On counting, numbers, problem solving and their beauty

BMM

<2018-01-07 Sun>

Contents

1 Introduction

If you, like me, have wondered what the life and use of a number is, then read on. Let's peep through the keyhole of this note with the hopes of seeing the very little that will be allowed on the livelihood of numbers. These notes explore some techniques of counting, especially the techniques for counting the number of steps taken in solving a problem so that the problem solver can have an idea of the amount of work that needs to be done. The post also explores some of the basic techniques involved in making mathematical formulae. After all, a formula is just an expression of a pattern that arises in a collection of objects and which provides a tool for reporting information about the collected objects without necessarily examining all the objects in the collection.

1.1 Euler's idea of quantities, numbers and mathematics

Leonard Euler, a very prominent 18th century mathematician, commenced one of his seminal works titled, *Elements of Algrebra*, by investigating something called **QUANTITY**. He said, "whatever is capable of increase or diminution is called magnitude, or quantity. It follows from this declarations that there are many kinds of magnitude or quantity. The amount of money one has in the bank is quantified. Expenditures have the sly nature of decreasing this sum while some kind of profitable earnings have the virtue of increasing this sum. Similarly, the temperature goes up and down, the gas in a car goes up and down as the car is used. Almost everything changes in a way that can be quantified. This makes quantities to show up everywhere.

Consequently, there are so many types of quantity that different disciples have been developed for measuring these quantities. The accountant under the discipline of accounting tracks down quantities. The physicist tries to track down forms of quantities when she investigates things like temperatures, speeds, and motion. The statistician, the probability expert, the geometer, the banker, the real estate manager, and even the shepherd are all involved in the investigation of quantity. The various branches of mathematics and even most of the scientists are fields for investigating different types of quantity. Additionally, Euler defined Mathematics as the "science which investigates the means of measuring quantity." It appears that all quantities can be expressed or dressed up as numbers. Therefore, these beautiful objects called numbers are at the center of it all.

1.2 The sect that worshiped numbers

Around the 6th century BCE, there was a sect of philosophers called the Pythagoreans who were so fascinated by the beauty and nature of numbers that they worshiped numbers. The Pythagoreans realized that everything can be represented with numbers. One major contribution of this school is in the field of music theory. It is said that Pythagoras was passing by a blacksmith's shop one day and instead of being disturbed by the sounds of clashing metals pounding other metals, he realized some kind of harmony in the sounds that resonated. He stepped in to see, further, for himself, what was going on. He realized that different hammers and tools of different shapes, lengths and sizes produced different kinds of sounds as they stroke other surfaces.

It's compelling to note that lengths, shapes and sizes of such tools can be quantified and these quantities can be expressed as numbers. Perhaps, the sect had a night vigil that day :-)

They did further research on stringed musical instruments like the harp, the guitar and other such instruments. They saw that two strings with the same length that had the same tension and thickness ended up sounding the same when stroke. They discovered that if the lengths were different, but the same tension and thickness maintained, the sound produced where different. This difference testified to the pitches and the various pitches in the hands of a the good made some good music. Long story short, the idea of music intervals was borne. Today there are intervals in music with names like the Perfect fifth(3:2), the unison(1:1) and the octave(2:1).

1.3 On counting and numbers

Counting seems to be one of the prominent application of numbers. I remember learning how to count with the assistance of counting sticks. Counting is an activity that involves enumerating the elements that make up a collection. The purpose may be to identify each element with a unique number or in some cases to get a sense of the total number of elements in a collections. Essentially, counting is a good tool for accounting. Counting enables the counter to get an account of the elements of a collection.

As mentioned above, numbers are like the cloth that beautify quantities, almost in the same way a beautiful table cloth dresses a table, especially if this table cloth still reveals the exact form of the underlying table. Arithmetic is the science of numbers and this science provides us with tools and operations for manipulating numbers. A counter ends up with the total number of elements in a collection because of the addition operation that is provided by arithmetic. The basic operations of addition, subtraction, multiplication, and division are wonderful operations that permitting counters to make numbers out of other numbers. In the course of applying these operations on numbers, beautiful patterns emerge some of which we shall see in this post.

1.4 An example of counting and an entry into computer science

Suppose that you are a school bus driver in a mythical city who is transporting kids to and back from school. Your task is to visit several neighborhoods and make a number of stops to pick up the students. The company operating the school bus can simply determine the total number of stops in all the neighborhoods by counting the total number of times the bus will stop at all the stop points in the set of neighborhoods they deal with.

As simple as this problem sounds, it is a gateway into computer science.

As it stands, the bus driver and her company is in the process of solving a transportation problem. Their task is to transport a bunch of kids to and from school. Their technique is to use a school bus, visit a series of stop points, then board and unboard the kids. Suppose further more that, by some stroke of luck, it happens that at the first neighborhood there is 1 stop to pick up the kids, at the second neighborhood there are 2 stops to make, then 3 stops for the next neighborhood and so on. That is, If there are 10 neighborhoods numbered and symbolized from 1..10, then each neighborhood also has the number of stops that matches its symbol. Neighborhood number

10 will therefore have 10 steps.

The task is to count the total number of stops that the bus driver will make during one process of picking up or dropping off the kids.

It's simple. The total number of stops for 10 neighborhoods is revealed in the sum below:

$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 5050$ stops

You, or this venerable bus driver will have to make 5050 stops in such a neighborhood. Luckily, the problem was posed for a mythical city and not for a real world situation, lest all the kids will always be late for school.

If there are 5 neighborhoods with stops: $1, 2, 3, 4, 5$, then the driver will have a total of:

$1 + 2 + 3 + 4 + 5 = 15$ stops

The fact that the numbers that are been added have an underlying pattern, that is each current number surpasses the previous by 1, we can find a formula that captures this pattern and spare ourselves the hassle of manually adding a bunch of numbers.

What happens if we intend to find the total number of stops required for 1000 neighborhoods?

Such a task will need us to find the following sum:

$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + ...1000 = 500500$

As the number get bigger, it becomes more tedious to list them all and get their sum.

