Homework 5

CS 4104 (Fall 2009)

Assigned on Monday, October 5, 2009.
Hardcopy due at the beginning of class on Monday, October 12, 2009.

Instructions:

- You are not allowed to consult any sources other than your textbook, the slides on the course web page, your own class notes, the TA, and the instructor. In particular, do not use a search engine.
- Do not forget to typeset your solutions. *Every mathematical expression must be typeset as a mathematical expression, e.g., the square of n must appear as n^2 and not as "n^2".* Students using LaTeX or LyX can use the corresponding versions of the homework problems to start entering their solutions.
- Describe your algorithms as clearly as possible. The style used in the book is fine, as long as your description is not ambiguous. Explain your algorithm in words. A step-wise description is fine. *However, if you submit detailed pseudo-code without an explanation, we will not grade your solutions.*
- Do not make any assumptions not stated in the problem. If you do make any assumptions, state them clearly, and explain why the assumption does not decrease the generality of your solution.
- Do not describe your algorithms only for a specific example you may have worked out.
- You must also provide a clear proof that your solution is correct (or a counter-example, where applicable). Type out all the statements you need to complete your proof. *You must convince us that you can write out the complete proof. You will lose points if you work out some details of the proof in your head but do not type them out in your solution.*
- Analyse your algorithm and state the running time. You will only get partial credit if your analysis is not tight, i.e., if the bound you prove for your algorithm is not the best upper bound possible.

Problem 1 (15 points) Solve the recurrence $T(n) = T(\lfloor\sqrt{n}\rfloor) + 1$. In words, the $T(n)$ is the running time of an algorithm that creates one sub-problem of the size equal to the square root of n, solves this sub-problem recursively, and spends one more unit of time. You can assume that $T(2) = 1$ and that $n \geq 2$.

Problem 2 (40 points) You are given three algorithms to solve the same problem of size n. Analyse each algorithm in $O()$ notation. Provide a clear proof of the analysis. Which algorithm would you choose and why? In other words, write down which algorithm is asymptotically the fastest and provide a proof why this algorithm is asymptotically the fastest of all three. If you can directly apply a formula we discussed in class, feel free to do so. For some sub-problems, you will have to come up with proofs from scratch, although your proofs will follow the general outlines we have used in class. Remember to prove your solution by induction, even if you use the "unrolling" method to guess a solution.

(i) Algorithm A solves the problem by dividing it into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.

(ii) Algorithm B solves the problem by dividing it into two sub-problems of size $n - 1$, recursively solving each subproblem, and then combining the solutions in constant time.

(iii) Algorithm C solves the problem by dividing it into three sub-problems of size $n/3$, recursively solving each sub-problem, and then combining the solutions in $O(n^2)$ time.

Problem 3 (45 points) Solve exercise 3 in Chapter 5 (pages 246–247) of your textbook. Let us call the equivalence class with more than $n/2$ cards the *important* class. It is enough for your algorithm to return a card that belongs to this class, if it exists, or no card at all. Note that the problem specifies that the only operation you can perform on a pair of cards is to decide if they are equivalent. You cannot perform any other operation, e.g., compare them in order to sort them. Your proof of correctness must clearly address why your algorithm will find a set of important class, if it exists. For instance, if you divide the cards into two sets of $n/2$ cards each and find the important class in each subset, why should the important class(es) found by the recursive calls have any relation to the important class for all n cards?