Problem 1902: Depth of BST Given Insertion Order

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

order

of length

n

, a

permutation

of integers from

1

to

n

representing the

order

of insertion into a

binary search tree

.

A binary search tree is defined as follows:

The left subtree of a node contains only nodes with keys

less than

the node's key.

The right subtree of a node contains only nodes with keys

greater than

the node's key.

Both the left and right subtrees must also be binary search trees.

The binary search tree is constructed as follows:

order[0]

will be the

root

of the binary search tree.

All subsequent elements are inserted as the

child

of

any

existing node such that the binary search tree properties hold.

Return

the

depth

of the binary search tree

.

A binary tree's

depth

is the number of

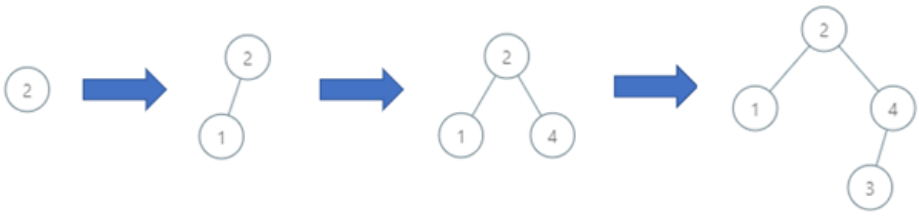
nodes

along the

longest path

from the root node down to the farthest leaf node.

Example 1:



Input:

order = [2,1,4,3]

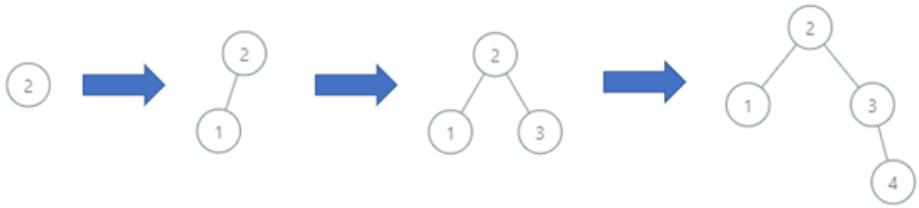
Output:

3

Explanation:

The binary search tree has a depth of 3 with path 2->3->4.

Example 2:



Input:

order = [2,1,3,4]

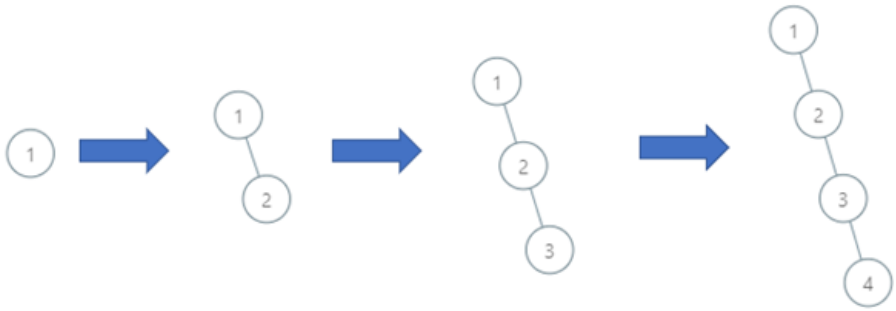
Output:

3

Explanation:

The binary search tree has a depth of 3 with path 2->3->4.

Example 3:



Input:

order = [1,2,3,4]

Output:

4

Explanation:

The binary search tree has a depth of 4 with path 1->2->3->4.

Constraints:

n == order.length

1 <= n <= 10

5

order

is a permutation of integers between

1

and

n

.

Code Snippets

C++:

```
class Solution {
public:
int maxDepthBST(vector<int>& order) {

}
};
```

Java:

```
class Solution {
public int maxDepthBST(int[] order) {

}
}
```

Python3:

```
class Solution:
def maxDepthBST(self, order: List[int]) -> int:
```

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function maxDepthBST(order: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxDepthBST(int[] order) {

}
}
```

C:

```
int maxDepthBST(int* order, int orderSize) {

}
```

Go:

```
func maxDepthBST(order []int) int {

}
```

Kotlin:

```
class Solution {
fun maxDepthBST(order: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxDepthBST(_ order: [Int]) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn max_depth_bst(order: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} order
# @return {Integer}
def max_depth_bst(order)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $order
* @return Integer
*/
function maxDepthBST($order) {

}
}
```

Dart:

```
class Solution {
int maxDepthBST(List<int> order) {

}
}
```

Scala:

```
object Solution {
def maxDepthBST(order: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_depth_bst(order :: [integer]) :: integer
def max_depth_bst(order) do

end
end
```

Erlang:

```
-spec max_depth_bst(Order :: [integer()]) -> integer().
max_depth_bst(Order) ->
.
```

Racket:

```
(define/contract (max-depth-bst order)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Depth of BST Given Insertion Order
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
class Solution {
public:
int maxDepthBST(vector<int>& order) {

}
};
```

Java Solution:

```
/**
* Problem: Depth of BST Given Insertion Order
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int maxDepthBST(int[] order) {

}
}
```

Python3 Solution:

```
"""
Problem: Depth of BST Given Insertion Order
Difficulty: Medium
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def maxDepthBST(self, order: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Depth of BST Given Insertion Order
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} order
* @return {number}
*/
var maxDepthBST = function(order) {

};
```

TypeScript Solution:

```
/**
* Problem: Depth of BST Given Insertion Order
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function maxDepthBST(order: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Depth of BST Given Insertion Order
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MaxDepthBST(int[] order) {

}
}
```

C Solution:

```
/*
* Problem: Depth of BST Given Insertion Order
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int maxDepthBST(int* order, int orderSize) {

}
```

Go Solution:

```
// Problem: Depth of BST Given Insertion Order
// Difficulty: Medium
```

```
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func maxDepthBST(order []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxDepthBST(order: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxDepthBST(_ order: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Depth of BST Given Insertion Order
// Difficulty: Medium
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn max_depth_bst(order: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} order
# @return {Integer}
def max_depth_bst(order)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $order
* @return Integer
*/
function maxDepthBST($order) {

}
}
```

Dart Solution:

```
class Solution {
int maxDepthBST(List<int> order) {

}
}
```

Scala Solution:

```
object Solution {
def maxDepthBST(order: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_depth_bst(order :: [integer]) :: integer
def max_depth_bst(order) do
```

```
end
end
```

Erlang Solution:

```
-spec max_depth_bst(Order :: [integer()]) -> integer().
max_depth_bst(Order) ->
.
```

Racket Solution:

```
(define/contract (max-depth-bst order)
(-> (listof exact-integer?) exact-integer?)
)
```