CS 624: Analysis of Algorithmns

Assignment 2

Due: Thursday, October 3, 2019

Questions 1–3 are copied here from HW1.

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

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

3. Prove that if a binary tree has depth 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", 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 depth of the tree. For instance,

      If a binary tree has depth 5, then it has ...

      ...

   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 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.

4. Exercise 6.5-7 (page 166).

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

6. Exercise 3.1 from the Lecture 4 handout (page 7).

7. Suppose you start with a rectangular array of numbers. Perform the following operations:

   First sort each row (smallest to largest).

   Then sort each column (smallest to largest).

   Show that after sorting the columns, each row is still sorted.

8. Assume that $c \geq 0$. Assume you had some kind of super-hardware that, when given two lists of length n that are sorted, merges them into one sorted list, and takes only n steps.

(a) Write down a recursive algorithm that uses this hardware to sort lists of length n.

(b) Write down a recurrence to describe the run time.

(c) For what values of c does this algorithm perform substantially better than $n \log n)$? Why is it highly implausible that this kind of super-hardware could exist for these values of c?