CS350 Homework 3

Due 10/24 @ 11:59

Problem 1

a. What might be an instance in which Shellsort would be preferable to Insertion Sort? Why?
Shell sort has an early stopping condition on its inner loop. This means it will rarely compare every element against every other element. This is particularly useful in an already sorted array but will also help in a random array. It also moves values greater distances allowing it to skip over many comparisons. This is extremely helpful in a reversed array.

b. What might be an instance in which Shellsort would be worse than Insertion Sort? Why?
Due to the early stopping condition and moving elements greater distances with each comparison, there is no array that will be faster with insertion sort.

c. The program included tests these hypotheses and finds they hold in the actual implementation.

Problem 2 Inversion algorithm and analysis.

```
INVERSION(A[n])
 1  copyA[n]toB[n]
 2  inversion = 0
 3  Mergesort(B(n), inversions)
 4       copyB[0...(n/2) − 1]C[0...(n/2 − 1)]
 5       copyB[(n/2)...n − 1]toD[0...(n/2 − 1)]
 6       Mergesort(C)
 7       Mergesort(D)
 8       inversions+ = Merge(B, C, D)
 9       return invserions
10  Merge(B[0...p + q + 1], C[0...p − 1], D[0...q − 1])
11       i = 0; j = 0; k = 0; inversions = 0
12       Whilei < len(C)andj < len(D)
13            ifC[i] < D[j]
14                 B[k] = C[i]
15                 i = i + 1
16            else
17                 B[k] = D[j]
18                 j = j + 1
19                 inversions+ = p − i The remaining elements in C are all greater than D[j]
20            k = k + 1
21       ifi == len(C)
22            copy the rest of D to B
23       else
24       copy the rest of C to B
25  return inversions
```

The general idea behind this algorithm is to count the number of swaps in merge sort. If C is the lower array and D is the higher array, every time an element in D is less than the next element in C, that is an inversion. Since C and D are both sorted, we know that every time an element in D is less than the next element in C, we also know it is less than all the elements in C. Thus we need to count each of those as one inversion.

Merge sort is an O(nlg(n)) algorithm so it is necessary to examine if we changed that. The only lines added that could effect this are lines 1 and 19. 19 does not supersede the basic operation for Merge which is the comparison on line 13, so it will not change the asymptotic complexity. Line 1 is a copy which is always going to be O(n) since it simply loops through the elements once and copies them. This means the complexity is n + nlg(n) and nlg(n) will dominate n. Thus the algorithm is O(nlg(n)).