Problem 1879: Minimum XOR Sum of Two Arrays

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

of length

n

.

The

XOR sum

of the two integer arrays is

(nums1[0] XOR nums2[0]) + (nums1[1] XOR nums2[1]) + ... + (nums1[n - 1] XOR nums2[n - 1])

(

0-indexed

).

For example, the

XOR sum

of

[1,2,3]

and

[3,2,1]

is equal to

(1 XOR 3) + (2 XOR 2) + (3 XOR 1) = 2 + 0 + 2 = 4

.

Rearrange the elements of

nums2

such that the resulting

XOR sum

is

minimized

.

Return

the

XOR sum

after the rearrangement

.

Example 1:

Input:

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

Output:

2

Explanation:

Rearrange

nums2

so that it becomes

[3,2]

. The XOR sum is (1 XOR 3) + (2 XOR 2) = 2 + 0 = 2.

Example 2:

Input:

nums1 = [1,0,3], nums2 = [5,3,4]

Output:

8

Explanation:

Rearrange

nums2

so that it becomes

[5,4,3]

. The XOR sum is (1 XOR 5) + (0 XOR 4) + (3 XOR 3) = 4 + 4 + 0 = 8.

Constraints:

n == nums1.length

n == nums2.length

1 <= n <= 14

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

7

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
```

```
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
```

```
# @return {Integer}
def minimum_xor_sum(nums1, nums2)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
```

```
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-xor-sum nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum XOR Sum of Two Arrays
* Difficulty: Hard
* Tags: array, 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 minimumXORSum(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum XOR Sum of Two Arrays
```

```
* Difficulty: Hard
* Tags: array, 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 minimumXORSum(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum XOR Sum of Two Arrays
Difficulty: Hard
Tags: array, 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 minimumXORSum(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum XOR Sum of Two Arrays
* Difficulty: Hard
* Tags: array, 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[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minimumXORSum = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum XOR Sum of Two Arrays
* Difficulty: Hard
* Tags: array, 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 minimumXORSum(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum XOR Sum of Two Arrays
* Difficulty: Hard
* Tags: array, 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 MinimumXORSum(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Minimum XOR Sum of Two Arrays
* Difficulty: Hard
* Tags: array, 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
*/

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

}
```

Go Solution:

```
// Problem: Minimum XOR Sum of Two Arrays
// Difficulty: Hard
// Tags: array, 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 minimumXORSum(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum XOR Sum of Two Arrays
// Difficulty: Hard
// Tags: array, 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 minimum_xor_sum(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-xor-sum nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```