Algorithms: Takeaways

Syntax

- Example of a constant time algorithm:

```
def blastoff(message):
count = 10
for i in range(count):
    print(count - i)
print(message)
```

- Example of a linear time algorithm:

```
def is_empty_1(ls):
if length(ls) == 0:
    return True
else:
    return False
```

Concepts

- An algorithm is a well-defined series of steps for performing a task. Algorithms usually have an input and output, and can either be simple or complicated.
- A linear search algorithm searches for a value in a list by reviewing each item in the list.
- When using more complex algorithms, it's important to make sure the code remains modular. Modular code consists of smaller chunks that we can reuse for other things.
- Abstraction is the idea that someone can use our code to perform in operation without having to worry about how it was written or implemented.
- When choosing from multiple algorithms, a programmer has to decide which algorithm best suits their needs. The most common factor to consider is time complexity. Time complexity is a measurement of how much time an algorithm takes with respect to its input size.

- An algorithm of constant time takes the same amount of time to complete, regardless of input size.
    - An example would be an algorithm to return the first element of a list.
- We refer to the time complexity of an algorithm that has to check `n` elements as linear time.
- Big-O Notation is the most commonly used notation when discussing time complexity. The following are most commonly used when discussing time complexity:
    - Constant time:
    - Linear time:
    - Quadratic:
    - Exponential:
    - Logarithmic:
- An algorithm with lower-order time complexities are more efficient. In other words, an algorithm of constant time is more efficient than linear time algorithms. Similarly, an algorithm which has complexity is more efficient than an algorithm with complexity.

Resources

- Time complexity
- Big-O notation