Problem 477: Total Hamming Distance

Problem Information

Difficulty: Medium
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 an integer array

nums

, return

the sum of

Hamming distances

between all the pairs of the integers in

nums

.

Example 1:

Input:

nums = [4,14,2]

Output:

6

Explanation:

In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just showing the four bits relevant in this case). The answer will be: HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.

Example 2:

Input:

nums = [4,14,4]

Output:

4

Constraints:

1 <= nums.length <= 10

4

0 <= nums[i] <= 10

9

The answer for the given input will fit in a

32-bit

integer.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (total-hamming-distance nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Total Hamming Distance
* Difficulty: Medium
* Tags: array, math
*
* 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 totalHammingDistance(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Total Hamming Distance
* Difficulty: Medium
* Tags: array, math
*
* 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 totalHammingDistance(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Total Hamming Distance
Difficulty: Medium
Tags: array, math

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 totalHammingDistance(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Total Hamming Distance
* Difficulty: Medium
* Tags: array, math
*
* 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 totalHammingDistance = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Total Hamming Distance
* Difficulty: Medium
* Tags: array, math
*
* 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 totalHammingDistance(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Total Hamming Distance
* Difficulty: Medium
* Tags: array, math
*
* 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 TotalHammingDistance(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Total Hamming Distance
* Difficulty: Medium
* Tags: array, math
*
* 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 totalHammingDistance(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Total Hamming Distance
// Difficulty: Medium
// Tags: array, math
//
// 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 totalHammingDistance(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Total Hamming Distance
// Difficulty: Medium
// Tags: array, math
//
// 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 total_hamming_distance(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (total-hamming-distance nums)
(-> (listof exact-integer?) exact-integer?)
)
```