Problem 1563: Stone Game V

Problem Information

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

Problem Description

There are several stones

arranged in a row

, and each stone has an associated value which is an integer given in the array

stoneValue

.

In each round of the game, Alice divides the row into

two non-empty rows

(i.e. left row and right row), then Bob calculates the value of each row which is the sum of the values of all the stones in this row. Bob throws away the row which has the maximum value, and Alice's score increases by the value of the remaining row. If the value of the two rows are equal, Bob lets Alice decide which row will be thrown away. The next round starts with the remaining row.

The game ends when there is only

one stone remaining

. Alice's score is initially

zero

.

Return

the maximum score that Alice can obtain

.

Example 1:

Input:

stoneValue = [6,2,3,4,5,5]

Output:

18

Explanation:

In the first round, Alice divides the row to [6,2,3], [4,5,5]. The left row has the value 11 and the right row has value 14. Bob throws away the right row and Alice's score is now 11. In the second round Alice divides the row to [6], [2,3]. This time Bob throws away the left row and Alice's score becomes 16 (11 + 5). The last round Alice has only one choice to divide the row which is [2], [3]. Bob throws away the right row and Alice's score is now 18 (16 + 2). The game ends because only one stone is remaining in the row.

Example 2:

Input:

stoneValue = [7,7,7,7,7,7,7]

Output:

28

Example 3:

Input:

stoneValue = [4]

Output:

0

Constraints:

1 <= stoneValue.length <= 500

1 <= stoneValue[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int stoneGameV(vector<int>& stoneValue) {

}
};
```

Java:

```
class Solution {
public int stoneGameV(int[] stoneValue) {

}
}
```

Python3:

```
class Solution:
def stoneGameV(self, stoneValue: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function stoneGameV(stoneValue: number[]): number {

};
```

C#:

```
public class Solution {
public int StoneGameV(int[] stoneValue) {

}
}
```

C:

```
int stoneGameV(int* stoneValue, int stoneValueSize) {

}
```

Go:

```
func stoneGameV(stoneValue []int) int {

}
```

Kotlin:

```
class Solution {
fun stoneGameV(stoneValue: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func stoneGameV(_ stoneValue: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn stone_game_v(stone_value: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} stone_value
# @return {Integer}
def stone_game_v(stone_value)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $stoneValue
* @return Integer
```

```
*/
function stoneGameV($stoneValue) {

}
}
```

Dart:

```
class Solution {
int stoneGameV(List<int> stoneValue) {

}
}
```

Scala:

```
object Solution {
def stoneGameV(stoneValue: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec stone_game_v(stone_value :: [integer]) :: integer
def stone_game_v(stone_value) do

end
end
```

Erlang:

```
-spec stone_game_v(StoneValue :: [integer()]) -> integer().
stone_game_v(StoneValue) ->
.
```

Racket:

```
(define/contract (stone-game-v stoneValue)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Stone Game V
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 stoneGameV(vector<int>& stoneValue) {

}
};
```

Java Solution:

```
/**
* Problem: Stone Game V
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 stoneGameV(int[] stoneValue) {

}
}
```

Python3 Solution:

```
"""
Problem: Stone Game V
Difficulty: Hard
Tags: array, dp, math

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 stoneGameV(self, stoneValue: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Stone Game V
* Difficulty: Hard
* Tags: array, dp, math
*
* 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[]} stoneValue
* @return {number}
*/
var stoneGameV = function(stoneValue) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Stone Game V
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 stoneGameV(stoneValue: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Stone Game V
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 StoneGameV(int[] stoneValue) {

}
}
```

C Solution:

```
/*
* Problem: Stone Game V
* Difficulty: Hard
```

```
* Tags: array, dp, math
*
* 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 stoneGameV(int* stoneValue, int stoneValueSize) {

}
```

Go Solution:

```
// Problem: Stone Game V
// Difficulty: Hard
// Tags: array, dp, math
//
// 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 stoneGameV(stoneValue []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun stoneGameV(stoneValue: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func stoneGameV(_ stoneValue: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Stone Game V
// Difficulty: Hard
// Tags: array, dp, math
//
// 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 stone_game_v(stone_value: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} stone_value
# @return {Integer}
def stone_game_v(stone_value)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $stoneValue
* @return Integer
*/
function stoneGameV($stoneValue) {

}
}
```

Dart Solution:

```
class Solution {
int stoneGameV(List<int> stoneValue) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def stoneGameV(stoneValue: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec stone_game_v(stone_value :: [integer]) :: integer
def stone_game_v(stone_value) do

end
end
```

Erlang Solution:

```
-spec stone_game_v(StoneValue :: [integer()]) -> integer().
stone_game_v(StoneValue) ->
.
```

Racket Solution:

```
(define/contract (stone-game-v stoneValue)
(-> (listof exact-integer?) exact-integer?)
)
```