Runtime Analysis

Problem 1: Bunny Pairings

If we sort and run a binary search on the Males, wherever the Binary Search stops can tell us how many males are larger than the female. All we need to do is see how many elements are to the right of the index that Binary Search stops at. We do have to make sure that there aren't any duplicates on the right, however, so if we notice that where we stop is the same as what's to the right of it, we continue the search. We read in the input with a O(n) for loop, sort the array in O(n log n) time, and the binary search is O(log n), giving us a runtime of O(n + nlogn + logn) = O(n log n).

Problem 2: Counting Inversions

Tracing through the problem on a whiteboard and using the "mergesort" hint, I noticed a pattern. Every time we would merge 2 arrays, any time an element from the right array would go before an element in the left array, this was an inversion. For every element not already merged in the left array, we add to the inversion count. We have O(n) runtime for the loop reading in the input, and O(n log n) for the mergesort. O(n + nlogn) = O(n log n). The runtime is O(n log n).

Problem 3: Bunny Escape Plan

An easy way of counting the total distance traveled is keeping track of how many bunnies are at each x. We then add the total number of bunnies at x to the sum when it rains at x, and then move all bunnies at x to x + 1. This way, we can sweep through the array and achieve a runtime of O(n) per simulation. While I nearly found the O(n) solution, I was unable to get it to consistently put out correct answers. I kept it in to show the work that I did, but the equation that I was using was giving me the wrong answer for the first storm in the first sample test case. We have O(n) runtime for the loop reading in the input, O(n) simulations, and at worst O(n) loop for calculating the movement. O(n + n^2). This gives us a runtime of O(n^2).

Problem 4: Counting Peaks

I made a modification to Triplets to solve this problem. Because we are looking for smaller numbers on both the left and right, we want to calculate the prefix tree and suffix tree in the same way, instead of removing values from the suffix tree. We have an O(n) loop to read in input, and countPeaks calls O(log n) FenwickTree methods n times, giving us a runtime of O(n + nlogn) = O(n log n).