Applied Mathematical Sciences, Vol. 6, 2012, no. 15, 697 - 705

Mayan and Babylonian Arithmetics Can Be Explained by the Need to Minimize Computations

Olga Kosheleva

Department of Teacher Education
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
olgak@utep.edu

Abstract

Most number systems use a single base – e.g., 10 or 2 – and represent each number as a combination of powers of the base. However, historically, there were two civilizations that used a more complex systems to represent numbers. They also used bases: Babylonians used 60 and Mayans used 20, but for each power, instead of a single digit, they used two. For example, a number 19 was represented by the Babylonians as $19_B = 1 \cdot 10 + 9$ and by the Mayans as $34_M = 3 \cdot 5 + 4$. In this paper, we show that such a representation is not just due to historic reasons: for the corresponding large bases, such a representation is actually optimal – in some reasonable sense.

1 Formulation of the Problem

Traditional numerical systems use a single base. Most numerical systems are based on using a single number as a base. For example, in the decimal system, each natural number is represented by a sequence of decimal digits such as 2011, so that:

- the last digit 1 means ones (i.e., multiples of 10^0);
- the next digit means 10s (i.e., multiples of 10^1;
- the next digit means 100s (i.e., multiples of 10^2);
- the next digit means 1000s (i.e., multiples of 10^3),

- etc.

and the whole number means

$$2011_{\mathbf{10}} = 2 \cdot \mathbf{10}^3 + 0 \cdot \mathbf{10}^2 + 1 \cdot \mathbf{10}^1 + 1 \cdot \mathbf{10}^0.$$

Similarly, in a binary system, with base 2, the number $1011_{\mathbf{2}}$ means

$$1 \cdot \mathbf{2}^3 + 0 \cdot \mathbf{2}^2 + 1 \cdot \mathbf{2}^1 + 1 \cdot \mathbf{2}^0 = 11_{\mathbf{10}}.$$

Mayan and Babylonian systems were more complex. However, there are two exceptional systems, system used by two civilizations that actually did a lot of computations, especially related to astronomy. In general, they use the base system: the Mayan used base $b = 20$ and the Babylonian used base $b = 60$ (our division of an hour into 60 minutes and a minute into 60 seconds goes back to the Babylonians). However, their systems were more complex than the base systems: namely, to represent each digit from 0 to $b - 1$, they used a representation using a different "sub-base":

- the Mayans used 5; see, e.g., [1, 2, 3, 4, 6], while
- the Babylonians used 10; see, e.g., [1, 3, 4, 7, 8, 9].

Thus, e.g., the number 19 was represented:

- by the Mayans as $34_M = 3 \cdot \mathbf{5} + 4$,
- and by the Babylonians as $19_B = 1 \cdot \mathbf{10} + 9$.

As a result, numbers had a complex representation. For example, to get the Mayan representation of $386_{\mathbf{10}}$, we:

- first represent it in base 20, as $386 = 19 \cdot \mathbf{20}^1 + 6 \cdot \mathbf{20}^0$, and
- then represent each "digit" of this representation in a base-5 system: $19 = 3 \cdot \mathbf{5} + 4 = 34_M$ and $6 = 1 \cdot \mathbf{5} + 1 = 11_M$,

resulting in $386_{\mathbf{10}} = 3411_M$.

In other words, we get $386_{\mathbf{10}} = 3 \cdot \mathbf{5} \cdot \mathbf{20}^1 + 4 \cdot \mathbf{20}^1 + 1 \cdot \mathbf{5} \cdot \mathbf{20}^0 + 1 \cdot \mathbf{20}^0$. In general, we have a representation of the type

$$l_n s_n \ldots l_0 s_{0,M} = l_n \cdot \mathbf{5} \cdot \mathbf{20}^n + s_n \cdot \mathbf{20}^n + \ldots + l_0 \cdot \mathbf{5} \cdot \mathbf{20}^0 + s_0 \cdot \mathbf{20}^0.$$

For the Babylonian system, we similarly have

$$l_n s_n \ldots l_0 s_{0,B} = l_n \cdot \mathbf{10} \cdot \mathbf{60}^n + s_n \cdot \mathbf{60}^n + \ldots + l_0 \cdot \mathbf{10} \cdot \mathbf{60}^0 + s_0 \cdot \mathbf{60}^0.$$

A natural question: why this complexity? At first glance, these complex numerical systems seem to require more computations than simply using the systems with base 20 and 60: we will show that addition and multiplication become more complicated.

