Computer Science Unplugged

Teacher’s Edition Parts IV, V and VI

An enrichment and extension programme for primary-aged children

Created by

Tim Bell, Ian H. Witten and Mike Fellows

Adapted for classroom use by Isaac Freeman, based on work by Robyn Adams and Jane McKenzie

Illustrations by Malcolm Robinson, Gail Williams, Matt Powell and Isaac Freeman
Introduction

This book adds 8 more activities to the Teacher’s Edition of Computer Science Unplugged. These activities were in the original 1999 edition, but have been updated to the style developed for classroom use in the 2002 Teacher’s Edition.

This book is available for free download for personal and educational use thanks to a generous grant by Google, Inc. It is distributed under a Creative Commons Attribution-NonCommercial-ShareAlike licence, which means that you are free to share (copy, distribute, and transmit) the book. It also allows you to remix the book. These are only available under the following conditions: you include attribution to the authors, you do not use this book for commercial purposes, and if you alter, transform or build upon this work, you share under the same or similar license. More details of this license can be found online by searching: CC BY-NC-SA 3.0.

We encourage the use of this material in educational settings, and you are welcome to print your own copy of the book and distribute worksheets from it to students. We welcome enquiries and suggestions, which should be directed to the authors (see csunplugged.org).

This book is being translated into several other languages. Please check the web site for information about the availability of translations.
Contents

Really hard problems—Intractability ......................................................... 109

The poor cartographer—Graph coloring .................................................. 112
Tourist town—Dominating sets ................................................................. 125
Ice roads—Steiner trees ................................................................. 133

Sharing secrets and fighting crime—Cryptography ............................... 144

Sharing secrets—Information hiding protocols ...................................... 148
The Peruvian coin flip—Cryptographic protocols ................................. 152
Kid Krypto—Public-key encryption .......................................................... 162

The human face of computing—Interacting with computers ............... 172

The chocolate factory—Human interface design ...................................... 175
Conversations with computers—The Turing test ................................... 187
Part VI

Really hard problems—Intractability
Intractability

Are there problems that are too hard even for computers? Yes. We will see in Activity 20 that just having a conversation—chatting—is something computers can't do, not because they can't speak but because they can't understand or think of sensible things to say, but that's not the kind of hard problem we're talking about here - it's not that computers can't have conversations, more that we don't know just how we do it ourselves and so we can't tell the computer what to do. But in this section we're going to look at problems where it's easy to tell the computer what to do—by writing a program—but the computer can't do what we want because it takes far too long: millions of centuries, perhaps. Not much good buying a faster computer: if it were a hundred times faster it would still take millions of years; even one a million times faster would take hundreds of years. That's what you call a hard problem—one where it takes far longer than the lifetime of the fastest computer imaginable to come up with a solution!

The activities in Part II on algorithms showed you how to find ways of making computer programs run more efficiently. In this section we look at problems for which no efficient solutions are known, problems that take computers millions of centuries to solve. And we will encounter what is surely the greatest mystery in computer science today: that no-one knows whether there's a more efficient way of solving these problems! It may be just that no-one has come up with a good way yet, or it may be that there is no good way. We don't know which. And that's not all. There are thousands of problems that, although they look completely different, are equivalent in the sense that if an efficient method is found to solve one, it can be converted into an efficient method to solve them all. In these activities you will learn about these problems.

For teachers

There are three activities in this section. The first involves coloring maps and counting how many colors are needed to make neighboring countries different. The second requires the ability to use a simple street map, and involves placing ice-cream vans at street corners so that nobody has to go too far to get an ice-cream. The third is an outdoor activity that uses string and pegs to explore how to make short networks connecting a set of points.

The activities provide a hands-on appreciation of the idea of complexity—how problems that are very simple to state can turn out to be incredibly hard to solve. And these problems are not abstruse. They are practical questions that arise in everyday activities such as mapping, school time-tableing, and road building. The computational underpinning rests on a notion called “NP-completeness” that is explained in the What's it all about? sections at the end of each activity. Although the activities themselves can be tackled in any order, these sections are intended to be read in the order in which they appear. By the time you reach the end you will have a firm grip on the most important open question in contemporary computer science.

The technical name for this part is “intractability” because problems that are hard to solve are called intractable. The word comes the Latin tractare meaning to draw or drag, leading to the modern usage of tractable as easy to handle, pliant, or docile. Intractable problems are ones that are not easily dealt with because it would take too long to come up with an answer. Although it may sound esoteric, intractability is of great practical interest because a breakthrough in this area would have major ramifications for many different lines of research. For example, most cryptographic codes rely on the
intractability of some problems, and a criminal who managed to come up with an efficient solution could have a field day decoding secrets and selling them, or—more simply—just making phoney bank transactions. We will look at these things in Part V—Cryptography.
Activity 13

The poor cartographer—Graph coloring

Summary
Many optimization problems involve situations where certain events cannot occur at the same time, or where certain members of a set of objects cannot be adjacent. For example, anyone who has tried to timetable classes or meetings will have encountered the problem of satisfying the constraints on all the people involved. Many of these difficulties are crystallized in the map coloring problem, in which colors must be chosen for countries on a map in a way that makes bordering countries different colors. This activity is about that problem.

Curriculum Links
✓ Mathematics: Number Level 2 and up. Exploring numbers in other bases.
  Representing numbers in base two.
✓ Mathematics: Algebra Level 2 and up. Continue a sequential pattern, and describe a rule for this pattern. Patterns and relationships in powers of two.

Skills
✓ Problem solving.
✓ Logical reasoning.
✓ Algorithmic procedures and complexity.
✓ Communication of insights.

Ages
✓ 7 and up

Materials
✓ a whiteboard or similar writing surface.
Each student will need:
✓ a copy of one or more of the worksheets,
✓ movable small colored markers (e.g. counters or poker chips), and
✓ four crayons of different colors (or colored pencils, felt tips etc.)
Graph Coloring

Introduction
This activity revolves around a story in which the students have been asked to help out a cartographer, or map-maker, who is coloring in the countries on a map. It doesn't matter which color a country is, so long as it's different to all bordering countries.

For example, this map shows four countries. If we color Northland red, then Westland and Eastland cannot be red, since their border with Northland would be hard to see. We could color Westland green, and it is also acceptable to color Eastland green because it does not share a border with Westland. (If two countries meet only at a single point, they do not count as sharing a border and hence can be made the same color.) Southland can be colored red, and we end up needing only two colors for the map.

In our story, the cartographer is poor and can't afford many crayons, so the idea is to use as few colors as possible.

Discussion
Describe the problem that the students will be working on, demonstrating the coloring process on a blackboard.

Give out a copy of the first worksheet. This map can be colored correctly using only two colors. Although restricting the number of colors to just two might sound particularly challenging, the task is quite simple compared with maps that require more colors because there is very little choice about what color each country can be.
Have the students try to color the map in with only two colors. In the process they may discover the “has-to-be” rule: once one country is colored in, any bordering country has to be the opposite color. This rule is applied repeatedly until all countries are colored in. It is best if the students can discover this rule for themselves, rather than being told it, as it will give them a better insight into the process.

As students complete each exercise they can be given the next sheet to try.

The students may also discover that it is better to use place-holders, such as colored counters, instead of coloring the countries straight away, since this makes it easier for them to change their mind.

For older students, ask them to explain how they know that they have found the minimum number of colors. For example, at least three colors are required for this map because it includes a group of three countries (the largest three), each of which has borders with the other two.

If a student finishes all the sheets early, ask them to try to devise a map that requires five different colors. It has been proved that any map can be colored with only four colors, so this task will keep them occupied for some time! In our experience students will quickly find maps that they believe require five colors, but of course it is always possible to find a four-color solution to their maps.
Worksheet Activity: Graph Coloring 1

Color in the countries on this map with as few colors as possible, but make sure that no two bordering countries are the same color.
Worksheet Activity: Graph Coloring 2

Color in the countries on this map with as few colors as possible, but make sure that no two bordering countries are the same color.
Worksheet Activity: Graph Coloring 3

Color in the countries on this map with as few colors as possible, but make sure that no two bordering countries are the same color.
Worksheet Activity: Graph Coloring 4

Color in the countries on this map with as few colors as possible, but make sure that no two bordering countries are the same color.
Variations and Extensions

There is a simple way to construct maps that require only two colors, as shown here. This map was drawn by overlaying closed curves (lines whose beginning joins up with their end). You can draw any number of these curves, of any shape, on top of each other, and you will always end up with a map that can be colored with two colors. Students can experiment with creating this type of map.

Four colors are always enough to color a map drawn on a sheet of paper or on a sphere (that is, a globe). One might wonder (as scientists are paid to do) how many colors are needed for maps drawn on weirder surfaces, such as the torus (the shape of a donut). In this case, one might need five colors, and five is always enough. Students might like to experiment with this.

There are many other entertaining variations on the map-coloring problem that lead off into directions where much is currently unknown. For example, if I am coloring a map on a sheet of paper by myself, then I know that if I work cleverly, four colors will be enough. But suppose that instead of working alone I am working with an incompetent (or even adversarial) partner, and we take turns at choosing the color for countries. Assume that I work cleverly, while my partner only works “legally” as we take turns coloring countries on the map. How many crayons need to be on the table in order for me in my cleverness to be able to make up for my partner’s legal but not very bright (or even subversive) moves? The maximum number isn’t known! In 1992 it was proved that 33 crayons will always be enough, and in 2008 this was improved by a proof that 17 would be sufficient, but we still don’t know that this many is ever actually required. (Experts conjecture that fewer than 10 colors are sufficient.) Students might enjoy acting out this situation, which can be played as a two-person game where you try to maximise the number of colours that your opponent needs.

In another variation of map coloring know as empire coloring, we start with two different maps on two sheets of paper having the same number of countries. Each country on one of the maps (say, the Earth) is paired with exactly one country on the other map (which might be colonies on the Moon). In addition to the usual coloring requirement of different colors for countries that share a border (for both maps) we add the requirement that each Earth country must be colored the same as its colony on the Moon. How many colors do we need for this problem? The answer is currently unknown.
What’s it all about?

The map coloring problem that we have explored in this activity is essentially to find the minimum number of colors—two, three, or four—that are necessary to color a particular map. The conjecture that any map can be colored using only four colors was formulated in 1852, but it was not proved until 1976. Computer science is full of unsolved problems, and knowing that the four-color theorem was proved after more than 120 years of attention from researchers is an encouragement to those working on other problems whose solution has eluded them for decades.

Map coloring belongs to a general class of problems known as “graph coloring.” In computer science, a graph is an abstract representation of relationships, as shown here.

As mentioned in Activity 9 on the Muddy City, the term graph is used in a different sense in mathematics to mean a chart displaying numerical data, such as a bar graph, but the graphs that computer scientists use are not related to these. In computer science, graphs are drawn using circles or large dots, technically called “nodes,” to denote objects, with lines between them to indicate some sort of relationship between the objects. The above graph happens to represent the map at the beginning of this activity. The nodes represent the countries, and a line between two nodes indicates that those two countries share a common border. On the graph, the coloring rule is that no connected nodes should be allocated the same color. Unlike a map, there is no limit to the number of colors that a general graph may require, because many different constraints may be drawn in as connecting lines, whereas the two-dimensional nature of maps restricts the possible arrangements. The “graph coloring problem” is to find the minimum number of colors that are needed for a particular graph.

In the graph on the right the nodes correspond to subjects in a school. A line between two subjects indicates that at least one student is taking both subjects, and so they should not be timetabled for the same period. Using this representation, the problem of finding a workable timetable using the minimum number of periods is equivalent to the coloring problem, where the different colors correspond to different periods. Graph coloring algorithms are of great interest in computer science, and are used for many real-world problems, although they are probably never used to color in maps!—our poor cartographer is just a fiction.

There are literally thousands of other problems based on graphs. Some are described elsewhere in this book, such as the minimal spanning tree of Activity 9 and the dominating sets of Activity 14. Graphs are a very general way of representing data and can be used to represent all sorts of situations, such as a map made up of roads and
intersections, connections between atoms in a molecule, paths that messages can take through a computer network, connections between components on a printed circuit board, and relationships between the tasks required to carry out a large project. For this reason, problems involving graph representations have long fascinated computer scientists.

Many of these problems are very difficult—not difficult conceptually, but difficult because they take a long time to solve. For example, to determine the most efficient solution for a graph coloring problem of moderate size—such as finding the best way to timetable a school with thirty teachers and 800 students—could take years, even centuries, for a computer using the best known algorithm. The problem would be irrelevant by the time the solution was found—and that’s assuming the computer doesn't break down or wear out before it finishes! Such problems are only solved in practice because we are content to work with sub-optimal, but still very good, solutions. If we were to insist on being able to guarantee that the solution found was the very best one, the problem would be completely intractable.

The amount of computer time needed to solve coloring problems increases exponentially with the size of the graph. Consider the map coloring problem. It can be solved by trying out all possible ways to color the map. We know that at most four colors are required, so we need to evaluate every combination of assigning the four colors to the countries. If there are \( n \) countries, there are \( 4^n \) combinations. This number grows very rapidly: every country that is added multiplies the number of combinations by four, and hence quadruples the solution time. Even if a computer were invented that could solve the problem for, say, fifty countries in just one hour, adding one more country would require four hours, and we would only need to add ten more countries to make the computer take over a year to find the solution. This kind of problem won't go away just because we keep inventing faster and faster computers!

Graph coloring is a good example of a problem whose solution time grows exponentially. For very simple instances of the problem, such as the small maps used in this activity, it is quite easy to find the optimal solution, but as soon as the number of countries increases beyond about ten, the problem becomes very difficult to do by hand, and with a hundred or more countries, even a computer can take many years to try out all the possible ways of coloring the map in order to choose the optimal one.

Many real-life problems are like this, but must be solved anyway. Computer scientists use methods that give good, but not perfect, answers. These heuristic techniques are often very close to optimal, very fast to compute, and give answers that are close enough for all practical purposes. Schools can tolerate using one more classroom than would be needed if the timetable were perfect, and perhaps the poor cartographer could afford an extra color even though it is not strictly necessary.

No-one has proved that there isn't an efficient way to solve this sort of problem on conventional computers, but neither has anyone proved that there is, and computer scientists are sceptical that an efficient method will ever be found. We will learn more about this kind of problem in the next two activities.
Further reading
Harel discusses the four-color theorem, including its history, in *Algorithmics*. More aspects of the map-coloring problem are discussed in *This is MEGA-Mathematics!* by Casey and Fellows. Kubale’s 2004 book, *Graph Colorings*, includes a history of the problem. There are many websites that cover this topic.
Solutions and hints

This is the only possible solution for the map on worksheet 1 (of course, the choice of colors is up to the student, but only two different colors are required).

