Why Teaching Informatics in Schools Is as Important as Teaching Mathematics and Natural Sciences[*]

Juraj Hromkovič and Björn Steffen

Department of Computer Science, ETH Zurich, Switzerland
{juraj.hromkovic,bjoern.steffen}@inf.ethz.ch

Abstract. In this paper, we aim to do more than arguing that informatics is a fascinating scientific discipline with interesting applications in almost all areas of everyday life. We pose the following questions: What are the educational requirements demanded from school subjects? Can we answer this question as satisfactory as we can do it, for instance, for mathematics, physics or chemistry? Does the teaching of informatics enrich education in ways other subjects cannot or do not sufficiently contribute?

Answering the questions above can not only be helpful for the discussion with politicians about integrating proper informatics and not only ICT skills in the educational systems, but it can also help us as teachers to focus on the fundamentals and on sustainable knowledge.

1 Introduction

If one deals with the problem of teaching a subject A in schools, one has to be able to answer the following three fundamental questions:

1. Does the teaching of A contribute to the understanding of our world and if yes, in which way and to what extent? How does A prepare the pupils for dealing with jobs and duties in society?
2. How important is the teaching of A for the ability to succeccfully study at a university? Do universities expect some fundamental knowledge of the discipline A?
3. What is the contribution of teaching A to the development of the way of thinking and the ability to solve various kinds of tasks and problems.

In the following three sections, we aim to give at least partial answers to these questions regarding the subject of informatics.

Before we deal with the three questions posed above, let us explain why we do not deal with imparting ICT skills in this article. Clearly, ICT skills have become important not only for studying, but also for future employments. Therefore, these skills need to be taught at school. However, in this article we focus on the unique, sustainable and lasting knowledge that informatics can provide.

[*] Partially supported by the Hasler Foundation.

To present scientific disciplines as collections of discoveries and research results gives a false impression. It is even more misleading to understand science merely in terms of its applications in everyday life. What would the description of physics look like if it was written in terms of the commercial products made possible by applying physical laws? Almost everything created by people—from buildings to household equipment and electronic devices—is based on the knowledge of physical laws. However, nobody mistakes the manufacturer of TVs or of computers or even users of electronic devices for physicists. We clearly distinguish between the basic research in physics and the technical applications in electrical engineering or other technical sciences. With the exception of the computer, the use of specific devices has never been considered a science.

Why does the public opinion equate the proficiency to use specific software packages to computer science? Why is it that in some countries teaching computer science is restricted to imparting ICT skills, i.e., learning to work with a word processor or to search on the internet? What is the value of such education, when software systems essentially change every year? Is the relative complexity of the computer in comparison with other devices the only reason for this misunderstanding?

Surely, computers are so common that the number of computer users is comparable to the number of car drivers. But why then is driving a car not a subject in secondary schools? Mobile phones are becoming small, powerful computers. Do we consider introducing the subject of using mobile phones into school education? We hope that this will not be the case. We do not intend to answer these rhetorical questions; the only aim in posing them is to expose the extent of public misunderstanding about computer science. Let us only note that experience has shown that teaching the use of a computer does not necessarily justify a new, special subject in school.

There are two main points in the relation between teaching core informatics and imparting ICT skills. First of all, none of our three principal questions posed above can be answered satisfactory if one reduces informatics to a computer driving licence. Secondly, reducing teaching of informatics to educating the usage of concrete software systems (word processor, spreadsheet, etc.) is the main reason for the current very bad image of informatics. Following our experience and some investigation in countries that replaced the teaching of informatics by imparting ICT skills, the pupils do not consider informatics as a science, they find it boring and not challenging enough to choose it as a subject for the study at a university. There is no other way out than to teach computer science in such a way that it is at least as attractive and deep as other natural sciences and mathematics.

2 Understanding the World Around Us

Physics helps us to understand a lot about the world around us. What is matter and what is energy? What is movement, space, and time? To understand at least some fundamental discoveries and laws of physics is of principal importance for understanding our world. Another example is, for instance, biology. It tries to uncover what life is and how it evolves. Many of the principal goals of the natural sciences are fascinating and challenging. The deep questions about the functioning of our world and about our role in this world are for most young people much more attractive than any colorful application on the screen. Understanding nontrivial concepts and getting deep insight into complex subjects makes them more pleased than mastering any skill.