1.5 A formula for the counting problem above

What if there was a formula that we can use by plugging in the number of elements we want to solve and then get the sum. That is a formula that works for sums of the form:

$1 + 2 + 3 + 4 + 5 + 6 + ... + n$

where n is the maximum number of this sum.

That is, we intend to get the sum of the first n numbers, where n is a nonnegative integer. This n starts from 0 and extends to positive infinity. The formula for such a task is said to give the sum of the first n nonnegative integers.

Such sums are also called series. A series is simply the sum of a sequence of numbers. In this case, the sequence of numbers is:

$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...n$

Let S_n symbolize the sequence of the first n nonnegative integers.

Let $S(n)$ symbolize a formula that calculates the sum we seek.

$S(5) = 15, n = 5$

$S(10) = 5050, n = 10$
Our task is to build this formula.

1.6 The use of the bus driver problem above in computer science

It was highlighted that the task of the bus driver to sequentially stop at specified neighborhoods to pick up school kids was the method/technique of her company to solve the transportation problem. The number of stops was required for accounting and to know the cost that is inquired in the process.

This problem translates directly to a computer science problem, where the bus driver and her technique will be a computer algorithm and the company will be a computer programmer who is interested in knowing the amount of computational resources, especially computing time, that is spent in the process of applying the steps of the algorithm. The driver will perform the task of picking up kids at every stop. The algorithm will similarly perform a specified task at each point that is like a stop.

As such, the algorithm under consideration will perform 1 operation on the first run or stop, 2 operations on the second stop, 3 operations on the third stop and so on, until the nth stop where it performs n operations. The programmer who is interested in knowing the total number of steps for this algorithm will sum up the steps and the number of times as follows:

$1 + 2 + 3 + 4 + 5 + 6 + 7 + ... + n$

It follows that getting that formula for this problem and other problems of this form will be the right thing to do.

2 Mathematical Tools for making formulae

Now that we have seen how a simple counting problem becomes applicable in computer science, it is time to examine the mathematical tools that can be used to ease the process of counting. These tools will help us develop formulae freely and eventually we can reuse the techniques to investigate other counting problems as we study algorithms or techniques for solving problems.

The main tool we need is one that enables us to determine if the formula that we get is correct. In other words, we need techniques for testing our formulae and for proving their correctness. Computer programs are easily tested on a given set of input in order to get a partial view of their correctness. But the real technique for determining the correctness of a program for an infinite number of input relies on the use of mathematical tools. To be

precise, we need mathematical proof methods for determine the correctness of our formulae, algorithms and even programs for an infinite collection of test cases.

A few proof techniques include proof by contradiction, proof by construction and proof by induction. We shall use proof by induction for our problems.

2.1 A short look at mathematical induction

Inductive reasoning broadly involves reasoning from a particular thing to a more general view of the thing and other things it's related to. Inductive reasoning helps in making generalizations. Theories and laws are often generalizations or the expressions of phenomena that are true for a collection of elements. In so doing, the phenomenon is generally true because it is also true for each particular element of the collection. Induction becomes that tool that makes possible the leap from the particular to the general. Some phenomenon becomes a law if it's been proven that there is a high chance or probability that it will work on all the elements of the collection under consideration.

How are you sure the sun will rise tomorrow?

It's simple. The sun rose yesterday, the day before yesterday and so on, hence it will rise tomorrow. This is a form of inductive reasoning. Induction plays an important role in physics and other physical sciences, but at most, induction in these physical sciences relies hugely on chance. One can't be absolutely sure that the phenomenon captured will hold. There is simply a higher probability.

However, mathematics is an exact science. Mathematical induction therefore presents investigators with certainty. A correct proof by mathematical induction is both absolute and its a law in the absolute sense. It will always be true provided the conditions that were proven remain true. With this in mind, let us investigate the concept of proof by mathematical induction.

Mathematical induction is a technique for proving that a given property or concept is true for all elements of a given collection. In most cases, the elements in the collection are uncountable. They are so many that we can't count them in all our lifetimes, but we still, however, want to be sure that the concept we are interested in is true for all the elements. Such is the beauty that's in mathematics. One can proof statements and concepts for uncountable collections of elements or mathematical objects.

The main idea behind proofs is to test the elements under investigations and determine that the property is true for all of them. If the elements are

lined up as if they were waiting to get serviced by a lab technician, then this technician has the singular role to test each element, keep the result, test the next and keep the result, and so on until all are tested and if all pass the tests, then the idea behind the test becomes a theory or a law covering the said behavior or concept for all those elements. But as we mentioned, there elements are uncountable. The technician doing the test will never halt or conclude.

2.1.1 The main idea behind mathematical induction

Instead of examining every object one after the other and performing tests, what if the technician or tester tries to capture some behavioral or structural pattern that's evident in the set of test subjects. That is, tester should find a behavior that binds the elements of the set of test subjects together. Interestingly, a collection is always a collection because of some property or behavior that uniquely identifies them.

Induction is mainly a technique for inferring the future from the present or for inferring the present from the past.

A subtle feature of proof by induction is that the tester does not need to test all the elements of the set. She make setting assumptions based on a common property shared by all test subjects, and in so doing, the assumption can help her make conclusions that are true for all elements of the set, even though we mentioned that the set may be uncountable.

Here is a link to one of my notes on mathematical induction and other related concepts like recursion and recurrences.

2.1.2 Steps of mathematical induction

Induction does not require the tester to manually test every subject. There are 3 main steps involved:

1. Test the simplest element of the set or the most basic basic/elementary member of the set. This is known as the base case of the proof. If the set of elements is a collection of numbers from 1 to n, then test on 1 may be the base case.

2. Assume that the concept or test is true for a given number of elements of the set, say n-1 or n.

   n-1, n is the assumption that enables one to make the leap into the general. This assumption is know as *Induction Hypothesis* for it sets a

stepping stone upon which we can march to the infinite and make the idea under consideration general.

3. The inductive step. Here, we test if the fact that the concept is true for the base case in 1) above and also true for the induction hypothesis of step 2) makes the concept or test statement true for the next element n or n+1.

