Problem 2407: Longest Increasing Subsequence II

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

Find the longest subsequence of

nums

that meets the following requirements:

The subsequence is

strictly increasing

and

The difference between adjacent elements in the subsequence is

at most

k

.

Return

the length of the

longest

subsequence

that meets the requirements.

A

subsequence

is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input:

nums = [4,2,1,4,3,4,5,8,15], k = 3

Output:

5

Explanation:

The longest subsequence that meets the requirements is [1,3,4,5,8]. The subsequence has a length of 5, so we return 5. Note that the subsequence [1,3,4,5,8,15] does not meet the requirements because 15 - 8 = 7 is larger than 3.

Example 2:

Input:

nums = [7,4,5,1,8,12,4,7], k = 5

Output:

4

Explanation:

The longest subsequence that meets the requirements is [4,5,8,12]. The subsequence has a length of 4, so we return 4.

Example 3:

Input:

nums = [1,5], k = 1

Output:

1

Explanation:

The longest subsequence that meets the requirements is [1]. The subsequence has a length of 1, so we return 1.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i], k <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int lengthOfLIS(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int lengthOfLIS(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def lengthOfLIS(self, nums: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def lengthOfLIS(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var lengthOfLIS = function(nums, k) {

};
```

TypeScript:

```
function lengthOfLIS(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int LengthOfLIS(int[] nums, int k) {

}
}
```

C:

```
int lengthOfLIS(int* nums, int numsSize, int k) {

}
```

Go:

```
func lengthOfLIS(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun lengthOfLIS(nums: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func lengthOfLIS(_ nums: [Int], _ k: Int) -> Int {

}

}
```

Rust:

```
impl Solution {
pub fn length_of_lis(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def length_of_lis(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function lengthOfLIS($nums, $k) {

}
}
```

Dart:

```
class Solution {
int lengthOfLIS(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def lengthOfLIS(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec length_of_lis(nums :: [integer], k :: integer) :: integer
def length_of_lis(nums, k) do

end
end
```

Erlang:

```
-spec length_of_lis(Nums :: [integer()], K :: integer()) -> integer().
length_of_lis(Nums, K) ->
.
```

Racket:

```
(define/contract (length-of-lis nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Increasing Subsequence II
* Difficulty: Hard
* Tags: array, tree, dp, queue
*
* 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 lengthOfLIS(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Increasing Subsequence II
* Difficulty: Hard
* Tags: array, tree, dp, queue
*
* 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 lengthOfLIS(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Increasing Subsequence II
Difficulty: Hard
Tags: array, tree, dp, queue

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 lengthOfLIS(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def lengthOfLIS(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Increasing Subsequence II
* Difficulty: Hard
* Tags: array, tree, dp, queue
*
* 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[]} nums
* @param {number} k
* @return {number}
*/
var lengthOfLIS = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Increasing Subsequence II
* Difficulty: Hard
* Tags: array, tree, dp, queue
*
* 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 lengthOfLIS(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Longest Increasing Subsequence II
* Difficulty: Hard
* Tags: array, tree, dp, queue
*
* 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 LengthOfLIS(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Longest Increasing Subsequence II
* Difficulty: Hard
* Tags: array, tree, dp, queue
*
* 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 lengthOfLIS(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Longest Increasing Subsequence II
// Difficulty: Hard
// Tags: array, tree, dp, queue
//
// 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 lengthOfLIS(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun lengthOfLIS(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func lengthOfLIS(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Increasing Subsequence II
// Difficulty: Hard
// Tags: array, tree, dp, queue
//
// 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 length_of_lis(nums: Vec<i32>, k: i32) -> i32 {

}


}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def length_of_lis(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function lengthOfLIS($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int lengthOfLIS(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def lengthOfLIS(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec length_of_lis(nums :: [integer], k :: integer) :: integer
def length_of_lis(nums, k) do

end
end
```

Erlang Solution:

```
-spec length_of_lis(Nums :: [integer()], K :: integer()) -> integer().
length_of_lis(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (length-of-lis nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```