Problem 2818: Apply Operations to Maximize Score

Problem Information

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

Problem Description

You are given an array

nums

of

n

positive integers and an integer

k

.

Initially, you start with a score of

1

. You have to maximize your score by applying the following operation at most

k

times:

Choose any

non-empty

subarray

nums[l, ..., r]

that you haven't chosen previously.

Choose an element

x

of

nums[l, ..., r]

with the highest

prime score

. If multiple such elements exist, choose the one with the smallest index.

Multiply your score by

x

.

Here,

nums[l, ..., r]

denotes the subarray of

nums

starting at index

l

and ending at the index

r

, both ends being inclusive.

The

prime score

of an integer

x

is equal to the number of distinct prime factors of

x

. For example, the prime score of

300

is

3

since

300 = 2 * 2 * 3 * 5 * 5

.

Return

the

maximum possible score

after applying at most

k

operations

.

Since the answer may be large, return it modulo

10

9

+ 7

.

Example 1:

Input:

nums = [8,3,9,3,8], k = 2

Output:

81

Explanation:

To get a score of 81, we can apply the following operations: - Choose subarray nums[2, ..., 2]. nums[2] is the only element in this subarray. Hence, we multiply the score by nums[2]. The score becomes 1 * 9 = 9. - Choose subarray nums[2, ..., 3]. Both nums[2] and nums[3] have a prime score of 1, but nums[2] has the smaller index. Hence, we multiply the score by nums[2]. The score becomes 9 * 9 = 81. It can be proven that 81 is the highest score one can obtain.

Example 2:

Input:

nums = [19,12,14,6,10,18], k = 3

Output:

4788

Explanation:

To get a score of 4788, we can apply the following operations: - Choose subarray nums[0, ..., 0]. nums[0] is the only element in this subarray. Hence, we multiply the score by nums[0]. The score becomes 1 * 19 = 19. - Choose subarray nums[5, ..., 5]. nums[5] is the only element in this subarray. Hence, we multiply the score by nums[5]. The score becomes 19 * 18 = 342. - Choose subarray nums[2, ..., 3]. Both nums[2] and nums[3] have a prime score of 2, but nums[2] has the smaller index. Hence, we multipy the score by nums[2]. The score becomes 342 * 14 = 4788. It can be proven that 4788 is the highest score one can obtain.

Constraints:

1 <= nums.length == n <= 10

5

1 <= nums[i] <= 10

5

1 <= k <= min(n * (n + 1) / 2, 10

9

)

Code Snippets

C++:

```
class Solution {
public:
```

```
int maximumScore(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int maximumScore(List<Integer> nums, int k) {

}
}
```

Python3:

```
class Solution:
def maximumScore(self, nums: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def maximumScore(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var maximumScore = function(nums, k) {

};
```

TypeScript:

```
function maximumScore(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaximumScore(IList<int> nums, int k) {

}
}
```

C:

```
int maximumScore(int* nums, int numsSize, int k) {

}
```

Go:

```
func maximumScore(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maximumScore(nums: List<Int>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumScore(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_score(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def maximum_score(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maximumScore($nums, $k) {

}
}
```

Dart:

```
class Solution {
int maximumScore(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def maximumScore(nums: List[Int], k: Int): Int = {

}

}
```

Elixir:

```
defmodule Solution do
@spec maximum_score(nums :: [integer], k :: integer) :: integer
def maximum_score(nums, k) do

end
end
```

Erlang:

```
-spec maximum_score(Nums :: [integer()], K :: integer()) -> integer().
maximum_score(Nums, K) ->
.
```

Racket:

```
(define/contract (maximum-score nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Apply Operations to Maximize Score
* Difficulty: Hard
* Tags: array, greedy, math, sort, stack
*
* 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 maximumScore(vector<int>& nums, int k) {
}
};
```

Java Solution:

```
/**
* Problem: Apply Operations to Maximize Score
* Difficulty: Hard
* Tags: array, greedy, math, sort, stack
*
* 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 maximumScore(List<Integer> nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Apply Operations to Maximize Score
Difficulty: Hard
Tags: array, greedy, math, sort, stack

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 maximumScore(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumScore(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Apply Operations to Maximize Score
* Difficulty: Hard
* Tags: array, greedy, math, sort, stack
*
* 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[]} nums
* @param {number} k
* @return {number}
*/
var maximumScore = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Apply Operations to Maximize Score
* Difficulty: Hard
* Tags: array, greedy, math, sort, stack
*
* 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 maximumScore(nums: number[], k: number): number {


};
```

C# Solution:

```
/*
* Problem: Apply Operations to Maximize Score
* Difficulty: Hard
* Tags: array, greedy, math, sort, stack
*
* 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 MaximumScore(IList<int> nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Apply Operations to Maximize Score
* Difficulty: Hard
* Tags: array, greedy, math, sort, stack
*
* 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 maximumScore(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Apply Operations to Maximize Score
// Difficulty: Hard
// Tags: array, greedy, math, sort, stack
//
// 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 maximumScore(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumScore(nums: List<Int>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumScore(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Apply Operations to Maximize Score
// Difficulty: Hard
// Tags: array, greedy, math, sort, stack
//
// 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 maximum_score(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def maximum_score(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maximumScore($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximumScore(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maximumScore(nums: List[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_score(nums :: [integer], k :: integer) :: integer
def maximum_score(nums, k) do

end
end
```

Erlang Solution:

```
-spec maximum_score(Nums :: [integer()], K :: integer()) -> integer().
maximum_score(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (maximum-score nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```