36. 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"

Program:

```
def count_prefixes(words, pref):

    return len([word for word in words if word.startswith(pref)])

# Test the function

words = ["pay", "attention", "practice", "attend"]

pref = "at"

print(count_prefixes(words, pref))

# Output: 2
```

31. Given an array of strings strs, group the anagrams together. You can return the answer in any order. An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1: Input: strs = ["eat","tea","tan","ate","nat","bat"]

Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Example 2: Input: strs = [""] Output: [[""]]

Program:

```
from collections import defaultdict

def group_anagrams(strs):

    grouped_anagrams = defaultdict(list)

        for word in strs:

        key = ''.join(sorted(word))

        grouped_anagrams[key].append(word)

    return list(grouped_anagrams.values())

# Example 1
```

```
strs1 = ["eat", "tea", "tan", "ate", "nat", "bat"]

print(group_anagrams(strs1))

# Output: [["eat", "tea", "ate"], ["tan", "nat"], ["bat"]]

# Example 2

strs2 = [""]

print(group_anagrams(strs2))

# Output: [[""]]
```

37. 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]]

Program:

```
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

# Test the function with the provided example

matrix = [[1, 1, 1], [1, 0, 1], [1, 1, 1]]

setZeroes(matrix)
```

print(matrix)

#output: [[1, 0, 1], [0, 0, 0], [1, 0, 1]]

37. You are given two 0-indexed arrays nums1 and nums2 of length n, both of which are permutations of [0, 1, ..., n - 1]. A good triplet is a set of 3 distinct values which are present in increasing order by position both in nums1 and nums2. In other words, if we consider pos1v as the index of the value v in nums1 and pos2v as the index of the value v in nums2, then a good triplet will be a set (x, y, z) where 0 <= x, y, z <= n - 1, such that pos1x < pos1y < pos1z and pos2x < pos2y < pos2z. Return the total number of good triplets.

Example 1: Input: nums1 = [2,0,1,3], nums2 = [0,1,2,3] Output: 1

Explanation: There are 4 triplets (x,y,z) such that pos1x < pos1y < pos1z. They are (2,0,1), (2,0,3), (2,1,3), and (0,1,3). Out of those triplets, only the triplet (0,1,3) satisfies pos2x < pos2y < pos2z. Hence, there is only 1 good triplet.

Program:

```
def count_good_triplets(nums1, nums2):

    n = len(nums1)

    count = 0

        for x in range(n):

        for y in range(x+1, n):

            for z in range(y+1, n):

                if nums1[x] < nums1[y] < nums1[z] and nums2[x] < nums2[y] < nums2[z]:

                    count += 1

    return count

# Example

nums1 = [2, 0, 1, 3]

nums2 = [0, 1, 2, 3]

print(count_good_triplets(nums1, nums2))

# Output: 1
```

38. 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

Program:

```
def intersection(nums1, nums2):

    set1 = set(nums1)

    set2 = set(nums2)

    return list(set1.intersection(set2))

# Test the function with examples

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

print(intersection(nums1, nums2))

# Output: [2]

nums1 = [4, 9, 5]

nums2 = [9, 4, 9, 8, 4]

print(intersection(nums1, nums2))

# Output: [9, 4]
```

38. Given an integer array nums and an integer k, return the kth largest element in the array. Note that it is the kth largest element in the sorted order, not the kth distinct element. Can you solve it without sorting?

Example 1: Input: nums = [3,2,1,5,6,4], k = 2 Output: 5

Example 2: Input: nums = [3,2,3,1,2,4,5,5,6], k = 4 Output: 4

Constraints: 1 <= k <= nums.length <= 105 -104 <= nums[i] <= 104

Program:

```python
import heapq

def findKthLargest(nums, k):

    heap = nums[:k]

    heapq.heapify(heap)

    for num in nums[k:]:

        if num > heap[0]:

            heapq.heappop(heap)

            heapq.heappush(heap, num)

    return heap[0]

# Test the function

nums1 = [3, 2, 1, 5, 6, 4]

k1 = 2

print(findKthLargest(nums1, k1))

# Output: 5
```

39. 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"

Program:

```python
import itertools

def count_good_strings(n, s1, s2, evil):

    MOD = 10**9 + 7

    count = 0

    for perm in itertools.product(range(26), repeat=n):
```

```python
        string = ''.join(chr(ord('a') + x) for x in perm)

        if s1 <= string <= s2 and evil not in string:

            count += 1

    return count % MOD

# Example Usage

n = 2

s1 = "aa"

s2 = "da"

evil = "b"

output = count_good_strings(n, s1, s2, evil)

print(output)

# Output: 51
```

39. Given an array nums of size n, return the majority element.The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array. Example 1: Input: nums = [3,2,3] Output: 3

Example 2: Input: nums = [2,2,1,1,1,2,2] Output: 2

Constraints: n == nums.length 1 <= n <= 5 * 104 -109 <= nums[i] <= 10

Program:

```python
def majority_element(nums):

    counts = {}

    for num in nums:

        counts[num] = counts.get(num, 0) + 1

        if counts[num] > len(nums) // 2:

            return num


# Test the function

nums1 = [3, 2, 3]
```

print(majority_element(nums1))

Output: 3

nums2 = [2, 2, 1, 1, 1, 2, 2]

print(majority_element(nums2))

 # Output: 2

40. 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]]

Program:

```
def transpose_matrix(matrix):

    return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]
```

Test the function

matrix1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(transpose_matrix(matrix1))

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

matrix2 = [[1, 2, 3], [4, 5, 6]]

print(transpose_matrix(matrix2))

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

40. You are given two 0-indexed integer arrays nums1 and nums2, each of size n, and an integer diff. Find the number of pairs (i, j) such that: 0 <= i < j <= n - 1 and nums1[i] - nums1[j] <= nums2[i] - nums2[j] + diff. Return the number of pairs that satisfy the conditions.

Example 1: Input: nums1 = [3,2,5], nums2 = [2,2,1], diff = 1 Output: 3

Explanation: There are 3 pairs that satisfy the conditions: 1. i = 0, j = 1: 3 - 2 <= 2 - 2 + 1. Since i < j and 1 <= 1, this pair satisfies the conditions. 2. i = 0, j = 2: 3 - 5 <= 2 - 1 + 1. Since i < j and -2 <= 2, this pair satisfies the conditions. 3. i = 1, j = 2: 2 - 5 <= 2 - 1 + 1. Since i < j and -3 <= 2, this pair satisfies the conditions. Therefore, we return 3

Program:

```
def count_pairs(nums1, nums2, diff):

    n = len(nums1)
```

```python
    count = 0

    for i in range(n):

        for j in range(i+1, n):

            if nums1[i] - nums1[j] <= nums2[i] - nums2[j] + diff:

                count += 1

    return count
# Example

nums1 = [3, 2, 5]

nums2 = [2, 2, 1]

diff = 1

print(count_pairs(nums1, nums2, diff))

# Output: 3
```