Problem 3133: Minimum Array End

Problem Information

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

Problem Description

You are given two integers

n

and

x

. You have to construct an array of

positive

integers

nums

of size

n

where for every

0 <= i < n - 1

,

nums[i + 1]

is

greater than

nums[i]

, and the result of the bitwise

AND

operation between all elements of

nums

is

x

.

Return the

minimum

possible value of

nums[n - 1]

.

Example 1:

Input:

n = 3, x = 4

Output:

6

Explanation:

nums

can be

[4,5,6]

and its last element is 6.

Example 2:

Input:

n = 2, x = 7

Output:

15

Explanation:

nums

can be

[7,15]

and its last element is 15.

Constraints:

1 <= n, x <= 10

8

Code Snippets

C++:

```
class Solution {
public:
long long minEnd(int n, int x) {

}
};
```

Java:

```
class Solution {
public long minEnd(int n, int x) {

}
}
```

Python3:

```
class Solution:
def minEnd(self, n: int, x: int) -> int:
```

Python:

```
class Solution(object):
def minEnd(self, n, x):
"""
:type n: int
:type x: int
:rtype: int
"""
```

JavaScript:

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

```
};
```

TypeScript:

```
function minEnd(n: number, x: number): number {

};
```

C#:

```
public class Solution {
public long MinEnd(int n, int x) {

}
}
```

C:

```
long long minEnd(int n, int x) {

}
```

Go:

```
func minEnd(n int, x int) int64 {

}
```

Kotlin:

```
class Solution {
fun minEnd(n: Int, x: Int): Long {

}
}
```

Swift:

```
class Solution {
func minEnd(_ n: Int, _ x: Int) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn min_end(n: i32, x: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} x
# @return {Integer}
def min_end(n, x)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $x
* @return Integer
*/
function minEnd($n, $x) {

}
}
```

Dart:

```
class Solution {
int minEnd(int n, int x) {

}
}
```

Scala:

```
object Solution {
def minEnd(n: Int, x: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_end(n :: integer, x :: integer) :: integer
def min_end(n, x) do

end
end
```

Erlang:

```
-spec min_end(N :: integer(), X :: integer()) -> integer().
min_end(N, X) ->
.
```

Racket:

```
(define/contract (min-end n x)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Array End
* 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:
long long minEnd(int n, int x) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Array End
* 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 long minEnd(int n, int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Array End
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 minEnd(self, n: int, x: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def minEnd(self, n, x):
"""
:type n: int
:type x: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Array End
* 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} n
* @param {number} x
* @return {number}
*/
var minEnd = function(n, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Array End
* 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 minEnd(n: number, x: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Array End
* 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 long MinEnd(int n, int x) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Array End
* 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
*/

long long minEnd(int n, int x) {

}
```

Go Solution:

```
// Problem: Minimum Array End
// 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 minEnd(n int, x int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minEnd(n: Int, x: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func minEnd(_ n: Int, _ x: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Array End
// 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 min_end(n: i32, x: i32) -> i64 {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} x
# @return {Integer}
def min_end(n, x)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $x
* @return Integer
*/
function minEnd($n, $x) {

}
}
```

Dart Solution:

```
class Solution {
int minEnd(int n, int x) {

}
}
```

Scala Solution:

```
object Solution {
def minEnd(n: Int, x: Int): Long = {

}
```

```
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_end(n :: integer, x :: integer) :: integer
def min_end(n, x) do

end
end
```

Erlang Solution:

```
-spec min_end(N :: integer(), X :: integer()) -> integer().
min_end(N, X) ->
.
```

Racket Solution:

```
(define/contract (min-end n x)
(-> exact-integer? exact-integer? exact-integer?)
)
```