Problem 3115: Maximum Prime Difference

Problem Information

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

Problem Description

You are given an integer array

nums

.

Return an integer that is the

maximum

distance between the

indices

of two (not necessarily different) prime numbers in

nums

.

Example 1:

Input:

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

Output:

3

Explanation:

nums[1]

,

nums[3]

, and

nums[4]

are prime. So the answer is

|4 - 1| = 3

.

Example 2:

Input:

nums = [4,8,2,8]

Output:

0

Explanation:

nums[2]

is prime. Because there is just one prime number, the answer is

|2 - 2| = 0

.

Constraints:

1 <= nums.length <= 3 * 10

5

1 <= nums[i] <= 100

The input is generated such that the number of prime numbers in the

nums

is at least one.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
```

```
function maximumPrimeDifference($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-prime-difference nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Prime Difference
* Difficulty: Medium
* Tags: array, math
*
* 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:
int maximumPrimeDifference(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Prime Difference
* Difficulty: Medium
* Tags: array, math
*
* 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 int maximumPrimeDifference(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Prime Difference
Difficulty: Medium
Tags: array, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Prime Difference
* Difficulty: Medium
* Tags: array, math
*
* 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 maximumPrimeDifference = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Prime Difference
* Difficulty: Medium
* Tags: array, math
*
* 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 maximumPrimeDifference(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Prime Difference
* Difficulty: Medium
* Tags: array, math
*
* 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 int MaximumPrimeDifference(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Prime Difference
* Difficulty: Medium
```

```
* Tags: array, math
*
* 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
*/

int maximumPrimeDifference(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Prime Difference
// Difficulty: Medium
// Tags: array, math
//
// 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 maximumPrimeDifference(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Prime Difference
// Difficulty: Medium
// Tags: array, math
//
// 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 maximum_prime_difference(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-prime-difference nums)
(-> (listof exact-integer?) exact-integer?)
)
```