Can computer science provide something comparable? Our answer is yes. The following two discoveries of computer science are of this kind.

The limits of automation and the concept of algorithms

All possible tasks (problems) can be partitioned into two classes, namely algorithmically solvable and algorithmically unsolvable tasks. Due to the introduction of the notion of the algorithm by A. Turing [13], we got an instrument for classifying computing problems into those solvable by computers and others unsolvable by computers. We discovered the existence of a rigorously defined limit of automation.

Quantitative laws of information processing and the concept of computational complexity

To perform a computation or to process information in order to solve a concrete task requires some amount of work. There exist quantitative laws of information processing. For each task of information processing, one can investigate the amount of computer work sufficient and necessary in order to solve the task. We discovered that more computational resources help us to solve more tasks and that there are arbitrarily hard tasks with respect to the amount of work required. Thousands of practical computing problems cannot be solved because the amount of work necessary is beyond the physical capabilities of our universe. The core scientific topic of informatics is to recognize how much information one can extract from the given data by a reasonable amount of work.

Both fundamental concepts mentioned above do not only reveal us the existence of some natural laws of information processing. Similar to physics, these concepts are also crucial for many of today's applications. For instance, the current e-commerce, based on public-key cryptography, would not exist without the fundamental concepts of algorithms and computational complexity.

Examples of how to introduce these concepts in an appropriate way for secondary schools is presented in numerous books: *Algorithmics: The Spirit of Computing* [7] *Das Affenpuzzle* [3] *Abenteuer Informatik* [6], *Computer Science Unplugged* [1], *Taschenbuch der Algorithmen* [14], *Algorithmic Adventures* [10], *Lehrbuch Informatik* [8], *Berechenbarkeit* [9], *Einführung in die Kryptologie* [5].

Another important issue about computational complexity is the fact that many computing problems can be solved efficiently, but it is a challenge to discover an efficient algorithm for them and this subject of informatics is very fruitful.

A classical problem from cryptology that is not practically solvable using a naive approach is for example the calculation of powers with very large exponents. But with an ingenious trick, the so-called fast exponentiation, this is efficiently doable. In Appendix A we present an approach we used in the textbook *Einführung in die Kryptologie* [5] to introduce the fast modular exponentiation to secondary-school students.

We conclude that informatics has a nontrivial scientific depth that is fascinating and can challenge young people with its goals and problems. Informatics expands science also by giving new dimensions to the fundamental categories of science such as determinism, nondeterminism, randomness, proof, simulation, correctness, efficiency etc. Many of its discoveries are surprising and attractive to young people. For instance, exchanging a deterministic control by a randomized one, whose decisions are partially influenced by random bits, can essentially decrease the amount of work necessary to reach the intended goals.

We only need to get more experience with teaching these topics understandably in our schools. There is no doubt that, in our society based on knowledge which is extracted daily from a huge amount of data, one cannot understand the world around us without some fundamental knowledge of informatics. The same is true for the control of technical devices in the technical world created by man.

3 Preparation for University Studies

For sure, ICT skills have become instruments that necessarily have to be mastered to successfully participate in many human activities. But this is not the matter of our discussion because they are mostly on the basic technical level like using a pen for writing. Here we want to look for the usefulness of sustainable knowledge in informatics.

First of all, most of the scientific disciplines are asked to handle a huge amount of data. This is not only true for experimental sciences such as biology, chemistry or physics, but also for economics, sociology, medicine etc. Because the amount of data is growing fast, one needs a well-structured way of saving it as well as efficient algorithms for searching, processing and communicating it. Many scientific disciplines, not only the natural sciences, generate knowledge from the data they collected by experiments, measurements, and assessments. To extract knowledge from the given data often requires a huge amount of computer work. This can only be done if one is able to develop specific efficient algorithms for these tasks or to ask computer scientists for their support. But this cannot be achieved without the ability to describe the problems exactly.

Of a similar importance is the application of simulations. In order to forecast some development, we create models and run simulations on them. This originally basic research instrument of physics, chemistry and engineering became a fundamental tool in sciences like economics, sociology, psychology, etc., which avoided the use of exact mathematical methods for a long period of time. Without simulations, many research projects would be unthinkable.

