Quantitative Reasoning in Computer Science

Samuel L. Marateck

Contents

1 Introduction

This is a proposal for a course in quantitative reasoning that combines some of the topics that are covered in the normal quantative reasoning course that was devised by Fred Greenleaf, plus some other topics, combined with the topics done in v22.0002, Introduction to Computers and Programming. The aim of the course is to bring students with deficient mathematical skills up to speed in their ability to use basic mathematical techniques in applications to computer science and other quantitative issues.

The topics that are covered in both the Greenleaf course and the proposed course are powers of ten, logarithms, permutations and combinations, and the the binomial probability distribution. Whereas the Greenleaf course and the proposed course cover this distribution, the proposed course also covers the binomial expansion, its relation to Pascal's triangle and this triangle's relation to the central limit theorem. Note that V22.0002 covers the **for** and **while** loops, **if** and **case** statements, methods and arrays.

Fred told me that the course described in his monumental text book, took seven years to develop, so if the proposed course is given the green light, consider the following description as just a first step. It consists of several case studies and we assume that it accompanies a Java text. The students would do the assignments in the proposed course in Java.

2 Determining Integer.MAX_VALUE

We'll start with some definitions. Numbers used in a Java program are examples of what are called *literals.* An integer is a number that does not contain a decimal point. Examples of integer literals are 423, -26 and 2009. Numerical literals are assigned to what are called the numerical *primitive* types. The integer family in Java contains four primitive types. The most commonly used are the **int** and **long** types. Their respective maximum values are 2147483647, given in a Java program by **Integer.MAX_VALUE**, and 9223372036854775807 given by **Long.MAX_VALUE**. The goal of this lesson is to determine how to both estimate and calculate the first of these values.

Let's start by reviewing multiplying and dividing powers of 10. In 10^3, 10 is called the base and 3 is called the exponent. When you multiply two numbers having the same base as in $10^3 * 10^5$, you add the exponents. So our product is 10^8. When you divide two numbers that have the same base as in $10^5/10^3$, you subtract the exponent of the denominator, here 3, from the exponent of the numerator, here 5. So the result is 10^2. We can generalize these results to any base. So $a^n * a^m$ is a^{n+m} and a^n/a^m is a^{n-m}. What happens when you evaluate $10^3/10^3$ following our rules, it's 10^0; but we know that the result is one. So any base raised to the zero power is 1, i.e., a^0 is 1.

The decimal system has base 10 and the digits range from 0 to 9. The number 632 can be expressed as 6 x 100 + 3 x 10 + 2 x 1. In power of 10 notation, it's $6x10^2 + 3x10^1 + 2x10^0$. The binary system, on the other hand, has base 2 and the digits can either be 0 or 1. The number 1111 in binary is $1x2^3 + 1x2^2 + 1x2^1 + 1x2^0$ or 8 + 4 + 2 + 1 or 15 in decimal. Although one can write computer programs without knowing the binary system, it is important to understand the binary system since computers use it to store data and perform calculations. The abbreviation for a binary digit is *bit.*

Calculating Integer.MAX_VALUE

In Java, an **int** occupies four bytes of memory and a byte consists of eight bits. Therefore an **int** requires 32 bits. The highest or left-most bit determines the sign and the remaining 31 bits store the binary number. The maximum **int** has thirty-one right-most bits set to 1. How do we determine its value? Certainly we can add all the powers of 2; but this can be arduous even if it's done on a computer. An easier way is to use a property of all number systems; but which we will use the binary system to demonstrate. We have seen that 1111 in binary is 15 in decimal. This is one less than the number obtained by setting the four 1's to 0,'s and preceding it with a 1, i.e., 10000. This is $1x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 0x2^0$ or simply 2^4 which is 16. So you raise 2 to the number of bits, here four and subtract 1. The largest **int** is therefore 2^{31} - 1. This result should not be strange, since in base 10, the highest three-digit number, for instance, is 10^3 - 1 or 1000 - 1, i.e., 999.

Before we calculate 2^{31} - 1, we review some mathematics. The number $10^3x10^3x10^3x10^3$ can be written as 10^{3x4} Now $2^{31} = 2x2^{30} = 2x(2^{10})^3$; but $2^{10} = 1024$. So $2^{31} = 2$ x 1024 x 1024 x 1024 which equals 2147483648, one more than **Integer.MAX_VALUE**. In a java expression, if you use an integer literal without appending a lowercase or uppercase L , it is considered an **int**. If you append an L, it's a **long**. So when the Java Virtual Machine executes **long m = 2*1024*1024*1024** it tries to evaluate the right-hand side as an **int** but fails since the result is greater than **Integer.MAX_VALUE**, so this statement produces an incorrect answer, -2147483648. If one of the

1024's is written as 1024L, for instance, **long m = 2*1024L*1024*1024**, the entire right-hand side is evaluated as a **long** and 2147483648 is assigned to **m**. When we subtract 1, we get the value of **Integer.MAX_VALUE**.

Although the preceding is best done on a computer, it could conceivably be done by hand. The next section demonstrates how to estimate **Integer.MAX_VALUE** by hand.

Estimating Integer.MAX_VALUE

Mathematicians and scientists like to do approximations to see if their method yields a result that is in the proper range or proper ball park*. To get an approximation to **Integer.MAX_VALUE** we approximate 1024 by 10^3 and obtain $2x10^3x10^3x10^3$ or $2x10^9$ as the maximum value of an **int**. The percent error in this result is the difference between the actual and approximate values, divided by the actual result; namely, (2147483647 - 2000000000)/2147483647 which equals 0.0687. So the error is 7 percent. In a later section we will learn how to estimate this from the original approximation of 1000 for 1024.

Using Logarithms to Calculate Integer.MAX_VALUE

Before we see another way to calculate **Integer.MAX_VALUE**, we will review more mathematics. Another term for an exponent is logarithm. So if $x = 10^y$, we say that the logarithm of x to the base 10 is y, or $log_{10}x = y$. Similarly $u = 2^v$ is equivalent to $log_2 u = v$. Exponents can be fractions. We know that $a^n * a^m = a^{n+m}$, so $10^{1/2} * 10^{1/2} = 10^{1/2+1/2} = 10^1$ or 10. But if $a^2 = 10$, then $a = \sqrt{10}$. This means that the exponent 1/2 indicates the square root. Now the square root of 10 is 0.331929865, so $log_{10}0.331929865 = 1/2$ or the log (exponent) of 0.331929865 to the base 10 is 1/2.

