Algorithms Winter 2007
Assignments Guidelines Instructor: Anna Karlin

Purpose

The purpose of this document is to provide basic useful guidelines for writing the solutions of the assignments. Accordingly, the document is intended to set the standards for the submitted assignments and to make the grading process more efficient and beneficial for all of us.

Structure

The following should serve as the structure of the solutions you submit. Please refer to the last section of this document for a detailed example.

Idea: Up to two sentences that summarize the main methods/techniques/algorithmic ideas that are used to solve the problem and to prove correctness. If there is some intuition for your solution that you want to include, that is the place.

Formally: Formal description of the solution, that is, most of the times, the algorithm you are asked to provide (you may omit this part of the solution only if the question requires you to prove a given claim). If the algorithm is very simple, you can simply describe it. Else, you should use pseudo-code. You might want to clarify your pseudo-code with a short simplifying description, but please refrain from reasoning or justifying the design of your algorithm. The intuition in the **Idea** section should be sufficient reasoning in that sense, while the proof of correctness (and complexity analysis) supplies the justification for your algorithm. The length of this part is usually no more than a third of a page.

Set up good notation. Define all the variables and terms you use. Many of the exercises in the book are phrased almost entirely in English. It will be your job to rephrase them mathematically when necessary. The first few lines might look something like "Let S be the set of students taking CSEP521". Make sure that you've clearly defined any variable you use. Choosing good names for your variables is also important. This is of special importance for complicated proofs, and can make a big difference. Some conventions are highly recommended: i, j are used for indexing, u, v usually denote nodes in a graph, n, m usually stand for combinatorial parameters of the input which are later used to express the complexity of the algorithm.

Pictures/figures. You may use pictures/figures to clarify your algorithm or a term that you use, but they are not sufficient by themselves as a description of your algorithm.

Complexity: Complexity analysis of any described algorithm. You might omit this part from your answer *only* when the solution you provide is neither an algorithm nor a proof for an algorithm that is given as part of the question (but a counter example or a proof of a given claim). In any other case, you must provide the complexity analysis even if the problem description doesn't specifically ask for it. The complexity analysis also serves as a proof of the termination of the algorithm. On one hand, that means that if the termination is not trivial, a detailed proof should be provided here. On the other hand, if the complexity of the algorithm can be trivially concluded from the *syntax* of your pseudo code, it is enough to briefly state the reasoning behind the analysis and the resulted complexity. Accordingly, the length of this part is usually up to 5 lines.

Correctness: Whether the problem specifically asks for it or not, you should always prove your solution is correct. The logical structure of the proof should be clear, concise and correct, and the proof should

be well-organized according to this structure. For example: don't forget to prove the base case when using induction; prove two direction for "iff" statements (mind that equality is such a statement); and so on. Make sure you prove everything that needs to be proven. The length of the proof is usually no more than a half a page; detailing level should be set accordingly, although it is probably impossible to precisely define the required detailing level in the scope of this document. Try to use common sense and make adjustments according to the feedback you get on your assignments. Feel free to attend the office hours if you need help with this. Few more tips:

Clarity is very important. You may lose points for style.

Use what you learn. Unless the question asks you to prove a theorem that was introduced in class, you may rely on the correctness of the theorems introduced in class. Moreover, doing so is encouraged, as it saves a lot of time and demonstrates your comprehension of the course material.

Try rewriting your proofs. Writing out your answer fully on a piece of scratch paper before writing up the version to hand in really does make a difference. You are more likely to catch mistakes or exceptions, and your proof will be better organized. It is highly recommended to type your solutions; this allows you more freedom in this rewriting process. *Usually, the correct practice is to spend more time on thinking how to phrase your solution than on writing it.*

Example - Bubble Sort

Idea: Incrementally adding items to the sorted list in their right place.

Formally: Let $x_1, \ldots, x_n$ be the given list of integers to sort. For a list L, let $L[i]$ denote the ith element in the list. Let $Insert(L, j, x_i)$ be a procedure that inserts the element x_i between the $(j-1)$th and the jth elements in the list L (if there are $j-1$ elements in the list, the element x_i is simply appended at the end of the list). Consider the following algorithm:

- Initialize L to be an empty list.
- For $i = 1, \ldots, n$
  - $j \leftarrow 1$
  - While $j < i$ and $L[j] < x_i$, $j++$.
  - $Insert(L, j, x_i)$.
- Output L.

Complexity: The outer loop executes n times. Each time, the inner loop executes at most n times and there is an invocation of the procedure $Insert$. As $Insert$ can be implemented to run with time complexity that is linear in the size of the list, which is always at most n, each iteration of the outer loop takes $O(n)$ time, and hence the algorithm's time complexity is $O(n^2)$.

Correctness: We prove by induction on the iteration of the outer loop that the list L is sorted. The base case is trivial as after the first iteration the list contains only x_1. Assume that at the beginning of the ith iteration L is sorted, $i > 1$. In the ith iteration we add the element x_i to the list. The place in which it is added is determined according to the termination of the while loop. If it is add in the ith place, we must have that $x_i > L[i-1]$, which implies that x_i is inserted in its right place. Else, we insert x_i between the $(j-1)$th and the jth elements of the list L, and $x_i > L[j-1]$ and $x_i \leq L[j]$, which implies that x_i is inserted in its right place.