PROGRESS OUTCOME 6

Linear versus binary searches

Context

Sharee has been investigating the impact of using different algorithms for searching and sorting data. She has experimented with two algorithms to determine which one works better for searching different data sets.

Insight 1: Applications of searching

Most people don't realise how much they rely on computers use of fast and accurate search algorithms for all kinds of information. So I researched and brainstormed the different types of everyday computer searches. We take it for granted that a web search will provide instant results and that our groceries will scan quickly at the supermarket checkout. In medicine, doctors and researchers need to search databases of diseases or genetic information. The police need quick access to information about offenders and public safety. Because of the computer's ability to quickly search large amounts of data for answers to problems, artificial intelligence is developing rapidly.

Insight 2: Different algorithms for searching data

During my research, I discovered many different search algorithms. Some are kept confidential, like the Google™ search algorithm, to maintain their competitive edge over other businesses. However, linear and binary searches are two basic search algorithms that most computer science students are familiar with.

To understand how they work, I tried each one with a friend. First, I printed out fifty random numbers and put them face down in random order. I then asked my friend to find a specific number using a linear search. That meant he had to start with the first number and look at each one in order until he found the number. We worked out that it could take fifty attempts to find the correct number, if it was the last one.

I then put the numbers in numerical order and asked him to find the number using a binary search by cutting the data set in half each time. He started in the middle and identified whether the specific number was higher or lower than the middle number and therefore which half of the data set it was in. We worked out that using this approach would take up to six attempts to find the number, because after cutting the data set in half six times you have only one piece of data left.

Insight 3: Determining best, average and worst cases

After doing the experiment with my friend, I used a spreadsheet to look at the best, average, and worst cases when searching data using these two algorithms. I also researched this on the internet to see if my conclusion was correct.

I learnt that the worst case for a linear search is that you must look through every piece of data, whereas a binary search cuts the data in half each time, so at worst you have to look through the data as many times as you can cut it in half (i.e., until there's one item left).

On average, that means it's much quicker for a computer to perform a binary search.
For example, if I have 1024 items of data to search through, it will take me at most 10 searches:

1) 1024/2 = 512

2) 512/2 = 256

3) 256/2 = 128

...

9) 4/2 = 2

10) 2/2 = 1

From my research, I learnt that this is a log base 2 function, so I used the log base 2 function in my spreadsheet to work out the worst case. I also read that the average and the worst case for a binary search are just about the same.

I tested this by downloading and running a binary search program 20 times for each data set and put the results into a spreadsheet. I saw that the average and worst cases were almost exactly the same. So I used the same spreadsheet formula for the average as for the worst case scenario (log base 2).

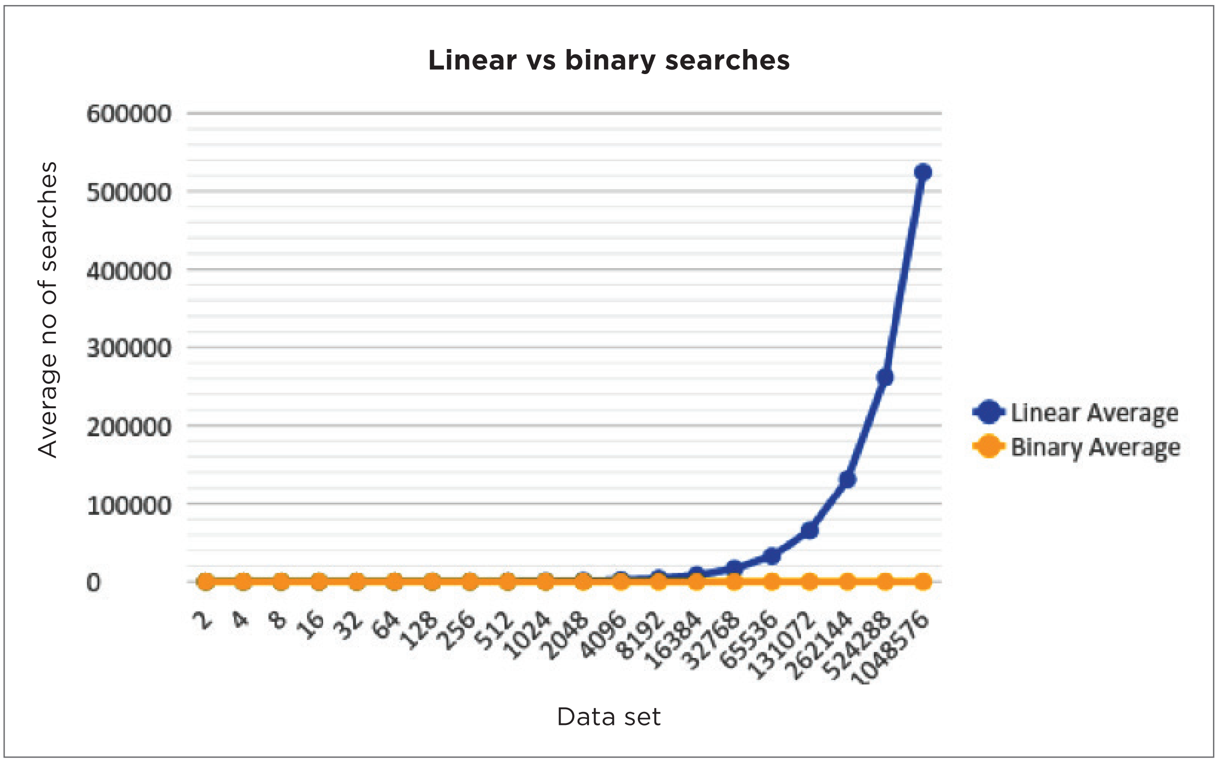
The table below shows my conclusions:

Linear Search			
Data set	Best	Worst	Average
2	1	1	1
4	1	3	2
8	1	7	4
16	1	15	8
32	1	31	16
64	1	63	32
128	1	127	64
256	1	255	128
512	1	511	256
1024	1	1023	512

Binary Search			
Data set	Best	Worst	Average
2	1	1	1
4	1	2	2
8	1	3	3
16	1	4	4
32	1	5	5
64	1	6	6
128	1	7	7
256	1	8	8
512	1	9	9
1024	1	10	10

Insight 4: Cost changes as problem size increases

I then added further data sets to the spreadsheet and made a graph comparing the two searches. I concluded that using a linear search with large amounts of data is much slower than a binary search because each time the data set doubles, the average number of searches for a linear search doubles, whereas it only adds one more search for a binary search.



Downloaded from http://technology.tki.org.nz or http://seniorsecondary.tki.org.nz/Technology/Digital-technologies
Google® is a registered trademark of Google Inc.
Copyright © Ministry of Education 2018, except for student work copyright © student
ISBN: 978-1-77669-236-1

MINISTRY OF EDUCATION
TE TĀHUHU O TE MĀTAURANGA

PROGRESS OUTCOME 6

Why compression matters

Context

Sarah has been investigating the concept of compression coding, by looking at the different ways images can be compressed and the trade-offs of using different methods of compression in relation to the size and quality of images. She researches the topic by doing several web searches and produces a report with her findings.

Insight 1: Reasons for compressing files

I realised that data compression is extremely useful as it reduces the amount of space needed to store files. Computers and storage devices have limited space, so using smaller files allows you to store more files. Compressed files also download more quickly, which saves time – and money, because you pay less for electricity and bandwidth. Compression is also important in terms of HCI (human-computer interaction), because if images or videos are not compressed they take longer to download and, rather than waiting, many users will move on to another website.

Insight 2: Representing images in an uncompressed form

Most people don't realise that a computer image is made up of tiny dots of colour, called pixels (short for "picture elements"). Each pixel has components of red, green and blue light and is represented by a binary number. The more bits used to represent each component, the greater the depth of colour in your image. For example, if red is represented by 8 bits, green by 8 bits and blue by 8 bits, 16,777,216 different colours can be represented, as shown below:

8 bits corresponds to 256 possible numbers in binary

red x green x blue = 256 x 256 x 256 = 16,777,216 possible colour combinations

Insight 3: Lossy compression and human perception

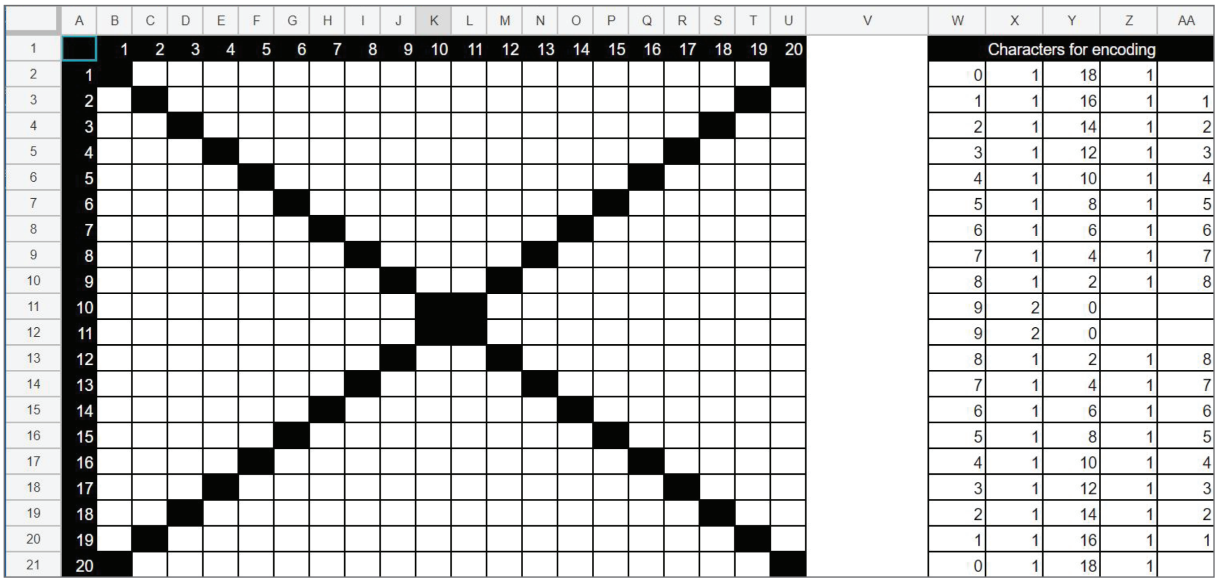
There are many different formats for file compression such as JPEG (image compression), MP3 (audio compression) and MPEG (audio and video compression). These formats use lossy compression, in which some of the original data is removed when the file is compressed. The benefit is a smaller file that downloads more quickly, but it does mean that you cannot convert the file back to its original full quality.

