Algorithm

1 Problems vs Instances

Algorithms solve problems and not instances of a problem. Lets say we have to sort a list of n integers.

```
L = [7,8,2,3,8]
n = 5
L = [2,3,7,8,8]
```

Sorting this list is easy since the number of elements in the list is so small. We can call this an instance of our problem. Reason why this is only an instance is because it's easy to solve as you don't need to think to much about it. However, what if we had a list containing 20,000 elements? That would be much harder and would require a lot more thought to solve. Remember we want to solve the actual problem of sorting not just a simple instance of the problem. Thats where algorithms come in! We would want our algorithm to be able to sort either 5 elements of 1,000,000 elements taking care of any instances of our problem.

2 Ways We Look at Algorithms

There are about three ways that we will be looking at algorithms.

Pseudo Code

Pseudo Code is a simple way for us to express our algorithms. In the book they will state words such as "keytypes".

```
"keytype" is the same as "i" such as L[i]
```

The next thing to keep in mind when reading the pseudo code in the book is indexes. Normally indexes start at 0, but in the book they start at 1.

```
Indexes -> start at 1
```

Next is what is called "number". The word number in the book refers to any data in the book like data types. In a programming language like Java you state these types, but in a language like Python you don't have to worry about it!

```
"Number" -> int, double, float, etc
```

Lastly, there is no object oriented stuff. They are just little pieces of code like how we saw when we first started coding!

Java

Java will be uses less frequently, but it is still a valid way for us to look at our algorithms. The reason why would look at it in Java is due to the book algorithms matching the syntax of Java.

```
    Example -> &&, ||, or using {} to mark the beginning and end of code, etc
```

Python

Python will be the last way we look at our algorithms. Python is useful for looking at algorithms as it is much more literal compared to languages such as Java.

```
    Example -> and, or, etc
```

Not much internal translation making it much easier to read.

3 Efficiency

Here we will be talking about effieceny. A question to ask is what exactly makes an algorithm efficient? There are two ways we can look at this.

```
    1. How long does it take a program to run?
```

If we have two algorithms solving a problem and the first algorithm takes 4ms and the second take 6ms. Obviously the first algorithm is faster than the second and we call this overall idea Time Complexity or how complex is this algorithm in relation to time. Now the other way we can look at his is

```
    2. How much memory is used?
```

Now lets take a look at a list

```
    2. How much memory is used?
```

Our first algorithm will take that list and make one copy while our second algorithm will take the list and make multiple copies. Algorithm 1 only has a few copies sitting in memory while algorithm 2 has multiple copies sitting in memory. Our first algorithm we would say is more efficient as it has less sitting in memory. Now we call this idea Space Complexity! So to recap we look at efficiency of algorithms in two ways:

```
    1. Time Complexity
    2. Space Complexity
```