Problem 1296: Divide Array in Sets of K Consecutive Numbers

Problem Information

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

Problem Description

Given an array of integers

nums

and a positive integer

k

, check whether it is possible to divide this array into sets of

k

consecutive numbers.

Return

true

if it is possible

.

Otherwise, return

false

.

Example 1:

Input:

nums = [1,2,3,3,4,4,5,6], k = 4

Output:

true

Explanation:

Array can be divided into [1,2,3,4] and [3,4,5,6].

Example 2:

Input:

nums = [3,2,1,2,3,4,3,4,5,9,10,11], k = 3

Output:

true

Explanation:

Array can be divided into [1,2,3] , [2,3,4] , [3,4,5] and [9,10,11].

Example 3:

Input:

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

Output:

false

Explanation:

Each array should be divided in subarrays of size 3.

Constraints:

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

5

1 <= nums[i] <= 10

9

Note:

This question is the same as 846:

https://leetcode.com/problems/hand-of-straights/

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-possible-divide nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide Array in Sets of K Consecutive Numbers
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Divide Array in Sets of K Consecutive Numbers
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Divide Array in Sets of K Consecutive Numbers
Difficulty: Medium
Tags: array, greedy, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide Array in Sets of K Consecutive Numbers
* Difficulty: Medium
* Tags: array, greedy, hash, sort
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Divide Array in Sets of K Consecutive Numbers
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

C# Solution:

```
/*
* Problem: Divide Array in Sets of K Consecutive Numbers
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

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

}
}
```

C Solution:

```
/*
* Problem: Divide Array in Sets of K Consecutive Numbers
* Difficulty: Medium
* Tags: array, greedy, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
```

Go Solution:

```
// Problem: Divide Array in Sets of K Consecutive Numbers
// Difficulty: Medium
// Tags: array, greedy, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func isPossibleDivide(nums []int, k int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Divide Array in Sets of K Consecutive Numbers
// Difficulty: Medium
// Tags: array, greedy, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-possible-divide nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```