Advice on presenting algorithms and proofs

Vassos Hadzilacos

Many of the problems in your assignments for this course will involve designing algorithms, justifying their correctness, and analyzing their running times. In this document I provide some advice on how to present your algorithms and proofs.

(1) Before describing an algorithm in pseudocode make sure that you give an informal and brief, but also clear and helpful, description of the basic idea(s) underlying your algorithm. This can include diagrams or examples, if this helps, but should not consist only of these. It should also not be a re-telling of the pseudocode in English.

By including such a description the reader (and grader) can know what it is that you are trying to do and, as a result, perhaps be more forgiving of small errors or omissions in your pseudocode. In the absence of an overall understanding of your ideas, the best a reader can do is to follow the pseudocode line-by-line. This is not only time-consuming, but also can make it impossible to tell if an error is due to a minor slip-up or the result of a completely wrong idea.

Explaining an idea in clear terms is a difficult and time-consuming, but very useful, skill. You may have to spend more time on it than on writing the pseudocode — in fact, doing it well could also help you write better pseudocode (and actual code).

(2) Do not write actual executable code; this is too much detail. Do not even write pseudocode for straightforward tasks. For example, it is better to write things like

- sort array A in increasing order
- find the edge in E with minimum weight
- reverse the list L
- $A^+ :=$ elements in A greater than x, in the order they appear in A

than to waste your and your grader's time writing pseudocode for these things.

Of course, you should account for the running time of such high-level descriptions correctly. For example, in your running time analysis for the first of these you might say (assuming A contains n elements): "Step (a) takes $O(n \log n)$ time using, say, Heapsort."

(3) Avoid idiosyncratic notation of specific languages. For example, if B and C are lists of some sort, write

- "$A :=$ concatenate B and C"

rather than

- "$A := B + C$"

which makes sense in Python but is misleading in, say, C, or

- "$A := A$.AddAll(B); $A := A$.AddAll(C)

which makes sense in Java but is unnecessarily obscure.

(4) Do not burden the reader with tedious proofs of obvious or straightforward facts; focus on justifying essential and non-obvious facts that are relevant to your point. If an argument entails a case analysis

where essentially the same argument (modulo straightforward changes, such as changing $<$ to $>$) applies to multiple cases do not waste your, and the reader's, time by repeating the argument. It suffices to say "by a similar argument, we can show that ...", possibly with some indication of what the appropriate changes are.

(5) On the other hand, do not say "it is obvious that X" unless (a) X is true, and (b) it should indeed be obvious (to a typical third- or fourth-year computer science UofT student) that X is true. Similarly, do not say "the proof of Case A is similar to that of Case B", unless that is indeed this case. Whether X is obvious, or two arguments are similar, is not a cut-and-dried matter; there is an element of judgment that you must exercise. Of course, you may be judged on your judgment!

(6) Keep in mind that:

- Induction is not needed in ***every*** proof.
- An argument does ***not*** become a proof just by virtue of involving induction.
- Clarity ***is*** needed in every proof.

(7) Regarding clarity: The reader (and, in your case, grader) of your proof can only see what you wrote, not what you meant to write. One of the advantages of working with a partner on an assignment, is that there is a second party to check your argument and make sure that it is understandable to someone other than the writer. When you work on homework with a partner, however, do not divide the work so that one party is in charge of solving the problem and the other is in charge of checking the solution: To derive the benefit of homework, which is critical to your learning in this course, both members of the group should assume both the role of problem-solver and the role of solution-checker. The goal of working with a partner is not to lessen the burden but to improve the learning.

(8) The purpose of a proof is to establish, without any doubt, that a certain conclusion follows from certain assumptions. A good proof should illuminate, rather than befuddle. The simpler a proof, the more illuminating it is. There are many ways to present a good proof; a proof is not about format or style, but about content.

(9) If your proof involves induction, be sure to tell the reader ***what exactly*** it is that you are proving by induction.

(10) If your proof, or part of it, involves an argument by contradiction, it is helpful if you ***start*** with something like "Suppose, for contradiction, that X". This way the reader knows why you are assuming X, and therefore that you are trying to prove not-X. It is confusing to follow a long chain of reasoning ending with "contradiction!" and then having to go back to see what it is that has been contradicted.

(11) It is fine to use diagrams to explain your argument (or algorithm), but your explanation should not be limited to the diagram. Furthermore, you should be careful not to be misled by diagrams, as they may capture only a special case and not cover all possibilities. The diagram alone should not be the proof, it should elucidate the proof.

(12) The imperatives "justify X", "prove X", "explain why X is true" all ask for the same thing: An argument that establishes the truth of X, given the relevant assumptions (that depend on the context in which these imperatives are given).