The map at the top of worksheet 2 can be colored correctly using three colors, while the one at the bottom requires four. Here are two possible solutions.
The map on worksheet 3 is a simpler three-color map, with a possible solution shown here.

Solution for worksheet 4 using just two colors (shaded and white).
Activity 14

Tourist town—Dominating sets

Summary
Many real-life situations can be abstracted into the form of a network or “graph” of the kind used for coloring in Activity 13. Networks present many opportunities for the development of algorithms that are practically useful. In this activity, we want to mark some of the junctions, or “nodes,” in such a way that all other nodes are at most one step away from one of the marked ones. The question is, how few marked nodes can we get away with? This turns out to be a surprisingly difficult problem.

Curriculum Links
✓ Mathematics Level 2: Position and orientation

Skills
✓ Maps.
✓ Relationships.
✓ Puzzle solving.
✓ Iterative goal seeking.

Ages
✓ 7 and up

Materials
Each group of students will need:

✓ a copy of the blackline master Ice Cream Vans, and
✓ several counters or poker chips of two different colors.

You will need

✓ an projector image of the blackline master Ice Cream Vans Solution on a whiteboard, or a whiteboard to draw it on.
Dominating Sets

Introduction
The Ice Cream Vans worksheet shows a map of Tourist Town. The lines are streets and the dots are street corners. The town lies in a very hot country, and in the summer season ice-cream vans park at street corners and sell ice-creams to tourists. We want to place the vans so that anyone can reach one by walking to the end of their street and then at most one block further. (It may be easier to imagine people living at the intersections rather than along the streets; then they must be able to get ice-cream by walking at most one block.) The question is, how many vans are needed and on which intersections should they be placed?

Discussion
1. Divide the students into small groups, give each group the Tourist Town map and some counters, and explain the story.

2. Show the students how to place a counter on an intersection to mark an ice-cream van, and then place counters of another color on the intersections one street away. People living at those intersections (or along the streets that come into them) are served by this ice-cream van.

3. Have the students experiment with different positions for the vans. As they find configurations that serve all houses, remind them that vans are expensive and the idea is to have as few of them as possible. It is obvious that the conditions can be met if there are enough vans to place on all intersections—the interesting question is how few you can get away with.

4. The minimum number of vans for Tourist Town is six, and a solution is shown here. But it is very difficult to find this solution! After some time, tell the class that six vans suffice and challenge them to find a way to place them. This is still quite a hard problem: many groups will eventually give up. Even a solution using eight or nine vans can be difficult to find.
5. The map of Tourist Town was made by starting with the six map pieces at the bottom of the Ice Cream Vans solution worksheet, each of which obviously requires only one ice-cream van, and connecting them together with lots of streets to disguise the solution. The main thing is not to put any links between the solution intersections (the open dots), but only between the extra ones (the solid dots). Show the class this technique on the board or using a projector.

6. Get the students to make their own difficult maps using this strategy. They may wish to try them on their friends and parents—they will find that they can create puzzles that they can solve but others can’t! These are examples of what is called a “one-way function”: it’s easy to come up with a puzzle that is very difficult to solve—unless you’re the one who created it in the first place. One-way functions play a crucial role in cryptography (see Activities 17 and 18).
Worksheet Activity: Ice Cream Vans

Work out how to place ice-cream vans on the street intersections so that every other intersection is connected to one that has a van on it.
Display this to the class to show how the puzzle was constructed.
Variations and extensions

There are all sorts of situations in which one might be faced with this kind of problem in town planning: locating mailboxes, wells, fire-stations, and so on. And in real life, the map won’t be based on a trick that makes it easy to solve. If you really have to solve a problem like this, how would you do it?

There is a very straightforward way: consider all possible ways of placing ice-cream vans and check them to see which is best. With the 26 street corners in Tourist Town, there are 26 ways of placing one van. It’s easy to check all 26 possibilities, and it’s obvious that none of them satisfies the desired condition. With two vans, there are 26 places to put the first, and, whichever one is chosen for the first, there are 25 places left to put the second (obviously you wouldn’t put both vans at the same intersection): $26 \times 25 = 650$ possibilities to check. Again, each check is easy, although it would be very tedious to do them all. Actually, you only need to check half of them (325), since it doesn’t matter which van is which: if you’ve checked van 1 at intersection A and van 2 at intersection B then there’s no need to check van 1 at B and van 2 at A. You could carry on checking three vans (2600 possibilities), four vans (14950 possibilities), and so on. Clearly, 26 vans are enough since there are only 26 intersections and there’s no point in having more than one van at the same place. Another way of assessing the number of possibilities is to consider the total number of configurations with 26 intersections and any number of vans. Since there are two possibilities for each street corner—it may or may not have a van—the number of configurations is $2^{26}$, which is 67,108,864.

This way of solving the problem is called a “brute-force” algorithm, and it takes a long time. It’s a widely held misconception that computers are so fast they can solve just about any problem quickly, no matter how much work it involves. But that’s not true. Just how long the brute-force algorithm takes depends on how quick it is to check whether a particular configuration is a solution. To check this involves testing every intersection to find the distance of the nearest van. Suppose that an entire configuration can be tested in one second. How long does it take to test all $2^{26}$ possibilities for Tourist Town? (Answer: $2^{26}$ is about 67 million; there are 86,400 seconds in a day, so $2^{26}$ seconds is about 777 days, or around two years.) And suppose that instead of one second, it took just one thousandth of a second to check each particular configuration. Then the same two years would allow the computer to solve a 36-intersection town, because $2^{36}$ is about 1000 times $2^{26}$. Even if the computer was a million times faster, so that one million configurations could be checked every second, then it would take two years to work on a 46-intersection town. And these are not very big towns! (How many intersections are there in your town?)

Since the brute-force algorithm is so slow, are there other ways to solve the problem? Well, we could try the greedy approach that was so successful in the muddy city (Activity 9). We need to think how to be greedy with ice-creams—I mean how to apply the greedy approach to the ice-cream van problem. The way to do it is by placing the first van at the intersection that connects the greatest number of streets, the second one at the next most connected intersection, and so on. However, this doesn’t necessarily produce a minimum set of ice-cream van positions—in fact, the most highly connected intersection in Tourist Town, which has five streets, isn’t a good place to put a van (check this with the class).

Let’s look at an easier problem. Instead of being asked to find a minimum configuration, suppose you were given a configuration and asked whether it was minimal or not. In some cases, this is easy. For example, this diagram shows a much simpler map whose solution is quite straightforward. If
you imagine the streets as edges of a cube, it’s clear that two ice-cream vans at diagonally opposite cube vertices are sufficient. Moreover, you should be able to convince yourself that it is not possible to solve the problem with fewer than two vans. It is much harder—though not impossible—to convince oneself that Tourist Town cannot be serviced by less than six vans. For general maps it is extremely hard to prove that a certain configuration of ice-cream vans is a minimal one.

**What’s it all about?**

One of the interesting things about the ice-cream problem is that no-one knows whether there is an algorithm for finding a minimum set of locations that is significantly faster than the brute-force method! The time taken by the brute-force method grows exponentially with the number of intersections—it is called an *exponential-time* algorithm. A *polynomial-time* algorithm is one whose running time grows with the square, or the cube, or the seventeenth power, or any other power, of the number of intersections. A polynomial-time algorithm will always be faster for sufficiently large maps—even (say) a seventeenth-power algorithm—since an exponentially-growing function outweighs any polynomially-growing one once its argument becomes large enough. (For example, if you work it out, whenever $n$ is bigger than 117 then $n^{17}$ is smaller than $2^n$). Is there a polynomial-time algorithm for finding the minimum set of locations?—no-one knows, although people have tried very hard to find one. And the same is true for the seemingly easier task of checking whether a particular set of locations is minimal: the brute-force algorithm of trying all possibilities for smaller sets of locations is exponential in the number of intersections, and polynomial-time algorithms have neither been discovered nor proved not to exist.

Does this remind you of map coloring (Activity 13)? It should. The ice-cream van question, which is officially called the “minimum dominating set” problem, is one of a large number—thousands—of problems for which it is not known whether polynomial-time algorithms exist, in domains ranging from logic, through jigsaw-like arrangement problems to map coloring, finding optimal routes on maps, and scheduling processes. Astonishingly, all of these problems have been shown to be equivalent in the sense that if a polynomial-time algorithm is found for one of them, it can be converted into a polynomial-time algorithm for all the others—you might say that they stand or fall together.

These problems are called *NP-complete*. NP stands for “non-deterministic polynomial.” This jargon means that the problem could be solved in a reasonable amount of time if you had a computer that could try out an arbitrarily large number of solutions at once (that’s the non-deterministic part). You may think this is a pretty unrealistic assumption, and indeed it is. It’s not possible to build this kind of computer, since it would have to be arbitrarily large! However, the concept of such a machine is important in principle, because it appears that NP-complete problems cannot be solved in a reasonable amount of time without a non-deterministic computer.

Furthermore, this group of problems is called *complete* because although the problems seem very different—for example, map-coloring is very different from placing ice-cream vans—it turns out that if an efficient way of solving one of them is found, then that method can be adapted to solve *any* of the problems. That’s what we meant by “standing or falling together.”

There are thousands of NP-complete problems, and researchers have been attacking them, looking for efficient solutions, for several decades without success. If an efficient solution had been discovered for just one of them, then we would have efficient solutions...
for them all. For this reason, it is strongly suspected that there is no efficient solution. But proving that the problems necessarily take exponential time is the most outstanding open question in theoretical computer science—possibly in all of mathematics—today.

**Further reading**

Harel’s book *Algorithmics* introduces several NP-complete problems and discusses the question of whether polynomial-time algorithms exist. Dewdney’s *Turing Omnibus* also discusses NP-completeness. The standard computer science text on the subject is Garey & Johnson’s *Computers and Intractability*, which introduces several hundred NP-complete problems along with techniques for proving NP-completeness. However, it is fairly heavy going and is really only suitable for the computer science specialist.
Activity 15

Ice roads —*Steiner trees*

**Summary**
Sometimes a small, seemingly insignificant, variation in the specification of a problem makes a huge difference in how hard it is to solve. This activity, like the Muddy City problem (Activity 9), is about finding short paths through networks. The difference is that here we are allowed to introduce new points into the network if that reduces the path length. The result is a far more difficult problem that is not related to the Muddy City, but is algorithmically equivalent to the cartographer’s puzzle (Activity 13) and Tourist Town (Activity 14).

**Skills**
- Spatial visualization.
- Geometric reasoning.
- Algorithmic procedures and complexity.

**Ages**
- 7 and up

**Materials**
Each group of students will need

- five or six pegs to place in the ground (tent pegs are good, although a coat hanger cut into pieces which are then bent over is fine),
- several meters of string or elastic,
- a ruler or tape measure, and
- pen and paper to make notes on.
Ice Roads

Introduction
The previous activity, Tourist Town, took place in a very hot country; this one is just the opposite. In the frozen north of Canada (so the story goes), in the winter on the huge frozen lakes, snowplows make roads to connect up drill sites so that crews can visit each other. Out there in the cold they want to do a minimum of road building, and your job is to figure out where to make the roads. There are no constraints: highways can go anywhere on the snow—the lakes are frozen and covered. It’s all flat.

The roads should obviously travel in straight stretches, since introducing bends would only increase the length unnecessarily. But it’s not as simple as connecting all the sites with straight lines, because adding intersections out in the frozen wastes can sometimes reduce the total road length—and it’s total length that’s important, not travel time from one site to another.

In this figure, (a) shows three drill sites. Connecting one of them to each of the others (as in (b)) would make an acceptable road network. Another possibility is to make an intersection somewhere near the center of the triangle and connect it to the three sites (c). And if you measure the total amount of road that has been cleared, this is indeed a better solution. The extra intersection is called a “Steiner” point after the Swiss mathematician Jacob Steiner (1796–1863), who stated the problem and was the first to notice that the total length can be reduced by introducing new points. You could think of a Steiner point as a new, fictitious, drill site.

Discussion
1. Describe the problem that the students will be working on. Using the example above, demonstrate to the students that with three sites, adding a new one sometimes improves the solution by reducing the amount of road-building.
2. The students will be using four points arranged in a square, as illustrated in (a). Go outside and get each group to place four pegs in the grass in a square about 1 meter by 1 meter.

3. Get the students to experiment with connecting the pegs with string or elastic, measuring and recording the minimum total length required. At this stage they should not use any Steiner points. (The minimum is achieved by connecting three sides of the square, as in (b), and the total length is 3 meters.)

4. Now see if the students can do better by using one Steiner point. (The best place is in the center of the square, (c). Then the total length is $2\sqrt{2} = 2.83$ meters.) Suggest that they might do even better using two Steiner points. (Indeed they can, by placing the two points as in (d), forming 120 degree angles between the incoming roads. The total length is then $1 + \sqrt{3} = 2.73$ meters.)

5. Can the students do better with three Steiner points? (No – two points are best, and no advantage is gained by using more.)

6. Discuss with the students why these problems seem hard. (It’s because you don’t know where to put the Steiner points, and there are lots of possibilities to try out.)
Worksheet Activity: Steiner Tree Example 1

Find a way of linking these drill sites with the shortest possible ice roads.
Worksheet Activity: Steiner Tree Example 2

Find a way of linking these drill sites with the shortest possible ice roads.
Variations and extensions

(a) ![Diagram of a rectangle with Steiner points](image)
(b) ![Diagram of a rectangle with Steiner points](image)
(c) ![Diagram of a rectangle with Steiner points](image)
(d) ![Diagram of a rectangle with Steiner points](image)

7. An interesting experiment for groups that finish the original activity early is to work with a rectangle about 1 meter by 2 meters (a). The students will find that adding one Steiner point makes things worse, but two give an improved solution. (The lengths are 4 meters for (b), $2\sqrt{5} = 4.47$ meters for (c), and $2 + \sqrt{3} = 3.73$ meters for (d).) See if they can figure out why the one-point configuration does so much worse for rectangles than for squares. (It’s because when the square is stretched into a rectangle, the amount of stretch gets added just once into (b) and (d), but both diagonals increase in (c).)
8. Older students can work on a larger problem. Two layouts of sites to connect with ice roads are given in the worksheets. They can experiment with different solutions either using new copies of the worksheet, or by writing with removable pen on a transparency over the top of the sheet. Alternatively, the maps can be marked out on the ground using pegs. They can let the class know when they think they’ve set a new record for the shortest distance. (The figures on the right show the minimal solution for the first example and two possible solutions for the second, whose total length is quite similar.) The fact that there are two such similar solutions illustrates why these kinds of problem are so hard—there are so many choices about where to put the Steiner points!

