Problem 2154: Keep Multiplying Found Values by Two

Problem Information

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

Problem Description

You are given an array of integers

nums

. You are also given an integer

original

which is the first number that needs to be searched for in

nums

.

You then do the following steps:

If

original

is found in

nums

,

multiply

it by two (i.e., set

original = 2 * original

).

Otherwise,

stop

the process.

Repeat

this process with the new number as long as you keep finding the number.

Return

the

final

value of

original

.

Example 1:

Input:

nums = [5,3,6,1,12], original = 3

Output:

24

Explanation:

- 3 is found in nums. 3 is multiplied by 2 to obtain 6. - 6 is found in nums. 6 is multiplied by 2 to obtain 12. - 12 is found in nums. 12 is multiplied by 2 to obtain 24. - 24 is not found in nums. Thus, 24 is returned.

Example 2:

Input:

nums = [2,7,9], original = 4

Output:

4

Explanation:

- 4 is not found in nums. Thus, 4 is returned.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i], original <= 1000

Code Snippets

C++:

```
class Solution {
public:
int findFinalValue(vector<int>& nums, int original) {

}
};
```

Java:

```
class Solution {
public int findFinalValue(int[] nums, int original) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findFinalValue(nums: number[], original: number): number {

};
```

C#:

```
public class Solution {
public int FindFinalValue(int[] nums, int original) {

}
}
```

C:

```
int findFinalValue(int* nums, int numsSize, int original) {

}
```

Go:

```
func findFinalValue(nums []int, original int) int {

}
```

Kotlin:

```
class Solution {
fun findFinalValue(nums: IntArray, original: Int): Int {

}
}
```

Swift:

```
class Solution {
func findFinalValue(_ nums: [Int], _ original: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_final_value(nums: Vec<i32>, original: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} original
# @return {Integer}
def find_final_value(nums, original)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $original
* @return Integer
*/
function findFinalValue($nums, $original) {

}
}
```

Dart:

```
class Solution {
int findFinalValue(List<int> nums, int original) {

}
}
```

Scala:

```
object Solution {
def findFinalValue(nums: Array[Int], original: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_final_value(nums :: [integer], original :: integer) :: integer
def find_final_value(nums, original) do
```

```
end
end
```

Erlang:

```
-spec find_final_value(Nums :: [integer()], Original :: integer()) ->
integer().
find_final_value(Nums, Original) ->
.
```

Racket:

```
(define/contract (find-final-value nums original)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Keep Multiplying Found Values by Two
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int findFinalValue(vector<int>& nums, int original) {

}
};
```

Java Solution:

```
/**
* Problem: Keep Multiplying Found Values by Two
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int findFinalValue(int[] nums, int original) {

}
}
```

Python3 Solution:

```
"""
Problem: Keep Multiplying Found Values by Two
Difficulty: Easy
Tags: array, hash, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def findFinalValue(self, nums: List[int], original: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Keep Multiplying Found Values by Two
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number} original
* @return {number}
*/
var findFinalValue = function(nums, original) {

};
```

TypeScript Solution:

```
/**
* Problem: Keep Multiplying Found Values by Two
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function findFinalValue(nums: number[], original: number): number {

};
```

C# Solution:

```
/*
* Problem: Keep Multiplying Found Values by Two
* Difficulty: Easy
```

```
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int FindFinalValue(int[] nums, int original) {

}
}
```

C Solution:

```
/*
* Problem: Keep Multiplying Found Values by Two
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int findFinalValue(int* nums, int numsSize, int original) {

}
```

Go Solution:

```
// Problem: Keep Multiplying Found Values by Two
// Difficulty: Easy
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func findFinalValue(nums []int, original int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun findFinalValue(nums: IntArray, original: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findFinalValue(_ nums: [Int], _ original: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Keep Multiplying Found Values by Two
// Difficulty: Easy
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn find_final_value(nums: Vec<i32>, original: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} original
# @return {Integer}
def find_final_value(nums, original)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $original
* @return Integer
*/
function findFinalValue($nums, $original) {

}
}
```

Dart Solution:

```
class Solution {
int findFinalValue(List<int> nums, int original) {

}
}
```

Scala Solution:

```
object Solution {
def findFinalValue(nums: Array[Int], original: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_final_value(nums :: [integer], original :: integer) :: integer
def find_final_value(nums, original) do

end
end
```

Erlang Solution:

```
-spec find_final_value(Nums :: [integer()], Original :: integer()) ->
integer().
find_final_value(Nums, Original) ->
.
```

Racket Solution:

```
(define/contract (find-final-value nums original)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```