Given two integers `num1` and `num2`, return *the **sum** of the two integers*.

Example 1:

```
Input: num1 = 12, num2 = 5
Output: 17
Explanation: num1 is 12, num2 is 5, and their sum is 12 + 5 = 17, so 17 is returned.
```

Example 2:

```
Input: num1 = -10, num2 = 4
Output: -6
Explanation: num1 + num2 = -6, so -6 is returned.
```

Constraints:

- `-100 <= num1, num2 <= 100`

Given a **positive** integer `n`, return *the smallest positive integer that is a multiple of **both** `2` and `n`*.

Example 1:

```
Input: n = 5
Output: 10
Explanation: The smallest multiple of both 5 and 2 is 10.
```

Example 2:

```
Input: n = 6
Output: 6
Explanation: The smallest multiple of both 6 and 2 is 6. Note that a number is a
multiple of itself.
```

Constraints:

- `1 <= n <= 150`

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to `target`*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

Constraints:

- `2 <= nums.length <= `10^4
- `-`10^9` <= nums[i] <= `10^9
- `-`10^9` <= target <= `10^9
- **Only one valid answer exists.**

Given a string `s` consisting of words and spaces, return *the length of the* ***last*** *word in the string.*

A **word** is a maximal substring consisting of non-space characters only.

Example 1:

```
Input: s = "Hello World"
Output: 5
Explanation: The last word is "World" with length 5.
```

Example 2:

```
Input: s = "   fly me   to   the moon  "
Output: 4
Explanation: The last word is "moon" with length 4.
```

Example 3:

```
Input: s = "luffy is still joyboy"
Output: 6
Explanation: The last word is "joyboy" with length 6.
```

Constraints:

- $1 <= s.length <= 10^4$
- `s` consists of only English letters and spaces `' '`.
- There will be at least one word in `s`.

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

```
Symbol       Value
I             1
V             5
X             10
L             50
C             100
D             500
M             1000
```

For example, `2` is written as `II` in Roman numeral, just two ones added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example 1:

```
Input: s = "III"
Output: 3
Explanation: III = 3.
```

Example 2:

```
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

Example 3:

```
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

Constraints:

- `1 <= s.length <= 15`
- `s` contains only the characters `('I', 'V', 'X', 'L', 'C', 'D', 'M')`.
- It is **guaranteed** that `s` is a valid roman numeral in the range `[1, 3999]`.

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Constraints:

- `1 <= s.length <=` 10^4
- `s` consists of parentheses only `'()[]{}'`.

Given an integer `x`, return `true` if `x` is palindrome integer.

An integer is a **palindrome** when it reads the same backward as forward.

- For example, `121` is a palindrome while `123` is not.

Example 1:

```
Input: x = 121
Output: true
Explanation: 121 reads as 121 from left to right and from right to left.
```

Example 2:

```
Input: x = -121
Output: false
Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore
it is not a palindrome.
```

Example 3:

```
Input: x = 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.
```

Constraints:

- $-2^{31} <= x <= 2^{31} - 1$

Given the array `nums`, for each `nums[i]` find out how many numbers in the array are smaller than it. That is, for each `nums[i]` you have to count the number of valid `j's` such that `j != i` **and** `nums[j] < nums[i]`.

Return the answer in an array.

Example 1:

```
Input: nums = [8,1,2,2,3]
Output: [4,0,1,1,3]
Explanation:
For nums[0]=8 there exist four smaller numbers than it (1, 2, 2 and 3).
For nums[1]=1 does not exist any smaller number than it.
For nums[2]=2 there exist one smaller number than it (1).
For nums[3]=2 there exist one smaller number than it (1).
For nums[4]=3 there exist three smaller numbers than it (1, 2 and 2).
```

