Problem 2333: Minimum Sum of Squared Difference

Problem Information

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

Problem Description

You are given two positive

0-indexed

integer arrays

nums1

and

nums2

, both of length

n

.

The

sum of squared difference

of arrays

nums1

and

nums2

is defined as the

sum

of

(nums1[i] - nums2[i])

2

for each

0 <= i < n

.

You are also given two positive integers

k1

and

k2

. You can modify any of the elements of

nums1

by

+1

or

-1

at most

k1

times. Similarly, you can modify any of the elements of

nums2

by

+1

or

-1

at most

k2

times.

Return

the minimum

sum of squared difference

after modifying array

nums1

at most

k1

times and modifying array

nums2

at most

k2

times

.

Note

: You are allowed to modify the array elements to become

negative

integers.

Example 1:

Input:

nums1 = [1,2,3,4], nums2 = [2,10,20,19], k1 = 0, k2 = 0

Output:

579

Explanation:

The elements in nums1 and nums2 cannot be modified because k1 = 0 and k2 = 0. The sum of square difference will be: (1 - 2)

2

+ (2 - 10)

2

+ (3 - 20)

2

+ (4 - 19)

2

= 579.

Example 2:

Input:

nums1 = [1,4,10,12], nums2 = [5,8,6,9], k1 = 1, k2 = 1

Output:

43

Explanation:

One way to obtain the minimum sum of square difference is: - Increase nums1[0] once. - Increase nums2[2] once. The minimum of the sum of square difference will be: (2 - 5)

2

+ (4 - 8)

2

+ (10 - 7)

2

+ (12 - 9)

2

= 43. Note that, there are other ways to obtain the minimum of the sum of square difference, but there is no way to obtain a sum smaller than 43.

Constraints:

n == nums1.length == nums2.length

1 <= n <= 10

5

0 <= nums1[i], nums2[i] <= 10

5

0 <= k1, k2 <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long minSumSquareDiff(vector<int>& nums1, vector<int>& nums2, int k1,
int k2) {

}
};
```

Java:

```
class Solution {
public long minSumSquareDiff(int[] nums1, int[] nums2, int k1, int k2) {

}
}
```

Python3:

```
class Solution:
def minSumSquareDiff(self, nums1: List[int], nums2: List[int], k1: int, k2:
int) -> int:
```

Python:

```
class Solution(object):
def minSumSquareDiff(self, nums1, nums2, k1, k2):
"""
:type nums1: List[int]
:type nums2: List[int]
:type k1: int
:type k2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @param {number} k1
* @param {number} k2
* @return {number}
*/
var minSumSquareDiff = function(nums1, nums2, k1, k2) {

};
```

TypeScript:

```
function minSumSquareDiff(nums1: number[], nums2: number[], k1: number, k2:
number): number {

};
```

C#:

```
public class Solution {
public long MinSumSquareDiff(int[] nums1, int[] nums2, int k1, int k2) {
}
}
```

C:

```
long long minSumSquareDiff(int* nums1, int nums1Size, int* nums2, int
nums2Size, int k1, int k2) {

}
```

Go:

```
func minSumSquareDiff(nums1 []int, nums2 []int, k1 int, k2 int) int64 {

}
```

Kotlin:

```
class Solution {
fun minSumSquareDiff(nums1: IntArray, nums2: IntArray, k1: Int, k2: Int):
Long {

}
}
```

Swift:

```
class Solution {
func minSumSquareDiff(_ nums1: [Int], _ nums2: [Int], _ k1: Int, _ k2: Int)
-> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_sum_square_diff(nums1: Vec<i32>, nums2: Vec<i32>, k1: i32, k2:
i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer} k1
# @param {Integer} k2
# @return {Integer}
def min_sum_square_diff(nums1, nums2, k1, k2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @param Integer $k1
* @param Integer $k2
* @return Integer
*/
function minSumSquareDiff($nums1, $nums2, $k1, $k2) {

}
}
```

Dart:

```
class Solution {
int minSumSquareDiff(List<int> nums1, List<int> nums2, int k1, int k2) {

}
}
```

Scala:

```
object Solution {
def minSumSquareDiff(nums1: Array[Int], nums2: Array[Int], k1: Int, k2: Int):
Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_sum_square_diff(nums1 :: [integer], nums2 :: [integer], k1 ::
integer, k2 :: integer) :: integer
def min_sum_square_diff(nums1, nums2, k1, k2) do

end
end
```

Erlang:

