1. **Know the concepts**. Solving a problem via memory or pattern recognition is much faster than solving it by reason alone. If you've solved a similar problem before, you'll be able to recall that solution intuitively. Failing that, if you at least keep up with current research and projects related to your own you'll have a much better idea where to turn for inspiration. Solving a problem "automatically" might seem like magic to others, but it's really an application of "practice practice practice" .
2. **Know the tools**. This is not an end in itself, but a way to maintain "flow" while programming. Every time you have to think about how to make your editor or version-control system or debugger do what you want, it bumps you out of your higher-level thought process. These "micro-interruptions" are small, but they add up quickly. People who learn their tools, practice using their tools, and automate things that the tools can't do by themselves can easily be several times as productive as those who do none of those things.
3. **Manage time.** Again it comes back to flow. If you want to write code, write code. If you want to review a bunch of patches, review a bunch of patches. If you want to brainstorm on new algorithms . . . you get the idea. Don't try to do all three together, and certainly don't interrupt yourself with email or IRC or Twitter or Quora. ;) Get your mind set to do one thing, then do that thing for a good block of time before you switch to doing something else.
4. **Prioritize.** This is the area where I constantly see people fail. Every problem worth tackling has many facets. Often, solving one part of the problem will make solving the others easier. Therefore, getting the order right really matters. I'm afraid there's no simple answer for how to recognize that order, but as you gain more experience within a problem domain - practice again - you'll develop a set of heuristics that will guide you.
5. **Reuse everything**. Reuse ideas. Reuse code. Every time you turn a new problem into a problem you already know how to solve - and computing is full of such opportunities - you can save time. Don't worry if the transformed solution isn't absolutely perfect for the current problem. You can refine later if you really need to, and most often you'll find that you're better off moving on to the next problem.

A lot of these really come down to efficiency. As you move through more problems per day, you'll gain more experience per day, which will let you move through more problems per day, and so on. It's a feedback loop; once you get on its good side, your effectiveness (and value) will increase drastically.