Problem 3630: Partition Array for Maximum XOR and AND

Problem Information

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

Problem Description

You are given an integer array

nums

.

Partition the array into

three

(possibly empty)

subsequences

A

,

B

, and

C

such that every element of

nums

belongs to

exactly

one subsequence.

Your goal is to

maximize

the value of:

XOR(A) + AND(B) + XOR(C)

where:

XOR(arr)

denotes the bitwise XOR of all elements in

arr

. If

arr

is empty, its value is defined as 0.

AND(arr)

denotes the bitwise AND of all elements in

arr

. If

arr

is empty, its value is defined as 0.

Return the

maximum

value achievable.

Note:

If multiple partitions result in the same

maximum

sum, you can consider any one of them.

Example 1:

Input:

nums = [2,3]

Output:

5

Explanation:

One optimal partition is:

A = [3], XOR(A) = 3

B = [2], AND(B) = 2

C = [], XOR(C) = 0

The maximum value of:

XOR(A) + AND(B) + XOR(C) = 3 + 2 + 0 = 5

. Thus, the answer is 5.

Example 2:

Input:

nums = [1,3,2]

Output:

6

Explanation:

One optimal partition is:

A = [1], XOR(A) = 1

B = [2], AND(B) = 2

C = [3], XOR(C) = 3

The maximum value of:

XOR(A) + AND(B) + XOR(C) = 1 + 2 + 3 = 6

. Thus, the answer is 6.

Example 3:

Input:

nums = [2,3,6,7]

Output:

15

Explanation:

One optimal partition is:

A = [7], XOR(A) = 7

B = [2,3], AND(B) = 2

C = [6], XOR(C) = 6

The maximum value of:

XOR(A) + AND(B) + XOR(C) = 7 + 2 + 6 = 15

. Thus, the answer is 15.

Constraints:

1 <= nums.length <= 19

1 <= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long maximizeXorAndXor(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public long maximizeXorAndXor(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximizeXorAndXor(nums: number[]): number {

};
```

C#:

```
public class Solution {
public long MaximizeXorAndXor(int[] nums) {

}
}
```

C:

```
long long maximizeXorAndXor(int* nums, int numsSize) {

}
```

Go:

```
func maximizeXorAndXor(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximizeXorAndXor(nums: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func maximizeXorAndXor(_ nums: [Int]) -> Int {

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def maximize_xor_and_xor(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maximizeXorAndXor($nums) {

}
}
```

Dart:

```
class Solution {
int maximizeXorAndXor(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def maximizeXorAndXor(nums: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximize_xor_and_xor(nums :: [integer]) :: integer
def maximize_xor_and_xor(nums) do

end
end
```

Erlang:

```
-spec maximize_xor_and_xor(Nums :: [integer()]) -> integer().
maximize_xor_and_xor(Nums) ->
.
```

Racket:

```
(define/contract (maximize-xor-and-xor nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Partition Array for Maximum XOR and AND
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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:
long long maximizeXorAndXor(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Partition Array for Maximum XOR and AND
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 long maximizeXorAndXor(int[] nums) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Partition Array for Maximum XOR and AND
Difficulty: Hard
Tags: array, greedy, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition Array for Maximum XOR and AND
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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[]} nums
* @return {number}
*/
var maximizeXorAndXor = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition Array for Maximum XOR and AND
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 maximizeXorAndXor(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Partition Array for Maximum XOR and AND
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 long MaximizeXorAndXor(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Partition Array for Maximum XOR and AND
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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
*/

long long maximizeXorAndXor(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Partition Array for Maximum XOR and AND
// Difficulty: Hard
// Tags: array, greedy, math
//
// 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 maximizeXorAndXor(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximizeXorAndXor(nums: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func maximizeXorAndXor(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Partition Array for Maximum XOR and AND
// Difficulty: Hard
// Tags: array, greedy, math
//
// 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 maximize_xor_and_xor(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def maximize_xor_and_xor(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maximizeXorAndXor($nums) {

}
}
```

Dart Solution:

```
class Solution {
int maximizeXorAndXor(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def maximizeXorAndXor(nums: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximize_xor_and_xor(nums :: [integer]) :: integer
def maximize_xor_and_xor(nums) do

end
end
```

Erlang Solution:

```
-spec maximize_xor_and_xor(Nums :: [integer()]) -> integer().
maximize_xor_and_xor(Nums) ->
.
```

Racket Solution:

```
(define/contract (maximize-xor-and-xor nums)
(-> (listof exact-integer?) exact-integer?)
)
```