

Problem 2542: Maximum Subsequence Score

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two

0-indexed

integer arrays

nums1

and

nums2

of equal length

n

and a positive integer

k

. You must choose a

subsequence

of indices from

nums1

of length

k

.

For chosen indices

i

0

,

i

1

, ...,

i

k - 1

, your

score

is defined as:

The sum of the selected elements from

nums1

multiplied with the

minimum

of the selected elements from

nums2

.

It can defined simply as:

(nums1[i

0

] + nums1[i

1

] +...+ nums1[i

k - 1

]) * min(nums2[i

0

] , nums2[i

1

], ... ,nums2[i

k - 1

])

.

Return

the
maximum
possible score.

A
subsequence

of indices of an array is a set that can be derived from the set
 $\{0, 1, \dots, n-1\}$

by deleting some or no elements.

Example 1:

Input:

nums1 = [1,3,3,2], nums2 = [2,1,3,4], k = 3

Output:

12

Explanation:

The four possible subsequence scores are: - We choose the indices 0, 1, and 2 with score = $(1+3+3) * \min(2,1,3) = 7$. - We choose the indices 0, 1, and 3 with score = $(1+3+2) * \min(2,1,4) = 6$. - We choose the indices 0, 2, and 3 with score = $(1+3+2) * \min(2,3,4) = 12$. - We choose the indices 1, 2, and 3 with score = $(3+3+2) * \min(1,3,4) = 8$. Therefore, we return the max score, which is 12.

Example 2:

Input:

nums1 = [4,2,3,1,1], nums2 = [7,5,10,9,6], k = 1

Output:

30

Explanation:

Choosing index 2 is optimal: $\text{nums1}[2] * \text{nums2}[2] = 3 * 10 = 30$ is the maximum possible score.

Constraints:

$n == \text{nums1.length} == \text{nums2.length}$

$1 \leq n \leq 10$

5

$0 \leq \text{nums1}[i], \text{nums2}[j] \leq 10$

5

$1 \leq k \leq n$

Code Snippets

C++:

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

Java:

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

```
}
```

```
}
```

Python3:

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

Python:

```
class Solution(object):  
    def maxScore(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 maxScore = function(nums1, nums2, k) {  
  
};
```

TypeScript:

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

C#:

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

```
}
```

```
}
```

C:

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

Go:

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

Kotlin:

```
class Solution {  
    fun maxScore(nums1: IntArray, nums2: IntArray, k: Int): Long {  
  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

    }
}
```

Dart:

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

Scala:

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

Elixir:

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

```

integer
def max_score(nums1, nums2, k) do
    end
end

```

Erlang:

```

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

```

Racket:

```

(define/contract (max-score nums1 nums2 k)
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?
    exact-integer?))

```

Solutions

C++ Solution:

```

/*
 * Problem: Maximum Subsequence Score
 * Difficulty: Medium
 * Tags: array, greedy, sort, 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 maxScore(vector<int>& nums1, vector<int>& nums2, int k) {
        }
    };

```

Java Solution:

```
/**  
 * Problem: Maximum Subsequence Score  
 * Difficulty: Medium  
 * Tags: array, greedy, sort, 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 maxScore(int[] nums1, int[] nums2, int k) {  
        // Implementation  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Maximum Subsequence Score  
Difficulty: Medium  
Tags: array, greedy, sort, 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 maxScore(self, nums1: List[int], nums2: List[int], k: int) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

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

```
:rtype: int
"""

```

JavaScript Solution:

```
/**
 * Problem: Maximum Subsequence Score
 * Difficulty: Medium
 * Tags: array, greedy, sort, 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} k
 * @return {number}
 */
var maxScore = function(nums1, nums2, k) {

};


```

TypeScript Solution:

```
/**
 * Problem: Maximum Subsequence Score
 * Difficulty: Medium
 * Tags: array, greedy, sort, 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 maxScore(nums1: number[], nums2: number[], k: number): number {

};
```

C# Solution:

```
/*
 * Problem: Maximum Subsequence Score
 * Difficulty: Medium
 * Tags: array, greedy, sort, 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 MaxScore(int[] nums1, int[] nums2, int k) {
        return 0;
    }
}
```

C Solution:

```
/*
 * Problem: Maximum Subsequence Score
 * Difficulty: Medium
 * Tags: array, greedy, sort, 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 maxScore(int* nums1, int nums1Size, int* nums2, int nums2Size, int k) {
    return 0;
}
```

Go Solution:

```
// Problem: Maximum Subsequence Score
// Difficulty: Medium
// Tags: array, greedy, sort, 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 maxScore(nums1 []int, nums2 []int, k int) int64 {
}
```

Kotlin Solution:

```
class Solution {
    fun maxScore(nums1: IntArray, nums2: IntArray, k: Int): Long {
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Maximum Subsequence Score
// Difficulty: Medium
// Tags: array, greedy, sort, 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 max_score(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> i64 {
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }
}
```

Dart Solution:

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

Scala Solution:

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

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-score nums1 nums2 k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?))
```