Problem 2954: Count the Number of Infection Sequences

Problem Information

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

Problem Description

You are given an integer

n

and an array

sick

sorted in increasing order, representing positions of infected people in a line of

n

people.

At each step,

one

uninfected person

adjacent

to an infected person gets infected. This process continues until everyone is infected.

An

infection sequence

is the order in which uninfected people become infected, excluding those initially infected.

Return the number of different infection sequences possible, modulo

10

9

+7

.

Example 1:

Input:

n = 5, sick = [0,4]

Output:

4

Explanation:

There is a total of 6 different sequences overall.

Valid infection sequences are

[1,2,3]

,

[1,3,2]

,

[3,2,1]

and

[3,1,2]

.

[2,3,1]

and

[2,1,3]

are not valid infection sequences because the person at index 2 cannot be infected at the first step.

Example 2:

Input:

n = 4, sick = [1]

Output:

3

Explanation:

There is a total of 6 different sequences overall.

Valid infection sequences are

[0,2,3]

,

[2,0,3]

and

[2,3,0]

.

[3,2,0]

,

[3,0,2]

, and

[0,3,2]

are not valid infection sequences because the infection starts at the person at index 1, then the order of infection is 2, then 3, and hence 3 cannot be infected earlier than 2.

Constraints:

2 <= n <= 10

5

1 <= sick.length <= n - 1

0 <= sick[i] <= n - 1

sick

is sorted in increasing order.

Code Snippets

C++:

```
class Solution {
public:
int numberOfSequence(int n, vector<int>& sick) {

}
};
```

Java:

```
class Solution {
public int numberOfSequence(int n, int[] sick) {

}
}
```

Python3:

```
class Solution:
def numberOfSequence(self, n: int, sick: List[int]) -> int:
```

Python:

```
class Solution(object):
def numberOfSequence(self, n, sick):
"""
:type n: int
:type sick: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} sick
* @return {number}
*/
var numberOfSequence = function(n, sick) {

};
```

TypeScript:

```
function numberOfSequence(n: number, sick: number[]): number {

};
```

C#:

```
public class Solution {
public int NumberOfSequence(int n, int[] sick) {

}
}
```

C:

```
int numberOfSequence(int n, int* sick, int sickSize) {

}
```

Go:

```
func numberOfSequence(n int, sick []int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfSequence(n: Int, sick: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func numberOfSequence(_ n: Int, _ sick: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_sequence(n: i32, sick: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} sick
# @return {Integer}
def number_of_sequence(n, sick)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $sick
* @return Integer
*/
function numberOfSequence($n, $sick) {

}
}
```

Dart:

```
class Solution {
int numberOfSequence(int n, List<int> sick) {

}
}
```

Scala:

```
object Solution {
def numberOfSequence(n: Int, sick: Array[Int]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec number_of_sequence(n :: integer, sick :: [integer]) :: integer
def number_of_sequence(n, sick) do

end
end
```

Erlang:

```
-spec number_of_sequence(N :: integer(), Sick :: [integer()]) -> integer().
number_of_sequence(N, Sick) ->
.
```

Racket:

```
(define/contract (number-of-sequence n sick)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Infection Sequences
* Difficulty: Hard
* Tags: array, math, sort
*
* 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 numberOfSequence(int n, vector<int>& sick) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Infection Sequences
* Difficulty: Hard
* Tags: array, math, sort
*
* 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 numberOfSequence(int n, int[] sick) {

}
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Infection Sequences
Difficulty: Hard
Tags: array, math, sort

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 numberOfSequence(self, n: int, sick: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfSequence(self, n, sick):
"""
:type n: int
:type sick: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count the Number of Infection Sequences
* Difficulty: Hard
* Tags: array, math, sort
*
* 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[]} sick
* @return {number}
*/
var numberOfSequence = function(n, sick) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Infection Sequences
* Difficulty: Hard
* Tags: array, math, sort
*
* 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 numberOfSequence(n: number, sick: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Count the Number of Infection Sequences
* Difficulty: Hard
* Tags: array, math, sort
*
* 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 NumberOfSequence(int n, int[] sick) {

}
}
```

C Solution:

```
/*
* Problem: Count the Number of Infection Sequences
* Difficulty: Hard
* Tags: array, math, sort
*
* 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 numberOfSequence(int n, int* sick, int sickSize) {

}
```

Go Solution:

```
// Problem: Count the Number of Infection Sequences
// Difficulty: Hard
```

```
// Tags: array, math, sort
//
// 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 numberOfSequence(n int, sick []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfSequence(n: Int, sick: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfSequence(_ n: Int, _ sick: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count the Number of Infection Sequences
// Difficulty: Hard
// Tags: array, math, sort
//
// 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 number_of_sequence(n: i32, sick: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} sick
# @return {Integer}
def number_of_sequence(n, sick)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $sick
* @return Integer
*/
function numberOfSequence($n, $sick) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfSequence(int n, List<int> sick) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfSequence(n: Int, sick: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_sequence(n :: integer, sick :: [integer]) :: integer
def number_of_sequence(n, sick) do

end
end
```

Erlang Solution:

```
-spec number_of_sequence(N :: integer(), Sick :: [integer()]) -> integer().
number_of_sequence(N, Sick) ->
.
```

Racket Solution:

```
(define/contract (number-of-sequence n sick)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```