Problem 1872: Stone Game VIII

Problem Information

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

Problem Description

Alice and Bob take turns playing a game, with

Alice starting first

.

There are

n

stones arranged in a row. On each player's turn, while the number of stones is

more than one

, they will do the following:

Choose an integer

x > 1

, and

remove

the leftmost

x

stones from the row.

Add the

sum

of the

removed

stones' values to the player's score.

Place a

new stone

, whose value is equal to that sum, on the left side of the row.

The game stops when

only

one

stone is left in the row.

The

score difference

between Alice and Bob is

(Alice's score - Bob's score)

. Alice's goal is to

maximize

the score difference, and Bob's goal is the

minimize

the score difference.

Given an integer array

stones

of length

n

where

stones[i]

represents the value of the

i

th

stone

from the left

, return

the

score difference

between Alice and Bob if they both play

optimally

.

Example 1:

Input:

stones = [-1,2,-3,4,-5]

Output:

5

Explanation:

- Alice removes the first 4 stones, adds (-1) + 2 + (-3) + 4 = 2 to her score, and places a stone of value 2 on the left. stones = [2,-5]. - Bob removes the first 2 stones, adds 2 + (-5) = -3 to his score, and places a stone of value -3 on the left. stones = [-3]. The difference between their scores is 2 - (-3) = 5.

Example 2:

Input:

stones = [7,-6,5,10,5,-2,-6]

Output:

13

Explanation:

- Alice removes all stones, adds 7 + (-6) + 5 + 10 + 5 + (-2) + (-6) = 13 to her score, and places a stone of value 13 on the left. stones = [13]. The difference between their scores is 13 - 0 = 13.

Example 3:

Input:

stones = [-10,-12]

Output:

-22

Explanation:

- Alice can only make one move, which is to remove both stones. She adds (-10) + (-12) = -22 to her score and places a stone of value -22 on the left. stones = [-22]. The difference between their scores is (-22) - 0 = -22.

Constraints:

n == stones.length

2 <= n <= 10

5

-10

4

<= stones[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function stoneGameVIII(stones: number[]): number {

};
```

C#:

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

```
}
}
```

C:

```
int stoneGameVIII(int* stones, int stonesSize){

}
```

Go:

```
func stoneGameVIII(stones []int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Scala:

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

}
}
```

Racket:

```
(define/contract (stone-game-viii stones)
(-> (listof exact-integer?) exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Stone Game VIII
* 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 stoneGameVIII(vector<int>& stones) {

}
};
```

Java Solution:

```
/**
* Problem: Stone Game VIII
* 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 stoneGameVIII(int[] stones) {

}
}
```

Python3 Solution:

```
"""
Problem: Stone Game VIII
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 stoneGameVIII(self, stones: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Stone Game VIII
* 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[]} stones
* @return {number}
*/
var stoneGameVIII = function(stones) {

};
```

TypeScript Solution:

```
/**
* Problem: Stone Game VIII
```

```
* 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 stoneGameVIII(stones: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Stone Game VIII
* 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 StoneGameVIII(int[] stones) {

}
}
```

C Solution:

```
/*
* Problem: Stone Game VIII
* 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 stoneGameVIII(int* stones, int stonesSize){

}
```

Go Solution:

```
// Problem: Stone Game VIII
// 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 stoneGameVIII(stones []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Stone Game VIII
// 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_viii(stones: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```

Racket Solution:

```
(define/contract (stone-game-viii stones)
(-> (listof exact-integer?) exact-integer?)

)
```