Due by the beginning of class, Sept. 24.

1. Suppose that you are given an k-*sorted* array, in which no element is farther than k positions away from its final (sorted) position. Give an algorithm which will sort such an array. Prove its correctness. Analyse its running time. Note: your algorithm should run faster than $\Theta(n \log n)$, that is, it should take advantage of the fact that the array is almost sorted.

2. Consider once again the k-sorted array of the previous problem. Show that any comparison based algorithm for sorting an almost sorted array makes $\Omega(n \log k)$ comparisons.

3. A *Weight Balanced Search Tree* maintains the following invariant. For each node u,

$$\text{weight(heavier child)} \leq 2\ \text{weight(lighter child)}.$$

   (Recall that the weight of a node is the number of descendants.)

   (a) Show that the tree can be searched in $\mathcal{O}(\log n)$ time.
   (b) Suppose that on an update, whenever a node goes out of balance, you rebuild the entire subtree rooted at that node. What is the cost of an update?
   (c) How many insertions are needed to unbalance a freshly balanced tree?
   (d) What is the cost of n insertions? (Hint: an insertion can contribute to the unbalancing of more than one node. How many nodes can an insertion help unbalance?)
   (e) What is the amortized cost of an insertion?

4. Consider a rectangular array of distinct elements. Each row is given in increasing sorted order. Now sort (in place) the elements in each column into increasing order. Prove that the elements in each row remain sorted.

5. Show that if you have a subroutine for $HamP$ which runs in polynomial time, you can solve the $HamC$ problem in polynomial time.

6. Show that if you have a subroutine for $HamP$ which runs in polynomial time, you can actually *find* a hamiltonian path in polynomial time.