Problem 3091: Apply Operations to Make Sum of Array Greater Than or Equal to k

Problem Information

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

Problem Description

You are given a

positive

integer

k

. Initially, you have an array

nums = [1]

.

You can perform

any

of the following operations on the array

any

number of times (

possibly zero

):

Choose any element in the array and

increase

its value by

1

.

Duplicate any element in the array and add it to the end of the array.

Return

the

minimum

number of operations required to make the

sum

of elements of the final array greater than or equal to

k

.

Example 1:

Input:

k = 11

Output:

5

Explanation:

We can do the following operations on the array

nums = [1]

:

Increase the element by

1

three times. The resulting array is

nums = [4]

.

Duplicate the element two times. The resulting array is

nums = [4,4,4]

.

The sum of the final array is

4 + 4 + 4 = 12

which is greater than or equal to

k = 11

.

The total number of operations performed is

3 + 2 = 5

.

Example 2:

Input:

k = 1

Output:

0

Explanation:

The sum of the original array is already greater than or equal to

1

, so no operations are needed.

Constraints:

1 <= k <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int minOperations(int k) {

}
};
```

Java:

```
class Solution {
public int minOperations(int k) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, k: int) -> int:
```

Python:

```
class Solution(object):
def minOperations(self, k):
"""
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} k
* @return {number}
*/
var minOperations = function(k) {

};
```

TypeScript:

```
function minOperations(k: number): number {

};
```

C#:

```
public class Solution {
public int MinOperations(int k) {

}
}
```

C:

```
int minOperations(int k) {

}
```

Go:

```
func minOperations(k int) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minOperations(_ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} k
# @return {Integer}
def min_operations(k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $k
* @return Integer
*/
function minOperations($k) {

}
}
```

Dart:

```
class Solution {
int minOperations(int k) {

}
}
```

Scala:

```
object Solution {
def minOperations(k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(k :: integer) :: integer
def min_operations(k) do

end
end
```

Erlang:

```
-spec min_operations(K :: integer()) -> integer().
min_operations(K) ->
.
```

Racket:

```
(define/contract (min-operations k)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minOperations(int k) {

}
};
```

Java Solution:

```
/**
* Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minOperations(int k) {
}
}
```

Python3 Solution:

```
"""
Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
Difficulty: Medium
Tags: array, greedy, math

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 minOperations(self, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, k):
"""
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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} k
* @return {number}
*/
var minOperations = function(k) {

};
```

TypeScript Solution:

```
/**
* Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minOperations(k: number): number {

};
```

C# Solution:

```
/*
* Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 MinOperations(int k) {

}

}
```

C Solution:

```
/*
* Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minOperations(int k) {

}
```

Go Solution:

```
// Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
// Difficulty: Medium
// Tags: array, greedy, math
//
// 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 minOperations(k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Apply Operations to Make Sum of Array Greater Than or Equal to k
// Difficulty: Medium
// Tags: array, greedy, math
//
// 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 min_operations(k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} k
# @return {Integer}
def min_operations(k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $k
* @return Integer
*/
function minOperations($k) {

}
}
```

Dart Solution:

```
class Solution {
int minOperations(int k) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(k :: integer) :: integer
def min_operations(k) do

end
end
```

Erlang Solution:

```
-spec min_operations(K :: integer()) -> integer().
min_operations(K) ->
.
```

Racket Solution:

```
(define/contract (min-operations k)
(-> exact-integer? exact-integer?)
)
```