

ASSIGNMENT -4

1. 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 $\text{difference}[i][j] = \text{words}[i][j+1] - \text{words}[i][j]$ where $0 \leq j \leq n - 2$. 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.
Example 1: Input: words = ["adc", "wzy", "abc"] Output: "abc" Explanation: - The difference integer array of "adc" is $[3 - 0, 2 - 3] = [3, -1]$. - The difference integer array of "wzy" is $[25 - 22, 24 - 25] = [3, -1]$. - The difference integer array of "abc" is $[1 - 0, 2 - 1] = [1, 1]$. The odd array out is [1, 1], so we return the corresponding string, "abc".
Example 2: Input: words = ["aaa", "bob", "ccc", "ddd"] Output: "bob" Explanation: All the integer arrays are [0, 0] except for "bob", which corresponds to [13, -13]. Constraints: $3 \leq \text{words.length} \leq 100$, $n = \text{words}[i].\text{length}$, $2 \leq n \leq 20$, words[i] consists of lowercase English letters.

PROGRAM:

```
def find_difference_word(words):

    def calculate_difference(word):
        return [ord(word[i + 1]) - ord(word[i]) for i in range(len(word) - 1)]

    target_array = calculate_difference(words[0])

    for word in words:
        if calculate_difference(word) != target_array:
            return word

words1 = ["adc", "wzy", "abc"]
words2 = ["aaa", "bob", "ccc", "ddd"]

print(find_difference_word(words1))
print(find_difference_word(words2))
```



abc
bob

OUTPUT:

TIME COMPLEXITY: $O(n)$

2. 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.

Example 1: Input: queries = ["word", "note", "ants", "wood"], dictionary = ["wood", "joke", "moat"] Output: ["word", "note", "wood"] Explanation: - Changing the 'r' in "word" to 'o' allows it to equal the dictionary word "wood". - Changing the 'n' to 'j' and the 't' to 'k' in "note" changes it to "joke". - It would take more than 2 edits for "ants" to equal a dictionary word. - "wood" can remain unchanged (0 edits) and match the corresponding dictionary word. Thus, we return ["word", "note", "wood"].

Example 2: Input: queries = ["yes"], dictionary = ["not"] Output: [] Explanation: Applying any two edits to "yes" cannot make it equal to "not". Thus, we return an empty array.

PROGRAM:

```
def find_matching_words(queries, dictionary):
```

```
    def is_match(word1, word2):
```

```
        if word1 == word2:
```

```
            return True
```

```
        if len(word1) != len(word2):
```

```
            return False
```

```
        diff_count = 0
```

```
        for char1, char2 in zip(word1, word2):
```

```
            if char1 != char2:
```

```
                diff_count += 1
```

```
            if diff_count > 2:
```

```
                return False
```

```
        return True
```

```
    result = []
```

```
    for query in queries:
```

```
        if query in dictionary:
```

```
            result.append(query)
```

```
        else:
```

```
            for word in dictionary:
```

```
                if is_match(query, word):
```

```
                    result.append(query)
```

```
                    break
```

```
    return result
```

```
queries1 = ["word", "note", "ants", "wood"]
```

```
dictionary1 = ["wood", "joke", "moat"]
```

```
queries2 = ["yes"]
```

```
dictionary2 = ["not"]
```

```
print(find_matching_words(queries1, dictionary1))
```

```
print(find_matching_words(queries2, dictionary2))
```

```
['word', 'note', 'wood']  
[]
```

OUTPUT:

TIME COMPLEXITY: $O(n^2 * m)$

3. You are given a 0-indexed array `nums` consisting of positive integers, representing targets on a number line. You are also given an integer `space`. You have a machine which can destroy targets. Seeding the machine with some `nums[i]` allows it to destroy all targets with values that can be represented as `nums[i] + c * space`, where `c` is any non-negative integer. You want to destroy the maximum number of targets in `nums`. Return the minimum value of `nums[i]` you can seed the machine with to destroy the maximum number of targets.

Example 1: Input: `nums = [3,7,8,1,1,5]`, `space = 2` Output: 1 Explanation: If we seed the machine with `nums[3]`, then we destroy all targets equal to 1,3,5,7,9,... In this case, we would destroy 5 total targets (all except for `nums[2]`). It is impossible to destroy more than 5 targets, so we return `nums[3]`.