Nobody discusses whether mathematics has to be a part of school education or not. But informatics is the scientific discipline making mathematics to a technology. Due to this, mathematics is applied everywhere.

Programming is a part of informatics with a growing importance. All technical systems are controlled by programs and therefore all students of technical sciences need to master programming. Slowly, but surely, this becomes true also for several areas of the natural sciences and even for humanities. Programming is not only the ability to implement given methods into programs. Much more important is the ability to use abstractions to describe problems, to analyse them and to find methods for solving them.

Definitely, we conclude that teaching informatics in school is not only a contribution for the study of related topics at university. The knowledge about the capabilities and limits of computer science and the way of working in informatics is at least as useful for the study at a university as any other classical subject of high school curricula.

4 A Way of Thinking and Working

One of the main arguments for teaching mathematics is the development of the exact way of thinking that finally results in the ability to use the exact language of mathematics for describing, analyzing and solving problems in all areas of our life. This ability becomes more and more important. Some colleagues tend to call informatics the "new" mathematics or at least a constructive mathematics. Jeannette Wing, head of the computer science department at Carnegie Mellon University, even envisions that "thinking like a computer scientist" should be a fundamental skill such as reading, writing and arithmetics [15]. Cohen and Haberman regard computer science as one of the five "languages" every citizen should acquire [2].

The reason for that is the way of working in computer science. Similarly as in mathematics, we begin with an abstract description of a problem and continue with its analysis. But additionally, computer scientists do not only discover an efficient way of solving it, but they also implement the discovered method and provide a product (program) for solving problems of this kind. This work is more constructive than the typical work of a mathematician and ties the exact way of thinking in mathematics with the pragmatic way of working in engineering.

In this way teaching informatics in school:

– supports the development of the exact way of thinking and working in mathematics and natural sciences, and
– brings new elements to education by teaching elements related to the way of thinking and working in engineering disciplines (introducing the concepts of implementation, verification (proving correctness), testing, modular design, etc.).
– supports interdisciplinarity, because the computing task considered (the extraction of information given data) have their origin in various scientific disciplines and industrial applications.

No other subject in school goes this long way from a problem formulation to a solution in the form of a product (program). This high level of constructivism is probably the main contribution of teaching computer science. This approach also contributes to the teaching of math as Syslo and Kwiatkowska point out [12].

It has to be said that teaching informatics in this way is very rewarding. One can fascinate young people and give them the great feeling of achievement.

Instead of being first of all an examinator, checking the success of the class by exams, the teacher can switch into the role of a supporter helping along the way from the problem to the solution. Whether the result of their work is correct can be directly verified by the pupils, no immediate judgement of the teacher is necessary. Additionally, team work can be educated in an excellent way.

5 Conclusion

To teach the discoveries of informatics, its methods, and ways of thinking and acting contributes to education at least in the same amount as teaching mathematics or other classical subjects. Additionally, teaching informatics brings new elements to the schools that we are missing already for a long time. Informatics contributes to science with new fundamental concepts and terms like algorithm, computational complexity, efficiency, verification, simulation, information security, etc., and gives a deeper insight on some fundamental notions of science such as determinism, nondeterminism, and randomness, to name a few.

Teaching informatics has became important to successfully study at a university in many scientific disciplines. More and more scientific disciplines expect and will expect fundamental knowledge of informatics and especially the ability to apply it in their own discipline.

The way of thinking and working in informatics enriches the human way of thinking and can essentially contribute to the success of young people in their life, whatever they will do in the future.

References

1. Bell, T., Fellows, M., Witten, I.H.: Computer Science Unplugged - Off-line activities and games for all ages (2006), `http://www.csunplugged.org`
2. Cohen, A., Haberman, B.: Chamsa: Five languages citizens of an increasingly technological world should acquire. ACM Inroads 1, 54–57 (2010)
3. Davis, H.: Das Affenpuzzle. Springer, Heidelberg (2001)
4. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions of Information Theory 22(6), 644–654 (1976)
5. Freiermuth, K., Hromkovič, J., Keller, L., Steffen, B.: Einführung in die Kryptologie. Vieweg+Teubner (2009)
6. Gallenbacher, J.: Abenteuer Informatik, 2nd edn. Elsevier, Amsterdam (2008)
7. Harel, D., Feldman, Y.: Algorithmics: The Spirit of Computing, 3rd edn. Addison Wesley, Reading (2004)
8. Hromkovič, J.: Lehrbuch Informatik. Vieweg+Teubner (2008)
9. Hromkovič, J.: Berechenbarkeit. Vieweg+Teubner (2011)

