Problem 1686: Stone Game VI

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 in a pile. On each player's turn, they can

remove

a stone from the pile and receive points based on the stone's value. Alice and Bob may

value the stones differently

.

You are given two integer arrays of length

n

,

aliceValues

and

bobValues

. Each

aliceValues[i]

and

bobValues[i]

represents how Alice and Bob, respectively, value the

i

th

stone.

The winner is the person with the most points after all the stones are chosen. If both players have the same amount of points, the game results in a draw. Both players will play

optimally

. Both players know the other's values.

Determine the result of the game, and:

If Alice wins, return

1

.

If Bob wins, return

-1

.

If the game results in a draw, return

0

.

Example 1:

Input:

aliceValues = [1,3], bobValues = [2,1]

Output:

1

Explanation:

If Alice takes stone 1 (0-indexed) first, Alice will receive 3 points. Bob can only choose stone 0, and will only receive 2 points. Alice wins.

Example 2:

Input:

aliceValues = [1,2], bobValues = [3,1]

Output:

0

Explanation:

If Alice takes stone 0, and Bob takes stone 1, they will both have 1 point. Draw.

Example 3:

Input:

aliceValues = [2,4,3], bobValues = [1,6,7]

Output:

-1

Explanation:

Regardless of how Alice plays, Bob will be able to have more points than Alice. For example, if Alice takes stone 1, Bob can take stone 2, and Alice takes stone 0, Alice will have 6 points to Bob's 7. Bob wins.

Constraints:

n == aliceValues.length == bobValues.length

1 <= n <= 10

5

1 <= aliceValues[i], bobValues[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int stoneGameVI(vector<int>& aliceValues, vector<int>& bobValues) {

}
};
```

Java:

```
class Solution {
public int stoneGameVI(int[] aliceValues, int[] bobValues) {

}
```

```
}
```

Python3:

```
class Solution:
def stoneGameVI(self, aliceValues: List[int], bobValues: List[int]) -> int:
```

Python:

```
class Solution(object):
def stoneGameVI(self, aliceValues, bobValues):
"""
:type aliceValues: List[int]
:type bobValues: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} aliceValues
* @param {number[]} bobValues
* @return {number}
*/
var stoneGameVI = function(aliceValues, bobValues) {

};
```

TypeScript:

```
function stoneGameVI(aliceValues: number[], bobValues: number[]): number {

};
```

C#:

```
public class Solution {
public int StoneGameVI(int[] aliceValues, int[] bobValues) {

}
}
```

C:

```
int stoneGameVI(int* aliceValues, int aliceValuesSize, int* bobValues, int
bobValuesSize) {

}
```

Go:

```
func stoneGameVI(aliceValues []int, bobValues []int) int {

}
```

Kotlin:

```
class Solution {
fun stoneGameVI(aliceValues: IntArray, bobValues: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func stoneGameVI(_ aliceValues: [Int], _ bobValues: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn stone_game_vi(alice_values: Vec<i32>, bob_values: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} alice_values
# @param {Integer[]} bob_values
# @return {Integer}
def stone_game_vi(alice_values, bob_values)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $aliceValues
* @param Integer[] $bobValues
* @return Integer
*/
function stoneGameVI($aliceValues, $bobValues) {

}
}
```

Dart:

```
class Solution {
int stoneGameVI(List<int> aliceValues, List<int> bobValues) {

}
}
```

Scala:

```
object Solution {
def stoneGameVI(aliceValues: Array[Int], bobValues: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec stone_game_vi(alice_values :: [integer], bob_values :: [integer]) ::
integer
def stone_game_vi(alice_values, bob_values) do

end
end
```

Erlang:

```
-spec stone_game_vi(AliceValues :: [integer()], BobValues :: [integer()]) ->
integer().
stone_game_vi(AliceValues, BobValues) ->
.
```

Racket:

```
(define/contract (stone-game-vi aliceValues bobValues)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Stone Game VI
* Difficulty: Medium
* Tags: array, greedy, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int stoneGameVI(vector<int>& aliceValues, vector<int>& bobValues) {

}
};
```

Java Solution:

```
/**
* Problem: Stone Game VI
* Difficulty: Medium
* Tags: array, greedy, math, sort, queue, heap
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int stoneGameVI(int[] aliceValues, int[] bobValues) {

}
}
```

Python3 Solution:

```
"""
Problem: Stone Game VI
Difficulty: Medium
Tags: array, greedy, math, sort, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def stoneGameVI(self, aliceValues: List[int], bobValues: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def stoneGameVI(self, aliceValues, bobValues):
"""
:type aliceValues: List[int]
:type bobValues: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Stone Game VI
* Difficulty: Medium
* Tags: array, greedy, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} aliceValues
* @param {number[]} bobValues
* @return {number}
*/
var stoneGameVI = function(aliceValues, bobValues) {

};
```

TypeScript Solution:

```
/**
* Problem: Stone Game VI
* Difficulty: Medium
* Tags: array, greedy, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function stoneGameVI(aliceValues: number[], bobValues: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Stone Game VI
* Difficulty: Medium
* Tags: array, greedy, math, sort, queue, heap
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int StoneGameVI(int[] aliceValues, int[] bobValues) {

}
}
```

C Solution:

```
/*
* Problem: Stone Game VI
* Difficulty: Medium
* Tags: array, greedy, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int stoneGameVI(int* aliceValues, int aliceValuesSize, int* bobValues, int
bobValuesSize) {

}
```

Go Solution:

```
// Problem: Stone Game VI
// Difficulty: Medium
// Tags: array, greedy, math, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func stoneGameVI(aliceValues []int, bobValues []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun stoneGameVI(aliceValues: IntArray, bobValues: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func stoneGameVI(_ aliceValues: [Int], _ bobValues: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Stone Game VI
// Difficulty: Medium
// Tags: array, greedy, math, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn stone_game_vi(alice_values: Vec<i32>, bob_values: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} alice_values
# @param {Integer[]} bob_values
# @return {Integer}
def stone_game_vi(alice_values, bob_values)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $aliceValues
* @param Integer[] $bobValues
* @return Integer
*/
function stoneGameVI($aliceValues, $bobValues) {

}
}
```

Dart Solution:

```
class Solution {
int stoneGameVI(List<int> aliceValues, List<int> bobValues) {

}
}
```

Scala Solution:

```
object Solution {
def stoneGameVI(aliceValues: Array[Int], bobValues: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec stone_game_vi(alice_values :: [integer], bob_values :: [integer]) ::
integer
def stone_game_vi(alice_values, bob_values) do

end
end
```

Erlang Solution:

```
-spec stone_game_vi(AliceValues :: [integer()], BobValues :: [integer()]) ->
integer().
stone_game_vi(AliceValues, BobValues) ->
.
```

Racket Solution:

```
(define/contract (stone-game-vi aliceValues bobValues)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```