

# Problem 3653: XOR After Range Multiplication Queries I

## Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given an integer array

nums

of length

n

and a 2D integer array

queries

of size

q

, where

$\text{queries}[i] = [l, r]$

i

, r

i

, k

i

, v

i

]

For each query, you must apply the following operations in order:

Set

idx = l

i

While

idx <= r

i

:

Update:

nums[idx] = (nums[idx] \* v

i

) % (10

9

+ 7)

Set

idx += k

i

Return the

bitwise XOR

of all elements in

nums

after processing all queries.

Example 1:

Input:

nums = [1,1,1], queries = [[0,2,1,4]]

Output:

4

Explanation:

A single query

[0, 2, 1, 4]

multiplies every element from index 0 through index 2 by 4.

The array changes from

[1, 1, 1]

to

[4, 4, 4]

The XOR of all elements is

$4 \wedge 4 \wedge 4 = 4$

Example 2:

Input:

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

Output:

31

Explanation:

The first query

[1, 4, 2, 3]

multiplies the elements at indices 1 and 3 by 3, transforming the array to

[2, 9, 1, 15, 4]

The second query

[0, 2, 1, 2]

multiplies the elements at indices 0, 1, and 2 by 2, resulting in

[4, 18, 2, 15, 4]

Finally, the XOR of all elements is

$4 \wedge 18 \wedge 2 \wedge 15 \wedge 4 = 31$

Constraints:

$1 \leq n == \text{nums.length} \leq 10$

3

$1 \leq \text{nums}[i] \leq 10$

9

$1 \leq q == \text{queries.length} \leq 10$

3

$\text{queries}[i] = [l$

i

, r

i

, k

i

, v

i

]

0 <= l

i

<= r

i

< n

1 <= k

i

<= n

1 <= v

i

<= 10

5

## Code Snippets

C++:

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

### Java:

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

### Python3:

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

### Python:

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


```

### JavaScript:

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

}
```

### TypeScript:

```
function xorAfterQueries(nums: number[], queries: number[][]): number {  
    };
```

### C#:

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

### C:

```
int xorAfterQueries(int* nums, int numsSize, int** queries, int queriesSize,  
int* queriesColSize) {  
}
```

### Go:

```
func xorAfterQueries(nums []int, queries [][]int) int {  
}
```

### Kotlin:

```
class Solution {  
    fun xorAfterQueries(nums: IntArray, queries: Array<IntArray>): Int {  
        }  
    }
```

### Swift:

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

### Rust:

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

### Ruby:

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

end
```

### PHP:

```
class Solution {

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

    }
}
```

### Dart:

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

### Scala:

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

```
}
```

### Elixir:

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

### Erlang:

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

### Racket:

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

## Solutions

### C++ Solution:

```
/*
 * Problem: XOR After Range Multiplication Queries I
 * Difficulty: Medium
 * 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 xorAfterQueries(vector<int>& nums, vector<vector<int>>& queries) {  
}  
};
```

### Java Solution:

```
/**  
 * Problem: XOR After Range Multiplication Queries I  
 * Difficulty: Medium  
 * 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 xorAfterQueries(int[] nums, int[][] queries) {  
}  
}
```

### Python3 Solution:

```
"""  
Problem: XOR After Range Multiplication Queries I  
Difficulty: Medium  
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 xorAfterQueries(self, nums: List[int], queries: List[List[int]]) -> int:  
        # TODO: Implement optimized solution  
        pass
```

### Python Solution:

```

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

```

### JavaScript Solution:

```

/**
 * Problem: XOR After Range Multiplication Queries I
 * Difficulty: Medium
 * 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 xorAfterQueries = function(nums, queries) {
}
```

### TypeScript Solution:

```

/**
 * Problem: XOR After Range Multiplication Queries I
 * Difficulty: Medium
 * 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 xorAfterQueries(nums: number[], queries: number[][]): number {

```

```
};
```

### C# Solution:

```
/*
 * Problem: XOR After Range Multiplication Queries I
 * Difficulty: Medium
 * 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 XorAfterQueries(int[] nums, int[][] queries) {
        ...
    }
}
```

### C Solution:

```
/*
 * Problem: XOR After Range Multiplication Queries I
 * Difficulty: Medium
 * 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
 */

int xorAfterQueries(int* nums, int numSize, int** queries, int queriesSize,
                    int* queriesColSize) {
    ...
}
```

### Go Solution:

```

// Problem: XOR After Range Multiplication Queries I
// Difficulty: Medium
// 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 xorAfterQueries(nums []int, queries [][]int) int {

}

```

### Kotlin Solution:

```

class Solution {
    fun xorAfterQueries(nums: IntArray, queries: Array<IntArray>): Int {
        return 0
    }
}

```

### Swift Solution:

```

class Solution {
    func xorAfterQueries(_ nums: [Int], _ queries: [[Int]]) -> Int {
        return 0
    }
}

```

### Rust Solution:

```

// Problem: XOR After Range Multiplication Queries I
// Difficulty: Medium
// 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 xor_after_queries(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> i32 {
        0
    }
}

```

```
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

    }
}
```

### Dart Solution:

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

### Scala Solution:

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

### Elixir Solution:

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

### Erlang Solution:

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

### Racket Solution:

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