Data Structures & Algorithms

Computer Science & the Programmer

1. Computer Science is the science of solving problems. The solution to that problem is called an algorithm. That algorithm is a step-by-step list of instructions that lead to a solution.
2. You can drive a car by understanding the break / gas pedals, steering wheel, turn signal, etc. The manufacturer of that car understands how every part works within the car and provides an easy to understand interface for the user.
3. As a programmer you understand the details and also must provide an easy to understand interface.

Why Study Data Structures

1. We will often want to define custom types of data. We will also need to provide an easy way for users to access that data.
2. By studying the most useful data structures we will get better at designing custom versions

Why Study Algorithms

1. By studying the most effective ways of solving problems we will begin to recognize how to utilize those solutions to solve numerous other problems.

Why is One Algorithm Better

1. Which is more readable
2. Which consumes the least amount of memory
3. Execution time

```
import timeit

def get_sum(max_num):
    sol = 0
    for i in range(1, max_num + 1):
        sol += i
    return sol

def get_sum_2(max_num):
    sol = 0
    i = 1
    while i < max_num:
        sol += i
        i += 1
    return sol

def get_sum_3(mn):
    return mn * (mn + 1) / 2

# Use timeit to verify execution time
# Pass the function, how many times to repeat the timer, number of times to
# execute the function and provide access to the functions
print("Testing get_sum")
print(timeit.repeat(stmt='get_sum(100000)', repeat=5, number=1, globals=globals()))
```

```
print("Testing get_sum_2")
print(timeit.repeat(stmt='get_sum_2(100000)', repeat=5, number=1, globals=globals()))

# This is the most efficient algorithm because it doesn't
# contain repeated steps which allows it to perform well
# as the number of values summed dramatically increases
print("Testing get_sum_3")
print(timeit.repeat(stmt='get_sum_3(100000)', repeat=5, number=1, globals=globals()))
```

Simply calculating time is not the best way to judge an algorithms performance because it is largely based on the computer / language using that algorithm.

It is better to quantify performance based on the number of steps the algorithm requires.

Big-O Notation, where the O refers to the Order of Magnitude, is a measure of how well an algorithm scales as the amount of data increases.

How well does it perform as it increases from a 10 element array versus a 10,000 element array.

Let's say you have this algorithm $45n^3 + 20n^2 + 19 = 84$ (if n is 1)

I want to define the part of the algorithm that has the biggest effect during the calculation of the answer.

If n=2 the answer goes from 84 to 459. It doesn't take long until + 19 doesn't matter.

If n=10 the answer is 47,019 and n^2 has little effect on our answer because $45n^3 = 45,000$

Because the n^3 has the greatest effect on our final answer and so we would say this algorithm has an order of n^3 or $O(N^3)$

I'll cover what all of these mean in the video that follows O(1), O(N), $O(N^2)$, O(log N), O(N log N)