Magic and Algorithms: The Australian Magician's Dream

Paul Curzon and Peter McOwan
Queen Mary University of London

The skills you need to be a great stage magician are exactly the same as those you need to be a great computer scientist: computational thinking. Magic tricks are algorithms and that is all a computer program is too. When early computers searched for data, they were actually doing a magic trick called the Australian Magician's Dream. Computer programmers really are wizards!

The Australian Magician's Dream

In this magic trick the magician is able to predict a revealed card that no one could possibly have known in advance.

Before doing the trick, take an ordinary shuffled pack of cards and place the 8 of Hearts in the 16th position from the top. Place a distinctive card (e.g., the Ace of Hearts) in the 32nd position. Place them face down on the table, so the 8 of Hearts should now be 16th from the top. Next take the 8 of Hearts from a second pack (ideally from an over-sized pack for extra effect) and place it in a sealed envelope under the magician's table where it will remain in full sight throughout the trick.

Get a volunteer from the audience to the front. Spread the cards in a line across the table face up so they can see it is a normal shuffled pack. Announce that first you need to roughly halve the pack. Spread your hands, to show what you mean, asking them to point to a card of their choice roughly in the middle. Secretly though make sure your hands are over the 16th and the 32nd card. This casually but importantly limits the spectator's choice when dividing the pack to a point between those two positions. Discard all the cards to the right of the one they point to, confirming with them that it was their free choice. Pick up the remaining cards and holding them face down explain that the night before you do magic shows you have weird dreams where magicians teach you new tricks. You had a dream last night where an Australian magician came to you and taught you the "Down-Under Deal": a way to predict a card that no one could possibly know in advance.

Now deal out the cards placing them into two piles in turn. While doing this say, "Down" as you put cards face down in the first pile. Say "Under" as you put cards face up in the second pile. Once all the cards have been dealt, discard the "Down" pile, noting you *always* throw away the down pile. Pick up the 'under' pile, turning them face down, and repeat the process. Continue to do this until you are left with one card on the table in the face up 'under' pile. It will be the 8 of Hearts. Say this is the selected card. Get the volunteer to confirm they had a free choice in where they split the pack. Turn the top few cards of the discard pile over to

show “had you split the pack one card differently it would have been a different card”. Have them confirm they had no idea of what the resulting card would be and get them to show it to the audience saying what card it is.

Now point out that the weird thing is that in your dream the Australian magician told you to place one particular card in an envelope. Ask the volunteer to look “Down Under the table” and reveal the card that was predicted. It is also the 8 of Hearts!

Thank the volunteer and ask the audience to give them a round of applause for helping.

Tricky Algorithms

What does a magic trick have to do with computing? Well, this kind of trick – which a magician would call a self-working trick – is exactly what a computer scientist calls an ‘algorithm’. It’s a series of instructions that if followed in the given order always results in a specific effect. In this case the effect is the magical effect that the card is the one predicted. Computer programs are just algorithms written in a language that a computer, rather than a human magician, can follow.

The algorithm behind the Australian Magician’s dream is essentially:

1. Place the chosen card in position 16
2. Discard roughly the bottom half of the pack
3. Repeat 4 times:
    Repeat until no cards left:
        Discard a card
        Keep a card
4. Reveal the card is the one predicted.

The steps in this particular algorithm aren’t just a sequence of steps to be followed one after the other. They include a **loop**: “Repeat 4 times”. A loop is just a way of writing that some instructions are to be repeated, to avoid having to write out the same thing lots of times. That is exactly the kind of instruction used by programmers in computer programs to tell a computer to repeat some instruction. There is actually a second loop: “Repeat until no cards left”. It is the thing done 4 times over by the other loop. Each of those 4 times you go through the pack until no cards are left, keeping then discarding cards.

Does the trick always work?

This trick works because if you repeatedly discard every second card you are guaranteed to end up with the 16th card. Do you trust me enough when I say that, that you are now willing to do the trick live? Or would you like some evidence? Science is all about not just trusting claims people make but asking for the hard evidence!

How can we be sure that it really does work? We can try it over and over again. If it works every time that gives us some confidence it really does work. Programmers call this `testing’. The more tests, the more confidence you have. But how can we be sure that the next time, the time we do it for an audience, won’t be the one time it doesn’t work? Could we test every possibility? Well that would mean trying it with every possible order a pack of cards could be in when we start the trick. For each of those orders, we would have to check it worked for

all the places the pack might be split. That is far too many possibilities to check than is practical.

