Problem 1707: Maximum XOR With an Element From Array

Problem Information

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

Problem Description

You are given an array

nums

consisting of non-negative integers. You are also given a

queries

array, where

queries[i] = [x

i

, m

i

]

.

The answer to the

i

th

query is the maximum bitwise

XOR

value of

x

i

and any element of

nums

that does not exceed

m

i

. In other words, the answer is

max(nums[j] XOR x

i

)

for all

j

such that

nums[j] <= m

i

. If all elements in

nums

are larger than

m

i

, then the answer is

-1

.

Return

an integer array

answer

where

answer.length == queries.length

and

answer[i]

is the answer to the

i

th

query.

Example 1:

Input:

nums = [0,1,2,3,4], queries = [[3,1],[1,3],[5,6]]

Output:

[3,3,7]

Explanation:

1) 0 and 1 are the only two integers not greater than 1. 0 XOR 3 = 3 and 1 XOR 3 = 2. The larger of the two is 3. 2) 1 XOR 2 = 3. 3) 5 XOR 2 = 7.

Example 2:

Input:

nums = [5,2,4,6,6,3], queries = [[12,4],[8,1],[6,3]]

Output:

[15,-1,5]

Constraints:

1 <= nums.length, queries.length <= 10

5

queries[i].length == 2

0 <= nums[j], x

i

, m

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> maximizeXor(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] maximizeXor(int[] nums, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximizeXor(nums: number[], queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] MaximizeXor(int[] nums, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximizeXor(int* nums, int numsSize, int** queries, int queriesSize,
int* queriesColSize, int* returnSize) {

}
```

Go:

```
func maximizeXor(nums []int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun maximizeXor(nums: IntArray, queries: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func maximizeXor(_ nums: [Int], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn maximize_xor(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def maximize_xor(nums, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function maximizeXor($nums, $queries) {

}

}
```

Dart:

```
class Solution {
List<int> maximizeXor(List<int> nums, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def maximizeXor(nums: Array[Int], queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximize_xor(nums :: [integer], queries :: [[integer]]) :: [integer]
def maximize_xor(nums, queries) do

end
end
```

Erlang:

```
-spec maximize_xor(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
maximize_xor(Nums, Queries) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum XOR With an Element From Array
* Difficulty: Hard
* Tags: array
*
* 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:
vector<int> maximizeXor(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum XOR With an Element From Array
* Difficulty: Hard
* Tags: array
*
* 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[] maximizeXor(int[] nums, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum XOR With an Element From Array
```

```
Difficulty: Hard
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum XOR With an Element From Array
* Difficulty: Hard
* Tags: array
*
* 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
* @param {number[][]} queries
* @return {number[]}
*/
```

```
var maximizeXor = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum XOR With an Element From Array
* Difficulty: Hard
* Tags: array
*
* 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 maximizeXor(nums: number[], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum XOR With an Element From Array
* Difficulty: Hard
* Tags: array
*
* 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[] MaximizeXor(int[] nums, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Maximum XOR With an Element From Array
* Difficulty: Hard
* Tags: array
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximizeXor(int* nums, int numsSize, int** queries, int queriesSize,
int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum XOR With an Element From Array
// Difficulty: Hard
// Tags: array
//
// 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 maximizeXor(nums []int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maximizeXor(nums: IntArray, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maximizeXor(_ nums: [Int], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Maximum XOR With an Element From Array
// Difficulty: Hard
// Tags: array
//
// 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(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def maximize_xor(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function maximizeXor($nums, $queries) {
}
}
```

Dart Solution:

```
class Solution {
List<int> maximizeXor(List<int> nums, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def maximizeXor(nums: Array[Int], queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximize_xor(nums :: [integer], queries :: [[integer]]) :: [integer]
def maximize_xor(nums, queries) do

end
end
```

Erlang Solution:

```
-spec maximize_xor(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
maximize_xor(Nums, Queries) ->
.
```

Racket Solution:

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