1.GIVEN AN INTEGER ARRAY NUM SORTED IN NON-DECREASING ORDER. YOU CAN PERFORM THE FOLLOWING OPERATION ANY NUMBER OF TIMES: CHOOSE TWO INDICES, I AND J, WHERE NUMS[I] < NUMS[J]. THEN, REMOVE THE ELEMENTS AT INDICES I AND J FROM NUMS. THE REMAINING ELEMENTS RETAIN THEIR ORIGINAL ORDER, AND THE ARRAY IS REINDEXED. RETURN THE MINIMUM LENGTH OF NUMS AFTER APPLYING THE OPERATION ZERO OR MORE TIMES.

EXAMPLE 1: INPUT: NUMS = [1,2,3,4]

CONSTRAINTS:

1 <= NUMS.LENGTH <= 105

1 <= NUMS[I] <= 109

NUMS IS SORTED IN NON-DECREASING ORDER.

```
def min_length_after_operations(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        if nums[left] < nums[right]:
            left += 1
            right -= 1
        else:
            right -= 1
    return (right - left + 1) * 2


# Example usage:
nums = [1, 2, 3, 4]
print(min_length_after_operations(nums))
```

OUTPUT: 0

2. 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"]

EXPLANATION: "AS" IS SUBSTRING OF "MASS" AND "HERO" IS SUBSTRING OF "SUPERHERO".

["HERO","AS"] IS ALSO A VALID ANSWER.

```
def find_substrings(words):
```

```
    substrings = set()  # Using a set to avoid duplicates
    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                substrings.add(words[i])
    return list(substrings)


# Example usage:
words = ["mass", "as", "hero", "superhero"]
print(find_substrings(words))
```

OUTPUT: ["as", "hero"]

3.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):
    if not mat:
        return mat
    m, n = len(mat), len(mat[0])
    distances = [[float('inf')] * n for _ in range(m)]
    queue = deque()
     for i in range(m):
        for j in range(n):
            if mat[i][j] == 0:
                distances[i][j] = 0
                queue.append((i, j))

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

```
    while queue:
        r, c = queue.popleft()

        for dr, dc in directions:
            nr, nc = r + dr, c + dc

            if 0 <= nr < m and 0 <= nc < n:
                if distances[nr][nc] > distances[r][c] + 1:
                    distances[nr][nc] = distances[r][c] + 1
                    queue.append((nr, nc))
    return distances
```

OUTPUT: [[0,0,0],[0,1,0],[1,2,1]]

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


nums1 = [1, 5, 1, 1, 6, 4]
wiggleSort(nums1)
print(nums1)


nums2 = [1, 3, 2, 2, 3, 1]
wiggleSort(nums2)
print(nums2)
```

OUTPUT: 1

5. 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 updateMatrix(mat):
    m, n = len(mat), len(mat[0])
    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    queue = []
    dist = [[float('inf')] * n for _ in range(m)]


    for i in range(m):
        for j in range(n):
            if mat[i][j] == 0:
                queue.append((i, j))
                dist[i][j] = 0


    index = 0
    while index < len(queue):
        x, y = queue[index]
        index += 1
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < m and 0 <= ny < n:
                if dist[nx][ny] > dist[x][y] + 1:
                    dist[nx][ny] = dist[x][y] + 1
                    queue.append((nx, ny))
```

```
    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]]
print(updateMatrix(mat2))
```

OUTPUT: 3