UW Math Circle

Week 9 – Big Numbers

Today, we will learn about some really big numbers. The study of big numbers is called *googology*, which is actually how Google got its name!

In school, you first learned about counting (+1), then addition, which is just repeated counting:

$$a + b = a + \underbrace{1 + \cdots + 1}_{b \text{ times}},$$

then multiplication, which is just repeated addition:

$$a \times b = \underbrace{a + \cdots + a}_{b \text{ times}},$$

and then maybe even exponentiation, which is just repeated multiplication:

$$a^b = \underbrace{a \times \cdots \times a}_{b \text{ times}}.$$

1. Before we get too far, let's just review exponents briefly.
   (a) What is 2^4?

   (b) Explain why $7^6 \times 7^{13} = 7^{19}$.

   (c) Explain why $(11^3)^4 = 11^{12}$.

   (d) We know that $(a+b)+c = a+(b+c)$ and $(a \times b) \times c = a \times (b \times c)$. Does $(a^b)^c = a^{(b^c)}$?

   (e) A *googol*, for which googology is named, is the number 10^{100}. This is more than the number of particles in the observable universe, which is approximately 10^{80}! Even still, come up with a somewhat realistic situation that completes the sentence: "There are a googol _____." (or more than a googol, if you prefer)

Have you ever wondered what happens if you keep defining more operations like this? Donald Knuth introduced his *up arrow notation* to generalize this. Let $a \uparrow b$ just be a fancy way of writing a^b. Then define

$$a \uparrow\uparrow b = \underbrace{a \uparrow (a \uparrow (\cdots \uparrow a))}_{b \text{ times}}$$

(where a appears b times). Note that since $(a \uparrow b) \uparrow c \neq a \uparrow (b \uparrow c)$ (this was problem 1d), make sure to keep the parentheses in mind, and start on the right!

2. (a) What is $3 \uparrow\uparrow 2$?

   (b) What is $2 \uparrow\uparrow 3$?

3. Which is bigger: $3 \uparrow\uparrow 3$ or $(3 \uparrow 3) \uparrow 3$?

4. (a) Calculate $1 \uparrow\uparrow 1$, $2 \uparrow\uparrow 2$, and $3 \uparrow\uparrow 3$, possibly using a calculator.

   (b) The OEIS (online encyclopedia of integer sequences) is an amazing website that collects almost any interesting integer sequence you think of, at `oeis.org`. Type the answers you calculated for $1 \uparrow\uparrow 1$, $2 \uparrow\uparrow 2$, and $3 \uparrow\uparrow 3$ into the OEIS search box. What is a fun fact about $4 \uparrow\uparrow 4$?

What is we used even more arrows? Let's define

$$a \uparrow\uparrow\uparrow b = \underbrace{a \uparrow\uparrow (a \uparrow\uparrow (\cdots \uparrow\uparrow a))}_{b \text{ times}}.$$

Sometimes, we also write $\uparrow^2$ instead of $\uparrow\uparrow$, and $\uparrow^3$ instead of $\uparrow\uparrow\uparrow$, because it's easier to write. In general, we can define for any number n,

$$a \uparrow^n b = \underbrace{a \uparrow^{n-1} (a \uparrow^{n-1} (\cdots \uparrow^{n-1} a))}_{b \text{ times}}.$$

5. What is $2 \uparrow^3 3$? You can use a calculator.

6. Which is bigger: $4 \uparrow\uparrow 4$ or $3 \uparrow^3 3$?

7. Explain why $2 \uparrow^n 2$ is the same number for all n.

Wow, numbers grow really fast with arrows! Now let's shift gears and talk about a new operation. We call it the Ackermann operation, denoted @, and it is defined as follows: To calculate $a \mathbin{@} b$:

- If $a = 0$, then $a \mathbin{@} b = b + 1$.
- If $a > 0$ and $b = 0$, then $a \mathbin{@} b = (a - 1) \mathbin{@} 1$.
- If $a > 0$ and $b > 0$, then $a \mathbin{@} b = (a - 1) \mathbin{@} (a \mathbin{@} (b - 1))$.

As an example, suppose you wanted to compute $1 \mathbin{@} 1$.

- $1 \mathbin{@} 1$ matches the third line in the definition above. So it is equal to $0 \mathbin{@} (1 \mathbin{@} 0)$. So, we also need to find $1 \mathbin{@} 0$.
- $1 \mathbin{@} 0$ matches the second line above, which tells us $1 \mathbin{@} 0 = 0 \mathbin{@} 1$.
- $0 \mathbin{@} 1$ matches the first line, so $1 \mathbin{@} 0 = 0 \mathbin{@} 1 = 1 + 1 = 2$.
- Therefore, again using the first line, $0 \mathbin{@} (1 \mathbin{@} 0) = 0 \mathbin{@} 2 = 2 + 1 = 3$.

So $1 \mathbin{@} 1 = 3$.

8. (a) Fill in the unshaded squares in the below table for $a \mathbin{@} b$. (Hint: Go row by row! You will be able to use previous rows to compute later rows faster. The squares from the above example are already filled in for you.)

