# Problem 1144: Decrease Elements To Make Array Zigzag

## Problem Information

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

## Problem Description

Given an array

nums

of integers, a

move

consists of choosing any element and

decreasing it by 1

.

An array

A

is a

zigzag array

if either:

Every even-indexed element is greater than adjacent elements, ie.

A[0] > A[1] < A[2] > A[3] < A[4] > ...

OR, every odd-indexed element is greater than adjacent elements, ie.

A[0] < A[1] > A[2] < A[3] > A[4] < ...

Return the minimum number of moves to transform the given array

nums

into a zigzag array.

Example 1:

Input:

nums = [1,2,3]

Output:

2

Explanation:

We can decrease 2 to 0 or 3 to 1.

Example 2:

Input:

nums = [9,6,1,6,2]

Output:

4

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 1000

## Code Snippets

### C++:

```
class Solution {
public:
int movesToMakeZigzag(vector<int>& nums) {

}
};
```

### Java:

```
class Solution {
public int movesToMakeZigzag(int[] nums) {

}
}
```

### Python3:

```
class Solution:
def movesToMakeZigzag(self, nums: List[int]) -> int:
```

### Python:

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

### JavaScript:

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

};
```

**TypeScript:**

```
function movesToMakeZigzag(nums: number[]): number {

};
```

**C#:**

```
public class Solution {
public int MovesToMakeZigzag(int[] nums) {

}
}
```

**C:**

```
int movesToMakeZigzag(int* nums, int numsSize) {

}
```

**Go:**

```
func movesToMakeZigzag(nums []int) int {

}
```

**Kotlin:**

```
class Solution {
fun movesToMakeZigzag(nums: IntArray): Int {

}
}
```

**Swift:**

```
class Solution {
func movesToMakeZigzag(_ nums: [Int]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn moves_to_make_zigzag(nums: Vec<i32>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @return {Integer}
def moves_to_make_zigzag(nums)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function movesToMakeZigzag($nums) {

}
}
```

**Dart:**

```
class Solution {
int movesToMakeZigzag(List<int> nums) {

}
}
```

### Scala:

```
object Solution {
def movesToMakeZigzag(nums: Array[Int]): Int = {

}
}
```

### Elixir:

```
defmodule Solution do
@spec moves_to_make_zigzag(nums :: [integer]) :: integer
def moves_to_make_zigzag(nums) do

end
end
```

### Erlang:

```
-spec moves_to_make_zigzag(Nums :: [integer()]) -> integer().
moves_to_make_zigzag(Nums) ->
.
```

### Racket:

```
(define/contract (moves-to-make-zigzag nums)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Decrease Elements To Make Array Zigzag
* Difficulty: Medium
* Tags: array, greedy
*
* 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 movesToMakeZigzag(vector<int>& nums) {

}
};
```

**Java Solution:**

```
/**
* Problem: Decrease Elements To Make Array Zigzag
* Difficulty: Medium
* Tags: array, greedy
*
* 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 movesToMakeZigzag(int[] nums) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Decrease Elements To Make Array Zigzag
Difficulty: Medium
Tags: array, greedy

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 movesToMakeZigzag(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Decrease Elements To Make Array Zigzag
* Difficulty: Medium
* Tags: array, greedy
*
* 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[]} nums
* @return {number}
*/
var movesToMakeZigzag = function(nums) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Decrease Elements To Make Array Zigzag
* Difficulty: Medium
* Tags: array, greedy
*
* 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 movesToMakeZigzag(nums: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Decrease Elements To Make Array Zigzag
* Difficulty: Medium
* Tags: array, greedy
*
* 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 MovesToMakeZigzag(int[] nums) {

}
}
```

### C Solution:

```
/*
* Problem: Decrease Elements To Make Array Zigzag
* Difficulty: Medium
* Tags: array, greedy
*
* 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 movesToMakeZigzag(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Decrease Elements To Make Array Zigzag
// Difficulty: Medium
// Tags: array, greedy
//
// 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 movesToMakeZigzag(nums []int) int {

}
```

**Kotlin Solution:**

```
class Solution {
fun movesToMakeZigzag(nums: IntArray): Int {

}
}
```

**Swift Solution:**

```
class Solution {
func movesToMakeZigzag(_ nums: [Int]) -> Int {

}
}
```

**Rust Solution:**

```
// Problem: Decrease Elements To Make Array Zigzag
// Difficulty: Medium
// Tags: array, greedy
//
// 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 moves_to_make_zigzag(nums: Vec<i32>) -> i32 {

}
}
```

**Ruby Solution:**

```
# @param {Integer[]} nums
# @return {Integer}
def moves_to_make_zigzag(nums)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function movesToMakeZigzag($nums) {

}
}
```

**Dart Solution:**

```
class Solution {
int movesToMakeZigzag(List<int> nums) {

}
}
```

**Scala Solution:**

```
object Solution {
def movesToMakeZigzag(nums: Array[Int]): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec moves_to_make_zigzag(nums :: [integer]) :: integer
def moves_to_make_zigzag(nums) do

end
end
```

### Erlang Solution:

```
-spec moves_to_make_zigzag(Nums :: [integer()]) -> integer().
moves_to_make_zigzag(Nums) ->
.
```

### Racket Solution:

```
(define/contract (moves-to-make-zigzag nums)
(-> (listof exact-integer?) exact-integer?)
)
```