Problem 866: Prime Palindrome

Problem Information

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

Problem Description

Given an integer n, return

the smallest

prime palindrome

greater than or equal to

n

.

An integer is

prime

if it has exactly two divisors:

1

and itself. Note that

1

is not a prime number.

For example,

2

,

3

,

5

,

7

,

11

, and

13

are all primes.

An integer is a

palindrome

if it reads the same from left to right as it does from right to left.

For example,

101

and

12321

are palindromes.

The test cases are generated so that the answer always exists and is in the range

[2, 2 * 10

8

]

.

Example 1:

Input:

n = 6

Output:

7

Example 2:

Input:

n = 8

Output:

11

Example 3:

Input:

n = 13

Output:

101

Constraints:

1 <= n <= 10

8

Code Snippets

C++:

```
class Solution {
public:
int primePalindrome(int n) {

}
};
```

Java:

```
class Solution {
public int primePalindrome(int n) {

}
}
```

Python3:

```
class Solution:
def primePalindrome(self, n: int) -> int:
```

Python:

```
class Solution(object):
def primePalindrome(self, n):
"""
:type n: int
```

```
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var primePalindrome = function(n) {

};
```

TypeScript:

```
function primePalindrome(n: number): number {

};
```

C#:

```
public class Solution {
public int PrimePalindrome(int n) {

}
}
```

C:

```
int primePalindrome(int n) {

}
```

Go:

```
func primePalindrome(n int) int {

}
```

Kotlin:

```
class Solution {
fun primePalindrome(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func primePalindrome(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn prime_palindrome(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def prime_palindrome(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function primePalindrome($n) {

}
}
```

Dart:

```
class Solution {
int primePalindrome(int n) {

}
}
```

Scala:

```
object Solution {
def primePalindrome(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec prime_palindrome(n :: integer) :: integer
def prime_palindrome(n) do

end
end
```

Erlang:

```
-spec prime_palindrome(N :: integer()) -> integer().
prime_palindrome(N) ->
.
```

Racket:

```
(define/contract (prime-palindrome n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Prime Palindrome
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int primePalindrome(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Prime Palindrome
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int primePalindrome(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Prime Palindrome
Difficulty: Medium
Tags: string, math
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def primePalindrome(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def primePalindrome(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Prime Palindrome
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* 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 primePalindrome = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Prime Palindrome
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function primePalindrome(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Prime Palindrome
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int PrimePalindrome(int n) {

}
}
```

C Solution:

```
/*
* Problem: Prime Palindrome
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

int primePalindrome(int n) {

}
```

Go Solution:

```
// Problem: Prime Palindrome
// Difficulty: Medium
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func primePalindrome(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun primePalindrome(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func primePalindrome(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Prime Palindrome
// Difficulty: Medium
// Tags: string, math
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn prime_palindrome(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def prime_palindrome(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function primePalindrome($n) {

}
}
```

Dart Solution:

```
class Solution {
int primePalindrome(int n) {

}
}
```

Scala Solution:

```
object Solution {
def primePalindrome(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec prime_palindrome(n :: integer) :: integer
def prime_palindrome(n) do

end
end
```

Erlang Solution:

```
-spec prime_palindrome(N :: integer()) -> integer().
prime_palindrome(N) ->
.
```

Racket Solution:

```
(define/contract (prime-palindrome n)
(-> exact-integer? exact-integer?)
)
```