Example 2:

```
Input: nums = [6,5,4,8]
Output: [2,1,0,3]
```

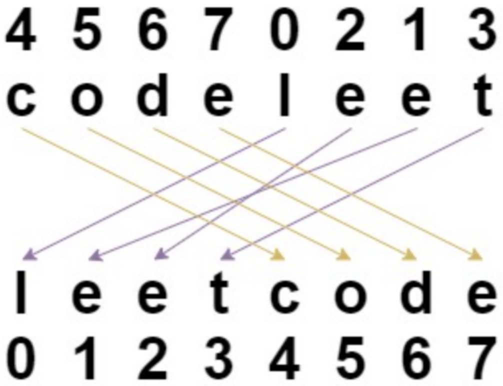
Example 3:

```
Input: nums = [7,7,7,7]
Output: [0,0,0,0]
```

You are given a string `s` and an integer array `indices` of the **same length**. The string `s` will be shuffled such that the character at the `i^th` position moves to `indices[i]` in the shuffled string.

Return *the shuffled string*.

Example 1:

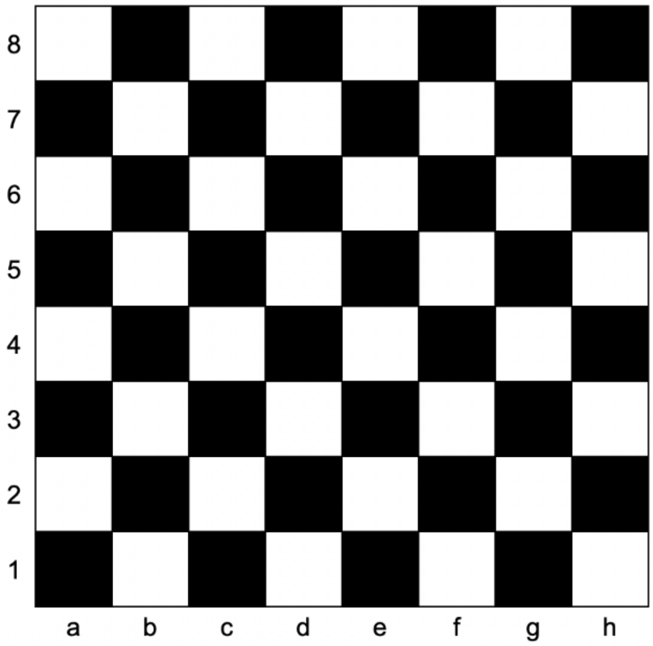


```
Input: s = "codeleet", indices = [4,5,6,7,0,2,1,3]
Output: "leetcode"
Explanation: As shown, "codeleet" becomes "leetcode" after shuffling.
```

Example 2:

```
Input: s = "abc", indices = [0,1,2]
Output: "abc"
Explanation: After shuffling, each character remains in its position.
```

You are given `coordinates`, a string that represents the coordinates of a square of the chessboard. Below is a chessboard for your reference.



Return `true` *if the square is white, and* `false` *if the square is black*.

The coordinate will always represent a valid chessboard square. The coordinate will always have the letter first, and the number second.

Example 1:

```
Input: coordinates = "a1"
Output: false
Explanation: From the chessboard above, the square with coordinates "a1" is black, so return false.
```

Example 2:

```
Input: coordinates = "h3"
Output: true
Explanation: From the chessboard above, the square with coordinates "h3" is white, so return true.
```

A **distinct string** is a string that is present only **once** in an array.

Given an array of strings `arr`, and an integer `k`, return *the* k^{th} ***distinct string*** *present in* `arr`. If there are **fewer** than `k` distinct strings, return *an* ***empty string*** `""`.

Note that the strings are considered in the **order in which they appear** in the array.

Example 1:

```
Input: arr = ["d","b","c","b","c","a"], k = 2
Output: "a"
Explanation:
The only distinct strings in arr are "d" and "a".
"d" appears 1st, so it is the 1st distinct string.
"a" appears 2nd, so it is the 2nd distinct string.
Since k == 2, "a" is returned.
```

Example 2:

```
Input: arr = ["aaa","aa","a"], k = 1
Output: "aaa"
Explanation:
All strings in arr are distinct, so the 1st string "aaa" is returned.
```

Example 3:

```
Input: arr = ["a","b","a"], k = 3
Output: ""
Explanation:
The only distinct string is "b". Since there are fewer than 3 distinct strings, we return an empty string "".
```

There is a malfunctioning keyboard where some letter keys do not work. All other keys on the keyboard work properly.

