Problem 1962: Remove Stones to Minimize the Total

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

piles

, where

piles[i]

represents the number of stones in the

i

th

pile, and an integer

k

. You should apply the following operation

exactly

k

times:

Choose any

piles[i]

and

remove

floor(piles[i] / 2)

stones from it.

Notice

that you can apply the operation on the

same

pile more than once.

Return

the

minimum

possible total number of stones remaining after applying the

k

operations

.

floor(x)

is the

largest

integer that is

smaller

than or

equal

to

x

(i.e., rounds

x

down).

Example 1:

Input:

piles = [5,4,9], k = 2

Output:

12

Explanation:

Steps of a possible scenario are: - Apply the operation on pile 2. The resulting piles are [5,4,

5

]. - Apply the operation on pile 0. The resulting piles are [

3

,4,5]. The total number of stones in [3,4,5] is 12.

Example 2:

Input:

piles = [4,3,6,7], k = 3

Output:

12

Explanation:

Steps of a possible scenario are: - Apply the operation on pile 2. The resulting piles are [4,3,

3

,7]. - Apply the operation on pile 3. The resulting piles are [4,3,3,

4

]. - Apply the operation on pile 0. The resulting piles are [

2

,3,3,4]. The total number of stones in [2,3,3,4] is 12.

Constraints:

1 <= piles.length <= 10

5

1 <= piles[i] <= 10

4

1 <= k <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int minStoneSum(vector<int>& piles, int k) {

}
};
```

Java:

```
class Solution {
public int minStoneSum(int[] piles, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minStoneSum(piles: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MinStoneSum(int[] piles, int k) {

}
}
```

C:

```
int minStoneSum(int* piles, int pilesSize, int k) {

}
```

Go:

```
func minStoneSum(piles []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minStoneSum(piles: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn min_stone_sum(piles: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} piles
# @param {Integer} k
# @return {Integer}
def min_stone_sum(piles, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $piles
* @param Integer $k
* @return Integer
*/
function minStoneSum($piles, $k) {

}
}
```

Dart:

```
class Solution {
int minStoneSum(List<int> piles, int k) {

}
}
```

Scala:

```
object Solution {
def minStoneSum(piles: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_stone_sum(piles :: [integer], k :: integer) :: integer
def min_stone_sum(piles, k) do

end
end
```

Erlang:

```
-spec min_stone_sum(Piles :: [integer()], K :: integer()) -> integer().
min_stone_sum(Piles, K) ->
.
```

Racket:

```
(define/contract (min-stone-sum piles k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Stones to Minimize the Total
* Difficulty: Medium
* Tags: array, greedy, 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:
int minStoneSum(vector<int>& piles, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Stones to Minimize the Total
* Difficulty: Medium
* Tags: array, greedy, 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 int minStoneSum(int[] piles, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Stones to Minimize the Total
Difficulty: Medium
Tags: array, greedy, 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 minStoneSum(self, piles: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove Stones to Minimize the Total
* Difficulty: Medium
* Tags: array, greedy, 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[]} piles
* @param {number} k
* @return {number}
*/
var minStoneSum = function(piles, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Stones to Minimize the Total
* Difficulty: Medium
* Tags: array, greedy, 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 minStoneSum(piles: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Remove Stones to Minimize the Total
* Difficulty: Medium
* Tags: array, greedy, 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 int MinStoneSum(int[] piles, int k) {

}
}
```

C Solution:

```
/*
* Problem: Remove Stones to Minimize the Total
* Difficulty: Medium
* Tags: array, greedy, 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
```

```
*/

int minStoneSum(int* piles, int pilesSize, int k) {

}
```

Go Solution:

```
// Problem: Remove Stones to Minimize the Total
// Difficulty: Medium
// Tags: array, greedy, 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 minStoneSum(piles []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minStoneSum(piles: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove Stones to Minimize the Total
// Difficulty: Medium
// Tags: array, greedy, 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 min_stone_sum(piles: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} piles
# @param {Integer} k
# @return {Integer}
def min_stone_sum(piles, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $piles
* @param Integer $k
* @return Integer
*/
function minStoneSum($piles, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minStoneSum(List<int> piles, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minStoneSum(piles: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_stone_sum(piles :: [integer], k :: integer) :: integer
def min_stone_sum(piles, k) do

end
end
```

Erlang Solution:

```
-spec min_stone_sum(Piles :: [integer()], K :: integer()) -> integer().
min_stone_sum(Piles, K) ->
.
```

Racket Solution:

```
(define/contract (min-stone-sum piles k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```