Problem 1918: Kth Smallest Subarray Sum

Problem Information

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

Problem Description

Given an integer array

nums

of length

n

and an integer

k

, return

the

k

th

smallest subarray sum

.

A

subarray

is defined as a

non-empty

contiguous sequence of elements in an array. A

subarray sum

is the sum of all elements in the subarray.

Example 1:

Input:

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

Output:

3

Explanation:

The subarrays of [2,1,3] are: - [2] with sum 2 - [1] with sum 1 - [3] with sum 3 - [2,1] with sum 3 - [1,3] with sum 4 - [2,1,3] with sum 6 Ordering the sums from smallest to largest gives 1, 2, 3,

3

, 4, 6. The 4th smallest is 3.

Example 2:

Input:

nums = [3,3,5,5], k = 7

Output:

10

Explanation:

The subarrays of [3,3,5,5] are: - [3] with sum 3 - [3] with sum 3 - [5] with sum 5 - [5] with sum 5 - [3,3] with sum 6 - [3,5] with sum 8 - [5,5] with sum 10 - [3,3,5], with sum 11 - [3,5,5] with sum 13 - [3,3,5,5] with sum 16 Ordering the sums from smallest to largest gives 3, 3, 5, 5, 6, 8,

10

, 11, 13, 16. The 7th smallest is 10.

Constraints:

n == nums.length

1 <= n <= 2 * 10

4

1 <= nums[i] <= 5 * 10

4

1 <= k <= n * (n + 1) / 2

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```

end
end
```

Erlang:

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

Racket:

```
(define/contract (kth-smallest-subarray-sum nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Kth Smallest Subarray Sum
* Difficulty: Medium
* Tags: array, 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:
int kthSmallestSubarraySum(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Kth Smallest Subarray Sum
* Difficulty: Medium
* Tags: array, 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 int kthSmallestSubarraySum(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Kth Smallest Subarray Sum
Difficulty: Medium
Tags: array, 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 kthSmallestSubarraySum(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Kth Smallest Subarray Sum
* Difficulty: Medium
* Tags: array, 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[]} nums
* @param {number} k
* @return {number}
*/
var kthSmallestSubarraySum = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Kth Smallest Subarray Sum
* Difficulty: Medium
* Tags: array, 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 kthSmallestSubarraySum(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Kth Smallest Subarray Sum
* Difficulty: Medium
* Tags: array, 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 int KthSmallestSubarraySum(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Kth Smallest Subarray Sum
* Difficulty: Medium
* Tags: array, 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
*/

int kthSmallestSubarraySum(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Kth Smallest Subarray Sum
// Difficulty: Medium
// Tags: array, 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 kthSmallestSubarraySum(nums []int, k int) int {
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Kth Smallest Subarray Sum
// Difficulty: Medium
// Tags: array, 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_subarray_sum(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (kth-smallest-subarray-sum nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```