9. Ladder networks like this provide another way to extend the problem.
Some minimal Steiner trees for ladder networks are shown here.

The one for a two-rung ladder is just the same as for a square. However, for a three-rung ladder the solution is quite different—as you will discover if you try to draw it out again from memory! The solution for four rungs is like that for two two-rung ladders joined together, whereas for five rungs it is more like an extension of the three-rung solution. In general, the shape of the minimal Steiner tree for a ladder depends on whether it has an even or odd number of rungs. If it is even, it is as though several two-rung ladders were joined together. Otherwise, it’s like a repetition of the three-rung solution. But proving these things rigorously is not easy.

10. Another interesting activity is to construct soap-bubble models of Steiner trees. You can do this by taking two sheets of rigid transparent plastic and inserting pins between them to represent the sites to be spanned, as shown here.

Now dip the whole thing into a soap solution. When it comes out, you will find that a film of soap connects the pins in a beautiful Steiner-tree network.
Unfortunately, however, it isn't necessarily a minimal Steiner tree. The soap film does find a configuration that minimizes the total length, but the minimum is only a local one, not necessarily a global one. There may be completely different ways of placing the Steiner points to give a smaller total length. For example, you can imagine the soap film looking like the first configuration in Extension 2 when it is withdrawn from the liquid on one occasion, and the second configuration on another.

**What's it all about?**

The networks that we've been working on are **minimal Steiner trees**. They’re called “trees” because they have no cycles, just as the branches on a real tree grow apart but do not (normally) rejoin and grow together again. They’re called “Steiner” trees because new points, Steiner points, can be added to the original sites that the trees connect. And they’re called “minimal” because they have the shortest length of any tree connecting those sites. In the Muddy City (Activity 9) we learned that a network connecting a number of sites that minimizes the total length is called a minimal spanning tree: Steiner trees are just the same except that new points can be introduced.

It’s interesting that while there is a very efficient algorithm for finding minimal spanning trees (Activity 14)—a greedy one that works by repeatedly connecting the two closest so-far-unconnected points—there is no general efficient solution to the minimal Steiner problem. Steiner trees are much harder because you have to decide where to put the extra points. In fact, rather surprisingly, the difficult part of the Steiner tree problem is not in determining the precise location of the Steiner points, but in deciding **roughly** where to put them: the difference between the two solutions to Example 2, for example. Once you know what regions to put the new points in, fine-tuning them to the optimum position is relatively simple. Soap films do that very effectively, and so can computers.

Finding minimal Steiner trees is part of a story that involved saving big money in the telephone business. Before 1967, when corporate customers in the US operated large private telephone networks, they leased the lines from a telephone company. The amount they are billed is not calculated on the basis of how the wires are actually used, but on the basis of the shortest network that would suffice. The reasoning is that the customer shouldn’t have to pay extra just because the telephone company uses a round-about route. Originally, the algorithm that calculated how much to charge worked by determining the minimal spanning tree. However, around 1967 it was noticed by a customer—an airline, in fact, with three major hubs—that if they requested a fourth hub at an intermediate point then the total length of the network would be reduced. The telephone company was forced to reduce charges to what they would have been if there was a telephone exchange at the Steiner point! Although, for typical configurations, the minimal Steiner tree is only 5% or 10% shorter than the minimal spanning tree, this can be a worthwhile saving when large amounts of money are involved. The Steiner tree problem is sometimes called the “shortest network problem” because it involves finding the shortest network that connects a set of sites.

If you have tackled the two preceding activities, the cartographer’s puzzle and tourist town, you will not be surprised to hear that the minimal Steiner tree problem is NP-complete. As the number of sites increases, so does the number of possible locations for Steiner points, and trying all possibilities involves an exponentially-growing search. This is another of the thousands of problems for which it simply isn’t known whether exponential search is the best that can be done, or whether there is an as-yet-undiscovered polynomial-time algorithm. What is known, however, is that if a polynomial-time algorithm is found for this problem, it can be turned into a polynomial-time algorithm for
graph coloring, for finding minimal dominating sets—and for all the other problems in the NP-complete class.

We explained at the end of the previous activity that the “NP” in NP-complete stands for “non-deterministic polynomial,” and “complete” refers to the fact that if a polynomial-time algorithm is found for one of the NP-complete problems it can be turned into polynomial-time algorithms for all the others. The set of problems that are solvable in polynomial time is called P. So the crucial question is, do polynomial-time algorithms exist for NP-complete problems—in other words, is \( P = NP? \) The answer to this question is not known, and it is one of the great mysteries of modern computer science.

Problems for which polynomial-time algorithms exist—even though these algorithms might be quite slow—are called “tractable.” Problems for which they do not are called “intractable,” because no matter how fast your computer, or how many computers you use together, a small increase in problem size will mean that they can’t possibly be solved in practice. It is not known whether the NP-complete problems—which include the cartographer’s puzzle, tourist town, and ice roads—are tractable or not. But most computer scientists are pessimistic that a polynomial-time algorithm for NP-complete problems will ever be found, and so proving that a problem is NP-complete is regarded as strong evidence that the problem is inherently intractable.

"I can’t find an efficient algorithm, I guess I’m just too dumb."  
"I can’t find an efficient algorithm, because no such algorithm is possible."  
"I can’t find an efficient algorithm, but neither can all these famous people."

What to do when you can’t find an efficient algorithm: three possibilities

What can you do when your boss asks you to devise an efficient algorithm that comes up with the optimal solution to a problem, and you can’t find one?—as surely happened when the airline hit upon the fact that network costs could be reduced by introducing Steiner points. If you could prove that there isn’t an efficient algorithm to come up with the optimal solution, that would be great. But it’s very difficult to prove negative results like this in computer science, for who knows what clever programmer might come along in the future and hit upon an obscure trick that solves the problem. So, unfortunately, you’re unlikely to be in a position to say categorically that no efficient algorithm is possible—that the problem is intractable. But if you can show that your problem is NP-complete, then it’s actually true that thousands of people in research laboratories have worked on problems that really are equivalent to yours, and also failed to come up with an efficient solution. That may not get you a bonus, but it’ll get you off the hook!
Of course, in real life these problems still need to be solved, and in that case people turn to heuristics – algorithms that don’t guarantee to give the best possible solution, but do give a solution within a very small percentage of the optimal. Heuristic algorithms can be very fast, and the wastage of not finding the best possible solution can be fairly small, so they are good enough to get on with the job. It’s just frustrating to know that there might be a slightly better timetable, or a slightly better layout of a network or roads.

Further reading

The cartoon is from Garey and Johnson’s classic book *Computers and Intractability*.

The “Computer recreations” column of *Scientific American*, June 1984, contains a brief description of how to make Steiner trees using soap bubbles, along with interesting descriptions of other analog gadgets for problem solving, including a spaghetti computer for sorting, a cat's cradle of strings for finding shortest paths in a graph, and a light-and-mirrors device for telling whether or not a number is prime. These also appear in a section about analog computers in Dewdney’s *Turing Omnibus*. 
Part V
Sharing secrets and fighting crime-
Cryptography
Sharing Secrets and Fighting Crime

You’ve heard of spies and secret agents using hidden codes or magic invisible writing to exchange messages. Well, that’s how the subject of “cryptography” started out, as the art of writing and deciphering secret codes. During the Second World War, the English built special-purpose electronic code-breaking machines and used them to crack military codes. And then computers came along and changed everything, and cryptography entered a new era. Massive amounts of computation, that would have been quite unimaginable before, could be deployed to help break codes. When people began to share computer systems with each other, there were new uses for secret passwords. When computers were linked up in networks, there were new reasons to protect information from people who would have liked to have got hold of it. When electronic mail arrived, it became important to make sure that people who sign messages are really who they say they are. Now that people can buy and sell goods using computers, we need secure ways of placing orders and sending cash on computer networks. And the growing threat of a terrorist attacking a computer system makes computer security ever more important.

Cryptography probably makes you think of computers storing secret passwords, and jumbling up the letters of messages so that the enemy can't read them. But the reality is very different. Modern computer systems don’t store secret passwords, because if they did, anyone who managed to get access to them would be able to break through all the security in the system. That would be disastrous: they could make phoney bank transfers, send messages pretending to be someone else, read everyone’s secret files, command armies, bring down governments. Nowadays, passwords are handled using the “one-way functions” that we talked about in Activity 14. And encryption is not just jumbling up the letters of messages; it’s done using techniques involving really hard problems—like the “intractable” ones introduced in Part IV.

Using cryptography, you can do things that you might think are impossible. In this section you will discover a simple way to calculate the average age of the people in a group without anyone having to let anyone else know what their age is. You will find out how two people who don’t trust each other can toss a coin and agree on the outcome even though they are in different cities and can’t both see the coin being tossed. And you will find a way to encode secret messages that can only be decoded by one person, even though everyone knows how to encode them.

For teachers

The activities that follow provide hands-on experience with modern cryptographic techniques—which are very different from what most people conjure up when they think of secrecy and computers.

There are two key ideas. The first is the notion of a “protocol,” which is a formal statement of a transaction. Protocols may bring to mind diplomats, even etiquette, but computers use them too! Seemingly difficult tasks can be accomplished by surprising simple protocols. Activity 16, which only takes a few minutes, shows how a group of people, cooperating together, can easily calculate their average age (or income), without anyone finding out any individual’s age (or income). The second key idea is the role that computational complexity—intractability—can play when interacting with others through computers. Activity 17 shows how two people who don’t necessarily trust each other can
agree on the outcome of a coin toss when they are connected only by telephone. (This activity also introduces, as an aside, the idea of Boolean logic circuits and how to work with them.) Activity 18 shows how people can use computational techniques to encrypt messages securely, even though the method for performing the encoding is public knowledge.

Some of these activities—particularly the last one—are hard work. You will have to motivate your class by instilling into the students a sense of wonder that such things can be done at all, for the activities really do accomplish things that most people would think were impossible. It is vital to create this sense of wonder, communicate it, and pause frequently to keep it alive throughout the activity so that students do not miss the (amazing!) forest for the (perhaps rather tiresome) trees. These activities are among the most challenging and technically intricate in the book. If they turn out to be too difficult, please skip to Part VI, which has a completely different, non-technical, character.

For the technically-minded
As computers encroach upon our daily lives, the application of cryptography is potentially rather tendentious. Most people simply don’t realize what modern cryptographic protocols are capable of. The result is that when large institutions—both governmental and commercial—set up systems that involve personal information, it tends to be technocrats who make the key decisions on how things are to be handled, what is to be collected, what is to be made available, and to whom. If people had a better understanding of the possibilities opened up by modern technology, they would be able to participate more actively in such decisions, and society might end up with a different information infrastructure.

This material on information-hiding protocols, cryptographic protocols, and public-key encryption is generally considered to be pretty advanced. But the ideas themselves are not difficult. It’s the technicalities, not the underlying concepts that are hard to understand. In practical situations involving electronic commerce, the technicalities are buried inside computer software, which renders the new technologies of encryption very easy to use. But it’s also important to understand the ideas on which they are based, in order to gain insight into what can be done.

Cryptographic systems are of great interest to governments, not just because they want to keep official communications secure, but because of concerns that encrypted communication could be used by people involved in illegal activities such as drug trafficking and terrorism. If such people use encryption then wire-tapping becomes useless unless a decryption method is available. These concerns have created a lot of debate between people concerned with law enforcement, who want to limit the strength of cryptographic systems, and civil libertarians, who are uncomfortable with the government having access to the private communications. For a while the US government has restricted the use of some cryptographic methods by deeming them to be munitions—like bombs and guns, anyone can set up a secure communication system given the right information and some technical ability, but they are dangerous in the wrong hands. At one stage there was extensive debate over the “Clipper Chip,” a system that has an extra password called a key escrow, which is held by a government agency that allows it to decode any message encrypted by the chip. The FBI and US Justice department wanted this chip to be widely used for communications, but this has drawn considerable opposition because of threats to privacy. All sorts of cryptographic systems are technically feasible, but they aren’t necessarily politically acceptable!
Cryptographic ideas have many applications other than keeping messages secret. Like verifying that messages really were sent by the people who said they sent them—this is “authentication,” and without it electronic commerce is impossible. There are ways to let people vote by computer without anyone else being able to find out who they voted for—even those who run the computer system—yet still prevent people from voting more than once. And you can even play cards over the phone—which may sound silly until you realize that making business deals is a lot like playing poker.

These things sound impossible. How could you even begin to shuffle a deck of cards over the phone if you’re in competition with the person at the other end and so can’t trust them? How could you possibly detect that someone has intercepted a message, modified it, and then passed it off as the original? Yet if you can’t do those things, you can’t conduct business electronically. You have to prevent technically-minded criminals from forging authorizations for withdrawals from bank accounts by intercepting the phone line between a point-of-sale terminal and the bank. You have to prevent business competitors from wreaking havoc by generating false orders or false contracts. With modern cryptographic techniques such miracles can be done, and these activities show how.

There are many interesting books about codes and code-breaking. *Codebreakers: the inside story of Bletchley Park* edited by Hinsley and Stripp, gives first-hand accounts of how some of the first computers were used to break codes during the Second World War, significantly shortening the war and saving many lives.
Activity 16

Sharing secrets—*Information hiding protocols*

**Summary**
Cryptographic techniques enable us to share information with other people, yet still maintain a surprisingly high level of privacy. This activity illustrates a situation where information is shared, and yet none of it is revealed: a group of students will calculate their average age without anyone having to reveal to anyone else what their age is.

**Skills**
- Calculating an average.
- Random numbers.
- Cooperative tasks.

**Ages**
- 7 years and up

**Materials**
Each group of students will need:

- a small pad of paper, and
- a pen.
Sharing Secrets

Introduction
This activity involves finding the average age of a group of students, without anyone having to reveal what their age is. Alternatively, one could work out the average income (allowance) of the students in the group, or some similar personal detail. Calculating these statistics works particularly well with adults, because older people can be more sensitive about details like age and income.

You will need at least three students in the group.