   If in the inductive hypothesis step, we assumed that the statement is true for $n-1$, and in the inductive step this assumption results in the truth of the statement for n, then it implies that the statement is true for the base case, the inductive case $n-1$, and for every other case n.

   Therefore, we say, the statement is true for the base case. Then we say if the statement is true for any n-1, then the statement will be true for any n.

2.1.3 The essence of proof by mathematical induction

Another look at the steps involved.

Show that the base case is true. Now assume that the statement is true for case $n-1$, and show that the truth of case n-1 secures the truth of case n.

By virtue of this reasoning, it is clear that the truth of case 1 implies the truth of case 2. That is, n-1 in this case is 1. Since n-1 secures the truth of case n, then 1 secures the truth of case 2. For the next case in line after $(n-1)$ is $(n-1)+1$ which is n. And case (1+1) is 2. Consequently, the truth for case 2 secures the truth for case 3 which secures the truth for case 4, and which then secure the truth of case 5, and so on to infinity.

As such, we can prove that a conceptual statement or hypothesis is true for all the elements of an infinite set just by capturing a pattern that recurs in the behavior or structure of the elements of the set.

2.2 Using mathematical induction on the counting problem

2.2.1 The formula for the counting problem

Side notes on formulae as simple abstract machines

Formulae can be regarded as abstract mathematical machines which are operated by the persons exercising with them. In this case, we are the operators.

In the case of a computer, there may be a stored procedure or simple computer program that performs the operation that is represented by a formula. So when we are thinking, we do not feel our minds with the details of the operation. If the formula that is being used has been proven to be correct and authentic for all the values it deals, then one is sure that any value we get as a result is correct. Hence, there is no need to worry about the detailed operational steps. In so doing, we focus simply on the results of operations when dealing formulae and think less of the step by step techniques for obtaining those values.

The sum of the first n numbers posited above is not a new problem. A forumala was discovered by a good man named Gauss who was a famous mathematician of the 18th century.

Let's develop Gauss's simple solution here.

$S(n)$ is a symbol of the formula that will generate the result we are looking for.

Let $S(n) = 1 + 2 + 3 + 4 + ... + n, n \geq 0$

That is, let S(n) be a formula or little abstract device where we shall plug in n and then get back the required result.

For example, $S(5) = 15$ and $S(10) = 5050$

Gauss first of all states $S(n)$ as shown below:

$S(n) = 1 + 2 + 3 + 4 + ... + n$ ..... (1)

He reversed $S(n)$ in (1 above to get:

$S(n) = n + (n-1) + (n-2) + ...1$ ..... (2)

Then, he added (1) and (2):

$2S(n) = (n+1) + (n-1+2) + (n-2+3) + ...(n+1)$

Observe that $(n-1+2) = (n+1)$ and $n-2+3) = (n+1)$.

This will happen for all the n elements of (1) and (2).

$\Rightarrow 2S(n) = (n+1) + (n+1) + (n+1) + ... + (n+1)$

$\Rightarrow 2S(n) = n(n+1)$

because there are n of these $(n+1)$ elements.

Now, $2S(n) = n(n+1)$

What happens if we divide both sides of the equation by 2 ?

$\frac{2S(n)}{2} = \frac{n(n+1)}{2}$

Therefore, $S(n) = \frac{n(n+1)}{2}$

Gauss concluded that $S(n) = \frac{n(n+1)}{2}$

2.2.2 Testing Gauss's solution

1. Manual Tests

n	S(n)
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

2. Proof by Induction

   $S(n) = 1 + 2 + 3 + 4 + 5 + ...n$ .....(a)

   (a) **Step 1**: Base Case. $n = 1$ $S(1) = 1$
   From the table in the section above, $S(1) = 1 for n = 1$.
   So we conclude that \(S(1) = 1).

   (b) **Step 2**: Inductive hypothesis.
   I assume or suppose that the formula, $S(n)$, is true.
   Let the symbol k represent any given number that I want to test against the formula $S(n)$.
   I hypothesize that for this test element(or number) k, it passes the test and this test result implies that the next value, that is $k + 1$, will also pass test. So, I say, all values of n will pass the test. And if all values of n pass the test on $S(n)$, them the formula $S(n)$ works for all values of n.

   (c) **Step 3**: Inductive Step.
   Now we conduct the necessary experiment that confirms the truth in the hypothesis stated in step (2) above and use the result to conclude the proof of the formula, $S(n) = frac12n(n + 1)$
   $S(k) = \frac{1}{2}k(k + 1)$ from the assumption in (2) above.
   We then try to see if the truth of $S(k)$ implies the truth of $S(k+1)$.
   From the manual steps of the procedure that returns that values of $S(n)$, we get the result by adding up the values from $1...n$. Therefore, the last value added is n itself. And this value n is added to the previous sum, that $S(n - 1)$.
   Therefore,

$S(k+1) = S(k) + (K+1)$

But, we assumed earlier above that S(k) is true. So we have to replace $S(k)$ with $\frac{k(k+1)}{2}$.

$\Rightarrow S(k+1) = \frac{1}{2}k(k+1) + (k+1)$

$\Rightarrow S(k+1) = \frac{k(k+1)}{2} + \frac{k+1}{1}$

$\Rightarrow S(k+1) = \frac{k(k+1)}{2} + \frac{k+1}{1} = \frac{k^2+k+2k}{2}$

$\Rightarrow S(k+1) = \frac{k^2+2k+k+2}{2}$

$\Rightarrow S(k+1) = \frac{k^2+3k+2}{2}$

Interestingly, the numerator of the fraction above is a polynomial which can be factorized into the value below:

$(k+1)(k+2) = k^2 + 2k + k + 2 = k^2 + 3k + 2$

$\Rightarrow S(k+1) = \frac{(k+1)(k+2)}{2}$

And it follows that $S(k+1)$ has the same form as$S(k)$, except that 1 has been added to the values of k.

Similarly, $S(k+1)$ passed the test when we assumed that $S(k)$ was true and when we added this assumed value to $(k+1)$.

Therefore, $S(k+1)$ is true wnen $S(1)$ and $S(k)$ are true. Hence, $S(n) = \frac{n(n+1)}{2}$ is true for all values of n.

