarily solvable.
# Only the filled cells need to be validated according to the
mentioned rules.


# Example 1:


# Input: board =
# [["5","3",".",".","7",".",".",".","."]
# ,["6",".",".","1","9","5",".",".","."]
# ,[".","9","8",".",".",".",".","6","."]
# ,["8",".",".",".","6",".",".",".","3"]
# ,["4",".",".","8",".","3",".",".","1"]
# ,["7",".",".",".",".","2",".",".",".","6"]
# ,[".","6",".",".",".",".",".","2","8","."]
# ,[".",".",".","4","1","9",".",".","5"]
# ,[".",".",".",".",".","8",".",".","7","9"]]
# Output: true
# Example 2:

# Input: board =
# [["8","3",".",".","7",".",".",".","."]
# ,["6",".",".","1","9","5",".",".","."]
# ,[".","9","8",".",".",".",".","6","."]
# ,["8",".",".",".","6",".",".",".","3"]
# ,["4",".",".","8",".","3",".",".","1"]
```

```
# ,["7",".",".",".","2",".",".",".","6"]
# ,[".","6",".",".",".",".",".","2","8","."]
# ,[".",".",".","4","1","9",".",".","5"]
# ,[".",".",".",".","8",".",".","7","9"]]
# Output: false
# Explanation: Same as Example 1, except with the 5 in the top left
corner being modified to 8. Since there are two 8's in the top left
3x3 sub-box, it is invalid.

class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:

        row = collections.defaultdict(set)
        col = collections.defaultdict(set)
        square = collections.defaultdict(set)

        for r in range(len(board)):
            for c in range(len(board[r])):

                if board[r][c] == ".":
                    continue

                if board[r][c] in row[r] or board[r][c] in col[c] or
board[r][c] in square[(r//3, c//3)]:
                    return False

                row[r].add( board[r][c] )
                col[c].add( board[r][c] )
                square[(r//3, c//3)]. add (board[r][c] )



        return True

# 54. Spiral Matrix
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an m x n matrix, return all elements of the matrix in spiral
order.



# Example 1:


# Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
# Output: [1,2,3,6,9,8,7,4,5]
# Example 2:
```

```python
# Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
# Output: [1,2,3,4,8,12,11,10,9,5,6,7]

class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        ans = []
        left = 0
        right = len(matrix[0])
        top = 0
        bottom = len(matrix)

        while left < right and top < bottom:

            #first get the top elements

            for i in range(left, right):
                ans.append(matrix[top][i])
            #move top down by 1
            top +=1

            #second, the right side
            for i in range(top, bottom):
                ans.append(matrix[i][right - 1])
            #move right borders to the left
            right -=1

            #This is needed incase of a the following examples
            #[1,2,3] or
            # [1,
            # 2,
            # 3]
            if not (left < right and top < bottom):
                break

            #get right to left
            for i in range(right -1, left -1, -1):
                ans.append(matrix[bottom-1][i])
            #bring bottom up
            bottom -= 1
            #bottom to top

            for i in range(bottom - 1, top -1, -1):
                ans.append(matrix[i][left])
            #move left by 1
            left +=1

        return ans
```

```python
# 73. Set Matrix Zeroes
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an m x n integer matrix matrix, if an element is 0, set its
entire row and column to 0's.

# You must do it in place.



# Example 1:


# Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
# Output: [[1,0,1],[0,0,0],[1,0,1]]
# Example 2:


# Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
# Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]

class Solution:
    def nullifyRows(self, row, len_of_row, matrix):

        for i in range(len_of_row):
            matrix[row][i] = 0

    def nullify_column(self, col, len_of_col, matrix):

        for i in range(len_of_col):
            matrix[i][col] = 0

    def setZeroes(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """

        rows_to_zero = set()
        cols_to_zero = set()

        for r in range(len(matrix)):

            for c in range(len(matrix[r])):

                if matrix[r][c] == 0:

                    #identify which rows to zero out
                    rows_to_zero.add(r)
```

```
                    cols_to_zero.add(c)

        for r in rows_to_zero:
            self.nullifyRows(r, len(matrix[r]), matrix)
        for c in cols_to_zero:
            self.nullify_column(c, len(matrix), matrix)



# Code
# Code
# Accepted
# Testcase
# Test Result
# Test Result
# 994. Rotting Oranges
# Solved
# Medium
# Topics
# Companies
# You are given an m x n grid where each cell can have one of three
values:

# 0 representing an empty cell,
# 1 representing a fresh orange, or
# 2 representing a rotten orange.
# Every minute, any fresh orange that is 4-directionally adjacent to a
rotten orange becomes rotten.

# Return the minimum number of minutes that must elapse until no cell
has a fresh orange. If this is impossible, return -1.



# Example 1:


# Input: grid = [[2,1,1],[1,1,0],[0,1,1]]
# Output: 4
# Example 2:

# Input: grid = [[2,1,1],[0,1,1],[1,0,1]]
# Output: -1
# Explanation: The orange in the bottom left corner (row 2, column 0)
is never rotten, because rotting only happens 4-directionally.
# Example 3:

# Input: grid = [[0,2]]
# Output: 0
# Explanation: Since there are already no fresh oranges at minute 0,
the answer is just 0.
```

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

        rows = len(grid)
        cols = len(grid[0])

        fresh = 0
        q = deque()
        visited = set()
        for r in range(rows):
            for c in range(cols):

                if grid[r][c] == 1:
                    fresh +=1

                if grid[r][c] == 2:
                    q.append((r,c))
                    visited.add((r,c))

        time = 0
        while q and fresh > 0:

            for i in range(len(q)):

                r, c = q.popleft()

                for dr, dc in [[-1,0], [0,-1], [1,0], [0,1]]:
                    row, col = r+dr, c+dc
                    if row not in range(rows) or col not in
range(cols) or grid[row][col] != 1 or (row, col)  in visited:
                        continue

                    q.append((row, col))
                    visited.add((row, col))
                    grid[row][col] == 2
                    fresh -=1

            time +=1
        return time if fresh == 0 else - 1

# 289. Game of Life
# Solved
# Medium
# Topics
# Companies
# According to Wikipedia's article: "The Game of Life, also known
simply as Life, is a cellular automaton devised by the British
mathematician John Horton Conway in 1970."

# The board is made up of an m x n grid of cells, where each cell has
```

```
an initial state: live (represented by a 1)
# or dead (represented by a 0). Each cell interacts with its eight
neighbors (horizontal, vertical, diagonal) using the following four
rules (taken from the above Wikipedia article):

# Any live cell with fewer than two live neighbors dies as if caused
by under-population.
# Any live cell with two or three live neighbors lives on to the next
generation.
# Any live cell with more than three live neighbors dies, as if by
over-population.
# Any dead cell with exactly three live neighbors becomes a live cell,
as if by reproduction.
# The next state is created by applying the above rules simultaneously
to every cell in the current state, where births
# and deaths occur simultaneously. Given the current state of the m x
n grid board, return the next state.



# Example 1:


# Input: board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
# Output: [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]
# Example 2:


# Input: board = [[1,1],[1,0]]
# Output: [[1,1],[1,1]]

class Solution:

    def countNeighbors(self, row, column, board):
        count = 0
        for r in range(row -1, row + 2):
            for c in range(column-1, column + 2):
                #(r == row and c == column ): ignore the cell whose
neigbor we are checking
                #r < 0 or r == len(board) : row out of bounds
                #c== len(board[r]) or c < 0: column out of bounds
                if (r == row and c == column ) or r < 0 or r ==
len(board) or c== len(board[r]) or c < 0:
                    continue

                #if board is 1 now or 3, that means it previous state
was 1
                if board[r][c] in [1,3]:
                    count +=1
        return count
```

```
    def gameOfLife(self, board: List[List[int]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        row = len(board)
        column = len(board[0])

        for r in range(row):
            for c in range(column):

                if board[r][c] == 1:
                    neigbors = self.countNeighbors(r,c,board)
                    #if alive and neighbors are 2 or 3 then it's
changing from 1 to 3
                    if neigbors in [2,3]:
                        board[r][c] = 3
                    #otherwise board remains unchanged
                #if board[r][c] == 0
                else:
                    neigbors = self.countNeighbors(r,c,board)
                    #if neighbors is exactly 3, cells to come to life
                    #if neiigbhors is 3 when zero, make it to 2
                    if neigbors == 3:
                        board[r][c] = 2


        #now update the board

        for r in range(row):
            for c in range(column):
                if board[r][c] in [2,3]:
                    board[r][c] = 1
                else:
                    board[r][c] = 0
```

