See the Sort

written by Jeremy Huggett | 26/04/2015

What's not to like about the idea of central European folk dance being used as a means of illustrating the operation of different sorting algorithms? That's what the Algo-rythmics did a few years ago – my personal favourite has to be the Quick Sort (below) with the hats changing with the operands, but do check them out (all six are on their Youtube page).

Within the last ten days, we've been reminded about the invisibility of algorithms which govern much of our online activity. We've seen Google alter its search ranking algorithm to prioritise mobile-friendly sites in its search results, Facebook change its newsfeed algorithm to give greater precedence to posts from friends (who'd have thought it?!), and the French Senate vote to require search engines to reveal the workings of their search ranking algorithms to ensure they deliver fair and non-discriminatory results. There's also been discussion of the role of trading algorithms in the 2010 'flash crash' and stock market movements in the last month or so in the US …

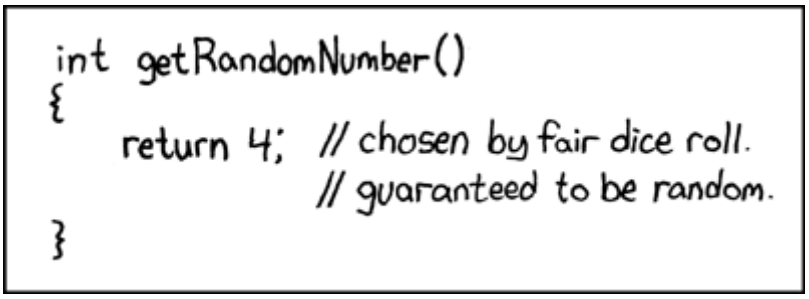
What this underlines is the way that algorithms affect our experiences, decisions, selections, relations, and information. At the same time, these algorithms also capture and contain embedded within them a range of concepts, assumptions, rules and methodologies. This isn't restricted to large Internet companies – it also impacts on our day-to-day use of software. For example, Ben Marwick has recently emphasised that our reliance as archaeologists on software like Excel or SPSS limits what we can do, highlighting amongst other issues the black boxing of the algorithms these packages use which makes the processes behind the results opaque. He argues that one of the benefits of open source is that the code and algorithms that the programs use are not hidden from us.

Of course, algorithms are not straightforward entities – you only have to compare the list of '10 Algorithms that Dominate our World' by George Dvorsky (ranging from Google Search, Facebook News Feed, NSA data collection, recommendation systems on Amazon, Netflix etc.) with the 10 proposed by Marcos Otero (sorts, fourier transforms, data compression, random number generation, etc.) to see that algorithms operate at different levels of conceptualisation. Indeed, algorithms frequently use other algorithms and these may use others again in a complex series of interrelationships. Consequently, algorithms operate at every level, from high frequency financial trading systems to the handling of basic input/output and mathematical functions within program code. That's the nature of programming. Different programming languages operate at different levels of disambiguation, but it ultimately comes down to the machine-code level of moving bits around registers in order to achieve some fundamental operation. For instance, a brief scan of the *99 Bottles of Beer* collection of programs which generate the lyrics to the song of that name quickly demonstrates that different languages bring their own approaches, levels of verbosity and elegance of solution while at the same time highlighting the way in which even the simplest task can be resolved through a wide variety of different algorithms.

Ben Marwick concludes that scripted analysis using an open source language is better for archaeologists, and science generally, than using proprietary systems. This is doubtless true: after all, open source means that we can inspect the code and algorithms that sit beneath the surface user interface. But how many of us would do that? Or would understand what we saw if we did so? Intriguingly, the Algo-rythmics examples demonstrate that seeing the sort is not the same as understanding it, in spite of the fact that this is what they are designed to do. The rhythmic dance movements don't easily convert into the underlying algorithmic code which drives them unless you've already got some familiarity with the sort routines being represented.

So we need to be clear about what we wish for. Seeing the code alone is not enough. Code with inline comments is probably not enough for that matter, either, as the comments are all too often written for other programmers rather than for those trying to understand what is going on. We might conclude that we need documentation that sits alongside the code, that explains what it seeks to achieve, the assumptions made, and the principles and methods selected to achieve those ends. But how often is this available and current?

Ironically, the transparency that ought to be implied by breaking tasks down into their component parts, formalising practices and methods, making things explicit, cannot be taken for granted (for example, Rieder and Röhle 2012). Seeing the algorithms doesn't guarantee that the processes they purport to model will be understandable. And how far should we deconstruct the code? At what point can we be confident that it does what we believe it to do, that it isn't fundamentally flawed by a lower-level function such as a very definitely non-random random number generator (below!)?



'Random' number generator (xkcd – CC BY-NC)

David Berry's *The Philosophy of Software* underlines the challenges involved in achieving the level of understanding required. For instance, his discussions of the *Underhanded C Contest* and the *International Obfuscated C Code Contest* (Berry 2011, 75ff) demonstrate the way that code can be designed to disguise or mislead – deliberately in these specific instances, but more likely inadvertently elsewhere. We certainly need to be able to have more than just faith and belief in the algorithms that drive our tools, but equally, we need to be realistic about what we can actually achieve. There's more to seeing the sort than seeing the sort …

References

Berry, D. 2011 *The Philosophy of Software*. Palgrave Macmillan.

Rieder, B. and Röhle, T. 2012 'Digital Methods: Five Challenges', in D. Berry (ed.) *Understanding*

Digital Humanities, pp. 67-84. Palgrave Macmillan.