Problem 3309: Maximum Possible Number by Binary Concatenation

Problem Information

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

Problem Description

You are given an array of integers

nums

of size 3.

Return the

maximum

possible number whose

binary representation

can be formed by

concatenating

the

binary representation

of

all

elements in

nums

in some order.

Note

that the binary representation of any number

does not

contain leading zeros.

Example 1:

Input:

nums = [1,2,3]

Output:

30

Explanation:

Concatenate the numbers in the order

[3, 1, 2]

to get the result

"11110"

, which is the binary representation of 30.

Example 2:

Input:

nums = [2,8,16]

Output:

1296

Explanation:

Concatenate the numbers in the order

[2, 8, 16]

to get the result

"10100010000"

, which is the binary representation of 1296.

Constraints:

nums.length == 3

1 <= nums[i] <= 127

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-good-number nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Possible Number by Binary Concatenation
* 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 maxGoodNumber(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Possible Number by Binary Concatenation
* 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 maxGoodNumber(int[] nums) {
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Possible Number by Binary Concatenation
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 maxGoodNumber(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Possible Number by Binary Concatenation
* 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 maxGoodNumber = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Possible Number by Binary Concatenation
* 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 maxGoodNumber(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Possible Number by Binary Concatenation
* 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 MaxGoodNumber(int[] nums) {

}

}
```

C Solution:

```
/*
* Problem: Maximum Possible Number by Binary Concatenation
* 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 maxGoodNumber(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Possible Number by Binary Concatenation
// 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 maxGoodNumber(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Possible Number by Binary Concatenation
// 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 max_good_number(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-good-number nums)
(-> (listof exact-integer?) exact-integer?)
)
```