Problem 3524: Find X Value of Array I

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an array of

positive

integers

nums

, and a

positive

integer

k

.

You are allowed to perform an operation

once

on

nums

, where in each operation you can remove any

non-overlapping

prefix and suffix from

nums

such that

nums

remains

non-empty

.

You need to find the

x-value

of

nums

, which is the number of ways to perform this operation so that the

product

of the remaining elements leaves a

remainder

of

x

when divided by

k

.

Return an array

result

of size

k

where

result[x]

is the

x-value

of

nums

for

0 <= x <= k - 1

.

A

prefix

of an array is a

subarray

that starts from the beginning of the array and extends to any point within it.

A

suffix

of an array is a

subarray

that starts at any point within the array and extends to the end of the array.

Note

that the prefix and suffix to be chosen for the operation can be

empty

.

Example 1:

Input:

nums = [1,2,3,4,5], k = 3

Output:

[9,2,4]

Explanation:

For

x = 0

, the possible operations include all possible ways to remove non-overlapping prefix/suffix that do not remove

nums[2] == 3

.

For

x = 1

, the possible operations are:

Remove the empty prefix and the suffix

[2, 3, 4, 5]

.

nums

becomes

[1]

.

Remove the prefix

[1, 2, 3]

and the suffix

[5]

.

nums

becomes

[4]

.

For

x = 2

, the possible operations are:

Remove the empty prefix and the suffix

[3, 4, 5]

.

nums

becomes

[1, 2]

.

Remove the prefix

[1]

and the suffix

[3, 4, 5]

.

nums

becomes

[2]

.

Remove the prefix

[1, 2, 3]

and the empty suffix.

nums

becomes

[4, 5]

.

Remove the prefix

[1, 2, 3, 4]

and the empty suffix.

nums

becomes

[5]

.

Example 2:

Input:

nums = [1,2,4,8,16,32], k = 4

Output:

[18,1,2,0]

Explanation:

For

x = 0

, the only operations that

do not

result in

x = 0

are:

Remove the empty prefix and the suffix

[4, 8, 16, 32]

.

nums

becomes

[1, 2]

.

Remove the empty prefix and the suffix

[2, 4, 8, 16, 32]

.

nums

becomes

[1]

.

Remove the prefix

[1]

and the suffix

[4, 8, 16, 32]

.

nums

becomes

[2]

.

For

x = 1

, the only possible operation is:

Remove the empty prefix and the suffix

[2, 4, 8, 16, 32]

.

nums

becomes

[1]

.

For

x = 2

, the possible operations are:

Remove the empty prefix and the suffix

[4, 8, 16, 32]

.

nums

becomes

[1, 2]

.

Remove the prefix

[1]

and the suffix

[4, 8, 16, 32]

.

nums

becomes

[2]

.

For

x = 3

, there is no possible way to perform the operation.

Example 3:

Input:

nums = [1,1,2,1,1], k = 2

Output:

[9,6]

Constraints:

1 <= nums[i] <= 10

9

1 <= nums.length <= 10

5

1 <= k <= 5

Code Snippets

C++:

```
class Solution {
public:
```

```
vector<long long> resultArray(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public long[] resultArray(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def resultArray(self, nums: List[int], k: int) -> List[int]:
```

Python:

```
class Solution(object):
def resultArray(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
var resultArray = function(nums, k) {

};
```

TypeScript:

```
function resultArray(nums: number[], k: number): number[] {

};
```

C#:

```
public class Solution {
public long[] ResultArray(int[] nums, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* resultArray(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go:

```
func resultArray(nums []int, k int) []int64 {

}
```

Kotlin:

```
class Solution {
fun resultArray(nums: IntArray, k: Int): LongArray {

}
}
```

Swift:

```
class Solution {
func resultArray(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn result_array(nums: Vec<i32>, k: i32) -> Vec<i64> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def result_array(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function resultArray($nums, $k) {

}
}
```

Dart:

```
class Solution {
List<int> resultArray(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def resultArray(nums: Array[Int], k: Int): Array[Long] = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec result_array(nums :: [integer], k :: integer) :: [integer]
def result_array(nums, k) do

end
end
```

Erlang:

```
-spec result_array(Nums :: [integer()], K :: integer()) -> [integer()].
result_array(Nums, K) ->
.
```

Racket:

```
(define/contract (result-array nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find X Value of Array I
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
```

```
public:
vector<long long> resultArray(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find X Value of Array I
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long[] resultArray(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find X Value of Array I
Difficulty: Medium
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def resultArray(self, nums: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def resultArray(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find X Value of Array I
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
var resultArray = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find X Value of Array I
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
function resultArray(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Find X Value of Array I
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long[] ResultArray(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find X Value of Array I
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* resultArray(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find X Value of Array I
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func resultArray(nums []int, k int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun resultArray(nums: IntArray, k: Int): LongArray {

}
}
```

Swift Solution:

```
class Solution {
func resultArray(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find X Value of Array I
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn result_array(nums: Vec<i32>, k: i32) -> Vec<i64> {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def result_array(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function resultArray($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> resultArray(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def resultArray(nums: Array[Int], k: Int): Array[Long] = {

}
```

```
}
```

Elixir Solution:

```
defmodule Solution do
@spec result_array(nums :: [integer], k :: integer) :: [integer]
def result_array(nums, k) do

end
end
```

Erlang Solution:

```
-spec result_array(Nums :: [integer()], K :: integer()) -> [integer()].
result_array(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (result-array nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```