1) 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.

sol:

```
def find_substrings(words):
    result = []
    n = len(words)
    for i in range(n):
        for j in range(n):
            if i != j and words[i] in words[j]:
                result.append(words[i])
                break
    return result
words = ["mass", "as", "hero", "superhero"]
print(find_substrings(words))
```

2) 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]]

Sol:

```
from collections import deque


def updateMatrix(mat):
    rows, cols = len(mat), len(mat[0])
    dist = [[float('inf')] * cols for _ in range(rows)]
    queue = deque()

    # Initialize the queue with all 0s positions and set their distance to 0
    for r in range(rows):
```

```
        for c in range(cols):
            if mat[r][c] == 0:
                dist[r][c] = 0
                queue.append((r, c))

    # Directions for moving up, down, left, right
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    # BFS from all 0s
    while queue:
        x, y = queue.popleft()

        for dr, dc in directions:
            new_x, new_y = x + dr, y + dc

            if 0 <= new_x < rows and 0 <= new_y < cols:
                if dist[new_x][new_y] > dist[x][y] + 1:
                    dist[new_x][new_y] = dist[x][y] + 1
                    queue.append((new_x, new_y))

    return dist
mat1 = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
print(updateMatrix(mat1))
mat2 = [[0, 0, 0], [0, 1, 0], [1, 1, 1]]
```

3) 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].

Sol:

```
from bisect import bisect_right
from collections import defaultdict
def makeArrayIncreasing(arr1, arr2):
    arr2 = sorted(set(arr2))
    dp = {-1: 0}
    for i in range(len(arr1)):
        new_dp = defaultdict(lambda: float('inf'))
        for last, steps in dp.items():

            if arr1[i] > last:
                new_dp[arr1[i]] = min(new_dp[arr1[i]], steps)

            idx = bisect_right(arr2, last)
            if idx < len(arr2):
                new_dp[arr2[idx]] = min(new_dp[arr2[idx]], steps + 1)
        dp = new_dp
    if dp:
        return min(dp.values())
    else:
        return -1
arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
print(makeArrayIncreasing(arr1, arr2))
```

4) 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.

Sol:

```
def repeatedStringMatch(a, b):
    from math import ceil

    # Calculate the minimum repeats required
    min_repeats = ceil(len(b) / len(a))

    # Check if b is a substring of the repeated a
    repeated_a = a * min_repeats
    if b in repeated_a:
        return min_repeats

    # Check the next repeat to account for boundary overlaps
    repeated_a += a
    if b in repeated_a:
        return min_repeats + 1

    return -1

# Example usage:
a = "abcd"
b = "cdabcdab"
print(repeatedStringMatch(a, b))  # Output: 3
```

5) 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.

Sol:

```
def missingNumber(nums):
    n = len(nums)
    expected_sum = n * (n + 1) // 2
    actual_sum = sum(nums)
    return expected_sum - actual_sum

# Example usage:
nums = [3, 0, 1]
print(missingNumber(nums))  # Output: 2
```