Problem 2607: Make K-Subarray Sums Equal

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

arr

and an integer

k

. The array

arr

is circular. In other words, the first element of the array is the next element of the last element, and the last element of the array is the previous element of the first element.

You can do the following operation any number of times:

Pick any element from

arr

and increase or decrease it by

1

.

Return

the minimum number of operations such that the sum of each

subarray

of length

k

is equal

.

A

subarray

is a contiguous part of the array.

Example 1:

Input:

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

Output:

1

Explanation:

we can do one operation on index 1 to make its value equal to 3. The array after the operation is [1,3,1,3] - Subarray starts at index 0 is [1, 3], and its sum is 4 - Subarray starts at index 1 is [3, 1], and its sum is 4 - Subarray starts at index 2 is [1, 3], and its sum is 4 - Subarray starts at index 3 is [3, 1], and its sum is 4

Example 2:

Input:

arr = [2,5,5,7], k = 3

Output:

5

Explanation:

we can do three operations on index 0 to make its value equal to 5 and two operations on index 3 to make its value equal to 5. The array after the operations is [5,5,5,5] - Subarray starts at index 0 is [5, 5, 5], and its sum is 15 - Subarray starts at index 1 is [5, 5, 5], and its sum is 15 - Subarray starts at index 2 is [5, 5, 5], and its sum is 15 - Subarray starts at index 3 is [5, 5, 5], and its sum is 15

Constraints:

1 <= k <= arr.length <= 10

5

1 <= arr[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long makeSubKSumEqual(int[] arr, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function makeSubKSumEqual(arr: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long MakeSubKSumEqual(int[] arr, int k) {

}
}
```

C:

```
long long makeSubKSumEqual(int* arr, int arrSize, int k) {

}
```

Go:

```
func makeSubKSumEqual(arr []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun makeSubKSumEqual(arr: IntArray, k: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} k
# @return {Integer}
def make_sub_k_sum_equal(arr, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @return Integer
*/
function makeSubKSumEqual($arr, $k) {

}
}
```

Dart:

```
class Solution {
int makeSubKSumEqual(List<int> arr, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec make_sub_k_sum_equal(arr :: [integer], k :: integer) :: integer
```

```
def make_sub_k_sum_equal(arr, k) do

end
end
```

Erlang:

```
-spec make_sub_k_sum_equal(Arr :: [integer()], K :: integer()) -> integer().
make_sub_k_sum_equal(Arr, K) ->
.
```

Racket:

```
(define/contract (make-sub-k-sum-equal arr k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make K-Subarray Sums Equal
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 makeSubKSumEqual(vector<int>& arr, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Make K-Subarray Sums Equal
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 makeSubKSumEqual(int[] arr, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Make K-Subarray Sums Equal
Difficulty: Medium
Tags: array, greedy, math, sort

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 makeSubKSumEqual(self, arr: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make K-Subarray Sums Equal
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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[]} arr
* @param {number} k
* @return {number}
*/
var makeSubKSumEqual = function(arr, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Make K-Subarray Sums Equal
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 makeSubKSumEqual(arr: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Make K-Subarray Sums Equal
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 MakeSubKSumEqual(int[] arr, int k) {

}
}
```

C Solution:

```
/*
* Problem: Make K-Subarray Sums Equal
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* 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 makeSubKSumEqual(int* arr, int arrSize, int k) {

}
```

Go Solution:

```
// Problem: Make K-Subarray Sums Equal
// Difficulty: Medium
// Tags: array, greedy, math, sort
//
// 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 makeSubKSumEqual(arr []int, k int) int64 {
}
```

Kotlin Solution:

```
class Solution {
fun makeSubKSumEqual(arr: IntArray, k: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make K-Subarray Sums Equal
// Difficulty: Medium
// Tags: array, greedy, math, sort
//
// 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 make_sub_k_sum_equal(arr: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} k
# @return {Integer}
def make_sub_k_sum_equal(arr, k)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @return Integer
*/
function makeSubKSumEqual($arr, $k) {

}
}
```

Dart Solution:

```
class Solution {
int makeSubKSumEqual(List<int> arr, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec make_sub_k_sum_equal(arr :: [integer], k :: integer) :: integer
def make_sub_k_sum_equal(arr, k) do

end
end
```

Erlang Solution:

```
-spec make_sub_k_sum_equal(Arr :: [integer()], K :: integer()) -> integer().
make_sub_k_sum_equal(Arr, K) ->
.
```

Racket Solution:

```
(define/contract (make-sub-k-sum-equal arr k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```