Problem 1176: Diet Plan Performance

Problem Information

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

Problem Description

A dieter consumes

calories[i]

calories on the

i

-th day.

Given an integer

k

, for

every

consecutive sequence of

k

days (

calories[i], calories[i+1], ..., calories[i+k-1] for all

0 <= i <= n-k

), they look at

T

, the total calories consumed during that sequence of

k

days (

calories[i] + calories[i+1] + ... + calories[i+k-1]

):

If

T < lower

, they performed poorly on their diet and lose 1 point;

If

T > upper

, they performed well on their diet and gain 1 point;

Otherwise, they performed normally and there is no change in points.

Initially, the dieter has zero points. Return the total number of points the dieter has after dieting for

calories.length

days.

Note that the total points can be negative.

Example 1:

Input:

calories = [1,2,3,4,5], k = 1, lower = 3, upper = 3

Output:

0

Explanation

: Since k = 1, we consider each element of the array separately and compare it to lower and upper. calories[0] and calories[1] are less than lower so 2 points are lost. calories[3] and calories[4] are greater than upper so 2 points are gained.

Example 2:

Input:

calories = [3,2], k = 2, lower = 0, upper = 1

Output:

1

Explanation

: Since k = 2, we consider subarrays of length 2. calories[0] + calories[1] > upper so 1 point is gained.

Example 3:

Input:

calories = [6,5,0,0], k = 2, lower = 1, upper = 5

Output:

0

Explanation

: calories[0] + calories[1] > upper so 1 point is gained. lower <= calories[1] + calories[2] <= upper so no change in points. calories[2] + calories[3] < lower so 1 point is lost.

Constraints:

1 <= k <= calories.length <= 10^5

0 <= calories[i] <= 20000

0 <= lower <= upper

Code Snippets

C++:

```
class Solution {
public:
int dietPlanPerformance(vector<int>& calories, int k, int lower, int upper) {

}
};
```

Java:

```
class Solution {
public int dietPlanPerformance(int[] calories, int k, int lower, int upper) {

}
}
```

Python3:

```
class Solution:
def dietPlanPerformance(self, calories: List[int], k: int, lower: int, upper:
```

```
int) -> int:
```

Python:

```
class Solution(object):
def dietPlanPerformance(self, calories, k, lower, upper):
"""
:type calories: List[int]
:type k: int
:type lower: int
:type upper: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} calories
* @param {number} k
* @param {number} lower
* @param {number} upper
* @return {number}
*/
var dietPlanPerformance = function(calories, k, lower, upper) {

};
```

TypeScript:

```
function dietPlanPerformance(calories: number[], k: number, lower: number,
upper: number): number {

};
```

C#:

```
public class Solution {
public int DietPlanPerformance(int[] calories, int k, int lower, int upper) {

}
}
```

C:

```
int dietPlanPerformance(int* calories, int caloriesSize, int k, int lower,
int upper) {

}
```

Go:

```
func dietPlanPerformance(calories []int, k int, lower int, upper int) int {

}
```

Kotlin:

```
class Solution {
fun dietPlanPerformance(calories: IntArray, k: Int, lower: Int, upper: Int):
Int {

}
}
```

Swift:

```
class Solution {
func dietPlanPerformance(_ calories: [Int], _ k: Int, _ lower: Int, _ upper:
Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn diet_plan_performance(calories: Vec<i32>, k: i32, lower: i32, upper:
i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} calories
# @param {Integer} k
# @param {Integer} lower
# @param {Integer} upper
# @return {Integer}
def diet_plan_performance(calories, k, lower, upper)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $calories
* @param Integer $k
* @param Integer $lower
* @param Integer $upper
* @return Integer
*/
function dietPlanPerformance($calories, $k, $lower, $upper) {

}
}
```

Dart:

```
class Solution {
int dietPlanPerformance(List<int> calories, int k, int lower, int upper) {

}
}
```

