1. One way to find the median of a list is to sort the list and then take the middle element.
   (a) Assume you use Bubble Sort on 15 elements. Exactly how many comparisons do you use (in the worst case)?
   (b) Assume you use Mergesort on 15 elements. Exactly how many comparisons do you use (in the worst case)?

2. You can actually find the median by running a sorting algorithm and stopping early, as soon as you know the median.
   (a) Assume you use Bubble Sort to find the median of 15 elements, but stop as soon as you know the median. Exactly how many comparisons do you use (in the worst case)?
   (b) Assume you use Mergesort to find the median of 15 elements, but stop as soon as you know the median. Exactly how many comparisons do you use (in the worst case)?

3. It turns out that you can find the Median of 15 elements with 28 comparisons (although there may be better algorithms). You can use this information to develop a (worst case) linear time Selection algorithm based on columns of size 15, rather than the columns of size 5 that we used in class.
   (a) Using columns of size 15 exactly how far from either end of the array is the median of medians guaranteed to be. Just give the high order term. (Recall that with columns of size 5 we got $\frac{3n}{10}$.)
   (b) Write down the recurrence for a Selection algorithm based on columns with 15 elements each. (You can ignore floors and ceilings, as we did in class.) You do not have to give the algorithm, but state where each of the terms in your recurrence comes from. (For example, you might say that the $n-1$ term comes from partition.)
   (c) Solve the recurrence, and give the high order term exactly.

4. Assume you have an algorithm that finds the median of n elements in $cn \lg \lg n - n$ comparison steps (for some constant c).
   (a) Give an efficient (recursive) algorithm for selection based on this. It will, of course, not be linear time.
   (b) Write a recurrence for the number of comparisons your algorithm uses.
   (c) Solve the recurrence using constructive induction. Just get the high order term exactly.
   (d) Why might it be a good algorithm despite not being linear time?