Problem 3290: Maximum Multiplication Score

Problem Information

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

Problem Description

You are given an integer array

a

of size 4 and another integer array

b

of size

at least

4.

You need to choose 4 indices

i

0

,

i

1

,

i

2

, and

i

3

from the array

b

such that

i

0

< i

1

< i

2

< i

3

. Your score will be equal to the value

a[0] * b[i

0

] + a[1] * b[i

1

] + a[2] * b[i

2

] + a[3] * b[i

3

]

.

Return the

maximum

score you can achieve.

Example 1:

Input:

a = [3,2,5,6], b = [2,-6,4,-5,-3,2,-7]

Output:

26

Explanation:

We can choose the indices 0, 1, 2, and 5. The score will be

3 * 2 + 2 * (-6) + 5 * 4 + 6 * 2 = 26

.

Example 2:

Input:

a = [-1,4,5,-2], b = [-5,-1,-3,-2,-4]

Output:

-1

Explanation:

We can choose the indices 0, 1, 3, and 4. The score will be

(-1) * (-5) + 4 * (-1) + 5 * (-2) + (-2) * (-4) = -1

.

Constraints:

a.length == 4

4 <= b.length <= 10

5

-10

5

<= a[i], b[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long maxScore(vector<int>& a, vector<int>& b) {

}
};
```

Java:

```
class Solution {
public long maxScore(int[] a, int[] b) {

}
}
```

Python3:

```
class Solution:
def maxScore(self, a: List[int], b: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxScore(self, a, b):
"""
:type a: List[int]
:type b: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} a
* @param {number[]} b
* @return {number}
*/
var maxScore = function(a, b) {

};
```

TypeScript:

```
function maxScore(a: number[], b: number[]): number {

};
```

C#:

```
public class Solution {
public long MaxScore(int[] a, int[] b) {

}
}
```

C:

```
long long maxScore(int* a, int aSize, int* b, int bSize) {

}
```

Go:

```
func maxScore(a []int, b []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxScore(a: IntArray, b: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func maxScore(_ a: [Int], _ b: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_score(a: Vec<i32>, b: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} a
# @param {Integer[]} b
# @return {Integer}
def max_score(a, b)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $a
* @param Integer[] $b
* @return Integer
*/
function maxScore($a, $b) {

}
}
```

Dart:

```
class Solution {
int maxScore(List<int> a, List<int> b) {

}
}
```

Scala:

```
object Solution {
def maxScore(a: Array[Int], b: Array[Int]): Long = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec max_score(a :: [integer], b :: [integer]) :: integer
def max_score(a, b) do

end
end
```

Erlang:

```
-spec max_score(A :: [integer()], B :: [integer()]) -> integer().
max_score(A, B) ->
.
```

Racket:

```
(define/contract (max-score a b)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Multiplication Score
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
long long maxScore(vector<int>& a, vector<int>& b) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Multiplication Score
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long maxScore(int[] a, int[] b) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Multiplication Score
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxScore(self, a: List[int], b: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxScore(self, a, b):
"""
:type a: List[int]
:type b: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Multiplication Score
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} a
* @param {number[]} b
* @return {number}
*/
var maxScore = function(a, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Multiplication Score
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxScore(a: number[], b: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Maximum Multiplication Score
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long MaxScore(int[] a, int[] b) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Multiplication Score
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long maxScore(int* a, int aSize, int* b, int bSize) {

}
```

Go Solution:

```
// Problem: Maximum Multiplication Score
// Difficulty: Medium
```

```
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxScore(a []int, b []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxScore(a: IntArray, b: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxScore(_ a: [Int], _ b: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Multiplication Score
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_score(a: Vec<i32>, b: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} a
# @param {Integer[]} b
# @return {Integer}
def max_score(a, b)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $a
* @param Integer[] $b
* @return Integer
*/
function maxScore($a, $b) {

}
}
```

Dart Solution:

```
class Solution {
int maxScore(List<int> a, List<int> b) {

}
}
```

Scala Solution:

```
object Solution {
def maxScore(a: Array[Int], b: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_score(a :: [integer], b :: [integer]) :: integer
def max_score(a, b) do

end
end
```

Erlang Solution:

```
-spec max_score(A :: [integer()], B :: [integer()]) -> integer().
max_score(A, B) ->
.
```

Racket Solution:

```
(define/contract (max-score a b)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```