# **Lesson 6 – Introduction to Algorithmic Thinking**

By Dr. Wayne Brown

Outline

* What is an algorithm?
* What is algorithmic thinking?
* How are algorithms communicated?
* Why are algorithms important?
* What are the important properties of algorithms?
* An example – how to create an algorithm

# **What is an algorithm?**

An algorithm is a precise, step-by-step set of instructions for solving a task. An algorithm does not solve a task; it gives you a series of steps that, if executed correctly, will result in a solution to a task. You use algorithms every day but you often do not explicitly think about the individual steps of the algorithm. For example, starting your car, putting on your clothes, logging into your computer, or following a recipe for cooking a food dish, are all accomplished using an algorithm, a step-by-step series of actions.

For an algorithm to be valid, each step (or instruction) must be:

* unambiguous – the instruction can only be interpreted in one unique way
* executable – the person or device executing the instruction must know how to accomplish the instruction without any extra information.
* ordered – the steps of an algorithm must be ordered in a proper sequence to correctly accomplish the task.

An example algorithm is shown in the seven steps below. This algorithm explains how to make "frozen lemon ice-box pie," (which is this author's favorite dessert).

Step 1: Pour 1 can of chilled pet milk into a mixing bowl.

Step 2: Beat the pet milk until peaks form.

Step 3: "Fold in" 1 cup of sugar and 1/2 cup of lemon juice.

Step 4: If you want a more lemony taste, add 1 tsp of grated lemon peal.

Step 5: Spread a graham cracker crust on the bottom of a 13" by 9" pan.

Step 6: Pour the pet milk mix into the pan.

Step 7: Place in freezer until frozen.

Before leaving this example, ask yourself if the steps to this algorithm meet the criteria listed above. That is, is each step unambiguous, executable, and in the correct order?

This reading explains the concept of algorithms, some important properties of algorithms, and how algorithms are created, executed, and validated.

## **What is Algorithmic Thinking?**

*Algorithmic thinking* is the ability to ***understand, execute***, ***evaluate***, and ***create*** algorithms. Let's discuss each of these ideas separately.

To be an *algorithmic thinker* you need the ability to ***understand*** and ***execute*** algorithms. Some people find it easy to follow a series of precise instructions while other people find it very challenging. Some people seem to lack the patience and diligence required to follow a step-by-step plan. However, it is a valuable skill that all people should master. Algorithmic thinking requires patience because each instruction must be executed in its correct sequence without skipping ahead or "glossing over" some of the instructions. In addition, algorithmic thinking requires diligence and perseverance. It is often tedious to follow the steps of a complex algorithm and people sometimes fail to complete an algorithm because they simply "give up."

*Algorithmic thinking* also requires the ability to ***evaluate*** algorithms. This involves determining if an algorithm really does solve a given task. This can be very challenging. For example, a "preflight check list" is an algorithm for preparing an aircraft for take off. Suppose you were given the job of determining if a new "preflight check list" for the F35 (Joint Strike Fighter) is ***correct*** (checks all systems on the aircraft properly) and ***complete*** (does not leave out any important checks). Hopefully you would agree that this is an important job – pilot's lives are dependent on getting it right – and that getting the check list correct and complete will not be easy. (As a side note, one way to create the preflight check list is to determine the fault of each new aircraft crash and add a new check to the list to prevent future crashes. But the cost in lives and money is too great. We need to get the algorithm correct and complete before the first plane takes to flight to prevent a single aircraft loss!)

And finally, *Algorithmic thinking* includes the ability to ***create*** new algorithms. This is probably the most challenging aspect of algorithmic thinking. Given a task, can you create a series of precise, step-by-step instructions that always solves the task correctly? Obviously the complexity of the task has a big impact on the complexity of an algorithm that will accomplish the task. Simple tasks can typically be accomplished with simple algorithms, while complex tasks typically require more complex algorithms.

A critical aspect of algorithm creation is the "target executer" of an algorithm. For example, the "target executer" of a preflight check list is a pilot. The average person on the street could not execute the preflight check list correctly because they would not understand the instructions. The three, previously stated, characteristics of an algorithm instruction (i.e., an instruction must be unambiguous, executable, and ordered) must be consistent with the knowledge and expertise of the "target executer." When you create an algorithm, make sure that each instruction can be unambiguously interpreted and executed by the "target executer."