Again, since $a^n * a^m = a^{n+m}$ we can say that if $A_1 = a^n$ and $A_2 = a^m$, then $log(A_1 * A_2) = n + m = log(A_1) + log(A_2)$, so in general for any base

$$log(A * B) = logA + logB \tag{1}$$

Similarly

$$log(A/B) = logA - logB \tag{2}$$

Also since $log(A * A * A...)$ for n copies of A

$$n * log(A) = log(A^n) \tag{3}$$

Finally, since $a^0 = 1$

$$log(1) = 0 \tag{4}$$

Using this, let's calculate 2^{31}. We will be using numbers that have a decimal point. This type of number is called a *floating point* number. Examples are 12.8, -0.04 and 27.0. Although one should

*Hence these approximations are called *ball park approximations.*

never use floating point numbers to calculate an integer result, we show the following as an exercise in using logarithms.

Let's calculate 2^{31} in decimal so we understand its magnitude. We start by solving $10^x = 2^y$ for x by taking the log_{10} of both sides.

$$log_{10}10^x = log_{10}2^y \tag{5}$$

or by using (3),

$$x * log_{10}10 = y * log_{10}2 \tag{6}$$

.

But since $10^1 = 10$ so $log_{10}10 = 1$. Therefore

$$x = y * log_{10}2 \tag{7}$$

.

We obtained the value of $log_{10}2$ from a Java program. Its value is 0.3010299956639812. Since y equals 31, $x = 31 * 0.3010299956639812$ or 9.331929865583417. These two floating point numbers are examples of what is called the **double** primitive type in Java. We will see in a later section that generally floating point numbers stored in the computer's memory are only an approximation to the literals used in a program. So $2^{31} = 10^{9.331929865583417}$ is also an approximation; but in this case, an excellent one. It equals $10^{9+0.331929865583417}$ or $10^9 * 10^{0.331929865583417}$. A log table gives an approximate value of $10^{0.331929865583417}$; but a Java program, in this case, gives the exact value[†], 2.147483648E9. The $E9$ is the equivalent of 10^9 and indicates that we should move the decimal point in 2.147483648 nine places to the right, getting 2147483648. This is the value of **Integer.MAX_VALUE + 1**. Finally, we find that **Integer.MAX_VALUE** = 2147483648 - 1 or 2147483647.

Problem Using the equivalent of 2^n write a program that uses a **for** loop to calculate the maximum integer for a n-bit memory location. Can you think of a way of using this program to determine how many bits there are in an **int**?

[†]The three statement needed to do this are: double x = 31*Math.log10(2); double y = Math.pow(10,x); System.out.println(y);.

3 Error Propagation

When scientists perform an experiment, they measure quantities and give their values as a number plus or minus an error which indicates the imprecision of the measurement. If the final result is the product of many measurements, it is effected by the errors in each of the measurements. This is called *error propagation*. In order to demonstrate how error propagation is calculated, we will estimate how the error caused by using 1000 instead of 1024 effects the error in the determination of **Integer.MAX_VALUE**.

Since the result for **Integer.MAX_VALUE** consists of cubing our initial approximation, we will investigate how the error in a effects N in the equation

$$N = a^3 \tag{8}$$

.

Let's call the error in $N, \Delta N$ and the error in $a, \Delta a$. So N plus its error is $N + \Delta N$ and a plus its error is $a + \Delta a$. Substituting in (5) we get

$$N + \Delta N = (a + \Delta a)^3 \tag{9}$$

We know that $(a + b)^2 = a^2 + 2ab + b^2$, so

$$(a + \Delta a)^2 = a^2 + 2a\Delta a + (\Delta a)^2 \tag{10}$$

But since Δa is very small in comparison with a, we can assume that $(\Delta a)^2$ is negligible. So

$$(a + \Delta a)^2 = a^2 + 2a\Delta a \tag{11}$$

We now have to multiply (10) by $a + \Delta a$, getting

$$(a + \Delta a)^2 * (a + \Delta a) = (a^2 + 2a\Delta a)(a + \Delta a) \tag{12}$$

or

$$a^3 + 2a^2\Delta a + a^2\Delta a + 2a(\Delta a)^2 \tag{13}$$

We can again neglect $(\Delta a)^2$, and using (9) get

$$N + \Delta N = a^3 + 3a^2\Delta a \tag{14}$$

This equation has two parts and is in effect two separate equations, $N = a^3$ and $\Delta N = 3a^2\Delta a$. We will use the second of these and divide by N on the left-hand side and its equivalent a^3 on the right-hand side. So

$$\Delta N/N = 3a^2\Delta a/a^3 \tag{15}$$

or

$$\Delta N/N = 3\Delta a/a \tag{16}$$

since $a^2/a^3 = 1/a$. The quantities $\Delta N/N$ and $\Delta a/a$ are called respectively, the percent errors in N and a. So we see that the percent error in N is three times the percent error in a. The error in using 1000 instead of 1024 is 24/1024 or 0.0234. If we multiply this by 3 we get 0.0702 which is consistent with the error in our approximation of **Integer.MAX_ VALUE**. We have used three significant digits here to show that the error propagation method is not exact. We will prove in a subsequent section that if $N = a^k$, the percent error in N is k times the percent error in a.

Let's do another error estimate. This time we'll take $N = bc$. So

$$N + \Delta N = (b + \Delta b)(c + \Delta c) \tag{17}$$

$$N + \Delta N = bc + c\Delta b + b\Delta c + \Delta b * \Delta c \tag{18}$$

Again we can neglect $\Delta b * \Delta c$, so we get

$$N + \Delta N = bc + c\Delta b + b\Delta c \tag{19}$$

or since $N = bc$, subtract N from both sides of the equation

$$\Delta N = c\Delta b + b\Delta c \tag{20}$$

We assume that the error in a does not effect the error in b. To explain the step that follows we give this analogy. Assume that the sun is directly overhead and shines on a vertical Δb. It doesn't cast a shadow on a horizontal Δc. They are at right angles to each other and form the two sides of a right triangle. The resultant is the hypotenuse. So by the Pythagorean theorem

$$\Delta N = \sqrt{((c\Delta b)^2 + (b\Delta c)^2)} \tag{21}$$

This is called adding Δb and Δc in quadrature. Dividing both sides by N we get

$$\Delta N/N = \sqrt{((c\Delta b)^2 + (b\Delta c)^2)/(bc)} \tag{22}$$

If we take bc in side the square root sign, we have to square bc in the denominator, so we get

$$\Delta N/N = \sqrt{((c\Delta b)^2/(bc)^2 + (b\Delta c)^2/(bc)^2)} \tag{23}$$

or

$$\Delta N/N = \sqrt{((\Delta b/b)^2/ + (\Delta c/c)^2)} \tag{24}$$

So the percent error in N is the percent errors of b and c added in quadrature. Why did we neglect the $(\Delta)^2$ terms in the previous calculations and here we include these terms? The reason is that here we are comparing $(\Delta)^2$ with other $(\Delta)^2$ terms so they are of the same order.

