Problem 2709: Greatest Common Divisor Traversal

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

, and you are allowed to

traverse

between its indices. You can traverse between index

i

and index

j

,

i != j

, if and only if

gcd(nums[i], nums[j]) > 1

, where

gcd

is the

greatest common divisor

.

Your task is to determine if for

every pair

of indices

i

and

j

in nums, where

i < j

, there exists a

sequence of traversals

that can take us from

i

to

j

.

Return

true

if it is possible to traverse between all such pairs of indices,

or

false

otherwise.

Example 1:

Input:

nums = [2,3,6]

Output:

true

Explanation:

In this example, there are 3 possible pairs of indices: (0, 1), (0, 2), and (1, 2). To go from index 0 to index 1, we can use the sequence of traversals 0 -> 2 -> 1, where we move from index 0 to index 2 because gcd(nums[0], nums[2]) = gcd(2, 6) = 2 > 1, and then move from index 2 to index 1 because gcd(nums[2], nums[1]) = gcd(6, 3) = 3 > 1. To go from index 0 to index 2, we can just go directly because gcd(nums[0], nums[2]) = gcd(2, 6) = 2 > 1. Likewise, to go from index 1 to index 2, we can just go directly because gcd(nums[1], nums[2]) = gcd(3, 6) = 3 > 1.

Example 2:

Input:

nums = [3,9,5]

Output:

false

Explanation:

No sequence of traversals can take us from index 0 to index 2 in this example. So, we return false.

Example 3:

Input:

nums = [4,3,12,8]

Output:

true

Explanation:

There are 6 possible pairs of indices to traverse between: (0, 1), (0, 2), (0, 3), (1, 2), (1, 3), and (2, 3). A valid sequence of traversals exists for each pair, so we return true.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean canTraverseAllPairs(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function canTraverseAllPairs(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool CanTraverseAllPairs(int[] nums) {

}
}
```

C:

```
bool canTraverseAllPairs(int* nums, int numsSize) {

}
```

Go:

```
func canTraverseAllPairs(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun canTraverseAllPairs(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool canTraverseAllPairs(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (can-traverse-all-pairs nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Greatest Common Divisor Traversal
* Difficulty: Hard
* Tags: array, graph, 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:
bool canTraverseAllPairs(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Greatest Common Divisor Traversal
* Difficulty: Hard
* Tags: array, graph, 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 boolean canTraverseAllPairs(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Greatest Common Divisor Traversal
Difficulty: Hard
Tags: array, graph, 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 canTraverseAllPairs(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Greatest Common Divisor Traversal
* Difficulty: Hard
* Tags: array, graph, 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 {boolean}
*/
var canTraverseAllPairs = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Greatest Common Divisor Traversal
* Difficulty: Hard
* Tags: array, graph, 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 canTraverseAllPairs(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Greatest Common Divisor Traversal
* Difficulty: Hard
* Tags: array, graph, 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 bool CanTraverseAllPairs(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Greatest Common Divisor Traversal
* Difficulty: Hard
* Tags: array, graph, 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
*/

bool canTraverseAllPairs(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Greatest Common Divisor Traversal
// Difficulty: Hard
// Tags: array, graph, 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 canTraverseAllPairs(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canTraverseAllPairs(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Greatest Common Divisor Traversal
// Difficulty: Hard
// Tags: array, graph, 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 can_traverse_all_pairs(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool canTraverseAllPairs(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-traverse-all-pairs nums)
  (-> (listof exact-integer?) boolean?)
  )
```