Think logarithmically![1]

Maciej M. Sysło
Nicolaus Copernicus University, Faculty of Mathematics and Computer Science,
Chopin str. 12/18, 87-100 Toruń, Poland, syslo@mat.umk.pl
University of Wrocław, Faculty of Mathematics and Computer Science,
F. Joliot-Curie str. 15, 50-383 Wrocław, Poland, syslo@ii.uni.wroc.pl

Dedicated to **John Napier**
on the occasion of the 400th anniversary
of inventing logarithm in his book
Mirifici Logarithmorum Canonis Descriptio

Abstract
In this paper we discuss a number of algorithmic topics which we use in our teaching and in learning of mathematics and informatics to illustrate and document the power of logarithm in designing very efficient algorithms and computations – logarithmic thinking is one of the most important key competencies for solving many real world practical problems. We demonstrate also how to introduce logarithm independently of mathematical formalism using a conceptual model for reducing a problem size by at least half. It is quite surprising that the idea which leads to logarithm, is present in Euclid's algorithm described almost 2000 years before John Napier invented logarithm.

Keywords
logarithm, binary search, binary representation, exponentiation, Euclid's algorithm, Fibonacci numbers, divide and conquer, complexity

INTRODUCTION
Logarithm is a very important operation and function in computer science, not only because 'logarithm' is an anagram of 'algorithm'. Today it is difficult to imagine how one could study and work in computer science, on any level of education and study, not knowing this concept. Logarithm is formally introduced in high schools (at least in Poland) as a mathematical concept, see the next Section. In this paper we demonstrate how to introduce logarithm in informatics as well as in mathematics using conceptual models which have a computing flavour and are related to practical applications.

We consider here five themes/questions:
- how many times do we have to read a page in a paper dictionary to find a given word?
- how many bits does an integer occupy in a computer?
- how many multiplications are needed to calculate the value of the exponential function?
- how fast can we find the greatest common divisor of two numbers?

[1] A shorter version of this paper, co-authored by Anna B. Kwiatkowska, was presented at the conference "KEYCIT – Key Competencies in Informatics and ICT (KEYCIT 2014)", University of Potsdam, Germany, July 1-4, 2014 and appeared in the conference proceedings: T. Brinda, N. Reynolds, R. Romeike, A. Schwill (eds.), *KEYCIT 2014*, *Commentarii informaticae didacticae* (CID) 7, Universitaetsverlag Potsdam 2015, pp. 371-380. Its translation into Polish appeared in *Delta* 12, 2014, pp. 10-13.

- how many steps does a divide and conquer algorithm need when applied to a problem of size *n*.

A common feature of the answers to these questions is that all of them touch logarithm, directly or indirectly, and moreover they contribute to better understanding this concept and its role in relation to its practical applications in designing efficient algorithms and computations.

In our presentation, as much as possible, we avoid to use any reference to logarithm as a mathematical concept. Then, after informal introduction of the logarithmic function, we define logarithm in an algorithmic (operational) way.

We consider **logarithmic thinking** in problem solving activities as one of the most important facets of algorithmic and computational thinking (see Wing, 2016) and as a key competence necessary for general education to master high school and academic informatics as well as ICT studies.

We refer the reader to our school textbook (Gurbiel et al., 2002-2003), books (Sysło, 1997, 2015) and (Sysło, 1997, 2016) and papers (Sysło & Kwiatkowska, 2006) and (Sysło & Kwiatkowska, 2014) for detailed presentations of the topics discussed here.

LOGARITHM AS A MATHEMATICAL CONCEPT

Logarithm appears in the core curriculum of mathematics in high schools in Poland on both levels, basic and extended, in the sections on real numbers and functions. On the basic level of competences students are expected to be able to apply in calculations formulas which involve logarithm of product, quotient, and power of numbers, and on the extended level – also the formula for changing the base of logarithm.

The logarithmic function appears only in the mathematics curricula on the extended level, and students are expected to draw a graph of this function and moreover to use logarithmic function in modelling physical and chemical phenomena and also in some practical situations (which are not described in details in the curriculum). No application of logarithm in informatics (computer science) is mentioned in the mathematics curriculum and reviewing mathematics syllabi and textbooks shows also that in general no connections of logarithm to its applications in computing are subject of mathematics instruction in the classroom.

The paper (Webb et al., 2011) reports on promoting student understanding of logarithm using the instructional design theory of Realistic Mathematics Education, based on a principle that engagement in mathematics for students should begin within a meaningful context. However, no informatics context is considered.

This situation motivated us to write a paper (Sysło & Kwiatkowska, 2006) on contribution of informatics education to mathematics education in schools, where we discuss several problems of mathematical flavour included in the informatics curriculum which can contribute to better understanding and appreciating mathematical concepts and their use in contemporary mathematics and its use in computing. One of such concepts is logarithm which students consider to be hard to understand and to master – some of them used to explain that logarithm is not only one of the functions they do not like, but also, it is an inverse function.