4 Proof By Induction

Mathematicians use the symbol $\sum$ to indicate a summation. For instance, the sum $1 + 2 + 3 + 4$ is indicated by $\sum_{n=1}^{n=4} n$. The value of n on the bottom of the $\sum$ indicates the initial value of n and the value of n on the top indicates its final value. We have seen that $2^0 + 2^1 + 2^2 + 2^3 = 2^4 - 1$. We rewrite this using the $\sum$ notation as $\sum_{n=0}^{n=3} 2^n = 2^{3+1} - 1$. If we let the upper limit of n be k, we can generalize this as

$$\sum_{n=0}^{n=k} 2^n = 2^{k+1} - 1 \tag{25}$$

One of the proofs used by mathematicians is called the *proof by induction.* Your hypothesized function of n must satisfy two criteria:

- It must be true for an initial value, let's say $n = 0$.
- If it's true for n, it must be true for $n + 1$.

Let's use induction to prove that equation (25) is correct. If k is zero, the sum is $2^1 - 1 = 2 - 1$ or 1. Since 2^0 is 1, the sum of only the first term is 1. So the first criterion is satisfied. To test the second criterion, we add the next term (2^{k+1}) to the sum $\sum_{n=0}^{n=k} 2^n$ and see if the augmented sum equals $\sum_{n=0}^{n=k+1} 2^n$, i.e., $2^{k+2} - 1$. So we get $2^{k+1} - 1 + 2^{k+1}$. Now $2^{k+1} + 2^{k+1} = 2 * 2^{k+1}$ which equals $2 * 2^k * 2^1 = 2^2 * 2^k$ or 2^{k+2}. So the augmented sum is $2^{k+2} - 1$. Thus the second criterion is satisfied.

5 Floating Point Numbers

Remember that numbers that have a decimal point are called *floating point numbers.* Let's determine the maximum number of digits a floating point number less than 1.0 can have. This will give us an idea of the number of meaningful digits a floating point number can have.

We consider the largest binary number less than 1.0 the computer can accommodate – it is 0.1111111..., i.e., all the bits are 1, and determine how this binary number translates to decimal one. We would expect this binary number to translate into the largest decimal number less than 1.0, i.e., 0.9999999... We will see that the number of 9's we get depends upon the number of bits the computer can accommodate in a memory location.

To determine how to calculate the decimal equivalent of, for instance, 1.11..., let's look at an example in the decimal system. The number 4.32 is the same as $4x1 + 3x0.1 + 2x0.01$ or $4x10^0 + 3x10^{-1} + 2x10^{-2}$. By analogy, 1.11 is $1x2^0 + 1x2^{-1} + 1x2^{-2}$. So to calculate the decimal equivalent of "0." followed by max 1's, we evaluate

$$\sum_{n=1}^{n=max} 1/2^n \tag{26}$$

in a hypothesized computer. Note that $\sum_{n=1}^{n=4} 1/2^n = 1/2^1 + 1/2^2 + 1/2^3 + 1/2^4$. The right-hand side represents the decimal equivalent of the binary number .1111. If we limit ourselves to a maximum of 16 bits, we get

m $\sum_{n=1}^{n=m} 1/2^n$

```
1   0.5
2   0.75
3   0.875
4   0.9375
5   0.96875
6   0.984375
7   0.9921875
8   0.99609375
9   0.9980469
10   0.99902344
11   0.9995117
12   0.99975586
13   0.9998779
14   0.99993896
15   0.9999695
16   0.99998474
```

Table 1.

The first 9 appears if the hypothesized computer can accommodate four bits. If it can accommodate 16 bits, the decimal number has four 9's in 0.99998474. The rest of the digits, 8474, are not significant. We should see now that there is a difference in the accuracy of how floating point numbers are stored and how integers are stored in the computers memory. Generally, a floating point number stored in the computer's memory is only an approximation to the floating point literal used in a program. Integers, on the other hand, are stored exactly as they appear in an integer literal.

Problem
Write a program to produce the output shown in Table 1.

6 Permutations and Combinations

We'll start by considering three marbles in an urn, one red (R), one green (G) and one blue (B). We'll select one marble at a time without replacing them. If we consider the order in which the marbles are selected, how many different orderings can we get. An example of an ordering would be RBG, another ordering would be BRG. For the first marble we have three choices. For each of thes three possibilities, we have two choices for the second marble. So if we choose R first, we could choose G or B next. For the last marble we have only one choice. So if we choose R first, B second, we could only choose G last. The number of orderings we can get is therefor $3 \cdot 2 \cdot 1$ or 6. As you probably know, the product can be written as 3!, pronounced three factorial. Here they are: RBG, RGB; BGR, BRG; GRB, GBR, where the semicolons separate orderings where the first colours are the same. An ordering of the colours is also called a permutation. So there are 3! permutations of these three objects. What result would we get if we choose three marbles from an urn containing four differently coloured marbles. Now we would have four choices for the first marble; and for each of these choices, three choices for the second marble; and for each of these three choices, two choices for the last marble. So the number of permutations would be $4 \cdot 3 \cdot 2$. If we choose three marbles from an urn containing five marbles, the number of permutations would be $5 \cdot 4 \cdot 3$ or sixty. Note that 0! is defined as 1.

Now let R represent a student named Roy; G, Grace; B, Bob; and V, Vivian. How would we calculate the number of study groups we could get if we choose study groups consisting of two students? A given group is independent of the ordering of the letters. So one group we could get is RG but this is the same as GR. So if we calculate the number of permutations, namely, $4 \cdot 3$, we would get a number too large because it includes the permutations of the letters for each committee. We compensate for this by dividing by the number of ways these two letters can be arranged (or permutated). So the result is $4 \cdot 3/2!$ or 6 and they are: RG, RB, RV, GB, GV and BV. This result is referred to as number of combinations of four objects taken two at a time, and is written as the combination symbol $\binom{4}{2}$, also referred to as "4 choose 2" . The branch of mathematics dedicated to the study of the properties of combination symbols and their use is called *combinatorics*. Let's do a few more examples.

