Problem 493: Reverse Pairs

Problem Information

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

Problem Description

Given an integer array

nums

, return

the number of

reverse pairs

in the array

.

A

reverse pair

is a pair

(i, j)

where:

0 <= i < j < nums.length

and

nums[i] > 2 * nums[j]

.

Example 1:

Input:

nums = [1,3,2,3,1]

Output:

2

Explanation:

The reverse pairs are: (1, 4) --> nums[1] = 3, nums[4] = 1, 3 > 2 * 1 (3, 4) --> nums[3] = 3, nums[4] = 1, 3 > 2 * 1

Example 2:

Input:

nums = [2,4,3,5,1]

Output:

3

Explanation:

The reverse pairs are: (1, 4) --> nums[1] = 4, nums[4] = 1, 4 > 2 * 1 (2, 4) --> nums[2] = 3, nums[4] = 1, 3 > 2 * 1 (3, 4) --> nums[3] = 5, nums[4] = 1, 5 > 2 * 1

Constraints:

1 <= nums.length <= 5 * 10

4

-2

31

<= nums[i] <= 2

31

- 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def reversePairs(self, nums):
```

```
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (reverse-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Pairs
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Reverse Pairs
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int reversePairs(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Reverse Pairs
Difficulty: Hard
Tags: array, tree, sort, search
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reverse Pairs
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Pairs
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function reversePairs(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Reverse Pairs
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int ReversePairs(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Reverse Pairs
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

int reversePairs(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Reverse Pairs
// Difficulty: Hard
// Tags: array, tree, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func reversePairs(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reverse Pairs
// Difficulty: Hard
// Tags: array, tree, sort, search
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (reverse-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```