1.Given a 2D integer array matrix, return the transpose of matrix. The transpose of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.

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

Output: [[1,4,7],[2,5,8],[3,6,9]]

Example 2: Input: matrix = [[1,2,3],[4,5,6]]

Output: [[1,4],[2,5],[3,6]]

```
def transpose(matrix):
    rows = len(matrix)
    cols = len(matrix[0])

    transposed_matrix = [[0] * rows for _ in range(cols)]

    for i in range(rows):
        for j in range(cols):
            transposed_matrix[j][i] = matrix[i][j]

    return transposed_matrix

matrix1 = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print(transpose(matrix1))
# OUTPUT: [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

matrix2 = [
    [1, 2, 3],
    [4, 5, 6]
]
print(transpose(matrix2))
```

 # OUTPUT: [[1, 4], [2, 5], [3, 6]]

2.GIVEN AN INTEGER N, RETURN THE NTH DIGIT OF THE INFINITE INTEGER SEQUENCE [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...].

EXAMPLE 1: INPUT: N = 3

OUTPUT: 3

EXAMPLE 2: INPUT: N = 11

OUTPUT: 0

EXPLANATION: THE 11TH DIGIT OF THE SEQUENCE 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... IS A 0, WHICH IS PART OF THE NUMBER 10.

```
def findNthDigit(n):
    digit_length = 1
    count = 9
    start = 1

    while n > digit_length * count:
        n -= digit_length * count
        digit_length += 1
        count *= 10
        start *= 10

    start += (n - 1) // digit_length
    s = str(start)

    return int(s[(n - 1) % digit_length])

print(findNthDigit(3))
```

 # OUTPUT: 3

```
print(findNthDigit(11))
```

 # OUTPUT: 0

3.GIVEN A SENTENCE THAT CONSISTS OF SOME WORDS SEPARATED BY A SINGLE SPACE, AND A SEARCHWORD, CHECK IF SEARCHWORD IS A PREFIX OF ANY WORD IN SENTENCE. RETURN THE INDEX OF THE WORD IN SENTENCE (1-INDEXED) WHERE SEARCHWORD IS

A PREFIX OF THIS WORD. IF SEARCHWORD IS A PREFIX OF MORE THAN ONE WORD, RETURN THE INDEX OF THE FIRST WORD (MINIMUM INDEX). IF THERE IS NO SUCH WORD RETURN - 1. A PREFIX OF A STRING S IS ANY LEADING CONTIGUOUS SUBSTRING OF S.

EXAMPLE 1: INPUT: SENTENCE = "I LOVE EATING BURGER", SEARCHWORD = "BURG" OUTPUT: 4

EXPLANATION: "BURG" IS PREFIX OF "BURGER" WHICH IS THE 4TH WORD IN THE SENTENCE.

```
def isPrefixOfWord(sentence, searchWord):
    words = sentence.split()

    for index, word in enumerate(words):
        if word.startswith(searchWord):
            return index + 1

    return -1

sentence1 = "i love eating burger"
searchWord1 = "burg"
print(isPrefixOfWord(sentence1, searchWord1))
```

 # OUTPUT: 4

```
sentence2 = "this problem is an easy problem"
searchWord2 = "pro"
print(isPrefixOfWord(sentence2, searchWord2))
```

 # OUTPUT: 2

```
sentence3 = "hello from the other side"
searchWord3 = "they"
print(isPrefixOfWord(sentence3, searchWord3))
```

 # OUTPUT: -1

4.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]

OUTPUT: 0 CONSTRAINTS: 1 <= NUMS.LENGTH <= 105 1 <= NUMS[I] <= 109 NUMS IS SORTED IN NON-DECREASING ORDER

```
def min_length_after_operations(nums):
    left = 0
    right = len(nums) - 1
    pairs = 0

    while left < right:
        pairs += 1
        left += 1
        right -= 1

    return len(nums) - 2 * pairs

nums1 = [1, 2, 3, 4]
print(min_length_after_operations(nums1))
```

 # OUTPUT: 0

5.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 find_substrings(words):
    substrings = []

    for i in range(len(words)):
        for j in range(len(words)):
```

```
            if i != j and words[i] in words[j]:
                substrings.append(words[i])
                break  # Move to the next word after finding the substring match

    return substrings

words1 = ["mass", "as", "hero", "superhero"]
print(find_substrings(words1))  # Output: ["as", "hero"]
```