Writing Algorithms

An algorithm is a set of instructions that is completed in a step-by-step way to solve a particular problem. But how do you go about writing one from scratch? There are different approaches to this. Some people like to write pseudocode, English-like syntax that resembles code, to explain the problem in a series of steps. Another approach is to use a flow chart that provides a graphical representation of the series of steps. The flow chart goes through the logical flow of the algorithm and shows the different decisions that need to be made in order to solve it.

Say you want to create an algorithm to determine how many food order tickets are in the queue to the kitchen on the rail board in a restaurant. If you had to do it without a computer you would just count the number of slips and get the total number of tickets. In code, it can be quite similar. Let's create some pseudocode to represent this.

```
let T = 0

for each ticket on rail
    Set T = T + 1

Return T
```

The pseudocode starts at 0 and then it checks to see how many tickets are on the rail. If a ticket is found it will then increase the counter by 1. And finally, it will return the total count.

One aspect of writing an algorithm is how efficient it is. This is referred to as optimizing the code. If I want to optimize the code above I need to look at how I can make it get to the answer faster. In the physical world, I could instead of counting one by one, count two tickets at a time. How can I represent this in my pseudocode? I just have to change the increment from 1 to 2.

```
let T = 0

for each pair of ticket on rail
    Set T = T + 2

Return T
```

This is a pretty easy optimization, but will it work in all scenarios? What happens, for example, if there are three orders on the rail? The code would work as follows:

Tickets = 0

Each pair = 1

Increment counter by 2

Return 2

This code is buggy. It does not account for the single ticket on the rail and only returns 2. I can fix the code by adding a condition to handle this edge case.

```
let T = 0

for each pair of ticket on rail
    Set T = T + 2

if 1 ticket remains then
    Set T = T + 1

Return T
```

Let's check the code to see if it meets the requirements:

Tickets = 0

Each pair = 1

Increment counter by 2

Check for any single tickets

Increment counter by 1

Return 3

Yes! The code works and it can handle the edge case. Algorithms are designed to solve problems but they should also be efficient. I needed to change my code to make sure it works as expected.

There are many different types of algorithms that have been designed to solve all kinds of different types of problems in computer science. When writing an algorithm, it can be solved in many different ways and each can have its own pros and cons.

Recursion

Recursion refers to a method or a function that will call itself. It is used to resolve problems by breaking the problem down into sub-problems. Let us take a look at some of the most popular types of recursive algorithms.

Divide and conquer

This consists of two parts. The first is breaking the problem down into smaller sub-problems and the second is solving the final solution.

Dynamic programming

This is mainly used for optimization problems. It is similar to the divide and conquer algorithm in that it splits the problems into sub-problems.

Greedy algorithm

This one finds the best solution in each and every step instead of approaching optimization in a global way.