We use lossy compression because most image and sound files have more data than the human eye or ear can perceive and if data is removed it isn't usually noticeable. If we compress a file too much, people will start to notice – for example, an image may look pixelated or blurry. Sometimes it's not appropriate to compress an image because the quality is extremely important, such as in a medical scan.

Insight 4: Lossless compression

Lossless compression compresses a file without removing any data. Instead a computer uses an algorithm to map the repeating patterns or information in the file (such as colours for an image or letters in a document). Most people use zip compression for general files. For images, lossless compression methods include PNG (Portable Network Graphics) and RLE (Run Length Encoding). Because lossless compression works by storing a map of repeating patterns of data, it is most effective when there is a lot of repeated data.

Below is an example of how lossless compression could work for a simple black and white image using RLE. The image on the left could be exactly replicated by a computer using the encoding map on the right. The computer has all the information it needs to recreate the image pixel by pixel (how many bits are black and how many bits are white).



Characters for encoding				
0	1	18	1	
1	1	16	1	1
2	1	14	1	2
3	1	12	1	3
4	1	10	1	4
5	1	8	1	5
6	1	6	1	6
7	1	4	1	7
8	1	2	1	8
9	2	0		
9	2	0		
8	1	2	1	8
7	1	4	1	7
6	1	6	1	6
5	1	8	1	5
4	1	10	1	4
3	1	12	1	3
2	1	14	1	2
1	1	16	1	1
0	1	18	1	

Downloaded from http://technology.tki.org.nz or http://seniorsecondary.tki.org.nz/Technology/Digital-technologies
Copyright © Ministry of Education 2018, except for student work copyright © student
ISBN: 978-1-77669-237-8

MINISTRY OF EDUCATION
TE TĀHUHU O TE MĀTAURANGA

PROGRESS OUTCOME 6

Designing a virtual golf game

Context

Physical education teacher Ms Henare has asked her year 11 students to create a computer game to help students work on a particular sports skill.

In consultation with Ms Henare and some year 9 students, Leilani has decided that a virtual golf game will be fun for students to use and help them with their golf swing.

Insight 1: Planning

I decided to use Scratch to develop the game. First I sketched out a plan using a flowchart, which outlined the problem and the specific tasks the program should perform. My aim was to design a game that is engaging, performs the tasks, functions as intended and is easy for the user to play.

My plan included creating a pseudocode that sets out the algorithm, breaking down the tasks into their necessary steps and identifying the sprites (characters) and components I would need to create for the game.

Insight 2: Setting out and commenting on the program code

I discussed my pseudocode with Ms Henare and began to develop the images, sprites and program code, as per my flowchart.

I had to plan how to make the game become increasingly more difficult (for example, by adding different levels). I also wanted to use a timer and keep a score (which would be a variable). I thought about the blocks I could use, such as actions to control the sprite movement (hitting the arrow keys and spacebar), and the variable for keeping the score. I sketched out my ideas and the bits of code and discussed these with Ms Henare, who was happy with my progress.

Next I started to develop the program. I annotated my code as I went, using the comment feature as I worked through the steps – the comments explain my design and implementation decisions. I also identified my variables and their names. This is a good way of reminding myself what the code is supposed to do (its function and behaviour). It's also useful if you have to explain the code to another coder and your reasons for programming it that way.

Insight 3: Testing and debugging the program

I worked through my blocks, testing the project to see if it ran as it should. I found a few errors, such as when you clicked the green flag, an animation of a golf club should have swung back and forth once, then stop. But it didn't stop – instead, the animation started all over again. I had to work through the blocks to figure out why this happened. I fixed the error and added a comment to the block for future reference.

Insight 4: Meeting end-user requirements

I asked a group of year 9 students to test the game for me. After discussing their feedback, we decided I needed to add another level for the more sophisticated golfer, as the game was too easy for some users. One error they found was that the "# of hits" display increased by more than 1 each time the golf club hit the ball. I had to work through my blocks to figure out how to fix this.

Insight 5: Testing the final outcome

After I'd finished my debugging, I asked Ms Henare and a group of year 9 students to test the game again. It functioned as intended, and they were happy with the outcome.

Downloaded from http://technology.tki.org.nz or http://seniorsecondary.tki.org.nz/Technology/Digital-technologies
Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab (http://scratch.mit.edu).
Copyright © Ministry of Education 2018, except for student work copyright © student
The program used in this exemplar is not officially endorsed by the Ministry of Education.
ISBN: 978-1-77669-238-5

MINISTRY OF EDUCATION
TE TĀHUHU O TE MĀTAURANGA