Discussion
1. Explain to the group that you would like to work out their average age, without anyone telling anyone else what their age is. Ask for suggestions about how this might be done, or even whether they believe it can be done.

2. Select about six to ten students to work with. Give the pad and pen to the first student, and ask them to secretly write down a randomly chosen three-digit number on the top sheet of paper. In this example, 613 has been chosen as the random number.

3. Have the first student tear off the first page, add their age to the random number, and write it on the second sheet on the pad. The first student’s age is 8, so the second sheet shows 621. They should keep the page that was torn off (and not show it to anyone.)

4. The pad is then passed to the second student, who adds their age to the number on the top, tears off the page, and writes the total on the next page. In the example, the second student is 10 years old.

5. Continue this process having a student tear off the top page and add their age to the number on it, until all the students have had the pad.
6. Return the pad to the first student. Have that student subtract their original random number from the number on the pad. In the example, the pad has been around five students, and the final number, 657, has the original number, 613, subtracted from it, giving the number 44. This number is the sum of the students' ages, and the average can be calculated by dividing by the number of students; thus the average age of our example group is 8.8 years old.

7. Point out to the students that so long as everyone destroys their piece of paper, no-one can work out an individual’s age unless two people decide to cooperate.

**Variations and extensions**

This system could be adapted to allow secret voting by having each person add one if they are voting yes, and zero if they are voting no. Of course, if someone adds more than one (or less than zero) then the voting would be unfair, although they would be running the risk of arousing suspicion if everyone voted yes, since the number of yes votes would be more than the number of people.

**What's it all about?**

Computers store a lot of personal information about us: our bank balance, our social networks, how much tax we owe, how long we have held a driver’s license, our credit history, examination results, medical records, and so on. Privacy is very important! But we do need to be able to share some of this information with other people. For example, when paying for goods at a store using a bank card, we recognize that the store needs to verify that we have the funds available.

Often we end up providing more information than is really necessary. For example, if we perform an electronic transaction at a store, they essentially discover who we bank with, what our account number is, and what our name is. Furthermore, the bank finds out where we have done our shopping. Banks could create a profile of someone by monitoring things like where they buy gas or groceries, how much they spend on these items each day, and when these places are visited. If we had paid by cash then none of this information would have been revealed. Most people wouldn’t worry too much about this information being shared, but there is the potential for it to be abused, whether for targeted marketing (for example, sending travel advertisements to people who spend a lot on air tickets), discrimination (such as offering better service to someone whose bank usually only takes on wealthy clients), or even blackmail (such as threatening to reveal the details of an embarrassing transaction). If nothing else, people might change the way they shop if they think that someone might be monitoring them.

This loss of privacy is fairly widely accepted, yet cryptographic protocols exist that allow us to make electronic financial transactions with the same level of privacy as we would get with cash. It might be hard to believe that money can be transferred from your bank account to a store’s account without anyone knowing where the money was coming from or going to. This activity makes such a transaction seem a little more plausible: both situations involve limited sharing of information, and this can be made possible by a clever protocol.

**Further reading**

A classic paper that highlights these issues was written by David Chaum, with the provocative title “Security without identification: transaction systems to make Big Brother obsolete.” The paper is quite readable, and gives simple examples of information
hiding protocols, including how completely private transactions can be made using “electronic cash.” It can be found in *Communications of the ACM*, October 1985.
Activity 17

The Peruvian coin flip—Cryptographic protocols

Summary
This activity shows how to accomplish a simple, but nevertheless seemingly impossible task—making a fair random choice by flipping a coin, between two people who don’t necessarily trust each other, and are connected only by a telephone.

Curriculum links
✓ Technology Level 1: Technological systems. Understand that technological systems have inputs, controlled transformations, and outputs.

Skills
✓ Boolean logic
✓ Functions
✓ Puzzle solving

Ages
✓ 9 and up

Materials
Each group of students will need:
✓ a copy of the reproducible sheet The Peruvian Coin Flip
✓ about two dozen small buttons or counters of two different colors
The Peruvian Coin Flip

Introduction

This activity was originally devised when one of the authors (MRF) was working with students in Peru, hence the name. You can customize the story to suit local conditions.

The soccer teams of Lima and Cuzco have to decide who gets to be the home team for the championship game. The simplest way would be to flip a coin. But the cities are far apart, and Alicia, representing Lima, and Benito, representing Cuzco, cannot spend the time and money to get together to flip a coin. Can they do it over the telephone? Alicia could flip and Benito could call heads or tails. But this won’t work because if Benito called heads, Alicia can simply say “sorry, it was tails” and Benito would be none the wiser. Alicia is not naturally deceitful but this, after all, is an important contest and the temptation is awfully strong. Even if Alicia were truthful, would Benito believe that if he lost?

This is what they decide to do. Working together, they design a circuit made up of and-gates and or-gates, as explained below. In principle they can do this over the phone, although admittedly in practice it could turn out to be more than a little tedious (email would work too!) During the construction process, each has an interest in ensuring that the circuit is complex enough that the other will be unable to cheat. The final circuit is public knowledge.

Children will get more out of this activity if they have learned binary number representation (see Activity 1, Count the dots), the concept of parity (see Activity 4, Card flip magic), and have seen the example of one-way functions in Activity14, Tourist Town.
Discussion

The rules of and-gates and or-gates are simple. Each “gate” has two inputs and one output. Each of the inputs can be either a 0 or a 1, which can be interpreted as false and true, respectively. The output of an and-gate is one (true) only if both inputs are one (true), and zero (false) otherwise. For example, the and-gate in has a one and a zero on its inputs (at the top), so the output (the square at the bottom) is a zero. The output of an or-gate is one (true) if either (or both) of the inputs is one (true), and zero (false) only if both the inputs are zero. Thus the output of the or-gate is a one for the inputs zero and one.

The output of one gate can be connected to the input of another (or several others) to produce a more complicated effect. For example, in the left-hand circuit the outputs from two or-gates are connected to the inputs of a third or-gate, with the effect that if any of the four inputs is a one then the output will be a one. In the right-hand circuit the outputs of each of the top two and-gates feeds into the lower two gates, so the whole circuit has two values in its output.

For the Peruvian coin flip we need even more complex circuits. The circuit on the worksheet has six inputs and six outputs. Here is a worked example for one particular set of input values.

The way that this circuit can be used to flip a coin by telephone is as follows. Alicia selects a random input to the circuit, consisting of six binary digits (zeros or ones), which she keeps secret. She puts the six digits through the circuit and sends Benito the six bits of output. Once Benito has the output, he must try to guess whether Alicia’s input has an
even or an odd number of ones—in other words, she must guess the *parity* of Alicia’s input. If the circuit is complex enough then Benito won’t be able to work out the answer, and his guess will have to be a random choice (in fact, he could even toss a coin to choose!) Benito wins—and the playoff is in Cuzco—if his guess is correct; Alicia wins—and the playoff is in Lima—if Benito guesses incorrectly. Once Benito has told Alicia his guess, Alicia reveals her secret input so that Benito can confirm that it produces the claimed output.

1. Divide the students into small groups, give each group the circuit and some counters, and explain the story. The situation will probably be more meaningful to the students if they imagine one of their sports captains organizing the toss with a rival school. Establish a convention for the counter colors—red is 0, blue is 1, or some such—and have the students mark it on the leg end at the top of the sheet to help them remember.

2. Show the students how to place counters on the inputs to show the digits that Alicia chooses. Then explain the rules of *and*-gates and *or*-gates, which are summarized at the bottom of the sheet (consider getting the students to color these in).

3. Show how to work through the circuit, placing counters at the nodes, to derive the corresponding output. This must be done accurately and takes some care; The table (which should *not* be given to the students) shows the output for each possible input for your own reference in case of any doubt.

<table>
<thead>
<tr>
<th>Input</th>
<th>000000</th>
<th>000001</th>
<th>000010</th>
<th>000011</th>
<th>000100</th>
<th>000101</th>
<th>000110</th>
<th>000111</th>
<th>001000</th>
<th>001001</th>
<th>001010</th>
<th>001011</th>
<th>001100</th>
<th>001101</th>
<th>001110</th>
<th>001111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>000000</td>
<td>010010</td>
<td>000000</td>
<td>010010</td>
<td>000100</td>
<td>000110</td>
<td>001010</td>
<td>001000</td>
<td>010010</td>
<td>010000</td>
<td>010110</td>
<td>010100</td>
<td>010110</td>
<td>011000</td>
<td>011010</td>
<td>011011</td>
</tr>
<tr>
<td>Input</td>
<td>001000</td>
<td>010100</td>
<td>000000</td>
<td>010100</td>
<td>001010</td>
<td>010110</td>
<td>011010</td>
<td>011000</td>
<td>011100</td>
<td>011101</td>
<td>011110</td>
<td>011111</td>
<td>001110</td>
<td>001111</td>
<td>001100</td>
<td>001101</td>
</tr>
<tr>
<td>Output</td>
<td>001000</td>
<td>010100</td>
<td>000000</td>
<td>010100</td>
<td>001010</td>
<td>010110</td>
<td>011010</td>
<td>011000</td>
<td>011100</td>
<td>011101</td>
<td>011110</td>
<td>011111</td>
<td>001110</td>
<td>001111</td>
<td>001100</td>
<td>001101</td>
</tr>
<tr>
<td>Input</td>
<td>010000</td>
<td>011000</td>
<td>010100</td>
<td>011100</td>
<td>011010</td>
<td>011110</td>
<td>011000</td>
<td>011100</td>
<td>011101</td>
<td>011110</td>
<td>011111</td>
<td>011000</td>
<td>011001</td>
<td>011010</td>
<td>011011</td>
<td>011100</td>
</tr>
<tr>
<td>Output</td>
<td>010000</td>
<td>011000</td>
<td>010100</td>
<td>011100</td>
<td>011010</td>
<td>011110</td>
<td>011000</td>
<td>011100</td>
<td>011101</td>
<td>011110</td>
<td>011111</td>
<td>011000</td>
<td>011001</td>
<td>011010</td>
<td>011011</td>
<td>011100</td>
</tr>
<tr>
<td>Input</td>
<td>100000</td>
<td>100001</td>
<td>100010</td>
<td>100011</td>
<td>100100</td>
<td>100101</td>
<td>100110</td>
<td>100111</td>
<td>101000</td>
<td>101001</td>
<td>101010</td>
<td>101011</td>
<td>101100</td>
<td>101101</td>
<td>101110</td>
<td>101111</td>
</tr>
<tr>
<td>Output</td>
<td>100000</td>
<td>010100</td>
<td>011000</td>
<td>011100</td>
<td>011010</td>
<td>011110</td>
<td>011000</td>
<td>011100</td>
<td>011101</td>
<td>011110</td>
<td>011111</td>
<td>011000</td>
<td>011001</td>
<td>011010</td>
<td>011011</td>
<td>011100</td>
</tr>
<tr>
<td>Input</td>
<td>101000</td>
<td>101001</td>
<td>101010</td>
<td>101011</td>
<td>101100</td>
<td>101101</td>
<td>101110</td>
<td>101111</td>
<td>110000</td>
<td>110001</td>
<td>110010</td>
<td>110011</td>
<td>110100</td>
<td>110101</td>
<td>110110</td>
<td>110111</td>
</tr>
<tr>
<td>Output</td>
<td>101000</td>
<td>011010</td>
<td>011010</td>
<td>011101</td>
<td>011100</td>
<td>011110</td>
<td>011000</td>
<td>011100</td>
<td>011101</td>
<td>011110</td>
<td>011111</td>
<td>011000</td>
<td>011001</td>
<td>011010</td>
<td>011011</td>
<td>011100</td>
</tr>
<tr>
<td>Input</td>
<td>110000</td>
<td>110001</td>
<td>110010</td>
<td>110011</td>
<td>110100</td>
<td>110101</td>
<td>110110</td>
<td>110111</td>
<td>111000</td>
<td>111001</td>
<td>111010</td>
<td>111011</td>
<td>111100</td>
<td>111101</td>
<td>111110</td>
<td>111111</td>
</tr>
<tr>
<td>Output</td>
<td>110000</td>
<td>011100</td>
<td>011100</td>
<td>011101</td>
<td>011100</td>
<td>011110</td>
<td>011000</td>
<td>011100</td>
<td>011101</td>
<td>011110</td>
<td>011111</td>
<td>011000</td>
<td>011001</td>
<td>011010</td>
<td>011011</td>
<td>011100</td>
</tr>
</tbody>
</table>

4. Now each group should elect an Alicia and a Benito. The group can split in half and each half side with Alicia or Benito respectively. Alicia should choose a random input for the circuit, calculate the output, and tell it to Benito. Benito guesses the parity of the input (whether it has an odd or even number of ones in it). It should become evident during this process that Benito’s guess is essentially random. Alicia
then tells everyone what the input was, and Benito wins if she guessed the correct parity. Benito can verify that Alicia’s didn’t change her chosen input by checking that it gives the correct output from the circuit.

At this point the coin toss has been completed.

Benito can cheat if, given an output, he can find the input that produced it. Thus it is in Alicia’s interests to ensure that the function of the circuit is one-way, in the sense discussed in Activity 14, to prevent Benito cheating. A one-way function is one for which the output is easy to calculate if you know what the input is, but the input is very difficult to calculate for a given output.

Alicia can cheat if she can find two inputs of opposite parity that produce the same output. Then, whichever way Benito guesses, Alicia can reveal the input that shows him to be wrong. Thus it is in Benito’s interests to ensure that the circuit does not map many different inputs to the same output.

5. See if the students can find a way for Alicia or Benito to cheat. From the first line of the table you can see that several different inputs generate the output 010010—for example, 000001, 000011, 000101, etc. Thus if Alicia declares the output 010010, she can choose input 000001 if Benito guesses that the parity is even, and 000011 if he guesses that it is odd.

With this circuit, it is hard for Benito to cheat. But if the output happens to be 011000, then the input must have been 100010—there is no other possibility (you can see this by checking right through the table). Thus if this is the number that Alicia happens to come up with, Benito can guess even parity and be sure of being correct. A computer-based system would use many more bits, so there would be too many possibilities to try (each extra bit doubles the number of possibilities).

