Problem 2523: Closest Prime Numbers in Range

Problem Information

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

Problem Description

Given two positive integers

left

and

right

, find the two integers

num1

and

num2

such that:

left <= num1 < num2 <= right

.

Both

num1

and

num2

are

prime numbers

.

num2 - num1

is the

minimum

amongst all other pairs satisfying the above conditions.

Return the positive integer array

ans = [num1, num2]

. If there are multiple pairs satisfying these conditions, return the one with the

smallest

num1

value. If no such numbers exist, return

[-1, -1]

.

Example 1:

Input:

left = 10, right = 19

Output:

[11,13]

Explanation:

The prime numbers between 10 and 19 are 11, 13, 17, and 19. The closest gap between any pair is 2, which can be achieved by [11,13] or [17,19]. Since 11 is smaller than 17, we return the first pair.

Example 2:

Input:

left = 4, right = 6

Output:

[-1,-1]

Explanation:

There exists only one prime number in the given range, so the conditions cannot be satisfied.

Constraints:

1 <= left <= right <= 10

6

Code Snippets

C++:

```
class Solution {
public:
```

```
vector<int> closestPrimes(int left, int right) {

}
};
```

Java:

```
class Solution {
public int[] closestPrimes(int left, int right) {

}
}
```

Python3:

```
class Solution:
def closestPrimes(self, left: int, right: int) -> List[int]:
```

Python:

```
class Solution(object):
def closestPrimes(self, left, right):
"""
:type left: int
:type right: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} left
* @param {number} right
* @return {number[]}
*/
var closestPrimes = function(left, right) {

};
```

TypeScript:

```
function closestPrimes(left: number, right: number): number[] {

};
```

C#:

```
public class Solution {
public int[] ClosestPrimes(int left, int right) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* closestPrimes(int left, int right, int* returnSize) {

}
```

Go:

```
func closestPrimes(left int, right int) []int {

}
```

Kotlin:

```
class Solution {
fun closestPrimes(left: Int, right: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func closestPrimes(_ left: Int, _ right: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn closest_primes(left: i32, right: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} left
# @param {Integer} right
# @return {Integer[]}
def closest_primes(left, right)

end
```

PHP:

```
class Solution {

/**
* @param Integer $left
* @param Integer $right
* @return Integer[]
*/
function closestPrimes($left, $right) {

}
}
```

Dart:

```
class Solution {
List<int> closestPrimes(int left, int right) {

}
}
```

Scala:

```
object Solution {
def closestPrimes(left: Int, right: Int): Array[Int] = {



}
}
```

Elixir:

```
defmodule Solution do
@spec closest_primes(left :: integer, right :: integer) :: [integer]
def closest_primes(left, right) do

end
end
```

Erlang:

```
-spec closest_primes(Left :: integer(), Right :: integer()) -> [integer()].
closest_primes(Left, Right) ->
.
```

Racket:

```
(define/contract (closest-primes left right)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Closest Prime Numbers in Range
* 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:
vector<int> closestPrimes(int left, int right) {

}
};
```

Java Solution:

```
/**
* Problem: Closest Prime Numbers in Range
* 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[] closestPrimes(int left, int right) {

}
}
```

Python3 Solution:

```
"""
Problem: Closest Prime Numbers in Range
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 closestPrimes(self, left: int, right: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def closestPrimes(self, left, right):
"""
:type left: int
:type right: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Closest Prime Numbers in Range
* 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} left
* @param {number} right
* @return {number[]}
*/
var closestPrimes = function(left, right) {

};
```

TypeScript Solution:

```
/**
* Problem: Closest Prime Numbers in Range
* 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 closestPrimes(left: number, right: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Closest Prime Numbers in Range
* 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[] ClosestPrimes(int left, int right) {

}
}
```

C Solution:

```
/*
* Problem: Closest Prime Numbers in Range
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* closestPrimes(int left, int right, int* returnSize) {

}
```

Go Solution:

```
// Problem: Closest Prime Numbers in Range
// 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 closestPrimes(left int, right int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun closestPrimes(left: Int, right: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func closestPrimes(_ left: Int, _ right: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Closest Prime Numbers in Range
// 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 closest_primes(left: i32, right: i32) -> Vec<i32> {
}
}
```

Ruby Solution:

```
# @param {Integer} left
# @param {Integer} right
# @return {Integer[]}
def closest_primes(left, right)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $left
* @param Integer $right
* @return Integer[]
*/
function closestPrimes($left, $right) {

}
}
```

Dart Solution:

```
class Solution {
List<int> closestPrimes(int left, int right) {

}
}
```

Scala Solution:

```
object Solution {
def closestPrimes(left: Int, right: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec closest_primes(left :: integer, right :: integer) :: [integer]
def closest_primes(left, right) do

end
end
```

Erlang Solution:

```
-spec closest_primes(Left :: integer(), Right :: integer()) -> [integer()].
closest_primes(Left, Right) ->
.
```

Racket Solution:

```
(define/contract (closest-primes left right)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```