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

package main

// 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
func RightSmallerThan(array []int) []int {
  if len(array) == 0 {
    return []int{}
  }

  rightSmallerCounts := make([]int, 0, len(array))
  for _, i := range array {
    rightSmallerCounts = append(rightSmallerCounts, i)
  }
  lastIdx := len(array) - 1
  bst := NewSpecialBST(array[lastIdx])
  rightSmallerCounts[lastIdx] = 0
  for i := lastIdx - 1; i >= 0; i-- {
    bst.Insert(array[i], i, rightSmallerCounts)
  }
  return rightSmallerCounts
}

type SpecialBST struct {
  Value           int
  LeftSubtreeSize int

  Left  *SpecialBST
  Right *SpecialBST
}

func NewSpecialBST(value int) *SpecialBST {
  return &SpecialBST{
    Value:           value,
    LeftSubtreeSize: 0,
    Left:            nil,
    Right:           nil,
  }
}

func (bst *SpecialBST) Insert(value, idx int, rightSmallerCounts []int) {
  bst.insert(value, idx, rightSmallerCounts, 0)
}

func (bst *SpecialBST) insert(value, idx int, rightSmallerCounts []int, numSmallerAtInsertTime int) {
  if value < bst.Value {
    bst.LeftSubtreeSize += 1
    if bst.Left == nil {
      bst.Left = NewSpecialBST(value)
      rightSmallerCounts[idx] = numSmallerAtInsertTime
    } else {
      bst.Left.insert(value, idx, rightSmallerCounts, numSmallerAtInsertTime)
    }
    return
  }

  numSmallerAtInsertTime += bst.LeftSubtreeSize
  if value > bst.Value {
    numSmallerAtInsertTime += 1
  }

  if bst.Right == nil {
    bst.Right = NewSpecialBST(value)
    rightSmallerCounts[idx] = numSmallerAtInsertTime
  } else {
    bst.Right.insert(value, idx, rightSmallerCounts, numSmallerAtInsertTime)
  }
}

```