AlgoExpert   Quad Layout   00:00:00 | 00:00:00

Prompt   Scratchpad   Our Solution(s)   Video Explanation

Difficulty: ■ **Category:** Sorting **Successful Submissions:** 1,425+

Count Inversions

Write a function that takes in an array of integers and returns the number of inversions in the array. An inversion occurs if for any valid indices `i` and `j`, `i < j` and `array[i] > array[j]`.

For example, given `array = [3, 4, 1, 2]`, there are `4` inversions. The following pairs of indices represent inversions: `[0, 2], [0, 3], [1, 2], [1, 3]`.

Intuitively, the number of inversions is a measure of how unsorted the array is.

Sample Input

```
array = [2, 3, 3, 1, 9, 5, 6]
```

Sample Output

```
5
// The following pairs of indices represent inversions:
// [0, 3], [1, 3], [2, 3], [4, 5], [4, 6]
```

Hints

Hint 1

The brute-force approach to solve this problem is to simply compare every pair of indices in the array and to determine how many of them represent inversions. This approach takes `O(n^2)` time, where n is the length of the array. Can you do better than this?

Hint 2

If the number of inversions is the degree to which the array is unsorted, and it if it takes `O(nlog(n))` time to sort an array using an optimal sorting algorithm, can you determine how unsorted the array is with a solution that runs in that time complexity?

Hint 3

Try thinking about how you would solve this problem if, instead of being given one array, you were given two separate arrays representing the main array's two halves. You would need to determine the number of inversions in the array created by merging the left array and the right array. The number of inversions in this example is actually equal to the number of inversions in the left array, the number of inversions in the right array, **and** the number of inversions when you merge the **sorted** left array and the **sorted** right array. Recall how Merge Sort works for a hint about how you can solve this problem.

Hint 4

Once you understand the information stated in Hint #3, you can use an algorithm that's very similar to Merge Sort to determine the number of inversions in any array. You'll recursively determine the number of inversions in the left and right halves of an array while sorting both the left and right halves, just like you do in Merge Sort. Once your two halves are sorted, you'll merge them together and count the number of inversions in the merged array. Take the example of these two sorted arrays: `a1 = [1, 3, 4]` and `a2 = [2, 2, 5]`. When you merge these two sorted arrays, you insert elements from the left and right array into one larger array. Whenever you insert an element from the right array before inserting an element from the left array, that means an inversion or multiple inversions have occurred. This is because elements in the right array are positioned after all elements in the left array (if these two arrays were originally left and right halves of another array). The remaining elements to be inserted from the left array when we insert an element from the right array are all inverted with this right-array element. See the Conceptual Overview section of this question's video explanation for a more in-depth explanation.

Optimal Space & Time Complexity

O(nlogn) time | O(n) space - where n is the length of the array