Given an array of string words, return all strings in words that is a substring of another word. You can return the answer in any order. A substring is a contiguous sequence of characters within a string Example 1: Input: words = ["mass","as","hero","superhero"] Output: ["as","hero"] Explanation: "as" is substring of "mass" and "hero" is substring of "superhero". ["hero","as"] is also a valid answer.

```
def stringMatching(words):
    return [word for word in words if any(other_word != word and other_word.find(word) != -1 for other_word in words)]

words = ["mass", "as", "hero", "superhero"]
print(stringMatching(words))
```

Given an m x n binary matrix mat, return the distance of the nearest 0 for each cell. The distance between two adjacent cells is 1. Input: mat = [[0,0,0],[0,1,0],[0,0,0]] Output: [[0,0,0],[0,1,0],[0,0,0]] Input: mat = [[0,0,0],[0,1,0],[1,1,1]] Output: [[0,0,0],[0,1,0],[1,2,1]]

```
from collections import deque

def updateMatrix(mat):
    rows, cols = len(mat), len(mat[0])
    queue = deque()

    for i in range(rows):
        for j in range(cols):
            if mat[i][j] == 0:
                queue.append((i, j))
            else:
                mat[i][j] = float('inf')

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    while queue:
```

```
        cell = queue.popleft()
        for d in directions:
            new_i, new_j = cell[0] + d[0], cell[1] + d[1]
            if 0 <= new_i < rows and 0 <= new_j < cols and mat[new_i][new_j] > mat[cell[0]][cell[1]] + 1:
                mat[new_i][new_j] = mat[cell[0]][cell[1]] + 1
                queue.append((new_i, new_j))

    return mat
mat1 = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
mat2 = [[0, 0, 0], [0, 1, 0], [1, 1, 1]]

print(updateMatrix(mat1))
print(updateMatrix(mat2))
```

Given two integer arrays arr1 and arr2, return the minimum number of operations (possibly zero) needed to make arr1 strictly increasing. In one operation, you can choose two indices 0 <= i < arr1.length and 0 <= j < arr2.length and do the assignment arr1[i] = arr2[j]. If there is no way to make arr1 strictly increasing, return -1. Example 1: Input: arr1 = [1,5,3,6,7], arr2 = [1,3,2,4] Output: 1 Explanation: Replace 5 with 2, then arr1 = [1, 2, 3, 6, 7].

```
            def min_operations(arr1, arr2):
    n, m = len(arr1), len(arr2)
    dp = {0: -arr1[0] - 1}
    for i in range(1, n):
        ndp = {}
        for key in dp:
            if arr1[i] > dp[key]:
                ndp[key] = max(ndp.get(key, 0), arr1[i])
            for j in range(m):
                if arr2[j] > dp[key]:
                    ndp[key + 1] = max(ndp.get(key + 1, 0), arr2[j])
```

```
            dp = ndp
        if dp:
            return n - max(dp)
        return -1

# Example
arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
print(min_operations(arr1, arr2))
```

Given two strings a and b, return the minimum number of times you should repeat string a so that string b is a substring of it. If it is impossible for b to be a substring of a after repeating it, return -1. Notice: string "abc" repeated 0 times is "", repeated 1 time is "abc" and repeated 2 times is "abcabc". Example 1: Input: a = "abcd", b = "cdabcdab" Output: 3 Explanation: We return 3 because by repeating a three times "abcdabcdabcd", b is a substring of it.

```
                def min_repeats(a, b):
    if b in a:
        return 1
    for i in range(1, len(b)):
        if b.startswith(a[-i:]):
            return i + 1
    return -1
a = "abcd"
b = "cdabcdab"
print(min_repeats(a, b))
```

Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array. Example 1: Input: nums = [3,0,1]Output: 2 Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

```
                def missing_number(nums):
```

```
    n = len(nums)
    total_sum = n * (n + 1) // 2
    actual_sum = sum(nums)
    return total_sum - actual_sum

# Example
nums = [3, 0, 1]
print(missing_number(nums))
```

You are given an n x n integer matrix grid.Generate an integer matrix maxLocal of size (n - 2) x (n - 2) such that: maxLocal[i][j] is equal to the largest value of the 3 x 3 matrix in grid centered around row i + 1 and column j + 1. In other words, we want to find the largest value in every contiguous 3 x 3 matrix in grid. Return the generated matrix. Input: grid = [[9,9,8,1],[5,6,2,6],[8,2,6,4],[6,2,2,2]] Output: [[9,9],[8,6]] Explanation: The diagram above shows the original matrix and the generated matrix. Notice that each value in the generated matrix corresponds to the largest value of a contiguous 3 x 3 matrix in grid.

```
            def generate_max_local(grid):
    n = len(grid)
    max_local = [[0] * (n - 2) for _ in range(n - 2)]

    for i in range(n - 2):
        for j in range(n - 2):
            max_local[i][j] = max(grid[i][j], grid[i][j + 1], grid[i][j + 2],
                                  grid[i + 1][j], grid[i + 1][j + 1], grid[i + 1][j + 2],
                                  grid[i + 2][j], grid[i + 2][j + 1], grid[i + 2][j + 2])

    return max_local

# Input matrix
grid = [[9, 9, 8, 1], [5, 6, 2, 6], [8, 2, 6, 4], [6, 2, 2, 2]]
output_matrix = generate_max_local(grid)
```

print(output_matrix)

