

Problem 2279: Maximum Bags With Full Capacity of Rocks

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You have

n

bags numbered from

0

to

$n - 1$

. You are given two

0-indexed

integer arrays

capacity

and

rocks

. The

i

th

bag can hold a maximum of

capacity[i]

rocks and currently contains

rocks[i]

rocks. You are also given an integer

additionalRocks

, the number of additional rocks you can place in

any

of the bags.

Return

the

maximum

number of bags that could have full capacity after placing the additional rocks in some bags.

Example 1:

Input:

capacity = [2,3,4,5], rocks = [1,2,4,4], additionalRocks = 2

Output:

3

Explanation:

Place 1 rock in bag 0 and 1 rock in bag 1. The number of rocks in each bag are now [2,3,4,4]. Bags 0, 1, and 2 have full capacity. There are 3 bags at full capacity, so we return 3. It can be shown that it is not possible to have more than 3 bags at full capacity. Note that there may be other ways of placing the rocks that result in an answer of 3.

Example 2:

Input:

capacity = [10,2,2], rocks = [2,2,0], additionalRocks = 100

Output:

3

Explanation:

Place 8 rocks in bag 0 and 2 rocks in bag 2. The number of rocks in each bag are now [10,2,2]. Bags 0, 1, and 2 have full capacity. There are 3 bags at full capacity, so we return 3. It can be shown that it is not possible to have more than 3 bags at full capacity. Note that we did not use all of the additional rocks.

Constraints:

$n == \text{capacity.length} == \text{rocks.length}$

$1 \leq n \leq 5 * 10$

4

$1 \leq \text{capacity}[i] \leq 10$

9

$0 \leq \text{rocks}[i] \leq \text{capacity}[i]$

$1 \leq \text{additionalRocks} \leq 10$

9

Code Snippets

C++:

```
class Solution {  
public:  
    int maximumBags(vector<int>& capacity, vector<int>& rocks, int  
    additionalRocks) {  
  
    }  
};
```

Java:

```
class Solution {  
    public int maximumBags(int[] capacity, int[] rocks, int additionalRocks) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def maximumBags(self, capacity: List[int], rocks: List[int], additionalRocks:  
        int) -> int:
```

Python:

```
class Solution(object):  
    def maximumBags(self, capacity, rocks, additionalRocks):  
        """  
        :type capacity: List[int]  
        :type rocks: List[int]  
        :type additionalRocks: int  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[]} capacity  
 * @param {number[]} rocks  
 * @param {number} additionalRocks  
 * @return {number}  
 */  
var maximumBags = function(capacity, rocks, additionalRocks) {  
  
};
```

TypeScript:

```
function maximumBags(capacity: number[], rocks: number[], additionalRocks:  
number): number {  
  
};
```

C#:

```
public class Solution {  
public int MaximumBags(int[] capacity, int[] rocks, int additionalRocks) {  
  
}  
}
```

C:

```
int maximumBags(int* capacity, int capacitySize, int* rocks, int rocksSize,  
int additionalRocks) {  
  
}
```

Go:

```
func maximumBags(capacity []int, rocks []int, additionalRocks int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun maximumBags(capacity: IntArray, rocks: IntArray, additionalRocks: Int):  
        Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maximumBags(_ capacity: [Int], _ rocks: [Int], _ additionalRocks: Int)  
        -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn maximum_bags(capacity: Vec<i32>, rocks: Vec<i32>, additional_rocks:  
        i32) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} capacity  
# @param {Integer[]} rocks  
# @param {Integer} additional_rocks  
# @return {Integer}  
def maximum_bags(capacity, rocks, additional_rocks)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $capacity  
     * @param Integer[] $rocks  
     * @param Integer $additionalRocks
```

```

    * @return Integer
    */
function maximumBags($capacity, $rocks, $additionalRocks) {

}
}

```

Dart:

```

class Solution {
int maximumBags(List<int> capacity, List<int> rocks, int additionalRocks) {

}
}

```

Scala:

```

object Solution {
def maximumBags(capacity: Array[Int], rocks: Array[Int], additionalRocks:
Int): Int = {

}
}

```

Elixir:

```

defmodule Solution do
@spec maximum_bags(capacity :: [integer], rocks :: [integer],
additional_rocks :: integer) :: integer
def maximum_bags(capacity, rocks, additional_rocks) do

end
end

```

Erlang:

```

-spec maximum_bags(Capacity :: [integer()], Rocks :: [integer()],
AdditionalRocks :: integer()) -> integer().
maximum_bags(Capacity, Rocks, AdditionalRocks) ->
.

