Problem 1569: Number of Ways to Reorder Array to Get Same BST

Problem Information

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

Problem Description

Given an array

nums

that represents a permutation of integers from

1

to

n

. We are going to construct a binary search tree (BST) by inserting the elements of

nums

in order into an initially empty BST. Find the number of different ways to reorder

nums

so that the constructed BST is identical to that formed from the original array

nums

.

For example, given

nums = [2,1,3]

, we will have 2 as the root, 1 as a left child, and 3 as a right child. The array

[2,3,1]

also yields the same BST but

[3,2,1]

yields a different BST.

Return

the number of ways to reorder

nums

such that the BST formed is identical to the original BST formed from

nums

.

Since the answer may be very large,

return it modulo

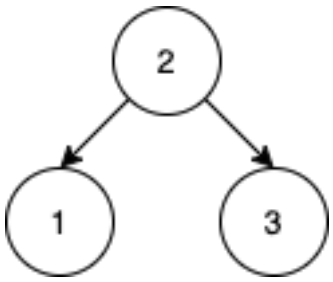
10

9

+ 7

.

Example 1:



Input:

nums = [2,1,3]

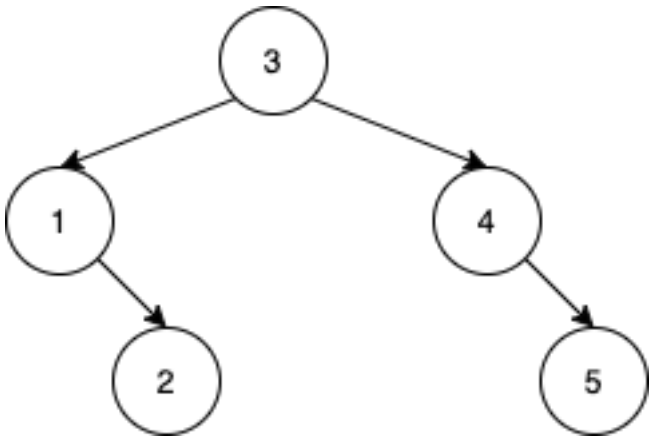
Output:

1

Explanation:

We can reorder nums to be [2,3,1] which will yield the same BST. There are no other ways to reorder nums which will yield the same BST.

Example 2:



Input:

nums = [3,4,5,1,2]

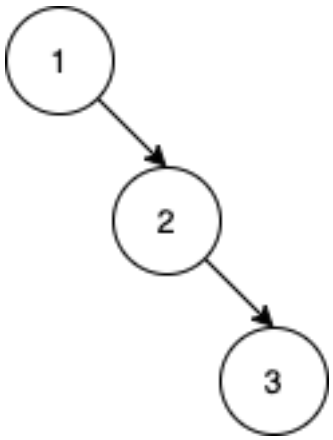
Output:

5

Explanation:

The following 5 arrays will yield the same BST: [3,1,2,4,5] [3,1,4,2,5] [3,1,4,5,2] [3,4,1,2,5] [3,4,1,5,2]

Example 3:



Input:

nums = [1,2,3]

Output:

0

Explanation:

There are no other orderings of nums that will yield the same BST.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= nums.length

All integers in

nums

are

distinct

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-of-ways nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Reorder Array to Get Same BST
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, search
*
```

```
* 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 numOfWays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Reorder Array to Get Same BST
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, search
*
* 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 numOfWays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Reorder Array to Get Same BST
Difficulty: Hard
Tags: array, tree, graph, dp, math, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Ways to Reorder Array to Get Same BST
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, search
*
* 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[]} nums
* @return {number}
*/
var numOfWays = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Reorder Array to Get Same BST
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, search
```

```
*
* 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 numOfWays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Reorder Array to Get Same BST
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, search
*
* 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 NumOfWays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Reorder Array to Get Same BST
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, search
*
* 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 numOfWays(int* nums, int numsSize) {
```

```
}
```

Go Solution:

```
// Problem: Number of Ways to Reorder Array to Get Same BST
// Difficulty: Hard
// Tags: array, tree, graph, dp, math, search
//
// 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 numOfWays(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Reorder Array to Get Same BST
// Difficulty: Hard
// Tags: array, tree, graph, dp, math, search
//
// 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_of_ways(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-of-ways nums)
(-> (listof exact-integer?) exact-integer?)
)
```