Problem 629: K Inverse Pairs Array

Problem Information

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

Problem Description

For an integer array

nums

, an

inverse pair

is a pair of integers

[i, j]

where

0 <= i < j < nums.length

and

nums[i] > nums[j]

.

Given two integers n and k, return the number of different arrays consisting of numbers from

1

to

n

such that there are exactly

k

inverse pairs

. Since the answer can be huge, return it

modulo

10

9

+ 7

.

Example 1:

Input:

n = 3, k = 0

Output:

1

Explanation:

Only the array [1,2,3] which consists of numbers from 1 to 3 has exactly 0 inverse pairs.

Example 2:

Input:

n = 3, k = 1

Output:

2

Explanation:

The array [1,3,2] and [2,1,3] have exactly 1 inverse pair.

Constraints:

1 <= n <= 1000

0 <= k <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (k-inverse-pairs n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: K Inverse Pairs Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int kInversePairs(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K Inverse Pairs Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int kInversePairs(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: K Inverse Pairs Array
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: K Inverse Pairs Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: K Inverse Pairs Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function kInversePairs(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: K Inverse Pairs Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int KInversePairs(int n, int k) {

}

}
```

C Solution:

```
/*
* Problem: K Inverse Pairs Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int kInversePairs(int n, int k) {

}
```

Go Solution:

```
// Problem: K Inverse Pairs Array
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func kInversePairs(n int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: K Inverse Pairs Array
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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


}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (k-inverse-pairs n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```