Inversion Count using Merge Sort

An **inversion** in an array is a pair of elements (arr[i], arr[j]) such that: - i < j - arr[i] > arr[j] This problem can be solved efficiently using a **modified merge sort** approach. **Idea:** - During the merge step of merge sort, if an element from the right half is placed before an element from the left half, it indicates that all remaining elements in the left half form inversions with this element. - Thus, inversion count is updated as (mid - left + 1). **Time Complexity:** O(n log n) **Space Complexity:** O(n)

Python Code:

```
class Solution:
    def merge(self, arr, low, mid, high):
        count = 0
        temp = []
        left = low
        right = mid + 1

        while left <= mid and right <= high:
            if arr[left] <= arr[right]:
                temp.append(arr[left])
                left += 1
            else:
                temp.append(arr[right])
                count += (mid - left + 1)  # inversion count
                right += 1

        while left <= mid:
            temp.append(arr[left])
            left += 1

        while right <= high:
            temp.append(arr[right])
            right += 1

        for i in range(len(temp)):
            arr[low + i] = temp[i]

        return count

    def mergeSort(self, arr, low, high):
        count = 0
        if low < high:
            mid = (low + high) // 2
            count += self.mergeSort(arr, low, mid)
            count += self.mergeSort(arr, mid + 1, high)
            count += self.merge(arr, low, mid, high)
        return count

    def inversionCount(self, arr):
        return self.mergeSort(arr, 0, len(arr) - 1)
```