At this point, the formula $S(n)$ has been proven to be true for all values of n.

2.3 Recurrences

2.3.1 About Recurrences

Recurrences are the next useful tools for capturing solutions to programs. These captured solutions, in the form of recurrences, serve as good models for further investigation into the nature of the problem. A recurrences will capture the exact nature of the solution and provide it as an artifact that one can play with and eventually carry out more experiments as needs arise.

The word *recurrence* is made of the words *re* and *currence*. The word *currency* is directly related to the word *currence*, which captures the state of being current. Currency expressed the idea that an event or a given thing is the standard or being used, at the moment. As such, *currency* applies to the monetary tokens that are standard. All in all, *currence* has to deal with items/events or tokens that are in use. *Re* commonly refers to doing something again as in *repeat*, *re-iterate*, *recycle* and *reconsider*.

It follows therefore that the term *recurrence* relates to idea of making something a standard over and over again. It refers to repeating a certain standard everytime in a series of events. This appears naturally in the cosmos of computation. There are often computational operations that involve the repetition of a particular operation a given number of times. For example, the exponentation operation is simply a process of repeated multiplication of a number by itself, this number is known as the base, a given number of times, known as the exponent or power.

$2^5 = 2x2x2x2x2 = 32$

In the example above, 2^5 symbolizes the repeated application of multiplication of 2, by itself, 5 times. This repetition of the multiplication operation can be seen as a recurrence of multiplication. Each round of multiplication is known as an iteration and it follows that there are 5 iterations in the example above. Therefore, there is an operation called multiplication, and there is a number of times, 5 in this case, to iteratively apply the multiplication operation. In the course of repeatedly(iteratively) applying multiplication to a given number, a running result is kept and it is this result that is reported at the end of the process. The table below shows the process of computing the exponent 2^5.

Iteration number	operaton	result
1	2 x 1	2
2	2 x 2	4
3	2 x 2 x 2	8
4	2 x 2 x 2 x 2	16
5	2 x 2 x 2 x 2 x 2	32

At the end of the 5th iteration, the result is 32 and that value will be returned as the emergent result of the exponentation operation.

Recurrences, are tools for capturing repetitive processes like this, but they are special in the sense that they capture processes which are self-referential. A self-referential process is a process that involves invoking itself as one of the steps in the solution. The exponententiation operation on 2^5) is thesame as the combination of 2 and the exponententiation of 2^4). That is $2^5 = 2x2^4$. It follows that $2^4 = 2x2^3$. It is evident that the exponententiation operation on a given value is identical to the application of the exponententiation process on smaller cases of the problem. Recurrences are great tools for capturing such self-embedding and self-referential processes.

2.3.2 Recursion and Recurrences and Induction

Recurrences are like devices that capture the abstract concept of *RECURSION*. Recursion captures the concept where an operation proceeds by repeatedly applying itself(the original process) on smaller problem input sets as part of the solution to the overall problem. It allows for concept or solution reuse on smaller instances of the problem. Through recursion, a solution to a problem evolves as a repetition of the original plan of action but on smaller input sets such that each further application or iteration reduces the problem further until a point is reached where there is no additional problem to solve. At this point, the smaller solutions that were acquired in each iteration that applied the process on a smaller input set, are then combined to form the overall solution.

Recursion and recurrences therefore naturally apply to the technique of *DIVIDE and CONQUER*. Divide a problem into smaller versions of that are identical to the original problem, then solve each of these smaller problems and combine their results. Since the smaller problems have thesame form as the original problem, the same machinery can be used to solve these smaller problems. Recurrences serve as abstract machines that take the form of the problem solvers which we shall apply to the divisions we arrive at. Recursion captures the commands that invoke the recurrences and on curated sizes of the original problem.

2.3.3 The relationship between recursion, recurrences and induction

As observed above, induction provides a method for marching from a particular case to a general case. It is through induction that theories and laws can be investigated for a theory or law is the embodiment of a rule that works for all members of a collection. Induction enables the testing of the rule on all members of the collection. If one were to think of testing these elements against certain rules as a physical process of conducting scientific experiments in actual labs, then the recurrence will take the place of an actual machine that works on a given test subject; in this case, a number under investigation will be a test subject.

Induction works on a set of elements that are related such that one can select the next element from the current element. Elements that are arranged in sequences are easy to investigate using induction. Again, a sequence is like a line up of elements where each element starting from the first, usually at position 0, has a successor which is the next position in the line up, or

a precedessor, which is in the previous position in the line up. The first element, however, has no predecessor and this simple property makes it the first of the sequence. Similarly, the last element in line has no successor.

Perhaps a look at the natural numbers will throw some light on the idea of sequences. The natural numbers are the counting numbers: 0, 1, 2, 3, ... They are also know as the positive counting integers and 0. Interestingly, the natural numbers are limitless in number. There is an infinite number of natural numbers. That is, these numbers start from 0 and never end. Below is a slice of the sequence of natural numbers:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,...

The line up above starts from 0, which is at the first location. There is no last number. It is evident that one can obtain the successor of every number, starting from 0 itself, by simply adding 1.

```
def showSucc(n)
    if n == 0
        return
    end

    showSucc(n-1)
    sum = n + 1
    puts "(#{n} + 1) = #{sum} and #{sum} is the successor of #{n}."
end

showSucc(10)
```

```
(1 + 1) = 2 and 2 is the successor of 1.
(2 + 1) = 3 and 3 is the successor of 2.
(3 + 1) = 4 and 4 is the successor of 3.
(4 + 1) = 5 and 5 is the successor of 4.
(5 + 1) = 6 and 6 is the successor of 5.
(6 + 1) = 7 and 7 is the successor of 6.
(7 + 1) = 8 and 8 is the successor of 7.
(8 + 1) = 9 and 9 is the successor of 8.
(9 + 1) = 10 and 10 is the successor of 9.
(10 + 1) = 11 and 11 is the successor of 10.
```

