Jordan Allard
CSCI 264-01
Homework 2

HW2 Problem 3 Writeup

a) Verbal Description

This algorithm calculates the weighted count of inversions for a given sequence by using a modified version of MergeSort. Similar to MergeSort, the algorithm recursively divides the array into partitions and counts the weight of the inversions for each partition, then adds them all together to get the total. The algorithm calculates the weight of inversions during the Merge portion of MergeSort, when the values of two sorted arrays are compared, by keeping track of the number and sum of all remaining values in the left array, then adding that sum and the product of the number and the right value to the total inversion count.

b) Pseudocode

```
CountInversions(nums, startIndex, endIndex):
      If startIndex = endIndex: return 0
      Let midIndex = (startIndex + endIndex) / 2
      Let leftCount = CountInversions(nums, startIndex, midIndex)
      Let rightCount = CountInversions(nums, midIndex + 1, endIndex)
      Let midCount = CountMidInversions(nums, startIndex, midIndex, endIndex)
      Return leftCount + rightCount + midCount

CountMidInversions(nums, startIndex, midIndex, endIndex):
      Let tempLeftArray = nums[startIndex, . . . , midIndex]
      Let leftSum = the sum of all elements in tempLeftArray (can be
calculated while filling tempLeftArray)
      Let tempRightArray = nums[midIndex + 1, . . . , endIndex]
      i = 0, j = 0, k = 0 midCount = 0
      While i < tempLeftArray.length and j < tempRightArray.length:
            If tempLeftArray[i] <= tempRightArray[j]:
                  nums[k] = tempLeftArray[i]
                  leftSum -= tempLeftArray[i]
                  i++, k++
            Else:
                  nums[k] = tempRightArray[j]
                  midCount += leftSum + (tempLeftArray.length -
i)*(tempRightArray[j])
                  j++, k++
      While i < tempLeftArray.length:
            nums[k] = tempLeftArray[i]
            k++, i++
      While j < tempRightArray.length:
            nums[k] = tempRightArray[j]
            k++, j++
      Return midCount
```

c) Proof of Correctness

When comparing two values from two sorted arrays, if the value from the right array is less than the value in the left array, that pair of values and the pairs made from the current right value and all following left values are guaranteed to be inversions. For example, when given two arrays [1, 3, 5, 7] and [2, 4, 6, 8], the algorithm will find that 2 is less than 3, meaning that (3, 2), and the pairs that follow, (5, 2) and (7, 2), must be inversions. The weighted count of these inversions is (3 + 2) + (5 + 2) + (7 + 2), which can be written as (3 + 5 + 7) + (2 + 2 + 2) or (3 + 5 + 7) + (3 * 2), which is the sum of all remaining left values and the right value multiplied by the number of remaining left values. This pattern

holds true each time the right value is less than the left value. In this example, the other inversions would be (5 + 4) + (7 + 4) = (5 + 7) + (2 * 4) and (7 + 6) = 7 + (1 * 6). Therefore, by keeping track of the number and sum of all remaining left values while comparing values from the left and right arrays, the algorithm will alwasy be able to correctly calculate the weighted count of inversions for that pairing.

Since MergeSort is guaranteed to compare every pair where right value has a possibility of being smaller than the left value, adding the weight calculation to the Merge comparison guarantees that the weight will be calculated for every inversion, so the total will be accurate.

d) Running Time Estimate

O(n log n)

e) Running Time Estimate Reasoning

In words: the running time of MergeSort is established to be O(n log n), and since the added calculations of this algorithm are all O(1) complexity, the overall time complexity remains unchanged.

Using the Master Theorem:

T(n) = 2T(n/2) + O(n)

a = 2, b = 2, f(n) = n

log 2 / log 2 = 1 --> f(n) = O(n^1) --> Case 2 applies

Thus, T(n) = O(n^1 log n) = O(n log n)