Trivial Runtime Analysis

In this lesson, you'll learn how to determine the run-time complexity of a given piece of code through some samples.

We'll cover the following

- Big-O
- Basic loops

Big-O

The goal of code analysis is to determine its run-time complexity with growing input size N.

If there is no input, then it's called a constant time algorithm. For example:

```
for (int i = 0; i < 1000000; i ++)
    x++;
```

How many times does the statement `x++` gets executed? A million times. But it's fixed or constant, so it's always a million operations no matter what. So the time complexity is $O(1)$ (*constant time*).

Some properties of Big-O notation:

- $O(c)$ is $O(1)$, where c is any constant
- $O(cN)$ is $O(N)$
- $O(N^2 + N)$ is $O(N^2)$, we only consider highest order term
- $O(N + c)$ is $O(N)$
- $O(N - c)$ is $O(N)$

> We hide the constant when expressing the algorithm's runtime complexity using Big-O. Sometimes this *hidden constant* becomes relevant as a big hidden constant will affect the actual execution time of the code. This will become

more evident as we learn algorithms that affect and/or are affected by the hidden constants.

Basic loops

Let's go through some code samples and analyze their runtime complexity.

```
for (int i = 0; i < N; i ++)
      x++;
```

All we need to do is count the number of times the statement `x++` will execute. Clearly, it's N, so the time complexity is $O(N), also called **linear**.

```
for (int i = 0; i < N; i++)
    for (int j = 0; j < i; j++)
        x++;
```

How many times the statement `x++` executes:

- When $i = 0$, 0 times
- When $i = 1$, 1 time
- When $i = 2$, 2 times and so on

This turns out to be $0 + 1 + 2 + 3 + .. + N - 1 = \frac{N \times (N-1)}{2}$

So the time complexity is $O(N^2)$, also called **quadratic**.

In the next lesson, we'll analyze logarithmic run-time cases.