Problem 2453: Destroy Sequential Targets

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

consisting of positive integers, representing targets on a number line. You are also given an integer

space

.

You have a machine which can destroy targets.

Seeding

the machine with some

nums[i]

allows it to destroy all targets with values that can be represented as

nums[i] + c * space

, where

c

is any non-negative integer. You want to destroy the

maximum

number of targets in

nums

.

Return

the

minimum value

of

nums[i]

you can seed the machine with to destroy the maximum number of targets.

Example 1:

Input:

nums = [3,7,8,1,1,5], space = 2

Output:

1

Explanation:

If we seed the machine with nums[3], then we destroy all targets equal to 1,3,5,7,9,... In this case, we would destroy 5 total targets (all except for nums[2]). It is impossible to destroy more than 5 targets, so we return nums[3].

Example 2:

Input:

nums = [1,3,5,2,4,6], space = 2

Output:

1

Explanation:

Seeding the machine with nums[0], or nums[3] destroys 3 targets. It is not possible to destroy more than 3 targets. Since nums[0] is the minimal integer that can destroy 3 targets, we return 1.

Example 3:

Input:

nums = [6,2,5], space = 100

Output:

2

Explanation:

Whatever initial seed we select, we can only destroy 1 target. The minimal seed is nums[1].

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= space <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int destroyTargets(vector<int>& nums, int space) {

}
};
```

Java:

```
class Solution {
public int destroyTargets(int[] nums, int space) {

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} space
* @return {number}
*/
var destroyTargets = function(nums, space) {

};
```

TypeScript:

```
function destroyTargets(nums: number[], space: number): number {

};
```

C#:

```
public class Solution {
public int DestroyTargets(int[] nums, int space) {

}
}
```

C:

```
int destroyTargets(int* nums, int numsSize, int space) {

}
```

Go:

```
func destroyTargets(nums []int, space int) int {

}
```

Kotlin:

```
class Solution {
fun destroyTargets(nums: IntArray, space: Int): Int {

}
}
```

Swift:

```
class Solution {
func destroyTargets(_ nums: [Int], _ space: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn destroy_targets(nums: Vec<i32>, space: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} space
# @return {Integer}
def destroy_targets(nums, space)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $space
* @return Integer
*/
function destroyTargets($nums, $space) {

}
```

```
}
```

Dart:

```
class Solution {
int destroyTargets(List<int> nums, int space) {

}
}
```

Scala:

```
object Solution {
def destroyTargets(nums: Array[Int], space: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec destroy_targets(nums :: [integer], space :: integer) :: integer
def destroy_targets(nums, space) do

end
end
```

Erlang:

```
-spec destroy_targets(Nums :: [integer()], Space :: integer()) -> integer().
destroy_targets(Nums, Space) ->
.
```

Racket:

```
(define/contract (destroy-targets nums space)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Destroy Sequential Targets
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int destroyTargets(vector<int>& nums, int space) {

}
};
```

Java Solution:

```
/**
* Problem: Destroy Sequential Targets
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int destroyTargets(int[] nums, int space) {

}
}
```

Python3 Solution:

```
"""
Problem: Destroy Sequential Targets
Difficulty: Medium
Tags: array, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def destroyTargets(self, nums: List[int], space: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Destroy Sequential Targets
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number} space
* @return {number}
*/
var destroyTargets = function(nums, space) {

};
```

TypeScript Solution:

```
/**
* Problem: Destroy Sequential Targets
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function destroyTargets(nums: number[], space: number): number {

};
```

C# Solution:

```
/*
* Problem: Destroy Sequential Targets
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int DestroyTargets(int[] nums, int space) {

}
}
```

C Solution:

```
/*
* Problem: Destroy Sequential Targets
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int destroyTargets(int* nums, int numsSize, int space) {

}
```

Go Solution:

```
// Problem: Destroy Sequential Targets
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func destroyTargets(nums []int, space int) int {

}
```

Kotlin Solution:

```
class Solution {
fun destroyTargets(nums: IntArray, space: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func destroyTargets(_ nums: [Int], _ space: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Destroy Sequential Targets
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn destroy_targets(nums: Vec<i32>, space: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} space
# @return {Integer}
def destroy_targets(nums, space)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $space
* @return Integer
*/
function destroyTargets($nums, $space) {

}
}
```

Dart Solution:

```
class Solution {
int destroyTargets(List<int> nums, int space) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def destroyTargets(nums: Array[Int], space: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec destroy_targets(nums :: [integer], space :: integer) :: integer
def destroy_targets(nums, space) do

end
end
```

Erlang Solution:

```
-spec destroy_targets(Nums :: [integer()], Space :: integer()) -> integer().
destroy_targets(Nums, Space) ->
.
```

Racket Solution:

```
(define/contract (destroy-targets nums space)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```