

Problem 1503: Last Moment Before All Ants Fall Out of a Plank

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

We have a wooden plank of the length

n

units

. Some ants are walking on the plank, each ant moves with a speed of

1 unit per second

. Some of the ants move to the

left

, the other move to the

right

When two ants moving in two

different

directions meet at some point, they change their directions and continue moving again.
Assume changing directions does not take any additional time.

When an ant reaches

one end

of the plank at a time

t

, it falls out of the plank immediately.

Given an integer

n

and two integer arrays

left

and

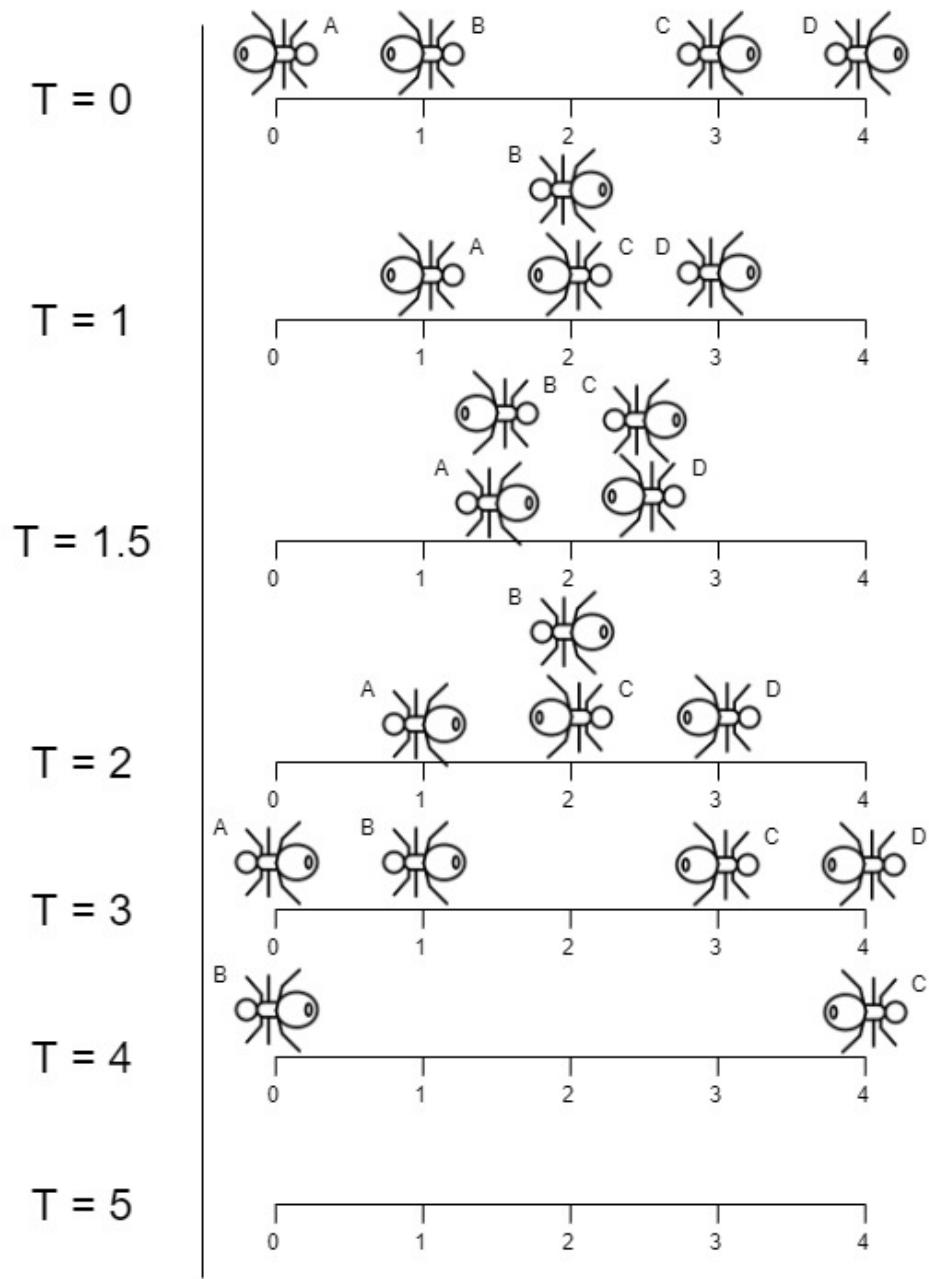
right

, the positions of the ants moving to the left and the right, return

the moment when the last ant(s) fall out of the plank

.

