Algorithms.
Observations and More Advice

This is a temporary and informal "living" document used to collect frequent observations and advice that should be helpful and of general interest.

It is important to thoroughly define all symbols that you introduce: variables, function symbols, etc. One cannot assume that readers will be able to infer them from context. They cannot know what you had intended, unless you say it. In the worst case, one or two undefined notations can make a whole text incomprehensible.

Also write in an explicit way. For instance, instead of only saying "the algorithm utilizes the fact ...", say *how* it utilizes the fact, and what it actually *does*.

The algorithm for Problem 2 was pretty obvious, so this is not a big secret: Put two pointers at both ends of the sorted sequence. If the sum is too small, move the left pointer to the right. If the sum is too large, move the right pointer to the left.

Alternatively one can use hashing in Problem 2, but the hidden internal costs are then higher. One needs extra space and randomization.

If we go for the elementary algorithm, the next concern is correctness:

The first question before making a proof should always be: What exactly needs to be proved? In other words: What is the assertion?

In the case of Problem 2 it suffices to show: Whenever we move a pointer, the discarded element cannot be in any solution. - No more and no less! Then the correcntness easily follows, and also the argument for this claim is simple. Altogether this makes the entire proof short and transparent.

A possible mistake in correctness proofs by exchange arguments is to consider the greedy solution and show that one exchange step makes it worse. This does not prove optimality of the greedy solution, because only special solutions close to it are captured. Rather, one should consider an *arbitrary* solution and do a whole *sequence* of exchange steps that lead to the greedy solution.

Unnecessary additional writing can hide the good things, rather than adding something useful. It is wise to look through the drafts: Are all arguments really needed? Can I remove something, and the rest is still a complete solution?

For the correctness of a dynamic programming algorithm it suffices to explain the specific "recursive" formula.

In equivalence proofs (e.g. within redutions) one should prove the two directions separately, otherwise it is easy to miss a step and to wrongly conclude equivalence where only one implication holds true.

The Most Important Points to Keep in Mind

and to focus on in your studies:

Part 1. definition of O-notation and comparison of standard functions, what do we count in a time analysis, reason why certain simple time bounds are optimal (we must at least read the whole input, etc.)

Part 2. recognizing that greedy rules easily fail, use of exchange arguments in optimality proofs of (correct) greedy algorithms

Part 3-5. understanding the entire machinery of dynamic programming: computing values of sub-instances, memoinzation rather than recursion, recovering a solution from the values by backtracing, reason why the algorithms for Subset Sum and Knapsack are not truly polynomial

Part 5-7. basic idea of divide and conquer, why are recurrences needed in the time analysis, idea of solving recurrences by repeated substituion (details are less important, as they depend on the equation)

Part 7. basic idea and formal defintion of (polynomial-time) reduction, understanding why only the time for the tranfsormations is counted

Part 8-9. definitions of P, NP, and NP-completeness. understanding when a problem is in NP, all the small properties that connect these complexity classes with polynomial-time reductions; what is needed to show that a problem is NP-complete, being able to do simple reductions between closely related problems (not the tricky ones!)

Part 10. properties of BFS and DFS and a few uses of them, in particular, why BFS solves the shortest-paths problem whereas DFS does not

Part 11. just try to follow every step of the presented algorithms and proofs, no aspects are highlighted here as more important than others

Part 12. paths in DAGs are often used, the rest is additional and more specialized material

Part 13. if you must set priorities: focus on the algorithm for fiding a pair of closest points