PROBLEM

Pairs violating the BST property

Medium   Accuracy: 44.43%   Submissions: 45+   Points: 4

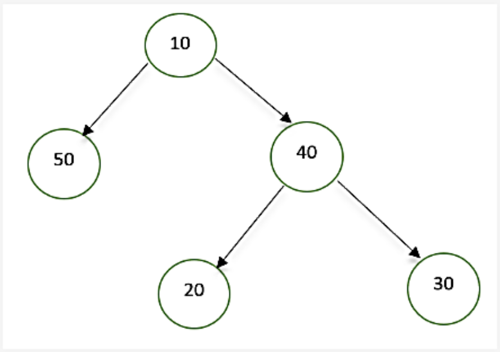
Given a binary tree with **n** nodes, find the number of **pairs violating the BST property**.

BST has the following properties:-

- Every node is greater than its left child and less than its right child.
- Every node is greater than the maximum value of in its left subtree and less than the minimum value in its right subtree.
- The maximum in the left sub-tree must be less than the minimum in the right subtree.

Example 1:

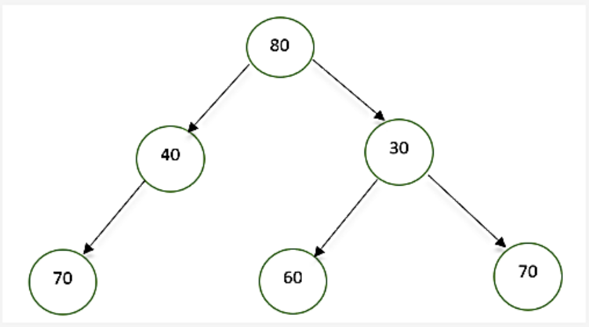
Input :
n = 5
Input tree



Output :
5
Explanation :
Pairs violating BST property are:-
(10,50), 10 should be greater than its left child value.
(40,30), 40 should be less than its right child value.
(50,20), (50,30) and (50,40), maximum of left subtree of 10 is 50 greater than 20, 30 and 40 of its right subtree.

Example 2:

Input :
n = 6
Input tree



Output :
5
Explanation :
Pairs violating BST property are:-
(80,30), greater than its right child.
(80,60), greater than node on its right side.
(80,70), greater than node on its right side.
(30,60), the value of 3 is not more than its left child.
(40,30), the value in the left subtree is greater than the value of the right subtree.

Your task :
You don't have to read input or print anything. Your task is to complete the function **pairsViolatingBST()** that takes the root of the tree and **n** as input and returns number of pairs violating BST property.

Expected Time Complexity: O(n*logn)
Expected Space Complexity: O(n)

Your Task :
1 <= n <= $2*10^4$
-10^9 <= node->data <= 10^9

CODE

```
from typing import Optional
from collections import deque
from bisect import bisect_right
"""
definition of binary tree node.
class Node:
   def _init_(self,val):
      self.data = val
      self.left = None
      self.right = None
"""
```

```
class Solution:
    def pairsViolatingBST(self, n : int, root :
Optional['Node']) -> int:
        inorder = []

        def INORDER(root):
            if not root:
                return
            INORDER(root.left)
            inorder.append(root.data)
            INORDER(root.right)

        INORDER(root)
        if len(inorder) <= 1:
            return 0

        pq = []
        for i, val in enumerate(inorder):
            pq.append((val, i))
        pq.sort()

        ans = 0
        x = []
        while pq:
            val, i = pq.pop(0)
            cnt = bisect_right(x, i)
            ans += i - cnt
            x.insert(cnt, i)

        return ans
        # code here
```