CHAPTER 2

ALGORITHMS

WHAT IS AN ALGORITHM?

We can't talk about AI or machine learning without talking about algorithms.

This is because algorithms are the basic building blocks of AI. In fact, algorithms are the building blocks of computer programs in general. But even more than that, they're the building blocks of how many of us live our lives.

When you get ready in the morning, what do you do? Maybe you have a set routine: you might wake up, take a shower, brush your teeth, then eat breakfast. These are the steps you take so that you can be ready for the day ahead of you.

This is what an algorithm is: a series of steps that allow you to perform a particular task.

In this case, the task is getting yourself ready for the day. But the task an algorithm solves can be much simpler or much more complicated.

As a result, most of us don't even realize how much of our lives are organized by algorithms.

Every algorithm takes in defined inputs (the things being acted upon) and has the goal of producing defined outputs (the results you want). For example, maybe you have a set algorithm for making yourself a sandwich for lunch. Your goal, or output, might be to make a sandwich that will fill you up. The inputs are all of the ingredients that will go into the sandwich, and your algorithm is how you order and arrange these things so that you can get your desired result, or output, of a sandwich for lunch.

Cleaning your room could be another example of a personal algorithm. The goal, or output, is a clean room, and the input is your room as it currently is, containing the items in the room that you will manipulate in some way. You might have a specific order for how you clean your room. Maybe you start by sweeping the floor, then you clean the windows, then you make the bed, then dust the dresser, and then your room is clean. This is your room-cleaning algorithm, or the steps you take to get to the desired output of a clean room.

But let's say you find that by sweeping the floor before dusting, you end up sweeping the floor twice because it gets dirty from the dust. Someone else who has their own room cleaning algorithm might

suggest doing things in a different way. They may start by making the bed, then cleaning the windows, dusting the counters, then sweeping the floor. And they might argue that their algorithm is more efficient because it's faster, since they only have to sweep the floor once.

This gives us a couple of key points about algorithms:

1. Different algorithms can exist for accomplishing the same task.

2. Algorithms are often judged by their efficiency, and efficiency can be evaluated in different ways. For example, you could judge the efficiency of an algorithm by how long it takes to do a task, how well it does a task, how much energy it uses to do a task, and so on.

Take a second to think about your algorithm for cleaning your room. What are the steps you follow?

TASK: ***Clean your room***

INPUT(S): ***Your room in its current state***

OUTPUT: ***A clean room***

Steps for algorithm:

1. _____

2. _____

3. _____

4. _____

5. _____

COMPUTER ALGORITHMS

So far we've been talking about personal algorithms. But the main space where you are likely to hear about algorithms is in relation to computers. The programs that computers run are full of algorithms. Like the personal algorithms that humans use, these computational algorithms exist to solve problems or perform tasks.

But the big difference between the algorithms that computers use and the informal ones that we use is that we can't describe our problems to computers in the same way that we describe them to ourselves. Instead, we have to be more direct and explicit with a computer about the problems we want it to solve.

There's a simple reason for this. Computers exist to make our lives easier. They tend to be very good at doing things that we're not good at (think how much more quickly you can do math on your cell phone, which is a just a tiny computer, than you can in your head). But computers tend to be pretty bad at the things that we're good at, like understanding context and nuance.

What this means is that if you tell a computer to sort a huge list of numbers from smallest to largest or to compute the shortest possible distance between two places, it can do the task easily. But if you try to ask a computer to clean your room, it won't know what you mean, because it doesn't know what it means to "clean", the equipment needed to "clean" or even what a room is.

In other words, computer algorithms have to be specific and clear. You can't tell a computer to clean your room, but you can tell a device programmed by a computer to pick up an object that is close to the ground and place it on the table (but keep in mind that you'll also have to tell it that a table is a flat surface that is above the ground, since it won't know what that is either).

Bubble Sort is the name of a common algorithm used to sort a list of numbers from smallest to largest. It starts at the beginning of a list, compares each pair of numbers in the list and swaps them if they are in the incorrect order, then repeats once it's reached the end of the list until the numbers are all sorted correctly. Here's what it would look like using our personal algorithm format:

TASK: ***Sort a list of numbers from smallest to largest***

INPUT(S): ***List of numbers: [7, 1, 40,_3, 5]***

OUTPUT: ***Sorted list of numbers***

Steps for Algorithm:

1. Compare the first element (7) with the second element (1).
2. 7 is larger than 1, so swap the order of the two. If not, do nothing.
3. Get a new list of **[1, 7, 40, 3, 5].**

4. Now compare the second element of the new list (7) with the third element of the new list **(40).**

5. If 7 is larger than 40, swap their order. If not, do nothing.

6. Get new list of **[1, 7, 40, 3, 5].**

7. Now compare the third element of the new list **(40)** with the fourth element of the new list **(3)**.

8. If 40 is larger than 3, swap their order. If not, do nothing.

9. Get new list of **[1, 7, 3, 40, 5].**

10. Compare the fourth element of the new list **(40)** with the fifth element of the new list **(5)**.

11. If 40 is larger than 5, swap their order. If not, do nothing.

12. Get new list of **[1, 7, 3, 5, 40].**

13. There are no more elements within the list. The sorted list is **[1, 7, 3, 5, 40]**. We can see that the last element is in the right place, but we still need to sort the others. From here, we start again from the beginning .

14. Compare the first element **(1)** with the second element **(7).**

15. If 1 is larger than 7, swap their order. If not, do nothing.

16. Get new list of **[1, 7, 3, 5, 40].**

17. Repeat until the entire list is sorted.

This is what computational thinking looks like. It involves breaking down a problem to its most basic parts and thinking sequentially to solve that problem.

Ordering a list is a perfect example of a problem that can be easily solved by a computational algorithm. But there are many problems which we will talk about later in the booklet that can't be solved as easily in the same way.

EMBODYING SOCIAL ALGORITHMS

Use the chart below to think through how algorithms show up in your everyday life.

DESCRIBE A PERSONAL ALGORITHM THAT YOU USE IN YOUR LIFE.	DESCRIBE ANOTHER ALGORITHM THAT YOU COULD USE TO DO THE SAME TASK.	WHICH OF THE TWO ALGORITHMS IS THE MOST EFFICIENT? HOW ARE YOU EVALUATING ITS EFFICIENCY?