1. Assume your machine has 64 bit words. Assume you can multiply two n word numbers in time $2n^2$ with a standard algorithm. Assume you can multiply two n word numbers in time $15n^{\lg 3}$ with a "fancy" algorithm.
   (a) Approximately, how large does n have to be for the fancy algorithm to be better?
   (b) How many bits is that?
   (c) How many decimal digits is that?

2. Use the same assumptions as for problem (1), except assume you can multiply two n word numbers in time only $5n^{\lg 3}$ with a "fancy" algorithm.
   (a) Approximately, how large does n have to be for the fancy algorithm to be better?
   (b) How many bits is that?
   (c) How many decimal digits is that?

3. Consider an array of size nine with the numbers in the following order $40, 20, 80, 60, 30, 90, 10, 70, 50$.
   (a) Form the heap using the algorithm described in class. Show the heap as a tree. Show the heap as an array. Exactly how many comparisons did heap creation use?
   (b) Start with the heap created in Part (a). Show the *array* after each element sifts down *after heap creation*. How many comparisons does each sift use? What is the total number of comparisons *after heap creation*?

4. Assume that your machine has a built-in data structure that inserts an item into a priority queue in one step. There is no charge to remove an item. It has both a min-priority queue and a max-priority queue available. You can access the largest element for one comparison step in a max-priority queue, and similarly you can access the smallest element for one comparison step in a min-priority queue.

   This machine can sort a list in linear time: Insert all of the elements into a priority queue and then remove them (one at a time). This is n queue inserts and no comparisons, for list of size n.

   What if your priority queue is restricted to having size s? You can assume that your algorithm may have many different priority queues (but each can have size at most s). You can distinguish elements in the priority queue (perhaps by their original index in the list).

   (a) i. Design an efficient algorithm based on merge sort. You may describe it in high level English or in pseudo code, but the algorithm must be clear.
   ii. Analyze its time: Give the number queue inserts and number of comparisons as precisely as reasonably possible. At least give the high order term exactly. Your answer should be a function of the list size, n, and the queue size, s.

   (b) i. Design an efficient algorithm based on heapsort. You may describe it in high level English or in pseudo code, but the algorithm must be clear.
   ii. Analyze its time: Give the number queue inserts and number of comparisons as precisely as reasonably possible. At least give the high order term exactly. Your answer should be a function of the list size, n, and the queue size, s.