Problem 1998: GCD Sort of an Array

Problem Information

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

Problem Description

You are given an integer array

nums

, and you can perform the following operation

any

number of times on

nums

:

Swap the positions of two elements

nums[i]

and

nums[j]

if

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

where

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

is the

greatest common divisor

of

nums[i]

and

nums[j]

.

Return

true

if it is possible to sort

nums

in

non-decreasing

order using the above swap method, or

false

otherwise.

Example 1:

Input:

nums = [7,21,3]

Output:

true

Explanation:

We can sort [7,21,3] by performing the following operations: - Swap 7 and 21 because gcd(7,21) = 7. nums = [

21

,

7

,3] - Swap 21 and 3 because gcd(21,3) = 3. nums = [

3

,7,

21

]

Example 2:

Input:

nums = [5,2,6,2]

Output:

false

Explanation:

It is impossible to sort the array because 5 cannot be swapped with any other element.

Example 3:

Input:

nums = [10,5,9,3,15]

Output:

true We can sort [10,5,9,3,15] by performing the following operations: - Swap 10 and 15 because gcd(10,15) = 5. nums = [

15

,5,9,3,

10

] - Swap 15 and 3 because gcd(15,3) = 3. nums = [

3

,5,9,

15

,10] - Swap 10 and 15 because gcd(10,15) = 5. nums = [3,5,9,

10

,

15

]

Constraints:

1 <= nums.length <= 3 * 10

4

2 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (gcd-sort nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: GCD Sort of an Array
* Difficulty: Hard
* Tags: array, graph, math, sort
*
```

```
* 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 gcdSort(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: GCD Sort of an Array
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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 gcdSort(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: GCD Sort of an Array
Difficulty: Hard
Tags: array, graph, math, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: GCD Sort of an Array
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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 gcdSort = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: GCD Sort of an Array
* Difficulty: Hard
* Tags: array, graph, math, sort
```

```
*
* 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 gcdSort(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: GCD Sort of an Array
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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 GcdSort(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: GCD Sort of an Array
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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 gcdSort(int* nums, int numsSize) {
```

```
}
```

Go Solution:

```
// Problem: GCD Sort of an Array
// Difficulty: Hard
// Tags: array, graph, math, sort
//
// 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 gcdSort(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: GCD Sort of an Array
// Difficulty: Hard
// Tags: array, graph, math, sort
//
// 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 gcd_sort(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (gcd-sort nums)
(-> (listof exact-integer?) boolean?)
)
```