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 to sort a list with 13 elements (i.e. $n = 13$). Exactly how many comparisons do you use (in the worst case)?
   (b) Assume you use Mergesort to sort a list with 13 elements (i.e. $n = 13$). 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 13 elements (i.e. $n = 13$), 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 13 elements (i.e. $n = 13$), but stop as soon as you know the median. Exactly how many comparisons do you use (in the worst case)?

3. The selection algorithm (to find the *kth* smallest value in a list), described in the class (and in the book), uses columns of size 5. Assume you implement the same selection algorithm using columns of size 13, rather than 5.
   (a) 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) It turns out that there is an algorithm that finds the median of 13 elements with 23 comparisons. Using this algorithm, briefly list each step of Selection with columns of size 13 and how many comparisons the step takes.
   (c) Write a recurrence for the number of comparisons the algorithm uses.
   (d) Solve the recurrence using constructive induction. Just get the high order term exactly.
   (e) How does this value compare to what we got in class with columns of size 5?

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?