Problem 3378: Count Connected Components in LCM Graph

Problem Information

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

Problem Description

You are given an array of integers

nums

of size

n

and a

positive

integer

threshold

.

There is a graph consisting of

n

nodes with the

i

th

node having a value of

nums[i]

. Two nodes

i

and

j

in the graph are connected via an

undirected

edge if

lcm(nums[i], nums[j]) <= threshold

.

Return the number of

connected components

in this graph.

A

connected component

is a subgraph of a graph in which there exists a path between any two vertices, and no vertex of the subgraph shares an edge with a vertex outside of the subgraph.

The term

lcm(a, b)

denotes the

least common multiple

of

a

and

b

.

Example 1:

Input:

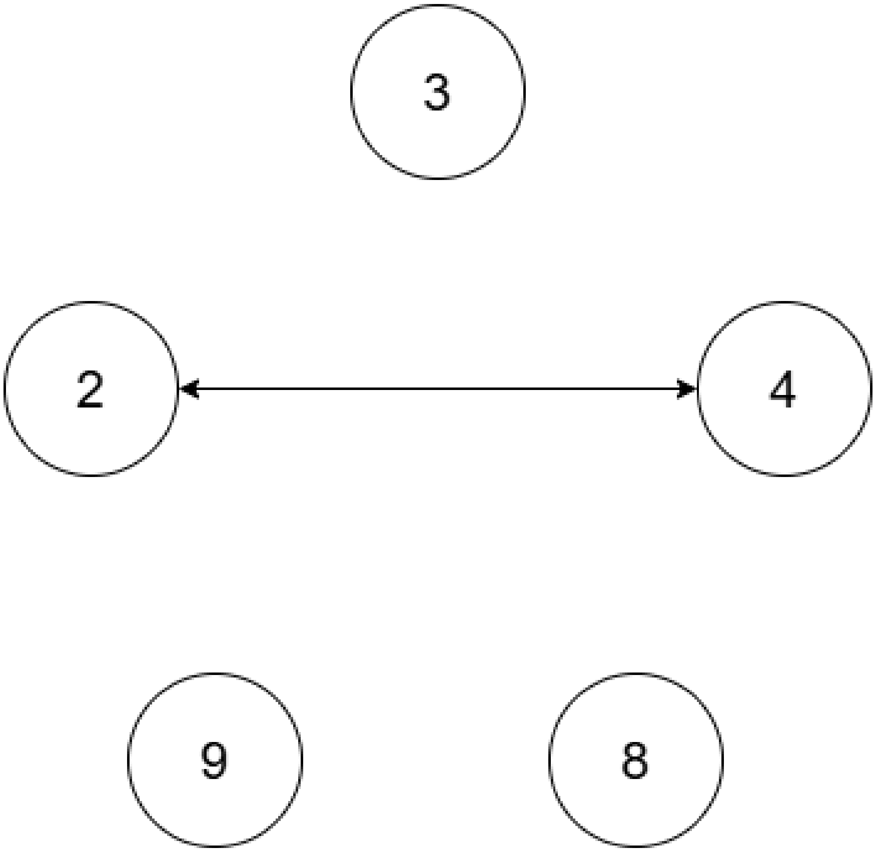
nums = [2,4,8,3,9], threshold = 5

Output:

4

Explanation:

)



The four connected components are

(2, 4)

,

(3)

,

(8)

,

(9)

.

Example 2:

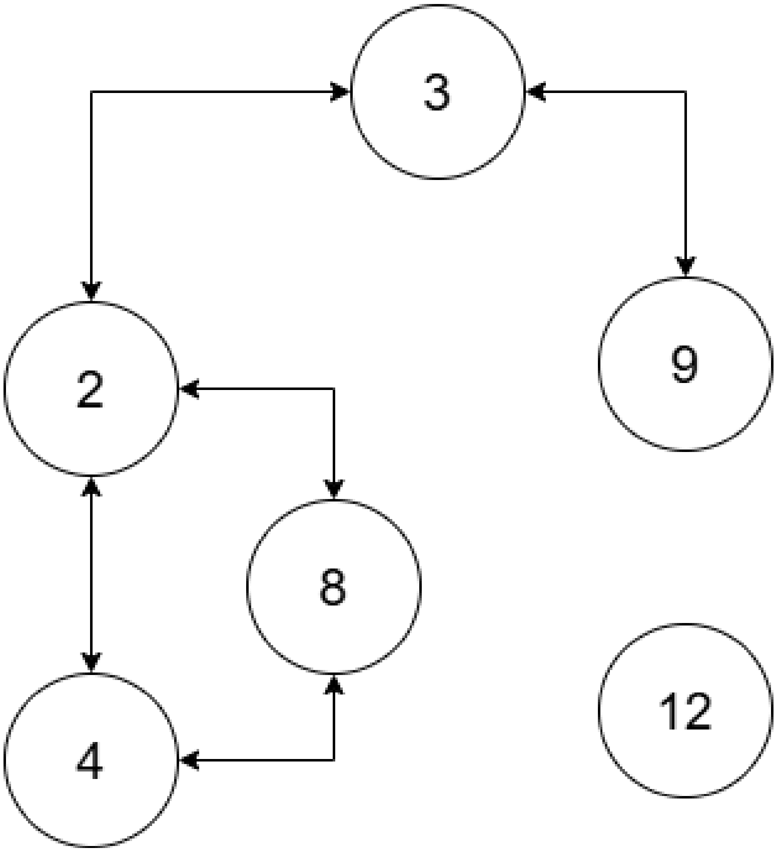
Input:

nums = [2,4,8,3,9,12], threshold = 10

Output:

2

Explanation:



The two connected components are

(2, 3, 4, 8, 9)

, and

(12)

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

All elements of

nums

are unique.

1 <= threshold <= 2 * 10

5

Code Snippets

C++:

```
class Solution {
public:
int countComponents(vector<int>& nums, int threshold) {

}
};
```

Java:

```
class Solution {
public int countComponents(int[] nums, int threshold) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countComponents(nums: number[], threshold: number): number {

};
```

C#:

```
public class Solution {
public int CountComponents(int[] nums, int threshold) {

}

}
```

C:

```
int countComponents(int* nums, int numsSize, int threshold) {

}
```

Go:

```
func countComponents(nums []int, threshold int) int {

}
```

Kotlin:

```
class Solution {
fun countComponents(nums: IntArray, threshold: Int): Int {

}
}
```

Swift:

```
class Solution {
func countComponents(_ nums: [Int], _ threshold: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_components(nums: Vec<i32>, threshold: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} threshold
# @return {Integer}
def count_components(nums, threshold)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $threshold
* @return Integer
*/
function countComponents($nums, $threshold) {

}
}
```

Dart:

```
class Solution {
int countComponents(List<int> nums, int threshold) {

}
}
```

Scala:

```
object Solution {
def countComponents(nums: Array[Int], threshold: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_components(nums :: [integer], threshold :: integer) :: integer
def count_components(nums, threshold) do

end
```

```
end
```

Erlang:

```
-spec count_components(Nums :: [integer()], Threshold :: integer()) ->
integer().
count_components(Nums, Threshold) ->
.
```

Racket:

```
(define/contract (count-components nums threshold)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Connected Components in LCM Graph
* Difficulty: Hard
* Tags: array, graph, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int countComponents(vector<int>& nums, int threshold) {

}
};
```

Java Solution:

```
/**
* Problem: Count Connected Components in LCM Graph
* Difficulty: Hard
* Tags: array, graph, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int countComponents(int[] nums, int threshold) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Connected Components in LCM Graph
Difficulty: Hard
Tags: array, graph, math, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Connected Components in LCM Graph
* Difficulty: Hard
* Tags: array, graph, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Connected Components in LCM Graph
* Difficulty: Hard
* Tags: array, graph, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function countComponents(nums: number[], threshold: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Connected Components in LCM Graph
* Difficulty: Hard
* Tags: array, graph, math, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int CountComponents(int[] nums, int threshold) {

}
}
```

C Solution:

```
/*
* Problem: Count Connected Components in LCM Graph
* Difficulty: Hard
* Tags: array, graph, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int countComponents(int* nums, int numsSize, int threshold) {

}
```

Go Solution:

```
// Problem: Count Connected Components in LCM Graph
// Difficulty: Hard
// Tags: array, graph, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countComponents(nums []int, threshold int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countComponents(nums: IntArray, threshold: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countComponents(_ nums: [Int], _ threshold: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Connected Components in LCM Graph
// Difficulty: Hard
// Tags: array, graph, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn count_components(nums: Vec<i32>, threshold: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} threshold
# @return {Integer}
def count_components(nums, threshold)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $threshold
* @return Integer
*/
function countComponents($nums, $threshold) {

}
}
```

Dart Solution:

```
class Solution {
int countComponents(List<int> nums, int threshold) {

}
}
```

Scala Solution:

```
object Solution {
def countComponents(nums: Array[Int], threshold: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_components(nums :: [integer], threshold :: integer) :: integer
def count_components(nums, threshold) do

end
end
```

Erlang Solution:

```
-spec count_components(Nums :: [integer()], Threshold :: integer()) ->
integer().
count_components(Nums, Threshold) ->
  .
```

Racket Solution:

```
(define/contract (count-components nums threshold)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```