How many groups of four could you get from a class of six students. The answer is $\binom{6}{4}$, i.e., $6 \cdot 5 \cdot 4 \cdot 3/4!$ or 15 groups. We can write a combination symbol in terms of factorials only, by writing the numerator, here $6 \cdot 5 \cdot 4 \cdot 3$, as a factorial. To do this we have to multiply by $2 \cdot 1$ getting $6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$; but then we must compensate for multiplying by $2 \cdot 1$ by dividing by $2 \cdot 1$. The result is $6!/(4! \cdot 2!)$. Note that the numbers in the denominator add up to the number in the numerator. So in general

$$\binom{m}{n} = m!/(n! \cdot (m-n)!) \tag{27}$$

.

What is $\binom{6}{2}$? It's $6!/(2! \cdot 4!)$; but this is the same as $\binom{6}{4}$. In general

$$\binom{m}{n} = \binom{m}{m-n} \tag{28}$$

.

How do we determine $\binom{m}{0}$. Using (27) we see that it equals $m!/(0! * m!) = m!/m! = 1$. In mathematics and science, it's nice to check an answer by using a method other than the original one used. Using (28), we get $\binom{m}{0} = \binom{m}{m} = 1$.

A Combinatorics Problem

How many different license plates can be made if each plate must contain three letters and three digits in any order, for example:

C	A	3	9	A	4

Let's begin by investigating the problem of a license plate that has two characters, one a letter and one a digit. So we can get a plate that has the letter first, for example:

C	4

Each of the 26 letters, can be matched with 10 different digits, so we can get 260 of these license plates. However, the digit could come first, followed by by a letter, for example:

4	C

and we can get 260 of these plates. The total number of plates we can get is $2 \cdot 26 \cdot 10$ or 520. Let's consider the slightly more difficult to solve situation of plates that have three characters, two of which have to be letters. We'll let A and B represent the letters and 7 the digit. The possible arrangements we can get are:

A	B	7

A	7	B

7	A	B

Since there are three possible configurations, the number of different license plates we can get is $3 \cdot 26 \cdot 26 \cdot 10$. How do we get the 3? There are three possible choices for the first position:

A		

	A	

		A

and for each of these, two possible choices for the seconds position making a total of six configurations. But since the two objects we are fitting in these positions are letters and we are including all 26 letters, we will be counting each configuration 2! ways or twice. So we divide by two. More generally, we are choosing two objects from three positions or 3 choose 2, $\binom{3}{2} = 3 \cdot 2/2 \cdot 1$ or 3. If you do the analysis for a four-character license plate where two of the characters have to be letters, you will see that the number of arrangements of the letters is $\binom{4}{2}$. Since there are two positions left, the number of ways we can arrange the digits is $\binom{2}{2}$ or 1. Therefore we only have to compute

the number of ways we can arrange the letters. (If we started by computing the number of ways we can arrange the digits, the number of ways we can arrange the letters would be $\binom{2}{2}$ or 1.) So for our original problem in which the license plate has six characters, three of which are letters and three are digits, the number of arrangements of all the characters is $\binom{6}{3}$ or 20, so the number of different license plates obtainable is $20 \cdot 26^3 \cdot 10^3$

What is the answer if we have a license plate that has room for seven characters, three of which are letters. Do we calculate $\binom{7}{3}$ for the letters or $\binom{7}{4}$ for the digits. It doesn't matter since by (28) $\binom{7}{3} = \binom{7}{4}$.

7 The Binomial Theorem

The binomial theorem provides a simple way of calculating $(a+b)^n$. It is given by

$$(a+b)^k = \sum_{n=0}^{n=k} \binom{k}{n} a^n b^{k-n} \tag{29}$$

To understand this, let's expand $(a+b)^3$ keeping the order of a and b. So $(a+b)^2 = a^2+ab+ba+b^2$ and $(a+b)^3 = (a+b)\cdot(a+b)^2 = aaa+aba+aab+abb+baa+bba+bab+bbb$. The terms that have two a's are aba, aab and baa; since we have three choices for the first position to place an a and for each of these choices, two choices to place the second a, and since the $a's$ are indistinguishable the number of these terms is just $\binom{3}{2}$. A similar analysis applies to the other groups of terms. So in general the coefficient of $a^n b^{k-n}$ is $\binom{k}{n}$.
Let's use (29) to calculate $(a+b)^2$

$$(a+b)^2 = \binom{2}{0} a^0 b^2 + \binom{2}{1} a^1 b^1 + \binom{2}{2} a^2 b^0 \tag{30}$$

or $b^2 + 2ab + a^2$. Similarly,

$$(a+b)^3 = \binom{3}{0} a^0 b^3 + \binom{3}{1} a^1 b^2 + \binom{3}{2} a^2 b^1 + \binom{3}{3} a^3 b^0 \tag{31}$$

or $b^3 + 3ab^2 + 3a^2b + a^3$. Let's now use the binomial expansion to show that if $N = a^k$, then

$$\Delta N/N = k\Delta a/a \tag{32}$$

Now

$$(a+\Delta a)^k = \binom{k}{0} a^k (\Delta a)^0 + \binom{k}{1} a^{k-1} \Delta a + \binom{k}{2} a^{k-2} (\Delta a)^2 \tag{33}$$

plus terms in high powers of Δa. We can neglect all terms but the first two since $(\Delta a)^2$ and higher Δa terms are negligible. Thus

$$(a+\Delta a)^k = \binom{k}{0} a^k (\Delta a)^0 + \binom{k}{1} a^{k-1} \Delta a \tag{34}$$

or

$$(a+\Delta a)^k = a^k + k a^{k-1} \Delta a \tag{35}$$

So

$$N + \Delta N = a^k + k a^{k-1} \Delta a \tag{36}$$

or

$$\Delta N = k a^{k-1} \Delta a \tag{37}$$

and finally,

$$\Delta N/N = (k a^{k-1} \Delta a)/a^k \ = k \Delta a/a \tag{38}$$

A problem

A time-honored story tells of an emperor who wanted to reward a servant for meritorious service. The servant requested that he receive the following reward. One grain of rice would be placed on on the first square of a chessboard, two on the second square, four on the third one, eight on the fourh one and so on so that each square contained twice as much rice as the previous one did. Of course, we have to suspend disbelief to assume that the grain could fit on a given square. How much grain would the servant receive. It's the same problem as determining the maximum unsigned integer that could fit in an n-bit memory location. The answer for the rice is $2^{64} - 1$. What is that in human terms? Again we approximate 2^{10} or 1024 by 1000. So $2^{64} = 2^4 \cdot 2^{60}$ or $16 \cdot 10^{3^6} = 16 \cdot 10^{18}$. The actual answer is 18,446,744,073,709,551,615. Our approximation of 1024 by 1000 produces an error of 0.0234. Since we raise 10^3 six times, the error is approximately six times 0.0234 or 14 percent. Note that each square contains more than the sum of the preceding squares.

