Problem 479: Largest Palindrome Product

Problem Information

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

Problem Description

Given an integer n, return

the

largest palindromic integer

that can be represented as the product of two

n

-digits integers

. Since the answer can be very large, return it

modulo

1337

.

Example 1:

Input:

n = 2

Output:

987 Explanation: 99 x 91 = 9009, 9009 % 1337 = 987

Example 2:

Input:

n = 1

Output:

9

Constraints:

1 <= n <= 8

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int largestPalindrome(int n) {

}
```

Go:

```
func largestPalindrome(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
```

```
*/
function largestPalindrome($n) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Largest Palindrome Product
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Largest Palindrome Product
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int largestPalindrome(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Palindrome Product
Difficulty: Hard
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Palindrome Product
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: Largest Palindrome Product
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function largestPalindrome(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Largest Palindrome Product
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Largest Palindrome Product
* Difficulty: Hard
```

```
 * Tags: math
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

int largestPalindrome(int n) {

}
```

Go Solution:

```
// Problem: Largest Palindrome Product
// Difficulty: Hard
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func largestPalindrome(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Largest Palindrome Product
// Difficulty: Hard
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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