Problem 2761: Prime Pairs With Target Sum

Problem Information

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

Problem Description

You are given an integer

n

. We say that two integers

x

and

y

form a prime number pair if:

1 <= x <= y <= n

x + y == n

x

and

y

are prime numbers

Return

the 2D sorted list of prime number pairs

[x

i

, y

i

]

. The list should be sorted in

increasing

order of

x

i

. If there are no prime number pairs at all, return

an empty array

.

Note:

A prime number is a natural number greater than

1

with only two factors, itself and

1

.

Example 1:

Input:

n = 10

Output:

[[3,7],[5,5]]

Explanation:

In this example, there are two prime pairs that satisfy the criteria. These pairs are [3,7] and [5,5], and we return them in the sorted order as described in the problem statement.

Example 2:

Input:

n = 2

Output:

[]

Explanation:

We can show that there is no prime number pair that gives a sum of 2, so we return an empty array.

Constraints:

1 <= n <= 10

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> findPrimePairs(int n) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> findPrimePairs(int n) {

}
}
```

Python3:

```
class Solution:
def findPrimePairs(self, n: int) -> List[List[int]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findPrimePairs(n: number): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> FindPrimePairs(int n) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** findPrimePairs(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func findPrimePairs(n int) [][]int {

}
```

Kotlin:

```
class Solution {
fun findPrimePairs(n: Int): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func findPrimePairs(_ n: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn find_prime_pairs(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer[][]}
def find_prime_pairs(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function findPrimePairs($n) {

}
}
```

Dart:

```
class Solution {
List<List<int>> findPrimePairs(int n) {

}
}
```

Scala:

```
object Solution {
def findPrimePairs(n: Int): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_prime_pairs(n :: integer) :: [[integer]]
def find_prime_pairs(n) do

end
end
```

Erlang:

```
-spec find_prime_pairs(N :: integer()) -> [[integer()]].
find_prime_pairs(N) ->
.
```

Racket:

```
(define/contract (find-prime-pairs n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Prime Pairs With Target Sum
* Difficulty: Medium
* Tags: array, math, sort
*
* 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<vector<int>> findPrimePairs(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Prime Pairs With Target Sum
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 List<List<Integer>> findPrimePairs(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Prime Pairs With Target Sum
Difficulty: Medium
Tags: array, math, sort

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 findPrimePairs(self, n: int) -> List[List[int]]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Prime Pairs With Target Sum
* Difficulty: Medium
* Tags: array, math, sort
*
* 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} n
* @return {number[][]}
*/
var findPrimePairs = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Prime Pairs With Target Sum
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 findPrimePairs(n: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Prime Pairs With Target Sum
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 IList<IList<int>> FindPrimePairs(int n) {

}
}
```

C Solution:

```
/*
* Problem: Prime Pairs With Target Sum
* Difficulty: Medium
* Tags: array, math, sort
*
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
```

```
*/
int** findPrimePairs(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Prime Pairs With Target Sum
// Difficulty: Medium
// Tags: array, math, sort
//
// 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 findPrimePairs(n int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun findPrimePairs(n: Int): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func findPrimePairs(_ n: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Prime Pairs With Target Sum
// Difficulty: Medium
// Tags: array, math, sort
//
```

```
// 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 find_prime_pairs(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer[][]}
def find_prime_pairs(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function findPrimePairs($n) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> findPrimePairs(int n) {

}
}
```

Scala Solution:

```
object Solution {
def findPrimePairs(n: Int): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_prime_pairs(n :: integer) :: [[integer]]
def find_prime_pairs(n) do

end
end
```

Erlang Solution:

```
-spec find_prime_pairs(N :: integer()) -> [[integer()]].
find_prime_pairs(N) ->
.
```

Racket Solution:

```
(define/contract (find-prime-pairs n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```