Indeed, when we add two digits in the decimal system (or in any other traditional system), the result – what goes into the corresponding digit of the result and whether we have a carry – does not depend on whether we are adding the last digits or the next-to-last digits or the first digits. In contrast, in the Mayan system, the result of adding two digits depends on whether we are adding last digits – in this case, 5 and above means a carry – or the next-to-last digit – in this case, 4 and above means a carry. Rules for multiplications are even more complicated. So, at first glance, these system may seem too complex to be practically useful.

What we do in this paper. In this paper, we analyze the computational complexity of the most complex of the two basic arithmetic operation – multiplication – in different notations, and we show that for large bases like 20 and 60, a complex system indeed leads to faster computations than simply using a positional system. Moreover, we show that for 20, the optimal complex system should be based (as it was) on the fact that $20 = 4 \cdot 5$, and for 60, the system should be based on the fact that $60 = 6 \cdot 10$.

The computational complexity will be estimated not based on any modern sophisticated algorithms, but rather based on the usual algorithms for multiplication, algorithms that use multiplication tables.

2 Analysis of the Problem

Why multiplication. Multiplication is much more complex than addition – because, crudely speaking, it consists of several addition. Thus, if we have a computational procedure that has approximately as many multiplications as additions, the largest amount of time is spent on multiplications. Hence, when we select a way to represent numbers, it makes sense to select the way for which multiplication requires the smallest amount of computation. So, to make the desired selection, let us analyze the complexity of multiplication.

Multiplication in the usual representation: analysis. Let us start by a representation in which we fix a base b, and represent each number as a sequence of digits ranging from 0 to $b-1$. In this representation, multiplication reduces to digit-by-digit multiplication – followed by summation of the results. Crudely speaking, each digit-by-digit multiplication means that we look up the

result in the multiplication table, and then place the result into the new table – in which we prepare everything for addition.

A multiplication table consists of $(b-1)^2$ results of multiplying each of the non-zero digits $1, \ldots, b-1$ by every other digit (multiplication by 0 always gives 0, so multiplication tables only contain non-zero digits).

The time to access this information is proportional to the largest distance to the farthest bit containing this information. In the brain, neurons are mostly located on the surface (see, e.g., [5]), so all the information is, in effect, contained in a two-dimensional area. To minimize the worst-case time, it is desirable to place the data in the most compact way, where the worst-case distance is the smallest. The most compact way of representing $(b-1)^2$ values is to store them in a 2-D square array $(b-1) \times (b-1)$, then the worst access time is proportional to $b-1$.

Placing the result into a place ready for addition requires one more computational step, so overall, the number of steps is b.

Multiplication in the more complex representation. Let us now assume that instead of storing the digits from 0 to $b-1$, we select a number b_s that divides b, i.e., for which $b = b_s \cdot b_l$ for some integer b_l, and then store two "sub-digits": a sub-digit that goes from 0 to $b_s - 1$ and a sub-digit that goes from 0 to $b_l - 1$. For example:

- the Mayans used $b_s = 5$ for $b = 20$; in this case, $b_l = 4$;
- the Babylonians used $b_s = 10$ for $b = 60$; in this case, $b_l = 6$.

Each original digit d is now represented by two "sub-digits" $d = d_l d_s$: a "small" sub-digit d_s (i.e., a sub-digit corresponding to values $0, \ldots, b_s - 1$ describing smaller numbers), and a "large" sub-digit d_l – i.e., a sub-digit corresponding to values $0, \ldots, b_l - 1$ describing larger numbers. Thus, to describe the product of two original digits $d = d_l d_s$ and $d' = d'_l d'_s$, we need to compute four products:

- one product of "large" sub-digits $d_l d'_l$,
- one product of "small" sub-digits $d_s d'_s$; and
- two products of sub-digits of different type $d_l d'_s$ and $d_s d'_l$.

Thus, we need three multiplication tables:

- a table for "large" sub-digits – it will be used once for each original digit;
- a table for "small" sub-digits – it will also be used once for each original digit, and

- a multiplication table describing results of multiplying a "large" sub-digit and a "small" sub-digit: this table will be used twice.

