Problem 2386: Find the K-Sum of an Array

Problem Information

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

Problem Description

You are given an integer array

nums

and a

positive

integer

k

. You can choose any

subsequence

of the array and sum all of its elements together.

We define the

K-Sum

of the array as the

k

th

largest

subsequence sum that can be obtained (

not

necessarily distinct).

Return

the K-Sum of the array

.

A

subsequence

is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Note

that the empty subsequence is considered to have a sum of

0

.

Example 1:

Input:

nums = [2,4,-2], k = 5

Output:

2

Explanation:

All the possible subsequence sums that we can obtain are the following sorted in decreasing order: 6, 4, 4, 2,

2

, 0, 0, -2. The 5-Sum of the array is 2.

Example 2:

Input:

nums = [1,-2,3,4,-10,12], k = 16

Output:

10

Explanation:

The 16-Sum of the array is 10.

Constraints:

n == nums.length

1 <= n <= 10

5

-10

9

<= nums[i] <= 10

9

1 <= k <= min(2000, 2

n

)

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
long long kSum(int* nums, int numsSize, int k) {

}
```

Go:

```
func kSum(nums []int, k int) int64 {

}
```

Kotlin:

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

}
```

```
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn k_sum(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find the K-Sum of an Array
```

```
* Difficulty: Hard
* Tags: array, 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 kSum(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the K-Sum of an Array
* Difficulty: Hard
* Tags: array, 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 kSum(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the K-Sum of an Array
Difficulty: Hard
Tags: array, 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 kSum(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the K-Sum of an Array
* Difficulty: Hard
* Tags: array, 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[]} nums
* @param {number} k
* @return {number}
*/
var kSum = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the K-Sum of an Array
* Difficulty: Hard
* Tags: array, 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 kSum(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the K-Sum of an Array
* Difficulty: Hard
* Tags: array, 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 KSum(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the K-Sum of an Array
* Difficulty: Hard
* Tags: array, 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 kSum(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Find the K-Sum of an Array
// Difficulty: Hard
// Tags: array, 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 kSum(nums []int, k int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the K-Sum of an Array
// Difficulty: Hard
// Tags: array, 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 k_sum(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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