Pascal's Triangle

It was apparently known in medieval times that the coefficients in the binomial expansion followed the following pattern, now known as Pascal's triangle

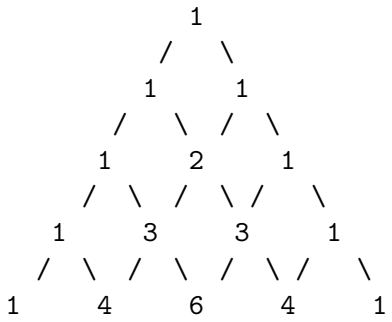
```
row
 0                              1
 1                          1       1
 2                      1       2       1
 3                  1       3       3       1
 4              1       4       6       4       1
 5          1       5      10      10       5       1
 6      1       6      15      20      15       6       1
```

The triangle is formed as follows: Each number in a given row is the sum of the numbers immediately to its left and right in the preceding row. For instance the 3 in row three is the sum of the 1 diagonally above it and to its left, and the 2 diagonally above it and to its right. If there is no number in the preceding row diagonally above a given number, the number in the preceding row is considered to be zero. The triangle simplifies the expansion of $(a+b)^n$ for any value of n as we now show.

The 1 in row 0 is the coefficient of $(a+b)^0$ which of course is one. The two 1's in the row 1 are respectively the coefficients of a and b in $(a+b)^1$. The 1 2 1 in row 2 are the coefficients of a^2, ab and b^2 respectively in the expansion of $(a+b)^2$. The 1 3 3 1 indicate that the expansion of $(a+b)^3$ is $a^3 + 3a^2b + 3ab^2 + b^3$. Observe that the powers of a and b must add up to the row number and the power of a decreases from the row number to zero, whereas the power of b increases from 0 to the row number. Thus $(a+b)^4 = a^4b^0 + 4a^3b^1 + 6a^2b^2 + 4a^1b^3 + a^0b^4$. If we set a^0 and b^0 both to 1, we get $a^4 + 4a^3b^1 + 6a^2b^2 + 4a^1b^3 + b^4$.

We consider the Pascal triangle to be a graph. In mathematics a graph consist of points (here represented by the numbers) called *vertices* and the lines connecting these points, called *edges*. Here the edges are represented by diagonal lines.

When a mathematicians and scientists study a situation, they look for a pattern. Looking at the Pascal triangle shown below, we see that the number of paths from the apex of the triangle, the solitary 1, to any vertex is given by the number at the vertex. So, for instance, there are four possible paths from the apex to each of the vertices labeled 4. We will be using this correspondence when we discuss binomial probability



8 Simple Probability

We begin with a few definitions

- An *experiment* is a process that enables us to see a result. Examples are: flipping a coin many times, tossing a pair of dice, and chosing a marble from an urn. In probability we try to predict experimental outcomes.

- An *event* is an outcome of an experiment. Example: flipping a coin and getting a head. Getting a red marble from an urn.

- A *simple event* is an event that cannot be broken down into other events. Example: tossing a die and getting a three.

- A *compound event* is an event that can be broken down into simple events. Example: tossing two coins or tossing a coin twice.

- Two events are *mutually exclusively* if one event does not effect the other one. Example: Flipping one coin does not effect the result of flipping another coin.

- Probability of A, written P(A), is the number of ways A can occur divided by the number of ways simple events can occur for the experiment. If you toss a coin, the result for a simple event could be a head, H, or a tail T. So if you toss two coins the compound events are HH, HT, TH, and TT. Let A represent two heads, then P(A) = 1/4 since there are four possible outcomes, and HH can occur only once. If A represents a head or a tail independent of order, the two possibilities are HT and TH. So the probability, P(A), of getting a head or tail in any order is P(A) = 2/4. If A represents getting two tails, P(A) = 1/4.

- P(A)= 0 means event A cannot happen. So if an urn contains red marbles the probability of getting a geen marble is 0. P(A) = 1 means an event must happen. So the probability of getting a red marble is 1.

- The sum of the probabilities in an experiment is 1. So if two coins are flipped P(HH) + P(HT) + P(TT) = 1/4 + 2/4 + 1/2 =1.
- The word "or" used in an experiment involving mutually exclusive events means that a "+" sign must be used in the calculation. So in our coin flipping experiment, P(HH or TT) = P(HH) + P(TT), i.e., 1/4 + 1/4 or 1/2.
- The word "and " used in an experiment involving mutually exclusive events means that multiplicaion must be used in the calculation. Let A represent getting a head first and a tail second in our experiment, then P(A) = P(H)*P(T) = 1/2*1/2 or 1/4. If A represents getting a head and tail in any order, then P(A) equals P(H)*P(T) or P(T)*P(H). So P(A) = 1/2*1/2 + 1/2*1/2 = 1/4 + 1/4 = 1/2.

Let's consider an experiment of rolling a die. A die has six sides, marked with from one to six dots. Let's call a success S, getting a two. Then a failure, F, is getting anything else. The probability of getting an S is 1/6 since there are six simple events corresponding to the six sides of the die and only one of them constitutes a S. The probability of getting an F is 5/6 since there are five ways of getting a failure. So P(S) + P(F) is 1/6 + 5/6 or one. Let p = P(S) and q = P(F). If you roll the dice twice, you could get SS or SF or FS or FF. The total probability is composed of $1/6 \cdot 1/6 + 1/6 \cdot 5/6 + 5/6 \cdot 1/6 + 5/6 \cdot 5/6 = 1/36 + 5/36 + 5/36 + 25/36$ This adds to 1. This is expected since the probability of all the events happening is 1.0.

What happens if the die is rolled three times? The configurations corresponding to one success (and of course two failures) are SFF, FSF and FFS. The probability of getting any of them is pq^2 or $1/6 \cdot 5/6 \cdot 5/6$. We see that there are three possible ways of getting one S and two F's. Let see how we can determine this without enumerating the F's and S's. It's more enlightening if we consider the F's. There are three positions we can place the first F and for each of these, two positions we can place the second F, the number of ways we can position the F's is $3 \cdot 2$. But the two F's are indistinguishable, so there are 2! ways of rearranging them. Thus the number of ways of getting two F's and one S is $\binom{3}{2}$. If we had considered the number of ways of getting one S instead, the answer is $\binom{3}{1}$. But these two combination symbols are the same. The probability of getting one S and two F's is $\binom{3}{2}p \cdot q^2$ or $3 \cdot 1/6 \cdot 5/6 \cdot 5/6$. We can now see that the probability of getting k S's and j F's is $\binom{(k+j)}{k}p^k \cdot q^j$ since $k + j$ add to the total number of events. If we let $n = k + j$, then the probability of getting k S's and $n - k$ F's, called P(k), is $\binom{n}{k}p^k \cdot q^{n-k}$ Again, this is the probability of getting k successes in n throws which is the same as getting $n - k$ failures in n throws.

