This homework is to be done in groups of 3 (or 2 if necessary), students may select their own groups.

There are additional “Problems not to be turned in” and reading assignments will be updated on the class web page.

1. (10 pts) Use an information theoretic (decision tree) argument to show that at least $2n - o(n)$ comparisons are required to merge two sorted lists $A$ and $B$ each containing $n$ elements. (Hint: First, get a good lower bound on the number of ways that two $n$-element lists with distinct elements can be merged, you may need to approximate a binomial coefficient. My solution shows a lower bound of $2n - O(\log n)$.)

2. (10 pts) The external path length of tree $T$, written $e(T)$, is the sum taken over all leaves of the depths of the leaves. Use induction to show that the external path length of every (non-empty) full binary tree $T$ is at least $\ell(T) \log \ell(T)$ where $\ell(T)$ is the number of leaves in $T$. (Hint: In the inductive step, find the two subtrees $T_L$ and $T_R$ of $T$ and show how to apply the inductive hypotheses to these trees to get a lower bound on the total depth of the leaves in $T$ in terms of how many leaves are in $T_L$. Then use calculus to minimize this bound and underestimate the total depth of the leaves.)

Five additional problems (not to be turned in)

1. Analyze the running time of the following Stooge Sort algorithm.

   ```
   Algorithm Stooge Sort (A, p, r):
   if $p + 1 \geq r$ then return;
   $t = \lfloor(r - p + 1)/3 \rfloor$ one-third of the number of elements
   Stooge Sort(A, p, r - t);
   Stooge Sort(A, p + t, r);
   Stooge Sort(A, p, r - t);
   ```

2. Prove that for all array sizes $n$, Stooge Sort $(A, 1, n)$ correctly sorts the array $A[1..n]$.

3. Problem 7-4 on page 188 (problem 7-4 on page 162 of 2nd edition) Quick sort stack depth

4. Problem 6-2 on page 167 (problem 6-2 on page 143 of 2nd ed) d-ary heaps.

5. Simulate by hand:

   - the heap operations (like exercises 6.3-1 and 6.5-1)
   - the partition algorithm (like exercise 7.1-1)
   - radix sort (like exercise 8.3-1)