Problem 1406: Stone Game III

Problem Information

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

Problem Description

Alice and Bob continue their games with piles of stones. There are several stones

arranged in a row

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

stoneValue

.

Alice and Bob take turns, with Alice starting first. On each player's turn, that player can take

1

,

2

, or

3

stones from the

first

remaining stones in the row.

The score of each player is the sum of the values of the stones taken. The score of each player is

0

initially.

The objective of the game is to end with the highest score, and the winner is the player with the highest score and there could be a tie. The game continues until all the stones have been taken.

Assume Alice and Bob

play optimally

.

Return

"Alice"

if Alice will win,

"Bob"

if Bob will win, or

"Tie"

if they will end the game with the same score

.

Example 1:

Input:

stoneValue = [1,2,3,7]

Output:

"Bob"

Explanation:

Alice will always lose. Her best move will be to take three piles and the score become 6. Now the score of Bob is 7 and Bob wins.

Example 2:

Input:

stoneValue = [1,2,3,-9]

Output:

"Alice"

Explanation:

Alice must choose all the three piles at the first move to win and leave Bob with negative score. If Alice chooses one pile her score will be 1 and the next move Bob's score becomes 5. In the next move, Alice will take the pile with value = -9 and lose. If Alice chooses two piles her score will be 3 and the next move Bob's score becomes 3. In the next move, Alice will take the pile with value = -9 and also lose. Remember that both play optimally so here Alice will choose the scenario that makes her win.

Example 3:

Input:

stoneValue = [1,2,3,6]

Output:

"Tie"

Explanation:

Alice cannot win this game. She can end the game in a draw if she decided to choose all the first three piles, otherwise she will lose.

Constraints:

1 <= stoneValue.length <= 5 * 10

4

-1000 <= stoneValue[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public String stoneGameIII(int[] stoneValue) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function stoneGameIII(stoneValue: number[]): string {

};
```

C#:

```
public class Solution {
public string StoneGameIII(int[] stoneValue) {

}
}
```

C:

```
char* stoneGameIII(int* stoneValue, int stoneValueSize) {

}
```

Go:

```
func stoneGameIII(stoneValue []int) string {

}
```

Kotlin:

```
class Solution {
fun stoneGameIII(stoneValue: IntArray): String {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String stoneGameIII(List<int> stoneValue) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec stone_game_iii(stone_value :: [integer]) :: String.t
def stone_game_iii(stone_value) do

end
end
```

Erlang:

```
-spec stone_game_iii(StoneValue :: [integer()]) -> unicode:unicode_binary().
stone_game_iii(StoneValue) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Stone Game III
* 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:
string stoneGameIII(vector<int>& stoneValue) {

}
};
```

Java Solution:

```
/**
* Problem: Stone Game III
* 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 String stoneGameIII(int[] stoneValue) {

}
}
```

Python3 Solution:

```
"""
Problem: Stone Game III
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 stoneGameIII(self, stoneValue: List[int]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Stone Game III
* 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 {string}
*/
var stoneGameIII = function(stoneValue) {

};
```

TypeScript Solution:

```
/**
* Problem: Stone Game III
* 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 stoneGameIII(stoneValue: number[]): string {

};
```

C# Solution:

```
/*
* Problem: Stone Game III
* 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 string StoneGameIII(int[] stoneValue) {

}
}
```

C Solution:

```
/*
* Problem: Stone Game III
* 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
```

```
*/

char* stoneGameIII(int* stoneValue, int stoneValueSize) {

}
```

Go Solution:

```
// Problem: Stone Game III
// 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 stoneGameIII(stoneValue []int) string {

}
```

Kotlin Solution:

```
class Solution {
fun stoneGameIII(stoneValue: IntArray): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Stone Game III
// 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_iii(stone_value: Vec<i32>) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
String stoneGameIII(List<int> stoneValue) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec stone_game_iii(stone_value :: [integer]) :: String.t
def stone_game_iii(stone_value) do

end
end
```

Erlang Solution:

```
-spec stone_game_iii(StoneValue :: [integer()]) -> unicode:unicode_binary().
stone_game_iii(StoneValue) ->
.
```

Racket Solution:

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