1. Exercise 6.5-8 (page 166). Note that the problem asks you to give an algorithm that runs in $O(\log n)$ time. So you not only have to give the algorithm, you also have to show that it really does run in $O(\log n)$ time.

2. Exercise 6.5-9 (page 166).

3. Exercise 6.1 in the Lecture 3 handout (on page 12 of the handout).

4. Exercise 7.3-2 (page 180).

5. Problem 7-4 (page 188). This requires careful explanation. It’s the kind of reasoning that is very important in software engineering.

6. Prove that if a binary tree has height $n$, then it has at most $2^n$ leaves.

   Please be careful about this. The result is not “obvious”. And you certainly can’t assume that the tree is “all filled out”\(^1\), or that a tree that is “all filled out” has the maximum possible number of leaves. That’s in fact what you are trying to prove, so you can’t assume it is true.

   A good way to prove this result is by induction:

   (a) The inductive hypothesis should be a sequence of statements, one for each possible height of the tree. For instance,

   \[
   \bullet \text{If a binary tree has height 5, then it has } \ldots
   \]

   Write down a few of these statements, each one completely (i.e., without “…”).

   (b) Now finish writing the proof by induction, using these statements as the inductive hypothesis. The proof is not hard, but you really have to be careful.

   And let me just say it once again: you can \emph{not} assume that any of the trees you are considering is “filled out completely”. If you do, the proof is automatically wrong, and I won’t even read it.

---

\(^1\)Our text calls a binary tree that is “all filled out” a \emph{complete} binary tree. However, this term is not entirely standard.