Problem 1497: Check If Array Pairs Are Divisible by k

Problem Information

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

Problem Description

Given an array of integers

arr

of even length

n

and an integer

k

.

We want to divide the array into exactly

n / 2

pairs such that the sum of each pair is divisible by

k

.

Return

true

If you can find a way to do that or

false

otherwise

.

Example 1:

Input:

arr = [1,2,3,4,5,10,6,7,8,9], k = 5

Output:

true

Explanation:

Pairs are (1,9),(2,8),(3,7),(4,6) and (5,10).

Example 2:

Input:

arr = [1,2,3,4,5,6], k = 7

Output:

true

Explanation:

Pairs are (1,6),(2,5) and(3,4).

Example 3:

Input:

arr = [1,2,3,4,5,6], k = 10

Output:

false

Explanation:

You can try all possible pairs to see that there is no way to divide arr into 3 pairs each with sum divisible by 10.

Constraints:

arr.length == n

1 <= n <= 10

5

n

is even.

-10

9

<= arr[i] <= 10

9

1 <= k <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean canArrange(int[] arr, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function canArrange(arr: number[], k: number): boolean {

};
```

C#:

```
public class Solution {
public bool CanArrange(int[] arr, int k) {

}
}
```

C:

```
bool canArrange(int* arr, int arrSize, int k) {

}
```

Go:

```
func canArrange(arr []int, k int) bool {

}
```

Kotlin:

```
class Solution {
fun canArrange(arr: IntArray, k: Int): Boolean {

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool canArrange(List<int> arr, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec can_arrange(Arr :: [integer()], K :: integer()) -> boolean().
can_arrange(Arr, K) ->
.
```

Racket:

```
(define/contract (can-arrange arr k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check If Array Pairs Are Divisible by k
* Difficulty: Medium
* Tags: array, hash
*
* 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 canArrange(vector<int>& arr, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Check If Array Pairs Are Divisible by k
* Difficulty: Medium
* Tags: array, hash
*
* 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 canArrange(int[] arr, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Check If Array Pairs Are Divisible by k
Difficulty: Medium
Tags: array, hash

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 canArrange(self, arr: List[int], k: int) -> bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check If Array Pairs Are Divisible by k
* Difficulty: Medium
* Tags: array, hash
*
* 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[]} arr
* @param {number} k
* @return {boolean}
*/
var canArrange = function(arr, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Check If Array Pairs Are Divisible by k
* Difficulty: Medium
* Tags: array, hash
*
* 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 canArrange(arr: number[], k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Check If Array Pairs Are Divisible by k
* Difficulty: Medium
* Tags: array, hash
*
* 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 CanArrange(int[] arr, int k) {

}
}
```

C Solution:

```
/*
* Problem: Check If Array Pairs Are Divisible by k
* Difficulty: Medium
* Tags: array, hash
*
* 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 canArrange(int* arr, int arrSize, int k) {

}
```

Go Solution:

```
// Problem: Check If Array Pairs Are Divisible by k
// Difficulty: Medium
// Tags: array, hash
//
// 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 canArrange(arr []int, k int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canArrange(arr: IntArray, k: Int): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check If Array Pairs Are Divisible by k
// Difficulty: Medium
// Tags: array, hash
//
// 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 can_arrange(arr: Vec<i32>, k: i32) -> bool {
```

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool canArrange(List<int> arr, int k) {

}
}
```

Scala Solution:

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

}
```

```
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec can_arrange(Arr :: [integer()], K :: integer()) -> boolean().
can_arrange(Arr, K) ->
.
```

Racket Solution:

```
(define/contract (can-arrange arr k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```