[2870. Minimum Number of Operations to Make Array Empty](https://leetcode.com/problems/minimum-number-of-operations-to-make-array-empty/)

Medium

1.2K

59

Companies

You are given a **0-indexed** array nums consisting of positive integers.

There are two types of operations that you can apply on the array **any** number of times:

* Choose **two** elements with **equal** values and **delete** them from the array.
* Choose **three** elements with **equal** values and **delete** them from the array.

Return *the****minimum****number of operations required to make the array empty, or*-1*if it is not possible*.

**Example 1:**

**Input:** nums = [2,3,3,2,2,4,2,3,4]

**Output:** 4

**Explanation:** We can apply the following operations to make the array empty:

- Apply the first operation on the elements at indices 0 and 3. The resulting array is nums = [3,3,2,4,2,3,4].

- Apply the first operation on the elements at indices 2 and 4. The resulting array is nums = [3,3,4,3,4].

- Apply the second operation on the elements at indices 0, 1, and 3. The resulting array is nums = [4,4].

- Apply the first operation on the elements at indices 0 and 1. The resulting array is nums = [].

It can be shown that we cannot make the array empty in less than 4 operations.

**Example 2:**

**Input:** nums = [2,1,2,2,3,3]

**Output:** -1

**Explanation:** It is impossible to empty the array.

**Constraints:**

* 2 <= nums.length <= 105
* 1 <= nums[i] <= 106

function minOperations($nums) {

        $freq = array\_count\_values($nums);

        $operations = 0;

        foreach ($freq as $count) {

            if($count % 3 == 0){

                $operations += (int) ($count / 3);

            }elseif($count % 2 == 0){

                $operations += (int) ($count / 2);

            }else{

                return -1;

            }

        }

        return $operations;

    }

[300. Longest Increasing Subsequence](https://leetcode.com/problems/longest-increasing-subsequence/)

Given an integer array nums, return *the length of the longest****strictly increasing***

***subsequence***

.

**Example 1:**

**Input:** nums = [10,9,2,5,3,7,101,18]

**Output:** 4

**Explanation:** The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

**Example 2:**

**Input:** nums = [0,1,0,3,2,3]

**Output:** 4

**Example 3:**

**Input:** nums = [7,7,7,7,7,7,7]

**Output:** 1

function lengthOfLIS($nums) {

        if (empty($nums)) {

            return 0;

        }

        $n = count($nums);

        $dp = array\_fill(0, $n, 1);

        for ($i = 1; $i < $n; $i++) {

            for ($j = 0; $j < $i; $j++) {

                if ($nums[$i] > $nums[$j]) {

                    $dp[$i] = max($dp[$i], $dp[$j] + 1);

                }

            }

        }

        return max($dp);

    }

[1. Two Sum](https://leetcode.com/problems/two-sum/)

Easy

54.2K

1.8K

Companies

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 <= 104
* -109 <= nums[i] <= 109
* -109 <= target <= 109
* **Only one valid answer exists.**

**Follow-up:**Can you come up with an algorithm that is less than O(n2) time complexity?

**Answar:**

function twoSum($nums, $target) {

        // $size = count($nums);

        // for($i=0; $i<$size; $i++){

        //     for($j=$i+1; $j<$size; $j++){

        //         if($nums[$i] + $nums[$j] == $target){

        //             return array($i,$j);

        //         }

        //     }

        // }

        $sampArr = [];

        foreach($nums as $key => $num){

            $restNum = $target - $num;

            if (array\_key\_exists($restNum, $sampArr)) {

                return [$sampArr[$restNum], $key];

            }

            $sampArr[$num] = $key;

        }

    }

[**26. Remove Duplicates from Sorted Array**](https://leetcode.com/problems/remove-duplicates-from-sorted-array/)

function removeDuplicates(&$nums) {

        $n = count($nums);

        if ($n == 0) {

            return 0;

        }

        $k = 1;

        for ($i = 1; $i < $n; $i++) {

            if ($nums[$i] != $nums[$i - 1]) {

                $nums[$k++] = $nums[$i];

            }

        }

        return $k;

    }