Problem 2317: Maximum XOR After Operations

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. In one operation, select

any

non-negative integer

x

and an index

i

, then

update

nums[i]

to be equal to

nums[i] AND (nums[i] XOR x)

.

Note that

AND

is the bitwise AND operation and

XOR

is the bitwise XOR operation.

Return

the

maximum

possible bitwise XOR of all elements of

nums

after applying the operation

any number

of times

.

Example 1:

Input:

nums = [3,2,4,6]

Output:

7

Explanation:

Apply the operation with x = 4 and i = 3, num[3] = 6 AND (6 XOR 4) = 6 AND 2 = 2. Now, nums = [3, 2, 4, 2] and the bitwise XOR of all the elements = 3 XOR 2 XOR 4 XOR 2 = 7. It can be shown that 7 is the maximum possible bitwise XOR. Note that other operations may be used to achieve a bitwise XOR of 7.

Example 2:

Input:

nums = [1,2,3,9,2]

Output:

11

Explanation:

Apply the operation zero times. The bitwise XOR of all the elements = 1 XOR 2 XOR 3 XOR 9 XOR 2 = 11. It can be shown that 11 is the maximum possible bitwise XOR.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

8

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum XOR After Operations
* Difficulty: Medium
* 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 maximumXOR(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum XOR After Operations
* Difficulty: Medium
* 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 maximumXOR(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum XOR After Operations
Difficulty: Medium
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 maximumXOR(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum XOR After Operations
* Difficulty: Medium
* 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 maximumXOR = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum XOR After Operations
* Difficulty: Medium
* 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 maximumXOR(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum XOR After Operations
* Difficulty: Medium
* 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 MaximumXOR(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum XOR After Operations
* Difficulty: Medium
* 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 maximumXOR(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum XOR After Operations
// Difficulty: Medium
// 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 maximumXOR(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum XOR After Operations
// Difficulty: Medium
// 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 maximum_xor(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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