Problem 1354: Construct Target Array With Multiple Sums

Problem Information

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

Problem Description

You are given an array

target

of n integers. From a starting array

arr

consisting of

n

1's, you may perform the following procedure :

let

x

be the sum of all elements currently in your array.

choose index

i

, such that

0 <= i < n

and set the value of

arr

at index

i

to

x

.

You may repeat this procedure as many times as needed.

Return

true

if it is possible to construct the

target

array from

arr

, otherwise, return

false

.

Example 1:

Input:

target = [9,3,5]

Output:

true

Explanation:

Start with arr = [1, 1, 1] [1, 1, 1], sum = 3 choose index 1 [1, 3, 1], sum = 5 choose index 2 [1, 3, 5], sum = 9 choose index 0 [9, 3, 5] Done

Example 2:

Input:

target = [1,1,1,2]

Output:

false

Explanation:

Impossible to create target array from [1,1,1,1].

Example 3:

Input:

target = [8,5]

Output:

true

Constraints:

n == target.length

1 <= n <= 5 * 10

4

1 <= target[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
bool isPossible(vector<int>& target) {

}
};
```

Java:

```
class Solution {
public boolean isPossible(int[] target) {

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

```
function isPossible(target: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool IsPossible(int[] target) {

}
}
```

C:

```
bool isPossible(int* target, int targetSize) {

}
```

Go:

```
func isPossible(target []int) bool {

}
```

Kotlin:

```
class Solution {
fun isPossible(target: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func isPossible(_ target: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_possible(target: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} target
# @return {Boolean}
def is_possible(target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $target
* @return Boolean
*/
function isPossible($target) {

}
}
```

Dart:

```
class Solution {
bool isPossible(List<int> target) {

}
}
```

Scala:

```
object Solution {
def isPossible(target: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_possible(target :: [integer]) :: boolean
def is_possible(target) do

end
end
```

Erlang:

```
-spec is_possible(Target :: [integer()]) -> boolean().
is_possible(Target) ->
.
```

Racket:

```
(define/contract (is-possible target)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Construct Target Array With Multiple Sums
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 isPossible(vector<int>& target) {

}
};
```

Java Solution:

```
/**
* Problem: Construct Target Array With Multiple Sums
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 isPossible(int[] target) {

}
}
```

Python3 Solution:

```
"""
Problem: Construct Target Array With Multiple Sums
Difficulty: Hard
Tags: array, queue, heap
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Construct Target Array With Multiple Sums
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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[]} target
* @return {boolean}
*/
var isPossible = function(target) {

};
```

TypeScript Solution:

```
/**
* Problem: Construct Target Array With Multiple Sums
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 isPossible(target: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Construct Target Array With Multiple Sums
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 IsPossible(int[] target) {

}
}
```

C Solution:

```
/*
* Problem: Construct Target Array With Multiple Sums
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 isPossible(int* target, int targetSize) {

}
```

Go Solution:

```
// Problem: Construct Target Array With Multiple Sums
// Difficulty: Hard
// Tags: array, queue, heap
//
// 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 isPossible(target []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isPossible(target: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isPossible(_ target: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Construct Target Array With Multiple Sums
// Difficulty: Hard
// Tags: array, queue, heap
```

```
//
// 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_possible(target: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} target
# @return {Boolean}
def is_possible(target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $target
* @return Boolean
*/
function isPossible($target) {

}
}
```

Dart Solution:

```
class Solution {
bool isPossible(List<int> target) {

}
}
```

Scala Solution:

```
object Solution {
def isPossible(target: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_possible(target :: [integer]) :: boolean
def is_possible(target) do

end
end
```

Erlang Solution:

```
-spec is_possible(Target :: [integer()]) -> boolean().
is_possible(Target) ->
.
```

Racket Solution:

```
(define/contract (is-possible target)
(-> (listof exact-integer?) boolean?)
)
```