Example 1:



Input:

$n = 4$, left = [4,3], right = [0,1]

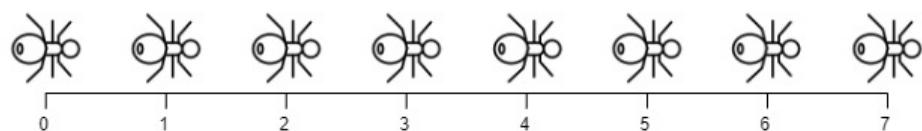
Output:

4

Explanation:

In the image above: -The ant at index 0 is named A and going to the right. -The ant at index 1 is named B and going to the right. -The ant at index 3 is named C and going to the left. -The ant at index 4 is named D and going to the left. The last moment when an ant was on the plank is $t = 4$ seconds. After that, it falls immediately out of the plank. (i.e., We can say that at $t = 4.0000000001$, there are no ants on the plank).

Example 2:



Input:

$n = 7$, $\text{left} = []$, $\text{right} = [0,1,2,3,4,5,6,7]$

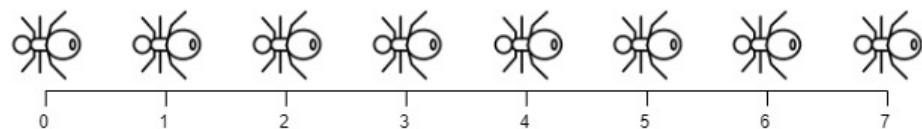
Output:

7

Explanation:

All ants are going to the right, the ant at index 0 needs 7 seconds to fall.

Example 3:



Input:

$n = 7$, $\text{left} = [0,1,2,3,4,5,6,7]$, $\text{right} = []$

Output:

7

Explanation:

All ants are going to the left, the ant at index 7 needs 7 seconds to fall.

Constraints:

$1 \leq n \leq 10$

4

$0 \leq \text{left.length} \leq n + 1$

$0 \leq \text{left}[i] \leq n$

$0 \leq \text{right.length} \leq n + 1$

$0 \leq \text{right}[i] \leq n$

$1 \leq \text{left.length} + \text{right.length} \leq n + 1$

All values of

left

and

right

are unique, and each value can appear

only in one

of the two arrays.

Code Snippets

C++:

```
class Solution {  
public:
```

```
int getLastMoment(int n, vector<int>& left, vector<int>& right) {  
    }  
};
```

Java:

```
class Solution {  
public int getLastMoment(int n, int[] left, int[] right) {  
    }  
}
```

Python3:

```
class Solution:  
    def getLastMoment(self, n: int, left: List[int], right: List[int]) -> int:
```

Python:

```
class Solution(object):  
    def getLastMoment(self, n, left, right):  
        """  
        :type n: int  
        :type left: List[int]  
        :type right: List[int]  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number} n  
 * @param {number[]} left  
 * @param {number[]} right  
 * @return {number}  
 */  
var getLastMoment = function(n, left, right) {  
};
```

TypeScript:

```
function getLastMoment(n: number, left: number[], right: number[]): number {  
    };
```

C#:

```
public class Solution {  
    public int GetLastMoment(int n, int[] left, int[] right) {  
        }  
    }
```

C:

```
int getLastMoment(int n, int* left, int leftSize, int* right, int rightSize)  
{  
}
```

Go:

```
func getLastMoment(n int, left []int, right []int) int {  
    }
```

Kotlin:

```
class Solution {  
    fun getLastMoment(n: Int, left: IntArray, right: IntArray): Int {  
        }  
    }
```

Swift:

```
class Solution {  
    func getLastMoment(_ n: Int, _ left: [Int], _ right: [Int]) -> Int {  
        }  
    }
```

Rust:

```
impl Solution {
    pub fn get_last_moment(n: i32, left: Vec<i32>, right: Vec<i32>) -> i32 {
        }
    }
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} left
# @param {Integer[]} right
# @return {Integer}
def get_last_moment(n, left, right)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer $n
     * @param Integer[] $left
     * @param Integer[] $right
     * @return Integer
     */
    function getLastMoment($n, $left, $right) {

    }
}
```

Dart:

```
class Solution {
    int getLastMoment(int n, List<int> left, List<int> right) {
        }
    }
```

Scala:

```
object Solution {
    def getLastMoment(n: Int, left: Array[Int], right: Array[Int]): Int = {
```

```
}
```

```
}
```

