```swift
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

class Program {
  // Average case: when the created BST is balanced
  // O(nlog(n)) time | O(n) space - where n is the length of the array
  // ---
  // Worst case: when the the created BST is like a linked list
  // O(n^2) time | O(n) space
  static func rightSmallerThan(_ array: [Int]) -> [Int] {
    if array.count == 0 {
      return [Int]()
    }

    var rightSmallerCounts = [Int]()
    for i in array {
      rightSmallerCounts.append(i)
    }
    let lastIdx = array.count - 1
    var bst = SpecialBST(array[lastIdx])
    rightSmallerCounts[lastIdx] = 0
    for i in (0 ..< lastIdx).reversed() {
      bst.insert(array[i], i, &rightSmallerCounts)
    }
    return rightSmallerCounts
  }

  class SpecialBST {
    var value: Int

    var leftSubtreeSize: Int = 0
    var left: SpecialBST?
    var right: SpecialBST?

    init(_ value: Int) {
      self.value = value
    }

    func insert(_ value: Int, _ idx: Int, _ rightSmallerCounts: inout [Int], _ numSmallerAtInsertTime: Int = 0) {
      if value < self.value {
        leftSubtreeSize += 1
        if let left = self.left {
          left.insert(value, idx, &rightSmallerCounts, numSmallerAtInsertTime)
        } else {
          left = SpecialBST(value)
          rightSmallerCounts[idx] = numSmallerAtInsertTime
        }
        return
      }

      var numSmaller = numSmallerAtInsertTime + leftSubtreeSize
      if value > self.value {
        numSmaller += 1
      }

      if let right = self.right {
        right.insert(value, idx, &rightSmallerCounts, numSmaller)
      } else {
        right = SpecialBST(value)
        rightSmallerCounts[idx] = numSmaller
      }
    }
  }
}
```