Problem 1502: Can Make Arithmetic Progression From Sequence

Problem Information

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

Problem Description

A sequence of numbers is called an

arithmetic progression

if the difference between any two consecutive elements is the same.

Given an array of numbers

arr

, return

true

if the array can be rearranged to form an

arithmetic progression

. Otherwise, return

false

.

Example 1:

Input:

arr = [3,5,1]

Output:

true

Explanation:

We can reorder the elements as [1,3,5] or [5,3,1] with differences 2 and -2 respectively, between each consecutive elements.

Example 2:

Input:

arr = [1,2,4]

Output:

false

Explanation:

There is no way to reorder the elements to obtain an arithmetic progression.

Constraints:

2 <= arr.length <= 1000

-10

6

<= arr[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
bool canMakeArithmeticProgression(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public boolean canMakeArithmeticProgression(int[] arr) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function canMakeArithmeticProgression(arr: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool CanMakeArithmeticProgression(int[] arr) {

}
}
```

C:

```
bool canMakeArithmeticProgression(int* arr, int arrSize) {

}
```

Go:

```
func canMakeArithmeticProgression(arr []int) bool {

}
```

Kotlin:

```
class Solution {
fun canMakeArithmeticProgression(arr: IntArray): Boolean {

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Boolean}
def can_make_arithmetic_progression(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Boolean
*/
function canMakeArithmeticProgression($arr) {

}
}
```

Dart:

```
class Solution {
bool canMakeArithmeticProgression(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def canMakeArithmeticProgression(arr: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_make_arithmetic_progression(arr :: [integer]) :: boolean
def can_make_arithmetic_progression(arr) do

end
end
```

Erlang:

```
-spec can_make_arithmetic_progression(Arr :: [integer()]) -> boolean().
can_make_arithmetic_progression(Arr) ->
.
```

Racket:

```
(define/contract (can-make-arithmetic-progression arr)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Can Make Arithmetic Progression From Sequence
* Difficulty: Easy
* 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 canMakeArithmeticProgression(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Can Make Arithmetic Progression From Sequence
* Difficulty: Easy
* 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 canMakeArithmeticProgression(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Can Make Arithmetic Progression From Sequence
Difficulty: Easy
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 canMakeArithmeticProgression(self, arr: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Can Make Arithmetic Progression From Sequence
* Difficulty: Easy
* 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[]} arr
* @return {boolean}
*/
var canMakeArithmeticProgression = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Can Make Arithmetic Progression From Sequence
* Difficulty: Easy
* 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 canMakeArithmeticProgression(arr: number[]): boolean {
```

```
};
```

C# Solution:

```
/*
* Problem: Can Make Arithmetic Progression From Sequence
* Difficulty: Easy
* 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 CanMakeArithmeticProgression(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Can Make Arithmetic Progression From Sequence
* Difficulty: Easy
* 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 canMakeArithmeticProgression(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Can Make Arithmetic Progression From Sequence
// Difficulty: Easy
```

```
// 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 canMakeArithmeticProgression(arr []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canMakeArithmeticProgression(arr: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Can Make Arithmetic Progression From Sequence
// Difficulty: Easy
// 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_make_arithmetic_progression(arr: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Boolean}
def can_make_arithmetic_progression(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Boolean
*/
function canMakeArithmeticProgression($arr) {

}
}
```

Dart Solution:

```
class Solution {
bool canMakeArithmeticProgression(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def canMakeArithmeticProgression(arr: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_make_arithmetic_progression(arr :: [integer]) :: boolean
def can_make_arithmetic_progression(arr) do
```

```
end
end
```

Erlang Solution:

```
-spec can_make_arithmetic_progression(Arr :: [integer()]) -> boolean().
can_make_arithmetic_progression(Arr) ->
.
```

Racket Solution:

```
(define/contract (can-make-arithmetic-progression arr)
(-> (listof exact-integer?) boolean?)
)
```