Given a string `text` of words separated by a single space (no leading or trailing spaces) and a string `brokenLetters` of all **distinct** letter keys that are broken, return *the* ***number of words*** *in* `text` *you can fully type using this keyboard*.

Example 1:

```
Input: text = "hello world", brokenLetters = "ad"
Output: 1
Explanation: We cannot type "world" because the 'd' key is broken.
```

Example 2:

```
Input: text = "leet code", brokenLetters = "lt"
Output: 1
Explanation: We cannot type "leet" because the 'l' and 't' keys are broken.
```

Example 3:

```
Input: text = "leet code", brokenLetters = "e"
Output: 0
Explanation: We cannot type either word because the 'e' key is broken.
```

Constraints:

- `1 <= text.length <= 10`^{4}
- `0 <= brokenLetters.length <= 26`
- `text` consists of words separated by a single space without any leading or trailing spaces.
- Each word only consists of lowercase English letters.
- `brokenLetters` consists of **distinct** lowercase English letters.

You are given a phone number as a string `number`. `number` consists of digits, spaces `' '`, and/or dashes `'-'`.

You would like to reformat the phone number in a certain manner. Firstly, **remove** all spaces and dashes. Then, **group** the digits from left to right into blocks of length 3 **until** there are 4 or fewer digits. The final digits are then grouped as follows:

- 2 digits: A single block of length 2.
- 3 digits: A single block of length 3.
- 4 digits: Two blocks of length 2 each.

The blocks are then joined by dashes. Notice that the reformatting process should **never** produce any blocks of length 1 and produce **at most** two blocks of length 2.

Return *the phone number after formatting.*

Example 1:

```
Input: number = "1-23-45 6"
Output: "123-456"
Explanation: The digits are "123456".
Step 1: There are more than 4 digits, so group the next 3 digits. The 1st block is "123".
Step 2: There are 3 digits remaining, so put them in a single block of length 3. The 2nd block is "456".
Joining the blocks gives "123-456".
```

Example 2:

```
Input: number = "123 4-567"
Output: "123-45-67"
Explanation: The digits are "1234567".
Step 1: There are more than 4 digits, so group the next 3 digits. The 1st block is "123".
Step 2: There are 4 digits left, so split them into two blocks of length 2. The blocks are "45" and "67".
Joining the blocks gives "123-45-67".
```

Example 3:

```
Input: number = "123 4-5678"
Output: "123-456-78"
Explanation: The digits are "12345678".
Step 1: The 1st block is "123".
Step 2: The 2nd block is "456".
Step 3: There are 2 digits left, so put them in a single block of length 2. The 3rd block is "78".
Joining the blocks gives "123-456-78".
```

You are given an integer array `digits`, where each element is a digit. The array may contain duplicates.

You need to find **all** the **unique** integers that follow the given requirements:

- The integer consists of the **concatenation** of **three** elements from `digits` in **any** arbitrary order.
- The integer does not have **leading zeros**.
- The integer is **even**.

For example, if the given `digits` were `[1, 2, 3]`, integers `132` and `312` follow the requirements.

Return *a **sorted** array of the unique integers.*

Example 1:

```
Input: digits = [2,1,3,0]
Output: [102,120,130,132,210,230,302,310,312,320]
Explanation: All the possible integers that follow the requirements are in the output array.
Notice that there are no odd integers or integers with leading zeros.
```

Example 2:

```
Input: digits = [2,2,8,8,2]
Output: [222,228,282,288,822,828,882]
Explanation: The same digit can be used as many times as it appears in digits.
In this example, the digit 8 is used twice each time in 288, 828, and 882.
```

Example 3:

```
Input: digits = [3,7,5]
Output: []
Explanation: No even integers can be formed using the given digits.
```