Problem 2429: Minimize XOR

Problem Information

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

Problem Description

Given two positive integers

num1

and

num2

, find the positive integer

x

such that:

x

has the same number of set bits as

num2

, and

The value

x XOR num1

is

minimal

.

Note that

XOR

is the bitwise XOR operation.

Return

the integer

x

. The test cases are generated such that

x

is

uniquely determined

.

The number of

set bits

of an integer is the number of

1

's in its binary representation.

Example 1:

Input:

num1 = 3, num2 = 5

Output:

3

Explanation:

The binary representations of num1 and num2 are 0011 and 0101, respectively. The integer

3

has the same number of set bits as num2, and the value

3 XOR 3 = 0

is minimal.

Example 2:

Input:

num1 = 1, num2 = 12

Output:

3

Explanation:

The binary representations of num1 and num2 are 0001 and 1100, respectively. The integer

3

has the same number of set bits as num2, and the value

3 XOR 1 = 2

is minimal.

Constraints:

1 <= num1, num2 <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minimizeXor(int num1, int num2) {

}
};
```

Java:

```
class Solution {
public int minimizeXor(int num1, int num2) {

}
}
```

Python3:

```
class Solution:
def minimizeXor(self, num1: int, num2: int) -> int:
```

Python:

```
class Solution(object):
def minimizeXor(self, num1, num2):
"""
:type num1: int
```

```
:type num2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} num1
* @param {number} num2
* @return {number}
*/
var minimizeXor = function(num1, num2) {

};
```

TypeScript:

```
function minimizeXor(num1: number, num2: number): number {

};
```

C#:

```
public class Solution {
public int MinimizeXor(int num1, int num2) {

}
}
```

C:

```
int minimizeXor(int num1, int num2) {

}
```

Go:

```
func minimizeXor(num1 int, num2 int) int {

}
```

Kotlin:

```
class Solution {
fun minimizeXor(num1: Int, num2: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimizeXor(_ num1: Int, _ num2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimize_xor(num1: i32, num2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num1
# @param {Integer} num2
# @return {Integer}
def minimize_xor(num1, num2)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num1
* @param Integer $num2
* @return Integer
*/
function minimizeXor($num1, $num2) {

}
```

```
}
```

Dart:

```
class Solution {
int minimizeXor(int num1, int num2) {

}
}
```

Scala:

```
object Solution {
def minimizeXor(num1: Int, num2: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimize_xor(num1 :: integer, num2 :: integer) :: integer
def minimize_xor(num1, num2) do

end
end
```

Erlang:

```
-spec minimize_xor(Num1 :: integer(), Num2 :: integer()) -> integer().
minimize_xor(Num1, Num2) ->
.
```

Racket:

```
(define/contract (minimize-xor num1 num2)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize XOR
* Difficulty: Medium
* Tags: greedy
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int minimizeXor(int num1, int num2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize XOR
* Difficulty: Medium
* Tags: greedy
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minimizeXor(int num1, int num2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize XOR
Difficulty: Medium
Tags: greedy
```

```
Approach: Greedy algorithm with local optimal choices
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def minimizeXor(self, num1: int, num2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimizeXor(self, num1, num2):
"""
:type num1: int
:type num2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimize XOR
* Difficulty: Medium
* Tags: greedy
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} num1
* @param {number} num2
* @return {number}
*/
var minimizeXor = function(num1, num2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize XOR
* Difficulty: Medium
* Tags: greedy
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function minimizeXor(num1: number, num2: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimize XOR
* Difficulty: Medium
* Tags: greedy
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinimizeXor(int num1, int num2) {

}
}
```

C Solution:

```
/*
* Problem: Minimize XOR
* Difficulty: Medium
* Tags: greedy
*
* Approach: Greedy algorithm with local optimal choices
```

```
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int minimizeXor(int num1, int num2) {

}
```

Go Solution:

```
// Problem: Minimize XOR
// Difficulty: Medium
// Tags: greedy
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func minimizeXor(num1 int, num2 int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimizeXor(num1: Int, num2: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimizeXor(_ num1: Int, _ num2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimize XOR
// Difficulty: Medium
// Tags: greedy
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn minimize_xor(num1: i32, num2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num1
# @param {Integer} num2
# @return {Integer}
def minimize_xor(num1, num2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $num1
* @param Integer $num2
* @return Integer
*/
function minimizeXor($num1, $num2) {

}
}
```

Dart Solution:

```
class Solution {
int minimizeXor(int num1, int num2) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def minimizeXor(num1: Int, num2: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimize_xor(num1 :: integer, num2 :: integer) :: integer
def minimize_xor(num1, num2) do

end
end
```

Erlang Solution:

```
-spec minimize_xor(Num1 :: integer(), Num2 :: integer()) -> integer().
minimize_xor(Num1, Num2) ->
.
```

Racket Solution:

```
(define/contract (minimize-xor num1 num2)
(-> exact-integer? exact-integer? exact-integer?)
)
```