48. Rotate Image
Solved
Medium
Topics
Companies
You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

```
Example 1:


Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
Example 2:


Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        row = len(matrix)
        column = len(matrix[0])

        #transpose
        for r in range(row):
            #above the diagonal
            for c in range(r+1, column):

                matrix[r][c], matrix[c][r] = matrix[c][r], matrix[r]
[c]

        #reverse each row vector

        for i in range(row):

            matrix[i].reverse()

    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """

        left = 0
        right = len(matrix[0]) - 1

        top = 0
        bottom = len(matrix) - 1

        while left < right and top < bottom:

            #go from left to right up to the last element
            #because we will modify that too

            for i in range(right - left):


                top_left = matrix[top][left + i]
```

```
                #move bottom_left to top_left
                matrix[top][left + i] = matrix[bottom - i][left]

                #move bottom right to bottom left
                matrix[bottom - i][left] = matrix[bottom][right - i]

                #move top right to bottom right
                matrix[bottom][right - i] = matrix[top + i][right]

                #move top_left to top right
                matrix[top + i][right] = top_left

            left +=1
            right -=1

            top += 1
            bottom -=1




# 128. Longest Consecutive Sequence
# Solved
# Medium
# Topics
# Companies
# Given an unsorted array of integers nums, return the length of the
longest consecutive elements sequence.

# You must write an algorithm that runs in O(n) time.



# Example 1:

# Input: nums = [100,4,200,1,3,2]
# Output: 4
# Explanation: The longest consecutive elements sequence is [1, 2, 3,
4]. Therefore its length is 4.
# Example 2:

# Input: nums = [0,3,7,2,5,8,4,6,0,1]
# Output: 9
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:

        numSet = set(nums)
        lon_con_seq = 0
```

```
        for i in range(len(nums)):
            #if there is not a previous num,
            #then it must be the first
            if nums[i] - 1 not in numSet:

                #a new set whose length is 1
                length = 1

                #check for the increasing subsequence

                while nums[i] + length in numSet:
                    length +=1

                lon_con_seq = max(lon_con_seq, length)

        return lon_con_seq


# 49. Group Anagrams
# Solved
# Medium
# Topics
# Companies
# Given an array of strings strs, group the anagrams together. You can
return the answer in any order.

# An Anagram is a word or phrase formed by rearranging the letters of
a different word or phrase, typically using all the original letters
exactly once.



# Example 1:

# Input: strs = ["eat","tea","tan","ate","nat","bat"]
# Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
# Example 2:

# Input: strs = [""]
# Output: [[""]]
# Example 3:

# Input: strs = ["a"]
# Output: [["a"]]

class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        #an hashmap of a list
        word_map = defaultdict(list)
```

```
        for word in strs:
            key = []
            for c in word:
                key.append(ord(c))

            key.sort()

            #put the key into the word_map
            #since a list can't be a key
            #we use a tuple
            word_map[ tuple(key) ].append( word)


        #return an list of list of the values
        return word_map.values()

    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        #an hashmap of a list
        word_map = defaultdict(list)


        for word in strs:
            char_key = [0] * 26
            for c in word:
                char_key[ord('z') - ord(c)] +=1

            #put the key into the word_map
            #since a list can't be a key
            #we use a tuple
            word_map[ tuple(char_key) ].append( word)


        #return an list of list of the values
        return word_map.values()



# Code
# Testcase
# Testcase
# Test Result
# 56. Merge Intervals
# Solved
# Medium
# Topics
# Companies
# Given an array of intervals where intervals[i] = [starti, endi],
merge all overlapping intervals,
#  and return an array of the non-overlapping intervals that cover all
the intervals in the input.
```

```python
# Example 1:

# Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
# Output: [[1,6],[8,10],[15,18]]
# Explanation: Since intervals [1,3] and [2,6] overlap, merge them
into [1,6].
# Example 2:

# Input: intervals = [[1,4],[4,5]]
# Output: [[1,5]]
# Explanation: Intervals [1,4] and [4,5] are considered overlapping.

class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        ans = []

        intervals.sort(key = lambda i: i[0])

        #put the first interval in it

        ans.append(intervals[0])

        #start from one

        for i in range(1, len(intervals)):

            previous = ans[-1]
            previous_start = previous[0]
            previous_end = previous[1]

            current_start = intervals[i][0]
            current_end = intervals[i][1]

            if current_start <= previous_end:
                #there is interval
                #modify the previous interval
                previous[0] = previous[0]
                previous[1] = max(previous_end, current_end)
            else:
                #no interval
                ans.append(intervals[i])

        return ans


# Code


# Testcase
```

```
# Testcase
# Test Result
# 57. Insert Interval
# Solved
# Medium
# Topics
# Companies
# Hint
# You are given an array of non-overlapping intervals intervals where
intervals[i] = [starti, endi]
# represent the start and the end of the ith interval and intervals is
sorted in ascending order by starti.
# You are also given an interval newInterval = [start, end] that
represents the start and end of another interval.

# Insert newInterval into intervals such that intervals is still
sorted in ascending order
# by starti and intervals still does not have any overlapping
intervals (merge overlapping intervals if necessary).

# Return intervals after the insertion.

# Note that you don't need to modify intervals in-place. You can make
a new array and return it.



# Example 1:

# Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
# Output: [[1,5],[6,9]]
# Example 2:

# Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval =
[4,8]
# Output: [[1,2],[3,10],[12,16]]
# Explanation: Because the new interval [4,8] overlaps with [3,5],
[6,7],[8,10].


class Solution:

    def insert(self, intervals: List[List[int]], newInterval:
List[int]) -> List[List[int]]:
        #add interval to list
        intervals.append(newInterval)
        #sort intervals
        intervals.sort(key = lambda i: i[0])

        ans = []
```

```
        ans.append(intervals[0])

        #start from one

        for i in range(1, len(intervals)):

            previous = ans[-1]
            previous_start = previous[0]
            previous_end = previous[1]

            current_start = intervals[i][0]
            current_end = intervals[i][1]

            if current_start <= previous_end:
                #there is interval
                #modify the previous interval
                previous[0] = previous[0]
                previous[1] = max(previous_end, current_end)
            else:
                #no interval
                ans.append(intervals[i])

        return ans


    def insert(self, intervals: List[List[int]], newInterval:
List[int]) -> List[List[int]]:

        ans = []
        intervals.sort(key = lambda i : i[0])
        for i in range(len(intervals)):

            #check if new_interval is at the right position
            #that is it is not overlapping
            #and it comes before the current interval
            if  newInterval[1] < intervals[i][0]:
                ans.append(newInterval)
                return ans + intervals[i:]
            #if it is not overlapping but it comes after the current
interval
            #append to array
            elif newInterval[0] > intervals[i][1]:
                ans.append(intervals[i])
            else:
                #now if there is an interval
                #modify interval
                newInterval[0] = min(newInterval[0], intervals[i][0])
                newInterval[1] = max(newInterval[1], intervals[i][1])
```

```
        #incase the interval list is empty and new interval is the
first interval
        ans.append(newInterval)

        return ans

# 452. Minimum Number of Arrows to Burst Balloons
# Solved
# Medium
# Topics
# Companies
# There are some spherical balloons taped onto a flat wall that
represents the XY-plane. The balloons are represented as a
# 2D integer array points where points[i] = [xstart, xend] denotes a
balloon whose horizontal diameter stretches between xstart
# and xend. You do not know the exact y-coordinates of the balloons.

# Arrows can be shot up directly vertically (in the positive y-
direction) from different points along the x-axis.
# A balloon with xstart and xend is burst by an arrow shot at x if
xstart <= x <= xend. There is no limit to the number of arrows
# that can be shot. A shot arrow keeps traveling up infinitely,
bursting any balloons in its path.

# Given the array points, return the minimum number of arrows that
must be shot to burst all balloons.



# Example 1:

# Input: points = [[10,16],[2,8],[1,6],[7,12]]
# Output: 2
# Explanation: The balloons can be burst by 2 arrows:
# - Shoot an arrow at x = 6, bursting the balloons [2,8] and [1,6].
# - Shoot an arrow at x = 11, bursting the balloons [10,16] and
[7,12].
# Example 2:

# Input: points = [[1,2],[3,4],[5,6],[7,8]]
# Output: 4
# Explanation: One arrow needs to be shot for each balloon for a total
of 4 arrows.
# Example 3:

# Input: points = [[1,2],[2,3],[3,4],[4,5]]
# Output: 2
# Explanation: The balloons can be burst by 2 arrows:
# - Shoot an arrow at x = 2, bursting the balloons [1,2] and [2,3].
# - Shoot an arrow at x = 4, bursting the balloons [3,4] and [4,5].
```

```python
class Solution:
    def findMinArrowShots(self, points: List[List[int]]) -> int:
        points.sort(key = lambda i:i[0])
        #the key is to get the intersection of the arrows
        ans = len(points)
        prev = points[0]

        for i in range(1, len(points)):

            #intersection found
            if points[i][0] <= prev[1]:
                #merge only intersction
                prev[1] = min(prev[1], points[i][1])
                ans -=1
            else:
                prev = points[i]
        return ans

# 71. Simplify Path
# Solved
# Medium
# Topics
# Companies
# Given an absolute path for a Unix-style file system, which begins
with a slash '/', transform this path into its simplified canonical
path.

# In Unix-style file system context, a single period '.' signifies the
current directory, a double period ".." denotes moving up one
directory level, and multiple slashes such as "//" are interpreted as
a single slash. In this problem, treat sequences of periods not
covered by the previous rules (like "...") as valid names for files or
directories.

# The simplified canonical path should adhere to the following rules:

# It must start with a single slash '/'.
# Directories within the path should be separated by only one slash
'/'.
# It should not end with a slash '/', unless it's the root directory.
# It should exclude any single or double periods used to denote
current or parent directories.
# Return the new path.



# Example 1:

# Input: path = "/home/"
```

```
# Output: "/home"

# Explanation:

# The trailing slash should be removed.


# Example 2:

# Input: path = "/home//foo/"

# Output: "/home/foo"

# Explanation:

# Multiple consecutive slashes are replaced by a single one.

# Example 3:

# Input: path = "/home/user/Documents/../Pictures"

# Output: "/home/user/Pictures"

# Explanation:

# A double period ".." refers to the directory up a level.

# Example 4:

# Input: path = "/../"

# Output: "/"

# Explanation:

# Going one level up from the root directory is not possible.

# Example 5:

# Input: path = "/.../a/../b/c/../d/./"

# Output: "/.../b/d"

# Explanation:

# "..." is a valid name for a directory in this problem.

class Solution:
    def simplifyPath(self, path):
        dirOrFiles = []
        path = path.split("/")
```

```
        for elem in path:
            if dirOrFiles and elem == "..":
                dirOrFiles.pop()
            elif elem not in [".", "", ".."]:
                dirOrFiles.append(elem)

        return "/" + "/".join(dirOrFiles)
    def simplifyPath(self, path: str) -> str:

        stack = []
        curr = ""
        for token in path + "/":

            if token == "/":
                if curr == "..":
                    if len(stack) > 0:
                        stack.pop()
                else:
                    if curr != "" and curr != ".":
                        stack.append(curr)
                curr = ""
            else:
                curr += token


        return "/" + "/".join(stack)

# 155. Min Stack
# Medium
# Topics
# Companies
# Hint
# Design a stack that supports push, pop, top, and retrieving the
minimum element in constant time.

# Implement the MinStack class:

# MinStack() initializes the stack object.
# void push(int val) pushes the element val onto the stack.
# void pop() removes the element on the top of the stack.
# int top() gets the top element of the stack.
# int getMin() retrieves the minimum element in the stack.
# You must implement a solution with O(1) time complexity for each
function.



# Example 1:

# Input
# ["MinStack","push","push","push","getMin","pop","top","getMin"]
```

```
# [[],[-2],[0],[-3],[],[],[],[]]

# Output
# [null,null,null,null,-3,null,0,-2]

# Explanation
# MinStack minStack = new MinStack();
# minStack.push(-2);
# minStack.push(0);
# minStack.push(-3);
# minStack.getMin(); // return -3
# minStack.pop();
# minStack.top();    // return 0
# minStack.getMin(); // return -2

class MinStack:

    def __init__(self):

        self.stack = []
        self.min_stack = []


    def push(self, val: int) -> None:
        self.stack.append(val)

        if self.min_stack:
            val = min(self.min_stack[-1], val)
        self.min_stack.append(val)


    def pop(self) -> None:

        if self.stack and self.min_stack.append:
            self.stack.pop()
            self.min_stack.pop()


    def top(self) -> int:

        return self.stack[-1]


    def getMin(self) -> int:

        return self.min_stack[-1]



# Your MinStack object will be instantiated and called as such:
# obj = MinStack()
```

```
# obj.push(val)
# obj.pop()
# param_3 = obj.top()
# param_4 = obj.getMin()

# 150. Evaluate Reverse Polish Notation
# Solved
# Medium
# Topics
# Companies
# You are given an array of strings tokens that represents an
arithmetic expression in a Reverse Polish Notation.

# Evaluate the expression. Return an integer that represents the value
of the expression.

# Note that:

# The valid operators are '+', '-', '*', and '/'.
# Each operand may be an integer or another expression.
# The division between two integers always truncates toward zero.
# There will not be any division by zero.
# The input represents a valid arithmetic expression in a reverse
polish notation.
# The answer and all the intermediate calculations can be represented
in a 32-bit integer.


# Example 1:

# Input: tokens = ["2","1","+","3","*"]
# Output: 9
# Explanation: ((2 + 1) * 3) = 9
# Example 2:

# Input: tokens = ["4","13","5","/","+"]
# Output: 6
# Explanation: (4 + (13 / 5)) = 6
# Example 3:

# Input: tokens = ["10","6","9","3","+","-
11","*","/","*","17","+","5","+"]
# Output: 22
# Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
# = ((10 * (6 / (12 * -11))) + 17) + 5
# = ((10 * (6 / -132)) + 17) + 5
# = ((10 * 0) + 17) + 5
# = (0 + 17) + 5
# = 17 + 5
# = 22
```

```python
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:

        stack_evaluator = []

        for i in range(len(tokens)):

            if tokens[i] not in {"+", "-", "*", "/"}:
                stack_evaluator.append( int(tokens[i]) )
            else:
                right = stack_evaluator.pop()
                left = stack_evaluator.pop()

                if tokens[i] == "+":
                    stack_evaluator.append(left + right)
                elif tokens[i] == "-":
                    stack_evaluator.append(left - right)
                elif tokens[i] == "*":
                    stack_evaluator.append(left * right)
                else:
                    # stack_evaluator.append(left // right)
                     # Perform integer division with truncation toward
zero
                    stack_evaluator.append(int(left / right))

        return stack_evaluator.pop()

# 138. Copy List with Random Pointer
# Solved
# Medium
# Topics
# Companies
# Hint
# A linked list of length n is given such that each node contains an
additional random pointer, which could point to any node in the list,
or null.

# Construct a deep copy of the list. The deep copy should consist of
exactly n brand new nodes,
# where each new node has its value set to the value of its
corresponding original node.
# Both the next and random pointer of the new nodes should point to
new nodes in the copied list such that the pointers in the original
list and
# copied list represent the same list state. None of the pointers in
the new list should point to nodes in the original list.

# For example, if there are two nodes X and Y in the original list,
where X.random --> Y, then for the corresponding two nodes x and y in
```

```
the copied list, x.random --> y.

# Return the head of the copied linked list.

# The linked list is represented in the input/output as a list of n
nodes. Each node is represented as a pair of [val, random_index]
where:

# val: an integer representing Node.val
# random_index: the index of the node (range from 0 to n-1) that the
random pointer points to, or null if it does not point to any node.
# Your code will only be given the head of the original linked list.

"""
# Definition for a Node.
class Node:
    def __init__(self, x: int, next: 'Node' = None, random: 'Node' =
None):
        self.val = int(x)
        self.next = next
        self.random = random
"""

class Solution:
    def copyRandomList(self, head: 'Optional[Node]') ->
'Optional[Node]':

        old_to_new = {None : None}

        current = head
        while current:

            copy = Node(current.val)

            old_to_new[current] = copy

            current = current.next

        current = head
        while current:

            copy = old_to_new[current]

            copy.next = old_to_new[current.next]
            copy.random = old_to_new[current.random]

            current = current.next

        return old_to_new[head]
```

```python
# 2. Add Two Numbers
# Solved
# Medium
# Topics
# Companies
# You are given two non-empty linked lists representing two non-
negative integers. The digits are stored in reverse order,
# and each of their nodes contains a single digit. Add the two numbers
and return the sum as a linked list.

# You may assume the two numbers do not contain any leading zero,
except the number 0 itself.



# Example 1:


# Input: l1 = [2,4,3], l2 = [5,6,4]
# Output: [7,0,8]
# Explanation: 342 + 465 = 807.
# Example 2:

# Input: l1 = [0], l2 = [0]
# Output: [0]
# Example 3:

# Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
# Output: [8,9,9,9,0,0,0,1]

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2:
Optional[ListNode]) -> Optional[ListNode]:

        def addHelper(node1, node2, carry):

            head = ListNode()

            if not node1 and not node2 and carry <=0:
                return None

            n1 = node1.val if node1 else 0
            n2 = node2.val if node2 else 0

            summation = n1 + n2 + carry
            head = ListNode(summation % 10)
```

```
            carry = summation // 10



            head.next = addHelper(node1.next if node1 else None,
node2.next if node2 else None, carry)
            return head

        return addHelper(l1, l2, 0)

# 2. Add Two Numbers
# Solved
# Medium
# Topics
# Companies
# You are given two non-empty linked lists representing two non-
negative integers. The digits are stored in reverse order,
# and each of their nodes contains a single digit. Add the two numbers
and return the sum as a linked list.

# You may assume the two numbers do not contain any leading zero,
except the number 0 itself.



# Example 1:


# Input: l1 = [2,4,3], l2 = [5,6,4]
# Output: [7,0,8]
# Explanation: 342 + 465 = 807.
# Example 2:

# Input: l1 = [0], l2 = [0]
# Output: [0]
# Example 3:

# Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
# Output: [8,9,9,9,0,0,0,1]

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2:
Optional[ListNode]) -> Optional[ListNode]:

        dummy = ListNode()
        head = dummy
```

```
        carry = 0
        while l1 or l2 or carry > 0:

            n1 = l1.val if l1 else 0
            n2 = l2.val if l2 else 0

            summation = n1 + n2 + carry

            head.next = ListNode(summation % 10)
            carry = summation // 10

            head = head.next

            l1 = l1.next if l1 else None
            l2 = l2.next if l2 else None

        return dummy.next




    def addTwoNumbers(self, l1: Optional[ListNode], l2:
Optional[ListNode]) -> Optional[ListNode]:

        def addHelper(node1, node2, carry):

            head = ListNode()

            if not node1 and not node2 and carry <=0:
                return None

            n1 = node1.val if node1 else 0
            n2 = node2.val if node2 else 0

            summation = n1 + n2 + carry
            head = ListNode(summation % 10)
            carry = summation // 10



            head.next = addHelper(node1.next if node1 else None,
node2.next if node2 else None, carry)
            return head

        return addHelper(l1, l2, 0)

# 19. Remove Nth Node From End of List
# Solved
# Medium
```

```python
# Topics
# Companies
# Hint
# Given the head of a linked list, remove the nth node from the end of
the list and return its head.



# Example 1:


# Input: head = [1,2,3,4,5], n = 2
# Output: [1,2,3,5]
# Example 2:

# Input: head = [1], n = 1
# Output: []
# Example 3:

# Input: head = [1,2], n = 1
# Output: [1]

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

        fast = head
        slow = head
        # advance fast to nth position
        for i in range(n):
            fast = fast.next

        if not fast:
            return head.next


        prev = None
        while fast:
            prev = slow
            slow = slow.next
            fast = fast.next
        # delete the node
        prev.next = slow.next
        return head
```

```python
# 82. Remove Duplicates from Sorted List II
# Solved
# Medium
# Topics
# Companies
# Given the head of a sorted linked list, delete all nodes that have
duplicate numbers, leaving only distinct numbers from the original
list. Return the linked list sorted as well.



# Example 1:


# Input: head = [1,2,3,3,4,4,5]
# Output: [1,2,5]
# Example 2:


# Input: head = [1,1,1,2,3]
# Output: [2,3]
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) ->
Optional[ListNode]:

        if not head:
            return head

        duplicates_table = {}
        current = head

        # First pass to mark duplicates
        while current:
            if current.val in duplicates_table:
                duplicates_table[current.val] = True
            else:
                duplicates_table[current.val] = False
            current = current.next

        # Second pass to remove duplicates
        dummy = ListNode(0, head)
        prev = dummy
        current = head

        while current:
            if duplicates_table[current.val]:
```

```
                prev.next = current.next
            else:
                prev = current
            current = current.next

        return dummy.next


  def deleteDuplicates(self, head: Optional[ListNode]) ->
Optional[ListNode]:

        dummy = ListNode(0, head)

        prev = dummy
        current = head

        while current:

            isDuplicate = False

            while current and current.next and current.val ==
current.next.val:
                current = current.next
                isDuplicate = True

            if isDuplicate:
                prev.next = current.next
            else:
                prev = prev.next
            current = current.next

        return dummy.next

# 92. Reverse Linked List II
# Solved
# Medium
# Topics
# Companies
# Given the head of a singly linked list and two integers left and
right where left <= right, reverse the nodes of the list from position
left to position right, and return the reversed list.



# Example 1:


# Input: head = [1,2,3,4,5], left = 2, right = 4
# Output: [1,4,3,2,5]
# Example 2:
```

```python
# Input: head = [5], left = 1, right = 1
# Output: [5]

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

        dummy = ListNode(0, head)

        leftPrevious = dummy
        current = head
        #step 1
        #take the current pointer to Node of reversing
        for _ in range(left - 1):
            leftPrevious = current
            current = current.next

        #reversal
        rightPrevious = None
        for _ in range(right - left + 1):

            cNext = current.next
            current.next = rightPrevious
            rightPrevious = current

            current = cNext

        #connection
        leftPrevious.next.next = current
        leftPrevious.next = rightPrevious

        return dummy.next



# Code
# Testcase
# Test Result
# Test Result
# 86. Partition List
# Solved
# Medium
# Topics
# Companies
# Given the head of a linked list and a value x, partition it such
```

```
that all nodes less than x come before nodes greater than or equal to
x.

# You should preserve the original relative order of the nodes in each
of the two partitions.



# Example 1:


# Input: head = [1,4,3,2,5,2], x = 3
# Output: [1,2,2,4,3,5]
# Example 2:

# Input: head = [2,1], x = 2
# Output: [1,2]

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

        current = head

        left_head = ListNode(0)  # Dummy head for the left list
        left_tail = left_head

        right_head = ListNode(0)  # Dummy head for the right list
        right_tail = right_head

        while current:
            if current.val < x:
                left_tail.next = ListNode(current.val)
                left_tail = left_tail.next
            else:
                right_tail.next = ListNode(current.val)
                right_tail = right_tail.next
            current = current.next

        # Connect the left and right lists
        left_tail.next = right_head.next

        return left_head.next

# 61. Rotate List
# Solved
```

```python
# Medium
# Topics
# Companies
# Given the head of a linked list, rotate the list to the right by k
places.



# Example 1:


# Input: head = [1,2,3,4,5], k = 2
# Output: [4,5,1,2,3]
# Example 2:


# Input: head = [0,1,2], k = 4
# Output: [2,0,1]

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next

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

        #get the size of list
        size_of_list = 0
        current = head
        while current:
            size_of_list +=1

            current = current.next
        #if k is more is 0, no need to perform rotation
        if size_of_list == 0 or k % size_of_list == 0:
            return head

        #get the rotation k
        k = k % size_of_list

        #advance fast pointer k steps forward
        slow_pointer = head
```

```
        fast_pointer = head
        for _ in range(k):
            fast_pointer = fast_pointer.next

        #get the position before the rotation
        while fast_pointer and fast_pointer.next:
            slow_pointer = slow_pointer.next
            fast_pointer = fast_pointer.next

        #break chain
        newHead = slow_pointer.next
        slow_pointer.next = None

        #connect
        fast_pointer.next = head

        return newHead



# 105. Construct Binary Tree from Preorder and Inorder Traversal
# Solved
# Medium
# Topics
# Companies
# Given two integer arrays preorder and inorder where preorder is the
preorder traversal of a binary tree and
# inorder is the inorder traversal of the same tree, construct and
return the binary tree.



# Example 1:


# Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
# Output: [3,9,20,null,null,15,7]
# Example 2:

# Input: preorder = [-1], inorder = [-1]
# Output: [-1]

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
```

```
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:

    def getIndex(self, arr: List[int], value: int):

        for i in range(len(arr)):
            if arr[i] == value:
                return i
        return -1


    def buildTree(self, preorder: List[int], inorder: List[int]) ->
Optional[TreeNode]:

        if not preorder or not inorder:
            return None

        # #get root value
        # root_value = preorder[0]
        # #get root value index in order traversal
        # root_index = self.getIndex(inorder, root_value)

        # root = TreeNode(root_value)
        # root.left = self.buildTree(preorder[1:root_index+1],
inorder[:root_index])
        # root.right = self.buildTree(preorder[root_index+1:],
inorder[root_index+1:])

        root_value = preorder.pop(0)