10. Hromkovič, J.: Algorithmic Adventures. Springer, Heidelberg (2009)
11. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
12. Syslo, M., Kwiatkowska, A.: Contribution of informatics education to mathematics education in schools. In: Mittermeir, R.T. (ed.) ISSEP 2006. LNCS, vol. 4226, pp. 209–219. Springer, Heidelberg (2006)
13. Turing, A.M.: On computable numbers with an application to the Entscheidungsproblem. Proceedings of London Mathematical Society 42(2), 230–265 (1936)
14. Vöcking, B., Alt, H., Dietzfelbinger, M., Reischuk, R., Scheideler, C., Vollmer, H., Wagner, D. (eds.): Taschenbuch der Algorithmen. eXamen.press, Springer (2008)
15. Wing, J.: Computational thinking. Comunications of the ACM 49(3) (2006)

A Fast Modular Exponentiation

Many cryptographic protocols, e. g. the RSA cipher [11] and the Diffie-Hellman protocol [4], rely on the calculation of modular powers with very large exponents, i. e. numbers with hundred or more digits. This is one prime example where the naive approach to calculate it leads to an inefficient algorithm, but with a important observation we can determine these powers efficiently.

This section shows an excerpt from our textbook *Einführung in die Kryptologie* [5], where we explain the method of the fast modular exponentiation to secondary school students.

If we calculate the power $a^x \bmod n$ naively like

$$a^x \bmod n = \underbrace{a \cdot a \cdot a \cdot \ldots \cdot a}_{x \text{ factors}} \mod n$$

with repeated multiplications, then $x - 1$ multiplications have to be carried out.

When x is large, 10^{200} for example, then more multiplications have to be made than the age of the universe ($\approx 10^{17}$ seconds) multiplied by the number of particles in the visible universe (below 10^{80}). This means that the calculation of a power like $a^{10^{200}}$ is not feasible, if we carry it out so clumsy. Therefore we want to build a more clever algorithm to efficiently calculate such powers.

First of all we observe that we do not need to work with large numbers such as a^x because we work modulo n and the students already know that they are allowed to reduce the size of numbers in the calculation below n by computing the reminder modulo n after each computation step[1]. Therefore, the size of the representation of the numbers in our calculation is always in $O(\log n)$ and we can measure the computational complexity (amount of computer work) by the number of arithmetic operations executed.

Therefore, we are allowed to focus on the number of operations needed to compute a^x and so we simplify our exponentiation by removing modular calculations mod n from our notation.

[1] At this point the students already worked trough a module about modular computations in the textbook [5].

We now ask the students to compute a^x with fewer than $x - 1$ multiplications for concrete numbers for x. They may discover it by their own or you can present a few motivating examples such as the following ones:

The power a^{16} can be expressed as

$$a^{16} = \left(\left((a^2)^2\right)^2\right)^2$$

and because of that one can calculate the power as follows:

$$\begin{aligned} a^2 &= a \cdot a \\ a^4 &= a^2 \cdot a^2 \\ a^8 &= a^4 \cdot a^4 \\ a^{16} &= a^8 \cdot a^8 \end{aligned}$$

by using only 4 multiplications. Similarly the following power

$$a^{24} = \left(\left((a^3)^2\right)^2\right)^2$$

can be determined with only 5 multiplications:

$$\begin{aligned} a^2 &= a \cdot a \\ a^3 &= a^2 \cdot a \\ a^6 &= a^3 \cdot a^3 \\ a^{12} &= a^6 \cdot a^6 \\ a^{24} &= a^{12} \cdot a^{12}. \end{aligned}$$

Then plenty of small challenges can be formulated. For instance, we can ask for

– calculating a^x for a concrete x with a given number of multiplications.
– searching for the smallest number of multiplications sufficient to compute a^x for concrete values of x.
– finding several different optimal ways for calculating a^x for concrete values of x.