Example 2: Input: `nums = [1,3,5,2,4,6]`, `space = 2` Output: 1 Explanation: Seeding the machine with `nums[0]`, or `nums[3]` destroys 3 targets. It is not possible to destroy more than 3 targets. Since `nums[0]` is the minimal integer that can destroy 3 targets, we return 1

PROGRAM:

```
def max_destroy_targets(nums, space):  
    nums.sort() # Sort the nums array in non-decreasing order  
    max_destroyed = 0  
    result = nums[0] # Initialize the result with the smallest value in nums  
  
    for num in nums:  
        destroyed = 0  
        for target in nums:  
            if abs(target - num) % space == 0:  
                destroyed += 1  
        if destroyed > max_destroyed:  
            max_destroyed = destroyed  
            result = num  
  
    return result  
  
nums1 = [3, 7, 8, 1, 1, 5]  
space1 = 2  
nums2 = [1, 3, 5, 2, 4, 6]  
space2 = 2  
  
print(max_destroy_targets(nums1, space1))  
print(max_destroy_targets(nums2, space2))
```

```
1
1
```

OUTPUT:

TIME COMPLEXITY: $O(n^2)$

4. 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. Return an integer array `answer`, where `answer[i]` is the second greater integer of `nums[i]`.

Example 1: Input: `nums = [2,4,0,9,6]` Output: `[9,6,6,-1,-1]` Explanation: 0th index: 4 is the first integer greater than 2, and 9 is the second integer greater than 2, to the right of 2. 1st index: 9 is the first, and 6 is the second integer greater than 4, to the right of 4. 2nd index: 9 is the first, and 6 is the second integer greater than 0, to the right of 0. 3rd index: There is no integer greater than 9 to its right, so the second greater integer is considered to be -1. 4th index: There is no integer greater than 6 to its right, so the second greater integer is considered to be -1. Thus, we return `[9,6,6,-1,-1]`.

Example 2: Input: `nums = [3,3]` Output: `[-1,-1]` Explanation: We return `[-1,-1]` since neither integer has any integer greater than it

PROGRAM:

```
def secondGreaterElement(nums):
```

```
    n = len(nums)
```

```
    result = [-1] * n
```

```
    stack = []
```

```
    for i in range(n):
```

```
        while stack and nums[i] > nums[stack[-1]]:
```

```
            result[stack.pop()] = nums[i]
```

```
        stack.append(i)
```

```
    return result
```

```
nums1 = [2, 4, 0, 9, 6]
```

```
nums2 = [3, 3]
```

```
print(secondGreaterElement(nums1))
```

```
print(secondGreaterElement(nums2))
```

```
[4, 9, 9, -1, -1]
[-1, -1]
```

OUTPUT:

TIME COMPLEXITY: $O(n)$

5. Average Value of Even Numbers That Are Divisible by Three Given an integer array nums of positive integers, return the average value of all even integers that are divisible by 3. Note that the average of n elements is the sum of the n elements divided by n and rounded down to the nearest integer.

Example 1: Input: nums = [1,3,6,10,12,15] Output: 9 Explanation: 6 and 12 are even numbers that are divisible by 3. $(6 + 12) / 2 = 9$.

Example 2: Input: nums = [1,2,4,7,10] Output: 0 Explanation: There is no single number that satisfies the requirement, so return 0.

PROGRAM:

```
def average_even_divisible_by_three(nums):
```

```
    count = 0
```

```
    total = 0
```

```
    for num in nums:
```

```
        if num % 2 == 0 and num % 3 == 0:
```

```
            total += num
```

```
            count += 1
```

```
    return total // count if count > 0 else 0
```

```
nums1 = [1, 3, 6, 10, 12, 15]
```

```
nums2 = [1, 2, 4, 7, 10]
```

```
print(average_even_divisible_by_three(nums1))
```

```
print(average_even_divisible_by_three(nums2))
```

```
9
0
```

OUTPUT:

TIME COMPLEXITY: $O(n)$

6. 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.

Example 1: Input: n = 16, target = 6 Output: 4 Explanation: Initially n is 16 and its digit sum is $1 + 6 = 7$. After adding 4, n becomes 20 and digit sum becomes $2 + 0 = 2$. It can be shown that we can not make n beautiful with adding non-negative integer less than 4.