        root_index = self.getIndex(inorder, root_value)

        root = TreeNode(root_value)
        root.left = self.buildTree(preorder, inorder[:root_index])
        root.right = self.buildTree(preorder, inorder[root_index+1:])

        return root


# 146. LRU Cache
# Solved
# Medium
# Topics
# Companies
# Design a data structure that follows the constraints of a Least
Recently Used (LRU) cache.

# Implement the LRUCache class:
```

```
# LRUCache(int capacity) Initialize the LRU cache with positive size
capacity.
# int get(int key) Return the value of the key if the key exists,
otherwise return -1.
# void put(int key, int value) Update the value of the key if the key
exists. Otherwise, add the key-value pair to the cache. If the number
of keys exceeds the capacity from this operation, evict the least
recently used key.
# The functions get and put must each run in O(1) average time
complexity.



# Example 1:

# Input
# ["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get",
"get"]
# [[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
# Output
# [null, null, null, 1, null, -1, null, -1, 3, 4]

# Explanation
# LRUCache lRUCache = new LRUCache(2);
# lRUCache.put(1, 1); // cache is {1=1}
# lRUCache.put(2, 2); // cache is {1=1, 2=2}
# lRUCache.get(1);    // return 1
# lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1,
3=3}
# lRUCache.get(2);    // returns -1 (not found)
# lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4,
3=3}
# lRUCache.get(1);    // return -1 (not found)
# lRUCache.get(3);    // return 3
# lRUCache.get(4);    // return 4

class LRUNode:
    def __init__(self, key : int = 0, value :int = 0, prev = None,
next = None):

        self.key = key
        self.value = value
        self.prev = prev
        self.next = next


class LRUCache:

    def __init__(self, capacity: int):
        #set left and right dummy nodes
```

```
        self.left = LRUNode()
        self.right = LRUNode()

        #set the pointers of let and right
        self.left.next = self.right
        self.right.prev = self.left
        #set capacity
        self.capacity = capacity
        #set cache map
        self.cache = {}

    def remove(self, node):
        #get the left and right pointers of the node
        left = node.prev
        right = node.next

        #break the link in between the nodes and update the pointer
        left.next = right
        right.prev = left

    def insert(self, node):

        #insertion can only happen from the right
        #get the rightmost node which is node before dummy right
        left = self.right.prev

        #update the pointers
        left.next = node
        node.prev = left
        #update the pointers
        node.next = self.right
        self.right.prev = node


    def get(self, key: int) -> int:

        #check if item is in list
        if key in self.cache:

            #if key is present, this is current the mostly recently
used
            #remove it and insert it back at the right

            mru = self.cache[key]
            self.remove(mru)
            self.insert(mru)

            return mru.value
        return -1
```

```
    def put(self, key: int, value: int) -> None:

        if key in self.cache:
            mru = self.cache[key]
            self.remove(mru)
            new_node = LRUNode(key, value, None, None)
            self.cache[key] = new_node
            self.insert(new_node)
        else:
            #key does not exist
            new_node = LRUNode(key, value, None, None)
            self.cache[key] = new_node
            self.insert(new_node)



        #now make sure we haven't exceeded capacity
        if len(self.cache) > self.capacity:

            #remove the least recently used
            lru = self.left.next
            self.remove(lru)

            #delete it in the cache map as well
            del self.cache[lru.key]



# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)

# 106. Construct Binary Tree from Inorder and Postorder Traversal
# Solved
# Medium
# Topics
# Companies
# Given two integer arrays inorder and postorder where inorder is the
inorder traversal of a binary tree and
# postorder is the postorder traversal of the same tree, construct and
return the binary tree.



# Example 1:


# Input: inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]
# Output: [3,9,20,null,null,15,7]
# Example 2:
```

```
# Input: inorder = [-1], postorder = [-1]
# Output: [-1]

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def getIndex(self, arr: List[int], value: int):

        for i in range(len(arr)):
            if arr[i] == value:
                return i
        return -1
    def buildTree(self, inorder: List[int], postorder: List[int]) ->
Optional[TreeNode]:

        if not inorder or not postorder:
            return None

        root_val = postorder.pop()
        root_index = self.getIndex(inorder, root_val)

        root = TreeNode(root_val)
        #we have to build right first becasue the right value is the
next to pop
        #if we build left first the nodes will be swapped
        root.right = self.buildTree(inorder[root_index+1:], postorder)
        root.left = self.buildTree(inorder[:root_index], postorder)

        return root

#tree  balanced
      #           1
      #     2           3
      # 4    5      6     7
# Code
# Testcase
# Testcase
# Test Result
# 116. Populating Next Right Pointers in Each Node
# Solved
# Medium
# Topics
# Companies
# You are given a perfect binary tree where all leaves are on the same
level, and every parent has two children. The binary tree has the
```

```
following definition:

# struct Node {
#   int val;
#   Node *left;
#   Node *right;
#   Node *next;
# }
# Populate each next pointer to point to its next right node. If there
is no next right node, the next pointer should be set to NULL.

# Initially, all next pointers are set to NULL.
"""
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left: 'Node' = None, right:
'Node' = None, next: 'Node' = None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
"""

class Solution:
    def connect(self, root: 'Optional[Node]') -> 'Optional[Node]':

        level_start = root

        while level_start:

            current = level_start

            #traverse each level
            while current:
                #if left exist, connct its next to the right
                if current.left:
                    current.left.next = current.right
                #if there is a right node, there must be next node
that has children
                #that the right node can connect to
                #otherwise no connection which is next.left
                if current.right and current.next:
                    current.right.next = current.next.left

                #move current to the next node in the level
                current = current.next
            #move level a step down
            level_start = level_start.left

        return root
```

```
#tree not balanced
      #           1
      #     2           3
      # 4     null  null    5
# 117. Populating Next Right Pointers in Each Node II
"""
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left: 'Node' = None, right:
'Node' = None, next: 'Node' = None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
"""

class Solution:
    def connect(self, root: 'Node') -> 'Node':

        current = root

        while current:

            dummy = Node(0)
            node_modifier = dummy

            #finished this level
            while current:

                if current.left:
                    node_modifier.next = current.left
                    node_modifier = node_modifier.next
                if current.right:
                    node_modifier.next = current.right
                    node_modifier  = node_modifier.next

                current = current.next

            #advance to the next level
            current = dummy.next

        return root

# 129. Sum Root to Leaf Numbers
# Solved
# Medium
# Topics
# Companies
```

```python
# You are given the root of a binary tree containing digits from 0 to
9 only.

# Each root-to-leaf path in the tree represents a number.

# For example, the root-to-leaf path 1 -> 2 -> 3 represents the number
123.
# Return the total sum of all root-to-leaf numbers. Test cases are
generated so that the answer will fit in a 32-bit integer.

# A leaf node is a node with no children.



# Example 1:


# Input: root = [1,2,3]
# Output: 25
# Explanation:
# The root-to-leaf path 1->2 represents the number 12.
# The root-to-leaf path 1->3 represents the number 13.
# Therefore, sum = 12 + 13 = 25.
# Example 2:


# Input: root = [4,9,0,5,1]
# Output: 1026
# Explanation:
# The root-to-leaf path 4->9->5 represents the number 495.
# The root-to-leaf path 4->9->1 represents the number 491.
# The root-to-leaf path 4->0 represents the number 40.
# Therefore, sum = 495 + 491 + 40 = 1026.

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right


class Solution:
  #except the recursive space, no use of extra space
    def sumNumbers(self, root: Optional[TreeNode]) -> int:

        def helper(root, val, ans):

            if not root.left and not root.right:
                last_val = ans.pop()
                ans.append(val + last_val)
```

```
                return

            if root.left:
                helper(root.left, val * 10 + root.left.val, ans)
            if root.right:
                helper(root.right, val * 10 + root.right.val, ans)
        ans = [0]
        helper(root, root.val, ans)
        return ans[0]

    #with recursion space
    #extra space was used
    def sumNumbers(self, root: Optional[TreeNode]) -> int:

        def helper(root, val, arr):

            if not root.left and not root.right:
                arr.append(val)
                return

            if root.left:
                helper(root.left, val * 10 + root.left.val, arr)
            if root.right:
                helper(root.right, val * 10 + root.right.val, arr)


        nums = []
        helper(root, root.val, nums)
        print(nums)
        ans = 0
        for num in nums:
            ans += num

        return ans

# 114. Flatten Binary Tree to Linked List
# Solved
# Medium
# Topics
# Companies
# Hint
# Given the root of a binary tree, flatten the tree into a "linked
list":

# The "linked list" should use the same TreeNode class where the right
child pointer points to the next node in the list and the left child
pointer is always null.
# The "linked list" should be in the same order as a pre-order
traversal of the binary tree.
```

```python
# Example 1:


# Input: root = [1,2,5,3,4,null,6]
# Output: [1,null,2,null,3,null,4,null,5,null,6]

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def __init__(self):
        self.prev = None
    def flatten(self, root: Optional[TreeNode]) -> None:
        """
        Do not return anything, modify root in-place instead.
        """

        if not root:
            return

        #go to the right and update the pointers from the back

        self.flatten(root.right)
        self.flatten(root.left)

        #now update the pointers
        root.right = self.prev
        self.prev = root
        root.left = None



# 173. Binary Search Tree Iterator
# Solved
# Medium
# Topics
# Companies
# Implement the BSTIterator class that represents an iterator over the
in-order traversal of a binary search tree (BST):

# BSTIterator(TreeNode root) Initializes an object of the BSTIterator
class. The root of the BST is given as part of the constructor.
# The pointer should be initialized to a non-existent number smaller
than any element in the BST.
# boolean hasNext() Returns true if there exists a number in the
traversal to the right of the pointer, otherwise returns false.
# int next() Moves the pointer to the right, then returns the number
```

```
at the pointer.
# Notice that by initializing the pointer to a non-existent smallest
number, the first call to next() will return the smallest element in
the BST.

# You may assume that next() calls will always be valid. That is,
there will be at least a next number in the in-order traversal when
next() is called.



# Example 1:


# Input
# ["BSTIterator", "next", "next", "hasNext", "next", "hasNext",
"next", "hasNext", "next", "hasNext"]
# [[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], [], [],
[]]
# Output
# [null, 3, 7, true, 9, true, 15, true, 20, false]

# Explanation
# BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9,
20]);
# bSTIterator.next();    // return 3
# bSTIterator.next();    // return 7
# bSTIterator.hasNext(); // return True
# bSTIterator.next();    // return 9
# bSTIterator.hasNext(); // return True
# bSTIterator.next();    // return 15
# bSTIterator.hasNext(); // return True
# bSTIterator.next();    // return 20
# bSTIterator.hasNext(); // return False

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class BSTIterator:

    def __init__(self, root: Optional[TreeNode]):

        self.stk = []

        #push the node and its left subtree to the stack
        self.pushSubTree(root)
```

```
    def next(self) -> int:
        #since next is always guaranteed to always have an elemnt
        #no need to check for edge cases

        nextNode = self.stk.pop()

        #the smallest number in the tree will always have the next
smallest
        #number in the leftmost part of its right subtree

        self.pushSubTree(nextNode.right)

        return nextNode.val

    def hasNext(self) -> bool:

        #returns true if stk is not empty
        return self.stk

    def pushSubTree(self, subtree : TreeNode):

        while subtree:
            self.stk.append(subtree)
            subtree = subtree.left



# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()


# 235. Lowest Common Ancestor of a Binary Search Tree
# Solved
# Medium
# Topics
# Companies
# Given a binary search tree (BST), find the lowest common ancestor
(LCA) node of two given nodes in the BST.

# According to the definition of LCA on Wikipedia: “The lowest common
ancestor is defined between two nodes p and q as
#  the lowest node in T that has both p and q as descendants (where we
allow a node to be a descendant of itself).”



# Example 1:
```

```python
# Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
# Output: 6
# Explanation: The LCA of nodes 2 and 8 is 6

class Solution:
  def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':

        if not root or root == p or root == q:
            return root
        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)

        if not left:
            return right
        elif not right:
            return left
        else:
            return root
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':
        if p.val < root.val and q.val < root.val:
            return self.lowestCommonAncestor(root.left, p,q)
        elif p.val > root.val and q.val > root.val:
            return self.lowestCommonAncestor(root.right, p,q)
        else:
            return root
        return None
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':

        current = root

        while current:
            #go to the left
            if p.val < current.val and q.val < current.val:
                current = current.left
            #go to the right
            elif p.val > current.val and q.val > current.val:
                current = current.right
            #a split has occured
            else:
                return current

        return None

# 199. Binary Tree Right Side View
# Solved
# Medium
```

```python
# Topics
# Companies
# Given the root of a binary tree, imagine yourself standing on the
right side of it, return the values of the nodes you can see ordered
from top to bottom.



# Example 1:


# Input: root = [1,2,3,null,5,null,4]
# Output: [1,3,4]
# Example 2:

# Input: root = [1,null,3]
# Output: [1,3]
# Example 3:

# Input: root = []
# Output: []
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:
        ans = []
        def helper(root, level):

            if root:
                if len(ans) == level:
                    ans.append(root.val)
                helper(root.right, level + 1)
                helper(root.left, level + 1)
        helper(root, 0)
        return ans

    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:

        ans = []

        queue = []
        if root:
            queue.append(root)

        while len(queue) > 0:
```

```
            size = len(queue)
            last_element = queue[len(queue) - 1]
            ans.append(last_element.val)

            for i in range(size):

                node = queue.pop(0)

                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

        return ans

# 102. Binary Tree Level Order Traversal
# Solved
# Medium
# Topics
# Companies
# Given the root of a binary tree, return the level order traversal of
its nodes' values. (i.e., from left to right, level by level).
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:

        result = []

        queue = []

        if root:
            queue.append(root)

        while len(queue) > 0:

            size = len(queue)
            sub_list = []
            for i in range(size):
                node = queue.pop(0)

                #append node's children first
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
```

```
                sub_list.append(node.val)

            result.append(sub_list)

        return result

# 103. Binary Tree Zigzag Level Order Traversal
# Solved
# Medium
# Topics
# Companies
# Given the root of a binary tree, return the zigzag level order
traversal of its nodes' values. (i.e., from left to right, then right
to left for the next level and alternate between).
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def zigzagLevelOrder(self, root: Optional[TreeNode]) ->
List[List[int]]:

        result = []

        queue = collections.deque()
        if root:
            queue.append(root)
        zig_zag = False
        while len(queue) > 0:

            size = len(queue)
            sub_list = collections.deque()
            for i in range(size):
                node = queue.popleft()

                #append node's children first
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
                if zig_zag:
                    sub_list.appendleft(node.val)
                else:
                    sub_list.append(node.val)
            zig_zag = not zig_zag
            result.append(sub_list)

        return result
```

```python
# 230. Kth Smallest Element in a BST
# Solved
# Medium
# Topics
# Companies
# Hint
# Given the root of a binary search tree, and an integer k, return the
kth smallest value (1-indexed) of all the values of the nodes in the
tree.
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:
        res = []

        def helper(root, k):
            if not root:
                return

            helper(root.left, k)
            if len(res) == k:
                return
            res.append(root.val)
            helper(root.right, k)
        helper(root, k)
        return res[-1]
    def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:

        def helper(root, counter, k):

            if not root:
                return
            helper(root.left, counter, k)

            counter[0] +=1
            print(root.val, counter[0])
            if counter[0] == k:
                counter[1] = root.val

            helper(root.right, counter, k)

        counter = [0, 0]

        helper(root, counter, k)
        return counter[1]

  #optimized
```

```
  def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:

        def helper(root, counter, k):

            if not root:
                return
            #tiny improvement
            #only try to find the kth smallest if we are still looking
for it
            #if found just return to the root
            if counter[0] < k:

                helper(root.left, counter, k)

                counter[0] +=1
                print(root.val, counter[0])
                if counter[0] == k:
                    counter[1] = root.val
                    return

                helper(root.right, counter, k)

        counter = [0, 0]

        helper(root, counter, k)
        return counter[1]

# 98. Validate Binary Search Tree
# Solved
# Medium
# Topics
# Companies
# Given the root of a binary tree, determine if it is a valid binary
search tree (BST).

# A valid BST is defined as follows:

# The left
# subtree
#  of a node contains only nodes with keys less than the node's key.
# The right subtree of a node contains only nodes with keys greater
than the node's key.
# Both the left and right subtrees must also be binary search trees.

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
```

```python
class Solution:
    #start with
    #min -infinity
    #max = infinity
    #everything must be within bounday of [-inf, inf]
    def isValidBSTHelper(self, root, minVal, maxVal) ->bool:

        if not root:
            return True

        if root.val <= minVal or root.val >= maxVal:
            return False

        #everything on the left must less than the root
        left = self.isValidBSTHelper(root.left, minVal, root.val)
        # #everything on the right must greater than the root
        right = self.isValidBSTHelper(root.right, root.val, maxVal)

        return left and right

    def isValidBST(self, root: Optional[TreeNode]) -> bool:

        return self.isValidBSTHelper(root, float('-inf'),
float('inf'))


# Code
# Testcase
# Testcase
# Test Result
# 200. Number of Islands
# Solved
# Medium
# Topics
# Companies
# Given an m x n 2D binary grid grid which represents a map of '1's
(land) and '0's (water), return the number of islands.

# An island is surrounded by water and is formed by connecting
adjacent lands horizontally or vertically. You may assume all four
edges of the grid are all surrounded by water.



# Example 1:

# Input: grid = [
#   ["1","1","1","1","0"],
#   ["1","1","0","1","0"],
#   ["1","1","0","0","0"],
#   ["0","0","0","0","0"]
```

```
# ]
# Output: 1
# Example 2:

# Input: grid = [
#   ["1","1","0","0","0"],
#   ["1","1","0","0","0"],
#   ["0","0","1","0","0"],
#   ["0","0","0","1","1"]

class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:

        rows = len(grid)
        columns = len(grid[0])

        explored = set()
        numIsland = 0

        def dfs(r,c):

            if r not in range(rows) or c not in range(columns) or
grid[r][c] != "1" or (r,c) in explored:
                return
            explored.add((r,c))
            dfs(r-1,c)
            dfs(r+1,c)
            dfs(r,c-1)
            dfs(r,c+1)

        def bfs(r, c):

            q = collections.deque()
            explored.add((r,c))
            q.append((r,c))

            while q:
                i, j = q.popleft()

                for dr, dc in [[1,0],[-1,0], [0,1], [0,-1]]:

                    if i + dr in range(rows) and j + dc in
range(columns) and grid[i + dr][j + dc] == "1" and (i + dr,j + dc) not
in explored:
                        q.append((i + dr,j + dc))
                        explored.add((i + dr,j + dc))


        for r in range(rows):
            for c in range(columns):
```

```
                if grid[r][c] == "1" and (r,c) not in explored:

                    #either bfs or dfs works
                    #but on leetcode dfs is faster and much cleaner
                    # bfs(r,c)
                    dfs(r,c)
                    numIsland +=1

        return numIsland

# 130. Surrounded Regions
# Solved
# Medium
# Topics
# Companies
# You are given an m x n matrix board containing letters 'X' and 'O',
capture regions that are surrounded:

# Connect: A cell is connected to adjacent cells horizontally or
vertically.
# Region: To form a region connect every 'O' cell.
# Surround: The region is surrounded with 'X' cells if you can connect
the region with 'X' cells and none of the region cells are on the edge
of the board.
# A surrounded region is captured by replacing all 'O's with 'X's in
the input matrix board.



# Example 1:

# Input: board = [["X","X","X","X"],["X","O","O","X"],
["X","X","O","X"],["X","O","X","X"]]

# Output: [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],
["X","O","X","X"]]

# Explanation:


# In the above diagram, the bottom region is not captured because it
is on the edge of the board and cannot be surrounded.

# Example 2:

# Input: board = [["X"]]

# Output: [["X"]]

class Solution:
    def solve(self, board: List[List[str]]) -> None:
```

```
        """
        Do not return anything, modify board in-place instead.
        """

        rows, columns = len(board), len(board[0])

        explored = set()

        def dfs(r,c):
            if r not in range(rows) or c not in range(columns) or
board[r][c] != "O":
                return

            board[r][c] = "T"

            dfs(r,c-1)
            dfs(r,c+1)
            dfs(r-1,c)
            dfs(r+1,c)

        #phase 1
        #find unsurrounded and mark is as "T" in the border region

        #check top border and find "O"
        # Phase 1: Mark all 'O's on the border and their connected
'O's as 'T'
        for top in range(columns):
            if board[0][top] == "O":
                dfs(0, top)
        for left in range(rows):
            if board[left][0] == "O":
                dfs(left, 0)
        for right in range(rows):
            if board[right][columns-1] == "O":
                dfs(right, columns-1)
        for bottom in range(columns):
            if board[rows-1][bottom] == "O":
                dfs(rows-1, bottom)

        #phase 2: turn unsurrounded 0 to x
        #this will in fact be the surrounded region
        for r in range(rows):
            for c in range(columns):

                if board[r][c] == "O":
                    board[r][c] = "X"

        #phase 3: turn to unsurrounded T back to 0
        for r in range(rows):
            for c in range(columns):
```

```
                if board[r][c] == "T":
                    board[r][c] = "O"

# 399. Evaluate Division
# Medium
# Topics
# Companies
# Hint
# You are given an array of variable pairs equations and an array of
real numbers values, where equations[i] = [Ai, Bi] and
#  values[i] represent the equation Ai / Bi = values[i]. Each Ai or Bi
is a string that represents a single variable.

# You are also given some queries, where queries[j] = [Cj, Dj]
represents the jth query where you must find the answer for Cj / Dj
= ?.

# Return the answers to all queries. If a single answer cannot be
determined, return -1.0.

# Note: The input is always valid. You may assume that evaluating the
queries will not result in division by zero and that there is no
contradiction.

# Note: The variables that do not occur in the list of equations are
undefined, so the answer cannot be determined for them.



# Example 1:

# Input: equations = [["a","b"],["b","c"]], values = [2.0,3.0],
queries = [["a","c"],["b","a"],["a","e"],["a","a"],["x","x"]]
# Output: [6.00000,0.50000,-1.00000,1.00000,-1.00000]
# Explanation:
# Given: a / b = 2.0, b / c = 3.0
# queries are: a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?
# return: [6.0, 0.5, -1.0, 1.0, -1.0 ]
# note: x is undefined => -1.0
# Example 2:

# Input: equations = [["a","b"],["b","c"],["bc","cd"]], values =
[1.5,2.5,5.0], queries = [["a","c"],["c","b"],["bc","cd"],["cd","bc"]]
# Output: [3.75000,0.40000,5.00000,0.20000]
# Example 3:

# Input: equations = [["a","b"]], values = [0.5], queries =
[["a","b"],["b","a"],["a","c"],["x","y"]]
# Output: [0.50000,2.00000,-1.00000,-1.00000]
```

```python
class Solution:
    def calcEquation(self, equations: List[List[str]], values:
List[float], queries: List[List[str]]) -> List[float]:

        graph = collections.defaultdict(list)

        for i in range(len(equations)):

            val = values[i]
            eq = equations[i]
            a = eq[0]
            b = eq[1]
            graph[a].append([b, val])
            graph[b].append([a, 1/val])

        def bfs(start, target):

            if start not in graph or target not in graph:
                return -1

            queue = collections.deque()
            visited = set()

            queue.append([start, 1])
            visited.add(start)

            while queue:

                nei, w = queue.popleft()

                if nei == target:
                    return w
                for neighbor, weight in graph[nei]:
                    if neighbor not in visited:
                        visited.add(neighbor)
                        queue.append([neighbor, w * weight])
            return -1

        return [bfs(q[0], q[1]) for q in queries]



# 133. Clone Graph
# Solved
# Medium
# Topics
# Companies
# Given a reference of a node in a connected undirected graph.

# Return a deep copy (clone) of the graph.
```

```
# Each node in the graph contains a value (int) and a list
(List[Node]) of its neighbors.

# class Node {
#     public int val;
#     public List<Node> neighbors;
# }


# Test case format:

# For simplicity, each node's value is the same as the node's index
(1-indexed). For example, the first node with val == 1,
# the second node with val == 2, and so on. The graph is represented
in the test case using an adjacency list.

# An adjacency list is a collection of unordered lists used to
represent a finite graph. Each list describes the set of neighbors of
a node in the graph.

# The given node will always be the first node with val = 1. You must
return the copy of the given node as a reference to the cloned graph.



