Problem 3366: Minimum Array Sum

Problem Information

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

Problem Description

You are given an integer array

nums

and three integers

k

,

op1

, and

op2

.

You can perform the following operations on

nums

:

Operation 1

: Choose an index

i

and divide

nums[i]

by 2,

rounding up

to the nearest whole number. You can perform this operation at most

op1

times, and not more than

once

per index.

Operation 2

: Choose an index

i

and subtract

k

from

nums[i]

, but only if

nums[i]

is greater than or equal to

k

. You can perform this operation at most

op2

times, and not more than

once

per index.

Note:

Both operations can be applied to the same index, but at most once each.

Return the

minimum

possible

sum

of all elements in

nums

after performing any number of operations.

Example 1:

Input:

nums = [2,8,3,19,3], k = 3, op1 = 1, op2 = 1

Output:

23

Explanation:

Apply Operation 2 to

nums[1] = 8

, making

nums[1] = 5

.

Apply Operation 1 to

nums[3] = 19

, making

nums[3] = 10

.

The resulting array becomes

[2, 5, 3, 10, 3]

, which has the minimum possible sum of 23 after applying the operations.

Example 2:

Input:

nums = [2,4,3], k = 3, op1 = 2, op2 = 1

Output:

3

Explanation:

Apply Operation 1 to

nums[0] = 2

, making

nums[0] = 1

.

Apply Operation 1 to

nums[1] = 4

, making

nums[1] = 2

.

Apply Operation 2 to

nums[2] = 3

, making

nums[2] = 0

.

The resulting array becomes

[1, 2, 0]

, which has the minimum possible sum of 3 after applying the operations.

Constraints:

1 <= nums.length <= 100

0 <= nums[i] <= 10

5

0 <= k <= 10

5

0 <= op1, op2 <= nums.length

Code Snippets

C++:

```
class Solution {
public:
int minArraySum(vector<int>& nums, int k, int op1, int op2) {

}
};
```

Java:

```
class Solution {
public int minArraySum(int[] nums, int k, int op1, int op2) {

}
}
```

Python3:

```
class Solution:
def minArraySum(self, nums: List[int], k: int, op1: int, op2: int) -> int:
```

Python:

```
class Solution(object):
def minArraySum(self, nums, k, op1, op2):
"""
:type nums: List[int]
:type k: int
:type op1: int
:type op2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @param {number} op1
* @param {number} op2
* @return {number}
*/
var minArraySum = function(nums, k, op1, op2) {

};
```

TypeScript:

```
function minArraySum(nums: number[], k: number, op1: number, op2: number):
number {

};
```

C#:

```
public class Solution {
public int MinArraySum(int[] nums, int k, int op1, int op2) {

}
}
```

C:

```
int minArraySum(int* nums, int numsSize, int k, int op1, int op2) {

}
```

Go:

```
func minArraySum(nums []int, k int, op1 int, op2 int) int {

}
```

Kotlin:

```
class Solution {
fun minArraySum(nums: IntArray, k: Int, op1: Int, op2: Int): Int {

}
}
```

Swift:

```
class Solution {
func minArraySum(_ nums: [Int], _ k: Int, _ op1: Int, _ op2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_array_sum(nums: Vec<i32>, k: i32, op1: i32, op2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @param {Integer} op1
# @param {Integer} op2
# @return {Integer}
def min_array_sum(nums, k, op1, op2)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @param Integer $op1
* @param Integer $op2
* @return Integer
*/
function minArraySum($nums, $k, $op1, $op2) {

}
}
```

Dart:

```
class Solution {
int minArraySum(List<int> nums, int k, int op1, int op2) {

}
}
```

Scala:

```
object Solution {
def minArraySum(nums: Array[Int], k: Int, op1: Int, op2: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_array_sum(nums :: [integer], k :: integer, op1 :: integer, op2 ::
integer) :: integer
def min_array_sum(nums, k, op1, op2) do

end
```

```
end
```

Erlang:

```
-spec min_array_sum(Nums :: [integer()], K :: integer(), Op1 :: integer(),
Op2 :: integer()) -> integer().
min_array_sum(Nums, K, Op1, Op2) ->
.
```

Racket:

```
(define/contract (min-array-sum nums k op1 op2)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Array Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minArraySum(vector<int>& nums, int k, int op1, int op2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Array Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minArraySum(int[] nums, int k, int op1, int op2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Array Sum
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minArraySum(self, nums: List[int], k: int, op1: int, op2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minArraySum(self, nums, k, op1, op2):
"""
:type nums: List[int]
:type k: int
:type op1: int
:type op2: int
```

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Array Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @param {number} k
* @param {number} op1
* @param {number} op2
* @return {number}
*/
var minArraySum = function(nums, k, op1, op2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Array Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minArraySum(nums: number[], k: number, op1: number, op2: number):
number {
};
```

C# Solution:

```
/*
* Problem: Minimum Array Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinArraySum(int[] nums, int k, int op1, int op2) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Array Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minArraySum(int* nums, int numsSize, int k, int op1, int op2) {

}
```

Go Solution:

```
// Problem: Minimum Array Sum
// Difficulty: Medium
// Tags: array, dp
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minArraySum(nums []int, k int, op1 int, op2 int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minArraySum(nums: IntArray, k: Int, op1: Int, op2: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minArraySum(_ nums: [Int], _ k: Int, _ op1: Int, _ op2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Array Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn min_array_sum(nums: Vec<i32>, k: i32, op1: i32, op2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @param {Integer} op1
# @param {Integer} op2
# @return {Integer}
def min_array_sum(nums, k, op1, op2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @param Integer $op1
* @param Integer $op2
* @return Integer
*/
function minArraySum($nums, $k, $op1, $op2) {

}
}
```

Dart Solution:

```
class Solution {
int minArraySum(List<int> nums, int k, int op1, int op2) {

}
}
```

Scala Solution:

```
object Solution {
def minArraySum(nums: Array[Int], k: Int, op1: Int, op2: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_array_sum(nums :: [integer], k :: integer, op1 :: integer, op2 ::
integer) :: integer
def min_array_sum(nums, k, op1, op2) do

end
end
```

Erlang Solution:

```
-spec min_array_sum(Nums :: [integer()], K :: integer(), Op1 :: integer(),
Op2 :: integer()) -> integer().
min_array_sum(Nums, K, Op1, Op2) ->
.
```

Racket Solution:

```
(define/contract (min-array-sum nums k op1 op2)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```