Problem 2935: Maximum Strong Pair XOR II

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. A pair of integers

x

and

y

is called a

strong

pair if it satisfies the condition:

|x - y| <= min(x, y)

You need to select two integers from

nums

such that they form a strong pair and their bitwise

XOR

is the

maximum

among all strong pairs in the array.

Return

the

maximum

XOR

value out of all possible strong pairs in the array

nums

.

Note

that you can pick the same integer twice to form a pair.

Example 1:

Input:

nums = [1,2,3,4,5]

Output:

7

Explanation:

There are 11 strong pairs in the array

nums

: (1, 1), (1, 2), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (3, 5), (4, 4), (4, 5) and (5, 5). The maximum XOR possible from these pairs is 3 XOR 4 = 7.

Example 2:

Input:

nums = [10,100]

Output:

0

Explanation:

There are 2 strong pairs in the array nums: (10, 10) and (100, 100). The maximum XOR possible from these pairs is 10 XOR 10 = 0 since the pair (100, 100) also gives 100 XOR 100 = 0.

Example 3:

Input:

nums = [500,520,2500,3000]

Output:

1020

Explanation:

There are 6 strong pairs in the array nums: (500, 500), (500, 520), (520, 520), (2500, 2500), (2500, 3000) and (3000, 3000). The maximum XOR possible from these pairs is 500 XOR 520

= 1020 since the only other non-zero XOR value is 2500 XOR 3000 = 636.

Constraints:

1 <= nums.length <= 5 * 10

4

1 <= nums[i] <= 2

20

- 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-strong-pair-xor nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Strong Pair XOR II
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Maximum Strong Pair XOR II
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int maximumStrongPairXor(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Strong Pair XOR II
Difficulty: Hard
Tags: array, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Strong Pair XOR II
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Strong Pair XOR II
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function maximumStrongPairXor(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Strong Pair XOR II
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MaximumStrongPairXor(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Strong Pair XOR II
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int maximumStrongPairXor(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Strong Pair XOR II
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maximumStrongPairXor(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Strong Pair XOR II
// Difficulty: Hard
// Tags: array, hash
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-strong-pair-xor nums)
(-> (listof exact-integer?) exact-integer?)
)
```