Problem 2197: Replace Non-Coprime Numbers in Array

Problem Information

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

Problem Description

You are given an array of integers

nums

. Perform the following steps:

Find

any

two

adjacent

numbers in

nums

that are

non-coprime

.

If no such numbers are found,

stop

the process.

Otherwise, delete the two numbers and

replace

them with their

LCM (Least Common Multiple)

.

Repeat

this process as long as you keep finding two adjacent non-coprime numbers.

Return

the

final

modified array.

It can be shown that replacing adjacent non-coprime numbers in

any

arbitrary order will lead to the same result.

The test cases are generated such that the values in the final array are

less than or equal

to

10

8

.

Two values

x

and

y

are

non-coprime

if

GCD(x, y) > 1

where

GCD(x, y)

is the

Greatest Common Divisor

of

x

and

y

.

Example 1:

Input:

nums = [6,4,3,2,7,6,2]

Output:

[12,7,6]

Explanation:

- (6, 4) are non-coprime with LCM(6, 4) = 12. Now, nums = [

12

,3,2,7,6,2]. - (12, 3) are non-coprime with LCM(12, 3) = 12. Now, nums = [

12

,2,7,6,2]. - (12, 2) are non-coprime with LCM(12, 2) = 12. Now, nums = [

12

,7,6,2]. - (6, 2) are non-coprime with LCM(6, 2) = 6. Now, nums = [12,7,

6

]. There are no more adjacent non-coprime numbers in nums. Thus, the final modified array is [12,7,6]. Note that there are other ways to obtain the same resultant array.

Example 2:

Input:

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

Output:

[2,1,1,3]

Explanation:

- (3, 3) are non-coprime with LCM(3, 3) = 3. Now, nums = [2,2,1,1,

3

,3]. - (3, 3) are non-coprime with LCM(3, 3) = 3. Now, nums = [2,2,1,1,

3

]. - (2, 2) are non-coprime with LCM(2, 2) = 2. Now, nums = [

2

,1,1,3]. There are no more adjacent non-coprime numbers in nums. Thus, the final modified array is [2,1,1,3]. Note that there are other ways to obtain the same resultant array.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

The test cases are generated such that the values in the final array are

less than or equal

to

10

8

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> replaceNonCoprimes(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number[]}
*/
```

```
var replaceNonCoprimes = function(nums) {

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> ReplaceNonCoprimes(int[] nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* replaceNonCoprimes(int* nums, int numsSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun replaceNonCoprimes(nums: IntArray): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (replace-non-coprimes nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Replace Non-Coprime Numbers in Array
* Difficulty: Hard
* Tags: array, math, stack
*
* 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:
vector<int> replaceNonCoprimes(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Replace Non-Coprime Numbers in Array
* Difficulty: Hard
* Tags: array, math, stack
*
* 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 List<Integer> replaceNonCoprimes(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Replace Non-Coprime Numbers in Array
Difficulty: Hard
Tags: array, math, stack

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Replace Non-Coprime Numbers in Array
* Difficulty: Hard
* Tags: array, math, stack
*
* 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 replaceNonCoprimes = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Replace Non-Coprime Numbers in Array
* Difficulty: Hard
* Tags: array, math, stack
*
* 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 replaceNonCoprimes(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Replace Non-Coprime Numbers in Array
* Difficulty: Hard
* Tags: array, math, stack
*
* 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 IList<int> ReplaceNonCoprimes(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Replace Non-Coprime Numbers in Array
* Difficulty: Hard
* Tags: array, math, stack
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* replaceNonCoprimes(int* nums, int numsSize, int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Replace Non-Coprime Numbers in Array
// Difficulty: Hard
// Tags: array, math, stack
//
// 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 replaceNonCoprimes(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun replaceNonCoprimes(nums: IntArray): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Replace Non-Coprime Numbers in Array
// Difficulty: Hard
// Tags: array, math, stack
//
// 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 replace_non_coprimes(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (replace-non-coprimes nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```