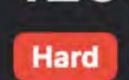
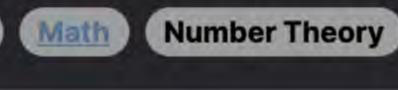
1250. Check If It Is a Good Array





Problem Description



The problem presents us with an array of positive integers called nums. Our goal is to check if it's possible to select a subset of these numbers, multiply each selected number by an integer, and then sum them up to get a total of 1. If this is possible, the array is considered "good". Otherwise, it is not good. The task is to determine if the given array is good or not, and we are to return True if it is good and False if it is not.

Intuition

To determine if an array is good or not, we can use a mathematical concept called the Greatest Common Divisor (GCD). The GCD of two numbers is the largest number that divides both of them without leaving a remainder. If we can extend this concept to find the GCD of all numbers in the array and the result is 1, this implies that we can form the number 1 as a linear combination of the array numbers, using the Bezout's identity theorem.

This means that if the GCD of the entire array is 1, there must be some subset of numbers in the array which can be multiplied by some integers to sum up to 1, since 1 is the only number that when used in linear combinations can produce any integer (and specifically the integer 1 in our case).

The Python code makes use of the reduce function and the gcd function from the math module. The reduce function is used to apply the gcd function cumulatively to the items of nums, from left to right, so that we are effectively finding the GCD of all the numbers in the array. If the GCD turns out to be 1, the function returns True. Otherwise, it returns False. This simple approach elegantly checks if the array is good using built-in functions to perform the necessary calculations.

The solution to this problem leverages the mathematical property of the Greatest Common Divisor (GCD) and a functional

programming construct in Python.

Solution Approach

Here's the step-by-step breakdown of the implementation:

• Step 1: Import the gcd function from the math module, gcd takes two numbers as arguments and returns their greatest common divisor.

Step 2: Use the reduce function from the functools module. The reduce function is a tool for performing a cumulative operation

over an iterable. In this case, it applies the gcd function starting with the first two elements of the nums array, and then

consecutively using the result as the first argument and the next element in the array as the second argument.

- Step 3: The reduce function will apply the gcd function progressively across all elements of the array. Essentially, this means it will calculate the gcd of the first and second elements, then take that result and calculate the gcd with the third element, and so on until it processes the entire array.
- Step 4: After reduce has processed all the elements, we evaluate whether the accumulated GCD is 1. If the result is 1, it signifies that there is some combination of the array elements along with respective multiplicands that could add up to 1 (thanks to Bezout's identity). If not, it implies that no such combination is possible.
- The algorithm relies on the following data structures and patterns: Arrays: The given input is an array that the algorithm iterates through.

• Step 5: The isGoodArray method returns True if the GCD is 1, signaling that the array is "good", or returns False otherwise.

 Functional Programming: Using reduce is an example of functional programming, as it applies a function cumulatively to the items of an iterable in a non-mutable way.

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

- And that's it. By efficiently applying the gcd function cumulatively across all elements in the nums array, we can ascertain whether the array is "good" with a single line of code after the necessary functions are imported:

return reduce(gcd, nums) == 1

```
Let's illustrate the solution approach with a small example. Suppose we are given the following array of positive integers:
```

Now, let's walk through the process step by step:

class Solution:

1 nums = [6, 10, 15]

array nums.

Example Walkthrough

• Step 1: We first import the gcd function from the math module, which will allow us to calculate the greatest common divisor of

from functools import reduce

1 from functools import reduce

from math import gcd

class Solution:

two given numbers.

• Step 4: The result (2) is then used with the next element in the array, which is 15. The gcd of 2 and 15 is 1.

• Step 5: Since the reduce function has finished processing all elements, we check if the accumulated GCD is 1. In our case, it is

• Step 2: We use the reduce function from the functools module to apply the gcd function cumulatively to the numbers in our

indeed 1.

• Step 3: First, reduce applies the gcd function to the first two elements, which are 6 and 10. The gcd of 6 and 10 is 2.

- According to Bezout's identity, because the final GCD is 1, there must be a combination of integer multiples of some or all numbers in
- array is "good". Here's a glimpse of how the code operates in this scenario:

our array nums that can add up to 1. Hence, for our input array, the function isGoodArray would evaluate to True, indicating that the

from math import gcd nums = [6, 10, 15]result = reduce(gcd, nums) is good = result == 1 # This would be True in this case

The simplicity of this algorithm lies in its use of the gcd function to examine the entire array in a single sweep. If the gcd of all

elements is 1, we can confidently say that the array is "good" as it meets the criteria outlined in the problem description.

