Problem 255: Verify Preorder Sequence in Binary Search Tree

Problem Information

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

Problem Description

Given an array of

unique

integers

preorder

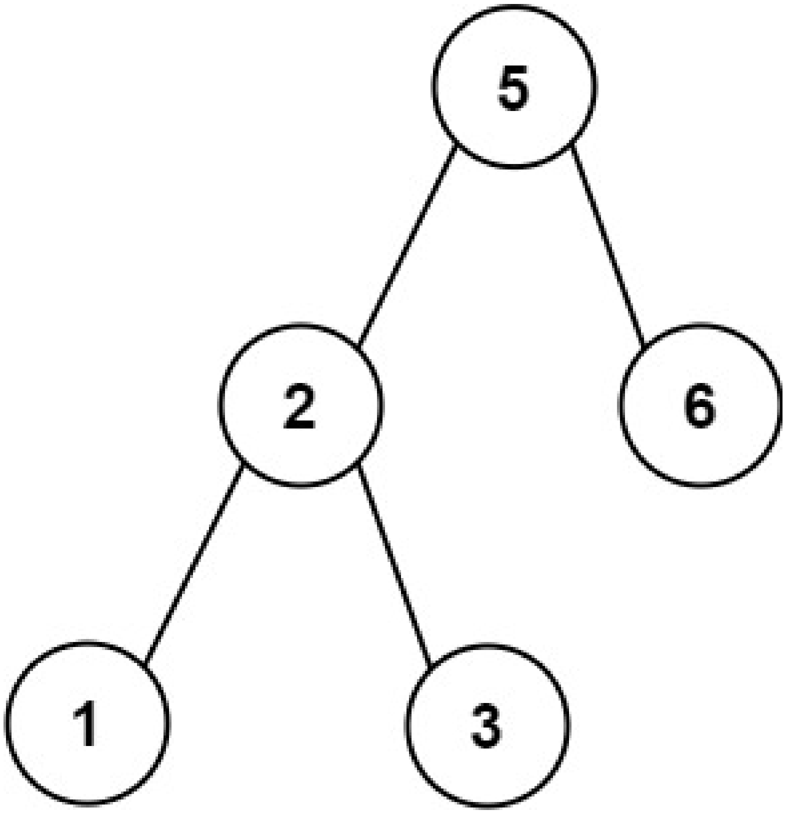
, return

true

if it is the correct preorder traversal sequence of a binary search tree

.

Example 1:



Input:

preorder = [5,2,1,3,6]

Output:

true

Example 2:

Input:

preorder = [5,2,6,1,3]

Output:

false

Constraints:

1 <= preorder.length <= 10

4

1 <= preorder[i] <= 10

4

All the elements of

preorder

are

unique

.

Follow up:

Could you do it using only constant space complexity?

Code Snippets

C++:

```
class Solution {
public:
bool verifyPreorder(vector<int>& preorder) {

}
};
```

Java:

```
class Solution {
public boolean verifyPreorder(int[] preorder) {

}
}
```

Python3:

```
class Solution:
def verifyPreorder(self, preorder: List[int]) -> bool:
```

Python:

```
class Solution(object):
def verifyPreorder(self, preorder):
"""
:type preorder: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} preorder
* @return {boolean}
*/
var verifyPreorder = function(preorder) {

};
```

TypeScript:

```
function verifyPreorder(preorder: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool VerifyPreorder(int[] preorder) {

}
}
```

C:

```
bool verifyPreorder(int* preorder, int preorderSize) {

}
```

Go:

```
func verifyPreorder(preorder []int) bool {

}
```

Kotlin:

```
class Solution {
fun verifyPreorder(preorder: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func verifyPreorder(_ preorder: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn verify_preorder(preorder: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} preorder
# @return {Boolean}
def verify_preorder(preorder)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $preorder
* @return Boolean
*/
function verifyPreorder($preorder) {

}
}
```

Dart:

```
class Solution {
bool verifyPreorder(List<int> preorder) {

}
}
```

Scala:

```
object Solution {
def verifyPreorder(preorder: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec verify_preorder(preorder :: [integer]) :: boolean
def verify_preorder(preorder) do

end
end
```

Erlang:

```
-spec verify_preorder(Preorder :: [integer()]) -> boolean().
verify_preorder(Preorder) ->
.
```

Racket:

```
(define/contract (verify-preorder preorder)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Verify Preorder Sequence in Binary Search Tree
* Difficulty: Medium
* Tags: array, tree, search, stack
*
* 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:
bool verifyPreorder(vector<int>& preorder) {

}
};
```

Java Solution:

```
/**
* Problem: Verify Preorder Sequence in Binary Search Tree
* Difficulty: Medium
* Tags: array, tree, search, stack
*
* 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 boolean verifyPreorder(int[] preorder) {

}
}
```

Python3 Solution:

```
"""
Problem: Verify Preorder Sequence in Binary Search Tree
Difficulty: Medium
Tags: array, tree, search, stack

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 verifyPreorder(self, preorder: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def verifyPreorder(self, preorder):
"""
:type preorder: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Verify Preorder Sequence in Binary Search Tree
* Difficulty: Medium
* Tags: array, tree, search, stack
*
* 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[]} preorder
* @return {boolean}
*/
var verifyPreorder = function(preorder) {

};
```

TypeScript Solution:

```
/**
* Problem: Verify Preorder Sequence in Binary Search Tree
* Difficulty: Medium
* Tags: array, tree, search, stack
*
* 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 verifyPreorder(preorder: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Verify Preorder Sequence in Binary Search Tree
* Difficulty: Medium
* Tags: array, tree, search, stack
*
* 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 bool VerifyPreorder(int[] preorder) {

}
}
```

C Solution:

```
/*
* Problem: Verify Preorder Sequence in Binary Search Tree
* Difficulty: Medium
* Tags: array, tree, search, stack
*
* 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
*/

bool verifyPreorder(int* preorder, int preorderSize) {

}
```

Go Solution:

```
// Problem: Verify Preorder Sequence in Binary Search Tree
// Difficulty: Medium
// Tags: array, tree, search, stack
//
// 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 verifyPreorder(preorder []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun verifyPreorder(preorder: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func verifyPreorder(_ preorder: [Int]) -> Bool {
}
}
```

Rust Solution:

```
// Problem: Verify Preorder Sequence in Binary Search Tree
// Difficulty: Medium
// Tags: array, tree, search, stack
//
// 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 verify_preorder(preorder: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} preorder
# @return {Boolean}
def verify_preorder(preorder)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $preorder
* @return Boolean
*/
function verifyPreorder($preorder) {

}
}
```

Dart Solution:

```
class Solution {
bool verifyPreorder(List<int> preorder) {

}
}
```

Scala Solution:

```
object Solution {
def verifyPreorder(preorder: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec verify_preorder(preorder :: [integer]) :: boolean
def verify_preorder(preorder) do

end
end
```

Erlang Solution:

```
-spec verify_preorder(Preorder :: [integer()]) -> boolean().
verify_preorder(Preorder) ->
.
```

Racket Solution:

```
(define/contract (verify-preorder preorder)
(-> (listof exact-integer?) boolean?)
)
```