Problem 1228: Missing Number In Arithmetic Progression

Problem Information

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

Problem Description

In some array

arr

, the values were in arithmetic progression: the values

arr[i + 1] - arr[i]

are all equal for every

0 <= i < arr.length - 1

.

A value from

arr

was removed that

was not the first or last value in the array

.

Given

arr

, return

the removed value

.

Example 1:

Input:

arr = [5,7,11,13]

Output:

9

Explanation:

The previous array was [5,7,

9

,11,13].

Example 2:

Input:

arr = [15,13,12]

Output:

14

Explanation:

The previous array was [15,

14

,13,12].

Constraints:

3 <= arr.length <= 1000

0 <= arr[i] <= 10

5

The given array is

guaranteed

to be a valid array.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int missingNumber(int[] arr) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function missingNumber(arr: number[]): number {

};
```

C#:

```
public class Solution {
public int MissingNumber(int[] arr) {

}
}
```

C:

```
int missingNumber(int* arr, int arrSize) {

}
```

Go:

```
func missingNumber(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun missingNumber(arr: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[] $arr
* @return Integer
*/
function missingNumber($arr) {

}
}
```

Dart:

```
class Solution {
int missingNumber(List<int> arr) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (missing-number arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Missing Number In Arithmetic Progression
* Difficulty: Easy
* Tags: array, math
*
* 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 missingNumber(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Missing Number In Arithmetic Progression
* Difficulty: Easy
* Tags: array, math
*
* 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 missingNumber(int[] arr) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Missing Number In Arithmetic Progression
Difficulty: Easy
Tags: array, math

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 missingNumber(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Missing Number In Arithmetic Progression
* Difficulty: Easy
* Tags: array, math
*
* 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 {number}
*/
var missingNumber = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Missing Number In Arithmetic Progression
* Difficulty: Easy
* Tags: array, math
*
* 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 missingNumber(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Missing Number In Arithmetic Progression
* Difficulty: Easy
* Tags: array, math
*
* 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 MissingNumber(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Missing Number In Arithmetic Progression
* Difficulty: Easy
* Tags: array, math
*
* 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
*/

int missingNumber(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Missing Number In Arithmetic Progression
// Difficulty: Easy
// Tags: array, math
//
// 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 missingNumber(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun missingNumber(arr: IntArray): Int {

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Missing Number In Arithmetic Progression
// Difficulty: Easy
// Tags: array, math
//
// 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 missing_number(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int missingNumber(List<int> arr) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (missing-number arr)
(-> (listof exact-integer?) exact-integer?)
)
```