Designing the right algorithm for a given application is a major creative act—that of taking a problem and pulling a solution out of the ether. The space of choices you can make in algorithm design is enormous, leaving you plenty of freedom to hang yourself.

This book is designed to make you a better algorithm designer. The techniques presented in Part I of this book provide the basic ideas underlying all combinatorial algorithms. The problem catalog of Part II will help you with modeling your application, and tell you what is known about the relevant problems. However, being a successful algorithm designer requires more than book knowledge. It requires a certain attitude—the right problem-solving approach. It is difficult to teach this mindset in a book, yet getting it is essential to becoming a successful designer.

The key to algorithm design (or any other problem-solving task) is to proceed by asking yourself questions to guide your thought process. What if we do this? What if we do that? Should you get stuck on the problem, the best thing to do is move onto the next question. In any group brainstorming session, the most useful person in the room is the one who keeps asking, “Why can’t we do it this way?” not the person who later tells them why, because she will eventually stumble on an approach that can’t be shot down.

Towards this end, we provide a sequence of questions to guide your search for the right algorithm for your problem. To use it effectively, you must not only ask the questions, but answer them. The key is working through the answers carefully by writing them down in a log. The correct answer to “Can I do it this way?” is never “no,” but “no, because...” By clearly articulating your reasoning as to why something doesn’t work, you can check whether you have glossed over a possibility that you didn’t want to think hard enough about. It is amazing how often the reason
you can’t find a convincing explanation for something is because your conclusion is wrong.

The distinction between strategy and tactics is important to keep aware of during any design process. Strategy represents the quest for the big picture—the framework around which we construct our path to the goal. Tactics are used to win the minor battles we must fight along the way. In problem-solving, it is important to check repeatedly whether you are thinking on the right level. If you do not have a global strategy of how you are going to attack your problem, it is pointless to worry about the tactics. An example of a strategic question is “Can I model my application as a graph algorithm problem?” A tactical question might be, “Should I use an adjacency list or adjacency matrix data structure to represent my graph?” Of course, such tactical decisions are critical to the ultimate quality of the solution, but they can be properly evaluated only in light of a successful strategy.

Too many people freeze up in their thinking when faced with a design problem. After reading or hearing the problem, they sit down and realize that they don’t know what to do next. Avoid this fate. Follow the sequence of questions provided below and in most of the catalog problem sections. We’ll tell you what to do next!

Obviously, the more experience you have with algorithm design techniques such as dynamic programming, graph algorithms, intractability, and data structures, the more successful you will be at working through the list of questions. Part I of this book has been designed to strengthen this technical background. However, it pays to work through these questions regardless of how strong your technical skills are. The earliest and most important questions on the list focus on obtaining a detailed understanding of your problem and do not require specific expertise.

This list of questions was inspired by a passage in [Wol79]—a wonderful book about the space program entitled The Right Stuff. It concerned the radio transmissions from test pilots just before their planes crashed. One might have expected that they would panic, so ground control would hear the pilot yelling Ahhhhhhhh- hhhh —, terminated only by the sound of smacking into a mountain. Instead, the pilots ran through a list of what their possible actions could be. I’ve tried the flaps. I’ve checked the engine. Still got two wings. I’ve reset the —. They had “the Right Stuff.” Because of this, they sometimes managed to miss the mountain.

I hope this book and list will provide you with “the Right Stuff” to be an algorithm designer. And I hope it prevents you from smacking into any mountains along the way.

1. Do I really understand the problem?

   (a) What exactly does the input consist of?
   (b) What exactly are the desired results or output?
   (c) Can I construct an input example small enough to solve by hand? What happens when I try to solve it?
   (d) How important is it to my application that I always find the optimal answer? Can I settle for something close to the optimal answer?