Problem 689: Maximum Sum of 3 Non-Overlapping Subarrays

Problem Information

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

Problem Description

Given an integer array

nums

and an integer

k

, find three non-overlapping subarrays of length

k

with maximum sum and return them.

Return the result as a list of indices representing the starting position of each interval (

0-indexed

). If there are multiple answers, return the lexicographically smallest one.

Example 1:

Input:

nums = [1,2,1,2,6,7,5,1], k = 2

Output:

[0,3,5]

Explanation:

Subarrays [1, 2], [2, 6], [7, 5] correspond to the starting indices [0, 3, 5]. We could have also taken [2, 1], but an answer of [1, 3, 5] would be lexicographically larger.

Example 2:

Input:

nums = [1,2,1,2,1,2,1,2,1], k = 2

Output:

[0,2,4]

Constraints:

1 <= nums.length <= 2 * 10

4

1 <= nums[i] < 2

16

1 <= k <= floor(nums.length / 3)

Code Snippets

C++:

```
class Solution {
public:
vector<int> maxSumOfThreeSubarrays(vector<int>& nums, int k) {
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxSumOfThreeSubarrays(nums: number[], k: number): number[] {
};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maxSumOfThreeSubarrays(int* nums, int numsSize, int k, int* returnSize)
{

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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



}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum of 3 Non-Overlapping Subarrays
* Difficulty: Hard
* Tags: array, graph, 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:
vector<int> maxSumOfThreeSubarrays(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum of 3 Non-Overlapping Subarrays
* Difficulty: Hard
* Tags: array, graph, 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[] maxSumOfThreeSubarrays(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum of 3 Non-Overlapping Subarrays
Difficulty: Hard
Tags: array, graph, 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 maxSumOfThreeSubarrays(self, nums: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Sum of 3 Non-Overlapping Subarrays
* Difficulty: Hard
* Tags: array, graph, 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
* @return {number[]}
*/
var maxSumOfThreeSubarrays = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum of 3 Non-Overlapping Subarrays
* Difficulty: Hard
* Tags: array, graph, 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 maxSumOfThreeSubarrays(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum of 3 Non-Overlapping Subarrays
* Difficulty: Hard
* Tags: array, graph, 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[] MaxSumOfThreeSubarrays(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum of 3 Non-Overlapping Subarrays
* Difficulty: Hard
* Tags: array, graph, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* maxSumOfThreeSubarrays(int* nums, int numsSize, int k, int* returnSize)
{

}
```

Go Solution:

```
// Problem: Maximum Sum of 3 Non-Overlapping Subarrays
// Difficulty: Hard
// Tags: array, graph, 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 maxSumOfThreeSubarrays(nums []int, k int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Sum of 3 Non-Overlapping Subarrays
// Difficulty: Hard
// Tags: array, graph, 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_of_three_subarrays(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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