6. Now ask the groups of students to devise their own circuits for this game. See if they can find a circuit that makes it easy for Alicia to cheat, and another that makes it easy for Benito to cheat. There is no reason why the circuit has to have six inputs, and it may even have different numbers of inputs and outputs.
Worksheet Activity: The Peruvian Con Flip

Choose some inputs for this circuit and work out what the outputs are.
Variations and extensions

1. An obvious problem in practice is the cooperation that is needed to construct a circuit acceptable to both Alicia and Benito. This might make the activity fun for the kids, but is likely to render the procedure inoperable in practice—particularly over the phone! However, there is a simple alternative in which Alicia and Benito construct their circuits independently and make them publicly available. Then Alicia puts her secret input through both circuits, and joins the two outputs together by comparing corresponding bits and making the final output a one if they are equal and zero otherwise. In this situation, neither participant can cheat if the other doesn’t, for if just one of the circuits is a one-way function then the combination of them both is also a one-way function.

The next two variations relate not to cryptographic protocols or the coin-tossing problem per se, but rather to the idea of circuits constructed out of and or or gates. They explore some important notions in the fundamentals not only of computer circuits, but of logic itself. This kind of logic is called Boolean algebra named after the mathematician George Boole (1815-64).

2. The students may have noticed that the all-zero input, 000000, is bound to produce the all-zero output, and likewise the all-one input 111111 is bound to produce the all-one output. (There may be other inputs that produce these outputs as well; indeed, there are for the example circuit—000010 produces all zeros, while 110111 produces all ones.) This is a consequence of the fact that the circuits are made up of and or or gates. By adding a not-gate, which takes just one input and produces the reverse as output (i.e. 0 → 1 and 1 → 0), the students can construct circuits that don’t have this property.

3. Two other important kinds of gate are and-not and or-not (usually abbreviated to nand and nor respectively), which are like and or or but followed by a not. Thus a and-not b is not (a and b). These do not allow any functionally different circuits to be achieved, since their effect can always be obtained with the corresponding and or or gate, followed by not. However, they have the interesting property that all other gate types can be made out of and-not gates, and also out of or-not gates.

Having introduced and-not and or-not, challenge the students to discover whether any of the gates can be made from other gates connected together, and further, if they can be made from just one type of gate connected together. The figure below shows how the three basic gates, not, and, and or, can be constructed from and-not gates, in the top row, and or-not gates, in the bottom row.
What’s it all about?

Recent years have seen huge increases in the amount of commerce being conducted over computer networks, and it is essential to guarantee secure interchange of electronic funds, confidential transactions, and signed, legally binding, documents. The subject of *cryptography* is about communicating in secure and private ways. Several decades ago, computer science researchers discovered the counter-intuitive result that secrecy can be guaranteed by techniques that ensure that certain information is kept *public*. The result is the so-called “public key cryptosystem” of Activity 18, Kid Krypto, which is now widely used as the main secure way of exchanging information. For example, you may have seen settings such as SSL (Secure Sockets Layer) or TLS (Transport Layer Security) in your web browser; these systems are based on public key systems that enable your web browser to set up a secure connection to a website such as a bank, even if someone is eavesdropping on all the data being sent.
Cryptography is not just about keeping things secret, but about placing controls on information that limit what others can find out, and about establishing trust between people who are geographically separated. Formal rules or “protocols” for cryptographic transactions have been devised to allow such seemingly impossible things as unforgeable digital signatures and the ability to tell others that you possess a secret (like a password) without actually revealing what it is. Flipping a coin over the telephone is a simpler but analogous problem, which also seems, on the face of it, to be impossible.

In a real situation, Alicia and Benito would not design a circuit themselves, but acquire a computer program that does the work internally. Probably neither would be interested in the innards of the software. But both would want to rest assured that the other is unable to influence the outcome of the decision, no matter how good their computer skills and how hard they tried.

In principle, any disputes would have to be resolved by appeal to a neutral judge. The judge would be given the circuit, Alicia’s original binary number, the output that she originally sent Benito, and the guess that Benito sent in return. Once the interchange is over, all this is public information, so both participants will have to agree that this is what the outcome was based on. The judge will be able to put Alicia’s original number through the circuit and check that the output is as claimed, and therefore decide whether the decision has been made fairly. Needless to say, the very fact that there is a clear procedure to check that the rules have been followed makes it unlikely that a dispute will arise. Compare with the situation where Alicia flips an actual coin and Benito calls heads or tails—no judge would take on that case!

A circuit as small as the one illustrated would not be much use in practice, for it is easy to come up with a table and use it to cheat. Using thirty-two binary digits in the input would provide better protection. However, even this does not guarantee that it is hard to cheat—that depends on the particular circuit. Other methods could be used, such as the one-way function introduced in Activity 14, Tourist Town. Methods used in practice often depend on the factoring of large numbers, which is known to be a hard problem (although, as we will learn at the end of the next activity, it is not NP-complete). It is easy to check that one number is a factor of another, but finding the factors of a large number is very time consuming. This makes it more complex for Alicia and Benito (and the judge) to work through by hand, although, as noted above, in practice this will be done by off-the-shelf software.

Digital signatures are based on a similar idea. By making public the output of the circuit for the particular secret input that she has chosen, Alicia is effectively able to prove that she is the one who generated the output—for, with a proper one-way function, no-one else can come up with an input that works. No-one can masquerade as Alicia! To make an actual digital signature, a more complex protocol is needed to ensure that Alicia can sign a particular message, and also to ensure that others can check that Alicia was the signatory even if she claims not to be. But the principle is the same.

Another application is playing poker over the phone, in an environment in which there is no referee to deal the cards and record both player’s hands. Everything must be carried out by the players themselves, with recourse to a judge at the end of the game in the event of a dispute. Similar situations arise in earnest with contract
negotiations. Obviously, players must keep their cards secret during the game. But they
must be kept honest—they must not be allowed to claim to have an ace unless they
actually have one! This can be checked by waiting until the game is over, and then
allowing each player to inspect the other’s original hand and sequence of moves. Another
problem is how to deal the cards while keeping each player’s hand secret until after the
game. Surprisingly, it is possible to accomplish this using a cryptographic protocol not
dissimilar to the coin-tossing one.

Cryptographic protocols are extremely important in electronic transactions, whether to
identify the owner of an debit card, to authorize the use of a cellphone for a call, or to
authenticate the sender of an email. The ability to do these things reliably is crucial to the
success of electronic commerce.

Further reading
Harel’s book *Algorithmics* discusses digital signatures and associated cryptographic
protocols. It also shows how to play poker over the phone, an idea that was first raised in
1981 in a chapter called “Mental poker”, in the book *The Mathematical Gardener*, edited
by D.A. Klarner. *Cryptography and data security* by Dorothy Denning is an excellent
computer science text on cryptography. Dewdney’s *Turing Omnibus* has a section on
Boolean logic that discusses the building blocks used for the circuits in this activity.
Activity 18

Kid Krypto—Public-key encryption

Summary
Encryption is the key to information security. And the key to modern encryption is that using only public information, a sender can lock up their message in such a way that it can only be unlocked (privately, of course) by the intended recipient.

It is as though everyone buys a padlock, writes their name on it, and puts them all on the same table for others to use. They keep the key of course—the padlocks are the kind where you just click them shut. If I want to send you a secure message, I put it in a box, pick up your padlock, lock the box and send it to you. Even if it falls into the wrong hands, no-one else can unlock it. With this scheme there is no need for any prior communication to arrange secret codes.

This activity shows how this can be done digitally. And in the digital world, instead of picking up your padlock and using it, I copy it and use the copy, leaving the original lock on the table. If I were to make a copy of a physical padlock, I could only do so by taking it apart. In doing so I would inevitably see how it worked. But in the digital world we can arrange for people to copy locks without being able to discover the key!

Sounds impossible? Read on.

Curriculum Links
✓ Technology Level 3: Technological systems. Understand that technological systems are represented by symbolic language tools and understand the role played by the black box in technological systems.

Skills
✓ Puzzle solving.
✓ Secret codes.

Ages
✓ 11 years and up

Materials
The students are divided into groups of about four, and within these groups they form two subgroups. Each subgroup is given a copy of the two maps on the worksheet Kid Krypto Maps. Thus for each group of students you will need:

✓ two copies of the Kid Krypto Maps.

You will also need:

✓ an overhead projector transparency of Kid Krypto Encoding, and
✓ a way to annotate the diagram.
Kid Krypto

Introduction

This is the most technically challenging activity in this book. While rewarding, it requires careful work and sustained concentration to complete successfully. Students should already have studied the example of one-way functions in Activity 14, Tourist Town, and it is helpful if they have completed the other activities in this section (Activity 16, Sharing Secrets, and Activity 17, the Peruvian coin flip). The activity also uses ideas covered in Activity 1, Count the dots, and Activity 5, Twenty guesses.

Amy is planning to send Bill a secret message. Normally we might think of secret messages as a sentence or paragraph, but in the following exercise Amy will send just one character — in fact, she will send one number that represents a character. Although this might seem like a simplistic message, bear in mind that she could send a whole string of such “messages” to make up a sentence, and in practice the work would be done by a computer. And sometimes even small messages are important —one of the most celebrated messages in history, carried by Paul Revere, had only two possible values. We will see how to embed Amy’s number in an encrypted message using Bill’s public lock so that if anyone intercepts it, they will not be able to decode it. Only Bill can do that, because only he has the key to the lock.

We will lock up messages using maps. Not Treasure Island maps, where X marks the spot, but street maps like the ones from Tourist Town (Activity 14), where the lines are streets and the dots are street corners. Each map has a public version—the lock—and a private version—the key.
**Discussion**

Shown on the worksheet *Kid Krypto Encoding* is Bill’s public map. It's not secret: Bill puts it on the table (or a web page) for everyone to see, or (equivalently) gives it to anyone who might want to send him a message. Amy has a copy; so has everyone else. The figure to the right shows Bill’s private map. It’s the same as his public map, except that some of the street corners are marked as special by enlarging them. He keeps this version of the map secret.

This activity is best done as a class, at least to begin with, because it involves a fair amount of work. Although not difficult, this must be done accurately, for errors will cause a lot of trouble. It is important that the students realize how surprising it is that this kind of encryption can be done at all—it seems impossible (doesn’t it?)—because they will need this motivation to see them through the effort required. One point that we have found highly motivating for school students is that using this method they can pass secret notes in class, and even if their teacher knows how the note was encrypted, the teacher won’t be able to decode it.

1. Display Bill's public map (*Kid Krypto Encoding* worksheet). Decide which number Amy is going to send. Now place random numbers on each intersection on the map, so that the random numbers add up to the number that Amy wishes to send. This figure gives an example of such numbers as the upper (non-parenthesised) number beside each intersection. Here, Amy has chosen to send the number 66, so all the unbracketed numbers add up to 66. If necessary, you can use negative numbers to get the total down to the desired value.

2. Now Amy must calculate what to send to Bill. If she sent the map with the numbers on, that would be no good, because if it fell into the wrong hands anybody could add them up and get the message.

Instead, choose any intersection, look at it and its three neighbors—four intersections in all—and total the numbers on them. Write this number at the intersection in parentheses or using a different color pen. For example, the rightmost intersection in
the example public map is connected to three others, labeled 1, 4, 11, and is itself labeled 6. Thus it has a total of 22. Now repeat this for all the other intersections in the map. This should give you the numbers in parentheses.

3. Amy will send to Bill his map, with only the parenthesised numbers on it.

Erase the original numbers and the counts, leaving only the numbers that Amy sends; or write out a new map with just those numbers on it. See if any of the students can find a way to tell from this what the original message was. They won’t be able to.

4. Only someone with Bill’s private key can decode the message to find the message that Amy originally wanted to send. On the coded message, mark the special enlarged nodes in Bill’s private map.

To decode the message, Bill looks at just the secret marked intersections and adds up the numbers on them. In the example, these intersections are labeled 13, 13, 22, 18, which add up to 66, Amy’s original message.

5. How does it work? Well, the map is a special one. Suppose Bill were to choose one of the marked intersections and draw around the intersections one street distant from it, and repeat the procedure for each marked intersection. This would partition the map into non-overlapping pieces, as illustrated here. Show these pieces to the students by drawing the boundaries on the map. The group of intersections in each partition is exactly the ones summed to give the transmitted numbers for the marked intersections, so the sum of the four transmitted numbers on those intersections will be the sum of all the original numbers in the original map; that it, it will be the original message!

Phew! It seems a lot of work to send one letter. And it is a lot of work to send one letter—encryption is not an easy thing to do. But look at what has been accomplished: complete secrecy using a public key, with no need for any prior arrangement between the participants. You could publish your key on a noticeboard and anyone could send you a secret message, yet no-one could decrypt it without the private key. And in real life all the calculation is done by a software package that you acquire (typically built into your web browser), so it’s only a computer that has to work hard.

Perhaps your class would like to know that they have joined the very select group of people who have actually worked through a public-key encryption example by hand—
practising computer scientists would consider this to be an almost impossible task and few people have ever done it!

Now, what about eavesdropping? Bill’s map is like the ones in the Tourist Town activity (Activity 14), where the marked intersections are a minimal way of placing ice-cream vans to serve all street corners without anyone having to walk more than one block. We saw in Tourist Town that it’s easy for Bill to make up such a map by starting with the pieces shown in his private map, and it’s very hard for anyone else to find the minimal way to place ice-cream vans except by the brute-force method. The brute-force method is to try every possible configuration with one van, then every configuration with two vans, and so on until you hit upon a solution. No-one knows whether there is a better method for a general map—and you can bet that lots of people have tried to find one!

Providing Bill starts with a complicated enough map with, say, fifty or a hundred intersections, it seems like no-one could ever crack the code—even the cleverest mathematicians have tried hard and failed. (But there is a caveat: see below under What’s it all about?)

6. Having been through one example with the whole class, divide the students into groups of, say, four. Give each pair of each group the public map on the Kid Krypto Maps. Each pair should choose a “message” (any integer), encode it with the public key, and give the resulting map to the other group. The other group can try to decode it, but they are unlikely to be successful until they are given (or work out!) the private map. Then give out the private map and see if they can now decode it correctly.

7. Now each pair can design their own map, keeping the private version secret and giving the public version to the other pair—or indeed “publishing” it on the classroom board. The principle for designing maps is just the same as was discussed in the Tourist Town activity, and extra streets can be added to disguise the solution. Just be careful not to add extra streets into any of the “special” points. That would create an intersection from which two ice-cream vans could be reached in one hop, which is all right for the tourist town situation but would cause havoc when encrypting. That is because the special points no longer decompose the map into non-overlapping pieces, as illustrated in the private map, and this is essential for the trick to work.
Worksheet Activity: Kid Krypto Maps

Use these maps as described in the text to encrypt and decrypt messages.
Worksheet Activity: Kid Krypto Encoding

Display this "map" to the class and use it to demonstrate the encoding of a message.
What’s it all about?

