Think logarithmically!

Maciej M. Sysło[1], **Anna Beata Kwiatkowska**[2]
[1]Faculty of Mathematics and Computer Science
University of Wrocław
Joliot Curie 15
50-383 Wrocław, Poland
syslo@ii.uni.wroc.pl
[2]Nicolaus Copernicus University
Chopin 12/18
87-100 Toruń, Poland
{syslo, aba}@mat.umk.pl

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

Abstract: We discuss here 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 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

1 Introduction

Logarithm is a very important operation and function in informatics, not only because 'logarithm' is an anagram of 'algorithm'. Today it is difficult to imagine how one could study and work in informatics not knowing this concept.

Logarithm is formally introduced in high schools as a mathematical concept and we demonstrate here how to introduce logarithm in informatics and 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 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?
- 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 designing practical 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 as one of the most important facets of algorithmic and computational thinking and as a key competence to master high school and academic informatics as well as ICT studies. We refer the reader to our books (Sysło, 1997, 1998) and papers (Sysło, Kwiatkowska, 2006, 2014) for detailed presentations of the topics discussed here.

2 Logarithm as a Mathematical Concept

Logarithm appears in the core curriculum of mathematics in high schools in Poland on both levels, basic and extended. On the basic level, students are expected to be able to apply 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 on the extended level and students are expected to draw a graph of this function and to use this function in modelling some phenomena and also in some practical situations. However, no applications of logarithm in informatics are included in the mathematics curriculum.

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 the 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 computing. In this paper we restrict our attention to one of such concepts – logarithm.

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

3 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, checking the smallest number of pages. Searching for a right page, students discover 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 Smith, until only one page remains, which should contain Smith's telephone number. Instead of a telephone book one can choose a dictionary.

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

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

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

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 take 'a bigger half'), for instance: 30, 15, 8, 4, 2, 1. The last column could be filled in later. For any game, numbers in the 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, we usually try to find this word on initial pages – such a strategy is called an **interpolation search**. We ask students to find in the Internet more information about this type of search and its complexity.

4 Binary Representation – A size of a number

Table 2: Binary representation

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

Students usually know how to find a **binary representation** for a given non-negative 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 remainder 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 Table 2.

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 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.

5 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 it tends to infinity with n going to infinity but it is incomparably slower function comparing with the linear growth of n. Table 3 is the best illustration of our words.

Table 3: Linear versus logarithmic growth

n	$\log_2 n$
1024	10
1 048 576	20
10^{10}	34
10^{100}	333
10^{300}	997

6 Exponentiation

Fast exponentiation x^n is a crucial step in many real-world computations, such as compound interest, and public key cryptography (e.g., RSA). Practical values of 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, e.g. $n = 123456789012345678901234567890$, 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 exponentiation, which, as in the previous two cases, reduces the exponent by half at each step. When the exponent is even, they quickly come up with the formula $x^2k = (xk)^2$ and when the exponent is odd we suggest to transform this case to the even case and they quickly find that $x^{2k+1} = (x^{2k})x$. Then they use these observations to find how to calculate x^{23}. Repeated application of these rules leads to the following calculations:

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

Therefore, to compute x^{23}, only 7 multiplications are needed, instead of 22.

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, 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 the length of the binary representation of n is about $\log_2 n$, the number of multiplications needed to calculate x^n is at most $2\log_2 n$.

Finally we can estimate how many multiplication performs the above algorithm for $n = 12345678901234567890123456789$0. 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. Table 3 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:

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

7 Euclid's algorithm

We begin this section with the main observation of this paper:

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

We first ask students to apply **Euclid's algorithm** to find the greatest common divisor GCD (n, m) of n and m ($n \geq m$), for instance for $n = 34$ and $m = 21$. The algorithm generates a sequence of remainders (in the third column of Table 4):

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

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

$$\begin{aligned} r_{-1} &= q_1 r_0 + r_1, && \text{where } 0 \leq r_1 < r_0 \\ r_0 &= q_2 r_1 + r_2, && \text{where } 0 \leq r_2 < r_1 \\ &\ldots \\ r_{k-2} &= q_k r_{k-1} + r_k, && \text{where } 0 \leq r_k < r_{k-1} \end{aligned}$$

and GCD(n, m) = r_{k-1}. The first equation corresponds to the first row in Table 4, and the last equation – to the last row. The quotients q_i and remainders r_i satisfy:

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

Table 4: GCD

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

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 Table 4. 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, ri is at least twice smaller than r_{i-2}. Therefore we want to show, that in general the remainder 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 remainder is not greater than m, which is at most $n/2$.

n: ——————————

m: ——— ——— ———

B. $m > n/2$. In this case, the remainder 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 that 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 GCD(n, m), where $n \geq m$ in'at most $2\log_2 n$ steps.

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

8 Fibonacci Numbers

The tendency of replacing a linear-time algorithm by a logarithmic-time algorithm, illustrated by the exponentiation, is also present e.g. in computing Fibonacci numbers. The recurrence relation defining Fibonacci numbers can be used to implement a linear-time algorithm. To obtain a logarithmic-time algorithm we have to use a system of two recurrence relations in which recursive calls have indices reduced by about half, see (Sysło, Kwiatkowska, 2014).

9 Divide and Conquer

The algorithmic methods discussed in this paper are based on divide-and-conquer technique in its broad sense – at each step of a method the problem size is reduced by at least half and there are a number of sub problems, which are to be solved on each level of the problem decomposition. In such situations complexity formula contains some logarithmic components and some linear terms. We usually illustrate such behaviour of divide and conquer using a merge sort together with its complexity analysis (for the problem size equal to a power of 2) see (Sysło, 1997).

10 Conclusions

In this paper we illustrate how we introduce logarithm, the most important concept in informatics, and show its properties and applications 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 problems.

References

Sysło, M. M. (1997). *Algorithms* (in Polish), WSiP, Warszawa.

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

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, accepted for *ISSEP 2014* (Istanbul).

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.

Biographies

Maciej M. Sysło, mathematician and computer scientist, 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: Steinhaus, Car (Poland), Mombusho (Japan), Humboldt (Germany), Fulbright (USA), Best Practices in Education Award, IFIP OSA.

Anna Beata Kwiatkowska – academic and school teacher of informatics and mathematics, instructor at in-service courses for teachers, supervisor of school students gifted in informatics who participate in various competitions and in the Olympiad in Informatics, organizer of the Bebras Competition in Poland.