```
-spec min_sum_square_diff(Nums1 :: [integer()], Nums2 :: [integer()], K1 ::
integer(), K2 :: integer()) -> integer().
min_sum_square_diff(Nums1, Nums2, K1, K2) ->
.
```

Racket:

```
(define/contract (min-sum-square-diff nums1 nums2 k1 k2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Sum of Squared Difference
* Difficulty: Medium
* Tags: array, greedy, sort, search, queue, heap
*
* 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:
long long minSumSquareDiff(vector<int>& nums1, vector<int>& nums2, int k1,
int k2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Sum of Squared Difference
* Difficulty: Medium
* Tags: array, greedy, sort, search, queue, heap
*
* 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 long minSumSquareDiff(int[] nums1, int[] nums2, int k1, int k2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Sum of Squared Difference
Difficulty: Medium
Tags: array, greedy, sort, search, queue, heap

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 minSumSquareDiff(self, nums1: List[int], nums2: List[int], k1: int, k2:
int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minSumSquareDiff(self, nums1, nums2, k1, k2):
"""
:type nums1: List[int]
:type nums2: List[int]
:type k1: int
:type k2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Sum of Squared Difference
* Difficulty: Medium
* Tags: array, greedy, sort, search, queue, heap
*
* 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
* @param {number} k1
* @param {number} k2
* @return {number}
*/
var minSumSquareDiff = function(nums1, nums2, k1, k2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Sum of Squared Difference
* Difficulty: Medium
* Tags: array, greedy, sort, search, queue, heap
*
* 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 minSumSquareDiff(nums1: number[], nums2: number[], k1: number, k2:
number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Sum of Squared Difference
* Difficulty: Medium
* Tags: array, greedy, sort, search, queue, heap
*
* 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 long MinSumSquareDiff(int[] nums1, int[] nums2, int k1, int k2) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Sum of Squared Difference
* Difficulty: Medium
* Tags: array, greedy, sort, search, queue, heap
*
* 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
*/

long long minSumSquareDiff(int* nums1, int nums1Size, int* nums2, int
nums2Size, int k1, int k2) {

}
```

Go Solution:

```
// Problem: Minimum Sum of Squared Difference
// Difficulty: Medium
// Tags: array, greedy, sort, search, queue, heap
//
// 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 minSumSquareDiff(nums1 []int, nums2 []int, k1 int, k2 int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minSumSquareDiff(nums1: IntArray, nums2: IntArray, k1: Int, k2: Int):
Long {

}
}
```

Swift Solution:

```
class Solution {
func minSumSquareDiff(_ nums1: [Int], _ nums2: [Int], _ k1: Int, _ k2: Int)
-> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Sum of Squared Difference
// Difficulty: Medium
// Tags: array, greedy, sort, search, queue, heap
//
// 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 min_sum_square_diff(nums1: Vec<i32>, nums2: Vec<i32>, k1: i32, k2:
i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer} k1
# @param {Integer} k2
# @return {Integer}
def min_sum_square_diff(nums1, nums2, k1, k2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @param Integer $k1
* @param Integer $k2
* @return Integer
*/
function minSumSquareDiff($nums1, $nums2, $k1, $k2) {

}
}
```

Dart Solution:

```
class Solution {
int minSumSquareDiff(List<int> nums1, List<int> nums2, int k1, int k2) {
}
}
```

Scala Solution:

```
object Solution {
def minSumSquareDiff(nums1: Array[Int], nums2: Array[Int], k1: Int, k2: Int):
Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_sum_square_diff(nums1 :: [integer], nums2 :: [integer], k1 ::
integer, k2 :: integer) :: integer
def min_sum_square_diff(nums1, nums2, k1, k2) do

end
end
```

Erlang Solution:

```
-spec min_sum_square_diff(Nums1 :: [integer()], Nums2 :: [integer()], K1 ::
integer(), K2 :: integer()) -> integer().
min_sum_square_diff(Nums1, Nums2, K1, K2) ->
.
```

Racket Solution:

```
(define/contract (min-sum-square-diff nums1 nums2 k1 k2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer? exact-integer?)
)
```