Problem 1863: Sum of All Subset XOR Totals

Problem Information

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

Problem Description

The

XOR total

of an array is defined as the bitwise

XOR

of

all its elements

, or

0

if the array is

empty

.

For example, the

XOR total

of the array

[2,5,6]

is

2 XOR 5 XOR 6 = 1

.

Given an array

nums

, return

the

sum

of all

XOR totals

for every

subset

of

nums

.

Note:

Subsets with the

same

elements should be counted

multiple

times.

An array

a

is a

subset

of an array

b

if

a

can be obtained from

b

by deleting some (possibly zero) elements of

b

.

Example 1:

Input:

nums = [1,3]

Output:

6

Explanation:

The 4 subsets of [1,3] are: - The empty subset has an XOR total of 0. - [1] has an XOR total of 1. - [3] has an XOR total of 3. - [1,3] has an XOR total of 1 XOR 3 = 2. 0 + 1 + 3 + 2 = 6

Example 2:

Input:

nums = [5,1,6]

Output:

28

Explanation:

The 8 subsets of [5,1,6] are: - The empty subset has an XOR total of 0. - [5] has an XOR total of 5. - [1] has an XOR total of 1. - [6] has an XOR total of 6. - [5,1] has an XOR total of 5 XOR 1 = 4. - [5,6] has an XOR total of 5 XOR 6 = 3. - [1,6] has an XOR total of 1 XOR 6 = 7. - [5,1,6] has an XOR total of 5 XOR 1 XOR 6 = 2. 0 + 5 + 1 + 6 + 4 + 3 + 7 + 2 = 28

Example 3:

Input:

nums = [3,4,5,6,7,8]

Output:

480

Explanation:

The sum of all XOR totals for every subset is 480.

Constraints:

1 <= nums.length <= 12

1 <= nums[i] <= 20

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int subsetXORSum(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int SubsetXORSum(int[] nums) {

}
}
```

C:

```
int subsetXORSum(int* nums, int numsSize) {

}
```

Go:

```
func subsetXORSum(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun subsetXORSum(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (subset-xor-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of All Subset XOR Totals
* Difficulty: Easy
* Tags: array, 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:
int subsetXORSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of All Subset XOR Totals
* Difficulty: Easy
* Tags: array, 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 int subsetXORSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of All Subset XOR Totals
Difficulty: Easy
Tags: array, 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 subsetXORSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of All Subset XOR Totals
* Difficulty: Easy
* Tags: array, 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 subsetXORSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of All Subset XOR Totals
* Difficulty: Easy
* Tags: array, 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 subsetXORSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of All Subset XOR Totals
* Difficulty: Easy
* Tags: array, 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 int SubsetXORSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of All Subset XOR Totals
* Difficulty: Easy
* Tags: array, 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
*/

int subsetXORSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of All Subset XOR Totals
// Difficulty: Easy
// Tags: array, 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 subsetXORSum(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun subsetXORSum(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of All Subset XOR Totals
// Difficulty: Easy
// Tags: array, 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 subset_xor_sum(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (subset-xor-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```