Elixir:

```
defmodule Solution do
  @spec get_last_moment(n :: integer, left :: [integer], right :: [integer]) :: integer
  def get_last_moment(n, left, right) do
    end
  end
```

Erlang:

```
-spec get_last_moment(N :: integer(), Left :: [integer()], Right :: [integer()]) -> integer().
get_last_moment(N, Left, Right) ->
  .
```

Racket:

```
(define/contract (get-last-moment n left right)
  (-> exact-integer? (listof exact-integer?) (listof exact-integer?)
    exact-integer?))
```

Solutions

C++ Solution:

```
/*
 * Problem: Last Moment Before All Ants Fall Out of a Plank
 * Difficulty: Medium
 * Tags: array
 *
 * 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 getLastMoment(int n, vector<int>& left, vector<int>& right) {

}
};


```

Java Solution:

```

/**
 * Problem: Last Moment Before All Ants Fall Out of a Plank
 * Difficulty: Medium
 * Tags: array
 *
 * 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 getLastMoment(int n, int[] left, int[] right) {

}
}


```

Python3 Solution:

```

"""
Problem: Last Moment Before All Ants Fall Out of a Plank
Difficulty: Medium
Tags: array

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 getLastMoment(self, n: int, left: List[int], right: List[int]) -> int:

```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
    def getLastMoment(self, n, left, right):
        """
        :type n: int
        :type left: List[int]
        :type right: List[int]
        :rtype: int
        """

```

JavaScript Solution:

```
/**
 * Problem: Last Moment Before All Ants Fall Out of a Plank
 * Difficulty: Medium
 * Tags: array
 *
 * 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} n
 * @param {number[]} left
 * @param {number[]} right
 * @return {number}
 */
var getLastMoment = function(n, left, right) {
}
```

TypeScript Solution:

```
/**
 * Problem: Last Moment Before All Ants Fall Out of a Plank
 * Difficulty: Medium

```

```

* Tags: array
*
* 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 getLastMoment(n: number, left: number[], right: number[]): number {
}

```

C# Solution:

```

/*
* Problem: Last Moment Before All Ants Fall Out of a Plank
* Difficulty: Medium
* Tags: array
*
* 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 GetLastMoment(int n, int[] left, int[] right) {
}
}

```

C Solution:

```

/*
* Problem: Last Moment Before All Ants Fall Out of a Plank
* Difficulty: Medium
* Tags: array
*
* 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 getLastMoment(int n, int* left, int leftSize, int* right, int rightSize)
{
}
```

Go Solution:

```
// Problem: Last Moment Before All Ants Fall Out of a Plank
// Difficulty: Medium
// Tags: array
//
// 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 getLastMoment(n int, left []int, right []int) int {
}
```

Kotlin Solution:

```
class Solution {
    fun getLastMoment(n: Int, left: IntArray, right: IntArray): Int {
        return 0
    }
}
```

Swift Solution:

```
class Solution {
    func getLastMoment(_ n: Int, _ left: [Int], _ right: [Int]) -> Int {
        return 0
    }
}
```

Rust Solution:

```
// Problem: Last Moment Before All Ants Fall Out of a Plank
// Difficulty: Medium
// Tags: array
//
```

```

// 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 get_last_moment(n: i32, left: Vec<i32>, right: Vec<i32>) -> i32 {
        }

    }
}

```

Ruby Solution:

```

# @param {Integer} n
# @param {Integer[]} left
# @param {Integer[]} right
# @return {Integer}
def get_last_moment(n, left, right)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer $n
     * @param Integer[] $left
     * @param Integer[] $right
     * @return Integer
     */
    function getLastMoment($n, $left, $right) {
        }

    }
}

```

Dart Solution:

```

class Solution {
    int getLastMoment(int n, List<int> left, List<int> right) {
        }
}

```

```
}
```

Scala Solution:

```
object Solution {  
    def getLastMoment(n: Int, left: Array[Int], right: Array[Int]): Int = {  
        }  
        }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec get_last_moment(n :: integer, left :: [integer], right :: [integer]) ::  
  integer  
  def get_last_moment(n, left, right) do  
  
  end  
end
```

Erlang Solution:

```
-spec get_last_moment(N :: integer(), Left :: [integer()], Right ::  
[integer()]) -> integer().  
get_last_moment(N, Left, Right) ->  
.
```

Racket Solution:

```
(define/contract (get-last-moment n left right)  
  (-> exact-integer? (listof exact-integer?) (listof exact-integer?)  
  exact-integer?)  
)
```