Check if each internal node of a BST has exactly one child

Given Preorder traversal of a BST, check if each non-leaf node has only one child. Assume that the BST contains unique entries.

Examples

```
Input: pre[] = {20, 10, 11, 13, 12}
Output: Yes
The give array represents following BST. In the following BST, every internal
node has exactly 1 child. Therefor, the output is true.
        20
       /
      10
       \
        11
          \
           13
           /
         12
```

In Preorder traversal, descendants (or Preorder successors) of every node appear after the node. In the above example, 20 is the first node in preorder and all descendants of 20 appear after it. All descendants of 20 are smaller than it. For 10, all descendants are greater than it. In general, we can say, if all internal nodes have only one child in a BST, then all the descendants of every node are either smaller or larger than the node. The reason is simple, since the tree is BST and every node has only one child, all descendants of a node will either be on left side or right side, means all descendants will either be smaller or greater.

Approach 1 (Naive)

This approach simply follows the above idea that all values on right side are either smaller or larger. Use two loops, the outer loop picks an element one by one, starting from the leftmost element. The inner loop checks if all elements on the right side of the picked element are either smaller or greater. The time complexity of this method will be O(n^2).

Approach 2

Since all the descendants of a node must either be larger or smaller than the node. We can do following for every node in a loop.

1. Find the next preorder successor (or descendant) of the node.
2. Find the last preorder successor (last element in pre[]) of the node.
3. If both successors are less than the current node, or both successors are greater than the current node, then continue. Else, return false.

C

```
#include <stdio.h>

bool hasOnlyOneChild(int pre[], int size)
{
    int nextDiff, lastDiff;

    for (int i=0; i<size-1; i++)
    {
        nextDiff = pre[i] - pre[i+1];
        lastDiff = pre[i] - pre[size-1];
        if (nextDiff*lastDiff < 0)
            return false;;
    }
    return true;
}

// driver program to test above function
int main()
{
    int pre[] = {8, 3, 5, 7, 6};
    int size = sizeof(pre)/sizeof(pre[0]);
    if (hasOnlyOneChild(pre, size) == true )
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

Java

```
// Check if each internal node of BST has only one child

class BinaryTree {

    boolean hasOnlyOneChild(int pre[], int size) {
        int nextDiff, lastDiff;

        for (int i = 0; i < size - 1; i++) {
            nextDiff = pre[i] - pre[i + 1];
            lastDiff = pre[i] - pre[size - 1];
            if (nextDiff * lastDiff < 0) {
                return false;
            };
        }
        return true;
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        int pre[] = new int[]{8, 3, 5, 7, 6};
        int size = pre.length;
        if (tree.hasOnlyOneChild(pre, size) == true) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }
}

// This code has been contributed by Mayank Jaiswal
```

Output:

```
 Yes
```

Approach 3

1. Scan the last two nodes of preorder & mark them as min & max.
2. Scan every node down the preorder array. Each node must be either smaller than the min node or larger than the max node. Update

min & max accordingly.

C

```
#include <stdio.h>

int hasOnlyOneChild(int pre[], int size)
{
    // Initialize min and max using last two elements
    int min, max;
    if (pre[size-1] > pre[size-2])
    {
        max = pre[size-1];
        min = pre[size-2]):
    else
    {
        max = pre[size-2];
        min = pre[size-1];
    }


    // Every element must be either smaller than min or
    // greater than max
    for (int i=size-3; i>=0; i--)
    {
        if (pre[i] < min)
            min = pre[i];
        else if (pre[i] > max)
            max = pre[i];
        else
            return false;
    }
    return true;
}

// Driver program to test above function
int main()
{
    int pre[] = {8, 3, 5, 7, 6};
    int size = sizeof(pre)/sizeof(pre[0]);
    if (hasOnlyOneChild(pre,size))
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

Java

```
// Check if each internal node of BST has only one child

class BinaryTree {

    boolean hasOnlyOneChild(int pre[], int size) {
        // Initialize min and max using last two elements
        int min, max;
        if (pre[size - 1] > pre[size - 2]) {
            max = pre[size - 1];
            min = pre[size - 2];
        } else {
            max = pre[size - 2];
            min = pre[size - 1];
        }

        // Every element must be either smaller than min or
        // greater than max
        for (int i = size - 3; i >= 0; i--) {
            if (pre[i] < min) {
                min = pre[i];
            } else if (pre[i] > max) {
                max = pre[i];
            } else {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        int pre[] = new int[]{8, 3, 5, 7, 6};
        int size = pre.length;
        if (tree.hasOnlyOneChild(pre, size) == true) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }
}

// This code has been contributed by Mayank Jaiswal
```

Output:

```
Yes
```