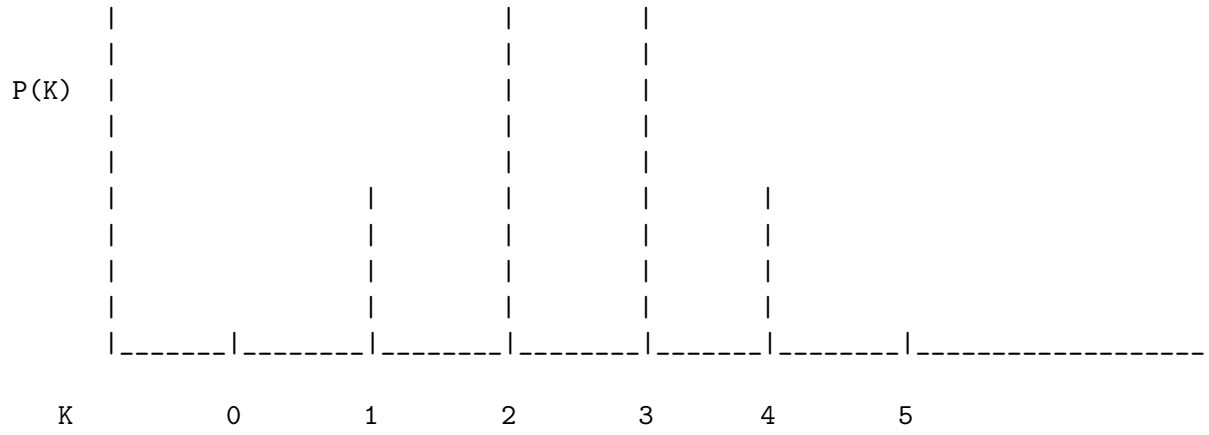
Let's now use $P(k) = \binom{n}{k}p^k \cdot q^{n-k}$ to calculate the probability of getting any number of heads if we toss a coin five times. Now both p and q are both 1/2.

- $P(0) = \binom{5}{0}(1/2)^0 \cdot (1/2)^5 = 1/32$

- $P(1) = \binom{5}{1}(1/2)^1 \cdot (1/2)^4 = 5/32$
- $P(2) = \binom{5}{2}(1/2)^2 \cdot (1/2)^3 = 10/32$
- $P(3) = \binom{5}{3}(1/2)^3 \cdot (1/2)^2 = 10/32$
- $P(4) = \binom{5}{4}(1/2)^3 \cdot (1/2)^2 = 5/32$
- $P(5) = \binom{5}{5}(1/2)^2 \cdot (1/2)^3 = 1/32$

Since the probability of success and failure are both 1/2, these probabilities can be written $\binom{5}{0}(1/2)^5$, $\binom{5}{1}(1/2)^5$, $\binom{5}{2}(1$ and $\binom{5}{5}(1/2)^5$, respectively.

If we plot P(k) versus k where k is the number of heads and P(k) the probability of getting k heads, we get the following probability distribution:



As n, the number of flips of the coin increases, the plot of P(k) versus k becomes more and more like a bell-shaped curve.

9 Galton's Board as an example of the Central Limit Theorem

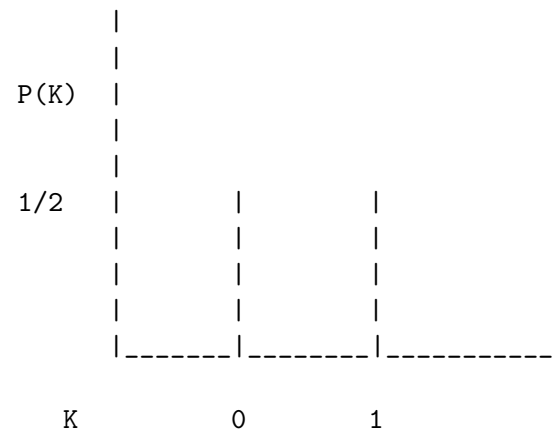
Francis Galton (1822 - 1911) a cousin of Charles Darwin, created a device consisting of a board that has pegs in the same positions as the numbers in Pascal's triangle. The board is on an incline, and marbles are rolled down from the apex of the triangle. See the animation at:

http://www.math.psu.edu/dlittle/java/probability/plinko/index.html

Each time a marble hits a peg, there is 50/50 chance of the marble rolling to the left or right, so the probability is 1/2 of the marble going either way. At the bottom of the board are bins to collect the marbles. Since the number of paths from the apex to a given peg is the corresponding number in the Pascal triangle, and if there are n rows of pegs, the probability $P(k)$ of marbles falling in the kth bin is $P(k) = \binom{n}{k}(1/2)^k$. As the value of n increases, the distribution of marbles in the bins approaches a bell-shaped curve.

Connection with the Central Limit Theorem The central limit theorem states that if a random sample of n values are taken from almost any distribution, a plot of the sum of these values will approach a bell-shaped curve as n approaches infinity.

Let's take flipping a coin as an experiment. If we assign 0 to a head and 1 to a tail, the probability distribution is:



Let's simulate a coin toss with a number that is randomly selected to be 0 or 1. The experiment consists of repeating the toss n times. We sum the 0's and 1's, and for example for n equals 10, we will get a sum that ranges from 0 (for ten heads) to 10 (for ten tails). If we repeat this experiment many times and plot the frequency of a given sum versus the sums, we will get a distribution that approached a bell-shaped curve as n approaches infinity.

If we form a string of the 0's and 1's, we can get for example, "0101010101". If we use these strings to guide a marble on the Galton board with ten rows, from the apex to the bins at the bottom, with a zero indicating bouncing to the right and a one indicating bouncing to the left, we get the same type of distribution as when the marbles simply roll down the board.

