Problem 2425: Bitwise XOR of All Pairings

Problem Information

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

Problem Description

You are given two

0-indexed

arrays,

nums1

and

nums2

, consisting of non-negative integers. Let there be another array,

nums3

, which contains the bitwise XOR of

all pairings

of integers between

nums1

and

nums2

(every integer in

nums1

is paired with every integer in

nums2

exactly once

).

Return

the

bitwise XOR

of all integers in

nums3

.

Example 1:

Input:

nums1 = [2,1,3], nums2 = [10,2,5,0]

Output:

13

Explanation:

A possible nums3 array is [8,0,7,2,11,3,4,1,9,1,6,3]. The bitwise XOR of all these numbers is 13, so we return 13.

Example 2:

Input:

nums1 = [1,2], nums2 = [3,4]

Output:

0

Explanation:

All possible pairs of bitwise XORs are nums1[0] ^ nums2[0], nums1[0] ^ nums2[1], nums1[1] ^ nums2[0], and nums1[1] ^ nums2[1]. Thus, one possible nums3 array is [2,5,1,6]. 2 ^ 5 ^ 1 ^ 6 = 0, so we return 0.

Constraints:

1 <= nums1.length, nums2.length <= 10

5

0 <= nums1[i], nums2[j] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int xorAllNums(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public int xorAllNums(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def xorAllNums(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
def xorAllNums(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var xorAllNums = function(nums1, nums2) {

};
```

TypeScript:

```
function xorAllNums(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public int XorAllNums(int[] nums1, int[] nums2) {

}
}
```

C:

```
int xorAllNums(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go:

```
func xorAllNums(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun xorAllNums(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func xorAllNums(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn xor_all_nums(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def xor_all_nums(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function xorAllNums($nums1, $nums2) {

}
}
```

Dart:

```
class Solution {
int xorAllNums(List<int> nums1, List<int> nums2) {

}
}
```

Scala:

```
object Solution {
def xorAllNums(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec xor_all_nums(nums1 :: [integer], nums2 :: [integer]) :: integer
def xor_all_nums(nums1, nums2) do
```

```
end
end
```

Erlang:

```
-spec xor_all_nums(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
xor_all_nums(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (xor-all-nums nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Bitwise XOR of All Pairings
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int xorAllNums(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Bitwise XOR of All Pairings
```

```
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int xorAllNums(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Bitwise XOR of All Pairings
Difficulty: Medium
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def xorAllNums(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def xorAllNums(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Bitwise XOR of All Pairings
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var xorAllNums = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Bitwise XOR of All Pairings
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function xorAllNums(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Bitwise XOR of All Pairings
* Difficulty: Medium
* Tags: array
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int XorAllNums(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Bitwise XOR of All Pairings
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int xorAllNums(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Bitwise XOR of All Pairings
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func xorAllNums(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun xorAllNums(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func xorAllNums(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Bitwise XOR of All Pairings
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn xor_all_nums(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def xor_all_nums(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function xorAllNums($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
int xorAllNums(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def xorAllNums(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec xor_all_nums(nums1 :: [integer], nums2 :: [integer]) :: integer
def xor_all_nums(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec xor_all_nums(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
xor_all_nums(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (xor-all-nums nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```