Problem 3647: Maximum Weight in Two Bags

Problem Information

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

Problem Description

You are given an integer array

weights

and two integers

w1

and

w2

representing the

maximum

capacities of two bags.

Each item may be placed in

at most

one bag such that:

Bag 1 holds

at most

w1

total weight.

Bag 2 holds

at most

w2

total weight.

Return the

maximum

total weight that can be packed into the two bags.

Example 1:

Input:

weights = [1,4,3,2], w1 = 5, w2 = 4

Output:

9

Explanation:

Bag 1: Place

weights[2] = 3

and

weights[3] = 2

as

3 + 2 = 5 <= w1

Bag 2: Place

weights[1] = 4

as

4 <= w2

Total weight:

5 + 4 = 9

Example 2:

Input:

weights = [3,6,4,8], w1 = 9, w2 = 7

Output:

15

Explanation:

Bag 1: Place

weights[3] = 8

as

8 <= w1

Bag 2: Place

weights[0] = 3

and

weights[2] = 4

as

3 + 4 = 7 <= w2

Total weight:

8 + 7 = 15

Example 3:

Input:

weights = [5,7], w1 = 2, w2 = 3

Output:

0

Explanation:

No weight fits in either bag, thus the answer is 0.

Constraints:

1 <= weights.length <= 100

1 <= weights[i] <= 100

1 <= w1, w2 <= 300

Code Snippets

C++:

```
class Solution {
public:
int maxWeight(vector<int>& weights, int w1, int w2) {

}
};
```

Java:

```
class Solution {
public int maxWeight(int[] weights, int w1, int w2) {

}
}
```

Python3:

```
class Solution:
def maxWeight(self, weights: List[int], w1: int, w2: int) -> int:
```

Python:

```
class Solution(object):
def maxWeight(self, weights, w1, w2):
"""
:type weights: List[int]
:type w1: int
:type w2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} weights
* @param {number} w1
* @param {number} w2
* @return {number}
*/
var maxWeight = function(weights, w1, w2) {
```

```
};
```

TypeScript:

```
function maxWeight(weights: number[], w1: number, w2: number): number {

};
```

C#:

```
public class Solution {
public int MaxWeight(int[] weights, int w1, int w2) {

}
}
```

C:

```
int maxWeight(int* weights, int weightsSize, int w1, int w2) {

}
```

Go:

```
func maxWeight(weights []int, w1 int, w2 int) int {

}
```

Kotlin:

```
class Solution {
fun maxWeight(weights: IntArray, w1: Int, w2: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxWeight(_ weights: [Int], _ w1: Int, _ w2: Int) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn max_weight(weights: Vec<i32>, w1: i32, w2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} weights
# @param {Integer} w1
# @param {Integer} w2
# @return {Integer}
def max_weight(weights, w1, w2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $weights
* @param Integer $w1
* @param Integer $w2
* @return Integer
*/
function maxWeight($weights, $w1, $w2) {

}
}
```

Dart:

```
class Solution {
int maxWeight(List<int> weights, int w1, int w2) {

}
}
```

Scala:

```
object Solution {
def maxWeight(weights: Array[Int], w1: Int, w2: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_weight(weights :: [integer], w1 :: integer, w2 :: integer) ::
integer
def max_weight(weights, w1, w2) do

end
end
```

Erlang:

```
-spec max_weight(Weights :: [integer()], W1 :: integer(), W2 :: integer()) ->
integer().
max_weight(Weights, W1, W2) ->
.
```

Racket:

```
(define/contract (max-weight weights w1 w2)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Weight in Two Bags
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxWeight(vector<int>& weights, int w1, int w2) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Weight in Two Bags
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxWeight(int[] weights, int w1, int w2) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Weight in Two Bags
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""
```

```
class Solution:
def maxWeight(self, weights: List[int], w1: int, w2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxWeight(self, weights, w1, w2):
"""
:type weights: List[int]
:type w1: int
:type w2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Weight in Two Bags
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} weights
* @param {number} w1
* @param {number} w2
* @return {number}
*/
var maxWeight = function(weights, w1, w2) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Weight in Two Bags
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxWeight(weights: number[], w1: number, w2: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Weight in Two Bags
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxWeight(int[] weights, int w1, int w2) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Weight in Two Bags
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

int maxWeight(int* weights, int weightsSize, int w1, int w2) {

}
```

Go Solution:

```
// Problem: Maximum Weight in Two Bags
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxWeight(weights []int, w1 int, w2 int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxWeight(weights: IntArray, w1: Int, w2: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxWeight(_ weights: [Int], _ w1: Int, _ w2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Weight in Two Bags
// Difficulty: Medium
// Tags: array, dp
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_weight(weights: Vec<i32>, w1: i32, w2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} weights
# @param {Integer} w1
# @param {Integer} w2
# @return {Integer}
def max_weight(weights, w1, w2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $weights
* @param Integer $w1
* @param Integer $w2
* @return Integer
*/
function maxWeight($weights, $w1, $w2) {

}
}
```

Dart Solution:

```
class Solution {
int maxWeight(List<int> weights, int w1, int w2) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def maxWeight(weights: Array[Int], w1: Int, w2: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_weight(weights :: [integer], w1 :: integer, w2 :: integer) ::
integer
def max_weight(weights, w1, w2) do

end
end
```

Erlang Solution:

```
-spec max_weight(Weights :: [integer()], W1 :: integer(), W2 :: integer()) ->
integer().
max_weight(Weights, W1, W2) ->
.
```

Racket Solution:

```
(define/contract (max-weight weights w1 w2)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```