Problem 2595: Number of Even and Odd Bits

Problem Information

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

Problem Description

You are given a

positive

integer

n

.

Let

even

denote the number of even indices in the binary representation of

n

with value 1.

Let

odd

denote the number of odd indices in the binary representation of

n

with value 1.

Note that bits are indexed from

right to left

in the binary representation of a number.

Return the array

[even, odd]

.

Example 1:

Input:

n = 50

Output:

[1,2]

Explanation:

The binary representation of 50 is

110010

.

It contains 1 on indices 1, 4, and 5.

Example 2:

Input:

n = 2

Output:

[0,1]

Explanation:

The binary representation of 2 is

10

.

It contains 1 only on index 1.

Constraints:

1 <= n <= 1000

Code Snippets

C++:

```
class Solution {
public:
vector<int> evenOddBit(int n) {

}
};
```

Java:

```
class Solution {
public int[] evenOddBit(int n) {

}
}
```

Python3:

```
class Solution:
def evenOddBit(self, n: int) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function evenOddBit(n: number): number[] {

};
```

C#:

```
public class Solution {
public int[] EvenOddBit(int n) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* evenOddBit(int n, int* returnSize) {

}
```

Go:

```
func evenOddBit(n int) []int {

}
```

Kotlin:

```
class Solution {
fun evenOddBit(n: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func evenOddBit(_ n: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn even_odd_bit(n: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer[]}
def even_odd_bit(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer[]
*/
function evenOddBit($n) {

}
}
```

Dart:

```
class Solution {
List<int> evenOddBit(int n) {

}
}
```

Scala:

```
object Solution {
def evenOddBit(n: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec even_odd_bit(n :: integer) :: [integer]
def even_odd_bit(n) do

end
end
```

Erlang:

```
-spec even_odd_bit(N :: integer()) -> [integer()].
even_odd_bit(N) ->
.
```

Racket:

```
(define/contract (even-odd-bit n)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Even and Odd Bits
* Difficulty: Easy
* 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:
vector<int> evenOddBit(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Even and Odd Bits
* Difficulty: Easy
* 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[] evenOddBit(int n) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Number of Even and Odd Bits
Difficulty: Easy
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 evenOddBit(self, n: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Even and Odd Bits
* Difficulty: Easy
* 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
* @return {number[]}
*/
var evenOddBit = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Even and Odd Bits
* Difficulty: Easy
* 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 evenOddBit(n: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Number of Even and Odd Bits
* Difficulty: Easy
* 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[] EvenOddBit(int n) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Number of Even and Odd Bits
* Difficulty: Easy
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* evenOddBit(int n, int* returnSize) {

}
```

Go Solution:

```
// Problem: Number of Even and Odd Bits
// Difficulty: Easy
// 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 evenOddBit(n int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun evenOddBit(n: Int): IntArray {

}
```

```
}
```

Swift Solution:

```
class Solution {
func evenOddBit(_ n: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Number of Even and Odd Bits
// Difficulty: Easy
// 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 even_odd_bit(n: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer[]}
def even_odd_bit(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer[]
```

```
*/
function evenOddBit($n) {

}
}
```

Dart Solution:

```
class Solution {
List<int> evenOddBit(int n) {

}
}
```

Scala Solution:

```
object Solution {
def evenOddBit(n: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec even_odd_bit(n :: integer) :: [integer]
def even_odd_bit(n) do

end
end
```

Erlang Solution:

```
-spec even_odd_bit(N :: integer()) -> [integer()].
even_odd_bit(N) ->
.
```

Racket Solution:

```
(define/contract (even-odd-bit n)
(-> exact-integer? (listof exact-integer?))
```

```
)
```