Show your work—to me, not your neighbor!

1. Starting or ending with the first letter in your name (first or last), circle seven consecutive letters in the following:

   FUNALGORMSYEBZCQDPKWXJV

   (For example, someone with the initials BW could go from T to B, from B to K, or from Z to W)

   Consider a seven-entry array that has your seven letters as entries.

   a) How many swaps would insertion sort perform to sort that array in ascending order? (Your answer should be a number, not a function). Is this number higher, lower or the same as the average case behavior for insertion sort on a random array of 7 letters?

   b) Start over with the same seven-entry array (before sorting). This time we will sort it with the bottom-up version of heapsort. Show the contents of the array one swap after heapsort first builds the heap. (You can show your work on the reverse side of this page.)

2. What is a sentinel? How did we implement mergesort to avoid the use of a sentinel?

3. Show the contents of an array whose initial entries are EASYQUESTION just after the N is swapped for the first time by quicksort.

4. Give the complexities of the following. Express your answer as $\Theta(f(n))$ for some function $f(n)$.

   a) The worst-case number of comparisons for selection sort.

   b) The worst-case number of swaps for selection sort.
c) The worst-case number of comparisons for quicksort.

d) The average-case number of comparisons for quicksort.

5. Let $f(n) =$ the number of decimal digits in $n!$ and $g(n) =$ the number of decimal digits in $n^n$. (For example, $4! = 24$ has 2 digits so $f(4) = 2$.) Circle the following relations which are true. (No explanation necessary, but one could get you partial credit if you’re wrong.)

\[
\begin{align*}
f(n) &= O(g(n)) \quad f(n) = o(g(n)) \quad f(n) = \Omega(g(n)) \quad f(n) = \omega(g(n)) \quad f(n) = \Theta(g(n))
\end{align*}
\]

6. Using the master theorem, determine the complexity of the solutions to the following recurrences. Show enough work so that I can follow your reasoning.

   a) $T(n) = 1$ for $n < 2$; $T(n) = 3T(n/2) + 6n^2$ for $n \geq 2$;
   
   b) $T(n) = 1$ for $n < 2$; $T(n) = 4T(n/2) + 7n^2$ for $n \geq 2$;
   
   c) $T(n) = 1$ for $n < 2$; $T(n) = 5T(n/2) + 8n^2$ for $n \geq 2$;

7. Determine the complexity of the solution of the following recurrence. An answer of the form $\Theta(f(n))$ is best, but $O$ or $\Omega$ answers are better than nothing. Justify your answer.

\[
T(n) = 1 \text{ for } n < 2; \quad T(n) = T(\lg n) + n \text{ for } n \geq 2.
\]

8. Consider the problem of finding the smallest and second smallest item in an array of $n$ entries. One algorithm is to run two passes of selection sort, passing through the array once to find the minimum, swapping it to the front, and passing through a second time (starting after the first entry) to find the second smallest.

Another algorithm would be the form a tournament to determine the smallest. Within each successive pair, the smaller letter moves on to the next round. For example, if the array had values \texttt{FINAL QUESTION}, the tournament would be:
What this means is first F and I are compared, with F being moved to an appropriate location, then N and A, etc. Then the next time through F and A are compared with A winning and moving on. Note that in any round, when there is an odd number of entries, the last one moves forward with no comparison. After the tournament is over, the smallest element is the winner. Then consider all the entries that lost to the winner. (In our example, N, F, E, and I.) Find the minimum of those to produce the second smallest entry.

a) Why does this always work?

b) How many comparisons (as a function of \( n \)) are performed in the tournament to determine the smallest letter? Give an exact answer. Compare this the number of comparisons on the first pass of selection sort.

c) After the tournament, as a function of \( n \), roughly how many more comparisons are necessary to determine the second largest. Express your answer as \( \Theta(f(n)) \) for some function \( f(n) \). Compare the complexity with the number of comparisons on the second pass of selection sort.

d) Considering all comparisons, how do the complexities of the two algorithms compare? (Use \( O \), \( \Omega \), or \( \Theta \).)