Chapter 1 - Algorithms

In this chapter we discuss what is an algorithm, time complexity, storage complexity and provide a few examples of algorithms.

What is an algorithm?

Let's first start with a discussion of what an algorithm is. Intuitively the definition is more or less clear, we are talking about some formal way to describe a computational procedure. According to the Merriam-Webster dictionary algorithm is "a set of steps that are followed in order to solve a mathematical problem or to complete a computer process".

Still this is probably not formal enough. How do we choose the next steps from the set of steps? Should we stop eventually? What is the result of an execution of an algorithm? There can be given many formal definitions of what constitutes an algorithm, however, at this point in the book, without introducing the abstract models of computation and what computation is, we will just give the following definition.

Definition 1.1 - Algorithm

Set of computational steps which specifies a formal computational procedure and has the following properties:

1. After each step is completed the next step is unambiguously defined or the algorithm stops its execution if there are no more steps left
2. It is defined on a set of inputs and for each input it stops after a finite number of steps
3. When it stops it produces a result which we call its output
4. Its steps and their order of execution can be formally and unambiguously specified using some language or notation

---

Algorithms can be expressed in a variety of ways. We can even specify the execution steps using the normal human language. Let's provide a few simple examples. First trivial example will be multiplying two numbers.

Example: Algorithm of integer number multiplication.

Steps:

1. *Given two integer numbers multiply them and return the result*

All the points from the definition of the algorithm are satisfied. There is only one step, after this step algorithm stops, the step is formally specified, all the integer numbers are valid inputs and a valid result will be produced for them. If we denote the algorithm for multiplication as `mult` then, for example

```
mult(2, 5) == 10
```

and we can formally specify the algorithm as follows:

```
mult(x, y) = x * y
```

So far while talking about algorithms we encountered no JavaScript or any other programming language notation. This is quite intentional since we now clearly see that the notion of an algorithm is more of a mathematical one and it is quite abstract. Of course we can express any algorithm using JavaScript but this will be just one of the possible formal representations, in this case also executable by a JavaScript engine.

A careful reader might be a bit puzzled by our confidence at this point. How can we state that any algorithm can be expressed using JavaScript? Can this be in fact proven given the definition of an algorithm we gave earlier? Is JavaScript in fact powerful enough to express all the algorithms there can be? It turns out it is, but we will leave this statement without any proof until the very end of the book where we will discuss abstract models of computation and give even more formal and strict definition of an algorithm.

Let's look again at the definition 1.1 of algorithm and its part that states that we should be able to specify the computational procedure formally. Now it is more or less clear why we require this property from an algorithm. Then we can use a formal language such as JavaScript to specify the algorithm of our interest and execute this formal specification by using a machine such as a laptop or even smartphone. For the number multiplication algorithm this we can specify it like this:

```
1 function mult(x, y) {
2   return x * y;
3 }
```

We see that the specification using JavaScript is much more clear and shorter than using the natural language. Later in the book primarily JavaScript will be used, but we will keep in mind that the discussed algorithms can actually be expressed using other formal notations. Many Computer Science books go as far as inventing their own "pseudocode" in order to not be dependent on implementation details of a particular programming language. We will not go as far as they do and will happily use JavaScript, hence the name of the book "Algorithms with JavaScript".

Another question that we can ask by now are there any computational procedures that are not algorithms? Yes, there are and we will provide examples immediately.

```
function getMaximumNumber() {
  let x = 0;
  while(true) {
    x++;
  }
  return x;
}
```

From this example we also see that not every computational procedure that can be formally expressed using some notation is an algorithm.

Another example of a non-algorithm will be the following division implementation defined on the set of all numbers:

```
function divide(x, y) {
  if (y == 0) {
    throw new Error("Cannot divide by zero");
  }
  return x / y;
}
```

This is not an algorithm because the result is not defined for all the pairs `(x, y)` where `y` is `0` However, it is easy to fix this and make the function divide specify an algorithm:

```
function divide(x, y) {
  return y == 0 ? Infinity : x / y;
}
```

In fact in JavaScript returning Infinity when dividing by 0 is the default behavior so we can just write:

```
function divide(x, y) {
  return x / y;
}
```

And this will be an algorithm, but only given the details of JavaScript engine implementation and the fact that it is possible to divide by `0` in JavaScript.

Now that we are finished with discussing the definition, let's try to take a look at a few more interesting algorithms.