We live in the "information age," where many of the tasks people need (or want) to do can be accomplished by computers. However, computers currently have no real understanding or cognition – they can only perform tasks that people have developed algorithms to solve. Therefore, to create algorithms that computers can execute, you must understand what a computer is capable of executing and be able to write a series of unambiguous instructions that a computer can execute successfully. For the remainder of this document we will restrict our discussion to the development of algorithms that can be executed on computers.

## **How are algorithms communicated?**

Each step (instruction) of an algorithm must be precisely stated. It is very difficult to explain algorithms precisely using only the English language. For example, please re-read the example algorithm for making "frozen lemon ice-box pie" and then answer the following questions:

* Does pet milk come in different size containers? How much milk is "1 can"?
* Does it matter what types of beaters are used to beat the pet milk?
* Is "tsp" a teaspoon or a table spoon?

If you tried to implement the recipe to make frozen lemon ice-box pie you might very well discover other ambiguities in the instructions. To eliminate these "communication problems," we need a precise language for specifying algorithms, especially since our "target executioner" is a dumb computer. A precise language for stating the instructions of an algorithm that can be executed by a computer is called a "programming language."

Many programming languages have been developed over the years and each language has its own special features and benefits. You have probably heard of some of these languages, such as C, C++, Pascal, BASIC, Java, Perl, Python, and Ada. It requires a significant amount of time to learn a new programming language and it is not a goal of CS110 to teach you how to program computers. We want cadets to become better problem solvers by becoming algorithmic thinkers. One of the common ways to express algorithms without using a programming language is called flowcharting[[1]](#footnote-0). Flowcharts provide a visual description of a step-by-step process. The major drawback to using flowcharts is that they are tedious to draw and hard to modify if they are hand drawn on paper. A computer program called RAPTOR was developed by Dr. Carlisle here at USAFA which allows a person to create an algorithm in flowchart form and then actually execute the flowchart to test its validity. Pretty cool! We will discuss the details of how to use RAPTOR in future lessons.

## **Why are Algorithms Important?**

Algorithms are important for many reasons:

* An algorithm documents the "how to" for accomplishing a particular task.
* If an algorithm is written well, it can be used to accomplish not only a single task but a whole group of related tasks.
* The existence of an algorithm means that the task can potentially be automated (i.e., performed by a computer).
* The automation of redundant, tedious, or dangerous tasks frees people from having to perform these boring, time-consuming, or potentially lethal tasks.
* The automation of some tasks makes new things possible (e.g., accessing web pages from all over the entire world in the blink of an eye).

Consider telephones for example. When telephones were invented, the problem of how to connect one phone caller to another caller arose. For many years this problem was solved manually by phone operators who pulled wires from a console and plugged them into the correct connecting wire. It has been estimated that if this manual method was still used today to handle the current daily phone calls made around the world, every person on the planet would have to be a phone operator! Thankfully, given the invention of fast computers and algorithms to instruct those computers, almost all phone calls made today are automatically connected without human intervention. What would your life be like without a phone? Can you imagine it? Without algorithms, that is the way life would be!

We study algorithms in CS110 because algorithmic thinking is transferable. If you can think and reason precisely and solve algorithmic problems in one domain (e.g. computer programming), then your ability to analyze and solve problems in other areas will improve.

**What are the important properties of algorithms?**

The following is a list of some of the important properties of algorithms. This list is by no means comprehensive or complete.

**Property 1**: For any given, non-trivial task (or set of related tasks), there are many possible algorithms for accomplishing the task.

This greatly confuses many students. "You mean there is more than one correct answer?" The answer is YES! Remember, an algorithm is not the solution to a problem; it is a step-by-step set of instructions for finding a solution. Is there only one way to get from Mitchell Hall to Arnold Hall?

**Property 2**: An algorithm does not encode the underlying theory behind the instruction steps.

An algorithm tells you how to accomplish a task. If the algorithm is correct, and you follow the instructions exactly, you will accomplish the task. But you may not understand why you did some of the steps. If a computer is executing an algorithm, this lack of understanding is not a problem. If a person is trying to understand an algorithm so that it can be updated or enhanced, this lack of understanding can make changes almost impossible. Complex algorithms often have associated documentation that provides detailed explanations of how they work.

