Problem 1040: Moving Stones Until Consecutive II

Problem Information

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

Problem Description

There are some stones in different positions on the X-axis. You are given an integer array

stones

, the positions of the stones.

Call a stone an

endpoint stone

if it has the smallest or largest position. In one move, you pick up an

endpoint stone

and move it to an unoccupied position so that it is no longer an

endpoint stone

.

In particular, if the stones are at say,

stones = [1,2,5]

, you cannot move the endpoint stone at position

5

, since moving it to any position (such as

0

, or

3

) will still keep that stone as an endpoint stone.

The game ends when you cannot make any more moves (i.e., the stones are in three consecutive positions).

Return

an integer array

answer

of length

2

where

:

answer[0]

is the minimum number of moves you can play, and

answer[1]

is the maximum number of moves you can play

.

Example 1:

Input:

stones = [7,4,9]

Output:

[1,2]

Explanation:

We can move 4 -> 8 for one move to finish the game. Or, we can move 9 -> 5, 4 -> 6 for two moves to finish the game.

Example 2:

Input:

stones = [6,5,4,3,10]

Output:

[2,3]

Explanation:

We can move 3 -> 8 then 10 -> 7 to finish the game. Or, we can move 3 -> 7, 4 -> 8, 5 -> 9 to finish the game. Notice we cannot move 10 -> 2 to finish the game, because that would be an illegal move.

Constraints:

3 <= stones.length <= 10

4

1 <= stones[i] <= 10

9

All the values of

stones

are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def numMovesStonesII(self, stones):
```

```
"""
:type stones: List[int]
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numMovesStonesII(int* stones, int stonesSize, int* returnSize) {

}
```

Go:

```
func numMovesStonesII(stones []int) []int {
}
```

Kotlin:

```
class Solution {
fun numMovesStonesII(stones: IntArray): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> numMovesStonesII(List<int> stones) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec num_moves_stones_ii(stones :: [integer]) :: [integer]
def num_moves_stones_ii(stones) do

end
end
```

Erlang:

```
-spec num_moves_stones_ii(Stones :: [integer()]) -> [integer()].
num_moves_stones_ii(Stones) ->
.
```

Racket:

```
(define/contract (num-moves-stones-ii stones)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Moving Stones Until Consecutive II
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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:
vector<int> numMovesStonesII(vector<int>& stones) {

}
};
```

Java Solution:

```
/**
* Problem: Moving Stones Until Consecutive II
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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[] numMovesStonesII(int[] stones) {

}
}
```

Python3 Solution:

```
"""
Problem: Moving Stones Until Consecutive II
Difficulty: Medium
Tags: array, dp, math, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Moving Stones Until Consecutive II
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 numMovesStonesII = function(stones) {
};
```

TypeScript Solution:

```
/**
* Problem: Moving Stones Until Consecutive II
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 numMovesStonesII(stones: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Moving Stones Until Consecutive II
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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[] NumMovesStonesII(int[] stones) {

}
}
```

C Solution:

```
/*
* Problem: Moving Stones Until Consecutive II
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numMovesStonesII(int* stones, int stonesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Moving Stones Until Consecutive II
// Difficulty: Medium
// Tags: array, dp, math, sort
//
// 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 numMovesStonesII(stones []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun numMovesStonesII(stones: IntArray): IntArray {

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Moving Stones Until Consecutive II
// Difficulty: Medium
// Tags: array, dp, math, sort
//
// 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 num_moves_stones_ii(stones: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> numMovesStonesII(List<int> stones) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_moves_stones_ii(stones :: [integer]) :: [integer]
def num_moves_stones_ii(stones) do

end
end
```

Erlang Solution:

```
-spec num_moves_stones_ii(Stones :: [integer()]) -> [integer()].
num_moves_stones_ii(Stones) ->
.
```

Racket Solution:

```
(define/contract (num-moves-stones-ii stones)
(-> (listof exact-integer?) (listof exact-integer?))
)
```