Problem 2217: Find Palindrome With Fixed Length

Problem Information

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

Problem Description

Given an integer array

queries

and a

positive

integer

intLength

, return

an array

answer

where

answer[i]

is either the

queries[i]

th

smallest

positive palindrome

of length

intLength

or

-1

if no such palindrome exists

.

A

palindrome

is a number that reads the same backwards and forwards. Palindromes cannot have leading zeros.

Example 1:

Input:

queries = [1,2,3,4,5,90], intLength = 3

Output:

[101,111,121,131,141,999]

Explanation:

The first few palindromes of length 3 are: 101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 202, ... The 90 th

palindrome of length 3 is 999.

Example 2:

Input:

queries = [2,4,6], intLength = 4

Output:

[1111,1331,1551]

Explanation:

The first six palindromes of length 4 are: 1001, 1111, 1221, 1331, 1441, and 1551.

Constraints:

1 <= queries.length <= 5 * 10

4

1 <= queries[i] <= 10

9

1 <= intLength <= 15

Code Snippets

C++:

```
class Solution {
public:
vector<long long> kthPalindrome(vector<int>& queries, int intLength) {
}
};
```

Java:

```
class Solution {
public long[] kthPalindrome(int[] queries, int intLength) {

}
}
```

Python3:

```
class Solution:
def kthPalindrome(self, queries: List[int], intLength: int) -> List[int]:
```

Python:

```
class Solution(object):
def kthPalindrome(self, queries, intLength):
"""
:type queries: List[int]
:type intLength: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} queries
* @param {number} intLength
* @return {number[]}
*/
var kthPalindrome = function(queries, intLength) {

};
```

TypeScript:

```
function kthPalindrome(queries: number[], intLength: number): number[] {

};
```

C#:

```
public class Solution {
public long[] KthPalindrome(int[] queries, int intLength) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* kthPalindrome(int* queries, int queriesSize, int intLength, int*
returnSize) {

}
```

Go:

```
func kthPalindrome(queries []int, intLength int) []int64 {

}
```

Kotlin:

```
class Solution {
fun kthPalindrome(queries: IntArray, intLength: Int): LongArray {

}
}
```

Swift:

```
class Solution {
func kthPalindrome(_ queries: [Int], _ intLength: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn kth_palindrome(queries: Vec<i32>, int_length: i32) -> Vec<i64> {

}
}
```

Ruby:

```
# @param {Integer[]} queries
# @param {Integer} int_length
# @return {Integer[]}
def kth_palindrome(queries, int_length)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $queries
* @param Integer $intLength
* @return Integer[]
*/
function kthPalindrome($queries, $intLength) {

}
}
```

Dart:

```
class Solution {
List<int> kthPalindrome(List<int> queries, int intLength) {

}
}
```

Scala:

```
object Solution {
def kthPalindrome(queries: Array[Int], intLength: Int): Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec kth_palindrome(queries :: [integer], int_length :: integer) ::
[integer]
def kth_palindrome(queries, int_length) do

end
end
```

Erlang:

```
-spec kth_palindrome(Queries :: [integer()], IntLength :: integer()) ->
[integer()].
kth_palindrome(Queries, IntLength) ->
.
```

Racket:

```
(define/contract (kth-palindrome queries intLength)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Palindrome With Fixed Length
* Difficulty: Medium
* Tags: array, string, 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<long long> kthPalindrome(vector<int>& queries, int intLength) {

}
};
```

Java Solution:

```
/**
* Problem: Find Palindrome With Fixed Length
* Difficulty: Medium
* Tags: array, string, 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 long[] kthPalindrome(int[] queries, int intLength) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Palindrome With Fixed Length
Difficulty: Medium
Tags: array, string, 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 kthPalindrome(self, queries: List[int], intLength: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def kthPalindrome(self, queries, intLength):
"""
:type queries: List[int]
:type intLength: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find Palindrome With Fixed Length
* Difficulty: Medium
* Tags: array, string, 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[]} queries
* @param {number} intLength
* @return {number[]}
*/
var kthPalindrome = function(queries, intLength) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Palindrome With Fixed Length
* Difficulty: Medium
* Tags: array, string, 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 kthPalindrome(queries: number[], intLength: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Palindrome With Fixed Length
* Difficulty: Medium
* Tags: array, string, 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 long[] KthPalindrome(int[] queries, int intLength) {

}
}
```

C Solution:

```
/*
* Problem: Find Palindrome With Fixed Length
* Difficulty: Medium
* Tags: array, string, 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().
*/
long long* kthPalindrome(int* queries, int queriesSize, int intLength, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Find Palindrome With Fixed Length
// Difficulty: Medium
// Tags: array, string, 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 kthPalindrome(queries []int, intLength int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun kthPalindrome(queries: IntArray, intLength: Int): LongArray {

}
}
```

Swift Solution:

```
class Solution {
func kthPalindrome(_ queries: [Int], _ intLength: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Palindrome With Fixed Length
// Difficulty: Medium
// Tags: array, string, 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 kth_palindrome(queries: Vec<i32>, int_length: i32) -> Vec<i64> {
}
}
```

Ruby Solution:

```
# @param {Integer[]} queries
# @param {Integer} int_length
# @return {Integer[]}
def kth_palindrome(queries, int_length)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $queries
* @param Integer $intLength
* @return Integer[]
*/
function kthPalindrome($queries, $intLength) {

}
}
```

Dart Solution:

```
class Solution {
List<int> kthPalindrome(List<int> queries, int intLength) {

}
}
```

Scala Solution:

```
object Solution {
def kthPalindrome(queries: Array[Int], intLength: Int): Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec kth_palindrome(queries :: [integer], int_length :: integer) ::
[integer]
def kth_palindrome(queries, int_length) do

end
end
```

Erlang Solution:

```
-spec kth_palindrome(Queries :: [integer()], IntLength :: integer()) ->
[integer()].
kth_palindrome(Queries, IntLength) ->
.
```

Racket Solution:

```
(define/contract (kth-palindrome queries intLength)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```