Problem 825: Friends Of Appropriate Ages

Problem Information

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

Problem Description

There are

n

persons on a social media website. You are given an integer array

ages

where

ages[i]

is the age of the

i

th

person.

A Person

x

will not send a friend request to a person

y

(

x != y

) if any of the following conditions is true:

age[y] <= 0.5 * age[x] + 7

age[y] > age[x]

age[y] > 100 && age[x] < 100

Otherwise,

x

will send a friend request to

y

.

Note that if

x

sends a request to

y

,

y

will not necessarily send a request to

x

. Also, a person will not send a friend request to themself.

Return

the total number of friend requests made

.

Example 1:

Input:

ages = [16,16]

Output:

2

Explanation:

2 people friend request each other.

Example 2:

Input:

ages = [16,17,18]

Output:

2

Explanation:

Friend requests are made 17 -> 16, 18 -> 17.

Example 3:

Input:

ages = [20,30,100,110,120]

Output:

3

Explanation:

Friend requests are made 110 -> 100, 120 -> 110, 120 -> 100.

Constraints:

n == ages.length

1 <= n <= 2 * 10

4

1 <= ages[i] <= 120

Code Snippets

C++:

```
class Solution {
public:
int numFriendRequests(vector<int>& ages) {

}
};
```

Java:

```
class Solution {
public int numFriendRequests(int[] ages) {

}
```

```
}
```

Python3:

```
class Solution:
def numFriendRequests(self, ages: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numFriendRequests(ages: number[]): number {

};
```

C#:

```
public class Solution {
public int NumFriendRequests(int[] ages) {

}
}
```

C:

```
int numFriendRequests(int* ages, int agesSize) {

}
```

Go:

```
func numFriendRequests(ages []int) int {

}
```

Kotlin:

```
class Solution {
fun numFriendRequests(ages: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func numFriendRequests(_ ages: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_friend_requests(ages: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} ages
# @return {Integer}
def num_friend_requests(ages)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $ages
* @return Integer
*/
function numFriendRequests($ages) {

}
}
```

Dart:

```
class Solution {
int numFriendRequests(List<int> ages) {

}
}
```

Scala:

```
object Solution {
def numFriendRequests(ages: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_friend_requests(ages :: [integer]) :: integer
def num_friend_requests(ages) do

end
end
```

Erlang:

```
-spec num_friend_requests(Ages :: [integer()]) -> integer().
num_friend_requests(Ages) ->
.
```

Racket:

```
(define/contract (num-friend-requests ages)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Friends Of Appropriate Ages
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 numFriendRequests(vector<int>& ages) {

}
};
```

Java Solution:

```
/**
* Problem: Friends Of Appropriate Ages
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 numFriendRequests(int[] ages) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Friends Of Appropriate Ages
Difficulty: Medium
Tags: array, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Friends Of Appropriate Ages
* Difficulty: Medium
* Tags: array, sort, search
*
* 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[]} ages
* @return {number}
*/
var numFriendRequests = function(ages) {

};
```

TypeScript Solution:

```
/**
* Problem: Friends Of Appropriate Ages
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 numFriendRequests(ages: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Friends Of Appropriate Ages
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 NumFriendRequests(int[] ages) {

}
}
```

C Solution:

```
/*
* Problem: Friends Of Appropriate Ages
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 numFriendRequests(int* ages, int agesSize) {

}
```

Go Solution:

```
// Problem: Friends Of Appropriate Ages
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 numFriendRequests(ages []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numFriendRequests(ages: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func numFriendRequests(_ ages: [Int]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Friends Of Appropriate Ages
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 num_friend_requests(ages: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} ages
# @return {Integer}
def num_friend_requests(ages)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $ages
* @return Integer
*/
function numFriendRequests($ages) {

}
}
```

Dart Solution:

```
class Solution {
int numFriendRequests(List<int> ages) {

}
}
```

Scala Solution:

```
object Solution {
def numFriendRequests(ages: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_friend_requests(ages :: [integer]) :: integer
def num_friend_requests(ages) do

end
end
```

Erlang Solution:

```
-spec num_friend_requests(Ages :: [integer()]) -> integer().
num_friend_requests(Ages) ->
.
```

Racket Solution:

```
(define/contract (num-friend-requests ages)
(-> (listof exact-integer?) exact-integer?)
)
```