It’s clear why you might want to send secret messages over computer networks that no-one but the intended recipient could decode, no matter how clever they were or how hard they tried. And of course there are all sorts of ways in which this can be done if the sender and receiver share a secret code. But the clever part of public-key encryption is that Amy can send Bill a secure message without any secret prior arrangement, just by picking up his “lock” from a public place like a web page.

Secrecy is only one side of cryptography. Another is authentication: When Amy receives a message from Bill, how does she know that it really comes from him and not from some imposter? Suppose she receives electronic mail that says, “Darling, I’m stuck here without any money. Please put $100 in my bank account, number 0241-45-784329 – love, Bill.” How can she know whether it really comes from Bill? Some public-key cryptosystems can be used for this, too. Just as Amy sends Bill a secret message by encoding it with his public key, he can send her a message that only he could have generated by encoding it with his private key. If Amy can decode it with Bill’s public key, then it must have come from him. Of course, anyone else could decode it too, since the key is public, but if the message is for her eyes only, Bill can then encode it a second time with Amy’s public key. This dual encoding provides both secrecy and authentication with the same basic scheme of public and private keys.

Now is the time to admit that while the scheme illustrated in this activity is very similar to an industrial-strength public-key encryption system, it is not in fact a secure one—even if quite a large map is used.

The reason is that although there is no known way of finding the minimal way to place ice-cream vans on an arbitrary map, and so the scheme is indeed secure from this point of view, there happens to be a completely different way of attacking it. The idea is unlikely to occur to school students, at least up to high school level, but you should at least know that it exists. You might say that the scheme we have been looking at is student-secure, but not mathematician-secure. Please ignore the next paragraph if you are not mathematically inclined!

Number the intersections on the map 1, 2, 3, ... Denote the original numbers that are assigned to intersections by \(b_1, b_2, b_3, \ldots\), and the numbers that are actually transmitted by \(t_1, t_2, t_3, \ldots\). Suppose that intersection 1 is connected to intersections 2, 3, and 4. Then the number that is transmitted for that intersection is

\[ t_1 = b_1 + b_2 + b_3 + b_4. \]

Of course, there are similar equations for every other intersection—in fact, there are the same number of equations as there are unknowns \(b_1, b_2, b_3, \ldots\). An eavesdropper knows the public map and the numbers \(t_1, t_2, t_3, \ldots\) that are transmitted, and can therefore write down the equations and solve them with an equation-solving computer program. Once the original numbers have been obtained, the message is just their sum—there is actually no need ever to discover the decryption map. The computational effort required to solve the equations directly using Gaussian elimination is proportional to the cube of the number of equations, but because these equations are sparse ones—most of the coefficients are zero—even more efficient techniques exist. Contrast this with the exponential computational effort that, as far as anyone knows, is the best one can do to come up with the decryption map.
We hope you don’t feel cheated! In fact, the processes involved in real public-key cryptosystems are virtually identical to what we have seen, except that the techniques they use for encoding are different—and really are infeasible to do by hand. The original public-key method, and still one of the most secure, is based on the difficulty of factoring large numbers.

What are the factors of the 100-digit number 9,412,343,607,359,262,946,971,172,136, 294,514,357,528,981,378,983,082,541,347,532,211,942,640,121,301,590,698,634,089, 611,468,911,681? Don’t spend too long!

They are 86,759,222,313,428,390,812,218,077,095,850,708,048, 977 and 108,488,104,853,637,470,612,961,399,842,972,948,409,834,611,525,790,577,216,753. There are no other factors: these two numbers are prime. Finding them is quite a job: in fact, it’s a several-month project for a supercomputer.

Now in a real public-key cryptosystem, Bill might use the 100-digit number as his public key, and the two factors as the private key. It would not be too difficult to come up with such keys: all you need is a way of calculating large prime numbers. Find two prime numbers that are big enough (that’s not hard to do), multiply them together, and—hey presto, there’s your public key. Multiplying huge numbers together is no big deal for a computer. Given the public key, no-one can find your private key, unless they have access to several months of supercomputer time. And if you’re worried that they might, use 200-digit primes instead of 100-digit ones—that’ll slow them down for years! The main thing is that the cost of cracking the key is higher than the value of the information it would unlock. In practice, 512-bit or larger keys are common for setting up secure connections, which is equivalent to about 155 decimal digits or more.

We still haven’t been given a way to encode a message using a prime-number based public key in such a way that it can’t be decoded without the private key. In order to do this, life is not quite as simple as we made out above. It’s not the two prime numbers that are used as the private key and their product as the public key, instead it’s numbers derived from them. But the effect is the same: you can crack the code by factoring the number. Anyway, it’s not difficult to overcome these difficulties and make the scheme into a proper encryption and decryption algorithm, but let’s not go into that here. This activity has already been enough work!

How secure is the system based on prime numbers? Well, factoring large numbers is a problem that has claimed the attention of the world’s greatest mathematicians for several centuries, and while methods have been discovered that are significantly better than the brute-force method of trying all possible factors, no-one has come up with a really fast (that is, polynomial-time) algorithm. (No-one has proved that such an algorithm is impossible, either.) Thus the scheme appears to be not just school-student secure, but also mathematician-secure. But beware: we must be careful. Just as there turned out to be a way of cracking Bill’s code without solving the Tourist Town problem, there may be a way of cracking the prime-number codes without actually factoring large numbers. People have checked carefully for this, and it seems OK.

Another worry is that if there are just a few possible messages, an interloper could encrypt each of them in turn using the public key, and compare the actual message with all the possibilities. Amy’s method avoids this because there are many ways of encrypting the same message, depending on what numbers were chosen to add up to the code value. In practice, cryptographic systems are designed so that there are just too many possible messages to even begin to try them all out, even with the help of a very fast computer.
It is not known whether a fast method for solving the prime factorization problem exists. No one has managed to devise one, but also it has not been proven that a fast method is impossible. If a fast algorithm for solving this problem is found, then many currently used cryptographic systems will become insecure. In Part IV we discussed NP-complete problems, which stand or fall together: if one of them is efficiently solvable then they all must be. Since so much (unsuccessful) effort has been put into finding fast algorithms for these problems, they would seem like excellent candidates for use in designing secure cryptosystems. Alas, there are difficulties with this plan, and so far the designers of cryptosystems have been forced to rely on problems (such as prime factorization) that might in fact be easier to solve than the NP-complete problems—maybe a lot easier. The answers to the questions raised by all this are worth many millions of dollars to industry and are regarded as vital to national security. Cryptography is now a very active area of research in computer science.

**Further reading**

Harel’s book *Algorithmics* discusses public-key cryptography; it explains how to use large prime numbers to create a secure public-key system. The standard computer science text on cryptography is *Cryptography and data security* by Dorothy Denning, while a more practical book is *Applied cryptography* by Bruce Schneier. Dewdney’s *Turing Omnibus* describes another system for performing public key cryptography.
Part VI

The human face of computing-
*Interacting with computers*
The Human Face of Computing

Why are computers so hard to get along with? Many people have stories about how difficult computers are to use, how they never seem to do what you really want them to, how they keep going wrong and make ridiculous mistakes. Computers seem to be made for wizards, not for ordinary people. But they should be made for ordinary people, because computers are everyday tools that help us to learn, work, and play better.

The part of a computer system that you interact with is called its “user interface.” It’s the most important bit! Although you might think of what the program actually does as the main thing and the user interface as just how you get into it, a program is no good at all if you can’t interact with it and make it do what you want. User interfaces are very difficult to design and build, and it has been estimated that when writing programs, far more effort goes into the interface than into any other part. Some software has excellent user interfaces, interfaces that need no complicated instructions and become almost invisible as you are drawn into using the application. But countless software products which are otherwise very good have been complete flops because they have strange user interfaces. And whole industries have been built around a clever interface idea—like the word processor or smartphones—that promotes access to computational functions which are really quite elementary in themselves.

But why do we have to have user interfaces at all? Why can’t we just talk to our computers the way we do to our friends? Good question. Maybe someday we will; maybe not. But certainly not yet: there are big practical limitations on how “intelligent” computers can be today. The activities that follow will help you understand the problems of user interface design, and help you to think more clearly about the limitations of computers and be wary of the misleading hype that is often used to promote computer products.

For teachers

Computing is not so much about calculation as it is about communication. Computing per se really has no intrinsic value; it is only worthwhile if the results are somehow communicated to the world outside the computer, and have some influence there. And a lot of the activities in this book are about communication. Representing data (Part I) shows how different kinds of information can be communicated to a computer or between computers. Representing processes (Part III) is about how to communicate processes to a computer to tell it how to accomplish certain tasks—after all, “programming” is really only explaining to a computer, in its own language! Cryptography (Part V) is about how to communicate in secret, or to communicate bits of secrets without revealing all.

The activities that follow are about how people communicate with computers. While the rest of the book is based on well understood technical ideas, this part is not. That makes it both easier, in that no special knowledge is required of the students, and more difficult, in that a certain level of maturity is needed to understand what the activities are about and relate them to a broader context. These activities contain more detailed explanations than most of the others because it is necessary to give you, the teacher, enough background material to be in a position to help draw out some of the implications in class discussion.

There are two activities in this section. The first is about the area known as the “human–computer interface,” commonly abbreviated to HCI. In order to “unplug” this aspect of
computing without depending on prior knowledge of a particular example of a computer system, we have invented a design exercise that does not really involve computers—but does introduce fundamental principles that are used in the design of human–computer interfaces. Because human interface design is culture-dependent, there are not necessarily any “right” answers in this activity, which may frustrate some students. The second activity is about the area known as “artificial intelligence,” or AI. It involves a guessing game that stimulates students into thinking about what computers can and can’t do.

**For the technically-minded**

Human–computer interaction has become one of the hottest research areas in computer science as people realize how much the success of a software product depends on its user interface. The subject draws heavily on a wide range of disciplines outside computer science, such as psychology, cognitive science, linguistics, sociology—even anthropology. Few computer scientists have training in these areas, and HCI represents an important growth area for people who are interested in the “softer” side of the subject.

Artificial intelligence is a topic that often raises hackles and causes disputes. In this book we have tried to steer a middle path between AI aficionados who believe that intelligent machines are just around the corner, and AI sceptics who believe that machines cannot in principle be intelligent. Our goal is to encourage students to think independently about such issues, and to promote a balanced view.

The activities here draw heavily on two eminently readable books, Don Norman's *The design of everyday things* and John Haugeland's *Artificial intelligence: the very idea*, which we enthusiastically recommend if you are interested in pursuing these issues further.

Computers involve another important kind of communication, one that is not touched upon in this book: communication between people who are building a computer system. Students who learn about computers and make their way into the job market—perhaps having graduated in computer science from university—are invariably surprised by how much interpersonal communication their job entails. Computer programs are the most complex objects ever constructed by humankind, with millions or perhaps billions of intricately interlocking parts, and programming projects are tackled by close-knit teams that work together and spend a great deal of their time communicating. Once the product is complete, there is the job of communicating with customers through user manuals, courses, “help” phonelines, online support, and the like—not to mention the problem of communicating with potential customers through demonstrations, displays, and advertising. We haven’t yet found a way to realistically “unplug” for students the interpersonal communication aspect of computing, so this book doesn’t address it. But it is the kind of thing that computer professionals who are visiting a classroom may be able to describe from their own experience and bring out in discussion.
Activity 19

The chocolate factory—Human interface design

Summary
The aim of this activity is to raise awareness of human interface design issues. Because we live in a world where poor design is rife, we have become accustomed (resigned?) to putting up with problems caused by the artifacts we interact with, blaming ourselves ("human error," “inadequate training,” “it’s too complicated for me”) instead of attributing the problems to flawed design. The issue is greatly heightened by computers because they have no obvious purpose—indeed, they are completely general purpose—and their appearance gives no clues about what they are for nor how to operate them.

Curriculum Links
✓ Technology: Understand that technological outcomes are products or systems developed by people and have a physical nature and a functional nature.
✓ Technology: Understand that technology is purposeful intervention through design.

Skills
✓ Design.
✓ Reasoning.
✓ Awareness of everyday objects.

Ages
✓ 7 and up

Materials
Each group of students will need:
✓ a copy of the sheets How do you open doors? and Stove top, and
✓ a copy of the images on the worksheet Icons, either displayed on a projector, shown on overhead projector transparency or on cards that can be displayed to the class, and
✓ one or more of the six cards on the Icon cards page. Cut the sheet into individual cards and divide them between the groups.
The Chocolate Factory

Introduction
The great chocolate factory is run by a race of elf-like beings called Oompa-Loompas. These Oompa-Loompas have terrible memories and no written language. Because of this, they have difficulty remembering what to do in order to run the chocolate factory, and things often go wrong. Because of this, a new factory is being designed that is supposed to be very easy for them to operate.

Discussion
1. Explain the story to the students and divide them into small groups.

2. The first problem the Oompa-Loompas face is getting through the doors carrying steaming buckets of liquid chocolate. They cannot remember whether to push or pull the doors to open them, or slide them to one side. Consequently they end up banging into each other and spilling sticky chocolate all over the place. The students should fill out the “doors” worksheet How do you open doors. More than one box is appropriate in each case. For some of the doors (including the first one) it is not obvious how to open them, in which case the students should record what they would try first. Once they have filled out their own sheets, have the whole group discuss the relative merits of each type of door, particularly with regard to how easy it is to tell how it works, and how suitable it would be to use if you are carrying a bucket of hot chocolate. Then they should decide what kind of doors and handles to use in the factory.

3. Follow this activity with a class discussion. The table below comments briefly on each door in the worksheet. Real doors present clues in their frames and hinges as to how they open, and there are conventions about whether doors open inwards or outwards. Identify the kinds of door handles used in your school and discuss their appropriateness (they may be quite inappropriate!) Can you think of a door that often confuses you? Why? Do doors normally open inwards or outwards into corridors?—and why? (Answer: They open into rooms so that when you come out you won’t bash the door into people walking along the corridor, although in some situations they open outwards to make evacuation easier in an emergency.)

