Hi everyone, I would like to take a few minutes tonight—and I plan to speak for no more than 10 or 15 minutes, so this should be relatively painless—to tell you about two things. First, I'll tell you a few reasons why I believe it's a good idea for computer scientists to study as much math as they can. Second, I'll tell you a few reasons why I believe mathematicians should understand some elements of computer science. (By the way, notice the asymmetry between these two statements: computer scientists should study as much math as they can, but mathematicians should understand some elements of computer science.)

So, let's leap straight into my first claim. Why should computer scientists study math? The main reason is that they need it: a significant proportion of computer science concepts cannot be described without sophisticated mathematics. There are other reasons too. For example, studying math is probably the best way of learning how to think logically and clearly, and this kind of thinking helps immensely when doing computer science. But I'm not going to talk about this reason or any other reasons, since the main reason is so important and obvious: a lot of computer science depends on math. If you can't do the math, you can't do the computer science. For example, consider the example of online mapping services like Google Maps and Bing Maps. One of the things these organisations try to do is compute the fastest route from one point to another. Computer scientists know how to do this using an algorithm invented by Edsger Dijkstra over 50 years ago, but the algorithm isn't efficient enough for online map services that receive millions of queries per day on maps that contain hundreds of thousands of roads. The modern shortest path algorithms actually used by today's mapping services require a healthy dose of math, including graph theory, linear algebra, geometry, and even topology (Delling 2011, Abraham 2010). If you want to work on mapping at Google, Microsoft, or MapQuest, it would be a good idea to know something about all of these topics. [mention online version with citations]

Obviously this MapQuest application is just one example, but it turns out that huge swathes of computer science depend on math in a similar way. Almost any application of pattern recognition or machine learning (this includes the automatic typing correction on your smartphone, face detection in its camera, and speech recognition when you call an automated telephone service) requires a good knowledge of linear algebra, probability, and statistics. Almost any application of cryptography, digital signatures, or computer security requires number theory, which in turn depends on group theory and certain other algebraic structures. The vast array of computational tasks in fields ranging from biology to finance require a good knowledge of optimization (including operations research), which in turn depends on multivariable calculus. And I have barely scratched the surface. [To explore further, see, for example Frailey (2006), Wing (2002), and the website of the Working Group on Integrating Mathematical Reasoning into Computer Science Curricula.] But I hope I started to convince you that the more math you know, the more computer science you can do.

[Mention Chuck Thacker's interview comments. Mention widespread disagreement on slashdot.]

What if you love computers and hate math? Don't panic. There are plenty of important areas of computer science that don't need much math, such as software engineering, user interface design, and operating systems. But my point is that if you're a computer scientist who does like math, you should devour as much as possible.

Now let's turn to my second point. Why is it a good idea for mathematicians to master some of the basic elements of computer science?

Before we can answer this, we need to know what the basic elements of computer science are. Well, it turns out that most people -- even people who think they know what computer science is -- actually do not know what computer science is. If you ask a non-computer scientist the question "what is computer science?", the most frequent answer is something similar to "programming computers". (I know this because I do this experiment on my intro class every time I teach COMP131, and something to do with programming is by far the most frequent response.) In fact, programming is only a lens through which we view the actual ideas of computer science. It's rather like studying French literature: of course you need to know French before you can study French literature properly, but a knowledge of the French language does not, by itself, tell you anything at all about French literature. In the same way, knowledge of a programming language itself doesn't tell you any fundamental ideas of computer science, but without the programming language, we have no way of implementing -- or even meaningfully discussing -- those fundamental ideas. And what are these fundamental ideas? Mostly, they are algorithms (such as how to efficiently sort a list of names into alphabetical order) and data structures (such as how to store the names in such a way that you can find them later with minimal effort). And perhaps the most important fundamental idea of all is abstraction, which means gradually refining complex problems into simpler problems, and thus converting a complex problem into a series of simple ones. So these are the elements of computer science: first there is programming, but only so we can discuss the other ideas, and then there are algorithms, data structures, and abstraction. This is what I believe mathematicians should try to know something about.

Why? The primary reason is that computational knowledge is likely to be very useful for a mathematician. Indeed, your performance at almost any job that uses mathematical knowledge (and this includes, for example, finance, consulting, modeling, business analysis, communications, insurance, and research in many applied fields like biotech) will be greatly enhanced by the ability to implement your ideas on a computer. Sometimes, skillful use of a spreadsheet is sufficient, and little computer science knowledge is necessary for that. But there are many problems that cannot be tackled with a spreadsheet. For those, you'll need some computer science basics. Specifically, you'll need to know how to develop some suitable abstractions of the problem, select a suitable algorithm and/or data structure for each abstraction, and implement these in a programming language (such as Matlab or Java).

What if you love math but hate computers? Again, don't panic. A stand-alone math major already gives you a wealth of opportunities, so I wouldn't recommend contending with computer science courses if you don't enjoy them. But if you are a math major who does like computers, I urge you to learn the basics of computer science, as they will greatly amplify your potential.

Now, I'm a computer science professor, so maybe my advice is biased. Of course I would tell you to study computer science! What do others have to say about this? It turns out that there are plenty of math professors who think their students should study some computer science. At Cornell, you can't even become a math major until you've taken a computer science course (Cornell University, 2010, p615)! At the college I went to in England, every math major had to complete a computer project. [Mention I was a math major. Didn't study computer science until some years later.] And that was 20 years ago -- these days, they have to do two computer projects (University of Cambridge, 2010, p3). By the way, requirements like this would not be a good idea at Dickinson, where we (quite rightly, in my opinion) emphasize the importance of a broad liberal arts education. I only mention these examples to suggest that I'm not necessarily reporting a biased opinion: plenty of math professors believe their students should study some computer science.

Now for some practical details. Suppose, for the sake of argument, that you believe everything I just said and would like to take my advice. What courses should you take at Dickinson? If you're a computer science major, I said you should take as much math as you can, provided you enjoy it. Whether or not you enjoy it, the major requires you to take Single Variable Calc, and Discrete Math, so you are stuck with those for sure. Beyond that, I recommend, in order of importance: Linear Algebra, Prob and Stat 1, Multi-Variable Calc, and Prob and Stat 2. And after that, as much of everything else as you can handle. If you're a math major, I said you should learn the basics of computer science, provided you enjoy it. Actually, it's possible you can get these basics within a math course. Depending on how they are taught, courses like Linear Algebra, Numerical Methods, and Operations Research might contain enough programming and algorithms to give you the basics. [Mention Matlab. Java/C++ not necessary.] So it might not even be necessary to study computer science separately, if those courses are taught with a computational emphasis. But if you want to guarantee that you acquire what I have called the basics of computer science, you would need to take the two-course introductory sequence (131 and 132), which gives you Java programming and elementary algorithms. Then you would need the data structures course (232), and preferably also the algorithms course (332). If you take all of those, you are beautifully poised to exploit computational ideas in any mathematically-oriented career, whether it be a PhD in mathematics, an actuarial position, a job on Wall Street, a bio-statistician, or teaching math in high school.

I've taken up enough of your time, but would certainly be happy to answer a couple of quick questions if there are any, and I want to end by thanking you very much for your attention. So thank you very much indeed, and enjoy your evening!