Problem 1250: Check If It Is a Good Array

Problem Information

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

Problem Description

Given an array

nums

of positive integers. Your task is to select some subset of

nums

, multiply each element by an integer and add all these numbers. The array is said to be

good

if you can obtain a sum of

1

from the array by any possible subset and multiplicand.

Return

True

if the array is

good

otherwise return

False

.

Example 1:

Input:

nums = [12,5,7,23]

Output:

true

Explanation:

Pick numbers 5 and 7. 5*3 + 7*(-2) = 1

Example 2:

Input:

nums = [29,6,10]

Output:

true

Explanation:

Pick numbers 29, 6 and 10. 29*1 + 6*(-3) + 10*(-1) = 1

Example 3:

Input:

nums = [3,6]

Output:

false

Constraints:

1 <= nums.length <= 10^5

1 <= nums[i] <= 10^9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean isGoodArray(int[] nums) {

}
}
```

Python3:

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

Python:

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

```
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function isGoodArray(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool IsGoodArray(int[] nums) {

}
}
```

C:

```
bool isGoodArray(int* nums, int numsSize) {

}
```

Go:

```
func isGoodArray(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun isGoodArray(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool isGoodArray(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-good-array nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check If It Is a Good Array
* Difficulty: Hard
* Tags: array, math
*
* 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:
bool isGoodArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Check If It Is a Good Array
* Difficulty: Hard
* Tags: array, math
*
* 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 boolean isGoodArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Check If It Is a Good Array
Difficulty: Hard
Tags: array, math
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check If It Is a Good Array
* Difficulty: Hard
* Tags: array, math
*
* 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 {boolean}
*/
var isGoodArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Check If It Is a Good Array
* Difficulty: Hard
* Tags: array, math
*
* 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 isGoodArray(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check If It Is a Good Array
* Difficulty: Hard
* Tags: array, math
*
* 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 bool IsGoodArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Check If It Is a Good Array
* Difficulty: Hard
* Tags: array, math
*
* 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
```

```
*/

bool isGoodArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Check If It Is a Good Array
// Difficulty: Hard
// Tags: array, math
//
// 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 isGoodArray(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isGoodArray(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check If It Is a Good Array
// Difficulty: Hard
// Tags: array, math
```

```
//
// 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 is_good_array(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool isGoodArray(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-good-array nums)
(-> (listof exact-integer?) boolean?)
)
```