2601. Prime Subtraction Operation

Medium

Topics

Companies

Hint

You are given a **0-indexed** integer array `nums` of length `n`.

You can perform the following operation as many times as you want:

- Pick an index `i` that you haven’t picked before, and pick a prime `p` **strictly less than** `nums[i]`, then subtract `p` from `nums[i]`.

Return *true if you can make* `nums` *a strictly increasing array using the above operation and false otherwise.*

A **strictly increasing array** is an array whose each element is strictly greater than its preceding element.

Example 1:

Input: nums = [4,9,6,10]
Output: true
Explanation: In the first operation: Pick i = 0 and p = 3, and then subtract 3 from nums[0], so that nums becomes [1,9,6,10].
In the second operation: i = 1, p = 7, subtract 7 from nums[1], so nums becomes equal to [1,2,6,10].
After the second operation, nums is sorted in strictly increasing order, so the answer is true.

Example 2:

Input: nums = [6,8,11,12]
Output: true
Explanation: Initially nums is sorted in strictly increasing order, so we don't need to make any operations.

Example 3:

Input: nums = [5,8,3]
Output: false
Explanation: It can be proven that there is no way to perform operations to make nums sorted in strictly increasing order, so the answer is false.

Constraints:

- `1 <= nums.length <= 1000`
- `1 <= nums[i] <= 1000`
- `nums.length == n`

Solution:

```
class Solution {

    public boolean primeSubOperation(int[] nums) {

        int maxElement = getMaxElement(nums);



        // Create Sieve of Eratosthenes array to identify prime numbers

        boolean[] sieve = new boolean[maxElement + 1];

        fill(sieve, true);

        sieve[1] = false;

        for (int i = 2; i <= Math.sqrt(maxElement + 1); i++) {

            if (sieve[i]) {

                for (int j = i * i; j <= maxElement; j += i) {

                    sieve[j] = false;

                }

            }

        }



        // Check if array can be made strictly increasing by subtracting
prime numbers

        int currValue = 1;

        int i = 0;

        while (i < nums.length) {
```

```java
            int difference = nums[i] - currValue;

            // Return false if current number is already smaller than required value
            if (difference < 0) {
                return false;
            }

            // Move to next number if difference is prime or zero
            if (sieve[difference] == true || difference == 0) {
                i++;
                currValue++;
            } else {
                currValue++;
            }
        }
        return true;
    }

    // Helper method to find maximum element in array
    private int getMaxElement(int[] nums) {
        int max = -1;
        for (int num : nums) {
            if (num > max) {
                max = num;
```

```
            }
        }
        return max;
    }

    // Helper method to initialize boolean array
    private void fill(boolean[] arr, boolean value) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = value;
        }
    }
}
```