# Example 1:


# Input: adjList = [[2,4],[1,3],[2,4],[1,3]]
# Output: [[2,4],[1,3],[2,4],[1,3]]
# Explanation: There are 4 nodes in the graph.
# 1st node (val = 1)'s neighbors are 2nd node (val = 2) and 4th node
(val = 4).
# 2nd node (val = 2)'s neighbors are 1st node (val = 1) and 3rd node
(val = 3).
# 3rd node (val = 3)'s neighbors are 2nd node (val = 2) and 4th node
(val = 4).
# 4th node (val = 4)'s neighbors are 1st node (val = 1) and 3rd node
(val = 3).

"""
# Definition for a Node.
class Node:
    def __init__(self, val = 0, neighbors = None):
        self.val = val
        self.neighbors = neighbors if neighbors is not None else []
"""

from typing import Optional

class Solution:
```

```
    def cloneGraph(self, node: Optional['Node']) -> Optional['Node']:

        table = {}

        def bfs(node):

            queue = collections.deque()

            if node:
                queue.append(node)

            while queue:

                nd = queue.popleft()
                table[nd] = Node(nd.val)

                for nei in nd.neighbors:
                    if nei not in table:
                        queue.append(nei)
        bfs(node)
        for old, new in table.items():
            for adj in old.neighbors:
                new.neighbors.append(table[adj])

        return table[node] if node else None

class Solution:
    def cloneGraph(self, node: Optional['Node']) -> Optional['Node']:

        table = {}
        def dfs(node):
            if not node or node in table:
                return

            table[node] = Node(node.val)

            for nei in node.neighbors:
                if nei not in table:
                    dfs(nei)
        dfs(node)
        for old, new in table.items():
            for adj in old.neighbors:
                new.neighbors.append(table[adj])

        return table[node] if node else None

# 207. Course Schedule
# Solved
# Medium
# Topics
# Companies
```

```
# Hint
# There are a total of numCourses courses you have to take, labeled
from 0 to numCourses - 1.
# You are given an array prerequisites where prerequisites[i] = [ai,
bi] indicates that you must take course bi first if you want to take
course ai.

# For example, the pair [0, 1], indicates that to take course 0 you
have to first take course 1.
# Return true if you can finish all courses. Otherwise, return false.



# Example 1:

# Input: numCourses = 2, prerequisites = [[1,0]]
# Output: true
# Explanation: There are a total of 2 courses to take.
# To take course 1 you should have finished course 0. So it is
possible.
# Example 2:

# Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
# Output: false
# Explanation: There are a total of 2 courses to take.
# To take course 1 you should have finished course 0, and to take
course 0 you should also have finished course 1. So it is impossible.

class Solution:
    def canFinish(self, numCourses: int, prerequisites:
List[List[int]]) -> bool:

        graph = {}
        #set up adjacency graph
        for i in range(numCourses):
            graph[i] = []

        for course, prereq in prerequisites:
            graph[course].append(prereq)

        #2->1->0 : True
        #1->0->1: False
        cycle, visited = set(), set()

        def dfs(course):

            if course in cycle:
                return False
            #optimization
            if course in visited:
```

```
                return True
            cycle.add(course)

            for pre in graph[course]:

                if not dfs(pre):
                    return False

            #remove course from cycle because it might need to be
called again
            cycle.remove(course)

            #optimization to avoid recalculation
            visited.add(course)

            return True

        for i in range(numCourses):
            if not dfs(i):
                return False
        return True

# 210. Course Schedule II
# Solved
# Medium
# Topics
# Companies
# Hint
# There are a total of numCourses courses you have to take, labeled
from 0 to numCourses - 1. You are given an array prerequisites
# where prerequisites[i] = [ai, bi] indicates that you must take
course bi first if you want to take course ai.

# For example, the pair [0, 1], indicates that to take course 0 you
have to first take course 1.
# Return the ordering of courses you should take to finish all
courses. If there are many valid answers, return any of them. If it is
impossible to finish all courses, return an empty array.



# Example 1:

# Input: numCourses = 2, prerequisites = [[1,0]]
# Output: [0,1]
# Explanation: There are a total of 2 courses to take. To take course
1 you should have finished course 0. So the correct course order is
[0,1].
# Example 2:

# Input: numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]
```

```
# Output: [0,2,1,3]
# Explanation: There are a total of 4 courses to take. To take course
3 you should have finished both courses 1 and 2. Both courses 1 and 2
should be taken after you finished course 0.
# So one correct course order is [0,1,2,3]. Another correct ordering
is [0,2,1,3].
# Example 3:

# Input: numCourses = 1, prerequisites = []
# Output: [0]

class Solution:
    def findOrder(self, numCourses: int, prerequisites:
List[List[int]]) -> List[int]:

        graph = {}

        for i in range(numCourses):
            graph[i] = []

        for course, prereq in prerequisites:
            graph[course].append(prereq)

        #2->1->0 : 0->1->2
        cycle = set()
        visited = set()
        output = []
        def dfs(course):

            if course in cycle:
                return False
            #no need for recomputation
            if course in visited:
                return True
            cycle.add(course)

            for pre in graph[course]:

                if not dfs(pre):
                    return False

            cycle.remove(course)
            visited.add(course)
            output.append(course)
            #This will work but it takes O(N) to check if it has been
appended
            #instead use a set
            # if course not in output:
            #     output.append(course)
```

```
            return True

        for i in range(numCourses):
            if not dfs(i):
                return []

        return output


# Code
# Testcase
# Testcase
# Test Result
# 909. Snakes and Ladders
# Solved
# Medium
# Topics
# Companies
# You are given an n x n integer matrix board where the cells are
labeled from 1 to n2 in a
# Boustrophedon style starting from the bottom left of the board (i.e.
board[n - 1][0]) and alternating direction each row.

# You start on square 1 of the board. In each move, starting from
square curr, do the following:

# Choose a destination square next with a label in the range [curr +
1, min(curr + 6, n2)].
# This choice simulates the result of a standard 6-sided die roll:
i.e., there are always at most 6 destinations, regardless of the size
of the board.
# If next has a snake or ladder, you must move to the destination of
that snake or ladder. Otherwise, you move to next.
# The game ends when you reach the square n2.
# A board square on row r and column c has a snake or ladder if
board[r][c] != -1. The destination of that snake or ladder is board[r]
[c]. Squares 1 and n2 do not have a snake or ladder.

# Note that you only take a snake or ladder at most once per move. If
the destination to a snake or ladder is the start of another snake or
ladder, you do not follow the subsequent snake or ladder.

# For example, suppose the board is [[-1,4],[-1,3]], and on the first
move, your destination square is 2. You follow the ladder to square 3,
but do not follow the subsequent ladder to 4.
# Return the least number of moves required to reach the square n2. If
it is not possible to reach the square, return -1.
```

```
# Example 1:


# Input: board = [[-1,-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1,-1],[-1,-1,-1,-
1,-1,-1],[-1,35,-1,-1,13,-1],[-1,-1,-1,-1,-1,-1],[-1,15,-1,-1,-1,-1]]
# Output: 4
# Explanation:
# In the beginning, you start at square 1 (at row 5, column 0).
# You decide to move to square 2 and must take the ladder to square
15.
# You then decide to move to square 17 and must take the snake to
square 13.
# You then decide to move to square 14 and must take the ladder to
square 35.
# You then decide to move to square 36, ending the game.
# This is the lowest possible number of moves to reach the last
square, so return 4.
# Example 2:

# Input: board = [[-1,-1],[-1,3]]
# Output: 1
class Solution:
    def snakesAndLadders(self, board: List[List[int]]) -> int:

        len_board = len(board)
        goal = len_board * len_board
        queue = collections.deque()
        def coordinates(square):
            r = (square - 1) // len_board
            c = (square - 1) % len_board

            if r % 2 == 0:
                return len_board - 1- r , c

            return len_board - 1-r, len_board-1-c

        queue.append((1,0))
        visited = set()
        visited.add(1)
        while queue:

            square, steps = queue.popleft()

            for i in range(1,7):
                nextsquare = square + i
                r,c = coordinates(nextsquare)
                if board[r][c] != -1:
                    nextsquare = board[r][c]
                if nextsquare == goal:
                    return steps + 1
```

```
                if nextsquare not in visited:
                    queue.append((nextsquare,steps+1))
                    visited.add(nextsquare)

        return -1

# 433. Minimum Genetic Mutation
# Solved
# Medium
# Topics
# Companies
# A gene string can be represented by an 8-character long string, with
choices from 'A', 'C', 'G', and 'T'.

# Suppose we need to investigate a mutation from a gene string
startGene to a gene string endGene where one mutation is defined as
one single character changed in the gene string.

# For example, "AACCGGTT" --> "AACCGGTA" is one mutation.
# There is also a gene bank bank that records all the valid gene
mutations. A gene must be in bank to make it a valid gene string.

# Given the two gene strings startGene and endGene and the gene bank
bank, return the minimum number of mutations needed to mutate from
startGene to endGene. If there is no such a mutation,
# return -1.

# Note that the starting point is assumed to be valid, so it might not
be included in the bank.



# Example 1:

# Input: startGene = "AACCGGTT", endGene = "AACCGGTA", bank =
["AACCGGTA"]
# Output: 1
# Example 2:

# Input: startGene = "AACCGGTT", endGene = "AAACGGTA", bank =
["AACCGGTA","AACCGCTA","AAACGGTA"]
# Output: 2

class Solution:
    def minMutation(self, startGene: str, endGene: str, bank:
List[str]) -> int:

        bank = set(bank)

        if endGene not in bank:
```

```
            return -1

        queue = collections.deque()
        queue.appendleft((startGene, 0))
        visited = set()
        visited.add(startGene)
        while queue:
            gene, level = queue.popleft()

            if gene == endGene:
                return level

            #create next level permutation
            #AACCGGTA
            for i in range(len(gene)):
                for chromo in ['A', 'C', 'G', 'T']:
                    mutation = gene[:i] + chromo + gene[i+1:]

                    if mutation not in visited and mutation in bank:
                        queue.append((mutation, level + 1))
                        visited.add(mutation)
        return -1

# 208. Implement Trie (Prefix Tree)
# Solved
# Medium
# Topics
# Companies
# A trie (pronounced as "try") or prefix tree is a tree data structure
used to efficiently store and retrieve keys in a dataset of strings.
# There are various applications of this data structure, such as
autocomplete and spellchecker.

# Implement the Trie class:

# Trie() Initializes the trie object.
# void insert(String word) Inserts the string word into the trie.
# boolean search(String word) Returns true if the string word is in
the trie (i.e., was inserted before), and false otherwise.
# boolean startsWith(String prefix) Returns true if there is a
previously inserted string word that has the prefix prefix, and false
otherwise.
class TrieNode():
    def __init__(self):
        self.children = {}
        self.end_of_word = False
class Trie:

    def __init__(self):
        self.root = TrieNode()
```

```
    def insert(self, word: str) -> None:
        current = self.root

        for char in word:
            if char not in current.children:
                current.children[char] = TrieNode()
            current = current.children[char]
        current.end_of_word = True


    def search(self, word: str) -> bool:
        current = self.root

        for char in word:
            if char not in current.children:
                return False
            current = current.children[char]
        return current.end_of_word

    def startsWith(self, prefix: str) -> bool:
        current = self.root
        for char in prefix:
            if char not in current.children:
                return False
            current = current.children[char]
        return True


# Your Trie object will be instantiated and called as such:
# obj = Trie()
# obj.insert(word)
# param_2 = obj.search(word)
# param_3 = obj.startsWith(prefix)

# 211. Design Add and Search Words Data Structure
# Solved
# Medium
# Topics
# Companies
# Hint
# Design a data structure that supports adding new words and finding
if a string matches any previously added string.

# Implement the WordDictionary class:

# WordDictionary() Initializes the object.
# void addWord(word) Adds word to the data structure, it can be
matched later.
# bool search(word) Returns true if there is any string in the data
```

```
structure that matches word or false otherwise. word may contain dots
'.' where dots can be matched with any letter.


# Example:

# Input
#
["WordDictionary","addWord","addWord","addWord","search","search","sea
rch","search"]
# [[],["bad"],["dad"],["mad"],["pad"],["bad"],[".ad"],["b.."]]
# Output
# [null,null,null,null,false,true,true,true]

# Explanation
# WordDictionary wordDictionary = new WordDictionary();
# wordDictionary.addWord("bad");
# wordDictionary.addWord("dad");
# wordDictionary.addWord("mad");
# wordDictionary.search("pad"); // return False
# wordDictionary.search("bad"); // return True
# wordDictionary.search(".ad"); // return True
# wordDictionary.search("b.."); // return True

class TrieNode():
    def __init__(self):
        self.children = {}
        self.end_of_word = False
class WordDictionary:

    def __init__(self):
        self.root = TrieNode()
        self.max_char = 0

    def addWord(self, word: str) -> None:
        self.max_char = max(self.max_char, len(word))
        current = self.root
        for char in word:
            if char not in current.children:
                current.children[char] = TrieNode()
            current = current.children[char]
        current.end_of_word = True

    def search(self, word: str) -> bool:
        if len(word) > self.max_char:
            return False

        def dfs(index, root):

            current = root
```

```
            for i in range(index, len(word)):

                if word[i] == ".":
                    for node in current.children.values():
                        if dfs(i+1, node):
                            return True
                    #if the loop executes and no true statement
                    #then no match
                    return False
                else:
                    if word[i] not in current.children:
                        return False
                    current = current.children[word[i]]

            return current.end_of_word

        return dfs(0, self.root)

# Your WordDictionary object will be instantiated and called as such:
# obj = WordDictionary()
# obj.addWord(word)
# param_2 = obj.search(word)


# Code
# Testcase
# Testcase
# Test Result
# 77. Combinations
# Solved
# Medium
# Topics
# Companies
# Given two integers n and k, return all possible combinations of k
numbers chosen from the range [1, n].

# You may return the answer in any order.



# Example 1:

# Input: n = 4, k = 2
# Output: [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
# Explanation: There are 4 choose 2 = 6 total combinations.
# Note that combinations are unordered, i.e., [1,2] and [2,1] are
considered to be the same combination.
# Example 2:

# Input: n = 1, k = 1
# Output: [[1]]
```

```python
# Explanation: There is 1 choose 1 = 1 total combination.
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        output = []

        def backtrack(start, combination):
            if len(combination) == k:
                output.append(combination.copy())
                return

            for i in range(start, n+1):

                combination.append(i)
                backtrack(i+1, combination)
                combination.pop()

        backtrack(1,[])

        return output

# 17. Letter Combinations of a Phone Number
# Solved
# Medium
# Topics
# Companies
# Given a string containing digits from 2-9 inclusive, return all
possible letter combinations that the number could represent. Return
the answer in any order.

# A mapping of digits to letters (just like on the telephone buttons)
is given below. Note that 1 does not map to any letters.




# Example 1:

# Input: digits = "23"
# Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
# Example 2:

# Input: digits = ""
# Output: []
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        output = []
        numToChar = {
            "2": "abc",
            "3": "def",
            "4": "ghi",
            "5": "jkl",
```

```
            "6": "mno",
            "7": "pqrs",
            "8": "tuv",
            "9": "wxyz"
        }

        def backtrack(start, combination):

            if start == len(digits) or len(combination) ==
len(digits):
                output.append(combination)
                return

            currentDigit = digits[start]
            currentString = numToChar[currentDigit]

            for c in currentString:
                backtrack(start+1, combination + c)

        backtrack(0, "")

        return output if digits else []

# 46. Permutations
# Medium
# Topics
# Companies
# Given an array nums of distinct integers, return all the possible
permutations. You can return the answer in any order.



# Example 1:

# Input: nums = [1,2,3]
# Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
# Example 2:

# Input: nums = [0,1]
# Output: [[0,1],[1,0]]
# Example 3:

# Input: nums = [1]
# Output: [[1]]

class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:

        output = []
        def backtrack(combination, arr):
```

```
            if len(combination) == len(nums):
                output.append(combination.copy())

            for i in range(len(arr)):

                combination.append(arr[i])

                left = arr[:i]
                right = arr[i+1:]
                newArr = left + right

                backtrack(combination, newArr)
                combination.pop()
        backtrack([], nums)

        return output


# Code
# Testcase
# Testcase
# Test Result
# 39. Combination Sum
# Solved
# Medium
# Topics
# Companies
# Given an array of distinct integers candidates and a target integer
target, return a list of all unique combinations of candidates where
the chosen numbers sum to target.
# You may return the combinations in any order.

# The same number may be chosen from candidates an unlimited number of
times. Two combinations are unique if the
# frequency
#  of at least one of the chosen numbers is different.

# The test cases are generated such that the number of unique
combinations that sum up to target is less than 150 combinations for
the given input.



# Example 1:

# Input: candidates = [2,3,6,7], target = 7
# Output: [[2,2,3],[7]]
# Explanation:
# 2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used
multiple times.
# 7 is a candidate, and 7 = 7.
```

```python
# These are the only two combinations.
# Example 2:

# Input: candidates = [2,3,5], target = 8
# Output: [[2,2,2,2],[2,3,3],[3,5]]
# Example 3:

# Input: candidates = [2], target = 1
# Output: []

class Solution:
    def combinationSum(self, candidates: List[int], target: int) ->
List[List[int]]:

        output = []
        def backtrack(start, combination, total):

            if total > target:
                return
            if total == target:
                output.append(combination.copy())
                return

            for i in range(start,len(candidates)):

                combination.append(candidates[i])
                backtrack(i, combination, total + candidates[i])
                combination.pop()

        backtrack(0,[], 0)

        return output

# 22. Generate Parentheses
# Medium
# Topics
# Companies
# Given n pairs of parentheses, write a function to generate all
combinations of well-formed parentheses.



# Example 1:

# Input: n = 3
# Output: ["((()))","(()())","(())()","()(())","()()()"]
# Example 2:

# Input: n = 1
# Output: ["()"]
class Solution:
```

```
    def generateParenthesis(self, n: int) -> List[str]:
        output = []
        def backtrack(start, open, close):

            if len(start) == 2*n:
                output.append(start)
                return

            if open < n:
                backtrack(start+"(", open+1, close)
            if close < open:
                backtrack(start+")", open, close+1)

        backtrack("", 0,0)

        return output




# 79. Word Search
# Solved
# Medium
# Topics
# Companies
# Given an m x n grid of characters board and a string word, return
true if word exists in the grid.

# The word can be constructed from letters of sequentially adjacent
cells, where adjacent cells are horizontally or vertically
neighboring. The same letter cell may not be used more than once.



# Example 1:


# Input: board = [["A","B","C","E"],["S","F","C","S"],
["A","D","E","E"]], word = "ABCCED"
# Output: true
# Example 2:


# Input: board = [["A","B","C","E"],["S","F","C","S"],
["A","D","E","E"]], word = "SEE"
# Output: true
# Example 3:


# Input: board = [["A","B","C","E"],["S","F","C","S"],
```

```
["A","D","E","E"]], word = "ABCB"
# Output: false

class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:

        row = len(board)
        column = len(board[0])

        visited = set()
        def backtrack(r, c, index):

            if r not in range(row) or c not in range(column) or (r,c)
in visited:
                return False

            if board[r][c] != word[index]:
                return False

            if index == len(word) - 1:
                return True

            visited.add((r,c))

            found = backtrack(r-1, c, index+1) or backtrack(r+1, c,
index+1) or backtrack(r, c-1, index+1) or backtrack(r, c+1, index+1)

            visited.remove((r,c))

            return found



        for r in range(row):
            for c in range(column):
                if backtrack(r,c, 0):
                    return True

        return False



class Solution:
    #instead of using a set to track visited cells
    #modify the cells
    def exist(self, board: List[List[str]], word: str) -> bool:

        row = len(board)
        column = len(board[0])

        # visited = set()
```

```
        def backtrack(r, c, index):

            if r not in range(row) or c not in range(column):
                return False

            if board[r][c] != word[index]:
                return False

            if index == len(word) - 1:
                return True

            oldVal = board[r][c]
            board[r][c] = "#"

            found = backtrack(r-1, c, index+1) or backtrack(r+1, c,
index+1) or backtrack(r, c-1, index+1) or backtrack(r, c+1, index+1)

            board[r][c] = oldVal

            return found



        for r in range(row):
            for c in range(column):
                if backtrack(r,c, 0):
                    return True

        return False


# 427. Construct Quad Tree
# Solved
# Medium
# Topics
# Companies
# Given a n * n matrix grid of 0's and 1's only. We want to represent
grid with a Quad-Tree.

# Return the root of the Quad-Tree representing grid.

# A Quad-Tree is a tree data structure in which each internal node has
exactly four children. Besides, each node has two attributes:

# val: True if the node represents a grid of 1's or False if the node
represents a grid of 0's. Notice that you can assign the val
# to True or False when isLeaf is False, and both are accepted in the
answer.
# isLeaf: True if the node is a leaf node on the tree or False if the
node has four children.
```

```
# class Node {
#     public boolean val;
#     public boolean isLeaf;
#     public Node topLeft;
#     public Node topRight;
#     public Node bottomLeft;
#     public Node bottomRight;
# }
# We can construct a Quad-Tree from a two-dimensional area using the
following steps:

# If the current grid has the same value (i.e all 1's or all 0's) set
isLeaf True and set val to the value of the grid and set the four
children to Null and stop.
# If the current grid has different values, set isLeaf to False and
set val to any value and divide the current grid into four sub-grids
as shown in the photo.
# Recurse for each of the children with the proper sub-grid.

"""
# Definition for a QuadTree node.
class Node:
    def __init__(self, val, isLeaf, topLeft, topRight, bottomLeft,
bottomRight):
        self.val = val
        self.isLeaf = isLeaf
        self.topLeft = topLeft
        self.topRight = topRight
        self.bottomLeft = bottomLeft
        self.bottomRight = bottomRight
"""

class Solution:
    def construct(self, grid: List[List[int]]) -> 'Node':

        def dfs(n, r, c):

            isLeaf = True
            for i in range(n):
                for j in range(n):

                    if grid[r][c] != grid[r+i][c+j]:
                        isLeaf = False

            if isLeaf:
                return Node(grid[r][c], True, None, None, None, None)

            #divide further
            n = n //2
            topLeft = dfs(n, r,c)
```

```
            topRight = dfs(n, r, c+n)
            bottomLeft = dfs(n, r+n, c)
            bottomRight = dfs(n, r+n, c+n)

            root = Node(0, False, topLeft, topRight, bottomLeft,
bottomRight)

            return root

        n = len(grid)
        return dfs(n, 0,0)

# 148. Sort List
# Solved
# Medium
# Topics
# Companies
# Given the head of a linked list, return the list after sorting it in
ascending order.

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def split(self, head):

        slow = head
        fast = head.next

        while fast and fast.next:

            slow = slow.next
            fast = fast.next.next

        return slow
    def merge(self, left, right):
        dummy = ListNode()
        current = dummy

        while left and right:
            if left.val < right.val:
                current.next = left
                left = left.next
            else:
                current.next = right
                right = right.next
            current = current.next
```

```
        if left:
            current.next = left
        if right:
            current.next = right
        return dummy.next
    def sortList(self, head: Optional[ListNode]) ->
Optional[ListNode]:

        if not head or not head.next:
            return head

        left = head
        right = self.split(head)
        tmp = right.next
        right.next = None
        right = tmp

        left = self.sortList(left)
        right = self.sortList(right)

        return self.merge(left, right)


# 53. Maximum Subarray
# Solved
# Medium
# Topics
# Companies
# Given an integer array nums, find the
# subarray
#  with the largest sum, and return its sum.



# Example 1:

# Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
# Output: 6
# Explanation: The subarray [4,-1,2,1] has the largest sum 6.
# Example 2:

# Input: nums = [1]
# Output: 1
# Explanation: The subarray [1] has the largest sum 1.
# Example 3:

# Input: nums = [5,4,-1,7,8]
# Outpuet: 23
# Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

class Solution:
```

```python
    def maxSubArray(self, nums: List[int]) -> int:

        globalMax = nums[0]
        currMax = 0

        for i in range(len(nums)):

            currMax = max(currMax+ nums[i], nums[i])
            globalMax = max(globalMax, currMax)

        return globalMax

# 918. Maximum Sum Circular Subarray
# Solved
# Medium
# Topics
# Companies
# Hint
# Given a circular integer array nums of length n, return the maximum
possible sum of a non-empty subarray of nums.

# A circular array means the end of the array connects to the
beginning of the array. Formally,
# the next element of nums[i] is nums[(i + 1) % n] and the previous
element of nums[i] is nums[(i - 1 + n) % n].

# A subarray may only include each element of the fixed buffer nums at
most once. Formally, for a subarray nums[i], nums[i + 1], ...,
nums[j],
# there does not exist i <= k1, k2 <= j with k1 % n == k2 % n.



# Example 1:

# Input: nums = [1,-2,3,-2]
# Output: 3
# Explanation: Subarray [3] has maximum sum 3.
# Example 2:

# Input: nums = [5,-3,5]
# Output: 10
# Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10.
# Example 3:

# Input: nums = [-3,-2,-3]
# Output: -2
# Explanation: Subarray [-2] has maximum sum -2.

#intuition
#since the array is circular, the goal is
```

```python
#1--> find the maximum subarray
#2--> find the minimum subarray
#the answer will either be the maximum number  maximum subarray or
everything but minimum subarray
#the answer will definitely be the maximum subarray if everything in
