0.1 Mathematics review

In this lesson, we will go over some of the basic mathematics concepts that we need in order to tackle data structures. The first in Big-O notation, which is very important in computer science, as well as in programming. Then, I will be talking a bit about induction and recursion. I want to get the following things out of this lesson:

- Have a great understanding of what Big-O is.
- Be able to do Big-O analysis quickly.
- Understand the concept of mathematical induction.
- Have a better understanding of recursion.

0.1.1 Big-O notation

Big-O notation is fundamental in computer science. It is used in order to evaluate the runtime or space complexity of an algorithm. Significant parts of computer science theory were developed before computers even existed. However, computer scientist needed a way to be able to tell how fast algorithms would be running. So they invented Big-O notation. Here is the mathematical definition of Big-O:

> $f(x)$ is $O(g(x))$ if and only if there are positive real numbers c, k such that $0 \leq f(x) \leq cg(x)$, for every $x \geq k$.

What this means is that $f(x)$ will be $O(g(x))$ only if they grow to infinity with the same speed. For example, if $f(x) = 4x$, then $f(x)$ is $O(n)$, because if we let $c >= 4$, then $0 \leq f(x) \leq cg(x)$. All Big-O is saying is: ignore constants, and look at the higher term of a function. For example, the function $h(x) = n^3 + 40942n^2 + \sqrt(n)$ is order $O(n^3)$. Now, when we look at the speed of functions in real life, constants do matter. However, in computer theory we are not so concerned about them, as the order of growth of a function is much more important. Doing Big-O analysis is not hard, it just requires some practice. Lets take a look at some examples:

```
1 int arraySum ( int * array, int size )
2 {
3   int total = 0;
4   for( int i = 0; i < size; i ++ )
5     total += array[i];
```

```
6   return total;
7 }
8
9 int halfArraySum ( int * array, int size )
10 {
11   int total = 0;
12   for( int i = 0; i < size/2; i ++ )
13     total += array[i];
14   return total;
15 }
```

When you want do Big-O analysis, the first thing you want to do, is to figure out what is the variable in the function. In the two functions above, the variable is *size*. In order to sum an array, we need to iterate through every element of the array, so that we can add it to total. So, $arraySum$ requires iterations $i(n) = n$, thus has order $O(n)$. However, also $halfArraySum$ has order $O(n)$, as it requires iterations $i(n) = \frac{n}{2}$. Remember? Constants do not matter. Imagine that we give an infinitely long array to our functions. Then, the two functions will take infinitely the same time. Lets look at some other examples:

```
1 /* ADD MORE EXAMPLES */
```