Problem 1486: XOR Operation in an Array

Problem Information

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

Problem Description

You are given an integer

n

and an integer

start

.

Define an array

nums

where

nums[i] = start + 2 * i

(

0-indexed

) and

n == nums.length

.

Return

the bitwise XOR of all elements of

nums

.

Example 1:

Input:

n = 5, start = 0

Output:

8

Explanation:

Array nums is equal to [0, 2, 4, 6, 8] where (0 ^ 2 ^ 4 ^ 6 ^ 8) = 8. Where "^" corresponds to bitwise XOR operator.

Example 2:

Input:

n = 4, start = 3

Output:

8

Explanation:

Array nums is equal to [3, 5, 7, 9] where (3 ^ 5 ^ 7 ^ 9) = 8.

Constraints:

1 <= n <= 1000

0 <= start <= 1000

n == nums.length

Code Snippets

C++:

```
class Solution {
public:
int xorOperation(int n, int start) {

}
};
```

Java:

```
class Solution {
public int xorOperation(int n, int start) {

}
}
```

Python3:

```
class Solution:
def xorOperation(self, n: int, start: int) -> int:
```

Python:

```
class Solution(object):
def xorOperation(self, n, start):
"""
:type n: int
:type start: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} start
* @return {number}
*/
var xorOperation = function(n, start) {

};
```

TypeScript:

```
function xorOperation(n: number, start: number): number {

};
```

C#:

```
public class Solution {
public int XorOperation(int n, int start) {

}
}
```

C:

```
int xorOperation(int n, int start) {

}
```

Go:

```
func xorOperation(n int, start int) int {

}
```

Kotlin:

```
class Solution {
fun xorOperation(n: Int, start: Int): Int {

}
```

```
}
```

Swift:

```
class Solution {
func xorOperation(_ n: Int, _ start: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn xor_operation(n: i32, start: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} start
# @return {Integer}
def xor_operation(n, start)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $start
* @return Integer
*/
function xorOperation($n, $start) {

}
}
```

Dart:

```
class Solution {
int xorOperation(int n, int start) {

}
}
```

Scala:

```
object Solution {
def xorOperation(n: Int, start: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec xor_operation(n :: integer, start :: integer) :: integer
def xor_operation(n, start) do

end
end
```

Erlang:

```
-spec xor_operation(N :: integer(), Start :: integer()) -> integer().
xor_operation(N, Start) ->
.
```

Racket:

```
(define/contract (xor-operation n start)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: XOR Operation in an Array
```

```
* Difficulty: Easy
* Tags: array, 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 xorOperation(int n, int start) {

}
};
```

Java Solution:

```
/**
* Problem: XOR Operation in an Array
* Difficulty: Easy
* Tags: array, 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 xorOperation(int n, int start) {

}
}
```

Python3 Solution:

```
"""
Problem: XOR Operation in an Array
Difficulty: Easy
Tags: array, 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 xorOperation(self, n: int, start: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def xorOperation(self, n, start):
"""
:type n: int
:type start: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: XOR Operation in an Array
* Difficulty: Easy
* Tags: array, 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} n
* @param {number} start
* @return {number}
*/
var xorOperation = function(n, start) {

};
```

TypeScript Solution:

```
/**
* Problem: XOR Operation in an Array
* Difficulty: Easy
* Tags: array, 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 xorOperation(n: number, start: number): number {

};
```

C# Solution:

```
/*
* Problem: XOR Operation in an Array
* Difficulty: Easy
* Tags: array, 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 XorOperation(int n, int start) {

}
}
```

C Solution:

```
/*
* Problem: XOR Operation in an Array
* Difficulty: Easy
* Tags: array, 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 xorOperation(int n, int start) {

}
```

Go Solution:

```
// Problem: XOR Operation in an Array
// Difficulty: Easy
// Tags: array, 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 xorOperation(n int, start int) int {

}
```

Kotlin Solution:

```
class Solution {
fun xorOperation(n: Int, start: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func xorOperation(_ n: Int, _ start: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: XOR Operation in an Array
// Difficulty: Easy
// Tags: array, 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 xor_operation(n: i32, start: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} start
# @return {Integer}
def xor_operation(n, start)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $start
* @return Integer
*/
function xorOperation($n, $start) {

}
}
```

Dart Solution:

```
class Solution {
int xorOperation(int n, int start) {

}
}
```

Scala Solution:

```
object Solution {
def xorOperation(n: Int, start: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec xor_operation(n :: integer, start :: integer) :: integer
def xor_operation(n, start) do

end
end
```

Erlang Solution:

```
-spec xor_operation(N :: integer(), Start :: integer()) -> integer().
xor_operation(N, Start) ->
.
```

Racket Solution:

```
(define/contract (xor-operation n start)
(-> exact-integer? exact-integer? exact-integer?)
)
```