Most of the concepts and topics discussed in this paper belong to discrete mathematics, known as mathematics of our times, mathematics of the computer era, mathematics of computing. We use the approach presented here in mathematics

instructions in schools K-12 as well as in teaching discrete mathematics at university level.

FIND A WORD, GUESS A NUMBER

A paper telephone book consists of 1000 pages. Find the page which contains the telephone number of Mr. Smith M.M. checking the smallest possible number of pages. Searching for a right page, students discover (or they already know) that the best strategy is **a binary search**, that is to keep splitting the remaining pages into two equal-size parts and to eliminate the part which does not contain the entry with Smith M.M., until only one page remains, which should contain Smith's telephone number. Instead of a telephone book one can choose an encyclopaedia or a dictionary.

The game of guessing an integer hidden in a given interval may serve the same purpose and can be used to activate whole class. We encourage students to play several rounds of this game in pairs and to fill in a table such as seen in Fig. 1.

Interval	Interval size	Hidden number	Number of questions asked	LOG	$\log_2$(Interval size)
[1, 80]	80	65	7	7	7
[51, 180]	130	100	7	8	8
[51, 180]	130	65			

Figure 1: A table for the results of the game to find a hidden number

Regardless of the hidden number, LOG is equal to the number of times the interval size is divided by 2 to obtain 1 (when an odd number is divided by 2 we always take 'a bigger half'), for instance: 80, 40, 20, 10, 5, 3, 2, 1. The last column in the table could be filled in later. For any game, numbers in last three columns should be close to each other.

Related topics and questions discussed with students:

- the importance of order among the elements in a dictionary and in an interval: how many times do we have to read a page in a paper telephone book of 1000 pages to find the owner of the phone number 1234567?
- in the case of a dictionary, when we have to find a word which begins with one of the initial letters in the alphabet, then we usually try to find this word on initial pages of the dictionary – such a strategy is called an **interpolation search**. We ask students to find in the Internet more information on this type of search, in particular regarding its complexity.

BINARY REPRESENTATION – A SIZE OF A NUMBER

Students usually know how to find a **binary representation** for a given nonnegative integer number n – such a representation is generated in successive divisions of n and the resulting quotients by 2. They divide n by 2 and take the reminder r (0 or 1) as the least significant digit of the representation. Then, apply this procedure to the quotient q and continue as far as the quotient is nonzero. For n = 23 we get $(10111)_2$, as in the table.

n	q	r
23	11	**1**
11	5	**1**
5	2	**1**
2	1	**0**
1	0	**1**

Then we ask students, how many binary digits has a decimal number n in its binary representation or equivalently, how much space in the computer memory we need for storing number n. To answer this question let us assume

that n needs k bits (however at this point we do not know the value of k). To find k, now we first have to determine the smallest and the largest numbers which can be represented on exactly k bits. The largest number has all k bits equal 1, hence we have:

$$(111\ldots1)_2 = 2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2^1 + 2^0 = 2^k - 1$$

It is easy to see that when we add 1 to this binary number we get 2^k, the next power of 2, hence we get the last equality above. On the other hand, the smallest integer which needs k bits has only 1 on its most significant position, therefore equals 2^{k-1}. Therefore we have the following inequalities:

$$2^{k-1} - 1 < n \leq 2^k - 1.$$

Now, adding 1 to all sides of these inequalities and taking $\log_2$ of all sides we get the inequalities:

$$k - 1 < \log_2 (n + 1) \leq k.$$

Since the number of bits k is an integer number, we have:

$$k = \lceil \log_2 (n + 1) \rceil,$$

where $\lceil x \rceil$ is the ceiling function and is equal to the smallest integer number greater than or equal to x.

We may conclude now that an integer number n occupies about $\log_2 n$ bits in a computer memory – this number is sometimes taken as the **size of *n*** in a computer. Moreover, since a binary search in an interval of size n corresponds to finding the binary representation of n, we conclude also, that the number of steps in a binary search in an interval of size n equals about $\log_2 n$.

Finally we may reverse our arguments and **define** $\log_2 n$ **algorithmically**:

> **Logarithm** $\log_2 n$ is equal to the number of steps in which, successive divisions of n and the resulting quotients by 2 lead to 1.

LOGARITHM

Now we have to convince students that the logarithmic function is very important in informatics – its importance lies in its rate of growth – although this function tends to infinity with n going to infinity but it is incomparably slower function comparing with the linear growth of n. The table, such as in Fig. 2 is the best illustration of our words, see also next section. We usually ask some students to verify some of the entries in the second column of such a table.