The first table contains the results of multiplying each of $b_l - 1$ "large" sub-digits by every "large" sub-digit. This table contains $(b_l - 1)^2$ results and therefore, access to this table requires time $b_l - 1$. Storing this result requires one more step, so we need time b_l to perform this sub-digit multiplication.

Similarly, the second table contains the results of multiplying each of $b_s - 1$ "small" sub-digits by every "small" sub-digit. This table contains $(b_s - 1)^2$ results and therefore, access to this table requires time $b_s - 1$. Storing this result requires one more step, so we need time b_s to perform this sub-digit multiplication.

Finally, the third table contains $(b_l - 1) \cdot (b_s - 1)$ results of multiplying two different types of sub-digits. As we have mentioned, in general, when we store N results, we need access time $\sqrt{N}$. Thus, for this table, the access time is $\sqrt{(b_l - 1) \cdot (b_s - 1)}$. We need to access this table twice, so we need access time $2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)}$. Storing the corresponding two results requires two more steps, so we need time $2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2$ to perform these two sub-digit multiplications.

By combining all these times, we conclude that the overall computation time is equal to

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2.$$

So, we arrive at the following conclusion:

Analysis: conclusion. For each base b, we select a complex method if there exists a value b_s that divides b and for which

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 < b.$$

If such a value b_s exists, as an optimal value, we select a value b_s for which the sum

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2.$$

is the smallest possible.

Comment. Once we have the original base b represented as a product of two integers, we can have the first one as b_s and the second one as b_l – or vice versa. For example, for $b = 20 = 4 \cdot 5$, we have two options:

- we can have $b_l = 4$ and $b_s = 5$, and
- we can also have $b_l = 5$ and $b_s = 4$.

From the above formulas, we can see that the computational complexity does not depend on which of the two numbers we take as b_s and which as b_l, Thus, in our selection based on the computational complexity, we will only select two factors of b, without specifying which one corresponds to "larger" sub-digits and which one corresponds to "smaller" sub-digits.

3 Explanation of Mayan and Babylonian Systems

Let us now apply to the above analysis to the cases $b = 20$ and $b = 60$ corresponding to the Mayan and the Babylonian number systems.

Case of $b = 20$: explanation of the Mayan system. For $b = 20$, the computational complexity (= number of computational steps) of the original method is equal to $b = 20$.

There are only two ways to represent 20 as a product of two non-trivial natural numbers (i.e., natural numbers different from 1):

- as $20 = 2 \cdot 10$, and
- as $20 = 4 \cdot 5$.

In the first case, without losing generality, we can assume that $b_s = 2$ and $b_l = 10$. Thus,

$$\sqrt{(b_l - 1) \cdot (b_s - 1)} = \sqrt{(2-1) \cdot (10-1)} = \sqrt{1 \cdot 9} = \sqrt{9} = 3$$

and so,

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 = 2 + 10 + 2 \cdot 3 + 2 = 20.$$

So, in this case, we do not gain anything: computations become more complex, but the computation time remains the same.

In the second case, without losing generality, we can assume that $b_s = 4$ and $b_l = 5$. Thus,

$$\sqrt{(b_l - 1) \cdot (b_s - 1)} = \sqrt{(4-1) \cdot (5-1)} = \sqrt{3 \cdot 2} = 2 \cdot \sqrt{3} \approx 3.46$$

and so,

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 \approx 4 + 5 + 2 \cdot 3.46 + 2 = 17.92 < 20.$$

So, this case indeed leads to the fastest computations – which is probably the reason why the Mayans select this case for their computations.

Case of $b = 60$: explanation of Babylonian system. For $b = 60$, the computational complexity (= number of computational steps) of the original method is equal to $b = 60$.

There are several ways to represent 60 as a product of two non-trivial natural numbers (i.e., natural numbers different from 1):

- as $60 = 2 \cdot 30$,
- as $60 = 3 \cdot 20$,
- as $60 = 4 \cdot 15$,
- as $60 = 5 \cdot 12$, and
- as $60 = 6 \cdot 10$.

In the first case, without losing generality, we can assume that $b_s = 2$ and $b_l = 30$. Thus,

$$\sqrt{(b_l - 1) \cdot (b_s - 1)} = \sqrt{(2 - 1) \cdot (30 - 1)} = \sqrt{1 \cdot 29} = \sqrt{29} \approx 5.4,$$

and so,

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 \approx 2 + 30 + 2 \cdot 5.4 + 2 = 44.8 < 60.$$

So, in this case, some selection is sub-digits leads to faster computations. Let us see which values b_s and $b + l$ lead to the fastest computations.

