

Problem 191: Number of 1 Bits

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a positive integer

n

, write a function that returns the number of

set bits

in its binary representation (also known as the

Hamming weight

).

Example 1:

Input:

$n = 11$

Output:

3

Explanation:

The input binary string

1011

has a total of three set bits.

Example 2:

Input:

n = 128

Output:

1

Explanation:

The input binary string

10000000

has a total of one set bit.

Example 3:

Input:

n = 2147483645

Output:

30

Explanation:

The input binary string

11111111111111111111111111111101

has a total of thirty set bits.

Constraints:

$1 \leq n \leq 2$

31

- 1

Follow up:

If this function is called many times, how would you optimize it?

Code Snippets

C++:

```
class Solution {  
public:  
    int hammingWeight(int n) {  
  
    }  
};
```

Java:

```
class Solution {  
    public int hammingWeight(int n) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def hammingWeight(self, n: int) -> int:
```

Python:

```
class Solution(object):
    def hammingWeight(self, n):
        """
        :type n: int
        :rtype: int
        """
```

JavaScript:

```
/**
 * @param {number} n
 * @return {number}
 */
var hammingWeight = function(n) {

};
```

TypeScript:

```
function hammingWeight(n: number): number {

};
```

C#:

```
public class Solution {
    public int HammingWeight(int n) {

    }
}
```

C:

```
int hammingWeight(int n) {

}
```

Go:

```
func hammingWeight(n int) int {

}
```

Kotlin:

```
class Solution {  
    fun hammingWeight(n: Int): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func hammingWeight(_ n: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn hamming_weight(n: i32) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer} n  
# @return {Integer}  
def hamming_weight(n)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer $n  
     * @return Integer  
     */  
    function hammingWeight($n) {  
  
    }  
}
```

```
}
```

Dart:

```
class Solution {  
  int hammingWeight(int n) {  
  
  }  
}
```

Scala:

```
object Solution {  
  def hammingWeight(n: Int): Int = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec hamming_weight(n :: integer) :: integer  
  def hamming_weight(n) do  
  
  end  
end
```

Erlang:

```
-spec hamming_weight(N :: integer()) -> integer().  
hamming_weight(N) ->  
.
```

Racket:

```
(define/contract (hamming-weight n)  
  (-> exact-integer? exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Number of 1 Bits
 * Difficulty: Easy
 * Tags: string
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public:
    int hammingWeight(int n) {

    }
};
```

Java Solution:

```
/**
 * Problem: Number of 1 Bits
 * Difficulty: Easy
 * Tags: string
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
    public int hammingWeight(int n) {

    }
}
```

Python3 Solution:

```
"""
Problem: Number of 1 Bits
Difficulty: Easy
Tags: string
```

```

Approach: String manipulation with hash map or two pointers
Time Complexity:  $O(n)$  or  $O(n \log n)$ 
Space Complexity:  $O(1)$  to  $O(n)$  depending on approach
"""

class Solution:
    def hammingWeight(self, n: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def hammingWeight(self, n):
        """
        :type n: int
        :rtype: int
        """

```

JavaScript Solution:

```

/**
 * Problem: Number of 1 Bits
 * Difficulty: Easy
 * Tags: string
 *
 * Approach: String manipulation with hash map or two pointers
 * 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 hammingWeight = function(n) {

};

```

TypeScript Solution:

```

/**
 * Problem: Number of 1 Bits
 * Difficulty: Easy
 * Tags: string
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

function hammingWeight(n: number): number {

};

```

C# Solution:

```

/*
 * Problem: Number of 1 Bits
 * Difficulty: Easy
 * Tags: string
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public int HammingWeight(int n) {

    }
}

```

C Solution:

```

/*
 * Problem: Number of 1 Bits
 * Difficulty: Easy
 * Tags: string
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach

```

```
*/

int hammingWeight(int n) {

}
```

Go Solution:

```
// Problem: Number of 1 Bits
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func hammingWeight(n int) int {

}
```

Kotlin Solution:

```
class Solution {
    fun hammingWeight(n: Int): Int {

    }
}
```

Swift Solution:

```
class Solution {
    func hammingWeight(_ n: Int) -> Int {

    }
}
```

Rust Solution:

```
// Problem: Number of 1 Bits
// Difficulty: Easy
// Tags: string
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn hamming_weight(n: i32) -> i32 {

    }
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def hamming_weight(n)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer $n
     * @return Integer
     */
    function hammingWeight($n) {

    }

}
```

Dart Solution:

```
class Solution {
    int hammingWeight(int n) {

    }
}
```

Scala Solution:

```
object Solution {  
  def hammingWeight(n: Int): Int = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec hamming_weight(n :: integer) :: integer  
  def hamming_weight(n) do  
  
  end  
end
```

Erlang Solution:

```
-spec hamming_weight(N :: integer()) -> integer().  
hamming_weight(N) ->  
.
```

Racket Solution:

```
(define/contract (hamming-weight n)  
  (-> exact-integer? exact-integer?)  
)
```