Problem 1874: Minimize Product Sum of Two Arrays

Problem Information

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

Problem Description

The

product sum

of two equal-length arrays

a

and

b

is equal to the sum of

a[i] * b[i]

for all

0 <= i < a.length

(

0-indexed

).

For example, if

a = [1,2,3,4]

and

b = [5,2,3,1]

, the

product sum

would be

1*5 + 2*2 + 3*3 + 4*1 = 22

.

Given two arrays

nums1

and

nums2

of length

n

, return

the

minimum product sum

if you are allowed to

rearrange

the

order

of the elements in

nums1

.

Example 1:

Input:

nums1 = [5,3,4,2], nums2 = [4,2,2,5]

Output:

40

Explanation:

We can rearrange nums1 to become [3,5,4,2]. The product sum of [3,5,4,2] and [4,2,2,5] is 3*4 + 5*2 + 4*2 + 2*5 = 40.

Example 2:

Input:

nums1 = [2,1,4,5,7], nums2 = [3,2,4,8,6]

Output:

65

Explanation:

We can rearrange nums1 to become [5,7,4,1,2]. The product sum of [5,7,4,1,2] and [3,2,4,8,6] is 5*3 + 7*2 + 4*4 + 1*8 + 2*6 = 65.

Constraints:

n == nums1.length == nums2.length

1 <= n <= 10

5

1 <= nums1[i], nums2[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minProductSum(nums1: number[], nums2: number[]): number {

};
```

C#:

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

}
}
```

C:

```


int minProductSum(int* nums1, int nums1Size, int* nums2, int nums2Size){

}
```

Go:

```
func minProductSum(nums1 []int, nums2 []int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
```

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

}
}
```

Scala:

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

}
}
```

Racket:

```
(define/contract (min-product-sum nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize Product Sum of Two Arrays
* Difficulty: Medium
* Tags: array, greedy, 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:
int minProductSum(vector<int>& nums1, vector<int>& nums2) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Minimize Product Sum of Two Arrays
* Difficulty: Medium
* Tags: array, greedy, 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 int minProductSum(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize Product Sum of Two Arrays
Difficulty: Medium
Tags: array, greedy, 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 minProductSum(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize Product Sum of Two Arrays
* Difficulty: Medium
* Tags: array, greedy, 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[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minProductSum = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Product Sum of Two Arrays
* Difficulty: Medium
* Tags: array, greedy, 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 minProductSum(nums1: number[], nums2: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimize Product Sum of Two Arrays
* Difficulty: Medium
* Tags: array, greedy, 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 int MinProductSum(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Minimize Product Sum of Two Arrays
* Difficulty: Medium
* Tags: array, greedy, 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
*/



int minProductSum(int* nums1, int nums1Size, int* nums2, int nums2Size){

}
```

Go Solution:

```
// Problem: Minimize Product Sum of Two Arrays
// Difficulty: Medium
// Tags: array, greedy, 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 minProductSum(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimize Product Sum of Two Arrays
// Difficulty: Medium
// Tags: array, greedy, 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 min_product_sum(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```

Racket Solution:

```
(define/contract (min-product-sum nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)

)
```