Debugging and Efficiency

Paul Stankovski

Here are some pointers that you will find very useful for this project and for the duration of this course. But even more importantly, the lessons you can learn here will be *very* valuable for you if you want to raise your industrial market value.

1 Debugging

This part regards the correctness of your code. The purpose of debugging is to catch logical errors.

Everybody makes mistakes when it comes to coding, so learning the debugging trade will serve you well if you would rather spend more of your time coding than debugging. Really good programmers are good at debugging. They seem to have a way of finding the errors quickly and without much effort. Do they use some kind of a trick to make it look so easy? Yes they do, and you will soon be able to use it too.

Debugging is hard if you have no idea what you are doing, but it is very easy if you have an efficient search strategy. This is the trick that good programmers use, to have an efficient search strategy that minimizes the time that you spend on debugging. Stick to this strategy—do not deviate from it!

Use binary search. First of all, find a set of parameters for which your code does not work as expected. Then choose some place close to the middle part of your code as a testing point. Test your program data to see if your program behaves correctly up to that point. If it does not, you will find at least one bug in the first half of your program. If it does behave correctly, the bugs can be found in the latter half.

Now simply repeat the search process with the remaining half of your code. You will then isolate the erroneous part(s) of your code to a successively smaller code area, maybe even to a single line of code or a statement.

Assert statements are a good way of making the testing points explicit. In this way you can also perform the test automatically in all future program executions (in debug mode).

Following the above strategy is easy, and you will always find the bug(s) in this way. Just looking at all of your code all at once and hoping to catch an error by chance is not a good strategy, it is a sign that you have given up. Stay focused. Employing the binary search strategy, you will live long and prosper.

2 Efficiency

While the mathematical part of the problem is solved when your code is correct (i.e. gives you the correct answer after *some* time), in practical applications you also need to deal with the efficiency of your code.

Short answer: Lack of efficiency is a bug. See Section 1.

Longer answer: Do not rule out the possibility that you have logical errors in your code. It is possible that the errors appear only for some specific sets of parameters, or when the parameters provided are very big.

If your code is very slow, you will first need to identify the bottlenecks. You can use the strategy given in Section 1 for locating these. Once you have done that, fix them. Figure out what the algorithm actually does, and how you can make that part more efficient. Once you see the structure, it is easy. You can do it, believe in yourself!

Choose data structures with care. If you know how much data you need to store beforehad, use a structure with a static size. Also, how will you use your data? Make sure that utilization of your data, or transformation from one storage structure to another, does not waste valuable time.

Memory allocation and deallocation can take more than a fair share of time. Avoid inefficient memory usage by minimizing the number of allocations. If you allocate a temporary buffer, consider passing a preallocated buffer as a parameter instead.

When you have nested loops, the code in the inner core part is expensive to run. If you have some tasks or calculation that can be performed outside the loop, move it out. Make the core part as small and efficient as possible with respect to calculations, memory allocation, and so on.

You can find many debugging hints and tutorials on-line.