Due at the start of class, Tuesday, September 21.

**Problem 1.** Assume that you have a list of \( n \) elements where you know exactly \( k \) of them are out of place, in the sense that if those \( k \) elements are removed the list is sorted (and if fewer than \( k \) elements are removed the list is not sorted).

(a) How many comparisons does bubble sort use (as functions of \( n \) and \( k \)):
   (i) In the best case?
   (ii) In the worst case?
   (iii) In the average case?

(b) How many exchanges does bubble sort use (as functions of \( n \) and \( k \)):
   (i) In the best case?
   (ii) In the worst case?
   (iii) In the average case?

**Problem 2.** Repeat Problem 1 for the “modified” bubble sort (where the algorithm ends an iteration at the last location where the previous iteration did an exchange, if any exchanges occurred). Do not worry about average number of comparisons (i.e., do not do part a.iii).

**Problem 3.** Repeat Problem 1 for insertion sort (using the version of insertion sort with a sentinel), but do not worry about average number of moves (i.e., do not do part b.iii).

**Problem 4.** Repeat Problem 1 for selection sort.

**NOTE 1:** Make all of your analyses as exact as possible.

**NOTE 2:** Average case analyses can be difficult.

(a) You can modify the definition of what it means for there to be \( k \) elements out of place. For example, you might want to define average case in a way that on average \( k \) elements are out of place. Or you might want to assume each element is randomly picked, and then returned to a random location, so there is a small chance that it is returned to its original location. In any case, you should state what you mean by average case, but you do not have to formalize it.

(b) You can assume \( k \) is small (relative to \( n \)).

(c) You can get just the high order term right.