**Data Structures and Algorithms**

**Covered**

**Algorithmic Thinking**

This involves breaking a problem down into a defined set of inputs and outputs along with a set of steps that allows you to convert that input to outputs.

* Identify inputs
* Layout steps
* Create output

**Characteristics of an Algorithm**

* The steps must follow a specific order
* The steps need to be distinct
* It should produce a result
* It should complete within a finite amount of time

**Runtime Complexity**

* Time Complexity: How fast
* Space complexity: How much memory (RAM) does it utilize. Here you want to watch the data structures you use, as well as

Algorithms have best case scenarios, the best possible runtime. Worst case scenarios, the worst possible runtime and average case scenarios, best + worst/2.

**Search Algorithms**

The best case scenario of linear and binary search is constant time O(1). While the worst case scenario of linear is O(n) and O(log n) accordingly.

* Linear Search – O(n) this involves iterating through each item in the list. It’s time complexity increases as n increases.
* Binary Search – This has logarithmic time complexity O(log n). Binary search compares the target value to the middle element of the array in a sorted list.

**Recursion**

This is a programming technique that involves returning a function call in a function until a stopping condition is met.

**Language Specificity**

Different programming languages have various recursion depths and other caveats which make some implementations more ideal than others.

For example, Python has a limited recursion depth. This means that there is a limit to the number of times a function can call itself before its halted. Hence, Python solutions are better of iterative than recursive.