Problem 3599: Partition Array to Minimize XOR

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

Your task is to partition

nums

into

k

non-empty

subarrays

. For each subarray, compute the bitwise

XOR

of all its elements.

Return the

minimum

possible value of the

maximum XOR

among these

k

subarrays.

Example 1:

Input:

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

Output:

1

Explanation:

The optimal partition is

[1]

and

[2, 3]

.

XOR of the first subarray is

1

.

XOR of the second subarray is

2 XOR 3 = 1

.

The maximum XOR among the subarrays is 1, which is the minimum possible.

Example 2:

Input:

nums = [2,3,3,2], k = 3

Output:

2

Explanation:

The optimal partition is

[2]

,

[3, 3]

, and

[2]

.

XOR of the first subarray is

2

.

XOR of the second subarray is

3 XOR 3 = 0

.

XOR of the third subarray is

2

.

The maximum XOR among the subarrays is 2, which is the minimum possible.

Example 3:

Input:

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

Output:

0

Explanation:

The optimal partition is

[1, 1]

and

[2, 3, 1]

.

XOR of the first subarray is

1 XOR 1 = 0

.

XOR of the second subarray is

2 XOR 3 XOR 1 = 0

.

The maximum XOR among the subarrays is 0, which is the minimum possible.

Constraints:

1 <= nums.length <= 250

1 <= nums[i] <= 10

9

1 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
int minXor(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int minXor(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minXor(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MinXor(int[] nums, int k) {
}
}
```

C:

```
int minXor(int* nums, int numsSize, int k) {

}
```

Go:

```
func minXor(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minXor(nums: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def min_xor(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function minXor($nums, $k) {

}
}
```

Dart:

```
class Solution {
int minXor(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def minXor(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_xor(nums :: [integer], k :: integer) :: integer
def min_xor(nums, k) do
```

```
end
end
```

Erlang:

```
-spec min_xor(Nums :: [integer()], K :: integer()) -> integer().
min_xor(Nums, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Partition Array to Minimize XOR
* Difficulty: Medium
* 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 minXor(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Partition Array to Minimize XOR
* Difficulty: Medium
* 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 minXor(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition Array to Minimize XOR
Difficulty: Medium
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 minXor(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition Array to Minimize XOR
* Difficulty: Medium
* 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} k
* @return {number}
*/
var minXor = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition Array to Minimize XOR
* Difficulty: Medium
* 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 minXor(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Partition Array to Minimize XOR
* Difficulty: Medium
* 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 MinXor(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Partition Array to Minimize XOR
* Difficulty: Medium
* 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
*/

int minXor(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Partition Array to Minimize XOR
// Difficulty: Medium
// 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 minXor(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minXor(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Partition Array to Minimize XOR
// Difficulty: Medium
// 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 min_xor(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def min_xor(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function minXor($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minXor(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minXor(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_xor(nums :: [integer], k :: integer) :: integer
def min_xor(nums, k) do

end
end
```

Erlang Solution:

```
-spec min_xor(Nums :: [integer()], K :: integer()) -> integer().
min_xor(Nums, K) ->
.
```

Racket Solution:

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