

Problem 2040: Kth Smallest Product of Two Sorted Arrays

Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given two

sorted 0-indexed

integer arrays

nums1

and

nums2

as well as an integer

k

, return

the

k

th

(

1-based

) smallest product of

$\text{nums1}[i] * \text{nums2}[j]$

where

$0 \leq i < \text{nums1.length}$

and

$0 \leq j < \text{nums2.length}$

.

Example 1:

Input:

$\text{nums1} = [2,5], \text{nums2} = [3,4], k = 2$

Output:

8

Explanation:

The 2 smallest products are: - $\text{nums1}[0] * \text{nums2}[0] = 2 * 3 = 6$ - $\text{nums1}[0] * \text{nums2}[1] = 2 * 4 = 8$ The 2

nd

smallest product is 8.

Example 2:

Input:

nums1 = [-4,-2,0,3], nums2 = [2,4], k = 6

Output:

0

Explanation:

The 6 smallest products are: - $\text{nums1}[0] * \text{nums2}[1] = (-4) * 4 = -16$ - $\text{nums1}[0] * \text{nums2}[0] = (-4) * 2 = -8$ - $\text{nums1}[1] * \text{nums2}[1] = (-2) * 4 = -8$ - $\text{nums1}[1] * \text{nums2}[0] = (-2) * 2 = -4$ - $\text{nums1}[2] * \text{nums2}[0] = 0 * 2 = 0$ - $\text{nums1}[2] * \text{nums2}[1] = 0 * 4 = 0$ The 6

th

smallest product is 0.

Example 3:

Input:

nums1 = [-2,-1,0,1,2], nums2 = [-3,-1,2,4,5], k = 3

Output:

-6

Explanation:

The 3 smallest products are: - $\text{nums1}[0] * \text{nums2}[4] = (-2) * 5 = -10$ - $\text{nums1}[0] * \text{nums2}[3] = (-2) * 4 = -8$ - $\text{nums1}[4] * \text{nums2}[0] = 2 * (-3) = -6$ The 3

rd

smallest product is -6.

Constraints:

$1 \leq \text{nums1.length}, \text{nums2.length} \leq 5 * 10$

4

-10

5

$\leq \text{nums1}[i], \text{nums2}[j] \leq 10$

5

$1 \leq k \leq \text{nums1.length} * \text{nums2.length}$

nums1

and

nums2

are sorted.

Code Snippets

C++:

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

    }
};
```

Java:

```
class Solution {
    public long kthSmallestProduct(int[] nums1, int[] nums2, long k) {

    }
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
    public long KthSmallestProduct(int[] nums1, int[] nums2, long k) {

    }
}
```

C:

```
long long kthSmallestProduct(int* nums1, int nums1Size, int* nums2, int
nums2Size, long long k) {

}
```

Go:

```
func kthSmallestProduct(nums1 []int, nums2 []int, k int64) int64 {

}
```

Kotlin:

```
class Solution {
    fun kthSmallestProduct(nums1: IntArray, nums2: IntArray, k: Long): Long {

    }
}
```

Swift:

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

    }
}
```

Rust:

```
impl Solution {
    pub fn kth_smallest_product(nums1: Vec<i32>, nums2: Vec<i32>, k: i64) -> i64
    {

    }
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer} k
```

```
# @return {Integer}
def kth_smallest_product(nums1, nums2, k)

end
```

PHP:

```
class Solution {

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

    }

}
```

Dart:

```
class Solution {
  int kthSmallestProduct(List<int> nums1, List<int> nums2, int k) {

  }

}
```

Scala:

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

  }

}
```

Elixir:

```
defmodule Solution do
  @spec kth_smallest_product(nums1 :: [integer], nums2 :: [integer], k ::
integer) :: integer
```

```
def kth_smallest_product(nums1, nums2, k) do

end

end
```

Erlang:

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

Racket:

```
(define/contract (kth-smallest-product nums1 nums2 k)
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?
    exact-integer?)
  )
```

Solutions

C++ Solution:

```
/*
 * Problem: Kth Smallest Product of Two Sorted Arrays
 * Difficulty: Hard
 * Tags: array, sort, search
 *
 * 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 kthSmallestProduct(vector<int>& nums1, vector<int>& nums2, long
    long k) {

    }

};
```

Java Solution:

```
/**
 * Problem: Kth Smallest Product of Two Sorted Arrays
 * Difficulty: Hard
 * Tags: array, sort, search
 *
 * 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 kthSmallestProduct(int[] nums1, int[] nums2, long k) {

    }
}
```

Python3 Solution:

```
"""
Problem: Kth Smallest Product of Two Sorted Arrays
Difficulty: Hard
Tags: array, sort, search

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

Python Solution:

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

```

:type nums2: List[int]
:type k: int
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Kth Smallest Product of Two Sorted Arrays
 * Difficulty: Hard
 * Tags: array, sort, search
 *
 * 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} k
 * @return {number}
 */
var kthSmallestProduct = function(nums1, nums2, k) {

};

```

TypeScript Solution:

```

/**
 * Problem: Kth Smallest Product of Two Sorted Arrays
 * Difficulty: Hard
 * Tags: array, sort, search
 *
 * 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 kthSmallestProduct(nums1: number[], nums2: number[], k: number):
number {

```

```
};
```

C# Solution:

```
/*
 * Problem: Kth Smallest Product of Two Sorted Arrays
 * Difficulty: Hard
 * Tags: array, sort, search
 *
 * 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 KthSmallestProduct(int[] nums1, int[] nums2, long k) {

    }
}
```

C Solution:

```
/*
 * Problem: Kth Smallest Product of Two Sorted Arrays
 * Difficulty: Hard
 * Tags: array, sort, search
 *
 * 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 kthSmallestProduct(int* nums1, int nums1Size, int* nums2, int
nums2Size, long long k) {

}
```

Go Solution:

```
// Problem: Kth Smallest Product of Two Sorted Arrays
// Difficulty: Hard
// Tags: array, sort, search
//
// 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 kthSmallestProduct(nums1 []int, nums2 []int, k int64) int64 {

}
```

Kotlin Solution:

```
class Solution {
    fun kthSmallestProduct(nums1: IntArray, nums2: IntArray, k: Long): Long {

    }
}
```

Swift Solution:

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

    }
}
```

Rust Solution:

```
// Problem: Kth Smallest Product of Two Sorted Arrays
// Difficulty: Hard
// Tags: array, sort, search
//
// 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 kth_smallest_product(nums1: Vec<i32>, nums2: Vec<i32>, k: i64) -> i64
    {
```

```
}  
}
```

Ruby Solution:

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

PHP Solution:

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

Dart Solution:

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

Scala Solution:

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

```
}  
}
```

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (kth-smallest-product nums1 nums2 k)  
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?  
    exact-integer?)  
  )
```