(2) **(Counting flips)** (20 points) A *flip* in an array $A[1:n]$ of numbers is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$. In other words, a flip is a pair of elements that are not in sorted order.

(a) (10 points) Find an array of length n over the elements $\{1, \ldots, n\}$ that maximizes the number of flips, and prove that it is optimal.

(b) (10 points) Suppose you run *insertion sort* on an array of length n that has k flips. Derive a tight big-O-bound on the running time of insertion sort as a function of both n and k, and explain your analysis.

(Note: Recall that insertion sort processes increasingly longer prefixes of the input array. After having sorted prefix $A[1:i]$, it sorts prefix $A[1:i+1]$ by inserting element $A[i+1]$ into its correct sorted position in $A[1:i]$.)

(c) **(bonus)** (10 points) By modifying *merge sort*, design an algorithm that *counts* the number of flips in an array of length n in $O(n \log n)$ time.

(Note: There can be $\Theta(n^2)$ flips in an n-element array, so your algorithm cannot explicitly list them and achieve the time bound. It is possible to count flips without listing them.)

a) An array that maximizes the number of flips would be one that is in decreasing order (a reverse sorted array) so for any given i and any other j that satisfies $i < j$ then $A[i] \geq A[j]$ and the number of flips for any given i would be $n-i$ and for the entire array there would be n^2 flips. We can prove this through contradiction by assuming the list isn't in decreasing order and that not all $A[i] \geq A[j]$ therefore there wouldn't be the maximum number of flips because there would be at least one instance where $A[i] < A[j]$
For an array that is reversed order the number of flips would be $(n-1)$ (inversions for the first element) + $(n-2)$ (inversions for second element) + $(n-3) \ldots + (2) + (1)$ which would be $\sum_{i=1}^{n-1} i$. If we had an other array that wasn't completely in reverse order then we would have a number of flips less than $\sum_{i=1}^{n-1} i$, therefore the array in complete reverse order must have the max number of flips

b) Insertion sort's asymptotic runtime consists of the time to make each comparison plus the time for each swap. In the case of insertion sort the number of comparisons would be $n+k$. Insertion sort must look at every element and compare it to the previous element at least once so we have n comparisons so far. Then, for any given element it will need to make a number of comparisons equal to the number of inversions specific to that element. For the entire array the number of inversions will be k so the time complexity for comparisons will be $O(n+k)$. The time complexity for swaps will be $O(k)$ since k swaps will need to be done to put the list in order. So, the asymptotic runtime will be $O(n+k) + O(k)$ (comparisons + swaps) which would reduce to $O(n+2k)$ and we can drop the constant infront of k to get $O(n+k)$. Since n and k are variables we can leave the time complexity in terms of both n and k since we don't know which one will be the dominating factor.

c) In the last portion of the mergesort algorithm there is a section that merges the two halves together. At this point we don't want to sort while we merge but rather compare every element in the first half will be compared to every element in the second half and if the first element is larger than the second element (indicating a flip) the a count variable is increased by one. Once all elements in the first half have been compared to all elements of the second half then return the newly found count. Once the algorithm returns to the original function call and the recursive calls have returned their count of flips a count variable will hold the number flips found throughout the array.