Problem 1121: Divide Array Into Increasing Sequences

Problem Information

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

Problem Description

Given an integer array

nums

sorted in non-decreasing order and an integer

k

, return

true

if this array can be divided into one or more disjoint increasing subsequences of length at least

k

, or

false

otherwise

.

Example 1:

Input:

nums = [1,2,2,3,3,4,4], k = 3

Output:

true

Explanation:

The array can be divided into two subsequences [1,2,3,4] and [2,3,4] with lengths at least 3 each.

Example 2:

Input:

nums = [5,6,6,7,8], k = 3

Output:

false

Explanation:

There is no way to divide the array using the conditions required.

Constraints:

1 <= k <= nums.length <= 10

5

1 <= nums[i] <= 10

5

nums

is sorted in non-decreasing order.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
bool canDivideIntoSubsequences(int* nums, int numsSize, int k) {

}
```

Go:

```
func canDivideIntoSubsequences(nums []int, k int) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (can-divide-into-subsequences nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide Array Into Increasing Sequences
* Difficulty: Hard
* Tags: array, sort
*
* 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:
bool canDivideIntoSubsequences(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Divide Array Into Increasing Sequences
* Difficulty: Hard
* Tags: array, sort
*
* 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 boolean canDivideIntoSubsequences(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Divide Array Into Increasing Sequences
Difficulty: Hard
Tags: array, sort
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide Array Into Increasing Sequences
* Difficulty: Hard
* Tags: array, sort
*
* 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[]} nums
* @param {number} k
* @return {boolean}
*/
var canDivideIntoSubsequences = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Divide Array Into Increasing Sequences
* Difficulty: Hard
* Tags: array, sort
*
* 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 canDivideIntoSubsequences(nums: number[], k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Divide Array Into Increasing Sequences
* Difficulty: Hard
* Tags: array, sort
*
* 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 bool CanDivideIntoSubsequences(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Divide Array Into Increasing Sequences
* Difficulty: Hard
* Tags: array, sort
*
* 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
*/

bool canDivideIntoSubsequences(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Divide Array Into Increasing Sequences
// Difficulty: Hard
// Tags: array, sort
//
// 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 canDivideIntoSubsequences(nums []int, k int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Divide Array Into Increasing Sequences
// Difficulty: Hard
```

```
// Tags: array, sort
//
// 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 can_divide_into_subsequences(nums: Vec<i32>, k: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
```

```
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-divide-into-subsequences nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```