$a \backslash b$	0	1	2	3	4	5	6	7	8	9
0		**2**								
1	**2**	**3**								
2										
3										
4										

   (b) (Challenge) Do $4 \mathbin{@} 1$, the lightly shaded square.

9. Type your answers for $0 \mathbin{@} 0$, $1 \mathbin{@} 1$, $2 \mathbin{@} 2$, and $3 \mathbin{@} 3$ into the OEIS in order to search for the sequence $n \mathbin{@} n$. Find the right sequence and read the fun facts. What is $4 \mathbin{@} 4$?

10. (Challenge) It turns out that an equivalent definition for $a \mathbin{@} b$ is:

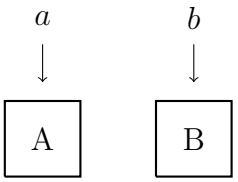
    - If $a = 0$, then $a \mathbin{@} b = b + 1$.
    - Otherwise, $a \mathbin{@} b = (2 \uparrow^{a-2} (b + 3)) - 3$.

    Show that this is equivalent by verifying the second and third cases from the original definition using this definition.

Why did Ackermann care about this function? In the 1920s, when the digital computer had not yet been invented and "computer" still referred to the profession in which a human worker sat in an office and did calculations, mathematicians were already thinking about computation. We all know how to compute things and follow instructions. But the mathematicians wanted to have a precise *model of computation*, so that not only could we say how to compute things, but maybe even show that certain problems cannot be computed, no matter how hard you try!

The mathematicians' first attempt is known today as *primitive recursion*. It's a little complicated, but let me try explain with two examples. For the first example, I will show a primitive recursive way to calculate $a + b$.

- At the beginning, boxes A and B hold the two *input* numbers, a and b.



- Then, we can calculate $a + b$ by following these instructions:

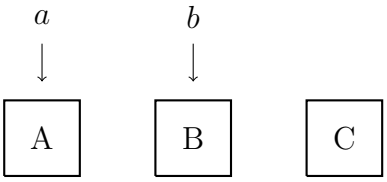
> Repeat the instructions inside this block for the number in box [B] times.
>
> > Increase the number in box [A] by 1.
>
> Output the number in box [A].

This is just a slightly longer way of writing the exact same ideas as

$$a + b = a + \underbrace{1 + \cdots + 1}_{b \text{ times}}.$$

For the second example, I will show a primitive recursive way to calculate $a \times b$.

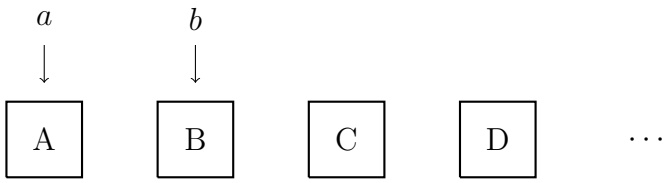
- At the beginning, boxes A and B again hold the two *input* numbers, a and b. This time, there is one more box that starts out empty.



- Then, we calculate $a \times b$ by following these instructions:

> Put the number (0) into box [C].
>
> Repeat the instructions inside this block for the number in box [A] times.
>
> > Repeat the instructions inside this block for the number in box [B] times.
> >
> > > Increase the number in box [C] by 1.
>
> Output the number in box [C].

In general, we say that something can be computed in a primitive recursive way if, given the input in some boxes as well as some number of extra empty boxes, as below,



we can calculate the answer using only the following 5 kinds of instructions:

> Put the number (__) into box [__].

> Copy the number in box [__] to box [__].

> Increase the number in box [__] by 1.

> Repeat the instructions inside this block for the number in box [__] times.

> Output the number in box [__].

Primitive recursive functions might sound too complicated and too simple at the same time, but they really can express almost any calculation you can think of! Mathematicians seriously thought for a while that everything computable is primitive recursive.

11. (Challenge) Write primitive recursive programs to compute the following:

    (a) On input a and b, compute a^b.

    (b) On input x, output $\begin{cases} x-1 & \text{if } x > 0, \\ 0 & \text{if } x = 0. \end{cases}$

    (c) On input x, output $\begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$

The Ackermann function was one of the first functions discovered to be perfectly computable (given enough time) by humans, in the sense that we can describe step-by-step instructions for how to do it at least, but is *not* primitive recursive! This discovery sent mathematicians on a decade-long search for better definitions of computability. In the 1930s, they ultimately settled on the equivalent definitions of *general recursion*, *lambda calculus*, and *Turing machines*, but those are beyond the scope of today's circle.

12. Give some intuition for why the Ackermann operation cannot be primitive recursive. (Hint: use problem 10, and think about 11a for inspiration)

13. If you have extra time, search for and read about Graham's function, TREE(n), busy beaver numbers, and/or Rayo's function. (These are listed in increasing order of growth rate, all of which grow faster than $A(n, n)$, and all of which have real importance in mathematics, except Rayo's function, which was just designed to win a big number competition at MIT.)