

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  $0 \leq j \leq n - 2$ . 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

```

main.py  Save Run Output
1 def find_odd_string(words):
2     diff_dict = {}
3
4     for word in words:
5         diff = [ord(word[i + 1]) - ord(word[i]) for i in range(len(word) - 1)]
6         if tuple(diff) in diff_dict:
7             diff_dict[tuple(diff)].append(word)
8         else:
9             diff_dict[tuple(diff)] = [word]
10
11    for key in diff_dict:
12        if len(diff_dict[key]) == 1:
13            return diff_dict[key][0]
14
15 words1 = ["adc", "wzy", "abc"]
16 words2 = ["aaa", "bob", "ccc", "ddd"]
17
18 print(find_odd_string(words1)) # Output: "abc"
19 print(find_odd_string(words2)) # Output: "bob"
20

```

abc  
bob  
=== Code Execution Successful ===

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`. 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. Constraints:  $1 \leq \text{queries.length}, \text{dictionary.length} \leq 100$   $n == \text{queries}[i].\text{length} == \text{dictionary}[j].\text{length}$   $1 \leq n \leq 100$  All `queries[i]` and `dictionary[j]` are composed of lowercase English letters. 2453. **Destroy Sequential Targets** 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  $\text{nums}[i]$  allows it to destroy all targets with values that can be represented as  $\text{nums}[i] + c * \text{space}$ , where  $c$  is any non-negative integer. You want to destroy the maximum number of targets in  $\text{nums}$ . Return the minimum value of  $\text{nums}[i]$  you can seed the machine with to destroy the maximum number of targets. Example 1: Input:  $\text{nums} = [3,7,8,1,1,5]$ ,  $\text{space} = 2$  Output: 1 Explanation: If we seed the machine with  $\text{nums}[3]$ , then we destroy all targets equal to 1,3,5,7,9,... In this case, we would destroy 5 total targets (all except for  $\text{nums}[2]$ ). It is impossible to destroy more than 5 targets, so we return  $\text{nums}[3]$ . Example 2: Input:  $\text{nums} = [1,3,5,2,4,6]$ ,  $\text{space} = 2$  Output: 1 Explanation: Seeding the machine with  $\text{nums}[0]$ , or  $\text{nums}[3]$  destroys 3 targets. It is not possible to destroy more than 3 targets. Since  $\text{nums}[0]$  is the minimal integer that can destroy 3 targets, we return 1. Example 3: Input:  $\text{nums} = [6,2,5]$ ,  $\text{space} = 100$  Output: 2 Explanation: Whatever initial seed we select, we can only destroy 1 target. The minimal seed is  $\text{nums}[1]$ . Constraints:  $1 \leq \text{nums.length} \leq 105$   $1 \leq \text{nums}[i] \leq 109$   $1 \leq \text{space} \leq 109$

```

main.py  Save Run Output Clear
1 def words_within_two_edits(queries, dictionary):
2     def is_within_two_edits(word1, word2):
3         if len(word1) != len(word2):
4             return False
5
6         diff_count = 0
7         for c1, c2 in zip(word1, word2):
8             if c1 != c2:
9                 diff_count += 1
10                if diff_count > 2:
11                    return False
12
13            return True
14
15        result = []
16
17        for query in queries:
18            if query in dictionary:
19                result.append(query)
20                continue
21
22            for word in dictionary:
23                if is_within_two_edits(query, word):
24                    result.append(query)
25                    break
26
27        return result
28
29 def destroy_sequential_targets(nums, space):
30     nums.sort()
31     min_seed = float('inf')
32     max_destroyed = 0
33
34     for num in nums:
35         destroyed = 1 + (num - nums[0]) // space
36         if destroyed > max_destroyed:
37             max_destroyed = destroyed
38             min_seed = num
39
40     return min_seed
41
42 queries1 = ["word", "note", "ants", "wood"]
43 dictionary1 = ["wood", "joke", "moat"]
44 print(words_within_two_edits(queries1, dictionary1))
45 queries2 = ["yes"]
46 dictionary2 = ["not"]
47 print(words_within_two_edits(queries2, dictionary2))
48
49 nums1 = [3, 7, 8, 1, 1, 5]
50 space1 = 2
51 print(destroy_sequential_targets(nums1, space1))
52

