Problem 2966: Divide Array Into Arrays With Max Difference

Problem Information

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

Problem Description

You are given an integer array

nums

of size

n

where

n

is a multiple of 3 and a positive integer

k

.

Divide the array

nums

into

n / 3

arrays of size

3

satisfying the following condition:

The difference between

any

two elements in one array is

less than or equal

to

k

.

Return a

2D

array containing the arrays. If it is impossible to satisfy the conditions, return an empty array. And if there are multiple answers, return

any

of them.

Example 1:

Input:

nums = [1,3,4,8,7,9,3,5,1], k = 2

Output:

[[1,1,3],[3,4,5],[7,8,9]]

Explanation:

The difference between any two elements in each array is less than or equal to 2.

Example 2:

Input:

nums = [2,4,2,2,5,2], k = 2

Output:

[]

Explanation:

Different ways to divide

nums

into 2 arrays of size 3 are:

[[2,2,2],[2,4,5]] (and its permutations)

[[2,2,4],[2,2,5]] (and its permutations)

Because there are four 2s there will be an array with the elements 2 and 5 no matter how we divide it. since

5 - 2 = 3 > k

, the condition is not satisfied and so there is no valid division.

Example 3:

Input:

nums = [4,2,9,8,2,12,7,12,10,5,8,5,5,7,9,2,5,11], k = 14

Output:

[[2,2,2],[4,5,5],[5,5,7],[7,8,8],[9,9,10],[11,12,12]]

Explanation:

The difference between any two elements in each array is less than or equal to 14.

Constraints:

n == nums.length

1 <= n <= 10

5

n

is a multiple of 3

1 <= nums[i] <= 10

5

1 <= k <= 10

5

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** divideArray(int* nums, int numsSize, int k, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun divideArray(nums: IntArray, k: Int): Array<IntArray> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

```
}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (divide-array nums k)
(-> (listof exact-integer?) exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide Array Into Arrays With Max Difference
* Difficulty: Medium
* Tags: array, greedy, 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:
vector<vector<int>> divideArray(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Divide Array Into Arrays With Max Difference
* Difficulty: Medium
* Tags: array, greedy, 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 int[][] divideArray(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Divide Array Into Arrays With Max Difference
Difficulty: Medium
Tags: array, greedy, 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 divideArray(self, nums: List[int], k: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide Array Into Arrays With Max Difference
* Difficulty: Medium
* Tags: array, greedy, 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 {number[][]}
*/
var divideArray = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Divide Array Into Arrays With Max Difference
* Difficulty: Medium
* Tags: array, greedy, 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 divideArray(nums: number[], k: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Divide Array Into Arrays With Max Difference
* Difficulty: Medium
* Tags: array, greedy, 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 int[][] DivideArray(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Divide Array Into Arrays With Max Difference
* Difficulty: Medium
* Tags: array, greedy, 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** divideArray(int* nums, int numsSize, int k, int* returnSize, int**
```

```
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Divide Array Into Arrays With Max Difference
// Difficulty: Medium
// Tags: array, greedy, 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 divideArray(nums []int, k int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun divideArray(nums: IntArray, k: Int): Array<IntArray> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Divide Array Into Arrays With Max Difference
// Difficulty: Medium
// Tags: array, greedy, 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 divide_array(nums: Vec<i32>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (divide-array nums k)
(-> (listof exact-integer?) exact-integer? (listof (listof exact-integer?)))
)
```