Problem 3312: Sorted GCD Pair Queries

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

and an integer array

queries

.

Let

gcdPairs

denote an array obtained by calculating the

GCD

of all possible pairs

(nums[i], nums[j])

, where

0 <= i < j < n

, and then sorting these values in

ascending

order.

For each query

queries[i]

, you need to find the element at index

queries[i]

in

gcdPairs

.

Return an integer array

answer

, where

answer[i]

is the value at

gcdPairs[queries[i]]

for each query.

The term

gcd(a, b)

denotes the

greatest common divisor

of

a

and

b

.

Example 1:

Input:

nums = [2,3,4], queries = [0,2,2]

Output:

[1,2,2]

Explanation:

gcdPairs = [gcd(nums[0], nums[1]), gcd(nums[0], nums[2]), gcd(nums[1], nums[2])] = [1, 2, 1]

.

After sorting in ascending order,

gcdPairs = [1, 1, 2]

.

So, the answer is

[gcdPairs[queries[0]], gcdPairs[queries[1]], gcdPairs[queries[2]]] = [1, 2, 2]

.

Example 2:

Input:

nums = [4,4,2,1], queries = [5,3,1,0]

Output:

[4,2,1,1]

Explanation:

gcdPairs

sorted in ascending order is

[1, 1, 1, 2, 2, 4]

.

Example 3:

Input:

nums = [2,2], queries = [0,0]

Output:

[2,2]

Explanation:

gcdPairs = [2]

.

Constraints:

2 <= n == nums.length <= 10

5

1 <= nums[i] <= 5 * 10

4

1 <= queries.length <= 10

5

0 <= queries[i] < n * (n - 1) / 2

Code Snippets

C++:

```
class Solution {
public:
vector<int> gcdValues(vector<int>& nums, vector<long long>& queries) {

}
};
```

Java:

```
class Solution {
public int[] gcdValues(int[] nums, long[] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function gcdValues(nums: number[], queries: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] GcdValues(int[] nums, long[] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* gcdValues(int* nums, int numsSize, long long* queries, int queriesSize,
int* returnSize) {

}
```

Go:

```
func gcdValues(nums []int, queries []int64) []int {

}
```

Kotlin:

```
class Solution {
fun gcdValues(nums: IntArray, queries: LongArray): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn gcd_values(nums: Vec<i32>, queries: Vec<i64>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} queries
# @return {Integer[]}
def gcd_values(nums, queries)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $queries
* @return Integer[]
*/
function gcdValues($nums, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> gcdValues(List<int> nums, List<int> queries) {

}
}
```

Scala:

```
object Solution {
def gcdValues(nums: Array[Int], queries: Array[Long]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec gcd_values(nums :: [integer], queries :: [integer]) :: [integer]
def gcd_values(nums, queries) do

end
end
```

Erlang:

```
-spec gcd_values(Nums :: [integer()], Queries :: [integer()]) -> [integer()].
gcd_values(Nums, Queries) ->
.
```

Racket:

```
(define/contract (gcd-values nums queries)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sorted GCD Pair Queries
* Difficulty: Hard
* Tags: array, dp, math, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<int> gcdValues(vector<int>& nums, vector<long long>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Sorted GCD Pair Queries
* Difficulty: Hard
* Tags: array, dp, math, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

class Solution {
public int[] gcdValues(int[] nums, long[] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Sorted GCD Pair Queries
Difficulty: Hard
Tags: array, dp, math, hash, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sorted GCD Pair Queries
* Difficulty: Hard
* Tags: array, dp, math, hash, sort, search
```

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

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

};
```

TypeScript Solution:

```
/**
* Problem: Sorted GCD Pair Queries
* Difficulty: Hard
* Tags: array, dp, math, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function gcdValues(nums: number[], queries: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sorted GCD Pair Queries
* Difficulty: Hard
* Tags: array, dp, math, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

public class Solution {
public int[] GcdValues(int[] nums, long[] queries) {

}
}
```

C Solution:

```
/*
* Problem: Sorted GCD Pair Queries
* Difficulty: Hard
* Tags: array, dp, math, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* gcdValues(int* nums, int numsSize, long long* queries, int queriesSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Sorted GCD Pair Queries
// Difficulty: Hard
// Tags: array, dp, math, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func gcdValues(nums []int, queries []int64) []int {

}
```

Kotlin Solution:

```
class Solution {
fun gcdValues(nums: IntArray, queries: LongArray): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sorted GCD Pair Queries
// Difficulty: Hard
// Tags: array, dp, math, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn gcd_values(nums: Vec<i32>, queries: Vec<i64>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} queries
# @return {Integer[]}
def gcd_values(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $queries
* @return Integer[]
*/
function gcdValues($nums, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> gcdValues(List<int> nums, List<int> queries) {

}
}
```

Scala Solution:

```
object Solution {
def gcdValues(nums: Array[Int], queries: Array[Long]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec gcd_values(nums :: [integer], queries :: [integer]) :: [integer]
def gcd_values(nums, queries) do

end
end
```

Erlang Solution:

```
-spec gcd_values(Nums :: [integer()], Queries :: [integer()]) -> [integer()].
gcd_values(Nums, Queries) ->
.
```

Racket Solution:

```
(define/contract (gcd-values nums queries)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```