Problem 455: Assign Cookies

Problem Information

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

Problem Description

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child

i

has a greed factor

g[i]

, which is the minimum size of a cookie that the child will be content with; and each cookie

j

has a size

s[j]

. If

s[j] >= g[i]

, we can assign the cookie

j

to the child

i

, and the child

i

will be content. Your goal is to maximize the number of your content children and output the maximum number.

Example 1:

Input:

g = [1,2,3], s = [1,1]

Output:

1

Explanation:

You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3. And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content. You need to output 1.

Example 2:

Input:

g = [1,2], s = [1,2,3]

Output:

2

Explanation:

You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2. You have 3 cookies and their sizes are big enough to gratify all of the children, You need to output 2.

Constraints:

1 <= g.length <= 3 * 10

4

0 <= s.length <= 3 * 10

4

1 <= g[i], s[j] <= 2

31

- 1

Note:

This question is the same as

2410: Maximum Matching of Players With Trainers.

Code Snippets

C++:

```
class Solution {
public:
int findContentChildren(vector<int>& g, vector<int>& s) {

}
};
```

Java:

```
class Solution {
public int findContentChildren(int[] g, int[] s) {

}
}
```

Python3:

```
class Solution:
def findContentChildren(self, g: List[int], s: List[int]) -> int:
```

Python:

```
class Solution(object):
def findContentChildren(self, g, s):
"""
:type g: List[int]
:type s: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} g
* @param {number[]} s
* @return {number}
*/
var findContentChildren = function(g, s) {

};
```

TypeScript:

```
function findContentChildren(g: number[], s: number[]): number {

};
```

C#:

```
public class Solution {
public int FindContentChildren(int[] g, int[] s) {
```

```
}
}
```

C:

```
int findContentChildren(int* g, int gSize, int* s, int sSize) {

}
```

Go:

```
func findContentChildren(g []int, s []int) int {

}
```

Kotlin:

```
class Solution {
fun findContentChildren(g: IntArray, s: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func findContentChildren(_ g: [Int], _ s: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_content_children(g: Vec<i32>, s: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} g
# @param {Integer[]} s
# @return {Integer}
def find_content_children(g, s)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $g
* @param Integer[] $s
* @return Integer
*/
function findContentChildren($g, $s) {

}
}
```

Dart:

```
class Solution {
int findContentChildren(List<int> g, List<int> s) {

}
}
```

Scala:

```
object Solution {
def findContentChildren(g: Array[Int], s: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_content_children(g :: [integer], s :: [integer]) :: integer
def find_content_children(g, s) do
```

```
end
end
```

Erlang:

```
-spec find_content_children(G :: [integer()], S :: [integer()]) -> integer().
find_content_children(G, S) ->
.
```

Racket:

```
(define/contract (find-content-children g s)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Assign Cookies
* Difficulty: Easy
* 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:
int findContentChildren(vector<int>& g, vector<int>& s) {

}
};
```

Java Solution:

```
/**
* Problem: Assign Cookies
```

```
* Difficulty: Easy
* 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 int findContentChildren(int[] g, int[] s) {

}
}
```

Python3 Solution:

```
"""
Problem: Assign Cookies
Difficulty: Easy
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 findContentChildren(self, g: List[int], s: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findContentChildren(self, g, s):
"""
:type g: List[int]
:type s: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Assign Cookies
* Difficulty: Easy
* 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[]} g
* @param {number[]} s
* @return {number}
*/
var findContentChildren = function(g, s) {

};
```

TypeScript Solution:

```
/**
* Problem: Assign Cookies
* Difficulty: Easy
* 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 findContentChildren(g: number[], s: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Assign Cookies
* Difficulty: Easy
* 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 int FindContentChildren(int[] g, int[] s) {

}
}
```

C Solution:

```
/*
* Problem: Assign Cookies
* Difficulty: Easy
* 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
*/

int findContentChildren(int* g, int gSize, int* s, int sSize) {

}
```

Go Solution:

```
// Problem: Assign Cookies
// Difficulty: Easy
// 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 findContentChildren(g []int, s []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findContentChildren(g: IntArray, s: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func findContentChildren(_ g: [Int], _ s: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Assign Cookies
// Difficulty: Easy
// 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 find_content_children(g: Vec<i32>, s: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} g
# @param {Integer[]} s
# @return {Integer}
def find_content_children(g, s)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $g
* @param Integer[] $s
* @return Integer
*/
function findContentChildren($g, $s) {

}
}
```

Dart Solution:

```
class Solution {
int findContentChildren(List<int> g, List<int> s) {

}
}
```

Scala Solution:

```
object Solution {
def findContentChildren(g: Array[Int], s: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_content_children(g :: [integer], s :: [integer]) :: integer
def find_content_children(g, s) do

end
end
```

Erlang Solution:

```
-spec find_content_children(G :: [integer()], S :: [integer()]) -> integer().
find_content_children(G, S) ->
.
```

Racket Solution:

```
(define/contract (find-content-children g s)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```