Check if a given array can represent Preorder Traversal of Binary Search Tree

Given an array of numbers, return true if given array can represent preorder traversal of a Binary Search Tree, else return false. Expected time complexity is O(n).

Examples:

```
Input:  pre[] = {2, 4, 3}
Output: true
Given array can represent preorder traversal
of below tree
    2
     \
      4
     /
    3

Input:  pre[] = {2, 4, 1}
Output: false
Given array cannot represent preorder traversal
of a Binary Search Tree.

Input:  pre[] = {40, 30, 35, 80, 100}
Output: true
Given array can represent preorder traversal
of below tree
     40
   /   \
 30    80
  \      \
  35     100


Input:  pre[] = {40, 30, 35, 20, 80, 100}
Output: false
Given array cannot represent preorder traversal
of a Binary Search Tree.
```

We strongly recommend that you click here and practice it, before moving on to the solution.

A **Simple Solution** is to do following for every node pre[i] starting from first one.

```
1) Find the first greater value on right side of current node.
   Let the index of this node be j. Return true if following
   conditions hold. Else return false
    (i)  All values after the above found greater value are
         greater than current node.
    (ii) Recursive calls for the subarrays pre[i+1..j-1] and
         pre[j+1..n-1] also return true.
```

Time Complexity of the above solution is $O(n^2)$

An **Efficient Solution** can solve this problem in O(n) time. The idea is to use a stack. This problem is similar to Next (or closest) Greater Element problem. Here we find next greater element and after finding next greater, if we find a smaller element, then return false.

```
1) Create an empty stack.
2) Initialize root as INT_MIN.
3) Do following for every element pre[i]
     a) If pre[i] is smaller than current root, return false.
     b) Keep removing elements from stack while pre[i] is greater
        then stack top. Make the last removed item as new root (to
        be compared next).
        At this point, pre[i] is greater than the removed root
        (That is why if we see a smaller element in step a), we
        return false)
     c) push pre[i] to stack (All elements in stack are in decreasing
        order)
```

Below is implementation of above idea.

C++

```
// C++ program for an efficient solution to check if
// a given array can represent Preorder traversal of
// a Binary Search Tree
#include<bits/stdc++.h>
using namespace std;

bool canRepresentBST(int pre[], int n)
{
    // Create an empty stack
    stack<int> s;

    // Initialize current root as minimum possible
    // value
    int root = INT_MIN;

    // Traverse given array
    for (int i=0; i<n; i++)
    {
        // If we find a node who is on right side
        // and smaller than root, return false
        if (pre[i] < root)
            return false;

        // If pre[i] is in right subtree of stack top,
        // Keep removing items smaller than pre[i]
        // and make the last removed item as new
        // root.
        while (!s.empty() && s.top()<pre[i])
        {
            root = s.top();
            s.pop();
        }

        // At this point either stack is empty or
        // pre[i] is smaller than root, push pre[i]
        s.push(pre[i]);
    }
    return true;
}

// Driver program
int main()
{
    int pre1[] = {40, 30, 35, 80, 100};
    int n = sizeof(pre1)/sizeof(pre1[0]);
    canRepresentBST(pre1, n)? cout << "true\n":
                              cout << "false\n";

    int pre2[] = {40, 30, 35, 20, 80, 100};
    n = sizeof(pre2)/sizeof(pre2[0]);
    canRepresentBST(pre2, n)? cout << "true\n":
                              cout << "false\n";

    return 0;
}
```

Java

```java
// Java program for an efficient solution to check if
// a given array can represent Preorder traversal of
// a Binary Search Tree
import java.util.Stack;

class BinarySearchTree {

    boolean canRepresentBST(int pre[], int n) {
        // Create an empty stack
        Stack<Integer> s = new Stack<Integer>();

        // Initialize current root as minimum possible
        // value
        int root = Integer.MIN_VALUE;

        // Traverse given array
        for (int i = 0; i < n; i++) {
            // If we find a node who is on right side
            // and smaller than root, return false
            if (pre[i] < root) {
                return false;
            }

            // If pre[i] is in right subtree of stack top,
            // Keep removing items smaller than pre[i]
            // and make the last removed item as new
            // root.
            while (!s.empty() && s.peek() < pre[i]) {
                root = s.peek();
                s.pop();
            }

            // At this point either stack is empty or
            // pre[i] is smaller than root, push pre[i]
            s.push(pre[i]);
        }
        return true;
    }

    public static void main(String args[]) {
        BinarySearchTree bst = new BinarySearchTree();
        int[] pre1 = new int[]{40, 30, 35, 80, 100};
        int n = pre1.length;
        if (bst.canRepresentBST(pre1, n) == true) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
        int[] pre2 = new int[]{40, 30, 35, 20, 80, 100};
        int n1 = pre2.length;
        if (bst.canRepresentBST(pre2, n) == true) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
    }
}

//This code is contributed by Mayank Jaiswal
```

Python

```
# Python program for an efficient solution to check if
# a given array can represent Preorder traversal of
# a Binary Search Tree

INT_MIN = -2**32

def canRepresentBST(pre):

    # Create an empty stack
    s = []

    # Initialize current root as minimum possible value
    root = INT_MIN

    # Traverse given array
    for value in pre:
        #NOTE:value is equal to pre[i] according to the
        #given algo

        # If we find a node who is on the right side
        # and smaller than root, return False
        if value < root :
            return False

        # If value(pre[i]) is in right subtree of stack top,
        # Keep removing items smaller than value
        # and make the last removed items as new root
        while(len(s) > 0 and s[-1] < value) :
            root = s.pop()

        # At this point either stack is empty or value
        # is smaller than root, push value
        s.append(value)

    return True

# Driver Program
pre1 = [40 , 30 , 35 , 80 , 100]
print "true" if canRepresentBST(pre1) == True else "false"
pre2 = [40 , 30 , 35 , 20 ,  80 , 100]
print "true" if canRepresentBST(pre2) == True else "false"

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
true
false
```