Problem 1705: Maximum Number of Eaten Apples

Problem Information

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

Problem Description

There is a special kind of apple tree that grows apples every day for

n

days. On the

i

th

day, the tree grows

apples[i]

apples that will rot after

days[i]

days, that is on day

i + days[i]

the apples will be rotten and cannot be eaten. On some days, the apple tree does not grow any apples, which are denoted by apples[i] == 0

and

days[i] == 0

.

You decided to eat

at most

one apple a day (to keep the doctors away). Note that you can keep eating after the first

n

days.

Given two integer arrays

days

and

apples

of length

n

, return

the maximum number of apples you can eat.

Example 1:

Input:

apples = [1,2,3,5,2], days = [3,2,1,4,2]

Output:

7

Explanation:

You can eat 7 apples: - On the first day, you eat an apple that grew on the first day. - On the second day, you eat an apple that grew on the second day. - On the third day, you eat an apple that grew on the second day. After this day, the apples that grew on the third day rot. - On the fourth to the seventh days, you eat apples that grew on the fourth day.

Example 2:

Input:

apples = [3,0,0,0,0,2], days = [3,0,0,0,0,2]

Output:

5

Explanation:

You can eat 5 apples: - On the first to the third day you eat apples that grew on the first day. - Do nothing on the fouth and fifth days. - On the sixth and seventh days you eat apples that grew on the sixth day.

Constraints:

n == apples.length == days.length

1 <= n <= 2 * 10

4

0 <= apples[i], days[i] <= 2 * 10

4

days[i] = 0

if and only if

apples[i] = 0

.

Code Snippets

C++:

```
class Solution {
public:
int eatenApples(vector<int>& apples, vector<int>& days) {

}
};
```

Java:

```
class Solution {
public int eatenApples(int[] apples, int[] days) {

}
}
```

Python3:

```
class Solution:
def eatenApples(self, apples: List[int], days: List[int]) -> int:
```

Python:

```
class Solution(object):
def eatenApples(self, apples, days):
"""
:type apples: List[int]
:type days: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} apples
* @param {number[]} days
* @return {number}
*/
var eatenApples = function(apples, days) {

};
```

TypeScript:

```
function eatenApples(apples: number[], days: number[]): number {

};
```

C#:

```
public class Solution {
public int EatenApples(int[] apples, int[] days) {

}
}
```

C:

```
int eatenApples(int* apples, int applesSize, int* days, int daysSize) {

}
```

Go:

```
func eatenApples(apples []int, days []int) int {

}
```

Kotlin:

```
class Solution {
fun eatenApples(apples: IntArray, days: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func eatenApples(_ apples: [Int], _ days: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn eaten_apples(apples: Vec<i32>, days: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} apples
# @param {Integer[]} days
# @return {Integer}
def eaten_apples(apples, days)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $apples
* @param Integer[] $days
* @return Integer
*/
function eatenApples($apples, $days) {

}
}
```

Dart:

```
class Solution {
int eatenApples(List<int> apples, List<int> days) {

}
}
```

Scala:

```
object Solution {
def eatenApples(apples: Array[Int], days: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec eaten_apples(apples :: [integer], days :: [integer]) :: integer
def eaten_apples(apples, days) do

end
end
```

Erlang:

```
-spec eaten_apples(Apples :: [integer()], Days :: [integer()]) -> integer().
eaten_apples(Apples, Days) ->
.
```

Racket:

```
(define/contract (eaten-apples apples days)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Eaten Apples
* Difficulty: Medium
* Tags: array, tree, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int eatenApples(vector<int>& apples, vector<int>& days) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Eaten Apples
* Difficulty: Medium
* Tags: array, tree, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int eatenApples(int[] apples, int[] days) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Eaten Apples
Difficulty: Medium
Tags: array, tree, greedy, queue, heap

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def eatenApples(self, apples: List[int], days: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def eatenApples(self, apples, days):
"""
:type apples: List[int]
:type days: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Eaten Apples
* Difficulty: Medium
* Tags: array, tree, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} apples
* @param {number[]} days
* @return {number}
*/
var eatenApples = function(apples, days) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Eaten Apples
* Difficulty: Medium
* Tags: array, tree, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function eatenApples(apples: number[], days: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Eaten Apples
* Difficulty: Medium
* Tags: array, tree, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int EatenApples(int[] apples, int[] days) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Eaten Apples
* Difficulty: Medium
* Tags: array, tree, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

int eatenApples(int* apples, int applesSize, int* days, int daysSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Eaten Apples
// Difficulty: Medium
// Tags: array, tree, greedy, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func eatenApples(apples []int, days []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun eatenApples(apples: IntArray, days: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func eatenApples(_ apples: [Int], _ days: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Eaten Apples
// Difficulty: Medium
// Tags: array, tree, greedy, queue, heap
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn eaten_apples(apples: Vec<i32>, days: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} apples
# @param {Integer[]} days
# @return {Integer}
def eaten_apples(apples, days)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $apples
* @param Integer[] $days
* @return Integer
*/
function eatenApples($apples, $days) {

}
}
```

Dart Solution:

```
class Solution {
int eatenApples(List<int> apples, List<int> days) {

}
}
```

Scala Solution:

```
object Solution {
def eatenApples(apples: Array[Int], days: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec eaten_apples(apples :: [integer], days :: [integer]) :: integer
def eaten_apples(apples, days) do

end
end
```

Erlang Solution:

```
-spec eaten_apples(Apples :: [integer()], Days :: [integer()]) -> integer().
eaten_apples(Apples, Days) ->
.
```

Racket Solution:

```
(define/contract (eaten-apples apples days)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```