the array is negative

class Solution:
    def maxSubarraySumCircular(self, nums: List[int]) -> int:

        globalMax = nums[0]
        globalMin = nums[0]

        currMin = 0
        currMax = 0

        total = 0
        for i in range(len(nums)):

            #calculate the maximum contigous subarray
            currMax = max(currMax + nums[i], nums[i])
            globalMax = max(globalMax, currMax)

            #calculate the minumum contigous subarray
            currMin = min(currMin + nums[i], nums[i])
            globalMin = min(currMin, globalMin)

            total += nums[i]

        return max(globalMax, total - globalMin) if globalMax > 0 else
globalMax

# 74. Search a 2D Matrix
# Solved
# Medium
# Topics
# Companies
# You are given an m x n integer matrix matrix with the following two
properties:

# Each row is sorted in non-decreasing order.
# The first integer of each row is greater than the last integer of
the previous row.
# Given an integer target, return true if target is in matrix or false
otherwise.

# You must write a solution in O(log(m * n)) time complexity.



# Example 1:
```

```python
# Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
# Output: true
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) ->
bool:

        rows = len(matrix)
        columns = len(matrix[0])

        LastIndexOfEachRow = columns - 1

        #use binary search to find the row to look for

        top = 0
        bottom = rows - 1

        while top <= bottom:

            mid = top + ((bottom - top) //2)

            #if target has greater elements than the mid area
            if target > matrix[mid][LastIndexOfEachRow]:
                top = mid + 1
            #otherwise target is above mid
            elif target < matrix[mid][0]:
                bottom = mid - 1
            #we are at a possible row that may contain the target
            else:
                break


        #check if a valid row was found
        #it can only be valid if top <= bottom

        if not (top <= bottom):
            return False
        #use binary search to find the target

        midRow = top + ((bottom - top) //2)
        left = 0
        right = LastIndexOfEachRow

        while left <= right:

            mid = left + ((right - left) //2)

            if target < matrix[midRow][mid]:
                right = mid - 1
            elif target > matrix[midRow][mid]:
```

```
                left = mid + 1
            else:
                return True

        return False

# 162. Find Peak Element
# Solved
# Medium
# Topics
# Companies
# A peak element is an element that is strictly greater than its
neighbors.

# Given a 0-indexed integer array nums, find a peak element, and
return its index. If the array contains multiple peaks, return the
index to any of the peaks.

# You may imagine that nums[-1] = nums[n] = -∞. In other words, an
element is always considered to be strictly greater than a neighbor
that is outside the array.

# You must write an algorithm that runs in O(log n) time.



# Example 1:

# Input: nums = [1,2,3,1]
# Output: 2
# Explanation: 3 is a peak element and your function should return the
index number 2.
# Example 2:

# Input: nums = [1,2,1,3,5,6,4]
# Output: 5
# Explanation: Your function can return either index number 1 where
the peak element is 2, or index number 5 where the peak element is 6.

class Solution:
    def findPeakElement(self, nums: List[int]) -> int:


        left = 0
        right = len(nums) - 1

        while left <= right:

            mid = left + ((right - left) // 2)

            if mid + 1 in range(0, len(nums)) and nums[mid] <
```

```
nums[mid+1]:
                left = mid + 1
            elif mid - 1 in range(0, len(nums)) and nums[mid] <
nums[mid-1]:
                right = mid - 1
            else:
                return mid

        return -1

# 33. Search in Rotated Sorted Array
# Solved
# Medium
# Topics
# Companies
# There is an integer array nums sorted in ascending order (with
distinct values).

# Prior to being passed to your function, nums is possibly rotated at
an unknown pivot index k
#  (1 <= k < nums.length) such that the resulting array is [nums[k],
nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-
indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index
3 and become [4,5,6,7,0,1,2].

# Given the array nums after the possible rotation and an integer
target, return the index of target if it is in nums, or -1 if it is
not in nums.

# You must write an algorithm with O(log n) runtime complexity.



# Example 1:

# Input: nums = [4,5,6,7,0,1,2], target = 0
# Output: 4
# Example 2:

# Input: nums = [4,5,6,7,0,1,2], target = 3
# Output: -1
# Example 3:

# Input: nums = [1], target = 0
# Output: -1

class Solution:
    def search(self, nums: List[int], target: int) -> int:

        left = 0
        right = len(nums) - 1
```

```
        while left <= right:

            # mid = left + ((right - left)//2)
            mid = (left + right) //2
            if target == nums[mid]:
                return mid
            if nums[left] <= nums[mid]:
                if target > nums[mid] or target < nums[left]:
                    left = mid + 1
                else:
                    right = mid - 1
            else:
                if target < nums[mid] or target > nums[right]:
                    right = mid -1
                else:
                    left = mid + 1

        return -1

# 34. Find First and Last Position of Element in Sorted Array
# Solved
# Medium
# Topics
# Companies
# Given an array of integers nums sorted in non-decreasing order, find
the starting and ending position of a given target value.

# If target is not found in the array, return [-1, -1].

# You must write an algorithm with O(log n) runtime complexity.



# Example 1:

# Input: nums = [5,7,7,8,8,10], target = 8
# Output: [3,4]
# Example 2:

# Input: nums = [5,7,7,8,8,10], target = 6
# Output: [-1,-1]
# Example 3:

# Input: nums = [], target = 0
# Output: [-1,-1]

class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:

        def binSearchHelper(arr, target, leftBias):
```

```
            left = 0
            right = len(arr) - 1

            index = -1
            while left <= right:

                mid = (left + right) //2

                if nums[mid] < target:
                    left = mid + 1
                elif nums[mid] > target:
                    right = mid -1
                else:
                    index = mid

                    if leftBias:
                        right = mid - 1
                    else:
                        left = mid + 1
            return index

        left = binSearchHelper(nums, target, True)
        right = binSearchHelper(nums, target, False)

        return [left, right]




# Code
# Testcase
# Testcase
# Test Result
# 153. Find Minimum in Rotated Sorted Array
# Solved
# Medium
# Topics
# Companies
# Hint
# Suppose an array of length n sorted in ascending order is rotated
between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7]
might become:

# [4,5,6,7,0,1,2] if it was rotated 4 times.
# [0,1,2,4,5,6,7] if it was rotated 7 times.
# Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time
results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

# Given the sorted rotated array nums of unique elements, return the
```

```
minimum element of this array.

# You must write an algorithm that runs in O(log n) time.



# Example 1:

# Input: nums = [3,4,5,1,2]
# Output: 1
# Explanation: The original array was [1,2,3,4,5] rotated 3 times.
# Example 2:

# Input: nums = [4,5,6,7,0,1,2]
# Output: 0
# Explanation: The original array was [0,1,2,4,5,6,7] and it was
rotated 4 times.
# Example 3:

# Input: nums = [11,13,15,17]
# Output: 11
# Explanation: The original array was [11,13,15,17] and it was rotated
4 times

class Solution:
    def findMin(self, nums: List[int]) -> int:

        left = 0
        right = len(nums) -1

        while left < right:

            mid = (left + right) //2

            if nums[mid] > nums[right]:
                left = mid + 1
            else:
                right = mid

        return nums[left]

# 215. Kth Largest Element in an Array
# Solved
# Medium
# Topics
# Companies
# Given an integer array nums and an integer k, return the kth largest
element in the array.

# Note that it is the kth largest element in the sorted order, not the
kth distinct element.
```

```python
# Can you solve it without sorting?



# Example 1:

# Input: nums = [3,2,1,5,6,4], k = 2
# Output: 5
# Example 2:

# Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
# Output: 4



import heapq

class Solution:
    #we use quick select
    #We define a partition
    #we recursively sort one side given a certain criteria
    #Average case : O(2N) -> O(N)
    #worst case: O(N^2)
    def findKthLargest(self, nums: List[int], k: int) -> int:
        k = len(nums) - k
        def quickselect(left, right):

            pivot = nums[right]
            p = left

            for i in range(left, right):
                if nums[i] <= pivot:
                    #swap p and nums[i]
                    nums[i], nums[p] = nums[p], nums[i]
                    p+=1
            nums[p], nums[right] = nums[right], nums[p]

            #now check which side to recursively sort
            #if we are at the right spot
            #we stop

            if p < k:
                return quickselect(p+1, right)
            elif p > k:
                return quickselect(left, p-1)
            else:
                return nums[p]

        return quickselect(0, len(nums)-1)
```

```python
class Solution:
    #use minheap
    #The idea is to keep k elements in the min heap
    #The initial pop will always be the kth largest element

    #Time: O(N log K)
    #Space: O(K)
    def findKthLargest(self, nums: List[int], k: int) -> int:

        min_heap = []
        for i in range(k):
            heapq.heappush(min_heap, nums[i])

        #now continue the push and popping in the heap after pushing
the first kth element

        for i in range(k, len(nums)):
            heapq.heappushpop(min_heap, nums[i])

        return heapq.heappop(min_heap)



class Solution:
    #use max heap to track of the array
    #then pop up to k times
    #The kth item is the kth largest
    #Time : O(N + KlogN)
    def findKthLargest(self, nums: List[int], k: int) -> int:
        #python does not have max heap
        #negative to the elements in the nums to simulate max heap

        #negate the items in nums
        for i in range(len(nums)):
            nums[i] = -nums[i]

        #convert nums to heap
        heapq.heapify(nums)

        for _ in range(k-1):
            heapq.heappop(nums)

        kth = heapq.heappop(nums)
        #renegate the kth element
        return  - kth

class Solution:
    #sort the array
    #key the kth array from the back
    #Time : O(NlogN)
```

```
    def findKthLargest(self, nums: List[int], k: int) -> int:

        nums.sort()

        return nums[len(nums) - k]



# Code
# Testcase
# Test Result
# Test Result
# 373. Find K Pairs with Smallest Sums
# Medium
# Topics
# Companies
# You are given two integer arrays nums1 and nums2 sorted in non-
decreasing order and an integer k.

# Define a pair (u, v) which consists of one element from the first
array and one element from the second array.

# Return the k pairs (u1, v1), (u2, v2), ..., (uk, vk) with the
smallest sums.



# Example 1:

# Input: nums1 = [1,7,11], nums2 = [2,4,6], k = 3
# Output: [[1,2],[1,4],[1,6]]
# Explanation: The first 3 pairs are returned from the sequence:
[1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
# Example 2:

# Input: nums1 = [1,1,2], nums2 = [1,2,3], k = 2
# Output: [[1,1],[1,1]]
# Explanation: The first 2 pairs are returned from the sequence:
[1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]


import heapq
class Solution:
    def kSmallestPairs(self, nums1: List[int], nums2: List[int], k:
int) -> List[List[int]]:

        min_heap = []
        visited = set()

        result = []
```

```
        visited.add((0,0))
        heapq.heappush(min_heap, (nums1[0]+ nums2[0], 0,0))

        while k and min_heap:
            key, i, j = heapq.heappop(min_heap)

            if i+1 < len(nums1) and (i+1, j) not in visited:
                heapq.heappush(min_heap, (nums1[i+1]+ nums2[j],
i+1,j))
                visited.add((i+1,j))

            if j+1 < len(nums2) and (i, j+1) not in visited:
                heapq.heappush(min_heap, (nums1[i]+ nums2[j+1],
i,j+1))
                visited.add((i,j+1))

            result.append([nums1[i], nums2[j]])

            k-=1

        return result


# Code
# Testcase
# Testcase
# Test Result
# 172. Factorial Trailing Zeroes
# Medium
# Topics
# Companies
# Given an integer n, return the number of trailing zeroes in n!.

# Note that n! = n * (n - 1) * (n - 2) * ... * 3 * 2 * 1.



# Example 1:

# Input: n = 3
# Output: 0
# Explanation: 3! = 6, no trailing zero.
# Example 2:

# Input: n = 5
# Output: 1
# Explanation: 5! = 120, one trailing zero.
# Example 3:

# Input: n = 0
```

```
# Output: 0

class Solution:
    #     Counting Factors of 5:

    # The number of trailing zeros in n! (n factorial)
    # is determined by the number of factors of 5 in the product 1 * 2
* 3 * ... * n.
    # For example, in 5! = 5 * 4 * 3 * 2 * 1, there is one factor of
5, so 5! has one trailing zero.
    # In 10! = 10 * 9 * 8 * ... * 1, there are two
    # factors of 5 (one from 5 and one from 10), so 10! has two
trailing zeros.
    #Time : O(Log base 5 N)
    def trailingZeroes(self, n: int) -> int:

        count = 0
        while n > 0:
            n //= 5
            count += n

        return count


# Code
# Testcase
# Testcase
# Test Result
# 50. Pow(x, n)
# Solved
# Medium
# Topics
# Companies
# Implement pow(x, n), which calculates x raised to the power n (i.e.,
xn).



# Example 1:

# Input: x = 2.00000, n = 10
# Output: 1024.00000
# Example 2:

# Input: x = 2.10000, n = 3
# Output: 9.26100
# Example 3:

# Input: x = 2.00000, n = -2
# Output: 0.25000
# Explanation: 2-2 = 1/22 = 1/4 = 0.25
```

```python
class Solution:
    #The idea calcuate half work and multiply it by itself
    #for example: 2^10 = 2*2*2*2*2*2*2*2*2*2
    #which is also equivalent to 2^5 * 2^5
    #if we know 2^5 then no need to recalcaute it
    #we can just multiply it and return it
    #Time : O(logN)
    def myPow(self, x: float, n: int) -> float:

        def helper(x, n):
            if x == 0:
                return 0
            if n == 0:
                return 1

            res = helper(x, n//2)
            res = res * res
            #if the number n is odd
            if n % 2 ==1:
                return x * res
            return res

        if n >= 0:
            return helper(x,n)
        return 1/ helper(x, abs(n))

# 137. Single Number II
# Solved
# Medium
# Topics
# Companies
# Given an integer array nums where every element appears three times
except for one, which appears exactly once. Find the single element
and return it.

# You must implement a solution with a linear runtime complexity and
use only constant extra space.



# Example 1:

# Input: nums = [2,2,3,2]
# Output: 3
# Example 2:

# Input: nums = [0,1,0,1,0,1,99]
# Output: 99

class Solution:
```

```
    # ones is use to count the number of single numbers
    #Twos is to count the number of twos number
    #If number appears three times both ones and twos turns to zero
    #Time : O(N)
    def singleNumber(self, nums: List[int]) -> int:

        ones = 0
        twos = 0

        for num in nums:

            ones = (ones ^ num) & ~twos
            twos = (twos ^ num) & ~ones


        return ones

# 201. Bitwise AND of Numbers Range
# Medium
# Topics
# Companies
# Given two integers left and right that represent the range [left,
right], return the bitwise AND of all numbers in this range,
inclusive.



# Example 1:

# Input: left = 5, right = 7
# Output: 4
# Example 2:

# Input: left = 0, right = 0
# Output: 0
# Example 3:

# Input: left = 1, right = 2147483647
# Output: 0

class Solution:
    #the purpose of this solution is to iteratively and operator every
number between left and right
    #for example: left = 5 and right = 7 then output is expected to be
5 & 6 & 7
    #large numbers might take long
    #this solution is faster
    #in the case of 5 and 7
    #5: 0011
    #6: 0110
    #7: 0111
```

```python
    #the idea is check if the rightmost of all the numbers  are the same if not we shift to the right
    #if they are same, then we can shift back by the amount we shifted to get the numbers to be the same
    def rangeBitwiseAnd(self, left: int, right: int) -> int:
        count = 0
        while left != right:
            left = left >> 1
            right = right >> 1
            count +=1
        return left << count

# 198. House Robber
# Solved
# Medium
# Topics
# Companies
# You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed,
# the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police
# if two adjacent houses were broken into on the same night.

# Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.



# Example 1:

# Input: nums = [1,2,3,1]
# Output: 4
# Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
# Total amount you can rob = 1 + 3 = 4.
# Example 2:

# Input: nums = [2,7,9,3,1]
# Output: 12
# Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
# Total amount you can rob = 2 + 9 + 1 = 12.

class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) ==1:
            return nums[0]
        dp = [1] * len(nums)
```

```
        dp[0] = nums[0]
        dp[1] = max(nums[0], nums[1])
        for i in range(2, len(nums)):

            dp[i] = max(nums[i] + dp[i-2], dp[i-1])

        return dp[len(nums)-1]

class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) ==1:
            return nums[0]
        prev1 = nums[0]
        prev2 = max(nums[0],nums[1])

        # [prev1, prev2, n, n+1]
        for i in range(2, len(nums)):
            temp = max(nums[i] + prev1, prev2)

            prev1 = prev2
            prev2 = temp

        return prev2




# 139. Word Break
# Medium
# Topics
# Companies
# Given a string s and a dictionary of strings wordDict, return true
if s can be segmented into a space-separated sequence of one or more
dictionary words.

# Note that the same word in the dictionary may be reused multiple
times in the segmentation.



# Example 1:

# Input: s = "leetcode", wordDict = ["leet","code"]
# Output: true
# Explanation: Return true because "leetcode" can be segmented as
"leet code".
# Example 2:

# Input: s = "applepenapple", wordDict = ["apple","pen"]
# Output: true
```

```python
# Explanation: Return true because "applepenapple" can be segmented as
"apple pen apple".
# Note that you are allowed to reuse a dictionary word.
# Example 3:

# Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]
# Output: false

class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:

        dp = [False] * (len(s) + 1)
        dp[0] = True
        for i in range(len(s)+1):
            for j in range(i):

                if dp[j] == True and s[j:i] in wordDict:
                    dp[i] = True
                    break

        return dp[len(s)]


# Code


# Testcase
# Testcase
# Test Result
# 322. Coin Change
# Solved
# Medium
# Topics
# Companies
# You are given an integer array coins representing coins of different
denominations and an integer amount representing a total amount of
money.

# Return the fewest number of coins that you need to make up that
amount. If that amount of money cannot be made up by any combination
of the coins, return -1.

# You may assume that you have an infinite number of each kind of
coin.



# Example 1:

# Input: coins = [1,2,5], amount = 11
# Output: 3
```

```python
# Explanation: 11 = 5 + 5 + 1
# Example 2:

# Input: coins = [2], amount = 3
# Output: -1
# Example 3:

# Input: coins = [1], amount = 0
# Output: 0

class Solution:
    #top down
    def coinChange(self, coins: List[int], amount: int) -> int:
        memo = {}

        def dp(remaining):
            if remaining == 0:
                return 0

            if remaining < 0:
                return float('inf')

            if remaining in memo:
                return memo[remaining]

            min_coins = float('inf')
            for i in range(len(coins)):
                num_coins = dp(remaining - coins[i])

                if num_coins != float('inf'):
                    min_coins = min(min_coins, num_coins +1)

            memo[remaining] = min_coins
            return min_coins

        return dp(amount) if dp(amount) != float('inf') else -1


    class Solution:
    #top down
    def coinChange(self, coins: List[int], amount: int) -> int:
        dp = [float('inf')] * (amount + 1)
        dp[0] = 0

        for i in range(1, amount +1):

            for coin in coins:
                if i - coin >= 0:
                    dp[i] = min(dp[i], 1+ dp[i - coin])

        return dp[amount] if dp[amount] != float('inf') else -1
```

```python
# 300. Longest Increasing Subsequence
# Medium
# Topics
# Companies
# Given an integer array nums, return the length of the longest
strictly increasing
# subsequence
# .



# Example 1:

# Input: nums = [10,9,2,5,3,7,101,18]
# Output: 4
# Explanation: The longest increasing subsequence is [2,3,7,101],
therefore the length is 4.
# Example 2:

# Input: nums = [0,1,0,3,2,3]
# Output: 4
# Example 3:

# Input: nums = [7,7,7,7,7,7,7]
# Output: 1

class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:

        if not nums:
            return 0

        dp = [1] * len(nums)
        for i in range(1,len(nums)):
            for j in range(i):
                if nums[j] < nums[i]:
                    dp[i] = max(dp[i], 1 + dp[j])

        return max(dp)


class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:

        if not nums:
            return 0
        max_length = 1
        dp = [1] * len(nums)
        for i in range(1,len(nums)):
            for j in range(i):
                if nums[j] < nums[i]:
```

```
                    dp[i] = max(dp[i], 1 + dp[j])
                    max_length = max(max_length, dp[i])
        return max_length



# Code
# Testcase
# Testcase
# Test Result
# 120. Triangle
# Solved
# Medium
# Topics
# Companies
# Given a triangle array, return the minimum path sum from top to
bottom.

# For each step, you may move to an adjacent number of the row below.
More formally, if you are on index i on the current row, you may move
to either index i or index i + 1 on the next row.



# Example 1:

# Input: triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]
# Output: 11
# Explanation: The triangle looks like:
#    2
#   3 4
#  6 5 7
# 4 1 8 3
# The minimum path sum from top to bottom is 2 + 3 + 5 + 1 = 11
(underlined above).
# Example 2:

# Input: triangle = [[-10]]
# Output: -10

class Solution:
    def minimumTotal(self, triangle: List[List[int]]) -> int:

        dp = [0] * (len(triangle) + 1)
        rows = len(triangle)

        for row in triangle[::-1]:
            for i, num in enumerate(row):
                dp[i] = num + min(dp[i], dp[i+1])
```

```
        return dp[0]



# Code
# Testcase
# Testcase
# Test Result
# 64. Minimum Path Sum
# Solved
# Medium
# Topics
# Companies
# Given a m x n grid filled with non-negative numbers, find a path
from top left to bottom right, which minimizes the sum of all numbers
along its path.

# Note: You can only move either down or right at any point in time.



# Example 1:


# Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
# Output: 7
# Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.
# Example 2:

# Input: grid = [[1,2,3],[4,5,6]]
# Output: 12

class Solution:
        #2d array for calculating bottom up
        # [[1,3,1],
        # [1,5,1],
        # [4,2,1]]

        #dp
        # [inf, inf, inf, inf],
        # [inf, inf, inf, inf],
        # [inf, inf, inf, inf]
        # [inf, inf, 0, inf]

        #we calculate each cell minimum path to bottom right
(destination)
    def minPathSum(self, grid: List[List[int]]) -> int:

        rows = len(grid)
        cols = len(grid[0])
```

```
        res = [[float('inf')] * (cols + 1) for r in range(rows + 1)]
        res[rows - 1][cols] = 0

        for r in range(rows -1, -1, -1):
            for c in range(cols-1, -1, -1):
                res[r][c] = grid[r][c] + min(res[r+1][c], res[r][c+1])
        return res[0][0]



# 63. Unique Paths II
# Solved
# Medium
# Topics
# Companies
# Hint
# You are given an m x n integer array grid. There is a robot
initially located at the top-left corner (i.e., grid[0][0]).
# The robot tries to move to the bottom-right corner (i.e., grid[m -
1][n - 1]). The robot can only move either down or right at any point
in time.

# An obstacle and space are marked as 1 or 0 respectively in grid. A
path that the robot takes cannot include any square that is an
obstacle.

# Return the number of possible unique paths that the robot can take
to reach the bottom-right corner.

# The testcases are generated so that the answer will be less than or
equal to 2 * 109.



# Example 1:


# Input: obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]
# Output: 2
# Explanation: There is one obstacle in the middle of the 3x3 grid
above.
# There are two ways to reach the bottom-right corner:
# 1. Right -> Right -> Down -> Down
# 2. Down -> Down -> Right -> Right
# Example 2:


# Input: obstacleGrid = [[0,1],[0,0]]
# Output: 1
```

```python
class Solution:
    #recursive implementation
    def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]])
-> int:

        M, N = len(obstacleGrid), len(obstacleGrid[0])

        #memoization to find recompuation top down
        dp = {(M-1, N-1):1}

        def dfs(r,c):

            if r == M or c == N or obstacleGrid[r][c] ==1:
                return 0

            if (r,c) in dp:
                return dp[(r,c)]

            dp[(r,c)] = dfs(r+1, c) + dfs(r, c+1)

            return dp[(r,c)]

        return dfs(0,0)


class Solution:
    #bottom up implementation
    def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]])
-> int:

        M, N = len(obstacleGrid), len(obstacleGrid[0])
        dp = [0] * N
        dp[N-1] = 1

        for r in reversed(range(M)):
            for c in reversed(range(N)):
                if obstacleGrid[r][c] == 1:
                    dp[c] = 0
                elif c + 1 < N:
                    dp[c] = dp[c] + dp[c+1]

        return dp[0]

# 5. Longest Palindromic Substring
# Medium
# Topics
# Companies
# Hint
# Given a string s, return the longest
# palindromic
```

```
# substring
#  in s.



# Example 1:

# Input: s = "babad"
# Output: "bab"
# Explanation: "aba" is also a valid answer.
# Example 2:

# Input: s = "cbbd"
# Output: "bb"

class Solution:
    def longestPalindrome(self, s: str) -> str:

        res = ""
        resLen = 0

        for i in range(len(s)):
            #do odd length
            l, r = i, i

            while l in range(len(s)) and r in range(len(s)) and s[l]
== s[r]:
                if (r-l+1) > resLen:
                    res = s[l:r+1]
                    resLen = r-l+1

                l-=1
                r+=1
            #do even length
            l, r = i, i+1
            while l in range(len(s)) and r in range(len(s)) and s[l]
== s[r]:
                if (r-l+1) > resLen:
                    res = s[l:r+1]
                    resLen = r-l+1

                l-=1
                r+=1

        return res

# 72. Edit Distance
# Solved
# Medium
# Topics
# Companies
```

```python
# Given two strings word1 and word2, return the minimum number of
operations required to convert word1 to word2.

# You have the following three operations permitted on a word:

# Insert a character
# Delete a character
# Replace a character


# Example 1:

# Input: word1 = "horse", word2 = "ros"
# Output: 3
# Explanation:
# horse -> rorse (replace 'h' with 'r')
# rorse -> rose (remove 'r')
# rose -> ros (remove 'e')
# Example 2:

# Input: word1 = "intention", word2 = "execution"
# Output: 5
# Explanation:
# intention -> inention (remove 't')
# inention -> enention (replace 'i' with 'e')
# enention -> exention (replace 'n' with 'x')
# exention -> exection (replace 'n' with 'c')
# exection -> execution (insert 'u')
class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        dp = [[float('inf')] * (len(word2)+1) for i in range(
len(word1)+1 )]

        for j in range(len(word2)+1):
            dp[len(word1)][j] = len(word2)-j

        for i in range(len(word1)+1):
            dp[i][len(word2)] = len(word1) - i


        for i in range(len(word1)-1, -1, -1):
            for j in range(len(word2)-1, -1,-1):
                if word1[i] == word2[j]:
                    dp[i][j] = dp[i+1][j+1]
                else:
                    dp[i][j] = 1 + min(dp[i+1][j], dp[i][j+1], dp[i+1]
[j+1])


        return dp[0][0]
```

```python
# 7. Reverse Integer
# Solved
# Medium
# Topics
# Companies
# Given a signed 32-bit integer x, return x with its digits reversed.
If reversing x causes the value to go outside the signed 32-bit
integer range [-231, 231 - 1], then return 0.

# Assume the environment does not allow you to store 64-bit integers
(signed or unsigned).



# Example 1:

# Input: x = 123
# Output: 321
# Example 2:

# Input: x = -123
# Output: -321
# Example 3:

# Input: x = 120
# Output: 21
class Solution:
    def reverse(self, x: int) -> int:
        reversed = 0
        isNeg = False
        if x < 0:
            x = -x
            isNeg = True