Also, each natural number's predecessor is obtaind by subtracting 1:

```
def showPred(n)
```

```
    if n == 0
        return
    end

    diff= n - 1
    puts "(#{n} - 1) = #{diff} and #{diff} is the predecessor of #{n}."
    showPred(n-1)
end

showPred(10)
```

```
(10 - 1) = 9 and 9 is the predecessor of 10.
(9 - 1) = 8 and 8 is the predecessor of 9.
(8 - 1) = 7 and 7 is the predecessor of 8.
(7 - 1) = 6 and 6 is the predecessor of 7.
(6 - 1) = 5 and 5 is the predecessor of 6.
(5 - 1) = 4 and 4 is the predecessor of 5.
(4 - 1) = 3 and 3 is the predecessor of 4.
(3 - 1) = 2 and 2 is the predecessor of 3.
(2 - 1) = 1 and 1 is the predecessor of 2.
(1 - 1) = 0 and 0 is the predecessor of 1.
```

The successors and predecessors of the natural numbers show us their relationship and this relationship can be exploited and used to walk up and down the sequence. Operating on these numbers with recurrences become simple. At each iteration, one can arrive at the predecessors or successors by simple arithmentic operations of addition, subtraction and the same operations work for all the numbers. As such, recurrences, recursion and becomes naturally available on natural numbers.

Let's use this property to investigate the addition operation and how we can use recurrences, recursion and induction to get the result of adding 2 numbers.

2.3.4 The addition operation using recurrences, recursion and induction

Remember, one of the most important aspects of proof by mathematical induction is the inductive hypothesis, where one assumes that a given formula works for a range of elements in the input. For example, assuming that the formula works "for all n". Such a leap is very helpful when dealing with recursion and recurrences.

Now, let's develop the addition operation.

What does is it mean to add 2 numbers?

Commonly, addition is a process of combining 2 groups of elements into one new group called the sum. The sum becomes the result of placing the constituent elements together.

Addition is a natural operation on natural numbers because of the predecessor and successor properties. In fact, the successor of any natural number is obtained by performing an addition operation. By adding 1 to any natural number, we obtain its successor. This gives an important insight into the nature of addition. Two numbers can therefore be added by adding the number 1 to one of the numbers repeatedly, a given number of times in which this number of times is equal to the magnitude of the other number in the addition process. That is, $X + Y$ is as simple as adding 1 to X, Y times.

$\Rightarrow 2 + 3$ is:

1. $2 + 1 = 3$ – the first iteration where 1 is added to $X = 2$
2. $(2 + 1) + 1 = 4$ – the second round where 1 is added to $X = 3$
3. $((2 + 1) + 1) + 1 = 5$ – the final round

After the third round, 1 has been added to 2 3 times, and this makes 5 the result.

As such, addition of 2 numbers is a process of finding an indexed successor or predecessor.

Suppose there is an operation or machinery called ADD that takes as input 2 natural numbers, call them X and Y. ADD then returns the sum. We are going to spill the guts of this ADD operation.

$ADD(2, 3) \Rightarrow 5$

Here is the definition and construction of ADD.

Let ADD(X, Y):

1. If X is 0, then go to step 4
2. let X1 = X-1 and Y1 = Y+1
3. ADD(X1, Y1), that is X=X1 and Y=Y1
4. return Y as the result.

The above definition says that to add 2 numbers using the ADD device, first test if the first number, that is X, is 0. If this first number is 0, then

the answer to $ADD(X, Y)$ is Y. The answer, if $X = 0$ is Y because anything added to zero is the thing itself.

Next, if the first number, X, is not zero, then the first thing to do is to reduce X and to obtain its predecessor. Then, we increase y by obtaining its successor. And finally, we repeat the ADD operation on the new elements we now have($X1$ and $Y1$).

Hence, ADD is a recurrence, for it is this abstract device or machinery that we use over and over again. Syntactically, the step that reapplies ADD on $X1$ and $Y1$ is a recursive call. And finally, if X is not 0, we assume that we still got operations to perform. It is thanks to the inductive nature of the natural numbers that we can confortably use the recursion and recurrences for this problem.

Additionally, recurrences and recursion can be easily translated into computer languages. As result, it becomes easy to develop computer programs that work in the same way that a recurrence works. After all, a recurrence is an abstract mathematical machine and computer programming provides facilities for performing most of the relatively more abstract mathematical modoels of phenomena.

Below is a computer program that performs the ADD operation described above:

```
def ADD(X, Y):
    if X == 0:
        return Y
    X1 = X-1
    Y1 = Y+1
    return ADD(X1, Y1)

for i in range(1, 10):
    print "ADD(%d, %d) is %d" % (i, i-1, ADD(i, i-1))
```

```
ADD(1, 0) is 1
ADD(2, 1) is 3
ADD(3, 2) is 5
ADD(4, 3) is 7
ADD(5, 4) is 9
ADD(6, 5) is 11
ADD(7, 6) is 13
ADD(8, 7) is 15
ADD(9, 8) is 17
```

One may wonder why there appears above the use of "+" in the very ADD operation that we are defining. The natural numbers come with the numbers and some primitive operations, two of which are: DECREMENT and INCREMENT. DECREMENT returns the predecessor of any given natural number, except 0. INCREMENT returns the successor of every natural number. DECREMENT receives a number, say X, and returns $X - 1$ as its result. INCREMENT receives a natural number, say X, and returns $X + 1$. Hence, the use of "+" or "-" is merely an invocation of the primitive DECREMENT and INCREMENT operations.

2.3.5 A brief look at iterations and the mathematical theory of summations

A common application of a recurrence has been shown to contain a minimal set of simple operations that are performed on varying input a given number of times. Simply put, there is a repetitive process going on and we need tools to capture and express such ideas. In programming languages like C, there exist constructs for expressing repitions. The **FOR**, **WHILE** and **DO WHILE** keywords are used to express and command the computer to perform certain operations repeatedly. Also, the ability of a procedure to invoke itself as one its steps is also another way of expressing or commanding the computer to repeat a process.

In the case of repeatedly applying the summation process to a sequence, mathematicians use the sigma symbol: Σ.

