Problem 1558: Minimum Numbers of Function Calls to Make Target Array

Problem Information

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

Problem Description

You are given an integer array

nums

. You have an integer array

arr

of the same length with all values set to

0

initially. You also have the following

modify

function:

```
func modify(arr, op, idx){
    //add by 1 index idx
    if (op == 0) {
        arr[idx] = arr[idx] + 1
    }
    //multiply by 2 all elements
    if (op == 1) {
        for(i = 0; i < arr.length; i++) {
            arr[i] = arr[i] * 2
        }
    }
}
```

You want to use the modify function to convert

arr

to

nums

using the minimum number of calls.

Return

the minimum number of function calls to make

nums

from

arr

.

The test cases are generated so that the answer fits in a

32-bit

signed integer.

Example 1:

Input:

nums = [1,5]

Output:

5

Explanation:

Increment by 1 (second element): [0, 0] to get [0, 1] (1 operation). Double all the elements: [0, 1] -> [0, 2] -> [0, 4] (2 operations). Increment by 1 (both elements) [0, 4] -> [1, 4] ->

[1, 5]

(2 operations). Total of operations: 1 + 2 + 2 = 5.

Example 2:

Input:

nums = [2,2]

Output:

3

Explanation:

Increment by 1 (both elements) [0, 0] -> [0, 1] -> [1, 1] (2 operations). Double all the elements: [1, 1] ->

[2, 2]

(1 operation). Total of operations: 2 + 1 = 3.

Example 3:

Input:

nums = [4,2,5]

Output:

6

Explanation:

(initial)[0,0,0] -> [1,0,0] -> [1,0,1] -> [2,0,2] -> [2,1,2] -> [4,2,4] ->

[4,2,5]

(nums).

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func minOperations(nums []int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Numbers of Function Calls to Make Target Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Numbers of Function Calls to Make Target Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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[] nums) {
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Numbers of Function Calls to Make Target Array
Difficulty: Medium
Tags: array, greedy

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, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Numbers of Function Calls to Make Target Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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 minOperations = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Numbers of Function Calls to Make Target Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Numbers of Function Calls to Make Target Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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[] nums) {

}

}
```

C Solution:

```
/*
* Problem: Minimum Numbers of Function Calls to Make Target Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Numbers of Function Calls to Make Target Array
// Difficulty: Medium
// Tags: array, greedy
//
// 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(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Numbers of Function Calls to Make Target Array
// Difficulty: Medium
// Tags: array, greedy
//
// 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(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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