```

Racket:

```
(define/contract (maximum-bags capacity rocks additionalRocks)
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?
    exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Maximum Bags With Full Capacity of Rocks
 * Difficulty: Medium
 * 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 maximumBags(vector<int>& capacity, vector<int>& rocks, int
additionalRocks) {

    }
};
```

Java Solution:

```
/**
 * Problem: Maximum Bags With Full Capacity of Rocks
 * Difficulty: Medium
 * 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 maximumBags(int[] capacity, int[] rocks, int additionalRocks) {
```

```
}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Bags With Full Capacity of Rocks
Difficulty: Medium
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 maximumBags(self, capacity: List[int], rocks: List[int], additionalRocks: int) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```
class Solution(object):

    def maximumBags(self, capacity, rocks, additionalRocks):
        """
        :type capacity: List[int]
        :type rocks: List[int]
        :type additionalRocks: int
        :rtype: int
        """
```

JavaScript Solution:

```
/**
 * Problem: Maximum Bags With Full Capacity of Rocks
 * Difficulty: Medium
 * 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[]} capacity
* @param {number[]} rocks
* @param {number} additionalRocks
* @return {number}
*/
var maximumBags = function(capacity, rocks, additionalRocks) {

```

```

};

```

TypeScript Solution:

```

/***
* Problem: Maximum Bags With Full Capacity of Rocks
* Difficulty: Medium
* 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 maximumBags(capacity: number[], rocks: number[], additionalRocks: number): number {

```

```

};

```

C# Solution:

```

/*
* Problem: Maximum Bags With Full Capacity of Rocks
* Difficulty: Medium
* 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 MaximumBags(int[] capacity, int[] rocks, int additionalRocks) {

}
}

```

C Solution:

```

/*
 * Problem: Maximum Bags With Full Capacity of Rocks
 * Difficulty: Medium
 * 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 maximumBags(int* capacity, int capacitySize, int* rocks, int rocksSize,
int additionalRocks) {

}

```

Go Solution:

```

// Problem: Maximum Bags With Full Capacity of Rocks
// Difficulty: Medium
// 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 maximumBags(capacity []int, rocks []int, additionalRocks int) int {
}

```

Kotlin Solution:

```

class Solution {
    fun maximumBags(capacity: IntArray, rocks: IntArray, additionalRocks: Int): Int {
        }
    }
}

```

Swift Solution:

```

class Solution {
    func maximumBags(_ capacity: [Int], _ rocks: [Int], _ additionalRocks: Int) -> Int {
        }
    }
}

```

Rust Solution:

```

// Problem: Maximum Bags With Full Capacity of Rocks
// Difficulty: Medium
// 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 maximum_bags(capacity: Vec<i32>, rocks: Vec<i32>, additional_rocks: i32) -> i32 {
        }
    }
}

```

Ruby Solution:

```

# @param {Integer[]} capacity
# @param {Integer[]} rocks
# @param {Integer} additional_rocks
# @return {Integer}
def maximum_bags(capacity, rocks, additional_rocks)

end

```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $capacity
     * @param Integer[] $rocks
     * @param Integer $additionalRocks
     * @return Integer
     */
    function maximumBags($capacity, $rocks, $additionalRocks) {

    }
}
```

Dart Solution:

```
class Solution {
  int maximumBags(List<int> capacity, List<int> rocks, int additionalRocks) {
    }
}
```

Scala Solution:

```
object Solution {
  def maximumBags(capacity: Array[Int], rocks: Array[Int], additionalRocks: Int): Int = {
    }
}
```

Elixir Solution:

```
defmodule Solution do
  @spec maximum_bags(integer :: [integer], integer :: [integer],
                     integer :: integer) :: integer
  def maximum_bags(capacity, rocks, additional_rocks) do
    end
  end
```

Erlang Solution:

```
-spec maximum_bags(Capacity :: [integer()], Rocks :: [integer()]),
AdditionalRocks :: integer()) -> integer().
maximum_bags(Capacity, Rocks, AdditionalRocks) ->
.
```

Racket Solution:

```
(define/contract (maximum-bags capacity rocks additionalRocks)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?))
)
```