10 A Number Theory Example

You can observe a feature of all three digit numbers in which the digits are different by doing the following: Arrange the digits in the number, for instance 623, so that they are in descending order, e.g., 632, and then arrange them so that they are in ascending order, e.g., 236. Subtract the latter from the former, here getting the result 396. If this process is continually repeated using the result of the subtraction, the result will eventually be 495. If this procedure is performed on 495, the result will be 495.

11 A Number Theory Project

Part A
Write a program in which the main method calls a method with the heading
public static int subtract(int num)
that takes a three digit integer **num**, for instance 123, and does the following

- Decomposes it into three digits, digit1, digit2 and digit3. One of the ways you can do this is by using **String s = "" + num**, and then use **s.charAt(0), s,charAt(1), s.charAt(2)** to get the individual digits.
- Using **Math.max()** and **Math.min()** determine the digits with the maximum and minimum value and assigns them to **maxDigit** and **minDigit** respectively.
- Determines the middle digit and assigns it to **midDigit**.
- Constructs an integer, **ascend** in which these digits appear in ascending order. For instance, if the original value of **num** was 673, the constructed integer would be 367.
- Repeats the last step but this time produces the digits in descending order. So in our example, it would produce 763. It assigns this to **descend**.
- Prints these two values.
- Subtracts these two numbers **descend - ascend** and returns this value in the method.

Part B

Write another program that uses a **while** loop and calls method **subtract** defined in part A from the main method until the method returns a value that doesn't change, i.e., 495. Try the following approach. Before the loop, assign 123 (or any other 3-digits number different from the one we describe next) to **previous** and 149 (or another 3-digit number in which all the digits are different) to **present**. Your program should print the value of **present**. Construct the **while** loop as follows:

```
while( previous != present)
{
     previous = present;
     ........ = subtract(...);
}
```

12 An Example of Number Theory Analysis

Can we predict the result of the subtraction in the previous problem. We'll let a be the largest digit, b, be the middle digit and c be the smallest digit. The number with the digits in descending order is $100a + 10b + c$. The number with the digits in ascending order is $100c + 10b + a$. The subtraction yields:

```
  100a + 10b + c
- 100c + 10b + a
  --------------
  100a - a + c - 100c
```

The result is $99a - 99c$ or $99(a - c)$. This means that the result of the subtraction will always be a multiple of 99. Let's check this. The two results we got in the subtractions in section 11 are 396 and 495. The first result, 396, is 4x99 and the second, 495, is 5x99. The reason we specify that the three digits in the integer should be different, is that the subtraction method does not work for 221 since 221 - 122 is 99. The fact that the three digits must be different means that the minimum value of $a - c$ is 2. The maximum it can be is is 9 - 0. Therefor the result of the subtraction is the set 297, 396, 495, 594, 693, 792, 891. The pair 297 and 792 are permutations of each other and will produce the same result in the subtraction process. The same applies to the pair 396 and 693, and the pair 495 and 594.

Can we determine what the sum of the digits in the result will be? Let's remember how subtraction is done by subtracting 356 from 653.

```
  653
- 356
  --------------
  297
```

As usual we start from the right and work our way to the left.. Since 6 is less than 3, we add 10 to 3 and subtract 6 from 13, getting 7. We then add 1 to 5 and subtract 6 from 10 + 5, getting 9. Finally, we add 1 to 3 and subtract 4 from 6, getting 2. Let's now generalize this. By definition c is less than a. So we have to add 10 to c in order to perform the subtraction of the left-most digit. We the add 1 to b and since b + 1 is greater than b and we subtract that from 10 + b. Finally, we subtract c - 1 from a. The result is

$$a - (c + 1) + 10 + b - (b + 1) + 10 + c - a \tag{39}$$

or

$$a - c - 1 + 10 + b - b - 1 + 10 + c - a \tag{40}$$

All the variables cancel each other and we are left with -1 + 10 - 1 + 10 or 18. So the sum of the digits in the result after the subtraction is always 18. Check that this is true for the set of allowable results.

13 Prime Numbers

Before we start, we introduce the modulo operator which in Java is represented by "%". Although in Java the "%" can operate on floating point numbers and integers, we will confine our remarks to the integers. For two integers a and b, $a\%b$ (pronounced "a mod b") is the remainder when a is divided by b. So 5%3 is 2, 6%3 is 0 and 3%4 is 3. When $a\%b$ is 0, we say that "b divides a" or that "a is divisible by b". A prime number is an integer that is divisible only by 1 and itself. The integer 2 is the first prime number. The digits that are prime are 2, 3, 5, and 7. the digit 9 is not prime since it is divisible by 3. Numbers that are not prime are called composite, so nine is a composite number

.

The Fundamental Theorem of Arithmetic
Every composite integer can be expressed as a product of primes[‡]. (This means that every integer is divisible by a prime.)

For instance, $9 = 3x3$ or 3^2, $27 = 3^3$. Similarly, $8 = 2^3$, and $12 = 2^2 * 3$.

For any integer a, if $a\%2$ is 0, then a is an even number. If an odd number is multiplied by an even number, the result is an even number, e.g., $3x4$ is 12.

Euclid's Proof that the Number of Primes is Infinite
Euclid proved that the number of primes is infinite. It is a proof by contradiction, i.e., assumes that a premise is true and then proves that it is false,

- Assume that that there exists a greatest prime p_{max}.
- Take the product P of all the primes where p_i represents any prime including p_{max}. The value of $P\%p_i$ is zero since any prime divides the product of all the primes.
- Add one to this product and call it n. Now $n\%p_i = 1$ because if you divide n by any of the primes that formed the product P, the remainder is one.
- The integer n is either prime or composite. If it's composite, then by the fundamental theorem of arithmetic, there must be a prime that divides n. None of the p_i do, nor does their product P. So there must be a prime greater than p_{max} that divides n. If n is prime, the same argument applies. Therefor our original assumption is false.

14 A Prime Number Project

The sieve of Eratosthenes is a method of generating a list of consecutive prime numbers without having to start a loop beginning with 2 each time.

[‡]For a given integer, this product is unique. This means that if for a given integer, there is more than one product, these products differ only by a permutation.

Here is how it works:

1. Dimension **x**, an array of **int**, to N and set N to 1000. As you know, all of the elements of an array of **int** are automatically set to zero.

2. In a **for** loop, starting with the loop index set to 2, test if the array element with that index is set to zero. If it is, in a **while** loop nested in this **for** loop, set all the elements of the array to 1 that have subscripts that are multiples of the starting index. So when the loop index is 2, you would set **x[4]**, **x[6]**, **x[8]**, etc., to 1.

3. Each successive value of the **for** loop should repeat this process until the **for** loop upper limit $\sqrt{N}$ is reached.

