Chapter 8. Some Overwhelming Numbers

> Mr Podsnap settled that whatever he put behind him he put out of existence…. Mr Podsnap had even acquired a peculiar flourish of his right arm in often clearing the world of its most difficult problems, by sweeping them behind him. Charles Dickens, *Our Mutual Friend*.

Introduction

In previous chapters we have paid little, if any, attention to the size of the problems we have considered, where "size" may refer to the number of variables or the total number of arithmetical operations or the time required to carry out the calculations. Indeed we have behaved like Mr. Podsnap and have simply ignored all such matters. In the present chapter we shall see that there are "small" problems that may be solved simply and in a timely fashion with the current technology – or even by hand – and "big" problems which become unmagineable as their size increases even with the technology which we have now or which may become available in the next few years. In this chapter we shall have a brief look at some of these problems, mostly big ones, and see why their consideration is of importance. First, though, we must define what we mean by the terms "small" and "big".

A very simple small problem is the addition of two integers. The addition of two n-digit integers requires n additions for the addition of the pairs of digits in the two numbers and not more than n further additions to accommodate the carries, giving a total of $2n$ additions. If we double the number of digits in each of the operands, then the total number of additions becomes $2\times(2n)$ or $4n$ which is twice the number of additions required for two n-digit integers. Similarly the multiplication of two n-digit integers requires the n^2 multiplications of all possible pairs of digits so that doubling the length of the operands quadruples the number of digit-by-digit multiplications. Thus for addition the number of operations is a linear function of the number of digits in each of the operands while for multiplication the number of operations is a quadratic function. Such problems in which the number of steps and thus the time required for solution is proportional to some power of the size of the problem are referred to as *polynomial-time*

problems. More formally for a problem of size n the number of steps required for the solution of a polynomial-time problem is not greater than An^k for some fixed integers A and k and for any value of n.

On the other hand, an *exponential-time* problem is one in which the number of steps required for its solution is some function such as 2^n, 3^n, …, n^n or $n!$. A very simple example of such a problem is that of enumerating the permutations of a number of distinct items. For example, if there are 5 items the number of different arrangements is 5! or 120, and if there are 10 items then the number of arrangements is 10! or 3628800. So doubling the number of items does not double or quadruple the number of arrangements to be listed but increases it by a multiplicative factor of (10!)/5! or 30240. Listings of this size, if they were ever required, could be handled quite easily with a computer. However, as the number of objects increases the problem soon becomes unmanageable even with the fastest computers. For example, if we could shuffle a deck of 52 playing cards once every nanosecond and if we had started with the creation of the Earth about five billion years ago, then by now we would have generated only about 10^{26} of the 52! or approximately 10^{68} required permutauions.

In this chapter we will give a few examples of exponential-time problems, discuss the relationship between polynomial- and exponential-time problems, and finally show how these matters are related to the important area of public key encryption. First, though, in order not to completely neglect polynomial-time problems, we shall give a very brief discussion of the well-known bubble sort algorithm.

Bubble sort

The bubble sort algorithm for sorting a list of numbers in non-decreasing order is an interesting example of a polynomial-time problem. (We might note that sorting in **J** may be accomplished by the expression `/:~X` for a list `X`.) For a list of length n the bubble sort algorithm may be described briefly as follows: Starting at the left of the list consider each overlapping pair of items in turn, interchanging them if the first item is greater than the second. This requires $n - 1$ comparisons and up to n - 1 interchanges, and places the largest item at the right-hand end of the list. Repeat this process starting at the left of the list but consider only the first $n - 2$ overlapping pairs. This will place the second largest item in the second last position in the list. A total of $n - 1$ repetitions of this procedure, decreasing the number of overlapping pairs considered by 1 in each repetition, will give the required sorted list. The total possible number of interchanges is