        while x > 0:

            lastNum = x % 10
            x = x //10

            reversed = reversed * 10 + lastNum

            if reversed not in range( 2**31 - 1):
                return 0

        return reversed if not isNeg else -reversed

# 97. Interleaving String
# Medium
# Topics
# Companies
# Given strings s1, s2, and s3, find whether s3 is formed by an
interleaving of s1 and s2.
```

```python
# An interleaving of two strings s and t is a configuration where s
and t are divided into n and m
# substrings
#  respectively, such that:

# s = s1 + s2 + ... + sn
# t = t1 + t2 + ... + tm
# |n - m| <= 1
# The interleaving is s1 + t1 + s2 + t2 + s3 + t3 + ... or t1 + s1 +
t2 + s2 + t3 + s3 + ...
# Note: a + b is the concatenation of strings a and b.



# Example 1:


# Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"
# Output: true
# Explanation: One way to obtain s3 is:
# Split s1 into s1 = "aa" + "bc" + "c", and s2 into s2 = "dbbc" + "a".
# Interleaving the two splits, we get "aa" + "dbbc" + "bc" + "a" + "c"
= "aadbbcbcac".
# Since s3 can be obtained by interleaving s1 and s2, we return true.
# Example 2:

# Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"
# Output: false
# Explanation: Notice how it is impossible to interleave s2 with any
other string to obtain s3.
# Example 3:

# Input: s1 = "", s2 = "", s3 = ""
# Output: true

class Solution:
    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:

        if len(s1) + len(s2) != len(s3):
            return False

        table = {}

        def dfs(i,j):

            if i == len(s1) and j == len(s2):
                return True

            if (i,j) in table:
                return table[(i,j)]
```

```
            k = i + j

            if i in range(len(s1)) and s1[i] == s3[k] and dfs(i+1, j)
== True:
                table[(i,j)] = True
                return True
            if j in range(len(s2)) and s2[j] == s3[k] and dfs(i, j+1)
== True:
                table[(i,j)] = True
                return True

            table[(i,j)] = False

            return False

        return dfs(0,0)

# 221. Maximal Square
# Medium
# Topics
# Companies
# Given an m x n binary matrix filled with 0's and 1's, find the
largest square containing only 1's and return its area.



# Example 1:


# Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],
["1","1","1","1","1"],["1","0","0","1","0"]]
# Output: 4
# Example 2:


# Input: matrix = [["0","1"],["1","0"]]
# Output: 1
# Example 3:

# Input: matrix = [["0"]]
# Output: 0
class Solution:
    def maximalSquare(self, matrix: List[List[str]]) -> int:
        if not matrix:
            return 0

        rows = len(matrix)
        cols = len(matrix[0])

        memo = {}
```

```
        def dfs(r, c):
            # Base case: if out of bounds
            if r >= rows or c >= cols:
                return 0

            if (r, c) in memo:
                return memo[(r, c)]

            right = dfs(r, c + 1)
            bottom = dfs(r + 1, c)
            diagonal = dfs(r + 1, c + 1)

            if matrix[r][c] == "1":
                memo[(r, c)] = 1 + min(right, bottom, diagonal)
            else:
                memo[(r, c)] = 0

            return memo[(r, c)]

        # Initialize maximum square length
        max_square_len = 0

        # Iterate over each cell in the matrix
        for r in range(rows):
            for c in range(cols):
                max_square_len = max(max_square_len, dfs(r, c))

        return max_square_len ** 2  # Return the area of the largest
square



# Code
# Testcase
# Testcase
# Test Result
# 75. Sort Colors
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an array nums with n objects colored red, white, or blue, sort
them in-place so that objects of the same color are adjacent, with the
colors in the order red, white, and blue.

# We will use the integers 0, 1, and 2 to represent the color red,
white, and blue, respectively.

# You must solve this problem without using the library's sort
```

```
function.



# Example 1:

# Input: nums = [2,0,2,1,1,0]
# Output: [0,0,1,1,2,2]
# Example 2:

# Input: nums = [2,0,1]
# Output: [0,1,2]

class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        left = 0
        right = len(nums)-1
        def partition(left, right):
            pivot = nums[right]
            p = left
            for i in range(left, right):
                if nums[i] <= pivot:
                    nums[i], nums[p] = nums[p], nums[i]
                    p+=1

            nums[p], nums[right] = nums[right], nums[p]

            return p

        def quicksort(left, right):
            if left < right:
                p = partition(left, right)
                quicksort(left, p - 1)
                quicksort(p + 1, right)

        quicksort(left, right)



class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        zeros, ones, n = 0, 0, len(nums)
        for num in nums:
            if num == 0:
```

```
                zeros += 1
            elif num == 1:
                ones += 1

        for i in range(0, zeros):
            nums[i] = 0

        for i in range(zeros, zeros + ones):
            nums[i] = 1

        for i in range(zeros + ones, n):
            nums[i] = 2

# 78. Subsets
# Solved
# Medium
# Topics
# Companies
# Given an integer array nums of unique elements, return all possible
# subsets
#  (the power set).

# The solution set must not contain duplicate subsets. Return the
solution in any order.



# Example 1:

# Input: nums = [1,2,3]
# Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
# Example 2:

# Input: nums = [0]
# Output: [[],[0]]

class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:

        result = []

        def dfs(start, arr):
            if start > len(nums):
                return
            result.append(arr.copy())

            for i in range(start, len(nums)):
                arr.append(nums[i])

                dfs(i+1, arr)
                arr.pop()
```

```
        dfs(0, [])

        return result


# Code


# Testcase
# Testcase
# Test Result
# 135. Candy
# Hard
# Topics
# Companies
# There are n children standing in a line. Each child is assigned a
rating value given in the integer array ratings.

# You are giving candies to these children subjected to the following
requirements:

# Each child must have at least one candy.
# Children with a higher rating get more candies than their neighbors.
# Return the minimum number of candies you need to have to distribute
the candies to the children.



# Example 1:

# Input: ratings = [1,0,2]
# Output: 5
# Explanation: You can allocate to the first, second and third child
with 2, 1, 2 candies respectively.
# Example 2:

# Input: ratings = [1,2,2]
# Output: 4
# Explanation: You can allocate to the first, second and third child
with 1, 2, 1 candies respectively.
# The third child gets 1 candy because it satisfies the above two
conditions.
class Solution:
    def candy(self, ratings: List[int]) -> int:

        arr = [1] * len(ratings)

        for i in range(1, len(ratings)):
            if ratings[i-1] < ratings[i]:
                arr[i] = arr[i-1] + 1
```

```
        for i in range(len(ratings)-2, -1, -1):

            if ratings[i] > ratings[i+1]:
                arr[i] = max(arr[i], arr[i+1] + 1)

        return sum(arr)


# Code
# Testcase
# Testcase
# Test Result
# 213. House Robber II
# Solved
# Medium
# Topics
# Companies
# Hint
# You are a professional robber planning to rob houses along a street.
Each house has a certain amount of money stashed.
# All houses at this place are arranged in a circle. That means the
first house is the neighbor of the last one. Meanwhile, adjacent
houses have a security system connected,
# and it will automatically contact the police if two adjacent houses
were broken into on the same night.

# Given an integer array nums representing the amount of money of each
house, return the maximum amount of money you can rob tonight without
alerting the police.



# Example 1:

# Input: nums = [2,3,2]
# Output: 3
# Explanation: You cannot rob house 1 (money = 2) and then rob house 3
(money = 2), because they are adjacent houses.
# Example 2:

# Input: nums = [1,2,3,1]
# Output: 4
# Explanation: Rob house 1 (money = 1) and then rob house 3 (money =
3).
# Total amount you can rob = 1 + 3 = 4.
# Example 3:

# Input: nums = [1,2,3]
# Output: 3
```

```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) == 1:
            return nums[0]

        if len(nums) == 2:
            return max(nums[0], nums[1])

        # Helper function to perform house robber on a linear list of
houses
        def rob_linear(nums):
            n = len(nums)
            if n == 0:
                return 0
            if n == 1:
                return nums[0]
            dp = [0] * n
            dp[0] = nums[0]
            dp[1] = max(nums[0], nums[1])
            for i in range(2, n):
                dp[i] = max(nums[i] + dp[i - 2], dp[i - 1])
            return dp[-1]

        # Rob excluding the first house or excluding the last house
        return max(rob_linear(nums[:-1]), rob_linear(nums[1:]))

#leetcode 75

# 2095. Delete the Middle Node of a Linked List
# Solved
# Medium
# Topics
# Companies
# Hint
# You are given the head of a linked list. Delete the middle node, and
return the head of the modified linked list.

# The middle node of a linked list of size n is the ⌊n / 2⌋th node
from the start using 0-based indexing, where ⌊x⌋ denotes the largest
integer less than or equal to x.

# For n = 1, 2, 3, 4, and 5, the middle nodes are 0, 1, 1, 2, and 2,
respectively.
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def deleteMiddle(self, head: Optional[ListNode]) ->
```

```
Optional[ListNode]:

        if not head or not head.next:
            return None

        slow = head
        fast = head

        dummy = ListNode(0, None)

        while slow and fast and fast.next:

            dummy.next = slow
            dummy = dummy.next

            slow = slow.next
            fast = fast.next.next

        dummy.next = dummy.next.next

        return head
```

```
# 647. Palindromic Substrings
# Solved
# Medium
# Topics
# Companies
# Hint
# Given a string s, return the number of palindromic substrings in it.

# A string is a palindrome when it reads the same backward as forward.

# A substring is a contiguous sequence of characters within the
string.



# Example 1:

# Input: s = "abc"
# Output: 3
# Explanation: Three palindromic strings: "a", "b", "c".
# Example 2:

# Input: s = "aaa"
# Output: 6
# Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa",
"aaa".
class Solution:
    def countSubstrings(self, s: str) -> int:
        res = 0
```

```
        for i in range(len(s)):
            #ven length
            l,r = i,i+1
            while l >= 0 and r < len(s) and s[l] == s[r]:
                res+=1
                l-=1
                r+=1
            #odd length
            l,r = i,i
            while l>= 0 and r < len(s) and s[l] == s[r]:
                res+=1
                l-=1
                r+=1

# 347. Top K Frequent Elements
# Solved
# Medium
# Topics
# Companies
# Given an integer array nums and an integer k, return the k most
frequent elements. You may return the answer in any order.



# Example 1:

# Input: nums = [1,1,1,2,2,3], k = 2
# Output: [1,2]
# Example 2:

# Input: nums = [1], k = 1
# Output: [1]

class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:

        freq_table = {}

        for num in nums:

            if num in freq_table:
                freq_table[num] +=1
            else:
                freq_table[num] = 1


        freq_table = sorted(freq_table.items(), key=lambda i: i[1],
reverse=True)

        res = []
```

```
        for num, count in freq_table:

            res.append(num)

            if len(res) == k:
                return res

        return res

import heapq
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:

        freq_table = {}
        for num in nums:
            if num in freq_table:
                freq_table[num] +=1
            else:
                freq_table[num] = 1

        heap = []
        for num, count in freq_table.items():
            heapq.heappush(heap, (count, num))

            if len(heap) > k:
                heapq.heappop(heap)

        res = []
        for count, num in heap:
            res.append(num)

        return res


# String Encode and Decode
# Design an algorithm to encode a list of strings to a single string.
The encoded string is then decoded back to the original list of
strings.

# Please implement encode and decode

# Example 1:

# Input: ["neet","code","love","you"]

# Output:["neet","code","love","you"]
# Example 2:

# Input: ["we","say",":","yes"]
```

```python
# Output: ["we","say",":","yes"]
from typing import List
class Solution:

    def encode(self, strs: List[str]) -> str:

        enc_s = ""

        for s in strs:
            enc_s += str(len(s)) +"#"+s

        return enc_s


    def decode(self, s: str) -> List[str]:

        res = []
        i = 0
        while i < len(s):

            j = i
            while s[j] != "#":
                j+=1

            length = int(s[i:j])
            word = s[j+1: j+length + 1]
            res.append(word)
            i = j+length + 1

        return res

word_list = ["i", "love", "you"]

sol = Solution()
con = sol.encode(word_list)
print(con)

rec = sol.decode(con)
print(rec)

# 1209. Remove All Adjacent Duplicates in String II
# Solved
# Medium
# Topics
# Companies
# Hint
# You are given a string s and an integer k, a k duplicate removal
consists of choosing k adjacent and equal letters
# from s and removing them, causing the left and the right side of the
deleted substring to concatenate together.
```

```python
# We repeatedly make k duplicate removals on s until we no longer can.

# Return the final string after all such duplicate removals have been
made. It is guaranteed that the answer is unique.



# Example 1:

# Input: s = "abcd", k = 2
# Output: "abcd"
# Explanation: There's nothing to delete.
# Example 2:

# Input: s = "deeedbbcccbdaa", k = 3
# Output: "aa"
# Explanation:
# First delete "eee" and "ccc", get "ddbbbdaa"
# Then delete "bbb", get "dddaa"
# Finally delete "ddd", get "aa"
# Example 3:

# Input: s = "pbbcggttciiippooaais", k = 2
# Output: "ps"

class Solution:
    def removeDuplicates(self, s: str, k: int) -> str:

        stk = []
        for ele in s:
            if not stk:
                stk.append((1, ele))
            elif stk:
                #if last element is equal to new element
                #and seen element is already at k-1
                #no need pushing
                #remove the already seen ones
                if stk[-1][1]== ele and stk[-1][0] == k-1:
                    for i in range(k-1):
                        stk.pop()
                #if it we have not seen up to k-1 characters
                #continue adding
                elif stk[-1][0] != k-1 and stk[-1][1]== ele:
                    newCount = stk[-1][0]+1
                    stk.append((newCount, ele))
                #if it is a new element
                #just append
                else:
                   stk.append((1, ele))
```

```
        res = ""
        for s in stk:
            res+=s[1]
        return res


# Code
# Testcase
# Testcase
# Test Result
# 978. Longest Turbulent Subarray
# Solved
# Medium
# Topics
# Companies
# Given an integer array arr, return the length of a maximum size
turbulent subarray of arr.

# A subarray is turbulent if the comparison sign flips between each
adjacent pair of elements in the subarray.

# More formally, a subarray [arr[i], arr[i + 1], ..., arr[j]] of arr
is said to be turbulent if and only if:

# For i <= k < j:
# arr[k] > arr[k + 1] when k is odd, and
# arr[k] < arr[k + 1] when k is even.
# Or, for i <= k < j:
# arr[k] > arr[k + 1] when k is even, and
# arr[k] < arr[k + 1] when k is odd.


# Example 1:

# Input: arr = [9,4,2,10,7,8,8,1,9]
# Output: 5
# Explanation: arr[1] > arr[2] < arr[3] > arr[4] < arr[5]
# Example 2:

# Input: arr = [4,8,12,16]
# Output: 2
# Example 3:

# Input: arr = [100]
# Output: 1
class Solution:
    def maxTurbulenceSize(self, arr: List[int]) -> int:
        def findLongestTurbulent():
            n = len(arr)
            dp1 = [1] * len(arr)
```

```
            dp2 = [1] * len(arr)
            for i in range(1, n):
                if i %2 == 0 and arr[i] < arr[i-1]:
                    dp1[i] = dp1[i-1] + 1
                elif i %2 == 1 and arr[i] > arr[i-1]:
                    dp1[i] = dp1[i-1] + 1

                elif i %2 ==0 and arr[i] > arr[i-1]:
                    dp2[i] = dp2[i-1] + 1
                elif i %2 == 1 and arr[i] < arr[i-1]:
                    dp2[i] = dp2[i-1] + 1
                else:
                    continue
            return max( max(dp1), max(dp2) )
        return findLongestTurbulent()

# 278. First Bad Version
# Solved
# Easy
# Topics
# Companies
# You are a product manager and currently leading a team to develop a
new product.
# Unfortunately, the latest version of your product fails the quality
check. Since each version is developed based on the previous version,
# all the versions after a bad version are also bad.

# Suppose you have n versions [1, 2, ..., n] and you want to find out
the first bad one, which causes all the following ones to be bad.

# You are given an API bool isBadVersion(version) which returns
whether version is bad.
# Implement a function to find the first bad version. You should
minimize the number of calls to the API.



# Example 1:

# Input: n = 5, bad = 4
# Output: 4
# Explanation:
# call isBadVersion(3) -> false
# call isBadVersion(5) -> true
# call isBadVersion(4) -> true
# Then 4 is the first bad version.
# Example 2:

# Input: n = 1, bad = 1
# Output: 1
```

```python
# The isBadVersion API is already defined for you.
# def isBadVersion(version: int) -> bool:

class Solution:
    def firstBadVersion(self, n: int) -> int:

        l, r = 1, n
        ans = -1
        while l <=r :

            mid = (l+r)//2

            if isBadVersion(mid):
                ans = mid
                r= mid -1
            else:
                l = mid +1
        return ans

# 1845. Seat Reservation Manager
# Solved
# Medium
# Topics
# Companies
# Hint
# Design a system that manages the reservation state of n seats that
are numbered from 1 to n.

# Implement the SeatManager class:

# SeatManager(int n) Initializes a SeatManager object that will manage
n seats numbered from 1 to n. All seats are initially available.
# int reserve() Fetches the smallest-numbered unreserved seat,
reserves it, and returns its number.
# void unreserve(int seatNumber) Unreserves the seat with the given
seatNumber.


# Example 1:

# Input
# ["SeatManager", "reserve", "reserve", "unreserve", "reserve",
"reserve", "reserve", "reserve", "unreserve"]
# [[5], [], [], [2], [], [], [], [], [5]]
# Output
# [null, 1, 2, null, 2, 3, 4, 5, null]

# Explanation
# SeatManager seatManager = new SeatManager(5); // Initializes a
SeatManager with 5 seats.
```

```
# seatManager.reserve();    // All seats are available, so return the
lowest numbered seat, which is 1.
# seatManager.reserve();    // The available seats are [2,3,4,5], so
return the lowest of them, which is 2.
# seatManager.unreserve(2); // Unreserve seat 2, so now the available
seats are [2,3,4,5].
# seatManager.reserve();    // The available seats are [2,3,4,5], so
return the lowest of them, which is 2.
# seatManager.reserve();    // The available seats are [3,4,5], so
return the lowest of them, which is 3.
# seatManager.reserve();    // The available seats are [4,5], so
return the lowest of them, which is 4.
# seatManager.reserve();    // The only available seat is seat 5, so
return 5.
# seatManager.unreserve(5); // Unreserve seat 5, so now the available
seats are [5].
import heapq
class SeatManager:

    def __init__(self, n: int):
        self.reservations = []
        for i in range(1, n+1):
            heapq.heappush(self.reservations, i)

    def reserve(self) -> int:
        if self.reservations:
            return heapq.heappop(self.reservations)
        return -1

    def unreserve(self, seatNumber: int) -> None:
        heapq.heappush(self.reservations, seatNumber)



# Your SeatManager object will be instantiated and called as such:
# obj = SeatManager(n)
# param_1 = obj.reserve()
# obj.unreserve(seatNumber)

# 645. Set Mismatch
# Solved
# Easy
# Topics
# Companies
# You have a set of integers s, which originally contains all the
numbers from 1 to n.
# Unfortunately, due to some error, one of the numbers in s got
duplicated to another number in the set, which results in repetition
of one number and loss of another number.
```

```python
# You are given an integer array nums representing the data status of
this set after the error.

# Find the number that occurs twice and the number that is missing and
return them in the form of an array.



# Example 1:

# Input: nums = [1,2,2,4]
# Output: [2,3]
# Example 2:

# Input: nums = [1,1]
# Output: [1,2]

class Solution:
    def findErrorNums(self, nums: List[int]) -> List[int]:

        unique_num = set()
        duplicate = [-1]
        n = len(nums)
        for num in nums:
            if num in unique_num:
                duplicate[0] = num
            unique_num.add(num)

        missin = -1
        for i in range(1, n+1):
            if i not in unique_num:
                missing = i
                break

        return [duplicate[0], missing]

# 1481. Least Number of Unique Integers after K Removals
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an array of integers arr and an integer k. Find the least
number of unique integers after removing exactly k elements.



# Example 1:

# Input: arr = [5,5,4], k = 1
# Output: 1
```

```
# Explanation: Remove the single 4, only 5 is left.
# Example 2:
# Input: arr = [4,3,1,1,3,3,2], k = 3
# Output: 2
# Explanation: Remove 4, 2 and either one of the two 1s or three 3s. 1
and 3 will be left.

class Solution:
    def findLeastNumOfUniqueInts(self, arr: List[int], k: int) -> int:

        freq_table = {}
        for num in arr:
            freq_table[num] = 1 + freq_table.get(num,0)
        freq_table = dict(sorted(freq_table.items(), key = lambda x :
x[1], reverse=False))

        for num, freq in freq_table.items():
            while k > 0 and freq_table[num] > 0:
                k-=1
                freq_table[num] -=1

        ans = set()
        for num, freq in freq_table.items():
            if freq > 0:
                ans.add(num)
        return len(ans)

# 881. Boats to Save People
# Solved
# Medium
# Topics
# Companies
# You are given an array people where people[i] is the weight of the
ith person, and an infinite number of
# boats where each boat can carry a maximum weight of limit. Each boat
carries at most two people at the same time, provided the sum of the
weight of those people is at most limit.

# Return the minimum number of boats to carry every given person.



# Example 1:

# Input: people = [1,2], limit = 3
# Output: 1
# Explanation: 1 boat (1, 2)
# Example 2:

# Input: people = [3,2,2,1], limit = 3
```

```
# Output: 3
# Explanation: 3 boats (1, 2), (2) and (3)
# Example 3:

# Input: people = [3,5,3,4], limit = 5
# Output: 4
# Explanation: 4 boats (3), (3), (4), (5)
class Solution:
    def numRescueBoats(self, people: List[int], limit: int) -> int:

        people.sort()
        l = 0
        r = len(people)-1
        ans = 0

        while l <= r:

            weight = people[l] + people[r]
            #if two people can enter
            if weight <= limit:
                l+=1
                r-=1
                ans+=1
            #otherwise let fattest person enter
            else:
                r-=1
                ans+=1
        return ans




# 1647. Minimum Deletions to Make Character Frequencies Unique
# Solved
# Medium
# Topics
# Companies
# Hint
# A string s is called good if there are no two different characters
in s that have the same frequency.

# Given a string s, return the minimum number of characters you need
to delete to make s good.

# The frequency of a character in a string is the number of times it
appears in the string. For example, in the string "aab", the frequency
of 'a' is 2, while the frequency of 'b' is 1.
```

```
# Example 1:

# Input: s = "aab"
# Output: 0
# Explanation: s is already good.
# Example 2:

# Input: s = "aaabbbcc"
# Output: 2
# Explanation: You can delete two 'b's resulting in the good string
"aaabcc".
# Another way it to delete one 'b' and one 'c' resulting in the good
string "aaabbc".
# Example 3:

# Input: s = "ceabaacb"
# Output: 2
# Explanation: You can delete both 'c's resulting in the good string
"eabaab".
# Note that we only care about characters that are still in the string
at the end (i.e. frequency of 0 is ignored).

class Solution:
    def minDeletions(self, s: str) -> int:

        cnt = Counter(s)
        freq = sorted(cnt.values(), reverse = True)
        s = set()
        ans = 0
        for v in freq:
            while v > 0 and v in s:
                v -=1
                ans +=1
            if v > 0:
                s.add(v)

        return ans

# 1011. Capacity To Ship Packages Within D Days
# Attempted
# Medium
# Topics
# Companies
# Hint
# A conveyor belt has packages that must be shipped from one port to
another within days days.

# The ith package on the conveyor belt has a weight of weights[i].
Each day, we load the ship with packages on the conveyor belt
#  (in the order given by weights). We may not load more weight than
```

```
the maximum weight capacity of the ship.

# Return the least weight capacity of the ship that will result in all
the packages on the conveyor belt being shipped within days days.



# Example 1:

# Input: weights = [1,2,3,4,5,6,7,8,9,10], days = 5
# Output: 15
# Explanation: A ship capacity of 15 is the minimum to ship all the
packages in 5 days like this:
# 1st day: 1, 2, 3, 4, 5
# 2nd day: 6, 7
# 3rd day: 8
# 4th day: 9
# 5th day: 10

# Note that the cargo must be shipped in the order given, so using a
ship of capacity 14 and
#  splitting the packages into parts like (2, 3, 4, 5), (1, 6, 7),
(8), (9), (10) is not allowed.
# Example 2:

# Input: weights = [3,2,2,4,1,4], days = 3
# Output: 6
# Explanation: A ship capacity of 6 is the minimum to ship all the
packages in 3 days like this:
# 1st day: 3, 2
# 2nd day: 2, 4
# 3rd day: 1, 4
# Example 3:

# Input: weights = [1,2,3,1,1], days = 4
# Output: 3
# Explanation:
# 1st day: 1
# 2nd day: 2
# 3rd day: 3
# 4th day: 1, 1

class Solution:
    def shipWithinDays(self, weights: List[int], days: int) -> int:
        l = max(weights)
        r = sum(weights)
        ans = r
        for capacity in range(l, r+1):
            d = 1
            total = 0
```

```
            for w in weights:
                if total + w > capacity:
                    total = w
                    d+=1
                else:
                    total+=w
            if d <= days:
                return capacity

        return ans



class Solution:
    def shipWithinDays(self, weights: List[int], days: int) -> int:
        l = max(weights)
        r = sum(weights)
        ans = r
        while l <= r:
            capacity = (l+r)//2
            total = 0
            d = 1
            for w in weights:
                if total + w > capacity:
                    total = w
                    d+=1
                else:
                    total+=w
            if d <= days:
                ans =  capacity
                r = capacity - 1
            else:
                l = capacity + 1

        return ans


# 535. Encode and Decode TinyURL
# Solved
# Medium
# Topics
# Companies
# Note: This is a companion problem to the System Design problem:
Design TinyURL.
# TinyURL is a URL shortening service where you enter a URL such as
https://leetcode.com/problems/design-tinyurl and it returns a short
URL such as
# http://tinyurl.com/4e9iAk. Design a class to encode a URL and decode
a tiny URL.
```

```python
# There is no restriction on how your encode/decode algorithm should
work. You just need to ensure that a URL can be encoded to a tiny URL
and the tiny URL can be decoded to the original URL.

# Implement the Solution class:

# Solution() Initializes the object of the system.
# String encode(String longUrl) Returns a tiny URL for the given
longUrl.
# String decode(String shortUrl) Returns the original long URL for the
given shortUrl. It is guaranteed that the given shortUrl was encoded
by the same object.


# Example 1:

# Input: url = "https://leetcode.com/problems/design-tinyurl"
# Output: "https://leetcode.com/problems/design-tinyurl"

# Explanation:
# Solution obj = new Solution();
# string tiny = obj.encode(url); // returns the encoded tiny url.
# string ans = obj.decode(tiny); // returns the original url after
decoding
class Codec:

    def __init__(self):
        self.base = "TinyUrl.com"
        self.short_to_long = {}
        self.long_to_short = {}
    def encode(self, longUrl: str) -> str:
        """Encodes a URL to a shortened URL.
        """
        if longUrl in self.long_to_short:
            return self.long_to_short[longUrl]

        encoding = longUrl + str(len(self.long_to_short))
        newUrl = self.base+"/"+encoding

        self.long_to_short[longUrl] = newUrl
        self.short_to_long[newUrl] = longUrl
        return self.long_to_short[longUrl]

    def decode(self, shortUrl: str) -> str:
        """Decodes a shortened URL to its original URL.
        """
        return self.short_to_long[shortUrl]


