Problem 946: Validate Stack Sequences

Problem Information

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

Problem Description

Given two integer arrays

pushed

and

popped

each with distinct values, return

true

if this could have been the result of a sequence of push and pop operations on an initially empty stack, or

false

otherwise.

Example 1:

Input:

pushed = [1,2,3,4,5], popped = [4,5,3,2,1]

Output:

true

Explanation:

We might do the following sequence: push(1), push(2), push(3), push(4), pop() -> 4, push(5), pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1

Example 2:

Input:

pushed = [1,2,3,4,5], popped = [4,3,5,1,2]

Output:

false

Explanation:

1 cannot be popped before 2.

Constraints:

1 <= pushed.length <= 1000

0 <= pushed[i] <= 1000

All the elements of

pushed

are

unique

.

popped.length == pushed.length

popped

is a permutation of

pushed

.

Code Snippets

C++:

```
class Solution {
public:
bool validateStackSequences(vector<int>& pushed, vector<int>& popped) {

}
};
```

Java:

```
class Solution {
public boolean validateStackSequences(int[] pushed, int[] popped) {

}
}
```

Python3:

```
class Solution:
def validateStackSequences(self, pushed: List[int], popped: List[int]) ->
bool:
```

Python:

```
class Solution(object):
def validateStackSequences(self, pushed, popped):
"""
:type pushed: List[int]
:type popped: List[int]
```

```
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} pushed
* @param {number[]} popped
* @return {boolean}
*/
var validateStackSequences = function(pushed, popped) {

};
```

TypeScript:

```
function validateStackSequences(pushed: number[], popped: number[]): boolean
{

};
```

C#:

```
public class Solution {
public bool ValidateStackSequences(int[] pushed, int[] popped) {

}
}
```

C:

```
bool validateStackSequences(int* pushed, int pushedSize, int* popped, int
poppedSize) {

}
```

Go:

```
func validateStackSequences(pushed []int, popped []int) bool {

}
```

Kotlin:

```
class Solution {
fun validateStackSequences(pushed: IntArray, popped: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func validateStackSequences(_ pushed: [Int], _ popped: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn validate_stack_sequences(pushed: Vec<i32>, popped: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} pushed
# @param {Integer[]} popped
# @return {Boolean}
def validate_stack_sequences(pushed, popped)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $pushed
* @param Integer[] $popped
* @return Boolean
*/
function validateStackSequences($pushed, $popped) {
```

```
}
}
```

Dart:

```
class Solution {
bool validateStackSequences(List<int> pushed, List<int> popped) {

}
}
```

Scala:

```
object Solution {
def validateStackSequences(pushed: Array[Int], popped: Array[Int]): Boolean =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec validate_stack_sequences(pushed :: [integer], popped :: [integer]) ::
boolean
def validate_stack_sequences(pushed, popped) do

end
end
```

Erlang:

```
-spec validate_stack_sequences(Pushed :: [integer()], Popped :: [integer()])
-> boolean().
validate_stack_sequences(Pushed, Popped) ->
.
```

Racket:

```
(define/contract (validate-stack-sequences pushed popped)
(-> (listof exact-integer?) (listof exact-integer?) boolean?)
```

```
)
```

Solutions

C++ Solution:

```
/*
* Problem: Validate Stack Sequences
* Difficulty: Medium
* Tags: array, stack
*
* 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:
bool validateStackSequences(vector<int>& pushed, vector<int>& popped) {

}
};
```

Java Solution:

```
/**
* Problem: Validate Stack Sequences
* Difficulty: Medium
* Tags: array, stack
*
* 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 boolean validateStackSequences(int[] pushed, int[] popped) {

}
}
```

Python3 Solution:

```
"""
Problem: Validate Stack Sequences
Difficulty: Medium
Tags: array, stack

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 validateStackSequences(self, pushed: List[int], popped: List[int]) ->
bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def validateStackSequences(self, pushed, popped):
"""
:type pushed: List[int]
:type popped: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Validate Stack Sequences
* Difficulty: Medium
* Tags: array, stack
*
* 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[]} pushed
```

```
* @param {number[]} popped
* @return {boolean}
*/
var validateStackSequences = function(pushed, popped) {

};
```

TypeScript Solution:

```
/**
* Problem: Validate Stack Sequences
* Difficulty: Medium
* Tags: array, stack
*
* 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 validateStackSequences(pushed: number[], popped: number[]): boolean
{

};
```

C# Solution:

```
/*
* Problem: Validate Stack Sequences
* Difficulty: Medium
* Tags: array, stack
*
* 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 bool ValidateStackSequences(int[] pushed, int[] popped) {

}
}
```

C Solution:

```
/*
* Problem: Validate Stack Sequences
* Difficulty: Medium
* Tags: array, stack
*
* 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
*/

bool validateStackSequences(int* pushed, int pushedSize, int* popped, int
poppedSize) {

}
```

Go Solution:

```
// Problem: Validate Stack Sequences
// Difficulty: Medium
// Tags: array, stack
//
// 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 validateStackSequences(pushed []int, popped []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun validateStackSequences(pushed: IntArray, popped: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func validateStackSequences(_ pushed: [Int], _ popped: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Validate Stack Sequences
// Difficulty: Medium
// Tags: array, stack
//
// 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 validate_stack_sequences(pushed: Vec<i32>, popped: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} pushed
# @param {Integer[]} popped
# @return {Boolean}
def validate_stack_sequences(pushed, popped)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $pushed
* @param Integer[] $popped
* @return Boolean
*/
function validateStackSequences($pushed, $popped) {
```

```
}
}
```

Dart Solution:

```
class Solution {
bool validateStackSequences(List<int> pushed, List<int> popped) {

}
}
```

Scala Solution:

```
object Solution {
def validateStackSequences(pushed: Array[Int], popped: Array[Int]): Boolean =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec validate_stack_sequences(pushed :: [integer], popped :: [integer]) ::
boolean
def validate_stack_sequences(pushed, popped) do

end
end
```

Erlang Solution:

```
-spec validate_stack_sequences(Pushed :: [integer()], Popped :: [integer()])
-> boolean().
validate_stack_sequences(Pushed, Popped) ->
.
```

Racket Solution:

```
(define/contract (validate-stack-sequences pushed popped)
(-> (listof exact-integer?) (listof exact-integer?) boolean?)
```

```
)
```