Problem 3681: Maximum XOR of Subsequences

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

where each element is a non-negative integer.

Select

two

subsequences

of

nums

(they may be empty and are

allowed

to

overlap

), each preserving the original order of elements, and let:

X

be the bitwise XOR of all elements in the first subsequence.

Y

be the bitwise XOR of all elements in the second subsequence.

Return the

maximum

possible value of

X XOR Y

.

Note:

The XOR of an

empty

subsequence is 0.

Example 1:

Input:

nums = [1,2,3]

Output:

3

Explanation:

Choose subsequences:

First subsequence

[2]

, whose XOR is 2.

Second subsequence

[2,3]

, whose XOR is 1.

Then, XOR of both subsequences =

2 XOR 1 = 3

.

This is the maximum XOR value achievable from any two subsequences.

Example 2:

Input:

nums = [5,2]

Output:

7

Explanation:

Choose subsequences:

First subsequence

[5]

, whose XOR is 5.

Second subsequence

[2]

, whose XOR is 2.

Then, XOR of both subsequences =

5 XOR 2 = 7

.

This is the maximum XOR value achievable from any two subsequences.

Constraints:

2 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum XOR of Subsequences
* 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:
int maxXorSubsequences(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum XOR of Subsequences
* 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 int maxXorSubsequences(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum XOR of Subsequences
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 maxXorSubsequences(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum XOR of Subsequences
* 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 maxXorSubsequences = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum XOR of Subsequences
* 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 maxXorSubsequences(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum XOR of Subsequences
* 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 int MaxXorSubsequences(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum XOR of Subsequences
* 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
*/

int maxXorSubsequences(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum XOR of Subsequences
// 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 maxXorSubsequences(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum XOR of Subsequences
// 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 max_xor_subsequences(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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