In the second case, without losing generality, we can assume that $b_s = 3$ and $b_l = 20$. Thus,

$$\sqrt{(b_l - 1) \cdot (b_s - 1)} = \sqrt{(3 - 1) \cdot (20 - 1)} = \sqrt{2 \cdot 19} = \sqrt{38} \approx 6.2,$$

and so,

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 \approx 3 + 20 + 2 \cdot 6.2 + 2 = 37.4.$$

In the third case, without losing generality, we can assume that $b_s = 4$ and $b_l = 15$. Thus,

$$\sqrt{(b_l - 1) \cdot (b_s - 1)} = \sqrt{(4 - 1) \cdot (15 - 1)} = \sqrt{3 \cdot 14} = \sqrt{42} \approx 6.5,$$

and so,

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 \approx 4 + 15 + 2 \cdot 6.5 + 2 = 34.$$

In the fourth case, without losing generality, we can assume that $b_s = 5$ and $b_l = 12$. Thus,

$$\sqrt{(b_l - 1) \cdot (b_s - 1)} = \sqrt{(5-1) \cdot (12-1)} = \sqrt{4 \cdot 11} = \sqrt{44} \approx 6.7,$$

and so,

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 \approx 5 + 12 + 2 \cdot 6.7 + 2 = 32.4.$$

Finally, in the last case, without losing generality, we can assume that $b_s = 5$ and $b_l = 6$. Thus,

$$\sqrt{(b_l - 1) \cdot (b_s - 1)} = \sqrt{(5-1) \cdot (6-1)} = \sqrt{4 \cdot 5} = \sqrt{20} \approx 4.5,$$

and so,

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 \approx 5 + 6 + 2 \cdot 4.5 + 2 = 22.$$

By comparing all the computational complexity results, we conclude that the case $b_s = 5$ and $b_l = 6$ indeed leads to the fastest computations – which is probably the reason why the Mayans select this case for their computations.

Case of $b = 10$. To doublecheck that our explanation is reasonable, let us show for the usual decimal system $b = 10$, we do not get any need for sub-digits. Indeed, for $b = 10$, the computational complexity (= number of computational steps) of the original method is equal to $b = 20$.

There are only one way to represent 10 as a product of two non-trivial natural numbers (i.e., natural numbers different from 1): as $10 = 2 \cdot 5$. In this case, without losing generality, we can assume that $b_s = 2$ and $b_l = 5$. Thus,

$$\sqrt{(b_l - 1) \cdot (b_s - 1)} = \sqrt{(2-1) \cdot (5-1)} = \sqrt{1 \cdot 4} = \sqrt{4} = 2$$

and so,

$$b_l + b_s + 2 \cdot \sqrt{(b_l - 1) \cdot (b_s - 1)} + 2 = 2 + 5 + 2 \cdot 2 + 2 = 11 > 10,$$

i.e., using sub-digits would slow down computations.

So, in the case of $b = 10$, the traditional representation (without sub-digits) is indeed the best one.

Acknowledgments

The author is thankful to Judith Munter for the useful discussion of Mayan mathematics.

References

[1] C. B. Boyer and U. C. Merzbach, *A History of Mathematics*, Wiley, New York, 1991.

[2] M. D. Coe, *The Maya*, Thames & Hudson, London, New York, 2011.

[3] G. Ifrah, *A universal history of numbers: From prehistory to the invention of the computer*, London, 1998.

[4] D. E. Knuth, *Seminumerical Algorithms*, Addison Wesley, Reading, Massachusetts, 1981.

[5] P. M. Milner, *Physiological Psychology*, Holt, New York, 1971.

[6] G. Morley, *Ancient Maya*, Stanford University Press, Stanford, 1946.

[7] O. Neugebauer, *The Exact Sciences in Antiquity*, Dover Publ., New York, 1969.

[8] B. L. van der Waerden, *Science Awakening*, P. Noorhoff, Groningen, 1954.

[9] B. L. van der Waerden, *Geometry and Algebra in Ancient Civilizations*, New York, 1983.

Received: August, 2011