$(n\text{-}1) + (n\text{-}2) + \ldots + 3 + 2 + 1$

which is equal to $n(n\text{-}1)/2$. If the list to be sorted is already in sorted order the number of interchanges is 0 and if the items are in random order the average numberof interchanges is $n(n\text{-}1)/4$. The following shows the result of each iteration in the sorting of the list

```
9 15 4 20 3 24 6 5
```

where the items given in bold are those items at the end of the list which are in the correct sorted position after each iteration:

```
9 15  4 20  3 24  6  5
9  4 15  3 20  6  5 24
4  9  3 15  6  5 20 24
4  3  9  6  5 15 20 24
3  4  6  5  9 15 20 24
3  4  5  6  9 15 20 24
3  4  5  6  9 15 20 24
3  4  5  6  9 15 20 24
```

The defined verb `Sort` given at the end of this chapter is a **J** implemenation of the bubble sort algorithm. Execution times on randomly generated lists of 250, 500 and 1000 items were approximately 0.9 seconds, 3.6 seconds and 14.4 seconds, respectively, showing that doubling the size of the problem quadruples the number of steps as is to be expected. We may note that sorting the 1000-item list using the **J** expresion /:~ *X* required only about 0.00008 seconds.

A couple of legends

An interesting, exponential-time problem is provided by the Tower of Hanoi puzzle which appears to have originated in late nineteenth-century France. It consists of three pegs fastened to a stand togeher with seven or eight wooden discs of different radii each with a hole in the centre. The discs are placed initially on one of the pegs with the largest disc on the bottom, followed by the second largest, and then the third largest, …, until the smallest disc is on top. The object of the puzzle is to transfer the discs from one peg to one of the other pegs by moving one disc at a time and without ever placing a disc on top of a smaller disc. It is a relatively simple matter, although we shall omit the proof, to show that with n discs the minimum number of moves required to transfer the discs from one peg to one of the other pegs is equal to $2^n - 1$. Therefore the minimum number of moves required to solve the puzzle for seven discs is 127.

Another way of thinking about solving the puzzle is to assume there is a solution for n-1 discs which will allow one to move the top n-1 discs in a single move from the first peg to the third peg, then move the bottom disc from the first peg to the second, and finally move the n-1 discs on the third peg to the second peg in a single move. Of course, the two all-at-once moves of the top n-1 discs would still have to be carried out in a similar manner. This argument suggests a

recursive solution to the Tower of Hanoi puzzle expressed in the defined **J** verb `Hanoi` given at the end of the chapter. The expression `Hanoi 3` will give the solution

```
+---+---+---+---+---+---+---+
|1 2|1 3|2 3|1 2|3 1|3 2|1 2|
+---+---+---+---+---+---+---+
```

which indicates a first move of the top disc from Peg 1 to Peg 2, then a move of the second disc to Peg 3, etc. The following gives the first 12 moves for a puzzle with 7 discs:

```
+---+---+---+---+---+---+---+---+---+---+---+---+
|1 2|1 3|2 3|1 2|3 1|3 2|1 2|1 3|2 3|2 1|3 1|2 3| ...
+---+---+---+---+---+---+---+---+---+---+---+---+
```

The following story, retold in George Gamow's delightful *One, Two, Three ... Infinity*, gives a fictitious account of the Tower of Hanoi puzzle:

> In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three damond needles, each a cubit high and as thick as the body of a bee, On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disc resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the tower of Bramah.. Day and night unceasingly the priest transfers the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple, and Bramins alike will crumble into dust, and with a thunder clap the world will vanish.

Since there are 64 discs, the number of moves is $2^{64} - 1$, which may be calculated exactly in **J** by the expression `_1+2^64x`, which is equal to 18,446,744,073,709,551,615. Gamow remarked that if the priests "worked day and night without holidays or vacation" making one move every second the time required to move the discs from one needle to another would be slightly more than fifty-eight thousand billion years. He goes on to say that the total age of the universe is estimated to be less than 2×10^{10} years which is considerably smaller tham the 5.8×10^{13} years required by the priests to move the discs.

