Problem 2126: Destroying Asteroids

Problem Information

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

Problem Description

You are given an integer

mass

, which represents the original mass of a planet. You are further given an integer array

asteroids

, where

asteroids[i]

is the mass of the

i

th

asteroid.

You can arrange for the planet to collide with the asteroids in

any arbitrary order

. If the mass of the planet is

greater than or equal to

the mass of the asteroid, the asteroid is

destroyed

and the planet

gains

the mass of the asteroid. Otherwise, the planet is destroyed.

Return

true

if

all

asteroids can be destroyed. Otherwise, return

false

.

Example 1:

Input:

mass = 10, asteroids = [3,9,19,5,21]

Output:

true

Explanation:

One way to order the asteroids is [9,19,5,3,21]: - The planet collides with the asteroid with a mass of 9. New planet mass: 10 + 9 = 19 - The planet collides with the asteroid with a mass of 19. New planet mass: 19 + 19 = 38 - The planet collides with the asteroid with a mass of 5. New planet mass: 38 + 5 = 43 - The planet collides with the asteroid with a mass of 3. New planet mass: 43 + 3 = 46 - The planet collides with the asteroid with a mass of 21. New planet mass: 46 + 21 = 67 All asteroids are destroyed.

Example 2:

Input:

mass = 5, asteroids = [4,9,23,4]

Output:

false

Explanation:

The planet cannot ever gain enough mass to destroy the asteroid with a mass of 23. After the planet destroys the other asteroids, it will have a mass of 5 + 4 + 9 + 4 = 22. This is less than 23, so a collision would not destroy the last asteroid.

Constraints:

1 <= mass <= 10

5

1 <= asteroids.length <= 10

5

1 <= asteroids[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
bool asteroidsDestroyed(int mass, vector<int>& asteroids) {

}
};
```

Java:

```
class Solution {
public boolean asteroidsDestroyed(int mass, int[] asteroids) {

}
}
```

Python3:

```
class Solution:
def asteroidsDestroyed(self, mass: int, asteroids: List[int]) -> bool:
```

Python:

```
class Solution(object):
def asteroidsDestroyed(self, mass, asteroids):
"""
:type mass: int
:type asteroids: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} mass
* @param {number[]} asteroids
* @return {boolean}
*/
var asteroidsDestroyed = function(mass, asteroids) {

};
```

TypeScript:

```
function asteroidsDestroyed(mass: number, asteroids: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool AsteroidsDestroyed(int mass, int[] asteroids) {

}
}
```

C:

```
bool asteroidsDestroyed(int mass, int* asteroids, int asteroidsSize) {

}
```

Go:

```
func asteroidsDestroyed(mass int, asteroids []int) bool {

}
```

Kotlin:

```
class Solution {
fun asteroidsDestroyed(mass: Int, asteroids: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func asteroidsDestroyed(_ mass: Int, _ asteroids: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn asteroids_destroyed(mass: i32, asteroids: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer} mass
# @param {Integer[]} asteroids
# @return {Boolean}
def asteroids_destroyed(mass, asteroids)

end
```

PHP:

```
class Solution {

/**
* @param Integer $mass
* @param Integer[] $asteroids
* @return Boolean
*/
function asteroidsDestroyed($mass, $asteroids) {

}
}
```

Dart:

```
class Solution {
bool asteroidsDestroyed(int mass, List<int> asteroids) {

}
}
```

Scala:

```
object Solution {
def asteroidsDestroyed(mass: Int, asteroids: Array[Int]): Boolean = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec asteroids_destroyed(mass :: integer, asteroids :: [integer]) :: boolean
def asteroids_destroyed(mass, asteroids) do

end
end
```

Erlang:

```
-spec asteroids_destroyed(Mass :: integer(), Asteroids :: [integer()]) ->
boolean().
asteroids_destroyed(Mass, Asteroids) ->
.
```

Racket:

```
(define/contract (asteroids-destroyed mass asteroids)
(-> exact-integer? (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Destroying Asteroids
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 asteroidsDestroyed(int mass, vector<int>& asteroids) {

}
};
```

Java Solution:

```
/**
* Problem: Destroying Asteroids
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 asteroidsDestroyed(int mass, int[] asteroids) {

}
}
```

Python3 Solution:

```
"""
Problem: Destroying Asteroids
Difficulty: Medium
Tags: array, greedy, sort

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 asteroidsDestroyed(self, mass: int, asteroids: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def asteroidsDestroyed(self, mass, asteroids):
"""
:type mass: int
:type asteroids: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Destroying Asteroids
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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} mass
* @param {number[]} asteroids
* @return {boolean}
*/
var asteroidsDestroyed = function(mass, asteroids) {

};
```

TypeScript Solution:

```
/**
* Problem: Destroying Asteroids
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 asteroidsDestroyed(mass: number, asteroids: number[]): boolean {
```

```
};
```

C# Solution:

```
/*
* Problem: Destroying Asteroids
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 AsteroidsDestroyed(int mass, int[] asteroids) {

}
}
```

C Solution:

```
/*
* Problem: Destroying Asteroids
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 asteroidsDestroyed(int mass, int* asteroids, int asteroidsSize) {

}
```

Go Solution:

```
// Problem: Destroying Asteroids
// Difficulty: Medium
```

```
// Tags: array, greedy, sort
//
// 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 asteroidsDestroyed(mass int, asteroids []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun asteroidsDestroyed(mass: Int, asteroids: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func asteroidsDestroyed(_ mass: Int, _ asteroids: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Destroying Asteroids
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 asteroids_destroyed(mass: i32, asteroids: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} mass
# @param {Integer[]} asteroids
# @return {Boolean}
def asteroids_destroyed(mass, asteroids)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $mass
* @param Integer[] $asteroids
* @return Boolean
*/
function asteroidsDestroyed($mass, $asteroids) {

}
}
```

Dart Solution:

```
class Solution {
bool asteroidsDestroyed(int mass, List<int> asteroids) {

}
}
```

Scala Solution:

```
object Solution {
def asteroidsDestroyed(mass: Int, asteroids: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec asteroids_destroyed(mass :: integer, asteroids :: [integer]) :: boolean
def asteroids_destroyed(mass, asteroids) do

end
end
```

Erlang Solution:

```
-spec asteroids_destroyed(Mass :: integer(), Asteroids :: [integer()]) ->
boolean().
asteroids_destroyed(Mass, Asteroids) ->
.
```

Racket Solution:

```
(define/contract (asteroids-destroyed mass asteroids)
(-> exact-integer? (listof exact-integer?) boolean?)
)
```