4. The key concept here is what is called the affordances of an object, which are its visible features—both fundamental and perceived—whose appearance indicates how the object should be used. Affordances are the kinds of operation that the object permits, or “affords.” For example, it is (mostly) clear from their appearance that chairs are for sitting, tables are for placing things on, knobs are for turning, slots are for inserting things into, buttons are for pushing. On a computer interface the affordances are the shapes of buttons, text boxes, menus and so on, which give the user a clue as to how they should be used. If a button is made to look like something else, then people won’t realise they can push it. This might seem obvious, but these problems aren’t hard to find on digital devices.

---

1 With apologies to Roald Dahl. You’ll know about the Oompa-Loompas if you’ve read his wonderful tale Charlie and the Chocolate Factory. If not, never mind: the plot is not relevant to this activity.
<table>
<thead>
<tr>
<th>Plain door</th>
<th>Can’t see how to open this one at all, except that since it has no handle, it must require pushing rather than pulling.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labeled door</td>
<td>The label is like a tiny user manual. But should a door need a user manual? And the Oompa Loompas can’t read.</td>
</tr>
<tr>
<td>Hinge door</td>
<td>At least you can see which is the side that opens.</td>
</tr>
<tr>
<td>Bar door</td>
<td>It’s fairly clear that you are supposed to push the bar, but which side? Or should you pull?</td>
</tr>
<tr>
<td>Handle door</td>
<td>Handles like this are usually for pulling—or sliding.</td>
</tr>
<tr>
<td>Knob door</td>
<td>The knob shows what to grasp, but not whether to push or pull; it probably doesn’t slide.</td>
</tr>
<tr>
<td>Panel door</td>
<td>It’s clear that you push this. What else could you do?</td>
</tr>
<tr>
<td>Glass door</td>
<td>The small vertical bar on this side signals “pull”; the longer horizontal one on the other signals “push”.</td>
</tr>
<tr>
<td>Sliding door</td>
<td>This one’s only for sliding.</td>
</tr>
</tbody>
</table>

Doors are very simple objects. Complex things may need explaining, but simple things should not. When simple objects need pictures, labels, or instructions, then design has failed.

5. **The pots containing different kinds of chocolate have to cook at different temperatures.** In the old chocolate factory the stoves were as shown in the *Stove top sheet*. The left-hand knob controlled the rear left heating element, the next knob controlled the front left element, the next one controlled the front right, and the right-hand knob controlled the rear right element. The Oompa-Loompas were always making mistakes, cooking the chocolate at the wrong temperature, and burning their sleeves when reaching across the elements to adjust the controls. The students should recall how the controls are laid out on their cookers at home and come up with a better arrangement for the new factory.

Follow this activity with a class discussion. This picture below shows some common arrangements. All but the one at the lower left have the controls at the front, to avoid having to reach across the elements. In the design at the top left, there are so many possible mappings from controls to burners (24 possibilities, in fact) that eight words of labeling are needed. The “paired” arrangement in the top center is better, with only four possible mappings (two for the left cluster and two for the right); it requires just four labeling words. The design at the top right specifies the control–burner relationship diagrammatically rather than linguistically (which is good for the Oompa-
The lower three designs need no labels. The left-hand one has a control by each burner, which is awkward and dangerous. The other two involve relocating the burners slightly, but for different reasons: in the center design they are moved to leave room for the controls, while in the right-hand one they are rearranged to make the correspondence clear.

The key concept here is the mapping of controls to their results in the real world. Natural mapping, which takes advantage of physical analogies and cultural standards, leads to immediate understanding. The spatial correspondences at the bottom of the picture are good examples—they are easily learned and always remembered. Arbitrary mappings, as in the top arrangements, need to be labeled, or explained and memorized.

6. The factory is full of conveyer belts carrying pots of half-made chocolate in various stages of completion. These conveyer belts are controlled manually by Oompa-Loompas, on instructions from a central control room. The people in the control room need to be able to tell the Oompa-Loompa to stop the conveyer belt, or slow it down, or start it up again.

In the old factory this was done with a voice system: the control room person’s voice came out of a loudspeaker by the conveyer belt controls. But the factory was noisy and it was hard to hear. The groups should design a scheme that uses visual signals.

One possibility is to put in lights to signal Stop!, Slow down and Start up. Students will probably work out that these should follow the normal traffic-light convention by using red for Stop!, yellow for Slow down and green for Start up. They should be arranged just like traffic lights, with red at the top and green at the bottom.
But now reveal to the class that in Oompa-Loompa land, traffic lights work differently from the way they do for us: yellow means stop, red means go, and lights go green to warn people that they will soon have a stop light. How does this affect things? (Answer: the factory should follow the Oompa-Loompa’s traffic-light convention—we should not try to impose our own.)

The key concepts here are those of transfer effects—people transfer their learning and expectations of previous objects into new but similar situations—and population stereotypes—different populations learn certain behaviours and expect things to work in a certain way. Although the traffic light scenario may seem far-fetched (though nothing is all that far-fetched in Oompa-Loompa land), there are many examples in our own world: in America light switches are on when they are up and off when they are down, whereas in Britain the reverse is true; calculator keypads and touchtone phones are laid out in different ways; and number formats (decimal point or comma) and date formats (day/month/year or month/day/year) vary around the world.

7. When one shift of Oompa-Loompas finishes work in the chocolate factory, they must clean up and put away pots and pans and jugs and spoons and stirrers ready for the next shift. There is a cupboard with shelves for them to put articles on, but the next shift always has trouble finding where things have been put away. Oompa-Loompas are very bad at remembering things and have trouble with rules like “always put the pots on the middle shelf,” “put the jugs to the left.” The groups of students should try to come up with a better solution.

The diagram on the right shows a good arrangement (which is sometimes used—but for rather different reasons—on yachts and other places where it is necessary to stop things sliding around). The key concept here is to use visible constraints to make it obvious where everything is supposed to go. It is clear from the size and shape of each hole which utensil it is intended for: the designer has made the constraints visible and used the physical properties of the objects to avoid the need to rely on arbitrary conventions.

8. In the main control room of the chocolate factory there are a lot of buttons and levers and switches that operate the individual machines. These need to be labeled, but because the Oompa-Loompas can’t read, the labels have to be pictorial—iconic—rather than linguistic.
To give the students a feeling for icons, the worksheet *Icons* shows some examples. The students should identify what the icons might mean (for example, the letter going into a mailbox might represent sending a message). There are no “correct” answers to this exercise; the idea is simply to identify possible meanings.

9. Now let’s design icons for the chocolate factory. The cards on worksheet *Icon cards* specify clusters of related functions, and each group of students receives one or more cards without the other groups knowing what they are. A control panel is to be designed for the function clusters that contains individual icons for each of the five or six operations. The groups then show their work to the other students, without saying what the individual operations are, to see if they can guess what the icons mean. Encourage the use of imagination, color, and simple, clear icons.
Worksheet Activity: How do you open doors?

Fill out the worksheet to show how you think each type of door opens.

Plain Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along

Labeled Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along

Hinge Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along

Bar Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along

Handle Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along

Knob Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along

Panel Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along

Glass Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along

Sliding Door
- [ ] Push [ ] Left side
- [ ] Pull [ ] Right side
- [ ] Slide it along
Worksheet Activity: Stove top

Redesign the stove so that the controls are easy to use. Front or back panels can be added to the design if desired.

OLD

NEW
Worksheet Activity: Icons

What do you think each of the icons (symbols) means?

**In an Office...**

- Calendar for 1995
- Calendar for January 4
- Filing cabinet
- Envelope
- Pyramid
- Key

**On a box...**

- Box with umbrella
- Box with an arrow pointing left
- Box with a label
- Box with a label for freezing point
- Box with a label for melting point
- Box with a no smoking symbol

© 2012 Computer Science Unplugged (csunplugged.org)
**Worksheet Activity: Icon cards**

Cut out the cards and give one to each group. Have each group design icons (symbols) to put on a control panel to represent each instruction.

**Ingredients**
- add • cocoa
- add • milk
- add • sugar
- add • extra sugar
- add • butter

**Extras**
- add • nuts
- add • caramel
- add • ginger
- add • raisins
- add • coconut

**Making**
- start mixing
- stop mixing
- start heating
- stop heating
- pour into moulds
- stamp a pattern (lots of different ones!)

**Tasting**
- taste it
- wonderful!—premium grade
- ok—regular grade
- yech—cooking chocolate
- yech, yech—throw out

**Sizing**
- small bar
- medium bar
- large bar
- humungous bar
- set bar size [in squares]
- make chocolate chips

**Packing**
- wrap with foil
- wrap with paper
- put into bag
- put into box
- start conveyor belt
- stop conveyor belt
Variations and extensions

Can the students set the time on a digital wristwatch or microwave oven? The mappings involved in the cooker layouts were simple because there were four controls for four burners. More difficulty occurs whenever the number of actions exceeds the number of controls. The controls on wristwatches or microwaves are often exceedingly complex, not because of the number of buttons (often there are only a few), but because of the number of states the device can get into. (“You would need an engineering degree from MIT to work this,” someone looking at his new wristwatch once told Don Norman, a leading user interface psychologist. Don has an engineering degree from MIT, and, given a few hours, he can figure out the watch. But why should it take hours?)

Students should keep an eye out for places where people get confused or frustrated using digital devices—mobile phones, video recorders, computers, remote controls—all these devices provide opportunities for frustrating users! Students should ask themselves, what is it about the device that confuses the users, and how might it have been designed better?

What’s it all about?

Human–computer interaction is about designing, evaluating, and implementing computer systems that allow people to carry out their activities productively and safely. In the old days, computers were for specialists and the users could be expected to be highly educated and specially trained in their use. Later people thought it was perfectly normal to buy a “dummies” book to find out how to use their computer. But now computers are everyday tools that we all must use, and far greater attention must be paid to the human interface.

Many disasters, some involving loss of life, have occurred because of inadequate interfaces: airplane crashes and even shoot-downs of civilian airplanes, freeway pile-ups because of errors in switching remotely-operated highway signs, nuclear power station disasters. On a smaller scale, most people experience frustration—often extreme frustration (a police officer once fired bullets into his computer screen)—with computers and other high-tech devices every day in the workplace. And it is not just computers: what about those shrink-wrapped packages that you could only open if you had sharp claws or a hooked beak, doors that hurt your wrist as you try to push your way through, milk cartons that always splash you when you open them, elevators where you can’t see how you’re supposed to push the button, home entertainment systems whose advertisements claim to do everything, but make it almost impossible to do anything?

We are becoming used to “human error” and to thinking of ourselves as somehow inadequate; people often blame themselves when things go wrong. But many so-called human errors are actually errors in design. People have limitations in how much information they can process, and designers need to account for these; bad design cannot be rectified by producing a detailed and complicated user manual and expecting people to study it intensively and remember it forever. Also, humans are fallible and design needs to take this into consideration.

Interface evaluation is an essential part of the design process. The present activity has involved some evaluation when the students tested their icon designs on others. A more thorough evaluation would test the design on real Oompa-Loompas (who may perceive icons differently) in a carefully-controlled psychology-style experiment.

Although the problems caused by technology form the butt of many jokes, human interface design is by no means a laughing matter. Inadequate interfaces cause problems
ranging from individual job dissatisfaction to stock-market disasters, from loss of self-esteem to loss of life.

Further reading
Don Norman’s book *The design of everyday things* is a delightful—and liberating—account of the myriad design problems in everyday products. Jeff Johnson’s *Designing with the mind in mind* is a thought-provoking insight into how people think, and how interfaces should be designed to take account of the human element.
Activity 20

Conversations with computers—The Turing test

Summary
This activity aims to stimulate discussion on the question of whether computers can exhibit “intelligence,” or are ever likely to do so in the future. Based on a pioneering computer scientist’s view of how one might recognize artificial intelligence if it ever appeared, it conveys something of what is currently feasible and how easy it is to be misled by carefully-selected demonstrations of “intelligence.”

Curriculum Links
✓ Technology Level 3: Technological systems. Understand that technological systems are represented by symbolic language tools and understand the role played by the black box in technological systems.

Skills
✓ Interviewing.
✓ Reasoning.

Ages
✓ 7 years and up

Materials
✓ A copy of the questions in the Turing Test Questions sheet that each student can see (either one for each pair of students, or a copy displayed on a projector/overhead projector), and
✓ one copy of the answers in the Turing Test Answers sheet.
Conversations with Computers

Discussion
This activity takes the form of a game in which the students must try to distinguish between a human and a computer by asking questions and analyzing the answers. The game is played as follows.

There are four actors: we will call them Gina, George, Herb and Connie (the first letter of the names will help you remember their roles). The teacher coordinates proceedings. The rest of the class forms the audience. Gina and George are go-betweens, Herb and Connie will be answering questions. Herb will give a human’s answers, while Connie is going to pretend to be a computer. The class’s goal is to find out which of the two is pretending to be a computer and which is human. Gina and George are there to ensure fair play: they relay questions to Herb and Connie but don’t let anyone else know which is which. Herb and Connie are in separate rooms from each other and from the audience.

What happens is this. Gina takes a question from the class to Herb, and George takes the same question to Connie (although the class doesn’t know who is taking messages to whom). Gina and George return with the answers. The reason for having go-betweens is to ensure that the audience doesn’t see how Herb and Connie answer the questions.

Before the class begins this activity, select people to play these roles and brief them on what they should do. Gina and George must take questions from the class to Herb and Connie respectively, and return their answers to the class. It is important that they don’t identify who they are dealing with, for example, by saying “She said the answer is...” Herb must give his own short, accurate, and honest answers to the questions he is asked. Connie answers the questions by looking them up on a copy of the Turing Test Answers sheet. Where the instructions are given in italics, Connie will need to work out an answer.

Gina and George should have pencil and paper, because some of the answers will be hard to remember.
1. Before playing the game, get the students’ opinions on whether computers are intelligent, or if the students think that they might be one day. Ask for ideas on how you would decide whether a computer was intelligent.

2. Introduce the students to the test for intelligence in which you try to tell the difference between a human and a computer by asking questions. The computer passes the test if the class can’t tell the difference reliably. Explain that Gina and George will communicate their questions to two people, one of whom will give their own (human) answers, while the other will give answers that a computer might give. Their job is to work out who is giving the computer’s answers.

3. Show them the list of possible questions in the *Turing Test Questions* sheet. This can either be copied and handed out, or placed on a projector.

   Have them choose which question they would like to ask first. Once a question has been chosen, get them to explain why they think it will be a good question to distinguish the computer from the human. This reasoning is the most important part of the exercise, because it will force the students to think about what an intelligent person could answer that a computer could not.