The above legend is reminiscent of Arthur C. Clark's short story "The Nine Billion Names of God" in which Tibetan monks are attempting to achieve the same end by finding all the names of the Supreme Being and have acquired a computer to enumerate and list the names. At the end of the project, the two

American maintenance engineers, who have been very skeptical all along, leave the monastery to return home. They look up, and "Overhead, without any fuss, the stars were going out."

In his book Gamow also recounts another legend which while it does not lead to an exponential-time problem does result in an exponential consumption of resources. An Indian king wishing to reward his grand vizier for inventing the game of chess asked him what he would desire. The vizier's reply seemed to be very modest in that it consisted of an amount of wheat resulting from placing one grain of wheat on the first square of the chessboard, 2 on the second, 4 on the third, and so on with the number of grains doubling on successive squares. Therefore the total number of grains of wheat required would be the sum of the geometric progression

$1 + 2 + 2^2 + 2^3 + \ldots + 2^{63}$

which is $2^{64} - 1$. This number, which is the same as the total number of moves in the 64-disc Tower of Hanoi problem, is approximately 1.844×10^{19}. Using Gamow's estimate of 5 million grains of wheat in a bushel and the total world's wheat production in 2009 as 2.7×10^{10} bushels, or 13.5×10^{15} grains, a little calculation will show that today the required number of grains of whaeat could be placed on only the first 51 squares of the chessboard. Incidentally, Gamow remarks that the king was a "victim of overwhelming numbers" which suggested the title of the present chapter. (A more modern example of the effects of repeated doubling is the arithmetical problem, said to be famous in the England of Charles Dickens, of the horsedealer who agrees to sell a horse for a sum calculated by doubling the price for each horseshoe nail, beginning with a farthing for the first nail.)

Some problems from the real world

A famous exponential-time problem is the travelling-salesman problem in which a person is required to visit each of n cities and to return to the starting city without visiting any city more than once and also minimizing the total time required or distance travelled. The problem was first posed in the 1930s by the Austrian mathematician Karl Menger and has become one of the most intensively studied problems in combinatorial optimization. A solution by evaluation of the costs for each of the $n!$ routes for n cities is obviously out of the question except for very small values of n. For example, for 10 cities the number of routes is 3,628,800 which could be handled on today's computers, but for 25 cities – not an unreasonable number in a practical situation – the number of routes is 25! which is 15,511,210,043,330,985,984,000,000 or more than 15 trillion trillion which is a much larger number than that encountered by the priests solving the Tower of Hanoi puzzle. However methods have been developed that have been applied to situations involving hundreds of cities. A related problem is the Hamiltonian-circuit problem of finding a tour,

regardless of length, through a number of cities which passes through each city once only. These problems are of more than simply academic interest and have applications in such diverse areas as the design of microchips and DNA sequencing.

Another problem which can be shown to have an exponitial-time algorithm is the simplex algorithm in linear programming which involves finding the maximum (or minimum) of a linear objective function with an arbitrary number, say n, of independent variables which are required to be non-negative and also satisfy a number of linear constraints. The variables are constrained to lie within or on the surface of an n-dimensional polygon with the optimal solution at one of the vertices. The simplex algorithm starts with an initial solution at one of the vertices and then proceeds along the edges from one vertex to another vertex such that the value of the objective function continually increases (or decreases for a minimization problem) until its optimal value is attained. The algorithm has been applied successfully to practical problems with up to several hundred variables, and although its exponential-time nature may be demonstrated with some specially prepared problems, its behaviour with practical problems is almost always a linear function of the number of variables.

An interesting and very importatnt example of an exponential-time problem occurs in the cryptographic method of the secure transmission of data known as public key ecryption. However before we consider this problem we shall give in the next section a few remarks on cryptography for the reader who is not familiar with the subject. Persons wishing a most readable introduction to cryptography are referred to Simon Singh's *The Code Book. The Science of Secrecy from Ancient Egypt to Quantum Crytography*. An informal and entertaining account of public key encryption is given in *In Code. A Mathematical Journey* by Sarah Flannery in collaboration with her father, David Flannery.