4. When the loop is complete, all the array elements with value 0 will have a loop index that is a prime number.

5. C. F. Gauss hypothesized that $\Pi(N)$ the number of primes less than or equal to N is defined as $\Pi(N) = N/log_e(N)$ as N approaches infinity. This was called the *prime number hypothesis.*[§] In another **for** loop print the prime numbers, a counter indicating its ordinal number (1, 2, 3, etc.) and the value of $\Pi(N)$. The variable **N** is the loop index which is the same as the prime number. Here, the maximum value of the loop index should be 1000.

   Your output should look something like the following. Don't worry about labeling the columns exactly as is shown here.

```
Prime number  total num of primes   predicted  # of primes
2             1                    3
3             2                    3
5             3                    3
7             4                    4
11            5                    5
```

   Note that the $log_e(N)$ and *sqrt* functions are part of the **Math** class, i.e., **Math.log()** and **Math.sqrt()**; to round the predicted number of primes, use **Math.round()** which produces a **long**.

[§]It was proved independently by Hadamard and de la Vallee Poussin in 1896 and is now called *the prime number theorem.*

15 The Riemann Zeta Function and Euler's Product (Optional)

There is a formula in mathematics involving something called the Euler product that may help you understand the sieve of Eratosthenes. It is also useful in understanding the fundamentals of data encryption, essential for securely transmitting data over the internet. The Euler product is equated to the what is called the Riemann ζ function (pronounced zeta) defined as

$$\zeta(s) = \sum_{n=1}^{n=\infty} 1/n^s \tag{41}$$

Since the upper limit is ∞ (infinity), this sum is called an infinite sum. for $s = 2$, the first four terms are

$$\zeta(2) = 1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 \tag{42}$$

Before we continue, we show why $2^3 4^3 = 8^3$. Consider $a^n b^n$. Take the log and get $log(a^n b^n)$. This equals $log(a^n) + log(b^n)$ by (1). Using (3) this equals $nlog(a) + nlog(b)$. Factoring out n we get $n(log(a) + log(b))$. Using (1) again, we get $n(log(ab))$ which equals $(log((ab)^n))$. Thus $log(a^n b^n) = log((ab)^n))$. If two logs are equal, their arguments, here $a^n b^n$ and $(ab)^n$ are equal. So

$$a^n b^n = (ab)^n \tag{43}$$

If two values x and y are equal and they aren't zero, then their inverses $1/x$ and $1/y$ are equal.

The symbol in mathematics for a product is Π. For instance $\Pi_{n=1}^{n=4}(1+n)$ is

$$\Pi_{n=1}^{n=4}(1+n) = (1+1)(1+2)(1+3)(1+4) \tag{44}$$

The Euler Product is defined in terms of $\zeta(s)$ as

$$\zeta(s) = \Pi_{allprimes}(1/(1-1/p^s) \tag{45}$$

Divide both sides by $\Pi_{allprimes} 1/(1-1/p^s)$ So

$$\zeta(s)/\Pi_{allprimes} 1/(1-1/p^s) = 1 \tag{46}$$

To calculate this we must know how to evaluate a term like $1/(1/A)$. Multiply both the numerator and denominator by A getting $A/(A/A)$ or A. So (42) becomes

$$\zeta(s) * \Pi_{allprimes}(1-1/p^s) = 1 \tag{47}$$

We have to show that the left-hand side is also 1. To do this we'll successively multiply $\zeta(s)$ by the factors of the product for successive primes, namely $1-1/2^s$, then $1-1/3^s$, then $1-1/5^s$, etc. So first we multiply

$$1/1^s + 1/2^s + 1/3^s + 1/4^s + 1/5^s + 1/6^s + 1/7^s + 1/8^s + ... \tag{48}$$

by $1 - 1/2^s$. The 1 part multiplying $\zeta(s)$ just gives $\zeta(s)$. Using (43) and (48) the $1/2^s$ part multiplying $\zeta(s)$ gives

$$1/2^s + 1/4^s + 1/6^s + 1/8^s + 1/(10)^s + 1/(12)^s + 1/(14)^s + 1/(16)^s + ... \tag{49}$$

since for instance $1/2^s * 1/2^s$ equals $1/4^s$. When this is subtracted from $\zeta(s)$, all of the multiples of 2 disappear giving

$$1/1^s + 1/3^s + 1/5^s + 1/7^s + ... \tag{50}$$

This is similar to step (2) in the sieve of Eratosthenes in which we set all the elements of the array to 1 that have subscripts that are multiples of the starting index When we multiply (50) by $1-1/3^s$, the 1 part gives us back (50), then we multiply (50) by $1/3^s$ giving

$$1/3^s + 1/9^s + 1/(15)^s + 1/(21)^s + ... \tag{51}$$

When we subtract this from (50), all the multiples of 3 disappear giving,

$$1/1^s + 1/5^s + 1/7^s + ... \tag{52}$$

When we multiply by $1 - 1/5^s$, all the factors of 5 disappear. The 1 part in all of these multiplications serves the part of the accumulator in statement like **accum = accum - powerOfS** executed in a loop. When we are finished, the only term in $\zeta(s)$ that remains is $1/1^s$ which equals 1.

The reason we take the product over only the primes is that all multiples of a given prime disappear. If we included non-primes in the product, the non-prime term would have no corresponding term in the modified $\zeta(s)$ to cancel.

16 Exponential Growth

This section was inspired by Professor Bartlett's Youtube presentation to be found at:

http://www.youtube.com/watch?v=F-QA2rkpBSY

If a process increases the value of quantity N at a steady rate r, e.g. 5 percent a year, it is described by an exponential function:

$$N = N_0 e^{rt} \tag{53}$$

where t represents the time, N_0 is the amount at $t = 0$ and N is the amount at time t. How do we calculate the doubling time, i.e., the time when N is $2 \cdot N_0$. We have to solve the equation

$$2 \cdot N_0 = N_0 e^{rt} \tag{54}$$

If we cancel the N_0 on both sides, we get

$$2 = e^{rt} \tag{55}$$

we solve this by taking the *log* to the base e of both sides[¶] and get

$$log_e 2 = rT \tag{56}$$

Now $log_e 2$ is about 0.69. If r is 0.05, then the doubling time T is

$$0.69/0.05 = T \tag{57}$$

Multiply the numerator and denominator by 100 and we get $T = 69/5$ or about fourteen years. If we approximate 69 by 70, then the

$$T = 70/r \tag{58}$$

So if the rate of growth is 10 percent, the doubling time is seven years

[¶]The quantity log_e is called ln