# Your Codec object will be instantiated and called as such:
```

```python
# codec = Codec()
# codec.decode(codec.encode(url))

# 2597. The Number of Beautiful Subsets
# Solved
# Medium
# Topics
# Companies
# Hint
# You are given an array nums of positive integers and a positive
integer k.

# A subset of nums is beautiful if it does not contain two integers
with an absolute difference equal to k.

# Return the number of non-empty beautiful subsets of the array nums.

# A subset of nums is an array that can be obtained by deleting some
(possibly none) elements from nums. Two subsets are different if and
only if the chosen indices to delete are different.



# Example 1:

# Input: nums = [2,4,6], k = 2
# Output:o 4
# Explanation: The beautiful subsets of the array nums are: [2], [4],
[6], [2, 6].
# It can be proved that there are only 4 beautiful subsets in the
array [2,4,6].
# Example 2:

# Input: nums = [1], k = 1
# Output: 1
# Explanation: The beautiful subset of the array nums is [1].
# It can be proved that there is only 1 beautiful subset in the array
[1].
class Solution:
    def beautifulSubsets(self, nums: List[int], k: int) -> int:
        ans = []
        def isBeautiful(arr, ele):
            for e in arr:
                if abs(e-ele) == k:
                    return False
            return True
        def dfs(arr, start):
            if len(arr) > 0:
                ans.append(arr.copy())
            for i in range(start, len(nums)):
```

```
                if isBeautiful(arr, nums[i]):
                    arr.append(nums[i])
                    dfs(arr, i+1)
                    arr.pop()

        dfs([], 0)
        return len(ans)


# 680. Valid Palindrome II
# Solved
# Easy
# Topics
# Companies
# Given a string s, return true if the s can be palindrome after
deleting at most one character from it.



# Example 1:

# Input: s = "aba"
# Output: true
# Example 2:

# Input: s = "abca"
# Output: true
# Explanation: You could delete the character 'c'.
# Example 3:

# Input: s = "abc"
# Output: false
class Solution:
    def validPalindrome(self, s: str) -> bool:

        def isPal(s, l, r):
            while l < r:
                if s[l] != s[r]:
                    return False
                l += 1
                r -= 1
            return True

        i, j = 0, len(s) - 1

        while i < j:
            if s[i] != s[j]:

                return isPal(s, i + 1, j) or isPal(s, i, j - 1)
            i += 1
            j -= 1
```

```
        return True


# Code
# Testcase
# Testcase
# Test Result
# 905. Sort Array By Parity
# Solved
# Easy
# Topics
# Companies
# Given an integer array nums, move all the even integers at the
beginning of the array followed by all the odd integers.

# Return any array that satisfies this condition.



# Example 1:

# Input: nums = [3,1,2,4]
# Output: [2,4,3,1]
# Explanation: The outputs [4,2,3,1], [2,4,1,3], and [4,2,1,3] would
also be accepted.
# Example 2:

# Input: nums = [0]
# Output: [0]
class Solution:
    def sortArrayByParity(self, nums: List[int]) -> List[int]:

        l = 0

        for i in range(len(nums)):
            if nums[i] %2 == 1:
                continue
            else:
                nums[l], nums[i] = nums[i], nums[l]
                l+=1

        return nums

# 739. Daily Temperatures
# Solved
# Medium
# Topics
# Companies
# Hint
# Given an array of integers temperatures represents the daily
```

```
temperatures, return an array answer such that answer[i]
# is the number of days you have to wait after the ith day to get a
warmer temperature. If there is no future day for which this is
possible, keep answer[i] == 0 instead.



# Example 1:

# Input: temperatures = [73,74,75,71,69,72,76,73]
# Output: [1,1,4,2,1,1,0,0]
# Example 2:

# Input: temperatures = [30,40,50,60]
# Output: [1,1,1,0]
# Example 3:

# Input: temperatures = [30,60,90]
# Output: [1,1,0]

class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        stk = []
        ans = [0] * len(temperatures)
        for i in range(len(temperatures)):

            while stk and temperatures[i] > stk[-1][1]:

                index, val = stk.pop()
                ans[index] = i - index
            stk.append((i, temperatures[i]))

        return ans



# 853. Car Fleet
# Medium
# Topics
# Companies
# There are n cars at given miles away from the starting mile 0,
traveling to reach the mile target.

# You are given two integer array position and speed, both of length
n, where position[i] is the starting mile of the ith car and speed[i]
is the speed of the ith car in miles per hour.

# A car cannot pass another car, but it can catch up and then travel
next to it at the speed of the slower car.

# A car fleet is a car or cars driving next to each other. The speed
```

```
of the car fleet is the minimum speed of any car in the fleet.

# If a car catches up to a car fleet at the mile target, it will still
be considered as part of the car fleet.

# Return the number of car fleets that will arrive at the destination.



# Example 1:

# Input: target = 12, position = [10,8,0,5,3], speed = [2,4,1,1,3]

# Output: 3

# Explanation:

# The cars starting at 10 (speed 2) and 8 (speed 4) become a fleet,
meeting each other at 12. The fleet forms at target.
# The car starting at 0 (speed 1) does not catch up to any other car,
so it is a fleet by itself.
# The cars starting at 5 (speed 1) and 3 (speed 3) become a fleet,
meeting each other at 6. The fleet moves at speed 1 until it reaches
target.
class Solution:
    def carFleet(self, target: int, position: List[int], speed:
List[int]) -> int:

        pair = [[p,s] for p, s in zip(position, speed)]

        stack = []

        for p,s in sorted(pair)[::-1]:
            time = (target - p) / s
            stack.append(time)
            #if there is an overlappig time,
            #defiinitely collision
            if len(stack ) >= 2 and stack[-1] <= stack[-2]:
                stack.pop()

        return len(stack)

# 875. Koko Eating Bananas
# Medium
# Topics
# Companies
# Koko loves to eat bananas. There are n piles of bananas, the ith
pile has piles[i] bananas. The guards have gone and will come back in
h hours.

# Koko can decide her bananas-per-hour eating speed of k. Each hour,
```

```
she chooses some pile of bananas and eats k bananas from that pile. If
the pile has less than k bananas, she eats all of them instead and
will not eat any more bananas during this hour.

# Koko likes to eat slowly but still wants to finish eating all the
bananas before the guards return.

# Return the minimum integer k such that she can eat all the bananas
within h hours.



# Example 1:

# Input: piles = [3,6,7,11], h = 8
# Output: 4
# Example 2:

# Input: piles = [30,11,23,4,20], h = 5
# Output: 30
# Example 3:

# Input: piles = [30,11,23,4,20], h = 6
# Output: 23

class Solution:
    def minEatingSpeed(self, piles: List[int], h: int) -> int:
        l, r = 1, max(piles)
        res = r

        while l <= r:
            k = (l + r) // 2

            totalTime = 0
            for p in piles:
                totalTime += math.ceil(float(p) / k)
            if totalTime <= h:
                res = k
                r = k - 1
            else:
                l = k + 1
        return res

# 153. Find Minimum in Rotated Sorted Array
# Solved
# Medium
# Topics
# Companies
# Hint
# Suppose an array of length n sorted in ascending order is rotated
between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7]
```

```
might become:

# [4,5,6,7,0,1,2] if it was rotated 4 times.
# [0,1,2,4,5,6,7] if it was rotated 7 times.
# Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time
results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

# Given the sorted rotated array nums of unique elements, return the
minimum element of this array.

# You must write an algorithm that runs in O(log n) time.



# Example 1:

# Input: nums = [3,4,5,1,2]
# Output: 1
# Explanation: The original array was [1,2,3,4,5] rotated 3 times.
# Example 2:

# Input: nums = [4,5,6,7,0,1,2]
# Output: 0
# Explanation: The original array was [0,1,2,4,5,6,7] and it was
rotated 4 times.
# Example 3:

# Input: nums = [11,13,15,17]
# Output: 11
# Explanation: The original array was [11,13,15,17] and it was rotated
4 times.

class Solution:
    def findMin(self, nums: List[int]) -> int:
        left, right = 0, len(nums) - 1
        while left < right:
            mid = left + (right - left) // 2
            if nums[mid] < nums[right]:
                right = mid
            else:
                left = mid + 1

        return nums[left]

# 981. Time Based Key-Value Store
# Medium
# Topics
# Companies
# Design a time-based key-value data structure that can store multiple
values for the same key at different time stamps and retrieve the
key's value at a certain timestamp.
```

```python
# Implement the TimeMap class:

# TimeMap() Initializes the object of the data structure.
# void set(String key, String value, int timestamp) Stores the key key
with the value value at the given time timestamp.
# String get(String key, int timestamp) Returns a value such that set
was called previously, with timestamp_prev <= timestamp.
# If there are multiple such values, it returns the value associated
with the largest timestamp_prev. If there are no values, it returns
"".


# Example 1:

# Input
# ["TimeMap", "set", "get", "get", "set", "get", "get"]
# [[], ["foo", "bar", 1], ["foo", 1], ["foo", 3], ["foo", "bar2", 4],
["foo", 4], ["foo", 5]]
# Output
# [null, null, "bar", "bar", null, "bar2", "bar2"]

# Explanation
# TimeMap timeMap = new TimeMap();
# timeMap.set("foo", "bar", 1);  // store the key "foo" and value
"bar" along with timestamp = 1.
# timeMap.get("foo", 1);         // return "bar"
# timeMap.get("foo", 3);         // return "bar", since there is no
value corresponding to foo at timestamp 3 and timestamp 2, then the
only value is at timestamp 1 is "bar".
# timeMap.set("foo", "bar2", 4); // store the key "foo" and value
"bar2" along with timestamp = 4.
# timeMap.get("foo", 4);         // return "bar2"
# timeMap.get("foo", 5);         // return "bar2"
class TimeMap:

    def __init__(self):
        self.map = {}

    def set(self, key: str, value: str, timestamp: int) -> None:

        if key not in self.map:
            self.map[key] = []

        self.map[key].append([value, timestamp])

    def get(self, key: str, timestamp: int) -> str:

        values = self.map.get(key, [])
```

```
        l, r = 0, len(values)-1
        res = ""
        while l <=r:
            m = (l+r)//2
            if values[m][1] <= timestamp:
                res = values[m][0]
                l = m+1
            else:
                r = m-1

        return res

# Your TimeMap object will be instantiated and called as such:
# obj = TimeMap()
# obj.set(key,value,timestamp)
# param_2 = obj.get(key,timestamp)

# 84. Largest Rectangle in Histogram
# Solved
# Hard
# Topics
# Companies
# Given an array of integers heights representing the histogram's bar
height where the width of each bar is 1, return the area of the
largest rectangle in the histogram.



# Example 1:


# Input: heights = [2,1,5,6,2,3]
# Output: 10
# Explanation: The above is a histogram where width of each bar is 1.
# The largest rectangle is shown in the red area, which has an area =
10 units.

class Solution:
    def largestRectangleArea(self, heights: List[int]) -> int:
        maxArea = 0
        stack = []  # pair: (index, height)

        for i, h in enumerate(heights):
            start = i
            while stack and stack[-1][1] > h:
                index, height = stack.pop()
                maxArea = max(maxArea, height * (i - index))
                start = index
            stack.append((start, h))
```

```
        for i, h in stack:
            maxArea = max(maxArea, h * (len(heights) - i))
        return maxArea


# 143. Reorder List
# Solved
# Medium
# Topics
# Companies
# You are given the head of a singly linked-list. The list can be
represented as:

# L0 → L1 → … → Ln - 1 → Ln
# Reorder the list to be on the following form:

# L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …
# You may not modify the values in the list's nodes. Only nodes
themselves may be changed.
# 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.
        """
        if not head or not head.next:
            return

        # Step 1: Find the middle of the list
        slow, fast = head, head.next
        while fast and fast.next:
            slow, fast = slow.next, fast.next.next

        # Step 2: Split the list into two halves
        first = head
        second = slow.next
        slow.next = None

        # Step 3: Reverse the second half
        prev = None
        while second:
            tmp = second.next
            second.next = prev
            prev = second
            second = tmp
```

```
        # Step 4: Merge the two halves
        second = prev
        while first and second:
            tmp1, tmp2 = first.next, second.next
            first.next = second
            second.next = tmp1
            first = tmp1
            second = tmp2

# 621. Task Scheduler
# Solved
# Medium
# Topics
# Companies
# Hint
# You are given an array of CPU tasks, each represented by letters A
to Z, and a cooling time, n. Each cycle or interval allows the
completion of one task.
# Tasks can be completed in any order, but there's a constraint:
identical tasks must be separated by at least n intervals due to
cooling time.

# Return the minimum number of intervals required to complete all
tasks.



# Example 1:

# Input: tasks = ["A","A","A","B","B","B"], n = 2

# Output: 8

# Explanation: A possible sequence is: A -> B -> idle -> A -> B ->
idle -> A -> B.

# After completing task A, you must wait two cycles before doing A
again. The same applies to task B. In the 3rd interval,
# neither A nor B can be done, so you idle. By the 4th cycle, you can
do A again as 2 intervals have passed.

# Example 2:

# Input: tasks = ["A","C","A","B","D","B"], n = 1

# Output: 6

# Explanation: A possible sequence is: A -> B -> C -> D -> A -> B.

# With a cooling interval of 1, you can repeat a task after just one
```

```
other task.

# Example 3:

# Input: tasks = ["A","A","A", "B","B","B"], n = 3

# Output: 10

# Explanation: A possible sequence is: A -> B -> idle -> idle -> A ->
B -> idle -> idle -> A -> B.

# There are only two types of tasks, A and B, which need to be
separated by 3 intervals. This leads to idling twice between
repetitions of these tasks.

import heapq
class Solution:
    def leastInterval(self, tasks: List[str], n: int) -> int:

        cnter = Counter(tasks)

        max_heap = [(-count, task) for task, count in cnter.items()]
        heapq.heapify(max_heap)

        q = deque()
        time = 0

        while max_heap or q:
            time += 1

            if max_heap:
                cnt, task = heapq.heappop(max_heap)
                cnt += 1
                order.append(task)
                if cnt != 0:
                    next_avail_time = time + n
                    q.append((cnt, next_avail_time, task))

            if q and q[0][1] == time:
                c, nat, t = q.popleft()
                heapq.heappush(max_heap, (c, t))

        return time

# 355. Design Twitter
# Solved
# Medium
# Topics
# Companies
# Design a simplified version of Twitter where users can post tweets,
follow/unfollow another user, and is able to see the 10 most recent
```

```
tweets in the user's news feed.

# Implement the Twitter class:

# Twitter() Initializes your twitter object.
# void postTweet(int userId, int tweetId) Composes a new tweet with ID
tweetId by the user userId. Each call to this function will be made
with a unique tweetId.
# List<Integer> getNewsFeed(int userId) Retrieves the 10 most recent
tweet IDs in the user's
#  news feed. Each item in the news feed must be posted by users who
the user followed or by the user themself. Tweets must be ordered from
most recent to least recent.
# void follow(int followerId, int followeeId) The user with ID
followerId started following the user with ID followeeId.
# void unfollow(int followerId, int followeeId) The user with ID
followerId started unfollowing the user with ID followeeId.

class Twitter:

    def __init__(self):
        self.account = defaultdict(set)
        self.feed = deque()

    def postTweet(self, userId: int, tweetId: int) -> None:
        self.feed.appendleft((userId, tweetId))

    def getNewsFeed(self, userId: int) -> List[int]:
        ans =  [tweetId for user, tweetId in self.feed if userId ==
user or user in self.account[userId]]
        return ans[:10]
    def follow(self, followerId: int, followeeId: int) -> None:
        self.account[followerId].add(followeeId)

    def unfollow(self, followerId: int, followeeId: int) -> None:
        self.account[followerId].discard(followeeId)

# 763. Partition Labels
# Solved
# Medium
# Topics
# Companies
# Hint
# You are given a string s. We want to partition the string into as
many parts as possible so that each letter appears in at most one
part.

# Note that the partition is done so that after concatenating all the
parts in order, the resultant string should be s.
```

```python
# Return a list of integers representing the size of these parts.



# Example 1:

# Input: s = "ababcbacadefegdehijhklij"
# Output: [9,7,8]
# Explanation:
# The partition is "ababcbaca", "defegde", "hijhklij".
# This is a partition so that each letter appears in at most one part.
# A partition like "ababcbacadefegde", "hijhklij" is incorrect,
because it splits s into less parts.
# Example 2:

# Input: s = "eccbbbbdec"
# Output: [10]

class Solution:
    def partitionLabels(self, s: str) -> List[int]:

        index_map = {}
        for i in range(len(s)):
            index_map[s[i]] = i

        max_i = 0
        l =0
        ans = []
        for i in range(len(s)):
            max_i = max(max_i, index_map[s[i]])
            if max_i == i:
                ans.append(i - l + 1)
                l = i +1

        return ans


class Solution:
    def partitionLabels(self, s: str) -> List[int]:

        cnt = {}
        st = set()
        for i in range(len(s)):
            cnt[s[i]] = cnt.get(s[i], 0) + 1


        l =0
        ans = []
        for r in range(len(s)):

            st.add(s[r])
```

```
            cnt[s[r]] -=1

            if cnt[s[r]] == 0:
                st.remove(s[r])
            if len(st) ==0:
                ans.append(r-l+1)
                l = r+1

        return ans

# 402. Remove K Digits
# Solved
# Medium
# Topics
# Companies
# Given string num representing a non-negative integer num, and an
integer k, return the smallest possible integer after removing k
digits from num.



# Example 1:

# Input: num = "1432219", k = 3
# Output: "1219"
# Explanation: Remove the three digits 4, 3, and 2 to form the new
number 1219 which is the smallest.
# Example 2:

# Input: num = "10200", k = 1
# Output: "200"
# Explanation: Remove the leading 1 and the number is 200. Note that
the output must not contain leading zeroes.
# Example 3:

# Input: num = "10", k = 2
# Output: "0"
# Explanation: Remove all the digits from the number and it is left
with nothing which is 0.
class Solution:
    def removeKdigits(self, num: str, k: int) -> str:
        stack = []

        for digit in num:
            while stack and k > 0 and stack[-1] > digit:
                stack.pop()
                k -= 1
            stack.append(digit)

        # If k > 0, remove remaining k digits from the end of the
```

```
stack
        stack = stack[:-k] if k > 0 else stack

        # Remove leading zeros
        result = ''.join(stack).lstrip('0')

        # Handle edge case where result might be empty
        return result if result else '0'


# Code
# Testcase
# Testcase
# Test Result
# 2706. Buy Two Chocolates
# Solved
# Easy
# Topics
# Companies
# Hint
# You are given an integer array prices representing the prices of
various chocolates in a store.
#You are also given a single integer money, which represents your
initial amount of money.

# You must buy exactly two chocolates in such a way that you still
have some non-negative leftover money.
# You would like to minimize the sum of the prices of the two
chocolates you buy.

# Return the amount of money you will have leftover after buying the
two chocolates. If there is no way for you to buy two chocolates
without ending up in debt, return money. Note that the leftover must
be non-negative.



# Example 1:

# Input: prices = [1,2,2], money = 3
# Output: 0
# Explanation: Purchase the chocolates priced at 1 and 2 units
respectively. You will have 3 - 3 = 0 units of money afterwards. Thus,
we return 0.
# Example 2:

# Input: prices = [3,2,3], money = 3
# Output: 3
# Explanation: You cannot buy 2 chocolates without going in debt, so
we return 3.
```

```python
class Solution:
    def buyChoco(self, prices: List[int], money: int) -> int:

        first_min, second_min = float("inf"), float("inf")

        for prc in prices:
            if prc < first_min:
                second_min = first_min
                first_min = prc
            elif prc < second_min:
                second_min = prc

        return (money - (first_min + second_min)) if (money -
(first_min + second_min)) >= 0 else money

class Solution:
    def buyChoco(self, prices: List[int], money: int) -> int:

        prices.sort()
        first_min, second_min = prices[0], prices[1]

        return (money - (first_min + second_min)) if (money -
(first_min + second_min)) >= 0 else money


# Code
# Testcase
# Testcase
# Test Result
# 1094. Car Pooling
# Solved
# Medium
# Topics
# Companies
# Hint
# There is a car with capacity empty seats. The vehicle only drives
east (i.e., it cannot turn around and drive west).

# You are given the integer capacity and an array trips where trips[i]
= [numPassengersi, fromi, toi]
# indicates that the ith trip has numPassengersi passengers and the
locations to pick them up and drop them off are fromi and toi
respectively. The locations are given as the number of kilometers due
east from the car's initial location.

# Return true if it is possible to pick up and drop off all passengers
for all the given trips, or false otherwise.



# Example 1:
```

```
# Input: trips = [[2,1,5],[3,3,7]], capacity = 4
# Output: false
# Example 2:

# Input: trips = [[2,1,5],[3,3,7]], capacity = 5
# Output: true


class Solution:
    def carPooling(self, trips: List[List[int]], capacity: int) ->
bool:
        stops = []
        for num, from_, to in trips:
            stops.append((from_, num))
            stops.append((to, -num))

        stops.sort()

        passengers = 0
        for _, num in stops:
            passengers += num
            if passengers > capacity:
                return False

        return True

# 896. Monotonic Array
# Solved
# Easy
# Topics
# Companies
# An array is monotonic if it is either monotone increasing or
monotone decreasing.

# An array nums is monotone increasing if for all i <= j, nums[i] <=
nums[j]. An array nums is monotone decreasing if for all i <= j,
nums[i] >= nums[j].

# Given an integer array nums, return true if the given array is
monotonic, or false otherwise.



# Example 1:

# Input: nums = [1,2,2,3]
# Output: true
# Example 2:

# Input: nums = [6,5,4,4]
```

```
# Output: true
# Example 3:

# Input: nums = [1,3,2]
# Output: false

class Solution:
    def isMonotonic(self, nums: List[int]) -> bool:
        if len(nums) == 0 or len(nums) ==1:
            return True
        def helper(nums, incrBias):

            for i in range(1, len(nums)):
                if incrBias and nums[i] < nums[i-1]:
                    return False
                elif not incrBias and nums[i] > nums[i-1]:
                    return False
            return True
        return helper(nums, True) or helper(nums, False)




# 1356. Sort Integers by The Number of 1 Bits
# Solved
# Easy
# Topics
# Companies
# Hint
# You are given an integer array arr. Sort the integers in the array
in ascending order by the
# number of 1's in their binary representation and in case of two or
more integers have the same number of 1's
# you have to sort them in ascending order.

# Return the array after sorting it.



# Example 1:

# Input: arr = [0,1,2,3,4,5,6,7,8]
# Output: [0,1,2,4,8,3,5,6,7]
# Explantion: [0] is the only integer with 0 bits.
# [1,2,4,8] all have 1 bit.
# [3,5,6] have 2 bits.
# [7] has 3 bits.
# The sorted array by bits is [0,1,2,4,8,3,5,6,7]
# Example 2:
```

```
# Input: arr = [1024,512,256,128,64,32,16,8,4,2,1]
# Output: [1,2,4,8,16,32,64,128,256,512,1024]
# Explantion: All integers have 1 bit in the binary representation,
you should just sort them in ascending order.

class Solution:
    def sortByBits(self, arr: List[int]) -> List[int]:

        def countBits(n):
            bits = 0
            while n >0:
                bt = n & 1
                if bt == 1:
                    bits+=1
                n = n >> 1
            return bits

        bitsMap = {}
        for n in arr:
            if n not in bitsMap:
                bitsMap[n] = countBits(n)

        def quickSort(l, r):
            if l < r:
                pivot = arr[r]
                p = l

                for i in range(l, r):
                    #sort by bits
                    #if bits are the same
                    #sort by value
                    if bitsMap[ arr[i] ] < bitsMap[pivot ] or
(bitsMap[arr[i]] == bitsMap[pivot] and arr[i] < pivot):
                        arr[p], arr[i] = arr[i], arr[p]
                        p+=1
                arr[p], arr[r] = arr[r], arr[p]

                quickSort(l, p-1)
                quickSort(p+1, r)

        quickSort(0, len(arr)-1)

        return arr



class Solution:
    def sortByBits(self, arr: List[int]) -> List[int]:
        return sorted(sorted(arr), key=lambda n: bin(n).count('1'))
```

```python
# 547. Number of Provinces
# Solved
# Medium
# Topics
# Companies
# There are n cities. Some of them are connected, while some are not.
If city a is connected directly with city b,
# and city b is connected directly with city c, then city a is
connected indirectly with city c.

# A province is a group of directly or indirectly connected cities and
no other cities outside of the group.

# You are given an n x n matrix isConnected where isConnected[i][j] =
1 if the ith city
# and the jth city are directly connected, and isConnected[i][j] = 0
otherwise.

# Return the total number of provinces.



# Example 1:


# Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]
# Output: 2
# Example 2:


# Input: isConnected = [[1,0,0],[0,1,0],[0,0,1]]
class Solution:
    def findCircleNum(self, isConnected: List[List[int]]) -> int:
        n = len(isConnected)
        explored = set()
        adj_list = {}
        for i in range(n):
            adj_list[i] = []
        for r in range(n):
            for c in range(n):
                if isConnected[r][c] == 1:
                    adj_list[r].append(c)
                    adj_list[c].append(r)
        def dfs(node):
            # if node not in explored:
            explored.add(node)
            for nei in adj_list[node]:
                if nei not in explored:
                    dfs(nei)
        def bfs(node):
            q = deque()
```

```
            explored.add(node)
            q.append(node)

            while q:
                n = q.popleft()
                for nei in adj_list[n]:
                    if nei not in explored:
                        explored.add(nei)
                        q.append(nei)

        res = 0
        for i in range(n):
            if i not in explored:
                #either bfs or dfs works
                #leetcode says dfs is faster
                dfs(i)
                # bfs(i)
                res+=1
        return res

import math
class Solution:
    #calculating distance: O(N)
    #sorting distances = O(NlogN)
    #total big O : NlogN
    def kClosest(self, points: List[List[int]], k: int) ->
List[List[int]]:

        distances = []
        for x, y in points:

            dist =  math.sqrt((x-0)**2 + (y-0)**2 )
            distances.append([dist, x,y])

        distances = sorted(distances, key = lambda x: x[0])

        result = []
        for i in range(k):
            _, x, y = distances[i]
            result.append([x,y])

        return result

import heapq
class Solution:
    #calculating distance: O(N)
    #heapify = O(N)
    #getting elements: O(KlogN)
    #total big O : KlogN
```

```
    def kClosest(self, points: List[List[int]], k: int) ->
List[List[int]]:

        minHeap = []
        for x, y in points:

            # dist =  math.sqrt((x-0)**2 + (y-0)**2 )
            #since we just need to know how far
            #we do not need to know the sqrt root
            #because for example the sqrt(5) will always be greater
than sqrt(4)
            #we just do absolute difference
            #Also subtractint zero does nothing
            dist =  (x-0)**2 + (y-0)**2
            minHeap.append([dist, x,y])

        heapq.heapify(minHeap)
        res = []
        for i in range(k):
            _, x, y = heapq.heappop(minHeap)

            res.append([x,y])

        return res



class Solution:
  #improvement, The heap size is never more than k
    def kClosest(self, points: List[List[int]], K: int) ->
List[List[int]]:
        heap = []
        for x, y in points:
            d = -(x*x + y*y)
            heappush(heap, (d, x, y))
            if len(heap) > K:
                heappop(heap)

        return [[x, y] for _, x, y in heap]
```

