Problem 3179: Find the N-th Value After K Seconds

Problem Information

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

Problem Description

You are given two integers

n

and

k

.

Initially, you start with an array

a

of

n

integers where

a[i] = 1

for all

0 <= i <= n - 1

. After each second, you simultaneously update each element to be the sum of all its preceding elements plus the element itself. For example, after one second,

a[0]

remains the same,

a[1]

becomes

a[0] + a[1]

,

a[2]

becomes

a[0] + a[1] + a[2]

, and so on.

Return the

value

of

a[n - 1]

after

k

seconds.

Since the answer may be very large, return it modulo

10

9

+ 7

.

Example 1:

Input:

n = 4, k = 5

Output:

56

Explanation:

Second

State After

0

[1,1,1,1]

1

[1,2,3,4]

2

[1,3,6,10]

3

[1,4,10,20]

4

[1,5,15,35]

5

[1,6,21,56]

Example 2:

Input:

n = 5, k = 3

Output:

35

Explanation:

Second

State After

0

[1,1,1,1,1]

1

[1,2,3,4,5]

2

[1,3,6,10,15]

3

[1,4,10,20,35]

Constraints:

1 <= n, k <= 1000

Code Snippets

C++:

```
class Solution {
public:
int valueAfterKSeconds(int n, int k) {

}
};
```

Java:

```
class Solution {
public int valueAfterKSeconds(int n, int k) {

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var valueAfterKSeconds = function(n, k) {

};
```

TypeScript:

```
function valueAfterKSeconds(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int ValueAfterKSeconds(int n, int k) {

}
}
```

C:

```
int valueAfterKSeconds(int n, int k) {

}
```

Go:

```
func valueAfterKSeconds(n int, k int) int {

}
```

Kotlin:

```
class Solution {
fun valueAfterKSeconds(n: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func valueAfterKSeconds(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn value_after_k_seconds(n: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def value_after_k_seconds(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function valueAfterKSeconds($n, $k) {

}
}
```

Dart:

```
class Solution {
int valueAfterKSeconds(int n, int k) {

}
}
```

Scala:

```
object Solution {
def valueAfterKSeconds(n: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec value_after_k_seconds(n :: integer, k :: integer) :: integer
def value_after_k_seconds(n, k) do

end
end
```

Erlang:

```
-spec value_after_k_seconds(N :: integer(), K :: integer()) -> integer().
value_after_k_seconds(N, K) ->
.
```

Racket:

```
(define/contract (value-after-k-seconds n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the N-th Value After K Seconds
* 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 valueAfterKSeconds(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the N-th Value After K Seconds
* 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 valueAfterKSeconds(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the N-th Value After K Seconds
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 valueAfterKSeconds(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the N-th Value After K Seconds
* 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} n
* @param {number} k
* @return {number}
*/
var valueAfterKSeconds = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the N-th Value After K Seconds
* 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 valueAfterKSeconds(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the N-th Value After K Seconds
* 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 ValueAfterKSeconds(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the N-th Value After K Seconds
* 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
*/

int valueAfterKSeconds(int n, int k) {

}
```

Go Solution:

```
// Problem: Find the N-th Value After K Seconds
// 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 valueAfterKSeconds(n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun valueAfterKSeconds(n: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func valueAfterKSeconds(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the N-th Value After K Seconds
// 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 value_after_k_seconds(n: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def value_after_k_seconds(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function valueAfterKSeconds($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
int valueAfterKSeconds(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def valueAfterKSeconds(n: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec value_after_k_seconds(n :: integer, k :: integer) :: integer
def value_after_k_seconds(n, k) do

end
end
```

Erlang Solution:

```
-spec value_after_k_seconds(N :: integer(), K :: integer()) -> integer().
value_after_k_seconds(N, K) ->
.
```

Racket Solution:

```
(define/contract (value-after-k-seconds n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```