N	$\lceil \log_2 n \rceil$
128	7
1024	10
65 536	16
1 048 576	20
10^{10}	34
10^{50}	167
10^{100}	333
10^{200}	665
10^{300}	997
10^{500}	1661

Figure 2: Linear versus logarithmic growth

EXPONENTIATION

Fast exponentiation, that is calculation of a power x^n, is a crucial step in many real-world computations, such as compound interest, population growth, and public key cryptography (e.g., RSA and PGP). Practical values of the exponent n, for instance in RSA, are really very big numbers having hundreds of digits. We first ask student to calculate, how long a PFLOPS super computer (it performs 10^{15} multiplications per second) will compute x^n for a 'small' exponent n consisting of 30 digits, for instance when n = 123456789012345678901234567890, using the 'school' method which depends on performing $n - 1$ multiplications. Using the Windows calculator students can easily find that it will take more than … 10^7 years.

Our task now is to direct students to a faster method for exponentiation which, as in the previous two cases, reduces the exponent by half at each step. When the exponent is an even integer, they quickly come up with the formula $x^{2k} = (x^k)^2$ and when the exponent is odd we suggest to transform this case to the even case and they quickly come up with the formula $x^{2k+1} = (x^{2k})x$. Then students use these observations to find a method x^{23} can be calculated. Repeated application of these rules leads to the following transformations of the power:

$$x^{23} = (x^{22})x = ((x^{11})^2)x = (((x^{10})x)^2)x = ((((x^5)^2)x)^2)x = (((((x^4)x)^2)x)^2)x = ((((((x^2)^2)x)^2)x)^2)x$$

Therefore, to compute the power x^{23}, only 7 multiplications are needed, instead of 22 (squaring a number needs one multiplication).

Next task is to estimate how many multiplications are used by this algorithm for arbitrary n. We ask students to compare the binary representation of n = 23 = $(10111)_2$ with the order of multiplications above, going from right to left. It becomes clear that except the left most position, each bit 1 corresponds to multiplication by x and each position corresponds to squaring. Therefore, the number of multiplications in computing x^n by the above algorithm is equal to the number of binary positions in the representation minus 1 plus the number of 1's in the representation minus 1. Since, as we know, the length of the binary representation of n is about $\log_2 n$, the number of multiplication needed to calculate x^n is at most $2\log_2 n$.

Finally we can estimate how many multiplication performs the algorithm we have described for n = 123456789012345678901234567890. We have $2\log_2 n < 194$. It is tremendous achievement in complexity – it takes a moment to perform 194 multiplications instead of waiting 10^7 years to get the result. The table in Fig. 2 shows that calculating x^n for n with hundreds of digits takes a few thousands of multiplications.

The exponentiation algorithm described above can be expressed as a recursive procedure, see (Sysło & Kwiatkowska, 2014) for further discussion:

$$x^n = \begin{cases} 1 & \text{for } n = 0 \\ (x^{n/2})^2 & \text{for } n \text{ – even} \\ (x^{n-1})x & \text{for } n \text{ – odd} \end{cases}$$

EUCLID'S ALGORITHM

We begin this section with its concluding statement and the main observation of this paper:

Euclid was very close to invent logarithm, almost 2000 years before John Napier did it!

n	m	r_i
34	**21**	13
21	13	8
13	8	5
8	5	3
5	3	2
3	2	**1**
2	1	0

We first ask students to apply **Euclid's algorithm** to find the greatest common divisor GCD(n, m) of two given numbers n and m ($n \geq m$), for instance for n = 34 and m = 21. The algorithm generates a sequence of reminders (in the third column):

$$r_{-1}, r_0, r_1, r_2, \ldots, r_k$$

which begins with the given numbers $r_{-1} = n$, $r_0 = m$ and terminates when the reminder becomes equal 0, $r_k = 0$. The reminders are generated according to the following equations:

$$r_{-1} = q_1 r_0 + r_1, \qquad \text{where } 0 \leq r_1 < r_0$$

$$r_0 = q_2 r_1 + r_2, \qquad \text{where } 0 \leq r_2 < r_1$$

$$\ldots$$

$$r_{k-2} = q_k r_{k-1} + r_k, \qquad \text{where } 0 \leq r_k < r_{k-1}$$

and GCD(n, m) = r_{k-1}. The first equation corresponds to the first row in the table, and so on, and the last equation corresponds to the last row. The quotients q_i and reminders r_i satisfy:

$$q_i = r_{i-2} \text{ div } r_{i-1}, \qquad \text{and} \qquad r_i = r_{i-2} \bmod r_{i-1}$$

Now we want to investigate with students how many steps needs Euclid's algorithm to find GCD(n, m). We suggest first to compare the numbers in columns 1 and 3 in the table above. Students should notice that in the same row, the number in the third column is at least twice smaller than the number in the first column, that is, in the equation $r_i = r_{i-2} \bmod r_{i-1}$, r_i is at least twice smaller than r_{i-2}.