```

- 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`. 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,-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. Constraints:  $1 \leq \text{nums.length} \leq 105$   $0 \leq \text{nums}[i] \leq 109$  2455. 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. Constraints:  $1 \leq \text{nums.length} \leq 1000$   $1 \leq \text{nums}[i] \leq 1000$  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. Example 3: Input: n = 1, target = 1 Output: 0 Explanation: Initially n is 1 and its digit sum is 1, which is already smaller than or equal to target. Constraints:  $1 \leq n \leq 1012$   $1 \leq \text{target} \leq 150$  The input will be generated such that it is always possible to make n beautiful.

```

main.py  Save Run Output Clear
1- def nextGreaterElements(nums):
2-     stack = []
3-     n = len(nums)
4-     result = [-1] * n
5-     for i in range(n - 1, -1, -1):
6-         while stack and stack[-1] <= nums[i]:
7-             stack.pop()
8-         if stack:
9-             result[i] = stack[-1]
10-        stack.append(nums[i])
11-    return result
12- def averageOfEvenNumbersDivisibleByThree(nums):
13-     total = 0
14-     count = 0
15-     for num in nums:
16-         if num % 2 == 0 and num % 3 == 0:
17-             total += num
18-             count += 1
19-     if count == 0:
20-         return 0
21-     return total // count
22- nums1 = [2, 4, 0, 9, 6]
23- print(nextGreaterElements(nums1))
24- nums2 = [3, 3]
25- print(nextGreaterElements(nums2))
26- nums3 = [1, 3, 6, 10, 12, 15]

```

Output

```

[4, 9, 9, -1, -1]
[-1, -1]
9
0
=== Code Execution Successful ===

```

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. 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. Example 3: Input:  $n = 1$ , target = 1 Output: 0 Explanation: Initially  $n$  is 1 and its digit sum is 1, which is already smaller than or equal to target. Constraints: •  $1 \leq n \leq 1012$  •  $1 \leq \text{target} \leq 150$  • The input will be generated such that it is always possible to make  $n$  beautiful

```

main.py
1 def min_addition_to_beautiful(n, target):
2     digit_sum = sum(int(digit) for digit in str(n))
3     difference = max(0, target - digit_sum)
4     return difference
5 print(min_addition_to_beautiful(16, 6))
6 print(min_addition_to_beautiful(467, 6))
7 print(min_addition_to_beautiful(1, 1))
8

```

- 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`. 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 0. Example 3: Input: `nums = [1,0,2,4,3]` Output: 2 Explanation: - Move item 2 to the empty space. Now, `nums = [1,2,0,4,3]`. - Move item 3 to the empty space. Now, `nums = [1,2,3,4,0]`. It can be proven that 2 is the minimum number of operations needed. Constraints: •  $n == \text{nums.length}$  •  $2 \leq n \leq 105$  •  $0 \leq \text{nums}[i] < n$  • All the values of `nums` are unique. 2460. Apply Operations to an Array 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. Constraints: •  $2 \leq \text{nums.length} \leq 2000$  •  $0 \leq \text{nums}[i] \leq 1000$

main.py	Save	Run	Output
<pre>1 def sort_array_by_moving_items_to_empty_space(nums): 2     n = len(nums) 3     count = 0 4     for i in range(n): 5         if nums[i] != 0 and nums[i] != i: 6             count += 1 7     return count 8 def apply_operations_to_array(nums): 9     n = len(nums) 10    for i in range(n - 1): 11        if nums[i] == nums[i + 1]: 12            nums[i] *= 2 13            nums[i + 1] = 0 14    non_zero = [x for x in nums if x != 0] 15    zero = [x for x in nums if x == 0] 16    return non_zero + zero 17    nums1 = [4, 2, 0, 3, 1] 18    print(sort_array_by_moving_items_to_empty_space(nums1)) 19    nums2 = [1, 2, 3, 4, 0] 20    print(sort_array_by_moving_items_to_empty_space(nums2)) 21    nums3 = [1, 0, 2, 4, 3] 22    print(sort_array_by_moving_items_to_empty_space(nums3)) 23    nums4 = [1, 2, 2, 1, 1, 0] 24    print(apply_operations_to_array(nums4)) 25    nums5 = [0, 1] 26    print(apply_operations_to_array(nums5))</pre>			<pre>3 4 3 [1, 4, 2, 0, 0, 0] [1, 0]  === Code Execution Successful ===</pre>