We don't have to do all those tests though if we use a bit of **logical reasoning**. The first thing to notice is that the values of the other cards don't matter at all. They could all be blank and it would not change what happens in the trick. We can just think about them based on their position rather than their value. In doing this we are creating a '**model**' of the pack of cards. This kind of modeling is an important part of computational thinking. Rather than describing the pack exactly, we are doing something called '**abstraction**': hiding some of the detail of the problem (the values of the cards) to make it easier. That narrows down the testing we have to do. We just need to check it works wherever we split the pack. Will we always end up with the 16th card? There are only 52 positions we could split the pack, so we now know we could get away with only testing those 52 cases and checking in every case the 16th card is the one left.

Programmers face a similar problem testing programs. They can't test all the things people could possibly do using their program. They therefore use logical reasoning to build a 'test plan': a set of tests that if passed will give a strong (if not perfect) assurance that the program works as it should.

52 times is still a lot of times to do the trick to be sure it works, though. Let's do some more reasoning. We can make a simplified picture of the pack and look at what happens as we throw away every second card. We represent each card by it's position in the pack at the start. This is our model of the pack:

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

What are we left with if we discard every second card starting from the first? Let's cross them out.

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

Just the even numbered positions are left and that means the 16th card is left in the pack:

2 4 6 8 10 12 14 16 18 ...

We can cross out every second card

~~2~~ 4 ~~6~~ 8 ~~10~~ 12 ~~14~~ 16 ~~18~~ ...

leaving

~~4~~ 8 ~~12~~ 16 ...

and then

~~8~~ 16

and finally

16

There is a problem though. If we had split the pack before the 16th card at the start then the 16th card would not have been left as it would of course already have been removed in that first cut. Then we would have ended up with the 8th card (if we'd cut above it).

Similarly if at the start of the down-under part we had more than 32 cards in the pack then by the time we got to the last stages we would have had the following cards left:

~~8~~ 16 ~~24~~ 32

and then

~~16~~ 32

and finally

32

With 32 or more cards at the start we would end up with the 32nd card not the 16th. Therefore we have to add a proviso. For our trick to work, our logical argument shows the cut must be made after the 16th card and before the 32nd. That's why it's important that 'roughly half' the pack is discarded, and why you spread your hands between the 16th and 32nd card when the volunteer chooses

We have used **computational modeling** to show that the trick does work: but only if there are at least 16 and not more than 31 cards in the pile when you start the deal. Computational modeling is just a process of making an abstract model of a computational process to explore it.

Punch Cards

In fact this trick has a deeper link to computing algorithms. A version of the trick's algorithm is actually the basis of a way early computers could search through data stored on punch cards. Punch cards are physical cards that were used as long-term memory for early computers – a place where data to be processed could be saved. They were also used to store programs.

Information was put on a punch card by punching holes into it in special patterns: using a code a bit like a spy's code. Spies might use arcane symbols for their codes, computers used a code of holes and no holes. Unlike a spy's secret code, with computer codes the idea is that the meaning of the symbols is well known. The code computers still use today for numbers is called binary.

To see how punch cards can be used to search for data using the 'magic' down under deal, first you need to make yourself a set of cards. You can download templates to print out from www.teachinglondoncomputing.org. Print them out, ideally directly on to thin card, or by sticking them onto card, though paper should still work. Rather than use holes and no-holes for our code we will use holes and notches. Cut out the notches and holes as shown by the dotted lines. Sprinkling a little talcum powder on the cards can stop them sticking together.

Touching bases

Let's see how you can use the Down-Under algorithm to pull any card quickly

from the pile. Take your pile of punch cards and mix them up so they are shuffled. To use the algorithm you first need to know the binary code. It is actually just a way to write numbers, but where you are only allowed to use the digits 0 and 1 rather than all the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 as we normally do. We use what is called base 10. The base tells us how many different digits – different symbols – that we have available. Binary is base 2: we just have 2 digits available. On our punch cards we are going to use holes for 1 and slots for 0.

Let's look at base 10 first so we can compare it to binary. In base 10 we use the digits to count up to 9 but then run out of digits so at that point have to use a new column. We go back to 0 but carry 1 in to the next column where that one now stands for ten:

0
1
2
…
9
10 At this point the 9 becomes 0 and we put a 1 in the 'tens' column.
11
…

Any digit in the second column stands for 10 times as much as the same digit in the first column. In Base 10 the number 16 is 1 lot of 10 (a 1 in the 10s column) and then 6 units. We add 10 and 6 to get the number 16. Similarly, 987 means 9 lots of 100, 8 lots of 10, and 7 1s are added together.

