

Problem 3131: Find the Integer Added to Array I

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two arrays of equal length,

nums1

and

nums2

Each element in

nums1

has been increased (or decreased in the case of negative) by an integer, represented by the variable

x

As a result,

nums1

becomes

equal

to

nums2

. Two arrays are considered

equal

when they contain the same integers with the same frequencies.

Return the integer

x

.

Example 1:

Input:

nums1 = [2,6,4], nums2 = [9,7,5]

Output:

3

Explanation:

The integer added to each element of

nums1

is 3.

Example 2:

Input:

nums1 = [10], nums2 = [5]

Output:

-5

Explanation:

The integer added to each element of

nums1

is -5.

Example 3:

Input:

nums1 = [1,1,1,1], nums2 = [1,1,1,1]

Output:

0

Explanation:

The integer added to each element of

nums1

is 0.

Constraints:

$1 \leq \text{nums1.length} == \text{nums2.length} \leq 100$

$0 \leq \text{nums1}[i], \text{nums2}[i] \leq 1000$

The test cases are generated in a way that there is an integer

x

such that

nums1

can become equal to

nums2

by adding

x

to each element of

nums1

.

Code Snippets

C++:

```
class Solution {
public:
    int addedInteger(vector<int>& nums1, vector<int>& nums2) {
        }
};
```

Java:

```
class Solution {
    public int addedInteger(int[] nums1, int[] nums2) {
        }
```

```
}
```

Python3:

```
class Solution:  
    def addedInteger(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):  
    def addedInteger(self, nums1, nums2):  
        """  
        :type nums1: List[int]  
        :type nums2: List[int]  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[]} nums1  
 * @param {number[]} nums2  
 * @return {number}  
 */  
var addedInteger = function(nums1, nums2) {  
  
};
```

TypeScript:

```
function addedInteger(nums1: number[], nums2: number[]): number {  
  
};
```

C#:

```
public class Solution {  
    public int AddedInteger(int[] nums1, int[] nums2) {  
  
    }  
}
```

C:

```
int addedInteger(int* nums1, int nums1Size, int* nums2, int nums2Size) {  
  
}
```

Go:

```
func addedInteger(nums1 []int, nums2 []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun addedInteger(nums1: IntArray, nums2: IntArray): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func addedInteger(_ nums1: [Int], _ nums2: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn added_integer(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} nums1  
# @param {Integer[]} nums2  
# @return {Integer}  
def added_integer(nums1, nums2)
```

```
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums1  
     * @param Integer[] $nums2  
     * @return Integer  
     */  
    function addedInteger($nums1, $nums2) {  
  
    }  
}
```

Dart:

```
class Solution {  
int addedInteger(List<int> nums1, List<int> nums2) {  
  
}  
}
```

Scala:

```
object Solution {  
def addedInteger(nums1: Array[Int], nums2: Array[Int]): Int = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec added_integer([integer], [integer]) :: integer  
def added_integer(nums1, nums2) do  
  
end  
end
```

Erlang:

```
-spec added_integer(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().  
added_integer(Nums1, Nums2) ->  
. .
```

Racket:

```
(define/contract (added-integer nums1 nums2)  
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Find the Integer Added to Array I  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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 addedInteger(vector<int>& nums1, vector<int>& nums2) {  
        }  
};
```

Java Solution:

```
/**  
 * Problem: Find the Integer Added to Array I  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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
```

```
*/\n\n\nclass Solution {\n    public int addedInteger(int[] nums1, int[] nums2) {\n\n        }\n    }\n}
```

Python3 Solution:

```
'''\n\nProblem: Find the Integer Added to Array I\n\nDifficulty: Easy\n\nTags: array\n\nApproach: Use two pointers or sliding window technique\nTime Complexity: O(n) or O(n log n)\nSpace Complexity: O(1) to O(n) depending on approach\n'''\n\n\nclass Solution:\n    def addedInteger(self, nums1: List[int], nums2: List[int]) -> int:\n        # TODO: Implement optimized solution\n        pass
```

Python Solution:

```
class Solution(object):\n    def addedInteger(self, nums1, nums2):\n        """\n        :type nums1: List[int]\n        :type nums2: List[int]\n        :rtype: int\n        """
```

JavaScript Solution:

```
/**\n * Problem: Find the Integer Added to Array I\n * Difficulty: Easy\n * Tags: array
```

```

/*
 * 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[]} nums1
 * @param {number[]} nums2
 * @return {number}
 */
var addedInteger = function(nums1, nums2) {

};

```

TypeScript Solution:

```

/**
 * Problem: Find the Integer Added to Array I
 * Difficulty: Easy
 * Tags: array
 *
 * 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 addedInteger(nums1: number[], nums2: number[]): number {

};

```

C# Solution:

```

/*
 * Problem: Find the Integer Added to Array I
 * Difficulty: Easy
 * Tags: array
 *
 * 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

```

```
*/\n\npublic class Solution {\n    public int AddedInteger(int[] nums1, int[] nums2) {\n\n        }\n    }\n}
```

C Solution:

```
/*\n * Problem: Find the Integer Added to Array I\n * Difficulty: Easy\n * Tags: array\n *\n * Approach: Use two pointers or sliding window technique\n * Time Complexity: O(n) or O(n log n)\n * Space Complexity: O(1) to O(n) depending on approach\n */\n\nint addedInteger(int* nums1, int numslSize, int* nums2, int nums2Size) {\n\n}
```

Go Solution:

```
// Problem: Find the Integer Added to Array I\n// Difficulty: Easy\n// Tags: array\n//\n// Approach: Use two pointers or sliding window technique\n// Time Complexity: O(n) or O(n log n)\n// Space Complexity: O(1) to O(n) depending on approach\n\nfunc addedInteger(nums1 []int, nums2 []int) int {\n\n}
```

Kotlin Solution:

```
class Solution {  
    fun addedInteger(nums1: IntArray, nums2: IntArray): Int {  
        }  
        }  
}
```

Swift Solution:

```
class Solution {  
    func addedInteger(_ nums1: [Int], _ nums2: [Int]) -> Int {  
        }  
        }  
}
```

Rust Solution:

```
// Problem: Find the Integer Added to Array I  
// Difficulty: Easy  
// Tags: array  
//  
// 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 added_integer(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {  
        }  
        }  
}
```

Ruby Solution:

```
# @param {Integer[]} nums1  
# @param {Integer[]} nums2  
# @return {Integer}  
def added_integer(nums1, nums2)  
  
end
```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $nums1
     * @param Integer[] $nums2
     * @return Integer
     */
    function addedInteger($nums1, $nums2) {

    }
}

```

Dart Solution:

```

class Solution {
    int addedInteger(List<int> nums1, List<int> nums2) {
        }
}

```

Scala Solution:

```

object Solution {
    def addedInteger(nums1: Array[Int], nums2: Array[Int]): Int = {
        }
}

```

Elixir Solution:

```

defmodule Solution do
    @spec added_integer(list(integer()), list(integer())) :: integer()
    def added_integer(nums1, nums2) do
        end
    end

```

Erlang Solution:

```

-spec added_integer(list(integer()), list(integer())) -> integer().
added_integer(Nums1, Nums2) ->
    .

```

Racket Solution:

```
(define/contract (added-integer nums1 nums2)
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?))
```