1.Odd String Difference You are given an array of equal-length strings words. Assume that the length of each string is n. Each string words[i] can be converted into a difference integer array difference[i] of length n - 1 where difference[i][j] = words[i][j+1] - words[i][j] where 0 <= j <= n - 2. Note that the difference between two letters is the difference between their positions in the alphabet i.e. the position of 'a' is 0, 'b' is 1, and 'z' is 25. For example, for the string "acb", the difference integer array is [2 - 0, 1 - 2] = [2, -1]. All the strings in words have the same difference integer array, except one. You should find that string. Return the string in words that has different difference integer array.

```
def find_difference_string(words):
    def get_difference_array(word):
        return [ord(word[i + 1]) - ord(word[i]) for i in range(len(word) - 1)]

    difference_arrays = [get_difference_array(word) for word in words]
    unique_array = None

    for arr in difference_arrays:
        if difference_arrays.count(arr) == 1:
            unique_array = arr
            break

    unique_string = words[difference_arrays.index(unique_array)]
    return unique_string

# Example
words = ["abc", "def", "ghi", "jklm"]
result = find_difference_string(words)
print(result)  # Output should be the string with a different difference array
```

2. Words Within Two Edits of Dictionary You are given two string arrays, queries and dictionary. All words in each array comprise of lowercase English letters and have the same length. In one edit you can take a word from queries, and change any letter in it to any other letter. Find all words from queries that, after a maximum of two edits, equal some word from dictionary. Return a list of all words from queries, that match with some word from dictionary after a maximum of two edits. Return the words in the same order they appear in queries.

from collections import Counter

```
def edit_distance(word1, word2):
    return sum((Counter(word1) - Counter(word2)).values())


def find_matching_words(queries, dictionary):
    result = []
    for query in queries:
        for word in dictionary:
            if edit_distance(query, word) <= 2:
                result.append(query)
                break
    return result


# Example usage
queries = ["dog", "cat", "bat"]
dictionary = ["dot", "cat", "hat"]
matching_words = find_matching_words(queries, dictionary)
print(matching_words)  # Output: ['cat']
```

3. Next Greater Element IV You are given a 0-indexed array of non-negative integers nums. For each integer in nums, you must find its respective second greater integer. The second greater integer of nums[i] is nums[j] such that: j > i nums[j] > nums[i] There exists exactly one index k such that nums[k] > nums[i] and i < k < j. If there is no such nums[j], the second greater integer is considered to be -1. For example, in the array [1, 2, 4, 3], the second greater integer of 1 is 4, 2 is 3, and that of 3 and 4 is -1.

```
def find_second_greater(nums):
    stack = []
    result = [-1] * len(nums)

    for i in range(len(nums)):
        while stack and nums[i] > nums[stack[-1]]:
            top = stack.pop()
            if stack:
```

```
            result[stack[-1]] = nums[i]
        stack.append(i)

    return result

# Example
nums = [1, 2, 4, 3]
result = find_second_greater(nums)
print(result)  # Output: [4, 3, -1, -1]
```

4. Minimum Addition to Make Integer Beautiful You are given two positive integers n and target. An integer is considered beautiful if the sum of its digits is less than or equal to target. Return the minimum non-negative integer x such that n + x is beautiful. The input will be generated such that it is always possible to make n beautiful.

```
def sum_of_digits(num):
    return sum(int(digit) for digit in str(num))

def min_addition_to_beautiful(n, target):
    x = 0
    while True:
        if sum_of_digits(n + x) <= target:
            return x
        x += 1

# Example Usage
n = 123
target = 10
result = min_addition_to_beautiful(n, target)
print(result)  # Output: 1
```

5. Sort Array by Moving Items to Empty Space You are given an integer array nums of size n containing each element from 0 to n - 1 (inclusive). Each of the elements from 1 to n - 1 represents an item, and the element 0 represents an empty space. In one operation, you can move any item to the empty space. nums is considered to be sorted if the numbers of all the items are in ascending order and the empty space is either at the beginning or at the end of the array. For

example, if n = 4, nums is sorted if: ● nums = [0,1,2,3] or ● nums = [1,2,3,0] ...and considered to be unsorted otherwise. Return the minimum number of operations needed to sort nums

```
def min_operations_to_sort(nums):
    n = len(nums)
    operations = 0
    for i in range(1, n):
        if nums[i] != i:
            operations += 1
    return operations


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