

# Problem 3699: Number of ZigZag Arrays I

## Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given three integers

$n$

,

$l$

, and

$r$

.

A

ZigZag

array of length

$n$

is defined as follows:

Each element lies in the range

[l, r]

.

No

two

adjacent elements are equal.

No

three

consecutive elements form a

strictly increasing

or

strictly decreasing

sequence.

Return the total number of valid

ZigZag

arrays.

Since the answer may be large, return it

modulo

10

9

+ 7

.

A

sequence

is said to be

strictly increasing

if each element is strictly greater than its previous one (if exists).

A

sequence

is said to be

strictly decreasing

if each element is strictly smaller than its previous one (if exists).

Example 1:

Input:

$n = 3, l = 4, r = 5$

Output:

2

Explanation:

There are only 2 valid ZigZag arrays of length

$n = 3$

using values in the range

[4, 5]

:

[4, 5, 4]

[5, 4, 5]

Example 2:

Input:

$n = 3, l = 1, r = 3$

Output:

10

Explanation:

There are 10 valid ZigZag arrays of length

$n = 3$

using values in the range

[1, 3]

:

[1, 2, 1]

,

[1, 3, 1]

,

[1, 3, 2]

[2, 1, 2]

,

[2, 1, 3]

,

[2, 3, 1]

,

[2, 3, 2]

[3, 1, 2]

,

[3, 1, 3]

,

[3, 2, 3]

All arrays meet the ZigZag conditions.

Constraints:

$3 \leq n \leq 2000$

$1 \leq l < r \leq 2000$

**Code Snippets**

### C++:

```
class Solution {
public:
    int zigZagArrays(int n, int l, int r) {

    }
};
```

### Java:

```
class Solution {
    public int zigZagArrays(int n, int l, int r) {

    }
}
```

### Python3:

```
class Solution:
    def zigZagArrays(self, n: int, l: int, r: int) -> int:
```

### Python:

```
class Solution(object):
    def zigZagArrays(self, n, l, r):
        """
        :type n: int
        :type l: int
        :type r: int
        :rtype: int
        """
```

### JavaScript:

```
/**
 * @param {number} n
 * @param {number} l
 * @param {number} r
 * @return {number}
 */
var zigZagArrays = function(n, l, r) {
```

```
};
```

### TypeScript:

```
function zigZagArrays(n: number, l: number, r: number): number {  
  
};
```

### C#:

```
public class Solution {  
    public int ZigZagArrays(int n, int l, int r) {  
  
    }  
}
```

### C:

```
int zigZagArrays(int n, int l, int r) {  
  
}
```

### Go:

```
func zigZagArrays(n int, l int, r int) int {  
  
}
```

### Kotlin:

```
class Solution {  
    fun zigZagArrays(n: Int, l: Int, r: Int): Int {  
  
    }  
}
```

### Swift:

```
class Solution {  
    func zigZagArrays(_ n: Int, _ l: Int, _ r: Int) -> Int {  
  
    }  
}
```

```
}
```

### Rust:

```
impl Solution {  
    pub fn zig_zag_arrays(n: i32, l: i32, r: i32) -> i32 {  
  
    }  
}
```

### Ruby:

```
# @param {Integer} n  
# @param {Integer} l  
# @param {Integer} r  
# @return {Integer}  
def zig_zag_arrays(n, l, r)  
  
end
```

### PHP:

```
class Solution {  
  
    /**  
     * @param Integer $n  
     * @param Integer $l  
     * @param Integer $r  
     * @return Integer  
     */  
    function zigZagArrays($n, $l, $r) {  
  
    }  
}
```

### Dart:

```
class Solution {  
    int zigZagArrays(int n, int l, int r) {  
  
    }  
}
```

### Scala:

```
object Solution {  
  def zigZagArrays(n: Int, l: Int, r: Int): Int = {  
  
  }  
}
```

### Elixir:

```
defmodule Solution do  
  @spec zig_zag_arrays(n :: integer, l :: integer, r :: integer) :: integer  
  def zig_zag_arrays(n, l, r) do  
  
  end  
end
```