240. Search a 2D Matrix II
Solved
Medium
Topics
Companies
Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.
Integers in each column are sorted in ascending from top to bottom.

```
Example 1:


Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],
[10,13,14,17,24],[18,21,23,26,30]], target = 5
Output: true
Example 2:


Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],
[10,13,14,17,24],[18,21,23,26,30]], target = 20
Output: false

class Solution:
  #time: O(rows * log cols)
    def searchMatrix(self, matrix: List[List[int]], target: int) ->
bool:

        rows, cols = len(matrix), len(matrix[0])

        for r in range(rows):

            if target >= matrix[r][0] and matrix[r][cols-1] >= target:

                #do binary search
                start,end = 0, cols-1

                while start <= end:
                    mid = (start + end) // 2

                    if matrix[r][mid] > target:
                        end = mid -1
                    elif matrix[r][mid] < target:
                        start = mid + 1
                    else:
                        return True

        return False


class Solution:
  #Time : O(M+N)
    def searchMatrix(self, matrix: List[List[int]], target: int) ->
bool:

        rows, cols = len(matrix), len(matrix[0])

        r, c = 0, cols-1

        while r < rows and c >= 0:
```

```
            if matrix[r][c] > target:
                c-=1
            elif matrix[r][c] < target:
                r +=1
            else:
                return True


        return False
'''
339. Nested List Weight Sum
Solved C
a Companies
You are given a nested list of integers nestedList.
 Each element is either an integer or a list whose elements
  may also be integers or other lists.
The depth of an integer is the number of lists
 that it is inside of. For example,
 the nested list [1, (2,2), [ [3) ,2),1)
  has each integer's value set to its depth.
Return the sum of each integer in nestedlist
 muitiplied by its depth.
Example 1:
nestedList =
depth =
Input: nestedList = [[1,1],2, [1,1]]
Output: 10
Explanation: Four 1's at depth 2, one 2 at depth 1. 1*2 + 1*2 + 2*1 +
1*2 + 1*2 = 10.
Example 2:
nestedList =
[1,
14,
depth =
Input: nestedList = [1, (4, [6]]]
Output: 27
27.
Explanation: One 1 at depth 1, one 4 at depth 2, and one 6 at depth 3.
1*1 + 4*2 + 6*3 =
'''


class Solution:
   def depthSum(self, nestedList: List (Nestedinteger]) -> int:
       def find_sum(nestedList, depth):
         ans = 0
         for el in
           if el.isInteger():
```

```
            ans += el.getinteger() * depth
         else:
            ans += find_sum(el.getList(), depth + 1)
      return ans
   return find_sun(nestedList, 1)
```