Therefore we want to show, that in general the reminder r from dividing n by m is not greater than $n/2$. We usually provide a geometric proof of this property in which there are two cases.

A. $m \leq n/2$. In this case, when n is divided by m, then the reminder is not greater than m, which is at most $n/2$.

n: ————————————

m: ————————————————

B. $m > n/2$. In this case, the reminder equals $n - m$ and since $m > n/2$, then $n - m$ is not greater than $n/2$.

n: ————————————

m: ——————————

Therefore in the sequence of numbers generated by Euclid's algorithm, each number is at least two times smaller than the number which appears two positions earlier. It reminds a sequence generated by a binary search except a sequence of the resulting numbers could be twice longer before it reaches 0 (the number smaller than 1). Hence we may conclude that:

Euclid's algorithm, finds NWD(n, m), where, $n \geq m$ in at most $2\log_2 n$ steps.

A challenging question for students is to find numbers n and m, for which Euclid's algorithm makes the largest number of steps. We have used a pair of such numbers in our example above.

FIBONACCI NUMBERS

The tendency of replacing a linear-time algorithm by a logarithmic-time algorithm, illustrated by the exponentiation, is also present in some other problems, e.g. in computing the values of Fibonacci numbers. The recurrence relation defining Fibonacci numbers can be used to design and implement a linear-time algorithm which avoids the use of very inefficient multiple (double) recursive calls. To obtain a logarithmic-time algorithm for Fibonacci numbers we have to use a system of two recurrence relations in which recursive calls have indices reduced by about half. To avoid inefficiency caused by mutual and multiple recursive calls in this system one can also implement this algorithm using iteration from the initial (boundary) cases, see details in (Sysło, 1998, 2015; Sysło & Kwiatkowska, 2014).

DIVIDE AND CONQUER

All the algorithmic methods discussed in this paper are based on divide and conquer technique in its broad sense – at each step of the solution process the problem size is reduced by at least half. In many applications of this strategy, the problem is reduced in size and moreover there are a number of subproblems (not only one like above) which are to be solved on each level of the problem decomposition. In such situations complexity formula contains some logarithmic components and also some polynomial terms. We usually illustrate such behaviour of divide and conquer technique using a merge sort algorithm together with its complexity analysis (for the problem size equal to a power of 2), see (Gurbiel at al., 2003; Sysło, 1997, 2016).

CONCLUSIONS

In this paper we focus on one of the most important concepts in informatics – logarithm – and show how we introduce it and present its properties and applications to students using a number of very popular building bricks of computer science. Logarithmic thinking is one of the most important key competencies when designing efficient solutions to real world practical problems.

All the topics discussed here are present in the textbook for informatics (Gurbiel et al., 2003) used in high schools in Poland. This textbook meets the curriculum and evaluation standards for school informatics approved by the Ministry of Education.

REFERENCES

Gurbiel, E., Hard-Olejniczak, G., Kołczyk, E., Krupicka, H., Sysło, M.M. (2003). *Informatics* (In Polish), Vols. 1 and 2, Textbook for high school. WSiP, Warszawa

Sysło, M.M. (1997). *Algorithms* (in Polish), WSiP, Warszawa; Helion, Gliwice 2016.

Sysło, M.M. (1998). *Pyramids, Cones and Other Algorithmic Constructions* (in Polish). WSiP, Warszawa; Helion, Gliwice 2015.

Sysło, M.M., Kwiatkowska, A.B. (2006). Contribution of Informatics Education to Mathematics Education in Schools. in: Mittermeir, R.T. (ed.) *ISSEP 2006*. LNCS, vol. 4226, pp. 209–219. Springer, Heidelberg.

Sysło, M.M., Kwiatkowska, A.B. (2014). Introducing Students to Recursion: a Multi-facet and Multi-tool Approach, in: Guelbahar, Y., Karatas, E., (eds.) *ISSEP 2014*. LNCS, vol. 8730, pp. 124-137. Springer, Heidelberg. 2014.

Webb, D.C., van der Kooij, H., Geist, M.R. (2011). Design research in the Netherlands: introducing logarithms using realistic mathematics education, *Journal of Mathematics Education at Teachers College*, Vol. 2, pp. 47-52.

Wing, J.M. (2006). Computational Thinking, *Communictions of the ACM* 49(3), 33-35.

Biography

Maciej M. Sysło, mathematician and computer scientist – academic and school teacher, author of informatics curricula, educational software, textbooks and guidebooks for teachers, member of national committees on education, Polish representative to IFIP TC3, recipient of awards and grants: Steinhaus (Poland), Car (Poland), Mombusho (Japan), Humboldt (Germany), Fulbright (USA), Best Practices in Education Award (2013), IFIP OSA (2014).