Solution 1   Solution 2   Solution 3

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

// 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
function rightSmallerThan(array) {
  if (array.length === 0) return [];

  const lastIdx = array.length - 1;
  const bst = new SpecialBST(array[lastIdx], lastIdx, 0);
  for (let i = array.length - 2; i >= 0; i--) {
    bst.insert(array[i], i);
  }

  const rightSmallerCounts = array.slice();
  getRightSmallerCounts(bst, rightSmallerCounts);
  return rightSmallerCounts;
}

function getRightSmallerCounts(bst, rightSmallerCounts) {
  if (bst === null) return;
  rightSmallerCounts[bst.idx] = bst.numSmallerAtInsertTime;
  getRightSmallerCounts(bst.left, rightSmallerCounts);
  getRightSmallerCounts(bst.right, rightSmallerCounts);
}

class SpecialBST {
  constructor(value, idx, numSmallerAtInsertTime) {
    this.value = value;
    this.idx = idx;
    this.numSmallerAtInsertTime = numSmallerAtInsertTime;
    this.leftSubtreeSize = 0;
    this.left = null;
    this.right = null;
  }

  insert(value, idx, numSmallerAtInsertTime = 0) {
    if (value < this.value) {
      this.leftSubtreeSize++;
      if (this.left === null) {
        this.left = new SpecialBST(value, idx, numSmallerAtInsertTime);
      } else {
        this.left.insert(value, idx, numSmallerAtInsertTime);
      }
    } else {
      numSmallerAtInsertTime += this.leftSubtreeSize;
      if (value > this.value) numSmallerAtInsertTime++;
      if (this.right === null) {
        this.right = new SpecialBST(value, idx, numSmallerAtInsertTime);
      } else {
        this.right.insert(value, idx, numSmallerAtInsertTime);
      }
    }
  }
}

exports.rightSmallerThan = rightSmallerThan;
```