The story of Sissa and Moore

According to the legend, the game of chess was invented by the Brahmin Sissa to amuse and teach his king. Asked by the grateful monarch what he wanted in return, the wise man requested that the king place one grain of rice in the first square of the chessboard, two in the second, four in the third, and so on, doubling the amount of rice up to the 64th square. The king agreed on the spot, and as a result he was the first person to learn the valuable—albeit humbling—lesson of *exponential growth*. Sissa's request amounted to $2^{64} - 1 = 18{,}446{,}744{,}073{,}709{,}551{,}615$ grains of rice, enough rice to pave all of India several times over!

All over nature, from colonies of bacteria to cells in a fetus, we see systems that grow exponentially—for a while. In 1798, the British philosopher T. Robert Malthus published an essay in which he predicted that the exponential growth (he called it "geometric growth") of the human population would soon deplete linearly growing resources, an argument that influenced Charles Darwin deeply. Malthus knew the fundamental fact that an exponential sooner or later takes over any polynomial.

In 1965, computer chip pioneer Gordon E. Moore noticed that transistor density in chips had doubled every year in the early 1960s, and he predicted that this trend would continue. This prediction, moderated to a doubling every 18 months and extended to computer speed, is known as *Moore's law*. It has held remarkably well for 40 years. And these are the two root causes of the explosion of information technology in the past decades: *Moore's law and efficient algorithms*.

It would appear that Moore's law provides a disincentive for developing polynomial algorithms. After all, if an algorithm is exponential, why not wait it out until Moore's law makes it feasible? But in reality the exact opposite happens: Moore's law is a huge incentive for developing efficient algorithms, because such algorithms are needed in order to take advantage of the exponential increase in computer speed.

Here is why. If, for example, an $O(2^n)$ algorithm for Boolean satisfiability (SAT) were given an hour to run, it would have solved instances with 25 variables back in 1975, 31 variables on the faster computers available in 1985, 38 variables in 1995, and about 45 variables with today's machines. Quite a bit of progress—except that each extra variable requires a year and a half's wait, while the appetite of applications (many of which are, ironically, related to computer design) grows much faster. In contrast, the size of the instances solved by an $O(n)$ or $O(n \log n)$ algorithm would be *multiplied by a factor of about 100* each decade. In the case of an $O(n^2)$ algorithm, the instance size solvable in a fixed time would be multiplied by about 10 each decade. Even an $O(n^6)$ algorithm, polynomial yet unappetizing, would more than double the size of the instances solved each decade. When it comes to the growth of the size of problems we can attack with an algorithm, we have a reversal: exponential algorithms make polynomially slow progress, while polynomial algorithms advance exponentially fast! For Moore's law to be reflected in the world we *need* efficient algorithms.

As Sissa and Malthus knew very well, exponential expansion cannot be sustained indefinitely in our finite world. Bacterial colonies run out of food; chips hit the atomic scale. Moore's law will stop doubling the speed of our computers within a decade or two. And then progress will depend on algorithmic ingenuity—or otherwise perhaps on novel ideas such as *quantum computation*, explored in Chapter 10.