Problem 2601: Prime Subtraction Operation

Problem Information

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

Problem Description

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

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {boolean}
*/
```

```
var primeSubOperation = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (prime-sub-operation nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Prime Subtraction Operation
* Difficulty: Medium
* Tags: array, greedy, math, sort, search
*
* 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 primeSubOperation(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Prime Subtraction Operation
* Difficulty: Medium
* Tags: array, greedy, math, sort, search
*
* 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 primeSubOperation(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Prime Subtraction Operation
Difficulty: Medium
Tags: array, greedy, math, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Prime Subtraction Operation
* Difficulty: Medium
* Tags: array, greedy, math, sort, search
*
* 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 primeSubOperation = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Prime Subtraction Operation
* Difficulty: Medium
* Tags: array, greedy, math, sort, search
*
* 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 primeSubOperation(nums: number[]): boolean {
};
```

C# Solution:

```
/*
* Problem: Prime Subtraction Operation
* Difficulty: Medium
* Tags: array, greedy, math, sort, search
*
* 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 PrimeSubOperation(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Prime Subtraction Operation
* Difficulty: Medium
* Tags: array, greedy, math, sort, search
*
* 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 primeSubOperation(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Prime Subtraction Operation
// Difficulty: Medium
// Tags: array, greedy, math, sort, search
//
// 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 primeSubOperation(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Prime Subtraction Operation
// Difficulty: Medium
// Tags: array, greedy, math, sort, search
//
// 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 prime_sub_operation(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```php
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (prime-sub-operation nums)
(-> (listof exact-integer?) boolean?)
)
```