1.Given an array nums containing n distinct numbers in the range [0, n], return *the only number in the range that is missing from the array.*

**Example 1:**

**Input:** nums = [3,0,1]

**Output:** 2

**Explanation:** n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

**Example 2:**

**Input:** nums = [0,1]

**Output:** 2

**Explanation:** n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

**Example 3:**

**Input:** nums = [9,6,4,2,3,5,7,0,1]

**Output:** 8

**Explanation:** n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

**Constraints:**

* n == nums.length
* 1 <= n <= 104
* 0 <= nums[i] <= n
* All the numbers of nums are **unique**.

2. Given a string s and a character c that occurs in s, return *an array of integers*answer*where*answer.length == s.length*and*answer[i]*is the****distance****from index*i*to the****closest****occurrence of character*c*in*s.

The **distance** between two indices i and j is abs(i - j), where abs is the absolute value function.

**Example 1:**

**Input:** s = "loveleetcode", c = "e"

**Output:** [3,2,1,0,1,0,0,1,2,2,1,0]

**Explanation:** The character 'e' appears at indices 3, 5, 6, and 11 (0-indexed).

The closest occurrence of 'e' for index 0 is at index 3, so the distance is abs(0 - 3) = 3.

The closest occurrence of 'e' for index 1 is at index 3, so the distance is abs(1 - 3) = 2.

For index 4, there is a tie between the 'e' at index 3 and the 'e' at index 5, but the distance is still the same: abs(4 - 3) == abs(4 - 5) = 1.

The closest occurrence of 'e' for index 8 is at index 6, so the distance is abs(8 - 6) = 2.

**Example 2:**

**Input:** s = "aaab", c = "b"

**Output:** [3,2,1,0]

**Constraints:**

* 1 <= s.length <= 104
* s[i] and c are lowercase English letters.
* It is guaranteed that c occurs at least once in s

3. Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

**Example 1:**

**Input:** nums = [-1,2,1,-4], target = 1

**Output:** 2

**Explanation:** The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

**Example 2:**

**Input:** nums = [0,0,0], target = 1

**Output:** 0

**Explanation:** The sum that is closest to the target is 0. (0 + 0 + 0 = 0).

**Constraints:**

* 3 <= nums.length <= 500
* -1000 <= nums[i] <= 1000
* -104 <= target <= 104

4. You are given an n x n 2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm), which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

**Example 1:**



**Input:** matrix = [[1,2,3],[4,5,6],[7,8,9]]

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

**Example 2:**



**Input:** matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]

**Output:** [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

**Constraints:**

* n == matrix.length == matrix[i].length
* 1 <= n <= 20
* -1000 <= matrix[i][j] <= 1000

5. Given an array nums with n objects colored red, white, or blue, sort them [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm)so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

**Example 1:**

**Input:** nums = [2,0,2,1,1,0]

**Output:** [0,0,1,1,2,2]

**Example 2:**

**Input:** nums = [2,0,1]

**Output:** [0,1,2]

**Constraints:**

* n == nums.length
* 1 <= n <= 300
* nums[i] is either 0, 1, or 2.

6. You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return *the****maximum****profit you can achieve*.

**Example 1:**

**Input:** prices = [7,1,5,3,6,4]

**Output:** 7

**Explanation:** Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Total profit is 4 + 3 = 7.

**Example 2:**

**Input:** prices = [1,2,3,4,5]

**Output:** 4

**Explanation:** Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

Total profit is 4.

**Example 3:**

**Input:** prices = [7,6,4,3,1]

**Output:** 0

**Explanation:** There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

**Constraints:**

* 1 <= prices.length <= 3 \* 104
* 0 <= prices[i] <= 104

7. Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.



**Example 1:**

**Input:** digits = "23"

**Output:** ["ad","ae","af","bd","be","bf","cd","ce","cf"]

**Example 2:**

**Input:** digits = ""

**Output:** []

**Example 3:**

**Input:** digits = "2"

**Output:** ["a","b","c"]

**Constraints:**

* 0 <= digits.length <= 4
* digits[i] is a digit in the range ['2', '9'].

8. Given a string s which represents an expression, *evaluate this expression and return its value*.

The integer division should truncate toward zero.

You may assume that the given expression is always valid. All intermediate results will be in the range of [-231, 231 - 1].

**Note:** You are not allowed to use any built-in function which evaluates strings as mathematical expressions, such as eval().

**Example 1:**

**Input:** s = "3+2\*2"

**Output:** 7

**Example 2:**

**Input:** s = " 3/2 "

**Output:** 1

**Example 3:**

**Input:** s = " 3+5 / 2 "

**Output:** 5

**Constraints:**

* 1 <= s.length <= 3 \* 105
* s consists of integers and operators ('+', '-', '\*', '/') separated by some number of spaces.
* s represents **a valid expression**.
* All the integers in the expression are non-negative integers in the range [0, 231 - 1].
* The answer is **guaranteed** to fit in a **32-bit integer**.

9. Given an encoded string, return its decoded string.

The encoding rule is: k[encoded\_string], where the encoded\_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there will not be input like 3a or 2[4].

The test cases are generated so that the length of the output will never exceed 105.

**Example 1:**

**Input:** s = "3[a]2[bc]"

**Output:** "aaabcbc"

**Example 2:**

**Input:** s = "3[a2[c]]"

**Output:** "accaccacc"

**Example 3:**

**Input:** s = "2[abc]3[cd]ef"

**Output:** "abcabccdcdcdef"

**Constraints:**

* 1 <= s.length <= 30
* s consists of lowercase English letters, digits, and square brackets '[]'.
* s is guaranteed to be **a valid** input.
* All the integers in s are in the range [1, 300].

10. In English, we have a concept called **root**, which can be followed by some other word to form another longer word - let's call this word **derivative**. For example, when the **root** "help" is followed by the word "ful", we can form a derivative "helpful".

Given a dictionary consisting of many **roots** and a sentence consisting of words separated by spaces, replace all the derivatives in the sentence with the **root** forming it. If a derivative can be replaced by more than one **root**, replace it with the **root** that has **the shortest length**.

Return *the sentence* after the replacement.

**Example 1:**

**Input:** dictionary = ["cat","bat","rat"], sentence = "the cattle was rattled by the battery"

**Output:** "the cat was rat by the bat"

**Example 2:**

**Input:** dictionary = ["a","b","c"], sentence = "aadsfasf absbs bbab cadsfafs"

**Output:** "a a b c"

**Constraints:**

* 1 <= dictionary.length <= 1000
* 1 <= dictionary[i].length <= 100
* dictionary[i] consists of only lower-case letters.
* 1 <= sentence.length <= 106
* sentence consists of only lower-case letters and spaces.
* The number of words in sentence is in the range [1, 1000]
* The length of each word in sentence is in the range [1, 1000]
* Every two consecutive words in sentence will be separated by exactly one space.
* sentence does not have leading or trailing spaces.