   Gina and George then relay the question, and return with an answer. The class should then discuss which answer is likely to be from a computer.

   Repeat this for a few questions, preferably until the class is sure that they have discovered who is the computer. If they discover who is the computer quickly, the game can be continued by having Gina and George toss a coin to determine if they will swap roles so the class no longer know which role the two have.

   The answers that Connie is reading from are not unlike the ones that some “intelligent” computer programs can generate. Some of the answers are likely to give the computer away quickly. For example, no-one is likely to recite the square root of two to 20 decimal places, and most people (including, perhaps, the students in the class) would not be able to answer that question at all. Some questions will reveal the computer when their answers are combined. For example, the “Do you like...” answers sound plausible on their own, but when you encounter more than one it becomes apparent that a simple formula is being used to generate the answers from the questions. Some of the answers indicate that the question was misinterpreted, although the class might reason that the person could have made the mistake.

   Many of the answers are very bland, but safe, and a follow-up question would probably reveal that the computer doesn’t really understand the subject. Answering “I don’t know” is reasonably safe for the computer, and might even make it seem more human—we might expect a student to answer “I don’t know” to some of the questions too, such as the request for the square root of two. However, if a computer gives this answer too often, or for a very simple question, then again it would reveal its identity.

   Since the goal of the computer is to make the questioners think that they are dealing with a person, some of the answers are deliberately misleading—such as the delayed and incorrect answers to the arithmetic problem. The questions and answers should provide plenty of fuel for discussion.
Worksheet Activity: Turing Test Questions

Choose questions from this list to ask the hidden human and “computer”.

1. What is the name of Bart Simpson’s baby sister?
2. What do you think of Roald Dahl?
3. Are you a computer?
4. What is the next number in the sequence 3, 6, 9, 12, 15?
5. What do you think of nuclear weapons?
6. What is $2 \times 78$?
7. What is the square root of two?
8. Add 34957 to 70764.
9. Do you like school?
10. Do you like dancing?
11. What day is it today?
12. What time is it?
13. How many days are there in February in a leap year?
14. How many days are there in a week?
15. For which country is the flag a red circle on a white background?
16. Do you like to read books?
17. What food do you like to eat?
Worksheet Activity: Turing Test Answers

Choose questions from this list to ask the hidden human and “computer”.

1. What is the name of Bart Simpson’s baby sister?
   
   I can’t remember.

2. What do you think of Roald Dahl?
   
   He writes funny books.

3. Are you a computer?
   
   Are you a computer?

4. What is the next number in the sequence 3, 6, 9, 12, 15?
   
   18.

5. What do you think of nuclear weapons?
   
   Nuclear weapons are very dangerous and should not be used.

6. What is $2 \times 78$?
   
   166 (This is deliberately incorrect!)

7. What is the square root of two?
   
   1.41421356237309504878

8. Add 34957 to 70764.
   
   Wait for about 20 seconds before giving the answer
   ...105621.

9. Do you like school?
   
   Yes, I like school.

10. Do you like dancing?
    
   Yes, I like dancing.

11. What day is it today?
    
   Give the correct day of the week.

12. What time is it?
    
   Give the correct time.
13. How many days are there in February in a leap year?

2000 and 2004 are leap years. *(This is deliberately incorrect!)*

14. How many days are there in a week?

**Seven.**

15. For which country is the flag a red circle on a white background?

I don’t know.

16. Do you like to read books?

Yes, I like to read books.

17. What food do you like to eat?

I’m not hungry, thanks.
Variations and extensions

The game can be played with as few as three people if Gina also takes the role of George and Connie. Gina takes the question to Herb, notes his answer, and also notes the answer from the Turing Test Answers sheet. She returns the two answers, using the letters A and B to identify who each answer came from.

In order to consider whether a computer could emulate a human in the interrogation, consider with the class what knowledge would be needed to answer each of the questions on Turing Test Answers. The students could suggest other questions that they would have liked to ask, and should discuss the kind of answers they might expect. This will require some imagination, since it is impossible to predict how the conversation might go.

By way of illustration, here are two example conversations. The former illustrates “factual” questions that a computer might be able to answer correctly, while the latter shows just how wide-ranging the discussion might become, and demonstrates the kind of broad knowledge that a computer might need to call upon.

There is a system called “Eliza” that is widely available on the web (it is a kind of “chatbot”, which is a system that you can have typed conversations with). Eliza simulates a session with a psychotherapist, and can generate remarkably intelligent conversation using some simple rules. Some sample sessions with Eliza are discussed below. Students might try out Eliza, or other chatbots, although be warned that some have been trained using language and subjects that might not be appropriate for school students.

What’s it all about?

For centuries philosophers have argued about whether a machine could simulate human intelligence, and, conversely, whether the human brain is no more than a machine running a glorified computer program. This issue has sharply divided people. Some find the idea preposterous, insane, or even blasphemous, while others believe that artificial intelligence is inevitable and that eventually we will develop machines that are just as intelligent as us. (As countless science fiction authors have pointed out, if machines do
eventually surpass our own intelligence they will themselves be able to construct even cleverer machines.) Artificial Intelligence (AI) researchers have been criticized for using their lofty goals as a means for attracting research funding from governments who seek to build autonomous war machines, while the researchers themselves decry the protests as a Luddite backlash and point to the manifest benefits to society if only there was a bit more intelligence around. A more balanced view is that artificial intelligence is neither preposterous nor inevitable: while no present computer programs exhibit “intelligence” in any broad sense, the question of whether they are capable of doing so is an experimental one that has not yet been answered either way.

The AI debate hinges on a definition of intelligence. Many definitions have been proposed and debated. An interesting approach to establishing intelligence was proposed in the late 1940s by Alan Turing, an eminent British mathematician, wartime counter spy and long-distance runner, as a kind of “thought experiment.” Turing’s approach was operational—rather than define intelligence, he described a situation in which a computer could demonstrate it. His scenario was similar to the activity described above, the essence being to have an interrogator interacting with both a person and a computer through a teletypewriter link (the very latest in 1940s technology!) If the interrogator could not reliably distinguish one from the other, the computer would have passed Turing’s test for intelligence. The use of a teletypewriter avoided the problem of the computer being given away by physical characteristics or tone of voice. One can imagine extending the exercise so that the machine had to imitate a person in looks, sound, touch, maybe even smell too—but these physical attributes seem hardly relevant to intelligence.

Turing’s original test was a little different from ours. He proposed, as a preliminary exercise, a scenario where a man and a woman were being interrogated, and the questioner had to determine their genders. The man’s goal was to convince the questioner that he was the woman, and the woman’s was to convince the questioner that she was herself. Then Turing imagined—for this was only proposed as a thought experiment—a computer being substituted for one of the parties to see if it could be just as successful at this “imitation game” as a person. We altered the setup for this classroom activity, because the kind of questions that students might ask to determine gender would probably not be appropriate, and besides, the exercise promotes sexual stereotyping—not to mention deception.

Imitating intelligence is a difficult job. If the roles were reversed and a person was trying to pass themselves off as a computer, they would certainly not be able to do so: they would be given away by their slow (and likely inaccurate) response to questions like “What is 123456 × 789012?”
However, it is surprisingly easy for a computer to acquire a thin veneer of conversational ability. A famous program developed in the early 60s, Eliza (mentioned earlier), simulated a non-directive psychotherapist in a dialog with the user. On the right is an example of a conversation with this system. Something approaching a natural conversation is achieved by devices such as

- generating canned responses (e.g. *Can you think of a specific example* in response to something beginning *They’re always …; I’m sorry to hear that ...* in response to *He says ...*);

- echoing the user’s statements (e.g. *Your boyfriend made you come here* in response to *My boyfriend made me come here*);

- recognizing key words (*Tell me more about your family* in response to *… mother …*);

- some stock phrases (*What would it mean to you if ...*);

- issues taken up from earlier parts of the conversation (*Does that have anything to do with the fact that ...*).

Of course, this does not mean that the Turing test has been passed, because the person is not asking questions with the aim of identifying whether the respondent is a machine or not: if they were, it would not be long before all was revealed.
A widely-quoted incident, which may be apocryphal but has now passed into the folklore of artificial intelligence, occurred when a computer sales vice-president allegedly came across a terminal that was normally connected directly to a particular person’s house, but on this occasion happened to be running the “psychotherapist” program. This conversation shows what happened. Although the vice-president was evidently taken in, it is clear that the situation predisposed him to believe that he was talking to a person. If he had suspected otherwise, he would soon have found out!

Another system that appears to be able to hold intelligent conversations is a program called “SHRDLU”, developed in the late 60s that accepted instructions to manipulate students’ blocks on a table top. In fact, the blocks and the table were simulated and presented in the form of a picture on a computer screen, although it would not have been too difficult to make things work with a real robot (at least in principle). The picture depicts the robot’s micro-world.
SHRDLU could carry on surprisingly fluent conversations, as shown in the dialog. Amongst other things, this illustrates how it can obey orders, disambiguate instructions, work out the referents of pronouns, and understand new terms (“steeple”).

Person: Pick up a big red block.
Robot: OK.

_The system answers “OK” when it carries out a command. In order to pick up the red block, it had to clear it off by finding a space for the green one and moving the green one away._

Person: Grasp the pyramid.
Robot: I don’t know which pyramid you mean.

_There are three pyramids in the scene, so the program does nothing._

Person: Find a block which is taller than the one you are holding and put it into the box.
Robot: By “it”, I assume you mean the block which is taller than the one I am holding.

_Printed out to keep us informed._

Robot: OK.

_The hand has to set down the red block to pick up the taller blue one._

Person: What does the box contain?
Robot: The blue pyramid and the blue block.

Person: What is the pyramid supported by?
Robot: Earlier the system responded to “the pyramid” by saying “I don’t know which pyramid you mean.” Now, it knows that we must mean the one which was just mentioned.

Robot: The box.

Person: Does a steeple ...

Robot: Sorry, I don’t know the word “steeple”.

_The system interrupts immediately when it sees a word it can’t figure out._

Person: A “steeple” is a stack which contains two green cubes and a pyramid.

_The quote marks indicate that we are defining a new word._

Robot: I understand.

Person: Are there any steeples now?
Robot: No.

_The new definition is used as a description to recognize steeples._

Person: Build one.
Robot: OK.

_The description given in the definition is enough to deduce a plan for building new objects as well as recognizing them. The word “one” means “a steeple”, and the plan is carried out._
However, the system is very fragile and the illusion it creates is easily shattered.

Here is a continuation of the conversation—which was not actually generated by the program, but certainly could have been. Although it was possible to define the new term *steeple* in the last conversation, only a very limited kind of term can be described because the robot has an extremely restricted “intelligence.”

Ultimately, it has to admit that it doesn’t even understand the words it uses itself (“Sorry, I don’t know the word ‘sorry’!”)

Now consider an alternate, completely imaginary, continuation of the conversation, which might have been generated by a student. Although tiny, this fragment demonstrates a wealth of world knowledge.

- The “robot” appreciates that a water pistol is less valuable if you already have one.
- Furthermore, it *expects the person to know that too* since it doesn’t bother to explain that this is a reason for declining the offer.
- Still, it sees that the person wants a block badly enough to part with something important.
- Also, it appreciates that—unlike water pistols—blocks are valuable in quantity.
- It tries to soften the person up by demeaning the frog.
- It implies that it is prepared to haggle.

Certainly this degree of artfulness is beyond today’s computers!

| Person: | Trade you a water pistol for the big red block. |
| Robot: | Sorry, I don't know the word “trade”. |
| Person: | A “trade” is a free exchange of property. |
| Robot: | Sorry, I don't know the word “free”. |
| Person: | A “free” act is done willingly, without being forced. |
| Robot: | Sorry, I don't know the word “act”. |
| Person: | “Acts” are what people do on purpose, and not by accident. |
| Robot: | Sorry, I don't know the word “people”. |
| Person: | Sorry, I thought you were smarter than you are. |
| Robot: | Sorry, I don't know the word “sorry”. |

| Person: | Trade you a water pistol for the big red block. |
| Robot? | I already have a water pistol, but I'll give you two blocks and a pyramid for your slimy old frog. |
The story on the right was generated by a more recent computer program (late 80s). Although it is immediately given away as computer-generated by its scarcity and terseness, it is not hard to imagine how it could be dressed up by adding all sorts of extra detail. What is interesting is not the superficial aspects of the story but the plot that it embodies. While this is a long way from any human-generated plot, it does seem to capture some human elements of conflict. These days there are a number of systems around for automatically generating stories, although the challenge in evaluating them is to determine how much of the material is just standard patterns with the gaps filled in, and how much is a plot that has been constructed creatively as above.

There is an annual competition for the Loebner prize, in which computer programs compete to pass the Turing test by fooling judges into thinking that they are human. As of 2012, no computer has yet won the gold or silver prizes, which involve consistently fooling the judges, but a bronze prize is awarded each year for the one judged to be the most human. In the first year of the competition (1991) a program managed to win a bronze award by, amongst other tricks, making typing mistakes to appear to be more human!

No artificial intelligence system has been created that comes anywhere near passing the full Turing test. Even if one did, many philosophers have argued that the test does not really measure what most people mean by intelligence. What it tests is behavioral equivalence: it is designed to determine whether a particular computer program exhibits the symptoms of intellect, which may not be the same thing as genuinely possessing intelligence. Can you be humanly intelligent without being aware, knowing yourself, being conscious, being capable of feeling self-consciousness, experiencing love, being ...

The AI debate is likely to be with us for many more decades.

Further reading

*Artificial intelligence: the very idea* by the philosopher John Haugeland is an eminently readable book about the artificial intelligence debate, and is the source of some of the illustrations in this activity (in particular, the SHRDLU conversations, and the discussion of them).

The original Turing test was described in an article called “Computing machinery and intelligence,” by Alan Turing, published in the philosophical journal *Mind* in 1950, and reprinted in the book *Computers and thought*, edited by Feigenbaum and Feldman. The article included the first two conversations.

The psychotherapist program was described in “ELIZA—A computer program for the study of natural language communication between man and machine,” by J. Weizenbaum, published in the computer magazine *Communications of the Association for Computing Machinery* in 1966.
The blocks-world robot program is described in a PhD thesis by Terry Winograd which was published as a book entitled *Understanding natural language* (Academic Press, New York, 1972).

The program that generated the story of Truman and Horace is described in “A planning mechanism for generating story text,” by Tony Smith and Ian Witten, published in the *Proceedings of the 10th International Conference on Computing and the Humanities* in 1990.