	100	10	1
x	9	8	7
=	900 +	80 +	7 = 987

Binary

Binary works in just the same way except that we run out of digits and have to use a new column when we get to 1, rather than going all the way up to 9. Instead of 1s, 10, and 100s, that means the columns stand for 1s, 2s, 4s, 8s and so on.

0
1
10 At this point the 1 becomes 0 and we add a 1 in the 'twos' column.
11
…

Joke: There are 10 kinds of people: those who understand binary and those who don't!

This means we write the number 5 in binary (so using only digits 1 and 0) as 101. It is 1 lot of 4 plus 0 lots of 2 plus 1 unit.

5 in Binary (where we show 5 columns) is 00101.

	16	8	4	2	1	
x	0	0	1	0	1	
=	0 +	0 +	4 +	0 +	1	= 5

Similarly, 16 in Binary is 10000.

	16	8	4	2	1	
x	1	0	0	0	0	
=	16 +	0 +	0 +	0 +	0	= 16

Binary punch cards

What does this have to do with our punch cards? Well we can store binary numbers on cards using holes for 0 and slots for 1. To put the number 5 onto a punch card, starting from the left, we need a hole (0), then another hole (0), then a slot (1), a hole (0) and finally a slot (1). For the number 16 we need a slot then the rest holes. With space for 5 holes we can store any number up to 31 on a card. With enough space we could store any number like this.

Once we've stored the number of a card on to it in binary as holes and slots we can then easily find any individual card we want. That is where the Down-under deal comes in. Take the stack of cards and make sure they are all the same way round with the cut off corner in the same place and holes lined up. Now put a pencil through the rightmost hole (the units column), shake out all the cards with a slot in that position. All those with a 1 in the binary at that position fall out, leaving only those with a 0. Now go back to the binary number of the card you are trying to find. If there is a 0 in its binary at that position then *discard* the 'down' pile – the cards that shake out. If there is a 1 in that position of the binary in the target number then *keep* the down pile. Do the same for each hole in turn.

Let's look at an example: finding card 16. In binary it is 10000. From the right that becomes

0: DISCARD those that fall
0: DISCARD those that fall
0: DISCARD those that fall
0: DISCARD those that fall
1: KEEP those that fall.

Repeatedly discard the down pile until on the 5th round you keep the down pile. It will be card 16. It's actually possible to find any card this way, just by spelling out the binary like that. Try and find card 5. In binary it is 00101. That becomes

1: KEEP those that fall
0: DISCARD those that fall
1: KEEP those that fall
0: DISCARD those that fall
0: DISCARD those that fall

You will be left with card 5.

So, how does that work?

It turns out that what is happening as you shake out the cards, is exactly the same thing as in the down under deal. To see this we need a little bit more **logical reasoning** and devise a rigorous argument about what's happening.

Take the first round discarding cards when looking for the number 16. Shaking

out those first punch cards and then getting rid of them throws away all cards that have a slot (a 1) in the first position of the binary number. That is the unit column. The numbers 1, 3, 5, 7 and so on have a slot (a 1) in that position – it's all the odd numbers. It's the same as in the down-under deal where we throw away alternate cards. But why does the binary version lead to the odd ones being thrown out?

Think about the way that we count in binary: 00, 01, 10, 11, …That unit column flips every second number as we count. That last unit position in the number counts 0, 1, 0, 1 and so on. It's actually no different to base 10 except that we have fewer digits before we go back to 0. In base 10 the units column counts 0, 1,… 9, 0, 1,…9,…

To think of it another way, you work out what a binary number stands for by adding up the contribution to the number of the separate digits (like 5= 4 + 0 + 1). That last unit digit is the only way to make odd numbers as all the other digits represent even numbers (2, 4, 8, 16, …)!

So we have shown that in round 1 the same thing happens as in the trick. Having taken out all the odd numbered cards we move on to the next hole on the punch cards and so the next position in the binary number. That shakes out all the numbers that include 2 in the addition making up the number. For example 6 is one of them. It is 110 in binary because 6 = 4+2+0. The numbers dropping out this time are 2 (10 in binary), 3 (11 in binary), 6 (110), 7 (111), 10 (1010 in binary), 11 (1011 in binary) and so on. The odd numbers have already been removed though, so the ones left that are shaken out this time are 2, 6, 10, … That is every second card that is left. It is the same sequence of cards as are removed in the second round of the down-under deal.

We can see why again if we think of the way the binary counting system works, the second column counts 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1 … You can see this pattern in the middle column of the following sequence for three digit binary numbers.

