```
1   // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3   // Average case: when the created BST is balanced
4   // O(nlog(n)) time | O(n) space - where n is the length of the array
5   // ---
6   // Worst case: when the the created BST is like a linked list
7   // O(n^2) time | O(n) space
8   function rightSmallerThan(array) {
9     if (array.length === 0) return [];
10
11    const rightSmallerCounts = array.slice();
12    const lastIdx = array.length - 1;
13    const bst = new SpecialBST(array[lastIdx]);
14    rightSmallerCounts[lastIdx] = 0;
15    for (let i = array.length - 2; i >= 0; i--) {
16      bst.insert(array[i], i, rightSmallerCounts);
17    }
18
19    return rightSmallerCounts;
20  }
21
22  class SpecialBST {
23    constructor(value) {
24      this.value = value;
25      this.leftSubTreeSize = 0;
26      this.left = null;
27      this.right = null;
28    }
29
30    insert(value, idx, rightSmallerCounts, numSmallerAtInsertTime = 0) {
31      if (value < this.value) {
32        this.leftSubTreeSize++;
33        if (this.left === null) {
34          this.left = new SpecialBST(value);
35          rightSmallerCounts[idx] = numSmallerAtInsertTime;
36        } else {
37          this.left.insert(value, idx, rightSmallerCounts, numSmallerAtInsertTime);
38        }
39      } else {
40        numSmallerAtInsertTime += this.leftSubTreeSize;
41        if (value > this.value) numSmallerAtInsertTime++;
42        if (this.right === null) {
43          this.right = new SpecialBST(value);
44          rightSmallerCounts[idx] = numSmallerAtInsertTime;
45        } else {
46          this.right.insert(value, idx, rightSmallerCounts, numSmallerAtInsertTime);
47        }
48      }
49    }
50  }
51
52  exports.rightSmallerThan = rightSmallerThan;
53
```