# Problem 3277: Maximum XOR Score Subarray Queries

## Problem Information

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

## Problem Description

You are given an array

nums

of

n

integers, and a 2D integer array

queries

of size

q

, where

queries[i] = [l

i

, r

i

]

.

For each query, you must find the

maximum XOR score

of any

subarray

of

nums[l

i

..r

i

]

.

The

XOR score

of an array

a

is found by repeatedly applying the following operations on

a

so that only one element remains, that is the

score

:

Simultaneously replace

a[i]

with

a[i] XOR a[i + 1]

for all indices

i

except the last one.

Remove the last element of

a

.

Return an array

answer

of size

q

where

answer[i]

is the answer to query

i

.

Example 1:

Input:

nums = [2,8,4,32,16,1], queries = [[0,2],[1,4],[0,5]]

Output:

[12,60,60]

Explanation:

In the first query,

nums[0..2]

has 6 subarrays

[2]

,

[8]

,

[4]

,

[2, 8]

,

[8, 4]

, and

[2, 8, 4]

each with a respective XOR score of 2, 8, 4, 10, 12, and 6. The answer for the query is 12, the largest of all XOR scores.

In the second query, the subarray of

nums[1..4]

with the largest XOR score is

nums[1..4]

with a score of 60.

In the third query, the subarray of

nums[0..5]

with the largest XOR score is

nums[1..4]

with a score of 60.

Example 2:

Input:

nums = [0,7,3,2,8,5,1], queries = [[0,3],[1,5],[2,4],[2,6],[5,6]]

Output:

[7,14,11,14,5]

Explanation:

Index

nums[l

i

..r

i

]

Maximum XOR Score Subarray

Maximum Subarray XOR Score

0

[0, 7, 3, 2]

[7]

7

1

[7, 3, 2, 8, 5]

[7, 3, 2, 8]

14

2

[3, 2, 8]

[3, 2, 8]

11

3

[3, 2, 8, 5, 1]

[2, 8, 5, 1]

14

4

[5, 1]

[5]

5

Constraints:

1 <= n == nums.length <= 2000

0 <= nums[i] <= 2

31

- 1

1 <= q == queries.length <= 10

5

queries[i].length == 2

queries[i] = [l

i

, r

i

]

0 <= l

i

<= r

i

<= n - 1

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

```
List[int]:
```

**Python:**

```
class Solution(object):
def maximumSubarrayXor(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 maximumSubarrayXor = function(nums, queries) {

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximumSubarrayXor(int* nums, int numsSize, int** queries, int
```

```
queriesSize, int* queriesColSize, int* returnSize) {

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

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

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum XOR Score Subarray Queries
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

### Java Solution:

```
/**
* Problem: Maximum XOR Score Subarray Queries
* Difficulty: Hard
* Tags: array, dp
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

### Python3 Solution:

```
"""
Problem: Maximum XOR Score Subarray Queries
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Maximum XOR Score Subarray Queries
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

### TypeScript Solution:

```
/**
* Problem: Maximum XOR Score Subarray Queries
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maximumSubarrayXor(nums: number[], queries: number[][]): number[] {

};
```

### C# Solution:

```
/*
* Problem: Maximum XOR Score Subarray Queries
* Difficulty: Hard
* Tags: array, dp
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

### C Solution:

```
/*
* Problem: Maximum XOR Score Subarray Queries
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

### Go Solution:

```
// Problem: Maximum XOR Score Subarray Queries
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

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

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Maximum XOR Score Subarray Queries
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

### Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
```

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

end
```

**PHP Solution:**

```
class Solution {

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

}
}
```

**Dart Solution:**

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

}
}
```

**Scala Solution:**

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

}
}
```

**Elixir Solution:**

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

```
def maximum_subarray_xor(nums, queries) do

end
end
```

### Erlang Solution:

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

### Racket Solution:

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