0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

It happens because in binary, the second digit only changes when the first binary digit has counted twice (0, 1) itself. Now if we have already removed every second card of that sequence we are left with not 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1 … but 0, 1, 0, 1, 0, 1, … That leaves us with:

0	0 0 0
2	0 1 0
4	1 0 0
6	1 1 0

This means that, given the cards left we are actually doing the same as in the first

round discarding every second card because the middle 1s are every second card in the above sequence.

The same thing happens on every round of removing cards. We are actually shaking out every second card that is left every time. The difference to the trick is that the numbers on the punch card don't refer to the card's position but to its binary hole and slot label. That means they can be shuffled and we still find the card. Another difference is that with the punch cards all the cards are removed in one go – in parallel. Whereas the down-under deal was very slow and boring as we went through every card in turn, the punch card version is very fast.

The card trick uses a '**sequential'** algorithm: we do one thing at a time, move one card at a time. Most computer programs are written like that – their instructions are followed one after the other. The punch card searching is an example of a '**parallel**' algorithm. Rather than doing one thing at a time, on some steps at least, we do lots of things at once, shaking out lots of cards. That means that while the deal is really slow, the punch card version is fast.

Overall, the punch card algorithm is fast because it uses what is known as a **divide and conquer** approach to solve the problem of finding a card. Divide and conquer is a general way of solving problems really quickly. The secret behind divide and conquer is that it repeatedly halves the size of the problem. What does that mean for our cards? On each round we remove half of the cards that remain. How do we search those left? We do the same again, removing half, and then half again, and so on.

There are other ways we could search for a punch card. The simplest is to check each card in turn to see if it is the one we are looking for. That is an algorithm called **linear search**. Our divide and conquer algorithm is much faster, precisely because it cuts the size of the problem in half on every step.

Suppose there were a million punch cards to search. How long would it take to find one particular one by checking each in turn? At best (very unlikely) you might find it immediately. At worst you would have to check all million. On average you would expect to do about half a million checks. That is not going to be quick! How much better does our divide and conquer, halving strategy, do? Well, after one round, we halve the pile, so only have 500,000 cards left. After two rounds of shaking out cards, 250,000 are left, then 125,000 cards left, then less than 64,000 cards (simplifying a little to make the numbers easier!), 32,000 cards, 16,000, 8000, 4000, 2000. After 10 rounds of shaking out half the cards there are only 1000 cards left, out of the original million, that it could still be. If we keep doing the same thing then 500 left after another round, 250, 125, less than 64, 32, 16, 8, 4, only 2 cards left in the pile, and on the 20th round there is only a single card left. It is the one you are looking for. So rather than checking half a million cards on average and a million at worst, with the punch card searching algorithm, you will be holding the card you were looking for after only 20 rounds of shaking out cards. Guaranteed!

That is the power of coming up with a good algorithm! If you can find a divide and conquer algorithm to solve a problem, you are on to a winner!

Inventing new magic

We have seen that the magic trick and a method for searching for data actually use the same underlying algorithm. The solution to one problem (entertaining an audience with a magical effect) is a solution to an apparently different problem

(finding a card in a pile of many). This is an illustration of an important part of **computational thinking**, the set of skills computer scientists develop: that of **transforming solutions between problem areas**. Someone who knew the trick could use it as the basis of solving a search problem. To translate problems like this, you need the skill of being able to **see patterns**. You have to be able to see the similarities in two apparently different things. In our example, it is about being able to see that what you are trying to do in the magic trick is actually the same as you are trying to do with the punch cards.

We can actually use the understanding we have gained from the search algorithm to come up with new versions of the magic trick too, using exactly this computational thinking skill. We've seen that we can find any punch card by discarding and keeping the cards shaken out. Perhaps, we could develop a variation of the trick based on this observation where we end up predicting a card that was originally placed at any position in the pack, not just the 16th. Try it! Computational thinking leads to our creating a new magic trick. In fact, several magic tricks have been inspired by algorithms in this way.

Sometimes you can do better than just use an old solution to solve your latest problem. Sometimes you can take a solution and codify it in a way that gives you a general way to solve lots of similar problems. That is really what the idea of a search algorithm is about. Once you have solved one search problem, you can **generalise** the solution as a search algorithm. In the general version it doesn't talk about playing cards or punch cards but just refers to the thing you are searching for. Then you can use it to solve any problem that turns out to be a search problem. This kind of **generalisation** is another computational thinking skill.

So we've seen that self-working tricks and computer programs are the same thing. People who invent new tricks are doing the same thing as those writing new programs: programmers really are wizards!

Use of this material

This booklet was distributed with support from the Mayor of London and Department for Education.

Department for Education

SUPPORTED BY MAYOR OF LONDON

COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE

BY NC SA