An repeated process usually has a starting point and an endpoint. The range between the start and the end is usually captured using numbers. For example, 0 could be the start and 9 the endpoint. As a result repeating a process while counting from 0 to 9 makes the process to be executed 10 times, for 0 to 9 spans a range of 10 digits.

Hence, programming constructs that involve repetition provide means for specifying the start and end cases.

From examples above the **range** keyword of the python programming language was used. We said, **for i in range(0, 9)** to specify the iteration of a process or procedure 10 times, where each iteration has a number and the first iteration had the number 0 and the last one, 9.

In mathematics, $\sum_{i=0}^{9} i$ means that we start counting from i = 0 and we need to perform the addition of i to each value it assumes, i.e i = 0, 1, 2, ... 9, until we reach the last count 9. This is identical to the python code fragment below:

```
sum = 0
for i in range(0, 9):
     sum = sum + i
print "The sum from 0...9 is %d" % sum

The sum from 0...9 is 36
```

The mathematician will say: $\sum_{i=0}^{9} i = 36$. Hence σ symbolizes repetitive addition of successive values as the iteration goes from the lower limit to the upper limit.

2.4 Conclusion of the mathematical tools needed

The main mathematical tools that were briefly observed above are:

1. mathematical induction
2. recurrences
3. recursion
4. summation
5. sequences

With these tools, one can continue the venerable task of counting and solving problems.

Recurrences and recursion enable the investigator to setup test equipment and devices. These tools allow one to setup up machinery that will be used to process the elements of the collection in question. The natural numbers was a practical example used in demonstrating some aspect of the nature of recurrences.

The relationship that exists between the natural numbers is an example of a sequential relationship. That is, there is a predecessor and a successor. There are other objects, especially in the computational world, that have more complex relationships. For example, if one were to capture the family tree of her family mathematically, then will arise more complex relationships that have branches that take bends to find successors and predecessors that are not as straigtforward as was the case in the sequence of natural numbers.

Also, if one were to sketch up a math representation of a city, its restaurants and roads that lead to them as lines, then the relationship that will

arise will be more complex that the straigtforward relationship between the natural numbers.

Recurrences can be used to work on related objects, but there will be different means of walking up, down, left, right etc,. along the paths that trace the various objects under consideration.

Recurrences, recursion, summations, sequences etc,. are tools we use for investigating the nature of computational problems including counting and programming problems. Mathematical induction enables us to carry out proofs on our findings and to cement theorems and laws that will later on save us time.

3 A couple of examples that use the mathematical tools considered above

Algorithms are central to computer science and programming. An algorithm is the technique that is used in solving particular problem. An algorithm embodies that precise steps that are needed in transforming a set of input into a desired set of output. In the business world, some companies have factories that transform raw materials(input) into finished products(output). The steps that are taken to realize these transformations are akin to the steps that an algorithm embodies, which the programmer or problem solver will implement in an attempt to transform input(raw materials) to output(finished products).

One of the most important operations for most business is the analyses of their operations so that they can get a sense of how much they are spending and eventually they can know if they make profits or losses. Most importantly, businesses try to minimize cost and maximize output and results.

Similarly, the computer programmer will need to analyse her algorithms in order to obtain a sense of expenses in terms of computing time, space and other costly resources like bandwidth, printer's ink, scanner's power consumption etc,.

One simple technique used to analyse algorithms is to obtain a total count of the main operations in a algorithm and to return a result of how much resources they consume. This result is often good enough to give a sense of the cost of the algorithms.

Fundamentally, analysing an algorithm involves a process of counting. Below are a few examples that show how this counting is done.

3.1 Counting the number of operations in a computer programming

Suppose that one has an algorithm that contains a main loop that does most of the work. The loop is the central piece of the algorithm such that all the costs incurred during its execution is related to this loop. How does one get to count the overall operations and get a sense of the costs.

The 2 main costs that are mostly considered are computing space and time. We often want algorithms that are fast. That is, we algorithms that run in the shortest time possible. Added to that is the need to use less space in storing the objects that the algorithm transforms. Storage is cheap but it's not free. Moreover, if too much is stored over a large area of memory, it will take more time to track down a piece of information. All in all, space and time are 2 essential resources that are regarded in the analysis of algorithms.

Back to our main loop example. Suppose that the main loop above has to apply a simple process *10 times and each time, this operation costs 1 unit of time*. It follows that **a cost of 1 unit over 10 iterations** will give us 10 units of time.

It is often easier to solve a problem by generalizing it. Suppose that the loop runs n times and each of the n times costs 1 unit of time. It follows again that the algorithm will cost n units in total.

The algorithms analyst will conclude that the algorithm runs in time that is directly proportional to the amount of input. That is, an input size of 10 will cost us 10 units of time. An input size of 50 will cost us 50 units of time. Thus, an input size of n will cost n units of time.

But such algorithms are in extinction at this point in human history :-)

It is more common to find algorithms whose main loops have more complicated operations. Most of these operations cost varyingly in the different rounds or iterations of the loop.

3.1.1 Example of a loop with varying costs

A simple example was demonstrated above using the school bus driver. Suppose that we have in place a main loop that runs 10 times, but in each iteration indexed by the a given number, say **i** such that i=1, 2, 3,... 10, the operations in that round cost **i** units. It follows that the overall cost will be the sum of all the costs of all the rounds. But the first round costs 1 unit, the second iteration costs 2 units, the third 3,... and the last iteration costs 10 units of time. This implies that the sum total is:

$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55$

And we are back to the expression that we got when we tried to analyse the cost of having the bus driver transport kids to and from school.

And such a sum is expressed as: $\sum_{i=1}^{1} 0i = 55$

In the most general sense, a main loop on n objects such that each iteration, i, contributes i units to the overall costs will have the following mathematical expression:

$\sum_{i=1}^{n} i$

3.1.2 Using recurrences, and recursion for the varying cost problem

$\sum_{i=1}^{n} i$ is the sum of the first n numbers. We worked on its solution and proof using Gauss's method. But let's use the idea of recurrences to get the job done.

Let S_n symbolize the operation that sums up the first n numbers. Therefore, $S_10 = 55$.