**Property 3**: Some algorithms are more efficient than other algorithms.

The execution of an algorithm requires some amount of time. If an algorithm is used often, its efficiency becomes an issue. It is common to create multiple algorithms to accomplish a particular task and then select the faster algorithm to include in a final product. One of the challenges of software development is deciding how much time and effort you should expend on the search and development of more efficient algorithms.

**Property 4**: Computer programs that are used over many years typically must be modified over time to adapt to changes in task requirements.

Computer programs are not typically static objects. They require "maintenance" to keep them current with changes in task requirements. For example, a computer program that calculates income taxes must change every year as the income tax laws change. Some studies have shown that up to 75% of software development costs are consumed by the maintenance of existing software.

In conclusion, when we develop new algorithms, we attempt to create one of many possible sequences of instructions that will solve our task at hand. Ideally we would like the algorithm to execute quickly and be well "commented" so that it can easily be maintained over time.

**An Example – How to create an algorithm**

This section will walk you through the process of creating an algorithm for a simple problem. Hopefully you will use the principles discussed here when it comes time for you to develop your own algorithms for different tasks.

Sample Problem: Two people work for the same company and have a total combined salary of W dollars, but one person earns Q dollars more than the other person. How much did they each earn?

**Principle 1:** Solve a *specific instance* of the problem *by hand* using pencil and paper.

A specific instance of a problem is when all the "variables" are given a specific value. Suppose we let W be $57.00 and let Q be $5.00. Now the problem reads:

|  |  |
| --- | --- |
| *Specific Problem Instance*: | Two people work for the same company and have a total combined salary of $57.00, but one person earns $5.00 dollars more than the other person. How much did they each earn? |

There are multiple ways to think about this problem. Two possible solutions are:

|  |  |
| --- | --- |
| **Arithmetic Reasoning:** | **Algebraic reasoning:** |
| Of the $57, Person1 gets $5 more then Person2.  Therefore, both equally share $57-$5 = $52.  $52 / 2 is $26, so each gets at least $26.  Person1 gets $31 ($26 + $5) Person2 gets $26 | Let X = Person1's pay  Let Y = Person2's pay.  Therefore, we know from the problem that:  X = Y + 5 and  X + Y = 57.  Solve these 2 equations using substitution:  ( Y + 5) + Y = 57  Y = 26  Therefore X = Y + 5 = 31 |

Please note: If you cannot solve this problem manually, it is impossible for you to write an algorithm to solve the problem!

**Principle 2:** Generalize your solution by replacing your specific instance values with "variables".

The two previously shown solutions now become:

|  |  |
| --- | --- |
| **Arithmetic Reasoning:** | **Algebraic reasoning:** |
| Of the total salary W, Person1 gets Q dollars more then Person2.  Therefore they both equally share (W - Q).  Each gets at least (W – Q) / 2.  Person1 gets (W – Q) / 2 + Q Person2 gets (W – Q) / 2 | Let X = Person1's pay  Let Y = Person2's pay.  Therefore, we know from the problem that:  X = Y + Q and  X + Y = W.  Solve these 2 equations using substitution:  ( Y + Q) + Y = W  Y = (W – Q)/2  Therefore X = (W – Q)/2 + Q |

**Principle 3:** Manually execute your algorithm on several test cases to verify that it produces correct answers. This is called *desk checking* or *walking through the algorithm*.

.

Example 1: W = 100, Q = 10

Person1 = (100 – 10)/2 + 10 = 55

Person2 = (100 – 10)/2 = 45

(This is correct because 55 + 45 = 100 and Person1 earns 10 more dollars than Person2)

Example 2: W = 31, Q = 2

Person1 = (31 – 2)/2 + 2 = 16.5

Person2 = (31 – 2)/2 = 14.5

(This is correct because 16.5 + 14.5 = 31 and Person1 earns 2 more dollars than Person2)

**Principle 4:** After a general solution to the task is known, write correct programming statements to implement your solution on the computer.

How to write correct programming statements will be covered in the next few lessons.

1. <http://en.wikipedia.org/wiki/Flowchart> [↑](#footnote-ref-0)