You are given an array of strings words and a string pref. Return the number of strings in words that contain pref as a prefix. A prefix of a string s is any leading contiguous substring of s. Example 1: Input: words = ["pay","attention","practice","attend"], pref = "at" Output: 2 Explanation: The 2 strings that contain "at" as a prefix are: "attention" and "attend".

```
def count_strings_with_prefix(words, pref):
    return sum(1 for word in words if word.startswith(pref))

# Example
words = ["pay", "attention", "practice", "attend"]
pref = "at"
output = count_strings_with_prefix(words, pref)
print(output)
```

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. Input: matrix = [[1,1,1],[1,0,1],[1,1,1]] Output: [[1,0,1],[0,0,0],[1,0,1]]

```
def setZeroes(matrix):
    rows, cols = len(matrix), len(matrix[0])
    zero_rows, zero_cols = set(), set()

    for i in range(rows):
        for j in range(cols):
            if matrix[i][j] == 0:
                zero_rows.add(i)
                zero_cols.add(j)

    for i in range(rows):
        for j in range(cols):
            if i in zero_rows or j in zero_cols:
                matrix[i][j] = 0
```

```
matrix = [[1, 1, 1], [1, 0, 1], [1, 1, 1]]
setZeroes(matrix)
print(matrix)
```

Given two integer arrays nums1 and nums2, return an array of their intersection . Each element in the result must be unique and you may return the result in any order. Example 1: Input: nums1 = [1,2,2,1], nums2 = [2,2] Output: [2] Example 2: Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4] Output: [9,4] Explanation: [4,9] is also accepted. Constraints: 1 <= nums1.length, nums2.length <= 1000 0 <= nums1[i], nums2[i] <= 1000

```
        def intersection(nums1, nums2):
    set1 = set(nums1)
    set2 = set(nums2)
    return list(set1.intersection(set2))

# Example 1
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersection(nums1, nums2))  # Output: [2]

# Example 2
nums1 = [4, 9, 5]
nums2 = [9, 4, 9, 8, 4]
print(intersection(nums1, nums2))
```

Given the strings s1 and s2 of size n and the string evil, return the number of good strings. A good string has size n, it is alphabetically greater than or equal to s1, it is alphabetically smaller than or equal to s2, and it does not contain the string evil as a substring. Since the answer can be a huge number, return this modulo 109 + 7. Example 1: Input: n = 2, s1 = "aa", s2 = "da", evil = "b" Output: 51 Explanation: There are 25 good strings starting with 'a': "aa","ac","ad",...,"az". Then there are 25 good strings starting with 'c': "ca","cc","cd",...,"cz" and finally there is one good string starting with 'd': "da".

```
            def countGoodStrings(n, s1, s2, evil):
```

```
    MOD = 10**9 + 7
    dp = [[[0] * 2 for _ in range(len(evil) + 1)] for _ in range(n + 1)]
    dp[0][0][0] = 1

    for i in range(1, n + 1):
        for j in range(len(evil) + 1):
            for k in range(2):
                for ch in range(ord(s1[i - 1]) if k == 0 else ord('a'), ord(s2[i - 1]) + 1 if k == 1 else ord('z') + 1):
                    nk = k
                    nj = j
                    while nj and evil[nj - 1] != chr(ch):
                        nj = dp[nj][nj - 1][nk]
                    if evil[nj] == chr(ch):
                        nj += 1
                    if nj == len(evil):
                        continue
                    dp[i][nj][nk | (ch > ord(s1[i - 1]))] += dp[i - 1][j][k]
                    dp[i][nj][nk | (ch > ord(s1[i - 1]))] %= MOD

    res = sum(sum(dp[n][j][1] for j in range(len(evil) + 1)) for n in range(n + 1)) % MOD
    return res

n = 2
s1 = "aa"
s2 = "da"
evil = "b"
output = countGoodStrings(n, s1, s2, evil)
print(output)  # Output: 51
```

Given an integer array nums, reorder it such that nums[0] < nums[1] > nums[2] < nums[3].... You may assume the input array always has a valid answer. Example 1: Input: nums = [1,5,1,1,6,4]

Output: [1,6,1,5,1,4] Explanation: [1,4,1,5,1,6] is also accepted. Example 2: Input: nums = [1,3,2,2,3,1] Output: [2,3,1,3,1,2]

```
def wiggleSort(nums):
    nums.sort()
    half = len(nums[::2])
    nums[::2], nums[1::2] = nums[:half][::-1], nums[half:][::-1]
```

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.Merge all the linked-lists into one sorted linked-list and return it. Input: lists = [[1,4,5],[1,3,4],[2,6]] Output: [1,1,2,3,4,4,5,6] Explanation: The linked-lists are: [1->4->5, 1->3->4, 2->6] merging them into one sorted list: 1->1->2->3->4->4->5->6

```
from queue import PriorityQueue

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def mergeKLists(lists):
    head = point = ListNode(0)
    q = PriorityQueue()
    for l in lists:
        if l:
            q.put((l.val, l))

    while not q.empty():
        val, node = q.get()
        point.next = ListNode(val)
        point = point.next
        node = node.next
```

```
    if node:
        q.put((node.val, node))

return head.next
```