Problem 2521: Distinct Prime Factors of Product of Array

Problem Information

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

Problem Description

Given an array of positive integers

nums

, return

the number of

distinct prime factors

in the product of the elements of

nums

.

Note

that:

A number greater than

1

is called

prime

if it is divisible by only

1

and itself.

An integer

val1

is a factor of another integer

val2

if

val2 / val1

is an integer.

Example 1:

Input:

nums = [2,4,3,7,10,6]

Output:

4

Explanation:

The product of all the elements in nums is: 2 * 4 * 3 * 7 * 10 * 6 = 10080 = 2

5

* 3

2

* 5 * 7. There are 4 distinct prime factors so we return 4.

Example 2:

Input:

nums = [2,4,8,16]

Output:

1

Explanation:

The product of all the elements in nums is: 2 * 4 * 8 * 16 = 1024 = 2

10

. There is 1 distinct prime factor so we return 1.

Constraints:

1 <= nums.length <= 10

4

2 <= nums[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
```

```
int distinctPrimeFactors(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int distinctPrimeFactors(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int DistinctPrimeFactors(int[] nums) {

}
}
```

C:

```
int distinctPrimeFactors(int* nums, int numsSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun distinctPrimeFactors(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (distinct-prime-factors nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Distinct Prime Factors of Product of Array
* Difficulty: Medium
* Tags: array, 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 distinctPrimeFactors(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Distinct Prime Factors of Product of Array
```

```
* Difficulty: Medium
* Tags: array, 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 distinctPrimeFactors(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Distinct Prime Factors of Product of Array
Difficulty: Medium
Tags: array, 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 distinctPrimeFactors(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Distinct Prime Factors of Product of Array
* Difficulty: Medium
* Tags: array, 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
* @return {number}
*/
var distinctPrimeFactors = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Distinct Prime Factors of Product of Array
* Difficulty: Medium
* Tags: array, 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 distinctPrimeFactors(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Distinct Prime Factors of Product of Array
* Difficulty: Medium
* Tags: array, 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 DistinctPrimeFactors(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Distinct Prime Factors of Product of Array
* Difficulty: Medium
* Tags: array, 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 distinctPrimeFactors(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Distinct Prime Factors of Product of Array
// Difficulty: Medium
// Tags: array, 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 distinctPrimeFactors(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun distinctPrimeFactors(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Distinct Prime Factors of Product of Array
// Difficulty: Medium
// Tags: array, 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 distinct_prime_factors(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (distinct-prime-factors nums)
(-> (listof exact-integer?) exact-integer?)
)
```