Problem 1753: Maximum Score From Removing Stones

Problem Information

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

Problem Description

You are playing a solitaire game with

three piles

of stones of sizes

a

,

b

,■■■■■■ and

c

respectively. Each turn you choose two

different non-empty

piles, take one stone from each, and add

1

point to your score. The game stops when there are

fewer than two non-empty

piles (meaning there are no more available moves).

Given three integers

a

,

b

,■■■■■ and

c

, return

the

maximum

score

you can get.

Example 1:

Input:

a = 2, b = 4, c = 6

Output:

6

Explanation:

The starting state is (2, 4, 6). One optimal set of moves is: - Take from 1st and 3rd piles, state is now (1, 4, 5) - Take from 1st and 3rd piles, state is now (0, 4, 4) - Take from 2nd and 3rd piles, state is now (0, 3, 3) - Take from 2nd and 3rd piles, state is now (0, 2, 2) - Take from 2nd and 3rd piles, state is now (0, 1, 1) - Take from 2nd and 3rd piles, state is now (0, 0, 0) There are fewer than two non-empty piles, so the game ends. Total: 6 points.

Example 2:

Input:

a = 4, b = 4, c = 6

Output:

7

Explanation:

The starting state is (4, 4, 6). One optimal set of moves is: - Take from 1st and 2nd piles, state is now (3, 3, 6) - Take from 1st and 3rd piles, state is now (2, 3, 5) - Take from 1st and 3rd piles, state is now (1, 3, 4) - Take from 1st and 3rd piles, state is now (0, 3, 3) - Take from 2nd and 3rd piles, state is now (0, 2, 2) - Take from 2nd and 3rd piles, state is now (0, 1, 1) - Take from 2nd and 3rd piles, state is now (0, 0, 0) There are fewer than two non-empty piles, so the game ends. Total: 7 points.

Example 3:

Input:

a = 1, b = 8, c = 8

Output:

8

Explanation:

One optimal set of moves is to take from the 2nd and 3rd piles for 8 turns until they are empty. After that, there are fewer than two non-empty piles, so the game ends.

Constraints:

1 <= a, b, c <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int maximumScore(int a, int b, int c) {

}
};
```

Java:

```
class Solution {
public int maximumScore(int a, int b, int c) {

}
}
```

Python3:

```
class Solution:
def maximumScore(self, a: int, b: int, c: int) -> int:
```

Python:

```
class Solution(object):
def maximumScore(self, a, b, c):
"""
:type a: int
:type b: int
:type c: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} a
* @param {number} b
* @param {number} c
* @return {number}
*/
var maximumScore = function(a, b, c) {

};
```

TypeScript:

```
function maximumScore(a: number, b: number, c: number): number {

};
```

C#:

```
public class Solution {
public int MaximumScore(int a, int b, int c) {

}
}
```

C:

```
int maximumScore(int a, int b, int c) {

}
```

Go:

```
func maximumScore(a int, b int, c int) int {

}
```

Kotlin:

```
class Solution {
fun maximumScore(a: Int, b: Int, c: Int): Int {
```

```
}
}
```

Swift:

```
class Solution {
func maximumScore(_ a: Int, _ b: Int, _ c: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_score(a: i32, b: i32, c: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @return {Integer}
def maximum_score(a, b, c)

end
```

PHP:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @return Integer
*/
function maximumScore($a, $b, $c) {

}
```

```
}
```

Dart:

```
class Solution {
int maximumScore(int a, int b, int c) {

}
}
```

Scala:

```
object Solution {
def maximumScore(a: Int, b: Int, c: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_score(a :: integer, b :: integer, c :: integer) :: integer
def maximum_score(a, b, c) do

end
end
```

Erlang:

```
-spec maximum_score(A :: integer(), B :: integer(), C :: integer()) ->
integer().
maximum_score(A, B, C) ->
.
```

Racket:

```
(define/contract (maximum-score a b c)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Score From Removing Stones
* Difficulty: Medium
* Tags: greedy, math, queue, heap
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int maximumScore(int a, int b, int c) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score From Removing Stones
* Difficulty: Medium
* Tags: greedy, math, queue, heap
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maximumScore(int a, int b, int c) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score From Removing Stones
```

```
Difficulty: Medium
Tags: greedy, math, queue, heap

Approach: Greedy algorithm with local optimal choices
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maximumScore(self, a: int, b: int, c: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumScore(self, a, b, c):
"""
:type a: int
:type b: int
:type c: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Score From Removing Stones
* Difficulty: Medium
* Tags: greedy, math, queue, heap
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} a
* @param {number} b
* @param {number} c
* @return {number}
```

```
*/
var maximumScore = function(a, b, c) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score From Removing Stones
* Difficulty: Medium
* Tags: greedy, math, queue, heap
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function maximumScore(a: number, b: number, c: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Score From Removing Stones
* Difficulty: Medium
* Tags: greedy, math, queue, heap
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MaximumScore(int a, int b, int c) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score From Removing Stones
* Difficulty: Medium
* Tags: greedy, math, queue, heap
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int maximumScore(int a, int b, int c) {

}
```

Go Solution:

```
// Problem: Maximum Score From Removing Stones
// Difficulty: Medium
// Tags: greedy, math, queue, heap
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func maximumScore(a int, b int, c int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumScore(a: Int, b: Int, c: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumScore(_ a: Int, _ b: Int, _ c: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Maximum Score From Removing Stones
// Difficulty: Medium
// Tags: greedy, math, queue, heap
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn maximum_score(a: i32, b: i32, c: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @return {Integer}
def maximum_score(a, b, c)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @return Integer
*/
function maximumScore($a, $b, $c) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int maximumScore(int a, int b, int c) {

}
}
```

Scala Solution:

```
object Solution {
def maximumScore(a: Int, b: Int, c: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_score(a :: integer, b :: integer, c :: integer) :: integer
def maximum_score(a, b, c) do

end
end
```

Erlang Solution:

```
-spec maximum_score(A :: integer(), B :: integer(), C :: integer()) ->
integer().
maximum_score(A, B, C) ->
.
```

Racket Solution:

```
(define/contract (maximum-score a b c)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```