A few notes on cryptography

Cryptography is the science and art of both concealing the meaning of a message by enciphering it according to some given algorithm and also deciphering an enciphered message to recover its original meaning. The two main ciphering methods are the *substitution cipher* in which each letter of a message is replaced by another character but retains its position in the message and the *transposition cipher* in which each letter remains unchanged but changes its position. The original message is referred to as the *plaintext* and the enciphered message as the *ciphertext*. We shall consider only substitution ciphers.

One of the earliest examples of a substitution cipher is attributed to Julius Caesar and is therefore known as the Caesar cipher. In this cipher each letter in a message is replaced by the

letter which occurs three places ahead of it in the alphabet. For example, if we wish to encipher the three names “Kyle”, “Maia” and “Ryan”, we could write the plaintext **kylemaiaryan** as a single word and the ciphertext would be **NBOHPDLDUBDQ**. The shift need not be limited to 3 places, and may be any value between 1 and 26 where a shift of 26 leaves the message unchanged. With a shift of 7 the plaintext given above would give the ciphertext **RFSLTHPHYFHU**. Caesar ciphers are relatively easy to break since there are only 25 of them and each may be tried in turn on a portion of the enciphered message until the correct one is found and then applied to the entire message. If the message is sufficiently long a frequency analysis of the relative frequencies of the letters in the enciphered text may assist in the decipherment. A more complicated substitution cipher is obtained by using a random permutation of the cipher alphabet so that, for example, each letter of the plaintext in the first row would be enciphered by the letter below it in the second row in the following table:

abcdefghijklmnopqrstuvwxyz

IOHCFPTVMGDAZJYUWKXRLNQSBE

With this substitution our example would give the ciphertext **DBAFZIMIKBIZ**. However such ciphers can still be broken by frequency analysis.

Caesar ciphers are examples of *monoalphabetic* ciphers since only one cipher alphabet is used in a given message. The cipher alphabet used is given by a *key* indicating the amount of the Caesar shift or with the last example the random permutation. However it is possible to define *polyalphabetic* ciphers in which more than one cipher alphabet is used in a message. The best known of these polyalphabetic ciphers is the Vigenère cipher named after the 16th century French diplomat Blaise de Vigenère who further developed and promoted a method of encipherment which had been suggested during the previous century by the Italian architect Leon Alberti. In the Vigenère cipher there are 26 cipher alphabets corresponding to Caesar shift alphabets with shifts of 1, 2, …, 26 letters, and the cipher alphabet chosen for each letter of the plaintext is given by a keyword. For example, if we wish to encipher the plaintext using the keyword **KEITH**, we could write the plaintext with the keyword repeated above it as many times as is necessary to give the following array:

KEITHKEITHKE

kylemaiaryan

This indicates that the first letter **k** of the plaintext would be enciphered with a Caesar shift of 10, giving a **U**, since it occurs below **K** which is the eleventh letter of the alphabet. Similarly the second letter **y** would be enciphered with a Caesar shift of 4 giving **C** since **E** is the fifth letter of the alphabet, and so on. Thus the ciphertext would be enciphered as **UCTWQKMIKFKR**. (The

enciphering is facilitated by the use of tables known not surprisingly as Vigenère tables which need not concern us here.) For many years the Vigenère cipher was considered unbreakable, and indeed was known as *le chiffre indéchiffrable*. However, a method of breaking it was found in the middle of the 19th century independently by Charles Babbage and by Friedrich Wilhelm Kasiski, a retired Prussian army officer.

