Problem 2568: Minimum Impossible OR

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

.

We say that an integer x is

expressible

from

nums

if there exist some integers

0 <= index

1

< index

2

< ... < index

k

< nums.length

for which

nums[index

1

] | nums[index

2

] | ... | nums[index

k

] = x

. In other words, an integer is expressible if it can be written as the bitwise OR of some subsequence of

nums

.

Return

the minimum

positive non-zero integer

that is not

expressible from

nums

.

Example 1:

Input:

nums = [2,1]

Output:

4

Explanation:

1 and 2 are already present in the array. We know that 3 is expressible, since nums[0] | nums[1] = 2 | 1 = 3. Since 4 is not expressible, we return 4.

Example 2:

Input:

nums = [5,3,2]

Output:

1

Explanation:

We can show that 1 is the smallest number that is not expressible.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minImpossibleOR(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int minImpossibleOR(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minImpossibleOR(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MinImpossibleOR(int[] nums) {

}
}
```

C:

```
int minImpossibleOR(int* nums, int numsSize) {

}
```

Go:

```
func minImpossibleOR(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun minImpossibleOR(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minImpossibleOR(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_impossible_or(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def min_impossible_or(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function minImpossibleOR($nums) {

}
}
```

Dart:

```
class Solution {
int minImpossibleOR(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def minImpossibleOR(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_impossible_or(nums :: [integer]) :: integer
def min_impossible_or(nums) do

end
end
```

Erlang:

```
-spec min_impossible_or(Nums :: [integer()]) -> integer().
min_impossible_or(Nums) ->
.
```

Racket:

```
(define/contract (min-impossible-or nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Impossible OR
* 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 minImpossibleOR(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Impossible OR
* 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 minImpossibleOR(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Impossible OR
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 minImpossibleOR(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Impossible OR
* 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
* @return {number}
*/
var minImpossibleOR = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Impossible OR
* 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 minImpossibleOR(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Impossible OR
* 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 MinImpossibleOR(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Impossible OR
* 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 minImpossibleOR(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Impossible OR
// 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 minImpossibleOR(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minImpossibleOR(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minImpossibleOR(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Impossible OR
// 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_impossible_or(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def min_impossible_or(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function minImpossibleOR($nums) {

}
}
```

Dart Solution:

```
class Solution {
int minImpossibleOR(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def minImpossibleOR(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_impossible_or(nums :: [integer]) :: integer
def min_impossible_or(nums) do

end
end
```

Erlang Solution:

```
-spec min_impossible_or(Nums :: [integer()]) -> integer().
min_impossible_or(Nums) ->
.
```

Racket Solution:

```
(define/contract (min-impossible-or nums)
(-> (listof exact-integer?) exact-integer?)
)
```