# 2. Data Structures and Algorithms :: Analysis of Algorithms

Remember that an **algorithm** is a step-by-step procedure for solving a problem or performing some operation. When designing an algorithm we are generally interested in finding the **most efficient** way to solve the problem.

But what do we mean by "most efficient" or in other words, what is the definition of **efficiency**? An exact definition of efficiency may by problem-specific, but for computer algorithms the two widely accepted measures of efficiency are:

1. How much **time** does the algorithm take?
2. How much **memory** does the algorithm use?

The former is referred to as the **time complexity** of the algorithm and the latter the **space complexity**. In this course we shall focus on time complexity.

# 2. Data Structures and Algorithms :: Time Complexity

The process of determining the time complexity (and efficiency in general) of an algorithm is referred to as **analyzing the algorithm** and in computer science, the study of algorithms and their efficiency is referred to as **analysis of algorithms**.

Now you might think that measuring the time complexity of an algorithm is a simple matter:

1. Write a program that implements the algorithm.
2. Start a stopwatch at the same time the program begins running.
3. Wait for the program to complete and when it does, stop the stopwatch.

That procedure would certainly tell you how long it takes for the algorithm to solve the problem, but there are a number of important variables that must be addressed:

1. What language do we use to write the program?
2. Which compiler do we use to compile the code?
3. Who do we get to write the program?
4. What machine do we run the program on?

## 2. Data Structures and Algorithms :: Time Complexity (continued)

These variables are important because:

1. *What language do we use to write the program?* The time it takes to execute a program is a function of the high-level programming language used to implement that program. FORTRAN is a very old language (it has been around since the mid 1950's) but it is, without doubt, the best language to use when implementing mathematical and science-related algorithms that involve a lot of number crunching (like predicting weather or simulating a nuclear explosion). On the other hand, BASIC is a very easy language to learn but it does not have a reputation for being fast. You would never write a program in BASIC to predict the weather.
2. *Which compiler do we use to compile the code?* Compilers vary in their ability to generate efficient machine language code, so the same identical C program compiled using two different compilers is likely to execute in different times.
3. *Who do we get to write the program?* A very experienced programmer will, most likely, write the code in such a way that it executes faster than the code that might be written by a less-experienced programmer.
4. *What machine do we run the program on?* There would be a significant time difference between running the program on a late-1990's era Pentium I computer versus a computer with the fastest and most advanced and microprocessor on the market.

There are other variables that would affect the time measurement, but these four should help you understand why that approach is undesirable.

## 2. Data Structures and Algorithms :: Time Complexity (continued)

Therefore, what we need is a way to analyze the time complexity of an algorithm that:

1. Does not depend on an implementation in a specific programming language.
2. Does not depend on the quality of the code generated by a compiler.
3. Does not depend on the skill level of a programmer.
4. Does not depend on the speed or architecture of a specific computer system.