Problem 2208: Minimum Operations to Halve Array Sum

Problem Information

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

Problem Description

You are given an array

nums

of positive integers. In one operation, you can choose

any

number from

nums

and reduce it to

exactly

half the number. (Note that you may choose this reduced number in future operations.)

Return

the

minimum

number of operations to reduce the sum of nums

by

at least

half.

Example 1:

Input:

nums = [5,19,8,1]

Output:

3

Explanation:

The initial sum of nums is equal to 5 + 19 + 8 + 1 = 33. The following is one of the ways to reduce the sum by at least half: Pick the number 19 and reduce it to 9.5. Pick the number 9.5 and reduce it to 4.75. Pick the number 8 and reduce it to 4. The final array is [5, 4.75, 4, 1] with a total sum of 5 + 4.75 + 4 + 1 = 14.75. The sum of nums has been reduced by 33 - 14.75 = 18.25, which is at least half of the initial sum, 18.25 >= 33/2 = 16.5. Overall, 3 operations were used so we return 3. It can be shown that we cannot reduce the sum by at least half in less than 3 operations.

Example 2:

Input:

nums = [3,8,20]

Output:

3

Explanation:

The initial sum of nums is equal to 3 + 8 + 20 = 31. The following is one of the ways to reduce the sum by at least half: Pick the number 20 and reduce it to 10. Pick the number 10 and reduce it to 5. Pick the number 3 and reduce it to 1.5. The final array is [1.5, 8, 5] with a total sum of 1.5 + 8 + 5 = 14.5. The sum of nums has been reduced by 31 - 14.5 = 16.5, which is at least half of the initial sum, 16.5 >= 31/2 = 15.5. Overall, 3 operations were used so we return 3. It can be shown that we cannot reduce the sum by at least half in less than 3 operations.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

7

Code Snippets

C++:

```
class Solution {
public:
int halveArray(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int halveArray(int[] nums) {

}
}
```

Python3:

```
class Solution:
def halveArray(self, nums: List[int]) -> int:
```

Python:

```
class Solution(object):
def halveArray(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
var halveArray = function(nums) {

};
```

TypeScript:

```
function halveArray(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int HalveArray(int[] nums) {

}
}
```

C:

```
int halveArray(int* nums, int numsSize) {

}
```

Go:

```
func halveArray(nums []int) int {
}
```

Kotlin:

```
class Solution {
fun halveArray(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func halveArray(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn halve_array(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def halve_array(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function halveArray($nums) {

}
}
```

Dart:

```
class Solution {
int halveArray(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def halveArray(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec halve_array(nums :: [integer]) :: integer
def halve_array(nums) do

end
end
```

Erlang:

```
-spec halve_array(Nums :: [integer()]) -> integer().
halve_array(Nums) ->
.
```

Racket:

```
(define/contract (halve-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Halve Array Sum
* 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 halveArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Halve Array Sum
* 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 halveArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Halve Array Sum
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 halveArray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def halveArray(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Halve Array Sum
* 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[]} nums
* @return {number}
*/
var halveArray = function(nums) {
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Halve Array Sum
* 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 halveArray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Halve Array Sum
* 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 HalveArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Halve Array Sum
* 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 halveArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Halve Array Sum
// 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 halveArray(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun halveArray(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func halveArray(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Halve Array Sum
// 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 halve_array(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def halve_array(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function halveArray($nums) {

}
}
```

Dart Solution:

```
class Solution {
int halveArray(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def halveArray(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec halve_array(nums :: [integer]) :: integer
def halve_array(nums) do

end
end
```

Erlang Solution:

```
-spec halve_array(Nums :: [integer()]) -> integer().
halve_array(Nums) ->
.
```

Racket Solution:

```
(define/contract (halve-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```