Problem 461: Hamming Distance

Problem Information

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

Problem Description

The

Hamming distance

between two integers is the number of positions at which the corresponding bits are different.

Given two integers

x

and

y

, return

the

Hamming distance

between them

.

Example 1:

Input:

x = 1, y = 4

Output:

2

Explanation:

1 (0 0 0 1) 4 (0 1 0 0) ↑ ↑ The above arrows point to positions where the corresponding bits are different.

Example 2:

Input:

x = 3, y = 1

Output:

1

Constraints:

0 <= x, y <= 2

31

- 1

Note:

This question is the same as

2220: Minimum Bit Flips to Convert Number.

Code Snippets

C++:

```
class Solution {
public:
int hammingDistance(int x, int y) {

}
};
```

Java:

```
class Solution {
public int hammingDistance(int x, int y) {

}
}
```

Python3:

```
class Solution:
def hammingDistance(self, x: int, y: int) -> int:
```

Python:

```
class Solution(object):
def hammingDistance(self, x, y):
"""
:type x: int
:type y: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} y
* @return {number}
*/
var hammingDistance = function(x, y) {
```

```
};
```

TypeScript:

```
function hammingDistance(x: number, y: number): number {

};
```

C#:

```
public class Solution {
public int HammingDistance(int x, int y) {

}
}
```

C:

```
int hammingDistance(int x, int y) {

}
```

Go:

```
func hammingDistance(x int, y int) int {

}
```

Kotlin:

```
class Solution {
fun hammingDistance(x: Int, y: Int): Int {

}
}
```

Swift:

```
class Solution {
func hammingDistance(_ x: Int, _ y: Int) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn hamming_distance(x: i32, y: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} x
# @param {Integer} y
# @return {Integer}
def hamming_distance(x, y)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @return Integer
*/
function hammingDistance($x, $y) {

}
}
```

Dart:

```
class Solution {
int hammingDistance(int x, int y) {

}
}
```

Scala:

```
object Solution {
def hammingDistance(x: Int, y: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec hamming_distance(x :: integer, y :: integer) :: integer
def hamming_distance(x, y) do

end
end
```

Erlang:

```
-spec hamming_distance(X :: integer(), Y :: integer()) -> integer().
hamming_distance(X, Y) ->
.
```

Racket:

```
(define/contract (hamming-distance x y)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Hamming Distance
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int hammingDistance(int x, int y) {

}
};
```

Java Solution:

```
/**
* Problem: Hamming Distance
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int hammingDistance(int x, int y) {

}
}
```

Python3 Solution:

```
"""
Problem: Hamming Distance
Difficulty: Easy
Tags: general

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def hammingDistance(self, x: int, y: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def hammingDistance(self, x, y):
"""
:type x: int
:type y: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Hamming Distance
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} x
* @param {number} y
* @return {number}
*/
var hammingDistance = function(x, y) {

};
```

TypeScript Solution:

```
/**
* Problem: Hamming Distance
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
function hammingDistance(x: number, y: number): number {

};
```

C# Solution:

```
/*
* Problem: Hamming Distance
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int HammingDistance(int x, int y) {

}
}
```

C Solution:

```
/*
* Problem: Hamming Distance
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int hammingDistance(int x, int y) {

}
```

Go Solution:

```
// Problem: Hamming Distance
// Difficulty: Easy
// Tags: general
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func hammingDistance(x int, y int) int {

}
```

Kotlin Solution:

```
class Solution {
fun hammingDistance(x: Int, y: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func hammingDistance(_ x: Int, _ y: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Hamming Distance
// Difficulty: Easy
// Tags: general
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn hamming_distance(x: i32, y: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} x
# @param {Integer} y
# @return {Integer}
def hamming_distance(x, y)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @return Integer
*/
function hammingDistance($x, $y) {

}
}
```

Dart Solution:

```
class Solution {
int hammingDistance(int x, int y) {

}
}
```

Scala Solution:

```
object Solution {
def hammingDistance(x: Int, y: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec hamming_distance(x :: integer, y :: integer) :: integer
def hamming_distance(x, y) do

end
end
```

Erlang Solution:

```
-spec hamming_distance(X :: integer(), Y :: integer()) -> integer().
hamming_distance(X, Y) ->
.
```

Racket Solution:

```
(define/contract (hamming-distance x y)
(-> exact-integer? exact-integer? exact-integer?)
)
```