

Problem 1049: Last Stone Weight II

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an array of integers

stones

where

stones[i]

is the weight of the

i

th

stone.

We are playing a game with the stones. On each turn, we choose any two stones and smash them together. Suppose the stones have weights

x

and

y

with

$x \leq y$

. The result of this smash is:

If

$x == y$

, both stones are destroyed, and

If

$x \neq y$

, the stone of weight

x

is destroyed, and the stone of weight

y

has new weight

$y - x$

.

At the end of the game, there is

at most one

stone left.

Return

the smallest possible weight of the left stone

. If there are no stones left, return

0

.

Example 1:

Input:

stones = [2,7,4,1,8,1]

Output:

1

Explanation:

We can combine 2 and 4 to get 2, so the array converts to [2,7,1,8,1] then, we can combine 7 and 8 to get 1, so the array converts to [2,1,1,1] then, we can combine 2 and 1 to get 1, so the array converts to [1,1,1] then, we can combine 1 and 1 to get 0, so the array converts to [1], then that's the optimal value.

Example 2:

Input:

stones = [31,26,33,21,40]

Output:

5

Constraints:

$1 \leq \text{stones.length} \leq 30$

$1 \leq \text{stones}[i] \leq 100$

Code Snippets

C++:

```
class Solution {
public:
    int lastStoneWeightII(vector<int>& stones) {

    }
};
```

Java:

```
class Solution {
    public int lastStoneWeightII(int[] stones) {

    }
}
```

Python3:

```
class Solution:
    def lastStoneWeightII(self, stones: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function lastStoneWeightII(stones: number[]): number {  
  
};
```

C#:

```
public class Solution {  
    public int LastStoneWeightII(int[] stones) {  
  
    }  
}
```

C:

```
int lastStoneWeightII(int* stones, int stonesSize) {  
  
}
```

Go:

```
func lastStoneWeightII(stones []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun lastStoneWeightII(stones: IntArray): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func lastStoneWeightII(_ stones: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn last_stone_weight_ii(stones: Vec<i32>) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} stones  
# @return {Integer}  
def last_stone_weight_ii(stones)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $stones  
     * @return Integer  
     */  
    function lastStoneWeightII($stones) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int lastStoneWeightII(List<int> stones) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def lastStoneWeightII(stones: Array[Int]): Int = {  
  
    }  
}
```

```
}
```

Elixir:

```
defmodule Solution do
  @spec last_stone_weight_ii(stones :: [integer]) :: integer
  def last_stone_weight_ii(stones) do

  end
end
```

Erlang:

```
-spec last_stone_weight_ii(Stones :: [integer()]) -> integer().
last_stone_weight_ii(Stones) ->
.
```

Racket:

```
(define/contract (last-stone-weight-ii stones)
  (-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Last Stone Weight II
 * 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 lastStoneWeightII(vector<int>& stones) {
```

```
}  
};
```

Java Solution:

```
/**  
 * Problem: Last Stone Weight II  
 * 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 lastStoneWeightII(int[] stones) {  
  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Last Stone Weight II  
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 lastStoneWeightII(self, stones: List[int]) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

```

class Solution(object):
def lastStoneWeightII(self, stones):
    """
    :type stones: List[int]
    :rtype: int
    """

```

JavaScript Solution:

```

/**
 * Problem: Last Stone Weight II
 * 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[]} stones
 * @return {number}
 */
var lastStoneWeightII = function(stones) {

};

```

TypeScript Solution:

```

/**
 * Problem: Last Stone Weight II
 * 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 lastStoneWeightII(stones: number[]): number {

};

```

C# Solution:

```
/*
 * Problem: Last Stone Weight II
 * 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 LastStoneWeightII(int[] stones) {

    }
}
```

C Solution:

```
/*
 * Problem: Last Stone Weight II
 * 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 lastStoneWeightII(int* stones, int stonesSize) {

}
```

Go Solution:

```
// Problem: Last Stone Weight II
// 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 lastStoneWeightII(stones []int) int {

}
```

Kotlin Solution:

```
class Solution {
    fun lastStoneWeightII(stones: IntArray): Int {

    }
}
```

Swift Solution:

```
class Solution {
    func lastStoneWeightII(_ stones: [Int]) -> Int {

    }
}
```

Rust Solution:

```
// Problem: Last Stone Weight II
// 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 last_stone_weight_ii(stones: Vec<i32>) -> i32 {

    }
}
```

Ruby Solution:

```
# @param {Integer[]} stones
# @return {Integer}
def last_stone_weight_ii(stones)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $stones
     * @return Integer
     */
    function lastStoneWeightII($stones) {

    }

}
```

Dart Solution:

```
class Solution {
  int lastStoneWeightII(List<int> stones) {

  }

}
```

Scala Solution:

```
object Solution {
  def lastStoneWeightII(stones: Array[Int]): Int = {

  }

}
```

Elixir Solution:

```
defmodule Solution do
  @spec last_stone_weight_ii(stones :: [integer]) :: integer
  def last_stone_weight_ii(stones) do

  end
end
```

```
end
```

Erlang Solution:

```
-spec last_stone_weight_ii(Stones :: [integer()]) -> integer().  
last_stone_weight_ii(Stones) ->  
.
```

Racket Solution:

```
(define/contract (last-stone-weight-ii stones)  
  (-> (listof exact-integer?) exact-integer?)  
  )
```