The function uses the greatest common divisor (gcd) to check if the array is 'good'.

// If at any point the gcd becomes 1, we can return true immediately.

```
Python Solution
```

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

public boolean isGoodArray(int[] nums) {

for (int num : nums) {

if (gcdValue == 1) {

return true;

// which is in the numeric header.

return greatestCommonDivisor == 1;

greatestCommonDivisor = std::gcd(number, greatestCommonDivisor);

// The array is "good" if the GCD is 1 after processing all elements.

int gcdValue = 0; // Initialize the gcd value to 0.

// Iterate over each element in the array to find the overall gcd.

```
# An array is 'good' if the gcd of all numbers in the array is 1.
           # Use the reduce function to apply the gcd function cumulatively to the items of 'nums',
           # from left to right, which reduces the array to a single value.
10
           # Then, check if the final gcd value is 1.
11
           return reduce(gcd, nums) == 1
12
13
```

gcdValue = gcd(num, gcdValue); // Update gcdValue using the current element and the accumulated gcd.

class Solution { // Method to check if the array is a good array based on the condition. // An array is considered good if the Greatest Common Divisor (GCD) of all its elements is 1.

10

11

12

13

14

Java Solution

```
15
           // The array is good if the final gcd value is 1.
16
           return gcdValue == 1;
17
18
       // Helper method to calculate the gcd of two numbers using Euclid's algorithm.
19
20
       private int gcd(int a, int b) {
           if (b == 0) {
21
               // If the second number b is 0, then gcd is the first number a.
23
               return a;
24
           } else {
25
               // Recursively call gcd with the second number and the remainder of a divided by b.
26
               return gcd(b, a % b);
27
28
29
30
C++ Solution
1 #include <vector>
   #include <numeric> // Required for std::gcd
   class Solution {
   public:
       // Function to determine if the array is a "good" array.
       // A "good" array is defined as an array where the greatest
       // common divisor of all its elements is 1.
       bool isGoodArray(std::vector<int>& nums) {
           int greatestCommonDivisor = 0; // Initialize to 0
10
11
           // Iterating through each element in the given array 'nums'.
           for (int number : nums) {
12
               // Update the greatest common divisor using std::gcd,
13
```

Typescript Solution

14

15

16

17

18

19

21

20 };

```
// Importing gcd function from a math utilities module
   // Note: TypeScript does not have a gcd function built-in, so you'll need to
   // either implement it yourself or include a library that provides it.
   function gcd(a: number, b: number): number {
     // Base case for the recursion
     if (b === 0) return a;
     // Recursively calling gcd with the remainder
     return gcd(b, a % b);
10
11
   // Function to determine if the array is a "good" array.
   // A "good" array is defined as an array where the greatest
  // common divisor of all its elements is 1.
   function isGoodArray(nums: number[]): boolean {
     let greatestCommonDivisor: number = 0; // Initialize to 0
17
18
     // Iterating through each element in the given array 'nums'.
19
     for (let number of nums) {
       // Update the greatest common divisor using the gcd function.
20
       greatestCommonDivisor = gcd(greatestCommonDivisor, number);
21
22
23
24
     // The array is "good" if the GCD is 1 after processing all elements.
25
     return greatestCommonDivisor === 1;
26 }
27
Time and Space Complexity
```

performs on the list elements.

Time Complexity The time complexity of the function is determined by the reduce function and the gcd (greatest common divisor) operations it

value. The gcd function runs in O(log(min(a, b))) time, where a and b are the numbers whose GCD is being calculated.

Assuming there are n elements in the nums list, the reduce function will perform n-1 gcd operations. Due to the nature of GCD

calculation, where after the first operation, the resulting GCD will often be lesser than or equal to the smallest number among the

operands, each subsequent gcd operation is typically faster than the last. However, for worst-case analysis, we'll consider each

• The reduce function applies the gcd function cumulatively to the items of the list, from start to end, to reduce the list to a single

operation to have the complexity of $O(\log(k))$, where k is the smallest element after each operation. Therefore, the time complexity is:

1 0(n*log(k))

Where n is the number of elements in the list and k is the smallest number in the list at each step of the reduction.

Space Complexity

The space complexity of the code is 0(1).

and updates the accumulator in-place.

- This space complexity comes from the fact that the gcd operations do not require additional space that scales with the input
- size, as they are computed in constant space. • The reduce function does not create any new data structures that depend on the input size; it just iterates over the existing list

Thus, the space complexity is:

1 0(1)