

How to write algorithms and their analysis

1. For all questions requiring an algorithm, unless otherwise specified, your solution should include, not only the algorithm, but also a running time analysis and a proof of correctness.
2. For all questions requiring a proof, you should provide not just a formal argument, even well-written, but also the statement of each lemma:
 - (a) input: the assumptions of the lemma
 - (b) output: the conclusion of the lemma.

If the answer needs to be fully detailed, you should take care to not forget the special cases in your proof.

3. For all questions requiring an algorithm, you should provide not just pseudo-code, even well-written, but also the specification of each function:
 - (a) input: for each variable, its type and what it represents, as well as the prerequisites
 - (b) output: type and what it represents, as well as the structural properties of the problem solved by the function.

If the answer needs to be fully detailed, you should also specify the exceptional cases.

4. Your meta-pseudo-code should be at a high level. For example, for finding the maximum element of an array A , it is sufficient to write:
`Scan array A to find its maximum element.`

When I read this, I can think of a few ways to implement it. They are all obvious (at the level of CS157), and they all take linear time in the size of the array. You would only need to provide more details if there was ambiguity.

5. Here is a technique for you to generate a clear, but non-ambiguous algorithm: First, you write pseudo-code. Then, you go through it and make it more compact, replacing each block by a description in English as much as possible. Iterate until you no longer see how to simplify the description.
6. Be as concise as possible.
7. A picture is worth a thousand words. Better draw a picture by hand than not have a picture!
8. A textbook quality solution is never obtained on the first writeup of a correct solution. Significant more work goes into thinking about the presentation, and several more iterations of writing are usually needed.
9. In class, the presentation is often discursive, of the form: "here is a first attempt, here is a counter example, here is a fix, here is another problem, here is a fix", etc. This is usually not a good way to write up a solution: we want to read the final result of your work, not the thinking process.