### Erlang:

```
-spec zig_zag_arrays(N :: integer(), L :: integer(), R :: integer()) ->  
integer().  
zig_zag_arrays(N, L, R) ->  
.
```

### Racket:

```
(define/contract (zig-zag-arrays n l r)  
  (-> exact-integer? exact-integer? exact-integer? exact-integer?)  
)
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Number of ZigZag Arrays I  
 * Difficulty: Hard  
 * Tags: array, dp  
 *  
 * 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 zigZagArrays(int n, int l, int r) {

    }

};

```

### Java Solution:

```

/**
 * Problem: Number of ZigZag Arrays I
 * Difficulty: Hard
 * Tags: array, dp
 *
 * 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 zigZagArrays(int n, int l, int r) {

    }

}

```

### Python3 Solution:

```

"""
Problem: Number of ZigZag Arrays I
Difficulty: Hard
Tags: array, dp

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 zigZagArrays(self, n: int, l: int, r: int) -> int:
    # TODO: Implement optimized solution
    pass
```

### Python Solution:

```
class Solution(object):
    def zigZagArrays(self, n, l, r):
        """
        :type n: int
        :type l: int
        :type r: int
        :rtype: int
        """
```

### JavaScript Solution:

```
/**
 * Problem: Number of ZigZag Arrays I
 * Difficulty: Hard
 * Tags: array, dp
 *
 * 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} n
 * @param {number} l
 * @param {number} r
 * @return {number}
 */
var zigZagArrays = function(n, l, r) {

};
```

### TypeScript Solution:

```
/**
 * Problem: Number of ZigZag Arrays I
```

```

* Difficulty: Hard
* Tags: array, dp
*
* 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 zigZagArrays(n: number, l: number, r: number): number {

};

```

### C# Solution:

```

/*
* Problem: Number of ZigZag Arrays I
* Difficulty: Hard
* Tags: array, dp
*
* 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 ZigZagArrays(int n, int l, int r) {

    }
}

```

### C Solution:

```

/*
* Problem: Number of ZigZag Arrays I
* Difficulty: Hard
* Tags: array, dp
*
* 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 zigZagArrays(int n, int l, int r) {  
  
}
```

### Go Solution:

```
// Problem: Number of ZigZag Arrays I  
// Difficulty: Hard  
// Tags: array, dp  
//  
// 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 zigZagArrays(n int, l int, r int) int {  
  
}
```

### Kotlin Solution:

```
class Solution {  
    fun zigZagArrays(n: Int, l: Int, r: Int): Int {  
  
    }  
}
```

### Swift Solution:

```
class Solution {  
    func zigZagArrays(_ n: Int, _ l: Int, _ r: Int) -> Int {  
  
    }  
}
```

### Rust Solution:

```
// Problem: Number of ZigZag Arrays I  
// Difficulty: Hard  
// Tags: array, dp  
//
```

```

// 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 zig_zag_arrays(n: i32, l: i32, r: i32) -> i32 {

}
}

```

### Ruby Solution:

```

# @param {Integer} n
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def zig_zag_arrays(n, l, r)

end

```

### PHP Solution:

```

class Solution {

/**
 * @param Integer $n
 * @param Integer $l
 * @param Integer $r
 * @return Integer
 */
function zigZagArrays($n, $l, $r) {

}

}

```

### Dart Solution:

```

class Solution {
int zigZagArrays(int n, int l, int r) {

}
}

```

```
}
```

### Scala Solution:

```
object Solution {  
  def zigZagArrays(n: Int, l: Int, r: Int): Int = {  
  
  }  
}
```

### Elixir Solution:

```
defmodule Solution do  
  @spec zig_zag_arrays(n :: integer, l :: integer, r :: integer) :: integer  
  def zig_zag_arrays(n, l, r) do  
  
  end  
end
```

### Erlang Solution:

```
-spec zig_zag_arrays(N :: integer(), L :: integer(), R :: integer()) ->  
integer().  
zig_zag_arrays(N, L, R) ->  
.
```

### Racket Solution:

```
(define/contract (zig-zag-arrays n l r)  
  (-> exact-integer? exact-integer? exact-integer? exact-integer?)  
)
```