Problem 3533: Concatenated Divisibility

Problem Information

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

Problem Description

You are given an array of positive integers

nums

and a positive integer

k

.

A

permutation

of

nums

is said to form a

divisible concatenation

if, when you

concatenate

the decimal representations

of the numbers in the order specified by the permutation, the resulting number is

divisible by

k

.

Return the

lexicographically smallest

permutation (when considered as a list of integers) that forms a

divisible concatenation

. If no such permutation exists, return an empty list.

Example 1:

Input:

nums = [3,12,45], k = 5

Output:

[3,12,45]

Explanation:

Permutation

Concatenated Value

Divisible by 5

[3, 12, 45]

31245

Yes

[3, 45, 12]

34512

No

[12, 3, 45]

12345

Yes

[12, 45, 3]

12453

No

[45, 3, 12]

45312

No

[45, 12, 3]

45123

No

The lexicographically smallest permutation that forms a divisible concatenation is

[3,12,45]

.

Example 2:

Input:

nums = [10,5], k = 10

Output:

[5,10]

Explanation:

Permutation

Concatenated Value

Divisible by 10

[5, 10]

510

Yes

[10, 5]

105

No

The lexicographically smallest permutation that forms a divisible concatenation is

[5,10]

.

Example 3:

Input:

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

Output:

[]

Explanation:

Since no permutation of

nums

forms a valid divisible concatenation, return an empty list.

Constraints:

1 <= nums.length <= 13

1 <= nums[i] <= 10

5

1 <= k <= 100

Code Snippets

C++:

```
class Solution {
public:
vector<int> concatenatedDivisibility(vector<int>& nums, int k) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

```
func concatenatedDivisibility(nums []int, k int) []int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
def concatenated_divisibility(nums, k) do

end
end
```

Erlang:

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

Racket:

```
(define/contract (concatenated-divisibility nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Concatenated Divisibility
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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<int> concatenatedDivisibility(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Concatenated Divisibility
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 int[] concatenatedDivisibility(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Concatenated Divisibility
Difficulty: Hard
Tags: array, graph, dp

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Concatenated Divisibility
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 concatenatedDivisibility = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Concatenated Divisibility
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 concatenatedDivisibility(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Concatenated Divisibility
* Difficulty: Hard
```

```
* Tags: array, graph, dp
*
* 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 int[] ConcatenatedDivisibility(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Concatenated Divisibility
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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().
*/
int* concatenatedDivisibility(int* nums, int numsSize, int k, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Concatenated Divisibility
// Difficulty: Hard
// Tags: array, graph, dp
//
// 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 concatenatedDivisibility(nums []int, k int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Concatenated Divisibility
// Difficulty: Hard
// Tags: array, graph, dp
//
// 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 concatenated_divisibility(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (concatenated-divisibility nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```