CS361 Sample Midterm

Note: more questions than an actual midterm.

1. Recall the definition of little-o: namely, $f = o(g)$ if $\lim_{n\to\infty} f(n)/g(n) = 0$. Now mark each of the following true or false. For the false ones, state whether the true relationship is Θ or not. Show your work!

$$
\begin{array}{lrcl}
a) & 2^{\sqrt{n}} & = & o(2^n) \\
b) & \log(\sqrt{n}) & = & o(\log n) \\
c) & 2^{n-1} & = & o(2^n) \\
d) & 2^{2^{n-1}} & = & o(2^{2^n})
\end{array}
$$

2. Consider the following function:

```
int f(int n) {
  if (n < 1) return 0;
  else {
    int val;
    val = n;
    val += f(n/4);
    val += f(n/4);
    return val;
  }
}
```

a) (5 points) Write down the recurrence relation for the *value returned by f*.

b) (10 points) Solve this recurrence within Θ.

c) (5 points) Suppose $f(n)$ is a constant A times your Θ-solution from part (b). (This might not be exact for small n, but that's OK.) Substitute this solution into your recurrence from part (a), and solve for A.

d) (5 points) Now write down the recurrence relation for the *running time of f*.

e) (10 points) Now solve this other recurrence within Θ.

f) (10 points) Now suppose we *memoize* the function by storing each $f(n)$ we calculate in a big array called alreadyDidIt[n], so any time we want to know $f(n)$ for some value of n we've already done, we can get $f(n)$ in $O(1)$ time. If I start with this array empty, so that I'm running f for the first time, what is the recurrence relation for f's running time now?

g) (5 points) What is the solution to this new recurrence within Θ?

3. We saw in class that I can "Heapify" a set of n items into a heap in $\Theta(n)$ time by calling downSift() on each one, starting with the items at the bottom of the heap and working my way up. I can also Heapify this by calling upSift() on each one, starting with the root. Within Θ, how long will this alternate procedure take?

4. Here is pseudocode for an iterative version of Selection Sort:

```
ssort(a[1...n]) {
  for i = 1 to n-1 {
    scan a[j] for j=i to n and find the location j of the smallest element
    if (j != i) swap a[j] with a[i]
  }
}
```

   (a) (10 points) Define a loop invariant for `ssort`. What kind of "progress" can we guarantee has been made by the time we have gone through the loop i times?

   (b) (10 points) Now prove that `ssort` works by showing inductively that the loop invariant holds for each value of i in the outer loop. Remember to include the base case, the inductive step, and termination (note the final value of i).

   (c) (10 points) Rewrite `ssort` as a recursive algorithm, i.e., without this outer `for` loop. On what part of the list does it call itself?

5. Suppose we want an algorithm that takes a list of n numbers and returns both their minimum and their maximum. Consider a divide-and-conquer strategy. We divide the list into two equal halves, and (recursively) find the minimum and maximum of each half. Then, the minimum of the whole list is the smaller of the two minima, and the maximum of the whole list is the larger of the two maxima. Finally, if the list is of size 2, a single comparison finds both the minimum and the maximum.

   For example, on the list $\{2, 5, 1, 9, 3, 6, 0, 7\}$, first we learn (by calling the algorithm recursively) that $\{2, 5, 1, 9\}$ has minimum 1 and maximum 9, and $\{3, 6, 0, 7\}$ has minimum 0 and maximum 7. Then comparing 0 and 1 gives the minimum 0, and comparing 9 and 7 gives the maximum 9.

   (a) (10 points) Let $f(n)$ be the number of comparisons this algorithm does for a list of size n. Write down the recurrence relation, and a base case, for $f(n)$.

   (b) (10 points) Solve this recurrence within Θ.

   (c) (10 points) Solve this recurrence and base case exactly, assuming that n is a power of 2. To do this, try multiplying your Θ-solution by A and adding a constant B. Plugging the resulting form into the recurrence and base case gives two equations for the two unknowns A and B.

6. Recall that a 2-3-4 tree is one where every node (except for the leaves) has 2, 3, or 4 children, and where all the leaves are at the same depth. Let q be the fraction of the nodes that are leaves. When the total number of nodes n is large, what are the largest and smallest possible values for q?

7. Recall that each node of a skip list has a list of pointers `node *next[height]` that point to nodes farther down the list. When we generate each node, we choose its level as follows:

```
height = 1;
while (coin flip == heads) height++;
```

   In other words, we give it an initial level of 1 and then start flipping coins; if the coin comes up "heads" we increment the level, and if it comes up "tails" we stop.

   Calculate *exactly* the average total storage space we need for all the `next` arrays of a skip list with n nodes — that is, n times the average total height of all n nodes. Hint: what is the average height of a single node?