And $S_n = 1 + 2 + 3 + 4 + ...n$

And $S_n = \sum_{i=1}^{n} i$

Let's begin our experiment by taking up a few values of n starting from 0 and going towards 10.

1. $S_0 = 0$
2. $S_1 = 0 + 1 = S_0 + 1 = 1$
3. $S_2 = 0 + 1 + 2 = S_1 + 2 = 3$
4. $S_3 = 0 + 1 + 2 + 3 = S_2 + 3 = 6$
5. $S_4 = 0 + 1 + 2 + 3 + 4 = S_3 + 4 = 10$
6. $S_5 = 0 + 1 + 2 + ... + 5 = S_4 + 5 = 15$
7. $S_6 = 0 + 1 + 2 + ... + 6 = S_5 + 6 = 21$
8. $S_7 = 0 + 1 + 2 + ... + 7 = S_6 + 7 = 28$
9. $S_8 = 0 + 1 + 2 + ... + 8 = S_7 + 8 = 36$
10. $S_9 = 0 + 1 + 2 + ... + 9 = S_8 + 9 = 45$

11. $S_10 = 0 + 1 + 2 + ... + 8 = S_9 + 10 = 55$

The list of operations above show that each step S_n is simply a sum of $S_n - 1 and n$. For example, $S_9 = S_8 + 9$, where $n=9$.

And $S_n - 1 = S_n - 1 + (n - 1)$

This goes on until the smallest case is arrived at which is S_0. In the case above $S_0 = 0$ and the recursion stops there and the various results that were obtained in other steps are combined to form the emergent solution.

As such, S_n is a recurrence, because S_n invokes S_{n-1} which in turn invokes $S_n - 2$ and so on. This is repeated until the calls bottom out at S_0.

Hence, we can express $\sum_{i=1}^{n} i$ as $S_n = S_n - 1 + n$.

3.1.3 Another interesting observation

- S_n is the sum of the predecessor $S_n - 1$ and n.
- S_{n-1} is $S_{n-2} + (n - 1)$
- S_{n-2} is $S_{n-3} + (n - 2)$
- S_{n-3} is $S_{n-4} + (n - 3)$
- S_{n-4} is $S_{n-5} + (n - 4)$
- S_{n-5} is $S_{n-6} + (n - 5)$
- S_{n-6} is $S_{n-7} + (n - 6)$
- S_{n-7} is $S_{n-8} + (n - 7)$
- S_{n-8} is $S_{n-9} + (n - 8)$
- S_{n-9} is $S_{n-10} + (n - 9)$

The subscripts of S increase towards n.

Therefore, there will be a case where we have $S_n - n$ and this number truly is just S_0. And as seen above, at $S_0 = 0$ we stop and collect and combine th various results.

Hence, the recurrence S_n in evolving into $S_n - 1 + n$ eventually reduces to S_0 in one of the steps.

If we unwind all of S_n, then we shall obtain the following:

1. $S_n = S_{n-1} + n$

2. $S_n = (S_{n-2} + n - 1) + n$

3. $S_n = ((S_{n-3} + n - 2) + n - 1) + n$

4. $S_n = (((S_{n-4} + n - 3) + n - 2) + n - 1) + n$

5. $S_n = ((((S_{n-5} + n - 4) + n - 3) + n - 2) + n - 1) + n$

6. $S_n = (((((S_{n-6} + n - 5) + n - 4) + n - 3) + n - 2) + n - 1) + n$

7. $S_n = ((((((S_{n-7} + n - 6) + n - 5) + n - 4) + n - 3) + n - 2) + n - 1) + n$
   $\cdots \quad \cdots \quad \cdots$

8. $S_n = (((...(S_{n-n} + n - (n-1)) + 2 + ... + n - 3 + n - 2 + n - 1 + n$

9. $S_n = 1 + 2 + 3 + 4 + ... + n - 3 + n - 2 + n - 1 + n$

It shows that the recurrence S_n is also a way to pack several iterative operations.

The recurrence reads: $*S_n$ is the sum of S_{n-1} and n.

The first part of the reading, that is S_{n-1} is simply a recusive call. It's the application of the same process that we read above, but on a smaller set of input, n-1, in this case. And this way of calling and applying the process on smaller sets goes on until the end is reached, whence the recurrence ends and the recursive application is said to bottom out.

3.2 On the complex nature of recurrences

It was mentioned above that problem solving in general, and in computer science, involves analysis in order to study the costs involved and to reduce them. Recurrences are good and they are simple straigtforward solutions to some problems. However, they appear to be costly. To compute S_n, one has to first of all compute S_{n-1} and then add the results to n. But to also compute any S_x, one has to compute S_{x-1}. If every S_i has cost C_i then we are looking at a total cost of:

$\sum_{i=1}^{n} C_i$

Each application of S_n is a procedure we are executing. And remember that forumlae are tools that help us capture patterns between procedures. Interestingly, S_n is repretive and it works on a structured arrangement of numbers. We can obtain a formula that will be less costly.

It was shown earlier above that $S_n = \frac{n(n+1)}{2}$.

Such a formula is easier to implement than $\sum_{i=1}^{n} i$

The following computer programs testify to the costliness $\sum_{i=1}^{n} C_i$.

The program below computes S_n by using recurrences and recursion.

```
def sumOfFirstn(n):
    if n == 0:
        return 0
    return n + sumOfFirstn(n-1)

print "sumOfFirstn(10) is %d" % sumOfFirstn(10)
```

```
sumOfFirstn(10) is 55
```

The program below computes the same sum S_n by using a simpler formula: $S_n = \frac{n(n+1)}{2}$

Using this formula, the result is obtained by a simple application of addition, multiplication and division by 2. Such is the beauty of formulae.

```
def sumOfFirstn(n):
    result = (n * (n+1))
    return (result/2)
print "sumOfFirstn(10) is %d" % sumOfFirstn(10)
```

```
sumOfFirstn(10) is 55
```

It is common to solve a problem by, first of all, finding a simple recurrence and then later on working on the recurrence to obtain a simpler formula as shown the example above. The process of finding simple and less costly formulae is known as solving recurrences.