After playing with several small challenges of the above mentioned kind, one can pose the following question:

> "It is nice to discover the best possible ways for calculating a^x for a given x. But this does not provide the possibility to automize the calculation, because for different x and y we may use different approaches to calculate a^x and a^y. Hence, we need an efficient algorithm that calculates a^x for any x in an uniform way."

One can start to search for some systematic way of calculating a^x for any x. With some help the class may be able to discover it on their own. A good starting point is an example of the following kind:

$$a^{21} = a^{16} \cdot a^4 \cdot a = \left(\left((a^2)^2\right)^2\right)^2 \cdot (a^2)^2 \cdot a.$$

We can calculate a^{21} by 6 multiplications as follows:

$$\begin{aligned} a^2 &= a \cdot a \\ a^4 &= a^2 \cdot a^2 \\ a^8 &= a^4 \cdot a^4 \\ a^{16} &= a^8 \cdot a^8 \\ a^{20} &= a^{16} \cdot a^4 \\ a^{21} &= a^{20} \cdot a. \end{aligned}$$

We observe that this calculation has two parts. The first one is used to compute a^2, a^4, a^8, and a^{16}, and the second one calculates the product of some of them. In this way one can discover that the binary representation of 21 is

$$10101,$$

where the first 1 is for 16 (2^4), the second 1 for 4 (2^2) and the last 1 stands for 1 (2^0). This means

$$a^{21} = a^{16} \cdot a^4 \cdot a,$$

i. e., the binary representation estimates, which of the partial precomputed powers a, a^2, …, a^{16} are used to compute the final result a^{21}. In general it is true that if

$$x = x_n \cdot 2^n + x_{n-1} \cdot 2^{n-1} + \cdots + x_1 \cdot 2^1 + x_0 \cdot 2^0 = \sum_{i=0}^{n} x_i \cdot 2^i,$$

then

$$a^x = a^{x_n \cdot 2^n} \cdot a^{x_{n-1} \cdot 2^{n-1}} \cdot \ldots \cdot a^{x_1 \cdot 2} \cdot a^{x_0} = \prod_{i=0}^{n} a^{x_i \cdot 2^i}.$$

The analysis of the computational complexity of this algorithm can be estimated easily. If $x = x_n x_{n-1} \ldots x_2 x_1 x_0$ is the binary representation of x, then we need first n multiplications to compute $a^2, \ldots, a^{2^n}$ and then as many multiplications as there are 1's in the binary representation of x. Since $\lfloor \log_2(x) \rfloor \leq n$,

$$2 \cdot \lfloor \log_2(x) \rfloor$$

multiplications are always sufficient to calculate a^x.

That does not need to be the end of our short teaching unit. One can deepen the acquired knowledge by posing and investigating for instance the following questions:

1. An algorithm for computing a^x can start as follows:

$$\begin{aligned} a^2 &= a \cdot a \\ a^3 &= a^2 \cdot a \\ a^9 &= a^3 \cdot a^3 \cdot a^3 \\ a^{27} &= a^9 \cdot a^9 \cdot a^9 \\ a^{81} &= a^{27} \cdot a^{27} \cdot a^{27} \\ a^{3^n} &= a^{3^{n-1}} \cdot a^{3^{n-1}} \cdot a^{3^{n-1}} \end{aligned}$$

   for $3^n \leq x < 3^{n-1}$. Can you complete the description of this algorithm in order to compute a^x and estimate its computational complexity? Can this algorithm be better or equally efficient compared to the algorithm based on the binary representation of x?
2. Is it profitable to execute the first part of the algorithm from question 1 as follows?

$$\begin{aligned} a^2 &= a \cdot a, & a^3 &= a^2 \cdot a, \\ a^6 &= a^3 \cdot a^3, & a^9 &= a^6 \cdot a^3, \\ a^{18} &= a^9 \cdot a^9, & a^{27} &= a^{18} \cdot a^9, \\ &\text{etc.} \end{aligned}$$

   Or asked differently, is it helpful to additionally compute and store the powers a^6, a^{18}, etc., as well?
3. Develop a new algorithm for the computation of a^x based on the 5-ary representation of x. How well does this algorithm behave compared to the previous ones?
4. Does the "binary representation" algorithm always (for all x) work in the most efficient way or do there exist values for x, for which one can compute a^x faster?