Example 2: Input: n = 467, target = 6 Output: 33 Explanation: Initially n is 467 and its digit sum is $4 + 6 + 7 = 17$. After adding 33, n becomes 500 and digit sum becomes $5 + 0 + 0 = 5$. It can be shown that we can not make n beautiful with adding non-negative integer less than 33

PROGRAM:

```
def min_beautiful_number(n, target):
```

```

current_sum = sum(int(digit) for digit in str(n))
x = 0

while current_sum > target:
    x += 1
    current_sum = sum(int(digit) for digit in str(n + x))

return x

```

n1, target1 = 16, 6
n2, target2 = 467, 6

```

print(min_beautiful_number(n1, target1))
print(min_beautiful_number(n2, target2))

```

```

4
33

```

OUTPUT:

TIME COMPLEXITY: $O(\log n)$

7. 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`. Example 1: Input: `nums = [4,2,0,3,1]` Output: 3 Explanation: - Move item 2 to the empty space. Now, `nums = [4,0,2,3,1]`. - Move item 1 to the empty space. Now, `nums = [4,1,2,3,0]`. - Move item 4 to the empty space. Now, `nums = [0,1,2,3,4]`. It can be proven that 3 is the minimum number of operations needed.
- Example 2: Input: `nums = [1,2,3,4,0]` Output: 0 Explanation: `nums` is already sorted so return

PROGRAM:

```

def min_operations_to_sort(nums):
    n = len(nums)
    empty_index = nums.index(0)

    if nums[:empty_index] == list(range(empty_index)):
        return sum(1 for i in range(empty_index, n) if nums[i] != i)

    if nums[empty_index + 1:] == list(range(empty_index + 1, n)):
        return sum(1 for i in range(empty_index) if nums[i] != i)

```

```
# If not sorted at either end, count out-of-place items
out_of_place = sum(1 for i, num in enumerate(nums) if i != num and num != 0)

return out_of_place

nums1 = [4, 2, 0, 3, 1]

print(min_operations_to_sort(nums1))
```

OUTPUT: 3

TIME COMPLEXITY: $O(n \cdot 2^n)$

8. You are given a 0-indexed array `nums` of size `n` consisting of non-negative integers. You need to apply `n - 1` operations to this array where, in the `i`th operation (0-indexed), you will apply the following on the `i`th element of `nums`:

- If `nums[i] == nums[i + 1]`, then multiply `nums[i]` by 2 and set `nums[i + 1]` to 0. Otherwise, you skip this operation. After performing all the operations, shift all the 0's to the end of the array.

- For example, the array `[1,0,2,0,0,1]` after shifting all its 0's to the end, is `[1,2,1,0,0,0]`.

Return the resulting array. Note that the operations are applied sequentially, not all at once.

Example 1: Input: `nums = [1,2,2,1,1,0]` Output: `[1,4,2,0,0,0]` Explanation: We do the following operations: - `i = 0`: `nums[0]` and `nums[1]` are not equal, so we skip this operation. - `i = 1`: `nums[1]` and `nums[2]` are equal, we multiply `nums[1]` by 2 and change `nums[2]` to 0. The array becomes `[1,4,0,1,1,0]`. - `i = 2`: `nums[2]` and `nums[3]` are not equal, so we skip this operation. - `i = 3`: `nums[3]` and `nums[4]` are equal, we multiply `nums[3]` by 2 and change `nums[4]` to 0. The array becomes `[1,4,0,2,0,0]`. - `i = 4`: `nums[4]` and `nums[5]` are equal, we multiply `nums[4]` by 2 and change `nums[5]` to 0. The array becomes `[1,4,0,2,0,0]`. After that, we shift the 0's to the end, which gives the array `[1,4,2,0,0,0]`.

Example 2: Input: `nums = [0,1]` Output: `[1,0]` Explanation: No operation can be applied, we just shift the 0 to the end.

PROGRAM:

```
def shiftZeros(nums):
    n = len(nums)
    for i in range(n - 1):
        if nums[i] == nums[i + 1]:
            nums[i] *= 2
            nums[i + 1] = 0

    zero_index = 0
    for i in range(n):
        if nums[i] != 0:
            nums[zero_index] = nums[i]
            zero_index += 1
```

```
for i in range(zero_index, n):  
    nums[i] = 0
```

```
return nums
```

```
nums1 = [1, 2, 2, 1, 1, 0]
```

```
nums2 = [0, 1]
```

```
print(shiftZeros(nums1))
```

```
print(shiftZeros(nums2))
```

OUTPUT:

```
[1, 4, 2, 0, 0, 0]  
[1, 0]
```

TIME COMPLEXITY: $O(n)$