Problem 904: Fruit Into Baskets

Problem Information

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

Problem Description

You are visiting a farm that has a single row of fruit trees arranged from left to right. The trees are represented by an integer array

fruits

where

fruits[i]

is the

type

of fruit the

i

th

tree produces.

You want to collect as much fruit as possible. However, the owner has some strict rules that you must follow:

You only have

two

baskets, and each basket can only hold a

single type

of fruit. There is no limit on the amount of fruit each basket can hold.

Starting from any tree of your choice, you must pick

exactly one fruit

from

every

tree (including the start tree) while moving to the right. The picked fruits must fit in one of your baskets.

Once you reach a tree with fruit that cannot fit in your baskets, you must stop.

Given the integer array

fruits

, return

the

maximum

number of fruits you can pick

.

Example 1:

Input:

fruits = [

1,2,1

]

Output:

3

Explanation:

We can pick from all 3 trees.

Example 2:

Input:

fruits = [0,

1,2,2

]

Output:

3

Explanation:

We can pick from trees [1,2,2]. If we had started at the first tree, we would only pick from trees [0,1].

Example 3:

Input:

fruits = [1,

2,3,2,2

]

Output:

4

Explanation:

We can pick from trees [2,3,2,2]. If we had started at the first tree, we would only pick from trees [1,2].

Constraints:

1 <= fruits.length <= 10

5

0 <= fruits[i] < fruits.length

Code Snippets

C++:

```
class Solution {
public:
int totalFruit(vector<int>& fruits) {

}
};
```

Java:

```
class Solution {
public int totalFruit(int[] fruits) {

}
}
```

Python3:

```
class Solution:
def totalFruit(self, fruits: List[int]) -> int:
```

Python:

```
class Solution(object):
def totalFruit(self, fruits):
"""
:type fruits: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} fruits
* @return {number}
*/
var totalFruit = function(fruits) {

};
```

TypeScript:

```
function totalFruit(fruits: number[]): number {

};
```

C#:

```
public class Solution {
public int TotalFruit(int[] fruits) {

}
}
```

C:

```
int totalFruit(int* fruits, int fruitsSize) {

}
```

Go:

```
func totalFruit(fruits []int) int {

}
```

Kotlin:

```
class Solution {
fun totalFruit(fruits: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func totalFruit(_ fruits: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn total_fruit(fruits: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} fruits
# @return {Integer}
def total_fruit(fruits)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $fruits
* @return Integer
*/
function totalFruit($fruits) {

}
}
```

Dart:

```
class Solution {
int totalFruit(List<int> fruits) {

}
}
```

Scala:

```
object Solution {
def totalFruit(fruits: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec total_fruit(fruits :: [integer]) :: integer
def total_fruit(fruits) do

end
end
```

Erlang:

```
-spec total_fruit(Fruits :: [integer()]) -> integer().
total_fruit(Fruits) ->
.
```

Racket:

```
(define/contract (total-fruit fruits)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Fruit Into Baskets
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 totalFruit(vector<int>& fruits) {

}
};
```

Java Solution:

```
/**
* Problem: Fruit Into Baskets
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 totalFruit(int[] fruits) {

}
}
```

Python3 Solution:

```
"""
Problem: Fruit Into Baskets
Difficulty: Medium
Tags: array, tree, hash

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 totalFruit(self, fruits: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def totalFruit(self, fruits):
"""
:type fruits: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Fruit Into Baskets
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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[]} fruits
* @return {number}
*/
var totalFruit = function(fruits) {

};
```

TypeScript Solution:

```
/**
* Problem: Fruit Into Baskets
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 totalFruit(fruits: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Fruit Into Baskets
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 TotalFruit(int[] fruits) {

}
}
```

C Solution:

```
/*
* Problem: Fruit Into Baskets
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 totalFruit(int* fruits, int fruitsSize) {

}
```

Go Solution:

```
// Problem: Fruit Into Baskets
// Difficulty: Medium
// Tags: array, tree, hash
//
// 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 totalFruit(fruits []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun totalFruit(fruits: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func totalFruit(_ fruits: [Int]) -> Int {
}
}
```

Rust Solution:

```
// Problem: Fruit Into Baskets
// Difficulty: Medium
// Tags: array, tree, hash
//
// 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 total_fruit(fruits: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} fruits
# @return {Integer}
def total_fruit(fruits)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $fruits
* @return Integer
*/
function totalFruit($fruits) {

}
}
```

Dart Solution:

```
class Solution {
int totalFruit(List<int> fruits) {

}
}
```

Scala Solution:

```
object Solution {
def totalFruit(fruits: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec total_fruit(fruits :: [integer]) :: integer
def total_fruit(fruits) do

end
end
```

Erlang Solution:

```
-spec total_fruit(Fruits :: [integer()]) -> integer().
total_fruit(Fruits) ->
.
```

Racket Solution:

```
(define/contract (total-fruit fruits)
(-> (listof exact-integer?) exact-integer?)
)
```