Problem 338: Counting Bits

Problem Information

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

Problem Description

Given an integer

n

, return

an array

ans

of length

n + 1

such that for each

i

(

0 <= i <= n

)

,

ans[i]

is the

number of

1

's

in the binary representation of

i

.

Example 1:

Input:

n = 2

Output:

[0,1,1]

Explanation:

0 --> 0 1 --> 1 2 --> 10

Example 2:

Input:

n = 5

Output:

[0,1,1,2,1,2]

Explanation:

0 --> 0 1 --> 1 2 --> 10 3 --> 11 4 --> 100 5 --> 101

Constraints:

0 <= n <= 10

5

Follow up:

It is very easy to come up with a solution with a runtime of

O(n log n)

. Can you do it in linear time

O(n)

and possibly in a single pass?

Can you do it without using any built-in function (i.e., like

__builtin_popcount

in C++)?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int[] CountBits(int n) {
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (count-bits n)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Counting Bits
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Counting Bits
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int[] countBits(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Counting Bits
Difficulty: Easy
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def countBits(self, n: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Counting Bits
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Counting Bits
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function countBits(n: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Counting Bits
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int[] CountBits(int n) {

}
}
```

C Solution:

```
/*
* Problem: Counting Bits
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Counting Bits
// Difficulty: Easy
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func countBits(n int) []int {


}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Counting Bits
// Difficulty: Easy
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
.
```

Racket Solution:

```
(define/contract (count-bits n)
(-> exact-integer? (listof exact-integer?))
)
```