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".

```
def count_strings_with_prefix(words, pref):
    return sum(1 for word in words if word.startswith(pref))

# Example
words = ["pay", "attention", "practice", "attend"]
pref = "at"
output = count_strings_with_prefix(words, pref)
print(output)
```

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: [[""]]

```
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))


# Example 2
strs2 = [""]
print(group_anagrams(strs2))
```

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]]

```
                                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 input
matrix = [[1, 1, 1], [1, 0, 1], [1, 1, 1]]
setZeroes(matrix)
print(matrix)
```

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.

```
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
```

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

```
def intersection(nums1, nums2):
    set1 = set(nums1)
    set2 = set(nums2)
    return list(set1.intersection(set2))
```

```
# Example 1
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersection(nums1, nums2))  # Output: [2]

# Example 2
nums1 = [4, 9, 5]
nums2 = [9, 4, 9, 8, 4]
print(intersection(nums1, nums2))  # Output: [9, 4]
```

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

```
import heapq

def findKthLargest(nums, k):
    return heapq.nlargest(k, nums)[-1]

# Example 1
nums1 = [3, 2, 1, 5, 6, 4]
k1 = 2
output1 = findKthLargest(nums1, k1)
print(output1)  # Output: 5

# Example 2
nums2 = [3, 2, 3, 1, 2, 4, 5, 5, 6]
k2 = 4
output2 = findKthLargest(nums2, k2)
print(output2)  # Output: 4
```

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".

```
def countGoodStrings(n, s1, s2, evil):
    MOD = 10**9 + 7
    dp = [[[0] * 2 for _ in range(len(evil) + 1)] for _ in range(n + 1)]
    dp[0][0][0] = 1
    for i in range(1, n + 1):
        for j in range(len(evil) + 1):
            for k in range(2):
                for x in range(ord(s1[i - 1]) if k == 0 else ord('a'), ord(s2[i - 1]) + 1 if k == 1 else ord('z') + 1):
                    nk = k or x < ord(s2[i - 1])
                    nj = j
                    while nj and evil[nj - 1] != chr(x):
                        nj = dp[nj][0][0]
                    dp[i][nj][nk] += dp[i - 1][j][k]
                    dp[i][nj][nk] %= MOD
    return sum(sum(row) for row in dp[n]) % MOD


# Example Usage
n = 2
s1 = "aa"
s2 = "da"
evil = "b"
output = countGoodStrings(n, s1, s2, evil)
print(output)  # Output: 51
```

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

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

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

iven 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] <= 109

```
from collections import Counter

def majority_element(nums):
    counts = Counter(nums)
    return max(counts, key=counts.get)

# Example 1
nums1 = [3, 2, 3]
print(majority_element(nums1))  # Output: 3

# Example 2
nums2 = [2, 2, 1, 1, 1, 2, 2]
print(majority_element(nums2))  # Output: 2
```