Problem 1690: Stone Game VII

Problem Information

Difficulty: Medium
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, they can

remove

either the leftmost stone or the rightmost stone from the row and receive points equal to the

sum

of the remaining stones' values in the row. The winner is the one with the higher score when there are no stones left to remove.

Bob found that he will always lose this game (poor Bob, he always loses), so he decided to

minimize the score's difference

. Alice's goal is to

maximize the difference

in the score.

Given an array of integers

stones

where

stones[i]

represents the value of the

i

th

stone

from the left

, return

the

difference

in Alice and Bob's score if they both play

optimally

.

Example 1:

Input:

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

Output:

6

Explanation:

- Alice removes 2 and gets 5 + 3 + 1 + 4 = 13 points. Alice = 13, Bob = 0, stones = [5,3,1,4]. - Bob removes 5 and gets 3 + 1 + 4 = 8 points. Alice = 13, Bob = 8, stones = [3,1,4]. - Alice removes 3 and gets 1 + 4 = 5 points. Alice = 18, Bob = 8, stones = [1,4]. - Bob removes 1 and gets 4 points. Alice = 18, Bob = 12, stones = [4]. - Alice removes 4 and gets 0 points. Alice = 18, Bob = 12, stones = []. The score difference is 18 - 12 = 6.

Example 2:

Input:

stones = [7,90,5,1,100,10,10,2]

Output:

122

Constraints:

n == stones.length

2 <= n <= 1000

1 <= stones[i] <= 1000

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```


int stoneGameVII(int* stones, int stonesSize){

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Scala:

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

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Stone Game VII
* Difficulty: Medium
* 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 stoneGameVII(vector<int>& stones) {

}
};
```

Java Solution:

```
/**
* Problem: Stone Game VII
* Difficulty: Medium
* 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 stoneGameVII(int[] stones) {

}
}
```

Python3 Solution:

```
"""
Problem: Stone Game VII
Difficulty: Medium
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 stoneGameVII(self, stones: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Stone Game VII
* Difficulty: Medium
* 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 stoneGameVII = function(stones) {

};
```

TypeScript Solution:

```
/**
* Problem: Stone Game VII
* Difficulty: Medium
```

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

};
```

C# Solution:

```
/*
* Problem: Stone Game VII
* Difficulty: Medium
* 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 StoneGameVII(int[] stones) {

}
}
```

C Solution:

```
/*
* Problem: Stone Game VII
* Difficulty: Medium
* 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 stoneGameVII(int* stones, int stonesSize){

}
```

Go Solution:

```
// Problem: Stone Game VII
// Difficulty: Medium
// 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 stoneGameVII(stones []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Stone Game VII
// Difficulty: Medium
// 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_vii(stones: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```