Problem 3473: Sum of K Subarrays With Length at Least M

Problem Information

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

Problem Description

You are given an integer array

nums

and two integers,

k

and

m

.

Return the

maximum

sum of

k

non-overlapping

subarrays

of

nums

, where each subarray has a length of

at least

m

.

Example 1:

Input:

nums = [1,2,-1,3,3,4], k = 2, m = 2

Output:

13

Explanation:

The optimal choice is:

Subarray

nums[3..5]

with sum

3 + 3 + 4 = 10

(length is

3 >= m

).

Subarray

nums[0..1]

with sum

1 + 2 = 3

(length is

2 >= m

).

The total sum is

10 + 3 = 13

.

Example 2:

Input:

nums = [-10,3,-1,-2], k = 4, m = 1

Output:

-10

Explanation:

The optimal choice is choosing each element as a subarray. The output is

(-10) + 3 + (-1) + (-2) = -10

.

Constraints:

1 <= nums.length <= 2000

-10

4

<= nums[i] <= 10

4

1 <= k <= floor(nums.length / m)

1 <= m <= 3

Code Snippets

C++:

```
class Solution {
public:
int maxSum(vector<int>& nums, int k, int m) {

}
};
```

Java:

```
class Solution {
public int maxSum(int[] nums, int k, int m) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MaxSum(int[] nums, int k, int m) {

}
}
```

C:

```
int maxSum(int* nums, int numsSize, int k, int m) {

}
```

Go:

```
func maxSum(nums []int, k int, m int) int {

}
```

Kotlin:

```
class Solution {
fun maxSum(nums: IntArray, k: Int, m: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def maxSum(nums: Array[Int], k: Int, m: Int): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
.
```

Racket:

```
(define/contract (max-sum nums k m)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of K Subarrays With Length at Least M
* 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:
int maxSum(vector<int>& nums, int k, int m) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of K Subarrays With Length at Least M
* 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 int maxSum(int[] nums, int k, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of K Subarrays With Length at Least M
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 maxSum(self, nums: List[int], k: int, m: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of K Subarrays With Length at Least M
* 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[]} nums
* @param {number} k
* @param {number} m
* @return {number}
*/
var maxSum = function(nums, k, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of K Subarrays With Length at Least M
* 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 maxSum(nums: number[], k: number, m: number): number {

};
```

C# Solution:

```
/*
* Problem: Sum of K Subarrays With Length at Least M
* 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 int MaxSum(int[] nums, int k, int m) {

}
}
```

C Solution:

```
/*
* Problem: Sum of K Subarrays With Length at Least M
* 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
*/

int maxSum(int* nums, int numsSize, int k, int m) {

}
```

Go Solution:

```
// Problem: Sum of K Subarrays With Length at Least M
// 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 maxSum(nums []int, k int, m int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSum(nums: IntArray, k: Int, m: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of K Subarrays With Length at Least M
// 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_sum(nums: Vec<i32>, k: i32, m: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def maxSum(nums: Array[Int], k: Int, m: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
max_sum(Nums, K, M) ->
.
```

Racket Solution:

```
(define/contract (max-sum nums k m)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```