Evan Zhang, z5383657

Question 3

3.1

The pairs are (2,1), (2,3) and (1,3).

Max of A[2,1] = 2, A[2,1,3] = 3 and A[1,3] = 3.

f(A) = 2 + 3 + 3 = 7.

3.2

J cannot equal n.

A[i] must be the max of A[i..j] and then the max of A[j+1]. For the current i's g(i,j) to increase, it will also need to be the max of A[j+2] and so on.

3.3

If j != 1 and j != n we can start to work backwards from A[j-1] to find each g(i,j).

First, find the maximum value of A[i](i.e. A[j-1]) and A[j] (the pair forms the current A[i..j]).

If A[j] is the max of the current A[i..j], set A[j] as max(A[i..j]) and set current g(i,j) = 0.

If A[i] is the max of the current A[i..j], set A[i] as max(A[i..j]) and set current g(i,j) = previous max's g(i,j) (if previous max was A[j], set to zero instead).

Then compare in sequence the numbers after A[j] (i.e. A[j..n]) with max(A[i..j]) until max(A[i..j]) is found to be smaller. Set this number as the starting point for the next max. For each number smaller than the current max, add 1 to the g(i,j). Next time, start from the set starting point instead of A[j+1].

Now, minus 1 from i and repeat until all values of i < j are completed.

Each element before A[j] and after A[j] is visited once in this algorithm and steps carried out at each one is in O(1) time. Therefore, this algorithm will be completed in O(n) time.

3.4

Begin the mergesort algorithm by splitting the n length array into n pieces. When merging/sorting at each step, count the number of numbers already inserted into the new array and pair them with the number which is being sorted into the new array. Add them if a pair already exists. Do not repeat this with previously sorted

Since mergesort is completed in O(nlogn) time and at each step of mergesort, only O(1) time additional commands are carried out, the total running time will be O(nlogn).