Public key encryption

All of the keys used in the above examples, and indeed in almost all, if not all, ciphering methods proposed until quite recently had two important characteristics. First they were *symmetric* in that the key used to encipher a message was used to decipher it. For example, if we received the ciphertext **KHQUB** known to have been enciphered with a Caesar shift of 3, we could decipher it to obtain the plaintext **henry**. Implicit in this symmetry property is the second property that the receiver must know the enciphering key, and therefore must have been sent the key in some manner. It is useless to encipher the key with another key and then transmit the enciphered key since the same problem arises in that the receiver must know the key used to encipher the original key. The solution adopted was to use couriers whose task it was to personally distribute keys to intended recipients of enciphered messages. The problems involved in such a procedure in commercial and peacetime settings were only exacerbated in times of war. The solutions that were proposed to the key distribution problem were to provide a public key that could be used to encipher a message and also a private key known only to the intended recipient that was required to decipher it. Of the several methods we shall consider very briefly only the RSA Public Key Method named after its originators Ronald Rivest, Adi Shamir and Leonard Adleman of Stanford University which was first announced by Martin Gardner in his August, 1977 column in *Scientific American* and later published in one of Gardner's many collections of his columns. We shall limit our very few remarks only to an illustration of the polynomial- and exponential-time aspects of the method. Readers wishing more details are referred to Singh's book cited above and to the references given there.

The basis of the RSA method is that the key used to encrypt a message is made public, as the name would suggest, and is available to any person wishing to send a message to another person or organization while another key, known only to the recipient is required to decrypt the message. The public key consists of the product of two very large prime numbers while the decryption key requires a knowledge of both of these primes. Now multiplication is a polynomial-time problem while factorization is an exponential-time problem. (As a simple example find the product of the two three-digit primes 613 and 691 and then find the two prime factors of the six-digit number

448211.) Thus a security may be achieved by generating a sufficiently large number which cannot be factored into its prime factors in a any reasonable time. Singh says that for some banking tyransactions a public key of the order of 10^{308} is necessary, and then remarks that "[T]he combined efforts of a hundred million personal computers would take more than one thousand years to crack such a cipher."

P vs NP

Problems for which there is a polynomil-time algorithm are said to be in class **P**. Another class of problems are those for which the solution, if known, may be written down and checked for correctness in polynomial time. These are referred to as being in class **NP** for non-deterministic polynomial. The Tower of Hanoi is obviously one such problem. The travelling-salesman may be converted to an **NP** problem by stating it as a "decision problem" by simply asking if a route may be found that is not greater in time or cost than a given amount. Any proposed solution may be obviously checked in polynomial time if it satisfies the conditions of passing through all cities once only within a minimum time or distance. Among the **NP** class of problems are those which are said to be **NP**-complete with the property that the input to any one of them may be converted to a form suitable for input to any other in polynomial time. Included in the large class of these problems are the travelling-salesman problem, the related Hamilton circuit problem, the multiprocessor scheduling problem, and the map-colouring problem of determining whether a map may be coloured with only three colours so that no two countries with the same colour have a common boundary. All of the **NP**-complete problems have exponential-time algorithms and it is believed that none has a polynomial-time algorithm. However if any one of them is found to have a polynomial-time solution, then all of them will have such a solution including the RSA public key method of the last section..

J programs

```
NB. Bubble sort
Sort=: 3 : 0
n=. #X=. y.
R=:<X
i=. 1
while. i < n do.
   j=. 0
   while. j < n-i do.
     if. >/x=. (j,j+1){X do.
       X=. x ((j+1),j)}X
     end.
 j=. >:j
```

```
 end.
i=. >:i
R=: R,<X
end.
X
)

NB. Tower of Hanoi
Hanoi=: 3 : 0
1 2 3 Hanoi y.
:
'P1 P2 P3'=. x.
n=. y.
R=. i.0
if. N > 0 do.
   R=. R, (P1,P3,P2) Hanoi n-1
   R=. R,<P1,P2
   R=. R, (P3,P2,P1) Hanoi n-1
end.
R
)
```