A formula like $S_n = \frac{n(n+1)}{2}$ is called the **CLOSED FORM** of a recurrence.

3.2.1 Another property of the complexity of recurrences

The idea of a mathematical function is very important in problem solving. A function is simply defined as a correspondence between 2 sets such that elements from one set called the domain and mapped or transformed to elements of a second set called the range of the function.

Matheamtical functions are also communicative devices. They can be used to specify the behaviors of operations. There are 4 main ways to present reports on functions:

1. Using formulae like: $S_n = \frac{n(n+1)}{2}$
2. Using graphs. That is, presentation pictorial reports
3. Using tables. Presentating the details as tables of values
4. Using prose or word descriptions

A useful application of functions is the business and medical fields. In the business field, if someon's investment is said to increase squarely by the years, then each year, the amount is the square of the previous amount.

We simply say, her investments X, grow by X^2 every year. Thesame can be used to report on the spread of diseases in the medical field. One can capture the cost of an operation by typing them to mathematical functions.

It is exciting to note that the solution to a recurrence actually reveals the cost of that recurrence as a mathematical formula.

Hence, $\sum_{i=1}^{n} i$ has cost that is proportional to: $S_n = \frac{n(n+1)}{2}$

This leads us to a simple concept of measuring the cost of operations called Big-Oh analysis. Follow this link to read more on that.

The gist of this technique is to extract the biggest term of a given forumala and attach the overall cost of the operation to that term. In the case of $S_n = \frac{n(n+1)}{2}$, the biggest term is n^2. This implies that the behavior of the sum of the first n numbers has a cost that is proportional to n^2. For any input size n, it will cost us computing time related to n^2 to get the result.

As a result, solving a recurrence for a closed form is also a technique for obtaining the cost of that operation.

3.3 Another solution to the recurrence $S_n = S_{n-1} + n$

Let's now use all the concepts that have been highlighted above to resolve the summation problem.

There are many ways to solve problems in general, and recurrences in particular. The most common and perhaps the most important way is to **GUESS** a solution and then try to prove that solution. In so doing, one arrives at the conclusion that the **GUESS** is right or wrong.

3.3.1 Another solution to $S_n = 1 + 2 + 3 + 4 + ... + n$

What is the formula for $S_n = 1 + 2 + ... + n$?

1. Using algebra This solution involves using algebra to guess a solution to the problem and then using mathematical induction to test our guess. We can then fine-tune the guess until we arrive at a correct result.

   (a) $S_1 = 1$
   (b) $S_2 = 3$
   (c) $S_3 = 6$
   (d) $S_4 = 10$
   (e) $S_5 = 15$

   Observe that for each n in S_n, the result is a couple of times bigger than the original n. Perhaps a table will clearly make this property observable.

n	S_n	Notes
1	1	The result is same as the input
2	3	The result is almost two times the input, 2
3	6	The result is 2 times the input, 3
4	10	The result is more than 2 times the input, 4.
5	5	The result is 3 times the input

   From the above table, we see that the result of S_n is a number times the original input.

   We can therefore guess that the result is the square of the input. That is, each n, $S_n \leq n^2$. Our task then is to fine the exact nature of S_n and how exactly close or far away it is from n^2.

   From the fundamental theorem of algebra, we know that each polynomial of degree n has at most n roots or solutions.

   Hence, our n^2 has 2 roots or solutions.

   We also have an operation S_n that generates values, so we can use our operation and our guess, n^2, to continue our investigation.

   One thing we can do is to form a set of equations were we try to map the values of n^2 to the values obtained from S_n.

   For n^2 and with the guarantee that it has at most 2 solutions, we are certain that we shall set up 2 simultaneous equations.

   Also, from algebra, any polynomial X of degree y has the form:

$A \times X^y + B \times X^{y-1} + C \times X^{y-2} + ... + Z \times X^0$

It follows, therefore, that n^2 which is our guess has the form:

$a \times n^2 + b \times n^1 + c \times n^0 = a \times n^2 + b \times n + c = a \cdot n^2 + b \cdot n + c$

Now let's link these n(s) to S_n:

- if n = 0, then $S_0 = 0 \Rightarrow S_0 = a \cdot (0)^2 + b \cdot (0) + c \Rightarrow 0 = c$ Hence, we can say, $S_n = a \cdot n^2 + b \cdot n$ since $c = 0$.
- If n = 1, then $S_1 = 1 \Rightarrow S_1 = a \cdot 1^2 + b \cdot 1 = a + b \Rightarrow 1 = a + b$
- If n = 2, then $S_2 = 3 \Rightarrow S_2 = a \cdot 2^2 + b \cdot 2 = 4a + 2b \Rightarrow 3 = 4a + 2b$

From the above, we obtain the following system of equations:

(a) $a + b = 1$

(b) $4a + 2b = 3$

(c) Now multiply equation on (1) above by 2.

(d) $\Rightarrow 2a + 2b = 2$

(e) Now subtract the equation on (4) above from the equation on (2)

(f) $\Rightarrow 2a + 0 = 1$

(g) $\Rightarrow a = \frac{1}{2}$

(h) Now substitute $a = \frac{1}{2}$ in (1) above

(i) $\Rightarrow b = \frac{1}{1} - \frac{1}{2} = \frac{1}{2}$

(j) Therefore, $a = \frac{1}{2}$, and $b = \frac{1}{2}$

It follows that $S_n = a \cdot n^2 + b \cdot n$ is actually given by the form:

$S_n = \frac{1}{2} \cdot n^2 + \frac{1}{2} \cdot n$

The quantity, $\frac{1}{2}$ is common to both terms, so we can factor it out to get:

$S_n = \frac{1}{2}(n^2 + n)$

And voila, we are back at the same solution that Gauss obtained. There is no need to carry out a proof, since we did that before.

3.4 Conclusion

As simple as counting sounds, it is a very important process and it is at the center of the analysis of algorithms and other interesting problems in other problem domains.

Numbers are infinite, hence we have at our disposal an inifinite amount of toys to play with. Moreoever, the tools we saw above provide us with building blocks for erecting our own systems and solutions to problems and also for investigating the nature of our systems.