Scala:

```
object Solution {
def dietPlanPerformance(calories: Array[Int], k: Int, lower: Int, upper:
Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec diet_plan_performance(calories :: [integer], k :: integer, lower ::
integer, upper :: integer) :: integer
def diet_plan_performance(calories, k, lower, upper) do

end
end
```

Erlang:

```
-spec diet_plan_performance(Calories :: [integer()], K :: integer(), Lower ::
integer(), Upper :: integer()) -> integer().
diet_plan_performance(Calories, K, Lower, Upper) ->
.
```

Racket:

```
(define/contract (diet-plan-performance calories k lower upper)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Diet Plan Performance
* Difficulty: Easy
* Tags: array
*
* 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 dietPlanPerformance(vector<int>& calories, int k, int lower, int upper) {

}
};
```

Java Solution:

```
/**
* Problem: Diet Plan Performance
* Difficulty: Easy
* Tags: array
*
* 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 dietPlanPerformance(int[] calories, int k, int lower, int upper) {

}
}
```

Python3 Solution:

```
"""
Problem: Diet Plan Performance
Difficulty: Easy
Tags: array

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 dietPlanPerformance(self, calories: List[int], k: int, lower: int, upper:
int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def dietPlanPerformance(self, calories, k, lower, upper):
"""
:type calories: List[int]
:type k: int
:type lower: int
:type upper: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Diet Plan Performance
* Difficulty: Easy
* Tags: array
*
* 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[]} calories
* @param {number} k
* @param {number} lower
* @param {number} upper
* @return {number}
*/
var dietPlanPerformance = function(calories, k, lower, upper) {

};
```

TypeScript Solution:

```
/**
* Problem: Diet Plan Performance
* Difficulty: Easy
* Tags: array
*
* 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 dietPlanPerformance(calories: number[], k: number, lower: number,
upper: number): number {

};
```

C# Solution:

```
/*
* Problem: Diet Plan Performance
* Difficulty: Easy
* Tags: array
*
* 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 DietPlanPerformance(int[] calories, int k, int lower, int upper) {

}
}
```

C Solution:

```
/*
* Problem: Diet Plan Performance
* Difficulty: Easy
* Tags: array
*
* 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 dietPlanPerformance(int* calories, int caloriesSize, int k, int lower,
int upper) {
}
```

Go Solution:

```
// Problem: Diet Plan Performance
// Difficulty: Easy
// Tags: array
//
// 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 dietPlanPerformance(calories []int, k int, lower int, upper int) int {

}
```

Kotlin Solution:

```
class Solution {
fun dietPlanPerformance(calories: IntArray, k: Int, lower: Int, upper: Int):
Int {

}
}
```

Swift Solution:

```
class Solution {
func dietPlanPerformance(_ calories: [Int], _ k: Int, _ lower: Int, _ upper:
Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Diet Plan Performance
// Difficulty: Easy
// Tags: array
//
// 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 diet_plan_performance(calories: Vec<i32>, k: i32, lower: i32, upper:
i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} calories
# @param {Integer} k
# @param {Integer} lower
# @param {Integer} upper
# @return {Integer}
def diet_plan_performance(calories, k, lower, upper)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $calories
* @param Integer $k
* @param Integer $lower
* @param Integer $upper
* @return Integer
*/
function dietPlanPerformance($calories, $k, $lower, $upper) {

}
}
```

Dart Solution:

```
class Solution {
int dietPlanPerformance(List<int> calories, int k, int lower, int upper) {
}
}
```

Scala Solution:

```
object Solution {
def dietPlanPerformance(calories: Array[Int], k: Int, lower: Int, upper:
Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec diet_plan_performance(calories :: [integer], k :: integer, lower ::
integer, upper :: integer) :: integer
def diet_plan_performance(calories, k, lower, upper) do

end
end
```

Erlang Solution:

```
-spec diet_plan_performance(Calories :: [integer()], K :: integer(), Lower ::
integer(), Upper :: integer()) -> integer().
diet_plan_performance(Calories, K, Lower, Upper) ->
.
```

Racket Solution:

```
(define/contract (diet-plan-performance calories k lower upper)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```